

Linux: System Administration





WORKFORCE DEVELOPMENT



LOGISTICS



Class Hours:

- Instructor will set class start and end times.
- There will be regular breaks in class.



Telecommunication:

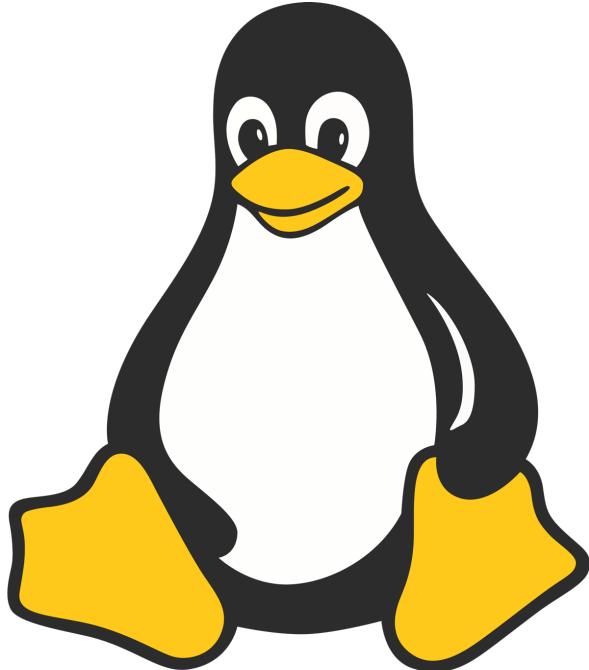
- Turn off or set electronic devices to silent (not vibrate)



Learning:

- Run the commands with the instructor as the slides are presented to you
- Ask questions and participate

Objectives



1. Become comfortable editing files with VI
2. Learn essential commands and skills for system monitoring
3. Understand how manage Linux Jobs and Processes
4. Apply operators and command chaining

Learning Commands

You will learn **many new commands** throughout this course.
Don't focus on memorizing them all — instead, focus on **understanding their purpose**.

Ask yourself:

"What problem does this command solve?"

When you face a challenge, **look for the right command** to help you solve it.

Always read the manual for new commands and options:

`man <command>`

To see **all available commands** on your system:

`compgen -c`

 *Mastery in Linux comes from curiosity and practice — not memorization.*

Linux: File Editing

Editing files is one of the **most basic and essential** system administration skills.

Why it matters:

- Configuration files control almost **everything** — networking, services, security, and startup behavior.
- Quick edits can **fix problems** or **apply changes** without reinstalling software.
- Many servers are **CLI-only** — you won't always have a GUI text editor.



Editing Files with VI

🧠 Why Learn VI / VIM

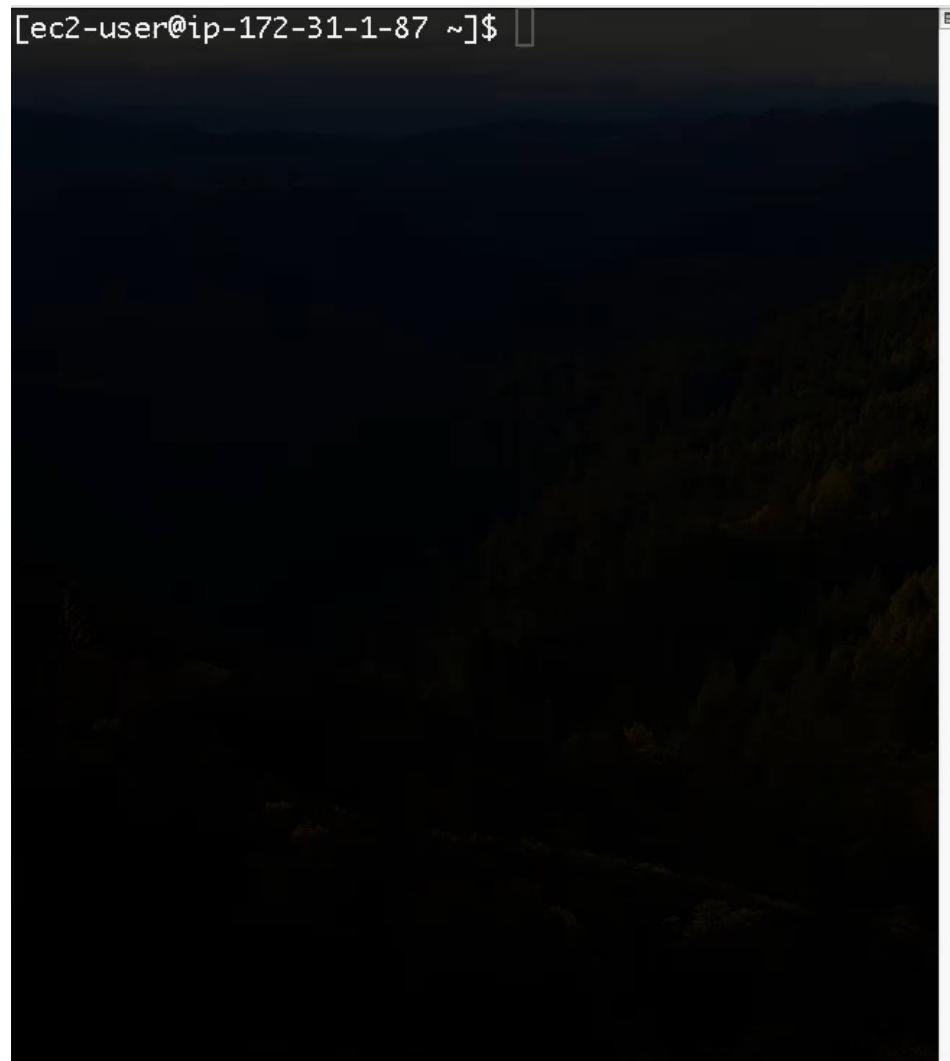
What It Is:

- Vim (Vi Improved) – a terminal-based **text editor** derived from the original *vi*.
- Available on virtually **every** Linux/Unix system — no installation needed.

Why It Matters:

- 🖥️ **Always Available** – found on all servers; perfect for SSH sessions.
- ⚡ **Fast & Lightweight** – no GUI overhead, runs even on minimal systems.

