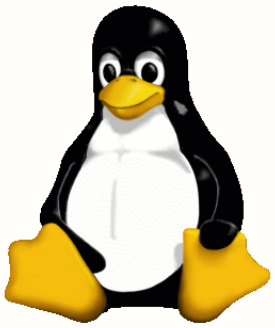
GrowthSchool

LINUX

COMPLETE NOTES



LEARN - ADMIN - AUTOMATE

Filesystem, permissions, processes, networking, scripting and ops

growthschool.cc | @growthschool.cc

Learning Roadmap

GrowthSchool

Work through these in order. Every page has explanation, real commands, a diagram or example, common mistakes, a student lab and a revision snapshot. Examples target a Shop API server.

HOW TO USE THESE NOTES

Practice on a real Linux box or a container (docker run -it ubuntu bash). Reading is not doing - type every command. Use pages 30-35 for fast revision before interviews.

1. Why Linux for DevOps

GrowthSchool

IN SIMPLE TERMS
Linux is a free operating system that runs most servers, containers, and cloud machines. As a DevOps engineer you will operate it every day, usually through the command line.

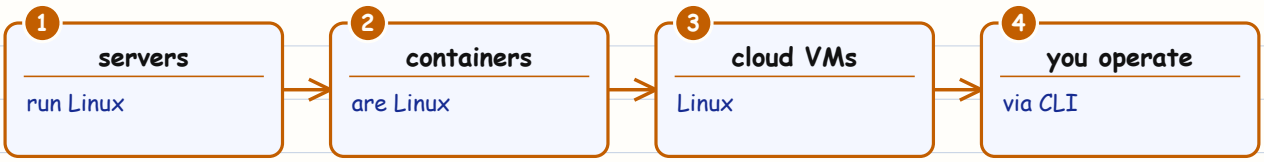
* It Runs the World

- Most servers, containers and cloud instances run Linux. Docker images are Linux userlands.
- If you deploy software, you operate Linux - directly or through containers and CI runners.

* What You Actually Do

- Navigate the filesystem, manage permissions, read logs, control services and write small scripts.
- The command line is faster, scriptable and works identically over SSH on a remote server.

Linux everywhere



REAL-WORLD EXAMPLE

```
uname -a          # kernel + arch
whoami; id        # who am I, my groups
cat /etc/os-release   # which distro + version
uptime            # load average + how long up
```

COMMON MISTAKE
Expecting a GUI on servers. Production Linux is headless; you work over SSH in a terminal. Get comfortable without a mouse.

REVISION SNAPSHOT

ASK YOURSELF Why is Linux fluency non-negotiable for DevOps?

DO THIS Identify your distro, kernel and current user with one command each.

2. The Shell & Command Line

GrowthSchool

IN SIMPLE TERMS
The shell is the program that reads the commands you type and runs them; the command line is the text screen where you type them.

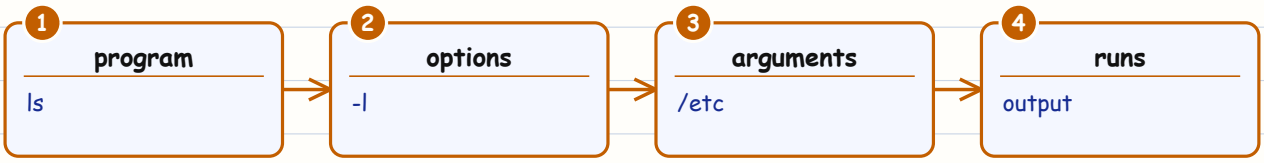
* What the Shell Is

- The shell (usually bash) reads commands and runs programs. A prompt shows user@host:path$.
- A command is: program + options (flags) + arguments, e.g. ls -l /etc.

* Work Faster

- Tab completes names; up-arrow recalls history; Ctrl-R searches it; man/--help explain commands.
- Ctrl-C cancels a command; Ctrl-L clears the screen; Ctrl-A/E jump to line start/end.

A command



REAL-WORLD EXAMPLE

```
ls -l /etc
man ls            # full manual
ls --help         # quick options
history | tail    # recent commands
type ls; which python3
```

COMMON MISTAKE
Retyping long commands. Use history, Ctrl-R and tab completion. Also read man pages instead of guessing flags - the answer is usually right there.

REVISION SNAPSHOT

ASK YOURSELF	What are the three parts of a typical command?
DO THIS	Use Ctrl-R to find and rerun a command from earlier in your session.

3. Filesystem Hierarchy

GrowthSchool

IN SIMPLE TERMS
The filesystem hierarchy is how Linux organises everything under one tree starting at "/", with fixed folders for config, logs, users, and programs.

* One Tree

- Linux has a single tree rooted at /. There are no drive letters; devices mount into the tree.
- Everything is a file - including devices (/dev) and kernel info (/proc, /sys).

* Know These Homes

- Config lives in /etc, logs in /var/log, users in /home, programs in /usr, temp in /tmp.
- Knowing where things live turns "where is that?" into an instant answer.

KEY DIRECTORIES

/ root of everything
/etc system configuration
/home user home directories
/var logs, spools, variable data
/usr installed programs + libs

MORE PATHS

/bin /sbin essential binaries
/tmp temporary files
/dev device files
/proc /sys kernel + process info
/opt /srv add-on + served data

REAL-WORLD EXAMPLE

```
ls /            # top-level layout
ls /etc         # system config files
ls /var/log     # system + app logs
df -h           # mounted filesystems
```

REVISION SNAPSHOT

ASK YOURSELF Where do config, logs and user homes live?

DO THIS Explore /etc, /var/log and /home and name one file in each.

4. Navigation & Paths

GrowthSchool

IN SIMPLE TERMS
Navigation is moving around that folder tree - knowing where you are (pwd) and changing directory (cd) using full or relative paths.

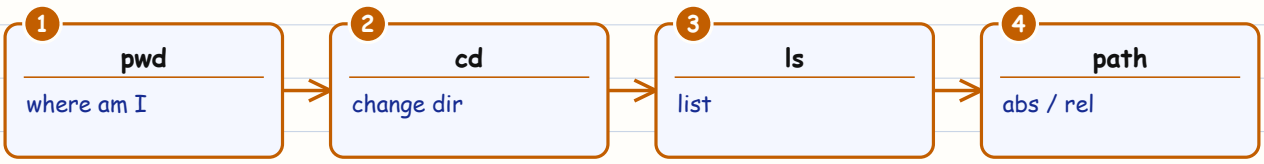
* Move Around

- pwd prints where you are; cd changes directory; ls lists. Absolute paths start with /.
- Relative paths are from your current directory. . is here, .. is up, ~ is your home.

* Speed Moves

- cd - jumps to the previous directory; cd with no argument goes home.
- ls -lah shows long listing, hidden files and human-readable sizes.

Move around



REAL-WORLD EXAMPLE

```
pwd
cd /var/log && ls -lah
cd ~            # home
cd -            # previous directory
cd ../../etc    # up two, then into etc
```

COMMON MISTAKE
Confusing absolute and relative paths. /log is not the same as ./log. When unsure, run pwd and use an absolute path.

REVISION SNAPSHOT

ASK YOURSELF	What do . , .. , ~ and - mean in cd?
DO THIS	Navigate to /var/log using an absolute path, then home using a shortcut.

5. File & Directory Operations

GrowthSchool

IN SIMPLE TERMS
File operations are the everyday actions of creating, copying, moving, and deleting files and folders.

