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

1.1 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.

1.2 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).

---

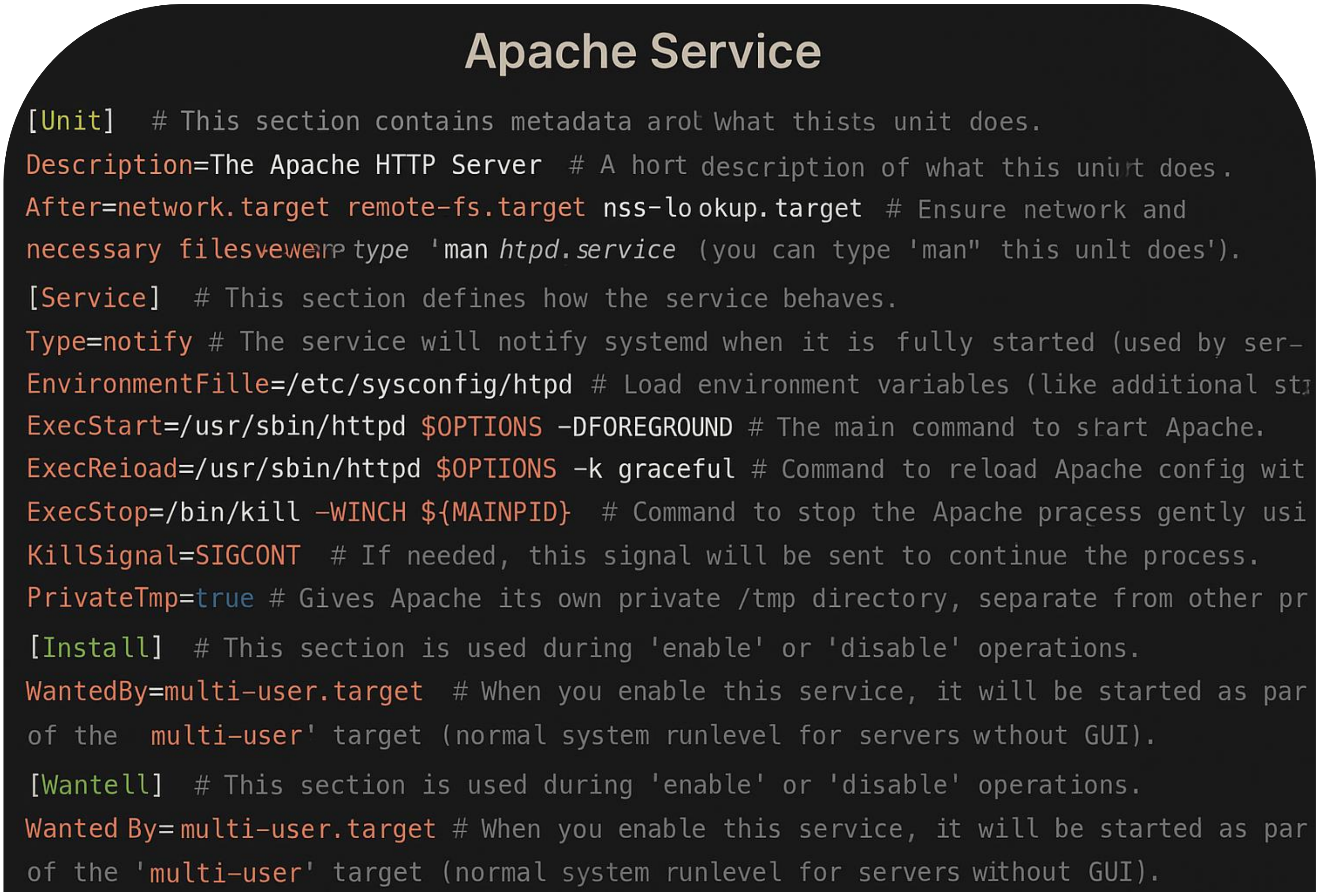
2. [Service] Section (for .service units)

- **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:



Apache Service

```
[Unit]  # This section contains metadata arot What thists unit does.
Description=The Apache HTTP Server  # A hort description of what this uniut does.
After=network.target remote-fs.target nss-lo okup.target  # Ensure network and
necessary filesvewenp type 'man htpd.service (you can type 'man" this unlt does').
[Service]  # This section defines how the service behaves.
Type=notify # The service will notify systemd when it is fully started (used by ser-
EnvironmentFille=/etc/sysconfig/htpd # Load environment variables (like additional st
ExecStart=/usr/sbin/httpd $OPTIONS -DFOREGROUND # The main command to start Apache.
ExecReioad=/usr/sbin/httpd $OPTIONS -k graceful # Command to reload Apache config wit
ExecStop=/bin/kill -WINCH ${MAINPID}  # Command to stop the Apache pracess gently usi
KillSignal=SIGCONT  # If needed, this signal will be sent to continue the process.
PrivateTmp=true # Gives Apache its own private /tmp directory, separate from other pr
[Install]  # This section is used during 'enable' or 'disable' operations.
WantedBy=multi-user.target  # When you enable this service, it will be started as par
of the  multi-user' target (normal system runlevel for servers wthout GUI).
[Wantell]  # This section is used during 'enable' or 'disable' operations.
Wanted By=multi-user.target # When you enable this service, it will be started as par
of the 'multi-user' target (normal system runlevel for servers without GUI).
```

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

3.1 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**

2. Partition Viewing and Monitoring commands:

- **lsblk**
- **blkid**

3. 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**.

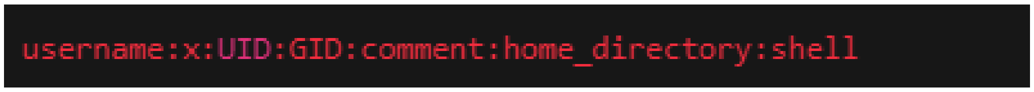
4.2- Linux User & Group Related Files:

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

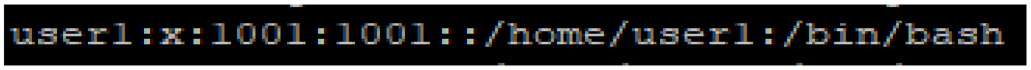
1- /etc/passwd: file is a critical system file that stores essential information about every user account on the Linux system.

- 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:x:UID:GID:comment:home_directory:shell
```



```
user1:x:1001:1001::/home/user1:/bin/bash
```

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.

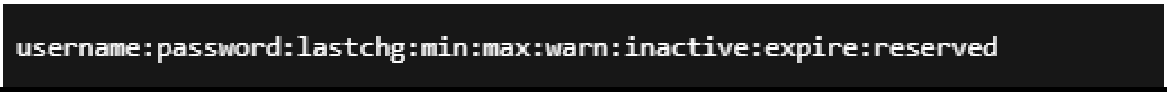
GID (Group ID): The primary group ID associated with the user. This links to an entry in the /etc/group file.

comment (GECOS field): optional information about the user

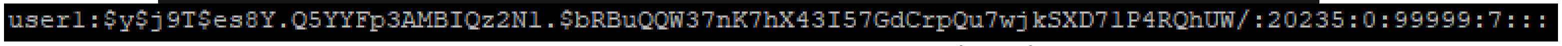
home_directory: The absolute path to the user's home directory.

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:password:lastchg:min:max:warn:inactive:expire:reserved
```



```
user1:$y$j9T$es8Y.Q5YYFp3AMBIQz2N1.$bRBuQQW37nK7hX43I57GdCrpQu7wjkSXD71P4RQhUW/:20235:0:99999:7:::
```

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.

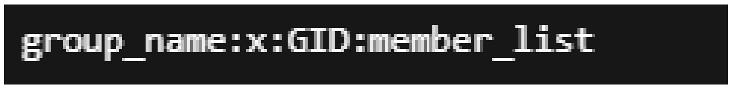
inactive (Inactive Period): The number of days after a password expires before the account is disabled.

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

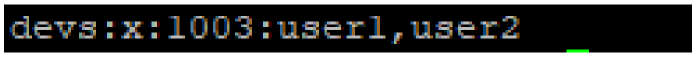
Reserved: Reserved for future use — normally left empty.

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:x:GID:member_list
```



```
devs:x:1003:user1,user2
```

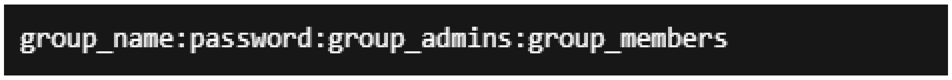
group_name: The name of the group. Example: devs.

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

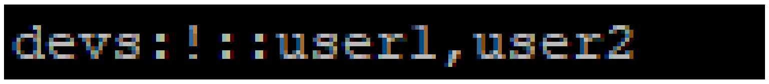
GID (Group ID); A unique number that identifies the group in the system. Example: 1002.

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:password:group_admins:group_members
```



```
devs:!::user1,user2
```

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:

```
-rwxr-xr--
```

- 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.
2. **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.
3. **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
4. **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.

5. **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).

6. **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.

7. **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)

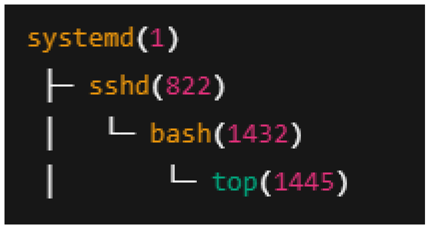
---

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
```



```
systemd(1)
 ├─ sshd(822)
 |    └─ bash(1432)
 |         └─ top(1445)
```

---

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.

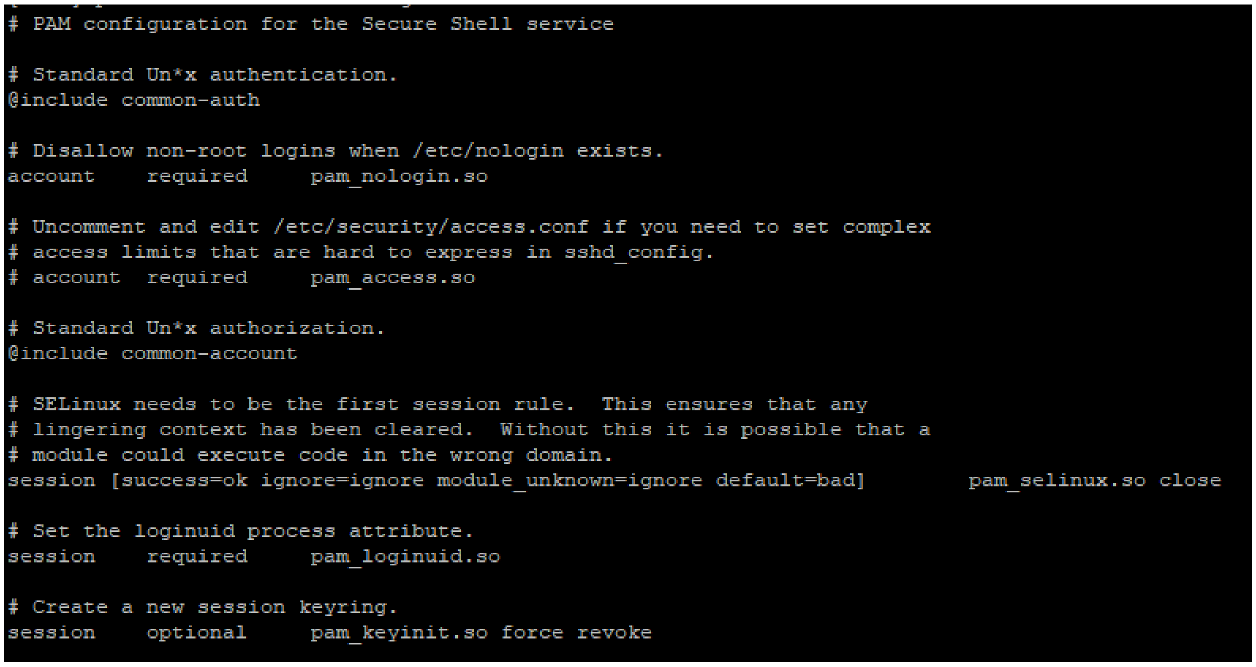
PAM configuration files are located in:

/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:



```
# PAM configuration for the Secure Shell service

# Standard Un*x authentication.
@include common-auth

# Disallow non-root logins when /etc/nologin exists.
account    required     pam_nologin.so

# Uncomment and edit /etc/security/access.conf if you need to set complex
# access limits that are hard to express in sshd_config.
# account  required     pam_access.so

# Standard Un*x authorization.
@include common-account

# SELinux needs to be the first session rule.  This ensures that any
# lingering context has been cleared.  Without this it is possible that a
# module could execute code in the wrong domain.
session [success=ok ignore=ignore module_unknown=ignore default=bad]        pam_selinux.so close

# Set the loginuid process attribute.
session    required     pam_loginuid.so

# Create a new session keyring.
session    optional     pam_keyinit.so force revoke
```

Each file consists of lines like:

```
<type>   <control>   <module-path>   [arguments]
```

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:

```
auth required pam_unix.so
```

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:

```
auth required pam_faillock.so preauth silent deny=3 unlock_time=600
auth [success=1 default=bad] pam_unix.so
auth [default=die] pam_faillock.so authfail
```

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.

```
ls -l report.txt
-rw-r--r-- 1 obaida engineers 2048 Jun 28 10:00 report.txt
```

- 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

```
SELINUX=enforcing   # or permissive/disabled
```

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:role:type:level
```

```
ls -Z /var/www/html/index.html
-rw-r--r--. root root unconfined_u:object_r:httpd_sys_content_t:s0 index.html
```

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:

```
/usr/sbin/apache2 {
  # Allow read access to config files
  /etc/apache2/** r,
  # Allow read/write to logs
  /var/log/apache2/** rw,
  # Deny everything else by default
}
```

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.

2. **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:

```
/var/log/messages          # General system messages (non-systemd)
/var/log/syslog            # Generic system logs (Debian/Ubuntu)
/var/log/auth.log          # Authentication logs (logins, sudo, ssh)
/var/log/kern.log          # Kernel messages
/var/log/dmesg             # Boot-time kernel messages
/var/log/boot.log          # Boot service status
/var/log/faillog           # Failed login attempts
/var/log/Xorg.0.log        # GUI/display logs
/var/log/httpd/            # Apache logs (or nginx, mysql, etc.)
```

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

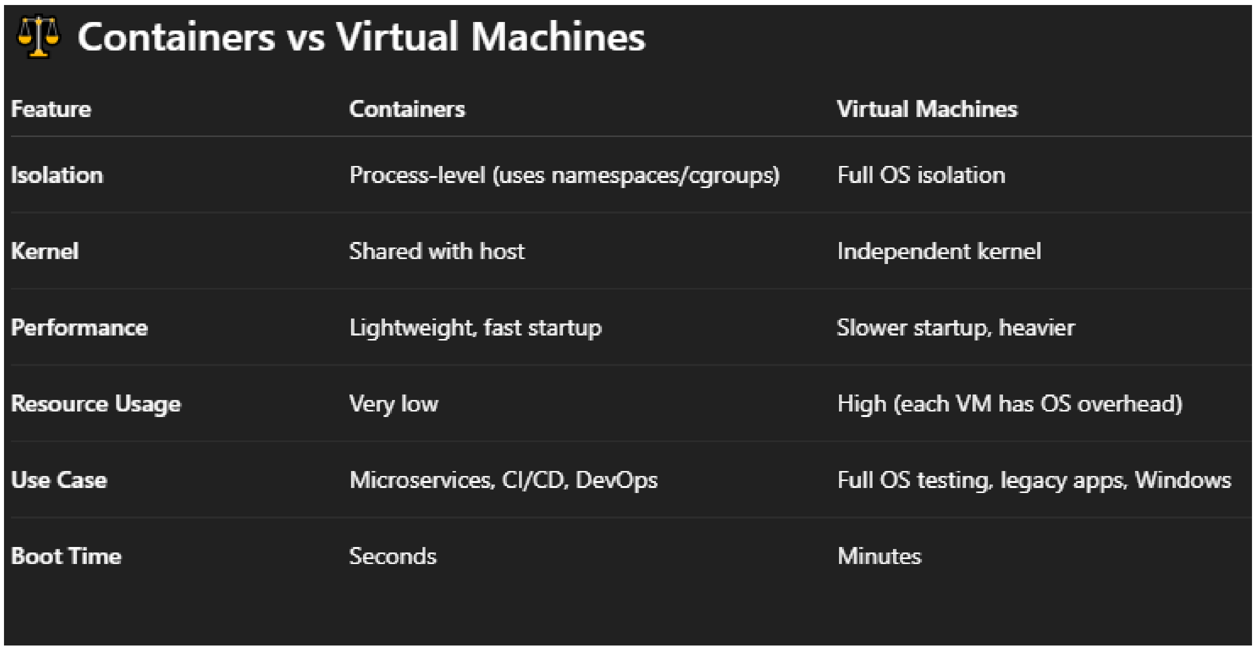
NAT: Default. Guest accesses external network via host. Simple setup.

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



⚖️ Containers vs Virtual Machines

Feature	Containers	Virtual Machines
Isolation	Process-level (uses namespaces/cgroups)	Full OS isolation
Kernel	Shared with host	Independent kernel
Performance	Lightweight, fast startup	Slower startup, heavier
Resource Usage	Very low	High (each VM has OS overhead)
Use Case	Microservices, CI/CD, DevOps	Full OS testing, legacy apps, Windows
Boot Time	Seconds	Minutes

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

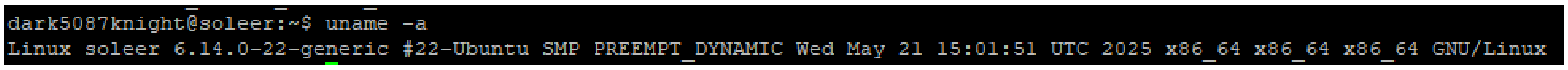
1.1- system information commands:

- **whoami**: command prints the current user’s username.

```
whoami [OPTION]

--help      -- display help message and exit
--version   -- show version info and exit
```

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

```
uname [OPTION]...

-a   --all               -- print all system info
-s   --kernel-name       -- print the kernel name
-n   --nodename          -- print the network node hostname
-r   --kernel-release    -- print the kernel release
-v   --kernel-version    -- print the kernel version
-m   --machine           -- print the machine hardware name (e.g. x86_64)
-p   --processor         -- print the processor type (may show "unknown")
-i   --hardware-platform -- print the hardware platform (may show "unknown")
-o   --operating-system -- print the OS name
--help                   -- show help and exit
--version                -- show version and exit
```



```
dark5087knight@soleer:~$ uname -a
Linux soleer 6.14.0-22-generic #22-Ubuntu SMP PREEMPT_DYNAMIC Wed May 21 15:01:51 UTC 2025 x86_64 x86_64 x86_64 GNU/Linux
```

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.

```
hostname [OPTION]... [new_hostname]

--help                  -- display help and exit
--version               -- output version info and exit

-s, --short             -- show short hostname
-a, --alias             -- show the alias name (rarely set)
-f, --fqdn              -- show the fully qualified domain name
-A, --all-fqdns         -- show all FQDNs (if multiple interfaces)
-i, --ip-address        -- show the IP address(es) for the host (legacy, not reliable)
-I, --all-ip-addresses  -- show all network interfaces' IPs (more reliable)
-d, --domain            -- show domain name
-y, --yp, --nis         -- show NIS/YP domain name
```

- **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.

```
hostnamectl [OPTIONS...] COMMAND...

--help                  -- show help message and exit
--version               -- show version info and exit
--no-legend             -- do not print legend
--no-pager              -- do not pipe output into a pager
--transient             -- operate on transient hostname
--static                -- operate on static hostname
--pretty                -- operate on pretty hostname

Commands:
status                  -- show current hostname and related info
set-hostname [NAME]     -- set static hostname (requires root)
set-icon-name [NAME]    -- set icon name for the system
set-chassis [NAME]      -- set chassis type (desktop, laptop, server, etc.)
set-deployment [NAME]   -- set deployment environment (production, testing, etc.)
set-location [NAME]     -- set physical location info
```

1.2- Navigation commands:

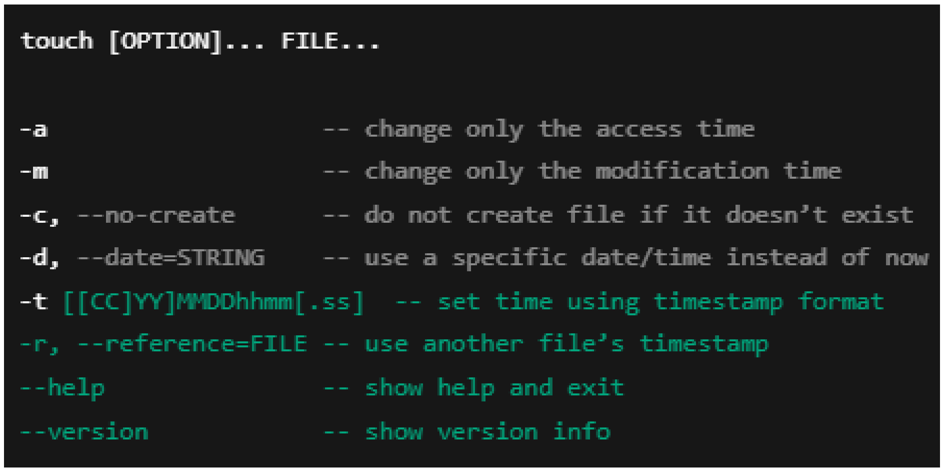
- **pwd (print working directory):** displays the full absolute path of the current directory you're in.

```
pwd [OPTION]...

-L, --logical       -- show the logical path (respects symlinks)
-P, --physical      -- show the physical path (resolves symlinks)
--help              -- display help and exit
--version           -- output version info and exit
```

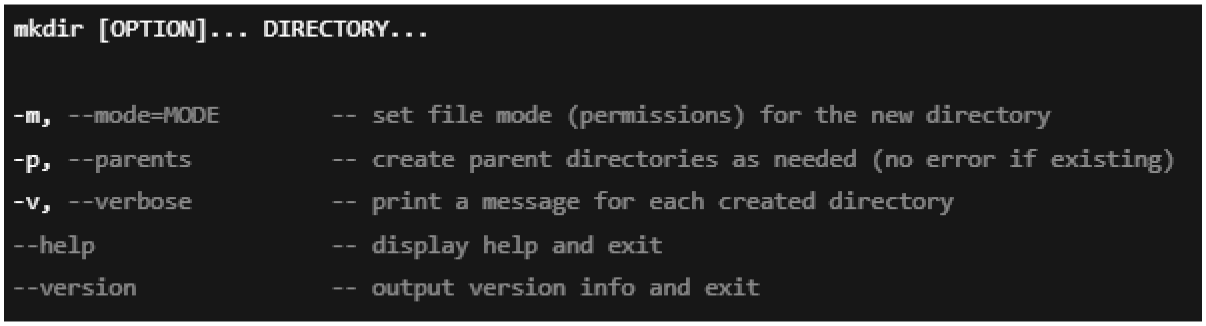
```
cd [DIRECTORY]

# Common special arguments
cd ~                -- go to the current user's home directory
cd /path/to/dir     -- go to an absolute path
cd ..               -- go up one level (parent directory)
cd ../..            -- go up two levels
cd -                -- switch to the previous working directory
cd ~user            -- go to another user’s home directory (if allowed)
```

```
ls [OPTION]... [FILE]...

-a, --all                  -- show hidden files (those starting with .)
-A, --almost-all           -- like -a, but omits . and ..
-l                         -- long listing format (shows permissions, size, date, owner)
-h, --human-readable       -- with -l, show sizes like 1K, 234M
-S                         -- sort by file size (largest first)
-t                         -- sort by modification time (newest first)
-r                         -- reverse order while sorting
-R                         -- list directories recursively
-d                         -- list directory itself, not its contents
-F                         -- append indicator (/ for dir, * for executable, etc.)
-i                         -- show inode numbers
--color=auto               -- color output (usually default)
--group-directories-first  -- put folders before files

--help                     -- show help
--version                  -- show version info
```

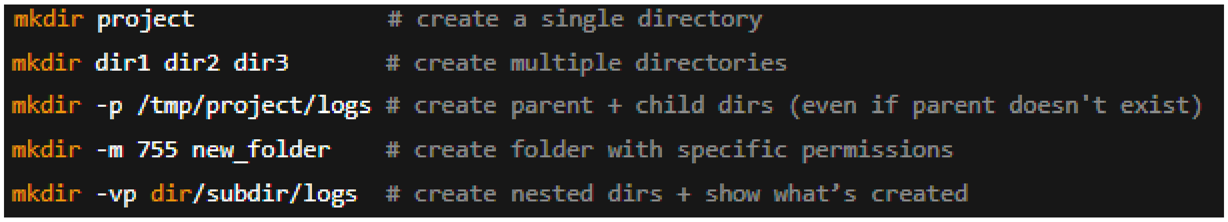
1.3- files and directories management commands:



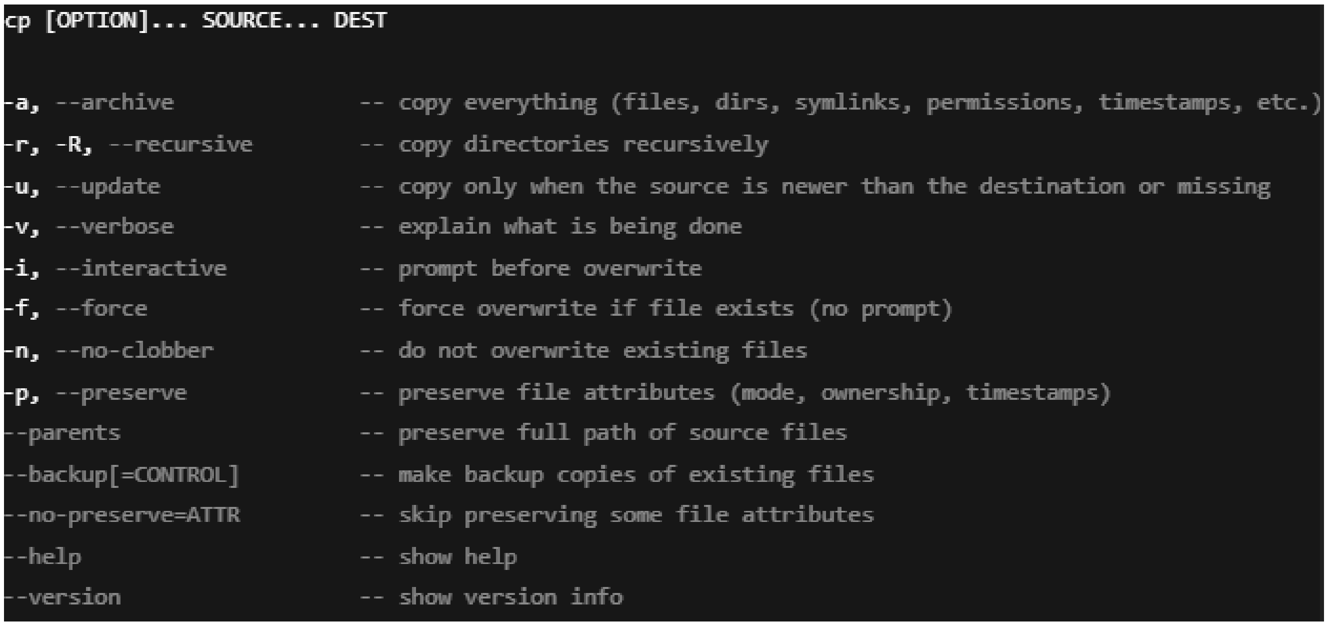
```
touch [OPTION]... FILE...

-a                     -- change only the access time
-m                     -- change only the modification time
-c, --no-create        -- do not create file if it doesn’t exist
-d, --date=STRING      -- use a specific date/time instead of now
-t [[CC]YY]MMDDhhmm[.ss]  -- set time using timestamp format
-r, --reference=FILE -- use another file’s timestamp
--help                 -- show help and exit
--version              -- show version info
```



```
mkdir [OPTION]... DIRECTORY...

-m, --mode=MODE         -- set file mode (permissions) for the new directory
-p, --parents           -- create parent directories as needed (no error if existing)
-v, --verbose           -- print a message for each created directory
--help                  -- display help and exit
--version               -- output version info and exit
```



```
mkdir project             # create a single directory
mkdir dir1 dir2 dir3      # create multiple directories
mkdir -p /tmp/project/logs # create parent + child dirs (even if parent doesn't exist)
mkdir -m 755 new_folder    # create folder with specific permissions
mkdir -vp dir/subdir/logs  # create nested dirs + show what’s created
```



```
cp [OPTION]... SOURCE... DEST

-a, --archive             -- copy everything (files, dirs, symlinks, permissions, timestamps, etc.)
-r, -R, --recursive       -- copy directories recursively
-u, --update              -- copy only when the source is newer than the destination or missing
-v, --verbose             -- explain what is being done
-i, --interactive         -- prompt before overwrite
-f, --force               -- force overwrite if file exists (no prompt)
-n, --no-clobber          -- do not overwrite existing files
-p, --preserve            -- preserve file attributes (mode, ownership, timestamps)
--parents                 -- preserve full path of source files
--backup[=CONTROL]        -- make backup copies of existing files
--no-preserve=ATTR        -- skip preserving some file attributes
--help                    -- show help
--version                 -- show version info
```

```
mv [OPTION]... SOURCE... DEST

-i, --interactive     -- prompt before overwrite
-f, --force           -- overwrite without prompt (default if -i not used)
-n, --no-clobber      -- do not overwrite existing files
-v, --verbose         -- show what's being moved
--backup[=CONTROL]    -- create backup before overwrite
--suffix=SUFFIX       -- override default backup suffix
--strip-trailing-slashes  -- remove trailing slashes in source
--target-directory=DIRECTORY  -- move all sources into the directory
--help                -- show help and exit
--version             -- show version and exit
```

```
mv file.txt /home/obaida/docs/       # move file to another directory
mv notes.txt notes_old.txt           # rename a file
```

---

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

```
file [OPTION]... FILE...

-b, --brief              -- do not prepend filenames to output lines
-c, --checking-print     -- print parsed content of magic file for debugging
-f FILE                  -- read filenames from FILE, one per line
-i, --mime               -- output MIME type strings instead of human-readable info
-L, --dereference        -- follow symlinks to determine file type
-s, --special-files      -- read special files (block or char devices)
--help                   -- show help and exit
--version                -- show version info
```

```
rm [OPTION]... FILE...

-f, --force             -- ignore nonexistent files, never prompt
-i                      -- prompt before every removal
-I                      -- prompt once before removing more than three files or recursively
-r, -R, --recursive     -- remove directories and their contents recursively
-d, --dir               -- remove empty directories
--one-file-system       -- when removing recursively, don't cross filesystem boundaries
--preserve-root         -- do not remove '/' (default)
--no-preserve-root      -- allow removal of '/'
--help                  -- display help and exit
--version               -- output version info and exit
```

```
rm file.txt                    # delete a single file
rm -i file.txt                 # prompt before deleting file.txt
rm -rf /tmp/myfolder           # force delete a directory and all contents recursively
rm -r dir1 dir2                # delete directories recursively with prompt depending on flags
rm -f *.log                   # force delete all .log files without prompt
rm -v file.txt                # verbose delete to see what's removed
```

```
rmdir [OPTION]... DIRECTORY...

-p, --parents       -- remove DIRECTORY and its ancestors if they are empty
-v, --verbose       -- output a message for each directory removed
--help              -- display help and exit
--version           -- output version info and exit
```

```
rmdir old_folder                # remove a single empty directory
rmdir -v tmp/cache             # verbose remove, will print confirmation
rmdir -p project/logs/temp     # remove temp and its parents if empty
rmdir -p dir1/dir2/dir3        # removes dir3, dir2, and dir1 if all empty
```

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

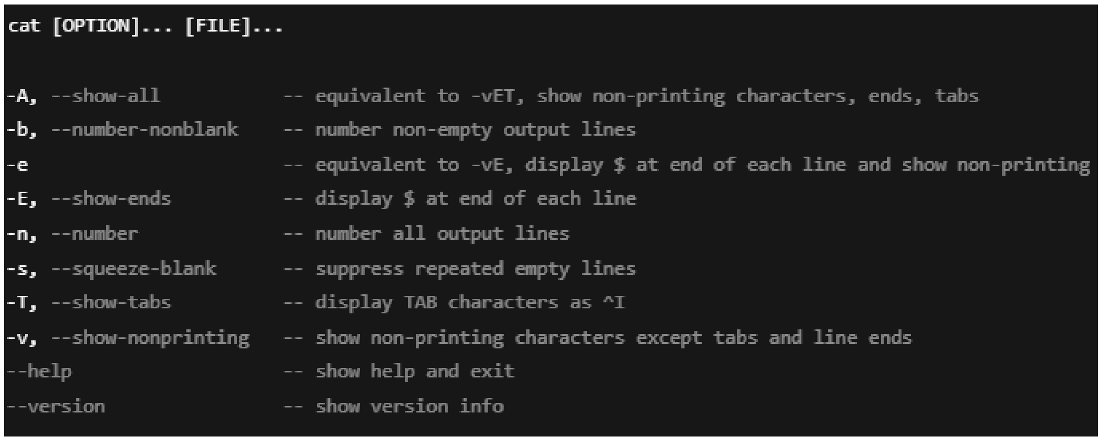
ln creates links between files. These can be:

- **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).

```
ln [OPTION]... TARGET [LINK_NAME]

-s, --symbolic          -- make symbolic links instead of hard links
-f, --force             -- remove existing destination files
-n, --no-dereference    -- treat LINK_NAME as a file even if it’s a symlink
-v, --verbose           -- show what’s being done
--backup[=CONTROL]      -- make a backup of each existing destination file
--help                  -- display help and exit
--version               -- show version info and exit
```

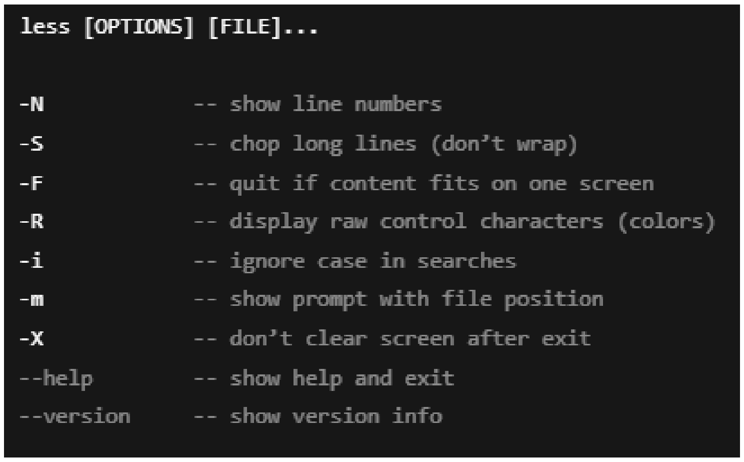
```
ln file.txt link.txt             # create a hard link named link.txt
ln -s /etc/nginx/nginx.conf ~/ng  # create a symlink to nginx config
ln -sf /path/old /path/new        # force-create symlink, overwrite if exists
ln -s $(pwd)/myfile mylink        # create symlink using absolute path
```

1.4- Viewing and displaying files:



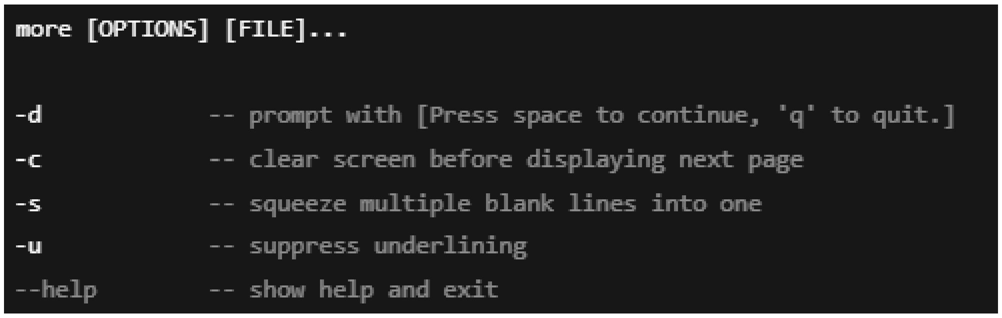
```
cat [OPTION]... [FILE]...

-A, --show-all          -- equivalent to -vET, show non-printing characters, ends, tabs
-b, --number-nonblank   -- number non-empty output lines
-e                      -- equivalent to -vE, display $ at end of each line and show non-printing
-E, --show-ends         -- display $ at end of each line
-n, --number            -- number all output lines
-s, --squeeze-blank     -- suppress repeated empty lines
-T, --show-tabs         -- display TAB characters as ^I
-v, --show-nonprinting  -- show non-printing characters except tabs and line ends
--help                  -- show help and exit
--version               -- show version info
```

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

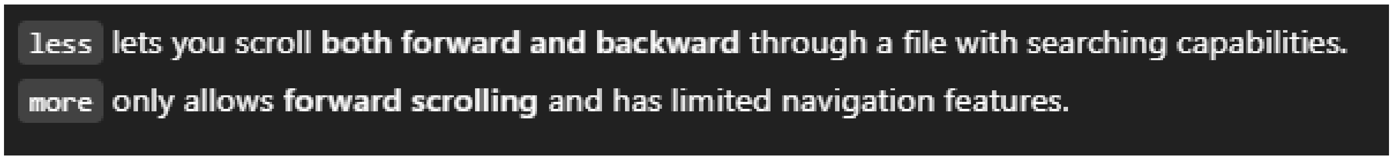


```
less [OPTIONS] [FILE]...

-N          -- show line numbers
-S          -- chop long lines (don’t wrap)
-F          -- quit if content fits on one screen
-R          -- display raw control characters (colors)
-i          -- ignore case in searches
-m          -- show prompt with file position
-X          -- don’t clear screen after exit
--help      -- show help and exit
--version   -- show version info
```

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



```
more [OPTIONS] [FILE]...

-d          -- prompt with [Press space to continue, 'q' to quit.]
-c          -- clear screen before displaying next page
-s          -- squeeze multiple blank lines into one
-u          -- suppress underlining
--help      -- show help and exit
```



`less` lets you scroll **both forward and backward** through a file with searching capabilities.
`more` only allows **forward scrolling** and has limited navigation features.

```
head [OPTION]... [FILE]...

-n, --lines=[NUM]      -- output the first NUM lines, default 10
-c, --bytes=[NUM]      -- output the first NUM bytes instead of lines
-q, --quiet            -- never print headers with file names
-v, --verbose          -- always print headers with file names
--help                 -- show help and exit
--version              -- show version info
```

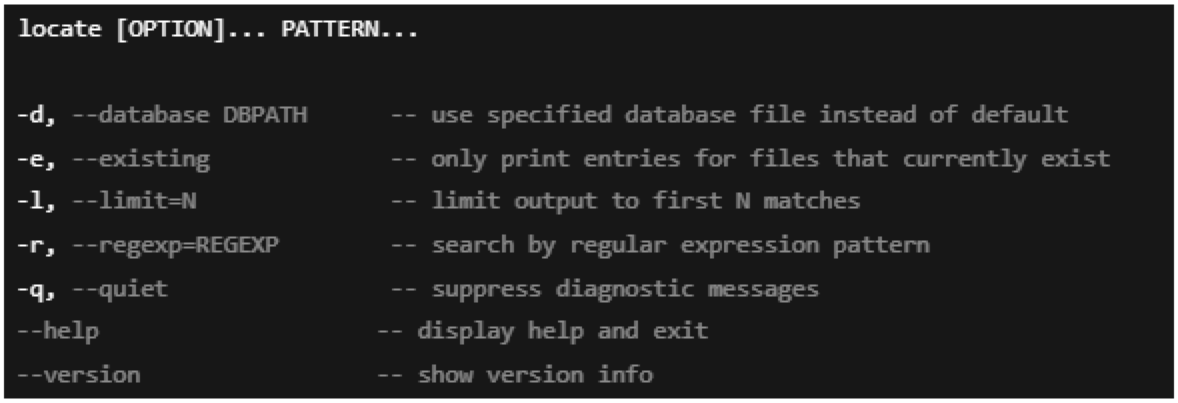
```
tail [OPTION]... [FILE]...

-n, --lines=[NUM]         -- output the last NUM lines, default 10
-c, --bytes=[NUM]         -- output the last NUM bytes instead of lines
-f, --follow              -- output appended data as the file grows (live update)
--pid=PID                 -- with -f, terminate after process PID dies
-q, --quiet, --silent     -- never print headers with file names
-v, --verbose             -- always print headers with file names
--help                    -- show help and exit
--version                 -- show version info
```

- **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.