[ec2-user@ip-172-31-1-87 ~]\$ █



Video: VI Demo

Editing Files with VI

⚡ Understanding the VI/VIM Demo

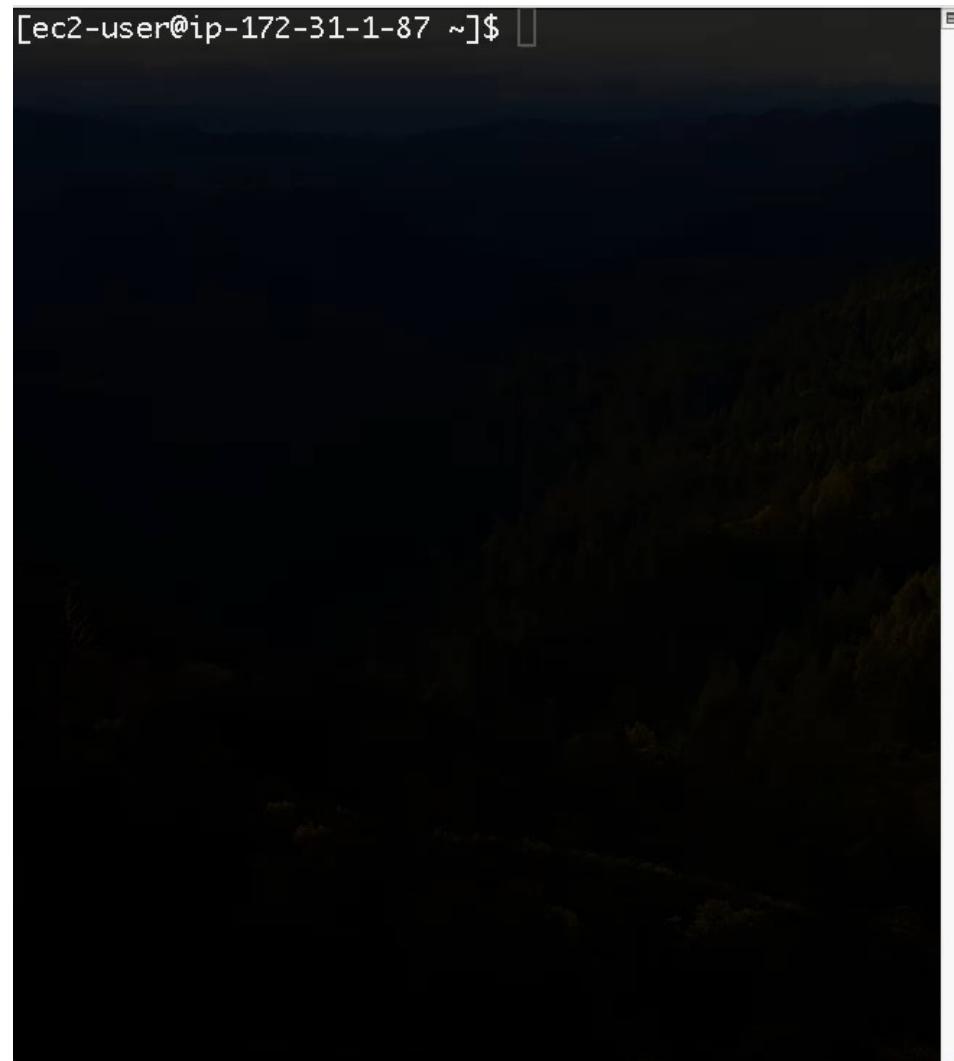
In the demo, we edited a file using just the keyboard:

- **i → Insert Mode**
Enables typing text into the file.
- **Esc → Command Mode**
Exits editing and returns control to VIM commands.
- **:wq → Write & Quit**
Saves the file (w) and exits (q).

💡 Key Idea:

VIM uses **modes** — you switch between typing text and running commands.
Once you learn the shortcuts, it's one of the **fastest editors** you can use on any server.

[ec2-user@ip-172-31-1-87 ~]\$ █



Video: VI Demo

Editing Files with VI

⚡ Common VI/VIM Shortcuts (beginner)

Navigation

- **gg** → Go to the **top** of the file
- **Shift+G** → Go to the **bottom** of the file

Editing

- **yy** → Copy (yank) a line
- **p** → Paste the copied line
- **dd** → Delete the current line
- **u** → Undo
- **Ctrl+r** → Redo

Exit

- **:wq** → Write (save) and quit (exit)
- **:q!** → Quit without saving

💡 *Tip: These are all you need to start feeling comfortable moving around and editing files in VIM.*

Editing Files with VI

Search & Replace in VI/VIM

Searching

- `/word` → Search **forward** for “word”
- `n` → Jump to the **next** match
- `N` → Jump to the **previous** match

Replacing

- `:%s/old/new/g` → Replace **all** occurrences of “old” with “new” in the file
- `:s/old/new/` → Replace only in the **current line**

Note:

Search and replace commands make VIM extremely powerful — you can modify hundreds of lines in seconds.

Editing Files with VI

⚠ Don't Panic if you see this screen

Vi simply creates a backup if the editor exits unexpectedly.

- R to recover the unsaved file
- D to delete the backup
- E to edit the saved version

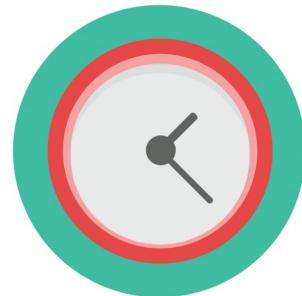
```
E325: ATTENTION
Found a swap file by the name ".test.txt.swp"
    owned by: ec2-user   dated: Sat Oct 25 16:57:42 2025
    file name: ~ec2-user/test.txt
    modified: YES
        user name: ec2-user   host name: ip-172-31-1-87.us-west-1.compute.internal
        process ID: 212922
While opening file "test.txt"
    CANNOT BE FOUND
(1) Another program may be editing the same file. If this is the case,
    be careful not to end up with two different instances of the same
    file when making changes. Quit, or continue with caution.
(2) An edit session for this file crashed.
    If this is the case, use ":recover" or "vim -r test.txt"
    to recover the changes (see ":help recovery").
    If you did this already, delete the swap file ".test.txt.swp"
    to avoid this message.

Swap file ".test.txt.swp" already exists!
[O]pen Read-Only, (E)dit anyway, (R)ecover, (D)elete it, (Q)uit, (A)bort: █
```

Lab 2.1 Editing Files

Estimated Time: 20 Minutes

Tip: Try the `vimtutor` command. It will teach you how to use vim.



