Linux+

**Topics:**  
 1- Filesystem Architecture

2- System Architecture and boot process

3- Managing filesystem, partition and Disks

4- User and group management

5- Process and service management

6- Network Management

7- Security and Access Management

8- Trouble shooting and system maintenance

9- Virtualization and Containerization

**Practical part:**

1- Basics Linux Commands

2- Systemd  
3-Managing filesystem, partition and Disks

4- User and group management

5- Process and Service Management

6- Network Management

7- Security and Access Management

8- Trouble shooting and system maintenance

9- Virtualization and Containerization

**Bash scripting**

--------

1-Filesystem Architecture

* 1. Filesystem Hierarchy Standards (FHS):

FHS - defines the linux directory structure .

Here are some of them:

* **/ (Root):** The root directory.
* **/bin:** Contains essential command line tools (ls,cp,mv)

Can be used by all users.

* **/sbin:** Contains system administration tools like network

configuration and disk management tools

(ifconfig,fdisk).

* **/boot:** Contains linux linux kernal boot process files.
* **/dev:** Contains special files that represents hardware.

devices (USB, printers, hard disks).

* **/etc:** Contains system-wide configurations files.
* **/tmp:** Contains temporary files for the programs.
* **/home:** Contains the personal directory and files for each

user .

* **/lib:** Contains essential system libraries used by binaries.
* **/lib64:** Contains 64-system libraries.
* **/media:** Contains directories for mounted media (USB).
* **/mnt:** Temporary mount point for manually mounted

filesystems by the administrator.

* **/opt:** Contains third party apps (not installed by the

package manager of the system apt or dnf).

* **/proc:** Virtual filesystem provide information about. running processes and the system.
* **/root:** The home directory for the root user.
* **/run:** Store temporary system information like process ID.
* **/srv:** contains data for system services.
* **/sys:** Virtual filesystem provides information about.

hardware and drivers.

* **/usr:** contains user installed programs libraries and.

Documents it often be larger than root.

* **/var:** store files that change frequently like logs files.
  1. Filesystem Types

These are the most common filesystems:

* Ext family:
* EXT2
* EXT3
* EXT4: this version of the EXT filesystem type has the following features:
* Journaling filesystem for linux.
* Designed as the successor to EXT3.
* XFS (x filesystem): Journaling filesystem and it has high performance scalability and availability.
* NTFS (New Technology Filesystem): It’s the primary filesystem for modern windows operation system
* FAT32
* Swap

2- system Architecture and boot process

2.1- Basic system Architecture

**Kernal:** The core component of the linux OS, it acts like a bridge between the hardware and the software layer.

**Kernal Roles:**

* **Memory Management:** Allocate memory to process as needed as well as deallocate.
* **Process Management:** Create schedule and terminate processes.
* **Device Management:** control the access to the hardware devices like disk drive network card and printers.
* **Filesystem Management:** Manage the filesystem and provide access to files and directories**.**
* **Hardware Abstraction layer (HAL):**  Provides a consistent interface for the software to interact with the hardware allowing different of software to interact with hardware without any modification on the software.
* **System Calls:** Provide set of functions that allow user level programs to integrate with the kernal wand request system services.
* **Security:** The kernel protects the system by controlling access to hardware, memory, and files.

It ensures that processes, users, and programs cannot interfere with each other without permission.

**Shell:** command lie interpreter that allows that allow user to interact with the computer system using commands   
  
**User Space**: It’s the environment wher the user level programs are being executed.

2.2- Boot process

BIOS/UEFI:

* **BIOS (Basic Input Output system):** A firmware interface used in older systems to initialize hardware components, perform a Power-On Self-Test (POST), and pass control to the bootloader to start the operating system.
* **UEFI (Unified Extensible Firmware Interface):** A modern replacement for BIOS that initializes hardware and loads the bootloader with additional features like secure boot, larger disk support, and a graphical interface.

Bootloaders:

GRUP (Grand Unified Bootloader): The most common bootloader in Linux systems. It takes control from BIOS or UEFI after the initial hardware initialization and loads the operating system kernel. Init Systems

* **sysVinit:** Is the traditional initialization system used in many old linux distros to start the system’s services and processes during the boot processes as well as manage and terminate them.
* **Systemd:** Is the sophisticated init system designed to replace the traditional sysvinit.

Systemd work with something called units

**Units:** in systemd are resources that it’s able to manage

Including: services, timers, mount, automount and more

systemd uses unit files to understand how to interact wit processes and services.

They are stored in three different locations and they are sorted from the highest priority to the least:

1. /etc/systemd/system
2. /run/systemd/system
3. /lib/systemd/system

Types of system unit files:

**1. Service Unit (.service)**

* **Purpose:** Defines a **background service or daemon**.
* **Example:** Web servers (apache2.service), database servers (mysql.service).
* **Simple Explanation:**  
  Used to start, stop, reload, or restart a service.

**2. Socket Unit (.socket)**

* **Purpose:** Defines a **socket** that can trigger a service.
* **Example:** cups.socket for the CUPS printing system.
* **Simple Explanation:**  
  Listens on a network or IPC socket and can start a service when needed.

**3. Target Unit (.target)**

* **Purpose:** Used to **group units together**.
* **Example:** multi-user.target for multi-user text mode.
* **Simple Explanation:**  
  Like a milestone or checkpoint; it collects other units to reach a certain system state.

**4. Device Unit (.device)**

* **Purpose:** Represents a **kernel device**.
* **Example:** Automatically created for devices like /dev/sda1.
* **Simple Explanation:**  
  Used for devices; can trigger services when a device becomes available.

**5. Mount Unit (.mount)**

* **Purpose:** Describes a **file system mount point**.
* **Example:** home.mount for /home.
* **Simple Explanation:**  
  Automatically mounts filesystems at boot or on demand.

**6. Automount Unit (.automount)**

* **Purpose:** Controls **on-demand mounting** of file systems.
* **Example:** home.automount for automatically mounting /home.
* **Simple Explanation:**  
  Delays the actual mounting until the path is accessed (lazy mounting).

**7. Timer Unit (.timer)**

* **Purpose:** Schedules when a **service unit** should be triggered.
* **Example:** logrotate.timer for periodic log rotation.
* **Simple Explanation:**  
  Like cron, but managed by systemd.

**8. Swap Unit (.swap)**

* **Purpose:** Describes a **swap space**.
* **Example:** dev-sda2.swap.
* **Simple Explanation:**  
  Manages swap partitions or swap files.

**9. Path Unit (.path)**

* **Purpose:** Watches for **changes in the filesystem**.
* **Example:** var-log-path.path watching /var/log.
* **Simple Explanation:**  
  Triggers a service when a file or directory changes.

**10. Slice Unit (.slice)**

* **Purpose:** Organizes **cgroups (control groups)** for resource management.
* **Example:** user.slice, system.slice.
* **Simple Explanation:**  
  Groups processes to control CPU, memory, I/O usage.

**11. Scope Unit (.scope)**

* **Purpose:** Manages **external processes** not started by systemd itself.
* **Example:** Processes started via systemd-run.
* **Simple Explanation:**  
  Used for transient or manually started processes outside normal services.

What is inside a unit file:

* 1. **[Unit] Section**
* **Purpose:** General description and dependencies.
* **Common Options:**
  + **Description**: A short description of the unit.
  + **After**: Specifies which units this unit should start after.
  + **Requires**: Specifies required dependencies (if this fails, the unit fails).

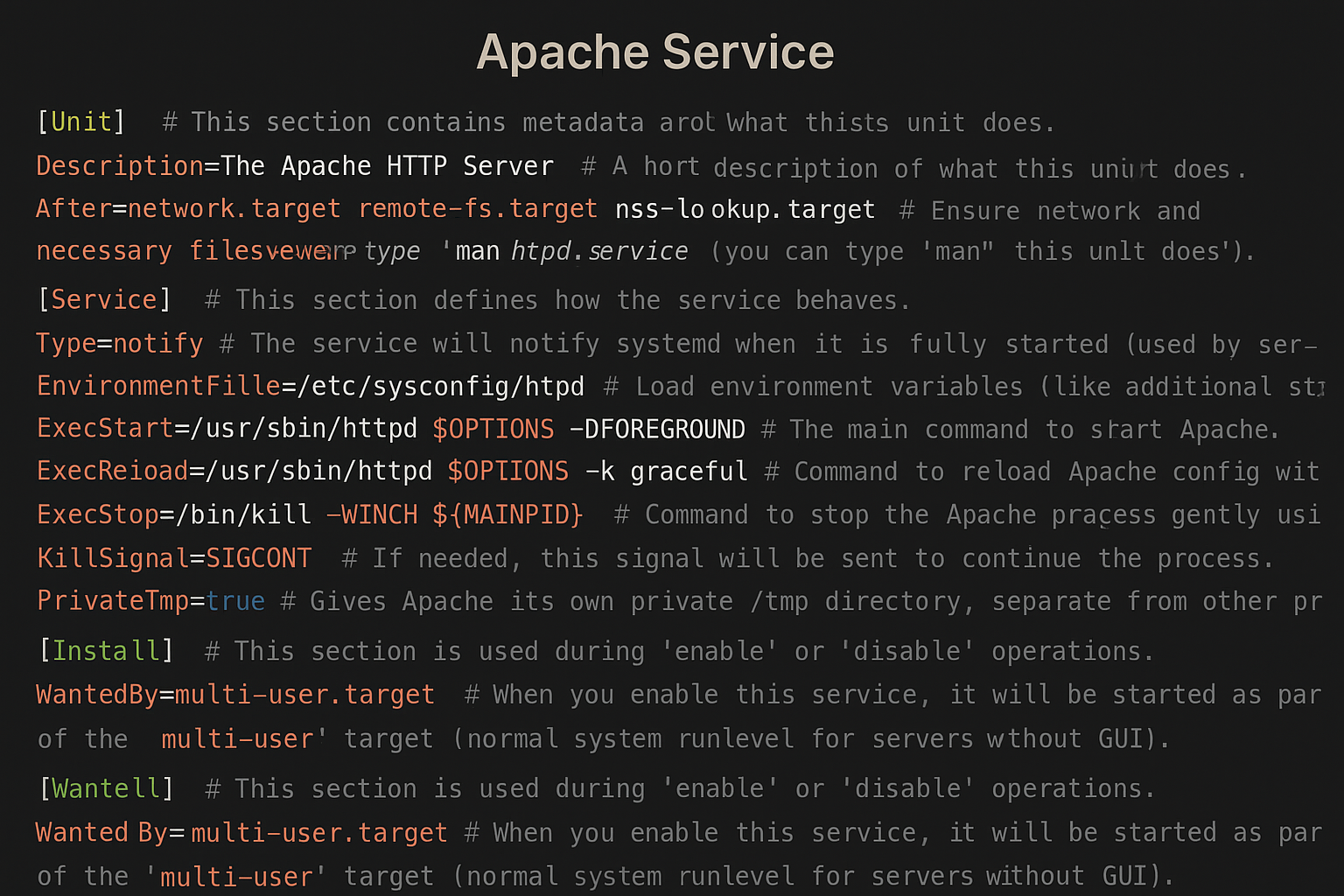
**2. [Service] Section (for .service units)**

* **Purpose:** Defines how the service behaves.
* **Common Options:**
  + **ExecStart**: Command to start the service.
  + **ExecStop**: Command to stop it.
  + **Restart**: What to do if the service fails (like always, on-failure).
  + **User**: Which user should run the service.

**3. [Install] Section (optional)**

* **Purpose:** Used when enabling/disabling the unit (with systemctl enable/disable).
* **Common Options:**
  + **WantedBy**= : Targets that will pull in this unit when enabled.

example:



common system commands:

**1. systemctl:** Controls system services and units (start, stop, enable, disable services).

**2. journalctl:** Views system logs (logs from all services and the system).

**3. hostnamectl:** Shows or sets the system’s hostname.

**4. timedatectl:** Controls system time, date, timezone, and NTP settings.

**5. localectl:** Manages system locale and keyboard layout settings.

**6. loginctl:** Manages user logins and seats (multi-user, graphical sessions).

**7. busctl:** Communicates directly with the D-Bus message bus (advanced, rarely used manually).

**8. systemd-analyze:** Shows performance info like boot-up time and critical chain analysis.

**9. systemd-cgls:** Displays control groups (cgroups) in a tree view.

**10. systemd-cgtop:** Live display of control group resource usage (like top but for cgroups).

**11. systemd-nspawn:** Launch lightweight containers (like a mini virtual machine).

**12. machinectl:** Manages and interacts with virtual machines and containers.

**13. systemd-run:** Run a command in a temporary scope or service.

**14. systemd-resolve (deprecated, replaced by resolvectl)**

Queries DNS resolution information.

**15. systemd-escape:** Escapes strings to make them suitable as systemd unit names.

\*\*every command is fully explained in the practical part\*\*

2.3 Managing boot Run levels Targets and Startup Processes

**Run levels in sysvinit**: Sysvinit organizes the boot processes in different run levels

0. Halt the the system

1. Single user mode
2. Multi user mode without NFS
3. Full multi user mode without GUI
4. Unused
5. Multi user mode with GUI and NFS
6. Reboot the system

**Boot Targets in system:** powerful mechanism for managing

the system state, they represent groups of services that

should be started or stopped together.

* **Multi user target:** most common state for multi user env.
* **Graphical target:** starts the graphical interface like the display.
* **Rescue target:** services for system recovery.
* **Emergency target:** only critical services for system maintenance.

Managing Startup Processes

**Sysvinit:** the sysvinit relies on set of scripts in the /etc/init.d to manage the startup and shutdown of system services for example (apache2,mysql) each script contains instructions for starting and stopping the corresponding service.

**Systmd**: it relies on the command systemctl to control and manage services

Usage:

sudo systemctl [option] [services]

Service field could be any service like docker appachi nginx or ssh

option filed could be 7 states

* enable: to make it start once the system run
* disable: to not make it start when the system run
* start: to start the service
* stop: to stop the service
* status: to see the status of the service
* restart: to restart the service stop the service and start it
* reload: Reloads only the configuration without stopping the service. The process stays running.

3- Managing partitions, filesystem and Disks

* 1. partitions

**partitions:** Logical division of physical storage.

**Types of partitions in linux OS:**

**1. Primary Partition**

* **Description**: The main type of partition that can contain data or an OS.
* **Limit**: You can have a maximum of **4 primary partitions** on a disk.
* **Usage**: Usually used for bootable systems (like /, /boot).

**2. Extended Partition**

* **Description**: A special partition type that acts as a container to hold multiple **logical partitions**.
* **Limit**: You can only have **one extended partition per disk**.
* **Usage**: Used when you need **more than 4 partitions** on a disk.

**3. Logical Partition**

* **Description**: Partitions created **inside an extended partition**.
* **Limit**: No hard limit like primary; usually **up to 128 logical partitions**.
* **Usage**: Used to organize data when you run out of primary partitions.

**4. Swap Partition**

* **Description**: Special partition for **swap space** (virtual memory).
* **Purpose**: Acts like RAM when the physical memory is full.
* **Usage**: Essential in most Linux installations for memory management.

**Common partitions in linux:**

**1. / (Root Partition)**

* **Description:** The top-level directory for the whole Linux filesystem.

**2. /boot**

* **Description:** Contains the files needed to boot the system (kernel, GRUB files, etc.).
* **Mandatory:** Recommended for easier boot management, especially in dual-boot setups.

**3. /home**

* **Description:** Stores user personal files and configs.
* **8. Swap**
* **Description:** Virtual memory when RAM is full.
* **Recommended Size:** Equal to or half of RAM (depends on system use).
* **Mandatory:** Highly recommended, especially on low-RAM systems.

**Disk Partitioning Formats:**

**1. MBR (Master Boot Record)**

* Old standard, Stores partition information in the first 512 bytes of the disk.
* Allows up to 4 primary partitions only, Maximum disk size: 2 TB.
* Used with BIOS systems.
* If you need more than 4 partitions, you must create an Extended Partition, which can hold additional Logical Partitions inside.

**Best for:** Very old systems or disks smaller than 2TB.

**2. GPT (GUID Partition Table)**

* Newer standard, replaces MBR, Part of the UEFI system.
* Supports disks larger than 2 TB.
* Can have up to 128 partitions, each partition has a unique identifier (GUID) for better management.
* More reliable—it stores multiple copies of the partition table for backup and recovery.

**Best for:** Modern systems, large disks, servers, and virtual machines.

**3. G — SGI (Silicon Graphics IRIX)**

* Special format for **SGI IRIX workstations/servers** (an old UNIX variant).
* Rare—almost no one uses this unless maintaining vintage SGI systems.
* Not usable for Linux/Windows or normal servers.

Used in: Silicon Graphics IRIX computers only.

Useless for modern Linux, Windows.

**4. s — SUN (Solaris/SPARC)**

* Used for **Sun Microsystems (SPARC architecture)** systems.
* Needed if the disk will be used on Solaris or older Sun servers.
* Not compatible with normal Linux PCs.

Used in: Old Solaris OS, SPARC servers.

Not for normal x86 Linux/Windows.

**Partition management commands:**

* 1. **Partition creation:**
* **fdisk**
* **parted**
* **gparted**
* **gdisk**
  1. **Partition Viewing and Monitoring commands:**
* **lsblk**
* **blkid**
  1. **Swap Management commands:**
* **mkswap**
* **swapon**
* **swapoff**
* **fallocate**

**\*\*all of them explained fully and deeply in the practical part\*\***

**File systems:**

We have gone through the filesystem types previously in

(2.1 filesystem architecture) so we will hump directly into

**filesystem management commands:**

* 1. **mkfs (Make File System):** Format a partition with a specific file system type.

**2. fsck (File System Consistency Check):** Check and repair file system errors.

**3. blkid :** Shows the UUID and type of the file system on a device.

**4. tune2fs (For ext2/ext3/ext4 file systems only):** Adjust tunable file system parameters.

**5. resize2fs (For ext2/ext3/ext4 only):** Resize (expand or shrink) an ext filesystem.

**6. xfs\_growfs (For XFS file systems only):** Grow an XFS file system to use additional space.

**7. mount / umount**: Mount or unmount file systems to/from directories.

**\*\*all of them explained fully and deeply in the practical part\*\***

**Disk Management commands:**

Most of the commands to manage the disk was mentioned earlier and here is the rest of command to manage and monitor the disk

* **df:** Shows the amount of **disk space used and available** on file systems.
* **du:** Shows the size of files and directories.

4- User and group management

**4.1- Users And Groups:**

**User:**  A user can be a person, a service, or the root of the system. Each user has its own permissions and file ownership inside the system.

**User Account:** represents an entity (a person or a service) that can log into the system or execute processes. Each user account has a unique User ID (UID) and is associated with specific permissions and access rights that determine what the user can do on the system.

**A user can:**

* Log into the system (if they have a shell and password).
* Own files and processes.
* Run commands and applications.
* Have permissions to access system resources (files, directories, devices).

**Why Do We Need Multiple Users in a Linux System?**

Linux is a **multi-user operating system**, meaning that it is designed to support multiple users at the same time. This is essential for several reasons:

**1. Security:** User isolation ensures that one user cannot access, modify, or delete another user’s files unless given explicit permission, prevents unauthorized access to sensitive data, Reduces the risk of accidental or malicious system damage.

**2. Resource Management:** Each user can be assigned specific disk quotas, CPU usage limits, or memory usage restrictions, prevents a single user or process from consuming all system resources, ensuring fair usage for everyone.

**3. Accountability:** Activities on the system can be tracked and logged per user, makes it easier to identify who made changes, ran certain commands, or accessed sensitive areas.

**4. Flexibility for Services and Applications:** System services (like web servers, databases) run under their own **dedicated service accounts**.

**5. Collaboration:** On shared systems (like servers or development environments), multiple users can work on the same machine while keeping their data **separate and private**.

**6. Maintainability:** Easier to manage system updates, permissions, and backups when each user has a distinct account and environment, avoids conflicts and confusion caused by shared or generic accounts.

1**. Root User (UID 0):** The superuser account that has unlimited privileges, can perform any administrative task:

* + Install/remove software.
  + Create/delete user accounts.
  + Access any file regardless of permissions.

Username: usually root, UID: always 0.

2. **Regular Users (UID ≥ 1000):** These are human users, like you and me, created for people who will use the system for daily tasks, have limited permissions:

* + Can only access their own files and certain shared resources.
  + Need sudo or su to perform administrative tasks.

3. System / Service Users (UID < 1000): Created automatically by the system or software packages, sed to run system services (daemons) like web servers, databases, and background processes, These users are not intended for human login.

* Examples:
  + nobody (UID 65534 — special case).
  + www-data (used by web servers like Apache or Nginx).
  + mysql (used by MySQL server).

**Group:** A group in Linux is a collection of user accounts. Groups are used to manage permissions collectively for a set of users, making it easier to control access to files, directories, and system resources.

**Purpose of Groups:**

1. **Simplified Permission Management:**
   * Allows administrators to set access rights for a group rather than for individual users.
   * Makes managing large numbers of users easier, especially in multi-user systems.
2. **Collaboration:**
   * Users who belong to the same group can share files and directories with controlled access.
   * Useful in environments like development teams, departments, or project groups.
3. **Security:**
   * Limits access to certain resources only to users who are members of specific groups.
   * Reduces the chance of unauthorized access.

**Types of groups:**

**1. Primary Group:**

* Each user has exactly one primary group.
* This group is automatically assigned when the user is created.
* By default, any new files or directories that the user creates are associated with this primary group.
* The primary group is defined in the /etc/passwd file for each user.

**2. Secondary (Supplementary) Groups:**

* A user can belong to one or more secondary groups in addition to their primary group.
* Secondary groups provide additional access rights to files and resources shared among group members.
* These memberships are listed in the /etc/group file.

**What is GID (Group ID) in Linux?**

* **GID** stands for **Group Identifier**.
* It is a **unique number assigned to each group** in the Linux system.
* Just like every user has a **UID (User ID)**, every group has a **GID**.

**4.2- Linux User & Group Related Files:**

1. /etc/passwd
2. /etc/shadow
3. /etc/group
4. /etc/gshadow

1- /etc/passwd: file is a critical system file that stores essential information about every user account on the Linux system.

* It is a **plain text file**, readable by all users.
* Each **line represents a single user account**.
* Fields are separated by colons (:).

**Structure of a Line in /etc/passwd:**

****

****

**username**: This is the login name of the user.

**password**: This field usually contains an x, which means the encrypted password is stored securely in the /etc/shadow file.

**UID (User ID)**: A unique number assigned to the user. The system uses this number (not the username) to identify the user. Example: 1001.

**GID (Group ID)**: The primary group ID associated with the user. This links to an entry in the /etc/group file.

**comment (GECOS field):** optional information about the user

**home\_directory**: The absolute path to the user's home directory.

**shell**: The absolute path to the user's default shell program.

2- /etc/shadow: Contains the hashed (encrypted) passwords and settings related to password aging and expiration policies.



**username**: The user’s login name (same as in /etc/passwd).

**Password**: The hashed (encrypted) user password.  
If this field contains:

* + A string of hash characters: The actual password hash.
  + ! or \*: Account is **locked** (cannot be used to log in).
  + Empty (""): No password set

**lastchg (Last Change)**: The number of days since Jan 1, 1970 (epoch) when the password was last changed.

**min (Minimum Age)**: the minimum number of days before the user can change the password again.

**max (Maximum Age)**: The maximum number of days the password is valid before the system requires it to be changed.

**warn (Warning Period)**: The number of days before password expiration to warn the user to change it.

**inactive (Inactive Period)**: The number of days after a password expires before the account is disabled.

**expire (Account Expiration Date)**: the number of days since Jan 1, 1970 when the account will be disabled.  
Empty means the account never expires.

**Reserved**: Reserved for future use — normally left empty.

**3- /etc/group:** file stores information about groups on the Linux.

* Each line represents one group.
* This file defines group names, GIDs, and the list of users who are members of that group.
* Used by the system to determine group membership and permissions.





