Terminal Toolkit: 51 Essential Linux Commands

A Compact Reference for Everyday Use

Aman Arabzadeh

May 26, 2025

What is the CLI?
The Command Line Interface (CLI) is a powerful way to interact with your Linux system using text-based commands. Unlike graphical interfaces, the CLI gives you speed, precision, and full control. This guide covers 51 essential Linux commands to help you work faster, automate tasks, and understand your system at a deeper level.

Contents

Introduction

Welcome to my *Pocket Guide to Essential Linux Commands*.
I earned my Computer Engineering degree in 2024, and throughout both school and work, Linux commands have been a core part of my daily toolkit. They're fast, flexible, and let you get more done without relying on a mouse. This guide is built for beginners who want a solid starting point, and for experienced users who need a quick way to refresh their memory. Each command comes with a short description, practical examples, and tips to help you get things done faster.
At the end, there's a cheat sheet that pulls everything together in one place. Let's get started and take control of the command line.

CHAPTER 1

Navigation & File Information

This chapter covers the basic commands you need to move around in the Linux filesystem and learn about files. Each section shows:

- What the command does.
- How to use it (syntax and options).
- A simple example with expected output.

1.1 pwd

Syntax: `pwd`
What it does: Shows the *full path* of the directory you are currently in.

Why use it?

If you ever forget where you are in the filesystem, `pwd` tells you exactly which directory you are in.
Example

```
$ pwd
/home/user/projects
```

Here, you see you are in the ”projects” folder inside ”user”.
Tip

To get the parent directory (one level up), you can run:

```
$ dirname "$PWD"
/home/user
```

1.2 cd

Syntax: `cd [DIR]`
What it does: Changes your current directory to the one you specify.

Common Uses

- `cd /etc`: Go directly to the `/etc` folder.
- `cd ..`: Move one level up (to the parent directory).
- `cd -`: Jump back to the previous directory you were in.
- `cd`: With no arguments, go to your home directory (e.g. `/home/username`).

Example

```
$ cd /etc
$ pwd
/etc

$ cd ..
$ pwd
/
```

Tips

- You can use `Tab` for filename completion. Type part of the folder name, then press `Tab`.
- Use `cd ~/Downloads` to go to the Downloads folder in your home directory.

1.3 ls

Syntax: `ls [OPTIONS] [FILE or DIRECTORY]`
What it does: Lists the files and folders inside a directory.

Key Options

- `-l`: Long format (shows permissions, owner, size, and modification date).
- `-a`: Show *all* files, including hidden ones (those starting with a dot).
- `-h`: With `-l`, show sizes in human-readable form (KB, MB).

Examples

```
$ ls
Documents  Downloads  script.sh

$ ls -l
-rw-r--r-- 1 user user 1024 May 10 08:00 script.sh
 drwxr-xr-x 2 user user 4096 May 12 14:21 Documents

$ ls -lah
total 20K
drwxr-xr-x 4 user user 4.0K May 15 09:30 .
-rw-r--r-- 1 user user 1.0K May 10 08:00 script.sh
drwxr-xr-x 2 user user 4.0K May 12 14:21 Documents
```

Tip

Combine options: `ls -alh` shows everything with details and human-readable sizes.

1.4 tree

Syntax: `tree [OPTIONS] [DIRECTORY]`
What it does: Draws a visual tree of the directory and its subdirectories.

Why it's useful

Quickly see the folder structure and how files are organized.
Example

```
$ tree -L 2
.
          Documents
                report.docx
                notes.txt
          script.sh
```

Shows two levels deep (top folder and one level of subfolders).
Tip

Install if missing: `sudo apt install tree` (Debian/Ubuntu). Use `-d` to list directories only (skip files).

1.5 file

Syntax: `file [OPTIONS] FILENAME`
What it does: Examines a file and tells you its type (text, image, executable, etc.).

