

Processes & Jobs (Core)

Linux Commands Course · Section 10

Goal

Learn how to **observe, control, and manage running programs** in Linux.

You'll understand process IDs, priorities, foreground/background jobs, and safe termination.

What Is a Process?

A **process** is a running instance of a program.

Every process has:

- A **PID** (Process ID)
- A **parent process**
- A **state** (running, sleeping, stopped, zombie)
- An **owner** and resource usage (CPU, memory)

All running processes form a hierarchy – you can view it at any time.

Viewing Processes – ps

`ps` lists running processes.

```
ps aux
```

Columns:

- USER – process owner
- PID – process ID
- %CPU, %MEM – resource usage
- STAT – process state
- COMMAND – what's running

Example:

USER	PID	%CPU	%MEM	COMMAND
root	1	0.0	0.1	systemd
student	213	0.1	0.5	bash
student	230	2.5	1.2	python3 script.py

Interactive Process Viewer – top

Displays a live updating view of running processes.

```
top
```

Common controls inside `top`:

- `P` – sort by CPU
- `M` – sort by memory
- `K` – kill a process by PID
- `Q` – quit

More colorful alternative (if installed): `htop`

```
htop
```

Use `F6` to sort columns, `F9` to kill, `F10` to quit.

Finding Processes – `pgrep` and `pkill`

Search processes by name:

```
pgrep bash
```

This prints PIDs for all matching processes.

Kill by name (no need for PID):

```
pkill firefox
```

Force kill with signal 9 (SIGKILL):

```
pkill -9 firefox
```

Killing by PID – kill

Stop a process using its PID.

Example:

```
ps aux | grep sleep  
kill 1234
```

Graceful termination (default SIGTERM 15):

```
kill 1234
```

Force kill if unresponsive:

```
kill -9 1234
```

Kill Multiple at Once – `killall`

Ends all processes with the same name.

```
killall python3
```

Useful for stopping multiple instances of a command quickly.

Adjusting Priority – nice

Each process has a **niceness** value (priority).
Lower value → higher priority.

Start a command with custom priority:

```
nice -n 10 script.sh
```

Default nice value = 0.
Range = **-20 (highest)** to **19 (lowest)**.

Changing Priority of Running Process – **renice**

Change niceness for an existing process:

```
renice +5 -p 2134
```

Example: lower CPU priority of a background task.

Only root can increase (raise) priority (negative nice values).

Foreground and Background Jobs

When you run a command normally, it runs in the **foreground**.

To run it in the **background**, append **&**:

```
sleep 60 &
```

Output example:

```
[1] 1234
```

[1] is the job number, 1234 is the PID.

Listing Jobs – jobs

Show jobs started from the current terminal:

```
jobs
```

Example output:

```
[1]+  Running  sleep 60 &
```

Jobs are tied to your shell session.

Bringing Jobs to Foreground or Background

Bring job to foreground:

```
fg %1
```

Send a suspended job to background again:

```
bg %1
```

Stop a running job temporarily with **Ctrl+Z**.

Disowning Jobs – disown

Detach a job from the current shell so it keeps running after you close the terminal.

```
disown %1
```

This removes it from the job table.

Example workflow:

```
sleep 300 &  
disown %1  
exit
```

The job keeps running even after logout.

Persistent Background Processes – `nohup`

`nohup` runs a command immune to hangups or logouts.

```
nohup long_task.sh &
```

Output is redirected to `nohup.out` by default.

Perfect for running long scripts or background services safely.

Signals Overview

Processes can receive signals (software interrupts).

Common ones:

Signal	Number	Meaning
SIGTERM	15	Graceful termination
SIGKILL	9	Force kill, cannot be trapped
SIGSTOP	19	Pause process
SIGCONT	18	Resume process

Send custom signals:

```
kill -STOP 2134 # pause
kill -CONT 2134 # resume
```


Combining Process Tools

Practical usage example:

```
ps aux | grep nginx  
sudo systemctl restart nginx  
pgrep nginx  
pkill -HUP nginx
```

- View → restart → verify → reload gracefully
-

Monitoring Processes Efficiently

Show top memory users:

```
ps -eo pid,comm,%mem --sort=-%mem | head
```

Show tree of process relationships:

```
pstree -p | less
```

These are great for debugging and audits.

Recap

- View: `ps aux`, `top`, `htop`
- Find / kill: `pgrep`, `pkill`, `kill`, `killall`
- Adjust priority: `nice`, `renice`
- Manage jobs: `&`, `jobs`, `fg`, `bg`, `disown`, `nohup`
- Know signals: graceful vs forceful termination

Mastering these makes you fluent in process control and multitasking.

Practice

1. Run `sleep 120` in background and list it with `jobs`.
 2. Bring it to foreground, then stop it with `Ctrl+Z`.
 3. Resume in background with `bg`.
 4. Disown it and close your terminal – confirm it stays active via `ps`.
 5. Start a command with low priority using `nice`.
 6. Use `top` to monitor and `kill` to stop it gracefully.
-

Next Up

Services & Logs (Core) – managing system daemons and tracking their behavior.