**group\_name**: The name of the group. Example: devs.

**password**  
Usually contains an x, indicating the group password (if any) is stored in /etc/gshadow.  
Group passwords are rarely used.

**GID (Group ID)**; A unique number that identifies the group in the system. Example: 1002.

**member\_list**: A comma-separated list of users who belong to this group as secondary (supplementary) members.  
This field can be empty if no additional users are members.

**/etc/gshadow:** file stores secure and sensitive information about groups in Linux, this includes encrypted group passwords, group administrators and the list of group members.





**group\_name:** The name of the group — same as in /etc/group. Example: developers.

**password**: The encrypted group password (if set).  
Special values:

* ! — means the group cannot be accessed via password.
* Empty field — means no password is set (the default and normal for most groups).

**group\_admins**: A comma-separated list of users who are group administrators — they can change the group password or membership using commands like gpasswd.

**group\_members**: A comma-separated list of users who are members of this group (same as the member list in /etc/group).

**4.3- Users and groups management**

**User:**

* **User Creation: useradd, adduser.**
* **User Management: usermod** **,** **chfn,** **id,** **chage.**
* **User Password Management: passwd, chage.**
* **User Deletion: userdel.**

**group:**

* **Group Creation: groupadd.**
* **Group Management: groupmod, gpasswd.**
* **Group Password Management: gpasswd.**
* **Group Deletion: groupdel.**

**4.3- File Ownership and Permissions:**

**Owner, Group, and Others**

Every file and directory in Linux is associated with three categories of users who have different access rights:

1. **Owner**: The user who owns the file. Usually, this is the user who created the file, but ownership can be changed. The owner has specific permissions for the file.
2. **Group**: The group assigned to the file. Any user who is a member of this group has group-level permissions on the file.
3. **Others**: All other users on the system who are neither the owner nor members of the group.

**The Relationship Between Users, Groups, and File Permissions**

Linux uses file permissions to control what each of these categories can do with a file or directory. These permissions are split into three types:

* **Read (r)** — permission to read the file or list the directory contents.
* **Write (w)** — permission to modify the file or create/delete files in the directory.
* **Execute (x)** — permission to run the file as a program or access a directory.

**Files And Directories Permissions Format:**

****

* The first character shows the file type (- for regular file, d for directory).
* The next three characters (rwx) are permissions for the owner.
* The next three (r-x) are permissions for the group.
* The last three (r--) are permissions for others.

**Files and Directories ownership and permissions management**

* **chown — changes file owner and optionally group.**
* **chgrp — changes group ownership only.**
* **chmod — modifies file or directory permissions.**
* **umask — sets default permission mask for new files and directories.**
* **getfacl — displays Access Control Lists (ACLs) for files.**
* **setfacl — modifies ACL entries to add or remove permissions.**

**Special permission bits set via chmod:**

* **sticky bit (+t) —** restricts file deletion within directories.
* **SUID (u+s) —** runs executable with file owner’s permissions.
* **SGID (g+s) —** runs executable with group permissions or makes new files inherit group ownership.

**Managing sudo permissions:** Sudo allows normal users to execute commands with root privileges, Sudo permissions are configured in the /etc/sudoers file or in files inside /etc/sudoers.d/.  
Only root can manage who gets sudo access.

**visudo:** Safely edits the /etc/sudoers file to configure sudo permissions. This command checks for syntax errors to avoid breaking sudo access.

**echo 'username ALL=(ALL) ALL'** >> /etc/sudoers — Directly adds a user to the sudoers file (not recommended; better to use visudo to prevent mistakes).

**sudo -l -U username —** Lists the sudo privileges available to the specified user.

**sudo groupadd sudo —** Creates a "sudo" group if it doesn't already exist (some distros use "wheel" instead of "sudo").

sudoers file syntax example:  
**username ALL=(ALL) ALL —** Allows the user to execute any command as any user on the system.

**%groupname ALL=(ALL) ALL —** Allows all members of the specified group to use sudo.

**4.4-Access Control List (ACL):** is an advanced permission mechanism in Linux that allows you to set specific permissions for individual users or groups, beyond the traditional owner-group-others model.

**Why Use ACL?**

* When standard Linux permissions are not enough
* To apply fine-grained permissions without changing file ownership or primary group.
* Useful in multi-user environments where file access needs are more complex.

**When ACL Is Needed:**

* You want to give read-only access to a specific user who is not the file’s owner or in the group.
* You want multiple users from different groups to access the same file without changing ownership or group settings.

**Key Concepts:**

* **Default ACL:** Applied automatically to new files/directories inside a directory.
* **Access ACL:** Applied directly to files/directories for specific users or groups.
* ACL allows specifying permissions like **rwx** for any user or group, not just the file's owner or primary group.

**ACL Commands:**

**getfacl** AND **setfacl**

5- Process and service management

**5.1- Process:** A process is an instance of a program that is being executed. It includes the program code and its current activity, represented by attributes such as the process ID (PID), memory usage, open files, and CPU time.  
Every command or application executed on a Linux system runs as a process.

**Key Point:** Processes are the basic unit of execution in Linux. Each process is isolated and managed by the kernel.

* **Daemon:** A daemon is a background process that is designed to run continuously without direct user interaction.  
  Daemons typically start during system boot and remain active to perform system-level tasks such as monitoring, scheduling, logging, or responding to network requests**.**
* Most daemon names end with the letter "d" (e.g., sshd, httpd, systemd).
* They are detached from the terminal and often run with root privileges.

**Key Point:** A daemon is a specialized process, usually running in the background and managed by the init system.

* **Service:** A service in Linux is a managed unit that represents a daemon or any other long-running process.  
  It is controlled by the system's init system (such as systemd) and can be started, stopped, enabled, or disabled based on system requirements.
* Services are defined by unit files in modern systems (e.g., nginx.service).
* A service is not always a daemon, but most services are daemons.

**Key Point**: Services are abstract representations of daemons, managed by init systems for consistent behavior and control.

**5.1- process:**

1. **Foreground Process:** is one that interacts directly with the user through the terminal or shell. It runs in the foreground and occupies the terminal session until it completes or is terminated.

* The user must wait for the process to finish before running another command (unless multi-terminal or backgrounding is used).
* Launched normally via a command.

1. **background process:** is a process that runs without blocking the terminal. It allows the user to continue using the shell while the process executes.

* Typically launched with an ampersand & at the end.
* Background processes can still output to the terminal unless redirected.

1. **interactive process** is initiated and controlled directly by the user via a shell or graphical interface like top. These are typically foreground processes but not always.

* Requires user input or interaction during execution.
* Can be paused with Ctrl+Z or terminated with Ctrl+C

1. **batch process** is scheduled to run at a later time without user interaction. It's often handled by a job scheduler like cron, at, or batch.

* Runs based on predefined rules or schedules.
* Commonly used for backups, updates, reports, etc.

1. **Daemon Process**

As defined earlier, a **daemon** is a long-running background process that starts at boot or on demand, and does not require user interaction. Most system-level tasks (like networking, logging) are handled by daemons.

* Managed by systemd, init, or other init systems.
* Runs under service names, often ends in d (e.g., sshd).

1. **Zombie process:** is a process that has completed execution but still has an entry in the process table. This happens when the parent process hasn’t yet read its exit status via wait().

* Takes up a PID but uses no CPU or memory.
* Can indicate a bug or mismanagement by the parent.

1. **Orphan process** is a process whose parent has terminated before it did. Orphans are automatically adopted by the init process (PID 1), which handles their cleanup.

**Process Lifecycle in Linux:**

**1. Process Creation:** Processes in Linux are created using a combination of the fork(), exec() and related system calls:

* fork(): Creates a child process by duplicating the parent process. Both processes continue executing.
* exec(): Replaces the current process memory with a new program.

Together, fork() followed by exec() is how most new processes start in Linux.

**Example:** When you type a command like ls, the shell:

* forks itself
* replaces the child process with the ls binary using exec().

**Process States:** Linux tracks every process using its state. These states help the scheduler and admin tools understand what a process is doing.

Here are the main process states:

**R (Running Actively):** running or ready to run

**S (Sleeping Waiting):** for an event (e.g. disk I/O)

**D (Uninterruptible Sleep):** Waiting for hardware, can’t be killed

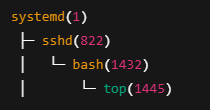
**T (Stopped Suspended):** (via signal or job control)

**Z (Zombie):** Completed execution, waiting for parent cleanup

**X (Dead (rare)):** Defunct process (very uncommon)

**3. Process Hierarchy (Parent & Child)**

* Every process (except init) has a parent process.
* The Linux kernel assigns a unique Process ID (PID) and tracks the Parent Process ID (PPID).

You can see this tree-like hierarchy with:

$ pstree -p

**4. Process Termination**

A process can terminate in one of three ways:

1. **Normal Exit**: The process completes successfully (calls exit() or returns from main()).
2. **Signal Exit**: The process is killed or terminated via signals (SIGKILL, SIGTERM, etc).
3. **Error/Crash**: The process crashes due to a segmentation fault or unhandled exception.

When a process ends, it sends a **termination status** to its parent.

If the parent process fails to collect this status, the child becomes a **zombie**.

**Process Monitoring and Management:**

**Monitoring Commands:**

* **ps**
* **top**
* **htop**
* **pstree**
* **pidstat**
* **iotop**

**Process Management:**

* **kill**
* **killall**
* **pkill**
* **nice**
* **renice**
* **trap**

**5.2- Service:  
service** is a long-running background process that is usually started at boot time and performs specific functions without direct user interaction.

* Most services are daemons (e.g., sshd, nginx, cron).
* Services are managed by the **init system**, which handles startup, shutdown, and monitoring.

**Key Concept:** A service = a daemon + management layer by systemd or init.

**Service States in system:**

**Active:** (running) Service is currently running

**Inactive:** Service is not running

**Failed:** Service encountered an error and stopped

**Enabled:** Will start at boot

**Disabled:** Will not start at boot

\*\*Services are managed by systemctl tool pack and it will be explained deeply in the practical part\*\*

**5.3- Job scheduling:**

Linux refers to the ability to automatically run tasks (commands or scripts) at specific times or intervals without manual intervention.

* Performing backups
* Cleaning log files
* Sending emails
* Updating systems
* Monitoring services

**1. cron – Recurring Jobs:** is used to schedule tasks that run repeatedly at fixed times, dates, or intervals (e.g., every day, every hour).

* Jobs are defined in **crontab files**.

Example use: Run a script every day at 3:00 AM

**2. at – One-Time Jobs:**  is used to schedule a job to run **once** at a specific future time, t is ideal for tasks that need to run later but only once (e.g., shutdown at midnight).

Example use: Schedule a one-time reboot at 2:00 AM

**3. anacron – Catch-Up for Missed Jobs:** is designed for systems that don’t run 24/7 (like laptops).**:** It ensures that scheduled tasks are not missed if the system was off at the scheduled time.

* It runs tasks **once a day**, **weekly**, or **monthly** with a delay after system boot.

Example use: Run daily maintenance tasks even if the system was off during the original schedule.

6- Network Management

7- Security and Access Management

**7.1- User Authentication Mechanisms:**

User authentication in Linux is not a simple username-password check — it's a modular, flexible system built around PAM. This structure allows administrators to enforce security policies, such as password complexity, account lockout after failed attempts, and even multi-factor authentication.

Failing to understand how authentication works in Linux can lead to dangerous misconfigurations — including locking out root, or leaving the system wide open to brute-force attacks.

**How Authentication Works (Flow)**

Here’s a simplified **login process flow** when a user attempts to log in (via TTY or SSH):

1. Login program (like login, sshd, gdm, etc.) is invoked.
2. It uses PAM to check credentials (via /etc/pam.d/ configs).
3. PAM executes a **series of modules** in order (the "PAM stack").
4. If all modules succeed → access is granted → shell is launched.
5. If any critical module fails → login is denied.

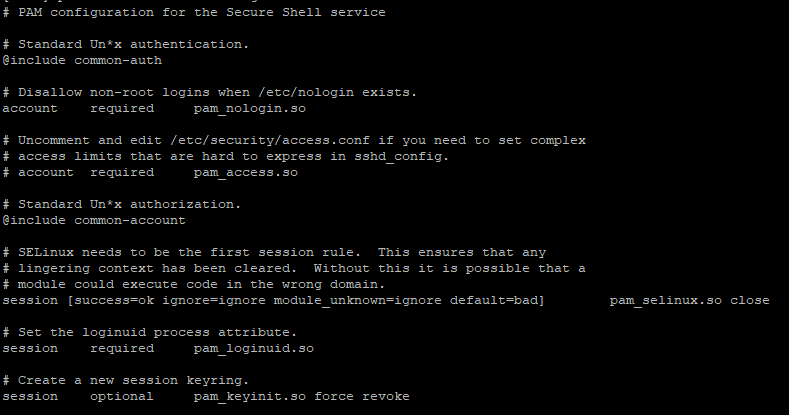
**PAM configuration files are located in:**

/etc/pam.d/

Each service (like sshd, login, su, etc.) has its own file, e.g.:

* /etc/pam.d/sshd
* /etc/pam.d/sudo

Each file consists of lines like:



Each file consists of lines like:



**Types:** Defines which phase of the authentication process the line applies to:

* **auth:** Verifies the user (password, tokens, etc.)
* **account:** Checks account validity (e.g., expired?)
* **password:** Handles password updates
* **session:** Manages session tasks (mount home dir, log actions)

**Control Flags:** Tells PAM how to handle success/failure of the module:

* **required:** Must succeed, but failure doesn’t stop immediately.
* **requisite:** Must succeed, and failure stops authentication.
* **sufficient:** If successful, no further modules are needed.
* **optional:** Not critical to success.

**module-path:** This is the absolute path to the PAM module being used. These modules are shared libraries (.so files) stored in

**arguments:** These are options passed to the module to change its behavior. They vary per module — just like function parameters in code.

**Example:**



**Important PAM Modules:**

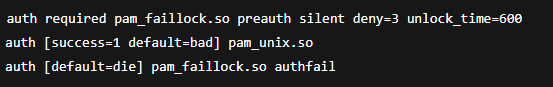
**pam\_unix.so:** Standard authentication using /etc/shadow

**pam\_pwquality.so:** Enforces password complexity

**pam\_tally2.so or pam\_faillock.so:** Tracks failed login attempts

**pam\_permit.so, pam\_deny.so:** Always allow / always deny (for testing)

Example snippet from /etc/pam.d/common-auth:



**User Authentication Methods:**

**1. Password Authentication:** Users log in by typing a username and password. Passwords are stored securely (hashed) in /etc/shadow.

**2. SSH Key Authentication:** Users use a pair of keys (public/private) instead of a password for secure remote login.

**3. Two-Factor Authentication (2FA):** Adds an extra step like a code from a phone app to the password for better security.

**7.2- Filesystem security:**

**chroot** (**change root):** It changes the apparent root directory / for a running process, isolating it from the rest of the system.

So, for a process inside a chroot environment:

/ = /some/custom/directory

**Why Use chroot?**

* Recovery Boot into live system and chroot into broken system to fix it
* Jailed services (e.g., SFTP) Restrict users to a specific directory
* Testing Test builds or software in isolated root
* Legacy isolation Used before containers were mainstream

**Limitations of chroot:**

* Not full isolation No kernel or namespace separation
* Can be broken by root Root user inside chroot can escape
* Needs full environment setup You must manually copy binaries, libs, config files
* No resource control No way to limit memory/CPU/IO usage

**7.3- Access Control:**

Access control mechanisms define who can do what on a system. Understanding these models is essential for building secure environments. There are four primary types:

**Types of Access control:**

**1. DAC (Discretionary Access Control):** is the default model used in most Linux and Unix systems. In this model, the owner of a file or resource decides who can access it and how.

How it works:

* Based on UIDs, GIDs, and file permissions.
* Controlled by traditional chmod, chown, and umask.

****

* obaida (user) owns the file and can modify permissions.
* Group engineers can read it.
* Others can read, but not write.

**Weakness:** If obaida is compromised, an attacker can change permissions or share the file.

**2. MAC (Mandatory Access Control):** MAC is a strict access control model where access rules are centrally defined and cannot be changed by regular users, not even by root (depending on policy).

Implemented by:

* **SELinux**
* **AppArmor**
* **TOMOYO, Smack**

**How it works:**

* Each object and subject is labeled.
* Policies define which labels (types, roles) can interact.

**Example (SELinux):**A file labeled httpd\_sys\_content\_t can be read by the Apache process (httpd\_t), but not written unless explicitly allowed.

Even if file permissions say 777, SELinux can still deny access.

**3. RBAC (Role-Based Access Control):**  RBAC grants permissions based on **roles assigned to users**. Instead of assigning permissions directly to users, they inherit them through roles.

**Where it’s used:**

* Not native to Linux, but implemented via tools like **SELinux roles**, **FreeIPA**, or **cloud platforms** (e.g., AWS IAM).

**Example:**

* A user with the backup\_admin role can access backup tools and folders, but not production databases.

**4. ABAC – Attribute-Based Access Control (Optional but good for completeness)**: ABAC grants access based on **attributes** (e.g., time of access, device, user location, classification).

**Used in:**

* Advanced systems (e.g., cloud, government frameworks).
* SELinux’s MLS (Multi-Level Security) mode can be seen as ABAC-style when sensitivity levels are involved.

Example: “Allow access if user is analyst, working from internal IP, and data is labeled public.”

**Example:**

“Allow access if user is analyst, working from internal IP, and data is labeled public.”

**MAC (Mandatory Access Control):**

* **SELinux (Security-Enhanced Linux):** is a Mandatory Access Control (MAC) system integrated into the Linux kernel. It enforces rules based on security policies, independently from standard file permissions.

Unlike DAC, SELinux policies are enforced system-wide, controlled by administrators and security labels — even the root user can't bypass them without permission.

**Why Use SELinux?**

Without SELinux With SELinux

Root can do anything **---- with SELinux** even root is restricted

Malware can spread freely **---with SELinux** Compromised service is contained

Only DAC protection **--- with SELinux** Fine-grained, context-aware control

SELinux is essential for secure environments: servers, government systems, banking, and any system that needs strict isolation and control.

**Basic Tools and Commands:**

* **sestatus:** View current status and policy
* **getenforce:** Get current mode
* **setenforce:** Enable enforcing mode

**SELinux Modes of Operation:**

**Enforcing:** SELinux fully enforces all policies; blocks violations

**Permissive:** SELinux logs policy violations but doesn't block them

**Disabled:** SELinux is turned off completely

Changes made via setenforce are temporary. To change it permanently:  
Edit /etc/selinux/config

**Policy-Based Security** is a security model where access control decisions are determined by predefined rules (policies) rather than being made dynamically or based solely on user discretion.

These **policies** define:

* Who (users or processes)
* Can do what (read, write, execute, network access)
* To which resources (files, ports, devices)
* Under what conditions

**Key Principles:**

1. **Centralized Control:** Policies are written by administrators and enforced consistently across the system.
2. **Non-Discretionary:** Users (even root) cannot override policies unless explicitly allowed.
3. **Automated Enforcement:** Access is granted or denied automatically, based on the policy — **no human intervention needed**.

**SELinux Components and Terminology:**

* **Security Context**

Every process, file, socket, port, etc., in SELinux has a **context** that looks like this:





**User:** SELinux user (e.g., system\_u, unconfined\_u)

**Role:** Usually object\_r for files

**Type:** Defines what the object is used for — MOST IMPORTANT

**Level:** Sensitivity (used in MLS environments, e.g., s0, s0-s15)

**SELinux Types and Domains:**

SELinux works by associating processes with domains and files with types.

**Example:**

* Apache process: httpd\_t
* Web content files: httpd\_sys\_content\_t

If the policy says: "httpd\_t can read httpd\_sys\_content\_t" → access allowed.  
If not? Access denied — even if file permission is 777.

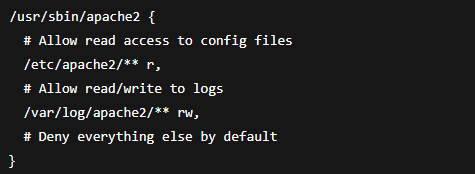
* **AppArmor (Application Armor):** is a Mandatory Access Control (MAC) system like SELinux, but it’s designed to be simpler to configure and understand.
* Instead of labeling everything with complex contexts, AppArmor works by attaching profiles to individual programs.
* These profiles define what files, capabilities, and network resources the program can access.
* It’s widely used in Ubuntu, openSUSE, and Debian.

**AppArmor Components and Terminology**

* **Profiles**: Text files defining the access rules for a program (e.g., /etc/apparmor.d/usr.sbin.apache2).
* **Modes**:
  + **Enforce**: Profile restrictions are actively enforced.
  + **Complain (or learning)**: Violations are logged but not blocked — useful for policy development.
* **Cache**: Compiled profiles loaded into the kernel.

**AppArmor Profiles Structure**

* Profiles contain file path rules specifying read, write, execute permissions.
* Example snippet from Apache profile:



**7.4- Data Encryption:**

**Data encryption:** is the process of converting plain data into an unreadable format using algorithms, so only authorized parties with the correct key can decode and read it.

**Types of Encryption**

1. **Symmetric Encryption**

* Same key used to encrypt and decrypt data.
* Fast and efficient for large data.
* Example algorithms: AES, DES.

1. **Asymmetric Encryption**

* Uses a key pair: public key (encrypt) and private key (decrypt).
* Used for secure key exchange and digital signatures.
* Example algorithms: RSA, ECC.

**Encryption in Linux**

* **File Encryption:** Tools like gpg, openssl to encrypt files.
* **Disk Encryption:** LUKS (Linux Unified Key Setup) for encrypting entire disks or partitions.
* **Network Encryption:** Protocols like SSH, TLS encrypt data in transit.

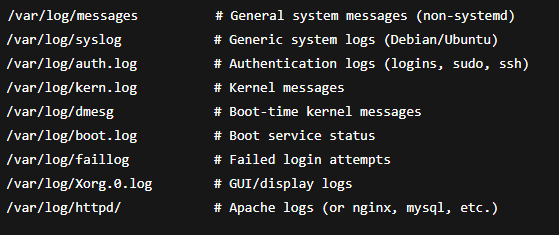
8- Trouble shooting and system maintenance

**8.1- Analize and Interpreting logs files:**

**Logs:** are system-generated records that track everything happening on the system — from startup to shutdown.

|  |  |
| --- | --- |
| **System Events** | Startup/shutdown, kernel loading, system crashes |
| **Service Status** | Start/stop/restart of services, failures, dependency issues |
| **Kernel Messages** | Driver issues, hardware errors, device detection, memory faults |
| **User Activity** | Logins, logouts, sudo usage, session creation |
| **Security Events** | Failed logins, brute-force attempts, auth rejections |
| **Network Activity** | Interface status, routing issues, DHCP, DNS resolution |
| **Application Logs** | Logs generated by programs (e.g., Apache, MySQL, Docker) |

**Logs locations:**

****

**Commands to show logs:**

* **journalctl**
* **dmesg**

**8.2- Backup and File Compression:**

**1. File Compression (Archiving & Compressing Files)**

* **Tools:** tar, gzip, bzip2, xz, zip, unzip
* **Purpose:** Reduce file size for storage or transfer
* **Used in:** backups, packaging, system recovery

**2. Backups (Copying and Preserving Data)**

* **Tools:** rsync, cp, dd, tar, cron, scp, dump/restore
* **Purpose:** Keep exact copies of files/systems
* **Used in:** disaster recovery, system migration, daily protection

**Compression** is reducing file size.  
**Archiving** is grouping multiple files into one.

**.tar** Archive only (no compression)

**.tar.gz** Archive + gzip compression

**.tar.bz2** Archive + bzip2 compression (better, slower)

**.tar.xz** Archive + xz compression (best ratio, slowest)

**.zip** Archive + compression (common in Windows)

**Backup:** is a copy of your important data that can be restored if the original is lost, damaged, or corrupted.

**Backups can include:**

* Files (documents, configs, scripts)
* Databases
* Entire system partitions or disk images
* Logs and snapshots

**Backup tools:**

rsync Incremental file and directory backup

cp Simple manual file/directory copy

tar Archive and compress files/directories

dd Clone entire disks or partitions (block-level)

cron Schedule automated backup tasks

scp / sftp Securely copy files to/from remote systems

dump / restore Backup and restore ext2/ext3/ext4 filesystems

**8.3- System Performance Monitoring**

9- Virtualization and Containerization

**9.1- Virtualization:** is the process of creating virtual instances of computing resources — such as servers, storage devices, networks, or even operating systems — on top of physical hardware.