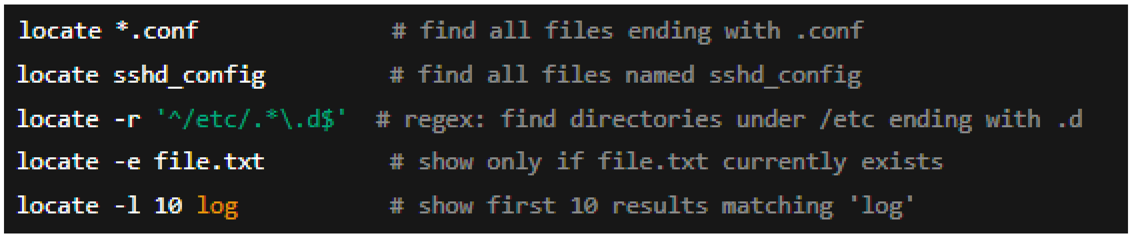
```
find [PATH] [OPTIONS] [EXPRESSION]

# Common expressions:
-name PATTERN              -- match file name (wildcards like *.log)
-type [f|d|l]              -- file (f), directory (d), symbolic link (l)
-size [+|-]N[c/k/M/G]      -- file size (>, <, = N bytes/kilobytes/megabytes/gigabytes)
-user USER                 -- files owned by USER
-group GROUP               -- files belonging to GROUP
-mtime [+|-]N              -- modification time (days ago)
-atime [+|-]N              -- access time (days ago)
-perm MODE                 -- match permissions
-exec COMMAND {} \;        -- execute command on each matched file
-delete                   -- delete matched files (use carefully)
```

```
find /var/log -name "*.log"                # find all .log files in /var/log
find /home -type d -name "backup"          # find directories named backup
find /tmp -type f -size +100M               # find files larger than 100 MB in /tmp
find / -user obaida                         # find all files owned by user obaida
find . -mtime -7                           # find files modified within last 7 days in current
find /tmp -name "*.tmp" -delete            # delete all .tmp files in /tmp (dangerous!)
find /var/www -name "*.php" -exec wc -l {} \;  # count lines of each PHP file in /var/www
```

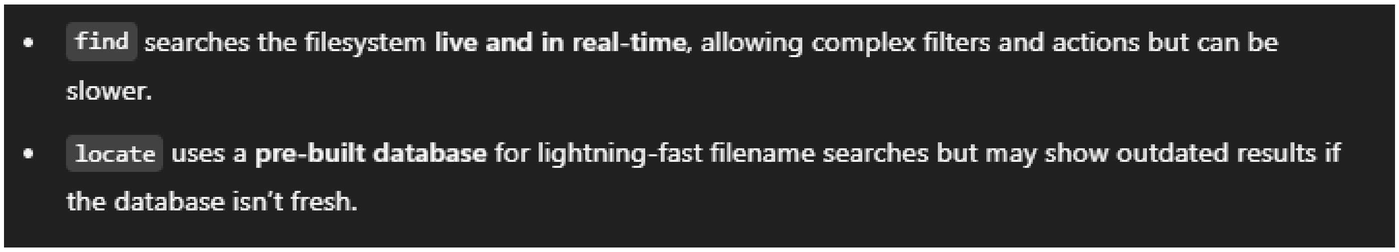
locate: quickly finds files by name using a **pre-built database** (usually updated daily), rather than scanning the filesystem live.



```
locate [OPTION]... PATTERN...

-d, --database DBPATH      -- use specified database file instead of default
-e, --existing             -- only print entries for files that currently exist
-l, --limit=N              -- limit output to first N matches
-r, --regexp=REGEXP        -- search by regular expression pattern
-q, --quiet                -- suppress diagnostic messages
--help                    -- display help and exit
--version                 -- show version info
```



```
locate *.conf              # find all files ending with .conf
locate sshd_config         # find all files named sshd_config
locate -r '^/etc/.*\.d$'  # regex: find directories under /etc ending with .d
locate -e file.txt         # show only if file.txt currently exists
locate -l 10 log           # show first 10 results matching 'log'
```



- `find` searches the filesystem **live and in real-time**, allowing complex filters and actions but can be slower.
- `locate` uses a **pre-built database** for lightning-fast filename searches but may show outdated results if the database isn't fresh.

1.5- File manipulation and searching commands:

```
echo [OPTION]... [STRING]...

-n                 -- do not output the trailing newline
-e                 -- enable interpretation of backslash escapes
-E                 -- disable backslash escape interpretation (default)
--help             -- display help and exit
```

```
echo Hello, Obaida!                       # print a simple message
echo $HOME                                # show value of a variable
echo -n "Loading..."                      # print without a newline
echo -e "Line1\nLine2"                   # print multiple lines with newline escape
echo -e "Path:\t$PATH"                   # show tabbed output
echo "Hello" > file.txt                   # write to file (overwrite)
echo "World" >> file.txt                  # append to file
```

```
grep [OPTIONS] PATTERN [FILE...]

-i, --ignore-case         -- ignore case while matching
-v, --invert-match        -- show lines that do NOT match
-r, --recursive           -- search directories recursively
-l, --files-with-matches  -- list only file names with matches
-L, --files-without-match-- list files with NO matches
-n, --line-number         -- show line numbers of matches
-c, --count               -- show number of matching lines
-o, --only-matching       -- show only matching part of line
-E, --extended-regexp     -- use extended regular expressions
--color=auto              -- highlight matches (usually default)
--help                    -- display help and exit
```

```
grep "error" /var/log/syslog              # find lines containing 'error'
grep -i "Failed" auth.log                 # case-insensitive search for 'Failed'
grep -v "INFO" app.log                    # show lines that do NOT contain 'INFO'
grep -n "user" /etc/passwd                # show line numbers with matches
grep -c "session" auth.log                # count lines with 'session'
grep -r "password" /etc                   # search all files under /etc
grep -l "root" *                          # show filenames that contain 'root'
grep -o "user[0-9]\+" users.txt           # show only matching usernames like user01, user123
```

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

```
cut [OPTION]... [FILE]...

-b, --bytes=LIST          -- select only these bytes
-c, --characters=LIST     -- select only these characters
-d, --delimiter=DELIM     -- use DELIM instead of TAB for field delimiter
-f, --fields=LIST         -- select only these fields (with -d)
--complement              -- complement the selection (invert result)
--output-delimiter=STR    -- change output delimiter
--help                    -- display help and exit
```

```
cut -d: -f1 /etc/passwd            # get usernames (1st field) from /etc/passwd
cut -d',' -f2 users.csv            # extract 2nd column from CSV file
cut -c1-5 file.txt                 # show first 5 characters of each line
cut -d' ' -f1 logfile.txt          # extract first word (field) per line
ps aux | cut -c 1-10               # show first 10 characters of each line in process list
```

---

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.

```
sed [OPTION]... 'SCRIPT' [FILE]...

-n, --quiet              -- suppress automatic printing
-e SCRIPT                -- add the script to be executed
-f SCRIPT_FILE           -- take script from a file
-i[SUFFIX]               -- edit files in-place (backup with SUFFIX)
--help                   -- show help and exit
--version                -- show version info
```

```
sed 's/error/ERROR/' logfile.txt          # replace first "error" with "ERROR" per line
sed 's/error/ERROR/g' logfile.txt         # replace all "error" with "ERROR" per line
sed -i 's/http/https/g' index.html        # edit file in-place, replacing http with https
sed -n '5p' file.txt                      # print only line 5
sed -n '3,7p' file.txt                    # print lines 3 to 7
sed '/^#/d' config.conf                   # delete lines starting with #
sed '/^$/d' file.txt                      # delete empty lines
echo "abc123" | sed 's/[0-9]//g'          # remove all digits
```

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.

```
awk [OPTIONS] 'pattern { action }' [FILE...]

-F fs          -- define input field separator (default is space or tab)
-v var=val     -- assign variables to use inside awk
-f script.awk  -- run awk script from file
--help         -- show help and exit
```

```
awk '{ print $1 }' file.txt                    # print first field of each line
awk -F: '{ print $1, $3 }' /etc/passwd         # show username and UID
ps aux | awk '{ print $1, $11 }'               # show USER and COMMAND columns
awk '$3 > 50 { print $1, $3 }' data.txt        # print lines where 3rd field > 50
awk 'BEGIN { print "Name\tScore" } { print $1, $2 }' scores.txt
awk 'NR==1 { print "Header:", $0 }' file.txt  # print first line with custom header
```

1.6- Key operators:

```
&    # Run command in background (asynchronously)
&&   # Logical AND: run next command only if previous succeeds
;    # Command separator: run commands sequentially regardless of success
>    # Redirect standard output (overwrite file)
>>   # Redirect standard output (append to file)
|    # Pipe: send output of command on left as input to command on right
||   # Logical OR: run next command only if previous fails
```

2- Systemd

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)

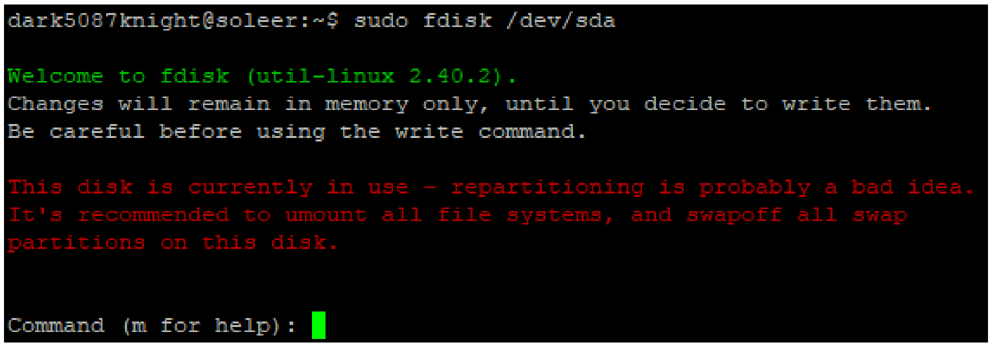
Controls if fdisk wipes (zeros) newly created partitions and old filesystem signatures from partitions.

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

```
$ fdisk -w always /dev/sda
$ fdisk -W never /dev/sda
```

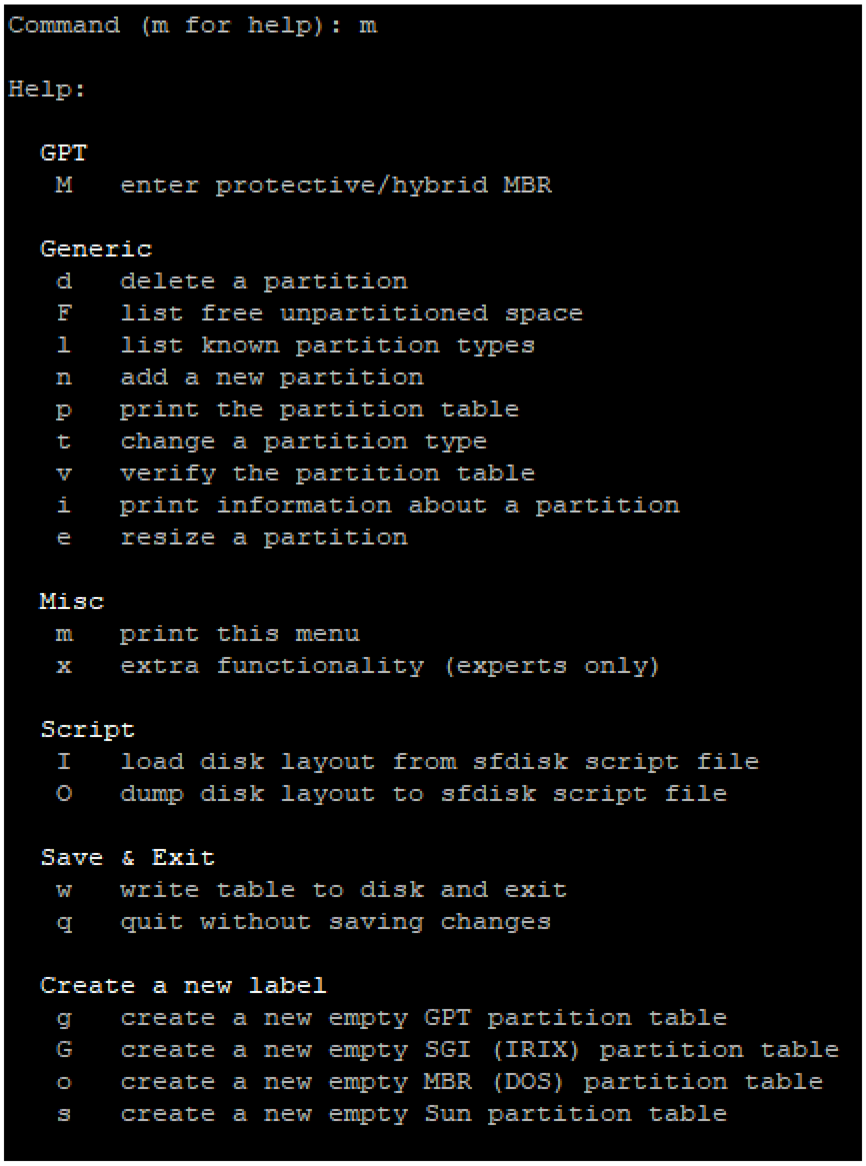
When we use the fdisk command on a disk in we go in an interactive mode so we can crate delete and modify partitions.

$ sudo fdisk /dev/sda



```
dark5087knight@soleer:~$ sudo fdisk /dev/sda

Welcome to fdisk (util-linux 2.40.2).
Changes will remain in memory only, until you decide to write them.
Be careful before using the write command.

This disk is currently in use - repartitioning is probably a bad idea.
It's recommended to umount all file systems, and swapoff all swap
partitions on this disk.


Command (m for help):
```

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



```
Command (m for help): m

Help:

  GPT
   M   enter protective/hybrid MBR

  Generic
   d   delete a partition
   F   list free unpartitioned space
   l   list known partition types
   n   add a new partition
   p   print the partition table
   t   change a partition type
   v   verify the partition table
   i   print information about a partition
   e   resize a partition

  Misc
   m   print this menu
   x   extra functionality (experts only)

  Script
   I   load disk layout from sfdisk script file
   O   dump disk layout to sfdisk script file

  Save & Exit
   w   write table to disk and exit
   q   quit without saving changes

  Create a new label
   g   create a new empty GPT partition table
   G   create a new empty SGI (IRIX) partition table
   o   create a new empty MBR (DOS) partition table
   s   create a new empty Sun partition table
```

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).

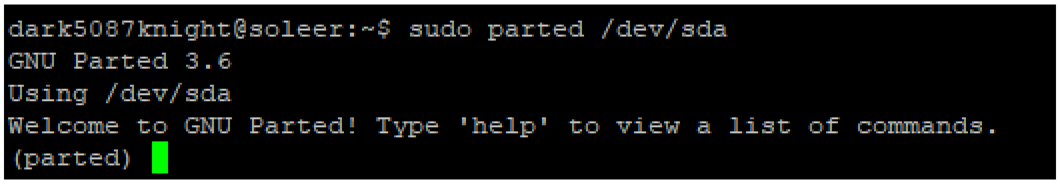
Perfect for:

- 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

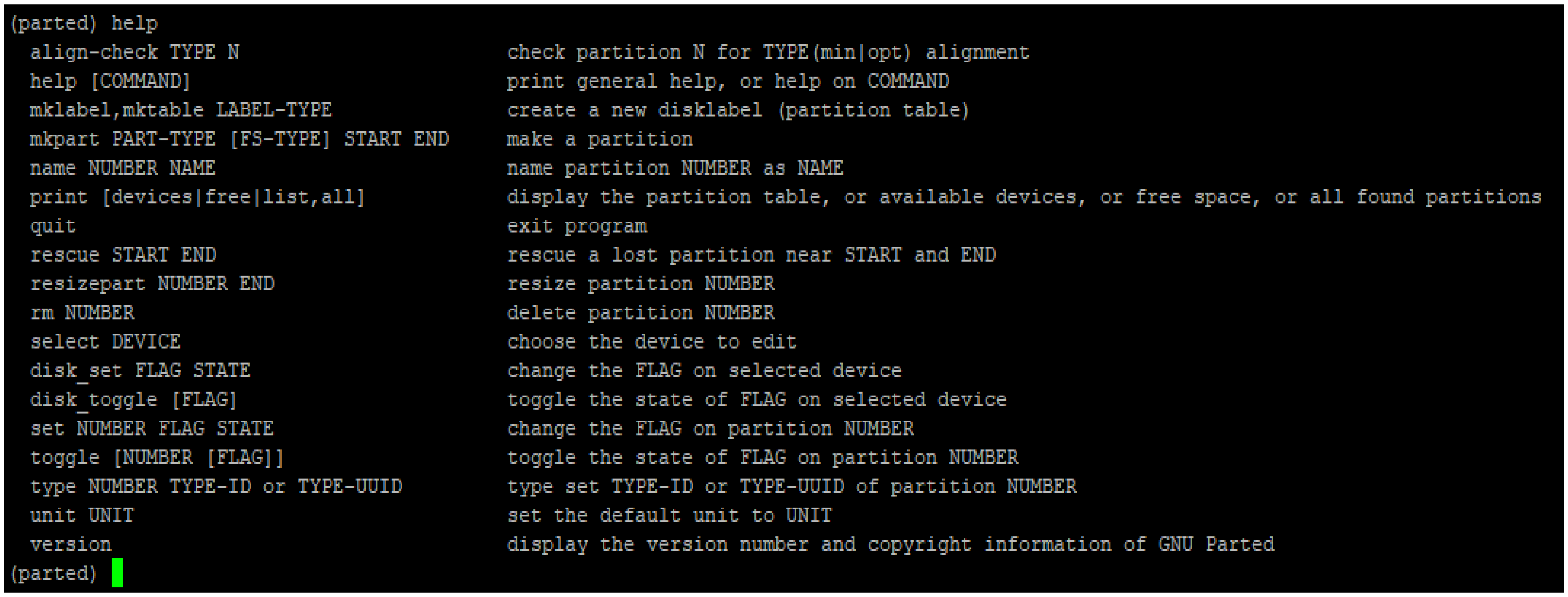
To enter the interactive mode you can simply use parted with the device

`$ sudo parted /dev/sda` and you will be in interactive mode



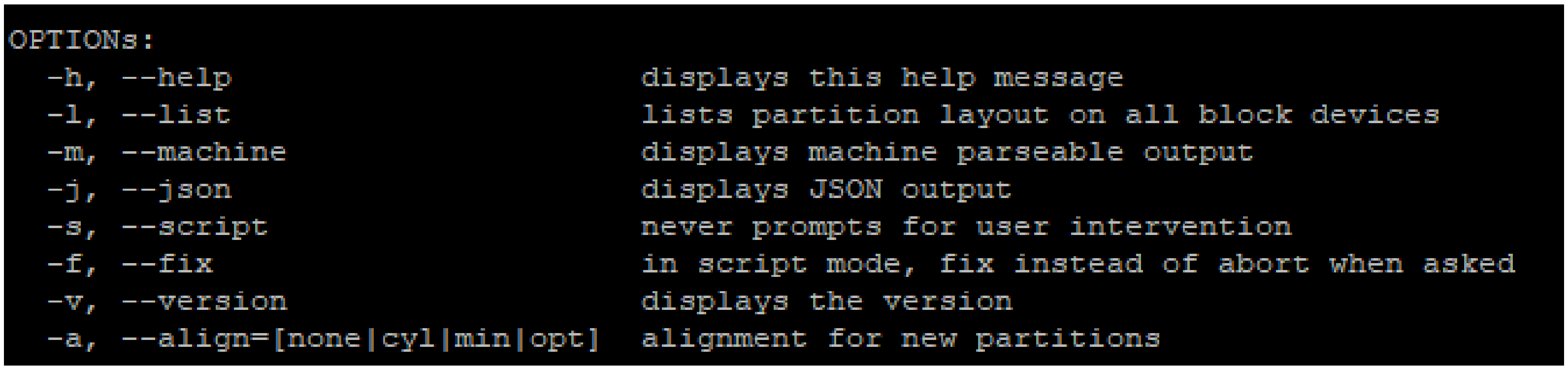
```
dark5087knight@soleer:~$ sudo parted /dev/sda
GNU Parted 3.6
Using /dev/sda
Welcome to GNU Parted! Type 'help' to view a list of commands.
(parted)
```

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



```
(parted) help
  align-check TYPE N                       check partition N for TYPE(min|opt) alignment
  help [COMMAND]                           print general help, or help on COMMAND
  mklabel,mktable LABEL-TYPE               create a new disklabel (partition table)
  mkpart PART-TYPE [FS-TYPE] START END     make a partition
  name NUMBER NAME                         name partition NUMBER as NAME
  print [devices|free|list,all]            display the partition table, or available devices, or free space, or all found partitions
  quit                                     exit program
  rescue START END                         rescue a lost partition near START and END
  resizepart NUMBER END                    resize partition NUMBER
  rm NUMBER                                delete partition NUMBER
  select DEVICE                            choose the device to edit
  disk_set FLAG STATE                      change the FLAG on selected device
  disk_toggle [FLAG]                       toggle the state of FLAG on selected device
  set NUMBER FLAG STATE                    change the FLAG on partition NUMBER
  toggle [NUMBER [FLAG]]                   toggle the state of FLAG on partition NUMBER
  type NUMBER TYPE-ID or TYPE-UUID         type set TYPE-ID or TYPE-UUID of partition NUMBER
  unit UNIT                                set the default unit to UNIT
  version                                  display the version number and copyright information of GNU Parted
(parted)
```

Common options:



```
OPTIONs:
  -h, --help                    displays this help message
  -l, --list                    lists partition layout on all block devices
  -m, --machine                 displays machine parseable output
  -j, --json                    displays JSON output
  -s, --script                  never prompts for user intervention
  -f, --fix                     in script mode, fix instead of abort when asked
  -v, --version                 displays the version
  -a, --align=[none|cyl|min|opt]  alignment for new partitions
```

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.

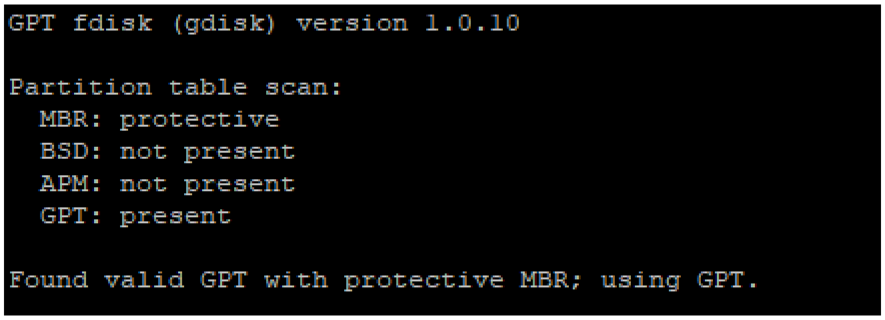
It's like fdisk, but designed specifically for **GPT disks only** (not MBR).
You can use it to **create, delete, resize, and repair GPT partitions**.
Useful for modern systems with UEFI and disks larger than 2TB.

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]


```
GPT fdisk (gdisk) version 1.0.10

Partition table scan:
  MBR: protective
  BSD: not present
  APM: not present
  GPT: present

Found valid GPT with protective MBR; using GPT.
```

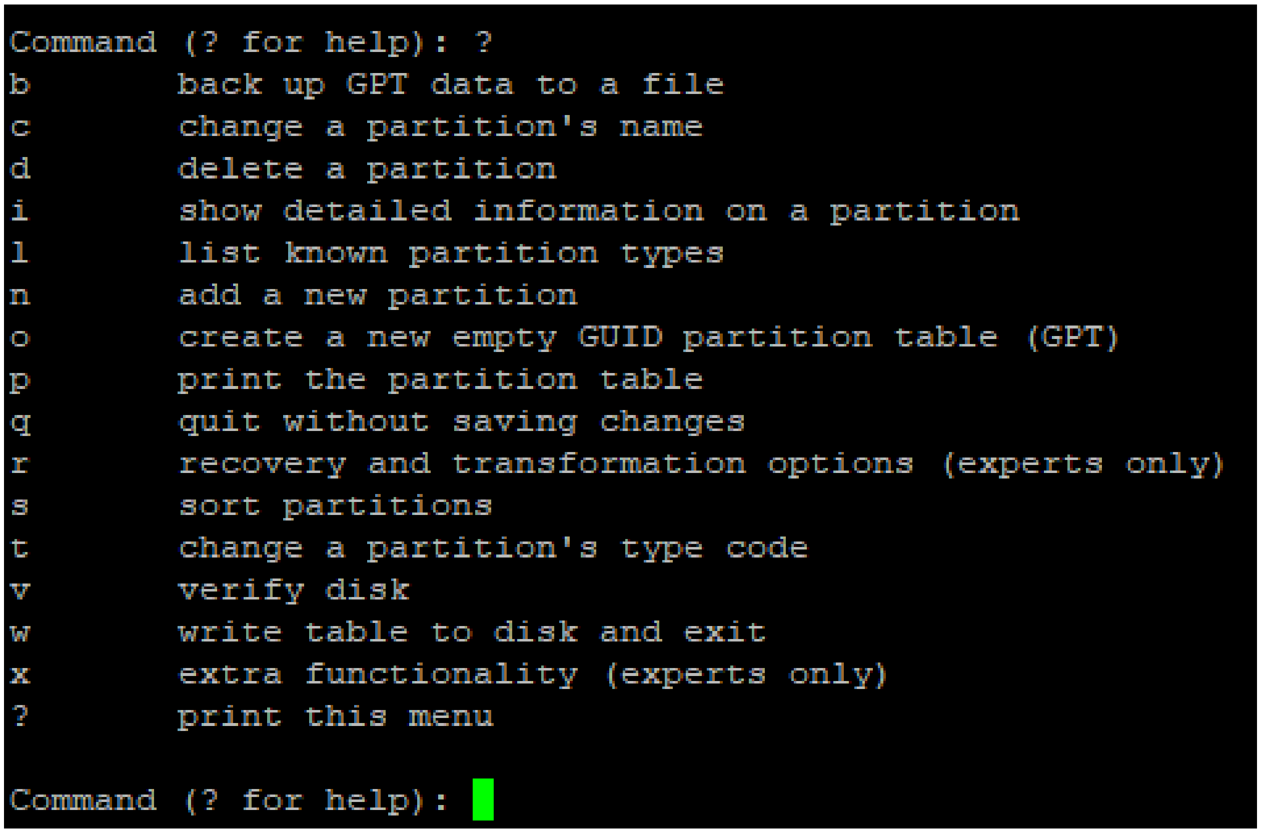
MBR: protective: Means this disk has a **"Protective MBR"**—a fake MBR to protect GPT disks from old tools, It prevents old MBR-only systems (like old BIOS) from mistakenly thinking the disk is empty.

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:



```
Command (? for help): ?
b       back up GPT data to a file
c       change a partition's name
d       delete a partition
i       show detailed information on a partition
l       list known partition types
n       add a new partition
o       create a new empty GUID partition table (GPT)
p       print the partition table
q       quit without saving changes
r       recovery and transformation options (experts only)
s       sort partitions
t       change a partition's type code
v       verify disk
w       write table to disk and exit
x       extra functionality (experts only)
?       print this menu

Command (? for help):
```

2. Partition Viewing and Monitoring commands:

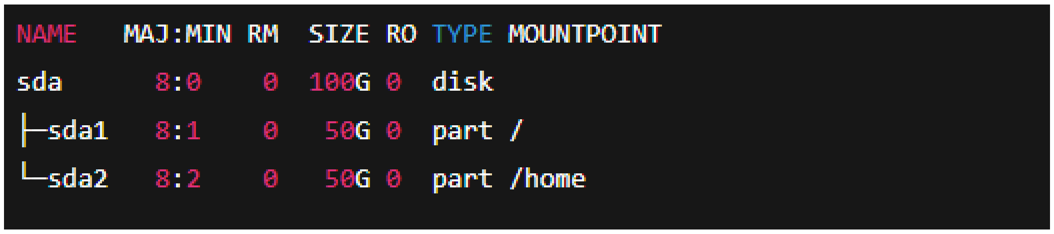
lsblk: It lists information about all available or specified block devices.

Block devices = disks, partitions, LVM volumes, flash drives, etc.

usage:

lsblk [options] [device...]

Default Output Columns:



```
NAME    MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sda       8:0    0  100G 0  disk
├─sda1    8:1    0   50G 0  part /
└─sda2    8:2    0   50G 0  part /home
```

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).

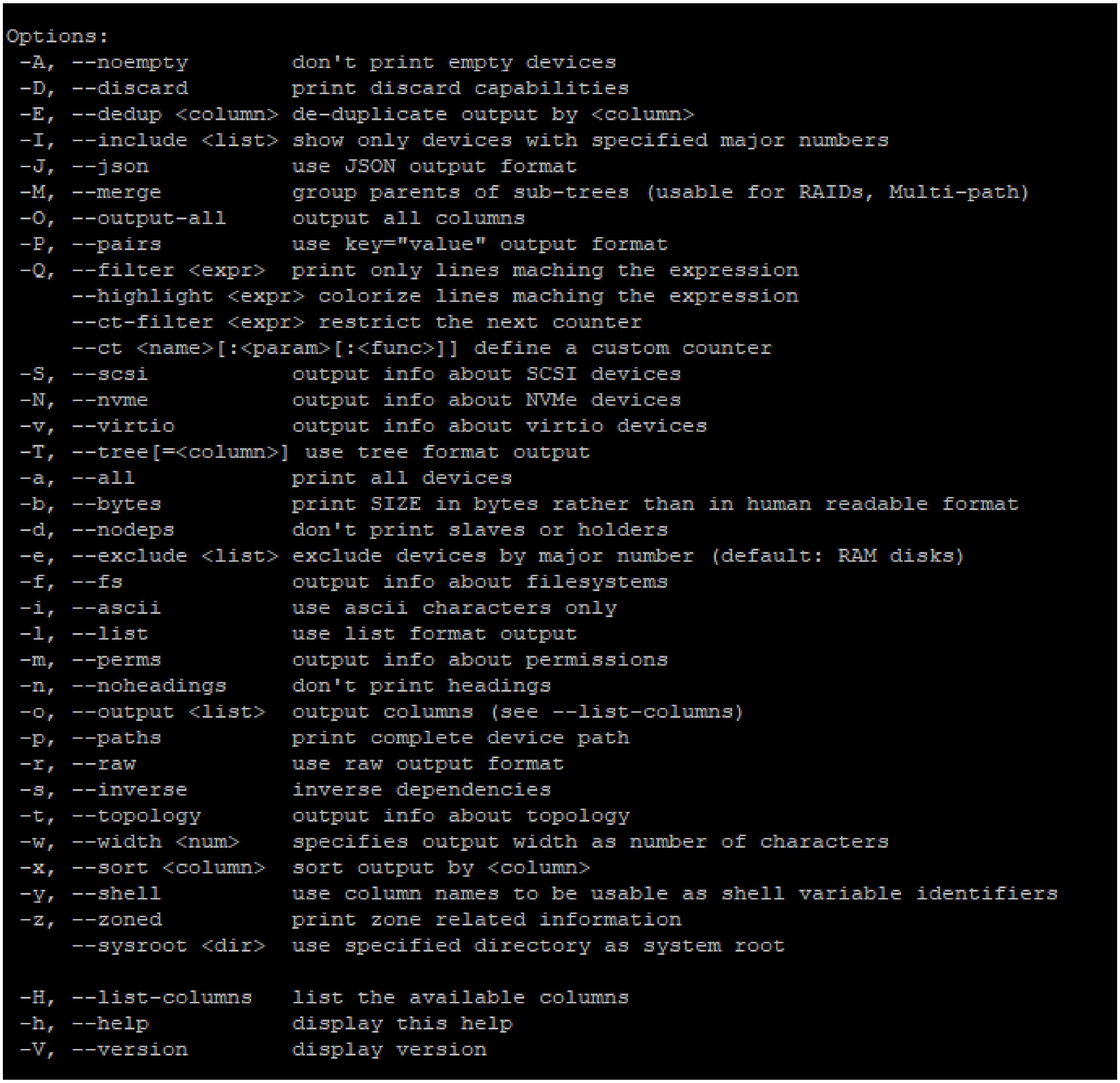
SIZE: The total size of the device or partition (for example: 100G, 512M, 2T).

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:



```
Options:
 -A, --noempty         don't print empty devices
 -D, --discard         print discard capabilities
 -E, --dedup <column>  de-duplicate output by <column>
 -I, --include <list>  show only devices with specified major numbers
 -J, --json            use JSON output format
 -M, --merge           group parents of sub-trees (usable for RAIDs, Multi-path)
 -O, --output-all      output all columns
 -P, --pairs           use key="value" output format
 -Q, --filter <expr>   print only lines maching the expression
     --highlight <expr> colorize lines maching the expression
     --ct-filter <expr> restrict the next counter
     --ct <name>[:<param>[:<func>]] define a custom counter
 -S, --scsi            output info about SCSI devices
 -N, --nvme            output info about NVMe devices
 -v, --virtio          output info about virtio devices
 -T, --tree[=<column>] use tree format output
 -a, --all             print all devices
 -b, --bytes           print SIZE in bytes rather than in human readable format
 -d, --nodeps          don't print slaves or holders
 -e, --exclude <list>  exclude devices by major number (default: RAM disks)
 -f, --fs              output info about filesystems
 -i, --ascii           use ascii characters only
 -l, --list            use list format output
 -m, --perms           output info about permissions
 -n, --noheadings      don't print headings
 -o, --output <list>   output columns (see --list-columns)
 -p, --paths           print complete device path
 -r, --raw             use raw output format
 -s, --inverse         inverse dependencies
 -t, --topology        output info about topology
 -w, --width <num>     specifies output width as number of characters
 -x, --sort <column>   sort output by <column>
 -y, --shell           use column names to be usable as shell variable identifiers
 -z, --zoned           print zone related information
     --sysroot <dir>   use specified directory as system root

 -H, --list-columns    list the available columns
 -h, --help            display this help
 -V, --version         display version
```

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

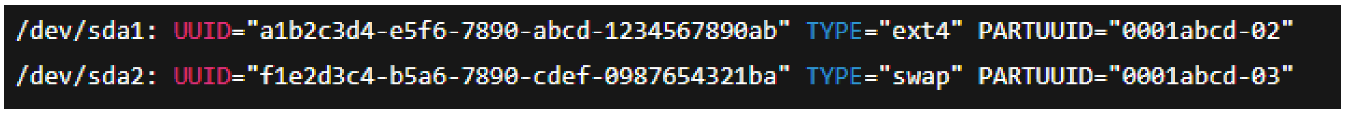
- Specifically shows device metadata (UUID, LABEL, FSTYPE).
- Focused ONLY on *identity info* — not size, not mountpoint, not topology.
- Perfect for finding which UUID matches which partition — this is what /etc/fstab and boot processes care about.
- Fast and simple output for automation scripts.

Usage:

blkid [options] [device...]