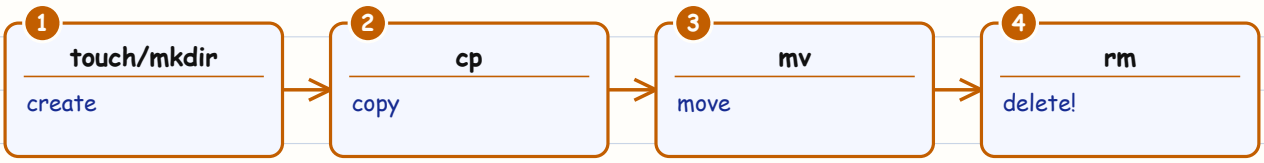
* Create, Copy, Move, Remove

- touch makes an empty file; mkdir -p makes nested dirs; cp copies; mv moves/renames; rm deletes.
- cp -r and rm -r work recursively on directories.

* Be Careful

- rm is permanent - there is no recycle bin. Double-check paths, especially with -r and wildcards.

File actions



REAL-WORLD EXAMPLE

```
mkdir -p shop/api/logs
touch shop/api/app.py
cp -r shop shop-backup
mv shop/api/app.py shop/api/server.py
rm -i shop-backup/api/server.py   # -i asks first
```

COMMON MISTAKE
Running rm -rf with a wrong or empty variable, e.g. rm -rf "$DIR"/ where DIR is unset -> rm -rf /.
Quote variables, avoid trailing slashes on variables, and prefer rm -i when unsure.

REVISION SNAPSHOT

ASK YOURSELF	Why is rm especially dangerous with -r and wildcards?
DO THIS	Create a nested folder tree, copy it, rename a file, then remove the copy safely.

6. Viewing & Editing Files

GrowthSchool

IN SIMPLE TERMS
Viewing and editing files means reading their contents (cat, less, tail) and changing them in a terminal editor like nano or vim.

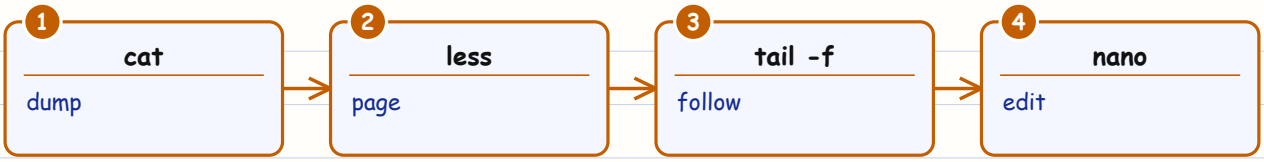
* Read Without Editing

- cat dumps a file; less pages through it (q to quit); head/tail show the start/end.
- tail -f follows a growing file live - essential for watching logs.

* Edit In Place

- nano is the easy editor; vim is powerful but has a learning curve (i to insert, Esc, :wq to save).
- On servers you edit config with these; know at least nano well.

View a file



REAL-WORLD EXAMPLE

```
less /var/log/syslog
head -n 20 access.log
tail -f /var/log/nginx/access.log
nano /etc/hosts
grep -n ERROR app.log | less
```

COMMON MISTAKE
cat-ing a huge log and flooding the terminal. Use less, head, tail, or grep to look at just the part you need.

REVISION SNAPSHOT

ASK YOURSELF	Which command follows a log file as new lines arrive?
DO THIS	Tail -f a log while triggering activity in another terminal.

7. Permissions & Ownership

GrowthSchool

IN SIMPLE TERMS
Permissions decide who can read, write, or run a file; ownership says which user and group a file belongs to.

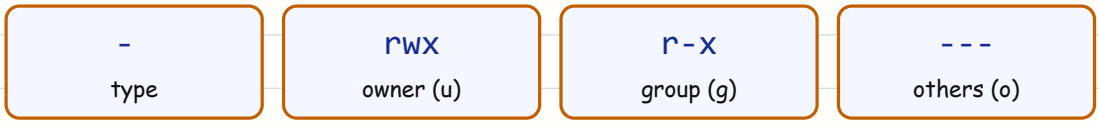
* The Model

- Every file has an owner (user), a group, and permissions for user/group/others.
- Permissions are read (r), write (w), execute (x). ls -l shows them as rwxr-x---.

* On Directories

- x on a directory means "can enter it"; r means "can list it"; w means "can create/delete inside".
- This is why a dir often needs x even just to cd into it.

Reading rwxr-x--- (chmod 750)



r=4 w=2 x=1 -> rwx=7, r-x=5, ---=0 -> 750

REAL-WORLD EXAMPLE

```
ls -l server.py
# -rwxr-x--- 1 asha devs 240 ... server.py
stat server.py       # detailed metadata
namei -l /home/asha/shop/app.py  # path perms
```

REVISION SNAPSHOT

ASK YOURSELF What does x mean on a file vs on a directory?

DO THIS Read ls -l output and state who can read/write/execute a file.

8. chmod, chown & umask

GrowthSchool

IN SIMPLE TERMS
chmod changes permissions, chown changes ownership, and umask sets the default permissions given to newly created files.

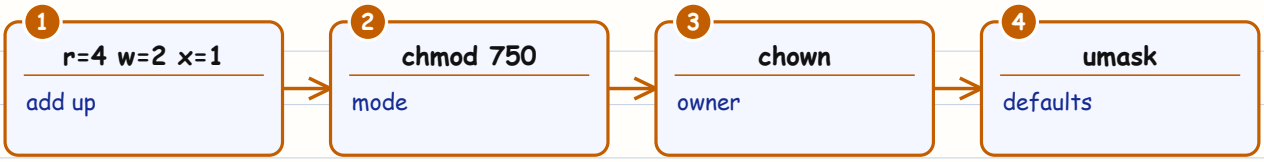
* Change Permissions

- chmod sets permissions numerically (750) or symbolically (u+x, go-w).
- r=4, w=2, x=1; add them per class: rwx=7, r-x=5, r--=4.

* Ownership & Defaults

- chown user:group file changes ownership (usually needs sudo).
- umask sets default permissions for new files; a umask of 022 yields 644 files, 755 dirs.

Set permissions



REAL-WORLD EXAMPLE

```
chmod 750 deploy.sh        # rwx r-x ---
chmod u+x,go-w deploy.sh   # symbolic
sudo chown asha:devs app.log
chmod -R 755 /var/www/shop
umask         # show current default mask
```

COMMON MISTAKE
chmod 777 to "fix" a permission error. That makes files world-writable - a security hole. Grant the least permission that works, usually 644 for files and 755 for dirs/scripts.

REVISION SNAPSHOT

ASK YOURSELF	What numeric mode is rwxr-x--- and how did you get it?
DO THIS	Make a script executable for owner+group only, then run it.

9. Users, Groups & sudo

GrowthSchool

IN SIMPLE TERMS
Users are individual accounts, groups bundle users together, and sudo lets a normal user run a single command with admin power.

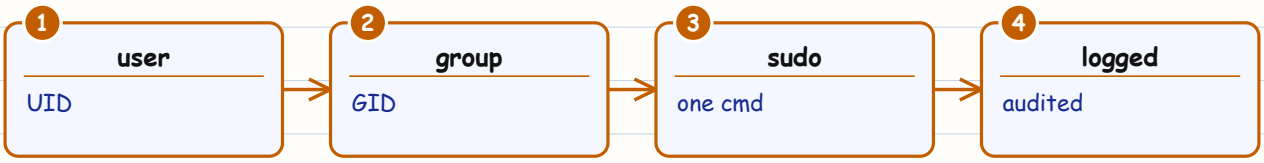
* Accounts

- Users have a UID; groups have a GID. /etc/passwd lists users, /etc/group lists groups.
- Add users with useradd/adduser; add to a group with usermod -aG group user.

