Prompt

I’m sharing some random or rough technical notes below.  
 Your job is to **restructure and rewrite them** into a **complete, well-organized guide** that’s easy to revise and helps me *understand every concept deeply*.

**Requirements:**

1. Keep **all original information**, don’t skip any point.
2. Arrange content in **clear sequence** with proper **headings and subheadings**.
3. For each concept or command, explain:  
   * **What it is** (definition)
   * **Why it is used** (purpose)
   * **When or in what case it is used** (scenario)
   * **How it works / Step-by-step explanation**
4. Include all commands in code blocks and describe what each one does.
5. Use short paragraphs and bullet points for clarity.
6. Add comparison tables or summary boxes where helpful.
7. Add “**Summary/Revision Table**” at the end (Step | Command | Description).
8. Keep it **technically accurate and beginner-friendly** — assume I’m revising for interviews or exams.
9. If relevant, add practical examples or file path references (like /etc, /usr/local, etc.).
10. Avoid filler words — make it concise but detailed and logically connected.

**Tone and Style:**

* Professional but simple (like a Linux instructor explaining to a trainee).
* Focus on **understanding, not memorizing.**
* **Add a quick troubleshooting section for common errors.**

**Input Notes:**

Notes

Ec2 instances

first create

create new key pair and download

launch

get public ip it is live ip and machine reboot it changed

install mobaxterm

go to session sssh and put ip

advance ssh give private key

ec2 -user it create bydefault user it will use key to access that instance

sudo -i it make a root

1️⃣ Linux Fundamental

Commands

**User and Root**

# root sign

$ user sign

su root

su - sameer switch user

sudo -i from user to root move

sudo to execute command as superuser do

whoami show user status

**File & Directory Management**

mkdir make directory

cd .. go one level up in directory

cd go to folder

touch make file

pwd print working directory

ls list files and folder

ls /dev/sd\* filter search

ls -l long format (size, owner, permissions)

ls -lh human readable long format KB

ls -a shows hidden (dot) files

ls -ia shows inode numbers

ls -ld long listing of directory itself

mv mv <source> <destination>

cp cp <source> <destination>(new name) only copy

cp -r copy also directory

rm remove files

rm -rf remove forcefully

rmdir remove dir

man man <command> , q for quit

– help quick help

what is one line

~ home cd ~

touch name{1..10} mkdir filename{1..50} to make multiple files at once

rm -rf \* to remove all (Be careful)

> take output as input date > new (create new file and put output in new

>> same but append

sync ensures all changes that were in **RAM** are written to the **hard disk**.

df -Th to check which file system

wc -l shows word count

ls | wc -l join 2 command shows word count

less single line scrolling shows page fit result, ls /etc\* | less, q for quit

more page wise scrolling ls /etc\* | more

**File Viewing & Editing**

echo echo “hello world” > filename (to write in file)

> overwrite

>> Append (to write multiple lines)

cat to see content of file

head head -n 1 file name (show first 1 line of file)

tail tail -n 1 file name (show last 1 line of file)

> Send output to a file (overwrite) ls > file.txt

>> Append output to a file echo "Hello" >> file.txt

**|** Piping (|) connect the output of one command as the input to another.

cat file.txt | grep "error"

**> |** ls | grep ".txt" > txtfiles.txt

< Use a file as input sort < file.txt

ls \*.txt List All ending with .txt

ls \* List All

ls ?.txt matches exactly one character from file good for finding duplication

grep find words grep "error" logfile.txt

mv (rename) mv old newname also move mv file /home/

find find location iname “.txt” search from one by one dir

. in current dir .. in parent dir search

locate locate sameer search from all OS, location /var/mlocate

locate find new file once its updated, it update when reboot or cron run

**Archiving and Zipping**

gzip gzip filename compress file

gunzip gunzip filename Decompress

bzip2 compress

bunzip2 decompress

zip zip folder.zip \* it make copy of compressed file archive + compressed

zcat view compressed folder

zip name.zip /home/user \* (it will get all from user dir and current dir too

unzip unzip -d /locatin -d for to unzip in other location

unzip -l folder.zip to view

tar tar -cvf dir.tar \* just archive

tar -tvf dir.tar view

tar -xvf dir.tar extract

tar + zip tar-zcvf dir.tar.gz archive + compress

**Backup**

cp -r copy or backup, file and folder too

Rsync

**Inode**

ls -i Inode Number ls -i <file>

state state<file> show full inode detail

df -i shows inode usage of file system

find. -inum find . -inum 12345 (find by inode num)

find. -inum<n>delete find . -inum 12345 delete (find and delete by inode num)

**LInks**

ln ln file.txt hard.txt ( Hard link)

ln -s ln -s file.txt soft.txt (Soft Link)

**Ownership and Permission**

ls -n UID and GID (Root=0, user/group 1000+)

chmod chmod 755 filename

chown chown NEWOWNER file

chown NEWOWNER:NEWGROUP file

chown :NEWGROUP file # change only group

chgrp chgrp NEWGROUP file

chattr check attributes

chattr +i immutable even by root

chattr -i remove immutable permission

chattr +a append

chattr -a remove append

lsattr list flags

ps -u ps -u <user> show process running of user

userdel -r <user> delete user + home directory

chmod -Rv chmod -Rv<dir> change permission of directory + its content, v= description means what it done actually (R=recursive=Apply to all)

**Advance Permission**

setfacl -m u:ali:rwx<file> permission to special user

get facl <file> know what permission

setfacl -m m:rwx <file> mask - upper limit

**Power Off**

shutdown -h now halt + poweroff now.

shutdown -r now reboot now.

shutdown -c cancel scheduled shutdown.

shutdown -r +5 reboot after 5 minute

cpu info cat /proc/cpuinfo

memory info cat /proc/meminfo

lscpu show info of cpu

lsusb show info of usb

**Partition**

lsblk show HD info

lsblk -f more HD info

**Runlevels**

Important command to practice

find

ps

rsync

FSH-1

**Linux File System**

**If mounted on then partition if not then direc**

/ → Root (everything starts here)

/bin → Basic commands (ls, cp, mv…)

/usr → User programs (software, libraries)

/lib → Libraries for programs

/var → Variable data (logs, caches)

/etc → Configuration files

/home → User directories

/tmp → Temporary files

/dev → Device files (disks, USBs)

/proc → Kernel & process info

/boot → Bootloader & kernel

/mnt & /media → Mount points (USB, HDD)

/opt → Optional/3rd party software

/srv → Service data (web/ftp)

FSH-2

Got it ✅ — you want a **single, unified, professional-level “Linux Filesystem Master Guide”** that combines:  
 1️⃣ the **first-level directories** (like /etc, /usr, /var),  
 2️⃣ the **second-level subdirectories** (like /usr/bin, /etc/ssh),  
 3️⃣ the **third-level key files** (like /etc/fstab, /usr/bin/python3, /var/log/messages),  
 plus comparison tables, clear hierarchy diagrams, and plain-English explanations — all in one cohesive document.

# **🧭 Linux Filesystem Master Guide (Three-Level Deep)**

A complete reference for system administrators, DevOps engineers, and learners.

## **🌳 Overview**

The Linux filesystem follows the **Filesystem Hierarchy Standard (FHS)** — a structured tree starting from / (root).  
 Every file, configuration, and program branches out from this single root directory.

## **🗂️ Top-Level (First-Level) Directories**

| **Directory** | **Purpose** | **Key Type of Content** |
| --- | --- | --- |
| /bin | Essential user commands | Core tools for all users |
| /sbin | Essential admin binaries | System management tools |
| /usr | User programs and shared data | Applications, libraries, docs |
| /etc | System configuration | Text configs and scripts |
| /var | Variable / changing data | Logs, caches, databases |
| /lib, /lib64 | Shared libraries | Dynamic libs (.so) for binaries |
| /home | User directories | Personal files & configs |
| /root | Root user’s home | Superuser files |
| /boot | Bootloader files | Kernel, initrd, GRUB |
| /dev | Device files | Hardware representation |
| /proc | Virtual process info | Kernel and runtime data |
| /sys | Virtual hardware info | Device tree interface |
| /tmp | Temporary files | Cleared on reboot |
| /run | Volatile runtime data | PID, socket, and lock files |

## **⚙️ Second-Level: Important Subdirectories**

| **Parent** | **Subdirectory** | **Description** |
| --- | --- | --- |
| /usr | /usr/bin | Non-essential user commands (vim, git, python3) |
|  | /usr/sbin | Admin tools and daemons (sshd, httpd) |
|  | /usr/share | Architecture-independent data (docs, icons, locales) |
|  | /usr/lib, /usr/lib64 | Libraries for /usr programs |
| /etc | /etc/ssh/ | SSH client and server configs |
|  | /etc/systemd/system/ | Custom service units |
|  | /etc/yum.repos.d/ | Repository definitions |
|  | /etc/network/ | Network settings |
| /var | /var/log/ | Logs for system and services |
|  | /var/lib/ | Databases and app state |
|  | /var/spool/ | Print/mail queues, cron jobs |
|  | /var/cache/ | Cached data for apps |
| /boot | /boot/grub2/ | GRUB configuration |
| /lib | /lib/modules/ | Kernel modules |
| /proc | /proc/sys/ | Tunable kernel parameters |
| /sys | /sys/class/ | Devices by type (net, block) |
| /home | /home/<user>/ | User directories (.bashrc, .ssh) |

## **🧩 Third-Level: Key Configuration & System Files**

### **📁 /etc — *Configuration Nucleus***

| **File** | **Function** |
| --- | --- |
| /etc/passwd | User account info |
| /etc/shadow | Encrypted passwords |
| /etc/group | Group definitions |
| /etc/fstab | Filesystem mount points |
| /etc/hostname | System hostname |
| /etc/hosts | Local host–IP mapping |
| /etc/resolv.conf | DNS configuration |
| /etc/crontab | Scheduled system tasks |
| /etc/sudoers | sudo privilege rules |
| /etc/profile | Login environment variables |
| /etc/bashrc | Global bash configuration |
| /etc/sysctl.conf | Kernel parameter tuning |
| /etc/ssh/sshd\_config | SSH daemon configuration |
| /etc/httpd/conf/httpd.conf | Apache configuration |
| /etc/php.ini | PHP settings |
| /etc/mysql/my.cnf | MySQL configuration |
| /etc/selinux/config | SELinux mode |
| /etc/default/grub | GRUB defaults |
| /etc/security/limits.conf | User limits |
| /etc/pam.d/system-auth | PAM authentication |
| /etc/login.defs | User creation defaults |

### **⚙️ /usr/bin — *User Executables***

| **File** | **Purpose** |
| --- | --- |
| /usr/bin/vim | Text editor |
| /usr/bin/python3 | Python interpreter |
| /usr/bin/git | Version control |
| /usr/bin/curl | HTTP requests |
| /usr/bin/wget | File download |
| /usr/bin/grep | Text search |
| /usr/bin/top | Process monitor |
| /usr/bin/systemctl | Manage systemd |
| /usr/bin/journalctl | View logs |
| /usr/bin/find | File search |

### **🔐 /usr/sbin — *Administrative Daemons***

| **File** | **Function** |
| --- | --- |
| /usr/sbin/sshd | SSH daemon |
| /usr/sbin/httpd | Apache web server |
| /usr/sbin/nginx | NGINX daemon |
| /usr/sbin/useradd | Add new user |
| /usr/sbin/ufw | Firewall management |
| /usr/sbin/firewalld | Dynamic firewall daemon |
| /usr/sbin/postfix | Mail server |
| /usr/sbin/cron | Scheduler daemon |

### **📚 /usr/share — *Shared Read-Only Data***

| **File / Path** | **Purpose** |
| --- | --- |
| /usr/share/man/man1/ls.1.gz | Manual page for ls |
| /usr/share/doc/httpd/README | Package documentation |
| /usr/share/fonts/ | System fonts |
| /usr/share/icons/ | Icon themes |
| /usr/share/locale/en\_US/ | Translations |
| /usr/share/applications/firefox.desktop | Desktop launcher |

### **📊 /var — *Dynamic Runtime Data***

| **File** | **Purpose** |
| --- | --- |
| /var/log/messages | System log |
| /var/log/secure | Security/authentication logs |
| /var/log/dmesg | Kernel boot messages |
| /var/log/httpd/access\_log | Apache access logs |
| /var/log/httpd/error\_log | Apache error logs |
| /var/lib/mysql/ibdata1 | MySQL database data |
| /var/spool/cron/root | Root’s cron jobs |
| /var/cache/dnf/ | Package cache |
| /var/tmp/ | Temp files persistent across reboot |

### **🧱 /lib, /usr/lib, /lib64 — *Shared Libraries***

| **Path** | **Description** |
| --- | --- |
| /lib/libc.so.6 | Standard C library |
| /lib/libpthread.so.0 | Threading library |
| /lib/modules/<kernel>/ | Kernel modules |
| /usr/lib/systemd/system/sshd.service | SSH systemd service unit |
| /usr/lib/python3.\*/ | Python standard libraries |

### **🖥️ /boot — *Boot Essentials***

| **File** | **Purpose** |
| --- | --- |
| /boot/vmlinuz-\* | Linux kernel image |
| /boot/initramfs-\* | Initial RAM disk |
| /boot/grub2/grub.cfg | GRUB configuration |
| /boot/System.map-\* | Kernel symbol table |

### **💽 /dev — *Device Interface***

| **File** | **Description** |
| --- | --- |
| /dev/sda | First disk |
| /dev/sda1 | First partition |
| /dev/null | Discards data |
| /dev/zero | Produces null bytes |
| /dev/random | Random data source |
| /dev/tty | Terminal device |

### **🧮 /proc — *Kernel and Process Info (Virtual)***

| **File** | **Description** |
| --- | --- |
| /proc/cpuinfo | CPU details |
| /proc/meminfo | Memory details |
| /proc/uptime | Uptime info |
| /proc/mounts | Mounted filesystems |
| /proc/sys/net/ipv4/ip\_forward | IP forwarding status |
| /proc/[PID]/cmdline | Process command line |
| /proc/[PID]/status | Process metadata |

### **⚙️ /sys — *Device Tree (Virtual)***

| **Path** | **Description** |
| --- | --- |
| /sys/class/net/eth0/ | Network interface info |
| /sys/block/sda/size | Disk size |
| /sys/devices/system/cpu/ | CPU topology |
| /sys/power/state | Power control |

### **🏠 /home and /root**

| **File** | **Purpose** |
| --- | --- |
| /home/user/.bashrc | User shell settings |
| /home/user/.ssh/authorized\_keys | SSH key access |
| /home/user/.vimrc | Vim editor config |
| /root/.bashrc | Root shell config |
| /root/.ssh/authorized\_keys | Root key access |

### **🔁 /tmp and /run**

| **File** | **Description** |
| --- | --- |
| /tmp/\*.tmp | Temporary files |
| /run/systemd/ | Systemd runtime data |
| /run/lock/ | Lock files |
| /run/user/1000/ | User session data |

## **⚖️ Category Comparison: Quick Reference**

| **Category** | **Examples** | **Editable** | **Volatile** | **Purpose** |
| --- | --- | --- | --- | --- |
| Configuration | /etc/ | ✅ | ❌ | System settings |
| Commands | /bin, /usr/bin | ❌ | ❌ | Executable binaries |
| Admin Tools | /sbin, /usr/sbin | ❌ | ❌ | System maintenance |
| Libraries | /lib, /usr/lib | ❌ | ❌ | Shared code for apps |
| Variable Data | /var/ | ✅ | ✅ | Logs, caches, DBs |
| Temporary | /tmp, /run | ✅ | ✅ | Short-term files |
| Virtual | /proc, /sys | ❌ | ⚡ | Kernel/runtime info |

## **🧠 Three-Level Visual Summary**

/

├── bin/ → core user commands

│ ├── ls, cp, mv

├── sbin/ → core admin commands

│ ├── fsck, reboot

├── etc/ → configs

│ ├── fstab

│ ├── ssh/sshd\_config

│ ├── yum.repos.d/\*.repo

├── usr/

│ ├── bin/python3, vim

│ ├── sbin/httpd, useradd

│ ├── share/

│ │ ├── man/, doc/, fonts/

│ └── lib/systemd/system/\*.service

├── var/

│ ├── log/messages, secure

│ ├── lib/mysql/

│ ├── spool/cron/

│ └── cache/yum/

├── lib/, lib64/ → shared libs (libc.so.6)

├── boot/

│ ├── vmlinuz-\*, grub2/grub.cfg

├── dev/

│ ├── sda, null, zero

├── proc/

│ ├── cpuinfo, meminfo

├── sys/

│ ├── class/net/

├── home/user/

│ ├── .bashrc, .ssh/

├── root/

│ ├── .bashrc, .ssh/

├── tmp/

├── run/

└── ...

## **🧩 Memory Hooks**

* **/etc → “Edit To Configure”**
* **/var → “Variable data (logs, DBs)”**
* **/usr → “User programs and shared resources”**
* **/lib → “Libraries used by commands”**
* **/proc → “Processes”**
* **/sys → “System hardware tree”**
* **/boot → “Boot files”**
* **/home → “Personal data”**

Would you like me to now generate this as a **visually formatted PDF** (with tables, color sections, and icons for each directory) — ready for printing or offline study?

## **/bin vs /usr/bin vs /sbin vs /usr/sbin**

| **Directory** | **Type of Commands** | **Accessible By** | **Purpose** |
| --- | --- | --- | --- |
| /bin | Essential user commands for system boot and repair | All users | Basic commands like ls, cp, mv, cat, bash |
| /usr/bin | Non-essential user commands (main user tools) | All users | Editors, compilers, utilities like vim, python3, gcc |
| /sbin | Essential system binaries | Root | Low-level admin commands like fsck, init, ip, mount |
| /usr/sbin | Non-essential system binaries | Root | Server and admin tools like sshd, apachectl, useradd |

## **Summary / Revision Table**

| **Directory** | **Purpose** | **Example Contents** | **Access** |
| --- | --- | --- | --- |
| /bin | Essential user commands | ls, cp, mv, bash | All users |
| /sbin | Essential system commands | fsck, mount, reboot | Root |
| /usr/bin | User applications | vim, python3, git | All users |
| /usr/sbin | Admin/daemon commands | sshd, httpd, useradd | Root |
| /usr/share | Read-only shared data | Docs, icons, fonts | Root |
| /etc | Configuration files | fstab, sshd\_config | Root |
| /var | Variable data (logs, DBs) | /var/log/, /var/lib/ | System |
| /lib | Essential libraries | /lib/libc.so.6 | System |
| /usr/lib | Libraries for user programs | /usr/lib/python3/ | System |
| /lib64 | 64-bit libraries | /lib64/libpthread.so.0 | System |

Find command

# **Linux Command: find — File & Directory Search Tool**

## **1️⃣ Purpose**

The find command is used to **search for files and directories** in a directory hierarchy based on **conditions** like:

* Name
* Type
* Owner
* Size
* Modification date
* Permissions

…and then perform actions (like ls, rm, chmod, chown, etc.) on them.

## **2️⃣ General Syntax (Synopsis)**

find [path] [options/conditions] [actions]

| **Part** | **Meaning** |
| --- | --- |
| **[path]** | Starting directory (e.g., /opt, /home, /pakistan) |
| **[options/conditions]** | Criteria to match (e.g., -name, -user, -type, -perm) |
| **[actions]** | What to do with matching files (e.g., -print, -exec, -delete) |

## **3️⃣ Commonly Used Options**

| **Option** | **Description** | **Example** |
| --- | --- | --- |
| -name | Find file by name (case-sensitive) | find /opt -name test.txt |
| -iname | Find file by name (case-insensitive) | find /opt -iname Test.txt |
| -type | Find by type: f=file, d=directory | find /opt -type d |
| -user | Find by owner | find /opt -user sameer |
| -group | Find by group | find /opt -group admin |
| -size | Find by file size | find /opt -size +10M |
| -perm | Find by permission | find /opt -perm 777 |
| -mtime | Find by modified date | find /opt -mtime -7 (within 7 days) |
| -empty | Find empty files or dirs | find /opt -empty |
| -exec | Run another command on the result | *(see below)* |

## **4️⃣ -exec Option — (Execute a Command on Each Match)**

### **Syntax:**

find [path] [conditions] -exec [command] {} \;

| **Symbol** | **Meaning** |
| --- | --- |
| **-exec** | Start of a secondary command (to execute on each found file) |
| **{}** | Placeholder → replaced by the name/path of each found file |
| **\** | Escape character → prevents the shell from interpreting ; |
| **;** | End of -exec command |

🧠 **In simple words:** Everything between -exec and \; is the command to run **for each file** found.

### **Example 1:**

find /opt -user sameer -exec rm -i {} \;

**Explanation:**

| **Part** | **Description** |
| --- | --- |
| /opt | Start searching from /opt |
| -user sameer | Find all files owned by user sameer |
| -exec rm -i {} \; | For each found file, run rm -i <filename> → asks confirmation before deleting |

So it **finds all files owned by sameer** inside /opt and **deletes them interactively**.

### **Example 2:**

find /opt -perm 777

**Meaning:** List all files and directories inside /opt that have **permission 777** (i.e., full read/write/execute access for everyone).

If you want to change them:

find /opt -perm 777 -exec chmod 755 {} \;

→ Changes each 777 file to 755.

## **5️⃣ Difference Between \; and +**

| **Terminator** | **Meaning** | **Behavior** |
| --- | --- | --- |
| \; | End of -exec command (one by one execution) | Runs the command separately for **each** file |
| + | Batch mode | Runs the command **once** for multiple files at a time (faster) |

**Example:**

find /opt -type f -exec chmod 644 {} +

→ Changes permissions for all files in fewer batches (faster).

## **6️⃣ Special Symbols Recap**

| **Symbol** | **Meaning** |
| --- | --- |
| **{}** | Represents each matched file or directory path |
| **\** | Escape symbol used to protect the following character (like ;) |
| **;** | Marks the end of the -exec command |
| **/** | Directory separator (path) |

## **7️⃣ Additional Practical Examples**

| **Command** | **Explanation** |
| --- | --- |
| find /var -type f -name "\*.log" | Find all .log files under /var |
| find /home -size +500M | Find files larger than 500 MB |
| find /etc -mtime -1 | Find files modified in the last 24 hours |
| find /tmp -empty -delete | Delete all empty files & folders in /tmp |
| find / -perm /222 | Find all files that are writable by anyone |

find /var/log/http -mtime -10 -exec rm -rf {} \;

Path System

**Absolute vs Relative Path in Linux**

**Absolute Path**

Starts from the **root /** directory.

Always gives the **full location** of a file or folder.

Works from **anywhere** in the system.

**Relative Path**

Starts from your current directory (.).

Does not start with /.

Depends on where you are.

**Example**

./file.txt (file in current folder)

../file.txt (go up one directory, then to file)

../../docs/file.txt (go up two levels, then into file)

Inode

**Inode**

Data structure in Linux that stores metadata about a file, but not the file name.

Every file has a unique **inode number** in its filesystem

## **What Inode Stores (Metadata)**

Memory hook: **POT-TIP**

* **P**ermissions (rwx)
* **O**wner (UID, GID)
* **T**imestamps (atime, mtime, ctime)
* **T**ype of file (regular, directory, symlink, etc.)
* **I**node number + link count
* **P**ointers → actual data blocks on disk

**Does NOT store**: File name or file data.

* **Inode** = All file info + Block pointer
* **Data Blocks** = Actual content of the file.
* **file nam**e= maps file name → inode number

**Links**

**Hard Link** both share same inode

**Symbolic link** new inode to new file

## **Inode Timestamps**

* **atime** = last **Access** Updates when file is **read**.
* **mtime** = last **Modify** Updates when the **file content** changes.
* **ctime** = last **Change** Updates when the **inode metadata** changes.
* **btime** = Birth (creation, only on some filesystems).

## **Inode & Disk Usage**

* Size = actual file content length.
* Blocks = space allocated

LInks

**Links**

A link is simply a reference (name) to a file.

Since files are really just inodes + data blocks, links decide how we access them.

### **Hard Link**

* Another name pointing to the same inode.
* File content is the same, no duplication.
* Both names are equal → deleting one does not delete the data (until last link is gone).
* No (across filesystems/partitions)

### **Soft Link (Symbolic Link)**

* A **separate file** with its **own inode**.
* Stores the **path** to another file (like a shortcut in Windows, no content).
* If the target is deleted → symlink becomes **broken**.
* Yes (across filesystems/partitions)

Ownership and permission

**Ownership and Permission**

-rw-r--r--. 1 owner group 1024 Sep 22 14:20 file.txt

**1st char** → file type

* - = regular file
* d = directory
* l = symlink
* c = character device
* b = block device, etc

**Next 9 chars** → permissions for user, group, others (rwx).

**Final symbol**:

* . = file has **extended attributes (xattr)**
* + = file has **ACLs (Access Control Lists)**
* (blank) = no extras

**Link count**

Number of hard links

**File link count** = number of hard links.

**Directory link count** = 2 + number\_of\_subdirectories.

why:

. = **this directory itself** (folderA)

.. = **the parent** (in this case /home/)

**Owner (User)**

Username (UID) of file’s owner.

**Group**

Group (GID) that owns the file.

**File size**

Size in bytes

**Modification time (mtime)**

Last content modification time.

Shows month day time (or year if old file).

**File name**

**Permissions**

Permissions decide who can do what with a file or directory.  
They exist for three roles (UGO):

* **User (u) → the owner of the file**
* **Group (g) → the file’s group**
* **Others (o) → everyone else**

## **Types of Permissions**

Each role (u, g, o) can have:

* **r (read = 4)**
  + File → view contents
  + Directory → list files inside(ls)
* **w (write = 2)**
  + File → edit, append, delete contents
  + Directory → add/remove/rename files
* **x (execute = 1)**
  + File → run as a program/script
  + Directory → enter the directory (cd)

**File permission**= Protect file content

**Directory permission**= Protect file names, deletion and renaming (Directory level)

**(if file permission is 000, and dir permission is 777)**

* Root: Can do rwx on file + delete rename create(ignore 000 of file)
* Owner: Can not rwx on file and yes dir level Delete, create and rename. but chmod for file permission (only owner and root can do it)
* Others: no rwx, but dir level

**Numeric values (r=4, w=2, x=1) → add them up for shorthand:**

### **Symbolic Method**

**chmod u+x file.txt # add execute for user**

**chmod g-w file.txt # remove write for group**

**chmod o=r file.txt # others can only read**

**chmod u=rwx,g=rx,o= # set all at once**

### **Numeric (Octal) Method**

**chmod 755 filename**

* **User = 7 (rwx)**
* **Group = 5 (r-x)**
* **Others = 5 (r-x)**

**Change Attribute**

special attribute on file directories at the file system level

These attributes give extra protection or behavior beyond normal permissions (chmod).

## **Why Use chattr?**

* Protect important system files so they are **not deleted or edited** (even by root).
* Make log files **append-only**, so entries can only be added, not removed.
* Prevent accidental changes.

## **chattr +i**

* chattr = “change attribute.”
* +i = adds the immutable flag to a file.
* **What it means:**If a file is immutable (+i set), then:
* You cannot modify it.
* You cannot delete it.
* You cannot rename it.
* Even root cannot overwrite it (without removing the flag)

## **Common Attributes**

* +i → Immutable (cannot be changed, deleted, renamed, or linked).
* -i → Remove immutable flag.
* +a → Append-only (you can only add to the file, not delete/overwrite).
* -a → Remove append-only flag.

## **Important Notes**

* chattr only works on ext2/ext3/ext4 filesystems (the most common in Linux).
* You need root (sudo) privileges for most attributes.
* It is stronger than normal chmod permissions.

**Advanced Permission**

**Access Control List (ACL)**

Special/custom permission to a specific user who is not the owner or part of the group.

**Modify**

setfacl -m u:ali:rw file.txt

### **Recursive ACL (directory + contents)**

setfacl -R -m u:ali:rw myfolder/

**Mask**

setfacl -m m:rw file.txt

# **setfacl Flags (with meaning)**

* **-m** → *Modify* or add an ACL entry
* **-x** → Remove a specific ACL entry
* **-b** → Remove all ACL entries (reset file to normal permissions)
* **-k** → Remove all default ACLs (from a directory)
* **-d** → Apply a **default ACL** (for new files in a directory)
* **-R** → Apply ACLs **recursively** (to directory and its contents)

Vim editor

**VIM Editor**

### **Moving in and out of modes**

* **i → insert mode (start typing)**
* **Esc → back to normal mode**

### **Saving & quitting**

* **:w → save file**
* **:q → quit**
* **:wq → save and quit**
* **:q! → quit without saving**

### **Editing**

* **x → delete a single character**
* **dd → delete a whole line**
* **u → undo**
* **Ctrl-r → redo**
* **yy → copy line (yank)**
* **p → paste below**

### **Search & Replace**

* **/word → search forward for “word”**
* **n → next match, N → previous match**

## **:%s/old/new/g → Search and Replace**

* **: → enter command mode**
* **% → apply to the whole file**
* **s → substitute (replace)**
* **/old/ → the text you want to replace**
* **/new/ → the new text you want instead**
* **g → global, meaning replace *all occurrences in each line***

***123G → Jump to line 123***

Runlevels

## **Runlevels**

the mode in which your Linux system is running.

**Commands**

runlevel to check runlevel

init init<No> to move from one to other

## **The Common Runlevels**

Traditionally in Linux (SysVinit systems):

* **0 → Shutdown** (system is off)
* **1 → Single user mode** (only one user, used for repair)
* **2 → Multi-user (without network in some distros)**
* **3 → Multi-user with networking (no graphics)**
* **4 → Not used (free for custom use)**
* **5 → Graphical mode (desktop login)**
* **6 → Reboot**

## **Example in Real Life**

* When you **shut down** your PC → Runlevel 0
* When you **restart** → Runlevel 6
* When you see only **black screen with commands** → Runlevel 3
* When you see your **desktop with icons and mouse** → Runlevel 5
* Runlevel shows previous and current.
* “N” means no previous run level.

**(*New RHEL 7 and 8 new method target***

**Virtual Console (In Runlevel 3)**

A **virtual console** is like a separate text-based login screen inside Linux.

* You can switch between them with shortcuts like **Ctrl + Alt + F1 … F6**.
* Each virtual console gives you an independent shell session (you can log in as different users on different consoles).

**Runlevel = system state/mode** that controls what services and environment are available.A **runlevel** defines the **mode of operation** of the Linux system(0 to 6).  
Runlevels decide **what services/processes** the system starts, not where you log in

**gnome Terminal (In Runlevel 5)**

* Around 400 Terminal
* Terminal name is pts (pseudo terminal)
* ps to check
* /dev/pts

### **Remote Access with PuTTY (pts)**

* **When you connect to a Linux system using PuTTY (SSH), the session uses a pseudo-terminal (pts) file under /dev/pts/#.**
* **Even if the Linux system is running in runlevel 5 (graphical mode), PuTTY itself can only show text/CLI output because it does not support graphics.**
* **To run GUI applications remotely, you would need:**
  + **X11 forwarding with an X server installed on your client machine (e.g., Xming, VcXsrv on Windows).**
  + **Or use remote desktop tools (like VNC, RDP, or X2Go).**

**Note:**

* In **runlevel 3**, You get **virtual consoles (VSCs)** for text-based login (tty1–tty6). /dev/tty is driver (To check, ps (process status))
* By default you interact with the system through virtual consoles
* In **runlevel 5** default is **GNOME, KDE, etc.**).
* if you are running programs in a console and you switch runlevels, those processes might get **terminated** because services are stopped/started fresh.

### **Graphics Without Switching Runlevel**

* Even in runlevel 3, you can start GUI temporarily:
  + Load GUI: startx
  + Unload GUI: pkill X

In minimal Linux installations, startx may not work because GUI packages are not installed.

### **Process of Boot**

1. **Power ON**
2. **Splash screen**→ if interrupted Init5 (move to level 5)
3. **/etc/inittab** → system chooses default runlevel (3 or 5 usually).