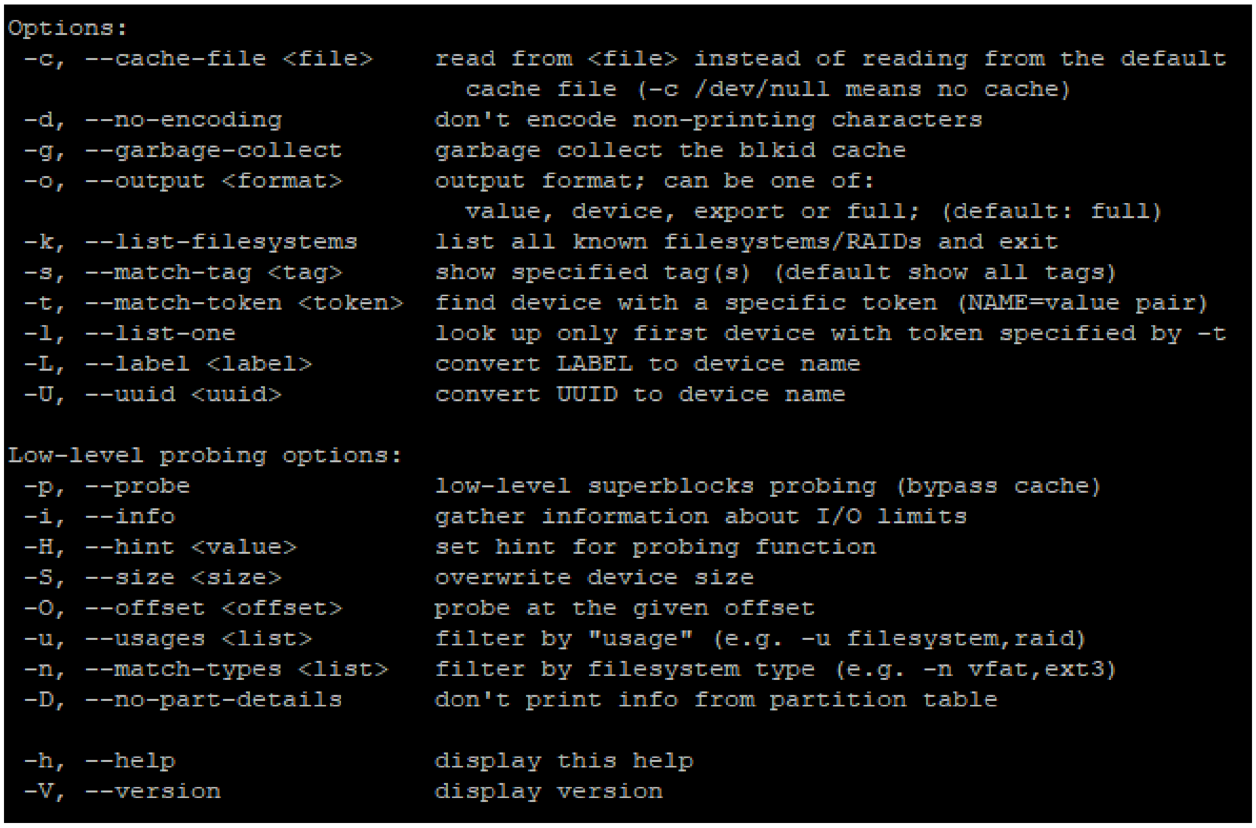
```
$ blkid
```

Output:

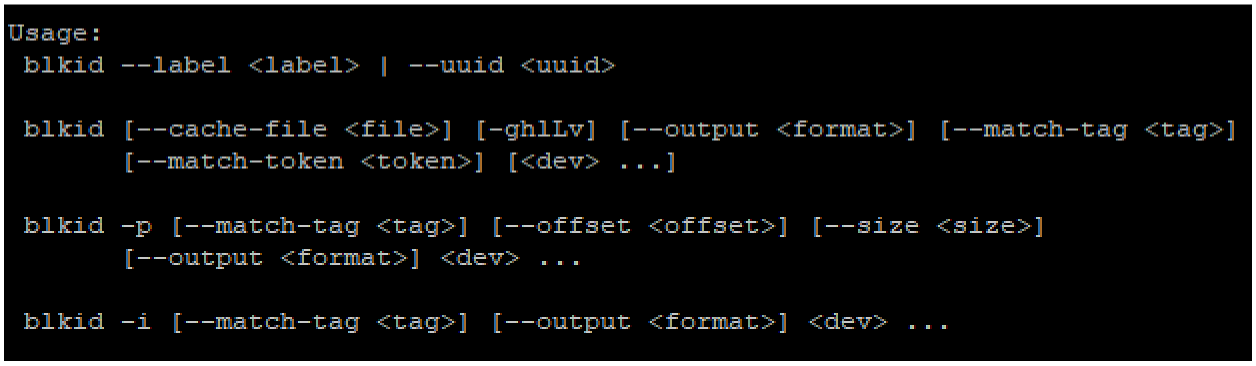


```
/dev/sda1: UUID="a1b2c3d4-e5f6-7890-abcd-1234567890ab" TYPE="ext4" PARTUUID="0001abcd-02"
/dev/sda2: UUID="f1e2d3c4-b5a6-7890-cdef-0987654321ba" TYPE="swap" PARTUUID="0001abcd-03"
```

Options:


```
Options:
 -c, --cache-file <file>    read from <file> instead of reading from the default
                              cache file (-c /dev/null means no cache)
 -d, --no-encoding          don't encode non-printing characters
 -g, --garbage-collect      garbage collect the blkid cache
 -o, --output <format>      output format; can be one of:
                              value, device, export or full; (default: full)
 -k, --list-filesystems     list all known filesystems/RAIDs and exit
 -s, --match-tag <tag>      show specified tag(s) (default show all tags)
 -t, --match-token <token>  find device with a specific token (NAME=value pair)
 -l, --list-one             look up only first device with token specified by -t
 -L, --label <label>        convert LABEL to device name
 -U, --uuid <uuid>          convert UUID to device name

Low-level probing options:
 -p, --probe                low-level superblocks probing (bypass cache)
 -i, --info                 gather information about I/O limits
 -H, --hint <value>         set hint for probing function
 -S, --size <size>          overwrite device size
 -O, --offset <offset>      probe at the given offset
 -u, --usages <list>        filter by "usage" (e.g. -u filesystem,raid)
 -n, --match-types <list>   filter by filesystem type (e.g. -n vfat,ext3)
 -D, --no-part-details      don't print info from partition table

 -h, --help                 display this help
 -V, --version              display version
```

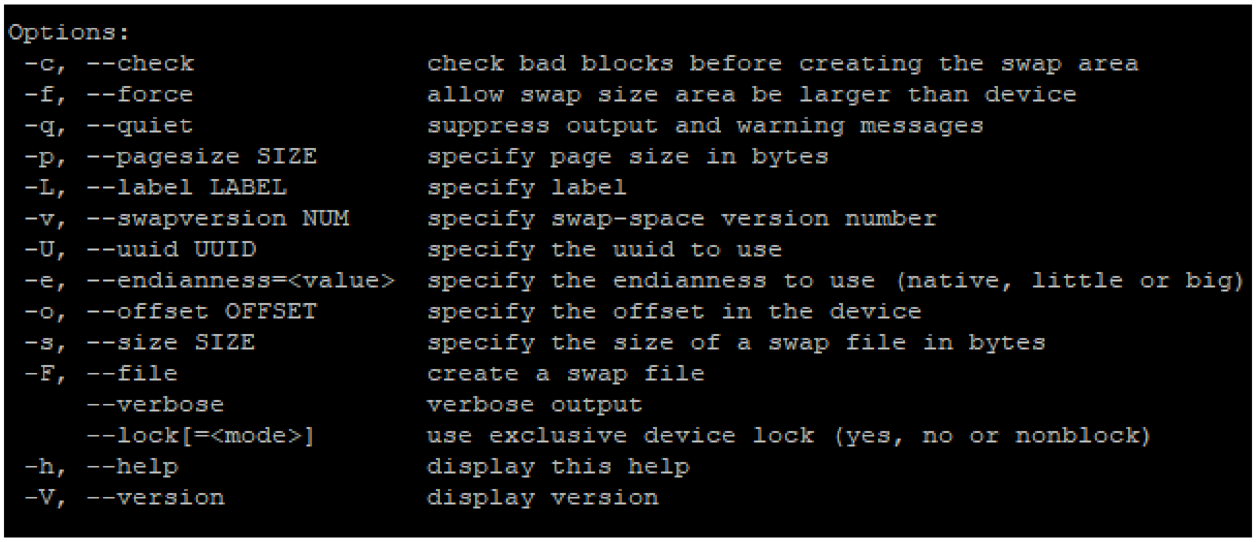
Usage of some options:


```
Usage:
 blkid --label <label> | --uuid <uuid>

 blkid [--cache-file <file>] [-ghlLv] [--output <format>] [--match-tag <tag>]
       [--match-token <token>] [<dev> ...]

 blkid -p [--match-tag <tag>] [--offset <offset>] [--size <size>]
       [--output <format>] <dev> ...

 blkid -i [--match-tag <tag>] [--output <format>] <dev> ...
```

Swap Management commands:

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

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



```
Options:
 -c, --check                check bad blocks before creating the swap area
 -f, --force                allow swap size area be larger than device
 -q, --quiet                suppress output and warning messages
 -p, --pagesize SIZE        specify page size in bytes
 -L, --label LABEL          specify label
 -v, --swapversion NUM      specify swap-space version number
 -U, --uuid UUID            specify the uuid to use
 -e, --endianness=<value>   specify the endianness to use (native, little or big)
 -o, --offset OFFSET        specify the offset in the device
 -s, --size SIZE            specify the size of a swap file in bytes
 -F, --file                 create a swap file
     --verbose              verbose output
     --lock[=<mode>]        use exclusive device lock (yes, no or nonblock)
 -h, --help                 display this help
 -V, --version              display version
```

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.

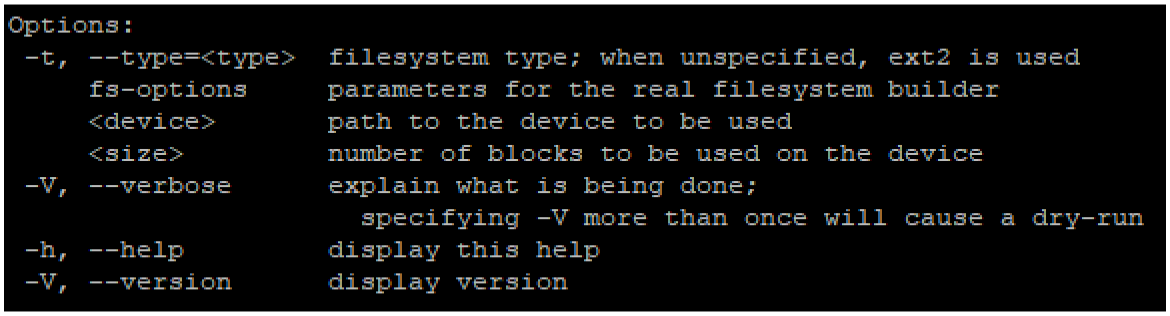
Usage:

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

OR a simpler form:

mkfs.<fstype> [options] <device>

options:



```
Options:
 -t, --type=<type>  filesystem type; when unspecified, ext2 is used
     fs-options     parameters for the real filesystem builder
     <device>       path to the device to be used
     <size>         number of blocks to be used on the device
 -V, --verbose      explain what is being done;
                      specifying -V more than once will cause a dry-run
 -h, --help         display this help
 -V, --version      display version
```

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).

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.

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

-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.

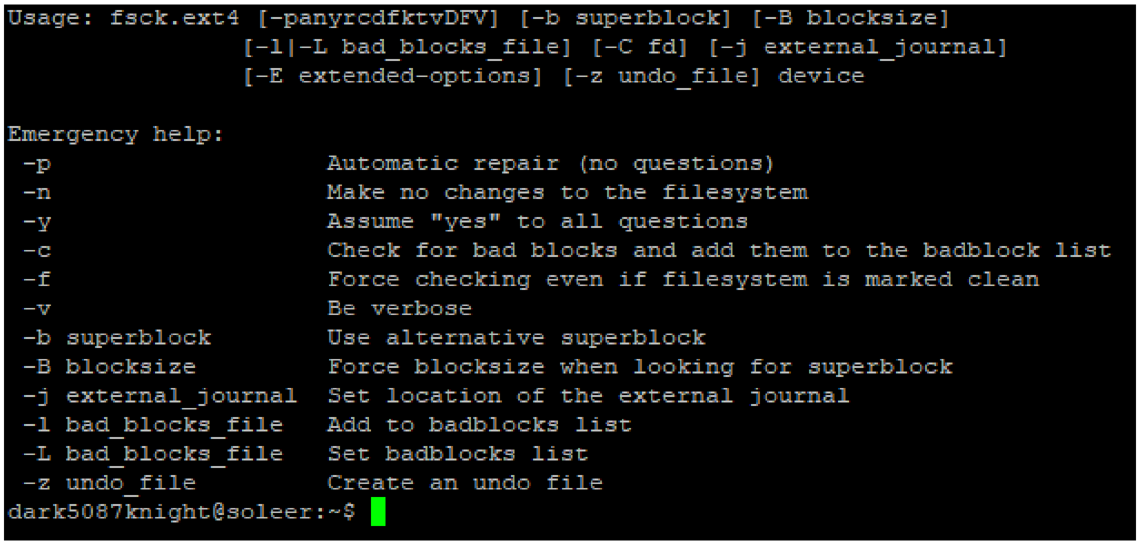
When to use fsck:

- 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:



```
Usage: fsck.ext4 [-panyrcdfktvDFV] [-b superblock] [-B blocksize]
                 [-l|-L bad_blocks_file] [-C fd] [-j external_journal]
                 [-E extended-options] [-z undo_file] device

Emergency help:
 -p                   Automatic repair (no questions)
 -n                   Make no changes to the filesystem
 -y                   Assume "yes" to all questions
 -c                   Check for bad blocks and add them to the badblock list
 -f                   Force checking even if filesystem is marked clean
 -v                   Be verbose
 -b superblock        Use alternative superblock
 -B blocksize         Force blocksize when looking for superblock
 -j external_journal  Set location of the external journal
 -l bad_blocks_file   Add to badblocks list
 -L bad_blocks_file   Set badblocks list
 -z undo_file         Create an undo file
dark5087knight@soleer:~$
```

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.

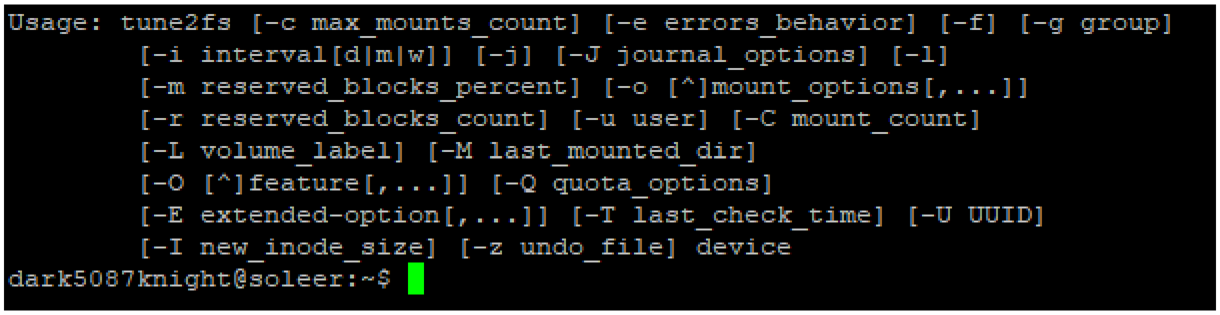
Usage:

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:



```
Usage: tune2fs [-c max_mounts_count] [-e errors_behavior] [-f] [-g group]
        [-i interval[d|m|w]] [-j] [-J journal_options] [-l]
        [-m reserved_blocks_percent] [-o [^]mount_options[,...]]
        [-r reserved_blocks_count] [-u user] [-C mount_count]
        [-L volume_label] [-M last_mounted_dir]
        [-O [^]feature[,...]] [-Q quota_options]
        [-E extended-option[,...]] [-T last_check_time] [-U UUID]
        [-I new_inode_size] [-z undo_file] device
dark5087knight@soleer:~$
```

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]

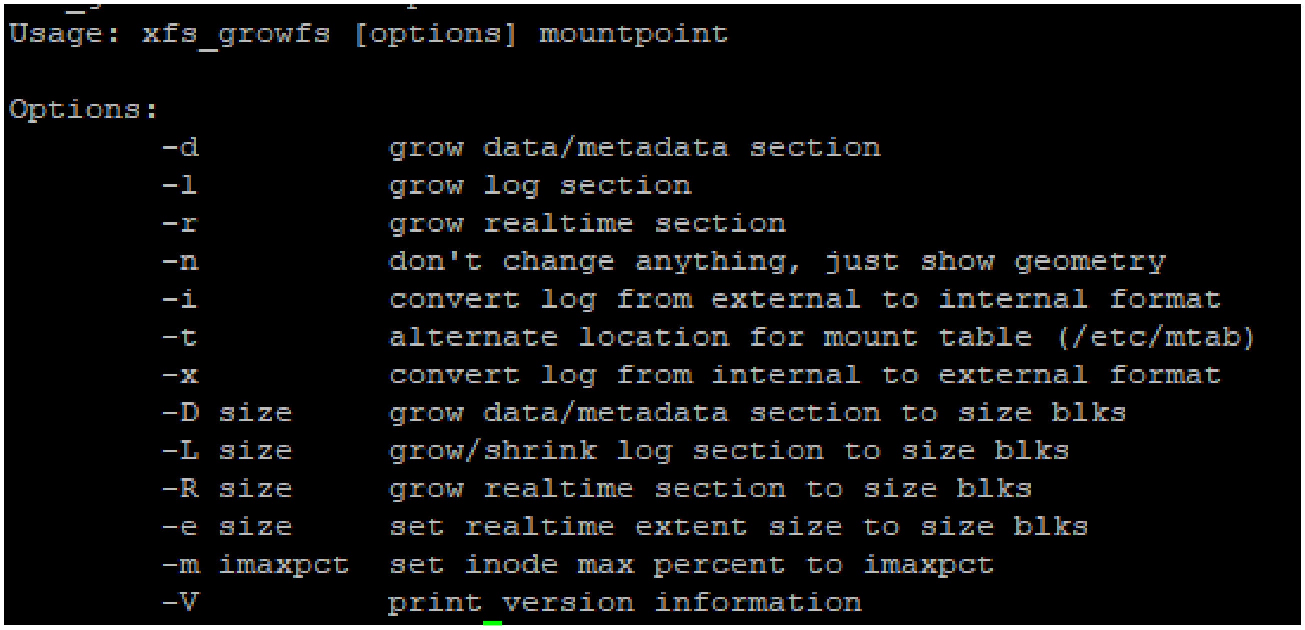
Options:

```
Options:

  -d debug-flags     Enable debugging output (for developers; rarely used).

  -f                 Force resizing even if the filesystem is marked with errors.
                     (Careful: normally you fix errors first using fsck.)

  -F                 Flush the filesystem device's buffer caches before resizing.
                     (Useful on some storage setups, but rarely needed for normal use.)

  -M                 Shrink the filesystem to its minimal possible size.
                     (Used to make the FS as small as it can be.)

  -p                 Show a progress bar during resizing.
                     (Helps you see the resizing progress in real time.)

  -P                 Print the minimum size the filesystem can be shrunk to, then exit.
                     (Does NOT resize, just prints info. Very useful before shrinking.)

  -h                 Show this help message and exit.
```

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



```
Usage: xfs_growfs [options] mountpoint

Options:
        -d          grow data/metadata section
        -l          grow log section
        -r          grow realtime section
        -n          don't change anything, just show geometry
        -i          convert log from external to internal format
        -t          alternate location for mount table (/etc/mtab)
        -x          convert log from internal to external format
        -D size     grow data/metadata section to size blks
        -L size     grow/shrink log section to size blks
        -R size     grow realtime section to size blks
        -e size     set realtime extent size to size blks
        -m imaxpct  set inode max percent to imaxpct
        -V          print version information
```

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:

```
Usage: mount [options] <device> <mount_point>

Options:

  -t type        Specify filesystem type (ext4, xfs, vfat, etc.).

  -o options     Set special mount options (like read-only, noexec, etc.).

  -r            Mount the filesystem as read-only.

  -w            Mount the filesystem as read-write (default).

  -L label      Mount by filesystem label instead of device name.

  -U uuid       Mount by UUID instead of device name.

  -a            Mount all filesystems listed in /etc/fstab.

  -n            Mount without writing to /etc/mtab (for advanced cases).

  -v            Verbose – show more information during mount.
```

```
Usage: umount [options] <device|mount_point>

Options:

  -l            Lazy unmount – disconnect when the filesystem is no longer busy.

  -f            Force unmount (even if busy or errors – risky).

  -v            Verbose – show what is being unmounted.

  -a            Unmount all filesystems listed in /etc/mtab (except critical ones like /).
```

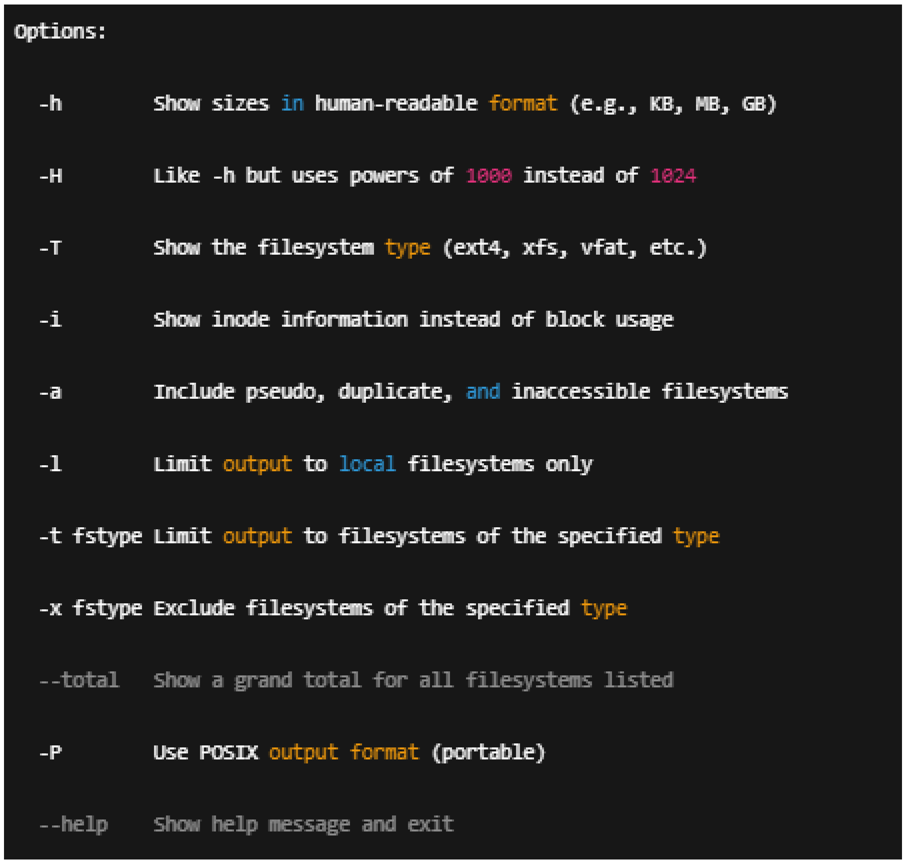
Disk Management commands:

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

usage:

df [options] [filesystem]

Options:



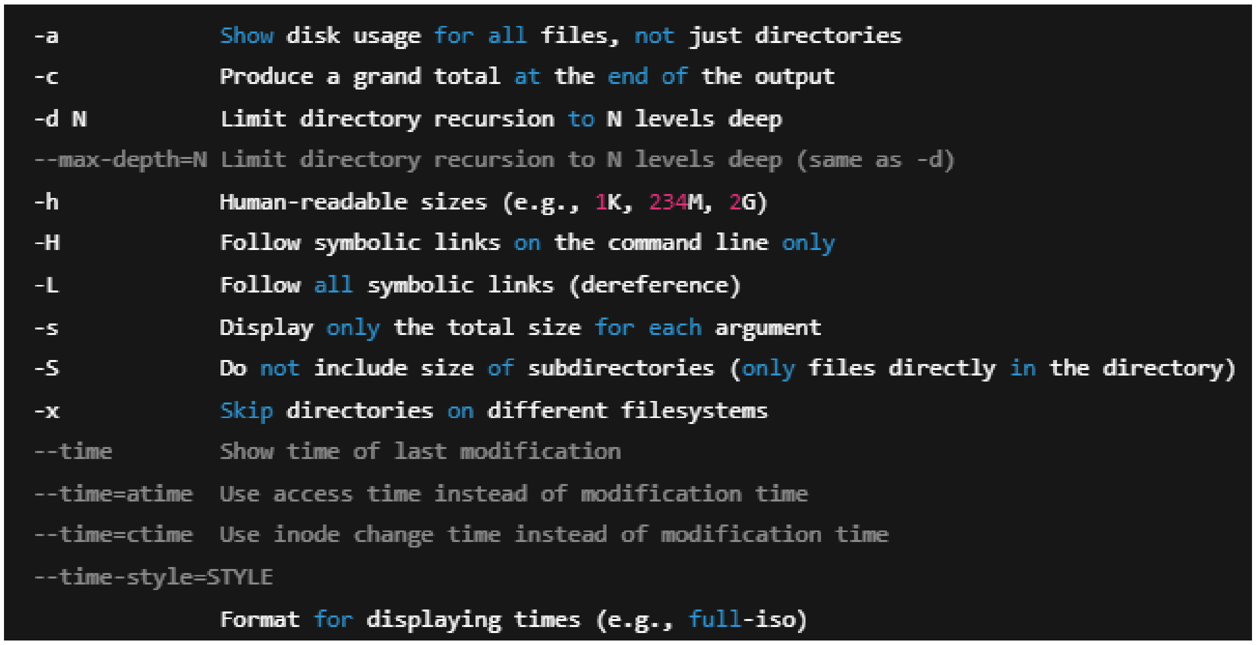
```
Options:

  -h        Show sizes in human-readable format (e.g., KB, MB, GB)

  -H        Like -h but uses powers of 1000 instead of 1024

  -T        Show the filesystem type (ext4, xfs, vfat, etc.)

  -i        Show inode information instead of block usage

  -a        Include pseudo, duplicate, and inaccessible filesystems

  -l        Limit output to local filesystems only

  -t fstype Limit output to filesystems of the specified type

  -x fstype Exclude filesystems of the specified type

  --total   Show a grand total for all filesystems listed

  -P        Use POSIX output format (portable)

  --help    Show help message and exit
```

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

Usage:

du [options] [file_or_directory]

options:



```
-a            Show disk usage for all files, not just directories
-c            Produce a grand total at the end of the output
-d N          Limit directory recursion to N levels deep
--max-depth=N Limit directory recursion to N levels deep (same as -d)
-h            Human-readable sizes (e.g., 1K, 234M, 2G)
-H            Follow symbolic links on the command line only
-L            Follow all symbolic links (dereference)
-s            Display only the total size for each argument
-S            Do not include size of subdirectories (only files directly in the directory)
-x            Skip directories on different filesystems
--time        Show time of last modification
--time=atime  Use access time instead of modification time
--time=ctime  Use inode change time instead of modification time
--time-style=STYLE
              Format for displaying times (e.g., full-iso)
```

4- User and group management

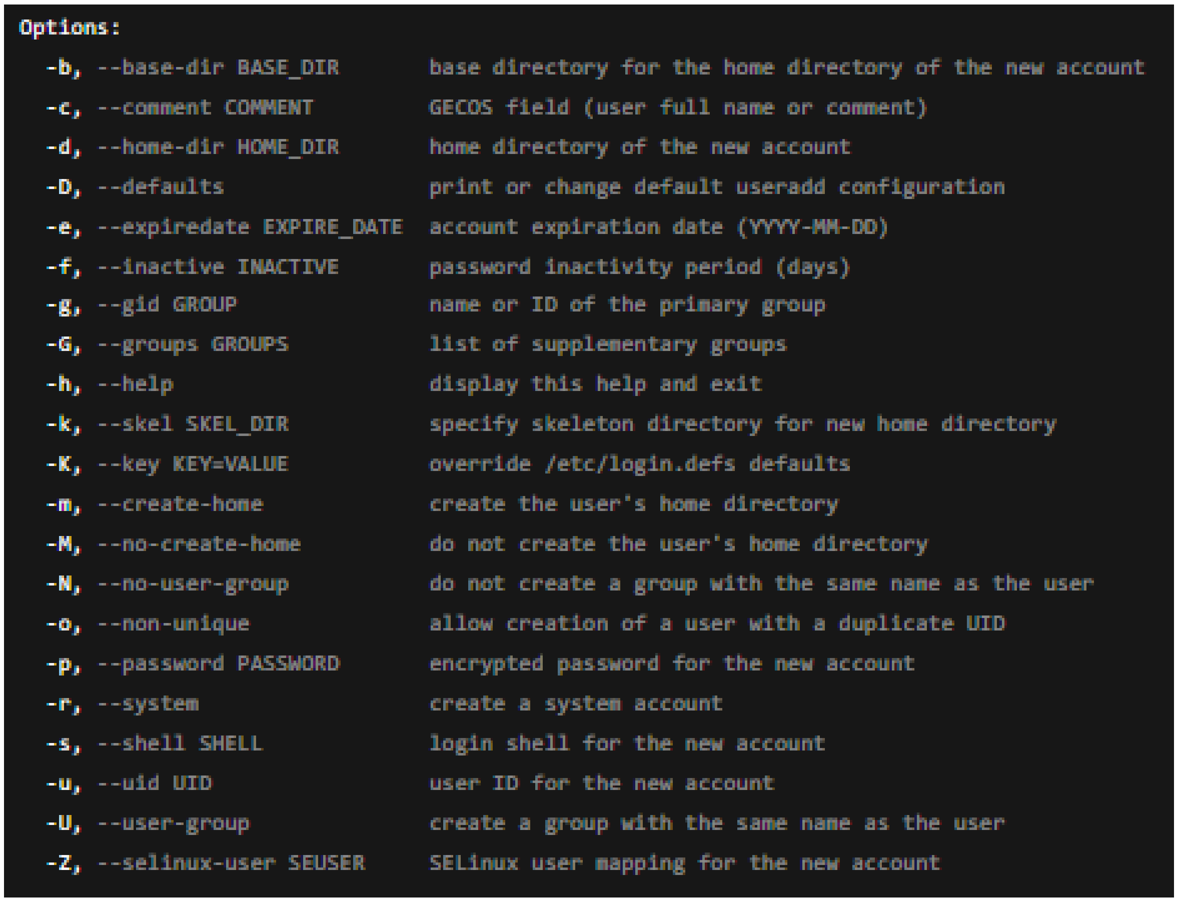
4.1- User management commands:

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:

```
useradd [OPTIONS] USERNAME
```



```
Options:
  -b, --base-dir BASE_DIR       base directory for the home directory of the new account
  -c, --comment COMMENT         GECOS field (user full name or comment)
  -d, --home-dir HOME_DIR       home directory of the new account
  -D, --defaults                print or change default useradd configuration
  -e, --expiredate EXPIRE_DATE  account expiration date (YYYY-MM-DD)
  -f, --inactive INACTIVE       password inactivity period (days)
  -g, --gid GROUP               name or ID of the primary group
  -G, --groups GROUPS           list of supplementary groups
  -h, --help                    display this help and exit
  -k, --skel SKEL_DIR           specify skeleton directory for new home directory
  -K, --key KEY=VALUE           override /etc/login.defs defaults
  -m, --create-home             create the user's home directory
  -M, --no-create-home          do not create the user's home directory
  -N, --no-user-group           do not create a group with the same name as the user
  -o, --non-unique              allow creation of a user with a duplicate UID
  -p, --password PASSWORD       encrypted password for the new account
  -r, --system                  create a system account
  -s, --shell SHELL             login shell for the new account
  -u, --uid UID                 user ID for the new account
  -U, --user-group              create a group with the same name as the user
  -Z, --selinux-user SEUSER     SELinux user mapping for the new account
```

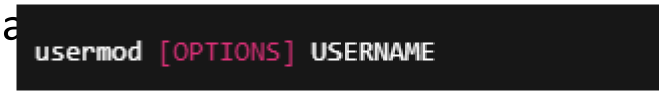
Example:

```
sudo useradd -m -s /bin/bash user1
```

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

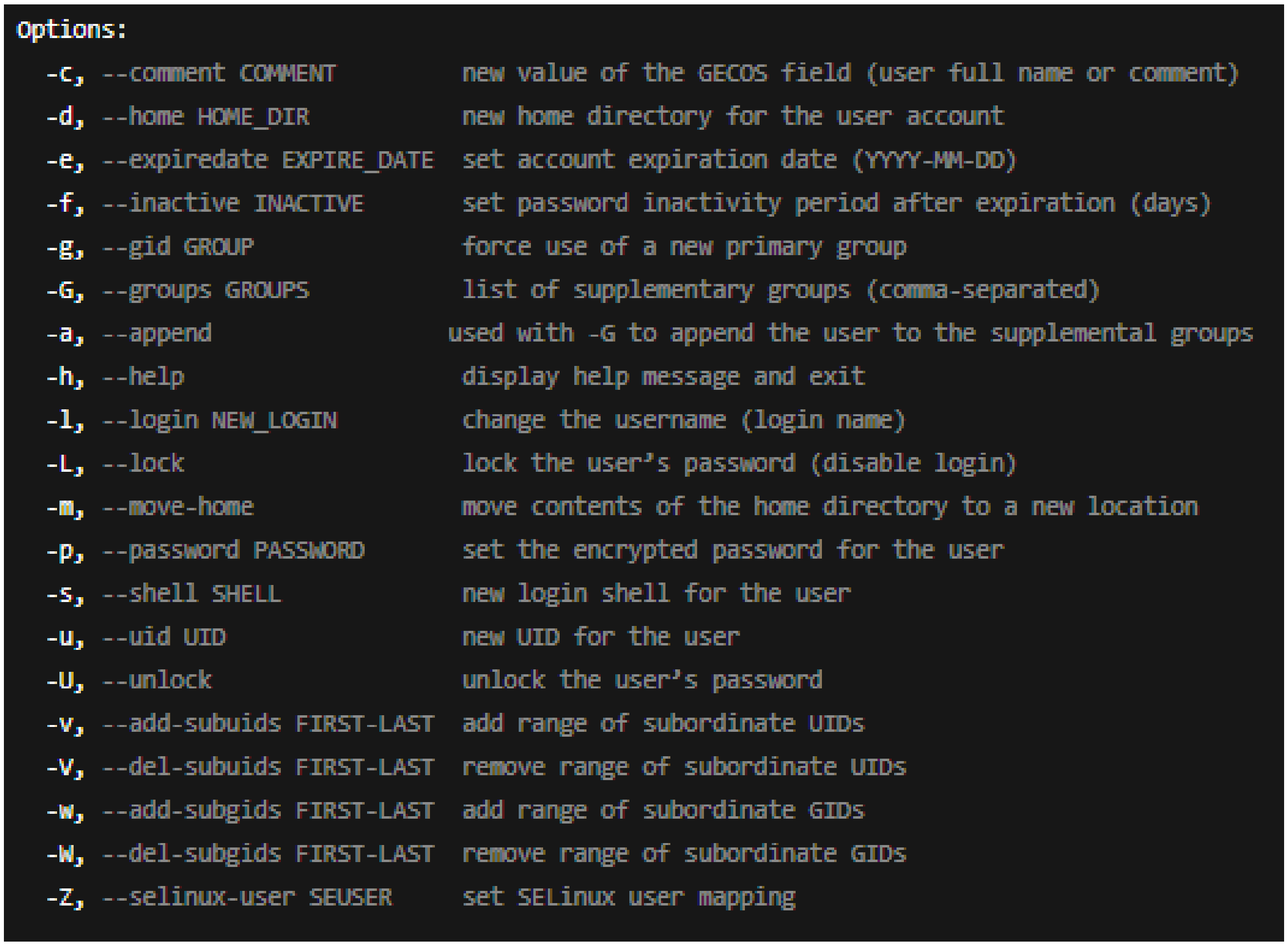
User Management:

- **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, a



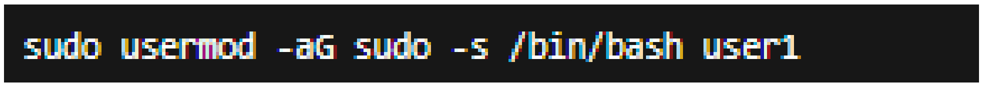
```
usermod [OPTIONS] USERNAME
```

Usage:



```
Options:
  -c, --comment COMMENT         new value of the GECOS field (user full name or comment)
  -d, --home HOME_DIR           new home directory for the user account
  -e, --expiredate EXPIRE_DATE  set account expiration date (YYYY-MM-DD)
  -f, --inactive INACTIVE       set password inactivity period after expiration (days)
  -g, --gid GROUP               force use of a new primary group
  -G, --groups GROUPS           list of supplementary groups (comma-separated)
  -a, --append                 used with -G to append the user to the supplemental groups
  -h, --help                    display help message and exit
  -l, --login NEW_LOGIN         change the username (login name)
  -L, --lock                    lock the user's password (disable login)
  -m, --move-home               move contents of the home directory to a new location
  -p, --password PASSWORD       set the encrypted password for the user
  -s, --shell SHELL             new login shell for the user
  -u, --uid UID                 new UID for the user
  -U, --unlock                  unlock the user's password
  -v, --add-subuids FIRST-LAST  add range of subordinate UIDs
  -V, --del-subuids FIRST-LAST  remove range of subordinate UIDs
  -w, --add-subgids FIRST-LAST  add range of subordinate GIDs
  -W, --del-subgids FIRST-LAST  remove range of subordinate GIDs
  -Z, --selinux-user SEUSER     set SELinux user mapping
```

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



```
sudo usermod -aG sudo -s /bin/bash user1
```

- **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.

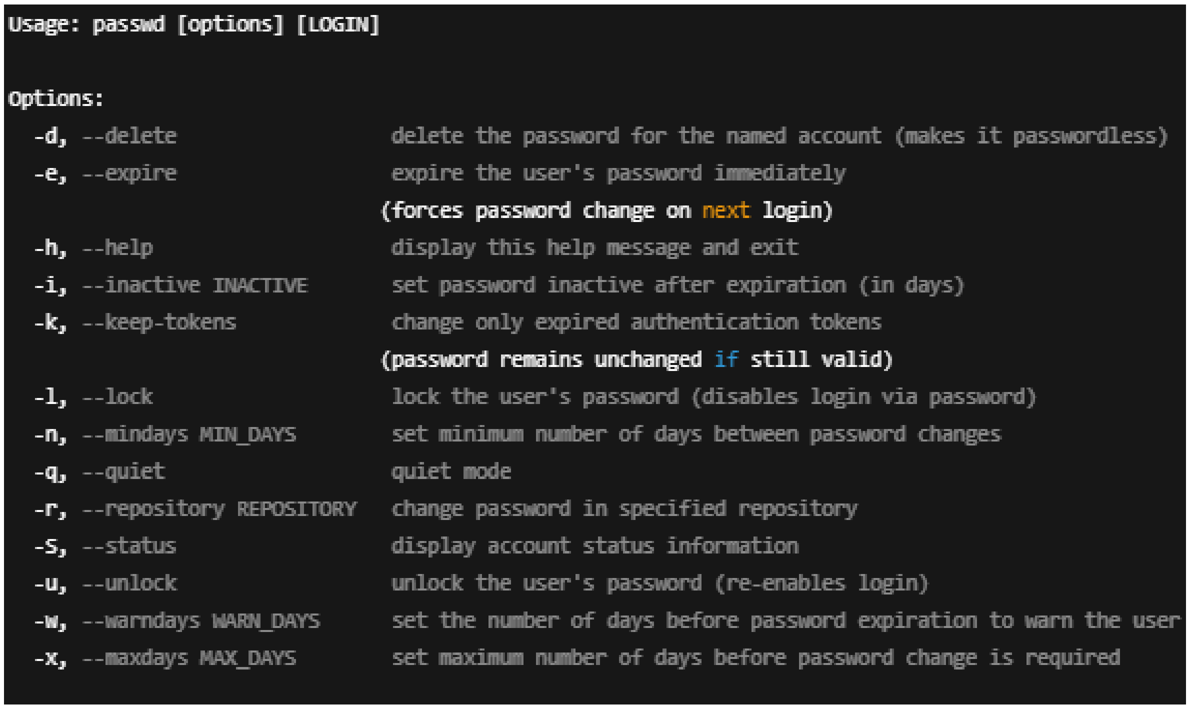
```
Usage: chfn [options] [LOGIN]

Options:
  -f, --full-name FULL_NAME       change the user's full name
  -r, --room ROOM_NUMBER          change the user's room number
  -w, --work-phone PHONE          change the user's work phone
  -h, --home-phone PHONE          change the user's home phone
  -o, --other OTHER               change the user's other information
      --help                     display this help and exit
```

- **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.

```
Usage: id [options] [user]

Options:
  -a                    (ignored; for compatibility)
  -G                    print all group IDs
  -g                    print only the effective group ID
  -n                    print a name instead of a numeric ID (used with -u, -g, or -G)
  -r                    print the real ID instead of the effective ID (used with -u, -g)
  -u                    print only the effective user ID
  -z                    print a NUL character after each output value
```

User Password Management:

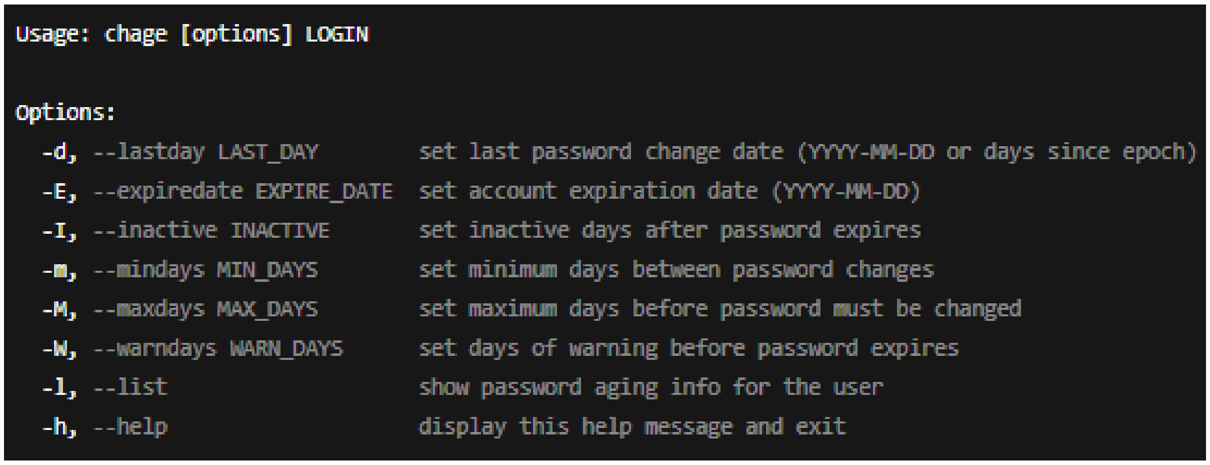
- **passwd:** is used to change a user's password or set password aging policies on Linux systems. It updates entries in /etc/shadow securely.



```
Usage: passwd [options] [LOGIN]

Options:
  -d, --delete                delete the password for the named account (makes it passwordless)
  -e, --expire                expire the user's password immediately
                             (forces password change on next login)
  -h, --help                  display this help message and exit
  -i, --inactive INACTIVE     set password inactive after expiration (in days)
  -k, --keep-tokens           change only expired authentication tokens
                             (password remains unchanged if still valid)
  -l, --lock                  lock the user's password (disables login via password)
  -n, --mindays MIN_DAYS      set minimum number of days between password changes
  -q, --quiet                 quiet mode
  -r, --repository REPOSITORY change password in specified repository
  -S, --status                display account status information
  -u, --unlock                unlock the user's password (re-enables login)
  -w, --warndays WARN_DAYS    set the number of days before password expiration to warn the user
  -x, --maxdays MAX_DAYS      set maximum number of days before password change is required
```

