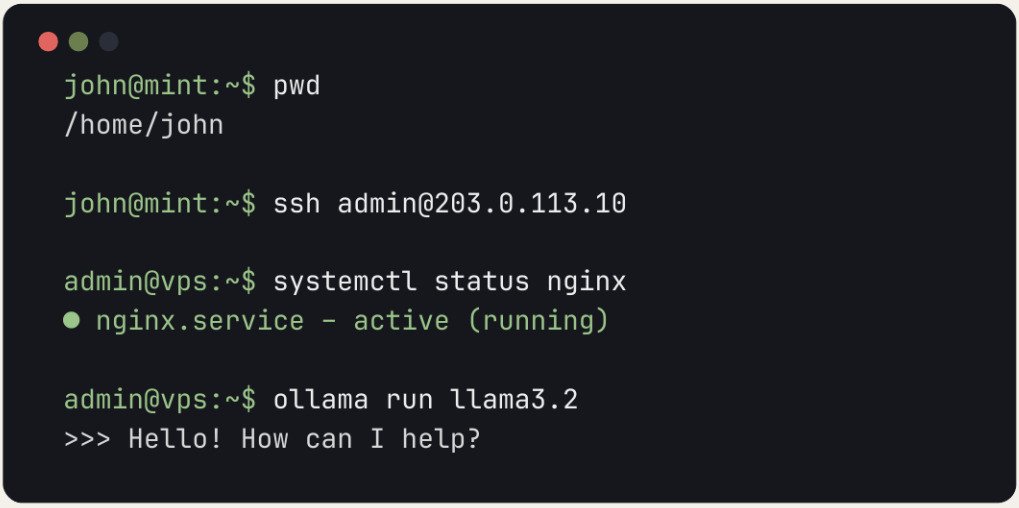
OVANAP TEAM

LINUX FROM ZERO

From your first command to your own server and AI assistant



A practical guide for beginners

OVANAP TEAM

Linux from Zero

From your first command to your own server and AI assistant

Ovanap · 2026

Linux from Zero. From your first command to your own server and AI assistant.

© Ovanap Team, 2026. All rights reserved.

Commands tested with Linux Mint 22.3 “Zena”, Ubuntu Server 26.04 LTS and Debian 13; Docker 29.x, Git 2.43–2.55, nginx 1.28 (Ubuntu repository). Fast-changing details live in the online appendix.

Trademarks mentioned in this book belong to their owners. Linux® is the registered trademark of Linus Torvalds.

Publisher: Ovanap (Monine AS, Norway)

First edition, 2026

Free sample

The introduction and Chapter 4, “The Terminal: Your First Conversation with the System”.
The full book has 26 chapters, 21 labs and a final project.

Contents

Introduction: Who This Book Is For and How to Use It 6

4. The Terminal: Your First Conversation with the System 13

Introduction: Who This Book Is For and How to Use It

Think of this short chapter as the map you study before a trip. By the end of it, you will know whether the book is for you, what you will be able to do by the last page, and how to use the command blocks, info boxes, labs, and online appendix. There are no commands here, only an explanation of how the book is organized.

Who this book is for

You know how to use a Windows computer and a web browser. Maybe you have heard that "all the serious stuff runs on Linux," or you want to understand what happens behind those friendly windows. Maybe you want your own website, your own server, or an AI assistant that runs on your computer and not on someone else's. If even one of these describes you, this book was written for you.

It does not matter where you live or what your first language is. Linux works the same way everywhere, and so does everything in this book.

My promise is modest but firm. Once you have read the book and done the hands-on work, you will be able to do fifteen things, in three groups.

The first group is about running your own computer:

1. Understand what an operating system is and why Linux matters so much.
2. Work in the terminal: talk to the system in text, not just with the mouse.

3. Manage files and directories safely.
4. Manage users and permissions.
5. Install and update programs, and find and stop a program that has frozen.
6. When something breaks, find the cause in the system logs.

These are the foundations for everything else.

The second group is about running your own server. A server is a computer that runs all the time and provides services to other computers. For example, it can host a website. It can sit in your room, or it can be a machine you rent far away.

7. Understand how networks work: what an IP address, a domain name, and a port are.
8. Rent your own server, connect to it safely over SSH, and prepare it for work.
9. Run a website on it with your own domain and a secure HTTPS connection.
10. Make backups and test that you can actually restore data from them.
11. Automate repetitive work: write scripts and schedule tasks.
12. Track versions of your files with Git.
13. Run programs in containers with Docker.

The third group is about running your own AI assistant:

14. Understand how large language models work, what a token and an API key are, and how much all of it costs.
15. Run an open AI model on your own server or computer, and combine everything you have learned into one working project.

At the end of the book, a final project takes you from an empty server to a working AI assistant that you own and control.

Who this book is not for

This book does not teach programming. It does not teach you to break into other people's systems. It teaches defense and safe working habits that keep your own server secure. Nor is it a theory course: each idea appears only when a concrete task needs it, and you put it to use right away.

