CYPHERPUNK SCHOOL — SOVEREIGNTY SERIES

The Command Line, From Zero

Linux CLI 101 — Your First 15 Commands

Never opened a terminal? By the end you'll move around, manage files, install software, and reach your server — and know you can't break it by looking

A practical guide to digital sovereignty

Cypherpunk School

cypherpunkschool.com

Sovereignty Series • Start Here: Foundation • June 2026

Contents

1. Why the Terminal — and Why You Can't Break It by Looking

If you've never opened a terminal, it can feel like the part of the computer where experts do dangerous things. Here's the truth: it's just a place where you *type* a command instead of *clicking* a button. That's the whole difference. And the everyday commands — looking at where you are, listing files, moving around — are about as dangerous as opening a folder in a file browser. You will not break your computer by exploring.

Almost every guide in this series, and most real privacy and self-hosting tools, expect you to type a few commands. Not because anyone's showing off — because the command line is where you get *direct* control. A graphical app shows you the three buttons its designer chose; the terminal gives you the actual machine. That directness is exactly why sovereignty lives here: nothing hidden, nothing decided for you.

> **The promise of this guide.** You don't need to "learn Linux." You need about **15 commands** — and they're all in here. By the end you'll move around the system, look at and manage files, install software, edit a file, and connect to a server. That's everything the rest of the series asks of you.

2. Opening a Terminal & Reading the Prompt

On most Linux desktops, press `Ctrl+Alt+T`, or search your apps for "Terminal." (On a Mac it's "Terminal" too; on Windows, install "WSL" or use the terminal on your Linux server.)

You'll see something like this — the *prompt*:

```
alice@server:~$
```

It's telling you four things: `alice` (your username), `server` (the machine's name), `~` (where you are — `~` means your home folder), and `$` ("ready for a command"). You type after the `$` and press Enter. That's it.

3. Where Am I? Moving Around

Three commands cover all movement:

```
pwd             # "print working directory" - where am I right now?
ls              # list what's in this folder
cd Documents    # change into the Documents folder
cd ..           # go up one level (.. means "the folder above")
cd ~            # go home (~ is your home folder)
cd /            # go to the very top (the root of the system)
```

A *path* is just an address. `/home/alice/Documents` is an absolute address (starts at `/`, the top). `Documents` on its own is relative to wherever you are now. That's the only "concept" here.

4. Looking at Things

```
ls -la         # list EVERYTHING, with details (permissions, size, dates)
cat notes.txt  # dump a file's contents to the screen
less notes.txt # view a long file scrollably (press q to quit)
```

`ls -la` is the one you'll use constantly — the `-l` gives details, the `-a` shows "hidden" files (names starting with a dot). Those flags after a command are how you adjust what it does.

5. Making & Changing Things

```
mkdir myfolder        # make a directory (folder)
touch file.txt        # create an empty file
cp file.txt copy.txt  # copy
mv file.txt docs/     # move (also how you RENAME: mv old.txt new.txt)
rm copy.txt           # remove a file
rm -r myfolder        # remove a folder and everything in it
```

> **`rm` does not ask, and there is no Recycle Bin.** When you remove something, it's gone. Read the line before you press Enter — especially with `rm -r` (recursive). Never run `rm -rf /` or `rm -rf` on a path you're not certain of; that pattern, with `sudo` in front, is the classic way people erase a system. Slow down on `rm`; everything else here is safe.

6. Permissions & Power

Linux files carry permissions — who can **r**ead, **w**rite, and e**x**ecute them. In `ls -la` you'll see strings like `rwxr-xr-x`. You rarely need to think about them until a guide tells you to make a file private or executable:

```
chmod 600 secret.txt   # only YOU can read/write it (good for key files)
chmod +x script.sh     # make a script runnable
```

And the big one: `sudo`. It means "do this as the administrator (root)." You'll prefix system changes with it (installing software, editing system files). It will ask for your password.

> **`sudo` is the power tool.** A command run with `sudo` can change *anything* on the machine. That's fine — you'll need it — but it's the moment to read the command twice. "With `sudo`" is the difference between a harmless mistake and a real one.