POP QUIZ:

What key takes you into Normal mode in vi?

- A. n
- B. esc
- C. nm
- D. enter



POP QUIZ:

What key takes you into Normal mode in vi?

- A. n
- B. **esc**
- C. nm
- D. enter

"What can you do in normal mode?"



POP QUIZ:

How can I properly exit a file without saving?

- A. Close the terminal session
- B. :q
- C. :qq
- D. :q!



POP QUIZ:

How can I properly exit a file without saving?

- A. Close the terminal session
- B. :q
- C. :qq
- D. :q!

"What happens if we close the terminal session?"



POP QUIZ:

What is the proper way to save and exit a file?

- A. :wq
- B. :save
- C. :w
- D. :zz



POP QUIZ:

What is the proper way to save and exit a file?

- A. :wq
- B. :save
- C. :w
- D. :zz

"How can I save without exiting?"



Linux: System Monitoring

System monitoring is how administrators stay ahead of problems. It reveals what the system is doing behind the scenes — which processes are using resources, how the network is behaving, and whether performance is steady or degrading over time. Without monitoring, issues are discovered only after users are affected.

A good admin doesn't guess; they observe. Knowing how to monitor your system means you can diagnose issues quickly, plan capacity intelligently, and maintain reliability with confidence.



System Monitoring

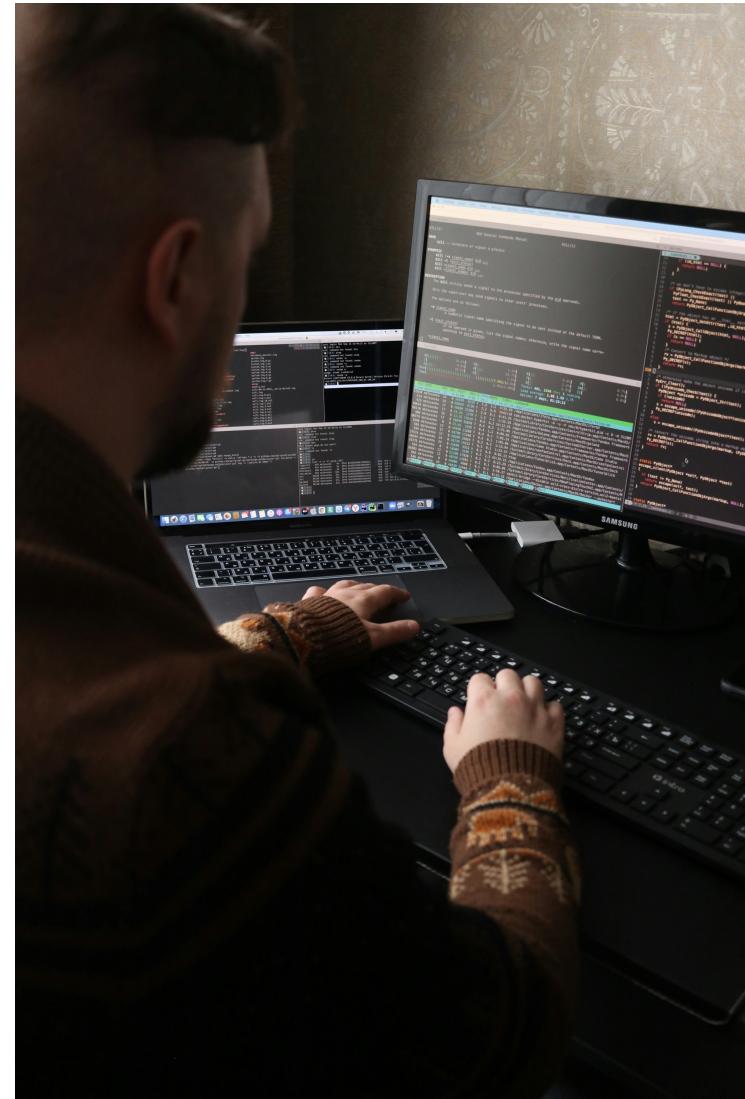
System monitoring is the practice of observing and analyzing system performance to ensure reliability, availability, and efficiency.

Why It Matters:

- Detect issues before they cause downtime
- Understand how CPU, memory, disk, and network resources are used
- Set up alerts for unusual activity or failures
- Support capacity planning and performance tuning
- Improve visibility for security and compliance

💡 Key Takeaway:

Monitoring turns system data into actionable insights — it's how engineers maintain stability and prevent outages.



Linux Processes

Program vs Process

Program

- A **set of instructions** stored on disk (e.g., /usr/bin/python)
- **Passive** — it does nothing until executed
- Exists as a **file**, not yet in memory

Process

- A **running instance** of a program
- **Active** — currently executing on the CPU
- Has its own **PID (Process ID)**, memory space, and entry in the **process table**
- Created when a program is launched

 *In short: A program is the recipe; a process is the dish being cooked.*

Linux Processes

Simulating Load with the `stress` Command

Purpose:

`stress` is a simple workload generator used to simulate CPU, memory, and I/O pressure — ideal for practicing monitoring and performance analysis.

Example Usage:

```
sudo dnf install stress -y # or apt install stress  
stress --cpu 2 --vm 1 --vm-bytes 128M --timeout 60s &
```

This command:

- Creates 2 CPU workers
- Allocates 128 MB of memory
- Runs for 60 seconds
- Great for creating processes to view with `top`

Linux Operating System

The top Command

A foundational monitoring command in Linux is **top**.

It provides a real-time view of CPU, memory, and process activity.