You do not need any experience. The book starts from zero, with what a command window looks like the first time you open one, and it takes nothing for granted. If you can use Windows and a browser, you have everything you need.

How to use this book

The chapters come in a deliberate order. Each one builds only on ideas from earlier chapters, so read them in order, at least the first time through. If you already know a topic, you can skip it. Before you do, read that chapter's "What you can do now" section and make sure every item is clear to you.

Almost every chapter ends with a lab: a longer, real task that you do from start to finish. In one lab, for example, you set up a practice environment. In another you write your first script or launch a website. Labs are not tests; they are where theory turns into skill.

You will run all your experiments in a practice environment: a separate place where mistakes have no consequences. In this book, that place is a virtual machine, which works like a separate computer inside your computer and runs as an ordinary program on Windows. You will install Linux Mint in it and do your first experiments there. If something goes wrong, you roll the virtual machine back to an earlier state and try again. Your real computer stays untouched. Chapter 2 shows how to set all this up.

The book uses the English interface of Linux Mint and every other program. That way, what you see on your screen matches the text, and it also matches most of the help you will find online.

The most important rule: reading is not enough; you have to do it. You remember a command you typed far better than one you only read, especially if you got it wrong the first time and had to track down the mistake. So type every command block yourself in your practice environment. Here, making mistakes and fixing them is part of learning, not a failure.

How to read the commands and info boxes

All commands in the book are grouped into numbered blocks. A block number has two parts: the chapter number and the block's position in that chapter. So "Command 4.3" means the third command block in Chapter 4. Each block comes with an explanation: what you do, why you do it, what changes in the system, and what the risk is. How to tell that it worked is shown by the expected result after the block or by a "Check." box.

Now and then you will see framed info boxes next to the text. They never repeat what the text says; each one adds something new:

- "Did you know?" – a fact or detail that widens the topic. You can keep learning without it.
- "Important." – something worth remembering before you move on.
- "Dangerous command." – a command that can delete or change something permanently. It always comes with a warning and, where one exists, a way back or a precaution.
- "Check." – a short action that confirms everything worked.
- "Common mistake." – a mistake beginners often make, and why it happens.
- "Undo." – how to reverse what you did.
- "Try it." – a small task you do right here and now.
- "In short." – the core of a section in a few sentences.
- "Updated online." – a sign that the details at this spot are kept up to date in the online appendix.

- "Debian – different." – a note for readers on Debian, where it behaves differently from Mint and Ubuntu.

Which versions the book was tested with

These are the versions used to test the commands in this book:

Component	Tested version
Desktop system	Linux Mint 22.3 "Zena" (updates until April 2029)
Server system	Ubuntu Server 26.04 LTS (supported until May 2031)
Debian family	Debian 13 "trixie"
Docker (a tool for running programs in containers)	29.x
Git (a version control tool)	2.43 (Mint 22.3), 2.53 (Ubuntu 26.04); latest: 2.55
nginx (a web server program)	1.28 (Ubuntu 26.04 repository); newest stable 1.30.x

If your versions are newer, that is fine. The principles do not age: file systems, permissions, networks, and services will work the same way ten years from now. What ages is the detail: where a button sits, which AI model versions exist, what things cost. For example, Linux Mint 23 (based on Ubuntu 26.04) is planned for late 2026. Some things in it will look different, but the principles in this book will still apply.

Those changing details live in the online appendix, a web page for this book that is updated after publication. You can open it at `https://linuxfromzero.com/appendix`, or point your phone camera at the QR code in Appendix G (the black-and-white square). Wherever information could go out of date, you will see an "Updated online" box.

What you can do now

- You know that this book is for people with no technical background, and you know what it does not teach.

- You know the three stages of the journey: your own computer, your own server, your own AI assistant.
- You know what a lab and a practice environment are, and why the work happens in a virtual machine.
- You can read a command block number, for example "Command 4.3."
- You recognize all ten info boxes and know what each one is for.
- You know which program versions the book was tested with.
- You know where to find updated details: in the online appendix, through the QR code or its web address.

Exercises

1. Look at the book's roadmap (Figure 0.1) and find the part that interests you most. Write down which chapter it starts in.
2. Write down one concrete goal you want to reach by the end of the book: a website, a server, or an AI assistant. Put the note somewhere you will find it when you reach the final project.
3. Leaf through the book and find the first "Dangerous command" box that warns about a real command. Read it, and note the precaution it recommends.
4. Point your phone camera at the book's QR code, or open `https://linuxfromzero.com/appendix`, and look through the online appendix. Check whether the desktop and server versions listed there still match the table in this chapter.

