



The Linux Command Line Handbook

52 commands · 8 errors · 35 how-tos · 6 comparisons

```
user@linux:~$ man everything
Building the mental model...
user@linux:~$ _
```

The Linux Command Line Handbook

Foundations, internals and real-world practice for the modern terminal

shell.trainer press

The Linux Command Line Handbook — 2026 edition.

Copyright © 2026 shell.trainer. All rights reserved. This book is sold DRM-free for personal use; please do not redistribute — every copy sold funds new chapters.

Companion website with interactive practice terminals: shell.trainer. Errata and updates are delivered free to all purchasers.

Linux® is the registered trademark of Linus Torvalds. All other trademarks belong to their owners.

Contents

Foundations	8
The Shell: How It Thinks	9
Files, Inodes and Names	10
The Permission Model	11
Composition: Pipes, Redirection and Text	12
The Command Reference	13
ls — List the contents of a directory, with options for...	14
cd — Move between directories: absolute and relative paths,...	17
pwd — Show the absolute path of the directory you are...	20
mkdir — Create one or many directories, including whole nested...	22
touch — Create empty files instantly, or update the timestamps...	25
rm — Delete files and directories — and understand -r, -f...	27
cp — Copy files and whole directory trees, preserving...	30
mv — Move files between directories and rename them — the...	33
cat — Print file contents, number lines, and join multiple...	36
grep — Find lines matching a pattern in files or piped input —...	39
find — Locate files anywhere in a directory tree by name,...	42
chmod — Change file permissions with numeric (755, 644) and...	45
echo — Print text and variables, and write or append to files...	48
head & tail — Print the first or last lines of a file — including...	51
tar — Create and extract archives: tar.gz, tar.bz2 — with the...	54
ssh — Log into remote machines, run remote commands, use keys...	57
curl — Download files, test APIs, send POST requests with...	60
wget — Download files and whole sites, resume interrupted...	63
ps — See what is running, find a process ID, and combine...	66
kill — Terminate processes politely or forcibly — and...	69
df & du — See free space per filesystem and find what is eating...	72
ln — Make one file reachable from several paths — and...	75
sed — Stream-edit text: substitute, delete lines, and edit...	78
awk — Pull columns out of any output, filter rows, and...	81
cut — Slice fields or character ranges out of each line —...	84
xargs — Turn a list of items into command arguments — the glue...	87

chown — Assign files to users and groups — the other half of...	90
history — List, search and re-run previous commands — plus...	93
ping — Check whether a host is reachable and measure...	96
ip — The modern tool for addresses, links and routes —...	99
ss — See listening ports, active connections and which...	102
dig — Ask DNS questions directly: A, MX, TXT records, and...	105
rsync — Sync directories locally or over SSH, transferring only...	108
scp — The quickest way to move a file to or from a remote...	111
top & htop — The live dashboard: CPU, memory, load average and the...	114
free — RAM and swap at a glance — and how to read "available"...	117
systemctl — Start, stop, enable and inspect services — the control...	119
journalctl — Query the system journal: by service, time window,...	122
sudo & su — Temporary privilege, audited — and why sudo won't...	125
apt — Search, install, upgrade and remove packages — plus the...	128
crontab — Run commands on a schedule — and read the five-field...	131
tee — Split a stream: watch output live while saving a copy —...	134
tr — Character-level transforms in pipelines: case, delete,...	136
uniq — Deduplicate adjacent lines and count occurrences —...	138
diff — See exactly what changed between two files or trees —...	140
less — The standard pager: scroll, search, jump — and follow...	142
gzip & zip — Single-file compression (gzip family) vs portable...	144
env & export — How configuration flows into processes — PATH, export,...	147
alias — Name your habits: shortcuts, safer defaults, and where...	150
jobs, & and nohup — Run things in the background, juggle them, and keep...	152
sha256sum & md5sum — Fingerprint files, verify downloads, and prove two...	155
which & type — Which binary actually runs when you type a name — and...	158