7. Installing Software

On Ubuntu/Debian systems, software comes from a package manager. Two commands get you almost everything:

```
sudo apt update           # refresh the list of available software
sudo apt install htop     # install a program (here, a system monitor)
```

Run `update` first (so it knows what's available), then `install` whatever the guide asks for. That's how every "install X" instruction works.

8. Editing a File

Sooner or later a guide says "edit this config file." The friendliest editor is `nano`:

```
nano config.txt
```

Type normally. The bottom of the screen shows the shortcuts: `Ctrl+O` then Enter to save ("write Out"), `Ctrl+X` to exit. That's enough to follow any guide.

> **Later, the power editors.** You'll hear about `vim` and `helix` — fast, keyboard-driven editors people love once they click. They have a learning curve, so they get their own short guides. For now, `nano` does everything you need; don't let anyone rush you off it.

9. Help & Speed

A few habits make the terminal feel easy instead of unforgiving:

- **Tab to autocomplete.** Start typing a file or command and press `Tab` — it finishes it for you. Saves typing and prevents typos.
- **Up-arrow for history.** Press ↑ to bring back commands you already ran. You rarely retype anything.
- **`Ctrl+C` stops it.** If something's running and you want out, `Ctrl+C` cancels it. Your escape hatch.
- **Get help.** `man ls` opens the manual for a command (`q` to quit); most commands also accept `--help` for a quick summary.
- **`clear`** wipes the screen when it gets cluttered.

> **Later, faster lookups.** You'll meet example-first help tools like `tldr` and `cheat.sh` in Part 3 — they're great once you know your way around. Learn `man` and `--help` first, though; they teach you the real thing, and they're always there even on a fresh server with no extras installed.

10. Reaching Your Server (SSH)

Most of what you'll self-host runs on a *headless* machine — a server with no monitor. You control it from your own computer over **SSH** (secure shell): an encrypted connection to its command line.

```
ssh alice@192.168.1.50      # connect as user "alice" to that address
```

It'll ask to confirm the server's identity the first time (type `yes`) and then for your password. Once connected, the prompt changes to the server's — and every command above works exactly the same, just on that machine. Type `exit` to disconnect.

> **This is the bridge.** SSH is what turns "commands on my laptop" into "running my own private services on a box in the corner." It's also the gate for the practice playground below — so if you can SSH, you're ready for the real thing.

11. You Can't Break It by Looking — the Only Two Real Dangers

Exploring is safe. Looking, listing, moving around, reading files — none of it harms anything. Only two things deserve real caution, and you now know both:

1. **`rm`** (especially `rm -r`) — it deletes permanently, no undo.
2. **`sudo`** — it acts with full power over the machine.

When a line has either of those, read it once more before Enter. Everything else, poke around freely. That's genuinely how everyone learns this.

12. Your First 15 Commands (keep this handy)

Command	What it does
`pwd`	Where am I?
`ls` / `ls -la`	List files / list all with details
`cd`	Change folder (`cd ..` up, `cd ~` home)
`cat` / `less`	Show a file / scroll a long file (`q` quits)
`mkdir`	Make a folder
`cp` / `mv`	Copy / move (or rename)
`rm`	Remove — careful, no undo
`chmod`	Change permissions
`sudo`	Run as administrator — careful
`apt update && apt install`	Refresh + install software
`nano`	Edit a file
`man` / `--help`	Get help on a command
`ssh`	Connect to a server
`Tab` / ↑ / `Ctrl+C`	Autocomplete / history / stop

13. Now Go Practice

Reading commands isn't the same as using them — the muscle memory comes from doing. The best free playground is **OverTheWire: Bandit** (overthewire.org/wargames/bandit), a gentle game where each level teaches a command by making you use it to find the password to the next. Level 0 is literally "SSH into this server" — which you can now do.

> **You're past the hardest part.** The barrier was never the commands — it was the fear of the blank black screen. You've crossed it. From here, every guide in this series is just typing a few of these instead of clicking. Welcome to the command line; it's yours now.

Continue the track

Part 2 — Streams, Pipes & the `-` Convention: the mental model that makes dense one-liners readable. Read it at

cypherpunkschool.com/guides/cli-102

More guides on self-hosting, privacy, local AI, and digital sovereignty:

cypherpunkschool.com
