1. Introduction & Basics

1. What is Linux ?

Answer: "Linux is an open-source, Unix-like operating system kernel. It acts as the bridge between software applications and the hardware. In a DevOps context, it is the preferred OS for servers because of its stability, security, and flexibility."

- **Simple Explanation:** It's the engine that runs most of the internet (servers, cloud).
- **Real-World Example:** Android phones, AWS EC2 instances, and Docker containers all run on Linux kernels.

2. Difference between Linux and Windows

Answer: "The main difference is the architecture and philosophy. Linux is open-source and uses a monolithic kernel, treating 'everything as a file'. Windows is proprietary and uses a drive-letter system (C:, D:). Linux is preferred for servers due to its command-line efficiency and lower resource overhead."

- **Key Differences:**
    - **File System:** Linux is a single tree (Starts at /); Windows uses drive letters (C:\).
    - **Case Sensitivity:** Linux is sensitive (File.txt $\neq$ file.txt); Windows is not.
    - **Kernel:** Linux is Monolithic; Windows is Hybrid/Micro.

3. What is a Shell and a Terminal ?

Answer: "The Terminal is the interface (or window) that accepts user input. The Shell is the command-line interpreter that actually processes that input and executes commands. The terminal sends keystrokes to the shell."

- **Analogy:** The Terminal is the car dashboard (where you sit); the Shell is the engine logic that reacts when you press the gas pedal.
- **Real-World Example:**
    - **Terminals:** PuTTY, iTerm2, Git Bash.
    - **Shells:** Bash (Bourne Again Shell), Zsh, Sh.

4. What is a Kernel in Linux ?

Answer: "The Kernel is the core component of the OS. It has complete control over everything in the system. It manages hardware resources like CPU and memory, handles device drivers, and manages system calls from software applications."

- **Simple Explanation:** It’s the boss. It tells the CPU what to do and ensures one program doesn’t crash the whole computer.
- **Real-World Example:** When you open Chrome, the Kernel allocates RAM to it. If Chrome tries to access memory it shouldn't, the Kernel kills the process (OOM Killer).

5. What are Daemons ?

Answer: "A Daemon is a background process that runs continuously to handle requests or perform tasks, without direct user intervention. In Linux, they usually end with the letter 'd' (e.g., sshd, httpd)."

- **Simple Explanation:** Service programs that run silently in the background waiting for work.
- **Real-World Examples:**
    - sshd: Listens for incoming SSH connections.
    - dockerd: Manages Docker containers.
    - crond: Runs scheduled tasks.

6. What is the Linux Booting Process ?

Answer: "The Linux boot process follows 6 stages:

1. **BIOS/UEFI:** Performs hardware checks (POST) and looks for a bootable device.
2. **MBR/GPT:** Loads the bootloader from the disk.
3. **Bootloader (GRUB):** Lets you select the OS/Kernel to load.
4. **Kernel:** Mounts the root file system and initializes hardware drivers.
5. **Init (Systemd):** The first process (PID 1). It starts all other services.
6. **Runlevel/Target:** The system reaches its final state (e.g., Graphical or Multi-user CLI)."

- **Simple Mnemonic: B**ig **M**achines **G**et **K**ernels **I**n **R**unning.
- **Tip:** Interviewers ask this to see if you know how to troubleshoot a server that won't start.

2. File System & Navigation

7. Basic Linux File System Structure

Answer: "Linux uses a hierarchical file system structure that resembles an inverted tree, starting with the root directory (`/`). All other directories and files are children of this root. It follows the Filesystem Hierarchy Standard (FHS)."

- **Simple Explanation:** In Windows, you have C: and D: drives. In Linux, everything starts at `/`. Even if you plug in a USB drive, it appears as a folder inside this tree.
- **Key Directories:**
    - `/bin`: User binaries (programs like `ls`).
    - `/home`: Personal user files (like `C:\Users`).
    - `/var`: Variable data (logs, websites).
    - `/tmp`: Temporary files.
    - `/root`: Home directory for the superuser (root).

8. What is `/etc` used for ?

Answer: "The `/etc` directory stores system-wide **configuration files**. It contains the settings that control how the operating system and installed applications behave. It generally does not contain binary programs."

- **Real-World Example:**
    - To change DNS settings: Edit `/etc/resolv.conf`.
    - To add/view users: Edit `/etc/passwd`.
    - To configure Nginx: Edit `/etc/nginx/nginx.conf`.
- **Interview Tip:** Never delete files in `/etc` unless you know exactly what you are doing.

9. Difference between Absolute and Relative Path

Answer: "An **Absolute Path** is the full path to a file starting from the root (`/`). A **Relative Path** is the path relative to your current location (current working directory)."

- **Example:**
    - **Absolute:** `/home/user/documents/file.txt` (Address works from anywhere).
    - **Relative:** `documents/file.txt` (Only works if you are already inside `/home/user`).
- **Tip:** In automation scripts (cron jobs), **always use absolute paths** to prevent errors.

10. Common Navigation Commands (`ls`, `cd`, `pwd`)

Answer: "These are the most fundamental commands for moving around the system:

- `pwd`: Print Working Directory (shows exactly where you are).
- `ls`: Lists files and directories in the current folder.
- `cd`: Changes the directory."
- **Common Flags:**
    - `ls -l`: Long listing (shows permissions, owner, size).
    - `ls -a`: All files (including hidden files starting with `.`).
    - `cd ~`: Go straight to your home directory.
    - `cd ..`: Go back/up one directory.