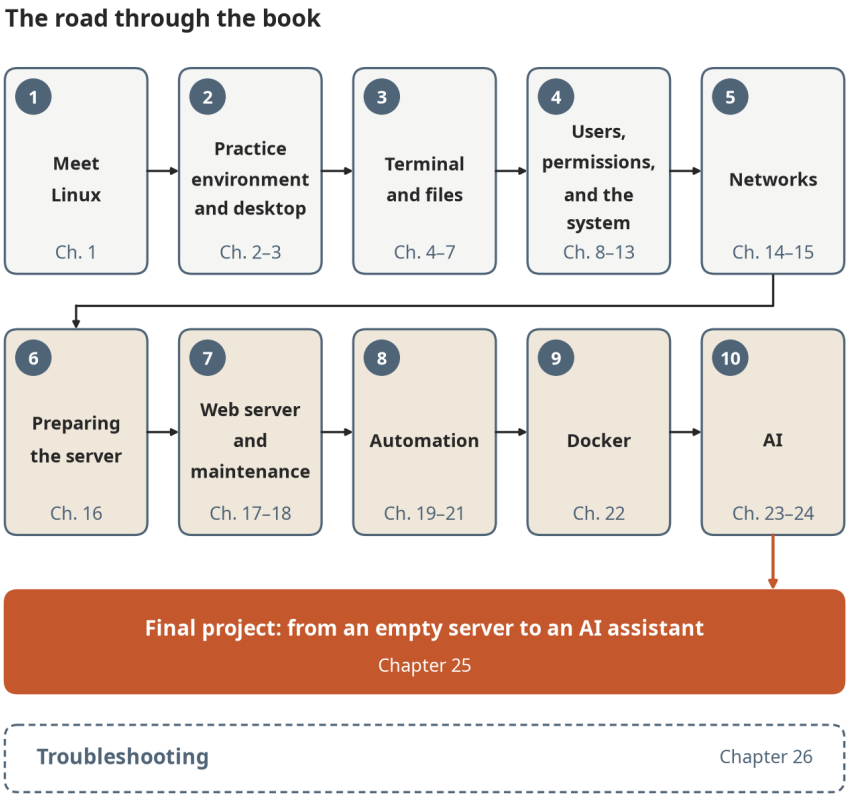


Figure 0.1 The road through this book: from meeting Linux to the final project

Before you move on

What does the label "Command 7.2" mean?

Answer: it is the second command block in Chapter 7. The first number is always the chapter, and the second is the block's position in that chapter.

4. The Terminal: Your First Conversation with the System

Until now you have "talked" to Linux with the mouse: clicking menus, dragging windows, pressing buttons. In this chapter you will open the dark window that scares many Windows users and discover that it is only a text conversation with the system. By the end of the chapter, you will be able to open the terminal, understand the line with your name and a dollar sign, run your first commands, and ask the system for help. Most important, you will see for yourself that a mistake in the terminal breaks nothing.

The store window and the storekeeper

In Windows, almost everything happens with the mouse. Want to create a folder? Right-click, "New Folder." Want to see how much disk space you have? "This PC," right-click, "Properties." A graphical user interface (GUI) is convenient as long as you do what its designers planned for. When you want something they did not plan for, you have to dig ever deeper through menus.

A comparison helps here. The graphical desktop is like a store window. You see what has been laid out, and you can point at it and take it. The terminal is a direct conversation with the storekeeper in the back. You say in words what you want, and the storekeeper answers in words. The back room has everything, even the things nobody put in the window (Figure 4.1 shows both sides next to each other).

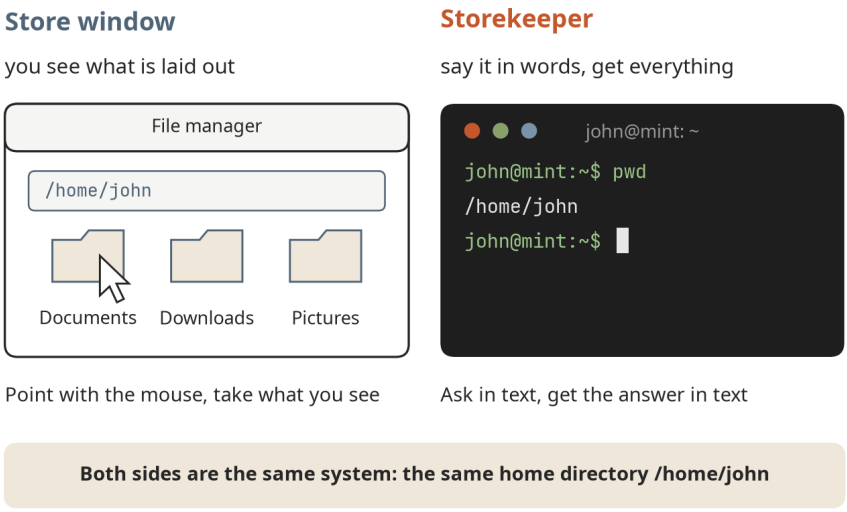


Figure 4.1 The store window and the storekeeper: the graphical interface and the terminal

This conversation has three rules.

First, you ask in text, and the system answers in text. Second, the system understands only exact sentences, called commands. Third, if a sentence is wrong, the system says "I don't understand" and does nothing. That last rule makes experimenting in the terminal safe: typing by itself changes nothing. Only specific commands change things, and before each one the book tells you what will change.