**Types of Virtualization:**

* **Full Virtualization:** VM emulates full hardware; guest OS is unaware it’s virtualized.
* **Para-Virtualization:** Guest OS is aware it's virtualized and communicates with the hypervisor.
* **Hardware-Assisted:** Uses CPU extensions (Intel VT-x, AMD-V) to boost VM performance.
* **OS-Level (Container):** No hardware emulation; isolates processes within the same kernel (e.g., Docker).

**Common Virtualization Tools in Linux:**

* **KVM (Kernel-based Virtual Machine)** full virtualization using hardware extensions.
* **QEMU (Emulator/virtualizer):** often used with KVM for device emulation.
* **Libvirt:** API/toolset to manage VMs across different hypervisors.
* **virt-manager:** GUI frontend to manage libvirt/KVM-based virtual machines.
* **virsh:** CLI tool to manage VMs via libvirt.
* **VirtualBox:** Cross-platform virtualization for desktop environments.

**Networking Modes: Bridged vs NAT**

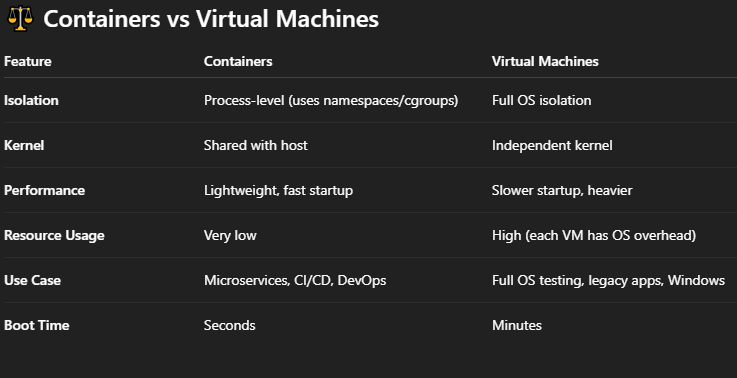
**NAT:** Default. Guest accesses external network via host. Simple setup.

**Bridged:** Guest gets an IP directly from LAN, like a physical machine. Good for servers.

9.2- **Containers:** are lightweight, isolated environments that run applications with their own filesystem, libraries, and dependencies — but share the same Linux kernel as the host OS.

**They isolate:**

* Processes
* Filesystems
* Networking
* Userspaces



**Common Container Runtimes:**

* Docker
* Podman
* LXC (Linux Containers)

**What is a Container Image?**

**container image** is a snapshot of a filesystem, plus metadata, used to instantiate a container.

Think of it like:

* A template → used to create containers.
* Read-only → containers add writable layers on top.

What is a Registry?

**Registry:** is a storage and distribution system for container images.

Common Registries Description:

**Docker Hub:** Public, default for Docker

**Quay.io:** Red Hat/Community registry

**GHCR:** GitHub Container Registry

**Harbor:** Self-hosted enterprise registry

Linux+

Practical

Topics:

1- Basics Linux Commands

2- Systemd  
3-Managing filesystem, partition and Disks

4- User and group management

5- Process and Service Management

6- Network Management

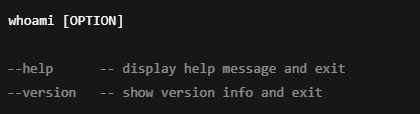
7- Security and Access Management

8- Trouble shooting and system maintenance

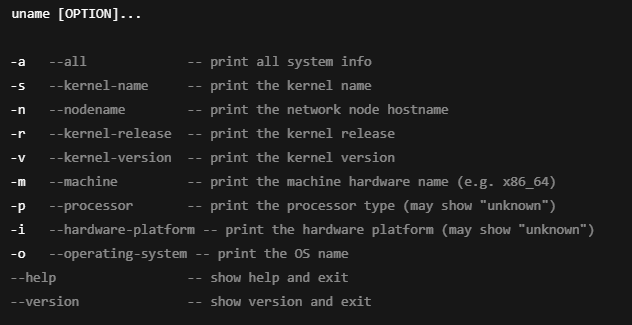
9- Virtualization and Containerization

**1- Basics Linux Commands**

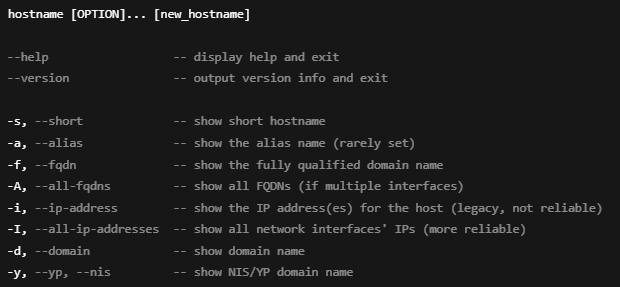
* 1. **system information commands:**
* **whoami:** command prints the current user’s username.

****

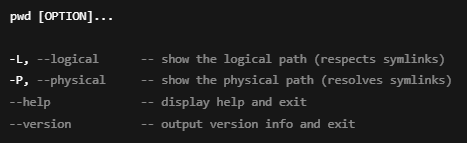
* **uname:** displays system information — like the kernel name, version, hardware type, and more depending on the options.

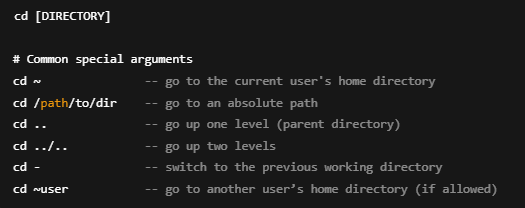


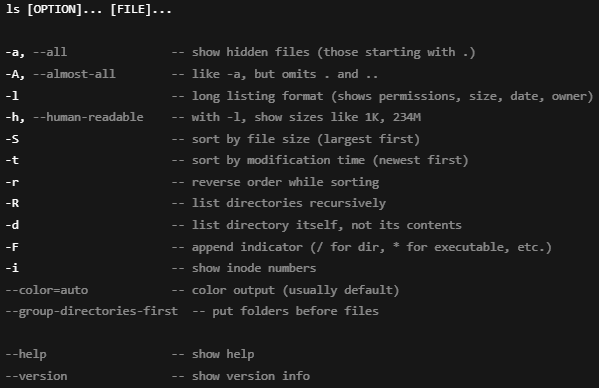
**hostname:** displays or sets system network identity information, including the hostname, domain name, and IP-related details depending on options. It helps query or modify the machine’s network identification and related info.



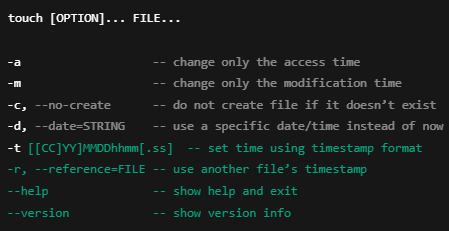
* **hostnamectl:** is a systemd utility to **query and change the system hostname and related metadata**, including static, transient, and pretty hostnames. It provides a modern, unified interface for hostname management and some related system identity info.
  1. Navigation commands:
* **pwd (print working directory):** displays the full absolute path of the current directory you’re in.

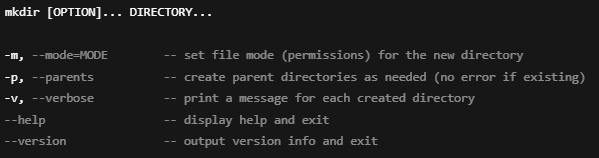


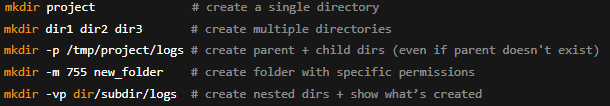
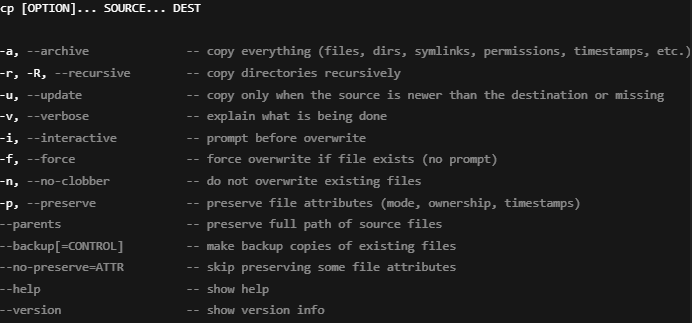


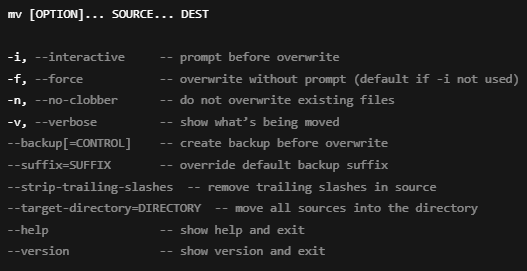


* 1. **files and directories management commands:**

****

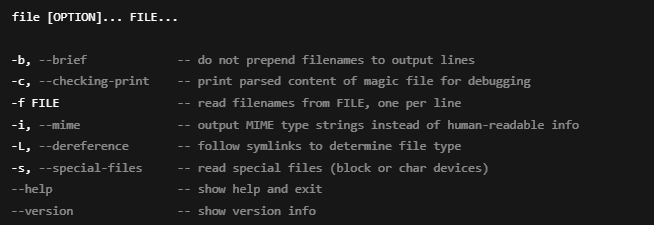
****

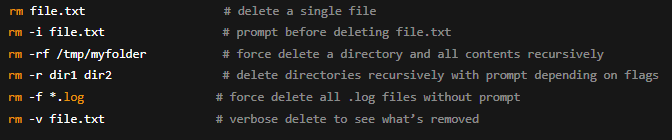
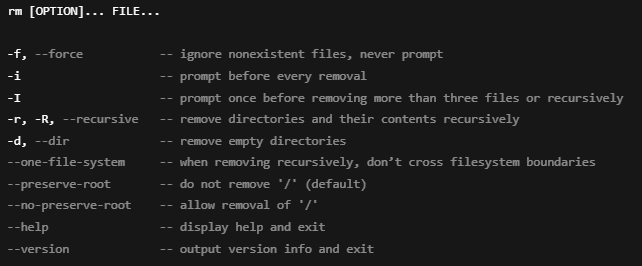
****

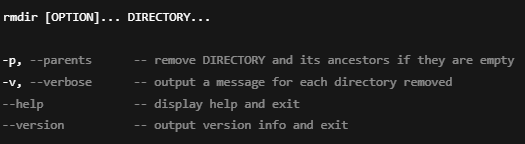
****

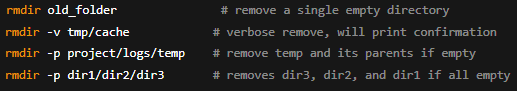
****

**file**: determines and displays the type of a file by examining its content (magic numbers, headers), rather than relying on its extension.

****

****

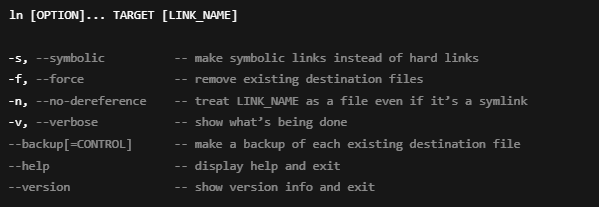
****

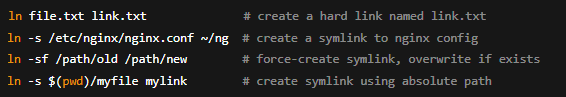
****

**ln:** is the command for creating hard and symbolic links — a core concept in Linux filesystems that every serious SysAdmin needs to understand cold.

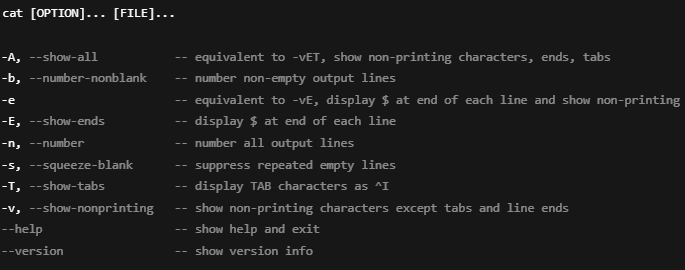
**ln** creates links between files. These can be:

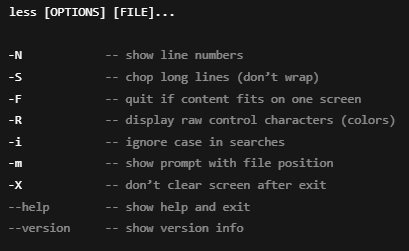
* **Hard links:** direct references to the same inode (i.e., another name for the same file).
* **Symbolic links (symlinks):** pointers to another file path (like shortcuts).



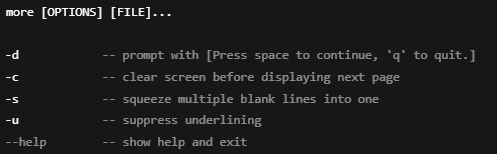


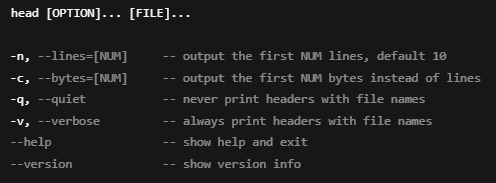
* 1. **Viewing and displaying files:**

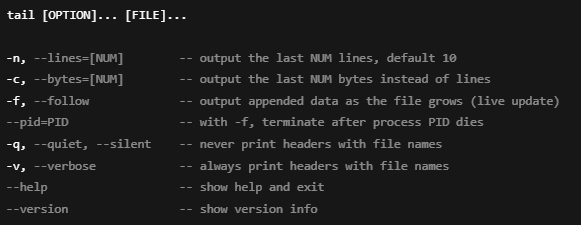
****

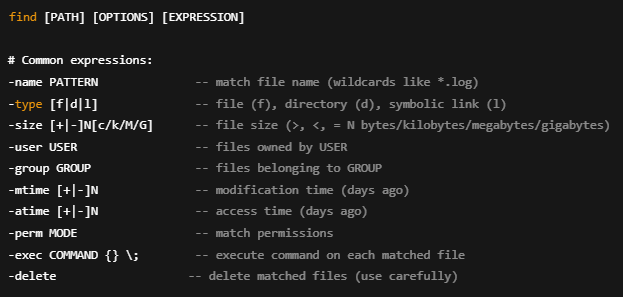
* less: is a pager program used to view files or output streams one screen at a time, allowing easy navigation forward and backward.

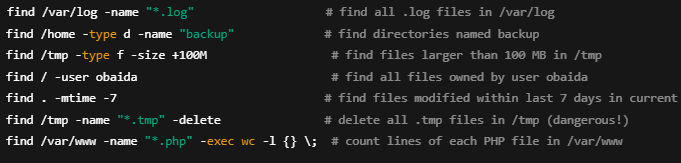
**more:** is a **basic pager utility** that displays file contents (or command output) one screen, allowing you to scroll forward but not backward.



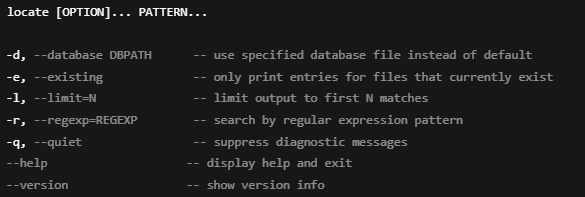


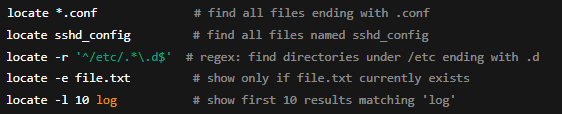


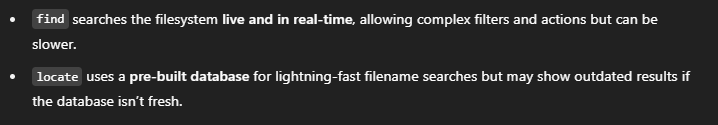
* **find:** searches the directory tree rooted at a specified location for files and directories that match given criteria, and can perform actions on those results.



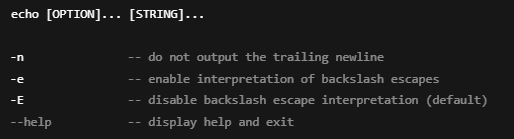
**locate:** quickly finds files by name using a **pre-built database** (usually updated daily), rather than scanning the filesystem live.

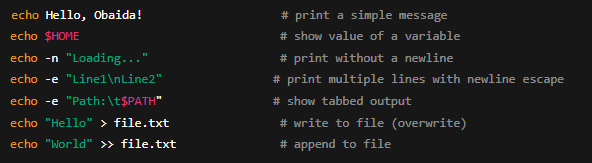


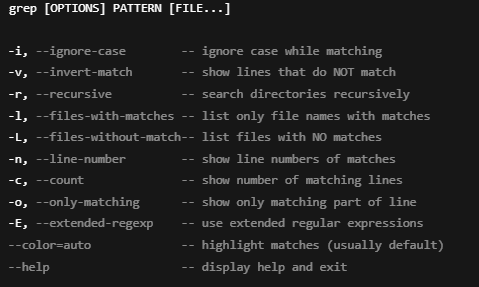


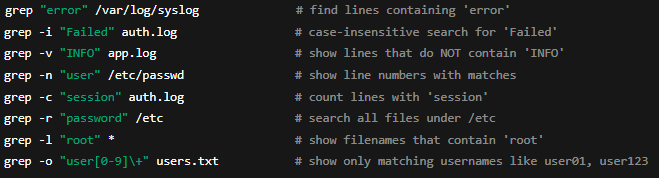


**1.5- File manipulation and searching commands:**

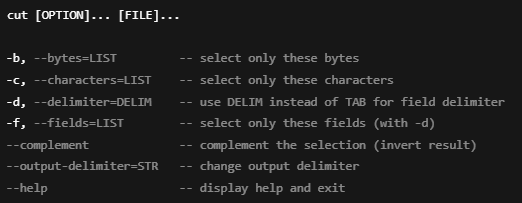


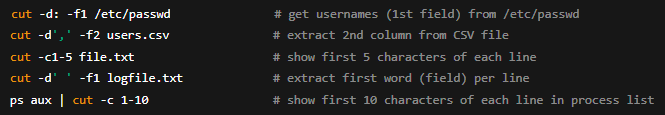




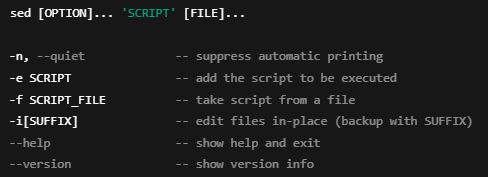
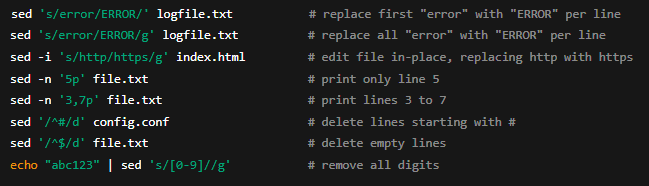


**cut:** extracts specific columns, fields, or **character ranges** from each line of input — either from a file or piped command output.





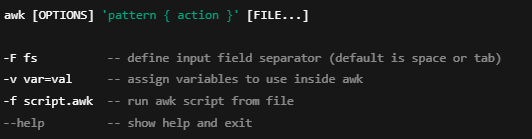
**sed:** is a stream editor that applies transformations to input line-by-line, often used for searching, replacing, deleting, and editing text in-place or through pipes.

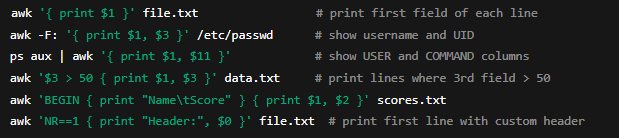


**awk**: is a powerful text-processing language that reads input line by line, splits it into fields, and applies actions and filters using a simple script-like syntax.

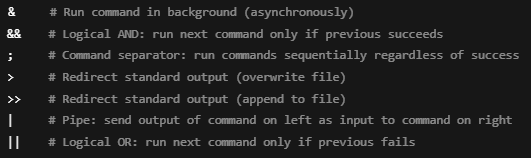
**When & Why We Use It:**

* To extract columns or fields from structured data (like logs, CSVs, or delimited output).
* To apply conditions and perform calculations on the fly.
* To format and report data in custom ways.
* Perfect when cut is too basic and sed is too linear.





**1.6- Key operators:**

****

2- Systemd

3-Managing filesystem, partition and Disks

**Partition management:**

**fdisk:** is an old tool used to create, delete, and manage disk partitions, mainly designed for MBR (Master Boot Record) disks. It can view GPT disks in some cases, but is not recommended for managing GPT — for that, tools like gdisk or parted are better.

Usage:

$ fdisk [options] <device>

now let’s break it down in parts

**<device>:** refers to the disk or storage device you want to partition.

For example:

* /dev/sda — The first **SATA/SCSI hard disk**.
* /dev/sdb — The second disk.
* /dev/nvme0n1 — An **NVMe SSD** (newer fast storage).
* /dev/vda — A **virtual disk** (common in VMs).
* /dev/loop0 — A **loopback device** (for mounting disk images).

**[Option]:** the options can be explored using man command

$ man fdisk

And here are the options

Common Options:

**-l [device] —** List partition tables, Shows partition info

$ fdisk -l /dev/sda: specific device

$ fdisk -l: all devices

**-w/-W<mode> —** Wipe mode (auto, always, never)

Controls if fdisk wipes (zeros) newly created partitions and old filesystem signatures from partitions.

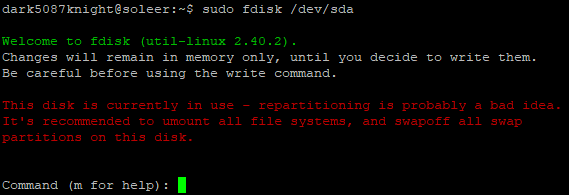
* **auto:** Default (only wipe if necessary)
* **always:** Always wipe
* **never:** Never wipe

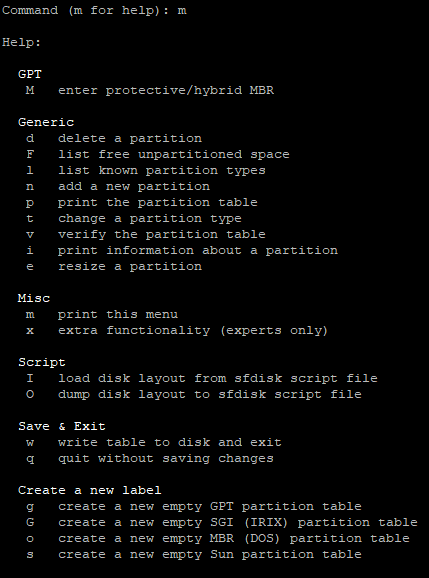
$ fdisk -w always /dev/sda

$ fdisk -W never /dev/sda

When we use the fdisk command on a disk in we go in an interactive mode so we can crate delete and modify partitions.