11. File Operations (`cp`, `mv`, `rm`, `touch`, `mkdir`)

Answer: "These commands manage files and directories:

- `touch filename`: Creates an empty file or updates timestamps.
- `mkdir dirname`: Creates a new directory.
- `cp source dest`: Copies a file (`cp -r` for directories).
- `mv source dest`: Moves or Renames a file.
- `rm filename`: Removes/Deletes a file."
- **Mistake:** `rm -rf /` is the most dangerous command; it forces deletion of everything from the root down. Always double-check before pressing Enter with `rm`.

12. Viewing Content (`cat`, `less`, `more`, `head`, `tail`)

Answer: "Tools to read file content without opening an editor:

- `cat`: Dumps the entire file content to the screen.
- `less`: Allows scrolling through large files line by line (efficient).
- `head`: Shows the first 10 lines.
- `tail`: Shows the last 10 lines."
- **Real-World Example:**
    - Use `tail -f /var/log/syslog` to watch log files in **real-time** as events happen.
- **Interview Tip:** If asked how to view a 10GB file, say `less`, never `cat`.

13. Searching in Files (`grep`, `find`, `locate`)

Answer: "`find` and `locate` search for **files** by name or metadata. `grep` searches for **text patterns** inside the actual content of the files."

- **Commands:**
  - **Find file:** `find / -name "config.xml"` (Scans disk, slower but accurate).
  - **Find text:** `grep "Error" /var/log/syslog` (Finds the word "Error" inside the log).
  - **Locate:** `locate file.txt` (Uses a pre-built database, faster than find).

14. What is a Symbolic Link and Hard Link ?

Answer: "A **Symbolic (Soft) Link** is like a shortcut in Windows; it points to the original file's path. A **Hard Link** is a direct reference to the physical data (Inode) on the disk. It acts like a mirror copy."

- **Real-World Use:** Soft links are used in web servers (like Nginx/Apache) to link configuration files from a "storage" folder to an "active" folder.

15. What is the difference between Hard Link and Soft Link ?

Answer: "The key differences are:

1. **Deletion:** If the original file is deleted, the Soft Link becomes broken (useless). The Hard Link remains active because it points to the actual data.
2. **Partitions:** Soft links can point to files on different drives/partitions. Hard links cannot.
3. **Inode:** Soft links have a unique Inode number; Hard links share the same Inode as the original."

- **Command:** `ln -s source link` (Soft), `ln source link` (Hard).

16. How to check System Info (`uname`, `uptime`, `whoami`, `hostname`)

Answer: "These commands provide quick diagnostics:

- `uname -r`: Prints the Kernel version.
- `uptime`: Shows how long the system has been running and the **load average**.
- `whoami`: Shows the current logged-in user.

- `hostname`: Displays the machine's network name."
- **Tip:** `uptime` is the first command to run when investigating a slow server to see if the CPU is overloaded.

3. File Permissions & Ownership

17. File permissions (`chmod`, `chown`, `umask`)

Answer: "These commands control who can access files:

- `chmod`: Changes file mode bits (Read/Write/Execute permissions).
- `chown`: Changes the owner and group of a file.
- `umask`: Sets the default permissions for newly created files."
- **Simple Explanation:** `chmod` decides *what* you can do (read/write). `chown` decides *who* owns the file. `umask` is the default safety setting for new files.
- **Commands:**
    - `chmod 755 script.sh`: Give owner full control, everyone else read/execute.
    - `chown user:group file.txt`: Change ownership to specific user and group.
    - `umask 022`: Standard default (folders 755, files 644).

18. How to debug "Permission Denied" errors

Answer: "Permission denied errors happen when the current user lacks the necessary Read (`r`), Write (`w`), or Execute (`x`) bit on a file or its parent directory. To debug:

1. Check user identity: `whoami`.
2. Check file permissions: `ls -l filename`.
3. Check directory permissions: Ensure the user has execute (`x`) access on the folder to enter it."

- **Real-World Example:** If a web server (Nginx) cannot read an HTML file, run `ls -l index.html` to see if the 'others' column has 'r' (read) permission. If not, Nginx (which runs as a different user) is blocked.
- **Tip:** Don't just run `chmod 777` to fix it. That is a security risk. Give only the specific permission needed.

19. What is sudo and why not to overuse it ?

Answer: "`sudo` (SuperUser DO) allows a permitted user to execute a command as the superuser (root) or another user. It should not be overused because running commands as root bypasses all security checks, making it easy to accidentally delete system files or install malicious software."

- **Simple Explanation:** `sudo` is like borrowing the manager's key card. It opens every door, but if you trip and fall, you break expensive things.
- **Interview Tip:** Never say "I run everything with sudo." Say "I use sudo only when administrative privileges are strictly required."

20. How to switch users (su, `sudo -i`)

Answer: "To switch user contexts:

- `su username`: 'Switch User'. Requires the *target* user's password.
- `su - username`: Switches user and also loads their environment variables (recommended).
- `sudo -i` or `sudo su -`: Switches to the root user using *your own* password (if you are in the sudoers group)."
- **Real-World Scenario:** If you are deploying an app as the 'app_user', you switch to that user (`su - app_user`) to ensure all files created belong to that user, not root.

4. Users & Groups

21. User Management (`useradd`, `passwd`, su, sudo)

Answer: "These are the core commands for managing user identities and access:

- `useradd`: Creates a new user account.
- `passwd`: Sets or updates a user's password.
- `su`: Switches the current user context to another user.
- `sudo`: Executes a single command with superuser (root) privileges."
- **Simple Explanation:** `useradd` hires the employee; `passwd` gives them a key card; `sudo` gives them the manager's override code for specific tasks.