When Things Break 160

Permission denied (publickey)	161
bash: command: command not found	164
No such file or directory	167
Permission denied	170
No space left on device	173
bind: Address already in use	176
Connection refused	178

bad interpreter: No such file or directory	180
Getting Things Done	182
Check disk space in Linux	183
Extract a .tar.gz archive	185
Free a port that is already in use	187
Find and replace text in files	189
Make a script executable	191
Search text inside files	193
Check memory usage	195
Copy files to or from a remote server	197
Search recursively with grep	199
Search with grep ignoring case	201
Search for multiple patterns with grep	203
Set a file to permission 755	205
Set a file to permission 644	207
Run a script in Linux	208
Connect to a Linux server with SSH	210
Find the process using a port	212
Find large files in Linux	213
Find files by name in Linux	214
Stop a process in Linux	216
Find a process ID in Linux	218
Search Bash command history	220
List hidden files in Linux	222
Copy a directory in Linux	224
Move files and directories in Linux	226
Delete a directory in Linux	228
Create nested directories in Linux	230
Count lines in a Linux file	232
View the end of a file	234
Watch a Linux log file in real time	236
List running processes in Linux	238
Stop a process by name	240
Download a file with curl	242

Download a file with wget	244
Rename a file in Linux	246
Identify a file type in Linux	248
Choosing Between Tools	250
rsync vs scp	251
curl vs wget	254
sed vs awk	257
symbolic link vs hard link	260
grep vs ripgrep (rg)	263
tar.gz vs zip	266

PART I

Foundations

Four short chapters on how the shell thinks. Everything in Part II rests on these ideas; read them once and hundreds of behaviors stop being arbitrary.

CHAPTER 1

The Shell: How It Thinks

The terminal is not a place where commands run — it is a conversation with a program called the shell, and the shell does far more than relay your words.

When you press Enter, the shell transforms your line before any command sees it. Variables are substituted, \$(command) constructs are executed and replaced by their output, quotes are interpreted and removed, and wildcards are expanded against the filesystem. Only then does the shell locate the program and start it, handing over the finished argument list. The command never sees a dollar sign or an asterisk — it sees results. Internalize this order of operations and an entire class of mysteries dissolves: why *.txt behaves differently with and without quotes, why a variable with spaces splits into several arguments, why echo can be used to preview any expansion.

The second pillar is the process model. Every command runs as a child process, inheriting three things at birth: the working directory, the environment variables, and three open streams. Inheritance flows strictly downward — a child can change nothing about its parent, which is why cd must be built into the shell and why a script cannot alter the environment of the terminal that ran it.

The three streams

Each process is born with standard input (0), standard output (1) and standard error (2). By default they point at your keyboard and screen, but the shell can rewire them before launching the program — to files, to other processes, to nowhere. The program neither knows nor cares: it reads 0 and writes 1 and 2 wherever they lead. All of redirection and piping is exactly this rewiring, performed by the shell, invisible to the command.

```
$ command > out.txt      # stream 1 -> file
$ command 2> err.txt     # stream 2 -> file
$ producer | consumer   # 1 of producer -> 0 of consumer
```

The last pillar is the exit code: every process ends by reporting a number, 0 for success, anything else for failure. The shell exposes it as \$?, and the operators && and || branch on it. This tiny convention is what lets independent programs coordinate — pipelines, scripts and CI systems are all built on processes telling each other, in one byte, how it went.

CHAPTER 2

Files, Inodes and Names

Half of Linux behavior becomes obvious the moment you learn one fact: the name of a file is not the file.

A file is an inode: a structure holding the metadata (permissions, owner, size, timestamps) and the pointers to the data blocks. Directories are files too — their content is simply a table mapping names to inode numbers. A filename is one entry in one directory pointing at an inode, and the inode keeps count of how many names point at it.

Read the standard operations through this lens. Deleting (`rm`) removes a name and decrements the count — data lives until the count is zero and nobody holds the file open. Moving within a filesystem (`mv`) rewrites a directory entry, touching no data, atomically. A hard link (`ln`) adds a second, equal name. A symbolic link is a small file containing a path, resolved on every access — breakable, but able to cross filesystems and point at directories. Copying (`cp`) is the odd one out: it creates a genuinely new inode and duplicates the data.