$ sudo fdisk /dev/sda



and to show the help table we write m and press enter

**parted:** is a modern disk partitioning tool that:

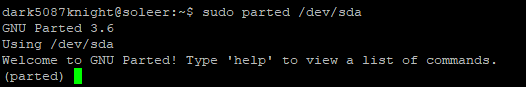
* Handles MBR and GPT partition tables.
* Supports disks larger than 2TB (unlike fdisk).
* Allows resizing, moving, copying partitions (which fdisk can’t do).

**Perfect for:**

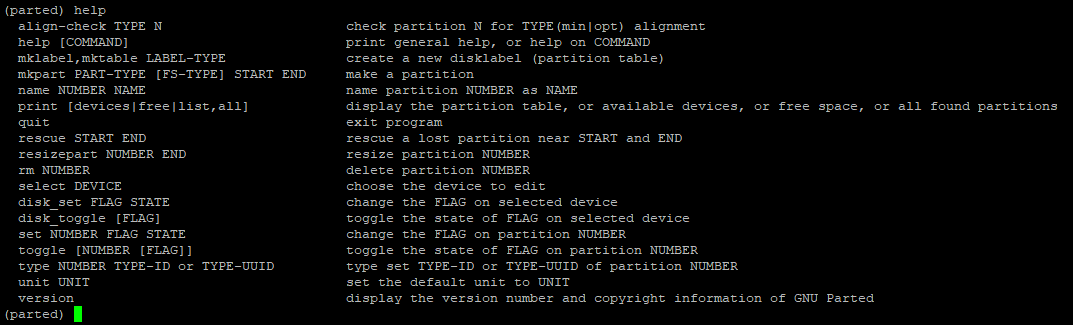
* Managing large modern disks (above 2TB).
* Working with GPT disks.
* Scripting/automation (parted can run in non-interactive mode)

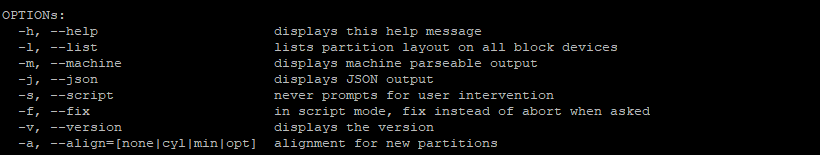
**Parted** has 2 modes of working the interactive mode where you interact with the tool and the script mode where you give the command or instruction directly

To enter the interactive mode you can simply use parted with the device  
$ sudo parted /dev/sda and you will be in interactive mode



And to show help and the use way type help and hit enter and the help menu will show up



**Common options:**

We can use the script mode using the option s and after give the command:

$ parted -s [device] [command [parameters]]

**Commands and parameters used with parted:**

**help:** Displays help information for all commands or for a specific one.

help [command]  
**[command]**: (optional) command you want help with.

help mkpart

Shows help about the mkpart command.

**print:** Displays the partition table or disk information.

print [devices|free|list]

* **devices** — List all available storage devices.
* **free** — Show all unallocated (free) space on the disk.
* **list** — Same as devices (list all disks).

print

Prints partition table of the current disk.

print free

Prints unallocated space on the disk.

**mklabel:** Creates a new partition table (disk label).

mklabel <label-type>

**<label-type>** — The type of partition table you want:

* + gpt — Modern, for large disks (over 2TB).
  + msdos — Old MBR format.
  + dvh — For SGI IRIX systems.
  + bsd — For BSD systems.
  + loop — For loopback files.
  + sun — For Sun/SPARC systems.

mklabel gpt

**name:** Names a partition (only works on GPT disks).

name <partition-number> <name>

* **<partition-number>** — The partition to name.
* **<name>** — The name you want to assign.

name 1 mypartition

**mkpart:** Creates a new partition.

mkpart [<part-type>] [<fs-type>] <start> <end>

* **<part-type>** — (optional) Partition type:
  + primary — Main partition.
  + logical — Logical partition (for MBR only).
  + extended — Extended partition (MBR only).
* **<fs-type>** — (optional) Filesystem hint:
  + Ex: ext4, fat32, ntfs (just a hint, does not format!).
* **<start>** — Start location (e.g., 1MiB, 0%).
* **<end>** — End location (e.g., 500MiB, 100%).

mkpart primary ext4 1MiB 1000MiB

**align-check:** Check if a partition is aligned properly important for SSDs.

align-check [type] <partition-number>

* **type** — minimal or optimal
* **partition-number** — Partition to check.

align-check optimal 1

**rm:** Deletes a partition.

rm <partition-number>  
**<partition-number>** — The number of the partition to

rm 1

Deletes partition 1.

**resizepart:** Resizes an existing partition to a new end location.

resizepart <partition-number> <end>

* **<partition-number>** — The partition you want to resize.
* **<end>** — New end position (e.g., 2000MiB, 100%).

resizepart 1 2000MiB

Expands partition 1 to 2000MiB.

**select:** Selects a different device without exiting parted.

select <device>

select /dev/sdb

**set:** Set or clear a flag on a partition.

set <partition-number> <flag> <on|off>

* **<partition-number>** — Partition to change.
* **<flag>** — Flag to set:
  + boot, esp, lvm, raid, swap, hidden, etc.
* **on|off** — Enable (on) or disable (off) the flag.

set 1 boot on

Marks partition 1 as bootable.

**unit:** Sets the display unit for sizes and positions.

unit <unit>

* **<unit>** — Unit to use:
  + s (sectors)
  + B (bytes)
  + kB, MB, GB, TB (metric)
  + KiB, MiB, GiB, TiB (binary)
  + % (percent of the disk)

unit MiB

Sets the unit to MiB.

**GParted** is a **graphical tool** (GUI) used to create, delete, resize, move, and manage disk partitions on Linux systems.

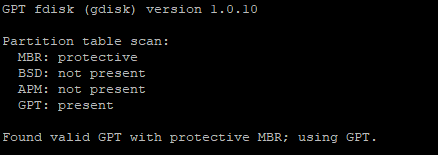
It does the same job as parted but with a **point-and-click interface** instead of command-line, Supports **MBR and GPT disks**, can format partitions with file systems like **ext4, NTFS, FAT32**, etc.., commonly used on **live CDs/USBs** to fix disks or prepare storage.

\*gparted will be covered later in the gui ubuntu or centos\*

**Gdisk:** is a **command-line tool** used to manage **GPT partition tables** on hard disks.

It's like fdisk, but designed specifically for **GPT disks only** (not MBR).  
You can use it to **create, delete, resize, and repair GPT partitions**.  
Useful for modern systems with UEFI and disks larger than 2TB.

**what gdisk can do that parted can't (or doesn't do well):**

* **MBR to GPT conversion — only gdisk can safely convert MBR to GPT in-place (without data loss).**
* **View or change GPT-specific details (like partition GUID, attributes, individual partition names more flexibly).**
* **Fix or repair broken GPT tables — gdisk has a powerful recovery (r) menu—parted does not.**

Usage:

$ sudo gdisk [device]

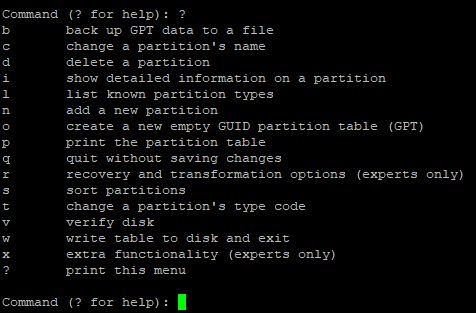
**MBR: protective**: Means this disk has a **"Protective MBR"**—a fake MBR to protect GPT disks from old tools, It prevents old MBR-only systems (like old BIOS) from mistakenly thinking the disk is empty.

**BSD: not present**: No BSD disk label found here.

**APM: not present**: No Apple Partition Map (APM)—this is used by old Mac systems.

**GPT: present**: A valid **GPT (GUID Partition Table)** was found, this is the **real partitioning system in use** on this disk.

The commands:



* 1. **Partition Viewing and Monitoring commands:**

**lsblk:**  It lists information about all available or specified block devices.  
**Block devices** = disks, partitions, LVM volumes, flash drives, etc.

usage:

lsblk [options] [device...]

**Default Output Columns:**

****

**NAME:** Device name — like sda, sda1, nvme0n1p1, etc.

**MAJ:MIN:** Major and minor device numbers — identifiers used by the kernel to manage devices.

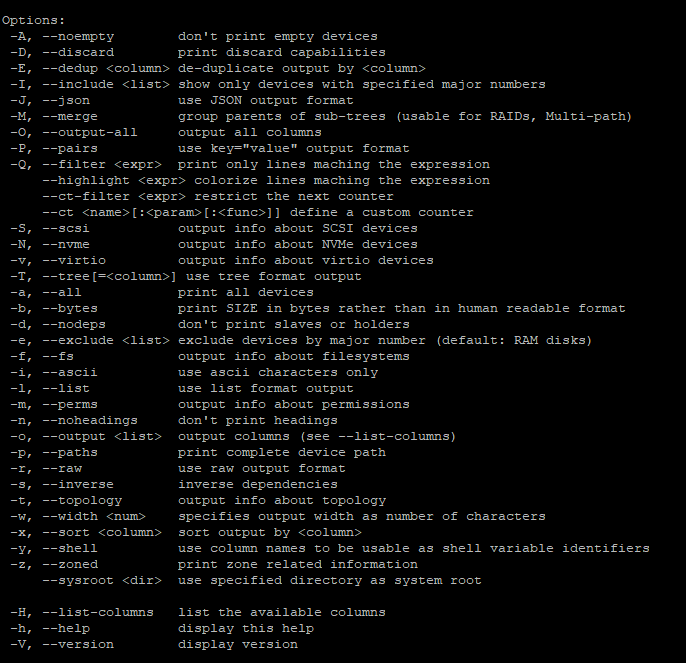
**RM:** Removable flag — 1 means the device is removable (like a USB stick), 0 means non-removable (like internal HDD/SSD).

**SIZE:** The total size of the device or partition (for example: 100G, 512M, 2T).

**RO:** Read-Only flag — 1 means the device is read-only (like a CD-ROM), 0 means it's writable.

**TYPE:** The type of the block device — such as disk (entire drive), part (partition), lvm (LVM volume), rom (read-only memory), etc.

**MOUNTPOINT:** Shows where the device or partition is mounted in the filesystem (such as /, /home, /boot, or blank if not mounted).

Options:

\

**blkid:** It shows UUID, LABEL, filesystem type, and other identifiers for block devices.

* Specifically shows device metadata (UUID, LABEL, FSTYPE).
* Focused ONLY on *identity info* — not size, not mountpoint, not topology.
* Perfect for finding which UUID matches which partition — this is what /etc/fstab and boot processes care about.
* Fast and simple output for automation scripts.

Usage:

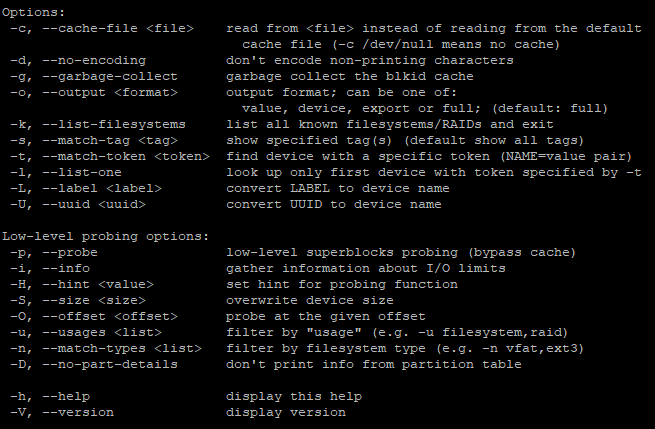
**blkid [options] [device...]**

$ blkid

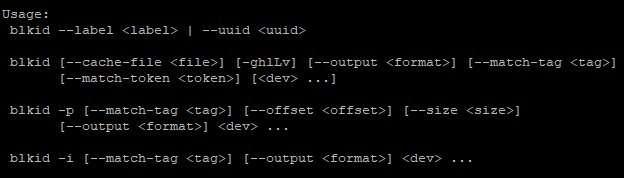
**Output:**



Options:



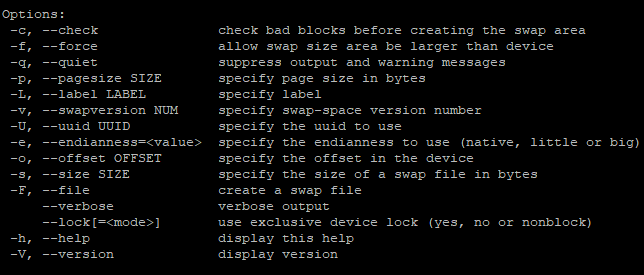
Usage of some options:



**Swap Management commands:**

**mkswap:** Formats a partition or file as a **swap area**

$ mkswap /dev/sda3 prepare this partition to be used as swap.



**swapon:** Turns on the swap space you prepared with mkswap.

$ swapon /dev/sda3 start using this swap partition.

**swapoff:** Turns off swap space — Linux will stop using it.

$ swapoff /dev/sda3 stop using this swap area now.

**Filesystem Management Commands:**

**mkfs:** is the **front-end command** to create (format) a file system on a block device like a hard disk, SSD, USB stick, or partition.

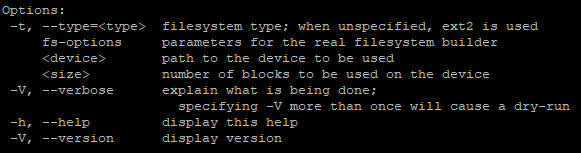
Usage:

mkfs [options] [-t <type>] [fs-options] <device> [<size>]

OR a simpler form:

mkfs.<fstype> [options] <device>

options:



Fs-options:

**ext2 / ext3 / ext4 (mkfs.ext4)**

**-b <block-size>**: Sets the block size of the filesystem (e.g. 1024, 2048, 4096), Larger block sizes are good for large files, but waste space for small files.

**-L <label>**: Assigns a label (name) to the filesystem.

**-m <reserved-percentage>:** Sets the percentage of reserved blocks for the root user, Default is 5%. You can reduce this to 0% for non-system disks.

**-O <features>**: Enable or disable specific filesystem features.

**-E <extended-options>**: Passes advanced options to tune filesystem creation.

**-T <usage-type>**: Hint about how the filesystem will be used. Possible values: news, small, largefile.

**-I <inode-size>**: Sets the size of each inode structure (commonly 128 or 256 bytes).

**-c**: Checks the disk for bad blocks before creating the filesystem.

**-F**: Force creation, even if the device appears to be in use or mounted (dangerous).

**XFS (mkfs.xfs)**

**-L <label>: Sets a label for the filesystem.**

**-m crc=<0|1>: Enables or disables metadata checksumming, Useful for filesystem integrity; default is enabled (1).**

**-d size=<value>: Manually sets the size of the data section, Usually not needed unless you're doing something unusual.**

**-n size=<value>: Sets the block size for the namespace (directory entries etc).**

**-l size=<value>: Sets the size of the log (journal) section.**

**-i size=<value>: Specifies inode size.**

**-f: Forces the creation of the filesystem, even if the device is mounted or contains data (dangerous).**

**-N: No action mode — shows what would be done without actually creating anything**

**VFAT/FAT16/FAT32**

**-L <label>**: Sets the filesystem label.

**-d <profile>**: Data block allocation profile — can be single, raid0, raid1, raid10, etc.

**-m <profile>**: Metadata block allocation profile — same possible values as for data.

**s <sectorsize>**: Specifies the leaf size and node size. Common values: 4096.

**-n <nodesize>**: Sets the size of internal Btrfs nodes.

**O <features>**: Enables or disables Btrfs features

**-f**:Forces the creation of a filesystem.

**VFAT/FAT16/FAT32**

**-F <12|16|32>**:Specifies the FAT type — either FAT12, FAT16, or FAT32.Usually FAT32 for modern large USB.

**n <label>**:Sets the volume label (up to 11 characters for FAT).

**-s <sectors>**:Sets the number of sectors per cluster.  
Controls the cluster size — can affect performance and disk usage efficiency.

**-I**: Forces the creation of a filesystem directly on the whole device (not on a partition).

**-c**: Checks the device for bad blocks before creating the filesystem.

**-v**: Enables verbose mode — shows detailed output of what the tool is doing.

**fsck: It is a system utility to check and repair file system errors on disks in Linux.**

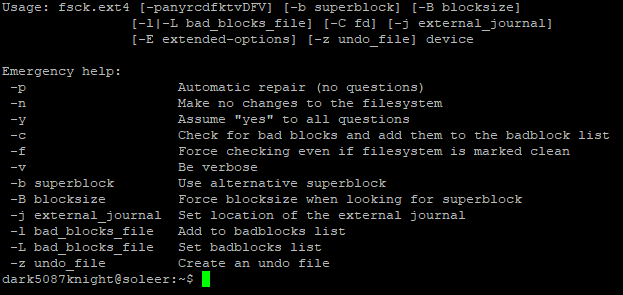
**When a file system (like ext4, xfs, etc.) gets corrupted, damaged, or inconsistent — for example due to sudden power loss, kernel panic, or unclean unmount — fsck comes to scan, check, and repair.**

**When to use fsck:**

* After a power failure or crash.
* When you see errors like "Superblock errors", "Inode errors", "Journal recovery failed".
* If the system refuses to mount a filesystem (e.g., root or home) because of inconsistencies.
* During system boot (sometimes automatically via systemd-fsck).

**Usage:**

**fsck [options] <device>**

**options:**

tune2fs: is a command-line utility to adjust tunable parameters on existing ext2/ext3/ext4 file systems — without needing to reformat or recreate the filesystem.

Usage:

**tune2fs [options] <device>**

**Use cases:**

* Change how much disk space is reserved for the root user.
* Change the filesystem label (name).
* Set how often the system checks the disk for errors.
* Set what happens if the disk has errors.
* Change the disk’s UUID.
* View detailed information about the filesystem.

**Options:**

****

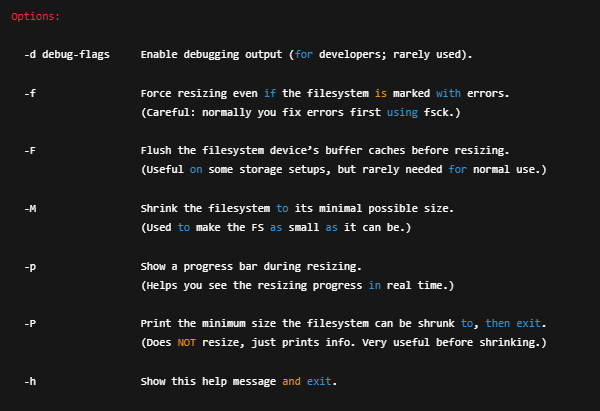
**resize2fs:** is used to resize (make bigger or smaller) an ext2, ext3, or ext4 filesystem.

* Expand the filesystem to fill a larger partition.
* Shrink the filesystem to make it smaller (careful: shrinking is risky if done wrong).

It works only on **unmounted filesystems** (or mounted ones if expanding online on certain setups).

Usage:

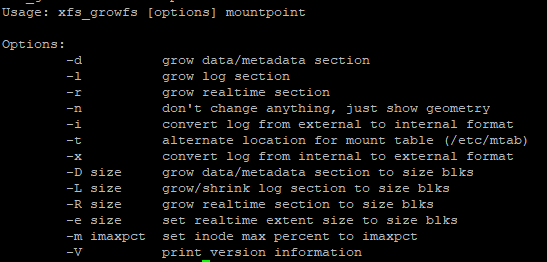
resize2fs [options] <device> [size]

Options:

**xfs\_growfs:** is used to expand the size of an XFS filesystem to fill the available space in its partition or logical volume.

Usage:

**xfs\_growfs [options] mount\_point**

****

**mount:** is used to attach (connect) a filesystem or storage device to the Linux filesystem tree.

When you plug in a disk, partition, USB, or ISO — you use mount to make its contents accessible (for example under /mnt or /media).

**umount:** is used to detach (disconnect) a mounted filesystem.

Before you remove or modify a disk, you must umount it to safely disconnect.

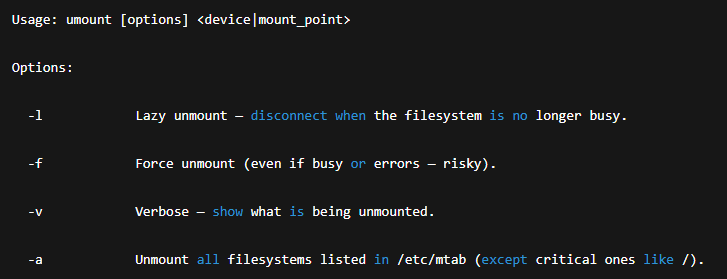
**Usage:**

**mount [options] <device> <mount\_point>**

**umount [options] <device|mount\_point>**

Options:





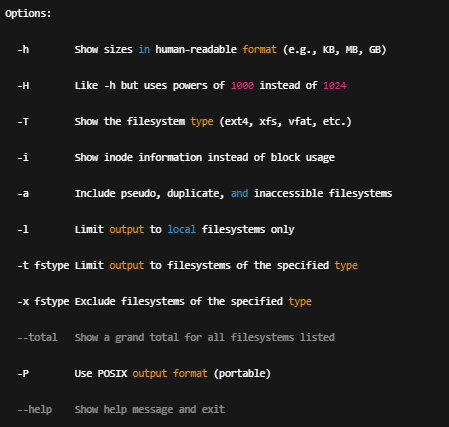
**Disk Management commands:**

**df:** It shows you how much disk space is used and available on mounted filesystems.

**usage:**

**df [options] [filesystem]**

**Options:**

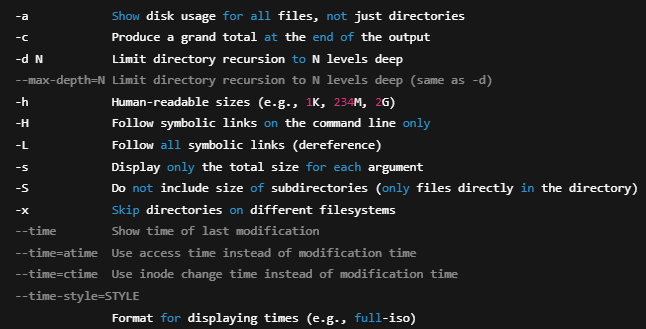


**du:** It shows how much space a file or directory uses on disk, recursively.

**Usage:**

**du [options] [file\_or\_directory]**

**options:**

****

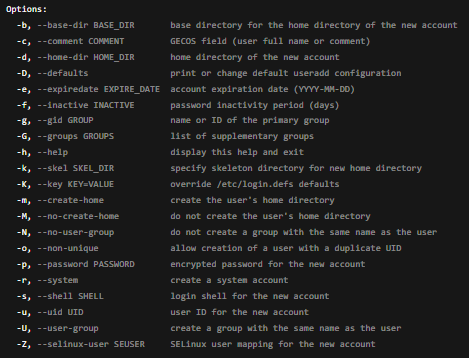
4- User and group management

**4.1- User management commands:**

**User creation:**

* **useradd:** is used to create new user accounts. It updates system files like /etc/passwd, /etc/shadow, /etc/group, and /etc/gshadow to add a new user to the system.

**Usage:**



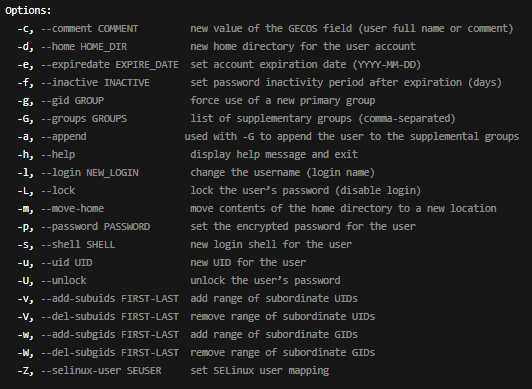
**Example:**

\*Note the command adduser is same as user add and it was built on it but its user friendly and interactive tool\*

**User Management:**

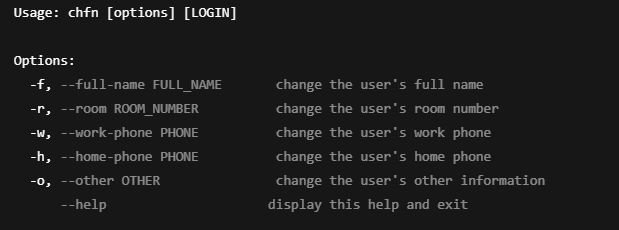
* **usermod:** is a Linux command-line tool used to modify or update an existing user account’s properties such as username, home directory, group memberships, login shell, UID, and more.