Try sorting the display:

- Press **P** → sort by **CPU usage**
- Press **M** → sort by **Memory usage**



Great for a quick snapshot of system health

```
top - 04:13:07 up 4 days, 6:53, 1 user, load average: 0.57, 0.13, 0
Tasks: 107 total, 3 running, 104 sleeping, 0 stopped, 0 zombie
%Cpu(s): 100.0 us, 0.0 sy, 0.0 ni, 0.0 id, 0.0 wa, 0.0 hi, 0.0 si
MiB Mem : 904.8 total, 281.6 free, 182.7 used, 440.6 buff/
MiB Swap: 0.0 total, 0.0 free, 0.0 used. 586.6 avail
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+
192150	ec2-user	20	0	3528	108	0	R	100.0	0.0	0:21.88
192151	ec2-user	20	0	3528	108	0	R	99.7	0.0	0:21.86
1	root	20	0	172592	17600	10832	S	0.0	1.9	0:19.32
2	root	20	0	0	0	0	S	0.0	0.0	0:00.11
3	root	0	-20	0	0	0	I	0.0	0.0	0:00.00
4	root	0	-20	0	0	0	I	0.0	0.0	0:00.00
5	root	0	-20	0	0	0	I	0.0	0.0	0:00.00
6	root	0	-20	0	0	0	I	0.0	0.0	0:00.00
8	root	0	-20	0	0	0	I	0.0	0.0	0:00.00

Linux Processes

The ps Command

ps (process status) displays information about active processes running on the system.

Why It's Useful:

- See which programs are currently executing
- Check process IDs (PIDs) and parent-child relationships
- Identify CPU and memory usage per process
- Useful for troubleshooting or verifying background services

Common Example:

```
ps aux
```

Shows all processes for all users in a detailed list.

 *Think of ps as a snapshot of what's running at a single moment in time.*

Linux Processes

```
[ec2-user@ip-172-31-1-87 ~]$ ps aux | head -5
USER          PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root            1  0.0  1.8 172592 17600 ?        Ss Oct20   0:19 /usr/lib/sys
root            2  0.0  0.0     0     0 ?        S   Oct20   0:00 [kthreadd]
root            3  0.0  0.0     0     0 ?        I< Oct20   0:00 [rcu_gp]
root            4  0.0  0.0     0     0 ?        I< Oct20   0:00 [rcu_par_gp]
[ec2-user@ip-172-31-1-87 ~]$
[ec2-user@ip-172-31-1-87 ~]$ ps aux --sort=-%mem | head -5
USER          PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root          827  0.0  8.5 131496 79000 ?        Ss Oct20   0:13 /usr/lib/sys
root        1568  0.0  2.1 1240436 19504 ?       Ssl Oct20   0:10 /usr/bin/amaz
root            1  0.0  1.8 172592 17600 ?        Ss Oct20   0:19 /usr/lib/sys
systemd+    1263  0.0  1.6  22548 14996 ?       Ss Oct20   0:00 /usr/lib/sys
[ec2-user@ip-172-31-1-87 ~]$
```

Linux Processes

Understanding ps aux Output

The ps aux command lists **all running processes** in a detailed snapshot — similar to top, but static.

Key Columns Explained:

- **USER** – Owner of the process
- **PID** – Process ID
- **%CPU / %MEM** – CPU and memory usage percentages
- **VSZ / RSS** – Virtual and resident memory sizes
- **TTY** – Terminal associated with the process (if any)
- **STAT** – Process state (e.g., R = running, S = sleeping, Z = zombie)
- **START** – When the process began
- **TIME** – Total CPU time used
- **COMMAND** – The program or command that started the process

 *ps aux provides a quick snapshot of system activity — while top continuously updates in real time.*

Linux Processes

Every process on a Linux system is **started by another process** — its **parent**. The relationship is tracked using **PIDs** and **PPIDs** (Parent Process IDs).

Example:

```
ps -ef
```

Common columns:

- **UID** – User who started the process
- **PID** – Process ID
- **PPID** – Parent Process ID
- **CMD** – Command that launched the process



Concept:

- The **parent process** creates a **child process** using the `fork()` system call.
- When a **parent process ends**, its children either receive a `SIGHUP` signal and terminate (if attached to the same terminal), or become **orphans** that are automatically adopted by `systemd` (PID 1).

 *Understanding parent-child relationships helps trace where a process came from and how it was started.*

Finding and Managing Processes

To locate specific processes, you can **search process lists** using ps and grep:
`ps -ef | grep stress`

If you only need the **Process ID (PID)**:
`pgrep sleep # prints every PID matching sleep`

Once you have the PID, you can take action:
`kill <PID> # Politely ask the process to terminate (SIGTERM)`
`kill -9 <PID> # Force kill (SIGKILL) — cannot be ignored`

You can also use the `pkill` command to kill any process that has `sleep` in it.
`pkill -9 stress # Force kills all processes with "stress" in the name`

 *Use -9 only if the process doesn't respond to a normal kill — it bypasses cleanup routines.*

Checking System Resource Usage

df – Disk Free Space

`df -h`

Shows available and used disk space for all mounted file systems.
(*-h = human-readable sizes like GB, MB*)

du – Disk Usage

`du -h ~`

Displays how much space a file or directory uses.

free – Memory Usage

`free -h`

Shows total, used, and available system memory and swap space.

 *These commands help you quickly assess disk and memory usage when troubleshooting or monitoring system health.*



Finding System Information

To learn more about your system, start with the **uname** command:

```
uname -a
```

Displays all system information, including kernel name, version, release, architecture, and hostname.

Other useful **uname** options:

- **uname -s** → Kernel name
- **uname -r** → Kernel release
- **uname -m** → Machine hardware (architecture)
- **uname -n** → System hostname

 Use **man uname** to view the manual and explore all available options.

You can also check system details with:

```
hostnamectl # System name, OS, and kernel info
```

Finding System Information

System Information details are essential when **installing software, compiling source code, or downloading the correct package version.**

It also helps with **troubleshooting**, ensuring commands, drivers, and binaries match your system's environment.

 *Knowing your system's architecture and kernel saves time and prevents compatibility issues.*