# **In RHel 7 onward**

(as in rhel 6, /etc/inittab, change default runlevel but )

**To check default runlevel**

systemctl get-default

**To change**

systemctl set-graphical.target

**Move to Multi-User (Runlevel 3, text mode)**

sudo systemctl isolate multi-user.target

**Move to Graphical (Runlevel 5, GUI mode**

sudo systemctl isolate graphical.target

Root Passwd

# **Securing Single-User Mode in RHEL 6**

### **1. Default Behavior**

* In **RHEL 6**, when rebooting and interrupting the boot process:  
  + Press a for append
  + If you choose **single-user mode** (runlevel 1),
  + The system **drops directly into a root shell** **without asking for password**.
  + This is a security risk.

### **2. Fix / Secure It**

Open the **init configuration file**:  
 vi /etc/sysconfig/init

Go to the **end of the file** quickly with:  
 Shift + G

Look for the variable:  
 SINGLE=/sbin/sushell

Replace it with:  
 SINGLE=/sbin/sulogin

### **3. Effect of Change**

* **sushell** = automatically spawns a root shell without authentication.
* **sulogin** = prompts for the **root password** before giving shell access in single-user mode.

**Init 1 (Single user, rescue mode)**  
older version dont need root password   
Newer version changed: reset via GRUB method.

# **Resetting Root Password in RHEL 5/6**

1. **Reboot** → interrupt GRUB splash.
2. **Select kernel** → press e.
3. Highlight **kernel line** → press e.
4. Append single or 1 at end.
5. **Boot** → press b.
6. At root shell → passwd root (set new password).
7. **Reboot** → reboot.

# **Resetting Root Password in RHEL 7**

### **1. Reboot and Interrupt GRUB**

* Reboot the system.
* At the **GRUB menu**, highlight the kernel line.
* Press **e** to edit it.

### **2. Edit Kernel Line**

* Find the line starting with linux16.
* At the end of that line, add:  
   rd.break
* Press **Ctrl + X** (or F10) to boot with this option.

### **3. System Boots into Emergency Shell**

* You will land in a shell prompt with the root filesystem mounted **read-only** at /sysroot.

### 

### 

### **4. Remount Root Filesystem**

mount -o remount,rw /sysroot

### **5. Switch into Real Root**

chroot /sysroot

### **6. Reset the Password**

passwd root

Enter the new root password twice.

### **7. Relabel SELinux (important in RHEL 7+)**

touch /.autorelabel

* Ensures SELinux contexts are updated at next boot, or you may not be able to log in.

### **8. Exit and Reboot**

exit

exit

reboot

Partitions

**Hard Disk**

**Track** = the circle (like a ring on the platter).

**Sector** = each slice of that circle (8 slices if you divide into 8).

**Sector** is the **smallest physical storage unit** on the disk (usually 512 bytes or 4 KB).

**Block** is the **smallest unit used by the operating system (file system)**.

* A block is made up of one or more **sectors**.
* Example: If block size = 4 KB and sector = 512 B → 1 block = 8 sectors.

**Bit = 0,1**

**1 bytes = 8 bit**

**1KB = 1000Bytes**

### **The Letters (a, b, c, …)**

* The letters show the **order in which the system detected the devices**:  
  + sda → the first detected disk
  + sdb → the second detected disk
  + sdc → the third detected disk

### **Partitions on Each Disk**

* Partitions are indicated with numbers:  
  + sda1 → first partition of the first disk
  + sda2 → second partition of the first disk
  + sdb1 → first partition of the second disk

The **driver** knows how to talk to the HDD/SSD.

The kernel then creates /dev/sda to give users and applications a way to access it.

# **The big picture**

* **Disk** = the whole storage (like a campus).
* **Partition** = a slice of that disk (like separate buildings).
* **Partition scheme** (MBR/GPT) = the **map** that tells where each slice starts/ends.

1. **MBR** = old partition scheme, max 2 TB disk & 4 partitions, works with BIOS.
2. **GPT** = modern scheme, huge disks & 128+ partitions, works with UEFI.

* **File system** (ext4, xfs, ntfs) = how files are organized inside a slice (like shelves & rules).
* **Mount point** = the folder where you “enter” that slice (a door with a name, e.g. /data).
* /, /etc, /var are not themselves file systems → they’re **folders (mount points)**.
* They live **inside a file system** (like ext4 or xfs) on a partition.

# **What ext4 *does inside a partition***

When you **format** a partition with ext4 (mkfs.ext4 /dev/sda1), Linux writes special structures inside that partition:

* **Superblock** 🏷️ → like the table of contents. Tells where everything else is.
* **Inodes** 📇 → small cards that store info about each file (owner, permissions, where data lives).
* **Data blocks** 📦 → the actual pieces of your files.
* **Directories** 📂 → lists that link filenames to their inode cards.
* **Journal** 📝 → a log that helps recover if power fails (so files don’t get corrupted).

👉 Together, these let Linux say:  
 “Okay, /home/ali/text.txt = inode #25 → points to blocks #101, 102, 103 → these blocks contain the text data.”

## **What is LVM?**

**LVM = Logical Volume Manager** It’s a **layer between physical disk partitions and the file system**.

So instead of being stuck with fixed partitions, LVM lets you:

* Resize volumes more easily (grow/shrink).

### **Basic Disk Partition Structure**

* A **hard disk** can have **a maximum of 4 primary partitions**.
* Out of these 4, **one** can be converted into an **extended partition**.

### **Key Points**

* You can create **only 4 primary partitions** in MBR.
* Instead of using all 4 as primary, you can make **3 primary + 1 extended**.
* Inside that **1 extended partition**, you can create **many logical partitions** (commonly up to 63).
* **Extended partition** is **not** a primary partition itself, but it **replaces one primary slot**.
* The first **four numbers (1–4)** are reserved for **primary partitions** (including the extended one).
* Logical partitions use a **separate numbering range (5 and above)** to keep them distinct.
* when you say **“9 drives but 8 partitions”**, it means:
* You have 9 **physical disks** (/dev/sda to /dev/sdi for example),
* But one of them (or one of its partitions) is **extended**, and **doesn’t hold data itself**.

## **1. Two Types of Partition Tables**

### **A. MBR (or MSDOS)**

* **Full form:** Master Boot Record
* **Old standard**, used for decades.
* **Limitations:**
  + Supports **only 4 primary partitions**.
  + To go beyond 4, one must be **converted to an extended partition** (which holds logical partitions).
  + Supports disks **up to 2 TB** only.
  + Stores partition info in the **first 512 bytes** of the disk.
* **Used in:** older BIOS-based systems.

### **B. GPT (GUID Partition Table)**

* **Modern standard** used with **UEFI** systems.
* Overcomes MBR limitations.
* **Features:**
  + Supports **up to 128 partitions** by default (no need for extended/logical).
  + Supports disks **larger than 2 TB**.
  + Stores **backup partition table** at the end of the disk for recovery.
  + Each partition has a **unique GUID (Global Unique ID)**.

## **🧰 2. Check Partition Table Type**

To check what partition table a disk uses:

sudo parted -l

or for a specific disk:

sudo parted /dev/sdb print

or (older tools)

sudo fdisk -l /dev/sdb

FDisk - partition

## **What is fdisk?**

* fdisk = **Fixed Disk Editor**.
* It’s a **command-line tool in Linux** used to create, delete, and manage **disk partitions**.
* Works mainly with **MBR** (older) partition scheme, but modern versions also handle GPT.

In short: fdisk is a **partition manager** for your disks.

### **Partition Table & Kernel Update**

* When using **fdisk**, pressing **w** writes (updates) the **partition table** on the hard disk.
* After updating the disk, the **kernel Table** must also be informed.  
  + In **RHEL 6**: use partx -a
  + In **RHEL 7**: use partprobe -x
* These commands update the **kernel’s in-memory partition table** (in **RAM**) so the OS recognizes new partitions.

### **💾 Partitions Overview**

* Example: 3 partitions → sdb1, sdb2, sdb3  
   → Each partition has its **own filesystem, 3 Filesystem**.
* Every hard disk contains **one partition table** only.
* 4 drivers

### **📂 Common Filesystem Types**

| **Filesystem** | **Features** |
| --- | --- |
| **ext2** | No journaling, older, simple |
| **ext3** | ext2 + journaling (faster recovery) |
| **ext4** | Larger file support, faster performance, delayed allocation |
| **xfs / btrfs** | Advanced, used in modern RHEL/CentOS versions |

**Journaling** is a feature in modern filesystems (like **ext3**, **ext4**, **xfs**) that keeps a **log (journal)** of changes before they are actually written to the main filesystem.

It helps the system **recover quickly** after a crash or power failure without checking the entire disk.

### **🧠 Important Linux Boot referencing Files**

| **File** | **Purpose** |
| --- | --- |
| **/etc/inittab** | Contains **runlevel information** (used in SysV init systems) |
| **/etc/fstab** | Contains **mounting information** for filesystems (File System Table). It is one of the **9 critical files** in Linux. |

### **📘 /etc/fstab – Fields**

Defines how partitions and filesystems mount automatically at boot.

| **Field** | **Description** |
| --- | --- |
| **Device/Driver** | e.g., /dev/sda1 or UUID=abcd-1234 |
| **Mount Point** | Directory where it’s mounted, e.g., /home, /data |
| **Type of Filesystem** | ext4, xfs, vfat, etc. |
| **Permission / Options** | defaults, ro, rw, etc. |
| **Fsdump** | Backup flag (0 or 1) |
| **Fsck On/Off Sequence** | Filesystem check order (0, 1, 2) |

### **🔑 UUID (Universally Unique Identifier)**

* A **128-bit unique number** assigned to each partition or device.
* Ensures reliable identification even if device names change (e.g., /dev/sda1 → /dev/sdb1).

View using:  
  
 sudo blkid

* Example:  
   UUID=3A1C-8F2A

# **Steps to Create a New Partition**

1️⃣ **List disks**

fdisk -l

2️⃣ **Open the disk in fdisk**

fdisk /dev/sdb

3️⃣ **Create new partition**

* Press **n** → new partition
* Choose **p** (primary) or **e** (extended)
* Enter partition number, start, and end sectors

4️⃣ **Write changes to disk**

* Press **w** → write and save partition table

5️⃣ **Update kernel partition table**

* **RHEL 6:** partx -a /dev/sdb
* **RHEL 7+:** partprobe -x /dev/sdb

6️⃣ **Create filesystem**

mkfs.ext4 /dev/sdb1

7️⃣ **Create mount point**

mkdir /mnt/data

8️⃣ **Mount partition**

mount /dev/sdb1 /mnt/data

9️⃣ **Add to /etc/fstab** (for permanent mount)

**Detects new disks without rebooting**

echo "- - -" > /sys/class/scsi\_host/hostX/scan

### **How a hard disk is structured**

A hard disk is divided into:  
 **Cylinders → Tracks → Sectors**

* **Cylinder** = one track on *each* platter surface (same radius). 1 cylinder = 8MB
* **Track** = circular path on a single platter surface.
* **Sector** = smallest storage unit (usually **512 bytes** or **4096 bytes (4 KB)**)

Partition task

**Task 1:**

Create a **separate 500 MB partition** and mount it for the Tomcat project team.”

We already had a main disk (/dev/sdb, 400 GB with LVM) and added a second small virtual disk (/dev/sda, 2 GB).

## **Steps Performed**

### **1. Added a New Virtual Disk**

* In VirtualBox, a new 2 GB disk was attached → appeared inside Linux as /dev/sda.
* Verified with:  
    
   fdisk -l

### **2. Created a Partition (/dev/sda1)**

Entered partitioning tool:  
 fdisk /dev/sda

* Inside fdisk:  
  + n → new partition
  + p → primary
  + Partition number: 1
  + First sector: **Enter** (default)
  + Last sector: +500MB (made the partition 500 MB)
  + w → write and exit

✅ Result: /dev/sda1 created as a 500 MB slice of the 2 GB disk.

### **3. Formatted the Partition**

Made it usable with **ext4 filesystem**:  
  
 mkfs.ext4 /dev/sda1

✅ Now the kernel knows how to store files inside /dev/sda1.

### **4. Created a Mount Point**

Prepared a folder to “attach” the partition:  
  
 mkdir -p /opt/tomcat

✅ /opt is the standard place for optional software → /opt/tomcat chosen for Tomcat project.

### **5. Mounted the Partition**

Mounted the new partition to the folder:  
  
 mount /dev/sda1 /opt/tomcat

Checked with:  
  
 df -h | grep tomcat

✅ /dev/sda1 now visible as a 1 GB filesystem under /opt/tomcat.

### **6. Made the Mount Permanent (fstab)**

Found UUID of /dev/sda1:  
  
 blkid /dev/sda1

* Example UUID: 5bfc7f98-2dcc-4902-926e-14191b78477e

Edited /etc/fstab:  
  
 nano /etc/fstab

Added this line:  
  
 UUID=5bfc7f98-2dcc-4902-926e-14191b78477e /opt/tomcat ext4 defaults 0 2

* + **UUID** → unique ID of partition
  + **/opt/tomcat** → mount point
  + **ext4** → filesystem type
  + **defaults** → standard options
  + **0** → skip old dump backups
  + **2** → run fsck check after root

Tested without reboot:  
  
 mount -a

✅ System accepted the entry → will auto-mount at every boot.

fstab- mount

## **What is /etc/fstab?**

* Full name: File System Table

It is a configuration file in Linux, located at: /etc/fstab

* It tells Linux which partitions to mount, where to mount them, and how to mount them every time the system boots.

**Here’s the one-liner summary of all steps you followed to set up the dedicated Tomcat partition:**

1. **Added new 2 GB virtual disk in VirtualBox → Linux detected it as /dev/sda.**
2. **Created 1 GB partition on /dev/sda using fdisk → became /dev/sda1.**
3. **Formatted /dev/sda1 with mkfs.ext4 → made it ready for files.**
4. **Created mount point mkdir -p /opt/tomcat → folder where partition will be attached.**
5. **Mounted partition with mount /dev/sda1 /opt/tomcat → gave 1 GB usable space at /opt/tomcat.**
6. **Got UUID with blkid /dev/sda1 → unique ID of the partition.**

**Edited /etc/fstab → added  
  
 UUID=<uuid> /opt/tomcat ext4 defaults 0 2**

1. **to make the mount automatic at every reboot.**
2. **Tested config with mount -a and df -h | grep tomcat → confirmed partition is mounted.**

**Fstab**

| Mount manually (noauto in fstab) | sudo mount /mnt/data |
| --- | --- |

| Mount with custom options | sudo mount -o rw,uid=1000,umask=022 /mnt/data |
| --- | --- |

| Remount with updated fstab | sudo mount -o remount /mnt/data |
| --- | --- |

* keep dump and fsck value same if 0 then both 0, if fsck is other than 0 then dump will be 1, both are related
* if changes do, and fsck is 0 then it will not crash because it not check during boot

## Field for Permissions → the 4th field (options)

* That’s where file-system mount options and permissions are defined
* defaults- Uses default options (rw, suid, dev, exec, auto, nouser, async).

BAckup-rsync

## **What “Backup” Really Means**

**Goal:** Keep a **safe copy** of your important files in another location — for example:

* Another **disk**
* Another **machine**
* The **cloud**

## **🧱 Types of Backup**

| **Type** | **Description** | **Pros** | **Cons** |
| --- | --- | --- | --- |
| **Full Backup** | Copies **everything** each time | Simple to restore | Large size, slow |
| **Incremental Backup** | Copies **only changed files** since the last backup | Fast, small | Needs previous backups to restore |
| **Differential Backup** | Copies **changes since the last full backup** | Easier restore than incremental | Larger size than incremental |

## **🔁 Rsync Overview**

* Performs **incremental backups**
* **First run:** full backup  
   **Next runs:** only copies changed files
* Can **back up data remotely**
* **Does not compress** data by default  
   → Use tar or zip for compression if needed
* Can back up data to:
  + Another machine (remote server)
  + **NAS / SAN**
  + **Tape drives** (e.g., LTO)

## **📦 Basic Copy vs Rsync**

### **Using cp**

cp -rv /home/\* /opt/

* -r → recursive (copy folders)
* -v → verbose (show progress)
* **Best practice:** always use **absolute paths**
* cp asks before overwriting files

### **Using rsync**

rsync /home/\* /opt/

* Doesn’t overwrite everything — only **updates changes**

To copy **existing** files again:  
 rsync -r /home/\* /opt/

## **⚙️ Common Rsync Syntax**

rsync -avz source destination

**Flags Explained:**

* -a → archive mode (preserves permissions, time, etc.)
* -v → verbose
* -z → compress data during transfer (optional, helps when sending remotely)

You can also remember as:

rsync -parv source destination

(p = preserve, a = archive, r = recursive, v = verbose)

## **🧩 Important Rsync Options**

| **Option** | **Meaning** |
| --- | --- |
| --delete | Deletes files from destination that are not in source (keeps both in sync) |
| --progress | Shows progress of each file transfer |
| --dry-run | Simulates backup without making changes |
| --exclude | Skips certain files or folders |
| -z | Enables compression (for remote transfers) |

**Remember:** Rsync compares **modification time** of files — if changed, it copies the **whole file**, not just modified content.

## **📂 File State & Time Details**

| **Type** | **Description** |
| --- | --- |
| **Access Time (a time)** | When a file was last read or accessed |
| **Modify Time (m time)** | When the content was last modified |
| **Change Time (c time)** | When permissions or metadata changed |

If **inode number** changes in destination → it means the file was **modified** or **recreated**.

## **🧠 How Rsync Works (Internally)**

1. **Compares** files between source and destination.
2. Checks **modification time** and **size**.
3. Copies only files that are **new or changed**.
4. Keeps metadata (permissions, owner, timestamps).

Efficient and ideal for scheduled backups or syncing directories.

## **🌐 Remote Backup Using Rsync**

### **1️⃣ Basic Remote Command**

rsync -avz source/\* IP:/destination/path/

* Will ask for **password** of remote machine.

Example:  
  
 rsync -avz /home/user/Documents 192.168.1.5:/backup/

### **2️⃣ Remote with User Authentication**

rsync -avz source/\* user@IP:/destination/path/

Example:  
  
 rsync -avz /home/user/data user@192.168.1.5:/backup/data

* Requires **SSH access** to the remote system.

SAN/ LUN Scanning

## **What is SAN (Storage Area Network)?**

A **SAN (Storage Area Network)** is a **dedicated, high-speed network** that connects **servers** to **shared storage devices** like disks, SSDs, or tape drives.  
 It’s mainly used in data centers for **centralized storage**, **backup**, and **high availability**.

## **⚙️ Components in the Diagram**

### **1. Linux Server**

* The **client or host machine** (like your Linux system in the image).
* It does not have internal storage for data—it connects to the SAN to use storage as if it’s local.
* Inside it has:  
  + **HBA (Host Bus Adapter)** card to connect to SAN storage.
  + **Network Interface (CAT 5/6 or Fibre Channel cable)** to communicate.

### **2. HBA Card (Host Bus Adapter)**

* The **HBA** acts like a **bridge** between the **server** and the **SAN storage**.
* It connects through **Fibre Channel (FC)** or sometimes **iSCSI** (Ethernet).
* It allows the server to see remote disks as if they are local.

📍Think of HBA like a **special network card** for connecting to **storage networks** (not internet).

### **3. FC Cable / Optical Fibre**

* These are **high-speed cables** (e.g., Fibre Channel or Optical).
* They physically connect the **HBA** on the Linux server to the **SAN storage box**.
* Provide **fast, reliable, low-latency** data transfer.

### **4. SAN Box (Storage Array)**

* The **SAN box** is a storage unit that contains multiple **hard drives or SSDs**.
* It is connected to servers through **Fibre Channel switches or directly**.
* Internally, SAN storage is divided into **LUNs (Logical Unit Numbers)**.

🧩 Each **LUN** acts like a **virtual disk** that the server can use.

### **5. LUN (Logical Unit Number)**

* **LUN = a logical disk presented to the server.**
* In your image:  
  + **LUN1 = 1 TB**
  + **LUN2 = 3 TB**
  + **LUN3 = 500 GB**
* These come from the storage pool inside the SAN box.
* Each LUN have a unique ID called LUN ID.

📍When the Linux server connects to SAN, it “sees” each LUN as a **separate hard drive** (e.g., /dev/sdb, /dev/sdc, etc.).

## **🔄 How It All Works (Step-by-Step Sequence)**

1. **Server Setup** Your **Linux server** has an **HBA card** installed.
2. **Physical Connection** The **HBA** is connected to the **SAN box** through **Fibre Channel (FC) cables** or **CAT 6 (iSCSI)**.
3. **Storage Configuration (in SAN box)** Inside the SAN, storage admin creates **LUNs** (e.g., 1TB, 3TB, 500GB).  
    These are logical partitions of the total storage.
4. **LUN Mapping/Masking** Each LUN is **assigned (mapped)** to a specific server’s **HBA WWN (World Wide Name)**.  
    This controls which server can see which LUN.

**5. Discovery (on Linux Server)** The Linux OS, via HBA, discovers these LUNs using commands:  
  
 lsscsi

lsblk

fdisk -l

## **What Is LUN Scanning?**

**LUN scanning** means **detecting or discovering new SAN disks (LUNs)** that are presented (assigned) to a Linux server from the **SAN storage** — **without rebooting the system**.

## **What is SCSI, Target, initiater?**

**SCSI** stands for **Small Computer System Interface**.  
 It’s a **protocol (communication standard)** that defines **how computers (servers) talk to storage devices (disks, tapes, SANs, etc.)**.

SCSI works in a **client–server model**:

* The **client** is the **initiator** (usually your Linux server or host).
* The **server** is the **target** (usually the storage box / SAN).

So:

**Initiator sends requests** → **Target provides storage responses/data**.

## **The Main Command (Universal Way)**

echo "- - -" > /sys/class/scsi\_host/hostX/scan

## **Let’s Break Down Each Part**

| **Part** | **Meaning** | **Explanation** |
| --- | --- | --- |
| echo | Send input | Sends text (parameters) to a file or device in Linux. |
| "- - -" | Channel, Target, LUN | The **three fields** represent **SCSI address components**. Each dash means “scan all.” |
| > | Redirect operator | Sends (echo) output into the specified file — here, into the kernel SCSI scan interface. |
| /sys/class/scsi\_host/hostX/scan | File path | Interface file inside Linux kernel representing each HBA (SCSI host). hostX means host0, host1, host2, etc. |

## **Yes — You Are Absolutely Right**

When we say **“host”** in the SCSI or SAN context,  
 it refers to the **SCSI host adapter** (usually your **HBA**) as recognized by the **Linux kernel** —  
 not the whole physical server itself.

So one **physical Linux server** can have:

* **Multiple HBA cards**, and
* **Each HBA** can have **one or more ports** (like Port A and Port B).

Each of those ports will appear in Linux as a **separate hostX** under:

/sys/class/scsi\_host/

Example:

host0

host1

host2

host3

These could represent:

| **HBA Card** | **Port** | **Linux Name** |
| --- | --- | --- |
| HBA 1 | Port A | host0 |
| HBA 1 | Port B | host1 |
| HBA 2 | Port A | host2 |
| HBA 2 | Port B | host3 |

So yes   
 the **Linux kernel** sees each **HBA port** as an independent **SCSI host**.

## **⚙️ When You Do LUN Scanning**

When you run:

echo "- - -" > /sys/class/scsi\_host/host2/scan

Here’s what really happens inside the system 👇

### **🔹 Step 1: You (the admin) trigger a scan on host2**

This means:

“Hey kernel, tell HBA port #2 to look for new storage connections.”

### **🔹 Step 2: The kernel sends SCSI inquiry commands via that HBA port**

* It checks all **channels**, all **targets**, and all **LUNs** (- - - = full scan).
* This is the “outgoing” scan — your host is asking the SAN.

### **🔹 Step 3: The SAN (target side) responds**

It sends back information about any **mapped LUNs** that belong to your HBA’s **WWN** (World Wide Name).

This is the **incoming response** that says:

“I have LUN 0 = 1TB, LUN 1 = 3TB, LUN 2 = 500GB.”

### **🔹 Step 4: The kernel adds new devices**

Linux creates new device files under /dev/, such as:

/dev/sdb

/dev/sdc

/dev/sdd

You can then check them with:

lsblk

lsscsi

fdisk -l

**6. File System Creation** System admin formats the LUN using:  
  
 mkfs.ext4 /dev/sdb

Then mounts it:  
  
 mount /dev/sdb /data

**7. Usage** The Linux system now uses this SAN disk for **data storage, backups, or databases** just like a local drive.

## **How They All Connect (in Sequence)**

| **Layer** | **Component** | **Function** |
| --- | --- | --- |
| **1. PCIe Bus** | Motherboard interface | Connects HBA to CPU |
| **2. HBA Card** | Installed in PCIe slot | Converts PCIe data to storage protocol (SAS/SCSI) |
| **3. SAS Cable/Protocol** | Between HBA and disks | Transfers data physically |
| **4. SCSI Commands** | Logical layer | Manages disk operations (read/write, LUNs, etc.) |

e2FSCK- repair

### **What is FSCK**

* **FSCK (File System Consistency Check)** is a tool used in Linux to **check and repair file system errors** on a disk.
* It checks **file system structure**, not the data inside files.

## **⚙️ Why We Need FSCK**

* To fix **file system corruption** (e.g., from power failure, improper shutdown, bad sectors).
* To ensure the file system remains **stable and readable**.
* Automatically runs at boot if the system detects issues or after a certain number of mounts.

## **💾 Checking for Bad Blocks**

Command:

badblocks -v /dev/sda1

* badblocks scans the hard drive partition /dev/sda1 for **bad sectors**.
* -v → verbose (shows progress).
* It only **detects**, doesn’t repair.

## **🔧 Repairing with FSCK**

### **Step 1: Unmount the partition**

umount /dev/sda1

You must unmount first because FSCK works directly on the **disk**, not on the mounted (RAM-linked) file system.

### **Step 2: Run FSCK or e2fsck**

e2fsck /dev/sda1

* Runs a manual check.
* May ask **Yes/No** confirmation for each issue.

e2fsck -yc /dev/sda1

* -y → automatically answers *Yes* to all repairs.
* -c → checks for bad blocks.
* Gives a detailed **repair report**.

e2fsck or fsck repairs the file system, **not user data**.

## **🪶 Note on /etc/fstab FSCK Order**

In /etc/fstab, the **last column** defines FSCK check order:

1 → root (/) partition checked first

2 → all other partitions checked after

0 → skip fsck for that partition

Example:

UUID=xxxx / ext4 defaults 0 1

UUID=yyyy /home ext4 defaults 0 2

TOp-Process monitering

## **Linux Process Management – Quick Revision Notes**

### **1. What is a Process?**

* Everything running in **RAM** is called a **process**.
* Any **program or file** opened in memory is also treated as a **process**.
* Each process is identified and managed by the **Linux kernel** using a unique **PID (Process ID)**.

### **2. Checking Processes and Open Files**

* lsof → Lists all **open files and processes** in the system.  
   *(lsof = list open files)*

### **3. Understanding PID, PPID, and PID 0**

* **PID (Process ID):** Unique number assigned by kernel to every running process.
* **PPID (Parent Process ID):** ID of the process that started another process.
* **PID 0:** Refers to the **kernel’s idle scheduler**, the root of all processes.
* **Example:**
  + Firefox launcher runs first (in process).
  + When you open Firefox, its actual process starts and gets a new PID.

### **4. View Running Processes**

| **Command** | **Description** |
| --- | --- |
| ps -ef | Shows all running processes in **full format** (columns: UID, PID, PPID, CMD). |
| ps -el | Displays processes in **long format**, with extra details like priority and state. |
| ps aux | Lists all processes for **all users**, memory & CPU usage included. |

### **5. Find Process IDs**

| **Command** | **Description** |
| --- | --- |
| pidof firefox | Shows the **PID** of Firefox directly. |
| pgrep firefox | Same as above; finds the process ID by **name pattern**. |

### **6. Kill or Stop a Process**

| **Command** | **Description** |
| --- | --- |
| kill <PID> | Gracefully stops a process using its PID. |
| pkill firefox | Kills the process **by name**. |
| kill -9 <PID> | Forcefully terminates a process (cannot be ignored). |

💡 -9 = SIGKILL → immediate termination.

### **7. View Process Hierarchy**

* pstree → Displays processes in **tree format**, showing parent-child relationships.

### **8. Init and Systemd**

| **Version** | **Description** |
| --- | --- |
| **RHEL 6 and earlier** | init process (PID 1) is the **father of all processes**. |
| **RHEL 7 and later** | systemd replaced init, and now acts as **PID 1** – the root of all processes in modern Linux systems. |

ps process status

ps -el process status extended list

top Displays **live info** about processes and system Res usage

clt+alt+F1 virtual console in runlevel 3 (F1...F6)

firefox& to send any process in background work within the terminal

nohup furefox work also outside terminal keep process in background

fg, bg, jobs to check background process

# **Linux System Monitoring & Process Management – Quick Revision Notes**

## **⚙️ 1. Checking System Load (Like Task Manager in Linux)**

| **Tool** | **Purpose** | **Command / Description** |
| --- | --- | --- |
| **top** | Real-time system monitoring (like Task Manager). Shows CPU, memory, and process load. | top |
| **iostat** | Monitor **CPU + disk I/O** (HDD/SSD load). | iostat |
| **vmstat** | Shows **memory, CPU, swap, I/O**, and process statistics. | vmstat 1 |
| **ipref / iperf3** | Network performance test between two systems (bandwidth). | iperf3 -s (server), iperf3 -c <ip> (client) |
| **iometer** | GUI tool for testing disk I/O performance. |  |
| **tcpdump** | Captures and analyzes **network packets**. | tcpdump -i eth0 |

## **📈 2. Load Average (from top, uptime)**

* Shows **number of processes** running or waiting for CPU.
* Example: load average: 0.25, 0.20, 0.15
  + 1-min, 5-min, 15-min average.
* Compare against number of **logical CPUs (threads)**.
  + Load ≤ CPU count → OK
  + Load > CPU count → Overloaded

## **💾 3. Swap – Virtual Memory**

**Swap** is a dedicated area on the disk used when **RAM is full**.

* Keeps inactive pages to free RAM for active processes.
* Can be **partition** or **file**.

Check swap usage:  
  
 free -h

## **🧩 4. /proc Directory**

* Virtual directory containing **runtime system info**.
* Stores details about CPU, memory, processes, and kernel parameters.

| **Example** | **Description** |
| --- | --- |
| /proc/cpuinfo | CPU information |
| /proc/meminfo | Memory usage details |
| /proc/swaps | Active swap info |
| /proc/loadavg | System load averages |
| /proc/<pid>/ | Information about specific process |

## **🧮 5. CPU vs Core vs Thread**

| **Term** | **Meaning** | **Example** |
| --- | --- | --- |
| **CPU** | The physical processor (entire chip). | Intel i7-6600U |
| **Core** | Independent processing unit within CPU. | 2 cores |
| **Thread** | Virtual execution line per core (Hyper-Threading). | 2 threads/core → 4 logical CPUs |

👉 1 physical CPU → 2 cores → 4 threads → **4 logical CPUs visible to Linux**

## **⚡ 6. Buffers and Cache**

| **Term** | **Purpose** | **What It Stores** |
| --- | --- | --- |
| **Buffer** | Temporary space for **data being written/read to disk**. | File system metadata, I/O blocks |
| **Cache** | Stores **recently accessed files/programs** to speed up reads. | File contents, executables |

🟢 **Key Points**

* Buffer = The “pending tray” where the person keeps notes to write them later.
* Cache = The “desk” where recently used books stay handy for quick access.
* Both can be **reclaimed instantly** when RAM is needed.

Check memory:

free -m

## **🧠 7. Daemons (Background Services)**

* **Daemon** = background process running continuously until stopped.  
   (Example: sshd, crond, systemd)

| **Type** | **Description** | **Impact if Killed** |
| --- | --- | --- |
| **System daemon** | Core service started by OS (e.g. systemd, udevd) | System may crash |
| **Application daemon** | App-specific background service (e.g. mysqld, nginx) | Only that app stops |

## **📊 8. Redirecting Output from top**

top runs interactively (real-time), so it doesn’t normally redirect output.  
 Use the **-n** flag for one snapshot:

top -n1 > system\_report.txt

✅ Saves a single screen of top output to a file.