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

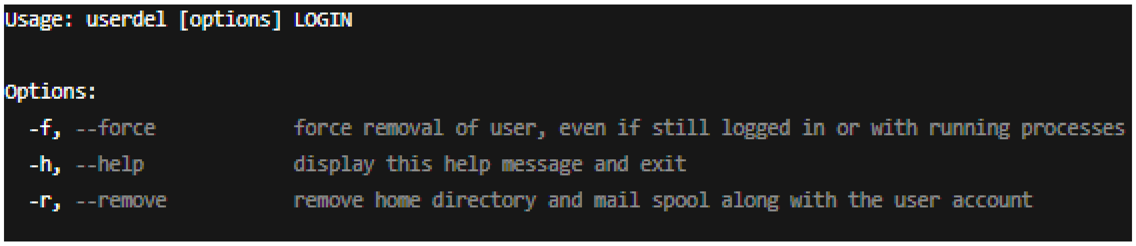
- **chage (Change Age):** manages user password aging policies, including expiration, warning periods, and account inactivity settings stored in /etc/shadow.



```
Usage: chage [options] LOGIN

Options:
  -d, --lastday LAST_DAY        set last password change date (YYYY-MM-DD or days since epoch)
  -E, --expiredate EXPIRE_DATE  set account expiration date (YYYY-MM-DD)
  -I, --inactive INACTIVE       set inactive days after password expires
  -m, --mindays MIN_DAYS        set minimum days between password changes
  -M, --maxdays MAX_DAYS        set maximum days before password must be changed
  -W, --warndays WARN_DAYS      set days of warning before password expires
  -l, --list                    show password aging info for the user
  -h, --help                    display this help message and exit
```

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.



```
Usage: userdel [options] LOGIN

Options:
  -f, --force               force removal of user, even if still logged in or with running processes
  -h, --help                display this help message and exit
  -r, --remove              remove home directory and mail spool along with the user account
```

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.

```
Usage: groupadd [options] GROUP

Options:
  -f, --force               exit successfully if the group already exists
  -g, --gid GID             use the specified group ID for the new group
  -h, --help                display this help message and exit
  -K, --key KEY=VALUE       override /etc/login.defs defaults
  -o, --non-unique          allow creating groups with duplicate GIDs
  -r, --system              create a system group (GID < 1000)
```

```
sudo groupadd developers
```

Group Management:

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

```
Usage: groupmod [options] GROUP

Options:
  -g, --gid GID             new group ID for the group
  -n, --new-name NEW_NAME   new name of the group
  -h, --help                display this help message and exit
  -o, --non-unique          allow using a non-unique GID
```

```
sudo groupmod -n devteam developers
```

Group Password Management:

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

```
Usage: gpasswd [options] GROUP

Options:
  -a, --add USER          add USER to the group
  -d, --delete USER       remove USER from the group
  -h, --help              display this help message and exit
  -r, --remove-password   remove the group password
  -R, --readonly          set the group to readonly (no password changes allowed)
  -A, --admin USERS       set the group administrators
```

```
sudo gpasswd -a user1 developers
```

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

```
sudo groupdel developers
```

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.

```
Usage: chown [OPTION]... [OWNER][:[GROUP]] FILE...

Options:
  -c, --changes             like verbose but only report when a change is made
  -f, --silent, --quiet     suppress most error messages
  -v, --verbose             output a diagnostic for every file processed
  --dereference             affect the referent of each symbolic link (default)
  -h, --no-dereference      affect symbolic links instead of referenced files
  --from=CURRENT_OWNER:CURRENT_GROUP
                           change the ownership only if current owner/group match
  -R, --recursive           operate on files and directories recursively
  --help                    display this help and exit
  --version                 output version information and exit
```

```
sudo chown user1 file.txt                # Change owner only
sudo chown user1:group1 file.txt        # Change owner and group
sudo chown :group1 file.txt             # Change group only
sudo chown -R user1:group1 /project     # Recursive ownership change
```

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

```
Usage: chgrp [OPTION]... GROUP FILE...

Options:
  -c, --changes             like verbose but only report when a change is made
  -f, --silent, --quiet     suppress most error messages
  -v, --verbose             output a diagnostic for every file processed
  --dereference             affect the referent of each symbolic link (default)
  -h, --no-dereference      affect symbolic links instead of referenced files
  --reference=RFILE         use RFILE's group instead of specifying a group
  -R, --recursive           operate on files and directories recursively
  --help                    display this help and exit
  --version                 output version information and exit
```

```
sudo chgrp developers file.txt           # Change group to 'developers'
sudo chgrp -R team /shared/folder        # Recursively change group ownership
sudo chgrp --reference=template file.txt # Copy group ownership from another file
```

- **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.

```
Usage: chmod [OPTION]... MODE[,MODE]... FILE...

Options:
  -c, --changes          like verbose but report only when a change is made
  -f, --silent, --quiet  suppress most error messages
  -v, --verbose          output a diagnostic for every file processed
      --no-preserve-root do not treat '/' specially (default)
      --preserve-root    fail to operate recursively on '/'
  -R, --recursive        change files and directories recursively
      --reference=RFILE  use RFILE's mode instead of specifying MODE
      --help             display this help and exit
      --version          output version information and exit
```

```
chmod 755 script.sh                  # Owner: rwx, Group: r-x, Others: r-x
chmod u+x file.txt                   # Add execute permission to owner
chmod g-w file.txt                   # Remove write permission from group
chmod o=r file.txt                   # Set others to read-only
chmod -R 700 /secure/data            # Recursive full access to owner only
chmod --reference=template file.txt  # Match permissions of another file
```

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

```
Usage: umask [options] [mask]

Options:
  -S, --symbolic     output in symbolic (rwx) notation instead of octal
  -p, --p            output in a format that may be reused as input
      --help         display this help and exit
      --version      output version information and exit
```

```
umask                    # Show current mask (usually 0022)
umask -S                 # Show current mask in symbolic format (e.g., u=rwx,g=rx,o=rx)
umask 077                # New files: 600, New dirs: 700 (only owner access)
umask 002                # Group-writable: files 664, dirs 775
```

Special permissions:

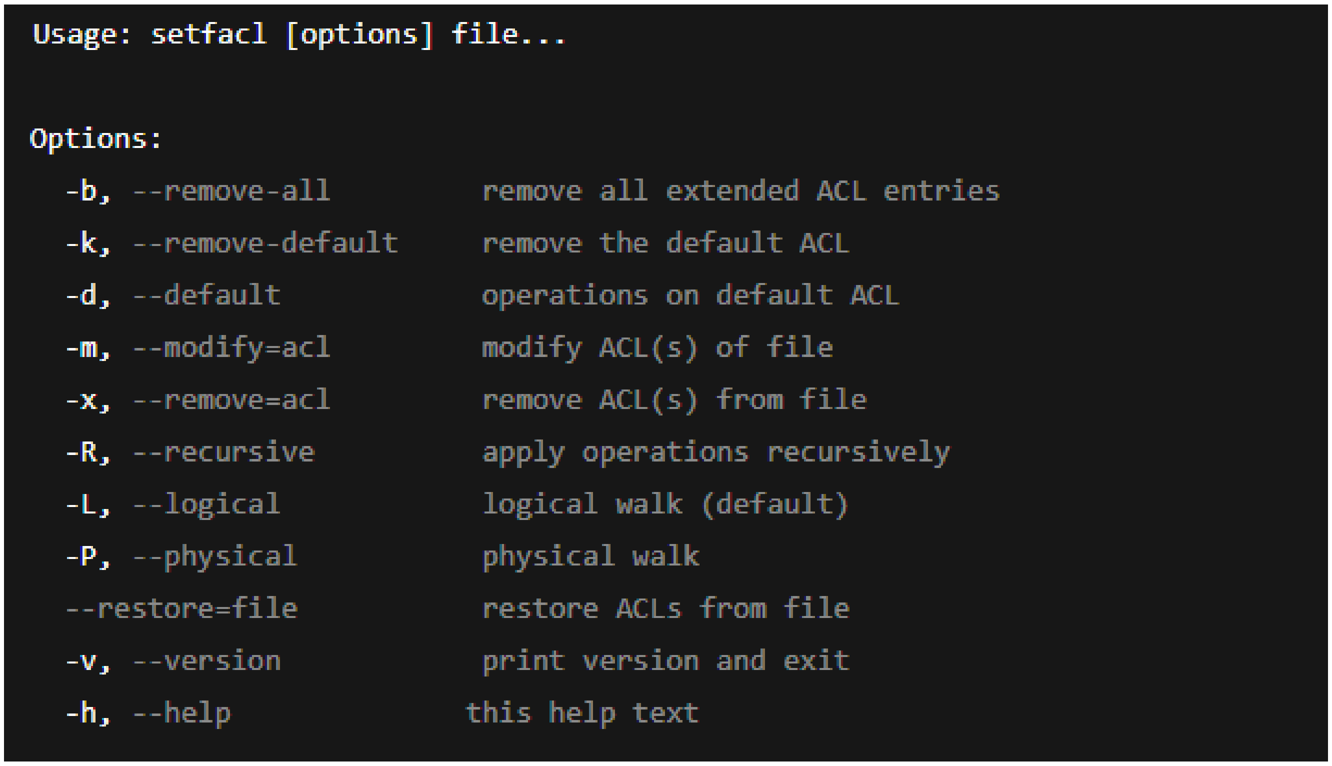
- **SUID:** Runs an executable with the file owner's permissions instead of the user's.
- **SGID:** Runs an executable with the file group's permissions or makes new files inherit the directory's group.
- **Sticky Bit**: Restricts deletion in a directory so only the file owner, directory owner, or root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

```
chmod 4755 /usr/bin/passwd     # SUID: allows normal users to change their own passwords
chmod 2755 /var/www/html       # SGID: new files inherit 'www-data' group in web server directory
chmod 1777 /tmp                # Sticky Bit: users can only delete their own files in /tmp
```

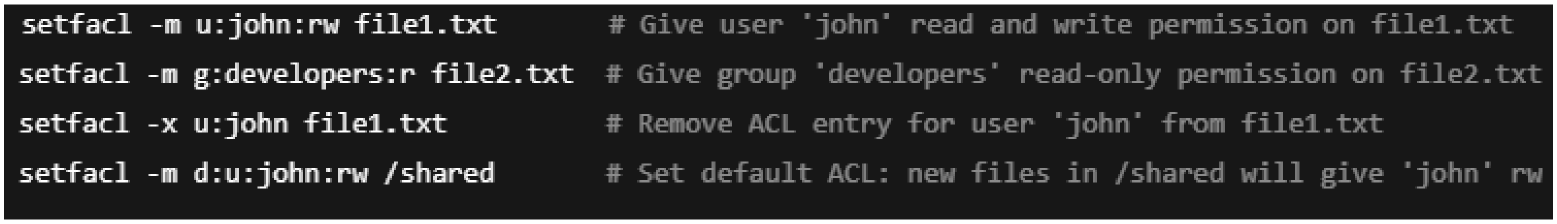
```
# SUID examples on a file
chmod u+s /usr/bin/example       # Set SUID
chmod u-s /usr/bin/example       # Remove SUID

# SGID examples on a file and directory
chmod g+s /usr/bin/example       # Set SGID on file
chmod g-s /usr/bin/example       # Remove SGID from file

chmod g+s /shared/directory      # Set SGID on directory (new files inherit group)
chmod g-s /shared/directory      # Remove SGID from directory

# Sticky bit examples on a directory
chmod o+t /tmp                  # Set Sticky bit
chmod o-t /tmp                  # Remove Sticky bit
```

4.4- ACL (Access Control List)

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



```
Usage: setfacl [options] file...

Options:
  -b, --remove-all          remove all extended ACL entries
  -k, --remove-default      remove the default ACL
  -d, --default             operations on default ACL
  -m, --modify=acl          modify ACL(s) of file
  -x, --remove=acl          remove ACL(s) from file
  -R, --recursive           apply operations recursively
  -L, --logical             logical walk (default)
  -P, --physical            physical walk
  --restore=file            restore ACLs from file
  -v, --version             print version and exit
  -h, --help               this help text
```



```
setfacl -m u:john:rw file1.txt       # Give user 'john' read and write permission on file1.txt
setfacl -m g:developers:r file2.txt  # Give group 'developers' read-only permission on file2.txt
setfacl -x u:john file1.txt          # Remove ACL entry for user 'john' from file1.txt
setfacl -m d:u:john:rw /shared       # Set default ACL: new files in /shared will give 'john' rw
```

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

```
Usage: getfacl [options] file...


Options:
  -a, --access            display the access ACL
  -d, --default           display the default ACL
  -c, --omit-header       do not display the comment header
  -e, --all-effective     print all effective rights
  -R, --recursive         list directories recursively
  -L, --logical           logical walk (default)
  -P, --physical          physical walk
  --restore=file          restore ACLs from file
  -v, --version           print version and exit
  -h, --help              this help text
```

```
getfacl file1.txt              # Show ACL of file1.txt
getfacl /shared/directory      # Show ACL of a directory
getfacl -R /var/www            # Recursively display ACLs under /var/www
```

5- Process and Service Management

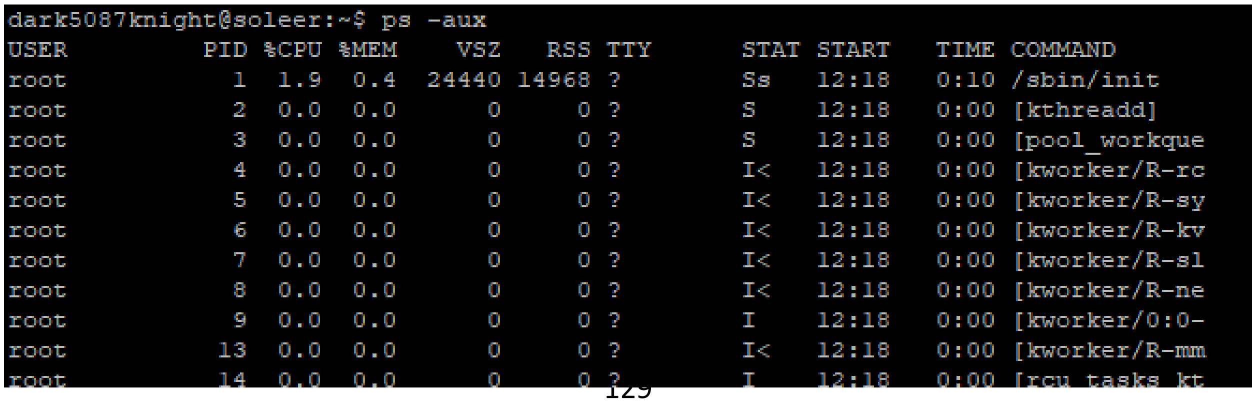
5.1- Process Management:

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.

```
ps [OPTIONS]

-a                        # Show processes from all users (except session leaders)
-u                        # Show user-oriented output (includes USER, %CPU, %MEM, etc.)
-x                        # Show processes without controlling terminals
-e or -A                 # Show all processes on the system
-f                        # Full format listing (includes PPID, STIME, CMD, etc.)
-l                        # Long format listing (more detailed info)
-p PID[,PID...]           # Show only processes with the given PID(s)
-U USER[,USER...]         # Show only processes for specified user(s)
-o FORMAT                 # Specify custom output columns
--sort=KEY                # Sort output by given column (e.g., --sort=-%mem)

--help                    # Show help message and exit
--version                 # Show version information and exit
```

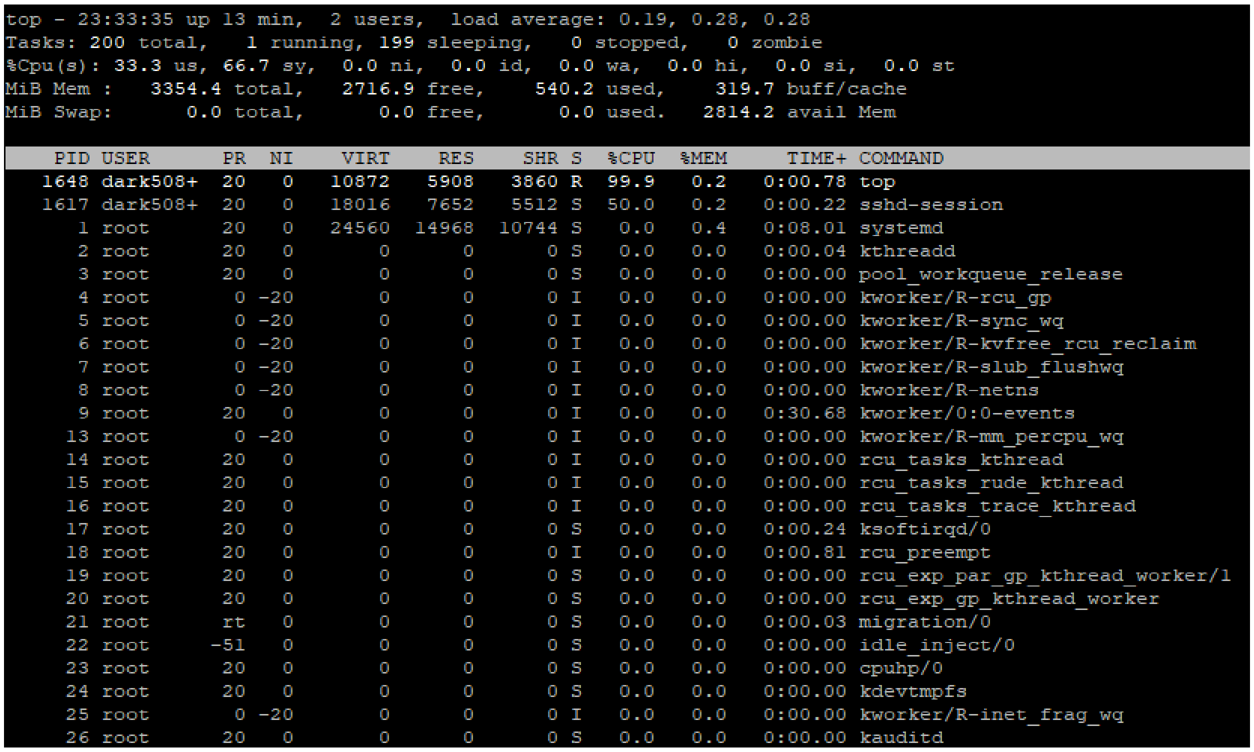
```
ps                     # Show processes for the current shell/session only
ps aux                 # Show all running processes with detailed user info
ps -ef                 # Full-format list of all processes
```



```
dark5087knight@soleer:~$ ps -aux
USER         PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root           1  1.9  0.4  24440 14968 ?        Ss   12:18   0:10 /sbin/init
root           2  0.0  0.0      0     0 ?        S    12:18   0:00 [kthreadd]
root           3  0.0  0.0      0     0 ?        S    12:18   0:00 [pool_workque
root           4  0.0  0.0      0     0 ?        I<   12:18   0:00 [kworker/R-rc
root           5  0.0  0.0      0     0 ?        I<   12:18   0:00 [kworker/R-sy
root           6  0.0  0.0      0     0 ?        I<   12:18   0:00 [kworker/R-kv
root           7  0.0  0.0      0     0 ?        I<   12:18   0:00 [kworker/R-sl
root           8  0.0  0.0      0     0 ?        I<   12:18   0:00 [kworker/R-ne
root           9  0.0  0.0      0     0 ?        I    12:18   0:00 [kworker/0:0-
root          13  0.0  0.0      0     0 ?        I<   12:18   0:00 [kworker/R-mm
root          14  0.0  0.0      0     0 ?        I    12:18   0:00 [rcu_tasks_kt
```

- **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.

```
top [OPTIONS]

-d, --delay=SECONDS          # Set delay between updates (default: 3 seconds)
-n, --number=ITERATIONS      # Number of iterations before exiting (use with -b)
-b, --batch                 # Batch mode: non-interactive output for logging
-p, --pid=PID,...           # Monitor only specified process IDs
-u, --user=USERNAME         # Show processes for the specified user only
-o, --order=FIELD           # Order by FIELD (e.g., %CPU, %MEM)
-H, --threads               # Show threads instead of processes
-M, --mem=UNIT              # Set memory units (k, m, g)
--help                      # Display this help and exit
--version                   # Show version information and exit
```

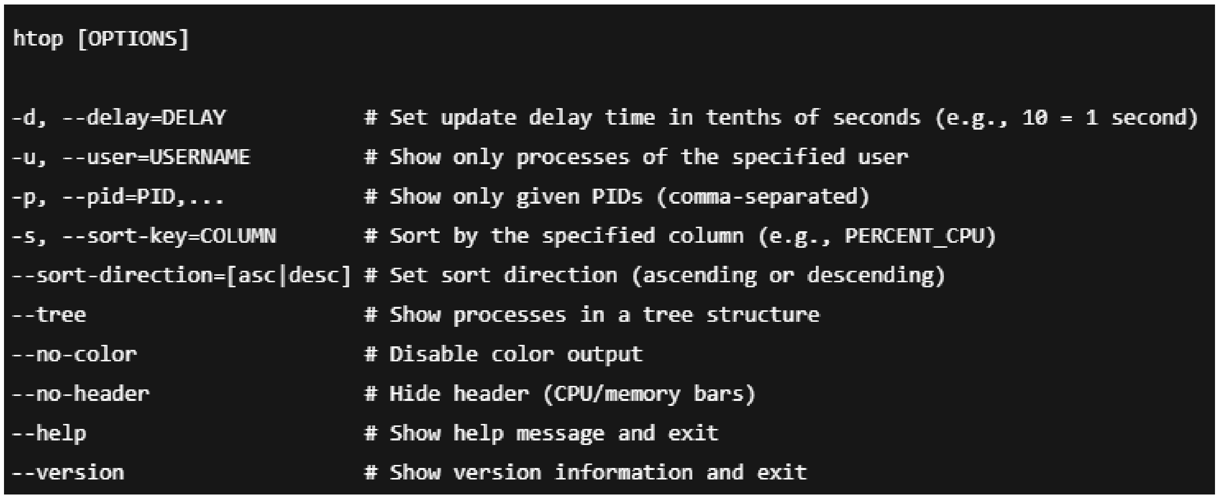


```
top - 23:33:35 up 13 min,  2 users,  load average: 0.19, 0.28, 0.28
Tasks: 200 total,   1 running, 199 sleeping,   0 stopped,   0 zombie
%Cpu(s): 33.3 us, 66.7 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
MiB Mem :   3354.4 total,   2716.9 free,    540.2 used,    319.7 buff/cache
MiB Swap:      0.0 total,      0.0 free,      0.0 used.   2814.2 avail Mem

    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
   1648 dark508+  20   0   10872   5908   3860 R  99.9   0.2   0:00.78 top
   1617 dark508+  20   0   18016   7652   5512 S  50.0   0.2   0:00.22 sshd-session
      1 root      20   0   24560  14968  10744 S   0.0   0.4   0:08.01 systemd
      2 root      20   0       0      0      0 S   0.0   0.0   0:00.04 kthreadd
      3 root      20   0       0      0      0 S   0.0   0.0   0:00.00 pool_workqueue_release
      4 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 kworker/R-rcu_gp
      5 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 kworker/R-sync_wq
      6 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 kworker/R-kvfree_rcu_reclaim
      7 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 kworker/R-slub_flushwq
      8 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 kworker/R-netns
      9 root      20   0       0      0      0 I   0.0   0.0   0:30.68 kworker/0:0-events
     13 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 kworker/R-mm_percpu_wq
     14 root      20   0       0      0      0 I   0.0   0.0   0:00.00 rcu_tasks_kthread
     15 root      20   0       0      0      0 I   0.0   0.0   0:00.00 rcu_tasks_rude_kthread
     16 root      20   0       0      0      0 I   0.0   0.0   0:00.00 rcu_tasks_trace_kthread
     17 root      20   0       0      0      0 S   0.0   0.0   0:00.24 ksoftirqd/0
     18 root      20   0       0      0      0 I   0.0   0.0   0:00.81 rcu_preempt
     19 root      20   0       0      0      0 S   0.0   0.0   0:00.00 rcu_exp_par_gp_kthread_worker/1
     20 root      20   0       0      0      0 S   0.0   0.0   0:00.00 rcu_exp_gp_kthread_worker
     21 root      rt   0       0      0      0 S   0.0   0.0   0:00.03 migration/0
     22 root     -51   0       0      0      0 S   0.0   0.0   0:00.00 idle_inject/0
     23 root      20   0       0      0      0 S   0.0   0.0   0:00.00 cpuhp/0
     24 root      20   0       0      0      0 S   0.0   0.0   0:00.00 kdevtmpfs
     25 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 kworker/R-inet_frag_wq
     26 root      20   0       0      0      0 S   0.0   0.0   0:00.00 kauditd
```

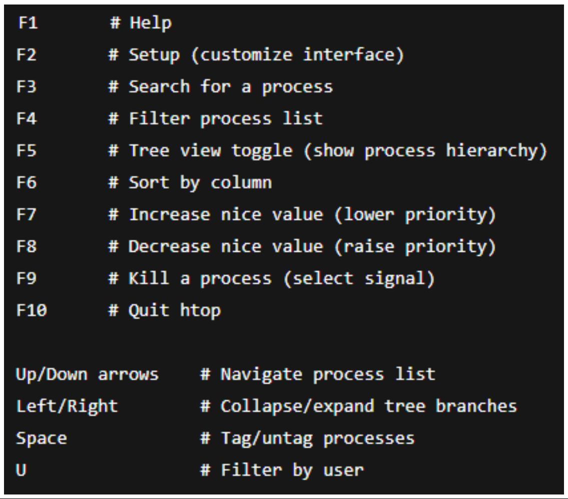
When running top it will take you to a screen showing process table and you can interact with it with the keyboard

- 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)

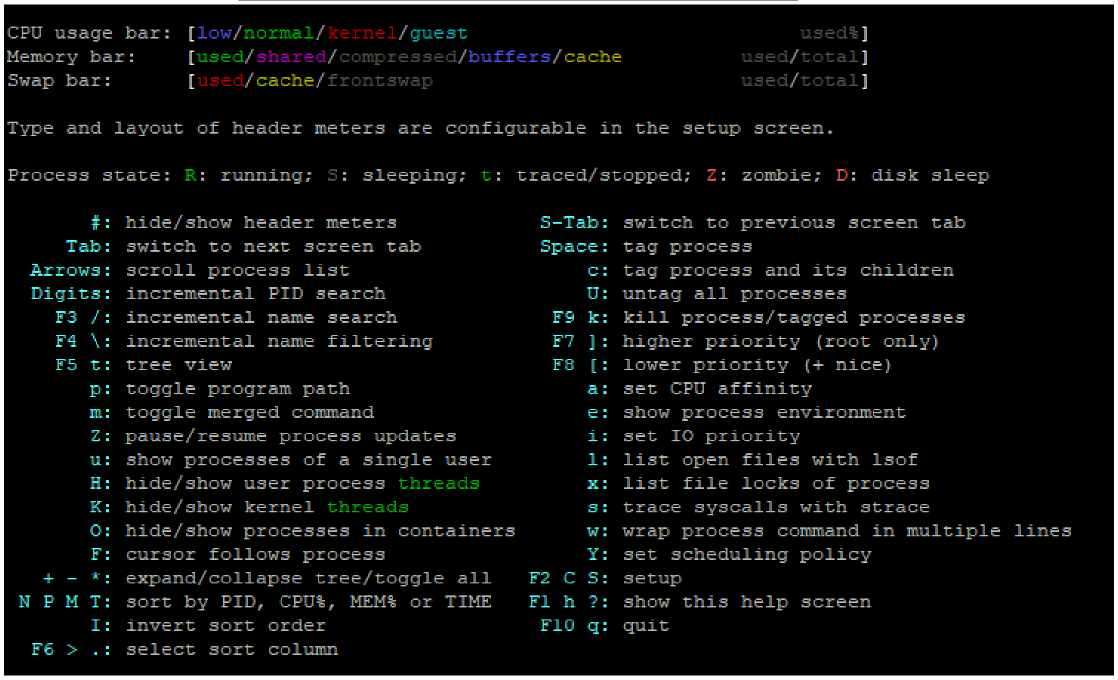
- **htop**: is interactive process viewer and system monitor, similar to top, but with a more user-friendly, and customizable interface. It displays system information, running processes, and resource usage in real-time.



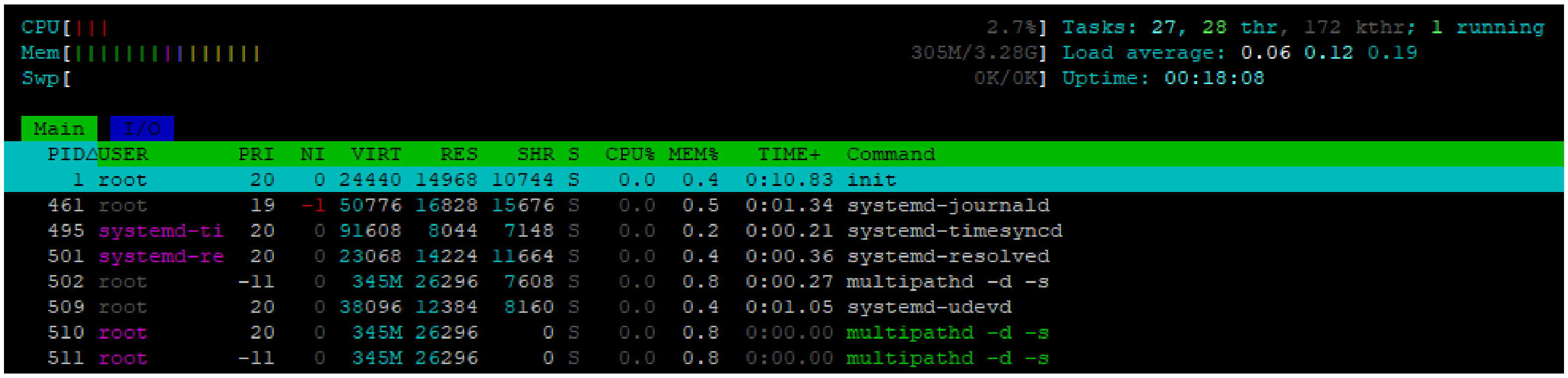
```
htop [OPTIONS]

-d, --delay=DELAY           # Set update delay time in tenths of seconds (e.g., 10 = 1 second)
-u, --user=USERNAME         # Show only processes of the specified user
-p, --pid=PID,...           # Show only given PIDs (comma-separated)
-s, --sort-key=COLUMN       # Sort by the specified column (e.g., PERCENT_CPU)
--sort-direction=[asc|desc] # Set sort direction (ascending or descending)
--tree                      # Show processes in a tree structure
--no-color                  # Disable color output
--no-header                 # Hide header (CPU/memory bars)
--help                      # Show help message and exit
--version                   # Show version information and exit
```



```
F1      # Help
F2      # Setup (customize interface)
F3      # Search for a process
F4      # Filter process list
F5      # Tree view toggle (show process hierarchy)
F6      # Sort by column
F7      # Increase nice value (lower priority)
F8      # Decrease nice value (raise priority)
F9      # Kill a process (select signal)
F10     # Quit htop

Up/Down arrows    # Navigate process list
Left/Right        # Collapse/expand tree branches
Space             # Tag/untag processes
U                 # Filter by user
```



```
CPU usage bar:  [low/normal/kernel/guest                            used%]
Memory bar:     [used/shared/compressed/buffers/cache          used/total]
Swap bar:       [used/cache/frontswap                          used/total]

Type and layout of header meters are configurable in the setup screen.

Process state: R: running; S: sleeping; t: traced/stopped; Z: zombie; D: disk sleep

      #: hide/show header meters            S-Tab: switch to previous screen tab
    Tab: switch to next screen tab          Space: tag process
 Arrows: scroll process list                    c: tag process and its children
 Digits: incremental PID search                 U: untag all processes
   F3 /: incremental name search             F9 k: kill process/tagged processes
   F4 \: incremental name filtering          F7 ]: higher priority (root only)
   F5 t: tree view                           F8 [: lower priority (+ nice)
      p: toggle program path                    a: set CPU affinity
      m: toggle merged command                  e: show process environment
      Z: pause/resume process updates           i: set IO priority
      u: show processes of a single user        l: list open files with lsof
      H: hide/show user process threads         x: list file locks of process
      K: hide/show kernel threads               s: trace syscalls with strace
      O: hide/show processes in containers      w: wrap process command in multiple lines
      F: cursor follows process                 Y: set scheduling policy
  + - *: expand/collapse tree/toggle all   F2 C S: setup
N P M T: sort by PID, CPU%, MEM% or TIME   F1 h ?: show this help screen
      I: invert sort order                  F10 q: quit
 F6 > .: select sort column
```



```
CPU[|||                                                     2.7%] Tasks: 27, 28 thr, 172 kthr; 1 running
Mem[||||||||||||||                                   305M/3.28G] Load average: 0.06 0.12 0.19
Swp[                                                      0K/0K] Uptime: 00:18:08

 Main  I/O
  PID△USER       PRI  NI  VIRT   RES   SHR S  CPU% MEM%   TIME+  Command
    1 root        20   0 24440 14968 10744 S   0.0  0.4  0:10.83 init
  461 root        19  -1 50776 16828 15676 S   0.0  0.5  0:01.34 systemd-journald
  495 systemd-ti  20   0 91608  8044  7148 S   0.0  0.2  0:00.21 systemd-timesyncd
  501 systemd-re  20   0 23068 14224 11664 S   0.0  0.4  0:00.36 systemd-resolved
  502 root       -11   0  345M 26296  7608 S   0.0  0.8  0:00.27 multipathd -d -s
  509 root        20   0 38096 12384  8160 S   0.0  0.4  0:01.05 systemd-udevd
  510 root        20   0  345M 26296     0 S   0.0  0.8  0:00.00 multipathd -d -s
  511 root       -11   0  345M 26296     0 S   0.0  0.8  0:00.00 multipathd -d -s
```

htop Column Headers Explained:

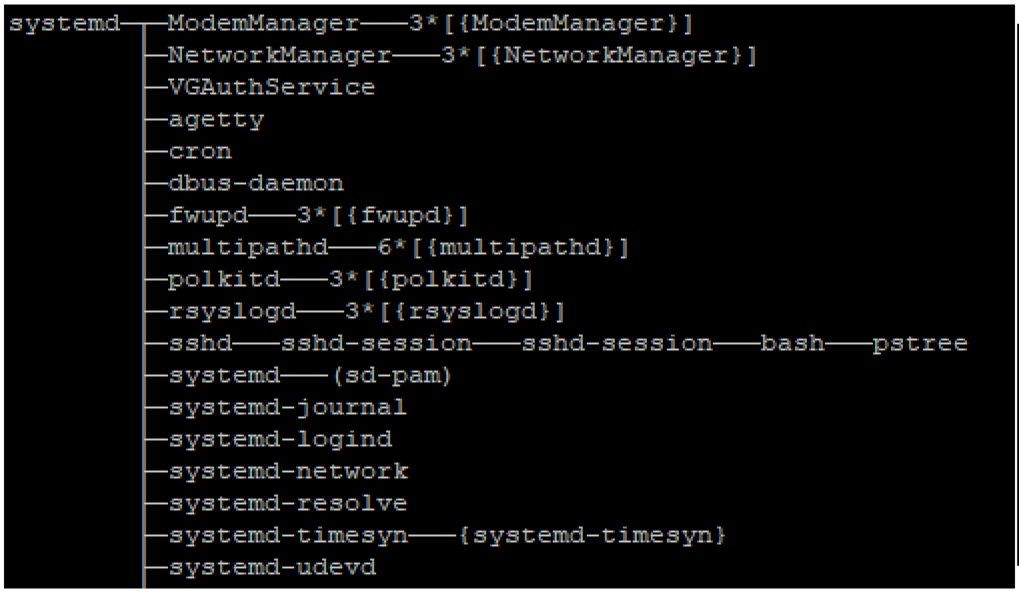
- **PID**: Process ID — the unique number assigned to each running process.
- **USER**: The owner (username) of the process.
- **PRI**: Priority — determines how much CPU time the process gets. Lower values mean higher priority.
- **NI**: Nice value — user-settable value that affects priority. Lower = 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.

```
pstree [OPTIONS] [PID or USER]


-a, --arguments           # Show command-line arguments for each process
-p, --show-pids           # Show PIDs next to process names
-u, --uid-names           # Show usernames instead of numeric UIDs
-n, --numeric-sort        # Sort child processes by PID instead of name
-h, --highlight-all       # Highlight the current process and its ancestors
-s, --show-parents        # Show only the parents of the specified PID
-G, --vt100               # Use VT100 line drawing characters
-l, --long                # Don't truncate long lines (full process names)
--help                    # Display help message and exit
```

```
pstree                    # Show all processes in a tree structure
pstree -p                 # Show the tree with PIDs
pstree -u                 # Show usernames instead of UID numbers
pstree -a                 # Show full command-line arguments
pstree -plua              # Combine options: PIDs, long lines, usernames, arguments
pstree 1234               # Show tree starting from process with PID 1234
pstree root               # Show processes owned by the user "root"
```



```
systemd─┬─ModemManager───3*[{ModemManager}]
        ├─NetworkManager───3*[{NetworkManager}]
        ├─VGAuthService
        ├─agetty
        ├─cron
        ├─dbus-daemon
        ├─fwupd───3*[{fwupd}]
        ├─multipathd───6*[{multipathd}]
        ├─polkitd───3*[{polkitd}]
        ├─rsyslogd───3*[{rsyslogd}]
        ├─sshd───sshd-session───sshd-session───bash───pstree
        ├─systemd───(sd-pam)
        ├─systemd-journal
        ├─systemd-logind
        ├─systemd-network
        ├─systemd-resolve
        ├─systemd-timesyn───{systemd-timesyn}
        ├─systemd-udevd
```

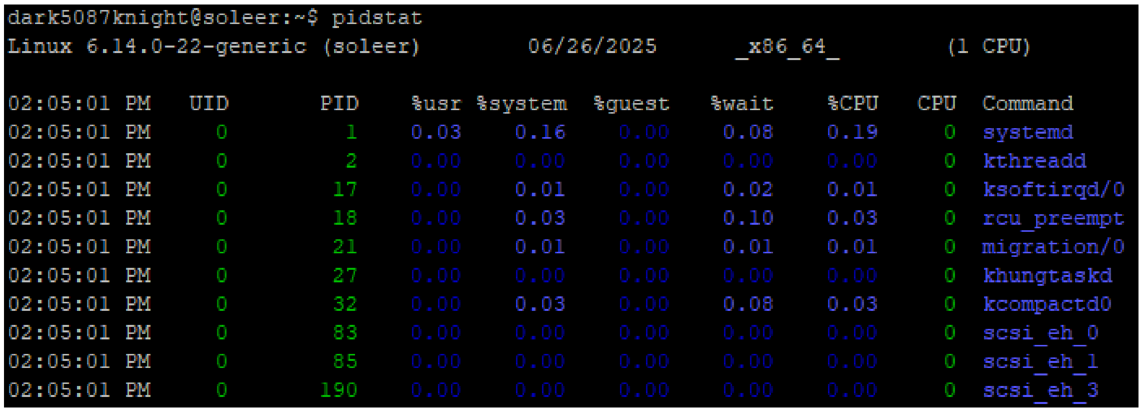
Running pstree with no options shows all system processes in a tree starting from PID 1, displaying only process names.