```
[ec2-user@ip-172-31-1-87 ~]$ hostnamectl
  Static hostname: ip-172-31-1-87.us-west-1.compute.internal
    Icon name: computer-vm
      Chassis: vm
    Machine ID: ec21f4760dd031654c602cf85faff450
      Boot ID: f9630b3d60e6446eaa91c5e276e907c2
  Virtualization: amazon
Operating System: Amazon Linux 2023.9.20251014
      CPE OS Name: cpe:2.3:o:amazon:amazon_linux:2023
        Kernel: Linux 6.1.155-176.282.amzn2023.x86_64
      Architecture: x86-64
  Hardware Vendor: Amazon EC2
      Hardware Model: t3.micro
Firmware Version: 1.0
[ec2-user@ip-172-31-1-87 ~]$
```

Useful System Commands

`lscpu`

Displays detailed information about the CPU architecture, including cores, threads, and virtualization support.

`lscpu`

`uptime`

Shows how long the system has been running, the number of users, and the load average.

`uptime`

`iostat`

Reports CPU utilization and disk I/O statistics, useful for performance analysis.

`iostat`

`vmstat`

Provides a snapshot of system performance, showing processes, memory, paging, block I/O, and CPU usage.

`vmstat`

 *These tools help you understand how the system is performing under different workloads.*

User Login and Session Commands

`who`

Shows who is currently logged in and where they're connected from.

`who`

`w`

Displays logged-in users plus what each user is currently doing.

`w`

`last`

Lists the login history, including previous users, login times, and session durations.

`last`

 *These commands help track active sessions, monitor user activity, and review login history for auditing or troubleshooting.*

Network Monitoring

The ss Command (Socket Statistics)

What It Is:

ss is a modern replacement for netstat — faster, built into most Linux distributions, and part of the core system utilities.

Common Uses:

```
ss -tuln  # Show TCP/UDP listening ports  
ss -tp    # Show active TCP connections with process info
```

Why It's Useful:

- Verify that services are listening on the correct ports
- Check active connections and who's connecting
- Helpful for network troubleshooting and security checks

 *ss gives you a clear, real-time view of network sockets — essential for validating running services.*

```
[ec2-user@ip-172-31-1-87 ~]$ sudo ss -tlnp  
State      Recv-Q      Send-Q      Local Address:Port      Peer Address:Port      Process  
LISTEN      0            128          0.0.0.0:22          0.0.0.0:*          users:(("sshd",pid=1572,fd=3))  
LISTEN      0            128          [::]:22            [::]:*          users:(("sshd",pid=1572,fd=4))  
[ec2-user@ip-172-31-1-87 ~]$
```

Network Monitoring

Network Interfaces and IP Information

Every system administrator must know how to view **network interfaces**, **IP addresses**, and **routing information**.

The `ip` command provides all of this in one place:

```
ip addr show      # Display IP addresses and interfaces  
ip link show     # Show network interfaces and their states  
ip route show    # View routing table information
```

 *The ip command replaces older tools like ifconfig and route — it's the modern standard for network management.*

Command	Layer	Shows	Connected To
<code>ip link show</code>	2 (Data Link)	Interfaces, MACs, state	Switch port
<code>ip addr show</code>	3 (Network)	IPs, subnets, broadcast	Logical network info
<code>ip route show</code>	3 (Network)	Routing table	Path decisions

Network Monitoring

When you run:

```
ip address show
```

you'll see something like:

```
inet 192.168.1.10/24 brd 192.168.1.255 scope global eth0
```

Here's what that means:

- 192.168.1.10 → your **IP address**
- /24 → the **CIDR notation** (Classless Inter-Domain Routing) — tells you the **subnet mask**
- /24 translates to 255.255.255.0, which means:
 - The first **24 bits** represent the **network portion**
 - The remaining **8 bits** are for **hosts**
- 192.168.1.255 → Broadcast Address

The CIDR suffix (/24, /16, /20, etc.) tells you **how large your subnet is** and which range of IPs belong to your **network**.

 CIDR = *the size and boundaries of your network.*

Network Monitoring

⌚ Understanding `ip route show`

Example output:

```
default via 192.168.1.1 dev eth0  
192.168.1.0/24 dev eth0 proto kernel scope link src 192.168.1.10
```

Let's decode it:

- **default via 192.168.1.1 dev eth0**
 - The **default route** is used when no other route matches.
 - via 192.168.1.1 → the **gateway or router** the system sends unknown traffic to (usually your router).
 - dev eth0 → the **network interface** used to reach it.
- **192.168.1.0/24 dev eth0**
 - This is your **local network route**.
 - /24 → subnet mask 255.255.255.0
 - dev eth0 → packets to this subnet stay **within the LAN** (no gateway needed, uses the switches).
 - src 192.168.1.10 → the **source IP** your system uses for this network.

💡 In short:

- default = route for *everything else* (internet, other networks)
- CIDR routes (like 192.168.1.0/24) = *local subnets* your system directly knows about

System Logs

Recent Kernel Messages

```
sudo dmesg | tail -20
```

Shows the last 20 messages from the kernel, such as hardware events or driver activity.

Recent System Logs

