Linux Day 3 — Processes, Signals, Priorities & Environment

> Friendly, visual, and DevOps-ready. This cheat sheet corrects the source slides, adds quick visuals, and ties every topic to real operations.

---

0) What you'll learn fast

- Processes (PID/PPID, daemons, shell jobs) and how to list/control them.
- Signals and safe process termination.
- Priorities/niceness & when to tune them.
- Real-time monitors (top/htop/btop) — key hotkeys.
- I/O redirection & pipes — with mental models.
- Environment variables — viewing, exporting, and persisting.
- Startup files order on Ubuntu/Debian & how to set defaults.
- Aliases & history for speed.

---

1) Processes — the mental model

Process = program in execution. It owns a unique **PID**, has a **parent (PPID)**, lives in RAM, and consumes CPU/I/O.

Kinds: - **Shell jobs** — commands you start in your terminal. - **Daemons/services** — background processes managed by **systemd**. - **Kernel threads** — internal OS helpers.

```
Parent (PPID) ➔ Child (PID)
If parent exits → child is adopted by systemd (PID 1)
```

DevOps scenario

- Your app rollout script spawns helpers. If the parent script dies, helpers get adopted by **systemd**, so they don't necessarily die — design cleanup traps (`trap` in bash) if needed.

---

2) Listing & finding processes

One-shot views

```
ps aux                 # everything (user, CPU, MEM, command)
ps -eo pid,ppid,ni,stat,cmd --sort=-%cpu | head  # pick columns
pgrep -fl nginx        # find by name (show PID + name)
pidof sshd             # PIDs of a daemon by exact name
```

Real-time views

```
top                    # built-in real-time monitor
htop || btop           # nicer TUI (if installed)
```

top hotkeys (must-know): `P` sort by CPU • `M` by MEM • `k` kill • `r` renice • `/` filter • `1` show CPU cores • `q` quit.

DevOps scenario

- On a high-CPU node, `top → P` + `r` to lower priority of a noisy batch job while keeping critical services responsive.

---

3) Job control (within your shell)

Foreground job blocks the terminal; **background** job frees it.

```
sleep 100 &    # start in background (job id %1)
jobs           # list jobs in this shell
Ctrl+Z         # pause (SIGTSTP) current job → background (stopped)
bg %1          # continue job %1 in background
fg %1          # bring job %1 to foreground
```

> Job control only affects processes started from **this** shell.

DevOps scenario

- You kicked off `kubectl logs -f` then need the prompt back → press **Ctrl+Z** then `bg` to keep streaming while you continue working.

---

4) Signals — talk to processes

Signals are messages to processes. Default if unspecified: **TERM** (polite stop).

Signal	Number	Meaning / When to use
`SIGTERM`	15	Ask to exit cleanly (default for `kill`).
`SIGHUP`	1	Reload config / re-open logs (daemons).
`SIGINT`	2	Interrupt from keyboard (**Ctrl+C**).
`SIGTSTP`	20	Stop from keyboard (**Ctrl+Z**).
`SIGCONT`	18	Continue a stopped process.
`SIGKILL`	9	Force-kill (no cleanup). **Last resort.**

```
kill -TERM <pid>              # polite stop
kill -HUP  $(pidof nginx)     # reload nginx
pkill -f myapp                # by pattern/name (cautious)
killall ssh                   # all processes by exact name
kill -l                       # list all signals
```

DevOps scenario

- Rotate app logs: send **HUP** to make daemons re-open log files after you moved/rotated them.

---

5) Priorities & Niceness (CPU scheduling)

Linux uses priorities from **-20 (highest)** to **+19 (lowest)**. Niceness is a *hint* to the scheduler.

```
ps -eo pid,ni,cmd --sort=ni       # show niceness
nice -n 10 heavy-task             # start new process “nicer”
renice +10 -p <pid>               # make running process nicer
```

Rules: regular users can only increase niceness (lower priority). Root can set any value.

DevOps scenario

- CI runner compiling assets on a shared node? Start it with `nice -n 10` so it won’t starve production workloads.

---

6) I/O redirection — the picture you'll remember

```
[stdin 0]  ──>  command  ──>  [stdout 1]
                         └─>  [stderr 2]
```

Operator	What it does	Example
`>`	overwrite stdout to file	`cmd > out.txt`
`>>`	append stdout	`cmd >> out.txt`
`<`	take stdin from file	`cmd < in.txt`
`2>`	stderr to file	`cmd 2> err.txt`
`&>`	stdout+stderr to file	`cmd &> all.txt`
`2>&1`	merge stderr into stdout	`cmd > out.txt 2>&1`
`/dev/null`	black hole	`cmd > /dev/null 2>&1`
`tee`	write to file **and** screen	`cmd | tee out.txt`

DevOps scenario

- Capture deployment output to artifact: `deploy.sh 2>&1 | tee deploy-$(date +%F).log` → live view + saved log.