- **Commands:**
    - `useradd -m john`: Creates user 'john' with a home directory.
    - `passwd john`: Prompts to set a password for 'john'.
- **Interview Tip:** Always mention `useradd -m`. If you forget `-m`, the user won't have a home directory (`/home/john`), which causes errors later.

22. How to create and use `.bash_profile`, `.bashrc`, `.profile`

Answer: "These are hidden configuration files in a user's home directory that run automatically to set up the environment (like variables, aliases, and paths).

- `.bash_profile` (or `.profile`): Executed **once** when a user logs in (Login Shell).
- `.bashrc`: Executed **every time** a new terminal window is opened (Non-Login Shell)."
- **Real-World Example:**
    - You put `export PATH=$PATH:/opt/maven/bin` in `.bash_profile` so tools work immediately after login.
    - You put `alias ll='ls -l'` in `.bashrc` so you can use shortcuts in every new terminal tab.
- **Command to apply changes:** `source ~/.bashrc` (Reloads the file without restarting the terminal).

23. What is the difference between Login and Non-Login Shell ?

Answer: "A **Login Shell** is the first shell session you get after successfully authenticating (entering username/password or SSH key). It reads `~/.bash_profile`. A **Non-Login Shell** is a shell started *after* you are already logged in (e.g., opening a new terminal tab or running a shell script). It reads `~/.bashrc`."

- **Analogy:**
    - **Login Shell:** Unlocking the front door of your house (happens once).
    - **Non-Login Shell:** Opening a room door inside the house (happens many times).
- **Why it matters:** If you define a variable in `.bash_profile`, it might not appear in a new terminal window unless `.bash_profile` explicitly loads `.bashrc`.

5. Package Management

24. Package Management (`apt`, `yum`)

Answer: "Package managers are tools that automate the process of installing, upgrading, configuring, and removing software. They resolve dependencies automatically.

- **`apt` (Advanced Package Tool):** Used by Debian-based systems (Ubuntu, Debian).
- **`yum` / `dnf`:** Used by Red Hat-based systems (RHEL, CentOS, Fedora)."
- **Simple Explanation:** It is like the "App Store" or "Google Play Store" for Linux servers. You tell it what you want, and it downloads and installs it for you.
- **Real-World Example:**
    - To install a web server on Ubuntu: `sudo apt install nginx`
    - To install a web server on CentOS: `sudo yum install httpd`
- **Interview Tip:** Always mention `sudo apt update` before installing. This ensures the package list is fresh so you don't download outdated software.

25. How to zip and unzip files (`tar`, `gzip`, `unzip`)

Answer: "These are tools to compress (shrink) files to save space or bundle multiple files into a single archive for easier transfer.

- **`tar`:** The most common tool to bundle files (often called a 'tarball'). It preserves permissions.
- **`gzip`:** Compresses the file to reduce size.
- **`unzip`:** Extracts `.zip` files (common when sharing files with Windows users)."
- **Simple Explanation:** `tar` puts your clothes in a suitcase; `gzip` sits on the suitcase to squash it down.
- **Commands:**
    - **Create Archive:** `tar -cvf archive.tar /folder` (Create, Verbose, File).
    - **Extract Archive:** `tar -xvf archive.tar` (Extract, Verbose, File).
    - **Unzip:** `unzip file.zip`.
- **Tip:** A common interview question is "What do the flags `-xvf` stand for?" (Extract, Verbose, File).

6. Process & Services

26. Process Management (`ps`, `top`, `htop`, `kill`)

Answer: "These tools are used to monitor and control running programs:

- `ps`: specific snapshot of current processes.
- `top`: Real-time view of system resources and running processes.
- `htop`: A more user-friendly, colorful version of `top`.
- `kill`: Sends a signal to stop a process."
- **Simple Explanation:** `ps` is like taking a photo of the traffic; `top` is like watching a live video feed of the traffic.
- **Command:** `ps aux | grep java` (Finds all Java processes).

27. What is the difference between a Process and a Service ?

Answer: "A **Process** is any running instance of a program (e.g., opening a text editor). It usually ends when the task is done or the user closes it. A **Service** (or Daemon) is a background process designed to run continuously, often starting automatically at boot (e.g., a web server)."

- **Analogy:** A "Process" is like a contractor you hire for a one-day job. A "Service" is like a security guard who works 24/7.

28. What is `kill -9` vs `kill -15` ?

Answer: "These are signals sent to stop a process:

- **`kill -15` (SIGTERM):** The 'polite' kill. It asks the process to stop, allowing it to save data and close files properly. This is the default.
- **`kill -9` (SIGKILL):** The 'force' kill. It immediately terminates the process without cleanup. Use this only if the process is stuck/frozen."
- **Real-World Example:** Try `kill -15 PID` first. If the server doesn't stop after a minute, use `kill -9 PID` as a last resort.

29. Understanding `init` vs `systemd`

Answer: "`init` (SysVinit) was the traditional initialization system for Linux, which started services sequentially (one by one), making boot times slower. `systemd` is the modern replacement. It starts services in parallel (simultaneously), resulting in faster boot times and better dependency management."

- **Key Difference:** `init` scripts are complex bash scripts; `systemd` uses simpler `.service` unit files.

30. Service Management (`systemctl`, `service`)

Answer: "`systemctl` is the main command to control services in modern Linux (Systemd). `service` is the older command (legacy).