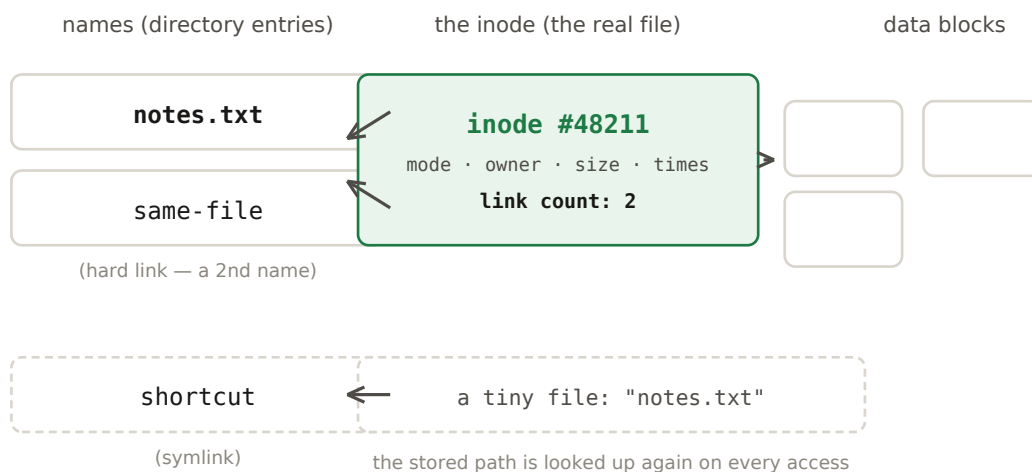


Figure 1 — Names, inodes and data: hard links are equal names for one inode; a symlink is a tiny file holding a path.

Why permissions live where they do

Permissions are inode metadata, which explains the two rules that surprise everyone. Deleting a file needs write permission on the directory (you are editing the directory's table), not on the file. And execute on a directory does not mean "run" — it means permission to resolve names through it, which is why a directory without `x` is sealed regardless of what is inside. Chapter 3 builds the full model.

```
$ ls -li notes.txt           # -i shows the inode number
$ ln notes.txt same-file    # second name, same inode
$ ln -s notes.txt shortcut  # small file containing a path
```

CHAPTER 3

The Permission Model

Nine bits, three special ones, and two identity numbers — the entire Linux access model fits in one page, and every 403 and "Permission denied" is explained by it.

Every inode stores an owner (UID), a group (GID) and a mode. The mode's nine main bits grant read, write and execute to three audiences: the owner, the group, and everyone else. When a process touches a file, the kernel picks exactly one audience — owner if UIDs match, else group, else other — and checks that audience's three bits. The octal notation is these bits in base 8: r=4, w=2, x=1, so 755 is rwx r-x r-x and 644 is rw- r-- r--.

For directories the letters shift meaning: r lists names, w creates and deletes entries, x traverses. Three special bits complete the model: setuid runs an executable with its owner's privileges (how passwd edits a root-owned file for you); setgid on a directory makes new files inherit the directory's group (the backbone of shared folders); the sticky bit on world-writable directories such as /tmp restricts deletion to each file's owner.

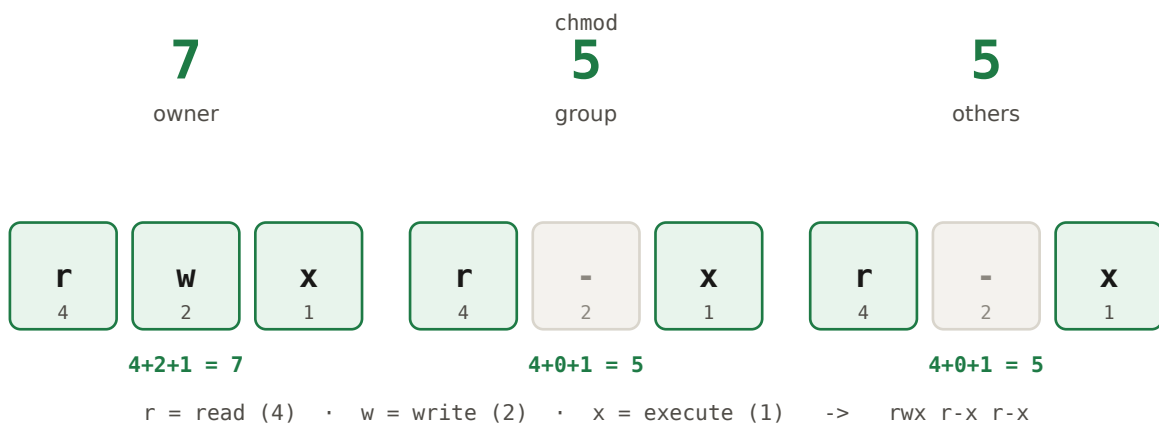


Figure 2 — How chmod 755 decomposes: one digit per audience, each the sum of its bits.