---

7) Pipes — build mini data flows

Pipe `|` sends stdout of the left command to stdin of the right.

```
journalctl -u myapp -n 500 | grep ERROR | tail -n 20
ps aux | sort -k3 -nr | head      # top 10 by CPU
kubectl get pods -A -o wide | column -t
```

DevOps scenario

- Quick SRE triage pipelines: chain `journalctl`, `grep`, `awk`, `jq`, `column` to surface what matters fast.

---

8) Environment variables — view, set, persist

Common: `$HOME`, `$USER`, `$PWD`, `$SHELL`, `$HOSTNAME`, `$PATH`, `$PS1`, `$$` (current shell PID).

View & set

```
echo "$PATH"
X=42            # shell variable (current shell only)
export X=42     # environment variable (inherited by children)
printenv | sort | less
```

Persist (Ubuntu/Debian)

- **Per-user login shells:** `~/.profile`
- **Per-user interactive shells:** `~/.bashrc`
- **System-wide login defaults:** `/etc/profile`
- **System-wide interactive bash:** `/etc/bash.bashrc`

Add exports in the right file. Example (per-user): add to `~/.bashrc`:

```
export PATH="$HOME/.local/bin:$PATH"
```

Then reload: `source ~/.bashrc`.

DevOps scenario

- Build tools in CI need `PATH` tweaks; export only what's needed and keep secrets out of shell history (use a `.env` file and `env -i`).

---

9) Aliases & command resolution

```
alias gs='git status'
unalias gs
alias ll='ls -alF --color=auto'
```

Find what will actually run:

```
type ls       # builtin/alias/function/file?
which ls      # path of external command
```

DevOps scenario

- Keep dangerous commands explicit. Example: avoid aliasing `rm` to `rm -rf`. Instead, create a **function** that asks for confirmation in prod hosts.

---

10) Bash history — speed + auditing

```
echo $HISTSIZE       # in-memory commands count
echo $HISTFILESIZE   # saved history lines
history              # list
afc -l               # edit & re-run (advanced)
history -w           # flush memory to ~/.bash_history
```

Tips: prefix a command with a space to skip history if `HISTCONTROL=ignorespace`.

DevOps scenario

- After an incident, `history | grep kube` helps reconstruct what was run.

---

11) Wildcards (globbing) — safe patterns

Pattern	Matches
`*`	any chars, any length
`?`	any single char
`[abc]`	one char from set
`{a,b}`	alternatives (brace expansion)

```
# audit first, then delete
ls /var/log/myapp/*.gz
rm -v /var/log/myapp/*.gz
```

DevOps scenario

- Rotate stale artifacts: `find build -type f -name '*.tar.gz' -mtime +14 -print` then replace `-print` with `-delete` after review.

---

12) Mini playbooks (practice bites)

A. Calmly stop then force if needed

```
pid=$(pgrep -f myapp | head -n1)
kill -TERM "$pid" && sleep 2 || true
pgrep -f myapp >/dev/null && kill -KILL "$pid"
```

B. Lower priority of a noisy process

```
renice +10 -p $(pgrep -f webpack)
```

C. Capture logs and screen

```
journalctl -u myapp -f | tee myapp-$(date +%F).log
```

D. Export env only for one command

```
ENV=staging DB_URL=postgres://... myapp --migrate
```

---

13) Quick reference wall

```
# Processes
ps aux | less
pgrep -fl name

# top hotkeys
P/M sort • r renice • k kill • 1 CPUs • / filter

# Signals
kill -TERM <pid>
kill -HUP  $(pidof svc)
kill -9    <pid>   # last resort

# Niceness
nice -n 10 cmd
renice +5 -p <pid>

# Redirection
cmd >out 2>err
cmd >out 2>&1
cmd | tee out
```

```
# Env
export VAR=val
printenv | sort
```

---

14) 60-second check-in

1) What's the difference between `SIGTERM` and `SIGKILL`? When would you use each? 2) Which file do you edit to persist PATH changes for your user's interactive shells? 3) Show a one-liner to list top 5 CPU processes using `ps` + `sort`. 4) How do you continue a job you just suspended with **Ctrl+Z**? 5) How do you send both stdout and stderr into the same file?

Final note

Practice the **mini-playbooks** and the check-in daily. In incidents, the right muscle memory beats slow Googling.