- **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.

```
pidstat [OPTIONS] [INTERVAL] [COUNT]

-u                          # Show CPU usage (default if no options are used)
-r                          # Show memory usage
-d                          # Show disk I/O statistics
-w                          # Show task switching activity
-h                          # Show thread statistics
-U USER                     # Show only processes belonging to a specific user
-p PID[,PID...]             # Show stats for specific process IDs
-G CMD[,CMD...]             # Show stats for specific command names
-I                          # Show I/O stats per thread
-T                          # Display per-thread statistics
-e                          # Show elapsed time and command line
--help                      # Show help message
```

```
pidstat                    # Show CPU usage for all processes
pidstat 2 5                # Show CPU stats every 2 seconds, 5 times
pidstat -u -r -d           # Show CPU, memory, and disk I/O stats
pidstat -p 1234            # Monitor process with PID 1234
pidstat -p 1234 1 10       # Monitor PID 1234 every second for 10 iterations
pidstat -u -G nginx        # Monitor CPU usage of all processes named nginx
```



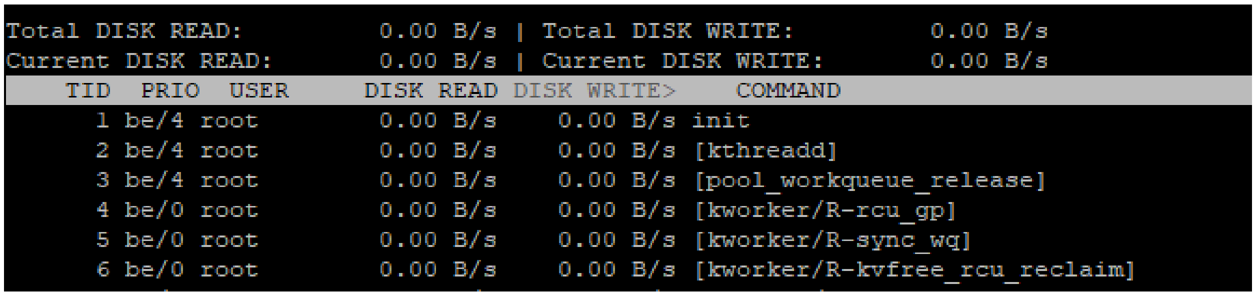
```
dark5087knight@soleer:~$ pidstat
Linux 6.14.0-22-generic (soleer)        06/26/2025      _x86_64_        (1 CPU)

02:05:01 PM   UID       PID    %usr %system  %guest   %wait    %CPU   CPU  Command
02:05:01 PM     0         1    0.03    0.16    0.00    0.08    0.19     0  systemd
02:05:01 PM     0         2    0.00    0.00    0.00    0.00    0.00     0  kthreadd
02:05:01 PM     0        17    0.00    0.01    0.00    0.02    0.01     0  ksoftirqd/0
02:05:01 PM     0        18    0.00    0.03    0.00    0.10    0.03     0  rcu_preempt
02:05:01 PM     0        21    0.00    0.01    0.00    0.01    0.01     0  migration/0
02:05:01 PM     0        27    0.00    0.00    0.00    0.00    0.00     0  khungtaskd
02:05:01 PM     0        32    0.00    0.03    0.00    0.08    0.03     0  kcompactd0
02:05:01 PM     0        83    0.00    0.00    0.00    0.00    0.00     0  scsi_eh_0
02:05:01 PM     0        85    0.00    0.00    0.00    0.00    0.00     0  scsi_eh_1
02:05:01 PM     0       190    0.00    0.00    0.00    0.00    0.00     0  scsi_eh_3
```

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.

```
iotop [OPTIONS]


-o, --only                # Show only processes or threads currently doing I/O
-b, --batch               # Run in non-interactive (batch) mode for logging
-n NUM                    # Number of iterations to run (used with -b)
-d SEC                    # Delay in seconds between updates (default: 1 second)
-p PID[,PID...]           # Monitor specific PIDs only
-u USER[,USER...]         # Monitor processes by specific user(s)
-k, --kilobytes           # Show I/O in KB/s instead of bytes/s
-t, --time                # Add a timestamp to each line (useful for logs)
--help                    # Show help and exit
```

```
iotop                    # Show all processes and their disk I/O
sudo iotop               # Must be run with root privileges
sudo iotop -o            # Show only processes doing actual I/O right now
sudo iotop -b -n 10      # Run 10 updates in batch mode for logging
sudo iotop -p 1234       # Monitor I/O usage of process with PID 1234
sudo iotop -u obaida     # Monitor I/O usage of processes owned by user "obaida"
```



```
Total DISK READ:         0.00 B/s | Total DISK WRITE:         0.00 B/s
Current DISK READ:       0.00 B/s | Current DISK WRITE:       0.00 B/s
    TID  PRIO  USER     DISK READ DISK WRITE>    COMMAND
      1 be/4 root        0.00 B/s    0.00 B/s init
      2 be/4 root        0.00 B/s    0.00 B/s [kthreadd]
      3 be/4 root        0.00 B/s    0.00 B/s [pool_workqueue_release]
      4 be/0 root        0.00 B/s    0.00 B/s [kworker/R-rcu_gp]
      5 be/0 root        0.00 B/s    0.00 B/s [kworker/R-sync_wq]
      6 be/0 root        0.00 B/s    0.00 B/s [kworker/R-kvfree_rcu_reclaim]
```

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.).

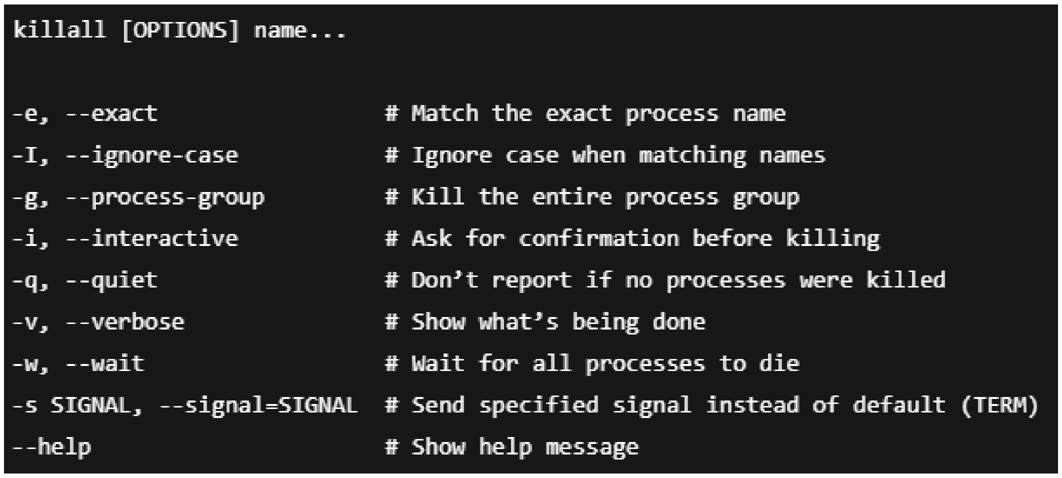
```
kill [OPTIONS] PID...

-s SIGNAL, --signal=SIGNAL   # Specify which signal to send (default is TERM)
-l, --list                   # List all signal names
-L                           # Same as --list, shows signals with numbers
--help                       # Show help and exit
```

```
TERM (15)   # Default, politely asks the process to terminate
KILL (9)    # Force kill, cannot be ignored
HUP (1)     # Hang up, often used to reload config
INT (2)     # Interrupt (like Ctrl+C)
STOP (19)   # Pause the process
CONT (18)   # Resume a paused process
```

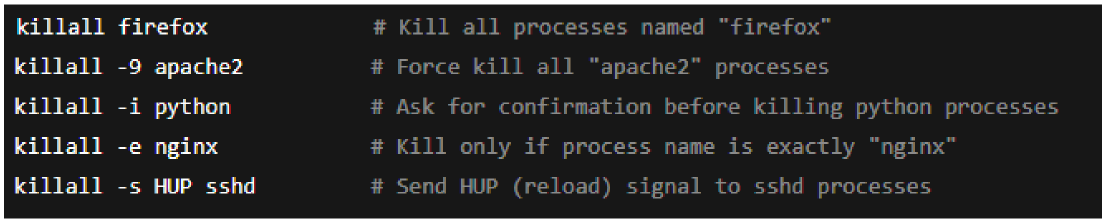
```
kill 1234                   # Send default TERM signal to PID 1234
kill -9 1234                # Force kill (KILL signal) to PID 1234
kill -s HUP 2222            # Send HUP (reload) signal to PID 2222
kill --list                 # List all available signals
kill -18 5678               # Resume a paused process (CONT)
```

- **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.

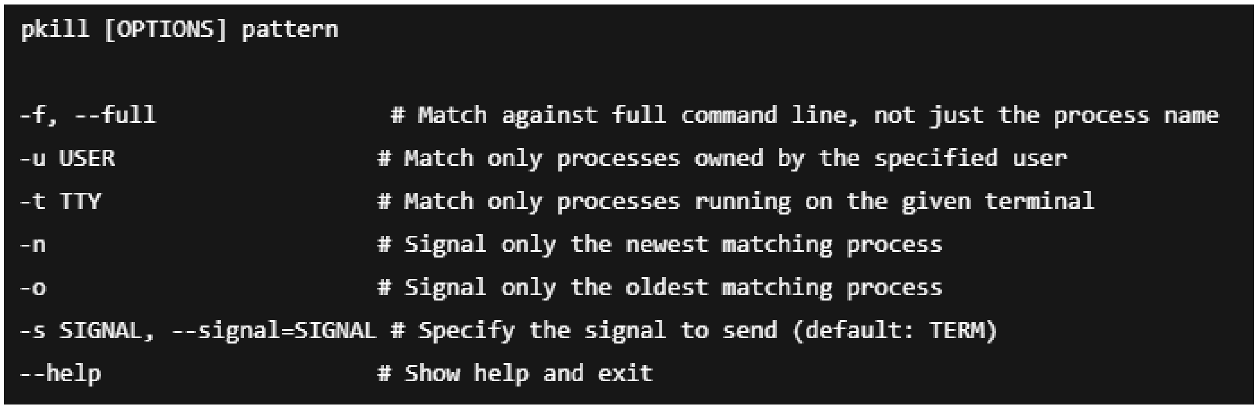


```
killall [OPTIONS] name...

-e, --exact                # Match the exact process name
-I, --ignore-case          # Ignore case when matching names
-g, --process-group        # Kill the entire process group
-i, --interactive          # Ask for confirmation before killing
-q, --quiet                # Don't report if no processes were killed
-v, --verbose              # Show what's being done
-w, --wait                 # Wait for all processes to die
-s SIGNAL, --signal=SIGNAL # Send specified signal instead of default (TERM)
--help                     # Show help message
```

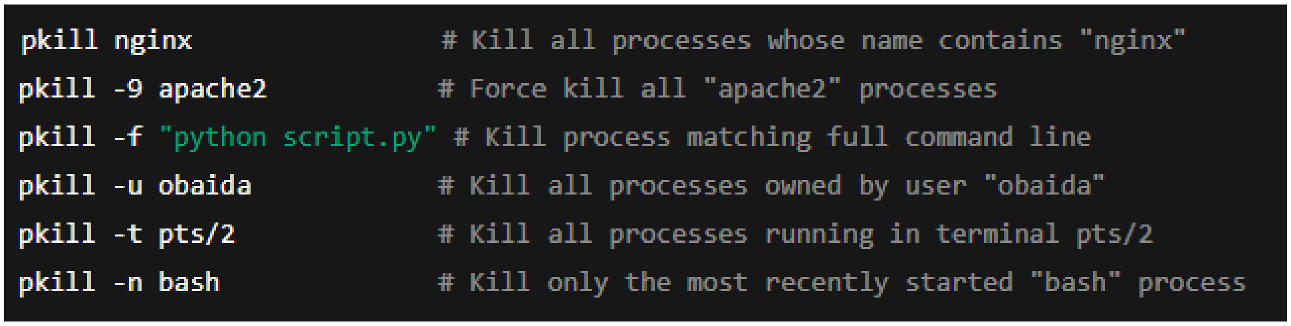


```
killall firefox            # Kill all processes named "firefox"
killall -9 apache2         # Force kill all "apache2" processes
killall -i python          # Ask for confirmation before killing python processes
killall -e nginx           # Kill only if process name is exactly "nginx"
killall -s HUP sshd        # Send HUP (reload) signal to sshd processes
```

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



```
pkill [OPTIONS] pattern

-f, --full                  # Match against full command line, not just the process name
-u USER                    # Match only processes owned by the specified user
-t TTY                     # Match only processes running on the given terminal
-n                         # Signal only the newest matching process
-o                         # Signal only the oldest matching process
-s SIGNAL, --signal=SIGNAL # Specify the signal to send (default: TERM)
--help                     # Show help and exit
```



```
pkill nginx                # Kill all processes whose name contains "nginx"
pkill -9 apache2           # Force kill all "apache2" processes
pkill -f "python script.py" # Kill process matching full command line
pkill -u obaida            # Kill all processes owned by user "obaida"
pkill -t pts/2             # Kill all processes running in terminal pts/2
pkill -n bash              # Kill only the most recently started "bash" process
```

- **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.

```
nice [OPTION] [COMMAND [ARG]...]

-n, --adjustment=N        # Set the nice value adjustment (-20 to 19; default is 10 if none given)
--help                   # Show help and exit
--version                # Show version information and exit
```

```
nice -n 10 ./backup.sh         # Run backup.sh with low CPU priority
nice ./long_task               # Run with default nice value (usually 10)
sudo nice -n -5 ./realtime_task  # Run with higher priority (negative nice, root only)
```

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

```
renice [OPTIONS] -n ADJUSTMENT -p PID...

-n, --adjustment=N          # New nice value (-20 to 19)
-p, --pid=PID              # Specify process IDs to adjust
-u, --user=USER            # Adjust nice for all processes owned by USER
-g, --pgrp=PGRP            # Adjust nice for process group
--help                     # Show help and exit
```

```
renice -n 10 -p 1234           # Lower priority of PID 1234 to nice 10
sudo renice -n -5 -p 5678      # Raise priority of PID 5678 (root required)
renice -n 15 -u obaida         # Change nice value for all processes owned by user "obaida"
renice -n 5 -g 4321            # Adjust nice value for process group 4321
```

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:

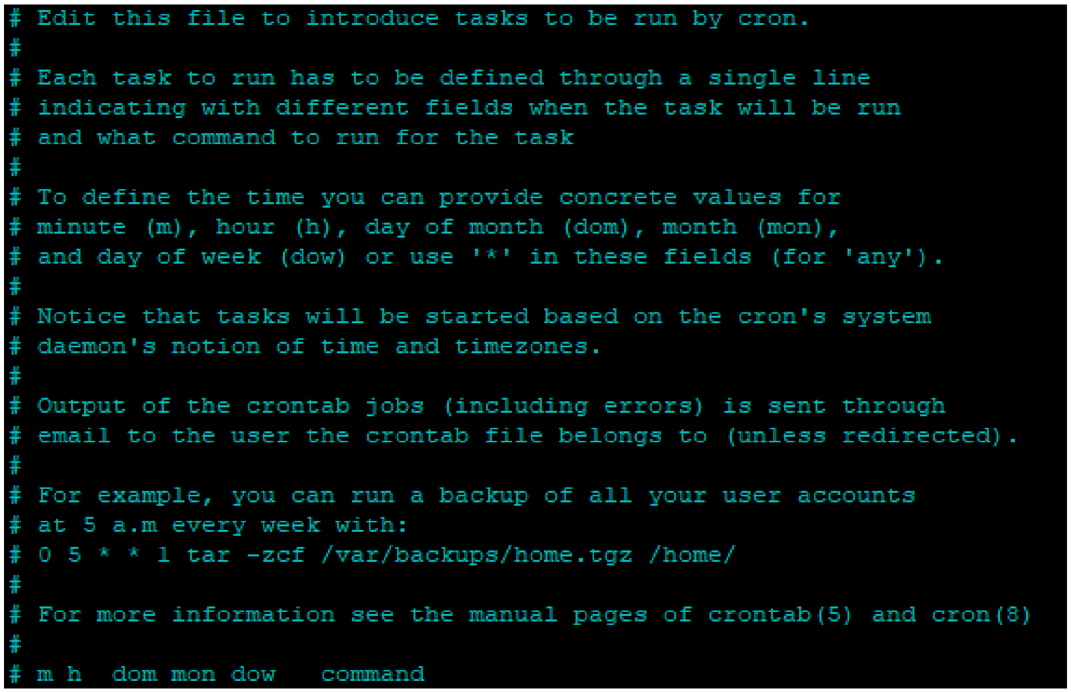
cron: is a time-based job scheduler in Linux that runs scripts or commands automatically at specified times and intervals.

Format:

```
* * * * * command_to_run
- - - - -
| | | | |
| | | | +----- Day of the week (0-7) (Sunday=0 or 7)
| | | +------- Month (1-12)
| | +--------- Day of the month (1-31)
| +----------- Hour (0-23)
+------------- Minute (0-59)
```

Usage:

```
crontab -e        # Edit current user's cron jobs
crontab -l        # List current user's cron jobs
crontab -r        # Remove current user's cron jobs
```



```
# Edit this file to introduce tasks to be run by cron.
#
# Each task to run has to be defined through a single line
# indicating with different fields when the task will be run
# and what command to run for the task
#
# To define the time you can provide concrete values for
# minute (m), hour (h), day of month (dom), month (mon),
# and day of week (dow) or use '*' in these fields (for 'any').
#
# Notice that tasks will be started based on the cron's system
# daemon's notion of time and timezones.
#
# Output of the crontab jobs (including errors) is sent through
# email to the user the crontab file belongs to (unless redirected).
#
# For example, you can run a backup of all your user accounts
# at 5 a.m every week with:
# 0 5 * * 1 tar -zcf /var/backups/home.tgz /home/
#
# For more information see the manual pages of crontab(5) and cron(8)
#
# m h  dom mon dow   command
```

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

```
0 2 * * * /home/obaida/backup.sh
```

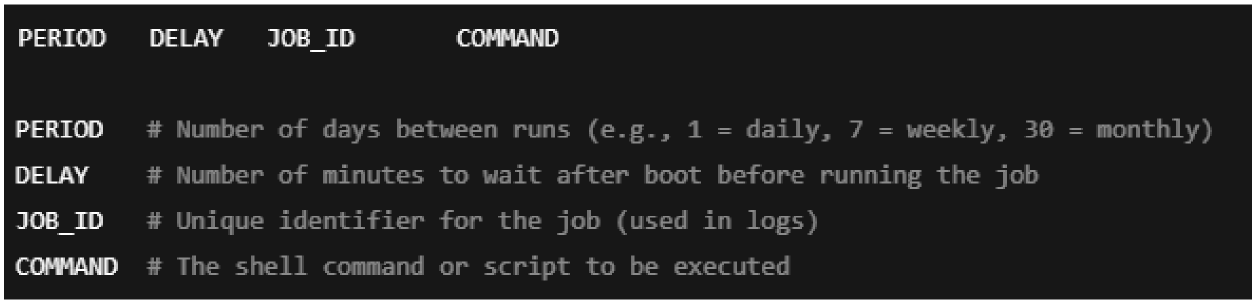
anacron: is a Linux tool used to schedule periodic jobs on systems that aren’t running 24/7.
Unlike cron, it guarantees that jobs will eventually run even if the system was off at the scheduled time.

When to Use It:

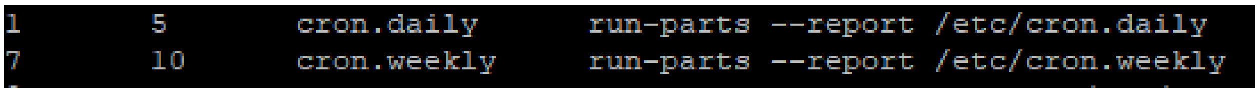
- You’re on a laptop, desktop, or non-server system that shuts down or suspends often.
- You need to guarantee periodic jobs still run, just maybe not at the exact time.
- Example: daily backup, log cleanup, or sync tasks.

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.



```
PERIOD   DELAY   JOB_ID      COMMAND

PERIOD   # Number of days between runs (e.g., 1 = daily, 7 = weekly, 30 = monthly)
DELAY    # Number of minutes to wait after boot before running the job
JOB_ID   # Unique identifier for the job (used in logs)
COMMAND  # The shell command or script to be executed
```



```
1       5       cron.daily      run-parts --report /etc/cron.daily
7       10      cron.weekly     run-parts --report /etc/cron.weekly
```

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

```
at [OPTION]... TIME

  -f FILE           # Read commands from a file instead of typing them manually
  -m                # Send mail to the user even if the job produces no output
  -q QUEUE          # Use a specific job queue (a-z, A-Z); default is 'a'
  -t TIME_SPEC      # Specify time in [[CC]YY]MMDDhhmm[.ss] format
  -u USERNAME       # Run the job as the specified user (requires root privileges)
  -V                # Show version information and exit
  -h, --help        # Display help message and exit
```

```
# Examples of using `at` with options and time formats:

at now + 10 minutes             # Schedule a command to run 10 minutes from now (interactive input)
at -f /home/obaida/script.sh 14:00   # Run commands from script.sh at 2:00 PM today.
at -m 23:30                     # Run a command at 11:30 PM and send mail even if no output.
at -q b now + 1 hour            # Schedule job in queue 'b' to run 1 hour from now.
at -t 202506271830 -f /home/obaida/cleanup.sh   # Run cleanup.sh at June 27, 2025 18:30 using preci
```

To list the scheduled jobs

```
atq [OPTION]...

  -q QUEUE        # Show jobs only in the specified queue (a-z, A-Z)
  -v              # Verbose output (may not be supported on all systems)
  -V              # Display version information and exit
  -h, --help      # Show help message and exit
```

To remove a scheduled job:

```
atrm [OPTION]... JOB_ID...
```

```
atrm 2
```

```
atrm 4 5 6
```

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

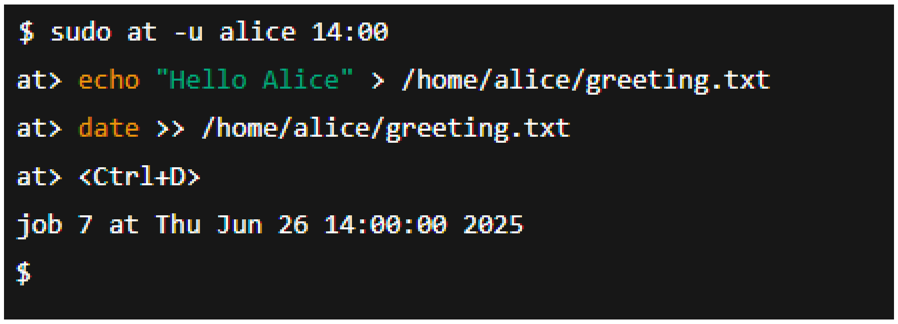
Important:

- 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

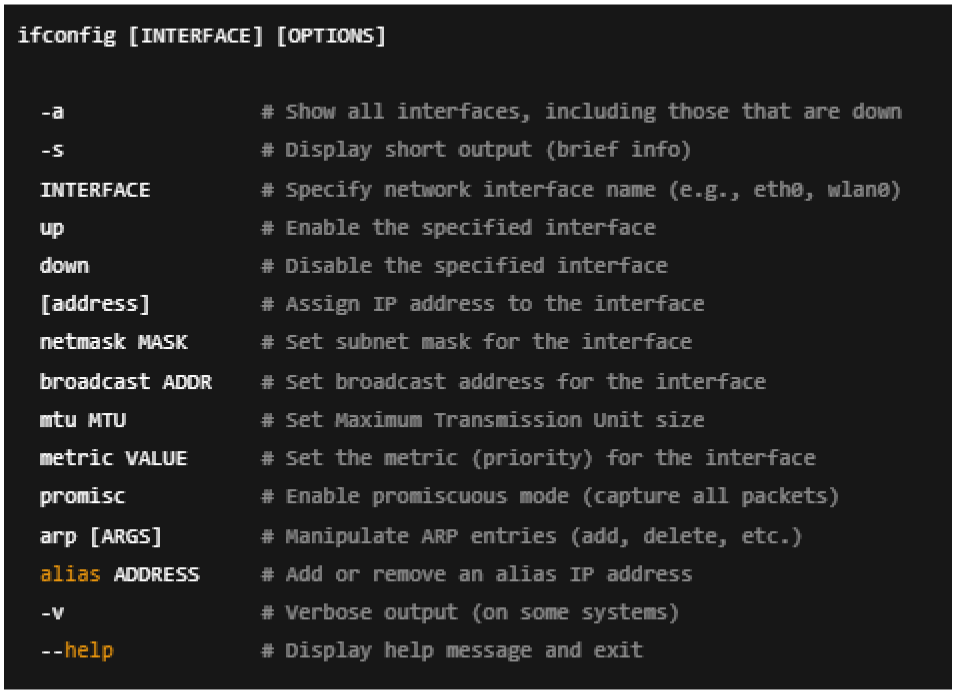
it will go to interactive mode to write the jobs must be done



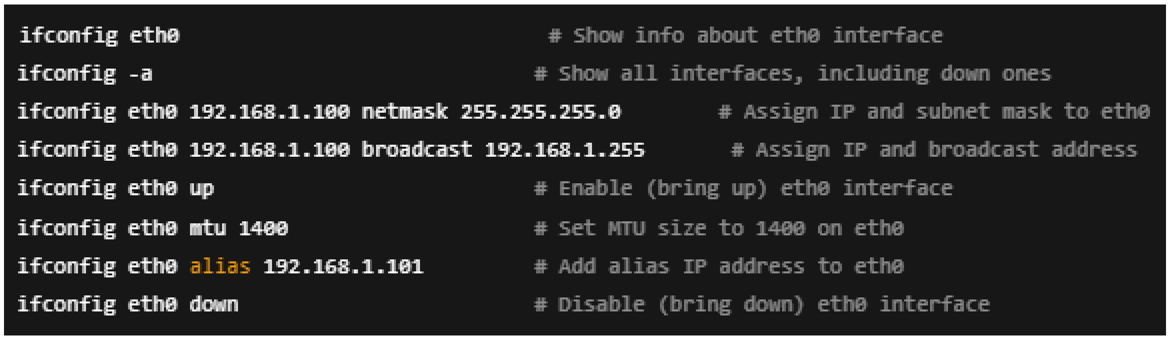
```
$ sudo at -u alice 14:00
at> echo "Hello Alice" > /home/alice/greeting.txt
at> date >> /home/alice/greeting.txt
at> <Ctrl+D>
job 7 at Thu Jun 26 14:00:00 2025
$
```

5.1- Network Configuration commands:

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



```
ifconfig [INTERFACE] [OPTIONS]

  -a                 # Show all interfaces, including those that are down
  -s                 # Display short output (brief info)
  INTERFACE          # Specify network interface name (e.g., eth0, wlan0)
  up                 # Enable the specified interface
  down               # Disable the specified interface
  [address]          # Assign IP address to the interface
  netmask MASK       # Set subnet mask for the interface
  broadcast ADDR     # Set broadcast address for the interface
  mtu MTU            # Set Maximum Transmission Unit size
  metric VALUE       # Set the metric (priority) for the interface
  promisc            # Enable promiscuous mode (capture all packets)
  arp [ARGS]         # Manipulate ARP entries (add, delete, etc.)
  alias ADDRESS      # Add or remove an alias IP address
  -v                 # Verbose output (on some systems)
  --help             # Display help message and exit
```



```
ifconfig eth0                                  # Show info about eth0 interface
ifconfig -a                                   # Show all interfaces, including down ones
ifconfig eth0 192.168.1.100 netmask 255.255.255.0        # Assign IP and subnet mask to eth0
ifconfig eth0 192.168.1.100 broadcast 192.168.1.255       # Assign IP and broadcast address
ifconfig eth0 up                              # Enable (bring up) eth0 interface
ifconfig eth0 mtu 1400                        # Set MTU size to 1400 on eth0
ifconfig eth0 alias 192.168.1.101             # Add alias IP address to eth0
ifconfig eth0 down                            # Disable (bring down) eth0 interface
```

- **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.

```
iwconfig [interface] [OPTIONS]

OPTIONS:
  essid <name>          # Set or get the network name (SSID)
  mode <mode>           # Set mode (managed, ad-hoc, master, monitor)
  freq <frequency>      # Set operating frequency/channel
  channel <num>         # Alias for frequency
  rate <bitrate>        # Set transmission rate (e.g. 54M)
  txpower <dbm>         # Set transmit power
  rts <threshold>       # Set RTS/CTS threshold
  frag <threshold>      # Set fragmentation threshold
  key <WEP_key>         # Set WEP encryption key
  power on|off          # Enable or disable power management
  ap <MAC>              # Set specific access point MAC address
```

```
iwconfig                           # Show wireless settings for all interfaces
iwconfig wlan0                     # Show info for wlan0 (SSID, signal, link quality)
iwconfig wlan0 essid "HomeNetwork" # Set the SSID to connect to
iwconfig wlan0 mode managed        # Set mode to client/managed mode
iwconfig wlan0 rate 54M            # Set Wi-Fi transmit speed to 54 Mbps
iwconfig wlan0 txpower 15          # Set transmit power to 15 dBm
iwconfig wlan0 power off           # Disable power management
```

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.

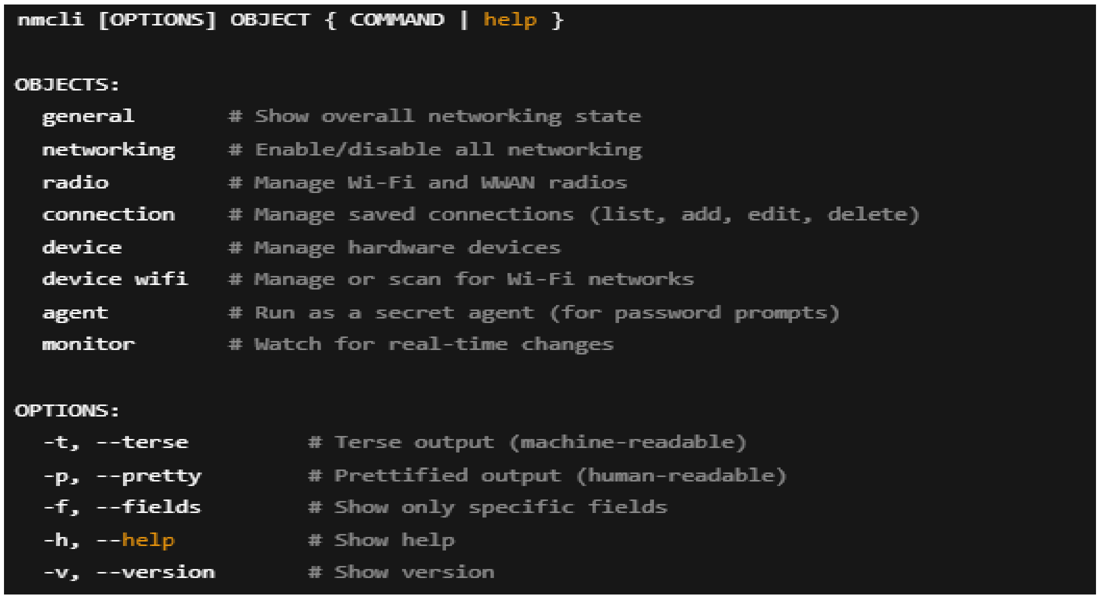
```
ip [ OPTIONS ] OBJECT { COMMAND | help }

OPTIONS:
  -s, -statistics         # Show detailed statistics
  -d, -details            # Show detailed information
  -o, -oneline            # Output in one line format
  -r, -resolve            # Resolve host addresses

OBJECTS:
  link        # Network interfaces
  addr        # IP addresses
  route       # Routing tables
  neigh       # ARP and neighbor cache
  tunnel      # Tunnels
  maddr       # Multicast addresses
  netns       # Network namespaces

COMMANDS vary per OBJECT, e.g.:

link      show, set, add, del, up, down
addr      show, add, del, flush
route     show, add, del
neigh     show, add, del
tunnel    add, del
```

```
ip addr show                        # Show all IP addresses and their interfaces
ip link set eth0 up                 # Bring interface eth0 up (enable)
ip link set wlan0 down              # Bring interface wlan0 down (disable)
ip addr add 192.168.1.100/24 dev eth0  # Assign IP address with subnet mask to eth0
ip addr del 192.168.1.100/24 dev eth0  # Remove IP address from eth0
ip route show                      # Display current routing table
ip route add default via 192.168.1.1  # Add default gateway route
ip neigh show                     # Show ARP (neighbor) cache entries
```

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

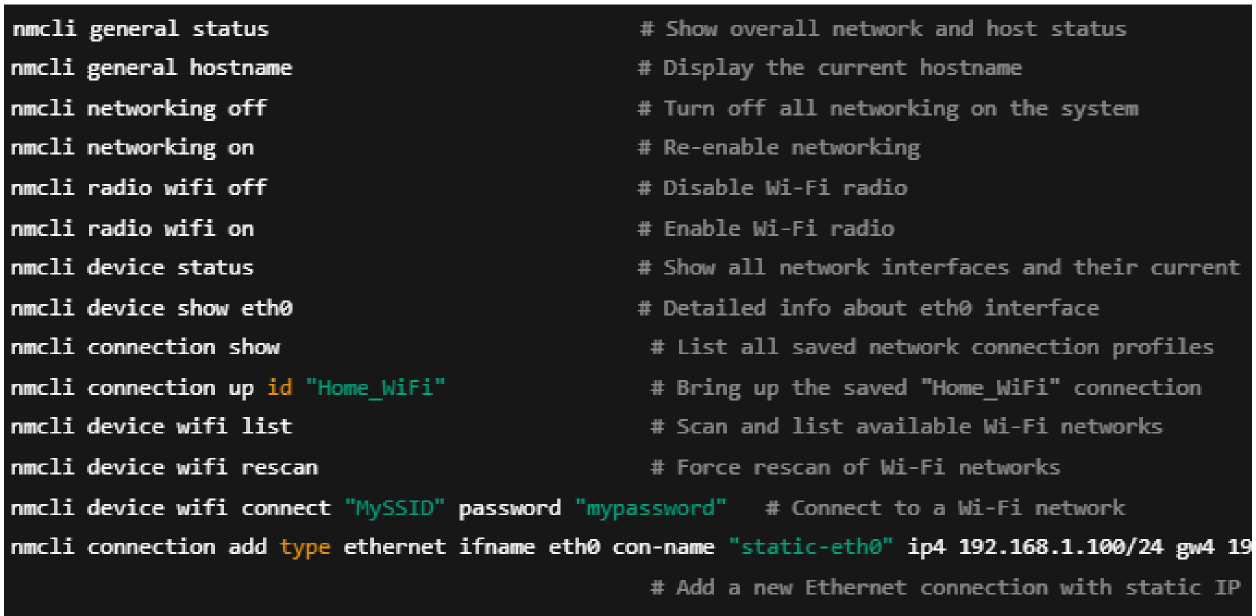
It's great for:

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



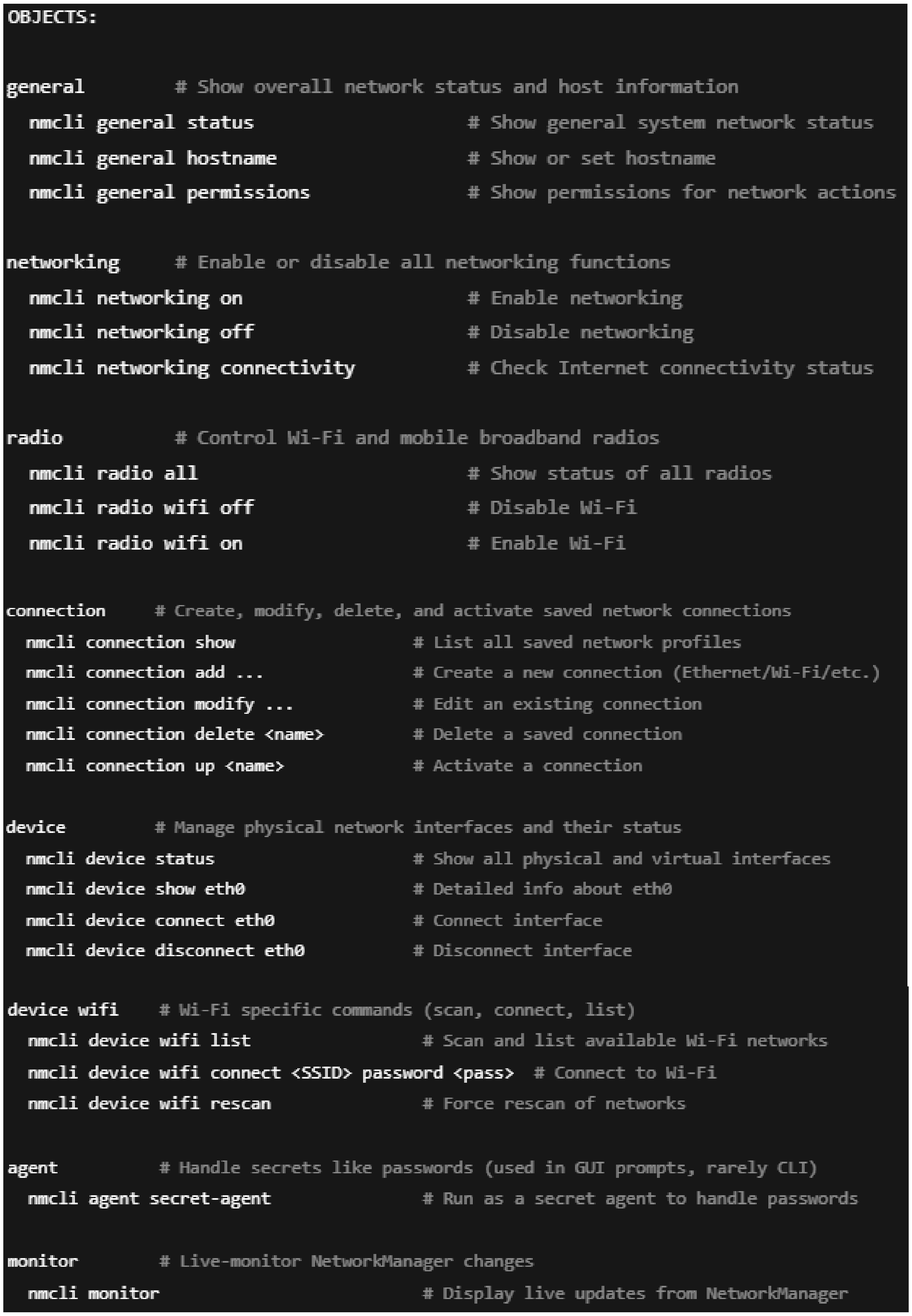
```
nmcli [OPTIONS] OBJECT { COMMAND | help }

OBJECTS:
  general        # Show overall networking state
  networking     # Enable/disable all networking
  radio          # Manage Wi-Fi and WWAN radios
  connection     # Manage saved connections (list, add, edit, delete)
  device         # Manage hardware devices
  device wifi    # Manage or scan for Wi-Fi networks
  agent          # Run as a secret agent (for password prompts)
  monitor        # Watch for real-time changes

OPTIONS:
  -t, --terse          # Terse output (machine-readable)
  -p, --pretty         # Prettified output (human-readable)
  -f, --fields         # Show only specific fields
  -h, --help           # Show help
  -v, --version        # Show version
```

Examples:



```
nmcli general status                                # Show overall network and host status
nmcli general hostname                              # Display the current hostname
nmcli networking off                                # Turn off all networking on the system
nmcli networking on                                 # Re-enable networking
nmcli radio wifi off                                # Disable Wi-Fi radio
nmcli radio wifi on                                 # Enable Wi-Fi radio
nmcli device status                                 # Show all network interfaces and their current
nmcli device show eth0                              # Detailed info about eth0 interface
nmcli connection show                                # List all saved network connection profiles
nmcli connection up id "Home_WiFi"                   # Bring up the saved "Home_WiFi" connection
nmcli device wifi list                               # Scan and list available Wi-Fi networks
nmcli device wifi rescan                             # Force rescan of Wi-Fi networks
nmcli device wifi connect "MySSID" password "mypassword"   # Connect to a Wi-Fi network
nmcli connection add type ethernet ifname eth0 con-name "static-eth0" ip4 192.168.1.100/24 gw4 19
                                                     # Add a new Ethernet connection with static IP
```