The recipes that follow from the model

```
$ chmod 700 ~/.ssh && chmod 600 ~/.ssh/id_ed25519 # SSH requires privacy
$ find . -type d -exec chmod 755 {} + # dirs: traversable
$ find . -type f -exec chmod 644 {} + # files: not executable
$ sudo chown -R www-data: /var/www/app/storage # writable by the service
```

And the anti-recipe: 777 "fixes" everything by abolishing protection — on any shared or networked machine it is an open door. There is always a narrower mode that expresses what you actually need; finding it is the skill this chapter exists to teach.

CHAPTER 4

Composition: Pipes, Redirection and Text

Unix bet that if every tool reads and writes plain text streams, tools never designed together could still cooperate. The bet paid off; this chapter is how to collect.

Redirection points a stream at a file: `>` writes (truncating first — the shell empties the target before the command runs, which is why redirecting a file onto itself destroys it), `>>` appends, `2>` captures errors, and `2>&1` merges errors into wherever stdout currently points. `/dev/null` discards. These five spellings cover practically all real usage.

A pipe connects stdout of one process to stdin of the next through a kernel buffer. Both processes run simultaneously; data flows as it is produced; nothing touches disk. Pipelines therefore read left to right as assembly lines, each stage a specialist: producers (cat, find, ps, curl), filters (grep, sed, awk, sort, uniq, cut), and consumers (head, wc, xargs, tee).

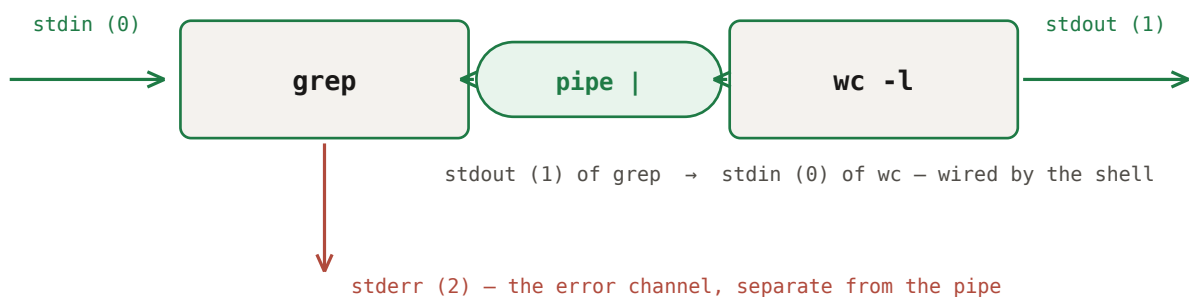


Figure 3 — A two-stage pipeline: the shell wires stdout of one process to stdin of the next; errors travel on their own channel.

Reading a real pipeline

```
$ awk '{print $1}' access.log | sort | uniq -c | sort -nr | head
#  extract IPs    -> group      -> count    -> rank      -> top 10
```

Each stage is trivial; the composition answers a question no single tool has a flag for. This is the pattern to internalize: when a task feels like it needs a program, try spelling it as a pipeline first. The reference chapters of Part II describe each specialist with exactly this composability in mind — the "pipes" column of your mental toolbox matters more than any individual flag.

PART II

The Command Reference

52 commands, one chapter each: what it is for, how it works underneath, the options that matter, worked examples, real-world scenarios, professional tips and the questions everyone asks.

CHAPTER 5

ls

List the contents of a directory, with options for hidden files, long format, sorting and more.