* sudo

- sudo runs one command as another user (usually root). It is safer than logging in as root.
- Every sudo use is logged - accountability and a smaller blast radius than a root shell.

Accounts & sudo



REAL-WORLD EXAMPLE

```
id asha
sudo adduser deploy
sudo usermod -aG docker deploy   # add to docker group
groups deploy
sudo -l          # what can I run as sudo?
```

COMMON MISTAKE
Working as root all day "to avoid permission errors". One typo can wreck the system. Use a normal user and sudo only for the specific commands that need it.

REVISION SNAPSHOT

ASK YOURSELF	Why prefer sudo over a persistent root login?
DO THIS	Create a user, add it to a group, and confirm with id and groups.

10. Processes & Signals

GrowthSchool

IN SIMPLE TERMS
A process is a running program; a signal is a message you send it, for example to ask it to stop gracefully or force it to quit.

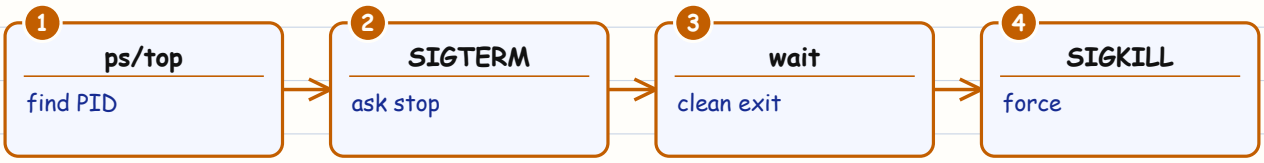
* Seeing Processes

- Every running program is a process with a PID. ps and top/htop list them with CPU/memory use.
- A parent process spawns children; killing a parent can orphan or end its children.

* Signals

- kill sends signals: SIGTERM (15) asks nicely to stop; SIGKILL (9) forces it; SIGHUP (1) reloads.
- Always try SIGTERM first so the app can shut down cleanly and flush data.

Manage a process



REAL-WORLD EXAMPLE

```
ps aux | grep shop-api
top            # live view (q to quit)
kill 4821      # SIGTERM (graceful)
kill -9 4821   # SIGKILL (last resort)
pkill -f shop-api
```

COMMON MISTAKE
Reaching for kill -9 first. It gives the process no chance to clean up, risking corrupt data or leftover lock files. Try SIGTERM, wait, then escalate.

REVISION SNAPSHOT

ASK YOURSELF	What is the difference between SIGTERM and SIGKILL?
DO THIS	Start a long-running command, find its PID, and stop it gracefully.

11. Jobs & Background Control

GrowthSchool

IN SIMPLE TERMS
Job control lets you run tasks in the background, pause and resume them, and keep them running after you log out.

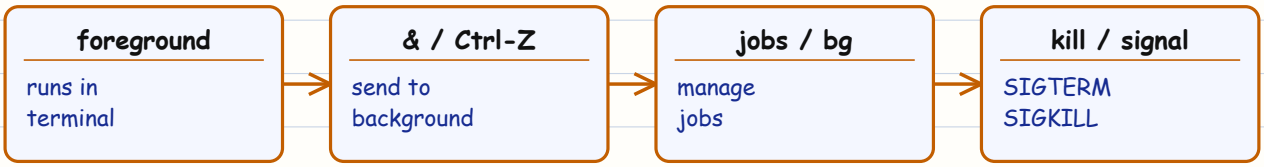
* Run in Background

- Append & to run a command in the background; Ctrl-Z suspends the foreground job.
- jobs lists them; fg brings one to the foreground; bg resumes a suspended job in the background.

* Survive Logout

- A background job dies when the shell exits, unless you use nohup or a terminal multiplexer (tmux).
- For real services, use systemd instead of nohup - it supervises and restarts them.

Process & Job Control Flow



REAL-WORLD EXAMPLE

```
./long-task.sh &     # background
jobs
fg %1                # foreground job 1
nohup ./worker.sh &  # survive logout
tmux new -s deploy   # persistent session
```

COMMON MISTAKE
Using nohup for a production service. It has no restart, no logging discipline and no status. Package long-running services as systemd units instead.

REVISION SNAPSHOT

ASK YOURSELF	How do you keep a task running after you log out?
DO THIS	Background a task, suspend and resume it, then run one with nohup.

12. systemd & Services

GrowthSchool

IN SIMPLE TERMS
systemd is the manager that starts and supervises background services on modern Linux; systemctl is how you control them.

* systemd Basics

- systemd is PID 1: it starts and supervises services (units). Manage them with systemctl.
- A unit file (.service) declares how to start, restart and depend on other units.

* Everyday Control

- start/stop/restart control a service now; enable/disable set whether it starts at boot; status shows state.

Linux Boot Sequence

REAL-WORLD EXAMPLE

```
sudo systemctl status nginx
sudo systemctl restart shop-api
sudo systemctl enable --now shop-api   # start + on boot
systemctl list-units --type=service
systemctl cat shop-api      # show the unit file
```

COMMON MISTAKE
Confusing start with enable. start runs it now; enable makes it start on boot. You usually want both: systemctl enable --now.

REVISION SNAPSHOT

ASK YOURSELF	What is the difference between systemctl start and enable?
DO THIS	Check a service status, restart it, and ensure it starts on boot.

13. Logs & journalctl

GrowthSchool

IN SIMPLE TERMS
Logs are the records of what the system and apps did; journalctl reads the logs collected by systemd.

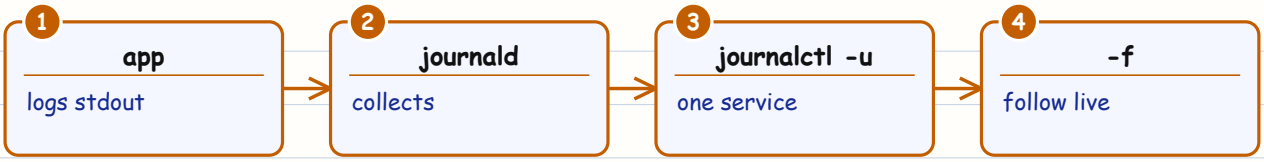
* Where Logs Live

- Traditional logs are text files in /var/log. systemd stores logs in the journal (journalctl).
- Good apps log to stdout/stderr; systemd/containers capture that automatically.

* Query the Journal

- journalctl -u service shows one service; -f follows live; --since filters by time.
- This is your first stop when a service misbehaves.

Read the logs



REAL-WORLD EXAMPLE

```
journalctl -u shop-api -f
journalctl -u shop-api --since "10 min ago"
journalctl -p err -b        # errors this boot
tail -f /var/log/nginx/error.log
journalctl --disk-usage
```

COMMON MISTAKE
Ignoring logs and guessing at causes. Nearly every failure explains itself in journalctl -u <service> or the app log. Read first, change second.

REVISION SNAPSHOT

ASK YOURSELF	How do you follow a single service's logs in real time?
DO THIS	Restart a service and watch its journal show the startup messages.

14. Networking Basics

GrowthSchool

IN SIMPLE TERMS
Networking basics are the tools to check your machine's addresses, routes, and whether it can reach other machines.