Example

```
$ file script.sh
script.sh: Bourne-Again shell script, ASCII text executable

$ file image.png
image.png: PNG image data, 800 x 600, 8-bit/color RGBA, non-interlaced
```

Tip

Use `-i` to show the MIME type instead of a description:

```
$ file -i script.sh
script.sh: text/x-shellscript; charset=us-ascii
```

CHAPTER 2

File Manipulation

This chapter covers how to copy, move, delete, and create files/directories safely.

2.1 cp

Syntax: `cp [OPTIONS] SOURCE... DESTINATION`
What it does: Copies files or directories from one place to another.

Common Options

- `-r`: Copy directories recursively (include all contents).
- `-i`: Prompt before overwrite.
- `-v`: Verbose mode, show each file copied.

Examples

```
$ cp file.txt /backup/
Copy "file.txt" into the /backup/ folder.

$ cp -r project/ /backup/project/
Copy the project folder and all its files into /backup/project/.
```

Tip

Use `cp -riv` to be safe and see progress.

2.2 mv

Syntax: `mv [OPTIONS] SOURCE... DESTINATION`
What it does: Moves or renames files and directories.

Common Options

- `-i`: Prompt before overwrite.
- `-n`: Do not overwrite existing files.
- `-v`: Show what is happening.

Examples

```
$ mv oldname.txt newname.txt
Rename a file.

$ mv file.txt /tmp/
Move file.txt into the /tmp directory.
```

Tip

Use `mv -iv` to avoid mistakes and see operations.

2.3 rm

Syntax: `rm [OPTIONS] FILE or DIRECTORY`
What it does: Deletes files or directories (be very careful!).

Common Options

- `-r`: Remove directories and their contents recursively.
- `-i`: Ask before each removal.
- `-f`: Force deletion without prompts.

Examples

```
$ rm file.txt
Delete a single file.

$ rm -r old_folder/
Delete a folder and everything in it.
```

Warning

Never run `rm -rf /` unless you know exactly what you're doing!

2.4 mkdir

Syntax: `mkdir [OPTIONS] DIRECTORY...`
What it does: Creates new directories (folders).

Common Options

- `-p`: Create parent directories as needed.
- `-v`: Show each directory as it is created.

Examples

```
$ mkdir newdir
Create a single folder named "newdir".

$ mkdir -p project/src/utils
Create project, src, and utils folders in one go.
```

2.5 rmdir

Syntax: `rmdir [OPTIONS] DIRECTORY...`
What it does: Removes empty directories only.

Example

```
$ rmdir olddir
Remove the folder "olddir" if it is empty.
```

Tip

To delete non-empty folders, use `rm -r` instead.

2.6 touch

Syntax: `touch [OPTIONS] FILE...`
What it does: Creates an empty file or updates the timestamp on existing files.

Example

```
$ touch notes.txt
Create an empty file named "notes.txt" or update its modification time.
```

Tip

Use `-c` to only change timestamps and not create files.

CHAPTER 3

Viewing & Editing Text

This chapter shows commands to view, navigate, and edit text files directly from the terminal.

3.1 cat

Syntax: `cat [OPTIONS] [FILE]...`
What it does: Joins files together and prints their contents to the screen.

Why use it?

Quick way to see small files or combine multiple files.
Example

```
$ cat file.txt
This is the first line.
This is the second line.
```

Tip

For long files, pipe through a pager:

```
$ cat file.txt | less
```

3.2 head

Syntax: `head [OPTIONS] [FILE]...`
What it does: Shows the first part of files (by default 10 lines).

Common Options

- `-n N`: Show the first N lines.
- `-c N`: Show the first N bytes.

Example

```
$ head -n 20 logfile.log
  [...] (first 20 lines) [...]
```

Tip

Use `head -c 200 file.txt` to see the first 200 bytes.

3.3 tail

Syntax: `tail [OPTIONS] [FILE]...`
What it does: Shows the last part of files (default 10 lines).