Usage:

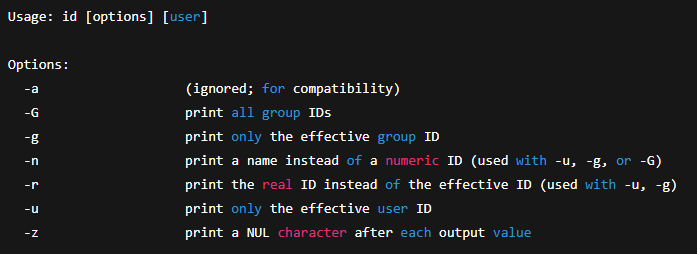


Example: Modify user1 to Have Bash Shell and Add to 'sudo' Group:

* **chfn(Change Finger Information):** is used to change a user’s personal information (called the GECOS fields) such as full name, office room number, work phone, and home phone in the /etc/passwd file.

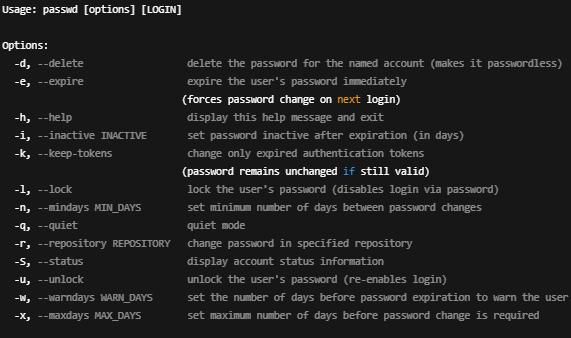


* **id:** displays the user ID (UID), primary group ID (GID), and all supplementary groups of a specified user or the current user if no username is given.

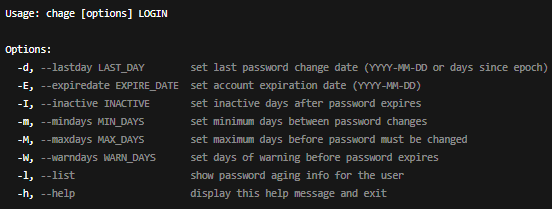


**User Password Management:**

* **passwd:** is used to change a user's password or set password aging policies on Linux systems. It updates entries in /etc/shadow securely.

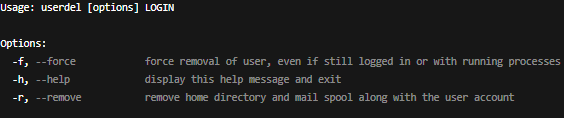


And if you used it directly you will be directed to interactive mode to change the user password $ passwd user1

* **chage (Change Age):** manages user password aging policies, including expiration, warning periods, and account inactivity settings stored in /etc/shadow.

**User Deletion:**

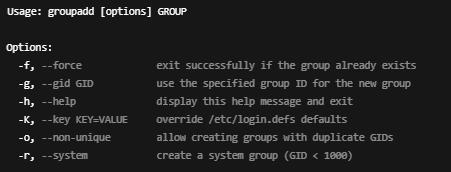
**userdel**: is used to delete a user account from the system, removing its entry from system files like /etc/passwd and optionally deleting the user's home directory and mail spool.



**4.2- Group Management commands:**

**Group Creation:**

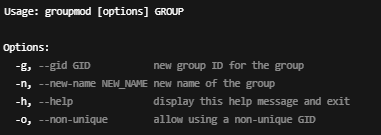
**groupadd**: is used to create a new group in the Linux system, updating the /etc/group file with the new group entry.

****

****

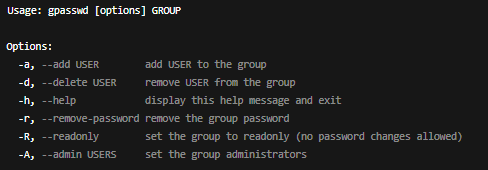
**Group Management:**

**groupmod:** modifies an existing group’s properties, such as its name or GID.





**Group Password Management:  
gpasswd:** is used to administer /etc/group and /etc/gshadow for group password management and group membership control.





**Group Deletion:**

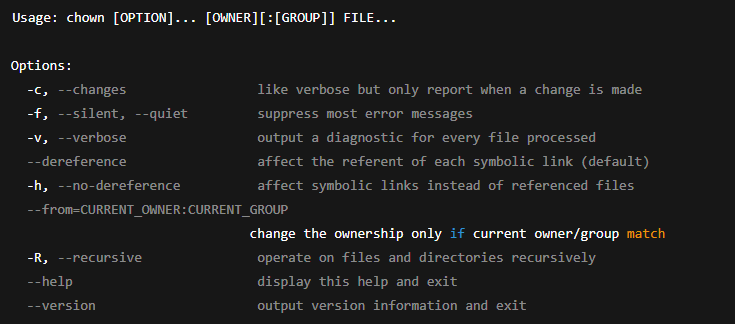
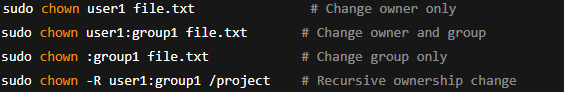
**groupdel:** deletes an existing group from the system by removing its entry from /etc/group and /etc/gshadow.

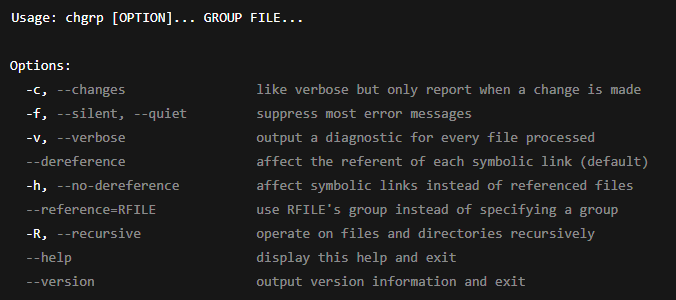
This command has no options only take group name as an argument

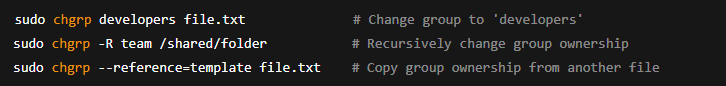


**4.3- File Ownership and Permissions Management:**

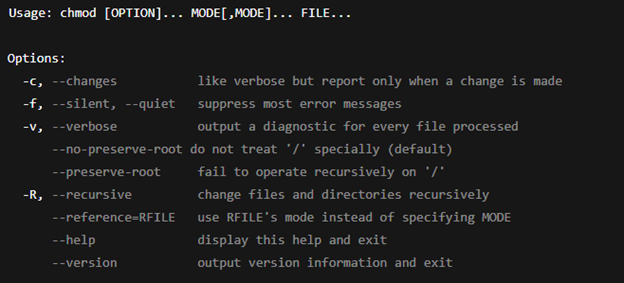
* **chown (change owner):** is used to change the ownership of files and directories specifically the user (owner), the group, or both.

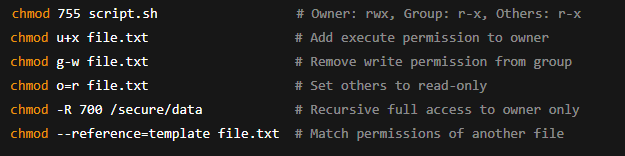


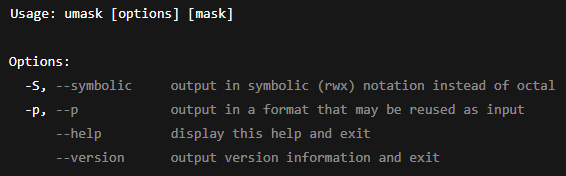
* **chgrp (change group):** is used to change the group ownership of a file or directory without modifying the user ownership.

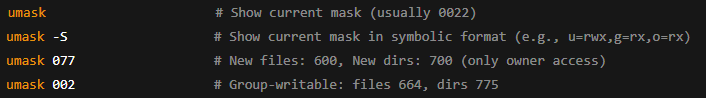
****

* **chmod (change mode):** is used to modify the permission bits of files and directories, controlling read (r), write (w), and execute (x) access for the owner, group, and others.



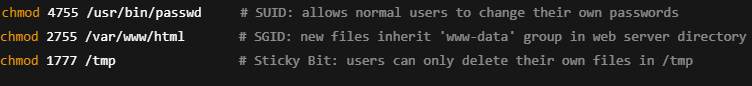


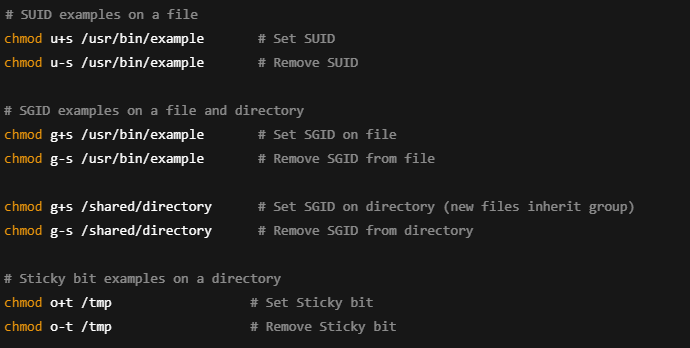
* **umask:** determines the default permission bits for newly created files and directories by "masking" (subtracting) permissions from the system defaults.



**Special permissions:**

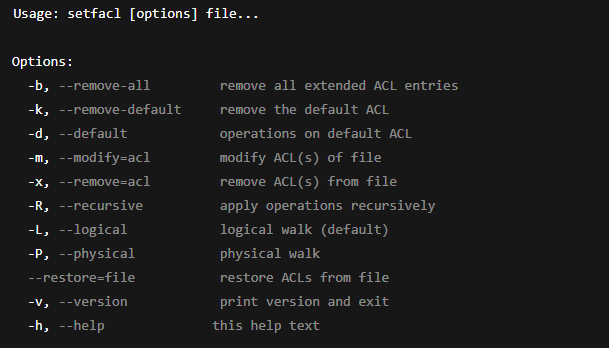
* **SUID:** Runs an executable with the file owner’s permissions instead of the user’s.
* **SGID:** Runs an executable with the file group’s permissions or makes new files inherit the directory’s group.
* **Sticky Bit**: Restricts deletion in a directory so only the file owner, directory owner, or root can delete files.



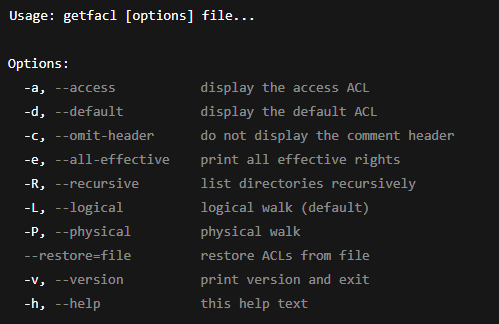


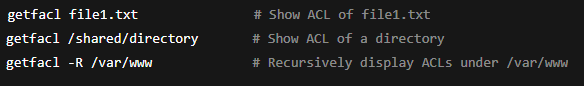
**4.4- ACL (Access Control List)**

* **setfacl**: is used to set Access Control List (ACL) permissions on files and directories.

****

* **getfacl:** displays the Access Control List (ACL) of files and directories.



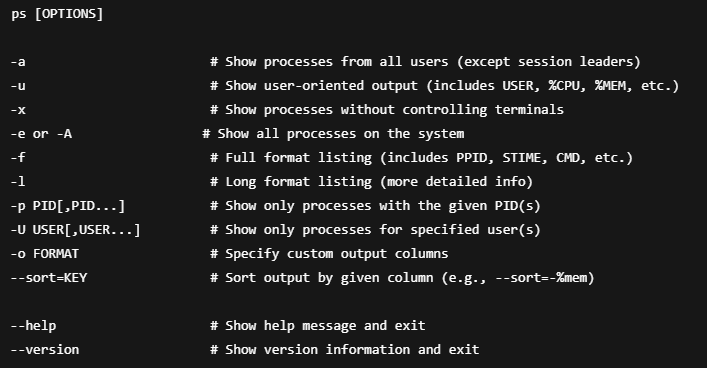


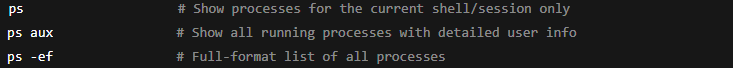
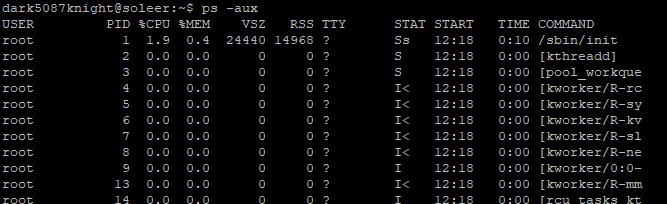
**5- Process and Service Management**

**5.1- Process Management:**

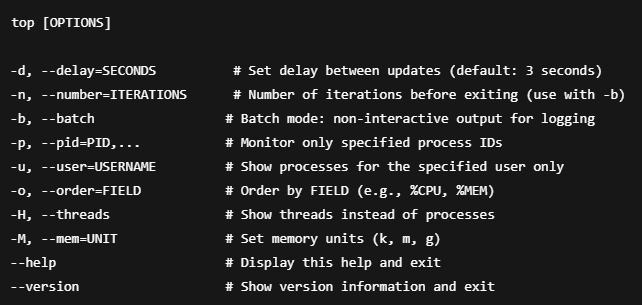
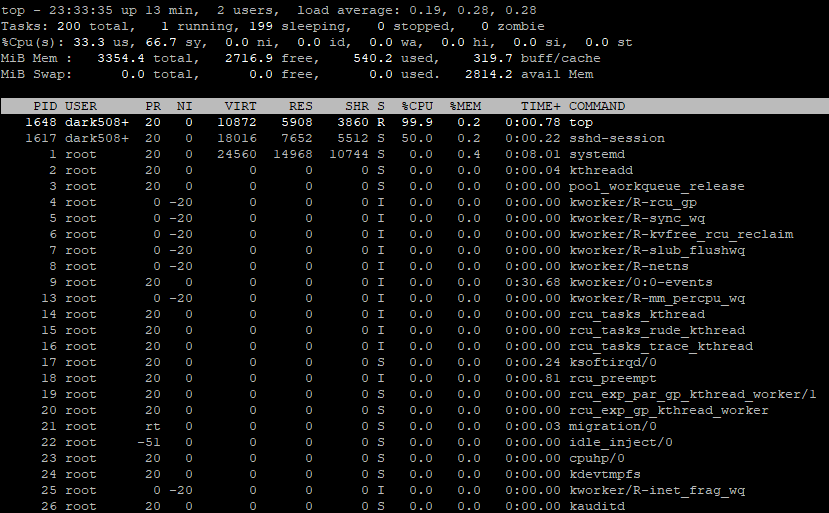
**Monitoring Commands:**

**ps** **(process status):** is a command-line utility used to view information about active processes on the system. Unlike top, it gives you a static snapshot of processes at the exact moment you run the command.

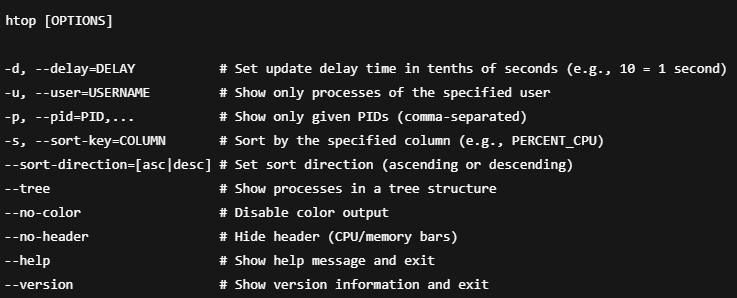


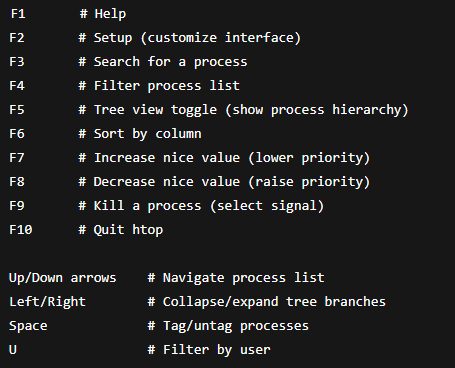


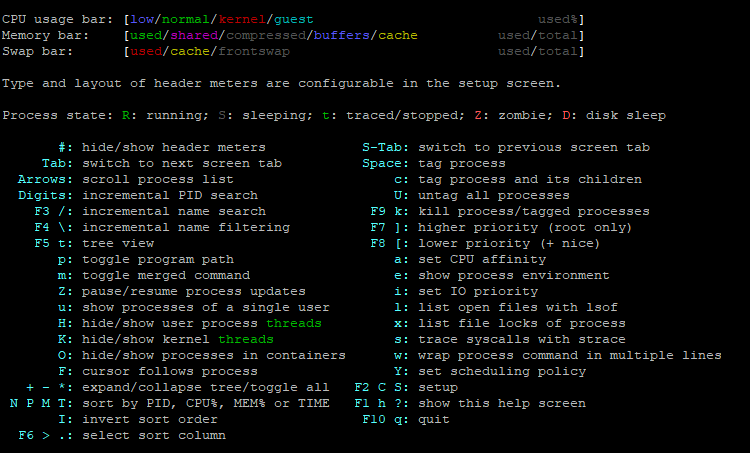
* **top:** is a real-time system monitoring utility that displays a dynamic, continuously updated view of the running processes on the system along with their resource usage (CPU, memory, etc.). It shows process IDs, CPU and memory consumption, uptime, and more.

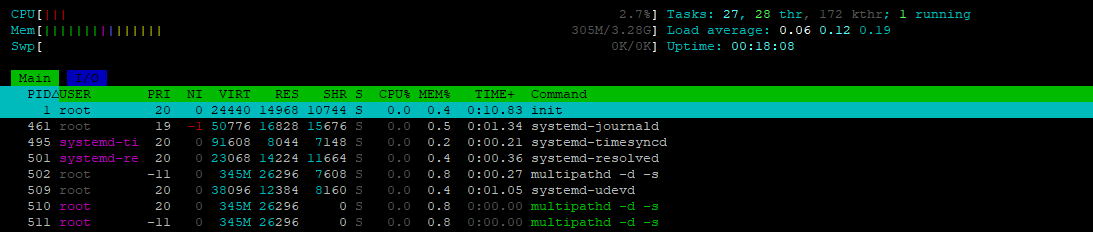


When running top it will take you to a screen showing process table and you can interact with it with the keyboard

* h or ? Show help screen with all commands
* q Quit top
* P Sort by CPU usage (high to low)
* M Sort by memory usage (high to low)
* T Sort by total CPU time used
* k Kill a process (enter PID and signal)
* r Renice a process (change priority)
* c Toggle command line display (full vs. name only)
* u Filter by user (enter username)
* 1 Show CPU cores separately or combined
* s Change update delay time (seconds)
* f or o Add/remove columns (customize output)
* **htop:** is interactive process viewer and system monitor, similar to top, but with a more user-friendly, and customizable interface. It displays system information, running processes, and resource usage in real-time.



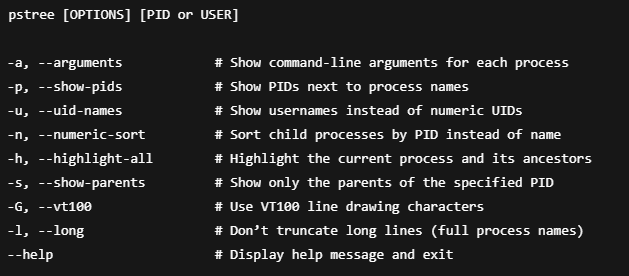
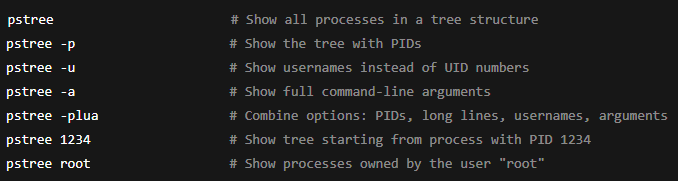


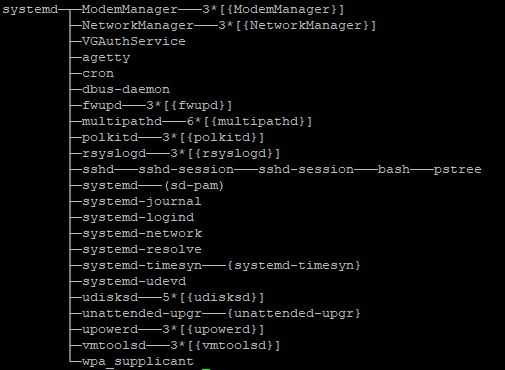


**htop Column Headers Explained:**

* **PID**: Process ID — the unique number assigned to each running process.
* **USER**: The owner (username) of the process.
* **PRI**: Priority — determines how much CPU time the process gets. Lower values mean higher priority.
* **NI**: Nice value — user-settable value that affects priority. Lower = higher priority.
* **VIRT**: Virtual memory — total memory the process can access, including swapped-out and allocated memory.
* **RES**: Resident memory — actual physical RAM being used
* **SHR:** Shared memory — memory shared with other processes.
* **S** :State — current status of the process (R, S, D, Z, T).
* **%CPU**: The percentage of CPU the process is currently using.
* **%MEM**: The percentage of physical RAM the process is using.
* **TIME+**: Total CPU time used by the process since it started.
* **COMMAND**: The name or path of the command that started the process (may show full command line if enabled).
* **pstree:** is a command-line utility that displays all currently running processes in a tree structure, showing the parent-child relationships between them.

Each process is listed with its name, and child processes are shown branching below their parent. It's a clear way to see how your system's process hierarchy is built.

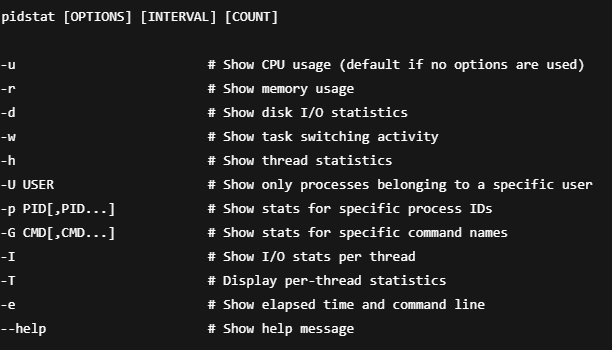


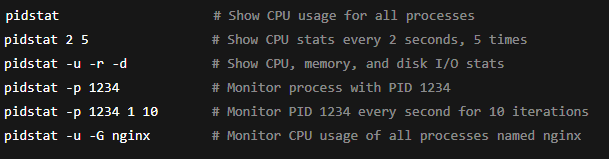


Running pstree with no options shows all system processes in a tree starting from PID 1, displaying only process names.

* **pidstat:** is a command-line tool used to monitor and report statistics about CPU, memory, I/O, and other resource usage by individual processes.