- Start: `systemctl start nginx`
- Stop: `systemctl stop nginx`
- Enable (Start on boot): `systemctl enable nginx`
- Status: `systemctl status nginx`"
- **Interview Tip:** Always use `systemctl` for modern distros like Ubuntu 16+, CentOS 7+, and RHEL 7+.

31. What are Zombie Processes ?

Answer: "A Zombie process (marked as `Z` in `top`) is a process that has completed execution but still has an entry in the process table. This happens because the parent process hasn't read the child's exit status yet."

- **Simple Explanation:** It's a "dead" process that is waiting for its parent to acknowledge it died. It doesn't use CPU or RAM, just a process ID (PID).
- **How to fix:** You cannot kill a zombie (it's already dead). You must kill the **parent** process to clean it up.

32. How to check open files and locks (`lsof`)

Answer: "`lsof` stands for **List Open Files**. It lists information about files opened by processes. Since 'everything is a file' in Linux, this includes network connections."

- **Real-World Example:**
    - "Why can't I delete this file?" -> Run `lsof /path/to/file` to see who is using it.

    - "Which process is using port 80?" -> Run `lsof -i :80`.

33. How to monitor system load (`uptime`, `load average`)

Answer: "`uptime` shows the current time, how long the system has been running, and the **load average** for the last 1, 5, and 15 minutes. **Load Average** represents the average number of processes waiting for CPU time."

- **Rule of Thumb:** If the load average is higher than the number of CPU cores you have, the system is overloaded.
    - Example: On a 4-core server, a load of 4.0 is 100% utilization. A load of 6.0 means processes are waiting (lag).

34. What is nohup and why it's used ?

Answer: "`nohup` (No Hang Up) runs a command that ignores the 'hangup' signal. This means the process keeps running even if you close the terminal or log out."

- **Command:** `nohup python script.py &`
- **Real-World Scenario:** You are running a long data migration script over SSH. If your internet disconnects, the script would normally die. using `nohup` ensures it finishes even if you disconnect.

7. Scripting & Shell Logic

35. Scripting Basics (Bash, .sh files, variables)

Answer: "Shell scripting is the process of writing a series of commands in a text file (usually with a `.sh` extension) to be executed by the shell. Variables are used to store data (like filenames or paths) that can be reused throughout the script."

- **Simple Explanation:** Instead of typing the same 10 commands every day, you put them in a file and run the file once.
- **Basic Syntax:**
    - `name="John"` (No spaces around `=`).
    - `echo "Hello $name"` (Use `$` to access the variable).

36. Loops and Conditionals in Bash

Answer: "These provide logic flow in scripts:

- **Conditionals (`if/else`):** Execute code only if a condition is true (e.g., 'If the file exists, delete it').
- **Loops (`for/while`):** Repeat a task multiple times (e.g., 'For every server in this list, run an update')."
- **Real-World Example:**
    - **Loop:** You have 50 log files to compress. Instead of doing it one by one, you write a `for` loop to compress all `*.log` files.
    - **Conditional:** Check if a directory exists before trying to create it to avoid errors.

37. What is a Shell Script vs. a Command ?

Answer: "A **Command** is a single instruction executed interactively in the terminal (e.g., `ls -l`). A **Shell Script** is a file containing a sequence of commands that are executed together to perform a complex task."

- **Analogy:**
    - **Command:** Telling a chef "Cut this onion."
    - **Script:** Giving the chef a full recipe card with steps (Cut onion, boil water, add pasta).

38. What is Shebang (`#!/bin/bash`) in shell scripts ?

Answer: "The Shebang (`#!`) is the absolute first line of a script. It tells the Linux kernel which interpreter should be used to execute the commands inside the file."

- **Examples:**
    - `#!/bin/bash`: Run using Bash.
    - `#!/usr/bin/python3`: Run using Python.
- **Tip:** If you forget the shebang, the system might try to run Python code as Bash commands, causing immediate errors.

39. What is a `trap` in bash scripting ?

Answer: "`trap` is a shell command used to catch signals (like `SIGINT` when you press Ctrl+C) and execute a specific function or cleanup task before the script exits."

- **Real-World Example:**
  - Your script creates a temporary file `/tmp/data.txt`. You use `trap` to ensure that even if the user cancels the script halfway (Ctrl+C), the script deletes that temporary file so it doesn't leave junk behind.
- **Command:** `trap "rm -f /tmp/tempfile" EXIT`

40. Exit code and $? in Linux

Answer: "Every command in Linux returns a status number called an **Exit Code** when it finishes.

- **`0`**: Success.
- **`1-255`**: Failure/Error. The variable `$?` stores the exit code of the *last* executed command."
- **Simple Explanation:** It's how the computer tells you if it did the job or failed.
- **Command:**
  - Run `ls existing_folder` -> then `echo $?` -> Output: `0`
  - Run `ls fake_folder` -> then `echo $?` -> Output: `2` (File not found)
- **Usage:** Used heavily in CI/CD pipelines to stop the build if a test fails.

41. How to write and use aliases (`alias`, `.bashrc`)

Answer: "An `alias` is a custom shortcut for a longer command. To make an alias permanent, it must be added to the user's shell configuration file (usually `.bashrc` or `.zshrc`)."

- **Simple Explanation:** It's like saving a contact in your phone. Instead of typing the full number every time, you just tap "Mom".
- **Commands:**
  - Temporary: `alias k='kubectl'`
  - Permanent: Add `alias ll='ls -l'` inside `~/.bashrc`, then run `source ~/.bashrc`.
- **Real-World Use:** DevOps engineers often shorten complex commands like `alias logs='tail -f /var/log/syslog'`.