Common Options

- `-n N`: Show the last N lines.
- `-f`: Follow the file as it grows (live updates).

Example

```
$ tail -f /var/log/syslog
  (shows new log entries in real time)
```

Tip

Press `Ctrl+C` to stop following.

3.4 less

Syntax: `less [OPTIONS] [FILE]...`
What it does: Opens files in an interactive viewer, letting you scroll, search, and navigate.

Key Commands

- `Space`: Next page
- `b`: Previous page
- `/pattern`: Search forward for "pattern".
- `?pattern?: Search backward.`
- `q`: Quit less

Example

```
$ less README.md
```

Tip

Combine with pipes: `grep "TODO" *.txt | less` to search across files interactively.

3.5 nano

Syntax: `nano [OPTIONS] [FILE]...`
What it does: Opens a simple terminal-based text editor for creating and modifying files.

Basic Controls

- `Ctrl+O`: Save (Write Out)
- `Ctrl+X`: Exit
- `Ctrl+K`: Cut line
- `Ctrl+U`: Paste line

Example

```
$ nano script.sh
```

Opens `script.sh` (or creates it if missing).

Tip

Use `Ctrl+G` to view help within Nano.

CHAPTER 4

Searching & Filtering

This chapter shows how to find and narrow down text and files using powerful command-line tools.

4.1 grep

Syntax: `grep [OPTIONS] PATTERN [FILE]...`
What it does: Searches one or more files for lines that match a text pattern.

Why use it?

Quickly locate words or phrases inside files—even across many subdirectories.
Common options

- `-R`: Search recursively in all subfolders.
- `-i`: Ignore case (find "Error" or "error").
- `-n`: Show the line number of each match.
- `-C N`: Display N lines of context around each match.

Example

```
$ grep -Rni "TODO" .
./script.sh:12:# TODO: add error handling
./docs/readme.md:5:- [ ] TODO: update install steps
```

Tip

Use `grep -C 2 "pattern" file.txt` to see two lines before and after each hit.

4.2 find

Syntax: `find PATH [OPTIONS]`
What it does: Locates files or directories matching name, type, size, date, etc.

Why use it?

Search anywhere on your system with flexible filters.
Common options

- `-name PATTERN`: Match names (e.g. `"*.log"`).
- `-type f|d`: Only files (f) or directories (d).
- `-mtime -N`: Modified within the last N days.
- `-exec CMD {} \;`: Run a command on each matched item.

Example

```
$ find /var -name "*.log" -mtime -7
/var/log/syslog
/var/log/auth.log
```

Tip

Use `find . -type f -size +10M` to find files larger than 10 MB.

4.3 locate

Syntax: `locate [OPTIONS] PATTERN`
What it does: Quickly looks up filenames using a prebuilt database.

Why use it?

Much faster than `find`—ideal for routine filename searches.
Example

```
$ locate bashrc
/home/user/.bashrc
/etc/skel/.bashrc
```

Tip

Run `sudo updatedb` to refresh the database before searching.

4.4 wc

Syntax: `wc [OPTIONS] [FILE]...`
What it does: Counts lines, words, and bytes (or characters) in files.

Why use it?

Get quick stats—useful for checking script lengths or log sizes.
Common options

- `-l`: Count lines only.
- `-w`: Count words only.
- `-m`: Count characters.

Example

```
$ wc -l *.txt
     30 notes.txt
     15 summary.txt
     45 total
```

Tip

Pipe other commands into `wc`; e.g., `grep -c "error" *.log` counts error lines.

4.5 sort

Syntax: `sort [OPTIONS] [FILE]...`
What it does: Orders lines of text alphabetically or numerically.

Why use it?

Prepare data for reports, remove duplicates, or sort logs chronologically.
Common options

- `-n`: Numeric sort (e.g. numbers instead of text).
- `-r`: Reverse order.
- `-k N`: Sort by the Nth field (column).