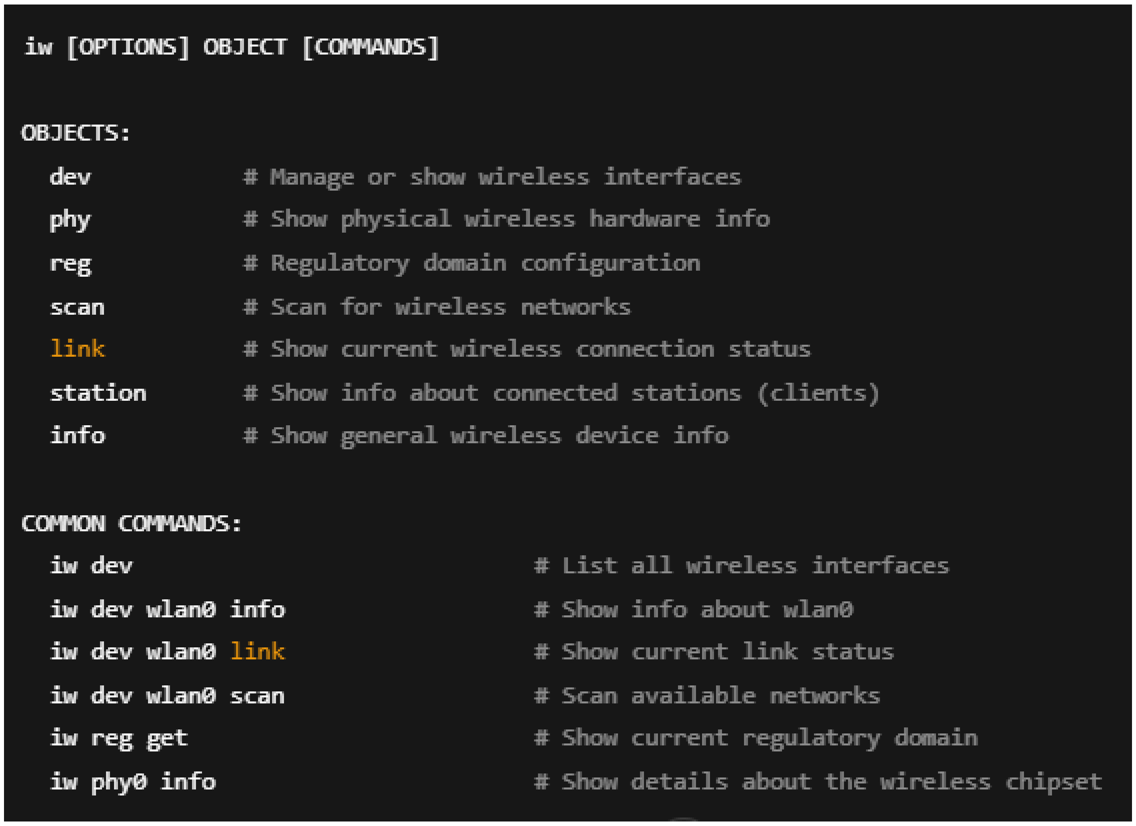
the Object field has different input for each one and here are the inputs:



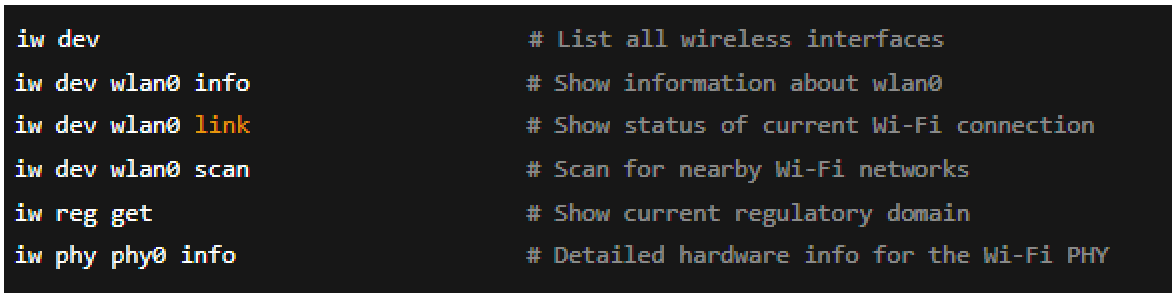
```
OBJECTS:

general        # Show overall network status and host information
  nmcli general status                    # Show general system network status
  nmcli general hostname                  # Show or set hostname
  nmcli general permissions               # Show permissions for network actions

networking     # Enable or disable all networking functions
  nmcli networking on                     # Enable networking
  nmcli networking off                    # Disable networking
  nmcli networking connectivity           # Check Internet connectivity status

radio          # Control Wi-Fi and mobile broadband radios
  nmcli radio all                         # Show status of all radios
  nmcli radio wifi off                    # Disable Wi-Fi
  nmcli radio wifi on                     # Enable Wi-Fi

connection     # Create, modify, delete, and activate saved network connections
  nmcli connection show                 # List all saved network profiles
  nmcli connection add ...              # Create a new connection (Ethernet/Wi-Fi/etc.)
  nmcli connection modify ...           # Edit an existing connection
  nmcli connection delete <name>        # Delete a saved connection
  nmcli connection up <name>            # Activate a connection

device         # Manage physical network interfaces and their status
  nmcli device status                   # Show all physical and virtual interfaces
  nmcli device show eth0                # Detailed info about eth0
  nmcli device connect eth0             # Connect interface
  nmcli device disconnect eth0          # Disconnect interface

device wifi    # Wi-Fi specific commands (scan, connect, list)
  nmcli device wifi list                # Scan and list available Wi-Fi networks
  nmcli device wifi connect <SSID> password <pass>  # Connect to Wi-Fi
  nmcli device wifi rescan              # Force rescan of networks

agent          # Handle secrets like passwords (used in GUI prompts, rarely CLI)
  nmcli agent secret-agent              # Run as a secret agent to handle passwords

monitor        # Live-monitor NetworkManager changes
  nmcli monitor                         # Display live updates from NetworkManager
```

- **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).



```
iw [OPTIONS] OBJECT [COMMANDS]

OBJECTS:
  dev           # Manage or show wireless interfaces
  phy           # Show physical wireless hardware info
  reg           # Regulatory domain configuration
  scan          # Scan for wireless networks
  link          # Show current wireless connection status
  station       # Show info about connected stations (clients)
  info          # Show general wireless device info

COMMON COMMANDS:
  iw dev                              # List all wireless interfaces
  iw dev wlan0 info                   # Show info about wlan0
  iw dev wlan0 link                   # Show current link status
  iw dev wlan0 scan                   # Scan available networks
  iw reg get                          # Show current regulatory domain
  iw phy0 info                        # Show details about the wireless chipset
```



```
iw dev                                # List all wireless interfaces
iw dev wlan0 info                     # Show information about wlan0
iw dev wlan0 link                     # Show status of current Wi-Fi connection
iw dev wlan0 scan                     # Scan for nearby Wi-Fi networks
iw reg get                            # Show current regulatory domain
iw phy phy0 info                      # Detailed hardware info for the Wi-Fi PHY
```

- **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.

```
iwlist [interface] [command]

COMMANDS:
  scan            # Scan for wireless networks
  frequency       # List available frequencies
  channel         # Alias for frequency
  rate            # Show supported bitrates
  encryption      # Show encryption capabilities (WEP/WPA/etc.)
  power           # Show power management status
  txpower         # Show transmit power levels
  retry           # Show retry limits
  ap              # List access points in range
  auth            # Show authentication modes
  event           # Show pending wireless events
```

```
iwlist wlan0 scan                      # Scan for all wireless networks on wlan0
iwlist wlan0 frequency                 # Show all supported frequencies
iwlist wlan0 rate                      # Show supported bitrates for wlan0
iwlist wlan0 encryption                # Show encryption settings for each SSID
iwlist wlan0 txpower                   # Show transmit power info
iwlist wlan0 power                     # Show power management status
```

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.

```
ping [OPTIONS] DESTINATION

OPTIONS:
  -c COUNT       # Send only COUNT echo requests, then stop
  -i INTERVAL    # Wait INTERVAL seconds between sending packets (default 1s)
  -s PACKET_SIZE # Specify the number of data bytes to send
  -t TTL         # Set Time To Live (max hops)
  -W TIMEOUT     # Time to wait for a reply, in seconds
  -q             # Quiet output — only summary at the end
  -4             # Use IPv4 only
  -6             # Use IPv6 only
  --help         # Show help message and exit
```

```
ping google.com                # Ping google.com continuously until interrupted
ping -c 4 8.8.8.8              # Send 4 pings to 8.8.8.8 and stop
ping -i 0.5 192.168.1.1        # Ping every 0.5 seconds to local router
ping -s 1000 example.com       # Send 1000 bytes of data per ping
ping -t 64 example.com         # Set TTL to 64 hops
ping -q google.com             # Quiet mode — only summary displayed
ping -6 ipv6.google.com        # Ping using IPv6
```

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.

```
traceroute [OPTIONS] DESTINATION

OPTIONS:
  -m MAX_TTL         # Set maximum number of hops (default 30)
  -q NUM_QUERIES     # Number of probe packets per hop (default 3)
  -w WAIT_TIME       # Wait time for each reply in seconds (default 5)
  -I                 # Use ICMP ECHO instead of UDP packets
  -T                 # Use TCP SYN packets (useful for firewall traversal)
  -p PORT            # Set destination port (default 33434)
  -n                 # Do not resolve IP addresses to hostnames
  -h, --help         # Show help and exit
```

```
traceroute google.com                # Trace route to google.com using default UDP probes
traceroute -m 20 example.com        # Limit max hops to 20
traceroute -q 1 example.com         # Send 1 probe per hop instead of 3
traceroute -w 2 example.com         # Wait 2 seconds for each reply
traceroute -I example.com           # Use ICMP echo requests (like ping)
traceroute -T -p 80 example.com     # Use TCP SYN probes to port 80
traceroute -n 8.8.8.8               # Show IP addresses only (no hostname resolution)
```

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

```
netstat [OPTIONS]

OPTIONS:
  -a                 # Show all sockets (listening and non-listening)
  -t                 # Show TCP connections only
  -u                 # Show UDP connections only
  -l                 # Show listening sockets
  -p                 # Show process PID and name using the socket
  -n                 # Show numerical addresses (no hostname resolution)
  -r                 # Show routing table
  -i                 # Show network interfaces and statistics
  -s                 # Show network protocol statistics
  --help             # Show help message and exit
```

```
netstat -a                   # Show all active sockets and connections
netstat -tuln                # Show all listening TCP and UDP ports numerically
netstat -p                  # Show processes using sockets (requires root)
netstat -r                   # Show the kernel routing table
netstat -i                   # Show network interface stats
netstat -s                   # Display network protocol statistics
netstat -tunap               # Show all TCP/UDP connections with PID and numeric addresses
```

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.

```
ss [OPTIONS]

OPTIONS:
  -t               # Show TCP sockets only
  -u               # Show UDP sockets only
  -a               # Show all sockets (listening and non-listening)
  -l               # Show only listening sockets
  -p               # Show process using the socket (requires root)
  -n               # Don't resolve service names or hostnames (numeric output)
  -r               # Show routing info (less common in ss)
  -s               # Summary statistics for sockets
  -4               # Show only IPv4 sockets
  -6               # Show only IPv6 sockets
  --help           # Show help message and exit
```

```
ss -tuln                   # Show listening TCP and UDP sockets, numeric addresses
ss -p                      # Show processes using sockets (requires sudo)
ss -s                      # Summary of socket statistics
ss -a                      # Show all sockets (listening and connected)
ss -t state ESTABLISHED    # Show only established TCP connections
ss -4 -t                   # Show IPv4 TCP sockets only
ss -6 -u                   # Show IPv6 UDP sockets only
```

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:

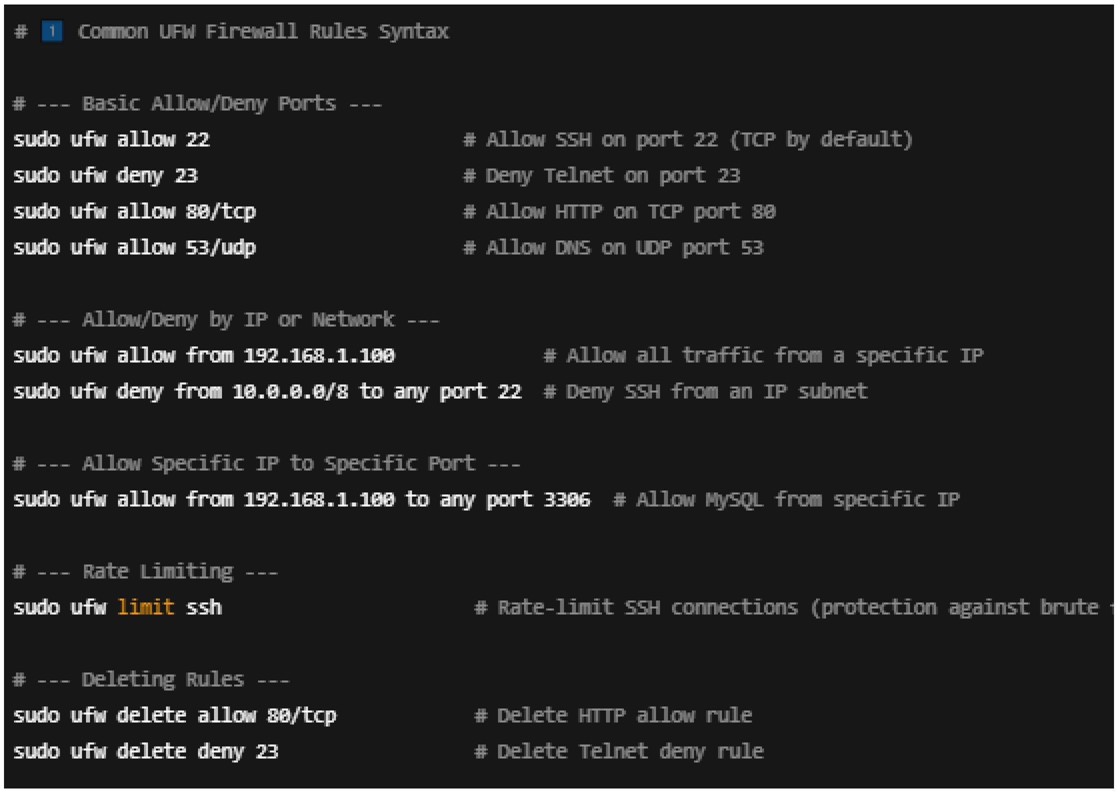
ufw: is a user-friendly front-end to iptables that simplifies firewall configuration, designed especially for Ubuntu and Debian-based systems.

```
ufw [OPTIONS] COMMAND [RULES]

OPTIONS:
  --force           # Skip confirmation prompts (useful in scripts)
  --dry-run         # Show what would be done without making changes
  --quiet           # Suppress output messages
  --help            # Display help message and exit
  --version         # Show ufw version info

COMMANDS:
  enable              # Enable the firewall
  disable             # Disable the firewall
  status              # Show current firewall status and rules
  allow [PORT/PROTO]  # Allow incoming traffic on port/protocol
  deny [PORT/PROTO]   # Deny incoming traffic on port/protocol
  delete [RULE]       # Remove a rule (by number or specification)
  logging [on|off]    # Enable or disable logging
  reset               # Reset firewall rules to default (disable and clear)
  reload              # Reload firewall rules
  default [allow|deny] # Set default incoming policy
  help                # Show help message
```

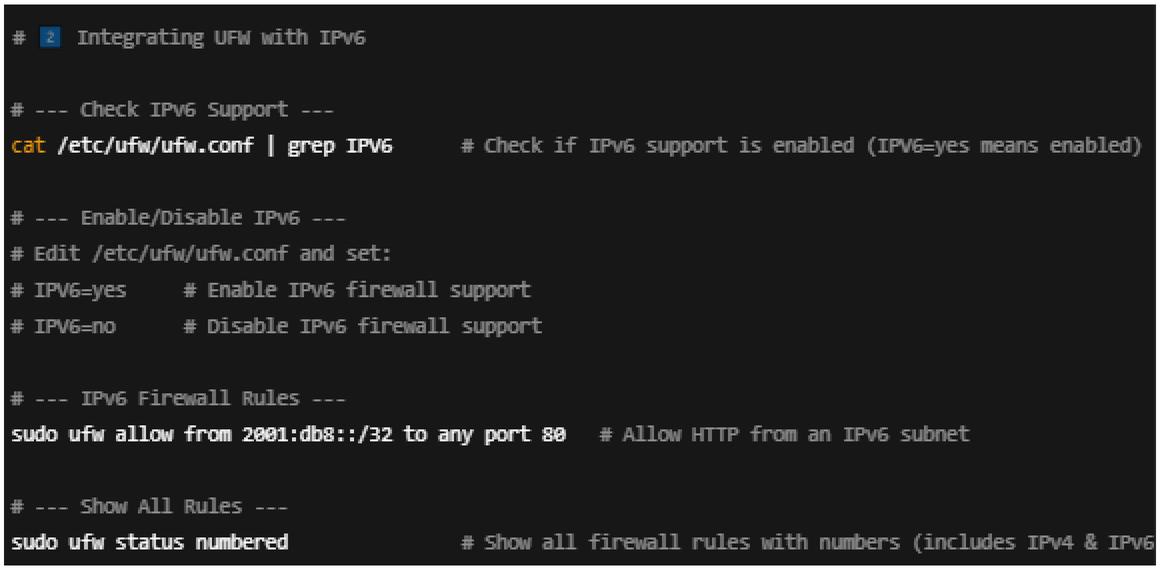
Examples:

```
sudo ufw --force enable
```

```
sudo ufw enable                       # Enable UFW firewall
sudo ufw status                      # Show active rules and status
sudo ufw allow 22                    # Allow incoming SSH on port 22
sudo ufw allow 80/tcp                # Allow incoming HTTP traffic
sudo ufw deny 23                     # Deny incoming Telnet traffic
sudo ufw delete allow 80/tcp         # Delete rule allowing HTTP
sudo ufw logging on                  # Enable firewall logging
sudo ufw reset                      # Disable and reset all firewall rules
```



```
# 1️⃣ Common UFW Firewall Rules Syntax

# --- Basic Allow/Deny Ports ---
sudo ufw allow 22                      # Allow SSH on port 22 (TCP by default)
sudo ufw deny 23                       # Deny Telnet on port 23
sudo ufw allow 80/tcp                  # Allow HTTP on TCP port 80
sudo ufw allow 53/udp                  # Allow DNS on UDP port 53

# --- Allow/Deny by IP or Network ---
sudo ufw allow from 192.168.1.100             # Allow all traffic from a specific IP
sudo ufw deny from 10.0.0.0/8 to any port 22  # Deny SSH from an IP subnet

# --- Allow Specific IP to Specific Port ---
sudo ufw allow from 192.168.1.100 to any port 3306  # Allow MySQL from specific IP

# --- Rate Limiting ---
sudo ufw limit ssh                      # Rate-limit SSH connections (protection against brute

# --- Deleting Rules ---
sudo ufw delete allow 80/tcp            # Delete HTTP allow rule
sudo ufw delete deny 23                 # Delete Telnet deny rule
```



```
# 2️⃣ Integrating UFW with IPv6

# --- Check IPv6 Support ---
cat /etc/ufw/ufw.conf | grep IPV6      # Check if IPv6 support is enabled (IPV6=yes means enabled)

# --- Enable/Disable IPv6 ---
# Edit /etc/ufw/ufw.conf and set:
# IPV6=yes     # Enable IPv6 firewall support
# IPV6=no      # Disable IPv6 firewall support

# --- IPv6 Firewall Rules ---
sudo ufw allow from 2001:db8::/32 to any port 80   # Allow HTTP from an IPv6 subnet

# --- Show All Rules ---
sudo ufw status numbered               # Show all firewall rules with numbers (includes IPv4 & IPv6
```

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.

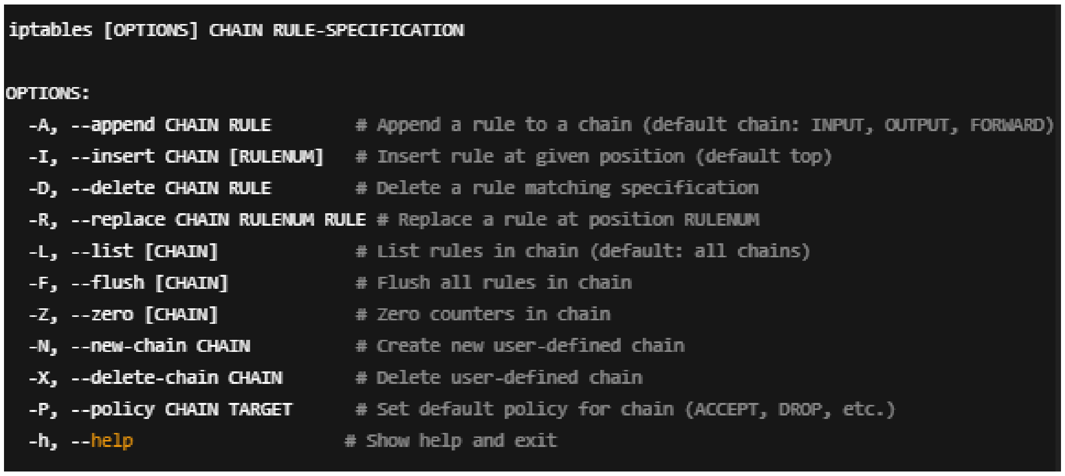
/etc/ufw/user.rules

- # 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.

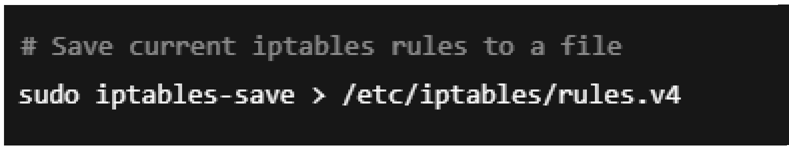
/etc/ufw/user6.rules

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

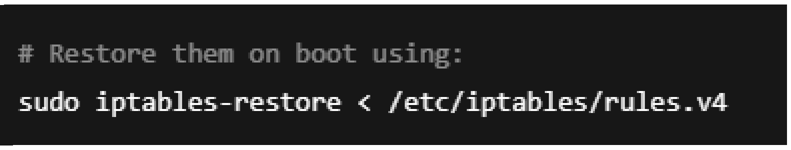
iptables**:** is a **firewall utility** that allows you to configure the Linux kernel’s **netfilter** firewall — letting you define rules for packet filtering, NAT, and mangling.



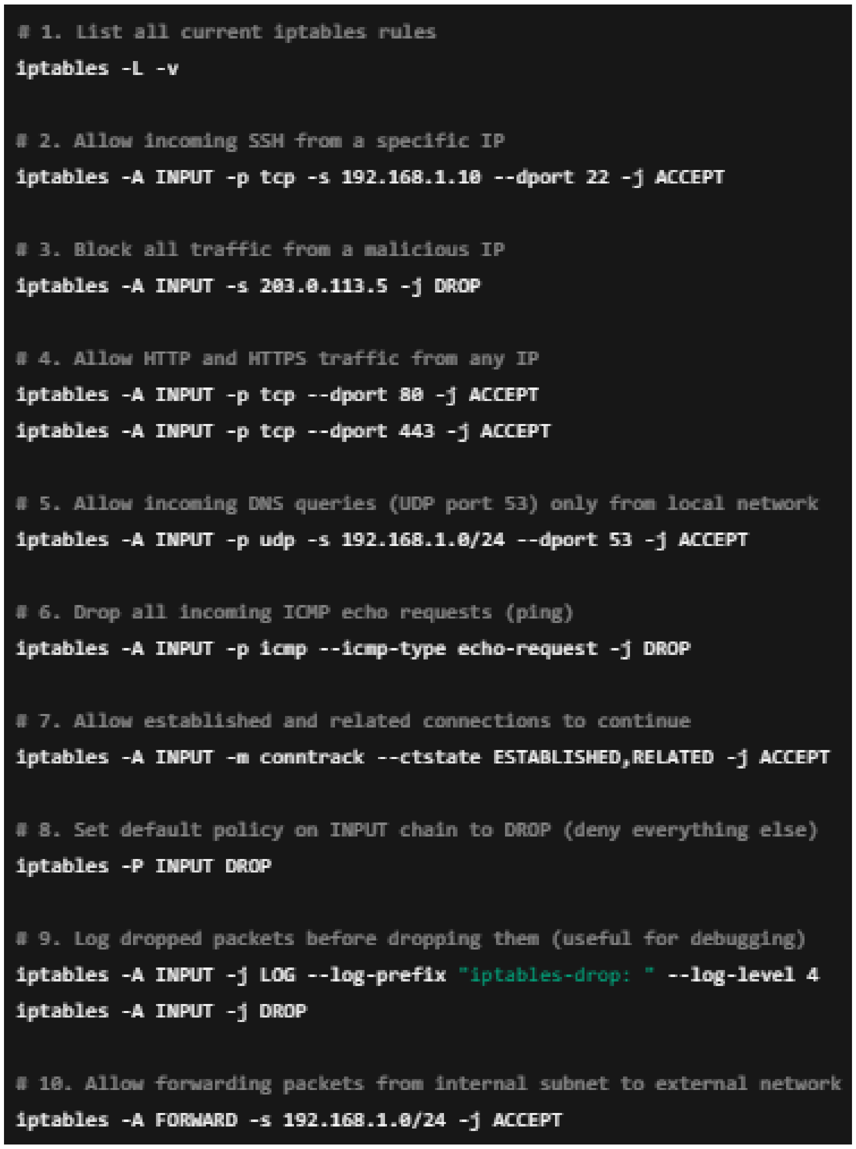
```
iptables [OPTIONS] CHAIN RULE-SPECIFICATION

OPTIONS:
  -A, --append CHAIN RULE       # Append a rule to a chain (default chain: INPUT, OUTPUT, FORWARD)
  -I, --insert CHAIN [RULENUM]  # Insert rule at given position (default top)
  -D, --delete CHAIN RULE       # Delete a rule matching specification
  -R, --replace CHAIN RULENUM RULE # Replace a rule at position RULENUM
  -L, --list [CHAIN]            # List rules in chain (default: all chains)
  -F, --flush [CHAIN]           # Flush all rules in chain
  -Z, --zero [CHAIN]            # Zero counters in chain
  -N, --new-chain CHAIN         # Create new user-defined chain
  -X, --delete-chain CHAIN      # Delete user-defined chain
  -P, --policy CHAIN TARGET     # Set default policy for chain (ACCEPT, DROP, etc.)
  -h, --help                   # Show help and exit
```



```
# Save current iptables rules to a file
sudo iptables-save > /etc/iptables/rules.v4
```



```
# Restore them on boot using:
sudo iptables-restore < /etc/iptables/rules.v4
```



```
# 1. List all current iptables rules
iptables -L -v

# 2. Allow incoming SSH from a specific IP
iptables -A INPUT -p tcp -s 192.168.1.10 --dport 22 -j ACCEPT

# 3. Block all traffic from a malicious IP
iptables -A INPUT -s 203.0.113.5 -j DROP

# 4. Allow HTTP and HTTPS traffic from any IP
iptables -A INPUT -p tcp --dport 80 -j ACCEPT
iptables -A INPUT -p tcp --dport 443 -j ACCEPT

# 5. Allow incoming DNS queries (UDP port 53) only from local network
iptables -A INPUT -p udp -s 192.168.1.0/24 --dport 53 -j ACCEPT

# 6. Drop all incoming ICMP echo requests (ping)
iptables -A INPUT -p icmp --icmp-type echo-request -j DROP

# 7. Allow established and related connections to continue
iptables -A INPUT -m conntrack --ctstate ESTABLISHED,RELATED -j ACCEPT

# 8. Set default policy on INPUT chain to DROP (deny everything else)
iptables -P INPUT DROP

# 9. Log dropped packets before dropping them (useful for debugging)
iptables -A INPUT -j LOG --log-prefix "iptables-drop: " --log-level 4
iptables -A INPUT -j DROP

# 10. Allow forwarding packets from internal subnet to external network
iptables -A FORWARD -s 192.168.1.0/24 -j ACCEPT
```

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.

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**

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

```
sudo mkdir -p /opt/chroot/jail/{bin,lib,lib64}
```

2. Copy a binary (like bash) into the jail

```
sudo cp /bin/bash /opt/chroot/jail/bin/
```

3. Copy required libraries (use ldd to find them)

```
ldd /bin/bash
```

```
linux-vdso.so.1 =>  (0x00007fff...)
libtinfo.so.6 => /lib64/libtinfo.so.6
libc.so.6 => /lib64/libc.so.6
```

Copy each of these into the appropriate jail subdirectory:

```
sudo cp /lib64/libtinfo.so.6 /opt/chroot/jail/lib64/
sudo cp /lib64/libc.so.6 /opt/chroot/jail/lib64/
```

4. Enter the chroot jail

```
sudo chroot /opt/chroot/jail /bin/bash
```

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:

```
# Check if AppArmor is enabled
sudo aa-status

# List all loaded profiles and their modes
sudo apparmor_status
```

Profile Location:

AppArmor profiles usually live in:

/etc/apparmor.d/

Example: A Simple Profile

```
#include <tunables/global>

profile /usr/bin/myapp flags=(complain) {
  # Allow reading /etc/myapp/
  /etc/myapp/** r,

  # Allow writing to /var/log/myapp.log
  /var/log/myapp.log w,

  # Allow using DNS
  network inet,

  # Deny everything else by default
  deny /** mrwklx,
}
```

Enable / Disable / Reload Profiles:

```
# Load a profile
sudo apparmor_parser -r /etc/apparmor.d/usr.bin.myapp

# Put profile into enforce mode
sudo aa-enforce /etc/apparmor.d/usr.bin.myapp

# Put profile into complain mode
sudo aa-complain /etc/apparmor.d/usr.bin.myapp

# Disable the profile
sudo ln -s /etc/apparmor.d/usr.bin.myapp /etc/apparmor.d/disable/
sudo apparmor_parser -R /etc/apparmor.d/usr.bin.myapp
```

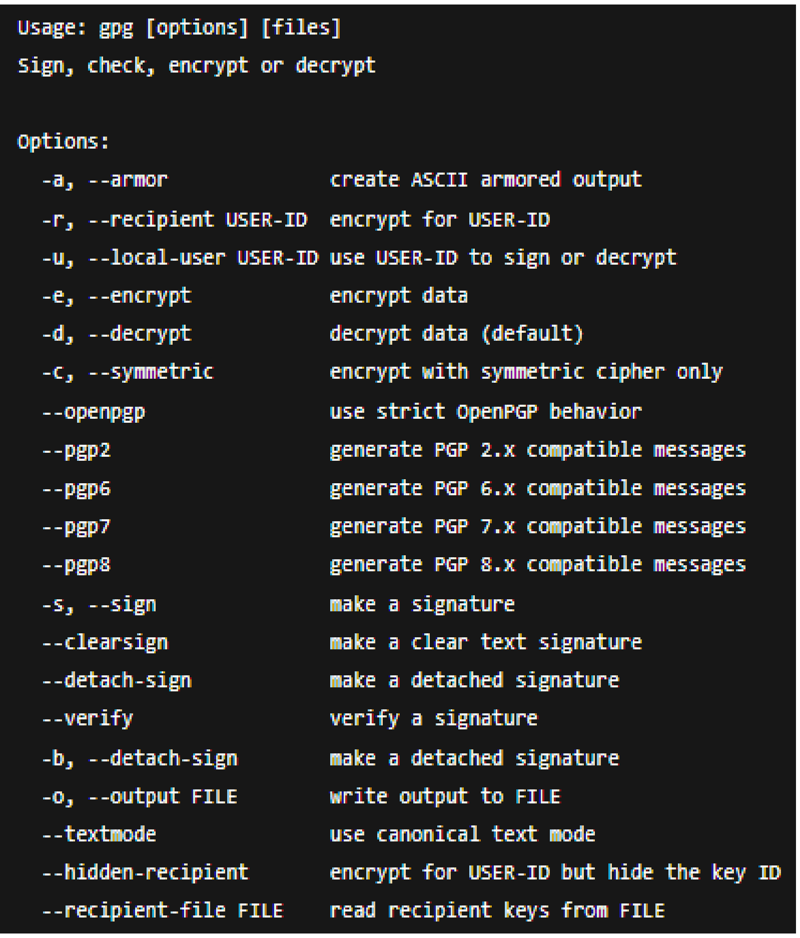
Tools to manage Apparmor:

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

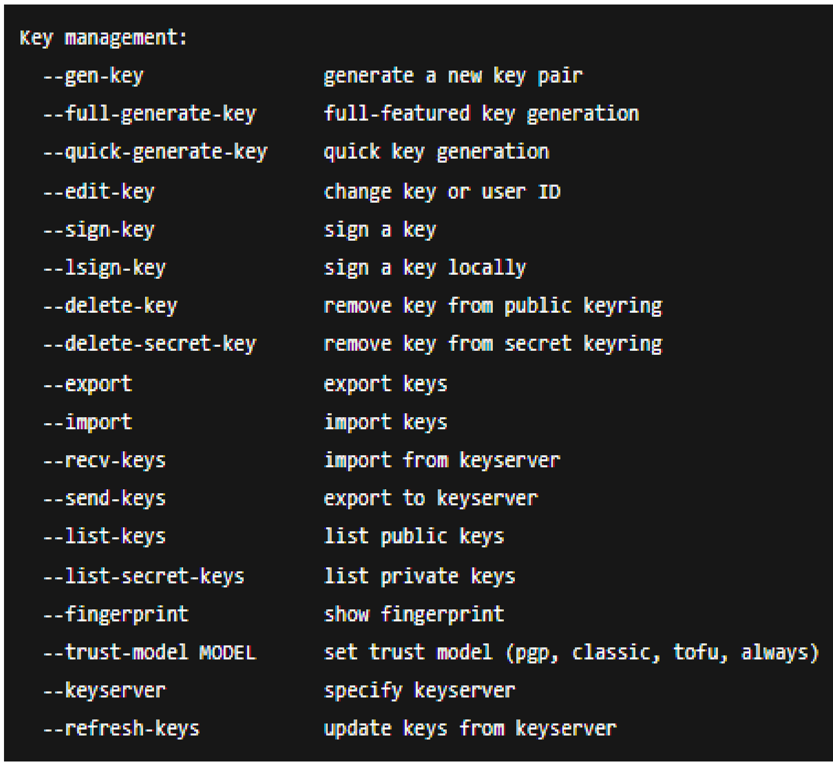
7.3- Data Encryption:

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

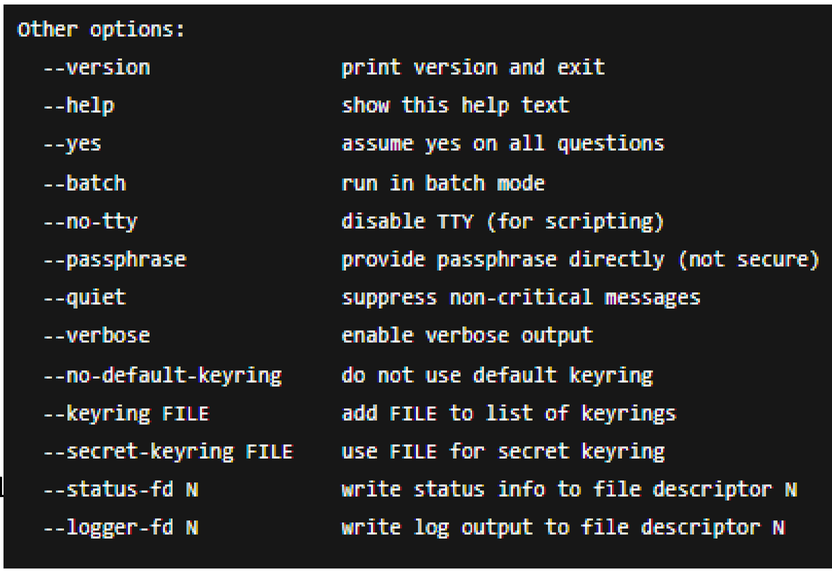
- Encrypt and decrypt data
- Create digital signatures to verify authenticity
- Manage keys for secure communication



```
Usage: gpg [options] [files]
Sign, check, encrypt or decrypt

Options:
  -a, --armor              create ASCII armored output
  -r, --recipient USER-ID  encrypt for USER-ID
  -u, --local-user USER-ID use USER-ID to sign or decrypt
  -e, --encrypt            encrypt data
  -d, --decrypt            decrypt data (default)
  -c, --symmetric          encrypt with symmetric cipher only
  --openpgp                use strict OpenPGP behavior
  --pgp2                   generate PGP 2.x compatible messages
  --pgp6                   generate PGP 6.x compatible messages
  --pgp7                   generate PGP 7.x compatible messages
  --pgp8                   generate PGP 8.x compatible messages
  -s, --sign               make a signature
  --clearsign              make a clear text signature
  --detach-sign            make a detached signature
  --verify                 verify a signature
  -b, --detach-sign        make a detached signature
  -o, --output FILE        write output to FILE
  --textmode               use canonical text mode
  --hidden-recipient       encrypt for USER-ID but hide the key ID
  --recipient-file FILE    read recipient keys from FILE
```

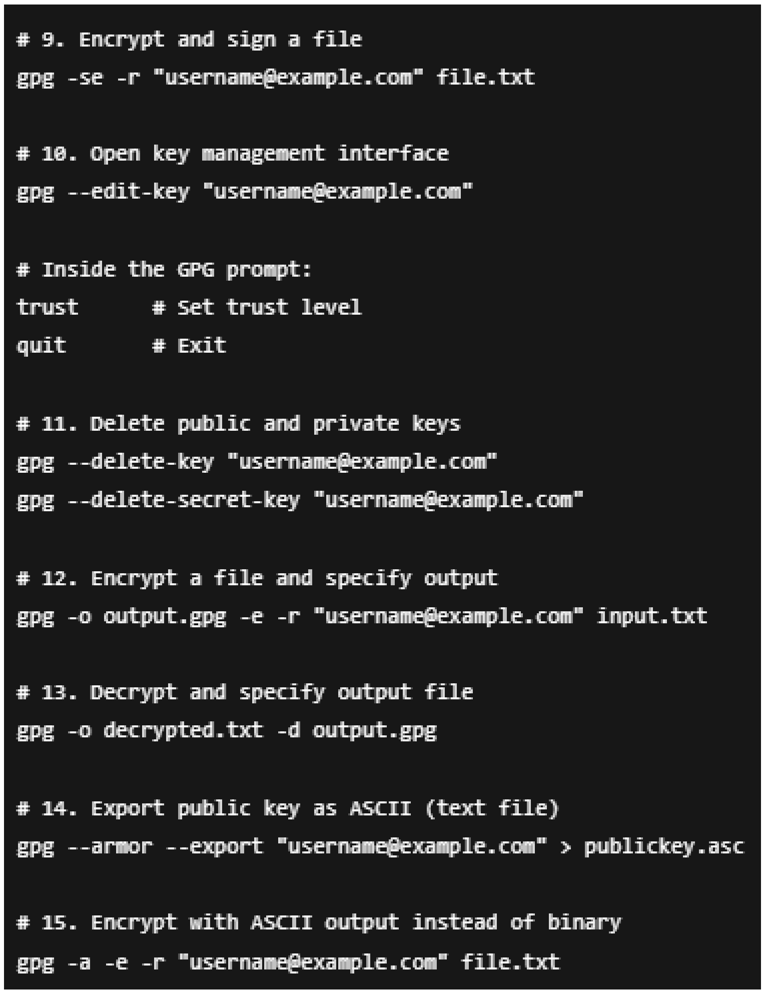


```
Key management:
  --gen-key               generate a new key pair
  --full-generate-key     full-featured key generation
  --quick-generate-key    quick key generation
  --edit-key              change key or user ID
  --sign-key              sign a key
  --lsign-key             sign a key locally
  --delete-key            remove key from public keyring
  --delete-secret-key     remove key from secret keyring
  --export                export keys
  --import                import keys
  --recv-keys             import from keyserver
  --send-keys             export to keyserver
  --list-keys             list public keys
  --list-secret-keys      list private keys
  --fingerprint           show fingerprint
  --trust-model MODEL     set trust model (pgp, classic, tofu, always)
  --keyserver             specify keyserver
  --refresh-keys          update keys from keyserver
```



```
Other options:
  --version                 print version and exit
  --help                    show this help text
  --yes                     assume yes on all questions
  --batch                   run in batch mode
  --no-tty                  disable TTY (for scripting)
  --passphrase              provide passphrase directly (not secure)
  --quiet                   suppress non-critical messages
  --verbose                 enable verbose output
  --no-default-keyring      do not use default keyring
  --keyring FILE            add FILE to list of keyrings
  --secret-keyring FILE     use FILE for secret keyring
  --status-fd N             write status info to file descriptor N
  --logger-fd N             write log output to file descriptor N
```

Examples



```
# 9. Encrypt and sign a file
gpg -se -r "username@example.com" file.txt

# 10. Open key management interface
gpg --edit-key "username@example.com"

# Inside the GPG prompt:
trust      # Set trust level
quit       # Exit

# 11. Delete public and private keys
gpg --delete-key "username@example.com"
gpg --delete-secret-key "username@example.com"

# 12. Encrypt a file and specify output
gpg -o output.gpg -e -r "username@example.com" input.txt

# 13. Decrypt and specify output file
gpg -o decrypted.txt -d output.gpg

# 14. Export public key as ASCII (text file)
gpg --armor --export "username@example.com" > publickey.asc

# 15. Encrypt with ASCII output instead of binary
gpg -a -e -r "username@example.com" file.txt
```

```
# 9. Encrypt and sign a file
gpg -se -r "username@example.com" file.txt

# 10. Open key management interface
gpg --edit-key "username@example.com"

# Inside the GPG prompt:
trust      # Set trust level
quit       # Exit

# 11. Delete public and private keys
gpg --delete-key "username@example.com"
gpg --delete-secret-key "username@example.com"

# 12. Encrypt a file and specify output
gpg -o output.gpg -e -r "username@example.com" input.txt

# 13. Decrypt and specify output file
gpg -o decrypted.txt -d output.gpg

# 14. Export public key as ASCII (text file)
gpg --armor --export "username@example.com" > publickey.asc

# 15. Encrypt with ASCII output instead of binary
gpg -a -e -r "username@example.com" file.txt
```

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?