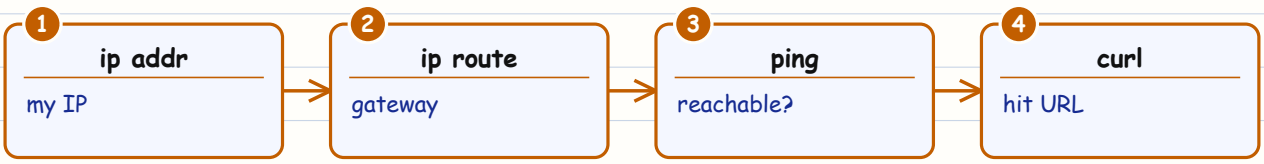
* Interfaces & IPs

- ip addr shows interfaces and IP addresses; ip route shows the routing table and default gateway.
- ping tests reachability; traceroute shows the path packets take.

* Test Connectivity

- curl and wget fetch URLs and are perfect for testing an API from the server itself.
- A failed request could be DNS, routing, firewall or the app - isolate each layer.

Test the network



REAL-WORLD EXAMPLE

```
ip addr
ip route
ping -c 3 growthschool.cc
curl -I https://growthschool.cc
curl -s http://localhost:8080/health
```

COMMON MISTAKE
Assuming "the network is down" when a single layer failed. Test in order: interface up? IP assigned? gateway reachable? DNS resolving? port open? app responding?

REVISION SNAPSHOT

ASK YOURSELF	Which command shows your IP addresses and which shows your default gateway?
DO THIS	Curl your own API health endpoint and a public site from the terminal.

15. DNS, Ports & Sockets

GrowthSchool

IN SIMPLE TERMS
DNS turns names into IP addresses; ports and sockets are the numbered doors that services listen on for connections.

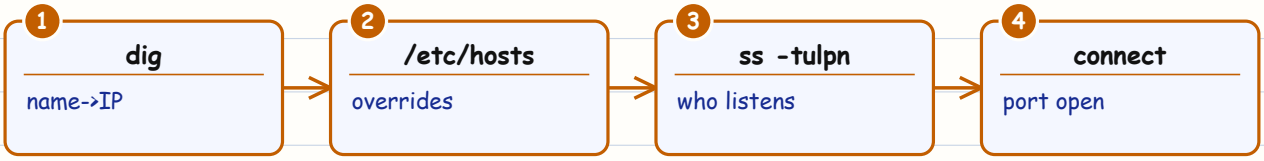
* Name Resolution

- DNS maps names to IPs. dig and nslookup query it; /etc/hosts overrides names locally.
- getent hosts <name> resolves using the systems full configuration.

* Listening Ports

- ss -tulpn lists listening TCP/UDP sockets and the process behind each - the modern netstat.
- If a service is "up" but unreachable, check it is actually listening on the expected port/address.

Name to port



REAL-WORLD EXAMPLE

```
dig +short growthschool.cc
getent hosts db.internal
ss -tulpn | grep 8080
cat /etc/resolv.conf     # configured DNS servers
```

COMMON MISTAKE
A service bound to 127.0.0.1 only, then wondering why remote clients cannot reach it. Bind to 0.0.0.0 (or the right interface) to accept external connections - carefully.

REVISION SNAPSHOT

ASK YOURSELF	Which command lists listening ports and the owning process?
DO THIS	Find which process is listening on a port with ss -tulpn.

16. SSH, SCP & rsync

GrowthSchool

IN SIMPLE TERMS
SSH gives you a secure remote terminal on another machine; SCP and rsync copy files to and from it over that secure link.

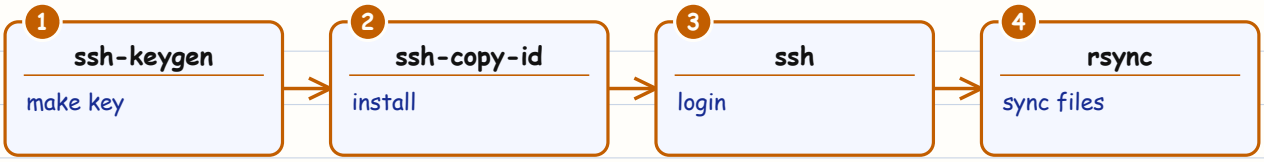
* SSH

- ssh gives a secure remote shell. Use key pairs, not passwords: ssh-keygen then copy the public key.
- Config aliases in ~/.ssh/config save typing host, user and key for each server.

* Copy Files

- scp copies files over SSH; rsync copies efficiently, transferring only differences.
- rsync -avz is the workhorse for syncing directories to/from servers.

Work remotely



REAL-WORLD EXAMPLE

```
ssh-keygen -t ed25519 -C "asha@laptop"
ssh-copy-id deploy@shop-server
ssh deploy@shop-server
scp app.tgz deploy@shop-server:/tmp/
rsync -avz --delete ./dist/ deploy@shop-server:/var/www/shop/
```

COMMON MISTAKE
Using rsync --delete against the wrong target - it removes files not present in the source. Dry-run first with -n, and double-check source/destination order.

REVISION SNAPSHOT

ASK YOURSELF	Why is rsync often better than scp for directories?
DO THIS	Generate an SSH key, add it to a server, and rsync a folder over.

17. Package Managers

GrowthSchool

IN SIMPLE TERMS
A package manager installs, updates, and removes software for you and handles its dependencies (apt, dnf, or apk depending on the distro).

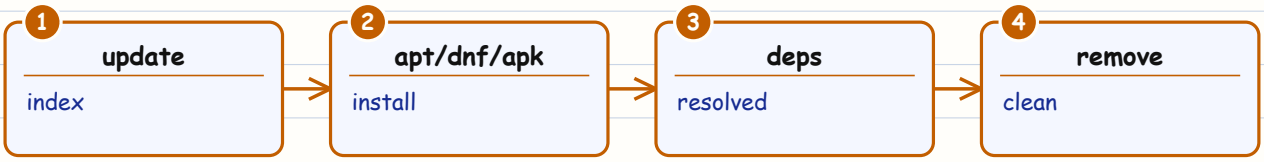
* Install Software

- Debian/Ubuntu use apt (.deb); RHEL/Fedora use dnf/yum (.rpm); Alpine uses apk.
- Update the index, then install/upgrade/remove. Package managers handle dependencies for you.

* Keep It Clean

- Prefer distro packages over random curl-to-bash installs; they are signed and updatable.
- Pin critical versions where reproducibility matters (e.g. in Docker images).

Install software



DISTRO-SPECIFIC
Commands differ per distro family: apt (Debian/Ubuntu), dnf/yum (RHEL/Fedora), apk (Alpine). The concept - indexed, signed, dependency-aware packages - is the same.

REAL-WORLD EXAMPLE

```
sudo apt update && sudo apt install -y nginx   # Debian/Ubuntu
sudo dnf install -y nginx                       # RHEL/Fedora
apk add --no-cache curl                         # Alpine
apt list --installed | grep nginx
```

REVISION SNAPSHOT

ASK YOURSELF	Which package manager belongs to which distro family?
DO THIS	Install, query and remove a small package on your distro.

18. Disk & Storage

GrowthSchool

IN SIMPLE TERMS
Disk and storage commands show how much space you have, what is using it, and which disks are mounted where.

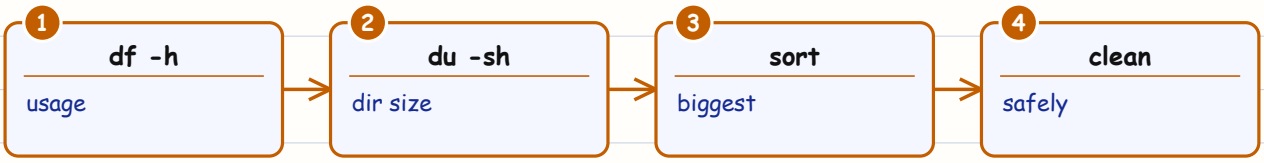
* See Usage

- df -h shows filesystem usage; du -sh shows the size of a directory. Watch for full disks.
- lsblk shows block devices and mounts; a full / disk breaks logging, builds and services.