```
sudo journalctl -n 20 --no-pager
```

Displays the last 20 log entries collected by the systemd journal.

journalctl is the **go-to command for viewing logs** from services, the kernel, and system events — all in one place.

 *Use these commands to quickly check system health and recent activity without digging through log files.*

Lab 2.2 System Monitoring

Estimated Time: 35 Minutes



POP QUIZ:

What command will print every process with common headers to standard output?

- A. pt --all
- B. top
- C. ps
- D. ps aux



POP QUIZ:

What non-interactive command will print every process with common headers to standard output?

- A. pt --all
- B. top
- C. ps
- D. ps aux

"What does ps without options do?"



POP QUIZ:

How can I view the size of every file in a directory?

- A. du -h /path
- B. df -h /path
- C. sizeof /path
- D. free /path



POP QUIZ:

How can I view the size of every file in a directory?

- A. du -h /path
- B. df -h /path
- C. sizeof /path
- D. free /path

"What is the difference between du and df?"



POP QUIZ:

Which commands can I use to find the correct architecture type of my system?

- A. uname -a
- B. vmstat
- C. lscpu
- D. hostnamectl



POP QUIZ:

Which commands can I use to find the correct architecture type of my system?

- A. `uname -a`
- B. `vmstat`
- C. `lscpu`
- D. `hostnamectl`

"What does vmstat do again?"



POP QUIZ:

Which is the go-to tool for viewing logs on your OS?

- A. /var/log
- B. dmesg
- C. journalctl
- D. who



POP QUIZ:

Which is the go-to tool for viewing logs on your OS?

- A. /var/log
- B. dmesg
- C. journalctl
- D. who

"What is the difference between these commands?"



POP QUIZ:

What command will show the TCP services that are listening and what ports they are bound to?

- A. sudo ss -tlnp
- B. ip services show
- C. ss -ulnp
- D. ip link show



POP QUIZ:

What command will show the TCP services that are listening and what ports they are bound to?

- A. sudo ss -tlnp
- B. ip services show
- C. ss -ulnp
- D. ip link show

"What do the -tlnp options do?"



POP QUIZ:

What command can show the range of ips in your network interface?

- A. ip link show
- B. ip address show
- C. ip route show
- D. all of the above



POP QUIZ:

What command can show the range of ips in your network interface?

- A. ip link show
- B. ip address show
- C. ip route show
- D. all of the above

"What do the other commands show?"



Linux: Jobs

Jobs in Linux represent the tasks your system is currently running — whether in the foreground, background, or paused. Understanding how to manage jobs gives you control over your workload without restarting commands or opening new terminals.

A skilled admin knows how to suspend, resume, and terminate jobs as needed. It's not just about multitasking; it's about efficiency. Mastering job control means you can manage processes confidently, stay organized, and keep the system responsive while you work.



Linux: Jobs

Any task running in Linux is considered a **job**.

A job running in the **foreground** uses your terminal and **blocks other commands** until it finishes.

A job running in the **background** allows **multiple tasks** to run simultaneously without interrupting your terminal.

 *Foreground jobs demand your attention — background jobs let you multitask.*



Linux: Jobs

Start in the **background**:

```
sleep 100 &
```

Pause a **foreground** job (sends **SIGTSTP**):

```
sleep 200
```

```
# Press: Ctrl + Z
```

List jobs:

```
jobs
```

Resume the most recent *stopped* job in the background
(sends **SIGCONT**):

```
bg %+
```

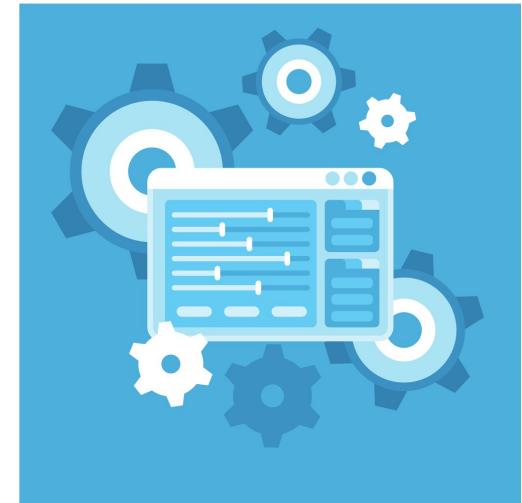
(Optional) Bring the most recent job to the foreground:

```
fg %+
```

Ctrl+Z = SIGTSTP (stop).

bg = SIGCONT (resume in background).

fg resumes in foreground.



Linux: Referencing and Managing Jobs

You can reference jobs using **job symbols** instead of PIDs:

- `%+` → Most recent job
- `%-` → Second most recent job
- `%3` → Specific job number (from `jobs` output)

You can **terminate a job** using either its **PID** or its **job symbol**:

```
kill %+      # Kill most recent job
kill %-      # Kill second most recent
kill %3      # Kill job number 3
```

 *Job symbols make it easy to manage background tasks without remembering PIDs.*

```
[ec2-user@ip-172-31-1-87 ~]$ sleep 100 &
[3] 217744
[ec2-user@ip-172-31-1-87 ~]$ sleep 200
^Z
[4]+  Stopped                  sleep 200
[ec2-user@ip-172-31-1-87 ~]$ jobs
[2]-  Stopped                  sleep 200
[3]  Running                  sleep 100 &
[4]+  Stopped                  sleep 200
[ec2-user@ip-172-31-1-87 ~]$ bg %-
[2]- sleep 200 &
[ec2-user@ip-172-31-1-87 ~]$ jobs
[2]   Done                    sleep 200
[3]-  Running                  sleep 100 &
[4]+  Stopped                  sleep 200
[ec2-user@ip-172-31-1-87 ~]$ kill %+
[4]+  Stopped                  sleep 200
[ec2-user@ip-172-31-1-87 ~]$ 
```

Linux: kill command

The `kill` command isn't just for ending processes — it's used to **send signals** to them.

You can list all available signals with:

```
kill -l
```

```
Kill -19 $PID # sends SIGSTOP to pause a process
```

Common Signals:

- **SIGTERM (15)** – Politely asks a process to terminate (default signal).
- **SIGKILL (9)** – Forcefully stops a process immediately (cannot be ignored).
- **SIGINT (2)** – Interrupts a process (like pressing **Ctrl+C**).
- **SIGHUP (1)** – Sent when a terminal disconnects; often used to reload configs.
- **SIGSTOP (19)** – Pauses a process (cannot be caught or ignored).
- **SIGCONT (18)** – Resumes a paused process.

 Think of `kill` as a way to **communicate** with processes — not always to destroy them.

Linux: No Hanging Up

The `nohup` command keeps a process running **even if the terminal is closed or disconnected**.

It prevents the kernel from sending the **SIGHUP** (hangup) signal when the **parent process** (your shell) terminates.

This is particularly useful for **long-running tasks** — like database updates or backups — that might otherwise stop if your SSH session ends.

When the parent shell exits, the process becomes an **orphan**, which is then **adopted by systemd** (or **init** on older systems), changing its **PPID** to 1.

Linux: nohup demo (gif)

```
[ec2-user@ip-172-31-1-87 ~]$ nohup sleep 200 &
[1] 226415
[ec2-user@ip-172-31-1-87 ~]$ nohup: ignoring input and appending output to 'nohu
p.out'