There is one more reason to learn this conversation, even though the store window is more comfortable: text can be saved. You cannot write a mouse movement down on paper, but you can write down a command, pass it to a friend, and repeat exactly the same thing a year later. You will not use this much in this chapter, but keep it in mind: everything you learn here will later grow into scripts, servers, and assistants that work for you. All of it starts with text you can read.

> **Important.** You are working in a virtual machine and you have a snapshot from Chapter 2. Even if you broke something, you could go back to it. In this chapter, though, you will not need that safety net: none of the commands here deletes or overwrites anything. You only look, ask, and read the answers.

Opening the terminal

In Chapter 3 you looked for applications in the menu, and the terminal is there too. The application is called "Terminal," and its icon is also pinned to the panel next to the menu. It is the window where the text conversation happens.

Steps:

1. Click the menu button in the bottom-left corner of the screen.
2. In the search box, start typing: `terminal`.
3. When you see "Terminal," click it.

A dark window opens with only a few characters in it. If that makes you a little uneasy, that is normal. As a test, close it with the window's close button, X. Now open it again, this time with the keyboard: press and hold `Ctrl` and `Alt`, then press `T`. The window opens instantly. `Ctrl`+`Alt`+`T` is probably the key combination Linux users press most often, and it is worth remembering, because you will open the terminal hundreds of times.

Did you know? The word "terminal" comes from the days when computers were huge and people connected to them through simple devices: a keyboard and a screen. Even earlier, teletypes (printing telegraph machines) played this role. That is why you will still sometimes see the abbreviation "tty" in Linux, an echo of that era. Today your terminal is only a window on the screen, but it still behaves the old-fashioned way: you type text, and you get text back.

What you see: the prompt

The terminal window shows one line and a blinking block. The line looks roughly like this:

```
john@mint:~$
```

This is the prompt, the system's way of asking "What next?" As long as you see it, the system is waiting. The blinking block is the cursor; it shows where the text you type will appear.

Here is the prompt, piece by piece:

- `john` – your username. Linux always knows who is working in the terminal, and it tells you.
- `@` – a separator. Read it as the word "at": john at mint.
- `mint` – the computer's name, which you gave the system during installation.
- `~` – a special sign that means "my home directory." A directory is what Windows calls a folder. Linux users say "directory," and this book uses that word from here on. The tilde is shorthand for a long path: `/home/john` and `~` mean the same place.
- `$` – a signal that the system is ready to accept a command. In later chapters you will sometimes see `#` in its place; that sign marks a different role, which you will learn about later. For now, one thing matters: if you see `$`, you can type.

Your username, the computer's name, and your location all fit on one short line. A single glance tells you who you are, which computer you

are on, and where you are, like the running header at the top of a book page. Figure 4.2 labels each part of the line with arrows.

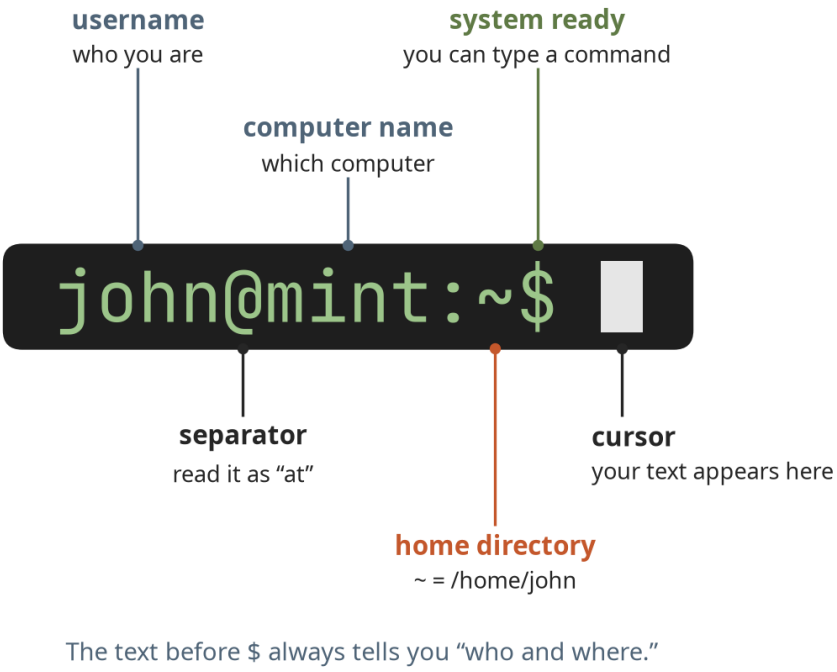


Figure 4.2 The parts of the prompt

> **Important.** Your prompt may look different: another username, another computer name, or a directory name instead of `~`. That is normal. The text before the dollar sign always tells you "who and where," and it changes when you change location. You will see this in the next chapter.

Terminal and shell: two different things

Before the first command, here is one more idea that will spare you confusion later: the thing you opened and the thing that talks to you are not the same.