* Find the Hog

- du -xh --max-depth=1 / | sort -h ranks directories by size to find what filled the disk.

Find the disk hog



REAL-WORLD EXAMPLE

```
df -h
du -sh /var/log
du -xh --max-depth=1 / 2>/dev/null | sort -h | tail
lsblk
ncdu /var        # interactive disk usage (if installed)
```

COMMON MISTAKE
Deleting a large log while a process still holds it open - space is not freed until the process closes the file. Truncate the file or restart the service instead.

REVISION SNAPSHOT

ASK YOURSELF	Which command finds which directory is filling the disk?
DO THIS	Locate the largest directory under / with du and sort.

19. Memory & CPU Diagnostics

GrowthSchool

IN SIMPLE TERMS
Memory and CPU diagnostics are the commands that show how busy your machine is and which processes use the most resources.

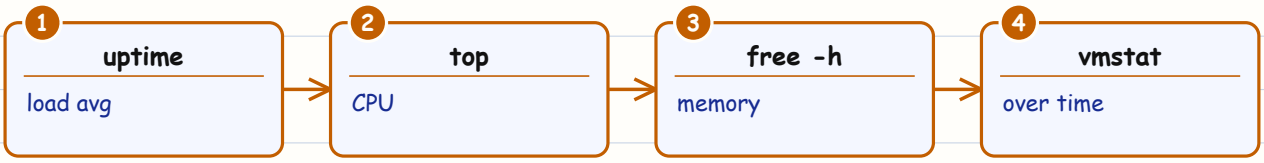
* Snapshot & Live

- free -h shows memory; top/htop show live CPU and per-process usage; uptime shows load average.
- Load average near the CPU count is busy; far above it means saturation or blocked I/O.

* Dig Deeper

- vmstat and iostat reveal CPU, memory, swap and disk I/O over time - useful for sustained issues.

Read the load



REAL-WORLD EXAMPLE

```
free -h
uptime
top -o %CPU
vmstat 1 5
ps aux --sort=-%mem | head
```

COMMON MISTAKE
Panicking about "low free memory". Linux uses free RAM for cache and releases it on demand. Look at available memory and swap activity, not just free.

REVISION SNAPSHOT

ASK YOURSELF	Why is high cached memory usually a good sign, not a problem?
DO THIS	Sort processes by memory and by CPU, and read the load average.

20. grep & Regex

GrowthSchool

IN SIMPLE TERMS
grep searches text for lines that match a pattern; a regex (regular expression) is the mini-language you use to describe that pattern.

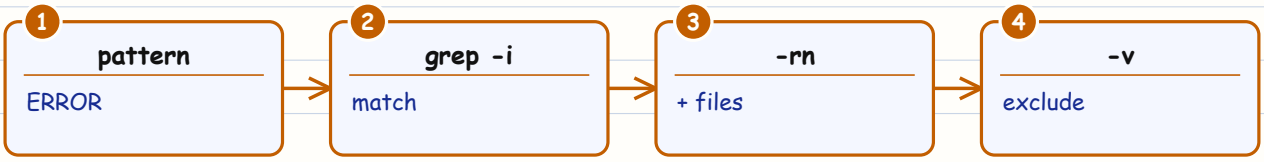
* Search Text

- grep finds lines matching a pattern. -i ignores case, -r recurses, -n shows line numbers, -v inverts.
- grep -E enables extended regex for alternation and quantifiers.

* Regex Building Blocks

- . any char, * zero-or-more, ^ start, $ end, [abc] class, \d-like via [0-9], | alternation (with -E).

grep a log



REAL-WORLD EXAMPLE

```
grep -i error app.log
grep -rn "TODO" src/
grep -E "40[0-9]|50[0-9]" access.log   # 4xx/5xx
grep -c "GET /health" access.log       # count matches
grep -v "healthcheck" access.log       # exclude noise
```

COMMON MISTAKE
Forgetting to quote patterns with special characters, letting the shell expand * or $ before grep sees them. Single-quote your regex.

REVISION SNAPSHOT

ASK YOURSELF	What do -r, -n, -i and -v do in grep?
DO THIS	Count how many 5xx responses appear in a sample access log.

21. find & locate

GrowthSchool

IN SIMPLE TERMS
find searches the filesystem for files by name, size, age, or type, and can run an action on each match.

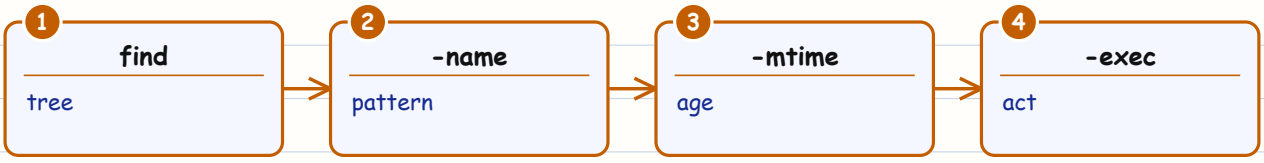
* find

- find searches the tree by name, type, size, time and more, and can act on results with -exec.
- It is precise and real-time (no index), so it is the reliable choice on servers.

* Act on Results

- find ... -delete removes matches; -exec runs a command per file. Test with -print first.

find files



REAL-WORLD EXAMPLE

```
find /var/log -name "*.log" -mtime +7
find . -type f -size +100M
find /tmp -type f -name "*.tmp" -delete
find src -name "*.py" -exec grep -l "TODO" {} +
```

COMMON MISTAKE
Running find ... -delete without first running it with -print to see what matches. Always preview the match set before you delete.

REVISION SNAPSHOT

ASK YOURSELF	How do you find files older than 7 days and act on them safely?
DO THIS	List all files over 100 MB under a directory, then preview a -delete.

22. sed & awk

GrowthSchool

IN SIMPLE TERMS
sed edits text streams (great for find-and-replace); awk processes text column by column for reports and sums.

* sed - Stream Editor

- sed transforms text line by line. s/old/new/g substitutes; -i edits files in place (use with care).
- Great for quick find-replace across files and config tweaks in scripts.

* awk - Field Processor

- awk splits each line into fields ($1, $2, ...) and is perfect for columns, sums and reports.
- Example: sum a column, print specific fields, or filter by a condition.

REAL-WORLD EXAMPLE

```
sed "s/DEBUG/INFO/g" app.log > app.info.log
sed -i "s/localhost/db/" config.env    # in place
awk '{print $1, $7}' access.log        # ip + path
awk '$9 >= 500 {c++} END {print c}' access.log   # 5xx count
```

COMMON MISTAKE
sed -i on the only copy of an important file with an untested expression. Test without -i first, or keep a backup: sed -i.bak "...".

REVISION SNAPSHOT

ASK YOURSELF	When do you reach for sed vs awk?
DO THIS	Use awk to print two columns from a log, then sum a numeric column.

23. Pipes & Redirection

GrowthSchool

IN SIMPLE TERMS
A pipe sends one command's output into another; redirection sends output into (or reads input from) a file.