42. What is PATH and how to modify it ?

Answer: "`PATH` is an environment variable that tells the shell which directories to search for executable files (commands) when you type a name. If a program is not in a directory listed in `PATH`, you must type its full location to run it."

- **Simple Explanation:** It's the list of places the computer looks for tools. If you buy a new tool but don't put it in the toolbelt (PATH), the computer can't find it.
- **Command to Modify:**
    - `export PATH=$PATH:/opt/new-tool/bin` (Adds `/opt/new-tool/bin` to the existing list).

43. Environment Variables (`export`, `.env`)

Answer: "Environment variables are dynamic values that affect the behavior of running processes. They are used to pass configuration settings (like API keys, database URLs, or debug modes) to applications without changing the code."

- **Command:** `export DB_HOST="localhost"` makes the variable available to child processes.
- **Real-World Example:** In Docker or Kubernetes, you never hardcode passwords. You inject them as environment variables (e.g., `MYSQL_PASSWORD`) so the app reads them at startup.

44. How to use `xargs` and Pipes (`|`)

Answer: "A **Pipe (`|`)** passes the *output* (text) of one command as input to the next. **`xargs`** is used when the second command doesn't accept text input but needs arguments (filenames). It converts the text stream into command-line arguments."

- **Simple Explanation:**
    - **Pipe:** Like a bucket brigade. One person passes water to the next.
    - **xargs:** Takes a list of items and feeds them one-by-one to a tool that processes them.
- **Command Example:**
    - `find . -name "*.log" | xargs rm` (Finds all log files and passes their names to `rm` to delete them).

45. Understanding stdin, stdout, stderr

Answer: "Linux has three standard data streams for every process:

1. **stdin (0):** Standard Input (usually keyboard input).
2. **stdout (1):** Standard Output (what you see on the screen).
3. **stderr (2):** Standard Error (where error messages go)."

- **Why it matters:** You often want to separate "real data" from "error messages".
- **Command:** `command > output.txt 2> errors.txt` (Saves success messages to `output.txt` and error messages to `errors.txt`).

46. How to use the `tee` command

Answer: "`tee` reads from standard input and writes it to **both** the standard output (screen) and one or more files simultaneously."

- **Simple Explanation:** It's a "T-junction" pipe. It splits the water flow so it goes to the sink AND the garden hose at the same time.
- **Real-World Use:** When installing software, you want to see the progress bar on screen, but also save the logs to a file for later review.
- **Command:** `echo "Log entry" | tee -a logfile.txt` (Appends to file while showing on screen).

47. What is `/dev/null` and why redirect output there ?

Answer: "`/dev/null` is a special file known as the "null device" or "bit bucket". Anything written to it is immediately discarded and disappears. It is used to suppress unwanted output."

- **Simple Explanation:** It is a black hole. Throw anything in, and it's gone forever.
- **Real-World Example:**
    - `cron` jobs often send emails every time they run. To stop this spam, you redirect output: `command > /dev/null 2>&1` (Silences both output and errors).

8. Disk Memory & Resource Management

48. Disk and Space Checking (`df`, `du`)

Answer: "These commands are used to monitor disk usage:

- **`df -h` (Disk Free):** Shows available disk space on all mounted filesystems in a human-readable format (GB/MB).
- **`du -sh` (Disk Usage):** Shows the size of a specific directory or file."
- **Simple Explanation:** `df` looks at the whole "hard drive" pie chart. `du` looks at how heavy one specific folder is.
- **Real-World Scenario:**
    - "Server is full!" -> Run `df -h` to see which partition is 100%.
    - "Which folder is using all the space?" -> Run `du -sh /var/log` to check log sizes.

49. How to check memory usage (`free -m`, `vmstat`)

Answer: "These commands diagnose RAM usage:

- **`free -m`:** Displays total, used, and free memory in MB. It also shows 'available' memory (which includes cache).
- **`vmstat`:** Provides a detailed breakdown of system performance, including memory, swap, IO, and CPU activity."
- **Key Metric:** In `free -m`, focus on the **"Available"** column, not just "Free". Linux caches files in RAM to speed things up, so "Free" might look low even if the system is healthy.

50. What is Swap Space and why it matters

Answer: "Swap space is a dedicated portion of the hard drive that acts as 'overflow' memory when physical RAM is full. If RAM is exhausted, the kernel moves inactive data to Swap to prevent a crash."

- **Trade-off:** Swap is much slower than RAM. If a server starts using Swap heavily (Swapping), performance will degrade significantly.
- **Interview Tip:** In Kubernetes (K8s), swap is usually disabled because it interferes with resource scheduling.

51. How to set file limits (`ulimit`)

Answer: "`ulimit` controls the resources available to the shell and processes started by it. The most common limit is **Open Files** (`-n`)."

- **Why it matters:** Linux has a default limit (often 1024) on how many files a user can open. High-traffic web servers (like Nginx) or databases need thousands of open connections (since every connection is a file).
- **Command:** `ulimit -n 65535` (increases the limit to 65k).

52. How to mount/unmount drives (`mount`, `umount`)

Answer: "In Linux, storage devices (USB, Hard Drives) must be attached to a specific directory in the file tree to be accessed.

- **`mount`:** Attaches the device filesystem to a directory.
- **`umount`:** Detaches the filesystem safely."
- **Command:**
    - `mount /dev/sdb1 /mnt/data` (Makes the drive accessible at `/mnt/data`).
    - `umount /mnt/data` (Safely removes it).
- **Tip:** Always `umount` before unplugging a drive to avoid data corruption.

