Linux Command Line Survival Guide

Linux becomes usable when you can do three things without guessing:

- know where you are
- move to where you need to be
- read what is there

This guide covers essential Linux commands for survival. It contains what you need for the Linux Basics lab and for basic system navigation in any Linux environment.

Nothing here is decorative. Everything here will be used.

The terminal

A terminal is the window (or app) that shows text input and output.

It is not Linux itself. It is the interface that lets you send input to programs and see their output.

If the terminal closes, programs tied to it often stop.

The shell

A shell is the program running inside the terminal that reads what you type, runs commands, and prints results.

Your prompt (often ending in `$`) is the shell telling you it is ready.

This matters because most Linux work is done by asking the shell to:

- run a program
- read or write a file
- show you the current state of the system

If you understand what the shell is doing, Linux stops feeling mysterious.

Filesystem

Linux uses a single, unified filesystem tree starting at the root directory (`/`). There are no drive letters. Extra disks and partitions appear by being mounted somewhere inside this tree.

Think of the filesystem as the map of where things live.

Paths

A path tells Linux where something is in the filesystem.

Helpful shortcuts:

Symbol	Meaning	Practical use
`/`	Root	Jump to a system folder: `cd /etc`
`~`	Home	Go to your home directory: `cd ~`
`.`	Current directory	Refer to “right here”
`..`	Parent directory	One level up: `cd ..`
`-`	Previous directory	Toggle back: `cd -`

Absolute paths start at / and are unambiguous.
Relative paths start from the current working directory of the shell.

Current working directory

Your shell is always “standing in” one directory at a time. That directory is your current working directory (cwd).

It is the path in the shell that all relative paths are based on.

Key rule:

> ***Every relative path is interpreted from your current working directory.***

If a command cannot find a file, first verify:

- where you are (`pwd`)
- what exists there (`ls`)

The PATH environment variable

When you type a command like ls or top, the shell has to find the program to run.
It does this by searching the directories listed in the PATH environment variable, in order.

Practical implications:

- If the "command not found" message appears, the program may not be installed or may not be on your `PATH`.
- If a command runs the “wrong” program, a different directory earlier in `PATH` may be providing a different executable.

Useful commands:

- echo `$PATH` — show the search path
- `which <command>` — show which executable the shell will run

Example:

```
which python3
```

Quick reference

Help and discovery:

- `man <command>`

Diagnostics:

- `uptime`
- `top`
- `ps aux`
- `free -h`

Filesystem capacity (start here when a system “feels wrong”):

Command	Purpose
`df -h`	Show filesystem free space (human-readable)
`du -sh <path>`	Show the total size of a directory
`du -h <path>`	Show the sizes of directory contents

Where you are and how you move (your shell context):

- `pwd`
- `cd`, `cd ..`, `cd ~`, `cd -`

See what exists:

- `ls`, `ls -l`, `ls -a`, `ls -lh`

Find things when you don’t know the path:

- `find <path> -name <pattern>`

Read safely:

- `less <file>`
- `more <file>`
- `cat <file>`

Search inside text:

- `grep <text> <file>`

Change files deliberately:

- `cp <src> <dst>`
- `mv <src> <dst>`
- `rm <file>`
- `rm -r <dir>`

Getting help first

man — manual pages

Use man before you guess flags.

Examples:

```
man ls
man grep
```

Inside man:

- Arrow keys/space to scroll
- `/` to search
- `q` to quit

Process control basics

When you run a program in the terminal, it occupies your shell until it finishes or you intervene.

Basic controls:

Action	Control	Effect
Interrupt	`Ctrl+C`	Interrupt the running program (`SIGINT`)
Suspend	`Ctrl+Z`	Suspend the program and return to the prompt (`SIGTSTP`)
Resume (foreground)	`fg`	Bring a suspended program back to the foreground
Resume (background)	`bg`	Resume a suspended program in the background
List jobs	`jobs`	List suspended or background jobs

Why this matters:

- You can stop or pause work without closing the terminal.
- You can recover control if a command runs longer than expected.
- You can distinguish "stuck" from "busy."

Diagnostics: What is happening right now

These commands do not “fix” anything. They show you what is true.

Command	What it shows
`uptime`	How long the system has been running, and the current load averages
`top`	Live view of CPU and memory consumers
`ps aux`	Complete process list (often used for inventory)
`free -h`	Memory usage summary (human-readable)

Filesystem space and pressure

df — filesystem usage

Use `df` when you suspect the disk is full or writes are behaving strangely.

```
df -h
```

du — directory usage

Use `du` when you need to know what is consuming space.

`du -sh <path>` — total size
`du -h <path>` — sizes of contents

Useful when you see “No space left on device.”

Know where you are

pwd — print working directory

Use `pwd` when you are unsure what “here” means. This reports the current working directory of the shell.