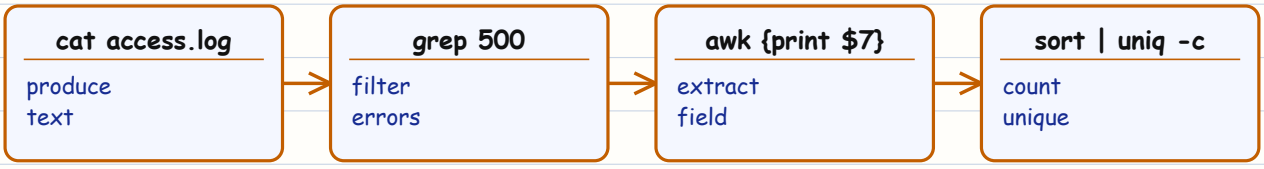
* Compose Small Tools

- A pipe | sends one command's output into the next. Chaining tools solves big problems simply.
- This "do one thing well, then compose" idea is the heart of the Unix philosophy.

* Redirect Streams

- > writes stdout to a file (overwrite), >> appends, 2> redirects stderr, 2>&1 merges them.
- /dev/null discards output; use it to silence noise you do not need.

Pipes: small tools chained into one job



REAL-WORLD EXAMPLE

```
cat access.log | grep " 500 " | awk "{print $7}" | sort | uniq -c | sort -rn | head
command > out.txt 2>&1     # stdout + stderr to file
noisy-cmd 2>/dev/null      # drop errors
echo "$LINE" >> report.txt
```

COMMON MISTAKE
Using > when you meant >>, silently truncating a file you wanted to append to. Also, > creates the file before the command runs, so redirecting a file into itself destroys it.

REVISION SNAPSHOT

ASK YOURSELF	What is the difference between >, >> and 2>&1?
DO THIS	Build a one-line pipeline that ranks the top request paths in a log.

24. Environment & Shell Config

GrowthSchool

IN SIMPLE TERMS
Environment variables carry settings to programs; shell config files (like ~/.bashrc) set up your aliases and preferences.

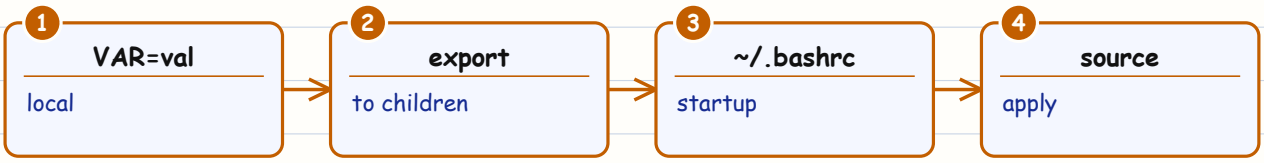
* Variables

- Shell variables (NAME=value) are local; export makes them environment variables for child processes.
- PATH lists where the shell looks for programs; HOME, USER and PWD are common env vars.

* Startup Files

- ~/.bashrc runs for interactive shells; put aliases, exports and prompt tweaks there.
- source a file to apply changes without logging out.

Environment



REAL-WORLD EXAMPLE

```
export APP_ENV=production
echo $PATH
alias ll="ls -lah"
echo "export PATH=\$PATH:/opt/shop/bin" >> ~/.bashrc
source ~/.bashrc
```

COMMON MISTAKE
Setting a variable without export and expecting a child process (or script) to see it. Without export it stays in the current shell only.

REVISION SNAPSHOT

ASK YOURSELF	What does export do that a plain assignment does not?
DO THIS	Add an alias and a PATH entry to ~/.bashrc and source it.

25. Bash Scripting Basics

GrowthSchool

IN SIMPLE TERMS
A bash script is a text file of commands you can run as one program to automate repetitive work.

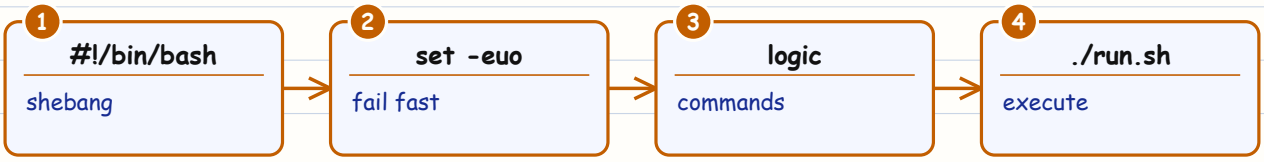
* Anatomy of a Script

- Start with a shebang #!/usr/bin/env bash, make it executable (chmod +x), then run ./script.sh.
- Add set -euo pipefail to fail fast on errors, unset variables and broken pipes.

* Input & Output

- Read arguments as $1, $2, "$@"; read user input with read; print with echo or printf.

A bash script



REAL-WORLD EXAMPLE

```
#!/usr/bin/env bash
set -euo pipefail
NAME="${1:-world}"
echo "Deploying ${NAME}..."
# run:  chmod +x deploy.sh && ./deploy.sh shop-api
```

COMMON MISTAKE
Omitting set -euo pipefail. Scripts then plough on after a failed command, doing damage with half-set variables. Fail fast and loud.

REVISION SNAPSHOT

ASK YOURSELF	What does set -euo pipefail protect you from?
DO THIS	Write a script that greets a name passed as the first argument.

26. Bash: Loops & Conditions

GrowthSchool

IN SIMPLE TERMS
Loops repeat commands over a list, and conditions run commands only when a test is true - the logic of any script.

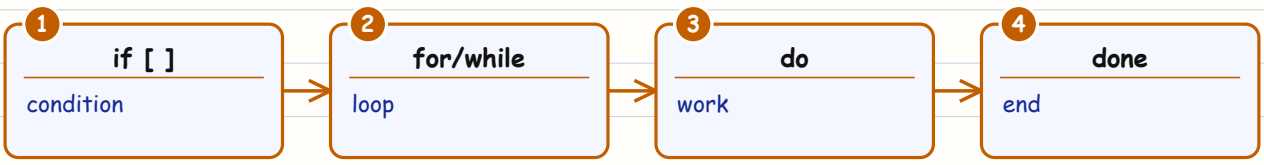
* Conditions

- if ["$x" = "y"]; then ... fi. Use [[...]] in bash for safer tests and pattern matching.
- Test files with -f (exists/file), -d (dir), -z (empty string), -n (non-empty).

* Loops

- for item in list; do ... done iterates; while [cond]; do ... done repeats until false.
- Loop over files, lines or command output to automate repetitive work.

Loops & tests



REAL-WORLD EXAMPLE

```
for f in *.log; do echo "rotating $f"; gzip "$f"; done
if [[ -f /etc/shop.conf ]]; then echo "config found"; fi
while read -r line; do echo ">> $line"; done < hosts.txt
if ! curl -sf localhost:8080/health; then echo DOWN; fi
```

COMMON MISTAKE
Looping over ls output (for f in $(ls)) which breaks on spaces/newlines. Glob directly (for f in *.log) or read files safely with a while-read loop.

REVISION SNAPSHOT

ASK YOURSELF	Why loop over a glob instead of the output of ls?
DO THIS	Write a loop that gzips every .log file in a directory.

27. Bash: Functions & Arguments

GrowthSchool

IN SIMPLE TERMS
Functions are reusable named blocks inside a script; arguments are the values you pass into a script or function.