# **Linux Command: top — Process & Resource Monitoring**

## **1️⃣ Purpose of top**

The **top** command displays real-time information about **running processes**, **CPU usage**, **memory usage**, and **system load**.  
 It’s one of the most used tools for **performance monitoring** and **process diagnosis**.

## **2️⃣ Basic Usage**

### **Command:**

top

Shows a live, updating view of system resource usage and running processes.

### **Save the Output to a File**

To record the current output for later review:

top -n1 > /pakistan/sameer

* -n1 → Run top for **one iteration only** (not continuous).
* Output is saved to /pakistan/sameer for later analysis.

## **3️⃣ Helpful Keys and Flags**

* **h** → Help (displays toggle keys and command list inside top).
* **r** → Change the **Nice value** (priority) of a running process.
* **Toggle Keys** → You can press keys to sort or change views (like CPU, memory, user, etc.).

## **4️⃣ Understanding Process States**

Each process shown in top has a **state**, represented by a single letter under the **S** column.

| **Symbol** | **State** | **Meaning** |
| --- | --- | --- |
| **R** | Running | Process actively using CPU. |
| **S** | Sleeping | Process is idle, waiting for CPU (not currently executing). |
| **D** | Uninterruptible | Waiting for I/O (e.g., disk access). Cannot be killed easily. |
| **Z** | Zombie | Process finished but still in memory (waiting for parent cleanup). |
| **T** | Stopped | Suspended or paused process. |
| **I** | Idle / Kernel thread | Background kernel process not currently active. |
| **Ss** | Session maintained by kernel (daemon or service). |  |

⚠️ **Zombie Process Fix:** Zombie processes stay in RAM until parent process reaps them.  
 If many zombies persist → only solution is **reboot**, since kill won’t work.

## **5️⃣ Process Priority and Nice Value**

Processes have **priority** levels that determine how much CPU time they get.

| **Field** | **Description** |
| --- | --- |
| **PR** | Actual priority value used by scheduler. |
| **NI** | Nice value (controls how “nice” a process is toward others). |

### **Nice Value (NI)**

* Defines process scheduling priority.
* Range: **-20 (highest priority)** → **+19 (lowest priority)**
* Default: **0**

Lower NI → higher priority → more CPU time  
 Higher NI → lower priority → less CPU time

Example:

| **NI** | **Meaning** | **Behavior** |
| --- | --- | --- |
| **-19** | Very high priority | CPU favors this process |
| **0** | Default | Normal scheduling |
| **+20** | Very low priority | Process runs slower (background tasks) |

To change a process’s nice value inside top, press **r**, then enter the **PID** and new **NI** value.

🧠 **Remember:** Only the **root user** can set negative nice values.

## **6️⃣ Key Fields in top Output**

| **Field** | **Description** |
| --- | --- |
| **PID** | Process ID |
| **USER** | Owner of the process |
| **PR** | Actual priority (affected by NI) |
| **NI** | Nice value |
| **VIRT** | Virtual memory used by the process |
| **RES** | Resident memory (actual RAM used) |
| **SHR** | Shared memory |
| **S** | Process state (R, S, Z, etc.) |
| **%CPU** | Percentage of CPU being used |
| **%MEM** | Percentage of RAM being used |
| **TIME+** | Total CPU time consumed by process |
| **COMMAND** | Name of the running process or command |

## **7️⃣ Memory Columns Explained (VIRT, RES, SHR)**

These three columns show how each process is using system memory.

### **🟩 VIRT — Virtual Memory**

**Definition:**

* Total virtual address space the process can access.
* Includes:  
  + Code + data + shared libraries
  + Memory-mapped files
  + Swap space
  + Shared memory regions

**Key Point:** Not all of this memory is physically used — much of it may be reserved.

**Example:**

VIRT = 2000M

→ The process has **2 GB** of total address space available.

**Analogy:** Booking a hotel with **100 rooms (VIRT)** — you have access to all, but may use only 10 (RES).

### **🟦 RES — Resident Memory**

**Definition:**

* Actual physical RAM being used by the process (not swapped).
* Directly impacts real memory consumption.

**Example:**

VIRT = 2000M

RES = 800M

→ The process is actively using **800 MB** out of its 2 GB virtual allocation.

**Analogy:** Out of 100 hotel rooms booked, you are actually occupying **40**.

### **🟨 SHR — Shared Memory**

**Definition:**

* Portion of **RES** shared with other processes.
* Includes shared libraries (like libc) or shared memory regions.

**Example:**

VIRT = 2000M

RES = 800M

SHR = 200M

→ Out of 800 MB resident memory, **200 MB** is shared with others.  
 → Private memory = 800 – 200 = **600 MB**.

**Analogy:** You and your friend are sharing 200 MB of data (shared space), while 600 MB is your own.

## **8️⃣ Full Example Breakdown**

| **Field** | **Value** | **Description** |
| --- | --- | --- |
| VIRT | 2000M | Total address space (RAM + swap + mappings) |
| RES | 800M | Actual physical RAM in use |
| SHR | 200M | Shared with other processes |
| Private Memory | 600M | Unique to this process (RES - SHR) |

🧠 **In short:**

* **VIRT** → What process *can* use.
* **RES** → What it *is* using.
* **SHR** → What it *shares*.

## **9️⃣ Memory Verification Tools**

To view memory usage details for a specific process:

pmap -d <pid>

Shows:

* Virtual size
* Resident set size
* Shared vs private pages

SWAP

**SWAP**

## **What is Swap Memory**

* **Swap** is a type of **virtual memory** used when the physical **RAM is full**.
* It acts as an **extension of RAM** and resides on the **hard disk (HDD/SSD)**.
* Swap has its own **virtual file system** used by the kernel.
* It **stores inactive processes** from RAM to free space for active processes.
* **Active processes** always stay in **RAM** (for faster CPU access).
* **Inactive processes** get moved from **RAM → Swap**, and back when needed.

## **💡 Key Concepts**

| **Concept** | **Explanation** |
| --- | --- |
| **Virtual Memory** | A combination of physical RAM and swap space. |
| **Paging** (Windows term) | Equivalent to swap in Linux. |
| **Swap usage** | Happens automatically when RAM pressure is high — sometimes even before RAM is 100% full. |
| **Speed** | Swap is much slower than RAM (because it’s on disk). |
| **Size rule** | Traditionally set to **2× RAM**, but not a strict rule — actual size depends on workload and app requirements. |
| **Process timing** | Moving between RAM and swap happens in fractions of a second. |

## **⚙️ Swap Usage Behavior**

* When RAM starts filling:  
  + Inactive pages in RAM are **moved to swap**.
  + Active apps remain in RAM for fast access.
* When an inactive app is used again:  
  + It’s **brought back from swap → RAM**.
* Some swap may be used even if RAM isn’t full (depends on system settings and kernel behavior).

## **🧾 Swap in Windows**

* In Windows, **swap** is called **paging** (or the **page file**).
* Works the same way: moves inactive memory pages to disk temporarily.

## **🔍 Commands to Check Swap**

| **Command** | **Description** |
| --- | --- |
| free -m | Shows total, used, and free memory including swap (in MB). |
| swapon -s | Lists active swap partitions or swap files. |
| sudo fdisk -l | Displays all disk partitions, including the one marked as **swap**. |

## **🧱 Extending Swap Space**

### **Steps:**

1. Create a new **partition** (like /dev/sda3) using fdisk or parted.

Format it as swap:  
  
 sudo mkswap /dev/sda3

Activate it:  
  
 sudo swapon /dev/sda3

To make it permanent (after reboot):  
 Add this line to /etc/fstab:  
  
 /dev/sda3 swap swap defaults 0 0

✅ You can also create a **swap file** instead of a partition if you don’t want to repartition.

## **🔍 Checking Which Process Uses Swap**

Find the **PID** (process ID):  
  
 pidof firefox

Go to that process directory:  
  
 cd /proc/<PID>/

Open the **status** file:  
  
 cat status | grep VmSwap

1. → The VmSwap field shows how much swap memory that process is using.

# 

# **How to Extend Swap (Add a New Swap Partition)**

## **1) Check current swap and disks**

**Usage / activity**

free -m # or: free -h

swapon --show # or: cat /proc/swaps

vmstat 1 # watch paging; si/so (swap-in/out)

**Disk/partition layout**

lsblk -f # devices, FS type, UUIDs

sudo fdisk -l # partition tables (MBR/GPT)

## **2) Enable/Disable swap (when needed)**

**All swap off/on**

sudo swapoff -a

sudo swapon -a

**Specific device/file**

sudo swapoff /dev/sdb1

sudo swapon /dev/sdb1

## **3) Plan (your scenario)**

* Existing swap: **/dev/sda1**
* Goal: **Add another swap** on **/dev/sdb1** to extend total swap
* Partitioning tool: **fdisk** (MBR) or **gdisk/parted** (GPT)

## **4) Create the new swap partition**

**Using fdisk on /dev/sdb (MBR example)**

sudo fdisk /dev/sdb

# n (new) → p (primary) → 1 (part no.) → choose size (e.g., +8G)

# t (type) → 82 (Linux swap)

# w (write)

**If GPT**: use gdisk/parted and set **type = 8200 (Linux swap)**

Rescan & verify:

sudo partprobe /dev/sdb || sudo udevadm settle

lsblk -f /dev/sdb

## **5) Format as swap (not mkfs)**

sudo mkswap /dev/sdb1

sudo blkid /dev/sdb1 # note the UUID

## **6) Make it persistent in /etc/fstab**

Open fstab:

sudo nano /etc/fstab

Add **one** of the following lines (UUID recommended):

UUID=<PASTE-UUID> none swap sw,pri=100 0 0

# or

/dev/sdb1 none swap sw,pri=100 0 0

**Notes**

* Mount point is **none**
* Filesystem type is **swap**
* You **do not mount** swap; you **swapon/swapoff** it

## **7) Activate and confirm**

sudo swapon -a

swapon --show

free -m

You should now see both /dev/sda1 and /dev/sdb1 and a larger total swap.

## **8) Tips & troubleshooting**

* If swapon -a doesn’t show the new swap:  
  + Recheck fstab syntax (spaces, UUID)
  + Ensure mkswap was run on the right device
  + Confirm partition type: **82** (MBR) / **8200** (GPT)
* Set swap priority with pri= (higher number used first)

Monitor swapping:  
  
 vmstat 1

top # or htop

Consider a **swap file** if you want to avoid repartitioning:  
  
 sudo fallocate -l 4G /swapfile

sudo chmod 600 /swapfile

sudo mkswap /swapfile

echo '/swapfile none swap sw,pri=100 0 0' | sudo tee -a /etc/fstab

sudo swapon -a

## **Quick Recap**

1. Check current swap: free -m, swapon --show
2. Create /dev/sdb1 (type **82/8200**)
3. mkswap /dev/sdb1
4. Add to /etc/fstab with none swap sw,pri=100
5. swapon -a → verify with swapon --show & free -m

2️⃣ Network User Admin

Basic-IP Config

## **Static IP Setup on RHEL 6**

RHEL 6 uses **network service**, not NetworkManager.

### **Steps:**

1️⃣ **Check Interface**

ifconfig

Output example:

eth0 Link encap:Ethernet HWaddr 08:00:27:A5:10:FB

inet addr:192.168.1.10 Bcast:192.168.1.255 Mask:255.255.255.0

2️⃣ **Go to config folder**

cd /etc/sysconfig/network-scripts/

3️⃣ **Edit config file**

sudo nano ifcfg-eth0

4️⃣ **Add these lines**

DEVICE=eth0

BOOTPROTO=static

ONBOOT=yes

IPADDR=192.168.1.100

NETMASK=255.255.255.0

GATEWAY=192.168.1.1

DNS1=8.8.8.8

DNS2=1.1.1.1

5️⃣ **Restart network**

service network restart

or

/etc/init.d/network restart

6️⃣ **Verify**

ifconfig eth0

route -n

✅ You should see:

Destination Gateway Genmask Flags Iface

0.0.0.0 192.168.1.1 0.0.0.0 UG eth0

7️⃣ **Test Connectivity**

ping -c 4 192.168.1.1

ping -c 4 8.8.8.8

ping -c 4 google.com

✅ Static IP and gateway are working properly.

# 

## **Static IP Setup on RHEL 9**

## **Permanent IP Configuration**

### **📍 For RHEL 8 and later:**

Path:

/etc/NetworkManager/system-connections/

### **Step-by-Step (Static IP)**

1️⃣ Open config file:

sudo nano /etc/NetworkManager/system-connections/enp0s3.nmconnection

2️⃣ Find the IPv4 section:

[ipv4]

method=auto

3️⃣ Replace with manual settings:

[ipv4]

method=manual

addresses=198.168.18.201/24

gateway=198.168.18.1

dns=8.8.8.8;1.1.1.1;

4️⃣ (Optional) Disable IPv6:

[ipv6]

method=ignore

5️⃣ Save and secure:

sudo chmod 600 /etc/NetworkManager/system-connections/enp0s3.nmconnection

6️⃣ Reload and activate:

sudo nmcli connection reload

sudo nmcli connection down enp0s3

sudo nmcli connection up enp0s3

7️⃣ Verify:

ip addr show enp0s3

✅ Result: IP is now **static and permanent**

# 

# 

# 

# **🌐 Linux Networking Notes (Complete & Organized)**

## **1. Basic Requirements for Internet on Linux**

To get internet access on a Linux machine, you need:

* An **IP address**
* A **network card** (hardware)
* A **driver** for that network card installed in the server

## **2. Identifying Network Devices**

### **📘 PCI Devices**

Command to list PCI devices:

lspci

Detailed (verbose) output:

lspci -vvv

### **🧩 Network Card and Ports**

* Network cards are connected through the **PCI bus**.
* Each card may have multiple **ports**:  
  + eth0, eth1, etc. for Ethernet
  + wl… prefix for wireless interfaces
* Each port is associated with a **driver**.

## **3. Checking Network Details**

### **Using ifconfig**

Show network interface details:

ifconfig

### **Using ethtool**

Check network card properties:

ethtool eth0

**Key fields:**

* **Duplex:** send and receive mode
* **Link detected: yes** → connection is active

## **4. Assigning an IP Address**

Example (temporary):

ifconfig enp0s3 198.168.18.201 netmask 255.255.255.0

✅ Assigns IP immediately  
 ⛔ **Not permanent** — resets after reboot or service restart.

### **Why it’s not permanent:**

ifconfig only updates the current (in-memory) configuration; it doesn’t save it to a file.

## **5. Network Management Commands**

### **nmcli**

Structure:

nmcli [object] [action] [options]

Example:

nmcli connection show

* **object:** connection
* **action:** show

## **6. Understanding Network Devices & Concepts**

### **1️⃣ Switch**

A **Switch** connects multiple devices within the same LAN.

* Works at **Layer 2 (Data Link Layer)**
* Uses **MAC addresses**
* Builds **local connections only**

**Example:** 5 office PCs + 1 printer connected to a switch → all can share files locally.

🧠 Features:

* Fast & low latency
* No internet connection capability

### **2️⃣ Router**

A **Router** connects **different networks** (e.g., LAN ↔ Internet).

* Works at **Layer 3 (Network Layer)**
* Uses **IP addresses**
* Performs **NAT** and **DHCP**
* Provides **firewall/security**

🧠 Example:  
 Router connects your home LAN (192.168.1.x) to the Internet.

### **3️⃣ Gateway**

A **Gateway** is the “door” from your local network to another network (often the Internet).

✅ Usually, **Router = Gateway**

**Example:**

* PC1 → 192.168.1.10
* PC2 → 192.168.1.11
* Router → 192.168.1.1 (Internet-connected)

Here, 192.168.1.1 is both **router** and **gateway**.

**Summary Table:**

| **Term** | **Function** |
| --- | --- |
| Switch | Connects devices inside your LAN |
| Router | Connects your LAN to other networks |
| Gateway | IP address of the router (exit point) |

## **7. Local Network (LAN)**

A **LAN** connects devices in a limited area like a home, office, or school.

🏠 Example:

| **Device** | **IP Address** | **Communicates?** |
| --- | --- | --- |
| Laptop | 192.168.18.138 | ✅ Yes |
| Phone | 192.168.18.105 | ✅ Yes |
| Smart TV | 192.168.18.120 | ✅ Yes |

All connected via same router (gateway).

## **8. Network Basics – Short Notes**

| **Concept** | **Description** | **Command** |
| --- | --- | --- |
| **Local Network (LAN)** | Private Wi-Fi/cable network | ip route |
| **Local IP** | Assigned by router; e.g. 192.168.x.x | ifconfig, ip addr show |
| **Broadcast IP** | Ends with .255 | — |
| **Router / Gateway** | Connects LAN → Internet | — |
| **Public IP** | Given by ISP | curl ifconfig.me |
| **NAT** | Converts private IPs → single public IP | — |
| **Check LAN devices** | List connected hosts | arp -a or ip neigh show |
| **IPv4 vs IPv6** | Short vs long format addresses | — |

## **9. Network Address Structure**

| **Example** | **Meaning** |
| --- | --- |
| 192.168.0.0 | Network ID |
| 192.168.0.255 | Broadcast ID |

🧮 If 300 machines exceed one subnet, a **new network** must be added and **router** connects them.  
 Router example:

* 192.168.0.1
* 192.168.1.1

## **10. PCI Bus Address Explanation**

Example:  
 enp4s1f1

Meaning:

* enp4 → Ethernet device on **PCI bus 4**
* s1 → **Slot 1**
* f1 → **Function 1** (2nd port on same NIC)

Each function = one **port**, starting from 0.

DNS resolve

**What is DNS?**

DNS (Domain Name System) is like the phonebook of the Internet.  
 It translates domain names (like www.google.com) into IP addresses (like 142.250.190.68) that computers use to communicate.

When you type a website name in your browser:

1. Your system first checks its local cache.
2. If not found, it asks the configured DNS server to resolve that name to an IP address.
3. The DNS server replies with the IP, and your system connects to it.

**Traditional Method (/etc/resolv.conf)**

This file contains the nameserver entries used for DNS lookup.

Example:

cat /etc/resolv.conf

## 

## 

## 

## **DNS Resolve in RHEL 8,9**

## **Step 1: Check Your Current DNS Settings**

Run:

cat /etc/resolv.conf

You’ll see something like:

# Generated by NetworkManager

nameserver 192.168.1.1

nameserver 8.8.8.8

💡 **Note:** In RHEL 9, this file is automatically managed by **NetworkManager**, so manual edits here may not persist after reboot.

## **⚙️ Step 2: Set DNS Server Permanently (Method 1 – Using nmcli)**

Find your active connection name:

nmcli connection show

Example output:

NAME UUID TYPE DEVICE

Wired connection 1 1c9a3d9b-5f8d-4d49-b0e3-123456abcdef ethernet enp0s3

Now set your DNS server:

sudo nmcli connection modify "Wired connection 1" ipv4.dns "8.8.8.8 1.1.1.1"

sudo nmcli connection modify "Wired connection 1" ipv4.ignore-auto-dns yes

Then reload the connection:

sudo nmcli connection down "Wired connection 1"

sudo nmcli connection up "Wired connection 1"

✅ **Effect:** NetworkManager updates /etc/resolv.conf automatically with:

nameserver 8.8.8.8

nameserver 1.1.1.1

## **📝 Step 3: Verify the Change**

Run:

cat /etc/resolv.conf

or

nmcli dev show | grep DNS

You should see:

IP4.DNS[1]: 8.8.8.8

IP4.DNS[2]: 1.1.1.1

## **🧰 Step 4: Test DNS Resolution**

Try resolving a hostname:

ping google.com -c 3

If DNS works, you’ll get replies:

PING google.com (142.250.74.206) ...

If it fails, try:

systemd-resolve --status

and check if DNS servers are correctly assigned.

IP Addr & GW

## **IP Address & Gateway Notes**

### **🔹 1. Commands to Check IP and Routing**

| **Command** | **Purpose** |
| --- | --- |
| ip a | Show all IP addresses and interfaces |
| ip addr list | Same as above — detailed IP info |
| ip r l | Display current routing table |
| route -n | Show routes with gateways and metrics (numeric form) |

Example output:

Destination Gateway Genmask Flags Iface

0.0.0.0 192.168.18.1 0.0.0.0 UG enp0s3

192.168.18.0 0.0.0.0 255.255.255.0 U enp0s3

### **🔹 2. Managing the Default Gateway**

| **Command** | **Description** |
| --- | --- |
| route add default gw 192.168.18.1 | Adds a new default gateway |
| route del default gw 192.168.18.1 | Deletes the default gateway |
| (Result) | If deleted → ping shows *network unreachable* |

### **🔹 3. What Is a Gateway?**

* A **gateway** is the device (usually a **router**) that connects your system to **external networks** beyond the local LAN.
* It is the **exit point** for packets going outside your local subnet.

**Simple Logic:**

* Inside LAN (same network): Direct communication → *no gateway needed*
* Outside LAN (different network): Must go through the gateway

Example:

PC (192.168.18.10) → Router (192.168.18.1) → Internet

### **🔹 4. Default Gateway Concept**

* Acts as a **catch-all route** for any destination **not found** in the routing table.

Defined in routing table as:  
  
 Destination Gateway Genmask Flags Iface

0.0.0.0 192.168.18.1 0.0.0.0 UG enp0s3

* Meaning: “Send all unknown traffic to 192.168.18.1.”

### **🔹 5. Multiple Gateways**

You can configure **more than one gateway**, but:

* Only **one default** gateway (for 0.0.0.0/0)
* Others are **specific to their networks**

**Example setup:**

| **Interface** | **Network** | **Gateway** | **Use** |
| --- | --- | --- | --- |
| eth0 | 192.168.18.0/24 | 192.168.18.1 | Local LAN/Internet |
| eth1 | 10.0.0.0/24 | 10.0.0.1 | VPN or backup network |

### **🔹 6. How Linux Chooses the Gateway**

Linux checks **routing metrics** and **destination matching**.

Example routing table:

Destination Gateway Genmask Metric Iface

0.0.0.0 192.168.18.1 0.0.0.0 100 eth0

0.0.0.0 10.0.0.1 0.0.0.0 200 eth1

192.168.18.0 0.0.0.0 255.255.255.0 0 eth0

10.0.0.0 0.0.0.0 255.255.255.0 0 eth1

➡️ The **lower metric value (100)** = **higher priority**.  
 So here, eth0 (192.168.18.1) will be used first.

### **🔹 7. Simplified Routing Summary**

| **Destination Type** | **IP Range** | **Gateway Used** | **Why** |
| --- | --- | --- | --- |
| **Local Network** | 192.168.18.0 – 192.168.18.255 | None | Direct communication |
| **External Networks** | Anything else (0.0.0.0/0) | 192.168.18.1 | Needs router to reach Internet |

IP Classes/CIDR

# **IPv4 Classes and CIDR – Complete Notes**

## **1. What Is an IP Address**

* An **IP address** is a unique number that identifies every device on a network.
* IPv4 address has **32 bits** (written as four numbers, e.g. 192.168.18.201).
* Each IP is divided into:  
  + **Network part** → identifies the network (like a street name)
  + **Host part** → identifies the specific device (like a house number)

## **2. What Is a Subnet Mask**

* A **subnet mask** separates the network and host parts of the IP address.
* Example:  
   255.255.255.0 → means first 24 bits are for network, last 8 bits for host.

## 

## **IP Address Classes (Old System)**

| Class | First Octet Range | Default Mask | Network Bits | Host Bits | Networks (Old) | Hosts per Network (Old) | Notes |
| --- | --- | --- | --- | --- | --- | --- | --- |
| A | 1–126 | 255.0.0.0 (/8) | 8 | 24 | 2⁷ = 128 → minus 2 → 126 | 2²⁴ − 2 = 16,777,214 | Very large networks |
| B | 128–191 | 255.255.0.0 (/16) | 16 | 16 | 2¹⁴ = 16,384 | 2¹⁶ − 2 = 65,534 | Medium networks |
| C | 192–223 | 255.255.255.0 (/24) | 24 | 8 | 2²¹ = 2,097,152 | 2⁸ − 2 = 254 | Small networks |

## 

## **Example — Class C Network**

Let’s take 192.168.18.0/24 → 8 host bits

* Total IPs = 2⁸ = **256**
* Subtract 2 (network + broadcast)
* **Usable = 256 − 2 = 254**

So usable range:

* **Network ID** → 192.168.18.0
* **Broadcast** → 192.168.18.255
* **Usable hosts** → .1 → .254

That’s why we say a Class C network supports **254 devices**.

**what is 192.168.18.1?**

192.168.18.1 is the first usable IP address in this subnet.

But its purpose depends on how your network is set up:

| Address | Typical Use | Explanation |
| --- | --- | --- |
| 192.168.18.0 | Network ID | Identifies the subnet itself — not assignable to any device. |
| 192.168.18.1 | Default Gateway (Router) | Commonly assigned to the router or gateway that connects your LAN to the internet or another network. |
| 192.168.18.2 – 192.168.18.254 | Host IPs (computers, phones, printers, etc.) | Assigned to devices inside your LAN. |
| 192.168.18.255 | Broadcast Address | Used to send a message to *all* devices in that subnet. |

# 

IP Alaising

**IP Aliasing**

IP aliasing means assigning **multiple IP addresses** to a single **network interface card (NIC)** on a Linux system.

ifconfig eth0:1 192.168.1.11 netmask 255.255.255.0 up

## **Permanent Way (RHEL 9)**

You must do it through **NetworkManager** (nmcli), not with ifconfig.

### **🧭 Step-by-step**

#### **1️⃣ Check your connection name**

nmcli con show

Example output:

NAME TYPE DEVICE

enp0s3 ethernet enp0s3

Your connection name is enp0s3.

#### **2️⃣ Add a second static IP**

Now run:

sudo nmcli con mod "enp0s3" ipv4.method manual ipv4.addresses "192.168.1.10/24,192.168.1.11/24" ipv4.gateway "192.168.1.1" ipv4.dns "8.8.8.8 1.1.1.1"

sudo nmcli con up "enp0s3"

That adds both IPs permanently to the same physical interface.

#### **3️⃣ Verify**

ip addr show enp0s3

You should now see:

inet 192.168.1.10/24

inet 192.168.1.11/24

#### **4️⃣ Confirm in NetworkManager**

nmcli con show enp0s3 | grep ipv4.addresses

It should display both addresses — this means it will survive reboot.

### **🧩 If your main IP is DHCP**

If your first IP comes from DHCP but you just want to **add a static alias**, then use:

sudo nmcli con mod "enp0s3" ipv4.method auto

sudo nmcli con mod "enp0s3" ipv4.additional-addresses "192.168.1.11/24"

sudo nmcli con down "enp0s3"; sudo nmcli con up "enp0s3"

✅ This keeps DHCP and adds the extra IP permanently.

**Driver**

/etc/NetworkManager/system-connections/

## **In RHEL 6**

RHEL 6 used **old-style network scripts**, not NetworkManager profiles.

All your interface configuration files were stored in:

/etc/sysconfig/network-scripts/

Each file started with ifcfg- followed by the interface name, like:

/etc/sysconfig/network-scripts/ifcfg-eth0

/etc/sysconfig/network-scripts/ifcfg-eth0:1 ← for alias

/etc/sysconfig/network-scripts/ifcfg-eth1

IP/MAC addr

## **MAC Address (Layer 2 — Data Link Layer)**

Definition:  
 A MAC (Media Access Control) address is the *hardware address* permanently assigned to your network card (NIC) by the manufacturer.

### **Purpose:**

* Uniquely identifies a device on a local network (LAN).
* Used for communication inside the same network segment (Layer 2).
* Works at the Ethernet frame level.

**Use Cases:**

| Use Case | Description |
| --- | --- |
| LAN Communication | Switches use MAC addresses to deliver frames within the same network (no IPs needed inside Layer 2). |
| ARP Protocol | When you know an IP but need the MAC, ARP (Address Resolution Protocol) maps IP → MAC. |
| Network Security | MAC filtering allows routers or Wi-Fi APs to permit or deny devices by MAC. |
| Device Identification | Used to identify a unique physical network interface (each NIC has one). |

## **IP Address (Layer 3 — Network Layer)**

**Definition:**  
 An IP (Internet Protocol) address is a *logical address* assigned to identify a device across different networks.

**Example:**

192.168.1.10 (IPv4)

fe80::a00:27ff:fe9d:5bd2 (IPv6)

**⚙️ Purpose:**

* Uniquely identifies a device across different networks (LAN/WAN/Internet).
* Used for routing and communication beyond the local network.
* Works at the Network Layer (Layer 3).

**Use Cases:**

| Use Case | Description |
| --- | --- |
| Internet Communication | Needed to send/receive data between networks. Routers forward packets based on IPs. |
| Remote Access (SSH, Ping) | You reach systems using their IP address. |
| Server Hosting | Websites, mail servers, etc., use IPs to be reachable globally. |
| Subnetting and Network Design | Organizing networks, assigning subnets, controlling traffic. |

## 

## **How They Work Together**

| **Layer** | **Address Type** | **Example** | **Role** |
| --- | --- | --- | --- |
| Layer 2 (Data Link) | MAC Address | 08:00:27:9D:5B:D2 | Physical device identifier (local only) |
| Layer 3 (Network) | IP Address | 192.168.1.10 | Logical identifier for routing (global/local) |

When you send data:

1. You target an **IP address**.
2. The system uses **ARP** to find the corresponding **MAC address** on the LAN.
3. Data is sent to that MAC, but labeled for the destination IP.

## 

## 

## **Situation:**

You have two computers in one local network (LAN):

| **Device** | **IP Address** | **MAC Address** |
| --- | --- | --- |
| PC-1 | 192.168.1.10 | AA:AA:AA:AA:AA:AA |
| PC-2 | 192.168.1.20 | BB:BB:BB:BB:BB:BB |

PC-1 wants to **ping** PC-2.

## **⚙️ Step-by-Step: What Really Happens**

### **🥇 Step 1 — PC-1 checks its own network**

PC-1 sees:

“192.168.1.20 is in my same subnet (192.168.1.x).  
 I can send directly — no need for gateway.”

So PC-1 knows the **destination IP**, but **not the MAC** yet.

### **🥈 Step 2 — PC-1 asks, “Who has this IP?”**

It sends a **broadcast** packet on the LAN:

Who has 192.168.1.20? Tell 192.168.1.10

This is called an **ARP Request**.  
 It goes to **everyone** on the network.

### **🥉 Step 3 — PC-2 replies**

Only PC-2 owns that IP, so it answers:

192.168.1.20 is at BB:BB:BB:BB:BB:BB

### **🏅 Step 4 — PC-1 remembers this**

PC-1 saves the mapping in its **ARP table**:

192.168.1.20 → BB:BB:BB:BB:BB:BB

So next time, it doesn’t need to ask again — it already knows.

### **🏆 Step 5 — Real data is sent**

Now PC-1 sends the actual **ICMP ping** packet, wrapped inside an Ethernet frame:

From MAC: AA:AA:AA:AA:AA:AA

To MAC: BB:BB:BB:BB:BB:BB

The frame goes to the **switch**, and the switch uses the **MAC table** to deliver it to PC-2’s port.

### **🧩 Step 6 — PC-2 receives and replies**

PC-2 receives the ping and sends back a reply frame:

From MAC: BB:BB:BB:BB:BB:BB

To MAC: AA:AA:AA:AA:AA:AA

Switch delivers it back to PC-1.  
 PC-1 shows:

Reply from 192.168.1.20: bytes=32 time<1ms

✅ Communication complete!

## Two Ways to Change MAC Address

You can test with:

sudo ip link set dev enp0s3 down

sudo ip link set dev enp0s3 address 00:11:22:33:44:55

sudo ip link set dev enp0s3 up

✅ Verify:

ip link show enp0s3

You’ll see:

link/ether 00:11:22:33:44:55 ...

⚠️ This change disappears after a reboot or after NetworkManager reload.

## **Permanent MAC Change (RHEL 9 / NetworkManager)**

1️⃣ **Find your connection name**

nmcli con show

2️⃣ **Set a custom MAC address**

sudo nmcli con mod "enp0s3" 802-3-ethernet.cloned-mac-address 00:11:22:33:44:55

3️⃣ **Apply changes**

sudo nmcli con down "enp0s3"; sudo nmcli con up "enp0s3"

4️⃣ **Verify**

ip link show enp0s3

✅ Now the OS always loads with your new MAC each time the connection activates.