Example:

```
pwd
```

What to notice:

- If a file path is relative (i.e., no leading /), it is interpreted relative to this directory.

Move between directories

cd — change directory

Moves your shell’s working directory.

Command	Meaning
`cd /path/to/dir`	Change to an absolute path
`cd relative/path`	Change to a path relative to your current directory
`cd ..`	Move up one directory
`cd ~`	Change to your home directory
`cd -`	Return to the previous directory

What to notice:

- A directory is not a “container.” It is a path you can enter.

- Hidden files often contain configuration or tool state.

Examples:

```
cd /etc
cd foo/bar
```

See what's in a directory

ls — list directory contents

Use ls to see what exists before you try to open, move, or delete anything.

Common forms:

Command	What it shows
`ls`	Names only
`ls -l`	Long listing (permissions, owner, size, time)
`ls -a`	Includes hidden entries that start with .
`ls -lh`	Long listing with human-readable sizes (e.g., 1.5M)

What to notice:

- A directory is not a "container." It is a path you can enter.
- Hidden files often contain configuration or tool state.

Finding files

find — locate files by name

Use `find` when you know something exists but not where.

Examples:

```
find /path -name "filename"
find . -name "*.txt"
find ~ -name "*.pdf"
```

Read files safely

Linux exposes the system state as files. Reading them is observation, not modification.

less — pager

Use `less` when the output is longer than a screen.

Useful keys:

- `q` quit
- `/text` search
- `n` next match
- `g` start
- `G` end

more — basic pager

Use `more` instead of `less` when you only need forward scrolling.

cat — print a file

Use cat for short files.

Example:

```
cat /proc/<PID>/status
```

Search text in files

grep — search for matching lines

Use `grep` when you need a specific line inside a file.

Examples:

- grep VmRSS /proc/<PID>/status
- grep -E 'VmRSS|VmSize' /proc/<PID>/status
- grep error /var/log/syslog

What to notice:

- grep does not change files. It changes what you see.

Copy, move, and delete

These commands change state. Use them carefully.

cp — copy files

Use `cp` to duplicate a file.

```
cp <src> <dst>
```

Example:

```
cp notes.txt notes.bak
```

mv — move or rename files

Use `mv` to move a file to a new location or rename it.

```
mv <src> <dst>
```

Examples:

```
mv oldname.txt newname.txt
mv file.txt /tmp/
```

rm — delete files and directories

Use `rm` only when you are certain.

```
rm <file>
```
`rm -f <file>` — force, no prompt
`rm -r <dir>` — remove a directory and its contents (DANGEROUS: deletes everything inside)

`rm` is permanent. There is no Trash by default.

Safety rules:

- Always `ls` the target first.
- Prefer deleting a specific file path rather than a broad directory.
- Never use `rm -rf` unless you know exactly what you're doing.

Pipelines and redirection

Linux tools are small and focused. A pipeline lets you connect them.

A pipeline passes the output of one command to the input of another command.

Example:

- `ps aux | grep bash`

Meaning:

- `ps aux` produces a large list
- `|` passes that output forward
- `grep bash` filters the list

A typical safe pattern is paging large output:

- `ps aux | less`

Another common pattern is making `/proc/<PID>/cmdline` readable. That file separates arguments with null bytes (`\0`):

- `cat /proc/<PID>/cmdline | tr '\0' ' '`

Working with /proc

`/proc` is a live, in-memory view of kernel state. It is how the kernel exposes process facts as files.

If a process allocates memory, `/proc/<PID>/status` updates immediately.

Common entries:

Entry	What it shows
`cmdline`	What was executed (command-line arguments)

`exe`	Which binary is running
`cwd`	Current working directory
`status`	Process identity and memory statistics
`fd/`	Open file descriptors

If `/proc/<PID>/` disappears, the process ended.

Essential techniques

Tab completion

Use Tab to complete commands and paths. Double-tap Tab to list options.

Case sensitivity

Linux is case-sensitive. Document.txt and document.txt are different files.

History search

Use `Ctrl+R` to search previous commands instead of pressing Up repeatedly.

Safety via less

If you only need to read a file, use less instead of an editor.

Text editing when unavoidable

nano — simple editor

Use `nano` when you must edit a file, and you want an editor that will not trap you.

```
nano <filename>
```

Essential keys:

Key	Action
`Ctrl+O`	Save (write out)

`Ctrl+X`	Exit
`Ctrl+K`	Cut line
`Ctrl+U`	Paste line

Use `nano` when you must edit a file and want the least-surprising editor. It shows the save/exit controls on screen, so you can always recover. This guide chooses nano not because it is powerful, but because it is hard to get trapped.

When things go wrong

When something fails, do not guess. Use this flow to decide what to check next.

Follow this order every time you hit a wall.