[ec2-user@ip-172-31-1-87 ~]$ ps -ef | grep [s]leep
ec2-user 226415 226381 0 00:28 pts/0    00:00:00 sleep 200
[ec2-user@ip-172-31-1-87 ~]$ echo $$
226381
[ec2-user@ip-172-31-1-87 ~]$
```

Notice the PPID is the Shells PID

Linux: Processes



Exploring lsof – List Open Files

The lsof command lists all open files and the processes using them.

Remember, in Linux **everything is a file** — from sockets and devices to logs and executables.

View which files a process has open:

```
lsof -p <PID>
```

See which process is holding a file open:

```
lsof <filename>
```

- Identify what started a process (the executable file is listed as txt)
- Troubleshoot “file is busy” or “device in use” errors

Tip: You can also use lsof -i to see open network connections and ports.

Linux: Processes

Slide: Process Priority and nice

The “nice” value controls how “polite” a process is when sharing CPU time.

- Lower **nice** value → higher priority (more CPU time)
- Higher **nice** value → lower priority (shares more)

 Try this demo:

```
sudo nice -n 10 stress --cpu 1 --timeout 30s &
sudo nice -n -10 stress --cpu 1 --timeout 30s &
ps -eo pid,cmd,%cpu,nice | grep [s]tress
```

 *The nicer a process is, the more it shares. The less nice it is, the more it wants!*

Linux: Processes

💀 Zombie Processes

A **zombie process** is one that has **finished execution** but still appears in the process table because its **parent hasn't collected its exit status**.



Characteristics:

- State shows as Z or <defunct> in ps
- Consumes **no CPU or memory**, just an entry in the process table
- Cleaned up automatically when parent exits or calls wait()

Example:

```
ps -eo pid,ppid,stat,cmd | grep Z # look for stat Z for zombie processes
```

💡 *A few zombies aren't harmful, but many can signal a misbehaving parent process. You can eventually run out PIDs to issue*

Linux: Processes

Slide: Linux Process Status Codes

Processes in Linux move through several states depending on what they're doing.

You can view them with:

```
ps -eo pid,stat,cmd
```

Code	Meaning	Description
R	Running	Actively using CPU or ready to run (in run queue).
S	Sleeping (Interruptible)	Waiting for an event (e.g., I/O or user input). Most processes are here.
D	Uninterruptible Sleep	Waiting on kernel or hardware I/O — can't be killed until it returns.
T	Stopped	Suspended by a signal (e.g., Ctrl+Z or kill -STOP).
Z	Zombie	Finished execution, but parent hasn't read its status.
X	Dead	Terminated and removed (rarely seen).

Lab 2.3 Linux Jobs

Estimated Time:

15 Minutes (Jobs)

15Minutes (Process Management)



POP QUIZ:

How can a job run in the background?

- A. add ! to end of command
- B. Ctrl+C
- C. add & to end of command
- D. Piping the command to nohup



POP QUIZ:

How can a job run in the background?

- A. add ! to end of command
- B. Ctrl+C
- C. add & to end of command
- D. Piping the command to nohup

"How else can we send a job to background?"



POP QUIZ:

A process refuses to terminate, how can we force it to terminate?

- A. Kill -9
- B. Kill -19
- C. nohup
- D. bg



POP QUIZ:

A process refuses to terminate, how can we force it to terminate?

- A. Kill -9
- B. Kill -19
- C. nohup
- D. bg

"What does kill -19 do?"



POP QUIZ:

How can the most recent job be started?

- A. jobs start recent
- B. fg %-
- C. bg %-
- D. bg %+



POP QUIZ:

How can the most recent job be started?

- A. jobs start recent
- B. fg %-
- C. bg %- "What is %- for?"
- D. bg %+



POP QUIZ:

Which process nice value will have the highest priority?

- A. -20
- B. 20
- C. 0
- D. -19



POP QUIZ:

Which process nice value will have the highest priority?

- A. -20 "Why not -20?"
- B. 20
- C. 0
- D. -19



Linux: Operators

Linux operators are the **symbols** that control how **commands interact** — they connect, redirect, and manage the flow of information between processes. Understanding them transforms single commands into powerful workflows.

A skilled admin uses operators to **chain tasks, redirect input and output, and handle errors gracefully**. It's not just about running commands; it's about precision and automation. Mastering operators means you can build efficient, reliable command sequences that do exactly what you intend.



Linux: Logical Operators

The special variable `$?` stores the **exit code** of the most recently executed command.

- A value of 0 indicates **success**.
- Any **non-zero** value indicates **failure** — the meaning of each code depends on the command (see the **manual** for details).

Example:

```
echo hello  
echo $? # will echo "0" indicating success  
0
```

Why It Matters:

Logical operators `&&` and `||` use the exit code to decide what happens next:

- `cmd1 && cmd2` → run cmd2 **only if cmd1 succeeds**
- `cmd1 || cmd2` → run cmd2 **only if cmd1 fails**

 *Exit codes are how Linux commands “communicate” success or failure to each other.*

Linux: Logical Operators

Examples:

```
echo hello && echo "Success" || echo "Failure"
```

- ✓ Both echo commands succeed → prints *hello* and *Success*

```
echo hello && ech "Success" || echo "Failure"
```

(ech is misspelled → fails)

- prints hello and Failure, since the second command returned a non-zero exit code.

```
false || echo "This runs because 'false' failed"
```

- false always returns failure → triggers the command after ||.

```
ech hello ; echo world
```

- The semicolon ; always runs the next command — success or failure doesn't matter. World will be printed after the error.

💡 *&& = run next if success // = run next if failure ; = always run next*

Linux: Logical Operators

`||` lets you run a fallback command when the previous one **fails**.

This is useful for detecting errors and responding immediately.