The terminal is a window. It only shows text and passes on your key presses, like a telephone that only carries a voice. Your real partner at the other end of the line is a program called the shell. The shell reads what you typed, works out what you want, starts the programs needed, and returns the answer. It is the "storekeeper" from the start of the chapter.

On your Mint system, the shell is called Bash. The name is a pun: *Bourne Again Shell* refers to the earlier Bourne shell, named after its author, Stephen Bourne. The history matters less than this: you will sometimes see the word `bash` in error messages or guides, and now you know it means your conversation partner, not yet another mysterious program.

Why tell the window and the partner apart? Because one can change while the other stays. Later in the book you will connect to other computers over the network. The terminal window will be on your computer, while the shell talks to you from a remote server: the same conversation over a longer "wire." The server chapters cover this.

Command 4.1: pwd – where am I

In a new place, the first question is always the same: where am I? The file manager shows your location in the location bar at the top. In the terminal, you ask for it with a command.

A command is a word the system understands as an instruction. Your first one is `pwd`, short for *print working directory*. The file system is the way all the files and directories on your disk are arranged, and the working directory is the place in it where you are "standing" right now. Think of a store: you stand in one department, not in all of them at once.

Before you type, one convention: you type a command after the `$` at the end of the prompt and finish it with the `Enter` key. `Enter` is the "say it" button. Until you press it, the system does nothing except show what you are typing. Also note that the prompt text is not part of the com-

mand. `pwd` is the command; `john@mint:~$` is only the place where you type it.

Command 4.1

```
pwd
```

What you do: type `pwd` and press `Enter`. *Why:* you want to know where in the file system you are standing. *What changes:* nothing – the system only prints a path. *Risk:* none.

The result you will see:

```
/home/john
```

Below it, a new prompt appears. The system has answered and is waiting again.

Read the answer like this: you are in the directory `john`, which is in the directory `home`, which sits at the very beginning of the system. The slash `/` separates the names, and the first `/` means "the very beginning." The next chapter covers this tree in detail. For now, one thing is enough: `/home/john` is the full path to your home directory, the same directory you opened in the file manager in Chapter 3.

> **Check.** Does your answer start with `/home/`? If so, all is well, whatever comes after it. If you see something else, don't worry; carry on. The next chapter explains paths and the tree in detail.

Notice that the `~` in the prompt means the same thing as the `/home/john` in the answer. The terminal tells you your location in two ways: as shorthand in the prompt and as the full path in the answer.

Command 4.2: ls – what is here

You know where you are. The logical next question is: what is here? In the file manager, you see the answer at once as icons in the open dir-

ectory. In the terminal, you ask for the contents with the `ls` command, short for *list*.

Command 4.2

```
ls
```

What you do: type `ls` and press `Enter`. *Why:* you want to see which directories and files are in your working directory. *What changes:* nothing, only text on the screen. *Risk:* none.

You will see the directories the system created for you, the same ones you saw in your home directory in the file manager: `Desktop`, `Documents`, `Downloads`, `Pictures`, and others. The exact list on your screen may differ. That is normal.

Now a new idea: the argument. On its own, `ls` lists the place where you are standing. But you can be more specific and say "list that place." The place you name after the command is called an argument. An argument adds an answer to "what?" or "where?" to the command.

Try it with an argument. Type:

```
ls /
```

Here `ls` is the command and `/` is the argument: "list the very beginning of the system." In the answer you will see the main directories of the whole system, for example:

```
bin  etc  home  usr  var  ...
```

Your list will have more names; I kept the most important ones and replaced the rest with an ellipsis. One of them, `home`, is already familiar: it is the directory that holds your home. Later chapters explain what the others are for. For now, notice that the same `ls` command with different arguments shows different places.

The rule: spaces separate a command from its arguments. If you type `ls/` without a space, the system tries to run a single word, `ls/`, and answers `bash: ls/: No such file or directory`: there is no command

or file with that name. In the terminal, a space is not decoration; it is a separator.

Command 4.3: ls -l – options

A command can take one more kind of addition: an option (also called a flag). If an argument answers "what?", an option answers "how?" The same question, "what is here?", can get a short answer or a detailed one, and the option decides which.

Options are written after a hyphen, usually as a single letter. The `ls` command with the `-l` option means "list, but in the long, detailed format."

Command 4.3

```
ls -l
```

What you do: type `ls`, a space, `-l`, and press `Enter`. *Why:* you want the same list, but with more detail about each item. *What changes:* only how the answer looks. *Risk:* none.

Your home directory already contains directories the system created, so the answer will look roughly like this (a shortened example):

```
total 32
drwxr-xr-x 2 john john 4096 Sep 24 10:15 Desktop
drwxr-xr-x 2 john john 4096 Sep 24 10:15 Documents
drwxr-xr-x 2 john john 4096 Sep 24 10:15 Downloads
...
```

The first line, `total`, shows how much disk space (in blocks) the listed items take up. Then comes one line per item: owner, size, date, name, and a few more columns that the chapter on permissions explains. The dates and numbers on your screen will be different. If a directory is completely empty, you will see only one line: `total 0`. Don't try to decode the columns yet; for now, notice that the `-l` option changed the shape of the answer.