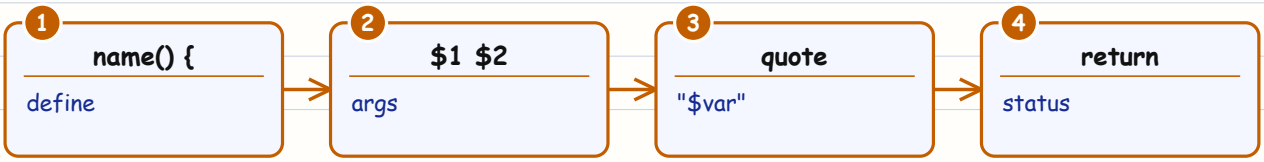
* Functions

- Define reusable blocks: name() { ...; }. Inside, $1..$n are the function's arguments.
- return sets an exit status (0 = success); echo returns data to a caller via command substitution.

* Robust Scripts

- Quote all variable expansions ("$var"); check argument counts; give clear usage/error messages.
- Exit non-zero on failure so callers and CI can detect it.

Functions



REAL-WORLD EXAMPLE

```
deploy() {
  local env="$1"
  [[ -z "$env" ]] && { echo "usage: deploy <env>"; return 1; }
  echo "deploying to ${env}"
}
deploy production
```

COMMON MISTAKE
Leaving variables unquoted so a value with spaces splits into multiple arguments. Always quote: "$1", "$env", "${arr[@]}".

REVISION SNAPSHOT

ASK YOURSELF	Why must you quote variable expansions in bash?
DO THIS	Write a function that validates its arguments and returns non-zero on misuse.

28. cron & Scheduling

GrowthSchool

IN SIMPLE TERMS
cron is the built-in scheduler that runs commands automatically at set times, like a daily backup.

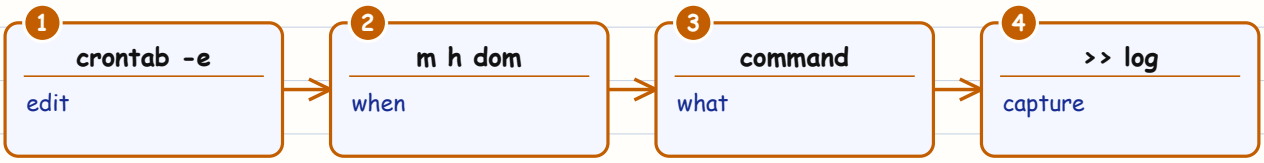
* cron Syntax

- crontab -e edits your schedule. Five fields: minute hour day-of-month month day-of-week + command.
- Example: 0 2 * * * runs daily at 02:00. Use * for "every".

* Reliable Jobs

- Use absolute paths and redirect output to a log; cron runs with a minimal environment.
- For systemd hosts, timers are a more observable alternative to cron.

Schedule with cron



REAL-WORLD EXAMPLE

```
crontab -e
# m h dom mon dow  command
0 2 * * *  /opt/shop/backup.sh >> /var/log/backup.log 2>&1
*/5 * * * * /opt/shop/healthcheck.sh
crontab -l        # list your jobs
```

COMMON MISTAKE
Assuming cron has your interactive PATH and env. It does not. Use absolute paths and set any needed variables explicitly, and always capture output to a log.

REVISION SNAPSHOT

ASK YOURSELF	What do the five cron time fields mean?
DO THIS	Schedule a script to run every 5 minutes and confirm it logs output.

29. Security Basics

GrowthSchool

IN SIMPLE TERMS
Security basics are the essential habits - SSH keys, firewalls, least privilege, and patching - that keep a server safe.

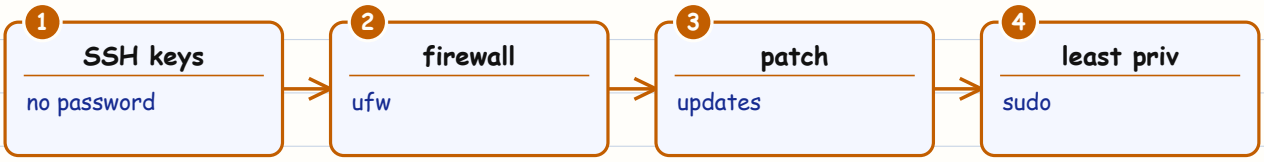
* Harden the Basics

- Use SSH keys and disable password login; keep packages patched; run services as non-root.
- Open only the ports you need with a firewall (ufw/firewalld); use least privilege everywhere.

* Watch & Limit

- Review auth logs for failed logins; use sudo (audited) instead of shared root; rotate credentials.
- Never commit secrets; store them in a secrets manager or protected env files.

Harden a server



REAL-WORLD EXAMPLE

```
sudo ufw allow 22/tcp && sudo ufw allow 443/tcp && sudo ufw enable
grep "Failed password" /var/log/auth.log | tail
sudo apt update && sudo apt upgrade -y
# /etc/ssh/sshd_config: PasswordAuthentication no
```

COMMON MISTAKE
Leaving password SSH login enabled on an internet-facing box. Bots brute-force it constantly. Use keys, disable password auth, and restrict the firewall.

REVISION SNAPSHOT

ASK YOURSELF	Name three quick wins that harden a fresh Linux server.
DO THIS	Enable a firewall allowing only SSH + HTTPS, then verify with ss.

30. Performance Troubleshooting

GrowthSchool

IN SIMPLE TERMS
Performance troubleshooting is a repeatable method for finding what is slowing a server down: CPU, memory, disk, or network.

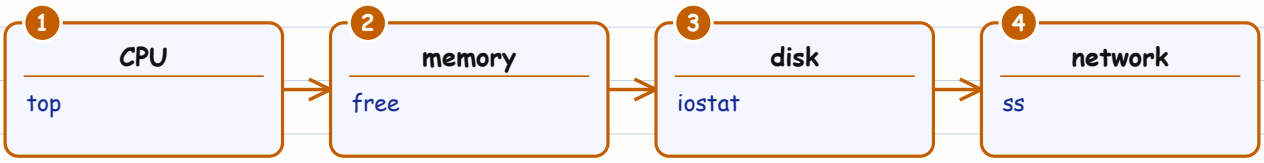
* A Repeatable Method

- Define the symptom, then check the four resources: CPU, memory, disk I/O, network - in that order.
- Use load average + top for CPU, free for memory, iostat for disk, ss/ip for network.

* From Symptom to Cause

- Slow app? Check CPU saturation, memory/swap, disk wait, then dependency latency (DB, network).
- Change one thing at a time and confirm the metric improves before moving on.

Find the bottleneck



REAL-WORLD EXAMPLE

```
uptime                  # load average trend
top -o %CPU             # who burns CPU
free -h                 # memory + swap
iostat -xz 1 3          # disk wait (%util, await)
ss -s                   # socket summary
```

COMMON MISTAKE
Changing several settings at once, so you never learn which one helped (or hurt). Isolate variables and measure after each change.

REVISION SNAPSHOT

ASK YOURSELF	What four resources do you check, and in what order?
DO THIS	Run the CPU/memory/disk/network checks and note a baseline for your box.

31. Real Incident: High CPU

GrowthSchool

IN SIMPLE TERMS
This page walks through a real incident - a server pinned at high CPU - and how to find and fix the cause step by step.

* The Scenario

- Shop API latency spikes. Alerts show load average of 12 on a 4-core box - CPU saturated.
- Users see timeouts. You SSH in to investigate calmly using a method, not guesswork.

* The Investigation

- top shows one shop-api worker at 380% CPU. ps and logs reveal a tight retry loop on a failing call.
- Fix: patch the loop / add backoff, restart the service, and confirm load returns to normal.

REAL-WORLD EXAMPLE

```
uptime
top -o %CPU            # find the hot process
ps -p 4821 -o pid,ppid,%cpu,cmd
journalctl -u shop-api --since "15 min ago" | tail
sudo systemctl restart shop-api
```

COMMON MISTAKE
Rebooting the whole server first. That hides the cause and it recurs. Identify the process, read its logs, fix the root cause, then restart just that service.