## **5️⃣ Restore Original MAC**

If you want to remove your custom MAC:

sudo nmcli con mod "enp0s3" 802-3-ethernet.cloned-mac-address ""

sudo nmcli con down "enp0s3"; sudo nmcli con up "enp0s3"

## **Command 1:**

sudo ip link set dev enp0s3 address 00:11:22:33:44:55

### **🔍 What it does**

This uses the **ip utility** (part of the iproute2 package) to **manually assign a new MAC address** to a network interface.

| **Part** | **Meaning** |
| --- | --- |
| sudo | Run as superuser (needed for network changes) |
| ip | Linux network configuration command |
| link | Refers to a network interface (link layer) |
| set | Modify its settings |
| dev enp0s3 | Target device name |
| address 00:11:22:33:44:55 | The new MAC address to assign |

## **Command 2:**

sudo nmcli con mod "enp0s3" 802-3-ethernet.cloned-mac-address 00:11:22:33:44:55

### **🔍 What it does**

This uses **NetworkManager’s CLI (nmcli)** to make the same change,  
 but at the **connection profile (OS-level)**, so it **persists** across reboots.

| **Part** | **Meaning** |
| --- | --- |
| nmcli | Command-line tool for NetworkManager |
| con mod "enp0s3" | Modify the saved connection profile named *enp0s3* |
| 802-3-ethernet.cloned-mac-address | Property that tells NM to use a custom MAC instead of the physical one |
| 00:11:22:33:44:55 | The MAC you want NetworkManager to apply |

802-3-ethernet.cloned-mac-address

| **Section** | **Meaning** |
| --- | --- |
| **802-3-ethernet** | Refers to **Ethernet (wired) settings** — based on the IEEE 802.3 Ethernet standard. |

## **The Real Meaning of “802-3”**

**IEEE 802.3** is the **international networking standard** for **wired Ethernet**.

### **802 Family (for context)**

| **IEEE Standard** | **Network Type** | **Example** |
| --- | --- | --- |
| **802.3** | **Ethernet (wired)** | LAN cables (RJ-45, fiber, etc.) |
| **802.11** | **Wi-Fi (wireless)** | WLANs (Wi-Fi 4, 5, 6, etc.) |
| **802.1Q** | VLAN tagging | Used in switches |
| **802.1D** | Spanning Tree Protocol | Prevents loops in switches |

ISO Model

## Notes:

Each **port on a network card** has its **own unique MAC address**.

## **What Is the “lo” Interface?**

lo stands for **Loopback Interface**.  
 It’s a **virtual network interface** inside your operating system — not a physical one.

It doesn’t connect to any cable, switch, or Wi-Fi.  
 It exists **entirely inside your own computer**.

## **⚙️ 2️⃣ Its IP Address**

By default, every Linux system assigns:

Interface: lo

IP Address: 127.0.0.1

Netmask: 255.0.0.0

This means any IP in the range **127.0.0.1 – 127.255.255.254** belongs to your local machine.

## **💡 3️⃣ Purpose: “Talk to Yourself”**

The loopback interface allows a computer to **send network traffic to itself** —  
 without ever leaving the machine.

It’s used for:

* **Testing and diagnostics**
* **Inter-process communication (IPC)** between applications
* **Local services** (like databases, web servers, or APIs) that don’t need to be accessed externally

## **OSI Model (Open Systems Interconnection Model)**

**Developed by:** ISO (International Organization for Standardization)  
 **Purpose:**

* To standardize how computers communicate over a network.
* Breaks communication into **7 layers**, each doing a specific job.

## **📘 OSI 7 Layers, Functions, and Common Protocols**

| **No.** | **Layer Name** | **Main Function** | **Example Protocols / Devices** |
| --- | --- | --- | --- |
| **7** | **Application Layer** | Interface between user and network; provides network services to applications (e.g., web, email). | **HTTP, HTTPS, FTP, SMTP, DNS, Telnet, SSH** |
| **6** | **Presentation Layer** | Translates, encrypts, or compresses data for the application. Ensures both systems understand data format. | **SSL/TLS, JPEG, PNG, MPEG, ASCII, EBCDIC** |
| **5** | **Session Layer** | Starts, manages, and ends communication sessions between applications. | **NetBIOS, RPC, PPTP, SOCKS** |
| **4** | **Transport Layer** | Responsible for end-to-end delivery, error checking, flow control, and segmentation. | **TCP, UDP, SCTP** |
| **3** | **Network Layer** | Handles logical addressing and routing of packets across networks. | **IP, ICMP, IPsec, RIP, OSPF, BGP** |
| **2** | **Data Link Layer** | Handles physical addressing (MAC), error detection, and frame transmission within the same network. | **Ethernet, ARP, PPP, HDLC, Frame Relay, VLAN (802.1Q)** |
| **1** | **Physical Layer** | Deals with hardware, signals, cables, and data transmission as bits (0s and 1s). | **Cables, Hubs, Repeaters, Wi-Fi (802.11), Fiber, Bluetooth** |

## **⚙️ Key Concepts**

### **🧩 Encapsulation (Sending Side):**

Each layer **adds its own header** to the data as it passes down the stack.

Application Data → Segments (TCP) → Packets (IP) → Frames (Ethernet) → Bits (Cable)

### **🧩 Decapsulation (Receiving Side):**

At the receiver, each layer **removes its header** in reverse order.

## **🌐 Example:**

Your computer (A) wants to open [**www.google.com**](http://www.google.com/) — that’s computer (B, Google’s server).

We’ll follow exactly **what happens in order**

### **🥇 Step 1 — You ask for something**

You type www.google.com in your browser and hit **Enter**.  
 Your computer wants to know: “Where is Google on the Internet?”

### **🥈 Step 2 — DNS (Find the address)**

Your system first checks:

* Do I already know Google’s IP? (from cache)
* If not → Ask a **DNS server** (like 8.8.8.8)

DNS replies:  
 ➡️ “www.google.com = 142.250.182.68”

Now your computer knows **where to send** the request.

### **🥉 Step 3 — Create the message (Application Layer)**

Your browser prepares the actual request:

GET / HTTP/1.1

Host: www.google.com

This is the **HTTP request** (human-readable message).

### **🏅 Step 4 — Wrap it in lower layers**

Your computer now wraps that data through the OSI layers:

1. **Transport layer (L4)** → adds TCP header (port numbers: 443 for HTTPS).
2. **Network layer (L3)** → adds IP addresses (yours + Google’s).
3. **Data Link layer (L2)** → adds MAC addresses for local delivery.
4. **Physical layer (L1)** → turns it all into signals (electrical or Wi-Fi).

### **🏆 Step 5 — Send it out**

* The signal goes from your PC → to your **switch/router**.
* The router checks the destination IP and sends it toward the Internet.
* Routers along the path forward the packet using **IP routing** until it reaches Google’s data center.

### **⚙️ Step 6 — Google receives it**

At Google’s server:

1. The Ethernet frame arrives.
2. It removes (decapsulates) each layer:  
   * MAC → IP → TCP → HTTP.
3. The **web server** (Layer 7) reads the HTTP request:  
    “Oh! Someone wants my homepage.”

### **💾 Step 7 — Google sends reply**

The server builds a **response**:

HTTP/1.1 200 OK

<html>Google Homepage...</html>

Then it wraps it back down through the layers (L7 → L1)  
 and sends it back along the same path — through routers → your ISP → your router → your PC.

### **📥 Step 8 — You receive and display it**

Your computer unwraps it in reverse (L1 → L7):

* Gets the IP packet
* Checks TCP sequence
* Decrypts HTTPS
* Browser displays the webpage

✅ You now see **Google’s homepage**!

## **🧩 9️⃣ Summary of the Whole Journey**

| **Step** | **What Happens** | **OSI Layers Involved** |
| --- | --- | --- |
| 1 | User requests website | 7 |
| 2 | DNS finds IP address | 7, 3 |
| 3 | HTTP request built | 7 |
| 4 | Data wrapped with TCP/IP | 4, 3 |
| 5 | Sent over network (MAC, signals) | 2, 1 |
| 6 | Server unwraps and processes | 1 → 7 |
| 7 | Server sends reply | 7 → 1 |
| 8 | Client unwraps and shows result | 1 → 7 |

Ports

## **How to port open**

start service service tomcat start

**How to close**

close service

**List of ports**

/etc/services

## **Protocol can be changed with inform to client**

## **Common Ports to Remember**

| **Service** | **Port** | **Protocol** | **Purpose** |
| --- | --- | --- | --- |
| **HTTP** | 80 | TCP | Used for normal websites (not secure) |
| **HTTPS** | 443 | TCP | Used for secure websites (encrypted) |
| **DNS** | 53 | TCP/UDP | Converts domain names to IP addresses |
| **SSH** | 22 | TCP | Secure remote login to Linux servers |
| **Telnet** | 23 | TCP | Old remote login (not secure) |
| **MySQL** | 3306 | TCP | Database service port for MySQL server |
| **FTP** | 21, 20 | TCP | File Transfer Protocol (21 = control, 20 = data) |
| **SMTP** | 25 | TCP | Sends emails (Mail Transfer) |
| **POP3** | 110 | TCP | Receives emails (old method) |
| **IMAP** | 143 | TCP | Modern email receive protocol |
| **IMAPS (secure IMAP)** | 993 | TCP | Encrypted email retrieval |
| **POP3S (secure POP3)** | 995 | TCP | Encrypted POP3 |
| **LDAP** | 389 | TCP | Directory services (used in AD, authentication) |
| **RDP (Remote Desktop)** | 3389 | TCP | Remote access to Windows systems |
| **Tomcat** | 8080 | TCP | Web application server (HTTP alternative port) |
| **Samba / SMB** | 137, 139, 445 | TCP/UDP | File sharing in Windows/Linux |
| **Squid** | 3128 | TCP | Proxy server port |

## 

## **What is a “Port”?**

* A **port** is not a physical thing (in this context).
* It’s a **virtual number** used by the operating system (OS) to identify **which program or service** should handle a specific type of network traffic.

So when data arrives at your computer’s IP, the **port number tells the OS which application** should receive it.

Every IP connection has **two main parts**:

IP address + Port number = Socket

Example:

192.168.1.10:80 → 142.250.182.68:443

| **Part** | **Meaning** |
| --- | --- |
| 192.168.1.10 | Your computer’s IP |
| :80 | Port number used by browser (HTTP) |
| 142.250.182.68 | Server IP (Google) |
| :443 | Port on Google’s server (HTTPS) |

## **Port Ranges**

| **Range** | **Type** | **Example Use** |
| --- | --- | --- |
| **0–1023** | **Well-known / System ports** | HTTP(80), HTTPS(443), FTP(21), SSH(22) |
| **1024–49151** | **Registered ports** | MySQL(3306), Nginx(8080), Postgres(5432) |
| **49152–65535** | **Dynamic / Ephemeral ports** | Used temporarily by clients (like browsers) |

When you open a website, your browser uses a **random port (e.g., 50123)** to talk to the server’s **fixed port (443)**.

## **Here’s What You’re Saying — and How It Actually Works**

You said:

When PC1 sends an HTTP request, all communication follows OSI layers and packet encapsulation. When it reaches the server, it checks the HTTP port; if the service is running, it decapsulates. Otherwise, it replies “not available.”

✅ That is **exactly correct in concept** — here’s the same process step-by-step explained clearly 👇

## **⚙️ Step-by-Step: How It Actually Happens**

### **🥇 1. Client (PC1) creates HTTP request**

Application layer creates the **HTTP** message:  
  
 GET /index.html HTTP/1.1

Host: example.com

* This is at **Layer 7 (Application)**.

### **🥈 2. OS encapsulates it**

Each lower layer adds its header:

| **Layer** | **What It Adds** | **Example** |
| --- | --- | --- |
| L4 (Transport) | TCP header (source port, destination port) | Src port: 50500 → Dst port: 80 |
| L3 (Network) | IP header (source + destination IPs) | 192.168.1.10 → 203.0.113.5 |
| L2 (Data Link) | Ethernet header (MAC source + destination) | AA:AA:AA:AA:AA:AA → BB:BB:BB:BB:BB:BB |

Now the **frame is ready** to go on the wire.

### **🥉 3. The packet travels through the network**

* Sent to the **router**, then over the internet using **IP routing**.
* Each router only looks at the **IP header (Layer 3)** — it doesn’t care about HTTP or ports.

### **🏅 4. Server receives the frame**

At the destination:

* The Ethernet frame is received by the NIC.
* Layers are **unwrapped (decapsulation)** in reverse order:

| **Layer** | **What Happens** |
| --- | --- |
| L2 | Checks MAC → if destination MAC matches, keep it |
| L3 | Checks IP → if destination IP matches, keep it |
| L4 | Checks destination **port number** (e.g., 80 for HTTP) |
| L7 | Sends data to web server application |

### **🧠 5. Port Check Happens Here**

* The **Transport Layer (TCP)** checks:  
   “Is there any process listening on port 80?”
* If **yes** → data handed to that program (like Nginx or Apache).

If **no** → the OS sends back:  
  
 TCP RST (Reset)

* meaning **“Port closed — no service running.”**

✅ So yes, **the port number is the key** that decides where the data goes or whether it’s rejected.

### **🏆 6. Web Server Receives and Replies**

* If HTTP service is active → it reads the message, processes it, and sends a **response packet**.

Response travels back to client:  
  
 203.0.113.5:80 → 192.168.1.10:50500

* The client decapsulates in reverse and the browser displays the page.

### **❌ If Service is Not Running**

If port 80 is **not open**, the OS immediately replies:

TCP RST (Connection refused)

Client then shows:

Connection refused / Site can’t be reached

## **💡 So You’re Absolutely Right**

Data is **encapsulated layer by layer**, sent across the network, then **decapsulated** on the server — but **only** if a service is listening on that port.  
 Otherwise, the OS rejects it.

### **🧩 Summary Flow**

1️⃣ PC1 creates HTTP request

2️⃣ TCP adds port info

3️⃣ IP adds addressing

4️⃣ Ethernet adds MAC

5️⃣ Packet travels through routers

6️⃣ Server receives → checks port

7️⃣ If service exists → decapsulate → respond

8️⃣ If not → send “connection refused”

## **Real Encapsulated Data**

### **Suppose you open a web page (HTTP)**

**Application Data:**

GET / HTTP/1.1

Host: www.google.com

**Encapsulated (simplified view):**

Ethernet Header:

Destination MAC: 00:50:56:C0:00:01

Source MAC: 00:0C:29:3E:8C:8B

Type: 0x0800 (IPv4)

IP Header:

Source IP: 192.168.1.10

Destination IP: 142.250.182.68

Protocol: TCP

TCP Header:

Source Port: 50500

Destination Port: 80

Flags: SYN (connection setup)

HTTP Data (Payload):

GET / HTTP/1.1

Host: www.google.com

When you view this in Wireshark, you’ll see exactly that structure:

Frame 1: 66 bytes

Ethernet II

Internet Protocol Version 4

Transmission Control Protocol

Hypertext Transfer Protocol

## **🧩 5️⃣ Reverse Happens at Receiver (Decapsulation)**

When the server receives the frame:

1. Ethernet header is removed → check MAC
2. IP header removed → check IP
3. TCP header removed → check port
4. Application data passed to the web server (HTTP)

Then the **same process happens in reverse** for the response.

this one line you wrote:

[ Ethernet Frame [ IP Packet [ TCP Segment [ Application Data ]]]]

is **the most important structure in networking** — it shows exactly **how data is wrapped (encapsulated)** before being sent.

Let’s break it down step by step in **simple, real-world language** 👇

## **🧠 Think of it like layers of a parcel 📦**

When you send something through the mail:

1. You write your message (content)
2. Put it in an envelope (TCP)
3. Write sender/receiver addresses (IP)
4. Put that envelope in a shipping box (Ethernet)
5. Send it through trucks/wires (Physical)

Same logic applies in networking —  
 each OSI layer **wraps the upper layer’s data** before sending it.

## 

## **Example**

## **🧩 1️⃣ Application Data (Layer 7)**

This is the **real message** — what the user or software actually wants to send.

Example:

GET /index.html HTTP/1.1

Host: www.google.com

It could also be:

* An email (SMTP)
* A file transfer (FTP)
* A web request (HTTP/HTTPS)
* A DNS query

👉 This is the *payload* created by your app — browser, mail client, etc.

## **🧩 2️⃣ TCP Segment (Layer 4 – Transport)**

The **Transport layer** (usually TCP or UDP) wraps the Application Data.

It adds a **TCP header** containing:

* Source port (random)
* Destination port (e.g., 80 for HTTP, 443 for HTTPS)
* Sequence number (for data ordering)
* Acknowledgment number
* Flags (SYN, ACK, FIN)
* Checksum (for errors)

### **Example (simplified):**

TCP Header:

Src Port: 50500

Dst Port: 80

Seq: 1

Ack: 0

Flags: SYN

Now you have:

[TCP Header + Application Data]

= called a **TCP Segment**

✅ Purpose: Ensures reliable delivery between two processes (applications).

## **🧩 3️⃣ IP Packet (Layer 3 – Network)**

Now the **Network layer** (IP) wraps that TCP segment inside an **IP header**.

It adds:

* Source IP (your PC)
* Destination IP (server)
* Protocol type (e.g., TCP = 6, UDP = 17)
* Time To Live (TTL)
* Header checksum

### **Example:**

IP Header:

Src IP: 192.168.1.10

Dst IP: 142.250.182.68

Protocol: TCP

Now you have:

[IP Header + TCP Segment + Application Data]

= called an **IP Packet**

✅ Purpose: Makes sure the packet can travel across networks to the right computer.

## **🧩 4️⃣ Ethernet Frame (Layer 2 – Data Link)**

Now the **Data Link layer** (Ethernet) wraps that IP packet in an **Ethernet frame** to send it over the **local network (LAN)**.

It adds:

* **Destination MAC address** (router or next hop)
* **Source MAC address** (your computer)
* **Type field** (to show this frame carries an IP packet)
* **Frame Check Sequence (FCS)** for error checking

### **Example:**

Ethernet Header:

Dest MAC: 00:50:56:C0:00:01

Src MAC: 00:0C:29:3E:8C:8B

Type: IPv4 (0x0800)

Now you have:

[Ethernet Header + IP Header + TCP Header + Application Data + FCS]

= called an **Ethernet Frame**

✅ Purpose: Delivers the data from one device to another **inside the same LAN**.

## **⚡ 5️⃣ Then, It’s Converted to Bits (Layer 1 – Physical)**

Finally, the **Physical layer** converts the frame into **electrical signals or light pulses** to send it through cables or Wi-Fi.

0101010101010010110101101100101...

Routers, switches, and NICs handle this part.

## **🔁 6️⃣ On the Receiver Side: Decapsulation**

When the frame reaches the destination (server):

1. **Ethernet layer** removes Ethernet header → passes up IP packet
2. **IP layer** removes IP header → passes up TCP segment
3. **TCP layer** removes TCP header → gives Application Data
4. **Application layer** reads your HTTP request

Then the same wrapping process happens in reverse for the reply.

### **At Layer 3 (IP)**

IP’s job is to deliver packets.  
 But IP can carry different kinds of data (called *payloads*):

* TCP
* UDP
* ICMP
* GRE
* ESP (IPsec)

So IP must tell the receiver:

“The payload I’m carrying is TCP (protocol = 6).”

When the packet arrives, the receiving system looks at that number and passes the payload to the correct handler — e.g. TCP.

### **At Layer 4 (TCP or UDP)**

Now, TCP or UDP must also tell the receiver:

“The payload I’m carrying belongs to HTTP, or DNS, or SSH…”

So TCP/UDP use **port numbers** for that:

* Port 80 → HTTP
* Port 443 → HTTPS
* Port 22 → SSH
* Port 53 → DNS

This way, the OS knows **which application** should get the data.

**TCP header example:**

Src Port: 50500

Dst Port: 80

Port Sniffing

## **Practical Troubleshooting Flow**

1. Check connectivity → ping, traceroute, mtr
2. Check DNS → dig, nslookup
3. Check bandwidth usage → iftop, nethogs, bmon
4. Check throughput → iperf3, speedtest-cli
5. Check errors on interface → ip -s link, ethtool -S eth0
6. Check firewall/routes → ip route, iptables -L
7. Capture traffic → tcpdump or Wireshark
8. Check server load → top, htop
9. Correlate all results → Find whether slowness is link, DNS, CPU, or remote-side issue.

## 

## **Network Scanning Basics**

### **1. Check your own network information**

| **Command** | **Purpose** |
| --- | --- |
| ifconfig | Show your local (private) IP, MAC address, and interface info |
| ip r l | Show the routing table → reveals network IP, gateway (router), and machine IP |
| curl -4 ifconfig.me | Display your **public IP** (Internet-facing address) |

🔹 Example:  
 Private IP → 192.168.18.50  
 Router IP → 192.168.18.1  
 Network → 192.168.18.0/24  
 Public IP → shown by curl -4 ifconfig.me

## **🔎 Nmap — Network Mapping**

### **2. Scan a single host**

nmap 192.168.18.1

→ Scans one IP (router in this case) and shows open ports.

### **3. Scan an entire network**

nmap 192.168.18.0/24

→ Scans **all devices** on that subnet (IPs .1 → .254).

/24 = subnet mask 255.255.255.0 → 256 total addresses.

### **4. What the results mean**

| **State** | **Meaning** |
| --- | --- |
| **open** | Service is accepting connections |
| **closed** | Port reachable but no service running |
| **filtered** | Firewall is blocking the probe |
| **host up** | Device responded to ping/probe |

### **5. Useful Nmap options**

| **Option** | **Function** |
| --- | --- |
| -sn | Ping sweep (discover live hosts, no port scan) |
| -sS | SYN (stealth) scan |
| -Pn | Skip ping (for hosts that block ICMP) |
| -p 80,443 | Scan specific ports |
| -sV | Detect service version |
| -O | Detect OS type |

Example:

sudo nmap -sS -Pn -sV 192.168.18.1

## **💬 Telnet and Netcat (Port Testing)**

### **6. Telnet — check if a specific port is open**

telnet 192.168.18.170 80

* If the screen clears → port **open**
* If it says *connection refused* → port **closed**
* If it hangs → port **filtered/firewalled**

### **7. Netcat (nc)**

nc 192.168.18.1 80

* No response = port open
* “Connection refused” or text output = port closed

Netcat is also used for creating or listening on custom ports:

nc -lvp 8080 # listen on port 8080

## **🌐 Connection Monitoring**

### **8. Netstat — view network connections**

| **Command** | **Meaning** |
| --- | --- |
| netstat -an | Shows all active connections (numeric) |
| netstat -tulnp | Shows listening ports with PID and program name |

**States:**

* LISTEN → waiting for incoming connection
* ESTABLISHED → active connection between two devices

## **📶 Ping — Test Reachability**

### **9. Basic ping**

ping 192.168.18.1

Checks if the device is up (ICMP echo request/reply).

### **10. Useful ping options**

| **Option** | **Description** |
| --- | --- |
| -c 6 | Send only 6 packets |
| -s 1024 | Increase packet size to test performance |
| -I eth1 | Send packets through a specific interface |

**Concepts:**

* Ping uses **ICMP**, not TCP/UDP (so it has **no port**).
* **Echo Request (ID 8)** → **Echo Reply (ID 0)**
* Works on **Network Layer (Layer 3)**.

## **🧭 Traceroute — Path Tracing**

### **11. Trace packet route to a destination**

traceroute google.com

Shows all intermediate routers (hops) between you and the target.  
 Each line = one hop’s IP and round-trip time.

**Tcpdump**

**tcpdump** is a packet-capture tool that listens to a network interface and records network traffic that matches a filter.

## **What tcpdump does (plain)**

* Captures packets seen on a network interface (Ethernet/Wi-Fi).
* Lets you filter which packets to capture (IP address, port, protocol, etc.).
* Prints packet summaries to terminal (human readable) or writes raw packets to a .pcap file for offline analysis (Wireshark/tshark).
* Can show packet contents (ASCII / hex) for debugging protocols.

## **Basic syntax**

sudo tcpdump [options] [expression]

* options change output, file write/read, snaplen, verbosity, etc.
* expression (BPF) filters traffic: host, src, dst, port, tcp, udp, icmp, portrange, logical and/or/not.

## **Common examples (practical)**

### **1. Show live traffic on default interface**

sudo tcpdump -nn

* -n numeric IPs/ports (no DNS/service name lookup). Double -nn to also avoid port->name resolution in some versions.

### **2. Capture traffic on a specific interface**

sudo tcpdump -i eth0 -nn

Use -i any to capture on all interfaces.

### **3. Capture only traffic to/from a host**

sudo tcpdump -i eth0 -nn host 192.168.1.50

### **4. Capture only TCP port 80 (HTTP) traffic**

sudo tcpdump -i eth0 -nn tcp port 80

Package MAnagement

## **Linux Package Management — Quick Notes**

### **💾 Installation Concept**

* **Installing a file** means **writing it to the hard disk**.
* In **Windows**, programs are distributed as .exe files.
* In **Linux**, programs come as **packages**, e.g. xyz-1.2.rpm.  
  + xyz → program name
  + 1.2 → version number
  + .rpm → Red Hat Package Manager format

## **🐧 Linux Distribution Types (by Package Format)**

Linux has two main package management families:

| **Type** | **Extension** | **Common Distros** | **Package Manager** |
| --- | --- | --- | --- |
| **RPM-based** | .rpm | Red Hat, CentOS, Fedora, SUSE, SLES | rpm, yum, dnf |
| **DEB-based** | .deb | Debian, Ubuntu, Linux Mint, Sabayon | dpkg, apt |

## **⚙️ Installation Commands**

| **Distribution** | **Command** | **Example** | **Description** |
| --- | --- | --- | --- |
| **CentOS / Red Hat / Fedora (RPM)** | rpm -ivh packagename.rpm | rpm -ivh zsh.rpm | Installs RPM package |
| **Debian / Ubuntu (.DEB)** | dpkg -i packagename.deb | dpkg -i zsh.deb | Installs DEB package |

**Options in RPM:**

* -i → install
* -v → verbose
* -h → show progress hash marks

## **📂 RPM Database**

* **RPM maintains a local database** of all installed software.

To list everything installed:  
  
 rpm -qa

* (-q = query, -a = all)

## **RPM (Red Hat Package Manager)**

**RPM also used database**

**/var/lib/rpm**

**Definition:**

rpm is the *low-level* package management tool for Red Hat-based systems.  
 It installs, removes, and queries .rpm package files directly.

**Key points**

* Works on individual package files (no dependency resolution).
* Checks signatures, installs/uninstalls, verifies files.
* Doesn’t automatically download or resolve dependencies.

**Common commands**

# Install a package file (must already have all dependencies)

sudo rpm -ivh package.rpm

# Upgrade or install

sudo rpm -Uvh package.rpm

([rpmpbne.net](http://rpmpbne.net) to search latest version of rpm

arch show architecture

x mean 64bit

i mean 32 bit

if you copy link location the wget link

wget link it automatically download it

)

# Remove a package

sudo rpm -e packagename

# Query installed package

rpm -q packagename

# List all installed RPMs

rpm -qa

# # Remove forcefully if give an error

rpm -ivh --force package name

**Think of it as:** 📦 The *engine* that installs and tracks packages, but you must handle dependencies manually.

## **🧰 2. YUM (Yellowdog Updater Modified)**

**Definition:** yum is the *high-level* package manager built **on top of RPM**.  
 It can automatically handle **dependencies**, **repositories**, and **updates**.

**Key points**

* Uses online repositories (mirrors) to fetch packages.
* Automatically installs dependencies.
* Supports update groups, rollbacks, and history.

**Common commands**

# Install a package (and dependencies)

sudo yum install zsh

# Remove a package

sudo yum remove zsh

# Update all packages

sudo yum update

# Search for a package

sudo yum search zsh

# Show package info

sudo yum info zsh

**Think of it as:** ⚙️ A *smarter layer* over RPM that manages repositories and dependency resolution.

## **🚀 3. DNF (Dandified YUM) — *Next-generation YUM***

**Definition:** dnf is the modern replacement for yum (default in **RHEL 8+, CentOS 8+, Fedora 22+**).  
 It’s faster, more stable, and built with a better dependency engine (libsolv).

**Key improvements**

* Better performance and memory efficiency.
* Clearer dependency error messages.
* Automatic history, rollback, and transaction safety.
* Compatible syntax with yum.

**Common commands**

sudo dnf install zsh

sudo dnf remove zsh

sudo dnf update

sudo dnf search zsh

sudo dnf info zsh

sudo dnf history

**Think of it as:** ⚡ The *modern replacement* for yum, still using RPM underneath but with smarter dependency handling.

**YUM** or **DNF** just make your life easy by handling **repositories** and **dependencies**.

But when the package is actually installed, it’s the **RPM system** that does the final job of unpacking files, registering them in the database, and tracking them.

/var/lib/rpm ← stores installed package info

### **After installation**

Once installed (by yum, dnf, or rpm), every package is recorded in the same **RPM database**:

Check it:

rpm -qa | grep zsh

## **Where the packages come from**

YUM gets the package from **repository servers** configured on your system.  
 These are just **HTTP/HTTPS or FTP URLs** that host .rpm files.

Each repo has a metadata file (repodata/repomd.xml) that lists:

* package names
* versions
* dependencies
* download locations

So when you run yum install zsh, YUM:

1. Reads your configured repo list from /etc/yum.repos.d/.
2. Contacts those repo URLs.
3. Downloads the correct .rpm file for **zsh** and any required dependencies.
4. Passes them to **RPM** for installation.

## **📁 2. Repository file locations**

Repository configuration files are stored in:

/etc/yum.repos.d/

Examples:

CentOS-Base.repo

CentOS-AppStream.repo

CentOS-Extras.repo

Each file contains something like:

[baseos]

name=CentOS Stream $releasever - BaseOS

baseurl=http://mirror.centos.org/centos/$releasever/BaseOS/$basearch/os/

enabled=1

gpgcheck=1

gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-CentOS-Stream-$releasever

This tells **YUM** where to fetch packages from.

## **🌍 3. Types of repositories**

| **Repo Type** | **Description** |
| --- | --- |
| **Base / BaseOS** | Core system packages |
| **AppStream** | User applications and utilities |
| **Extras / EPEL** | Extra or community packages |
| **Updates** | Latest security and bug fixes |
| **Third-party** | External repos (like Remi, ELRepo, etc.) |

## **⚙️ 4. Verify which repo provided a package**

After installation, you can check which repo it came from:

yum info zsh

# or

dnf info zsh

Example output:

Name : zsh

Version : 5.8

Release : 1.el9

From repo : baseos

Summary : Powerful command line shell

## **🧠 5. Extra tips**

To **list all repos**:  
  
 yum repolist all

To **see only enabled repos**:  
  
 yum repolist enabled

To **show the actual download URL**:  
  
 yumdownloader zsh

To **add a new repo**, place a .repo file in /etc/yum.repos.d/.

## **Steps to Install .rpm from USB (Offline)**

### **🧩 Step 1. Plug in the USB**

Once you insert the USB, check that it’s detected:

lsblk

Example output:

sdb1 8:17 1 4G 0 part /run/media/user/USB

### **💾 Step 2. Mount the USB (if not auto-mounted)**

If you don’t see it in /run/media/..., mount it manually:

sudo mkdir /mnt/usb

sudo mount /dev/sdb1 /mnt/usb

Now your USB contents are available at /mnt/usb.

### **📁 Step 3. Copy the .rpm file to your system (optional)**

You can install directly from USB or copy it first:

cp /mnt/usb/zsh.rpm /home/user/

### **⚙️ Step 4. Install the RPM package**

Now run:

sudo rpm -ivh zsh.rpm

**Options meaning:**

| **Option** | **Description** |
| --- | --- |
| -i | Install the package |
| -v | Verbose mode (shows details) |
| -h | Displays progress with hash marks (#) |

### **🧠 Step 5. If dependencies are missing**

rpm only installs one package and won’t auto-install dependencies.  
 If it shows an error like *“Failed dependencies…”*, use:

sudo dnf install /mnt/usb/zsh.rpm

👉 DNF can check the same directory for dependent .rpm files and install them together.