```
curl some-endpoint.com || echo "some-endpoint.com not available"
```

If you want to **stop the script** when a command fails, use a block with `exit 1`:

```
curl some-endpoint.com || { echo "Endpoint not available"; exit 1; }
```

 *// can provide simple error messages or stop execution entirely when something goes wrong.*

Linux: File Descriptors and Redirection

Every Linux process uses three standard file descriptors:

- 0 – **stdin** (standard input)
- 1 – **stdout** (standard output)
- 2 – **stderr** (standard error)

Redirecting Output:

```
echo "hello world" > myfile  
echo "hello again" > myfile  
cat myfile
```

> writes (overwrites) output to a file.

You can also write explicitly with 1> — it's the default for stdout.

Appending Output:

```
echo "new line" >> myfile
```

>> appends instead of overwriting.



Linux: Redirecting Standard Error (stderr)

Standard error uses **file descriptor 2**, separate from standard output.

```
ls /fakefolder 2> errors.log
```

→ The error message is **redirected** to errors.log instead of the screen.

```
ls /etc 2> errors.log
```

→ No error here — the output still prints to the screen (stdout), and errors.log remains **empty**.

```
ls /fakefolder > output.log
```

✗ Command fails → error message still prints to screen.
output.log is **empty** because there was no standard output.

💡 *2> only captures **errors** — normal output still goes to the terminal.*

Linux: File Descriptors and Redirection

Redirecting Both stdout and stderr

You can redirect **standard output** (1) and **standard error** (2) separately:

`command 1> out.txt 2> error.txt`

- Output goes to `out.txt`
- Errors go to `error.txt`

To redirect **both** to the same file:

`command > all.txt 2>&1`

- `>` redirects `stdout` to `all.txt`
- `2>&1` sends `stderr` (2) to the same place as `stdout` (1)

 The `&` means “use the file descriptor, not a file name.”

Thus `2>&1` means “send errors where standard output is currently going.”

Linux: Ignoring Errors Gracefully

Sometimes you don't care if a command fails — or you just want to hide its errors.

`command 2> /dev/null || true`

`2> /dev/null` discards **error messages**

`|| true` forces the command to **succeed** even if it fails

`/dev/null` is a **special device** that discards everything written to it — like a *black hole* for output you don't care about.

`|| true` is useful when `set -e` is enabled (covered later), so one failure doesn't stop the entire script.

 *Use with caution — this hides problems you might later need to debug.*

Linux: Input Redirection

Symbol	Name	Description
<	Input redirection	Takes input from a file instead of the keyboard. Example: cat < file.txt
<<	Here Document (heredoc)	Passes a block of text to a command as input until a delimiter. Example: cat << EOF Hello EOF
<<<	Here String	Sends a single line/string to a command as input. Example: cat <<< "Hello world"

<, <<, and <<< are all forms of input redirection. They feed data *into* commands through `stdin` (file descriptor 0) instead of you typing it interactively.

Linux: Input Redirection

```
[ec2-user@ip-172-31-1-87 ~]$ cat math.txt  
5+5  
10*100  
4+4  
[ec2-user@ip-172-31-1-87 ~]$ bc < math.txt # inputs the file into bc  
10  
1000  
8  
[ec2-user@ip-172-31-1-87 ~]$ bc << EOF # builds a doc for input until EOF  
> 10+10  
> 4+4  
> EOF  
20  
8  
[ec2-user@ip-172-31-1-87 ~]$ bc <<< 5*5 # inputs the string into bc  
25  
[ec2-user@ip-172-31-1-87 ~]$ cat << EOF > data.txt  
> 10, 20, 30  
> 40, 50, 60  
> EOF  
[ec2-user@ip-172-31-1-87 ~]$ cat data.txt  
10, 20, 30  
40, 50, 60  
[ec2-user@ip-172-31-1-87 ~]$
```

Linux: The Pipe Operator (|)

The **pipe** (|) sends the **output** of one command into the **input** of another, allowing multiple commands to work together efficiently.

Basic Examples:

```
ls | grep "log" # Shows only files containing "log".  
ps aux | grep ssh # Filters running processes for "ssh"
```

Advanced Example (Multiple Pipes + Redirection):

```
cat /var/log/syslog | grep error | wc -l > error_count.txt || echo "Log check failed"
```

 You can chain multiple pipes, redirect outputs, and even handle failures — all in one line.



Lab 2.4 Operators

Estimated Time: 20 Minutes



POP QUIZ:

What will the following command output?

`ls | grep txt 2> /dev/null`

- A. nothing
- B. All files with txt in its name
- C. error
- D. grep txt



POP QUIZ:

What will the following command output?

ls | grep txt 2> /dev/null

- A. nothing
- B. All files with txt in its name
- C. error
- D. grep txt

"Why does it output even with 2>?"



POP QUIZ:

What will the following command output?

```
echo "hello" 1> hello.txt && echo "hello"
```

- A. hello hello
- B. hello 1> hello.txt hello
- C. error
- D. hello



POP QUIZ:

What will the following command output?

```
echo "hello" 1> hello.txt && echo "hello"
```

- A. hello hello
- B. hello 1> hello.txt hello
- C. error
- D. hello

"Which echo output to stdout ?"



POP QUIZ:

What will the following command output?
`echo "hello world" 2> /dev/null || echo "fail"`

- A. hello world
- B. hello 1> hello.txt hello
- C. error
- D. hello



POP QUIZ:

What will the following command output?

```
echo "hello world" 2> /dev/null || echo "fail"
```

"What is the purpose of /dev/null ?"

- A. hello world
- B. hello 1> hello.txt hello
- C. error
- D. hello



POP QUIZ:

What command will count how many lines are in a file?

- A. grep -c myfile.txt
- B. linecount myfile.txt
- C. wc -l < myfile.txt
- D. myfile.txt | wc -l



POP QUIZ:

What command will count how many lines are in a file?

- A. grep -c myfile.txt
- B. linecount myfile.txt
- C. **wc -l < myfile.txt**
- D. myfile.txt | wc -l

"How can we make A. and D. work?"



Lab 2.5 Monitoring Challenge

Estimated Time: 60 Minutes



Linux Day 2 Complete

🎉 Great Work Today!

Today you learned how to:

- Edit and manage files using Vim
- Monitor system performance and processes
- Control and manage Linux jobs
- Use **operators** to connect, automate, and handle commands efficiently

Give yourself a pat on the back — you've taken another big step toward becoming an **effective and confident System Administrator**.

💡 *Keep practicing — mastery in Linux comes from curiosity, repetition, and exploration.*