Now let's sum up what a command line is made of. Take the command `ls -l /` (you can run it too) and split it into three parts. `ls` is the command: it says WHAT to do. `-l` is the option: it says HOW to do it. `/` is the argument: it says WHAT TO ACT ON. The diagram shows the same three parts (Figure 4.3 is a larger version with more examples):

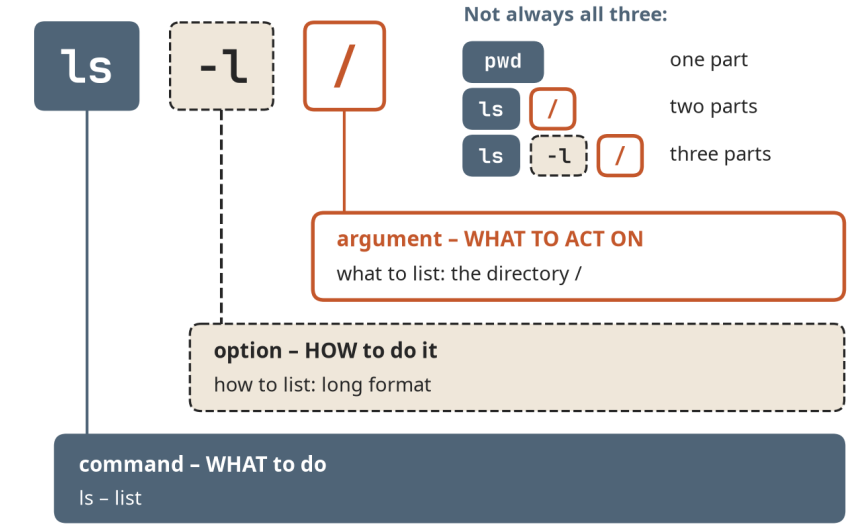


Figure 4.3 The parts of a command line: command, options, arguments

```
ls    -l    /
│     │     └── argument: WHAT to list (the directory /)
│     └──────── option: HOW to list (long format)
└────────────── command: WHAT to do (ls - list)
```

One line, three roles. Not every command has all three: `pwd` had only the first part, `ls /` had a command and an argument, and `ls -l /` has all three. Read a command line like a short sentence: action, manner, object.

One more detail will save you frustration later: Linux distinguishes uppercase from lowercase. `LS` is not the same as `ls`, and the `-l` option

is not the same as `-L`. Windows did not prepare you for this, but here it is a firm rule: if two command or file names differ in the case of even one letter, they are different names.

> **Common mistake.** You type `ls-l` without a space, or `LS` in capitals, and the system replies that there is no such command. This is not a fault; it is the price of an exact language. Always separate the command, option, and argument with spaces, and type letters exactly as they appear in the book.

A deliberate mistake: badcommand

Now make a mistake on purpose. This is the most important part of the chapter. You need to experience, once, what happens when the system does not understand you, and see for yourself that everything is fine afterward.

Type a word that is not in any dictionary:

```
badcommand
```

Press `Enter`. The system answers:

```
badcommand: command not found
```

Then it shows the prompt again. That is the whole "damage." The system looked for a command with that name, did not find it, and said so. It did not delete anything or switch anything off. It answered like a storekeeper who says, "We don't have that item," and then asks what else you need. Figure 4.4 shows the terminal after the mistake.



That is all that happened: the system said "not found" and is waiting again.

Figure 4.4 The terminal after an error: the system tells you what went wrong

> **Common mistake.** `command not found` almost always means one of two things: a typo (check letter by letter, including uppercase and lowercase), or a program that is not installed on your system yet. In the first case, fix the spelling. In the second, install the program; you already know how to do that with the Software Manager from Chapter 3. If the name you typed resembles known commands, Mint prints a longer version of the same message instead: `Command '…' not found, did you mean:`, a list of similar commands, and the hint `Try: sudo apt install <deb name>`. Do not run it for now; it is only a suggestion. This message never means the system is "broken."

Notice how this error message is built: first the system repeats what it was looking for, then it gives the reason. Most Linux error messages follow this pattern: "what" first, then "why not." When you meet longer error messages later, read them the same way. Don't be afraid; look for these two parts.

This deliberate mistake is worth your time for one more reason. Many people find the terminal frightening because the first time they saw it, something had "broken" and error text was pouring down the screen. Now you know that an error message is not a mark of shame but an

answer: the system is trying to help you, briefly. Experienced users make dozens of mistakes in the terminal every day. What sets them apart from beginners is not that they never make mistakes, but that they read the answer and try again.

Ctrl+C: the stop button

Sometimes the terminal seems "stuck": you type and nothing happens, or a command keeps running and you want it to stop. For those moments, there is one key combination: `Ctrl+C`.