REVISION SNAPSHOT

ASK YOURSELF	Walk the steps from "high load alert" to root cause.
DO THIS	Simulate CPU load (e.g. yes >/dev/null &) and locate it with top, then stop it.

32. Real Incident: Disk Full

GrowthSchool

IN SIMPLE TERMS
This page walks through a real incident - a full disk - and how to find what filled it and safely reclaim space.

* The Scenario

- Writes start failing with "No space left on device". The app cannot log or accept uploads.
- df -h shows / at 100%. You must find and safely reclaim space without deleting needed data.

* The Investigation

- du ranks the biggest directories; a runaway /var/log file or old Docker images is a common culprit.
- Truncate/rotate logs, prune unused images, then add rotation/monitoring so it does not recur.

REAL-WORLD EXAMPLE

```
df -h
du -xh --max-depth=1 / 2>/dev/null | sort -h | tail
sudo truncate -s 0 /var/log/huge.log   # free space, keep file
docker system df && docker system prune
journalctl --vacuum-size=200M
```

COMMON MISTAKE
rm on a huge log a running process still holds open - space is not freed until the process releases it. Truncate the file (or restart the writer) instead.

REVISION SNAPSHOT

ASK YOURSELF	Why can deleting an open log file fail to free space?
DO THIS	Fill a test file with fallocate, watch df, then reclaim the space.

33. Linux Command Cheat Sheet

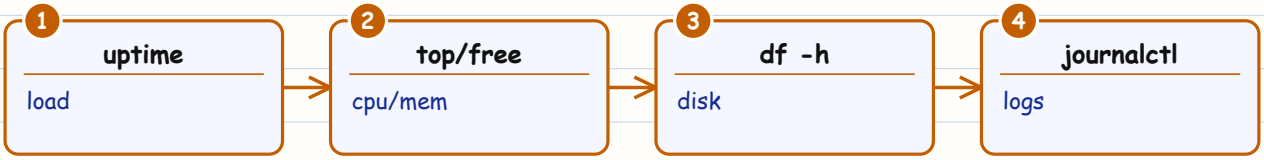
GrowthSchool

IN SIMPLE TERMS
This is a quick-reference list of the Linux commands you will use every day.

* Daily Drivers

- ls/cd/pwd, cp/mv/rm, chmod/chown, ps/top/kill, systemctl, journalctl, grep/find/awk/sed, ss, df/du.

Triage a server



TRIAGE LOOP
uptime -> top -> free -h -> df -h -> journalctl -u <svc>. These five commands orient you on almost any misbehaving Linux server in under a minute.

ONE-PAGE COMMAND REFERENCE

```
ls -lah     cd -          cp -r  mv  rm -i      mkdir -p
chmod 750   chown u:g     stat   umask          sudo -l
ps aux      top/htop      kill -TERM  pkill     free -h
systemctl status/restart/enable --now          journalctl -u x -f
grep -rn    find -name -mtime -exec            sed -i  awk
ss -tulpn   ip addr/route  dig  curl -I        ping -c 3
df -h  du -sh  lsblk       tar czf/xzf         rsync -avz
crontab -e    for/while/if    export  source   chmod +x
```

COMMON MISTAKE
Memorising exotic flags while fumbling the daily ones. Drill ls/cd/grep/ps/systemctl/journalctl until they are reflex - those carry 90% of real server work.

STUDENT LAB
Cover this page and, from memory, write the command to: follow a service log, find the biggest directory, list listening ports, and make a script executable. Check yourself against the sheet.

REVISION SNAPSHOT

ASK YOURSELF	Which five commands triage a struggling server fastest?
DO THIS	Recall fifteen commands from memory grouped by purpose.

34. Capstone Challenge

GrowthSchool

IN SIMPLE TERMS
This capstone ties everything together by setting up and operating a real Linux server for the Shop API.

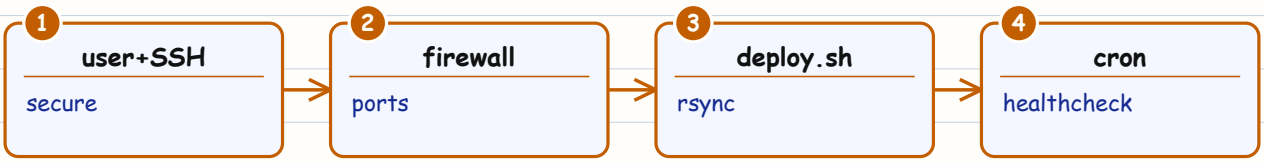
* Provision a Shop API Host

- Create a deploy user with sudo, SSH-key-only login, and a firewall allowing SSH + HTTPS.
- Install nginx, set correct ownership/permissions on /var/www/shop, and enable services on boot.

* Automate & Operate

- Write a bash deploy script (set -euo pipefail) that rsyncs a build and restarts the service.
- Add a cron healthcheck that logs status, and a log-rotation config so the disk never fills.

Capstone steps



REAL-WORLD EXAMPLE

```
sudo adduser deploy && sudo usermod -aG sudo deploy
sudo ufw allow 22,443/tcp && sudo ufw enable
rsync -avz --delete ./dist/ deploy@host:/var/www/shop/
sudo systemctl restart nginx && systemctl is-active nginx
```

STUDENT LAB
Deliverables: key-only SSH, a firewall, a running enabled service, an idempotent deploy script, a cron healthcheck writing to a log, and log rotation configured.

REVISION SNAPSHOT

ASK YOURSELF	Is your server key-only, firewalled, and self-healing on reboot?
DO THIS	Complete every capstone deliverable on a VM or container and document it.

35. Interview & Revision Notes

GrowthSchool

IN SIMPLE TERMS
This page gathers the key ideas and common interview questions so you can revise quickly.

* Explain Clearly

- Permissions (rwx + numeric), SIGTERM vs SIGKILL, systemctl start vs enable, pipes + redirection.
- How you triage high CPU or a full disk, and why you avoid running as root.

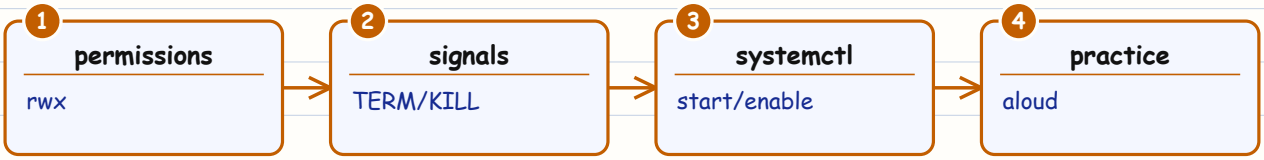
* Common Questions

- How do you find what is using a port / the disk / the CPU? How do you follow a service's logs?
- What does set -euo pipefail do? How do SSH keys work? What is the difference between > and >>?

* Your Next Step

- Run everything on a real box. Recreate the two incidents deliberately and fix them from memory.

Revise out loud



REAL-WORLD EXAMPLE

```
# 30-second self test
chmod 640 file  # who can read/write now?
ss -tulpn | grep 8080   # what is listening?
journalctl -u svc -f    # follow logs
# SIGTERM vs SIGKILL? start vs enable? > vs >>?
```

REVISION SNAPSHOT

ASK YOURSELF	Can you explain rwx numeric permissions in 30 seconds?
DO THIS	Answer every "Common Question" above out loud without notes.