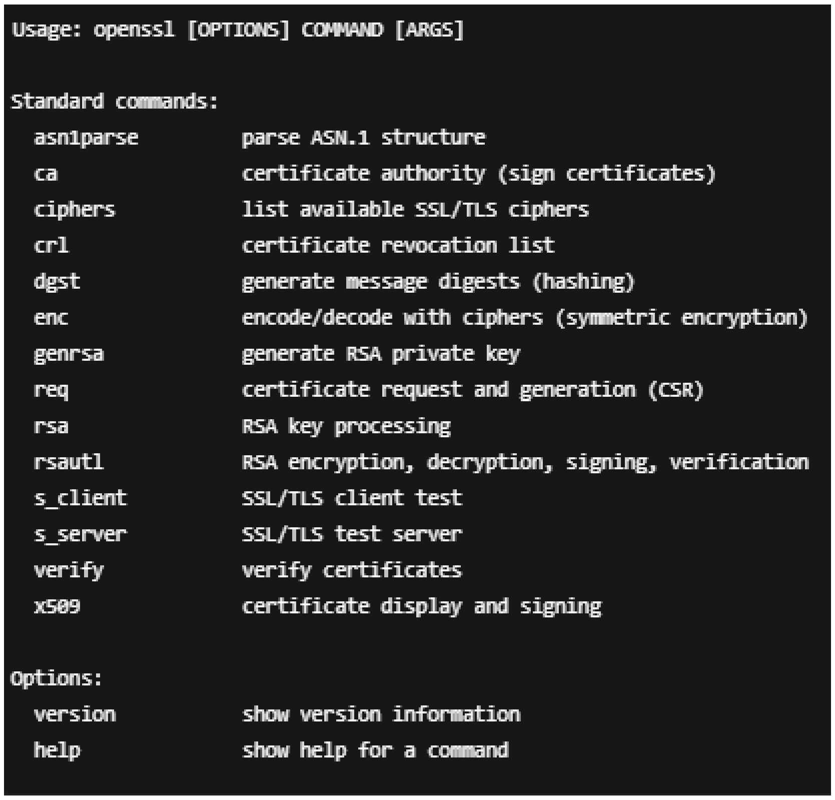
Create SSL/TLS certificates: Self-signed or with a CA

Encrypt/decrypt files or strings: Symmetric/asymmetric support

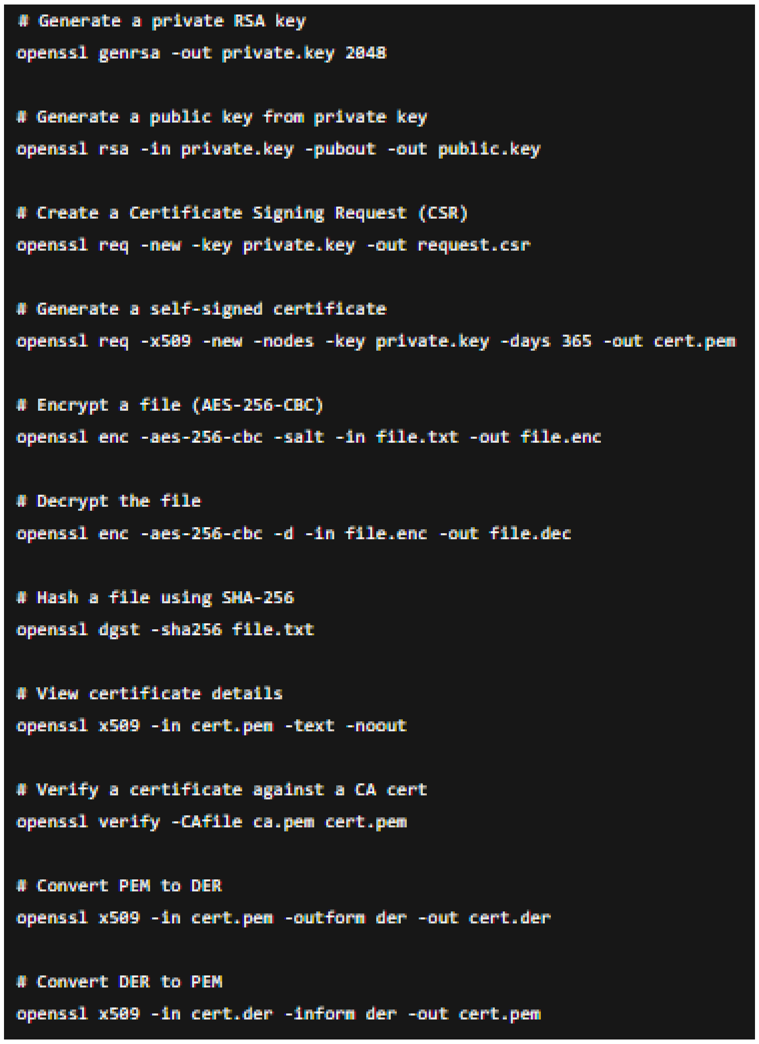
Hash data: MD5, SHA-1, SHA-256, etc.

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

Generate key pairs: RSA, EC, DSA

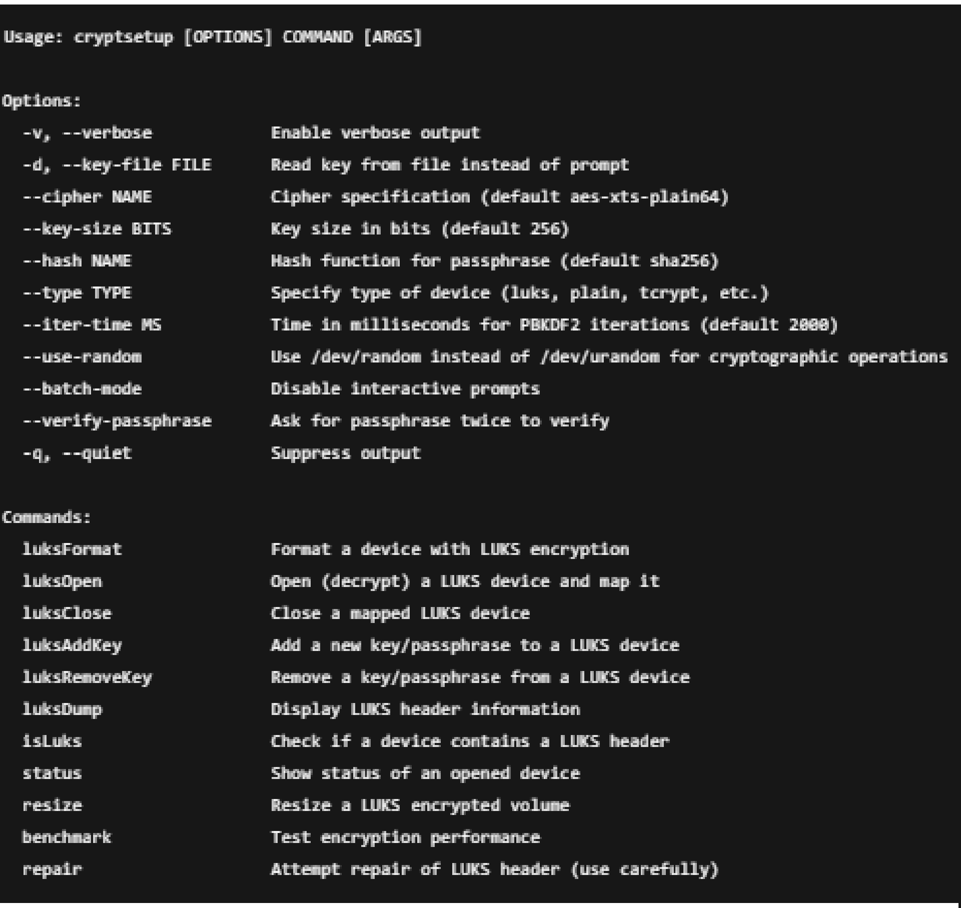


```
Usage: openssl [OPTIONS] COMMAND [ARGS]

Standard commands:
  asn1parse        parse ASN.1 structure
  ca               certificate authority (sign certificates)
  ciphers          list available SSL/TLS ciphers
  crl              certificate revocation list
  dgst             generate message digests (hashing)
  enc              encode/decode with ciphers (symmetric encryption)
  genrsa           generate RSA private key
  req              certificate request and generation (CSR)
  rsa              RSA key processing
  rsautl           RSA encryption, decryption, signing, verification
  s_client         SSL/TLS client test
  s_server         SSL/TLS test server
  verify           verify certificates
  x509             certificate display and signing

Options:
  version          show version information
  help             show help for a command
```

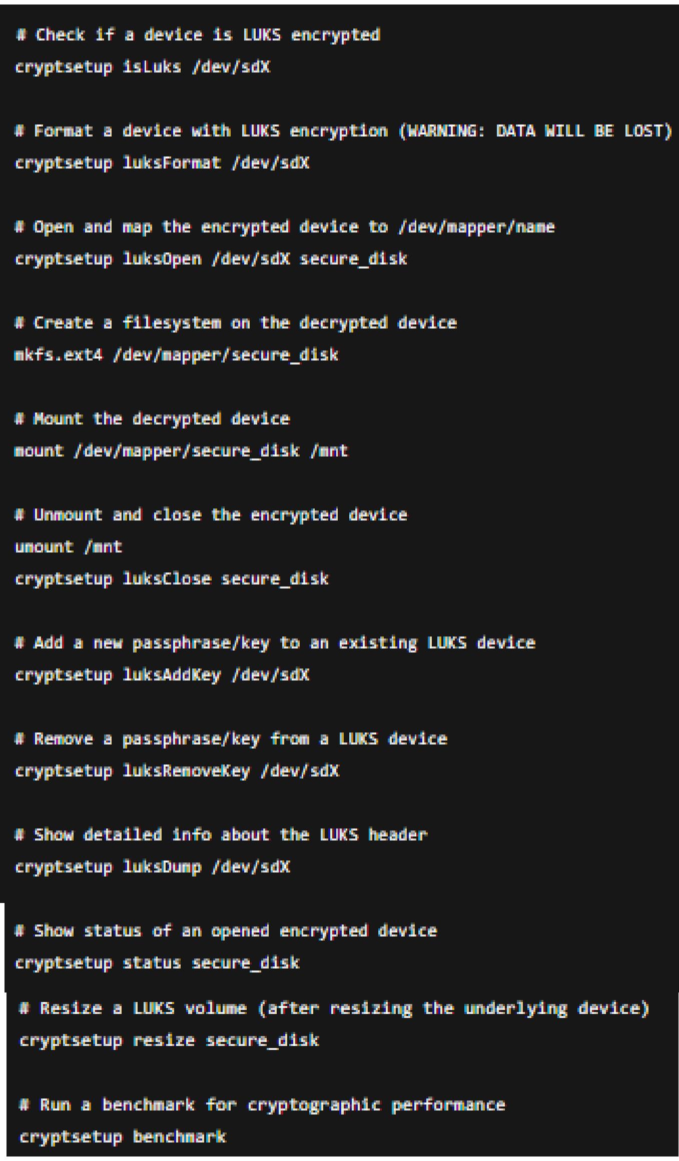


```
# Generate a private RSA key
openssl genrsa -out private.key 2048

# Generate a public key from private key
openssl rsa -in private.key -pubout -out public.key

# Create a Certificate Signing Request (CSR)
openssl req -new -key private.key -out request.csr

# Generate a self-signed certificate
openssl req -x509 -new -nodes -key private.key -days 365 -out cert.pem

# Encrypt a file (AES-256-CBC)
openssl enc -aes-256-cbc -salt -in file.txt -out file.enc

# Decrypt the file
openssl enc -aes-256-cbc -d -in file.enc -out file.dec

# Hash a file using SHA-256
openssl dgst -sha256 file.txt

# View certificate details
openssl x509 -in cert.pem -text -noout

# Verify a certificate against a CA cert
openssl verify -CAfile ca.pem cert.pem

# Convert PEM to DER
openssl x509 -in cert.pem -outform der -out cert.der

# Convert DER to PEM
openssl x509 -in cert.der -inform der -out cert.pem
```

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



```
Usage: cryptsetup [OPTIONS] COMMAND [ARGS]

Options:
  -v, --verbose           Enable verbose output
  -d, --key-file FILE     Read key from file instead of prompt
  --cipher NAME           Cipher specification (default aes-xts-plain64)
  --key-size BITS         Key size in bits (default 256)
  --hash NAME             Hash function for passphrase (default sha256)
  --type TYPE             Specify type of device (luks, plain, tcrypt, etc.)
  --iter-time MS          Time in milliseconds for PBKDF2 iterations (default 2000)
  --use-random            Use /dev/random instead of /dev/urandom for cryptographic operations
  --batch-mode            Disable interactive prompts
  --verify-passphrase     Ask for passphrase twice to verify
  -q, --quiet             Suppress output

Commands:
  luksFormat              Format a device with LUKS encryption
  luksOpen                Open (decrypt) a LUKS device and map it
  luksClose               Close a mapped LUKS device
  luksAddKey              Add a new key/passphrase to a LUKS device
  luksRemoveKey           Remove a key/passphrase from a LUKS device
  luksDump                Display LUKS header information
  isLuks                  Check if a device contains a LUKS header
  status                  Show status of an opened device
  resize                  Resize a LUKS encrypted volume
  benchmark               Test encryption performance
  repair                  Attempt repair of LUKS header (use carefully)
```

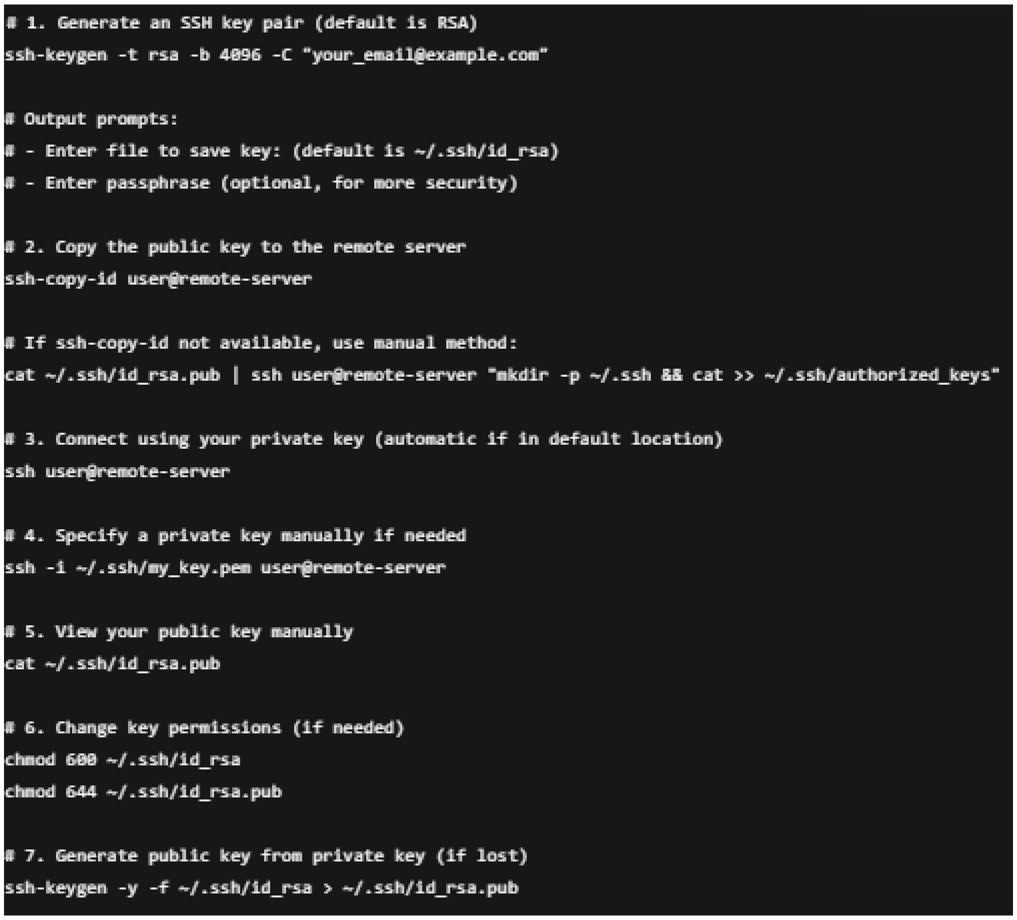


```
# Check if a device is LUKS encrypted
cryptsetup isLuks /dev/sdX

# Format a device with LUKS encryption (WARNING: DATA WILL BE LOST)
cryptsetup luksFormat /dev/sdX

# Open and map the encrypted device to /dev/mapper/name
cryptsetup luksOpen /dev/sdX secure_disk

# Create a filesystem on the decrypted device
mkfs.ext4 /dev/mapper/secure_disk

# Mount the decrypted device
mount /dev/mapper/secure_disk /mnt

# Unmount and close the encrypted device
umount /mnt
cryptsetup luksClose secure_disk

# Add a new passphrase/key to an existing LUKS device
cryptsetup luksAddKey /dev/sdX

# Remove a passphrase/key from a LUKS device
cryptsetup luksRemoveKey /dev/sdX

# Show detailed info about the LUKS header
cryptsetup luksDump /dev/sdX

# Show status of an opened encrypted device
cryptsetup status secure_disk
# Resize a LUKS volume (after resizing the underlying device)
cryptsetup resize secure_disk

# Run a benchmark for cryptographic performance
cryptsetup benchmark
```

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.

```
ssh-keygen [OPTIONS]

Options:
  -t type              Specify key type: rsa, dsa, ecdsa, ed25519
  -b bits              Key length (e.g. 2048 or 4096 for RSA)
  -f filename          Output file for private key
  -N passphrase        Provide new passphrase (empty for no passphrase)
  -C comment           Add comment to the public key
  -q                   Quiet mode (no output)
  -y                   Read private key file and print public key
```

Typical SSH Key Workflow:



```
# 1. Generate an SSH key pair (default is RSA)
ssh-keygen -t rsa -b 4096 -C "your_email@example.com"

# Output prompts:
# - Enter file to save key: (default is ~/.ssh/id_rsa)
# - Enter passphrase (optional, for more security)

# 2. Copy the public key to the remote server
ssh-copy-id user@remote-server

# If ssh-copy-id not available, use manual method:
cat ~/.ssh/id_rsa.pub | ssh user@remote-server "mkdir -p ~/.ssh && cat >> ~/.ssh/authorized_keys"

# 3. Connect using your private key (automatic if in default location)
ssh user@remote-server

# 4. Specify a private key manually if needed
ssh -i ~/.ssh/my_key.pem user@remote-server

# 5. View your public key manually
cat ~/.ssh/id_rsa.pub

# 6. Change key permissions (if needed)
chmod 600 ~/.ssh/id_rsa
chmod 644 ~/.ssh/id_rsa.pub

# 7. Generate public key from private key (if lost)
ssh-keygen -y -f ~/.ssh/id_rsa > ~/.ssh/id_rsa.pub
```

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: sftp [options] [user@]host

Options:
  -b file           Batch mode: run commands from file
  -C                Enable compression
  -i identity_file  Use this file for authentication (private key)
  -o option         SSH option (e.g. -oPort=2222)
  -P port           Connect to non-standard port
  -q                Quiet mode
  -R num_requests   Number of outstanding requests
  -S program        Specify SSH program (default: ssh)
  -v                Verbose/debug mode
```

Usage

```
# Connect to remote server
sftp user@hostname_or_ip

# After login, use these commands:
cd path             Change directory on remote server
lcd path            Change directory on local machine
ls                  List files on remote server
lls                 List files locally
get file            Download a file from remote to local
put file            Upload a file from local to remote
mget *.txt          Download multiple files (wildcards allowed)
mput *.log          Upload multiple files
rename old new      Rename a file remotely
rm file             Delete file on remote
mkdir dirname       Create a directory on remote
rmdir dirname       Remove remote directory
pwd                 Print remote working directory
lpwd                Print local working directory
exit                Exit sftp shell
bye                 Same as exit
help                Show list of sftp commands
```

examples

```
# Connect to a remote server
sftp user@192.168.1.10

# Upload a file
put myfile.txt

# Download a file
get notes.pdf

# Change to a specific directory before upload
cd /var/www/html
put index.html

# Use a private key to connect
sftp -i ~/.ssh/id_rsa user@192.168.1.10

# Specify a different SSH port
sftp -P 2222 user@192.168.1.10

# Run multiple commands from a file
sftp -b batch.txt user@host
```

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.

```
Usage: scp [options] [[user@]host1:]file1 [[user@]host2:]file2

Options:
  -r              Copy directories recursively
  -P port         Specify SSH port
  -i identity     Use identity (private key) for SSH
  -C              Enable compression
  -v              Verbose output (debugging)
  -q              Quiet mode
  -l limit        Limit bandwidth in Kbit/s
  -o option       Pass SSH option (e.g. -o StrictHostKeyChecking=no)
```

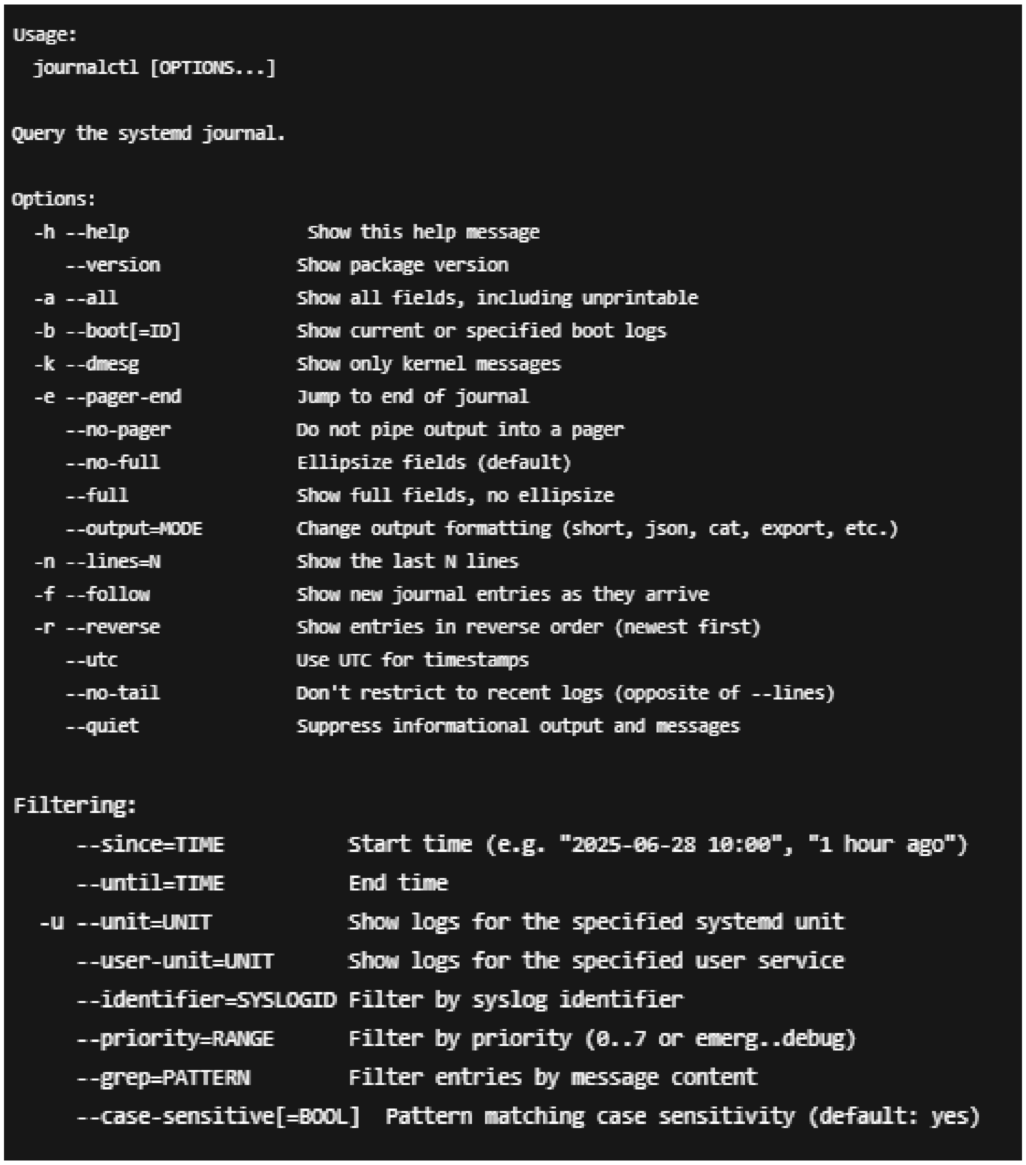
```
# 1. Copy a local file to a remote server
scp file.txt user@192.168.1.10:/home/user/

# 2. Copy a file from remote to local
scp user@192.168.1.10:/var/log/syslog ./syslog_backup

# 3. Copy a directory recursively to remote
scp -r myfolder/ user@192.168.1.10:/home/user/backup/

# 4. Copy using a specific SSH port (e.g., 2222)
scp -P 2222 file.txt user@192.168.1.10:/tmp/

# 5. Copy using a specific private key
scp -i ~/.ssh/id_rsa secret.txt user@192.168.1.10:/srv/secure/

# 6. Pull multiple files from remote
scp user@192.168.1.10:"/etc/*.conf" .

# 7. Copy a remote file from one server to another (via local machine)
scp user1@host1:/path/file user2@host2:/path/   # not peer-to-peer; routed through your local
```

8- Trouble shooting and system maintenance

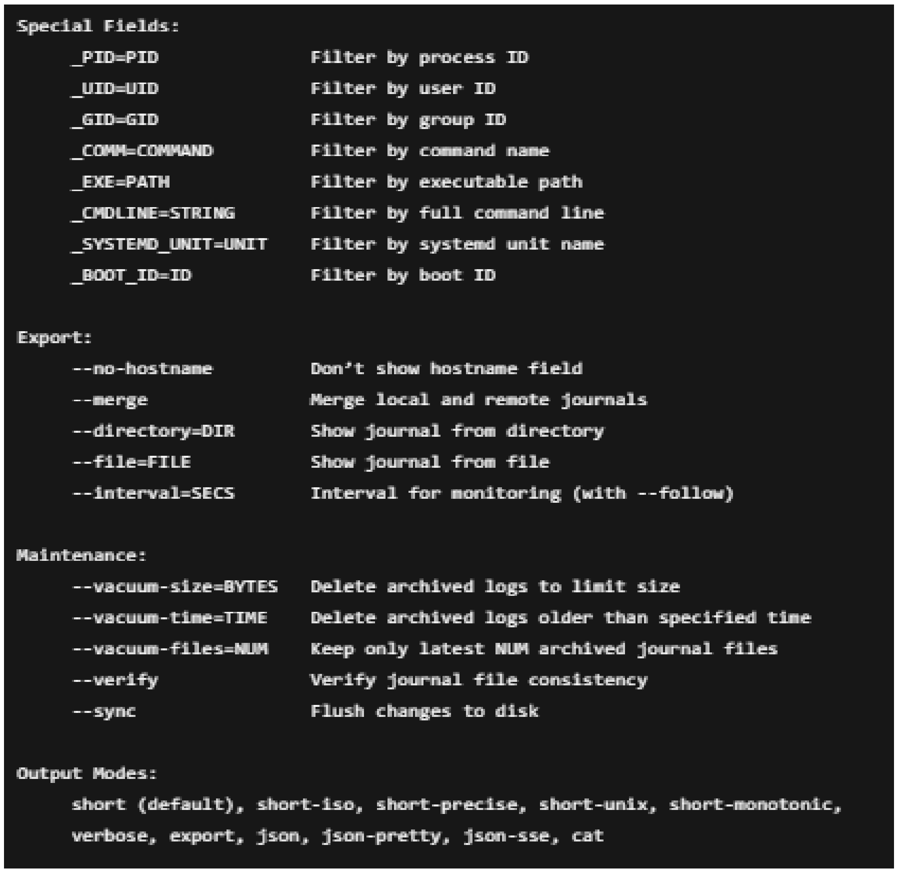
8.1- Analize and Interpreting logs files:

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

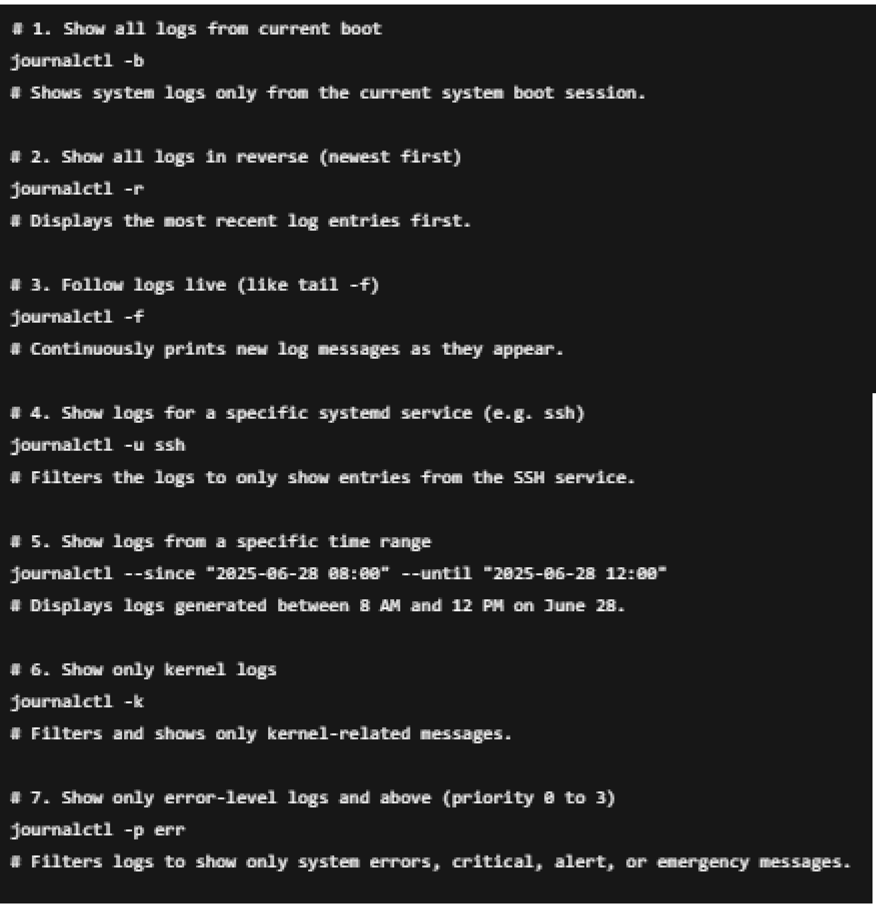
It reads the binary journal logs that contain detailed system events: services, kernel, user sessions, boots, shutdowns, etc.



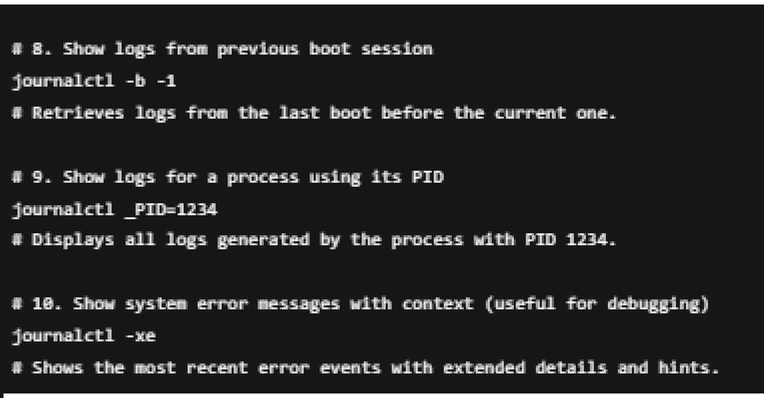
```
Usage:
  journalctl [OPTIONS...]

Query the systemd journal.

Options:
  -h --help                 Show this help message
     --version             Show package version
  -a --all                 Show all fields, including unprintable
  -b --boot[=ID]           Show current or specified boot logs
  -k --dmesg               Show only kernel messages
  -e --pager-end           Jump to end of journal
     --no-pager            Do not pipe output into a pager
     --no-full             Ellipsize fields (default)
     --full                Show full fields, no ellipsize
     --output=MODE         Change output formatting (short, json, cat, export, etc.)
  -n --lines=N             Show the last N lines
  -f --follow              Show new journal entries as they arrive
  -r --reverse             Show entries in reverse order (newest first)
     --utc                 Use UTC for timestamps
     --no-tail             Don't restrict to recent logs (opposite of --lines)
     --quiet               Suppress informational output and messages

Filtering:
     --since=TIME          Start time (e.g. "2025-06-28 10:00", "1 hour ago")
     --until=TIME          End time
  -u --unit=UNIT           Show logs for the specified systemd unit
     --user-unit=UNIT      Show logs for the specified user service
     --identifier=SYSLOGID Filter by syslog identifier
     --priority=RANGE      Filter by priority (0..7 or emerg..debug)
     --grep=PATTERN        Filter entries by message content
     --case-sensitive[=BOOL]  Pattern matching case sensitivity (default: yes)
```



```
Special Fields:
     _PID=PID              Filter by process ID
     _UID=UID              Filter by user ID
     _GID=GID              Filter by group ID
     _COMM=COMMAND         Filter by command name
     _EXE=PATH             Filter by executable path
     _CMDLINE=STRING       Filter by full command line
     _SYSTEMD_UNIT=UNIT    Filter by systemd unit name
     _BOOT_ID=ID           Filter by boot ID

Export:
     --no-hostname         Don't show hostname field
     --merge               Merge local and remote journals
     --directory=DIR       Show journal from directory
     --file=FILE           Show journal from file
     --interval=SECS       Interval for monitoring (with --follow)

Maintenance:
     --vacuum-size=BYTES   Delete archived logs to limit size
     --vacuum-time=TIME    Delete archived logs older than specified time
     --vacuum-files=NUM    Keep only latest NUM archived journal files
     --verify              Verify journal file consistency
     --sync                Flush changes to disk

Output Modes:
     short (default), short-iso, short-precise, short-unix, short-monotonic,
     verbose, export, json, json-pretty, json-sse, cat
```

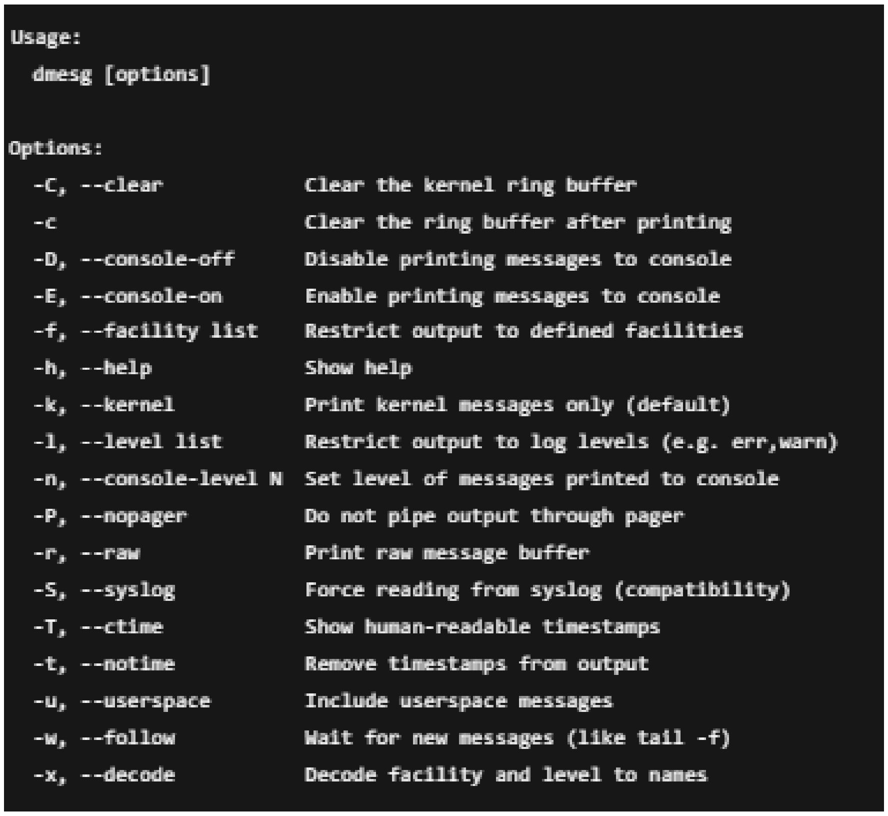


```
# 1. Show all logs from current boot
journalctl -b
# Shows system logs only from the current system boot session.

# 2. Show all logs in reverse (newest first)
journalctl -r
# Displays the most recent log entries first.

# 3. Follow logs live (like tail -f)
journalctl -f
# Continuously prints new log messages as they appear.

# 4. Show logs for a specific systemd service (e.g. ssh)
journalctl -u ssh
# Filters the logs to only show entries from the SSH service.

# 5. Show logs from a specific time range
journalctl --since "2025-06-28 08:00" --until "2025-06-28 12:00"
# Displays logs generated between 8 AM and 12 PM on June 28.

# 6. Show only kernel logs
journalctl -k
# Filters and shows only kernel-related messages.

# 7. Show only error-level logs and above (priority 0 to 3)
journalctl -p err
# Filters logs to show only system errors, critical, alert, or emergency messages.
```

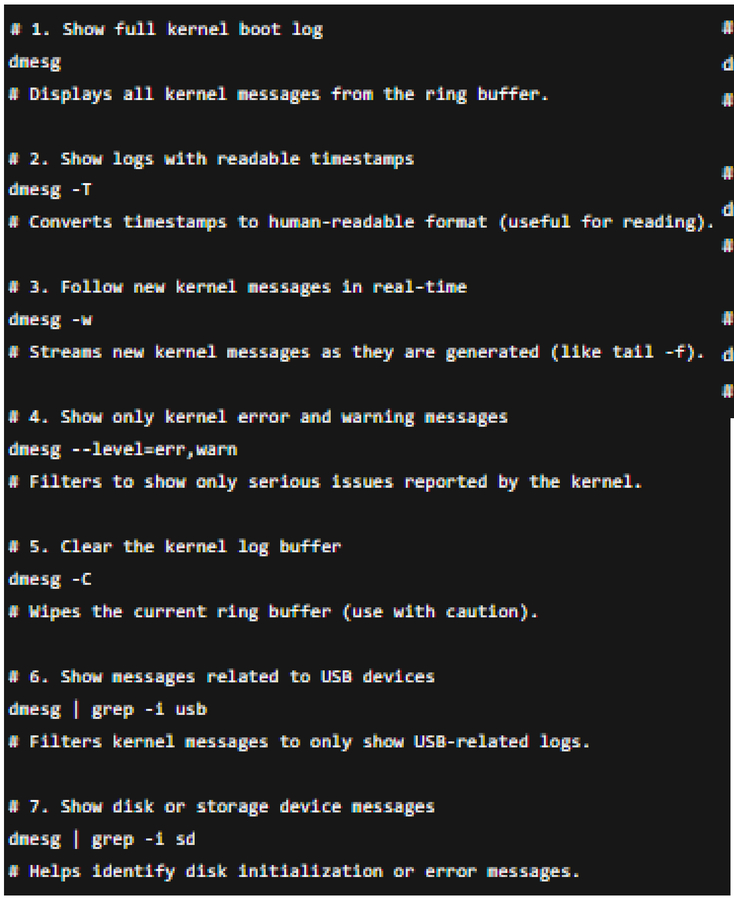


```
# 8. Show logs from previous boot session
journalctl -b -1
# Retrieves logs from the last boot before the current one.