Example

```
$ sort names.txt | uniq
user
Bob
Charlie
```

Tip

To reverse-sort numerically on column 2:
lstinline!sort -k2 -nr data.csv!.

CHAPTER 5

Shell & Environment

This chapter covers commands for printing messages, managing variables, and customizing your shell session.

5.1 echo

Syntax: `echo [OPTIONS] [STRING]...`
What it does: Prints text or the value of variables to your terminal.

Why use it?

Quickly display messages or debug variable contents in scripts and the shell.
Example

```
$ echo "Hello, world!"
Hello, world!

$ name="user"
$ echo "User: $name"
User: user
```

Tip

Use `-n` to omit the trailing newline: `echo -n "Loading..."` continues on the same line.

5.2 export

Syntax: `export NAME[=VALUE]`
What it does: Sets or updates an environment variable for this shell and sub-processes.

Why use it?

Make variables (like PATH or EDITOR) available to any program you launch from this shell.
Example

```
$ export PATH=$PATH:/opt/bin
$ echo $PATH
/usr/local/bin:...:/opt/bin
```

Tip

Put `export` lines in `~/.bashrc` (or `.bash_profile`) and then run `source ~/.bashrc` to apply them automatically.

5.3 env

Syntax: `env [OPTIONS] [COMMAND [ARG]...]`
What it does: Shows current environment variables or runs a command with a modified (or empty) environment.

Why use it?

Inspect your environment or launch programs without inherited variables.
Example

```
$ env | grep PATH
PATH=/usr/local/bin:...

$ env -i bash --noprofile --norc
# starts a new shell with no environment variables set
```

Tip

Use `env -i` to temporarily clear all variables and test behavior in a clean shell.

5.4 alias

Syntax: `alias NAME='COMMAND'`
What it does: Defines a shortcut name for a longer command or set of options.

Why use it?

Save typing and enforce your preferred flags (e.g., always use color with `ls`).
Example

```
$ alias ll='ls -lAh'
$ ll
drwxr-xr-x 2 user user 4.0K May  ...
```

Tip

To make aliases permanent, add them to `~/.bashrc` and reload with `source ~/.bashrc`.

5.5 history

Syntax: `history [OPTIONS]`
What it does: Lists the commands you've previously run in this shell session.

Why use it?

Quickly find and re-run recent commands without retyping.
Example

```
$ history 20
  981  ls
  982  cd projects
  983  git status

$ !982
# reruns command number 982 (cd projects)
```

Tip

Use $! (e.g. 983!) to rerun a specific entry, or to repeat the very last command.

5.6 source

Syntax: `source FILE`
What it does: Executes commands from FILE in the current shell (does not spawn a new process).

Why use it?

Reload your shell configuration (e.g., `.bashrc`) without closing the terminal.
Example

```
$ source ~/.bashrc
# applies any new aliases, exports, or functions immediately
```

Tip

The shorthand dot syntax `. ~/.bashrc` is equivalent to `source ~/.bashrc`.

CHAPTER 6

Permissions & Ownership

This chapter explains how to view and change who can access your files and how.

6.1 chmod

Syntax: `chmod [OPTIONS] MODE FILE...`
What it does: Updates the read/write/execute permissions on one or more files or directories.

Why use it?

Control who can read, write, or run your scripts and data files.
Modes

- Numeric (e.g. `755`): Owner rwx, Group r-x, Others r-x.
- Symbolic (e.g. `u+x`): Add execute for user.

Example

```
$ chmod 755 script.sh
# Owner can rwx, others can read and execute.
```

Tip

Use symbolic for fine-grained changes, e.g. `chmod u+w,g-w file.txt`

6.2 chown

Syntax: `chown [OPTIONS] OWNER[:GROUP] FILE...`
What it does: Changes the owner and/or group of files or directories.

Why use it?

Ensure the correct user or group controls access, especially in shared environments.
Example

```
$ sudo chown user:staff file.txt
# Now user owns the file, group is staff.
```

Tip

Add `-R` to apply changes recursively to a folder and its contents.