53. How to safely shut down or reboot a Linux system

Answer: "Using the correct commands ensures that running processes are stopped gracefully and data is synced to the disk before power is cut.

- **`shutdown -h now`:** Immediately shuts down the system.
- **`reboot`** (or `shutdown -r now`): Restarts the system."
- **Why not just pull the plug?** Pulling the plug can corrupt the filesystem (journal errors) or lose data that was sitting in RAM cache but not yet written to disk.

54. How to use `watch` command

Answer: "`watch` runs a specified command repeatedly (default every 2 seconds) and displays the output in the terminal. It is perfect for real-time monitoring."

- **Real-World Example:**
    - You are copying a massive 50GB file. You want to see the file size grow in real-time.
    - **Command:** `watch du -sh target_file`

    - You want to see if a pod is coming up in Kubernetes: `watch kubectl get pods`.

9. Scheduling & Automation

55. Scheduling with cron

Answer: "`cron` is a time-based job scheduler in Unix-like systems. It allows users to schedule jobs (commands or scripts) to run automatically at specific times, dates, or intervals. The configuration file is called a `crontab`."

- **Simple Explanation:** It's like setting a recurring alarm clock for your server to do chores (like backups) while you sleep.
- **Syntax:** `* * * * * command_to_run`
    - The 5 stars represent: **Minute** (0-59), **Hour** (0-23), **Day of Month** (1-31), **Month** (1-12), **Day of Week** (0-6).
- **Real-World Example:**
    - "Run a database backup every day at 3:00 AM."
    - `0 3 * * * /home/user/scripts/backup_db.sh`
- **Command:** `crontab -e` (Edits the schedule file).

56. What are runlevels / targets in Linux ?

Answer: "Runlevels (in older SysVinit) or Targets (in modern Systemd) define the state in which a Linux system boots. They determine which services and resources are started."

- **Simple Explanation:** It works like "Safe Mode" vs. "Normal Mode" in Windows. It tells the OS whether to start just the text console or the full graphical interface.
- **Key Targets:**
    - **multi-user.target (Runlevel 3):** Text-only mode with networking. This is the standard for servers (saves resources).
    - **graphical.target (Runlevel 5):** Graphical User Interface (GUI) mode.
    - **poweroff.target (Runlevel 0):** System shutdown.
    - **reboot.target (Runlevel 6):** System reboot.
- **Command:** `systemctl get-default` (Shows the current target).

10. SSH, SCP & Remote Access

57. What is SSH ?

Answer: "SSH (Secure Shell) is a cryptographic network protocol used to securely access and manage network devices over an unsecured network. It encrypts the connection, preventing hackers from stealing passwords or data."

- **Simple Explanation:** It is a secure tunnel that lets you use a remote computer's terminal as if you were sitting right in front of it.
- **Real-World Example:** DevOps engineers use SSH daily to log into AWS EC2 instances (`ssh ec2-user@1.2.3.4`) to deploy code or fix issues.

58. Difference between SSH and Telnet

Answer: "The main difference is security. **SSH encrypts** all data (including login credentials), making it secure. **Telnet sends data in plain text**, meaning anyone on the network can intercept and read passwords."

- **Analogy:** Telnet is like shouting your password across a crowded room. SSH is whispering it in a secret code that only the listener understands.
- **Interview Tip:** Telnet is obsolete. If asked "When should you use Telnet?", the answer is usually "Never for remote management over the internet."

59. How to connect using SSH (`ssh user@host`)

Answer: "The standard command to connect to a remote machine is `ssh username@hostname_or_IP`. If the server uses a non-standard port (default is 22), specify it with `-p`."

- **Commands:**
  - Standard: `ssh root@192.168.1.10`
  - Custom Port: `ssh -p 2222 root@192.168.1.10`

60. What is SCP and how to use it ?

Answer: "SCP (Secure Copy Protocol) is a command-line tool used to securely transfer files between a local computer and a remote host (or between two remote hosts) using the SSH protocol."

- **Simple Explanation:** It's like the "Copy-Paste" feature, but over a network.
- **Commands:**
    - **Upload:** `scp localfile.txt user@remote_ip:/tmp` (Copies file *to* the server).
    - **Download:** `scp user@remote_ip:/var/log/syslog .` (Copies file *from* server to here).

61. What is SFTP vs SCP ?

Answer: "**SCP** is best for a quick, one-time file transfer (non-interactive). **SFTP** (Secure File Transfer Protocol) is an **interactive session** (like FTP) that allows you to browse directories, delete files, and resume interrupted transfers."

- **Real-World Example:** Use SCP to quickly push a config file. Use SFTP if you need to explore the remote folder structure to find what you want to download.

62. What is an SSH key pair (public/private key) ?

Answer: "An SSH key pair consists of two cryptographic keys used for authentication:

1. **Public Key:** Placed on the server (the lock). Anyone can see it.
2. **Private Key:** Kept secret on your computer (the physical key). Never share it."
- **Why use it?** It is more secure than passwords because passwords can be guessed (Brute Force), but 4096-bit keys are nearly impossible to crack.

63. How to generate SSH key (`ssh-keygen`)

Answer: "The `ssh-keygen` command creates a new public/private key pair on your local machine."

- **Command:** `ssh-keygen -t rsa -b 4096`
    - `-t rsa`: Type of encryption.
    - `-b 4096`: Bits (strength).
- **Tip:** Press Enter to accept the default location (`~/.ssh/id_rsa`).