### **✅ Step 6. Verify installation**

Check that it’s installed:

rpm -qa | grep zsh

the command

rpm -qlp packagename.rpm

is used to **list the contents of an RPM package file** **before installing it**.

## **🧩 Explanation of each option**

| **Option** | **Meaning** |
| --- | --- |
| -q | Query mode (asks the RPM database or package for information) |
| -l | List files contained in the package |
| -p | Operate **on a package file (.rpm)** instead of an installed package |

So together:

rpm -qlp file.rpm → **List all files that will be installed** by this RPM package, even if it’s not yet installed.

## **🧠 Example**

rpm -qlp zsh.rpm

Output might look like:

/bin/zsh

/usr/share/man/man1/zsh.1.gz

/etc/zshrc

This means:

* /bin/zsh → main executable
* /etc/zshrc → configuration file
* /usr/share/man/man1/zsh.1.gz → manual page

## **🧾 Related Useful Commands**

| **Command** | **Description** |
| --- | --- |
| rpm -ql zsh | List files of an **installed** package |
| rpm -qi zsh | Show information (version, build date, summary) of an installed package |
| rpm -qpi zsh.rpm | Show info about an **uninstalled** RPM file |

**Command line Downloader (wget)**

## **What it does**

wget retrieves files from the **web** using **HTTP, HTTPS, or FTP**.  
 It can download:

* single files (e.g., .rpm, .deb, .iso, .zip)
* entire websites (recursive mode)
* large files in the background or resume interrupted downloads

It’s a **non-interactive** downloader—perfect for scripts or servers.

## **⚙️ Basic syntax**

wget [options] <URL>

### **🔹 Example**

wget https://mirror.centos.org/centos/9-stream/BaseOS/x86\_64/os/Packages/zsh.rpm

→ Downloads zsh.rpm to your current directory.

## **🧠 Common options**

| **Option** | **Description** |
| --- | --- |
| -c | Continue (resume) a partially downloaded file |
| -O <filename> | Save the file with a custom name |
| -P <directory> | Save into a specific folder |
| --limit-rate=200k | Limit download speed |
| -b | Run in the background (creates a log file) |
| -r | Recursive download (entire site or folder) |
| --no-check-certificate | Skip SSL certificate checks (for broken HTTPS) |
| --user=username --password=pass | Authenticate for protected downloads |

## **📥 Examples**

### **1️⃣ Download a single file**

wget https://example.com/file.rpm

### **2️⃣ Save it with another name**

wget -O myfile.rpm https://example.com/file.rpm

### **3️⃣ Resume an interrupted download**

wget -c https://example.com/bigfile.iso

## **RPM Installation & Dependency Handling**

### **1. Basic Installation Command**

rpm -i <package-name>.rpm

**Purpose:** Installs a new .rpm package on the system.

### **2. Installing Without Dependencies**

rpm -i --nodeps <package-name>.rpm

* **Meaning:** --nodeps = *ignore dependencies*
* **When to use:** ✅ Safe to use when missing files are **non-critical**, e.g. documentation files.  
   ❌ **Do not use** when the missing file is **critical**, e.g. shared library files (.so files).  
   Using --nodeps in such cases can cause the program to malfunction.

### **3. Handling Dependencies Manually**

If dependency errors appear:

1. **Check for the missing dependency** on the **DVD** (installation media).
2. If found, **copy that dependency RPM** file to your local directory.
3. Then **install the dependency first**, followed by the main package.

## **💾 Using DVD as a Local Repository**

* The **installation DVD** contains multiple RPM packages.
* To install offline:  
  1. **Copy** all required packages from the DVD to a **local folder** (e.g., /mnt/localrepo/).
  2. Use rpm -i commands from that folder.
* This avoids repeated DVD mounting during installations.

## **🔍 Checking Installed Package Information**

rpm -qi <package-name>

**Purpose:** Shows detailed info about an installed package  
 (e.g., version, release, installation date, vendor, and description).

## **📁 Finding Which Package Owns a File**

rpm -qf <full-file-path>

**Example:**

rpm -qf /usr/bin/rdesktop

**Output:** Shows the name of the RPM package that installed that file.  
 Useful for troubleshooting or identifying file origins.

YUM

# **End-to-End Guide: Networking, YUM/DNF, RPM, and Installing Docker**

## **1) Make sure networking works first**

There’s no package install without working DNS, gateway, and an IP.

### **A. Quick checks**

ip addr # See interfaces and IPs

ip route # See default gateway (look for "default via ...")

ping -c 2 8.8.8.8 # Test raw connectivity

ping -c 2 google.com # Test DNS + connectivity

cat /etc/resolv.conf # DNS servers in use

### **B. Configure/repair networking (RHEL/CentOS)**

* **RHEL 6**
  + Edit ifcfg files: /etc/sysconfig/network-scripts/ifcfg-<ifname>

Restart network:  
  
 service network restart

* **RHEL 7/8/9**

Prefer **NetworkManager**:  
  
 nmtui # Text UI (easy)

nmcli device status # See devices

nmcli connection show # See profiles

nmcli connection modify "<PROFILE>" ipv4.addresses "192.168.1.10/24" \

ipv4.gateway "192.168.1.1" \

ipv4.dns "8.8.8.8 1.1.1.1" \

ipv4.method manual

nmcli connection up "<PROFILE>"

Restart services (when needed):  
  
 systemctl restart NetworkManager

# (If using legacy network-scripts:)

systemctl restart network

**Remember**

* **DNS** lives in /etc/resolv.conf (though NetworkManager often writes it for you).
* Default **gateway** is visible in ip route.
* Use nmtui/nmcli or edit ifcfg-\* files; don’t mix methods at the same time.

## **2) How YUM/DNF really works (what your screenshot described)**

You run something like:  
  
 yum install dovecot # (RHEL/CentOS 6/7)

# or

dnf install dovecot # (RHEL/CentOS 8/9)

1. The package manager reads **repo definitions** from /etc/yum.repos.d/\*.repo.  
   * Key fields inside a .repo file:  
     + baseurl= or mirrorlist=
     + enabled=1
     + gpgcheck=1 and gpgkey=...
2. It resolves **DNS** and connects (HTTP/HTTPS/FTP) to the repo URL.
3. It downloads **metadata**, especially repodata/repomd.xml (this is the “inventory” that lists other metadata files and lets the manager resolve dependencies).
4. It computes the **dependency graph** for your request.
5. It downloads required **.rpm** files into the local cache (traditionally under /var/cache/yum/ for yum, /var/cache/dnf/ for dnf).
6. It performs a transaction via **rpm** (install/upgrade/remove) with order and scriptlets handled correctly.
7. Cache retention is controlled by settings (e.g., keepcache); RPMs aren’t *always* deleted—behavior depends on configuration.

**From the screenshot:** steps about loading .repo → contacting fastest mirror → reading repomd.xml → downloading RPMs → installing with rpm -ivh (internally) are accurate conceptually.

## **3) The core commands you’ll use**

### **A. Query what’s already installed**

rpm -qa | grep docker

rpm -qa | grep dovecot

### **B. Install via YUM/DNF (resolves dependencies automatically)**

yum install <pkg> # RHEL/CentOS 6/7

dnf install <pkg> # RHEL/CentOS 8/9

**Correction to a common myth** During yum install, the prompt is **Yes/No** (y/N). There isn’t a built-in single-letter **“d = just download”** choice.  
 If you want **download only**, use:

yum install --downloadonly --downloaddir=/path <pkg> # needs yum-plugin-downloadonly on older systems

# or

yumdownloader <pkg> # from yum-utils

# or (dnf):

dnf download <pkg>

### **C. Manual RPM install (not dependency-smart)**

rpm -ivh package.rpm # install

rpm -Uvh package.rpm # upgrade or install

rpm -e package\_name # erase

Prefer **yum localinstall package.rpm** or **dnf install package.rpm** so dependencies are resolved automatically.

### **D. Repo maintenance & troubleshooting**

yum repolist all

yum clean all

rm -rf /var/cache/yum/\*

dnf clean all

## **4) Installing Docker (with repos you already have)**

If your local repos contain docker (or you’ve set up the Docker CE repo), do:

rpm -qa | grep docker # see if any docker packages exist

yum install docker # RHEL/CentOS 6/7 (package name may be docker or docker-engine)

# or CE on newer systems (with Docker repo enabled):

# dnf install docker-ce docker-ce-cli containerd.io

Enable and start:

systemctl enable --now docker # RHEL 7/8/9

service docker start # RHEL 6

docker version

docker run hello-world

If yum install docker fails with “No package docker available”, your current repos don’t provide it. Either:

* Enable the official Docker CE repo, or
* Use a locally mirrored repo that contains Docker packages compatible with your OS version.

## **5) Understanding .repo files (the heart of YUM/DNF)**

Location: /etc/yum.repos.d/\*.repo

Minimal example:

[my-base]

name=My Local/Base Repo

baseurl=http://repo.myco.local/centos/$releasever/os/$basearch/

enabled=1

gpgcheck=1

gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-myco

* **baseurl vs mirrorlist**: Use one. mirrorlist points to a service that hands out mirrors.
* **repomd.xml**: The top metadata file yum/dnf fetch first; it references the rest of the metadata.
* **Dependencies** come from metadata (primary.xml, filelists.xml, etc.), which repomd.xml indexes.

## **6) Typical workflow (from zero to installed)**

1. **Fix networking** (IP, gateway, DNS) and confirm with ping/curl.

**Verify repos** exist and are enabled:  
  
 ls -1 /etc/yum.repos.d/

yum repolist

**Install the package**:  
  
 yum install <pkg> # or dnf install <pkg>

**If you only need the RPMs**:  
  
 yum install --downloadonly --downloaddir=/tmp/rpms <pkg>

# or

yumdownloader --resolve <pkg>

**If you already have an RPM file**:  
  
 yum localinstall /path/to/package.rpm # dependency-aware

**Verify**:  
  
 rpm -qa | grep <pkg>

which <pkg-binary>

<pkg-binary> --version

## **7) When “YUM won’t work” (triage checklist)**

* **DNS issues**
  + Check /etc/resolv.conf. Try dig repo-domain.com or nslookup.
* **Firewall/proxy**

Ensure outbound **80/443** allowed. If using a proxy, export:  
  
 export http\_proxy=http://user:pass@proxy:port

export https\_proxy=http://user:pass@proxy:port

* **Repo URL wrong or unreachable**
  + Open the baseurl in a browser or with curl -I http://... to verify.

**Stale/locked cache** yum clean all

rm -rf /var/cache/yum/\*

If a stuck process holds the lock:  
  
 ps aux | grep yum

sudo pkill yum # last resort; make sure no transaction is mid-flight

* **GPG key problems**
  + Import the correct key listed in the .repo file.
* **SELinux/SSL cert issues**
  + Check /var/log/yum.log, /var/log/messages, and errors on screen.

## **8) Quick reference (you mentioned these)**

* **Check routes**: ip r (or ip route)
* **Configure via**:  
  + **nmtui** (simple)
  + **nmcli** (scriptable)
  + **Edit ifcfg** files in /etc/sysconfig/network-scripts/ (legacy/static)
* **Key config files/places**:  
  + /etc/yum.repos.d/\*.repo → repo definitions
  + /etc/resolv.conf → DNS servers
  + /var/cache/yum/ or /var/cache/dnf/ → download cache
  + repodata/repomd.xml on the repo server → top metadata index

## **9) Small truth-checks vs your raw notes**

* ✔️ YUM reads .repo files, fetches repomd.xml, resolves dependencies, downloads RPMs, then installs using rpm transactions.
* ✔️ /var/cache/yum/ is used for downloads (dnf uses /var/cache/dnf/).
* ⚠️ **“y / d / n”** at prompt: YUM’s standard prompt is **y/N**. **“d = just download”** is **not** a built-in choice—use --downloadonly/yumdownloader.
* ⚠️ **“rpm -ivh once it wget”**: YUM/DNF don’t literally call the wget command; they use libcurl internally. The effect (download → install) is correct, but the mechanism is not a shell wget.
* ⚠️ **“It will remove the .rpm pkg”**: Cache behavior depends on settings. Don’t assume automatic deletion.

# **Package & Repo Guide (CentOS Stream 10/EL10)**

## **0) Know your tool on EL10**

* **DNF is the real package manager.**
  + yum exists as a wrapper, but the cache & behavior are **DNF**.
* **Cache paths:** /var/cache/dnf/... (not /var/cache/yum).

## **1) Check if a package exists in your enabled repos**

dnf repolist enabled # which repos are active

dnf search docker # search by name/summary

dnf provides '\*/dockerd' # search by file/provide

**Meaning:**

* If dnf search docker shows *No match*, your current repos don’t ship it.

On **CentOS Stream 10**, Docker Engine packages are **not** in default repos. You’ll typically see only podman and podman-docker.

## **2) If not found → enable extra repos (example: EPEL)**

**What is EPEL?** A repository hosted by Fedora with thousands of extra RPMs (not a single file).

### **Enable EPEL**

dnf install -y epel-release

* This installs **epel-release** (e.g., epel-release-latest-10.noarch.rpm), which simply drops repo files into /etc/yum.repos.d/ (even on EL10 it’s the same path).

### **Re-check after enabling**

dnf clean all && dnf makecache

dnf repolist enabled | grep -i epel

dnf search <pkg>

**Note:** Enabling EPEL won’t magically add Docker Engine on EL10; it just adds many community packages. Docker CE for EL10 is currently not published by Docker’s official repo. Use **Podman** on EL10 or switch to EL9 if you require Docker CE.

## **3) Install the package (when available)**

dnf install -y <package-name>

* DNF resolves dependencies automatically and downloads/install RPMs.

## **4) Understand DNF cache (why you don’t see RPMs yet)**

### **Metadata vs Packages**

* dnf makecache downloads **metadata only** (you’ll see files like appstream.solv, epel.solv, packages.db).
* **Actual .rpm files** appear **only after** you install or download a package.

### **Where RPMs show up**

/var/cache/dnf/<reponame-somehash>/packages/

* If you don’t see .rpm there, you haven’t downloaded any from that repo, or cache was cleaned.

### **Keep downloaded RPMs (optional)**

sudo sh -c 'printf "[main]\nkeepcache=True\n" > /etc/dnf/dnf.conf'

dnf clean all && dnf makecache

* Future installs will keep RPMs in /var/cache/dnf/.../packages/.

### **Prove it quickly**

# Download from EPEL without installing (saves to current directory):

dnf --enablerepo=epel download htop

# Or install (puts RPMs in cache if keepcache=True):

dnf --enablerepo=epel install -y htop

# Now see cached RPMs:

find /var/cache/dnf -name "\*.rpm"

## **5) Special case: Docker on CentOS Stream 10**

* Default repos on EL10 **don’t** provide Docker Engine (docker-ce).

Recommended path on EL10:  
  
 dnf -y install podman podman-docker

systemctl enable --now podman.socket

export DOCKER\_HOST=unix:///run/podman/podman.sock

docker run hello-world # works via Podman compatibility

* If you **must** have Docker CE, use an EL9 distro (Rocky/Alma/CS9) with Docker’s official repo. EL10 builds aren’t generally available yet.

## **6) Quick decision checklist**

1. **Search**: dnf search <pkg> → Not found?
2. **Enable extra repo** (e.g., dnf install epel-release).
3. **Refresh**: dnf clean all && dnf makecache.
4. **Search again**.
5. **Install** if found, or choose an **equivalent** (e.g., Podman on EL10).
6. **Want to see RPMs in cache?** Set keepcache=True and install/download.

## **7) Useful commands (one-liners)**

# See enabled repos

dnf repolist enabled

# Search by name/summary

dnf search <term>

# See which package provides a file

dnf provides '\*/binary-or-file'

# Enable EPEL

dnf install -y epel-release

# Clean & rebuild metadata

dnf clean all && dnf makecache

# Install

dnf install -y <pkg>

# Download only (to current dir)

dnf download <pkg>

### **Key reminders from your notes (clarified)**

* **EPEL is a repo, not a single RPM.** The RPM you install is **epel-release** (it just adds repo files).
* Metadata in /var/cache/dnf (like epel.solv) ≠ actual RPM packages.
* You’ll only see .rpm in cache **after** installing/downloading something from that repo.
* On EL10, use **Podman/podman-docker** for Docker-like workflows.

if there is firewall and can not go to internet

then we hva to build local repository how t install offline

first we need to add dvd then mount it

there is packages having rpm

we need to copy in any local mount point so we have to create partition or add new HDD

n live system we have enter disk discover command

echo “- - -” > /sys/class/scsi\_host/host0/scan

then make partition by fdisk

then partprobe

then format mkfs.xfs

then make dir in new partition then mount that dir with partition

the make it persistent in /etc/fstab

then take cp by

rsync -parv hdv location to new partition location

it will take back and it is now local repo

now we have to create repomd.xml file with create repo command

if create repo is not install then yum install create repo

then

we will create create repo in that location where rpm copied

createrepo -v location

once it done then it will also get dependencies……..

its done and now it is same as it is online

now we have to use it

first create a repo in yum.repos.d named local repo and put entries in it

(list entries and description what it do like gptcheck- it check integrity of file, in local we keep it 0)

now when we use yum install mysql

then it will go to yum.repos.d then go to local  
  
where he get base url which is url of local repo

here note file have protocol file:// like https://

if i need to specily get from local the command is

yum –enablerepo=local

YUM- Local Repo

# **Build & Use an Offline Local YUM/DNF Repository (from DVD/ISO → Local Disk)**

## **0) Scenario & Goal**

* **Scenario:** No internet due to firewall.
* **Goal:** Mirror RPMs from DVD/ISO to a local disk, generate metadata, and **add a repo file under /etc/yum.repos.d/** so yum/dnf installs work offline.

## **1) Get a Source of RPMs (DVD/ISO)**

### **1.1 Mount the DVD/ISO**

mkdir -p /mnt/dvd

# Physical DVD:

mount /dev/sr0 /mnt/dvd

# OR ISO file:

# mount -o loop /path/to/your.iso /mnt/dvd

**Why:** We need the **Packages/** folder from the media—this is the raw RPM source.

**Check:**

ls /mnt/dvd/Packages | head

## **2) Prepare Local Storage for the Repository**

### **2.1 (If adding a new disk) Discover disk on a live system**

# Rescan SCSI buses (run for each host that exists)

echo "- - -" > /sys/class/scsi\_host/host0/scan

echo "- - -" > /sys/class/scsi\_host/host1/scan # if present

echo "- - -" > /sys/class/scsi\_host/host2/scan # if present

lsblk

**Why:** Tells the kernel to detect newly attached disks without rebooting.

### **2.2 Partition the disk**

fdisk /dev/sdb # n → p → accept sizes → w

partprobe # notify kernel about new partition table

lsblk -f # confirm e.g., /dev/sdb1

**Why:** Create a slice on the disk to hold the repo.

### **2.3 Create filesystem, mount point, and mount it**

mkfs.xfs /dev/sdb1

mkdir -p /repo/local

mount /dev/sdb1 /repo/local

df -h /repo/local

**Why:** Formats storage and makes it accessible at a stable path.

### **2.4 Make the mount persistent across reboots**

blkid /dev/sdb1

# Add to /etc/fstab using UUID to avoid device-name changes:

echo "UUID=$(blkid -s UUID -o value /dev/sdb1) /repo/local xfs defaults 0 0" >> /etc/fstab

umount /repo/local

mount -a

df -h /repo/local

**Why:** Ensures the repo directory is always available after reboots.

## **3) Copy RPMs from DVD/ISO to Local Disk**

### **3.1 Copy packages**

rsync -parv /mnt/dvd/Packages/ /repo/local/Packages/

# (Many admins like -avh too: rsync -avh /mnt/dvd/Packages/ /repo/local/Packages/)

**Why:** Brings the entire RPM set local, so installs won’t need the DVD in the drive.

**What it does:** Copies **all RPMs** while preserving attributes (-p perms, -a archive, -r recursive, -v verbose).

## **4) Create Repo Metadata (repodata/repomd.xml)**

### **4.1 Ensure createrepo exists (or createrepo\_c)**

With no internet, install it **from the DVD** if it’s available:

rpm -q createrepo createrepo\_c || true

# If missing, search on DVD (paths vary by distro version):

# Example:

# rpm -ivh /mnt/dvd/Packages/createrepo-\*.rpm || rpm -ivh /mnt/dvd/Packages/createrepo\_c-\*.rpm

**Why:** createrepo generates the **repodata/** folder and **repomd.xml** index that YUM/DNF reads.

### **4.2 Build metadata in the repo root**

createrepo -v /repo/local

# or:

createrepo\_c -v /repo/local

ls /repo/local/repodata

**Why:** Without metadata, yum/dnf can’t “see” what packages exist in your folder.

**Note:** createrepo only indexes; it doesn’t magically add dependencies. Dependency resolution happens during install; you must have all needed RPMs present (or add more sources, like the DVD as another repo).

## **5) Tell YUM/DNF About Your Local Repo (your key step)**

### **5.1 Create a repo file inside /etc/yum.repos.d/**

Create **/etc/yum.repos.d/local.repo**:

[local]

name=Local Offline Repo

baseurl=file:///repo/local

enabled=1

gpgcheck=0

**Why we are doing this & what each entry does:**

* **[local]** → The **repo ID**. Used with --enablerepo=local.
* **name=** → Human-readable label shown in repolist.
* **baseurl=** → Where packages & metadata live.  
  + file:///repo/local means “use the local filesystem path /repo/local”.
  + (This is like https://… but for files; **your note is right**: it uses file:// protocol.)
* **enabled=** → 1 means the repo is active by default. Set to 0 to disable.
* **gpgcheck=** → 0 skips signature checking (handy offline if keys aren’t set).  
  + If you **do** want verification: set gpgcheck=1 and add gpgkey=file:///path/to/RPM-GPG-KEY.
* (Optional) **metadata\_expire=** → How long cached metadata is valid (e.g., 6h).
* (Optional) **priority=** (with yum-plugin-priorities) → Prefer one repo over others.

You wrote “gptcheck”—the correct key is **gpgcheck** (verifies package signatures).

### **5.2 (Optional) Add the DVD itself as a repo (without copying)**

Create **/etc/yum.repos.d/dvd.repo**:

[dvd]

name=Install DVD

baseurl=file:///mnt/dvd

enabled=0

gpgcheck=0

**Why:** Allows you to **enable it on demand** to satisfy missing deps directly from the disc:

dnf --enablerepo=dvd makecache # or yum --enablerepo=dvd makecache

## **6) Clean Cache & Load the New Repo**

dnf clean all || yum clean all

dnf makecache || yum makecache

dnf repolist all || yum repolist all

**Why:** Forces the package manager to read **repodata/repomd.xml** from your new repo and show it in the list.

## **7) Use the Local Repo**

### **7.1 Install from only the local repo (great for testing)**

# DNF:

dnf --disablerepo="\*" --enablerepo="local" install mysql

# YUM:

yum --disablerepo="\*" --enablerepo="local" install mysql

**Why:** Guarantees the package comes from **your local repo**.

Package names vary by distro (e.g., mysql, mysql-server, or mariadb-server). Search:

dnf --enablerepo=local search mysql

# or

yum --enablerepo=local search mysql

## **8) Maintain & Update Your Local Repo**

**When you add/remove RPMs** under /repo/local/Packages/, **update metadata**:  
  
 createrepo --update /repo/local

* **Why:** Regenerates indexes so yum/dnf sees changes.

**See which repo a package would come from:** dnf info <pkg> | grep -i repo # or yum info <pkg> | grep -i repo

* **Troubleshooting:**
  + Missing deps → add more RPMs or temporarily enable the dvd repo.
  + mount -a error → fix /etc/fstab (UUID, fs type, or mount path).
  + Can’t see new disk → re-run the echo "- - -" rescan on **all** scsi\_hosts you have.

# **One-Page “Do & Why” Checklist (Fast Recall)**

1. **Mount DVD/ISO** **Do:** mount /dev/sr0 /mnt/dvd (or loop-mount ISO)  
    **Why:** Source of RPMs in /mnt/dvd/Packages.
2. **Discover & prep disk** *(if using a new HDD)* **Do:** echo "- - -" > /sys/class/scsi\_host/hostX/scan → fdisk → partprobe  
    **Why:** Make the new disk/partition visible to the OS.
3. **Format & mount repo location** **Do:** mkfs.xfs /dev/sdb1 → mkdir -p /repo/local → mount /dev/sdb1 /repo/local  
    **Why:** Create a stable place to store packages and metadata.
4. **Persist mount** **Do:** Add UUID line to /etc/fstab, then mount -a  
    **Why:** Repo is available after reboots.
5. **Copy RPMs** **Do:** rsync -parv /mnt/dvd/Packages/ /repo/local/Packages/  
    **Why:** Local mirror of packages—no DVD needed later.
6. **Generate metadata** **Do:** createrepo -v /repo/local  
    **Why:** Creates repodata/repomd.xml so YUM/DNF can use your repo.
7. **Create the repo file (your key step)** **Do:** **/etc/yum.repos.d/local.repo** with:

[local]

name=Local Offline Repo

baseurl=file:///repo/local

enabled=1

gpgcheck=0

**Why:** This is how yum/dnf learns about your local repo.  
 *(Use file:// protocol; gpgcheck=0 if you don’t have keys offline.)*

1. **Refresh cache & verify** **Do:** dnf/yum clean all && dnf/yum makecache && repolist  
    **Why:** Loads your new repo into the resolver.
2. **Install from local** **Do:** dnf/yum --disablerepo="\*" --enablerepo=local install <pkg>  
    **Why:** Ensures installs use your offline repo only.

## **Final Summary**

* You mounted the **DVD/ISO** → prepared **local storage** → **copied RPMs** → ran **createrepo** to generate repodata/repomd.xml.
* You **created a repo file inside /etc/yum.repos.d/** (local.repo) with **baseurl=file:///repo/local**, enabled=1, gpgcheck=0.
* You **cleaned caches** and **installed packages** using --enablerepo=local.
* Every step includes **why** you’re doing it, so when you review later, you’ll recall the concept behind the command—not just the syntax.

Int Prep-Prc Monitering

# **Interview Preparation Notes: Linux Server Troubleshooting & Performance Diagnosis**

## **Common Interview Questions**

1. **What are your daily activities?**
2. **If a server is in high utilization, how do you diagnose it?** → Follow systematic diagnosis steps (explained below)
3. **What critical issue did you face in the last month?**

# **Server High Utilization Diagnosis – Case-Based Approach**

## **🧩 Case 1 – High CPU and Memory Utilization (Application-Level Load)**

### **1️⃣ Initial Check**

Command:  
  
 top

* **CPU Section**
  + id (idle value) = 0 → means 100% CPU load
  + id = 99 or 100 → means no load
* **Memory Section**
  + Check **used**, **free**, and **available** memory.
* **Swap Section**
  + If swap usage is full, system is under memory pressure.

### **2️⃣ Cross-check Memory with:**

free -m

### **3️⃣ Identify the Culprit Process**

* In top, check the **COMMAND** column for any process consuming high CPU or memory.  
   Example: java consuming 100% memory.

### **4️⃣ Action**

* Engage the **application team** if a process (e.g., Java, Python) is overutilizing resources.
* Root cause: Application tuning issue, not hardware or OS.

🧠 **Summary for Case 1:** If one process is consuming most CPU/memory → Engage app team for tuning (e.g., JVM, threads, garbage collection).

## **🧩 Case 2 – High System Load but No Process Using High CPU**

Sometimes load average is high but no single process seems responsible.

### **1️⃣ Check Disk Utilization**

Commands:  
  
 df -h # Check disk space

iostat -x # Check detailed disk I/O utilization

* If a partition is full → clean old files/logs.
* If high I/O utilization → investigate disk or RAID performance.
* Tool: iometer (if installed) → monitors I/O in detail.
* If issue persists → escalate to **hardware vendor** to check RAID or disk faults.

### **2️⃣ Check Network Utilization**

* Tools:  
  + tcpdump – Packet capture and inspection.
  + iperf – Bandwidth and latency test.
  + nmap – Network scanning and connection check.
  + wireshark – Graphical packet analysis.
  + sar – Combined report of CPU, I/O, and network.

🧠 **Summary for Case 2:** If CPU/memory look fine but load remains high → check **disk I/O first**, then **network load**.

## **🧩 Case 3 – All Resources Appear Normal but System Still Slow**

This often points to **memory leakage** or excessive **cache/buffer** usage.

### **1️⃣ Check Buffer and Cache**

* **Buffer:** Preloaded data in RAM (e.g., YouTube “grey loading”).
* **Cache:** Temporary data stored in swap or memory before being written to disk.

### **2️⃣ Reason for Growth**

* Poorly designed application → starts **child processes** that don’t terminate after parent closes.
* Leads to **memory leakage**.

### **3️⃣ Detection**

Tool:  
  
 valgrind

* → Detects memory leaks in applications.

### **4️⃣ Fix / Clean Cache**

Command (run only with team approval):  
  
 sync; echo 3 > /proc/sys/vm/drop\_caches

🧠 **Summary for Case 3:** If everything looks fine (CPU, memory, disk, network) → check for **buffer/cache overflow** and **memory leaks**.

## **🧩 Case 4 – All Above Areas Clear but Server Still Unstable**

Likely a **hardware problem**.

### **1️⃣ Check Hardware Health**

**RAM Test:** memtest

* **CPU Health:**
  + i7z → Check CPU frequency and overclocking.  
    Use CPU temperature monitoring tools.

### **2️⃣ Possible Causes**

* Overheating CPU
* Faulty RAM module
* Disk controller or RAID hardware issue

🧠 **Summary for Case 4:** If OS and applications are clean → suspect **hardware fault** and perform physical diagnostics.

# **🔧 Performance Monitoring Tools (Summary)**

| **Tool** | **Description** |
| --- | --- |
| top | Real-time CPU, memory, swap, process monitoring |
| htop | Interactive version of top |
| ps aux | Shows all running processes |
| iostat | Disk I/O statistics |
| sar | Historical performance reports (CPU, I/O, network) |
| vmstat | Monitors memory, CPU, and swap activity |
| pidstat | Per-process live statistics |
| mpstat | CPU usage per core |
| strace | Tracks system calls made by a process |
| ipcs | Shared memory/semaphore stats |
| lsof | Lists open files by process |
| tcpdump | Network packet sniffer |
| iometer | Advanced disk I/O load analyzer |

# **🧾 Common Commands Explained**

### **Iostat**

iostat -x -d -p | less

iostat -x -d -p /dev/sda 2 3

→ Shows extended disk stats every 2 seconds (3 outputs total).

### **Pidstat**

Monitors live performance of each process:

pidstat

### **Vmstat**

Shows CPU + memory + swap + buffer/cache:

vmstat 2

### **Sar**

Collects and displays historical system activity.

sar -u 2 # CPU stats every 2 sec

sar -d 2 # Disk I/O stats

cd /var/log/sa

sar -f saXX # View stored reports

### **File and Process Checks**

pidof ssh # Find PID

pmap -d <pid> # Show memory maps for that PID

strace ls # See system calls made by 'ls'

# **🔁 Quick Summary – Step-by-Step Flow**

| **Step** | **Focus Area** | **Tools / Commands** | **Purpose** |
| --- | --- | --- | --- |
| 1 | CPU, Memory, Swap | top, free -m | Identify resource-heavy process |
| 2 | Disk Utilization | df -h, iostat | Clean or diagnose storage bottleneck |
| 3 | Network Load | tcpdump, iperf, sar | Check for high network activity |
| 4 | Cache & Buffer | vmstat, drop\_caches | Detect & clear memory leaks |
| 5 | Hardware Health | memtest, i7z | Identify faulty hardware |
| 6 | Application | valgrind | Debug memory leakage issues |

FSH-Var file/ logs

## **Linux /var/Log Files Overview**

## **What Are Log Files?**

* Logs are **records of daily activities** and events happening in the system.
* They help in **monitoring, troubleshooting, and auditing** system behavior.
* All main logs are stored in the **/var/log/** directory.