6.3 sudo

Syntax: `sudo [OPTIONS] COMMAND`
What it does: Runs a command with another user's privileges (root by default).

Why use it?

Perform administrative tasks (installing software, editing system configs) without logging in as root.
Example

```
$ sudo apt update
# Updates package lists as root.
```

Tip

Use `sudo visudo` to safely edit the sudoers file and avoid syntax errors.

6.4 umask

Syntax: `umask [OPTIONS] [MASK]`
What it does: Sets the default permission mask for new files and directories.

Why use it?

Define stricter or looser default permissions when you create new files/folders.
Example

```
$ umask 022
# New files: 644 (-rw-r--r--), directories: 755 (drwxr-xr-x)
```

Tip

Add your preferred `umask` line to `~/.bashrc` so it applies every session.

CHAPTER 7

Processes & Monitoring

This chapter covers how to view, manage, and prioritize running processes on your system.

7.1 ps

Syntax: `ps [OPTIONS]`
What it does: Shows a static snapshot of processes running in your shell or system.

Why use it?

Quickly see what commands are running, their PIDs, CPU/memory usage, and ownership.
Common options

- `aux`: All users, full-format listing.
- `-ef`: POSIX-style listing with parent–child relationships.

Example

```
$ ps aux
USER        PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
user       2345  0.1  1.2  22500  5120 pts/0    S+   09:10   0:00 bash
```

Tip

Pipe to `grep` to filter, e.g. `ps aux | grep apache2` finds all Apache processes.

7.2 top

Syntax: `top`
What it does: Opens an interactive, real-time view of system processes.

Why use it?

Monitor live CPU, memory, and process usage to spot resource hogs.
Example

```
$ top
```

Tip

Press `P` to sort by CPU, `M` to sort by memory, `q` to quit.

7.3 kill

Syntax: `kill [OPTIONS] PID...`
What it does: Sends a signal (by default SIGTERM) to one or more processes.

Why use it?

Gracefully or forcefully stop misbehaving processes by PID.
Common signals

- `-15`: SIGTERM (default, allows cleanup).
- `-9`: SIGKILL (immediate, cannot be caught).

Example

```
$ kill -9 1234
# Force-kill process 1234 if it  w o n t  terminate normally.
```

Tip

Try `kill PID` first (no `-9`), then escalate if the process ignores SIGTERM.

7.4 jobs

Syntax: `jobs [OPTIONS]`
What it does: Lists background and suspended jobs in your current shell.

Why use it?

Track processes you've put in the background or paused with Ctrl+Z.
Example

```
$ sleep 100 &
[1] 2345
$ jobs
[1]+  Running                 sleep 100 &
```

Tip

Use `fg %1` to bring job 1 to the foreground, or `bg %1` to resume in the background.

7.5 `nice` and `renice`

Syntax: `nice [OPTIONS] COMMAND`
What it does: Launches a command with an adjusted CPU scheduling priority.

Why use it?

Lower-priority (higher "nice" value) jobs yield CPU to more important tasks.
Example

```
$ nice -n 10 tar czf archive.tgz dir/
# Runs tar at a lower priority.
```

Tip

To change priority of an existing process, use `renice 5 -p PID` (requires appropriate permissions).

CHAPTER 8

Networking & Transfer

This chapter covers basic tools for testing connectivity, configuring interfaces, and securely transferring data.

8.1 ping

Syntax: `ping [OPTIONS] DESTINATION`
What it does: Sends ICMP "echo request" packets to check if a host is reachable.

Why use it?

Verify network connectivity and measure round-trip time to a server.
Common options

- `-c N`: Send N packets then stop.
- `-i SECONDS`: Wait SECONDS between each packet.
- `-W SECONDS`: Timeout in seconds for each reply.