64. Where are SSH keys stored (`~/.ssh/`) ?

Answer: "SSH keys are stored in the user's home directory inside a hidden folder named `.ssh`."

- **Key Files:**
    - `~/.ssh/id_rsa` (Private Key - **SECRET**).
    - `~/.ssh/id_rsa.pub` (Public Key - Safe to share).
    - `~/.ssh/authorized_keys` (File on the remote server where valid public keys are listed).

65. How to copy your key to remote server (`ssh-copy-id`)

Answer: "The `ssh-copy-id` command is a utility that automatically copies your **Public Key** to the remote server's `authorized_keys` file, enabling passwordless login."

- **Command:** `ssh-copy-id user@remote_ip`
- **Manual Method:** If `ssh-copy-id` isn't available, you can manually paste the content of `id_rsa.pub` into `~/.ssh/authorized_keys` on the server.

66. How to add SSH key to ssh-agent

Answer: "`ssh-agent` is a background program that handles passwords for SSH private keys. If your private key has a passphrase, adding it to the agent means you only have to type the passphrase once per session, not every time you connect."

- **Commands:**
    1. Start the agent: `eval $(ssh-agent -s)`
    2. Add your key: `ssh-add ~/.ssh/id_rsa`
- **Real-World Scenario:** Essential for CI/CD pipelines (like Jenkins) where automation needs to connect to servers without human intervention to type a password.

67. What are `known_hosts` and fingerprint warnings ?

Answer: "The `~/.ssh/known_hosts` file stores the 'fingerprint' (identity) of every server you have connected to. A **Fingerprint Warning** ('Remote host identification has changed!') appears if a server's key changes. This safeguards against Man-in-the-Middle attacks."

- **Simple Explanation:** Your computer remembers the server's face. If the server suddenly wears a mask (changes keys), your computer blocks the connection to keep you safe.

- **How to fix:** If you know the server was legitimately reinstalled, remove the old key: `ssh-keygen -R hostname`.

68. How to set up passwordless SSH login

Answer: "Passwordless login uses public key authentication instead of a password.
Steps:

1. Generate a key pair on your local machine: `ssh-keygen`.
2. Copy the public key to the remote server: `ssh-copy-id user@remote_ip`.
3. Now, `ssh user@remote_ip` will log you in instantly without prompting for a password."

- **Why do it?** It is mandatory for automation scripts (Ansible, Terraform) that cannot type passwords.

69. How to run a command remotely via SSH

Answer: "You can execute a command on a remote server without fully logging in by appending the command after the SSH address."

- **Command:** `ssh user@1.2.3.4 "ls -l /var/www"`
- **Real-World Example:**
    - "I need to restart the web server on 5 different machines."
    - You can write a script that runs `ssh user@server "sudo systemctl restart nginx"` for each one.

70. How to check SSH logs (`/var/log/auth.log`)

Answer: "SSH login attempts (successes and failures) are recorded in specific log files.

- **Ubuntu/Debian:** `/var/log/auth.log`
- **RHEL/CentOS:** `/var/log/secure`"
- **Command:** `tail -f /var/log/auth.log`
- **Real-World Scenario:** If you suspect hackers are trying to brute-force your server, check this log. You might see thousands of "Failed password" attempts.

71. What is `rsync` and how is it different from SCP ?

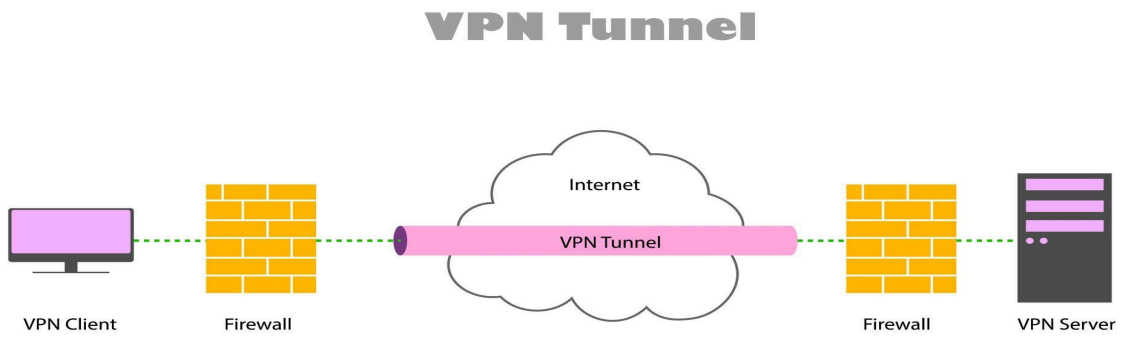
Answer: "**SCP** blindly copies files (overwriting everything). **`rsync`** (Remote Sync) is smarter: it checks differences and only transfers the parts of files that have changed. It also supports resuming interrupted transfers."

- **Simple Explanation:** SCP copies the whole book. Rsync only copies the page you edited.
- **Command:** `rsync -avz local_folder/ user@remote:/backup/`
    - `-a`: Archive (keep permissions).
    - `-v`: Verbose.
    - `-z`: Compress (faster).

72. How to use SSH tunneling or port forwarding

Answer: "SSH Tunneling allows you to forward a network port from a local machine to a remote machine (or vice versa) through an encrypted SSH connection. It is often used to access internal services (like a database) that are blocked by a firewall."