Hold `Ctrl` and press `C`. It means "stop what is happening now." The system stops the running command and gives you back the prompt. Try it now: in the open terminal, on an empty line, press `Ctrl+C`. A new prompt appears. The system has dropped the "unfinished thought" and is waiting again.

`Ctrl+C` has one more use worth remembering. If you realize halfway through typing a command that it is wrong, you do not need to delete it character by character. Press `Ctrl+C`, and the whole line is canceled so you can start over.

There is one situation where beginners really need `Ctrl+C`, and you will run into it soon. The `echo` command prints the text you give it. You will try it properly in the lab; here only the quotation mark matters. If you type text with an opening quotation mark and forget to close it (for example, `echo "Hello`) and press `Enter`, the shell prints nothing. Instead of the prompt, a single `>` sign appears, and the system waits. This is not a freeze. Bash knows the sentence is unfinished and is waiting for the rest: the closing quotation mark. You can type it and get the result, or press `Ctrl+C` to return to a clean prompt. Remember: `>` means "waiting for the rest," not "broken."

One warning: in the terminal, `Ctrl+C` is NOT copy, as it is in Windows. If you try to copy text from the terminal this way, you copy nothing and only stop the running command. In the terminal, copy with `Ctrl+Shift+C` and paste with `Ctrl+Shift+V`. An easy way to remember: in the terminal, copy and paste always get an extra `Shift`.

> **Dangerous command.** There are no dangerous commands in this chapter. But let's agree on a rule for the whole book now: if a command starts with a word you have not learned yet and asks for a password, stop and read the explanation first. A password prompt means the command will change something important. Chapter 8 explains more.

Help: --help, man and tldr

The storekeeper you are talking to has a good habit: the instructions are always at hand. You can ask for them in three ways, and you will learn all three now.

The first way is to ask the command itself. Almost every command responds to the `--help` option, which means "tell me how to use you." Notice the two hyphens before the word. Short one-letter options get one hyphen (`-l`); whole words get two (`--help`). Almost every command follows this convention.

Command 4.4

```
ls --help
```

What you do: ask the `ls` command to describe itself. *Why:* you want to see all of its options. *What changes:* several screens of text appear, and the top lines "scroll away" upward. *Risk:* none.

The first lines of the answer will look like this:

```
Usage: ls [OPTION]... [FILE]...
List information about the FILEs (the current directory by default).
```

Text that has scrolled up is not gone: you can scroll back with the mouse wheel.

Here is how to read it. "Usage: `ls` [OPTION]... [FILE]..." means: the command, then possibly some options, then possibly a file or directory. Square brackets mean "optional," and the ellipsis means "there can be several." The second line says what the command does in general: it

lists information about the files you name, or about the current directory if you name none. "By default" means "this is what happens if you specify nothing."

Below that is a long list of options in alphabetical order. You do not need most of them yet. Find `-l`, an option you already know. That feeling of "I recognize this one" is exactly what `--help` is for: not reading everything, but quickly checking whether the command has what you need.

The second way is the system manual. Linux has its own reference library, which you open with the `man` command (short for *manual*). Type `man`, a space, and the command name to get a detailed description:

```
man ls
```

A full screen of text opens. Don't be alarmed: this is not a new window but a viewer program. Every manual page has the same structure, and it is worth learning to recognize:

- **NAME** – the command name and a one-sentence description ("ls - list directory contents").
- **SYNOPSIS** – how the command is written, with all its possible parts, in the same square-bracket notation you already saw in the `--help` output.
- **DESCRIPTION** – a detailed explanation of what the command does, and descriptions of all the options.

The most important thing to know before you open `man` is how to get out. In the viewer, keys work differently from what you are used to. Press `q` (for *quit*), and the manual closes and returns you to the prompt. You can scroll with the up and down arrows and the `Page Up` and `Page Down` keys. You can also search the manual for a word: press `/`, type the word (for example, `sort`), and press `Enter`. The manual jumps to the first place the word appears, and the `n` key takes you to the next one. Learning `man` really means learning one letter: `q`. Searching with `/` is a bonus for when you feel more confident.

I will mention the third way only briefly, because it is not installed with the system. A community project called `tldr` (*too long; didn't read*) shows, instead of the whole manual, a few of the most common examples for each command with short explanations. It comes in handy when you want quick examples. You can install it once you learn to install packages in Chapter 10 (the package is also called `tldr`). For now, it is enough to know the tool exists.

> **Did you know?** `--help`, `man` and `tldr` answer different questions. `--help`: "What options are there?" `man`: "How exactly does it all work?" `tldr`: "How do I do this in practice?" Most experienced users reach for the first daily, the second now and then, and the third more and more often.

Command 4.5: history – what I have already asked

The last command in this chapter looks at the past. The shell remembers the commands you have typed. While the window is open, it keeps them in memory; when you close the window, Bash writes them to a hidden file in your home directory (`.bash_history`). That is why a new window also shows commands from earlier windows, up to a fairly large limit. Ask to see the list:

Command 4.5

```
history
```

What you do: ask to see the list of commands you have already typed. *Why:* so you can reuse an earlier command without retyping it. *What changes:* a numbered list appears on the screen. *Risk:* none.