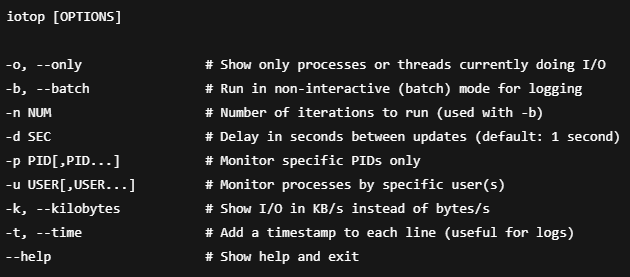
It's part of the sysstat package and is especially useful for tracking how a specific process behaves over time.

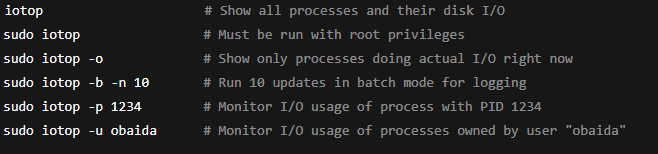


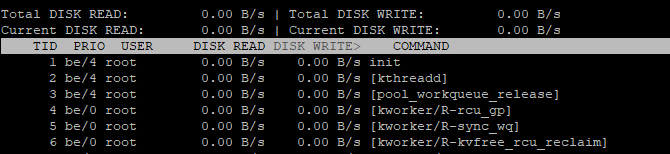


If you run pidstat with **no options**, it defaults to **-u**, which means it shows **CPU usage per process**.

**iotop:** is a real-time command-line utility that shows **which processes are using the disk (I/O)** and how much I/O they’re doing. It’s like top, but for disk usage instead of CPU/memory.





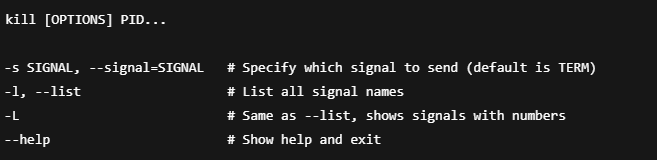


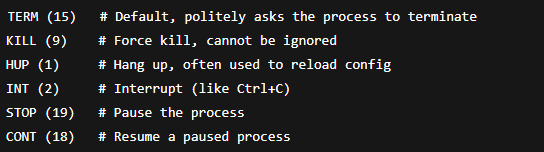
Running iotop with no options shows all processes in real time (including idle ones), refreshing every 1 second — root access required.

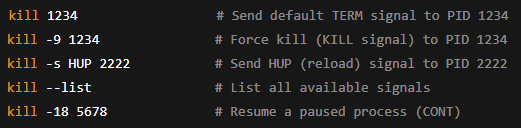
**Process Management commands:**

* **kill:** a command-line tool used to send signals to processes, most commonly to stop (terminate) them.

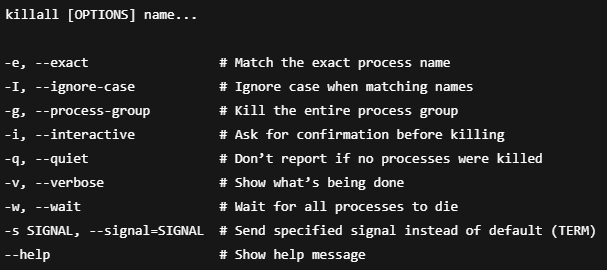
Despite the name, it doesn't always “kill” — it can send different types of signals (like pause, continue, terminate, etc.).

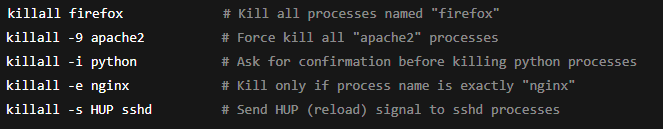
****

****

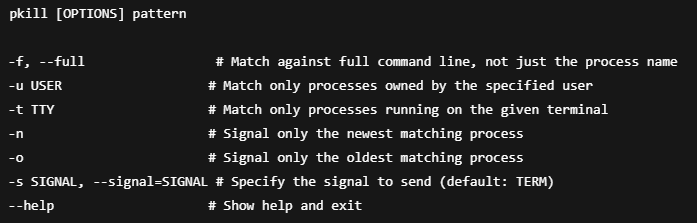
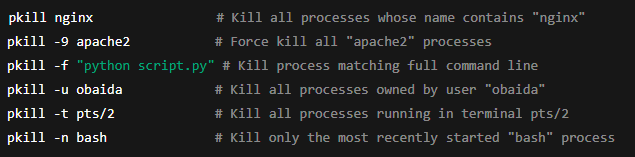
****

* **kill:** Use when you know the exact PID.
* **killall:** Use when you want to kill all processes by exact name.
* **pkill:** Use when you want to kill processes by partial name or other filters (user, terminal, etc.).
* **killall:** sends signals to all processes that match a given name, instead of using PIDs like kill.

****

****

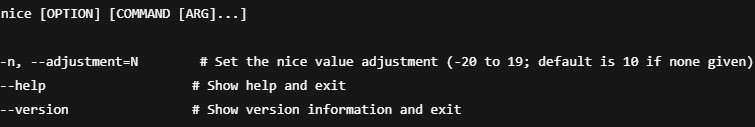
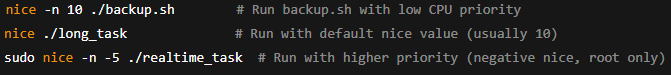
* **pkill:** sends signals to processes by matching their names or attributes using patterns or rules — it’s like kill + grep in one tool.

****

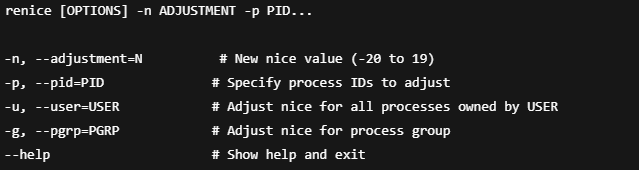
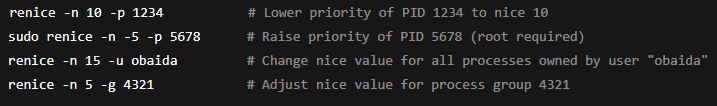
* **nice:** is a command-line utility to start a process with a modified scheduling priority, making it “nicer” to other processes by lowering its CPU priority (or, less commonly, increasing it).

**Key Points:**

* Nice values range from **-20 (highest priority)** to **19 (lowest priority)**.
* Normal processes start with a **nice value of 0**.
* Only root can set **negative nice values** (higher priority).
* The higher the nice value, the lower the priority.

* **renice:** changes the **nice value** (CPU scheduling priority) of one or more running processes, letting you increase or decrease their priority on the fly.



**5.2- Service management:**

for service management the main tool used to start, stop, restart, etc. is systemctl

**Main Service Management Commands:**

**systemctl start UNIT**: Starts the specified service immediately without enabling it to start at boot.

**systemctl stop UNIT**: Stops the specified service immediately.

**systemctl restart UNIT**: Restarts the service by stopping and starting it again (useful after config changes).

**systemctl reload UNIT**: Reloads the service configuration without stopping it (only works if the service supports reload).

**systemctl enable UNIT**: Enables the service to automatically start at boot.

**systemctl disable UNIT**: Disables the service from starting at boot.

**systemctl status UNIT**: Shows the current status, logs, and activity info of the service.

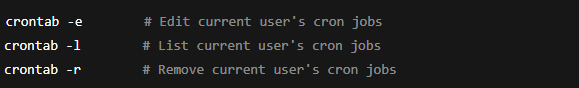
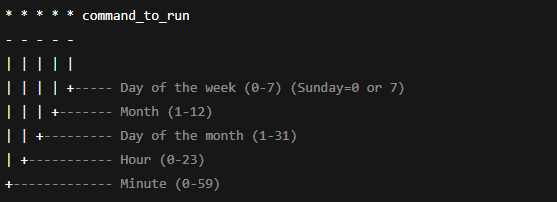
**Other Service Management Commands:**

* **systemctl is-active UNIT:** Checks if the service is currently running (active).
* **systemctl is-enabled UNIT:** Checks if the service is set to start automatically at boot.
* **systemctl reload-or-restart UNIT:** Reloads the service if possible; otherwise, restarts it.
* **systemctl try-restart UNIT:** Restarts the service only if it’s already running.
* **systemctl cat UNIT:** Shows the full content of the service unit file, including overrides.
* **systemctl edit UNIT:** Opens an editor to create or modify override settings for the service.
* **systemctl show UNIT:** Displays detailed properties and status of the service.
* **systemctl reset-failed [UNIT]:** Clears the failed state of one or more services.

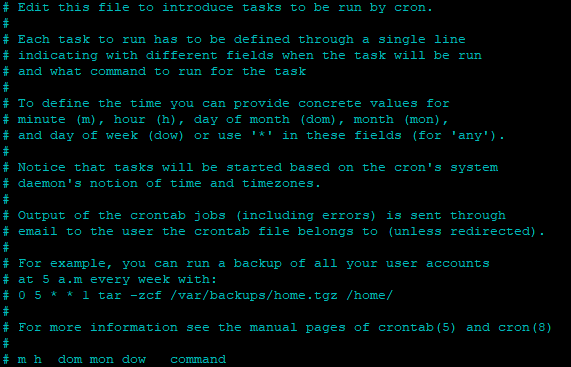
**5.3- Job Scheduling:**

**cron:** is a time-based job scheduler in Linux that runs scripts or commands automatically at specified times and intervals.

**Format:**

****

**Usage:**





Runs the backup.sh script every day at 2:00 AM.

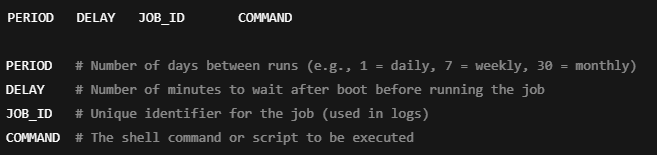
**anacron:** is a Linux tool used to schedule periodic jobs on systems that aren’t running 24/7.  
Unlike cron, it guarantees that jobs will eventually run even if the system was off at the scheduled time.

**When to Use It:**

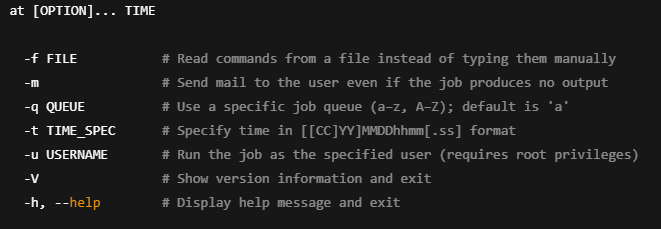
* You’re on a laptop, desktop, or non-server system that shuts down or suspends often.
* You need to guarantee periodic jobs still run, just maybe not at the exact time.
* Example: daily backup, log cleanup, or sync tasks.

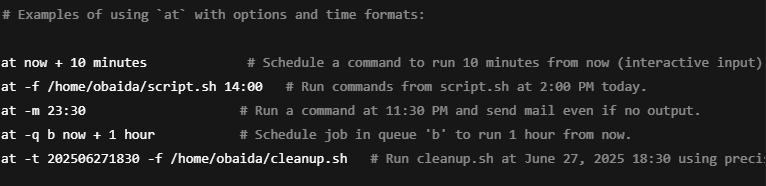
**How It Works:**

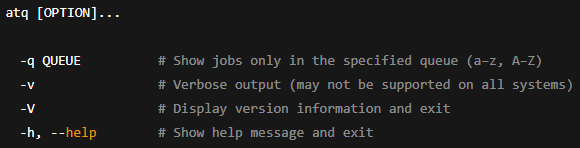
* anacron reads from a config file (usually /etc/anacrontab).
* It checks how many days it has been since a job last ran.
* If the job is due, it runs it **after a delay** (even if the system booted late).
* It **records timestamps** to track execution history.





* **at:** is a command used to schedule a one-time task to run at a specific time in the future.

****

**To list the scheduled jobs**

**To remove a scheduled job:**

**  
**

****

**Time structure:**

**1. Relative Times:** You can specify time relative to now:

* now + 5 minutes
* now + 2 hours
* now + 1 day
* now + 1 week

**2. Absolute Times:** Specify exact clock times (today or tomorrow if time already passed):

* 14:00 → Today at 2 PM (or tomorrow if already past 2 PM)
* 10:30 PM → Today at 10:30 PM
* midnight → The upcoming midnight
* noon → The upcoming noon
* teatime → 16:00 (4 PM)

**3. Dates:** Specify specific dates:

* June 26 → Next June 26th (today or next year if passed)
* June 26 14:00 → June 26th at 2 PM
* 6/26/2025 → June 26, 2025
* June 26 2025 14:00 → June 26, 2025 at 2 PM

**4. Combination of Date and Time**

* 10:00 AM June 30
* 23:59 December 31

**5. Timestamp Format (for scripting):** [[CC]YY]MMDDhhmm[.ss]

* CC = first two digits of year (century)
* YY = last two digits of year
* MM = month (01-12)
* DD = day (01-31)
* hh = hour (00-23)
* mm = minute (00-59)
* .ss = optional seconds (00-59)

Examples:

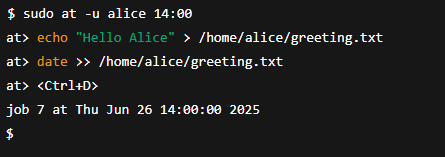
* 202506261430 → June 26, 2025 at 14:30
* 202506261430.30 → June 26, 2025 at 14:30:30

**Important:**

* Only users listed in /etc/at.allow can schedule at jobs.
* Users in /etc/at.deny are forbidden from using at.
* Root can override these.

When running

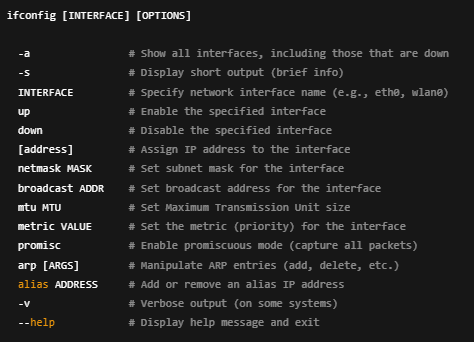
at [OPTION]... TIME

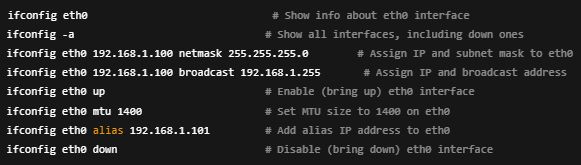
it will go to interactive mode to write the jobs must be done   


6- Network Management

**5.1- Network Configuration commands:**

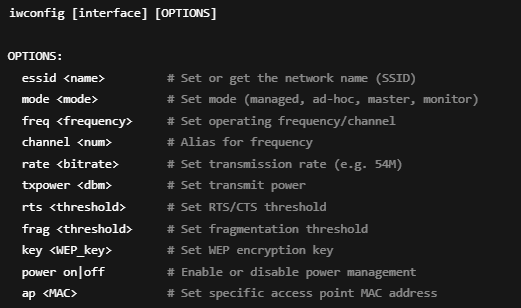
* **ifconfig (interface configuration):** is a command-line utility used to **view and configure** network interfaces in Linux/Unix systems.

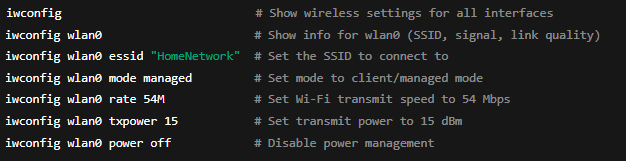




* **iwconfig:** is a **legacy CLI tool** used to **configure wireless network interfaces** — it’s the **wireless equivalent of ifconfig**, and is part of the **wireless-tools** package.

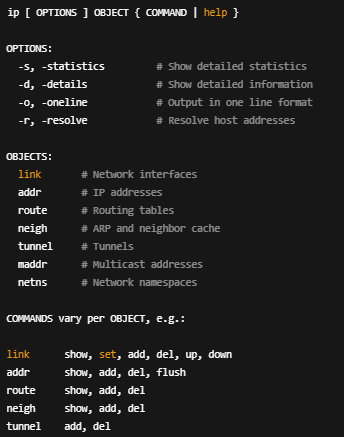
It lets you **view and set Wi-Fi parameters** like SSID, mode, frequency, bitrate, encryption keys, and more.

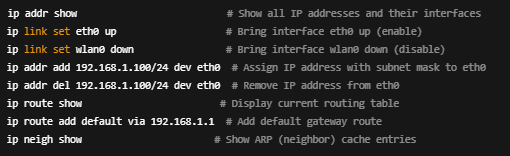




**Notes:**

* iwconfig **does NOT handle WPA/WPA2 encryption** — you'd need wpa supplicant for that.
* It’s **deprecated** on most modern systems in favor of iw, nmcli, and wpa\_cli.
* **ip:** is a part of the iproute2 package and is the modern tool for managing network interfaces, IP addresses, routing, and other network settings in Linux. It replaces older tools like ifconfig, route, and arp by providing a unified, powerful, and flexible interface for network configuration and monitoring.

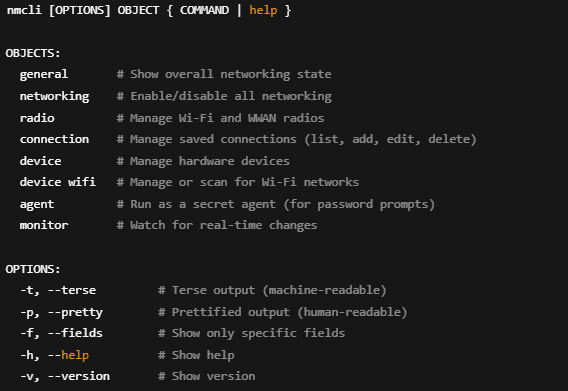


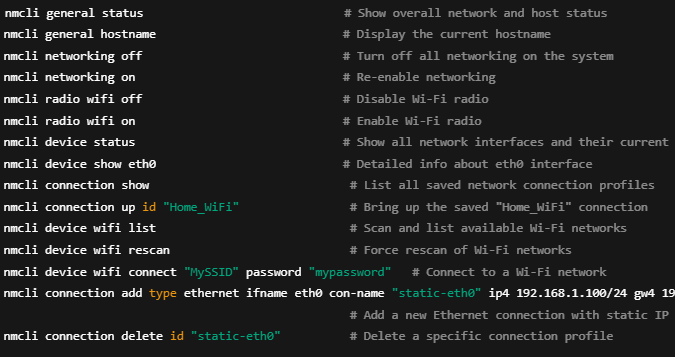


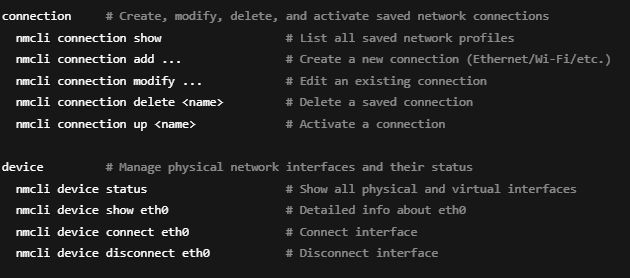
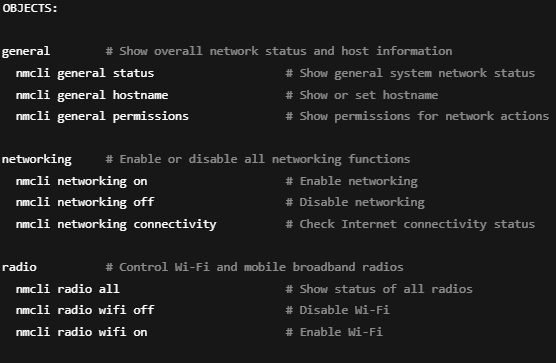
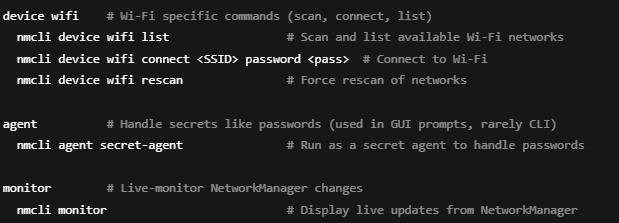
* **nmcli (Network Manager CLI):** is a **command-line tool used to manage network connections, devices, and settings** using **NetworkManager**.

It's great for:

* Viewing network status
* Creating/modifying connections (Wi-Fi, Ethernet, VPN, etc.)
* Activating/deactivating interfaces
* Scripting network changes without using a GUI

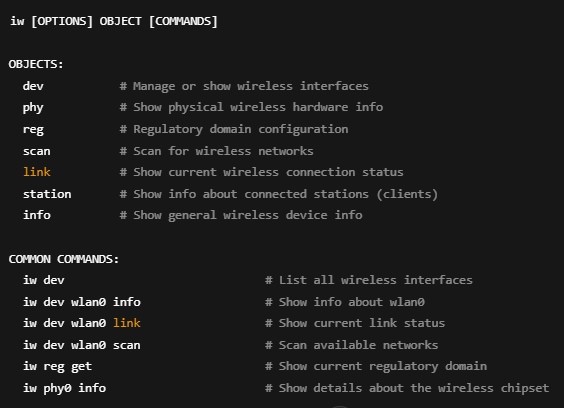


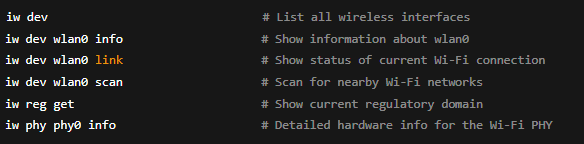
Examples:

the Object field has different input for each one and here are the inputs:

* **iw:** is a **modern CLI utility to configure and get information from wireless devices** using the **nl80211** kernel interface.

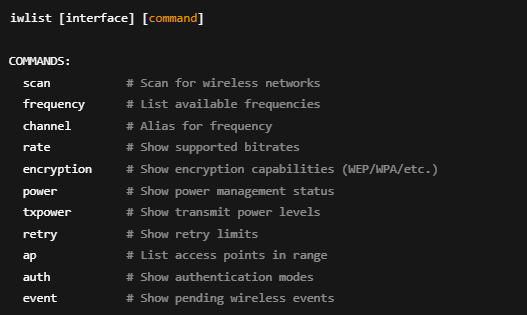
It’s lower-level than nmcli, meaning it gives **raw control over wireless interfaces**, but **doesn’t handle connections** (like DHCP, encryption, or authentication).

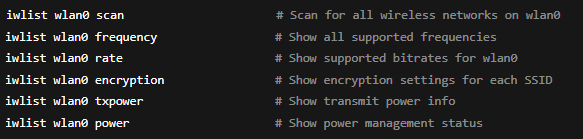




* **iwlist:** is a **legacy wireless tool** used to **retrieve more detailed information** from wireless interfaces, such as scan results, frequencies, signal levels, bit rates, and encryption info.

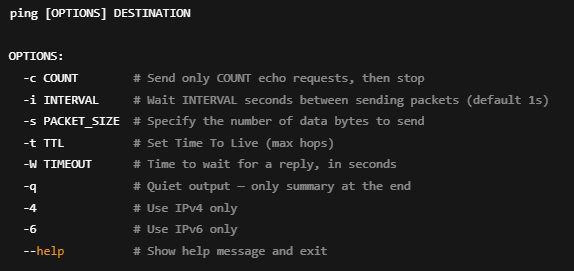
It's part of the **wireless-tools** package (not iproute2), and it works alongside iwconfig.