- **Command (Local Forwarding):** `ssh -L 8080:localhost:3306 user@remote_server`
    - This forwards your local port **8080** to the remote server's port **3306** (MySQL). Now you can connect to the remote DB using `localhost:8080`.



11. Volume

73. What is LVM (Logical Volume Manager) in Linux ?

Answer: "LVM (Logical Volume Manager) is a tool for logical volume management that provides a layer of abstraction between the physical storage devices (hard drives) and the file system. It allows you to create flexible storage volumes that can be resized (expanded or shrunk) dynamically without unmounting or restarting the system."

- **Simple Explanation:** Imagine you have two 5GB hard drives. Without LVM, you have two separate 5GB spaces. With LVM, you can glue them together to make one big 10GB space. If you run out of space later, you can buy a third drive and just "add" it to the pile to make it 15GB.
- **Real-World Example:**
    - **Scenario:** Your database server is running out of space on `/var/lib/mysql`.
    - **Without LVM:** You have to buy a bigger drive, turn off the server, copy all data to the new drive, and restart. (Downtime: Hours).
    - **With LVM:** You plug in a new drive, add it to the Volume Group, and extend the Logical Volume on the fly. (Downtime: Zero).
- **Key Concepts/Commands:**
    - **PV (Physical Volume):** The raw disk (`pvcreate /dev/sdb`).
    - **VG (Volume Group):** The pool of storage (`vgcreate my_pool /dev/sdb`).
    - **LV (Logical Volume):** The slice you actually use (`lvcreate -L 10G -n my_vol my_pool`).

12. Scenario-Based Questions

74. You SSH into a server, but a script fails with "Permission Denied."

Answer: "This usually means the script is not executable, or the user doesn't have the right read/execute permissions on the file or directory. **Troubleshooting Steps:**

1. Run `ls -l script.sh` to check permissions. Look for `x` (execute) flags.
2. If missing, run `chmod +x script.sh` to make it executable.
3. If permissions look fine but it still fails, check if the parent directory allows access, or if SELinux is blocking it."

- **Simple Explanation:** You are trying to open a door, but you don't have the key (permission) or the door is jammed (not executable).

75. A log file is growing too large and eating up disk space. What would you do ?

Answer: "I would first identify the large file using `du -sh` or `find`. To fix it immediately **without deleting the file** (which might crash the app writing to it), I would 'truncate' it: `> /var/log/largefile.log` (Empties the file while keeping it open). For a long-term fix, I would configure **logrotate** to automatically compress and archive old logs."

- **Real-World Mistake:** Never just `rm` a log file while the server is running. The process will keep writing to the "deleted" file reference, and the disk space won't be freed until you restart the service.

76. A background service is not running, but it should start automatically.

Answer: "I would check two things:

1. **Status:** `systemctl status service_name` to see why it stopped (look for error logs).
2. **Enablement:** `systemctl is-enabled service_name` to see if it's set to start at boot. If not, I would run `systemctl enable service_name`."

77. A file was accidentally deleted from `/etc`. Can you recover or find out who did it ?

Answer: "**To Recover:** If we have backups or use a configuration management tool (like Ansible/Git), I would restore it from there. If not, I might check `/lost+found` or try file recovery tools (rare in production). **To Find Who:** I would check the user's `.bash_history` or the system audit logs (`/var/log/audit/audit.log`) if `auditd` is enabled."

78. You notice 100% CPU usage on a Linux server. How do you investigate ?

Answer: "I would run `top` or `htop` and sort by CPU usage (P). I would identify the Process ID (PID) consuming the resources.

- If it's a valid process (like a database backup), I might let it finish or `renice` it.
- If it's a stuck or runaway process, I would kill it using `kill -15 PID`."

79. Your Bash script is failing silently. No error, no output.

Answer: "To debug a silent failure, I would run the script with the `-x` flag (`bash -x script.sh`). This enables **debug mode**, printing every command to the terminal before it executes, allowing me to see exactly where it stops or fails."

80. A VM is showing "No space left on device", but `df -h` says there is space.

Answer: "This is a classic 'Inode Exhaustion' issue. The disk has space (MB), but it has run out of index slots (Inodes) because of too many small files. **Fix:** Run `df -i` to check Inode usage. If it's 100%, I would search for directories with millions of tiny files (often session files or mail queues) and delete them."

81. You are trying to `scp` a file but it fails with "Connection Refused".

Answer: "Connection refused usually means the request reached the server, but no service was listening on that port. **Checks:**

1. Is the SSH service running on the remote host? (`systemctl status sshd`)
2. Am I using the correct port? (Default 22, but secure servers often change this).
3. Is a firewall (UFW/IPTables) blocking my IP?"

82. A user says they can't run a script due to "Permission Denied". You're root.

Answer: "Even if I am root, the **user** needs specific permissions.

1. Check file ownership: `ls -l script.sh`. Does the user own it?
2. Check group permissions: Is the user in the group that owns the file?
3. Check execute bit: Does the file have `chmod +x` set for that user/group?"

83. You deployed an app, but it fails because environment variables are missing.

Answer: "I would check how the variables are being loaded.

1. Run `printenv` or `env` in the running shell to see if they exist.
2. If it's a systemd service, the variables must be defined in the `.service` file (EnvironmentFile), not just in `.bashrc`, because services don't load user shell profiles."

84. How would you delete empty files from a directory ?

Answer: "I would use the `find` command with the `-empty` flag."

- **Command:** `find /path/to/dir -type f -empty -delete`
- **Why:** This is safer and faster than writing a loop to check file sizes manually.