IN PLAIN ENGLISH

Imagine walking into a room and switching on the light: `ls` is that switch. You're not touching anything or opening any drawer — you're just seeing what's there. It will be the command you type most in your life, usually on pure reflex, just to orient yourself before doing anything else.

The `ls` command lists the contents of a directory. It is almost certainly the command you will run most often in a Linux terminal: before you move, copy, delete or open anything, you usually run `ls` first to see what is there. By default it lists the current directory in alphabetical order, hiding any file whose name starts with a dot.

How `ls` works

A directory in Linux is itself a file — one whose contents are a list of name-to-inode pairs. When you run `ls`, it opens that directory file and reads out the names. That is all a plain `ls` does, which is why it is instantaneous even in a directory of a million entries: it never touches the files themselves.

The long listing is a different animal. Sizes, permissions, owners and dates do not live in the directory — they live in each file's inode. So `ls -l` must call `stat()` once per entry to fetch that metadata, which is why `-l` is visibly slower on huge or network-mounted directories. One more thing worth internalizing: when you type `ls *.txt`, the shell — not `ls` — expands the wildcard and passes the resulting names as arguments. `ls` never sees the asterisk. This "the shell expands globs" rule explains dozens of otherwise mysterious behaviors across every command.

Syntax

```
ls [OPTIONS] [PATH...]
```

The options that matter

-l	Long format: permissions, owner, group, size and modification date of each entry.
-a	Show all files, including hidden ones (names starting with a dot, like <code>.bashrc</code>).
-h	With <code>-l</code> , print sizes in human-readable units (K, M, G) instead of bytes.
-t	Sort by modification time, newest first.
-r	Reverse the sort order.
-R	List subdirectories recursively.
-S	Sort by file size, largest first.
-d	List the directory itself instead of its contents (useful with wildcards).

Worked examples

```
$ ls
```

List the current directory. Directories are usually shown in a different color.

```
$ ls -l
```

Long listing. The first column shows the type and permissions: `d` means directory, `-` means regular file.

```
$ ls -la
```

Long listing including hidden files. This is one of the most common option combinations in daily use.

```
$ ls -lh
```

Long listing with human-readable sizes: 4.0K, 12M, 1.2G instead of raw byte counts.

```
$ ls -lt
```

Sort by modification time so the most recently changed files appear at the top. Add `-r` (`ls -ltr`) to put the newest at the bottom, right above your prompt.

```
$ ls /var/log
```

You can list any path without moving to it first.

```
$ ls -d */
```

Show only the directories in the current location.

```
$ ls *.txt
```

Use wildcards to list only files matching a pattern.

In the real world

Auditing a server directory

You SSH into a server and need to know what changed recently: `ls -lt` shows files by modification time, newest first, and `ls -lah` adds hidden files and readable sizes. Within seconds you know what was touched, when, and how big it is — before opening a single file.

Checking permissions before a deploy

A web deploy fails with 403 errors. `ls -l` on the web root instantly shows whether files are readable (644) and directories enterable (755), and who owns them. Most permission incidents are diagnosed with `ls -l` alone, before `chmod` is ever needed.

Professional habits and common mistakes

- > Make `ls -lah` your reflex instead of plain `ls` on servers — hidden config files (`.env`, `.htaccess`) are exactly the ones that cause surprises.
- > `ls -ltr` (with `r` for reverse) puts the newest files at the bottom, right above your prompt — no scrolling up on long listings.
- > The first character of `ls -l` output is the file type: `-` file, `d` directory, `l` symlink. The arrow after a symlink name shows where it points.
- > Counting entries? `ls | wc -l` is quick, but remember it counts lines, so filenames with newlines (rare but legal) will miscount.

Questions everyone asks

What does `ls -la` show?

`ls -la` combines two options: `-l` prints a detailed (long) listing with permissions, owner, size and date, and `-a` includes hidden files — those whose names begin with a dot, such as `.bashrc` or `.gitignore`.

How do I sort `ls` output by date?

Use `ls -lt` to sort by modification time with the newest first. Many users prefer `ls -ltr`, which reverses the order so the most recent files appear at the bottom, directly above the prompt.

Why are some files not shown by `ls`?

By default `ls` hides any file whose name starts with a dot. These are conventionally used for configuration. Run `ls -a` to include them.