You will see numbered lines with your commands: `pwd`, `ls`, `badcommand`, and the others you tried today. Even the mistake `badcommand` is there; the history records everything, not only what worked.

The arrow keys are even more practical than the list. The up arrow `↑` brings the last command you typed back to the prompt. Press `↑` once, and the last command appears; press it again, and you get the one before. The down arrow `↓` moves back toward the newest ones. When you find the one you need, press `Enter` to run it again. You can also edit it first: move to the right spot with the left and right arrows, change a letter, and press `Enter`.

This small habit will let you work faster than many users of the graphical desktop: you type a command once, and after that it is up arrow, maybe a small edit, and `Enter`.

> **In short.** The whole chapter in four lines: `pwd` – where I am, `ls` – what is here, `--help` / `man` – how it works, `history` and `↑` – what I have already done. Everything else is a variation on these four.

LAB-03: your first conversation with the system

A lab is a small, self-contained piece of practice. Take your time. Each step has an expected result; if yours is different, read the explanation before you move on.

Starting point. Linux Mint in the virtual machine, logged in as john, terminal not open. Snapshot taken (Chapter 2).

Steps.

1. Open the terminal with `Ctrl+Alt+T`. *Expected:* a dark window with the line `john@mint:~$` (your username and computer name may differ). If it did not open, open it from the menu search by typing `terminal`.
2. Ask where you are: type `pwd` and press `Enter`. *Expected:* `/home/john`, or `/home/` followed by your username. If you see a different path, carry on; the next chapter explores the path tree.
3. See what is here: type `ls`. *Expected:* a list of directories (`Documents`, `Downloads` and others). Then type `ls /` and find the word `home` in

the list. Found it? Good: you are looking at the same system, only as text.

4. Ask for more detail: type `ls -l`. *Expected:* a first line `total` with a number, then one line per directory with several columns. If you see only `total 0`, the directory is empty, which is also a correct result.
5. Make a mistake on purpose: type `badcommand`. *Expected:* `badcommand: command not found` and a new prompt. Then press `Ctrl+C`, and another clean prompt appears. See for yourself: the system works, and you broke nothing.
6. Ask for help: type `ls --help` and read the first two lines. Then type `man ls`, find the word **NAME**, scroll down a few lines with the down arrow, and leave with the `q` key. *Expected:* you are back at the prompt.
7. Look back: type `history` and find `pwd` in the list. Then press the `↑` arrow several times until `pwd` appears at the prompt again, and press `Enter`. *Expected:* you see `/home/john` again. You repeated the command without typing it.
8. Say hello to the system: type `echo "Hello, Linux!"` and press `Enter`. *Expected:* the line `Hello, Linux!`. The `echo` command prints whatever you give it in quotation marks. This is the first "sentence" the system repeats back to you word for word, and a good way to check that the terminal "hears" you.

Result. The terminal is open. You have run and repeated at least seven kinds of commands, made one mistake and survived it, opened and closed the help system, and looked through your history. You can close the terminal window. Your commands changed nothing in the system; only entries in the history remain.

What you can do now

- Open the terminal from the menu and with `Ctrl+Alt+T`.
- Read the prompt: who you are, which computer you are on, and where you are.

- Run commands, knowing that `Enter` is the "say it" button.
- Tell apart the command (`ls`), the option (`-l`, `--help`), and the argument (`/`), and recognize them in any line.
- Ask the system "where am I?" (`pwd`) and "what is here?" (`ls`).
- Read the error `command not found` and know that it breaks nothing.
- Stop a running command with `Ctrl+C`, and leave the manual with `q`.
- Look for help at three levels: `--help`, `man`, and later `tldr`.
- Go back to earlier commands with `history` and the ↑ arrow.

Exercises

1. Open the terminal and use `pwd`, `ls`, and `ls /` to answer three questions: where you are standing, what is in your home directory, and which directories are at the beginning of the system. Write the answers down on paper; you will compare them with Chapter 5.
2. Look through `man ls` and find the option that sorts the list in some way other than alphabetically (hint: search for the word "sort"). Leave with `q` and try the option you found with `ls`.
3. Type any five commands from this chapter, then use the ↑ arrow to bring each of them back and run it again. Count how many keystrokes you saved.
4. The `echo` command repeats text. Make the terminal print your name and today's date on one line. Then try the same thing without quotation marks and compare the result.

Before you move on

Question. You typed the command `lsz` and got an answer starting with `Command 'lsz' not found, did you mean:`, followed by a list of similar commands. What does this text mean, and what should you check first?

Answer. The system looked for the command `lsz` and did not find it. Nothing happened, and you broke nothing. The list after the message is

a set of hints Mint offers when a name resembles known commands; `ls` is among them. First check the spelling, including uppercase and lowercase letters: most likely you meant to type `ls`. If the spelling is correct, the program is not on your system and would need to be installed.