Example

```
$ ping -c 4 example.com
PING example.com (93.184.216.34): 56 data bytes
64 bytes from 93.184.216.34: icmp_seq=0 ttl=56 time=10.2 ms
...
--- example.com ping statistics ---
4 packets transmitted, 4 received, 0% packet loss
```

Tip

Use `ping -i 0.5` for half-second intervals if you need quicker feedback.

8.2 ip

Syntax: `ip [OPTIONS] OBJECT COMMAND`
What it does: Displays or modifies network interfaces, routes, and addresses.

Why use it?

Modern replacement for `ifconfig` and `route`, with more features.
Common usages

- `ip addr show`: List all interfaces and their IPs.
- `ip link set eth0 up`: Enable interface `eth0`.
- `ip route show`: Display routing table.

Example

```
$ ip addr show
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500
    inet 192.168.1.10/24 brd 192.168.1.255 scope global eth0
```

Tip

Use `ip -br addr` for a concise, "brief" summary of interfaces.

8.3 ssh

Syntax: `ssh [OPTIONS] [USER@]HOST`
What it does: Opens a secure shell session on a remote machine.

Why use it?

Securely log in to remote servers, forward ports, and run commands.
Common options

- `-i KEYFILE`: Specify your private key file.
- `-p PORT`: Connect to a non-standard SSH port.
- `-L local:remote`: Forward local port to remote.

Example

```
$ ssh -i ~/.ssh/id_rsa user@server.example.com
Last login: Fri May  ... on pts/0
user@server:~$
```

Tip

Add hosts and keys to `~/.ssh/config` for shorter commands.

8.4 scp

Syntax: `scp [OPTIONS] SOURCE [USER@]HOST:DEST`
What it does: Securely copies files or directories over SSH.

Why use it?

Transfer files between local and remote machines without setting up FTP.
Common options

- `-r`: Recursively copy directories.
- `-P PORT`: Specify SSH port for the remote host.
- `-C`: Enable compression during transfer.

Example

```
$ scp -r project/ user@host:/var/www/
# Copies the local project folder to /var/www/ on the remote host.
```

Tip

Use `scp -C` to speed up transfers over slow links by compressing data.

8.5 curl

Syntax: `curl [OPTIONS] URL`
What it does: Transfers data to or from a server using various protocols (HTTP, FTP, etc.).

Why use it?

Download files, test APIs, or upload data from the command line.
Common options

- `-O`: Save with the remote filename.
- `-o FILE`: Save output to a specific file.
- `-I`: Fetch and display only HTTP headers.
- `-L`: Follow redirects.

Example

```
$ curl -L -o latest.tar.gz https://example.com/downloads/app.tar.gz
```

Tip

Use `curl -I URL` to quickly check response headers without downloading the body.

CHAPTER 9

Archiving & Compression

This chapter covers how to bundle files into archives and compress or decompress them efficiently.

9.1 tar

Syntax: `tar [OPTIONS] ARCHIVE [FILES\,...]`
What it does: Creates or extracts tar archives, optionally with compression.

Why use it?

Bundle a directory tree into a single file, with optional gzip or bzip2 compression.
Common options

- `c`: Create a new archive.
- `x`: Extract files from an archive.
- `z`: Use gzip compression.
- `j`: Use bzip2 compression.
- `v`: Verbose—show files processed.
- `f FILE`: Specify archive filename.

Examples

```
$ tar czf archive.tgz project/
# Create gzip-compressed archive.tgz from project/ directory

$ tar xzf archive.tgz
# Extract files from archive.tgz into the current directory
```

Tip

For bzip2 compression (smaller but slower): `tar cjf archive.tar.bz2 project/`

9.2 gzip & gunzip

Syntax: `gzip [OPTIONS] FILE...` / `gunzip [OPTIONS] FILE...`
What it does: Compresses single files to .gz and decompresses them.

Why use it?

Quickly compress or decompress individual files.
Common options

- `-k`: Keep original file after compression/decompression.
- `-c`: Write output to stdout (useful in pipes).
- `-d`: Decompress (same as gunzip).