## **📁 2. Important Log Files in /var/log/**

| **Log File** | **Purpose / Contains** | **Key Commands** |
| --- | --- | --- |
| **/var/log/messages** | General system activity – boot info, kernel, network, and system events | head, tail, tail -f, less, more |
| **/var/log/secure** | **Authentication-related logs** – login attempts, sudo, SSH, etc. | cat /var/log/secure, grep ssh /var/log/secure |
| **/var/log/dmesg** | Boot-time hardware detection messages saved from kernel ring buffer | cat /var/log/dmesg |
| **/var/log/boot.log** | System boot messages (startup services) | cat /var/log/boot.log |
| **/var/log/cron** | Cron job execution details | cat /var/log/cron |
| **/var/log/maillog** | Mail server activity | cat /var/log/maillog |

💡 **Tip:**

* /var/log/secure is critical for **security auditing** (tracks successful/failed logins).
* /var/log/messages is best for **general troubleshooting**.

## **🧩 3. Useful Commands to View Logs**

| **Command** | **Description** |
| --- | --- |
| head -20 /var/log/messages | Shows the first 20 lines of the log file |
| tail -20 /var/log/messages | Shows the last 20 lines |
| tail -f /var/log/messages | Monitors the file **live** as new entries appear |
| less / more | Used to scroll through long logs |
| grep -i error /var/log/messages | Filters logs by keyword, e.g., “error” |
| grep "Failed" /var/log/secure | Find failed SSH or sudo attempts |

## **⚙️ 4. Hardware Logs and System Info**

### **🔹 dmesg**

* Shows **hardware detection messages** from the kernel.
* Reads directly from the **kernel ring buffer** (in **RAM**, not a file).

Common use:  
  
 dmesg | less

* On some systems, saved copy: /var/log/dmesg

🧠 **Remember:** dmesg = temporary live kernel hardware log (RAM)  
 /var/log/dmesg = saved version at boot time

### **🔹 dmidecode**

Displays **full hardware information** (motherboard, BIOS, CPU, RAM, etc.) directly from BIOS.  
  
 dmidecode | less

* More detailed than dmesg.

## **🔄 5. Log Rotation**

### **🔹 What Is Log Rotation?**

* **Older log files** are automatically backed up and replaced by new ones to save disk space.
* These backup files are called **rotated logs**.

📘 Example:

/var/log/messages

/var/log/messages.1

/var/log/messages.2.gz

/var/log/messages.3.gz

/var/log/messages.4.gz

→ When 5th rotation happens, the **oldest one is deleted**.

### **🔹 Default Rotation Policy**

* By default: **4 rotated files** are kept.
* The **5th one** is deleted.
* You can **change this policy**.

## **⚙️ 6. Log Rotation Configuration**

### **🔹 Configuration Files**

* **Main configuration:** /etc/logrotate.conf
* **Service-specific configurations:** /etc/logrotate.d/

### **🔹 Check Logrotate Directory**

ls -d /etc/logrotate.d

This directory contains rotation rules for specific services (e.g., syslog, cron, yum, httpd, etc.)

### **🔹 Logrotate Service**

* Rotation is handled by **logrotate service**.

It runs automatically via **cron jobs** or can be executed manually:  
  
 logrotate /etc/logrotate.conf

### **🔹 Policy Control Options**

In config files, you can set:

rotate 4 # number of old logs to keep

daily/weekly/monthly # rotation frequency

compress # compress old logs

missingok # skip missing files without error

create # create new log after rotation

FSH- Other files

# **Linux Important System Directories – Final Notes**

## **📂 1. /mnt (Mount Directory)**

* Optional directory used for **temporary mounting** of external storage such as USB drives, ISO files, or network shares.
* Mainly used by administrators for short-term mounting.
* After unmounting, /mnt becomes empty again.

**Example commands:** mount /dev/sdb1 /mnt

umount /mnt

## **⚙️ 2. /tmp (Temporary Directory)**

* Stores **temporary files** created by applications during use.
* Applications manage their own temp files; some are deleted automatically, others remain depending on app design.
* Most files are cleared automatically **after reboot**.
* /tmp appears **green** because it has the **sticky bit** permission.  
  + Sticky bit (t) means everyone can create files, but only the **owner or root** can delete them.

**Check permission:** ls -ld /tmp

* Output → drwxrwxrwt (the final **t** shows sticky bit).

## **🧩 3. /lost+found (Recovery Directory)**

* Used for **data recovery** by the filesystem.
* When the system runs a filesystem check (fsck) after an error or crash, **lost or orphaned files** are placed here.
* Every partition (like /, /home, /boot) has its own lost+found folder.

## **🔥 4. /boot (Boot Directory)**

* Contains all **boot-related files** required to start Linux.
* **Main contents:**
  + **vmlinuz** → The **kernel file** (compiled in binary, written in **C**, and **compressed** in bzip/gzip).
  + **grub/** → Contains the **GRUB bootloader** configuration and files.

**Commands:** du -sh /boot # Check total size

du -ah | sort -n -r | head -n 20 # Show top 20 largest files

* **du** stands for **disk usage**.

## **🧠 5. /dev (Device Directory)**

* Contains **device files** that act as interfaces between hardware and the **kernel**.
* These are **special files**, not normal data files.
* **Types of files:**
  + **Character (c)** → Devices that send/receive data **byte by byte**.
  + **Block (b)** → Devices that transfer data **in blocks**.
  + **Socket (s)** → Used for process communication.

**Filter devices:** ls -l /dev | grep ^c # Character devices

* **Examples:**
  + lp0 → Printer driver file (special executable file in RAM).
  + sda → Hard-disk driver file.
  + **/dev/loop0** → is just a virtual “disk” interface that points to a **file** (your Parrot ISO).
  + When you mount -o loop,ro parrot.iso /mnt/parrot, the kernel (your **CentOS** kernel) treats that file like a read-only disk and exposes its filesystem so you can **browse the files** inside the ISO.
  + **/dev/loop0 = pretend disk.** It lets Linux treat a **file** (like a ParrotOS ISO) as if it were a real disk.
  + When you **mount** the Parrot ISO on CentOS, you’re only **opening the ISO to read its files**.
  + Your server still **runs CentOS’s kernel**. Nothing from Parrot gets installed or started just by mounting.
  + Think of it like **unzipping and browsing** a ParrotOS DVD—you're **looking**, not **booting** it.
  + **/dev/null** is a special “black hole” file.
  + **Write to it** → data is instantly discarded.
  + **Read from it** → you immediately get **EOF** (as if the file is empty).

### **Related (don’t confuse)**

### **/dev/zero**: reading gives infinite **zero bytes**.

### **/dev/random** / **/dev/urandom**: reading gives random bytes.

* These are **driver files** controlled by the **kernel** — just like how fdisk uses /dev/sda to access the hard drive.

## **🧾 6. Kernel Control**

* The **kernel controls all hardware** through these special files in /dev.
* Each file (like lp0 or sda) acts as a communication point between hardware and user space.
* Without these driver files, tools like fdisk or printers cannot function.

## **🧠 Quick Summary Table**

| **Directory** | **Purpose** | **Key Points** |
| --- | --- | --- |
| /mnt | Temporary mount point | Used for manual short-term mounting |
| /tmp | Temporary files | Sticky bit permission, cleared on reboot |
| /lost+found | Data recovery | Used by fsck after crash |
| /boot | Boot files | Contains kernel (vmlinuz) and GRUB |
| /dev | Device interface files | Includes drivers like lp0, sda |

## **📚 Important Commands**

| **Command** | **Function** |
| --- | --- |
| du -sh | Show total directory size |
| `du -ah | sort -n -r |
| `ls -l | grep ^c` |
| mount /dev/sdb1 /mnt | Mount external drive |
| umount /mnt | Unmount drive |

## **🧩 Key Keywords for Quick Recall**

* **Mount point** → /mnt
* **Sticky bit** → /tmp
* **Recovered data** → /lost+found
* **Kernel file (vmlinuz)** → /boot
* **Driver files (lp0, sda)** → /dev

NOte

we can not create separate partition other than root

like /dev. /etc, /bin, /sbin, /lib

contian critical file which should always available to OS to run

Tape Device mnge

# **🎞️ Tape Device Management in Linux (/dev/nst0)**

## **🧩 1. Overview**

When a **tape device (LTO – Linear Tape-Open)** is connected to a Linux server, it is detected as:  
  
 /dev/nst0

* + n → **non-rewinding** tape device.
  + st0 → first SCSI tape device.
* **Tape drives** use physical **cassette cartridges** to store large amounts of compressed data.
* Usually connected via:  
  + **SCSI interface**
  + **Fibre Channel**
  + **SAS** or **network-attached interface** (through a storage server)

## **💾 2. Key Features**

* Tape devices are **sequential access** storage (unlike HDDs).
* Designed for **large, long-term backups**.
* Data is **compressed automatically** by the drive’s hardware.
* No need to manually format the tape — it manages its own structure and headers.

## **🧱 3. Backup and Restore Commands**

### **🟢 To Take a Backup**

Use the tar command to write data to the tape:

tar -cvf /dev/nst0 /home/\*

* c → Create archive
* v → Verbose (show files)
* f → File (device path /dev/nst0)

✅ This creates a **full backup** of /home onto the tape.

### **🔵 To Restore Backup**

Use the same device path to extract:

tar -xvf /dev/nst0 /\*

* x → Extract files from tape
* v → Show details
* f → Read from tape device

✅ This restores data (or full OS backup) from the tape.

## **⚙️ 4. Managing the Tape Device**

Tape drives are managed using the **mt (magnetic tape)** command.  
 It controls tape movement, status, and erase operations.

### **🧭 Common mt Commands**

| **Purpose** | **Command** | **Description** |
| --- | --- | --- |
| **Forward tape** | mt -f /dev/nst0 forward | Moves the tape forward |
| **Rewind tape** | mt -f /dev/nst0 rewind | Brings tape back to start |
| **Check status** | mt -f /dev/nst0 status | Displays tape drive status (position, errors, etc.) |
| **Eject tape** | mt -f /dev/nst0 eject | Physically ejects the cassette |
| **Erase tape** | mt -f /dev/nst0 erase | Erases entire tape contents |

✅ Tip: Some systems use fsf, bsf, rewind, offline, etc., as motion options:

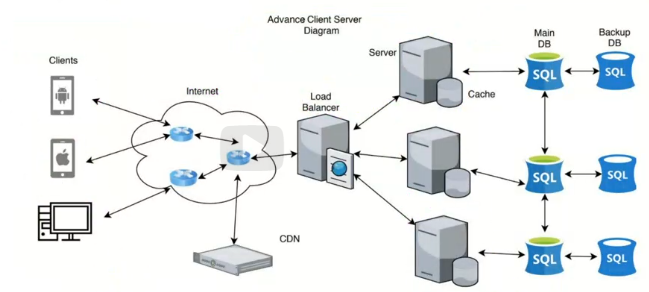
mt -f /dev/nst0 fsf 1 # Move forward one file

mt -f /dev/nst0 rewind # Rewind to beginning

## **🧾 5. Notes**

* /dev/st0 → **Rewinding device** (automatically rewinds after each operation).
* /dev/nst0 → **Non-rewinding device** (keeps position after write/read).
* mt manages **headers and tape movement**, while tar handles the **backup content**.
* Always rewind (mt -f /dev/nst0 rewind) before removing the tape to avoid read/write errors next time.

Interview



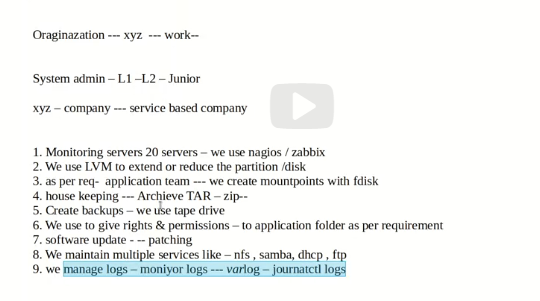
nagious and zabious are monitoring ttol

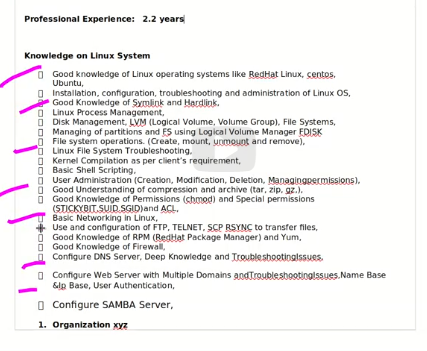
it is dashboard which monitor server it is configured with all servers

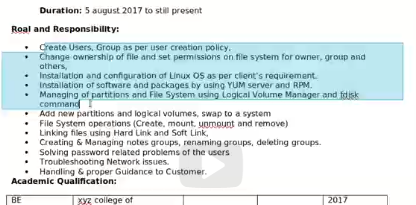
if nay issue find it generate email

then proactive action taken

and ticket generated for seniors







100 percent profile at first

update time to time as it go up

LVM

**Summary**

1. Add & discover new HDD
2. ls /dev/sdc → confirm disk is visible
3. df -h → verify no /dev/sdc partitions mounted
4. fdisk /dev/sdc → create **primary** partition **1**
5. In fdisk: press t → set **ID = 8e (Linux LVM)**
6. partprobe -s /dev/sdc → notify kernel of partition table changes
7. **Do not format** /dev/sdc1
8. LVM stack reminder: **PV → VG → LV**
9. pvcreate /dev/sdc1
10. vgcreate <vgname> /dev/sdc1
11. lvcreate -L 300M -n <lvname> <vgname>
12. ls /dev/<vgname>/<lvname> → confirm new LV device
13. mkfs.ext4 /dev/<vgname>/<lvname> → **format the LV (not /dev/sdc1)**
14. mkdir -p /<new\_dir> && mount /dev/<vgname>/<lvname> /<new\_dir>
15. df -Th → verify mount & filesystem type
16. vi /etc/fstab → add:  
     /dev/<vgname>/<lvname> /<new\_dir> ext4 defaults 0 0

**LVM = Logical Volume Manager.**

It’s a Linux storage layer that sits between physical disks/partitions and the filesystems you mount. It lets you pool space and carve it up flexibly.

### **Core idea**

* Instead of tying a filesystem directly to a single partition, you:  
  1. Make a **Physical Volume (PV)** on a disk/partition (e.g., /dev/sdc1).
  2. Combine one or more PVs into a **Volume Group (VG)** (a shared storage pool).
  3. Allocate chunks from the VG into **Logical Volumes (LVs)** (these behave like “virtual partitions” you format and mount).

### **Why use it**

* **Flexible resizing:** grow/shrink LVs (and filesystems, if supported) without repartitioning.
* **Add space easily:** add another disk as a new PV → extend the VG → extend an LV.
* **Snapshots:** create point-in-time copies of an LV for backup/testing.
* **Better layout:** split workloads across disks, mirror/stripe (with LVM RAID features).

### **Mental model**

[ Disks/Partitions ] -> PVs -> [ VG (pool) ] -> LVs -> Filesystems (/home, /var, ...)

# 

# 

# 

# **Migrate /home to a New LVM Logical Volume (when the current /home is non-LVM)**

## **Scenario & Goal**

* **Scenario:** Current /home lives on a non-LVM partition and the partition is full.
* **Goal:** Create an LVM stack on a new disk/partition and **move /home** onto the new LVM logical volume.

## **Key Idea (Why we’re doing each step)**

* We can’t easily grow a **non-LVM** /home.
* So we **add a new disk**, make an **LVM** stack (PV → VG → LV), **format the LV**, **mount it**, and **migrate /home** there.

## **Assumptions (your example values)**

* New HDD = **~500 MB** (device shows up as **/dev/sdc**).
* We create **one partition of ~300 MB**: **/dev/sdc1**.
* Partition **type/ID = 8e** (Linux LVM).
* LVM layers (concept recall):  
  + **PV (Physical Volume):** the physical layer (disk/partition).  
      
      
     Concept: you noted PV ≈ 500 MB overall; in this scenario we’re actually initializing **/dev/sdc1 (~300 MB)** for LVM.
  + **VG (Volume Group):** pools PV space (≈ the PV size used).
  + **LV (Logical Volume):** slice carved from the VG; **LV = ~300 MB** for /home.

## **Step-by-Step (What to do)**

### **1) Add the new HDD (do not format the whole disk)**

* Ensure the OS sees the disk as **/dev/sdc**.

### **2) Create an LVM-type partition**

* Use **fdisk /dev/sdc** to create **/dev/sdc1** (size **~300 MB**).
* Set **partition type = 8e (Linux LVM)**.
* Write changes and re-read the table (e.g., partprobe).

### **3) Build the LVM stack**

* **PV:** initialize the partition as a Physical Volume → **/dev/sdc1**.
* **VG:** create a Volume Group using that PV (VG size ≈ PV size used).
* **LV:** create a Logical Volume of **~300 MB** for /home.

**Shape to remember (three layers):** **PV** (physical) → **VG** (pool) → **LV** (usable slice)

### **4) Format the LV (not the partition)**

* **Important recall:** *You format the LV device, not /dev/sdc1.*
* The formatted device looks like: **/dev/<vg>/<lv>** (e.g., /dev/vg/home\_lv).

### **5) Create a mount directory and mount the LV**

* Make a directory (e.g., a temporary mount point), then **mount** **/dev/<vg>/<lv>** there.

### **6) Migrate /home to the new LV**

* Copy/move the /home data onto the mounted LV (this is the migration step).
* After verifying the data, mount this LV **as /home**.

## **Very Important Recall Points (from your notes)**

* **Device you format:** **/dev/<vg>/<lv>** **Do not** format **/dev/sdc1**.
* **New device path after LVM:** **/dev/<vg>/<lv>** (example form).
* **Mounting:** create a new directory, then **mount that directory to the LV**.

## **“Where do we add the new driver /dev/<vg>/<lv> and the new mounted name?”**

* You **mount** the LV device **/dev/<vg>/<lv>** **on your chosen mount point** (the “new mounted name”—for example, /home once you switch over).
* (If you want it to persist across reboots, you add the device **and its mount point** to your system’s mounts configuration. Keep this line in mind conceptually: **device = /dev/<vg>/<lv> → mount point = /home**.)

## **One-Screen Summary**

* **Problem:** /home on non-LVM is full.
* **Fix flow:** Add disk → make **/dev/sdc1** (type **8e**) → **PV** → **VG** → **LV (300 MB)** → **format LV** (not the partition) → **mount LV** → **migrate /home** → **use LV as /home**.
* **Always remember:** Work with **/dev/<vg>/<lv>** for formatting and mounting.

LVM-2

## **🧠 Correct Mental Model of LVM**

| **Layer** | **Meaning** | **Example** | **Purpose** |
| --- | --- | --- | --- |
| **PV (Physical Volume)** | The actual storage partition or disk initialized for LVM | /dev/sda1 | The physical base layer |
| **VG (Volume Group)** | A storage pool made from one or more PVs | vg0 | Combines multiple PVs into one large storage space |
| **LV (Logical Volume)** | A virtual partition created inside the VG | /dev/vg0/lv1 | The “drive” you format and mount for OS/apps |

🔹 Example mental model:  
 If you create a PV from /dev/sda1 (500 MB):

vgcreate vg0 /dev/sda1

→ VG = 500 MB total.  
 Then create an LV (e.g., 100 MB). You can later extend it up to available VG free space.

## **⚙️ Step-by-Step: Creating LVM from Scratch**

### **1. Identify and Prepare the New Disk**

lsblk

fdisk -l

➡ Lists all attached disks and partitions.

### **2. Create a Partition**

fdisk /dev/sdc

Inside fdisk:

* Create **primary partition** (e.g., /dev/sdc1)
* Press t to **change ID** → set it to **8e** (Linux LVM)
* Write changes and exit.

Then notify kernel:

partprobe

### **3. Create the LVM Layers**

#### **a. Create Physical Volume (PV)**

pvcreate /dev/sdc1

Initializes the partition for LVM.

#### **b. Create Volume Group (VG)**

vgcreate sameer /dev/sdc1

Creates VG named sameer using the PV.

#### **c. Create Logical Volume (LV)**

lvcreate -L 300M -n lv1 sameer

Creates an LV (lv1) of 300 MB inside VG sameer.

### **4. Format and Mount LV**

mkfs.xfs /dev/sameer/lv1 # XFS for RHEL 7/8

mkdir /data

mount /dev/sameer/lv1 /data

Add to /etc/fstab for auto-mount if needed.

## **🔍 Checking and Managing LVM**

| **Purpose** | **Command** | **Description** |
| --- | --- | --- |
| List all detected LVM devices | lvmdiskscan | Scans system for LVM-capable partitions |
| Show Physical Volumes | pvdisplay | Details about PVs |
| Show Volume Groups | vgdisplay | Shows VG size, free space, etc. |
| Show Logical Volumes | lvdisplay | Shows LV info |
| Simplified summary | pvs, vgs, lvs | Compact table view |
| Scan for volumes | pvscan, vgscan, lvscan | Detects and lists all volumes |
| Show devices with LVs | lvs -o +devices | Shows LVs with mapped devices |

## **🔁 Correct Order of Operations**

### **Creation Order:**

PV → VG → LV

### **Deletion Order:**

umount → LV → VG → PV

#### **Example:**

umount /data

lvremove /dev/sameer/lv1

vgremove sameer

pvremove /dev/sdc1

Then clean /etc/fstab if the entry exists.

## **📈 Extending LVM**

### **Case 1: VG Has Free Space**

lvextend -L +50M /dev/sameer/lv1

or use all free space:

lvextend -l +100%FREE /dev/sameer/lv1

Then extend filesystem:

For **ext4**:  
  
 resize2fs /dev/sameer/lv1

For **XFS** (RHEL 7/8):  
  
 xfs\_growfs /dev/sameer/lv1

### **Case 2: VG Is Full → Add New Partition or Disk**

#### **a. Make new partition (e.g., /dev/sda2)**

pvcreate /dev/sda2

#### **b. Add it to existing VG**

vgextend sameer /dev/sda2

#### **c. Extend LV again**

lvextend -l +100%FREE -r /dev/sameer/lv1

(-r auto-resizes filesystem)

## **Option 2 — Without -r (Manual Resize)**

If you forget or skip -r, then you must manually resize the filesystem after extending the LV.

### **Step 1: Extend only the Logical Volume**

lvextend -l +100%FREE /dev/sameer/lv1

At this point:

* The LV (logical volume) has grown.
* But the **filesystem inside it is still the old size.**

### **Step 2: Resize the Filesystem Manually**

#### **If your filesystem is ext4:**

resize2fs /dev/sameer/lv1

#### **If your filesystem is XFS (like RHEL 7/8):**

xfs\_growfs /dev/sameer/lv1

💡 **Tip:** You must keep the LV *mounted* while using xfs\_growfs,  
 but for resize2fs (ext4) it can be run on unmounted or mounted (preferably unmounted for safety).

### **Case 3: Add New Hard Disk (if no partitions left)**

* Add new HDD → /dev/sdb
* Partition → /dev/sdb1
* Create PV → pvcreate /dev/sdb1
* Add to existing VG → vgextend sameer /dev/sdb1

Now your VG size = sum of all PVs (old + new).

## **📉 Reducing LV (only for ext4)**

⚠️ Not possible for **XFS** — XFS supports only extension, not reduction.

### **Steps:**

umount /data

e2fsck -f /dev/sameer/lv1

resize2fs /dev/sameer/lv1 200M

lvreduce -L 200M /dev/sameer/lv1

mount /dev/sameer/lv1 /data

## **💡 Scenario Example**

**Situation:** Ticket booking app stores data on /dev/sameer/lv1.  
 If LV becomes full, service is affected.

**Solution Path:**

1. Check VG free space:  
    vgdisplay sameer
2. If free space exists → extend LV.
3. If VG full → create new partition /dev/sda2.
4. pvcreate /dev/sda2
5. vgextend sameer /dev/sda2
6. lvextend -l +90%FREE /dev/sameer/lv1
7. resize2fs /dev/sameer/lv1

If no free partitions or disks, reduce less-used LVs or attach a new HDD.

## **🗺️ Mapping LVM Devices (Understand /dev/mapper)**

* /dev/mapper holds **device-mapper nodes** that represent LVs.

Example:  
  
 /dev/mapper/cs\_vbox-root

/dev/mapper/cs\_vbox-home

are mapped from:  
  
 /dev/cs\_vbox/root

/dev/cs\_vbox/home

* These are the actual block devices your OS mounts.

To view complete mapping:

lsblk

## **💾 Swap on LVM**

### **Check Current Swap**

swapon --show

cat /proc/swaps

free -h

If swap path shows /dev/mapper/...swap, then it’s an LV.

### **Create Swap on LV**

lvcreate -L 4G -n swap cs\_vbox

mkswap /dev/cs\_vbox/swap

swapon /dev/cs\_vbox/swap

echo '/dev/cs\_vbox/swap none swap sw 0 0' | sudo tee -a /etc/fstab

### **Resize Swap LV**

swapoff /dev/cs\_vbox/swap

lvresize -L +2G /dev/cs\_vbox/swap

mkswap /dev/cs\_vbox/swap

swapon /dev/cs\_vbox/swap

## **🧩 Golden Rules Summary**

1. A **PV** belongs to exactly **one VG**.
2. **VG size** = sum of all its PVs.
3. **LVs live inside VGs**; filesystems live on LVs.
4. If **LV full**, extend it using VG’s free space.

If **VG full**, add new PV(s) using:  
  
pvcreate /dev/sdXn

vgextend vg0 /dev/sdXn

1. For XFS: only **extend**, cannot **shrink**.

LVM-3

## **Step-by-Step LVM Creation**

## **🔹 Backup and Restore of LVM Metadata**

### **Why Backup Metadata**

When LVM is created, it stores **metadata in the first sector** of the hard disk.  
 If this metadata gets corrupted, the LVM setup may fail.  
 To prevent loss, backup is taken from the **Volume Group** (VG).

### **10. Backup VG Metadata**

**Command:**

vgcfgbackup <vgname>

* Takes a backup of VG metadata.
* Backup file stored at:

ls /etc/lvm/backup/

### **11. LVM Configuration File**

**Command:**

vi /etc/lvm/lvm.conf

* Contains LVM configuration and backup directory path.
* Inside /backup section, it shows where backups are saved.

### **12. Excluding Devices from LVM**

If you **don’t want** to use some disks (like /dev/sdc or /dev/sde) for LVM,  
 you can **exclude them** by editing filter in:

vi /etc/lvm/lvm.conf

Modify:

filter = [ "r|/dev/sdc|", "r|/dev/sde|", "a|.\*|" ]

### **13. Restore VG Metadata**

**Command:**

vgcfgrestore <vgname>

* Restores the VG metadata from backup if corruption occurs.

## **🔹 Snapshot (Point-in-Time Copy)**

**Command:**

lvcreate -L 1G -s -n snapname /dev/<vgname>/<lvname>

* Creates a **snapshot** (not an actual backup).
* It’s a **pointer copy** that refers to the data state at the time of snapshot creation.
* Useful for testing or temporary backup.

**Check Snapshot:**

ls /dev/<vgname>/

## **🔹 Understanding PE (Physical Extent) and LE (Logical Extent)**

* **PE (Physical Extent):** A small slice of space on PV. Default = **4MB**.  
   Example: PV = 100MB → 25 PE (each 4MB).
* **LE (Logical Extent):** LV’s mapping unit corresponding to PE (1:1 mapping).

**Changing PE size during VG creation:**

vgcreate -s 8M <vgname> /dev/sdx

**Example using PE count:**

lvcreate -l 25 -n <lvname> <vgname>

→ 25 PEs × 4MB = 100MB LV

**Example using all free space:**

lvcreate -l 100%FREE -n <lvname> <vgname>

📝 **Note:** If PE = 4MB and you make LV = 102MB → it becomes 104MB (rounded to PE multiple).

## **🧩 Concept Recap**

| **Layer** | **Example** | **Function** |
| --- | --- | --- |
| **PV (Physical Volume)** | /dev/sdb1 | Base partition prepared for LVM |
| **VG (Volume Group)** | vgcreate myvg /dev/sdb1 | Storage pool combining PVs |
| **LV (Logical Volume)** | /dev/myvg/lv1 | Virtual partition for filesystem |
| **PE / LE** | 4MB / 4MB | Fundamental allocation unit |

## **🖼️ Picture Topic (from shared image “122.PNG”)**

*(Include this picture in your final notes — it summarizes the LVM command flow used in exam practice.)*

**Visual Flow:**

#exam --- QS

# hdd create partition fdisk

# 100 extents with each PE = 32MB

# ---lv6

# fdisk /dev/sdb --3500M

# pvcreate /dev/sdb

# vgcreate -s 32M ...

# lvcreate -l 100 ...

# mkfs.xfs ...

# mkdir

# mount

# add in fstab

# df -h

### 

### 

### 

### **Steps to make LV**

### **1. Create Partition**

**Command:**

fdisk /dev/sdb

* Create a new partition (example size: **3500M**).
* Inside fdisk, change the **type ID** to **8e (Linux LVM)** — because kernel recognizes it as an LVM type for recovery and management.
* Save and exit using **w**.

### **2. Create Physical Volume (PV)**

**Command:**

pvcreate /dev/sdb

* Converts the partition /dev/sdb into an **LVM physical volume**.
* PV is the **base layer** in LVM, on which VG is built.

### **3. Create Volume Group (VG)**

**Command:**

vgcreate -s 32M <vgname> /dev/sdb

* Creates a **volume group** and sets **Physical Extent (PE)** size = 32MB.
* Each PE = 32MB.
* Example: 100 extents → 100 × 32MB = **3200MB total space**.

### **4. Create Logical Volume (LV)**

**Command:**

lvcreate -l 100 -n <lvname> <vgname>

* Creates a logical volume using **100 extents** (each extent = 32MB).
* So LV size = **3200MB**.
* -l = number of extents.
* If using -L, that means size in MB or GB.

**Example:**

lvcreate -L 3500M -n lv6 vg1

### **5. Format Logical Volume**

**Command:**

mkfs.xfs /dev/<vgname>/<lvname>

* Formats the LV with **XFS** filesystem.
* After formatting, the LV is ready to mount.

### **6. Create Mount Directory**

**Command:**

mkdir /mnt/lv6

### **7. Mount the LV**

**Command:**

mount /dev/<vgname>/<lvname> /mnt/lv6

### **8. Add Entry in fstab**

* To make mount **permanent** across reboots.  
   **Edit:**

vi /etc/fstab

Add:

/dev/<vgname>/<lvname> /mnt/lv6 xfs defaults 0 0

### **9. Check Mounted Volumes**

**Command:**

df -h

* Verifies mount and shows size, used, and available space.

✅ **Final Summary:** These are the **exact steps and commands** in order — from creating a partition to mounting and verifying LV, including **metadata backup, restore, snapshot, and PE/LE concept**, with **no skipped information** from your material.

Remote Access

## **Remote Access in Linux**

Remote access allows one computer to **control or view another system** over a network.  
 There are different tools depending on the **source (client)** and **destination (server)** operating systems.

### **⚙️ 1. rdesktop (Linux → Windows Remote Access)**

**Definition:** rdesktop is an **open-source Remote Desktop Protocol (RDP) client** for Linux.  
 It lets you **connect from a Linux system to a Windows machine** using the Windows Remote Desktop feature.

#### **🔸 Purpose**

* Used when you want to **control or access a Windows desktop** from your Linux system.
* Works through the **RDP (Remote Desktop Protocol)** service enabled on the Windows host.

#### **🔸 Basic Command**

rdesktop <Windows-IP-address>

#### **🔸 Example**

rdesktop 192.168.18.2

👉 This command opens a remote desktop session to the Windows system at IP 192.168.18.2.

#### **🔸 Common Options**

| **Option** | **Description** |
| --- | --- |
| -u <username> | Specify username for Windows login |
| -p <password> | Provide password (optional; prompts securely if omitted) |
| -f | Full-screen mode |
| -r sound:local | Redirect sound from Windows to Linux |
| -r disk:share=/path | Share a local folder to Windows session |
| -g 1280x720 | Set window resolution |

#### **🔸 Example with Options**

rdesktop -u admin -p 1234 -f 192.168.18.2

➡️ Connects as user admin, full screen, to Windows IP 192.168.18.2.