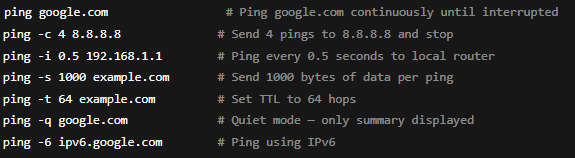




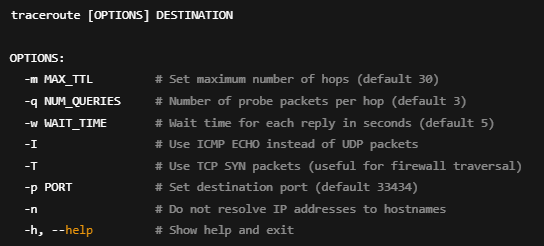
**5.2- Network Troubleshooting Commands:**

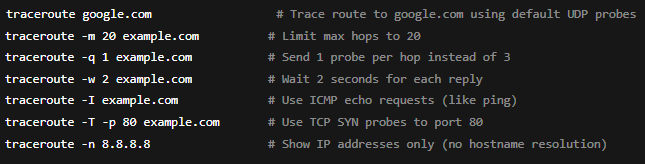
**ping:** sends ICMP echo request packets to a target host and listens for echo replies to check network connectivity and measure round-trip time.



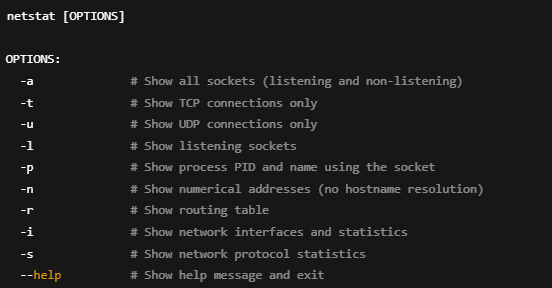


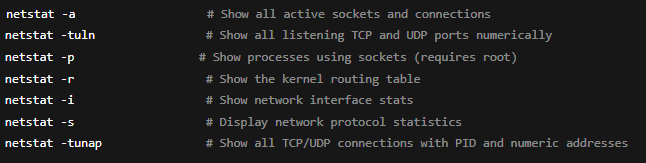
**traceroute:** traces the route packets take from your machine to a destination host by sending packets with increasing TTL (Time To Live) values and recording each hop’s response time.





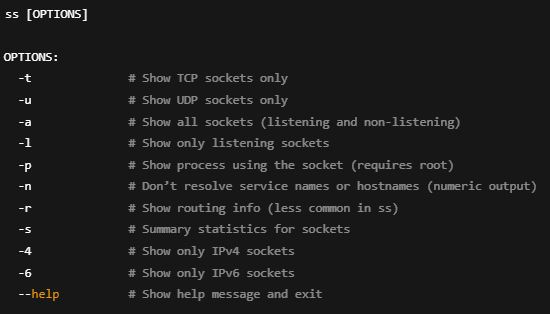
* **netstat:** displays active network connections, listening ports, routing tables, interface statistics, masquerade connections, and multicast memberships.

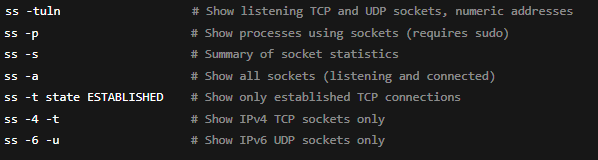




netstat is often replaced by ss on newer Linux distros for faster and more detailed socket info.

* **ss**: is a utility to **investigate sockets** on your system — showing detailed information about TCP, UDP, raw sockets, UNIX sockets, and more, much faster than netstat.



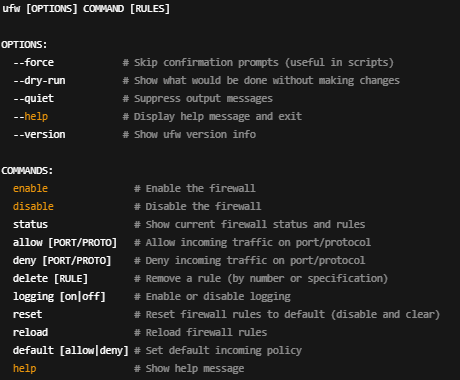


**Why ss beats netstat:**

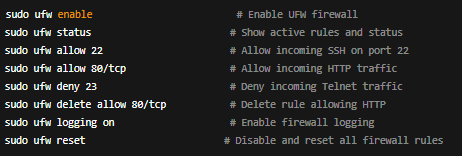
* **Faster output** because it reads data directly from kernel sockets.
* **More detailed info** about socket queues and memory.
* Supports **filtering by state** more easily.
* Actively maintained and standard on modern Linux.

**5.3- Firewal Management tools:**

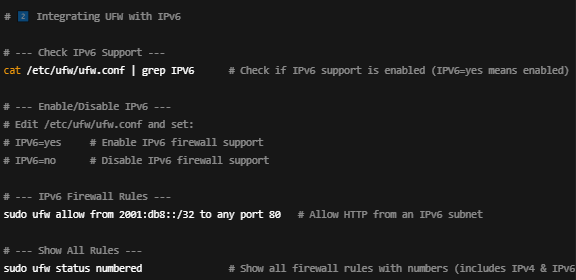
**ufw:** is a user-friendly front-end to iptables that simplifies firewall configuration, designed especially for Ubuntu and Debian-based systems.



Examples:







**And we can edit the rules directly from configuration files**

**Here are the configuration files we can edit:**  
/etc/ufw/ufw.conf

* # Core config file — controls basic settings like IPv6 support and logging
* # You can toggle IPV6=yes/no and enable/disable logging here.

/etc/ufw/before.rules

* # Rules applied BEFORE user-defined rules
* # Great place to add low-level firewall rules or custom chains.

/etc/ufw/after.rules

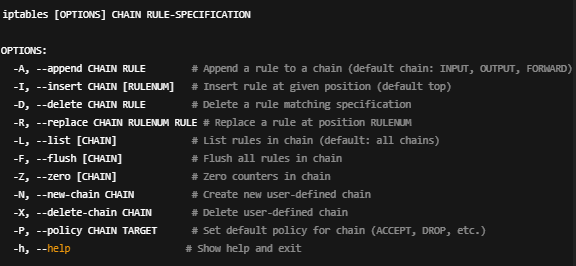
* # Rules applied AFTER user-defined rules
* # Useful for tweaking rules that should override user rules or for NAT.

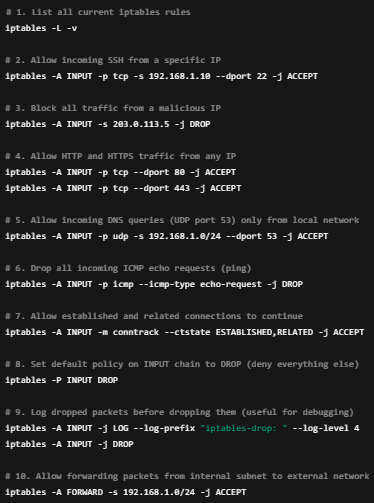
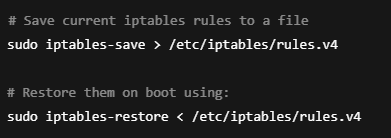
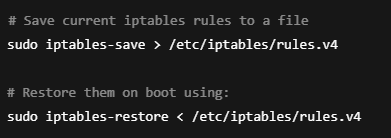
/etc/ufw/user.rules

* # This file stores all user-added rules (what you add with 'ufw allow', etc.)
* # Editing this manually is possible but risky — better to use ufw commands.

/etc/ufw/user6.rules

* # Same as user.rules but for IPv6 rules.

**iptables:** is a **firewall utility** that allows you to configure the Linux kernel’s **netfilter** firewall — letting you define rules for packet filtering, NAT, and mangling.

7- Security and Access Management

You can manually edit the rules.v4 file with a text editor (nano, vim, etc.).

Common file locations:

* **/etc/iptables/rules.v4 → IPv4 rules**
* **/etc/iptables/rules.v6 → IPv6 rules**

**When and Why to Use It:**

* To build custom, flexible firewall rules from scratch.
* For advanced filtering, NAT , and packet manipulation.
* When you need fine-grained control over inbound and outbound traffic.
* Used in many servers, embedded devices, and environments where detailed firewall policies are necessary.

**chroot:** changes the apparent root directory for a running process. This creates a restricted environment called a “chroot jail” — the process can’t see or access files outside of it.

**Step-by-Step: Creating a chroot Jail:**

**1. Create the jail directory**

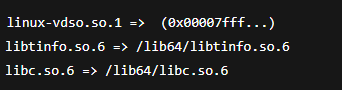
****

* 1. **Copy a binary (like bash) into the jail**

****

* 1. **Copy required libraries (use ldd to find them)**

****

****

Copy each of these into the appropriate jail subdirectory:



* 1. Enter the chroot jail



**SELinux:**

* **sestatus:** shows detailed SELinux configuration and runtime status. It’s like a SELinux dashboard in your terminal — giving you everything from mode to policy to mount points.

**The output will have:**

**SELinux status:** Is SELinux active on the system (yes or no)

**SELinuxfs mount:** Mount point of the SELinux filesystem (used by kernel for context info)

**SELinux root:** directory Where SELinux configuration files are stored (/etc/selinux)

**Loaded policy name:** targeted or mls (type of policy in use)

**Current mode:** enforcing, permissive, or disabled — live system mode

**Mode from config file:** What mode will be used after reboot (from /etc/selinux/config)

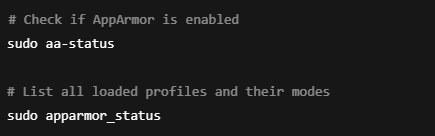
**Policy MLS status:** Is multi-level security enabled?

**deny\_unknown:** Whether unknown labels are denied access

* **getenforce:** is a simple, one-line command that shows the current SELinux mode: Enforcing, Permissive and Disabled

**AppArmor (Application Armor):**

**Checking AppArmor Status:**

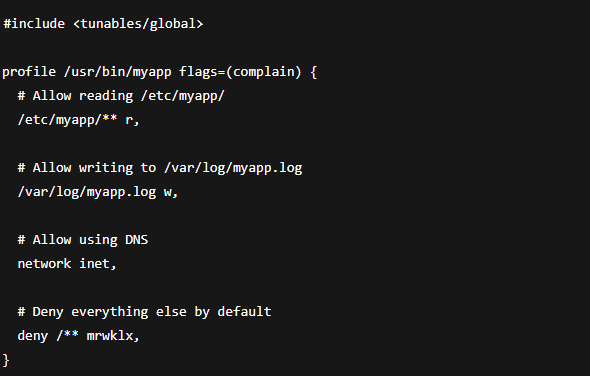
****

**Profile Location:**

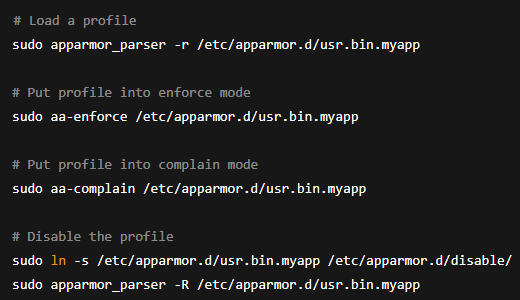
AppArmor profiles usually live in:

/etc/apparmor.d/

Example: A Simple Profile



**Enable / Disable / Reload Profiles:**



**Tools to manage Apparmor:**

aa-status Shows all profiles and modes

aa-enforce Puts profile in enforce mode

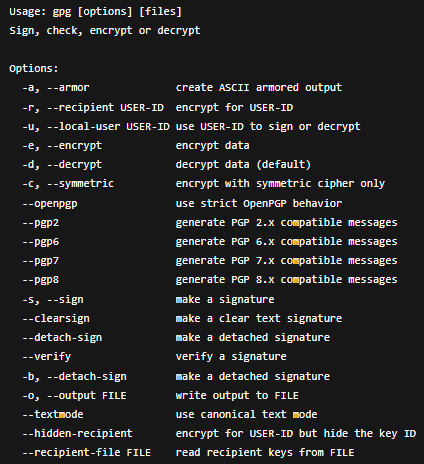
aa-complain Puts profile in complain mode

aa-logprof Interactive tool to build profiles from logs

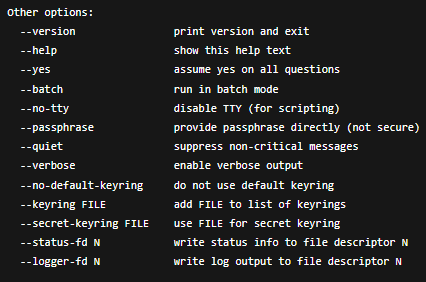
apparmor\_parser Loads and validates profiles

**7.3- Data Encryption:**

**GPG (GNU Privacy Guard):** is a free, open-source encryption tool that follows the OpenPGP standard (RFC 4880). It allows you to:

* Encrypt and decrypt data
* Create digital signatures to verify authenticity
* Manage keys for secure communication





 Examples

**File Extensions**

* Encrypted file: .gpg or .asc (if ASCII armored)
* Signed file: .sig, .gpg

**OpenSSL:** is a robust, full-featured open-source toolkit that implements:

* SSL/TLS protocols
* Cryptographic algorithms
* PKI (Public Key Infrastructure) features like certificate creation, signing, and verification
* And general-purpose encryption/decryption

**Why Use OpenSSL?**

**Create SSL/TLS certificates:** Self-signed or with a CA

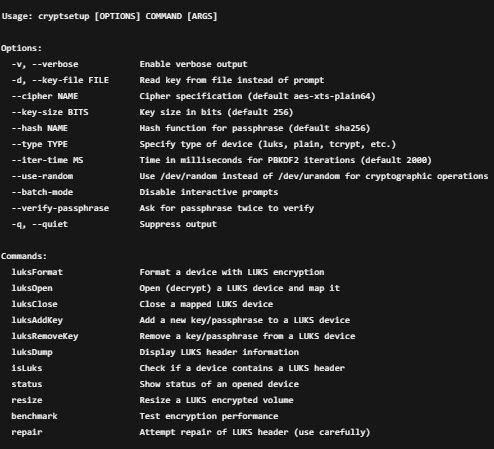
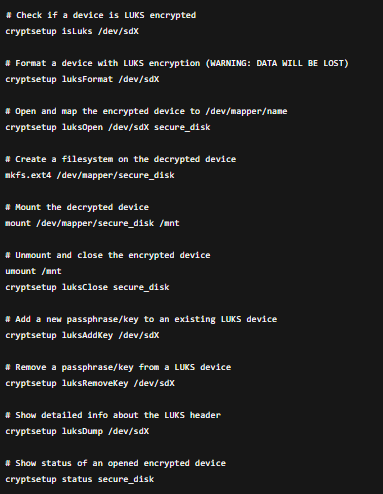
**Encrypt/decrypt files or strings:** Symmetric/asymmetric support

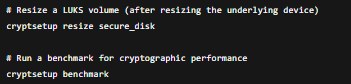
**Hash data:** MD5, SHA-1, SHA-256, etc.

**Inspect/convert certificates:** View or convert PEM, DER, PKCS12 forma

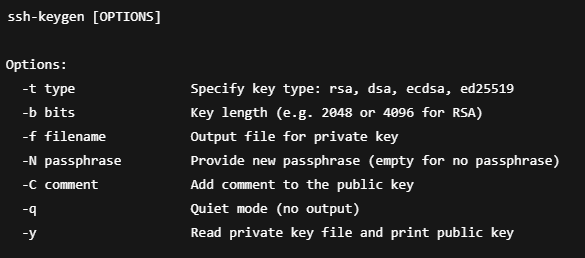
**Generate key pairs:** RSA, EC, DSA

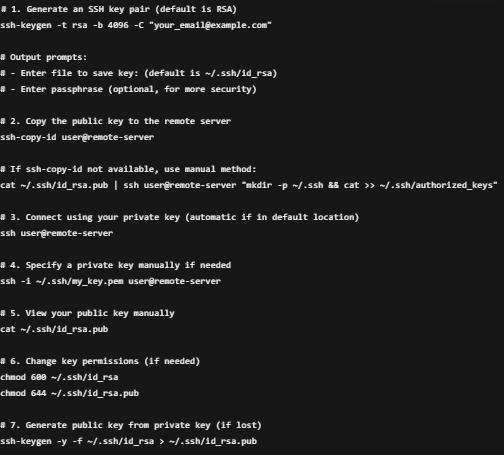
**cryptsetup:** is the command-line utility used to manage disk encryption on Linux systems, specifically for LUKS (Linux Unified Key Setup) volumes.





**ssh-keygen**: is a command-line utility used to generate, manage, and convert authentication keys for SSH (Secure Shell). It creates public-private key pairs that are used for secure, passwordless login and encrypted communication with remote systems.

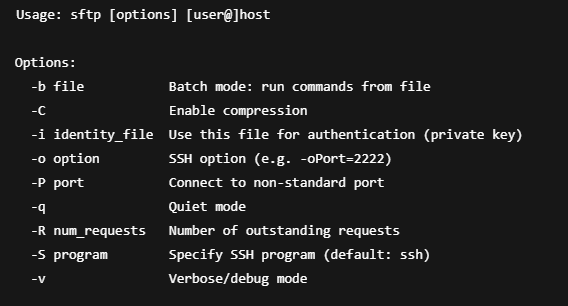


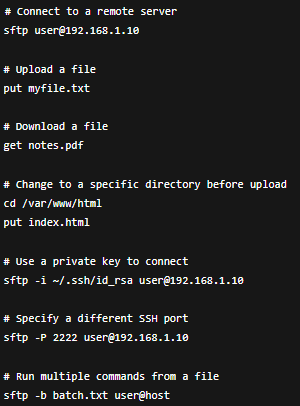
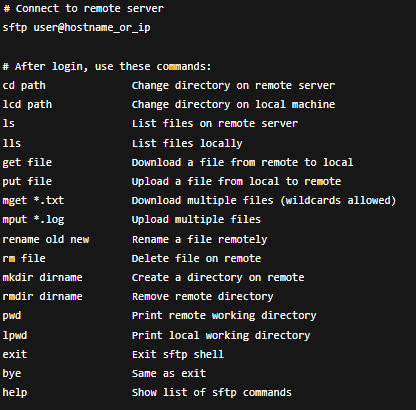
**Typical SSH Key Workflow:**

**7.3- File Transmission:**

**sftp:** it’s a secure file transfer tool that works over an SSH connection.

It’s not FTP over SSH, and it’s not related to FTP or FTPS — it’s an entirely different protocol built into SSH.

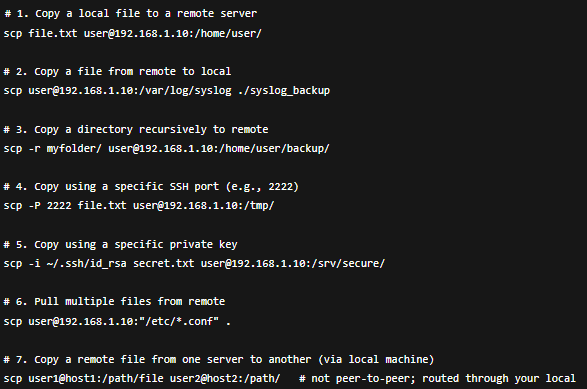


****Usage examples

**scp:** It’s a command-line tool used to copy files and directories securely over an SSH connection — like a secure cp between machines.

It uses SSH encryption, so it’s much safer than legacy ftp or rsh.



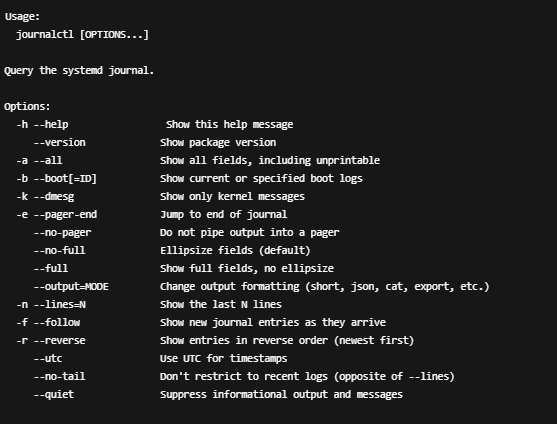
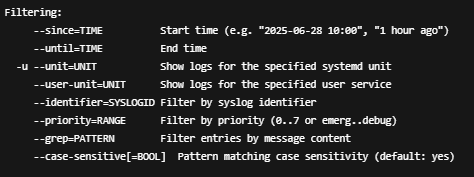


8- Trouble shooting and system maintenance

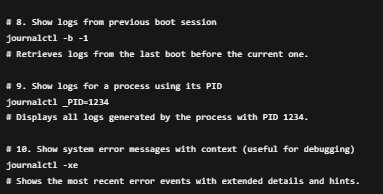
**8.1- Analize and Interpreting logs files:**

**journalctl:** is the command-line tool used to query and display logs collected by the systemd-journald service.

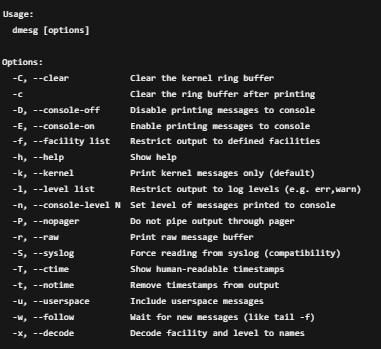
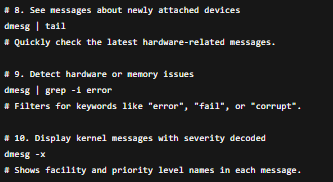
It reads the binary journal logs that contain detailed system events: services, kernel, user sessions, boots, shutdowns, etc.



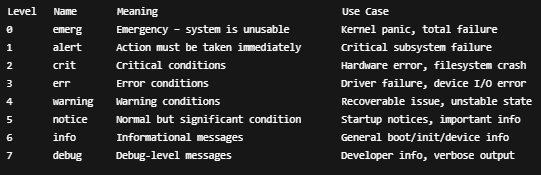




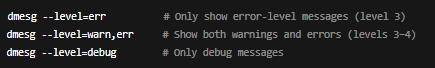
* **dmesg**: stands for “diagnostic message” — it displays kernel ring buffer messages, which are logs generated by the Linux kernel, especially during boot time and hardware events.

****

**dmesg / kernel log levels:**

****

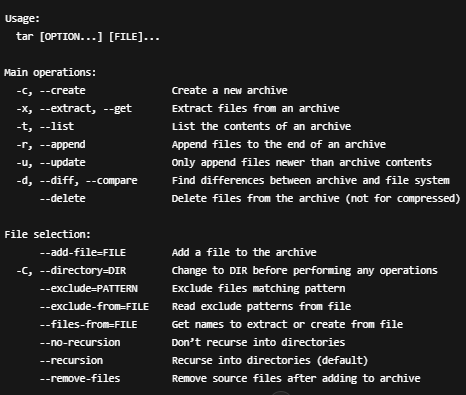
**How to Filter by Log Level with dmesg:**

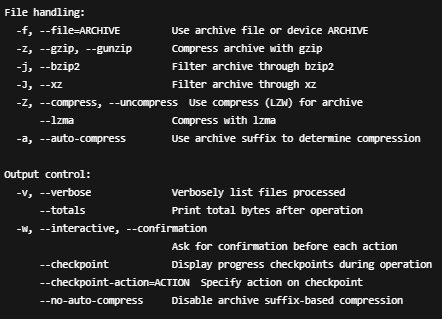
****

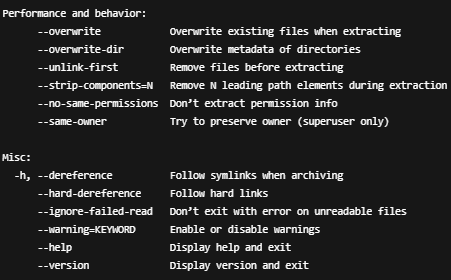
**8.2- Backup and File Compression:**

**1. File Compression (Archiving & Compressing Files)**

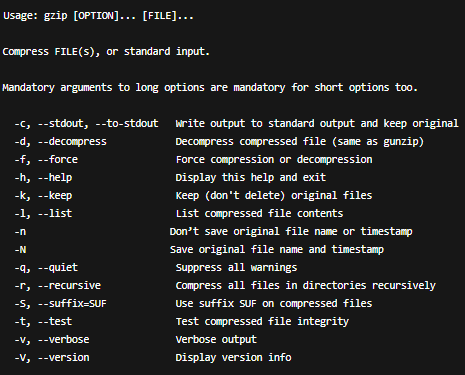
* **tar:** It collects multiple files and directories into a single archive file (called a "tarball") for easy storage, transfer, or backup.