“The command didn’t work.”

Symptom: command not found

Do this:

- Check spelling (Linux is case-sensitive)
- Try:

```
which <command>
man <command>
```

If still stuck:

- The command may not be installed. Stop and ask before proceeding.

“No such file or directory.”

First question: Am I where I think I am?

Do this:

1. `pwd` — confirm your location.
2. `ls` — confirm the file exists.
3. Use tab completion to avoid typos.

If the file exists but is not here:

```
find . -name "<filename>"
```

“Permission denied.”

Do not escalate immediately.

Check:

```
ls -l <file>
```
`ps` (for processes)

Rule:

- If you didn’t create it, you probably don’t own it.
- Do not use `sudo` unless you understand why.

“The terminal looks stuck.”

Try these in order:

1. `q` — many tools quit this way
2. `Ctrl+C` — interrupt
3. `Ctrl+D` — end input/exit shell

If nothing responds:

- Close the terminal
- Open a new one

“The output is unreadable or overwhelming.”

Do not scroll blindly.

Do this:

- Send it to a pager: command | less
- Or filter it: command | grep <text>

“Something is slow or behaving oddly.”

Observe before acting.

Do this:

```
uptime
```

```
top
df -h
free -h
```

If you cannot describe what you see, you are not ready to fix it yet.

Golden rule

If you don’t know what to do next, observe more.

Linux almost always exposes the information you need — but only if you stop guessing.

Mental model to keep

Linux is not opaque.

If something matters, there is:
- a directory you can enter
- a file you can read
- a process you can identify

If you cannot find those, you are not looking at the right level yet.

5-Minute Self-Test: Surviving Linux

This self-test checks whether you can map a situation to the correct command without guessing.

Rules:
- Do not run commands while answering.
- For each scenario, name one primary command from this guide that you would use first.
- If multiple commands could work, choose the best first tool.
- If you cannot justify your choice, you do not understand the situation yet.

Time limit: 5 minutes.

Scenario 1 — Where am I, really?

You run a command, and it fails because a relative path cannot be resolved. You are not sure which directory your shell is currently in.

Question: Which command tells you the truth immediately?

Scenario 2 — The file should be here

You are told a file exists in the current directory, but cat filename returns “No such file or directory.”

Question: Which command do you run first to verify whether the file exists?

Follow-up: Which variant would you use to check for hidden files?

Scenario 3 — I know the file exists, but not where

You know that a PDF file exists somewhere in your home directory, but you do not know which folder it’s in.

Question: Which command from this guide is designed specifically for this situation?

Scenario 4 — This output is unreadable

You run a command, and it floods your terminal with hundreds of lines of output, scrolling off the screen.

Question: What is the safest way to view this output so you can search and scroll?

Scenario 5 — The terminal looks stuck

You run a command, and the terminal stops responding to input. You do not know whether it is waiting, hung, or just running.

Question: What is the first keyboard shortcut you try?

Follow-up: If that does not work, what is the subsequent escalation?

Scenario 6 — Something feels slow

Programs are responding slowly, and you want to know whether the system is under load right now.

Question: Which single command gives you the clearest immediate picture?

Scenario 7 — Am I allowed to read this?

You try to read a file, but you get “Permission denied.”

Question: Which command helps you understand why before doing anything risky?

Scenario 8 — What is this process doing?

You know a process ID and want to see what program is running and how much memory it is using.

Question: Which location in the filesystem gives you authoritative, live information?

Scenario 9 — I need help, not guesses

You remember that a command exists, but cannot remember the flags.

Question: What is the correct way to get authoritative documentation?

Scenario 10 — Danger check

You are about to delete a file you created earlier, but you want to make sure it is the correct file.

Question: What command do you run immediately before `rm`?

Scoring (optional)

- 8–10 correct: You can survive Linux.
- 6–7 correct: You can function, but you will guess under pressure.
- ≤5 correct: You are still thinking in abstractions instead of evidence.

If you miss a question, do not memorize the answer. Go back to the section of the guide that explains why that command exists and what problem it solves.

Self-Test Answer Key

Here it is, formatted consistently with the rest of the survival guide:

Question	Tool	When to use it
1	`pwd`	Print your current working directory
2	`ls`	Verify whether a file is present (ls -a to include hidden files)
3	`find`	Search for files by name or pattern
4	`less (pager)`	Safely view large output (`command | less`)
5	`Ctrl+C`	Interrupt a running command (if still stuck: try q, then `Ctrl+D`, then close the terminal)

6	`top`	Get a live view of system load and resource usage
7	`ls -l <file>`	Show file ownership and permissions
8	`/proc/<PID>/`	Inspect the live kernel state for a process
9	`man <command>`	Read authoritative documentation
10	`ls <file>`	Verify a target immediately before deletion