### **🖧 2. SSH (Linux → Linux Remote Access – CLI)**

**Definition:** SSH (**Secure Shell**) is a **command-line tool** used for **secure remote login** between Linux/Unix systems.

#### **🔸 Purpose**

* Used when you only need **terminal access (text mode)** to another Linux system.
* Encrypts all communication (more secure than telnet).

#### **🔸 Command**

ssh <username>@<Linux-IP-address>

#### **🔸 Example**

ssh root@192.168.18.3

➡️ Opens a terminal session to the Linux host at 192.168.18.3 as user root.

#### **🔸 Other Useful SSH Commands**

| **Command** | **Description** |
| --- | --- |
| ssh-copy-id user@ip | Copy your SSH key for passwordless login |
| scp file user@ip:/path | Copy files securely between systems |
| ssh -p 2222 user@ip | Connect using a non-default port |

### **🖼️ 3. VNC (Linux → Linux Remote Access – GUI)**

**Definition:** VNC (**Virtual Network Computing**) allows **graphical (desktop)** remote access between Linux systems.

#### **🔸 Components**

**VNC Server** — runs on the system you want to access.  
 Command:  
  
 vncserver

1. Starts the VNC service on the remote system.

**VNC Viewer** — runs on the client system (your machine).  
 Command:  
  
 vncviewer <Remote-IP>:<Display-Number>

#### **🔸 Example**

vncviewer 192.168.18.3:1

➡️ Connects to display :1 of the remote Linux system at 192.168.18.3.

#### **🔸 Common VNC Tools**

* TigerVNC
* RealVNC
* TightVNC

#### **🔸 Use Case**

* Useful for **graphical interface** access between two Linux systems.
* Often used for system administration, GUI testing, or remote support.

Patching-Rollback

## **1. Checking for Package Updates**

### **Command**

yum check-update <package-name>

### **Description**

* This command checks if any updates are available for a specific package.
* It queries all configured repositories and shows the latest available versions.

## **2. Viewing Available Updates (All Packages)**

### **Command**

yum check-update

### **Description**

* Displays a list of all packages that have available updates.
* Commonly used before patching to prepare an update report for team review or approval.

## **3. Updating Packages**

### **General Update**

yum update <package-name>

* Updates the given package to the latest version available in enabled repositories.
* It automatically resolves dependencies.

### **Using a Specific Repository**

yum --enablerepo=<repo-name> update <package-name>

* Useful when you want to update a package only from a specific repository (for example, a testing repo).

### **Exclude Certain Packages During Update**

yum update -x mysql,kernel

* The -x option excludes packages you don’t want to update (common during OS patching).

## **4. Searching and Getting Information About Packages**

### **Search a Package**

yum search <package>

* Searches for a package by name or keyword in enabled repositories.

### **View Package Information**

yum info <package>

* Displays details such as version, size, repository, and a short description.

### **Check Installed Packages**

rpm -qa | grep -i <package-name>

* Lists all installed packages; grep filters a specific one.
* Useful for verifying if a package is already installed.

**5 — Installing a Specific Version of a Package**

There are **3 valid methods** to install a particular version when it’s not available locally:

### **A. Using a Direct .rpm File**

✅ **Correct**

**When to use:**

* When you manually download a package (for example, from Red Hat or CentOS mirrors) and want to install without using YUM repos.

**Command:**

rpm -ivh <package-name>-<version>.rpm

**What happens:**

* rpm installs that exact file version.
* However, **it doesn’t auto-resolve dependencies**, so you may need to install missing dependencies manually.

**Tip:** If dependencies are missing, it’s better to use:

yum localinstall <package-name>.rpm

This uses YUM’s dependency solver while installing your local RPM — same result, less headache.

### **B. Installing via YUM Repository**

✅ **Correct (and preferred method)**

**When to use:**

* When multiple versions are available in a configured repository.

**Command:**

yum install <package-name>-<version>

Example:

yum install httpd-2.4.6-99.el7.centos

**What happens:**

* YUM fetches that **exact version** and resolves dependencies automatically.
* If the version isn’t available in your default repo, you can create or add a new .repo file under /etc/yum.repos.d/.

**Repo file example (/etc/yum.repos.d/custom.repo):**

[custom-repo]

name=Custom Local Repo

baseurl=file:///mnt/customrepo/

enabled=1

gpgcheck=0

### **C. Installing from Source Code**

✅ **Technically correct (but least recommended for production)**

**When to use:**

* When no RPM or repo version is available — e.g., experimental or newer open-source builds.

**Process:**

tar -xvf package.tar.gz

cd package

./configure

make

make install

**Drawbacks:**

* No automatic dependency management.
* Package won’t appear in yum or rpm -qa because it bypasses the system package manager.
* You’ll need to track updates manually.

## 

## **6. Kernel Updates**

### **Update Command**

yum update kernel

### **Key Points**

* Multiple kernels can coexist in /boot.
* The newest kernel **is not overwritten**; it’s added as a new entry in the boot menu.
* After reboot, you can choose between old and new kernels from the GRUB screen.

### **Verify Current Kernel**

uname -r

* Shows which kernel is currently running.

If the new kernel fails after update, boot with the old kernel from GRUB and investigate issues.

## **7. Operating System Update**

### **Command**

yum update

### **Description**

* Updates *all* system packages, including dependencies.
* **Avoid running this on production systems** without approval — it can break compatibility if applications rely on specific versions.

## **8. Software Patching Workflow**

1. **Take approval** from your team.
2. **Stop services** if required (in clustered systems, stop one node at a time to avoid downtime).
3. **Take backups** before patching.

**Check the current version:** mysql --version

**Check available updates:** yum check-update mysql

**Update the package:** yum update mysql

## **9. Rollback and Undo Operations**

YUM keeps a **transaction history**, which allows rollback to previous states.

### **View History**

yum history

* Displays the list of all transactions with an **ID**.

### **View Details of a Transaction**

yum history info <ID>

* Shows which packages were installed, updated, or removed in that transaction.

### **Rollback to a Specific Transaction**

yum history rollback <ID>

* Reverts the system to the state of that specific transaction (removes newer updates done after it).

### **Undo a Specific Installation**

yum history undo <ID>

* Uninstalls or reverses only the changes made by that particular transaction.

### **Alternative Method**

rpm -qa | grep <package-name>

* Identify the installed package version and remove it manually if necessary.

## **10. GUI-Based Patching (Optional)**

* YUM and DNF provide graphical interfaces (like yumex or GNOME Software) for patching and updates.
* These are rarely used on servers but available for desktop systems.

✅ **Summary**

| **Task** | **Command** | **Purpose** |
| --- | --- | --- |
| Check available updates | yum check-update | Shows pending updates |
| Update specific package | yum update <pkg> | Updates only selected package |
| Update from specific repo | yum --enablerepo=<repo> update <pkg> | Use defined repository |
| Exclude packages | yum update -x mysql,kernel | Skips listed packages |
| View installed version | `rpm -qa | grep <pkg>` |
| Kernel update | yum update kernel | Adds new kernel in /boot |
| Rollback updates | yum history rollback <ID> | Reverts to specific transaction |
| Undo installation | yum history undo <ID> | Reverses specific changes |

Source Code Installation

# **Understanding Source Code Installation in Linux**

## **1. Concept Overview**

Source-code installation (also known as **tarball installation**) is a **manual method** of installing software directly from its source files (usually distributed as .tar.gz or .tar.bz2 archives).  
 It allows **customization and fine-tuning**—for example, enabling or disabling modules like PHP, SSL, or Java during the build.

Unlike **RPM** or **YUM/DNF** package installations that come precompiled and standardized, **source-based installation compiles code on your system**, ensuring compatibility and control.

## **2. When to Use Source Code Installation**

| **Situation** | **Why Choose Source Code Installation** |
| --- | --- |
| You want to **customize features** or enable specific modules | RPM/YUM versions are fixed; source code lets you build with options. |
| Latest version is **not available in repositories** | Developers often release tarballs before RPMs. |
| You need to **run multiple versions** of the same service (e.g., Apache 2.4.63 and 2.4.64) | Source-based builds can coexist in separate directories like /usr/local/apache2\_64. |
| You are **developing or testing** custom modules | Easier to recompile and link new modules. |

## **3. Prerequisites and Dependencies**

Before building any source package, you must have:

* **gcc / g++** – compiler tools
* **make** – build automation utility
* **pcre**, **apr**, **apr-util**, **zlib**, etc. – common libraries used by Apache and similar software

If any dependency is missing, the ./configure script will fail. Install them using:

yum install gcc make pcre-devel apr-devel apr-util-devel

## **4. Step-by-Step Installation Example (Apache HTTPD 2.4.64)**

Let’s assume your current Apache version is **2.4.63**, and you want to install **2.4.64** manually.

### **Step 1: Download the Source Package**

Get the latest tarball from Apache’s archive:

wget https://downloads.apache.org/httpd/httpd-2.4.64.tar.bz2

👉 wget downloads files from the web to your current directory.

### **Step 2: Extract the Archive**

tar jxvf httpd-2.4.64.tar.bz2

cd httpd-2.4.64

* tar extracts compressed files.
* j flag = bzip2, x = extract, v = verbose, f = file name.
* You’ll now be inside the source directory containing INSTALL, README, and configuration files.

### **Step 3: Read Installation Instructions (Recommended)**

less INSTALL

This file provides official notes from developers — always check it before proceeding.

### **Step 4: Configure the Build**

./configure

#### **What It Does:**

* Checks for system dependencies and paths.
* Creates a **Makefile** based on your system’s configuration.
* You can **customize build options** here.

#### **Example Customization:**

./configure --enable-php --enable-java

Here, you’re enabling support for PHP and Java modules during build.

#### **Check Command Success:**

echo $?

If output = 0, the previous command ran successfully.

### **Step 5: Compile the Source**

make

#### **What Happens:**

* The **Makefile** (generated by ./configure) defines build instructions.
* make compiles the source code into binaries using gcc.
* You can again check the result:

echo $?

### **Step 6: Install the Compiled Program**

make install

#### **What Happens:**

* Copies compiled binaries, libraries, and configuration files into /usr/local/<package\_name>/.

Example for Apache:  
  
 /usr/local/apache2/

💡 This step is **equivalent to** installing an RPM using rpm -i.  
 However, unlike RPM, everything remains inside /usr/local/ and is not distributed across /etc, /bin, or /usr/lib.

## **5. Running and Managing the Installed Service**

### **Start the Apache Server:**

cd /usr/local/apache2/bin

./httpd -k start

### **Stop / Restart:**

./httpd -k stop

./httpd -k restart

You can now access the web server at:  
 [**http://localhost**](http://localhost) (or your server’s IP)

## **6. Uninstalling Software**

| **Type** | **Uninstallation Method** |
| --- | --- |
| **RPM/YUM installation** | rpm -e <package\_name> removes all registered files. |
| **Source code installation** | Delete the installation folder manually:  rm -rf /usr/local/apache2 |

⚠️ Because there’s no package manager tracking, **manual removal** is required for source installations.

## **7. Directory Comparison: RPM vs Source Installation**

| **Feature / File Type** | **RPM / YUM Installation** | **Source Code Installation** |
| --- | --- | --- |
| Binary executables | /usr/bin | /usr/local/apache2/bin |
| Libraries (.so) | /usr/lib or /lib64 | /usr/local/apache2/lib |
| Configuration files | /etc/httpd/conf/ | /usr/local/apache2/conf/ |
| Logs | /var/log/httpd | /usr/local/apache2/logs |
| Service start command | systemctl start httpd | ./httpd -k start |

## **8. Advantages of Source Code Installation**

✅ Full control and customization.  
 ✅ Ability to run multiple versions.  
 ✅ Useful for developers and testing environments.  
 ✅ Independence from distribution repositories.

## **9. Disadvantages / Drawbacks**

❌ No automatic dependency management (must install manually).  
 ❌ Harder to uninstall or upgrade.  
 ❌ No automatic security or bug updates.  
 ❌ Not integrated with systemctl or service commands unless manually configured.

## **10. Summary (Revision-Friendly)**

| **Step** | **Command** | **Description** |
| --- | --- | --- |
| 1 | wget <url> | Download source tarball |
| 2 | tar jxvf <file> | Extract source code |
| 3 | cd <folder> | Move into extracted directory |
| 4 | ./configure [options] | Prepare Makefile & check dependencies |
| 5 | make | Compile the source |
| 6 | make install | Install binaries under /usr/local |
| 7 | ./httpd -k start | Run Apache server manually |
| 8 | echo $? | Verify command success (0 = OK) |
| 9 | rm -rf /usr/local/<folder> | Uninstall manually if needed |

Cron scheduling

## **Linux Services, Cron Scheduler, and System Hardening Guide**

### **1. Linux Services (init vs systemd)**

#### **1.1 What is a Service?**

A **service** in Linux is a background process (daemon) that performs system-level or application-specific functions. Examples include:

* sshd – handles remote login
* httpd – runs Apache web server
* crond – executes scheduled tasks

#### **1.2 Why Services are Important**

* Provide continuous background functionality
* Allow administrators to start/stop or restart processes
* Automate essential operations such as networking, logging, or scheduling

### **1.3 Service Management in RHEL 6 (init system)**

**init** is the parent process (PID 1) that starts all other system processes. It uses runlevels to manage service startup.

**Commands:**

service <service-name> start

service <service-name> stop

service <service-name> restart

service <service-name> status

* start – loads and starts the service
* stop – terminates service process
* restart – restarts the service
* status – shows service running state

**Enable Service on Boot:**

chkconfig <service-name> on

This ensures the service starts automatically at boot.

### **1.4 Service Management in RHEL 7+ (systemd)**

**systemd** replaced init starting from RHEL 7. It also runs as PID 1 but uses a faster, parallelized approach.

**Commands:**

systemctl start <service-name>

systemctl stop <service-name>

systemctl restart <service-name>

systemctl status <service-name>

**Enable/Disable at Boot:**

systemctl enable <service-name>

systemctl disable <service-name>

**Location of Unit Files:**

* /etc/systemd/system/
* /usr/lib/systemd/system/

**Logs:**

journalctl -u <service-name>

### **1.5 Comparison Table**

| **Feature** | **RHEL 6 (init)** | **RHEL 7+ (systemd)** |
| --- | --- | --- |
| Main Process | init (PID 1) | systemd (PID 1) |
| Script Location | /etc/rc.d/init.d/ | /usr/lib/systemd/system/ |
| Command | service | systemctl |
| Auto-start | chkconfig | systemctl enable |
| Boot Model | Runlevel | Target |
| Parallel Startup | No | Yes |
| Logging | /var/log/messages | journalctl |

### **2. Cron Scheduler**

#### **2.1 What is cron?**

**cron** is a job scheduler in Linux that executes tasks automatically at specified intervals. The service responsible for it is crond.

**Check cron service:**

systemctl status crond

If not running:

systemctl start crond

systemctl enable crond

#### **2.2 Why cron is used**

* Automates repetitive tasks (e.g., backups, log rotation)
* Ensures jobs run on schedule even without user intervention

#### **2.3 When to use**

* For recurring system tasks
* To run scripts or commands at a specific time (daily, weekly, monthly)

### **2.4 crontab Structure and Scheduling**

Each user (including root) can have a personal crontab file.

**Edit crontab:**

crontab -e # For current user

crontab -u user -e # For specific user

**View scheduled jobs:**

crontab -l

**Syntax:**

\* \* \* \* \* <command>

| | | | |

| | | | +-- Day of week (0-7, Sunday = 0 or 7)

| | | +---- Month (1-12)

| | +------ Day of month (1-31)

| +-------- Hour (0-23)

+---------- Minute (0-59)

**Example:** Run /opt/backup.sh daily at 2:00 AM

00 2 \* \* \* /opt/backup.sh

**Special Patterns:**

* \*/1 – every minute
* \*/5 – every 5 minutes
* 1-5 – from Monday to Friday
* 0 2 \*/2 \* \* – every alternate day at 2:00 AM

### **2.5 Backup and Logs of cron jobs**

**Backup crontab:**

cat /var/spool/cron/root > /mnt/cron\_backup.txt

**Check execution logs:**

cat /var/log/cron

### **2.6 How cron Works (Behind the Scenes)**

* crond service loads jobs into memory (RAM)
* Every minute, it checks /var/spool/cron/ for due jobs
* If a time match is found, it executes the job
* If not, it sleeps until the next minute

### **2.7 Controlling User Access to cron**

**Deny cron access:**

echo username >> /etc/cron.deny

**Allow specific users:**

echo username >> /etc/cron.allow

If /etc/cron.allow exists, only users in that file can schedule jobs. If it doesn’t exist, /etc/cron.deny decides who is blocked.

**Security/Hardening Role:** Prevents unauthorized users from running scheduled scripts.

### **3. Other Linux Schedulers**

| **Scheduler** | **Type** | **When Used** | **Example** |
| --- | --- | --- | --- |
| **cron** | Repetitive | For regular intervals | Daily backups |
| **at** | One-time | Schedule a job once | at 3:00 PM tomorrow |
| **anacron** | Missed cron recovery | For systems not always on | Laptop updates |

#### **Example system anacron directories:**

/etc/cron.daily

/etc/cron.hourly

/etc/cron.weekly

System uses these directories to run scheduled maintenance scripts.

system also do scheduling

### **4. System Hardening**

#### **4.1 What is Hardening?**

**System Hardening** means securing the operating system by reducing its vulnerability surface.

#### **4.2 Why it’s done**

* Prevent unauthorized access
* Limit exploit opportunities
* Improve overall security posture

#### **4.3 When to apply**

* Before deploying a server to production
* After OS installation or security audits

#### **4.4 Common Hardening Steps**

Stop **unwanted services**:  
  
systemctl disable telnet

systemctl stop telnet

Remove **unused packages**:

yum remove ftp vsftpd

- Set \*\*strong passwords\*\* and \*\*account lock policies\*\*

- Secure \*\*single-user mode\*\* (require root password)