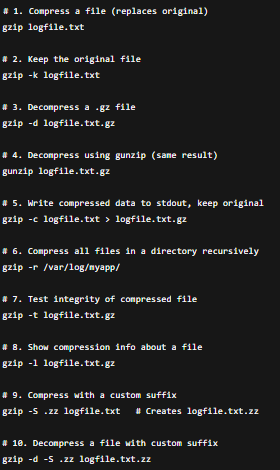
****

****

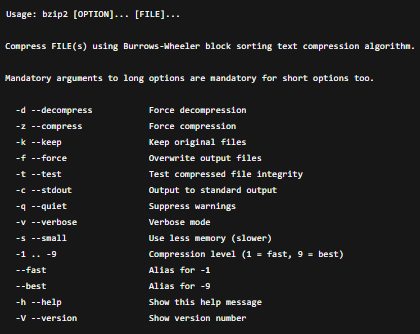
****

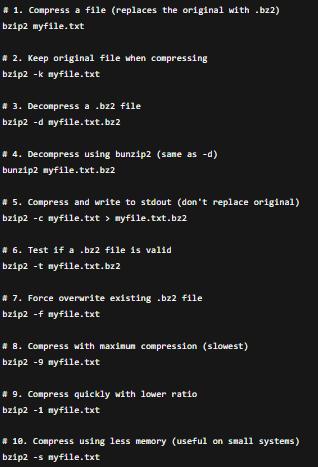
**gzip:** stands for GNU zip, and it is used to compress a single file using the DEFLATE algorithm.  
It reduces the file size for storage or transfer.

* It replaces the original file with a .gz version
* Commonly used with tar to produce .tar.gz archives
* Only compresses one file at a time

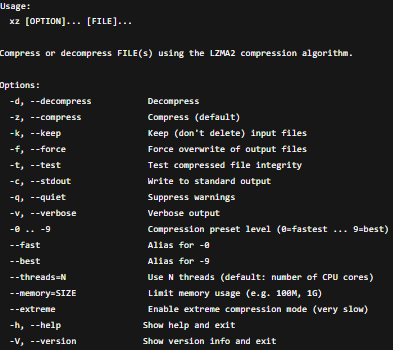
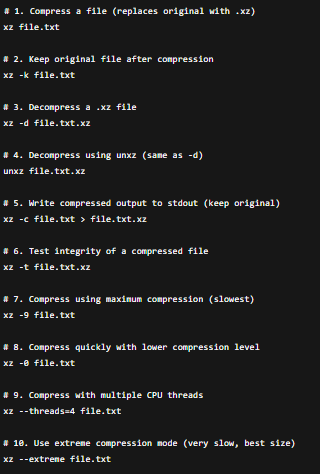
****

* **bzip2:** is a compression tool that compresses one file at a time, using the Burrows–Wheeler algorithm.  
  It typically provides better compression ratios than gzip, but is slower.

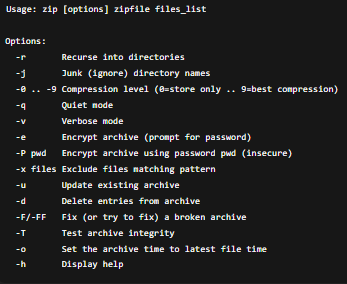
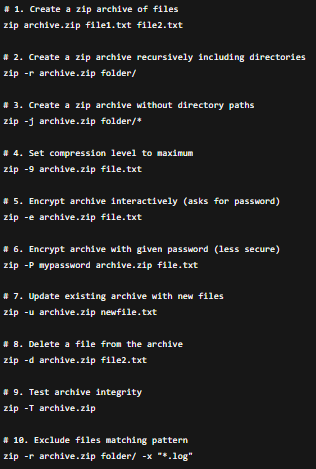
Files compressed with bzip2 usually end with .bz2.



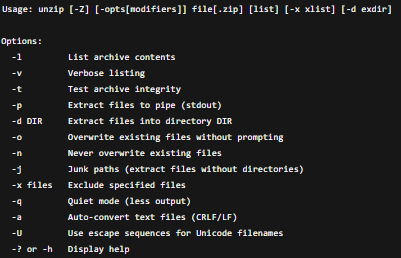
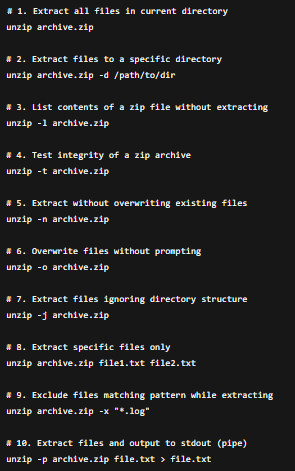
* **xz:** is a compression tool that uses the LZMA2 algorithm, delivering higher compression ratios than gzip and bzip2 but at the cost of slower compression times.

Files compressed by xz typically have the .xz extension.

* **zip:** compresses files individually and stores them in a single archive file (.zip).  
  Supports compression, encryption, and file comments.  
  Very handy when you want archives that are easily opened on Windows, macOS, and Linux.



* **unzip:** extracts files from .zip archives, It can also list archive contents, test integrity, and selectively extract files.



Part 1

Variables

* 1. **Types of Values Can Be Assigned to Variables in Bash:**

In Bash, all variables are treated as strings by default — even when you assign a number. However, how you use the variable (e.g., in arithmetic, conditionals, etc.) determines how Bash interprets it.

* **String Values (Default):** Strings are the default type in Bash.

You can assign **text or alphanumeric strings** directly:

$ city="Baghdad"

$ message="Hello, my name is $name and I live in $city."

* **Integer Values:** You can assign **integer numbers** as values — Bash treats them as strings **until** you use them in arithmetic.

$ age=25

$ total=$((age + 5))

Bash does not support **floating-point** math natively.

* **Boolean-like Usage:** Bash doesn’t have a native Boolean type (true, false), but you can simulate it:

$ flag=true

You can also use 0 and 1 for success/failure semantics.

* **Command Substitution:** You can set a variable to the output of a command using `command` or $(command).

$ current\_date=$(date)

$ echo "Today is $current\_date"

* **Array Values:** Bash supports one-dimensional indexed arrays:

$ fruits=("apple" "banana" "cherry")

$ echo ${fruits[1]} # banana

Arrays are not used as often in basic scripts but are very powerful in automation logic.

* 1. **Types of Variables in Bash:**

1. **User-defined variables:** Created by the user inside scripts or the terminal.
2. **Environment (System) variables:** Provided by the shell or OS and available system-wide.
3. **Special shell variables:** Automatically set by the shell to control behaviors or hold runtime information.

**1- User-defined variables:**

* **Local Variables (Default Behavior**): These are the standard variables you define in your shell or script.
* Exist only in the current shell/session.
* Not available to subprocesses.
* Lost when the shell or script ends.

$ greeting="Hello"

* **Read-Only Variables:** These are constants. Once set, their values cannot be changed or unset.

$ readonly pi=3.14

$ pi=3.14159 # Error

* **Exported Variables:** are user-defined, but promoted to environment variables using export, so child processes can inherit them.

$ export PATH="$PATH:/opt/mytools"

* **Unset Variables (Deleted):** A user-defined variable can be removed entirely using unset.

$ name="Obaida"

$ unset name

$ echo $name # Nothing will print

Part 2

Mathematical and Logical Operations

**2.1- Mathematical Operations**

| **Operator** | **Meaning** | **Example** | **Result** |
| --- | --- | --- | --- |
| **+** | **Addition** | **3 + 2** | **5** |
| **-** | **Subtraction** | **3 - 2** | **1** |
| **\*** | **Multiplication** | **3 \* 2** | **6** |
| **/** | **Division** | **6 / 2** | **3** |
| **%** | **Modulo** | **5 % 2** | **1** |
| **\*\*** | **Exponent** | **2 \*\* 3** | **8** |

**Tools to apply Mathematical Operations:**

**1. $(( expression )):** Most Common & Recommended

Performs integer arithmetic.

$ a=10 ; b=3

$ result=$((a + b))

Supports: + - \* / % \*\*, Cleanest syntax, Recommended for scripts

**2. let:** Works similarly to $(( )), but does not require $ to access variables inside the expression.

$ let result=a\*b

Still widely used, slightly less readable and modern than $(( ))

**3. expr:** Useful in older shells or POSIX scripts. Requires spaces between operators and uses backticks or $( ) to capture output.

$ a=5 ; b=2

$ result=$(expr $a + $b)

Slower than built-in methods, Watch for spacing errors  
Doesn't support \*\* (exponentiation)

**4. bc:** For Floating Point Math (Optional Tool)

Bash can’t handle floats directly. To calculate with decimal numbers, use the bc tool:

$ echo "scale=2; 5 / 3" | bc # Output: 1.66

scale determines the number of decimal places.

**2.2- Logical Operations:**

**1. AND (&&) —** Both conditions must be true

Example 1: Command chaining

$ mkdir /tmp/test && cd /tmp/test #cd will only execute if mkdir succeeds.

Example 2: Inside condition

$ a=15; if [[ $a -gt 10 && $a -lt 20 ]]

**2. OR (||) —** At least one condition must be true

Example 1: Command fallback

$ rm file.txt || echo "File does not exist" #echo runs if rm fails.

Example 2: Inside condition

$ if [[ $a -lt 0 || $a -gt 100 ]]

**3.** NOT (!): Negate a condition

Example: Run if file does *not* exist

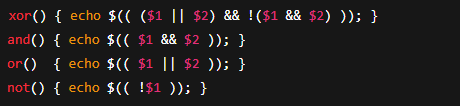
$ if ! [ -f "config.txt" ]; then

Another Example: Not equal check

$ user="guest" ; if [[ ! $user == "admin" ]]

**Boolean Logical Operations:**

* Boolean AND $ (( a && b ))
* Boolean OR $ (( a || b ))
* Boolean NOT $ (( ! a ))



**2.3 Comparison Operations in Bash:**

**Rules of Bash Comparison:**

| Data Type | Syntax Used | Best Tool |
| --- | --- | --- |
| Numbers | [ ], (( )) | Arithmetic |
| Strings | [ ], [[ ]] | String-safe |
| Files | [ ] only | File tests |

**1. Numeric Comparison Operators (Used in [ ] or [[ ]])**

| Operator | Meaning | Example |
| --- | --- | --- |
| -eq | Equal to | [ $a -eq $b ] |
| -ne | Not equal to | [ $a -ne $b ] |
| -gt | Greater than | [ $a -gt $b ] |
| -lt | Less than | [ $a -lt $b ] |
| -ge | Greater or equal | [ $a -ge $b ] |
| -le | Less or equal | [ $a -le $b ] |

**2. Arithmetic Comparison:**

| **Operator** | **Meaning** |
| --- | --- |
| **==** | **Equal** |
| **!=** | **Not equal** |
| **<** | **Less than** |
| **>** | **Greater than** |
| **<=** | **Less or equal** |
| **>=** | **Greater or equal** |

**3. String Comparison (Inside [ ] or [[ ]])**

| Operator | Meaning | Example |
| --- | --- | --- |
| = / == | Equal | [[ $a == $b ]] |
| != | Not equal | [[ $a != $b ]] |
| < | Less (alphabetical) | [[ $a < $b ]] |
| > | Greater (alphabetical) | [[ $a > $b ]] |
| -z | String is empty | [[ -z $a ]] |
| -n | String is not empty | [[ -n $a ]] |

**4. File Comparison Operators: These are for checking file properties:**

| Operator | Checks If... |
| --- | --- |
| -w | File is writable |
| -x | File is executable |
| file1 -nt file2 | file1 is newer than file2 |
| file1 -ot file2 | file1 is older than file2 |

| Operator | Checks If... |
| --- | --- |
| -e | File exists |
| -f | Regular file |
| -d | Directory |
| -s | File is not empty |
| -r | File is readable |

**Part 3**

**if Statements**

**Basic Syntax:**

$ if CONDITION; then

$ # commands if true

$ fi

**You can also add else and elif (else-if):**

$ if CONDITION; then

$ # if true

$ elif OTHER\_CONDITION; then

$ # if first false, this is checked

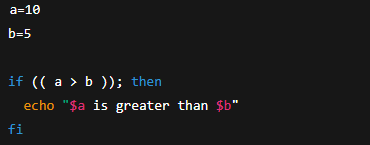
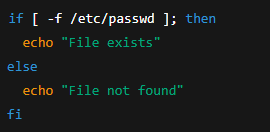
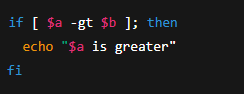
$ else

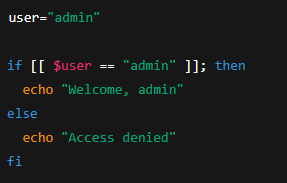
$ # if all conditions fail

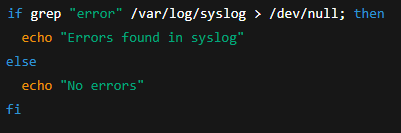
$ fi

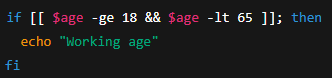
**Accepted CONDITION Types:**

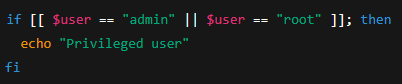
| **Syntax** | **Use When...** | **Example** |
| --- | --- | --- |
| [ ] | Basic POSIX test | [ $a -eq 5 ] |
| [[ ]] | Advanced string & logic tests | [[ $user == "root" ]] |
| (( )) | Arithmetic expressions | (( a > b )) |
| command | Exit status-based checks | if grep "error" log.txt |

****

****

****

****

****

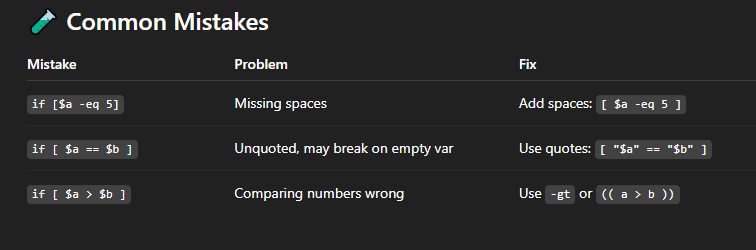
**Notes & Gotchas**

**Don’t forget:**

* Spaces must be around brackets and operators.
* Quote variables to avoid word-splitting:



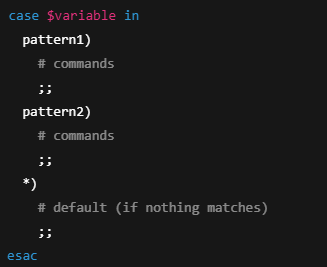
Don’t use == inside [ ] on all systems — some require just =.

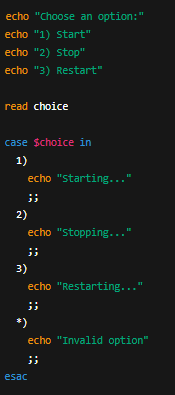


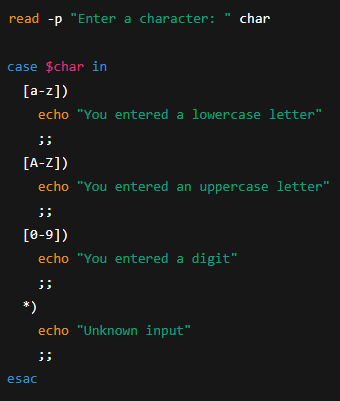
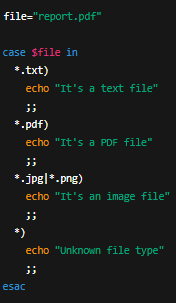
**Part 4**

**case Statement**

**Syntax:**

****

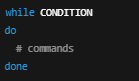
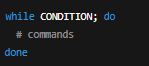
**Examples:**

**bb**

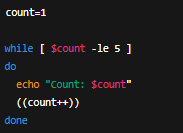
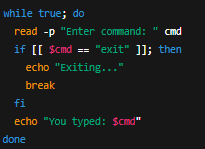
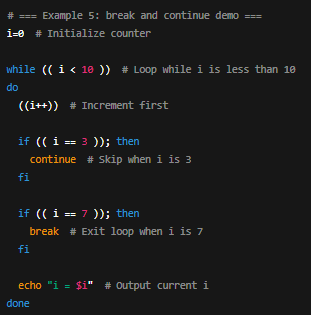
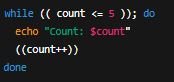
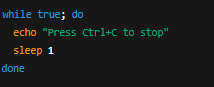
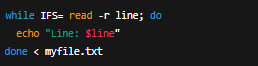
**Part 5:**

**While Loop**

**Syntax:**

** **

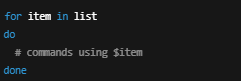
**Examples:**

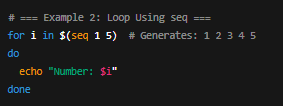
**   **

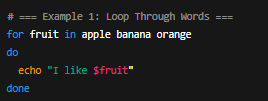
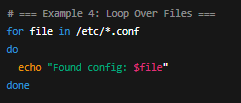
**Part 6**

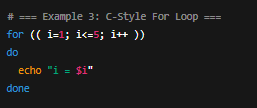
**for Loops**

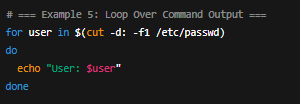
**Syntax:**

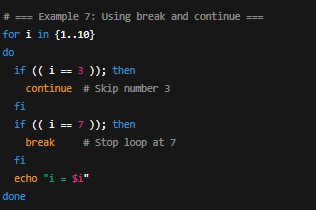
****

**Examples:**

****

****

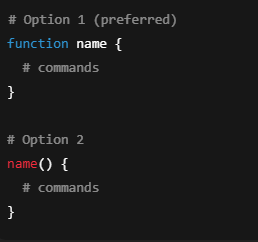
****

****

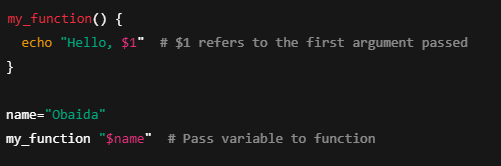
**Part 7**

**Functions**

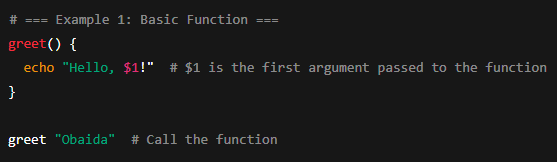
**Syntax**

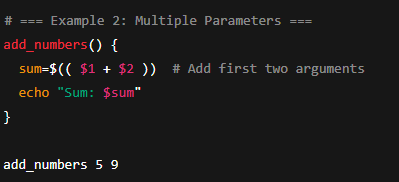
****

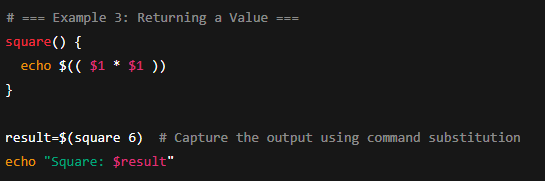
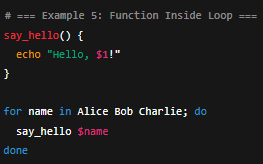
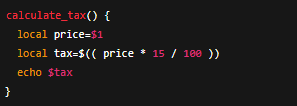
**Passing a variable syntax**

****

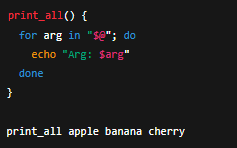
**Examples:**

****

****

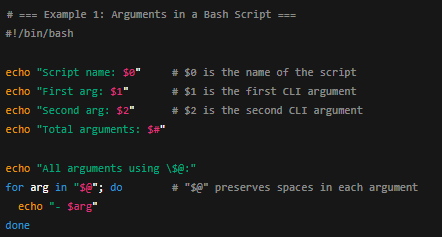
****

**Local variables inside function**

**Access All Arguments in a Loop:**

**Part 8**

**Arguments**

****

**Key and value argument syntax using getopts**

****

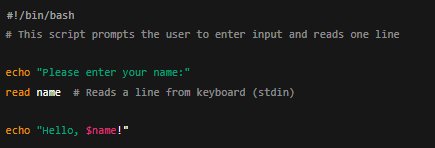
**Part 9**

**Data stream**

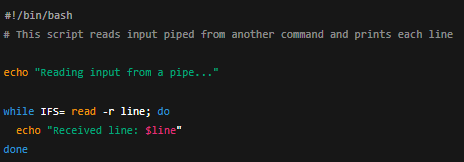
**The Three Standard Data Streams**

|  |  |  |  |
| --- | --- | --- | --- |
| **Stream** | **Description** | **File Descriptor** | **Default Direction** |
| **stdin** | **Standard Input — data going into a command** | **0** | **Keyboard input (usually)** |
| **stdout** | **Standard Output — normal output from command** | **1** | **Screen (terminal)** |
| **stderr** | **Standard Error — error messages output** | **2** | **Screen (terminal)** |

**read**

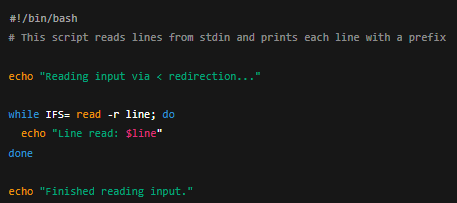
****

**Pipe**

****

****

**<**

****

****

**Stdout will be skipped**

**Stder:** is the standard error stream.

**Why use stderr?**

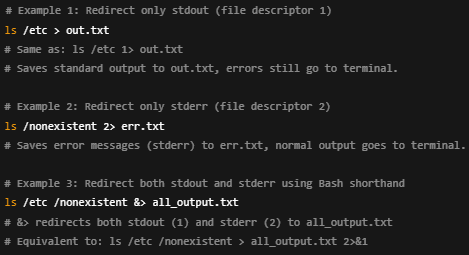
* **Separating errors from normal output allows:**
* **Logging errors separately.**
* **Keeping program output clean.**
* **Handling errors programmatically.**

**Directing the output of a command:**

**1 is for the standard output**

**2 for the standard error**

**& for Both**

****

**Part 10**

**Exit Code**

Every command in Linux returns an exit code (a.k.a. return status) when it finishes.  
This code tells the shell whether the command succeeded or failed.

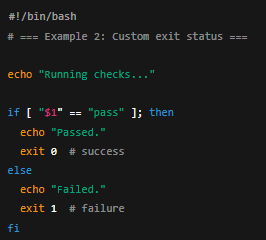
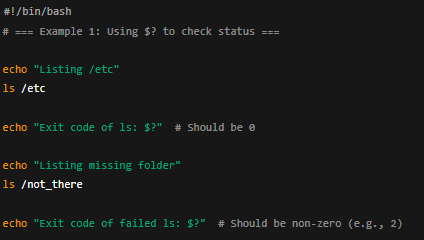
* 0 means success
* Anything else (1–255) means failure

To **get the exit code** of the last command, use:

****

To set the exit code manually in a script, use:

****

****

**Part 11**

**Where to Store Bash Scripts and**

**How to Run Them Like Commands**

**Why This Matters:**

Saving your scripts in the right place and setting them up properly allows you to:

* Run them from anywhere on your system
* Organize your tools and automation clearly
* Avoid polluting system directories

**Common Directories for Saving Scripts**

| Path | Purpose |
| --- | --- |
| /usr/local/bin/ | Preferred system-wide location for user scripts |
| /usr/bin/ | System-managed binaries (avoid putting personal scripts here) |
| ~/bin/ | Per-user scripts (great for personal use) |
| Any custom folder (e.g., ~/scripts/) | Just add it to your $PATH to make scripts globally accessible |