Examples

```
$ gzip file.txt
# Compresses to file.txt.gz and removes original file.txt

$ gunzip -k file.txt.gz
# Decompresses to file.txt and keeps file.txt.gz
```

Tip

Combine with tar: `tar cf - project/ | gzip -c > project.tar.gz`

9.3 zip & unzip

Syntax: `zip [OPTIONS] ARCHIVE FILE...` / `unzip [OPTIONS] ARCHIVE`
What it does: Creates or extracts ZIP archives (common in Windows).

Why use it?

Interoperate with Windows users or tools that prefer .zip format.
Common options

- `-r`: Recursively include directories.
- `-e`: Encrypt archive (prompt for password).
- `-l`: List contents of a ZIP file without extracting.

Examples

```
$ zip -r project.zip project/
# Create project.zip containing the project/ folder

$ unzip -l project.zip
# List contents without extracting

$ unzip project.zip
# Extract all files into the current directory
```

Tip

To unzip to a specific folder: `unzip project.zip -d /path/to/dest/`

CHAPTER 10

Disk Usage

This chapter explains how to check overall filesystem space and drill down into directory sizes.

10.1 df

Syntax: `df [OPTIONS] [FILE or MOUNTPOINT]...`
What it does: Shows total, used, and available space on mounted filesystems.

Why use it?

Quickly see which disks or partitions are running low on space.
Common options

- `-h`: Human-readable sizes (KB/MB/GB).
- `-T`: Display filesystem type.
- `-a`: Include all filesystems, even with 0 blocks.

Example

```
$ df -h
Filesystem      Size  Used Avail Use% Mounted on
/dev/sda1        50G   20G   28G  42% /
tmpfs           1.9G  4.0M  1.9G   1% /run
```

Tip

To check a specific mount point or directory, append its path: `df -h /home`

10.2 du

Syntax: `du [OPTIONS] [FILE or DIRECTORY]...`
What it does: Estimates disk usage of files and directories.

Why use it?

Identify which folders are consuming the most space.
Common options

- `-h`: Human-readable sizes.
- `-s`: Summarize total for each argument (no per-file breakdown).
- `-c`: Produce a grand total at the end.
- `-d N`: Limit depth to N levels of subdirectories.

Example

```
$ du -sh *
1.2G  Videos
  600M  Documents
  150M  Downloads
```

Tip

To see a sorted list of top space users, combine with `sort`: `du -h --max-depth=1 /var | sort -h`

CHAPTER 11

Comparing Files

This chapter shows how to spot differences between two files, whether by lines or raw bytes.

11.1 diff

Syntax: `diff [OPTIONS] FILE1 FILE2`
What it does: Compares two text files line by line and reports the changes needed to make them match.

Why use it?

Quickly review edits, patches, or version changes between text files or code.
Common options

- `-u`: Unified format (shows context lines).
- `-c`: Context format.
- `-i`: Ignore case differences.
- `-w`: Ignore all whitespace differences.

Example

```
$ diff -u old.txt new.txt
--- old.txt 2025-05-01 12:00:00
+++ new.txt 2025-05-02 08:30:00
@@ -1,3 +1,4 @@
 Line one
+Added this line
 Line two
-Line three
+Changed line three
```

Tip

Pipe output to `less` to scroll: `diff -u old.txt new.txt | less`

11.2 cmp

Syntax: `cmp [OPTIONS] FILE1 FILE2`
What it does: Compares two files byte by byte and reports the first difference or all differences.

Why use it?

Verify binary files are identical or pinpoint exact byte offsets where they differ.
Common options

- `-l`: List all differing byte positions and values.
- `-s`: Silent mode (no output, just return code).