- Protect \*\*bootloader (grub)\*\* with a password

- Disable \*\*root SSH login\*\*:

```bash

vi /etc/ssh/sshd\_config

PermitRootLogin no

Enable **firewall**:  
  
systemctl enable firewalld

systemctl start firewalld

### **5. Troubleshooting and Common Issues**

| **Problem** | **Cause** | **Solution** |
| --- | --- | --- |
| Service not starting | Misconfigured unit file | Check with journalctl -xe |
| Cron job not running | Script not executable | Run chmod +x script.sh |
| Cron not logging | Syslog disabled | Enable rsyslog and restart cron |
| User denied from cron | In cron.deny list | Remove username from /etc/cron.deny |
| Firewall blocking | Port closed | Check firewall-cmd --list-all |

### **6. Summary / Revision Table**

| **Step** | **Command** | **Description** |
| --- | --- | --- |
| 1 | systemctl start <srv> | Start a service |
| 2 | systemctl enable <srv> | Auto-start at boot |
| 3 | service <srv> status | Check service (RHEL6) |
| 4 | crontab -e | Edit scheduled jobs |
| 5 | crontab -l | List user cron jobs |
| 6 | systemctl status crond | Check cron service |
| 7 | cat /var/log/cron | View cron logs |
| 8 | echo user >> /etc/cron.deny | Block user cron access |
| 9 | yum remove <pkg> | Remove unnecessary packages |
| 10 | systemctl enable firewalld | Enable OS firewall |

User Management

# **Linux User Management**

## **1. Overview**

**User management** in Linux controls who can access the system, how they authenticate, and what resources they can use.  
 Each account is defined by several text files that together describe:

* The user’s identity (username, UID, GID).
* Their authentication (passwords in /etc/shadow).
* Their permissions (group membership).
* Their environment (home directory, shell, startup files).

Whenever you create, modify, or delete a user, Linux updates these configuration files and directory structures automatically.

## **2. Core Files Involved**

| **File** | **Description** |
| --- | --- |
| /etc/passwd | Lists all user accounts with their UID, GID, home directory, and shell. |
| /etc/shadow | Stores encrypted passwords and password-aging rules. |
| /etc/group | Defines groups and their members. |
| /etc/gshadow | Holds secure group passwords. |
| /etc/login.defs | System-wide default rules for account creation and password policy. |
| /etc/default/useradd | Default values used by the useradd command. |
| /etc/skel | Template directory copied to every new user’s home folder. |

Understanding these files is critical, because almost all useradd, passwd, and usermod operations modify them.

## **3. Creating a New User**

### **Command**

useradd khan

### **Description**

**What it is:** useradd is the standard Linux utility for creating new local users.

**Why it’s used:** To register a new user account and automatically create its UID, GID, home directory, and default shell.

**When to use:** Whenever a new employee or service account needs access to the system.

**How it works:**

1. Reads global defaults from /etc/login.defs and /etc/default/useradd.
2. Allocates the next available **UID** and **GID**.
3. Adds an entry in:  
   * /etc/passwd (basic info)
   * /etc/shadow (password placeholder)
   * /etc/group and /etc/gshadow (group info)
4. Creates a home directory (e.g., /home/khan).
5. Copies template files from /etc/skel into the new home directory.
6. Sets ownership and permissions (typically 700).

## **4. Setting a Password**

### **Command**

passwd khan

### **Description**

**What it is:** The passwd command manages user authentication passwords.

**Why it’s used:** To assign or update a password, making the user account functional for login.

**When to use:** Immediately after user creation or any time password reset is required.

**How it works:**

1. Prompts for the new password twice.
2. Encrypts it (usually with SHA-512).
3. Writes the encrypted hash into /etc/shadow.

After this, khan becomes a valid, login-ready Linux user.

## **5. /etc/skel — Default User Environment**

### **Description**

**What it is:** A *skeleton* directory that contains default shell configuration files.

**Why it’s used:** To ensure every new user starts with a consistent environment (PATHs, aliases, color prompts, etc.).

**When to use:** During user creation. useradd automatically copies its contents into each new home directory.

**How it works:** Files from /etc/skel → copied to /home/<username>.

### **Typical Contents**

| **File** | **Purpose** |
| --- | --- |
| .bash\_profile | Executed at login; sets environment variables and PATH. |
| .bashrc | Executed for every shell; defines aliases and colors. |
| .bash\_logout | Executed at logout; clears screen or temporary files. |

### **Rebuilding /etc/skel**

sudo mkdir -p /etc/skel

sudo cp /etc/bashrc /etc/skel/.bashrc

sudo cp /etc/profile /etc/skel/.bash\_profile

sudo touch /etc/skel/.bash\_logout

## **6. /etc/login.defs — Default Account Rules**

### **Description**

**What it is:** A configuration file defining global password and UID/GID policies.

**Why it’s used:** Ensures consistent security standards (like password age limits) across all users.

**When to use:** Automatically referenced whenever a new user is added.

**Important Fields**

| **Field** | **Description** |
| --- | --- |
| PASS\_MAX\_DAYS | Max number of days before password expires. |
| PASS\_MIN\_DAYS | Min days before a password can be changed again. |
| PASS\_WARN\_AGE | Days before expiry to warn the user. |
| UID\_MIN / UID\_MAX | Range of UIDs for normal (non-system) users. |
| GID\_MIN / GID\_MAX | Range of GIDs for normal groups. |
| CREATE\_HOME | Controls if useradd creates a home directory. |
| ENCRYPT\_METHOD | Specifies password hashing algorithm (e.g., SHA512). |

## **7. /etc/default/useradd — useradd Behavior**

### **Description**

**What it is:** Defines the default options useradd applies to all new users.

**Why it’s used:** To pre-configure values like default shell or home path so administrators don’t need to type them every time.

**Important Fields**

| **Field** | **Function** |
| --- | --- |
| HOME | Base directory for home folders (/home). |
| SHELL | Default login shell (/bin/bash). |
| SKEL | Skeleton directory path (/etc/skel). |
| INACTIVE | Days after password expiry before disabling account. |
| EXPIRE | Account expiration date. |
| CREATE\_MAIL\_SPOOL | Whether to create /var/mail/<user>. |

## **8. /etc/passwd — User Information Database**

### **Description**

**What it is:** The main text file containing one line per user.

**Why it’s used:** To map usernames to UIDs, home directories, and shells.

**Format**

username:x:UID:GID:comment:/home/dir:/bin/bash

| **Field** | **Description** |
| --- | --- |
| username | Login name. |
| x | Placeholder (password stored in /etc/shadow). |
| UID | Unique user ID. |
| GID | Primary group ID. |
| comment | User description (GECOS field). |
| home directory | User’s personal directory. |
| shell | Default shell. |

**Usage Tips**

* Comment the line or set shell to /sbin/nologin to disable login.

View users:  
  
 cat /etc/passwd

## **9. /etc/shadow — Encrypted Passwords**

### **Description**

**What it is:** Stores hashed passwords and password expiration data.

**Why it’s used:** To protect credentials; readable only by root.

**Format (9 fields)**

| **Field** | **Meaning** |
| --- | --- |
| 1 | Username |
| 2 | Encrypted password |
| 3 | Last password change (days since 1-1-1970) |
| 4 | Min days before next change |
| 5 | Max days before expiry |
| 6 | Warning period |
| 7 | Inactive days before disable |
| 8 | Account expiry date |
| 9 | Reserved for future use |

**Edit safely**

vipw -s

## **10. /etc/group — Group Management**

### **Description**

**What it is:** A text file listing all groups and their members.

**Why it’s used:** Groups control file access and permission sharing among users.

**Format**

groupname:x:GID:user1,user2

**Useful Commands**

getent group # List all groups

getent group khan # Show specific group info

**Add or Modify Membership**

usermod -G <group> <user> # Add to secondary group

usermod -g <group> <user> # Change primary group

## **11. Viewing Users**

### **Command**

compgen -u

### **Description**

Lists all users on the system (reads the first field of /etc/passwd).

## **12. Modifying User Accounts**

### **Command**

usermod -u <uid> -d <home\_dir> -c <comment> -s <shell> username

### **Description**

Updates an existing user’s properties such as UID, home directory, comment, or shell.

## **13. Creating Customized Users**

### **Command**

useradd -u <uid> -d <home\_dir> -c <comment> -s <shell> username

### **Description**

Creates a user with manually defined attributes instead of using defaults.

## **14. Deleting Users**

| **Action** | **Command** | **Description** |
| --- | --- | --- |
| Delete user only | userdel khan | Removes entry but keeps files. |
| Delete user + home dir | userdel -r khan | Removes both user and home directory. |

## **15. Locking and Disabling Accounts**

| **Task** | **Command** | **Description** |
| --- | --- | --- |
| Lock a user | passwd -l <username> | Prevents login by prepending ! to the hash in /etc/shadow. |
| Unlock a user | passwd -u <username> | Removes the ! to restore access. |
| Disable all users | touch /etc/nologin | Blocks all non-root logins temporarily. |

To disable a single account permanently, set shell to /sbin/nologin in /etc/passwd.

## **16. Aliases — Command Shortcuts**

### **Command Examples**

alias khol='ls'

Creates a temporary shortcut for ls.

**Make permanent** Add to /etc/bashrc or ~/.bashrc:

alias khol='ls'

**Remove alias**

unalias khol

## **17. Symlinks — Custom Command Names**

### **Example**

ln -s /usr/sbin/useradd /usr/sbin/userbnao

userbnao khan

userbnao now behaves exactly like useradd.  
 Symlinks are useful for creating short or language-specific command alternatives.

## **18. Troubleshooting**

| **Problem** | **Likely Cause** | **Solution** |
| --- | --- | --- |
| Login fails | Shell set to /sbin/nologin | Change to /bin/bash. |
| Password rejected | Corrupt /etc/shadow entry | Reset with passwd <user>. |
| .bashrc missing | /etc/skel not configured | Rebuild /etc/skel. |
| “User exists” error | Old line remains in /etc/passwd | Remove or comment it. |
| Account locked | Previously locked by passwd -l | Unlock with passwd -u. |

## **19. Summary / Revision Table**

| **Step** | **Command** | **Description** |
| --- | --- | --- |
| 1 | useradd <user> | Create a new user. |
| 2 | passwd <user> | Set or change password. |
| 3 | compgen -u | List all users. |
| 4 | userdel -r <user> | Delete user and home dir. |
| 5 | usermod -G <group> <user> | Add to secondary group. |
| 6 | passwd -l <user> | Lock account. |
| 7 | passwd -u <user> | Unlock account. |
| 8 | touch /etc/nologin | Disable all users temporarily. |
| 9 | getent group | Display groups. |
| 10 | vipw -s | Safely edit /etc/shadow. |

## **20. Quick Concept Recap**

* **User = Identity** (in /etc/passwd)
* **Password = Authentication** (in /etc/shadow)
* **Group = Permission set** (in /etc/group)
* **Shell & Home = Environment** (from /etc/skel)
* **login.defs & useradd defaults = Policy templates**

Advance permission

# **Linux Advanced File Permissions (SUID, SGID, Sticky Bit & Sudoers)**

## **1. Basic File Permissions Recap**

Linux controls file access using **three classes** of users and **three permission types**.

| **Class** | **Description** |
| --- | --- |
| **u** | User (Owner) |
| **g** | Group |
| **o** | Others |
| **a** | All (u+g+o) |

| **Symbol** | **Permission** | **Value** | **Meaning** |
| --- | --- | --- | --- |
| **r** | Read | 4 | Can view file contents |
| **w** | Write | 2 | Can modify or delete file |
| **x** | Execute | 1 | Can run the file as a program |

**Example:**

-rwxr-xr--

* User → rwx (7)
* Group → r-x (5)
* Others → r-- (4)

**Octal form:** 754

## **2. File-Level Permission Control**

To modify permissions:

chmod ugo+rwx <filename>

This example gives **read, write, and execute** permissions to **user, group, and others**.

## **3. Introduction to Advanced Permissions**

Standard permissions (rwx) are not enough for special cases like allowing normal users to run administrative commands (fdisk, mount, passwd, etc.).  
 Such commands require **root privileges** — but Linux provides **Advanced Permissions** to safely allow controlled access.

### **The three advanced permission bits are:**

| **Permission** | **Symbol** | **Value** | **Applied On** | **Purpose** |
| --- | --- | --- | --- | --- |
| **SUID (Set User ID)** | s/S | 4 | Files | Run file as file owner |
| **SGID (Set Group ID)** | s/S | 2 | Files/Dirs | Run as group / inherit group |
| **Sticky Bit** | t/T | 1 | Directories | Protect users’ files from deletion |

## **4. Understanding SUID (Set User ID)**

### **Definition**

SUID allows a normal user to **run a program with the privileges of the file owner (usually root)**.

**Example:** The passwd command lets users change passwords but needs to modify /etc/shadow (root-owned).  
 It works because SUID is set.

-rwsr-xr-x 1 root root /usr/bin/passwd

* s in the user field → SUID with execute permission.

### **When and Why Used**

* Needed for system commands that require root privileges temporarily.
* Avoid giving full root access to users.
* Security must be handled carefully (never set SUID on untrusted binaries).

### **Setting and Removing SUID**

| **Action** | **Command** |
| --- | --- |
| Add SUID | chmod u+s <filename> |
| Remove SUID | chmod u-s <filename> |

Note: When viewing permissions:

* **s** → SUID + execute permission
* **S** → SUID without execute permission (inactive)

### **Octal Representation**

SUID = 4  
 Example:

chmod 4777 <filename>

* The **first digit (4)** represents the **advanced permission**.
* Remaining digits (777) represent normal ugo permissions.

### **View and Remove All SUID Files**

find / -perm -4000 -type f -print

find / -perm -4000 -exec chmod u-s {} \;

## **5. SGID (Set Group ID)**

### **Definition**

SGID runs a file with the **group privileges** of the file, not the user who executes it.  
 For directories, it ensures **new files inherit the directory’s group**.

-rwxr-sr-x 1 root staff /usr/bin/write

* s in the **group** field → SGID active.

### **Usage**

* Used for **shared project directories** or **multi-user group work**.
* All files created inside share the same group ownership.

### **Commands**

| **Action** | **Command** |
| --- | --- |
| Add SGID | chmod g+s <filename> |
| Remove SGID | chmod g-s <filename> |
| Octal Value | chmod 2777 <filename> |

### **View SGID Files**

find / -perm -2000 -type f -print

## **6. Sticky Bit**

### **Definition**

The **Sticky Bit** is used on **shared directories** to prevent users from deleting each other’s files.

**Example:** /tmp directory:

drwxrwxrwt 9 root root /tmp

t → sticky bit with execute permission.

### **Usage**

* Only file owners and root can delete or rename their own files inside that directory.
* Common on: /tmp, /var/tmp, /shared.

### **Commands**

| **Action** | **Command** |
| --- | --- |
| Add Sticky Bit | chmod o+t <directory> |
| Remove Sticky Bit | chmod o-t <directory> |
| Octal Value | chmod 1777 <directory> |

### **View Sticky Bit Directories**

find / -perm -1000 -type d -print

## **7. Quick Summary of “s/S” and “t/T” Indicators**

| **Symbol** | **Meaning** |
| --- | --- |
| s | SUID/SGID **with execute (x)** |
| S | SUID/SGID **without execute** |
| t | Sticky Bit **with execute (x)** |
| T | Sticky Bit **without execute** |

## **8. Combining Advanced Permissions**

You can apply multiple advanced permissions simultaneously.

chmod u+s,g+s,o+t <filename>

Or using octal:

chmod 7777 <filename>

(4 + 2 + 1 = 7 for advanced bits)

## **9. SUID vs Sudoers**

| **Feature** | **SUID** | **Sudoers** |
| --- | --- | --- |
| Level | File-level | User-level |
| Privilege | File runs as file owner | Specific user can run specific commands as root |
| Control | Global – affects everyone | Controlled per user |
| Risk | Security risk if misused | Safer, requires password |
| Config File | None | /etc/sudoers |

## **10. Configuring Sudo Access**

### **File Location**

/etc/sudoers

### **Safe Editing**

visudo

visudo is a **debugging-safe editor** — it checks syntax before saving.

### **Grant Permission for Specific Command**

Example:

sameer ALL=/sbin/fdisk -l

Now user sameer can run:

sudo /sbin/fdisk -l

### **Grant Full Root Privileges**

sameer ALL=(ALL:ALL) ALL

#### **Breakdown:**

sameer ALL=(ALL:ALL) ALL

│ │ │ │

│ │ │ └──► Commands user can run (ALL = any command)

│ │ └──────────────► Run as any user or group (ALL = any)

│ └────────────────────► Applies to all hosts

└────────────────────────────► Username

**Meaning:** User **sameer** can execute **any command**, on **any host**, as **any user/group** using **sudo**.  
 ➡️ Gives **full root privileges**.

### **Give Access to Specific Service Management**

Example: Allow user sameer to start or stop Apache (httpd) **without password**:

sameer ALL=(ALL) NOPASSWD: /usr/bin/systemctl start httpd, /usr/bin/systemctl stop httpd

### **Security Note**

* Avoid giving ALL=(ALL:ALL) ALL unless necessary.
* Use NOPASSWD sparingly — it bypasses password checks.
* Check syntax using visudo to avoid lockouts.

## **11. Root vs Admin**

* **Root** is not the only admin.
* Any user with **UID = 0** is treated as **superuser**.

### **Check UID**

id <username>

### **Change User ID**

usermod -u 0 <username>

⚠️ Be careful: after this change, that user has **full root privileges**.

## **12. Revision Summary Table**

| **Feature** | **Symbol** | **Octal** | **Command** | **Description** |
| --- | --- | --- | --- | --- |
| SUID | s/S | 4 | chmod u+s file | Run as owner |
| SGID | s/S | 2 | chmod g+s file | Run as group |
| Sticky Bit | t/T | 1 | chmod o+t dir | Protect deletions |
| SUID Files | — | — | find / -perm -4000 -print | Locate SUID files |
| SGID Files | — | — | find / -perm -2000 -print | Locate SGID files |
| Sticky Dirs | — | — | find / -perm -1000 -print | Locate Sticky directories |
| Edit sudoers | — | — | visudo | Safe edit |
| Root privilege to user | — | — | sameer ALL=(ALL:ALL) ALL | Full sudo |
| Command-specific sudo | — | — | sameer ALL=/sbin/fdisk -l | Limited sudo |
| UID 0 user | — | — | usermod -u 0 user | Make user admin |

PATH/ UMASK

# **Linux PATH Environment Variable Guide**

## **1. What is PATH?**

The **PATH** variable in Linux defines the **directories where the shell looks for executable commands**.  
 When you type a command (like ls or cat), Linux doesn’t search your entire disk — it looks only inside directories listed in your **PATH**.

**Example:** When you type:

ls

The shell checks each directory in $PATH (in order) to find the ls binary — usually in /bin or /usr/bin.

## **2. View the Current PATH**

### **Command**

echo $PATH

### **Example Output**

/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/snap/bin

This means Linux searches these folders (from left to right) whenever you run a command.

## **3. Find Where a Command Comes From**

To know which exact file executes when you type a command:

which <command>

### **Example**

which fdisk

Output:

/usr/sbin/fdisk

→ This means the system runs fdisk from /usr/sbin.

## **4. How PATH Works**

* When you enter a command, the shell searches each directory listed in $PATH in sequence.
* The **first matching executable** is run.

If the command is **not found** in any of those directories, you’ll get:  
  
 bash: command: command not found

To run commands outside of $PATH, you must give their **absolute path**:  
  
 /opt/mytool/run.sh

## **5. Temporarily Add a Directory to PATH**

You can extend your current PATH using:

PATH=$PATH:/opt

### **Explanation**

* $PATH keeps the existing directories.
* /opt is appended at the end.
* This change is **temporary** — it will work only until you **log out or reboot**.

### **Verify**

echo $PATH

You should now see /opt added to the end.

## **6. Permanently Add a Directory to PATH**

If you want your added path (like /opt) to remain **after reboot**, you must save it in a configuration file that loads automatically when the system starts or when a user logs in.

### **Option 1 — For All Users (Global Change)**

#### **File: /etc/profile**

This file is executed **for all users** at login.

#### **Steps:**

Open file with root privileges:  
  
 sudo vi /etc/profile

Scroll to the bottom and add:  
  
PATH=$PATH:/opt

export PATH

Save and exit (:wq).

Reload to apply immediately:  
  
source /etc/profile

Verify:  
  
 echo $PATH

### **Option 2 — For a Specific User Only**

#### **File: ~/.bashrc (or ~/.bash\_profile)**

#### **Steps:**

Edit your bashrc:  
  
 nano ~/.bashrc

Add at the end:  
  
PATH=$PATH:/opt

export PATH

Save file and reload:  
  
source ~/.bashrc

Check:  
  
 echo $PATH

Now the directory /opt will **automatically load into PATH** every time that user logs in.

## **7. Confirm Changes on Reboot**

After reboot, run:

echo $PATH

You should see your new directory (/opt) included, confirming that it has been successfully persisted.

## **8. Summary Table**

| **Command** | **Description** |
| --- | --- |
| echo $PATH | Displays all directories in current PATH |
| which <cmd> | Shows full path of a command binary |
| PATH=$PATH:/opt | Temporarily add /opt to PATH |
| source /etc/profile | Reload global profile |
| source ~/.bashrc | Reload user profile |
| /etc/profile | Global environment configuration |
| ~/.bashrc | User-specific configuration |
| export PATH | Makes PATH available to all child processes |

## **9. Key Points to Remember**

* $PATH controls where Linux looks for executables.
* Use **absolute path** when command isn’t inside $PATH.
* which helps trace the source of any command.
* Use /etc/profile for **system-wide** persistence.
* Use ~/.bashrc for **user-only** persistence.
* Always use source after editing to apply changes instantly.

**NOte:**

# **Bash shells & configs — quick notes**

## **~ (tilde)**

* ~ = your **home** (e.g., /home/sameer).
* ~/.bashrc = file inside your home.

## **Shell types (think: first entry vs extra tab)**

* **Login shell** = the shell you get when you **first log in**.  
   Examples: console login, ssh user@host, su - user, bash --login
* **Interactive non-login shell** = an **extra interactive shell** after login.  
   Examples: open a new terminal tab, run bash, su user (no -)

## **Which files are read**

* **Login shell** reads:  
   /etc/profile → then ~/.bash\_profile (which often calls ~/.bashrc)
* **Interactive non-login** reads:  
   /etc/bashrc → then ~/.bashrc  
   *(Debian/Ubuntu name: /etc/bash.bashrc)*

## **su vs su -**

* su user → **interactive, non-login** (quick switch). Reads ~/.bashrc.
* su - user → **login shell** (full, clean env). Reads /etc/profile + ~/.bash\_profile.

## **Reloading configs (no restart)**

* Reload system interactive settings:  
   source /etc/bashrc
* Reload your interactive settings:  
   source ~/.bashrc
* Reload login/profile stuff in current shell (rare):  
   source /etc/profile

## **Why source /etc/bashrc?**

* Immediately apply new aliases/prompt/functions you added to /etc/bashrc (current terminal only).

# **What “system” vs “your (user) interactive” means**

## **Scope & files**

* **System settings** = defaults for **all users**.  
   Live in **/etc** (needs sudo).  
  + Login env: **/etc/profile** (+ /etc/profile.d/\*.sh)
  + Interactive env: **/etc/bashrc** (Debian/Ubuntu name: /etc/bash.bashrc)
* **Your settings** = only for **your account**.  
   Live in **your home (~)**.  
  + Login env: **~/.bash\_profile** (may source ~/.bashrc)
  + Interactive env: **~/.bashrc**

## **What goes where (rule of thumb)**

* **Login/profile stuff** (set once per login): PATH, LANG, EDITOR, umask, ulimit, proxies → put in **/etc/profile** or **~/.bash\_profile**.
* **Interactive stuff** (used while typing): aliases, prompt (PS1), completions, set -o vi, shell options, functions → put in **/etc/bashrc** or **~/.bashrc**.

## **Which file to reload**

* **System interactive** changes → source /etc/bashrc
* **Your interactive** changes → source ~/.bashrc
* **Login/profile** changes (rare to reload) → source /etc/profile (or log out/in)

# **How to tell which kind of setting you’re looking at**

### **Quick identifiers**

* Path starts with **/etc/...** → **system** (affects everyone).
* Path starts with **~/...** → **your** file (affects only you).
* File name **\*profile\*** → **login/profile** context.
* File name **\*bashrc\*** → **interactive** context.

Kernal Management

# **Linux Kernel & Driver Management — Complete Guide**

## **1. What Is the Kernel?**

The **kernel** is the **core component of the Linux operating system**.  
 It acts as a **bridge between hardware and software**, managing how applications interact with system resources like CPU, memory, and devices.

In simple terms:

The kernel = “heart of the operating system.”

## **2. Key Functions of the Kernel**

| **Function** | **Description** |
| --- | --- |
| **Process Management** | Handles process creation, scheduling, and termination. |
| **Memory Management** | Allocates and tracks physical & virtual memory. |
| **Device Management** | Communicates with hardware devices using drivers. |
| **File System Management** | Controls how data is read/written to storage. |
| **Networking** | Manages network protocols and data transmission. |
| **System Security** | Enforces permissions, authentication, and isolation. |

## **3. Kernel Location and Structure**

* The kernel binary is located in the **/boot** directory.

Typically named:  
  
 vmlinuz-<version>

Example:  
  
 /boot/vmlinuz-5.15.0-1051-aws

**Explanation of terms:**

* vmlinuz → “Virtual Memory Linux, compressed”
* It’s a **binary executable file**, developed in **C and C++**.
* Compressed using **bzip2**, **gzip**, or **xz** to save space.
* On boot, it is **decompressed and loaded into RAM** by the **bootloader (GRUB)**.

## **4. Kernel Composition**

The kernel internally includes:

| **Component** | **Description** |
| --- | --- |
| **Process Scheduler** | Decides which process runs next (CPU time allocation). |
| **Memory Manager** | Handles RAM, swap, caching, and virtual memory. |
| **Device Drivers** | Interface between hardware and kernel. |
| **File System Layer** | Manages data read/write on disk. |
| **Networking Stack** | Implements TCP/IP and other network protocols. |
| **System Call Interface (SCI)** | Gateway for user programs to request kernel services. |

## **5. Kernel Drivers (Modules)**

### **Definition**

Drivers are **kernel modules** that extend the kernel’s functionality — they allow it to communicate with new or specific hardware **without recompiling the entire kernel**.

These are also called:

* **KO** → Kernel Object file (.ko)
* **KLM** → Kernel Loadable Module
* **LKM** → Loadable Kernel Module

Each driver handles one hardware or subsystem (e.g., network card, sound card, USB, etc.).

## **6. Kernel Driver Location**

All driver modules are stored under:

/lib/modules/<kernel\_version>/kernel/

Example:

/lib/modules/6.6.25-200.fc39.x86\_64/kernel/

This directory contains subfolders like:

* drivers/net → Network drivers
* drivers/sound → Audio drivers
* drivers/usb → USB drivers
* fs/ → File system support modules

This structure helps identify **which hardware features your kernel supports**.

## **7. Commands to Explore Kernel Modules**

### **(a) Count all available drivers**

ls -R | grep -i .ko | wc -l

* -R → Recursive listing (all subdirectories)
* grep -i .ko → Filters all kernel object files
* wc -l → Counts total number of drivers

### **(b) Using find command**

find . -iname "\*.ko" | wc -l

This also searches recursively for all .ko files.

### **(c) Show folder hierarchy**

tree .

Displays directory structure in tree form (requires tree package).

## **8. Viewing Loaded Drivers (Modules in RAM)**

When the system boots, only **required drivers** are loaded into RAM.

To see currently loaded drivers:

lsmod

**Output example:**

Module Size Used by

e1000e 303104 0

snd\_hda\_intel 57344 1

usb\_storage 73728 1

* **Module** → Driver name
* **Size** → Memory used in KB
* **Used by** → Number of processes using it

## **9. Adding and Removing Kernel Modules**

### **(a) Insert a driver manually**

insmod <driver\_name.ko>

Loads a driver into the kernel **manually** (full path required).

Example:

insmod /lib/modules/$(uname -r)/kernel/drivers/usb/usbcore.ko

**Note:** insmod does not handle dependencies automatically.

### **(b) Add driver with dependency management**

modprobe <driver\_name>

* Smarter than insmod
* Automatically loads all dependent modules

Example:

modprobe e1000e

### **(c) Remove driver from RAM**

rmmod <driver\_name>

Removes a loaded driver **if not in use**.

Example:

rmmod e1000e

If the driver is currently in use, you’ll see:

rmmod: ERROR: Module is in use

To force remove (use cautiously):

rmmod -f <driver\_name>

### **(d) List module info**

modinfo <driver\_name>

**Example:**

modinfo e1000e

**Output:**

filename: /lib/modules/6.6.25-200.fc39.x86\_64/kernel/drivers/net/e1000e/e1000e.ko.xz

version: 3.8.7

license: GPL

description: Intel(R) PRO/1000 Network Driver

author: Intel Corporation

## **10. Permanently Deleting a Driver**

If a driver is obsolete or unsupported and you wish to delete it permanently:

Locate the .ko file:  
  
 find /lib/modules/$(uname -r) -name "<driver\_name>.ko"

Delete the file:  
  
 sudo rm -rf /lib/modules/<kernel\_version>/kernel/drivers/.../<driver\_name>.ko

Update module dependencies:  
  
 sudo depmod -a

⚠️ Warning: Deleting kernel modules permanently can make devices stop working — use only when certain.

## **11. Useful Kernel & Module Commands Summary**

| **Command** | **Description** |
| --- | --- |
| uname -r | Show running kernel version |
| ls /boot | View installed kernel binaries |
| lsmod | List loaded kernel modules |
| insmod <file.ko> | Insert module manually |
| modprobe <module> | Insert module with dependencies |
| rmmod <module> | Remove module from RAM |
| modinfo <module> | View module details |
| depmod -a | Rebuild module dependency map |
| find /lib/modules/$(uname -r) -name "\*.ko" | List all modules for current kernel |

## **12. Driver Management Workflow**

**Check if driver is loaded:** lsmod | grep <driver>

**Load driver if missing:** modprobe <driver>

**Verify it loaded:** lsmod | grep <driver>

**Get info:** modinfo <driver>

**Remove if not required:** rmmod <driver>

## **13. Kernel Module Configuration Files**

Persistent module configurations are stored in:

/etc/modprobe.d/

You can add .conf files here to **auto-load or blacklist** specific drivers at boot.

Exmple:

blacklist nouveau

(to disable NVIDIA open-source driver)

## **14. Summary / Revision Table**

| **Concept** | **Command / Path** | **Description** |
| --- | --- | --- |
| Kernel binary | /boot/vmlinuz-\* | Core OS file, compressed binary |
| Kernel version | uname -r | Shows current running kernel |
| Kernel modules directory | /lib/modules/<version>/kernel/ | Contains all drivers |
| List loaded modules | lsmod | Displays modules loaded into RAM |
| Load module | modprobe <name> | Load with dependencies |
| Remove module | rmmod <name> | Unload from RAM |
| Module details | modinfo <name> | Show version, author, license, etc. |
| Count drivers | `find . -iname "\*.ko" | wc -l` |
| Update dependencies | depmod -a | Rebuild module dependency table |
| Blacklist driver | /etc/modprobe.d/blacklist.conf | Prevent loading unwanted module |

## **15. Key Takeaways**

* **Kernel = Core OS component** managing hardware & system resources.
* **Drivers = Kernel modules (.ko)** that let the kernel talk to devices.
* **Modules can be dynamically loaded/unloaded** without reboot.
* **lsmod → see loaded**, **modprobe → add**, **rmmod → remove**, **modinfo → inspect**.
* Driver files are stored under /lib/modules/<kernel\_version>/kernel/.
* Use /etc/modprobe.d/ for persistent configuration or blacklisting.

Kernel Tunning

# **Linux Kernel Driver Patching & Kernel Tuning — Complete Guide**

This guide combines your notes on **driver patching, kernel types, tuning, and persistence**, written in the same structured, exam-friendly format as your previous notes.

## **1. What Is a Driver?**

A **driver** is specialized software that enables the **operating system (OS)** to communicate with **hardware devices** such as printers, keyboards, network cards, and sound chips.  
 Without drivers, the OS cannot understand or control the hardware.

## **2. Types of Drivers**

| **Type** | **Where It Works** | **Communicates With** | **Examples** | **Purpose** |
| --- | --- | --- | --- | --- |
| **Hardware-level driver** | Runs very close to the device (firmware or kernel) | Directly with the hardware | GPU firmware, BIOS driver, printer firmware | Controls low-level physical operations like timing, voltage, and signaling |
| **OS-level driver** | Part of the operating system (Linux, Windows, macOS) | Between apps and hardware-level driver | .ko in Linux, .sys in Windows | Provides a controlled interface for the OS and applications |

## **3. Why Both Are Needed**

* Hardware only understands **machine code and electrical signals**.
* Operating systems and apps use **high-level commands** (“play sound,” “print file”).  
   Drivers act as translators.

### **Data Flow Example (Sound Playback)**

App (Spotify)

↓

System Call / Audio API (PulseAudio, DirectSound)

↓

OS-level driver (.ko / ALSA)

↓

[Kernel Space Boundary]

↓

Kernel (handles memory, buffers, interrupts)

↓

Hardware-level driver / firmware (DAC inside sound card)

↓

Speaker (produces sound)

🔸 **Kernel** = traffic controller  
 🔸 **OS driver** = interpreter between app and hardware  
 🔸 **Hardware driver** = executes physical operations

## **4. Checking Drivers and Hardware**

### **(a) Check kernel messages (driver logs)**

dmesg | less

### **(b) Identify hardware chipsets**

lspci -vvv

To filter sound or audio devices:

lspci | grep -i audio

This shows the **hardware-level chipset** (e.g., Realtek ALC887, Intel HDA).

## **5. Driver Patching Workflow (Example: Sound Driver)**

### **Step 1 — Identify hardware chipset**

lspci | grep -i audio

### **Step 2 — Obtain OS-level driver (.ko or source code)**

Download source package or official driver from vendor website.

### **Step 3 — Compile from source**

./configure

make

sudo make install

This builds a .ko (kernel object) module and installs it into:

/lib/modules/<kernel\_version>/kernel/

### **Step 4 — Load driver into RAM**

modprobe <driver\_name>

### **Step 5 — Make it load automatically (persistent)**

Edit:

sudo vi /etc/modprobe.d/dist.conf

Add:

alias audio <driver\_name>

Now kernel automatically loads that module at boot using dependency list:

/lib/modules/<kernel\_version>/modules.dep

## **6. Blacklisting Drivers (Prevent Loading)**

To block unwanted or conflicting drivers:

**File:**

/etc/modprobe.d/blacklist.conf

**Example entry:**

blacklist nouveau

blacklist r8169

Blacklisted modules are skipped during kernel initialization.

## **7. Kernel Files in /boot Directory**

When a new kernel is installed, three key files appear under /boot:

| **File** | **Description** |
| --- | --- |
| **vmlinuz-**\* | The main kernel binary (compressed executable) |
| **initramfs-**\* | Initial RAM filesystem — contains temporary drivers and tools needed before mounting root filesystem |
| **config-**\* | Kernel configuration file; shows which features/drivers are built-in or modular |

### **Understanding the config File**

Each line starts with a **parameter** and ends with a flag:

| **Symbol** | **Meaning** |
| --- | --- |
| =y | Built into the kernel |
| =m | Available as a module (.ko) |
| =n | Not enabled |

If a driver shows =n, you can’t use it until you install or compile its .ko module separately.

## **8. Kernel Version and Updating**

### **Check running kernel**

uname -r

### **Install or update kernel (RHEL/CentOS/Fedora)**

yum update kernel

### **Install manually**

rpm -Uvh kernel-<version>.rpm

Linux never overwrites the existing kernel — it installs new ones side-by-side.  
 You can select the desired kernel from the GRUB boot menu.

## **9. Kernel Compilation from Source (Overview)**

Download kernel source from https://kernel.org

Extract and configure:  
  
make menuconfig

Build and install:  
  
 make

make modules\_install

make install

Reboot and verify:  
  
 uname -r

## **10. Kernel Tuning**

**Kernel tuning** means adjusting kernel parameters that control performance, memory, and system behavior.

These parameters reside in:

/proc

and

/sys

directories.

Changes here are **temporary** (stored in RAM).

### **(a) View All Tunable Parameters**

sysctl -a

### **(b) Change a parameter temporarily**

echo 60000 > /proc/sys/fs/file-max

or

sysctl -w fs.file-max=60000

### **Example 1 — File Descriptor Limit**

Check maximum open files allowed:

cat /proc/sys/fs/file-max

Change limit:

echo 600000 > /proc/sys/fs/file-max

### **Example 2 — Swapiness (Swap Memory Usage)**

Check current value:

cat /proc/sys/vm/swappiness

or

sysctl -a | grep swap

Change temporarily:

echo 60 > /proc/sys/vm/swappiness

or

sysctl -w vm.swappiness=60

### **Persist Kernel Tuning (Permanent Settings)**

Edit:

/etc/sysctl.conf

Add:

vm.swappiness=60

fs.file-max=600000

Apply changes:

sysctl -p

## **11. Kernel Tuning via /proc and /sys**

* /proc → runtime kernel data, process and memory info
* /sys → hardware and driver parameter interface  
   Both reflect **real-time values** in RAM.

Example:

cat /proc/cpuinfo

cat /proc/meminfo

cat /proc/sys/net/ipv4/ip\_forward

## **12. Kernel Types**

| **Type** | **Used By** | **Description** |
| --- | --- | --- |
| **Monolithic Kernel** | Linux | All core services (drivers, file system, memory, etc.) run in the same address space. Supports **loadable modules (.ko)** and allows **live patching** without reboot. |
| **Microkernel** | Solaris, AIX, HP-UX | Only minimal services in kernel space; everything else runs in user space. Requires frequent patching and context switches. |

## **13. Kernel Live Patching (Concept)**

In Linux (monolithic kernels), **live patching** lets you update kernel code or modules without rebooting.  
 Tools such as kpatch, ksplice, and livepatch are used for enterprise systems to fix vulnerabilities dynamically.

## **14. Inspecting Kernel Runtime Activity**

| **Command** | **Purpose** |
| --- | --- |
| lsof | Lists open files and sockets (to see kernel-level usage) |
| free -m | Shows RAM usage (may differ from open file count) |
| top or htop | Displays kernel-managed processes and memory |
| /proc files | Show kernel status in real-time |

## **15. Important Kernel & Tuning Files**

| **File / Command** | **Purpose** |
| --- | --- |
| /boot/vmlinuz-\* | Kernel binary |
| /boot/initramfs-\* | Temporary filesystem at boot |
| /boot/config-\* | Kernel configuration options |
| /lib/modules/<ver>/modules.dep | Lists all module dependencies |
| /etc/modprobe.d/ | Configuration for loading/blacklisting modules |
| /etc/sysctl.conf | Persistent kernel tuning |
| /proc/sys/\* | Runtime tunable kernel parameters |
| /sys/\* | Hardware interface and device parameters |

## **16. Key Concepts Recap**

| **Concept** | **Command / Path** | **Description** |
| --- | --- | --- |
| Check running kernel | uname -r | Show kernel version |
| Load driver | modprobe <driver> | Load .ko module into RAM |
| Blacklist driver | /etc/modprobe.d/blacklist.conf | Prevent module from loading |
| Check hardware | lspci -vvv | View hardware chipsets |
| View kernel logs | dmesg | View messages and driver load info |
| Kernel tuning | /proc, sysctl -a | Modify performance parameters |
| Persist tuning | /etc/sysctl.conf | Save permanent kernel values |
| Reapply tuning | sysctl -p | Reload kernel tuning into RAM |
| Open files info | lsof | Shows file handles managed by kernel |

## **17. Summary: Safe Kernel & Driver Management Workflow**

1. **Identify hardware** → lspci
2. **Check OS driver (.ko)** → search in /lib/modules/<ver>
3. **Compile or patch if needed** → make, make install
4. **Load driver** → modprobe <driver>
5. **Blacklist unwanted drivers** → /etc/modprobe.d/blacklist.conf
6. **Tune kernel parameters** → sysctl -a, modify /etc/sysctl.conf
7. **Verify current kernel** → uname -r
8. **Update safely** → yum update kernel or rpm -U

3️⃣ Services

AWS CC,

## **Cloud Computing**

### **🔹 What It Is**

Cloud computing = Using **internet-based servers** instead of your own hardware.  
 → You **rent computing power, storage, and software** as needed.

Before Cloud → Buy & manage servers yourself.  
 Now → Rent resources from providers (AWS, Azure, Google Cloud).

## **🏢 Inside a Data Centre (Layer by Layer)**

**1. Building Infrastructure**

* **Power:** Grid + UPS + generators (no downtime).
* **Cooling (HVAC):** Keeps 18–27°C.
* **Fire Safety:** Gas systems (no water).
* **Security:** Guards, CCTV, biometrics.

**2. IT Infrastructure**

* **Servers:** Real computers for data & apps.
* **Racks:** Hold multiple servers neatly (42U).
* **Networking:** Switches, routers, firewalls, fiber.
* **Storage:** SAN/NAS/Object storage for data.

**3. Support Systems**

* **PDUs:** Distribute power to racks.
* **Monitoring:** Tracks heat, power, humidity.
* **Access Control & NOC:** 24/7 supervision.

## **🌍 Types of Data Centres**

| **Type** | **Description** | **Example** |
| --- | --- | --- |
| **Enterprise** | Owned by one org | Banks, hospitals |
| **Colocation** | You rent rack space | Equinix, Digital Realty |
| **Cloud** | Used by cloud providers | AWS, Azure, GCP |
| **Edge** | Small, near users (low latency) | Telecom hubs |
| **Hyperscale** | Huge, global scale | Amazon, Google, Meta |

## **⚙️ Key Concepts**

**Latency vs Bandwidth**

* Latency → Speed of one message (travel time).
* Bandwidth → Data capacity per second (highway width).

## **🧱 Data Centre vs Colocation vs Cloud**

| **Concept** | **You Own** | **Provider Owns** | **You Pay For** | **Analogy** |
| --- | --- | --- | --- | --- |
| Data Centre | Servers | Building | Power, cooling | Building |
| Colocation | Servers | Building & Network | Space, power | Renting a room for your own PCs |
| Cloud | None | Everything | Virtual compute | Renting virtual computers |

👉 All clouds exist **inside** data centres.

## **☁️ Cloud Service Models**

### **1️⃣ IaaS – *Infrastructure as a Service***

* Rent **virtual hardware** (VMs, storage).
* You manage OS, app, and data.
* **Example:** AWS EC2, Azure VM.
* **Analogy:** Empty apartment — you furnish it.
* ✅ Full control | ⚠️ More maintenance.

### **2️⃣ PaaS – *Platform as a Service***

* Ready platform to **develop & deploy apps**.
* You focus on code; provider manages OS & servers.
* **Example:** AWS Elastic Beanstalk, Google App Engine.
* **Analogy:** Rented kitchen with staff.
* ✅ Fast & auto-scaling | ⚠️ Limited config.

### **3️⃣ SaaS – *Software as a Service***

* Fully functional **software online**.
* Just log in and use.
* **Example:** Gmail, Google Docs, Zoom.
* **Analogy:** Ordering food — no cooking.
* ✅ Zero setup | ⚠️ Less customization.

### **4️⃣ FaaS – *Function as a Service / Serverless***

* Upload small code functions → run only when triggered.
* Pay per execution.
* **Example:** AWS Lambda, Azure Functions.
* **Analogy:** Food truck — works only when order comes.
* ✅ No servers | ⚠️ Not for long apps.

## **📊 Who Manages What**

| **Layer** | **On-Prem** | **IaaS** | **PaaS** | **SaaS** | **FaaS** |
| --- | --- | --- | --- | --- | --- |
| Networking | You | Cloud | Cloud | Cloud | Cloud |
| Servers | You | Cloud | Cloud | Cloud | Cloud |
| OS | You | You | Cloud | Cloud | Cloud |
| App | You | You | You | Cloud | You |
| Data | You | You | You | You | You |

## **🧩 Choosing the Right Model**

| **Situation** | **Best Choice** | **Why** |
| --- | --- | --- |
| Move old servers to cloud | IaaS | Same setup, less cost |
| Build a web app | PaaS | Focus on code |
| Use email/CRM | SaaS | Ready-made |
| Run small functions | FaaS | Pay per run |

## **🧮 Cloud Stack**

User

│

├── SaaS → Full apps (Gmail, Docs)

├── PaaS → App platforms (App Engine, DB, Firewall, OS)

├── IaaS → Virtual hardware (EC2, VMs, server, Hardware, router)

└── Physical DC → Real servers, power

## **⚡ Before vs After Cloud**

| **Aspect** | **Before** | **After** |
| --- | --- | --- |
| Hardware | Buy servers | Rent instantly |
| Setup | Weeks/months | Minutes |
| Cost | CAPEX (big upfront) | OPEX (pay-as-you-go) |
| Scaling | Manual | Auto-scale |
| Maintenance | You manage | Provider handles |
| Backup | Manual | Built-in |
| Access | Local | Global |
| Security | Your duty | Shared responsibility |
| Speed | Slow to innovate | Fast & flexible |

## **AWS Global Infrastructure (Summary)**

### **🌍 Region**

* A **region** is a **geographic area** where AWS has multiple data centres.
* Each region is **independent** for data storage, compliance, and latency.
* Example: us-east-1 (N. Virginia), ap-south-1 (Mumbai), eu-central-1 (Frankfurt).

### **🧩 Availability Zone (AZ)**

* Each region has **2–6 Availability Zones**.
* Each AZ = one or more **data centres** with separate power, cooling, and network.
* Data is **replicated** across AZs for backup & fault tolerance.
* Zones are **physically separated** but **connected with low-latency links**.
* Purpose: high availability + disaster recovery.
* Example: ap-south-1a, ap-south-1b, ap-south-1c.

### **🏢 Data Centre**

* The **physical buildings** where AWS servers run.
* Each AZ contains one or more data centres.

### **🌐 Global Infrastructure**

**Region + Availability Zones = AWS Global Infrastructure** ➡️ Provides **redundancy**, **scalability**, and **disaster recovery** worldwide.

Tab 56

Tab 57

Tab 58

Tab 59