Example

```
$ cmp -l file1.bin file2.bin
   12  101  102
   45  255  254
```

Tip

Use the exit status to script comparisons: `cmp -s file1 file2 && echo "Match" || echo "Differ"`

APPENDIX A

Cheat Sheet

Table A.1.: Common Linux Commands Cheat Sheet

Command	Key Flags	Purpose
pwd	—	Print the full path of the current directory.
cd	—	Change to a different directory.
ls	-l, -a, -h	List files with details, hidden files, human-readable sizes.
tree	-L, -d	Show directory structure as a tree (limit depth, directories only).
file	-i	Identify file type or MIME type.
cp	-r, -i, -v	Copy files/directories recursively, prompt, verbose.
mv	-i, -n, -v	Move/rename safely, no overwrite, verbose.
rm	-r, -f, -i	Remove recursively, force, or prompt.
mkdir	-p, -v	Create directories with parents, verbose.
rmdir	—	Remove empty directories only.
touch	-c	Create or update file timestamps without creating new.
cat	—	Concatenate and display file contents.
head	-n, -c	Show first N lines or bytes of a file.
tail	-n, -f	Show last N lines or follow file updates.
less	—	View and search through files interactively.
nano	—	Simple terminal text editor.
grep	-R, -i, -n, -C	Search text patterns recursively, ignore case, show line numbers, context.
find	-name, -type, -mtime, -exec	Locate files by name, type, date; run commands on results.
locate	—	Fast filename lookup using database.
wc	-l, -w, -m	Count lines, words, and characters.
sort	-n, -r, -k	Sort numerically, reverse, by column.

Continued on next page

Continued from previous page

Command	Key Flags	Purpose
`echo`	`-n`	Display text or variable values without newline.
`export`	—	Set environment variables for child processes.
`env`	`-i`	Show or run command in a clean environment.
`alias`	—	Define command shortcuts in shell.
`history`	—	List and reuse past commands.
`source`	—	Execute a script in the current shell session.
`chmod`	`-R, u+x, g-w`	Change file permissions recursively or symbolically.
`chown`	`-R`	Change file owner and group recursively.
`sudo`	—	Run commands with elevated (root) privileges.
`umask`	—	Set default permission mask for new files.
`ps`	`aux, -ef`	Show snapshot of running processes (detailed, POSIX).
`top`	—	Real-time interactive process viewer.
`kill`	`-15, -9`	Send terminate or kill signals to processes.
`jobs`	—	List background and suspended jobs in shell.
`nice`	`-n`	Start a process with modified CPU priority.
`renice`	`-n, -p`	Adjust priority of running processes.
`ping`	`-c, -i, -W`	Send ICMP echo requests; count, interval, timeout.
`ip`	`addr, link, route, -br`	Manage interfaces, addresses, routes; brief output.
`ssh`	`-i, -p, -L`	Secure shell: keyfile, port, port forwarding.
`scp`	`-r, -P, -C`	Secure copy: recursive, port, compression.
`curl`	`-O, -o, -I, -L`	Transfer data: save files, headers only, follow redirects.
`tar`	`c, x, z, j, v, f`	Create/extract archives with gzip/bzip2, verbose, specify file.
`gzip`	`-k, -c, -d`	Compress or decompress .gz files, keep originals, output to stdout.
`unzip`	`-l, -d`	List or extract ZIP archives to specified directory.
`df`	`-h, -T, -a`	Report filesystem disk usage, human-readable, type, all.
`du`	`-h, -s, -c, -d`	Estimate directory sizes, human-readable, summary, total, depth.
`diff`	`-u, -c, -i, -w`	Compare text files with context, ignore case/whitespace.
`cmp`	`-l, -s`	Compare files byte by byte, list offsets, silent mode.

References & Sources

- GNU Coreutils Manual — https://www.gnu.org/software/coreutils/manual/coreutils.html
- Linux Man Pages — https://man7.org/linux/man-pages/
- TLDP Advanced Bash Guide — https://tldp.org/LDP/abs/html/
- Stack Overflow — https://stackoverflow.com
- Personal experience and experimentation