# 9. Show logs for a process using its PID
journalctl _PID=1234
# Displays all logs generated by the process with PID 1234.

# 10. Show system error messages with context (useful for debugging)
journalctl -xe
# Shows the most recent error events with extended details and hints.
```

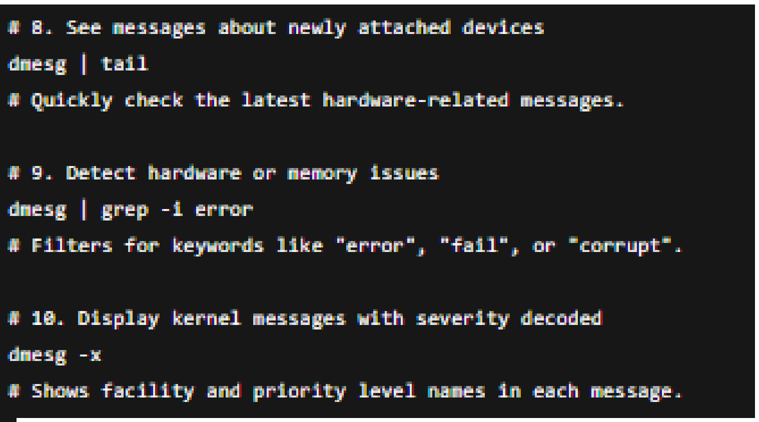
- **dmesg**: stands for "diagnostic message" — it displays kernel ring buffer messages, which are logs generated by the Linux kernel, especially during boot time and hardware events.



```
Usage:
  dmesg [options]

Options:
  -C, --clear            Clear the kernel ring buffer
  -c                     Clear the ring buffer after printing
  -D, --console-off      Disable printing messages to console
  -E, --console-on       Enable printing messages to console
  -f, --facility list    Restrict output to defined facilities
  -h, --help             Show help
  -k, --kernel           Print kernel messages only (default)
  -l, --level list       Restrict output to log levels (e.g. err,warn)
  -n, --console-level N  Set level of messages printed to console
  -P, --nopager          Do not pipe output through pager
  -r, --raw              Print raw message buffer
  -S, --syslog           Force reading from syslog (compatibility)
  -T, --ctime            Show human-readable timestamps
  -t, --notime           Remove timestamps from output
  -u, --userspace        Include userspace messages
  -w, --follow           Wait for new messages (like tail -f)
  -x, --decode           Decode facility and level to names
```



```
# 1. Show full kernel boot log
dmesg
# Displays all kernel messages from the ring buffer.

# 2. Show logs with readable timestamps
dmesg -T
# Converts timestamps to human-readable format (useful for reading).

# 3. Follow new kernel messages in real-time
dmesg -w
# Streams new kernel messages as they are generated (like tail -f).

# 4. Show only kernel error and warning messages
dmesg --level=err,warn
# Filters to show only serious issues reported by the kernel.

# 5. Clear the kernel log buffer
dmesg -C
# Wipes the current ring buffer (use with caution).

# 6. Show messages related to USB devices
dmesg | grep -i usb
# Filters kernel messages to only show USB-related logs.

# 7. Show disk or storage device messages
dmesg | grep -i sd
# Helps identify disk initialization or error messages.
```



```
# 8. See messages about newly attached devices
dmesg | tail
# Quickly check the latest hardware-related messages.

# 9. Detect hardware or memory issues
dmesg | grep -i error
# Filters for keywords like "error", "fail", or "corrupt".

# 10. Display kernel messages with severity decoded
dmesg -x
# Shows facility and priority level names in each message.
```

dmesg / kernel log levels:

```
Level   Name       Meaning                             Use Case
0       emerg      Emergency – system is unusable      Kernel panic, total failure
1       alert      Action must be taken immediately    Critical subsystem failure
2       crit       Critical conditions                 Hardware error, filesystem crash
3       err        Error conditions                    Driver failure, device I/O error
4       warning    Warning conditions                  Recoverable issue, unstable state
5       notice     Normal but significant condition    Startup notices, important info
6       info       Informational messages              General boot/init/device info
7       debug      Debug-level messages                Developer info, verbose output
```

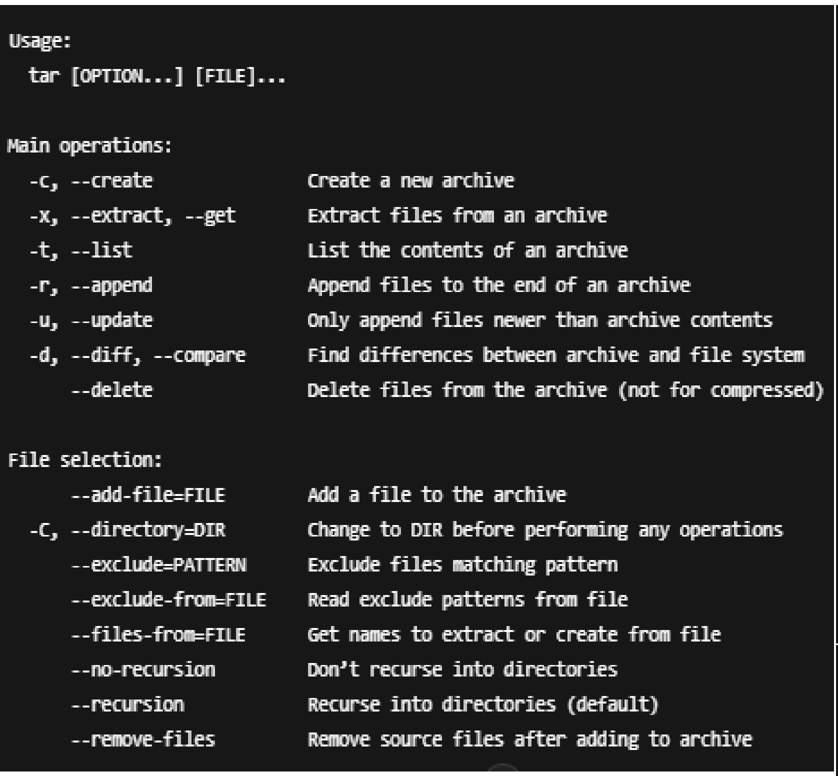
How to Filter by Log Level with dmesg:

```
dmesg --level=err           # Only show error-level messages (level 3)
dmesg --level=warn,err      # Show both warnings and errors (levels 3–4)
dmesg --level=debug         # Only debug messages
```

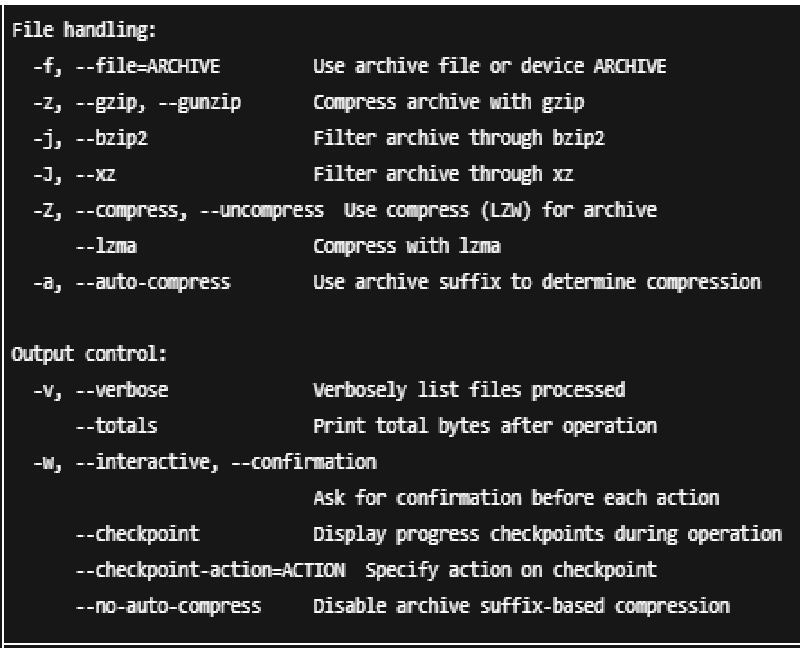
8.2- Backup and File Compression:

1. File Compression (Archiving & Compressing Files)

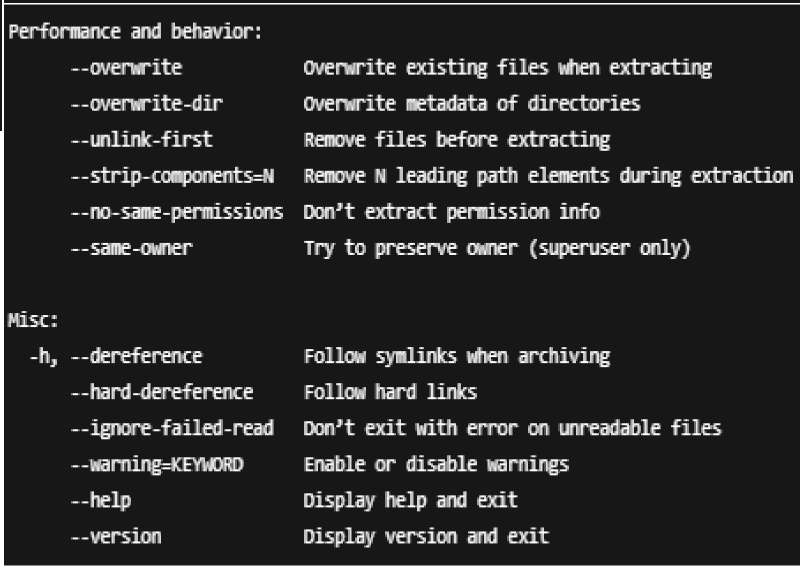
- **tar:** It collects multiple files and directories into a single archive file (called a "tarball") for easy storage, transfer, or backup.



```
Usage:
  tar [OPTION...] [FILE]...

Main operations:
  -c, --create              Create a new archive
  -x, --extract, --get      Extract files from an archive
  -t, --list                List the contents of an archive
  -r, --append              Append files to the end of an archive
  -u, --update              Only append files newer than archive contents
  -d, --diff, --compare     Find differences between archive and file system
      --delete              Delete files from the archive (not for compressed)

File selection:
      --add-file=FILE       Add a file to the archive
  -C, --directory=DIR       Change to DIR before performing any operations
      --exclude=PATTERN     Exclude files matching pattern
      --exclude-from=FILE   Read exclude patterns from file
      --files-from=FILE     Get names to extract or create from file
      --no-recursion        Don't recurse into directories
      --recursion           Recurse into directories (default)
      --remove-files        Remove source files after adding to archive
```



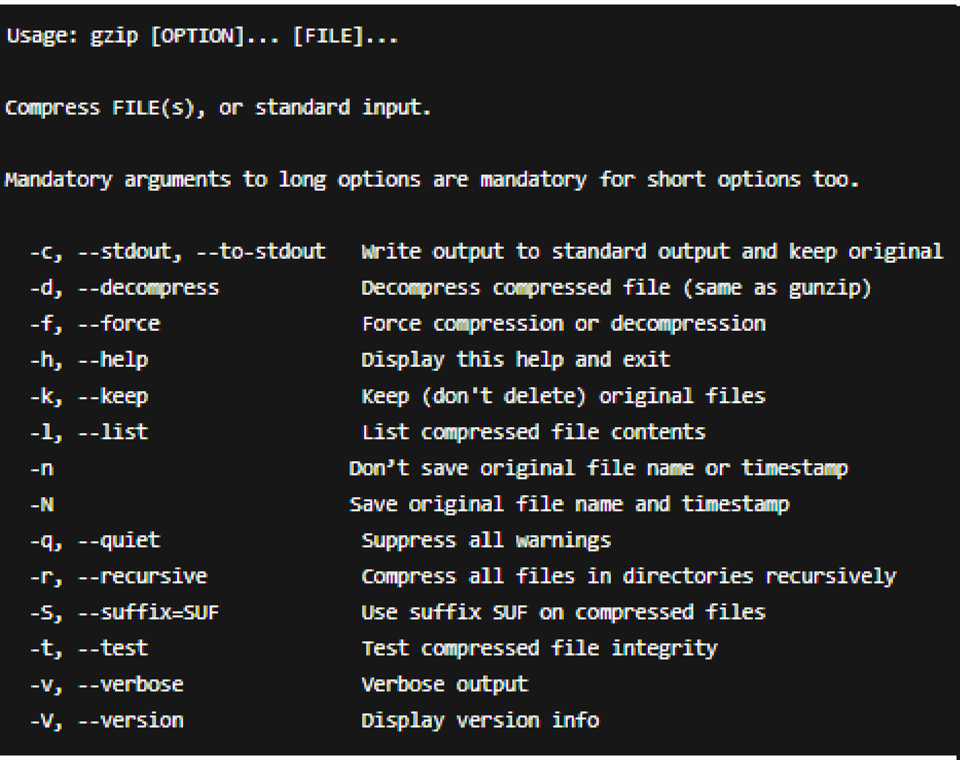
```
File handling:
  -f, --file=ARCHIVE          Use archive file or device ARCHIVE
  -z, --gzip, --gunzip        Compress archive with gzip
  -j, --bzip2                 Filter archive through bzip2
  -J, --xz                    Filter archive through xz
  -Z, --compress, --uncompress  Use compress (LZW) for archive
      --lzma                  Compress with lzma
  -a, --auto-compress         Use archive suffix to determine compression

Output control:
  -v, --verbose               Verbosely list files processed
      --totals                Print total bytes after operation
  -w, --interactive, --confirmation
                              Ask for confirmation before each action
      --checkpoint            Display progress checkpoints during operation
      --checkpoint-action=ACTION  Specify action on checkpoint
      --no-auto-compress      Disable archive suffix-based compression
```



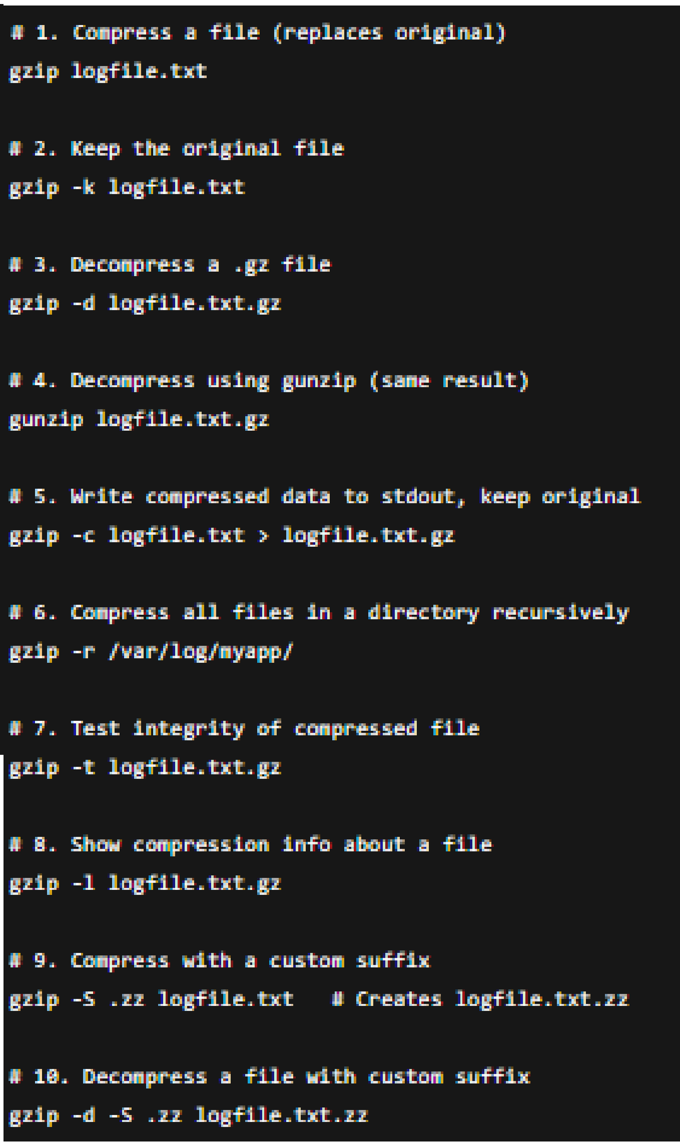
```
Performance and behavior:
      --overwrite             Overwrite existing files when extracting
      --overwrite-dir         Overwrite metadata of directories
      --unlink-first          Remove files before extracting
      --strip-components=N    Remove N leading path elements during extraction
      --no-same-permissions   Don't extract permission info
      --same-owner            Try to preserve owner (superuser only)

Misc:
  -h, --dereference           Follow symlinks when archiving
      --hard-dereference      Follow hard links
      --ignore-failed-read    Don't exit with error on unreadable files
      --warning=KEYWORD       Enable or disable warnings
      --help                  Display help and exit
      --version               Display version and exit
```

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



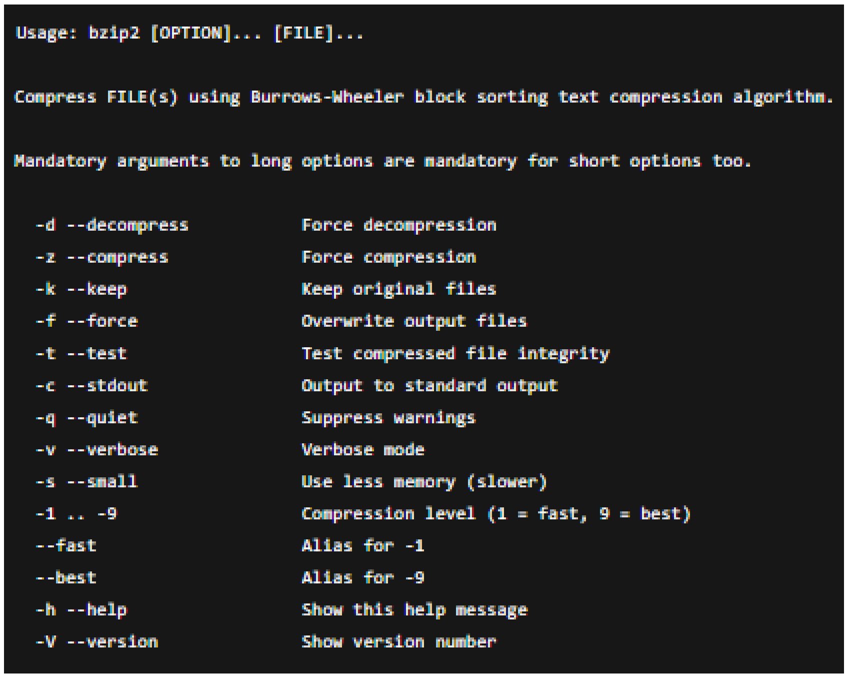
```
Usage: gzip [OPTION]... [FILE]...

Compress FILE(s), or standard input.

Mandatory arguments to long options are mandatory for short options too.

  -c, --stdout, --to-stdout   Write output to standard output and keep original
  -d, --decompress            Decompress compressed file (same as gunzip)
  -f, --force                 Force compression or decompression
  -h, --help                  Display this help and exit
  -k, --keep                  Keep (don't delete) original files
  -l, --list                  List compressed file contents
  -n                         Don't save original file name or timestamp
  -N                         Save original file name and timestamp
  -q, --quiet                 Suppress all warnings
  -r, --recursive             Compress all files in directories recursively
  -S, --suffix=SUF            Use suffix SUF on compressed files
  -t, --test                  Test compressed file integrity
  -v, --verbose               Verbose output
  -V, --version               Display version info
```



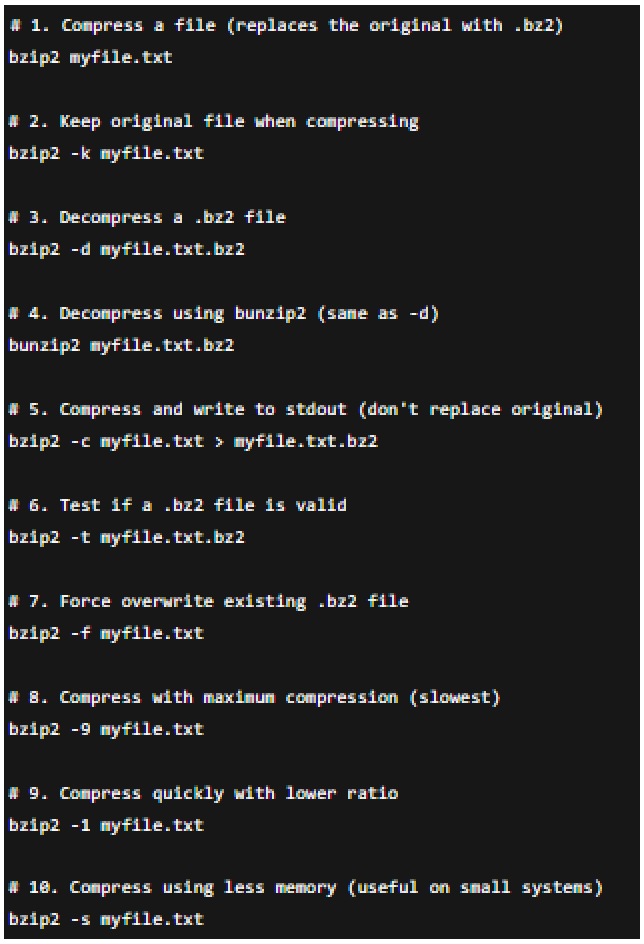
```
# 1. Compress a file (replaces original)
gzip logfile.txt

# 2. Keep the original file
gzip -k logfile.txt

# 3. Decompress a .gz file
gzip -d logfile.txt.gz

# 4. Decompress using gunzip (same result)
gunzip logfile.txt.gz

# 5. Write compressed data to stdout, keep original
gzip -c logfile.txt > logfile.txt.gz

# 6. Compress all files in a directory recursively
gzip -r /var/log/myapp/

# 7. Test integrity of compressed file
gzip -t logfile.txt.gz

# 8. Show compression info about a file
gzip -l logfile.txt.gz

# 9. Compress with a custom suffix
gzip -S .zz logfile.txt   # Creates logfile.txt.zz

# 10. Decompress a file with custom suffix
gzip -d -S .zz logfile.txt.zz
```

- **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.



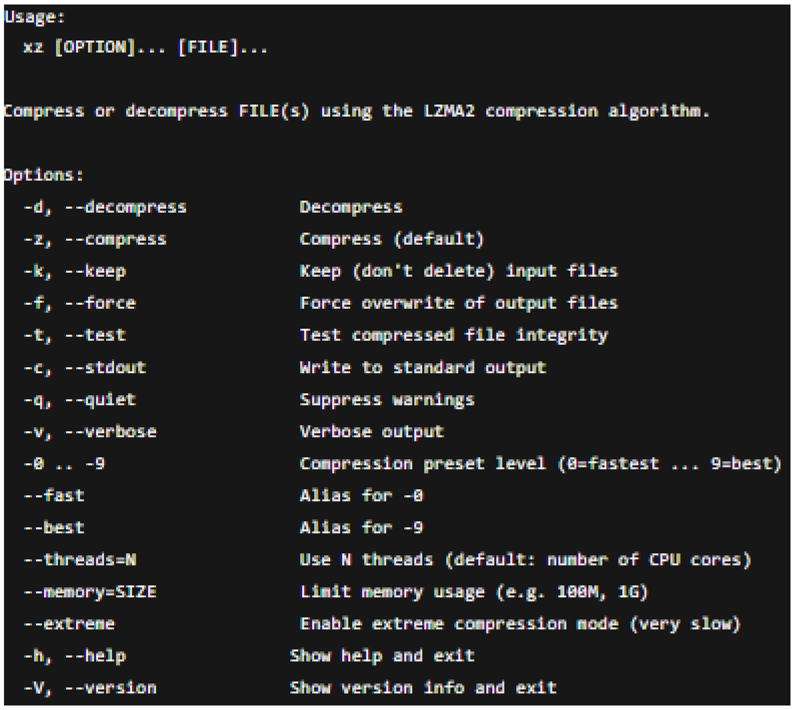
```
Usage: bzip2 [OPTION]... [FILE]...

Compress FILE(s) using Burrows-Wheeler block sorting text compression algorithm.

Mandatory arguments to long options are mandatory for short options too.

  -d --decompress          Force decompression
  -z --compress            Force compression
  -k --keep                Keep original files
  -f --force               Overwrite output files
  -t --test                Test compressed file integrity
  -c --stdout              Output to standard output
  -q --quiet               Suppress warnings
  -v --verbose             Verbose mode
  -s --small               Use less memory (slower)
  -1 .. -9                 Compression level (1 = fast, 9 = best)
  --fast                   Alias for -1
  --best                   Alias for -9
  -h --help                Show this help message
  -V --version             Show version number
```



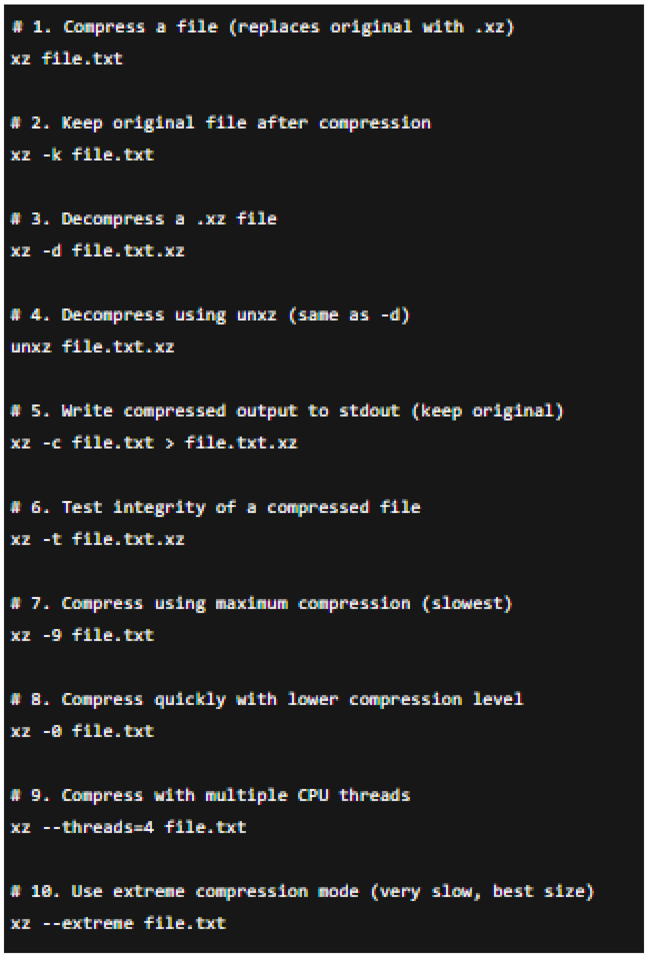
```
# 1. Compress a file (replaces the original with .bz2)
bzip2 myfile.txt

# 2. Keep original file when compressing
bzip2 -k myfile.txt

# 3. Decompress a .bz2 file
bzip2 -d myfile.txt.bz2

# 4. Decompress using bunzip2 (same as -d)
bunzip2 myfile.txt.bz2

# 5. Compress and write to stdout (don't replace original)
bzip2 -c myfile.txt > myfile.txt.bz2

# 6. Test if a .bz2 file is valid
bzip2 -t myfile.txt.bz2

# 7. Force overwrite existing .bz2 file
bzip2 -f myfile.txt

# 8. Compress with maximum compression (slowest)
bzip2 -9 myfile.txt

# 9. Compress quickly with lower ratio
bzip2 -1 myfile.txt

# 10. Compress using less memory (useful on small systems)
bzip2 -s myfile.txt
```

- **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.

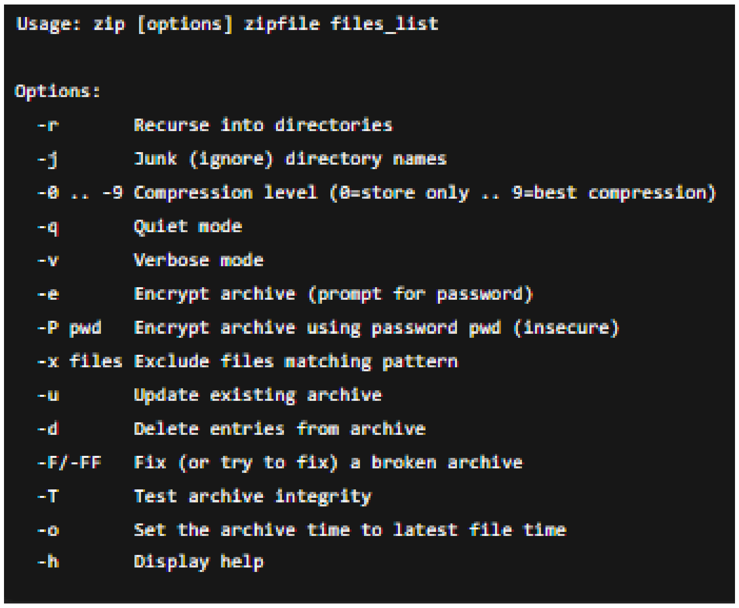


```
Usage:
  xz [OPTION]... [FILE]...

Compress or decompress FILE(s) using the LZMA2 compression algorithm.

Options:
  -d, --decompress           Decompress
  -z, --compress             Compress (default)
  -k, --keep                 Keep (don't delete) input files
  -f, --force                Force overwrite of output files
  -t, --test                 Test compressed file integrity
  -c, --stdout               Write to standard output
  -q, --quiet                Suppress warnings
  -v, --verbose              Verbose output
  -0 .. -9                   Compression preset level (0=fastest ... 9=best)
  --fast                     Alias for -0
  --best                     Alias for -9
  --threads=N                Use N threads (default: number of CPU cores)
  --memory=SIZE              Limit memory usage (e.g. 100M, 1G)
  --extreme                  Enable extreme compression mode (very slow)
  -h, --help                Show help and exit
  -V, --version             Show version info and exit
```

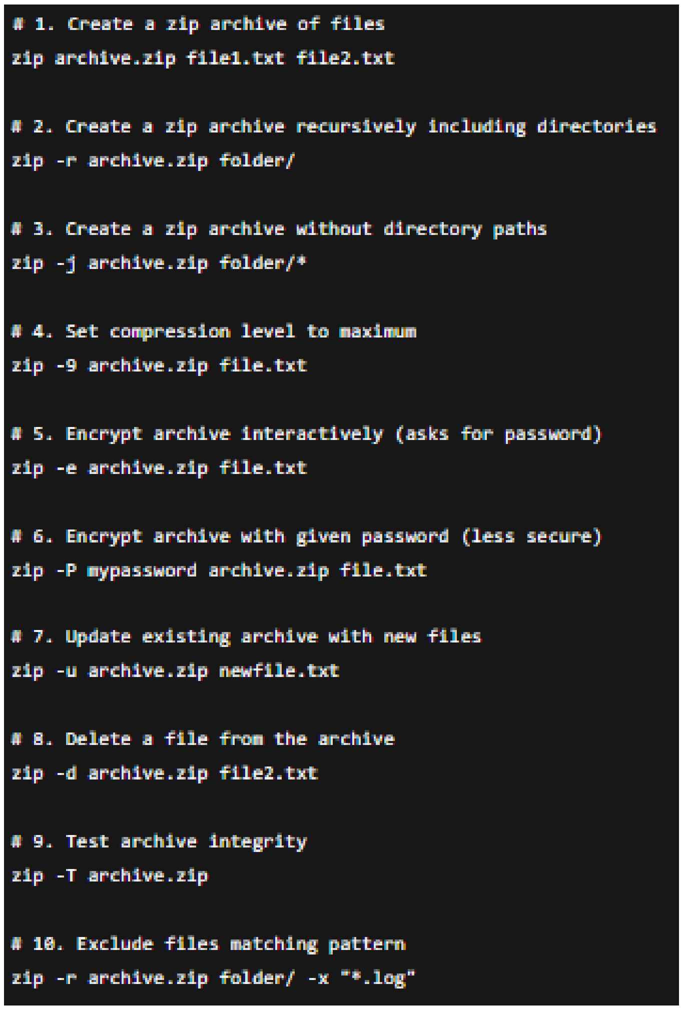


```
# 1. Compress a file (replaces original with .xz)
xz file.txt

# 2. Keep original file after compression
xz -k file.txt

# 3. Decompress a .xz file
xz -d file.txt.xz

# 4. Decompress using unxz (same as -d)
unxz file.txt.xz

# 5. Write compressed output to stdout (keep original)
xz -c file.txt > file.txt.xz

# 6. Test integrity of a compressed file
xz -t file.txt.xz

# 7. Compress using maximum compression (slowest)
xz -9 file.txt

# 8. Compress quickly with lower compression level
xz -0 file.txt

# 9. Compress with multiple CPU threads
xz --threads=4 file.txt

# 10. Use extreme compression mode (very slow, best size)
xz --extreme file.txt
```

- **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.

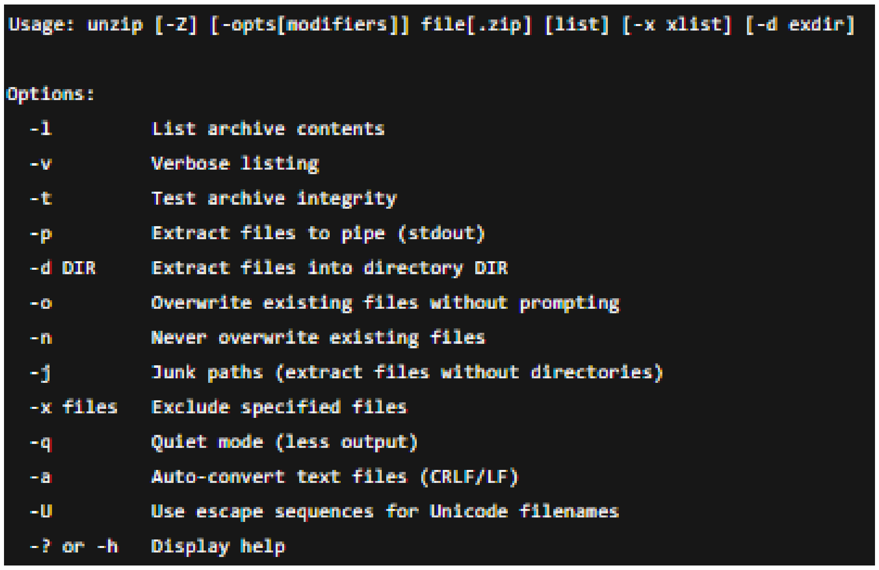


```
Usage: zip [options] zipfile files_list

Options:
  -r        Recurse into directories
  -j        Junk (ignore) directory names
  -0 .. -9 Compression level (0=store only .. 9=best compression)
  -q        Quiet mode
  -v        Verbose mode
  -e        Encrypt archive (prompt for password)
  -P pwd    Encrypt archive using password pwd (insecure)
  -x files Exclude files matching pattern
  -u        Update existing archive
  -d        Delete entries from archive
  -F/-FF    Fix (or try to fix) a broken archive
  -T        Test archive integrity
  -o        Set the archive time to latest file time
  -h        Display help
```

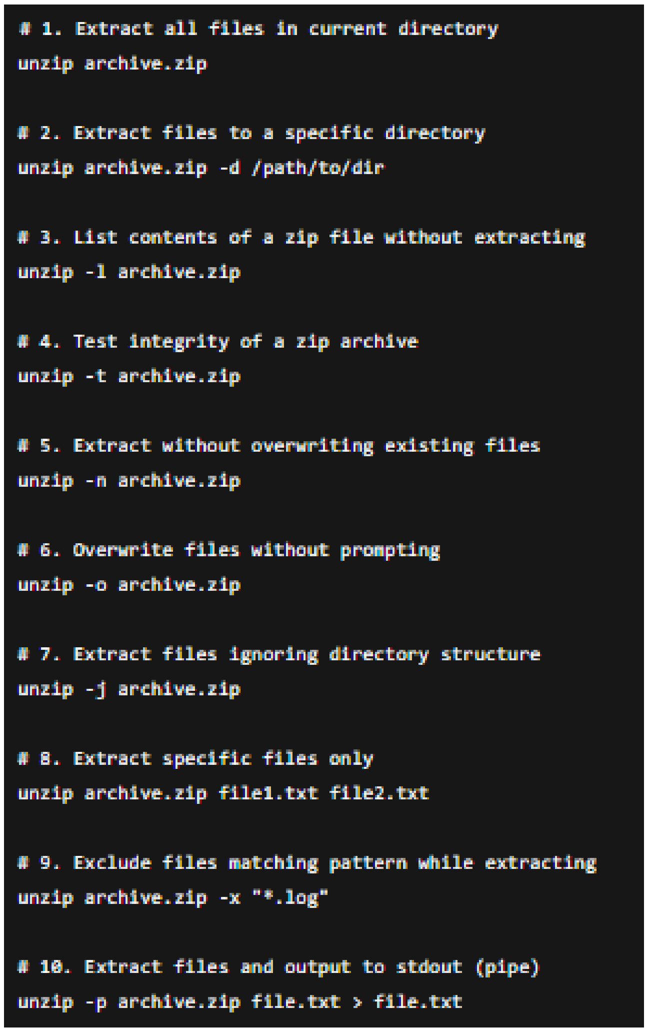


```
# 1. Create a zip archive of files
zip archive.zip file1.txt file2.txt

# 2. Create a zip archive recursively including directories
zip -r archive.zip folder/

# 3. Create a zip archive without directory paths
zip -j archive.zip folder/*

# 4. Set compression level to maximum
zip -9 archive.zip file.txt

# 5. Encrypt archive interactively (asks for password)
zip -e archive.zip file.txt

# 6. Encrypt archive with given password (less secure)
zip -P mypassword archive.zip file.txt

# 7. Update existing archive with new files
zip -u archive.zip newfile.txt

# 8. Delete a file from the archive
zip -d archive.zip file2.txt

# 9. Test archive integrity
zip -T archive.zip

# 10. Exclude files matching pattern
zip -r archive.zip folder/ -x "*.log"
```

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



```
Usage: unzip [-Z] [-opts[modifiers]] file[.zip] [list] [-x xlist] [-d exdir]

Options:
  -l          List archive contents
  -v          Verbose listing
  -t          Test archive integrity
  -p          Extract files to pipe (stdout)
  -d DIR      Extract files into directory DIR
  -o          Overwrite existing files without prompting
  -n          Never overwrite existing files
  -j          Junk paths (extract files without directories)
  -x files    Exclude specified files
  -q          Quiet mode (less output)
  -a          Auto-convert text files (CRLF/LF)
  -U          Use escape sequences for Unicode filenames
  -? or -h    Display help
```



```
# 1. Extract all files in current directory
unzip archive.zip

# 2. Extract files to a specific directory
unzip archive.zip -d /path/to/dir

# 3. List contents of a zip file without extracting
unzip -l archive.zip

# 4. Test integrity of a zip archive
unzip -t archive.zip

# 5. Extract without overwriting existing files
unzip -n archive.zip

# 6. Overwrite files without prompting
unzip -o archive.zip

# 7. Extract files ignoring directory structure
unzip -j archive.zip

# 8. Extract specific files only
unzip archive.zip file1.txt file2.txt

# 9. Exclude files matching pattern while extracting
unzip archive.zip -x "*.log"

# 10. Extract files and output to stdout (pipe)
unzip -p archive.zip file.txt > file.txt
```

Part 1

Variables

1.1- 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.

1.2- 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

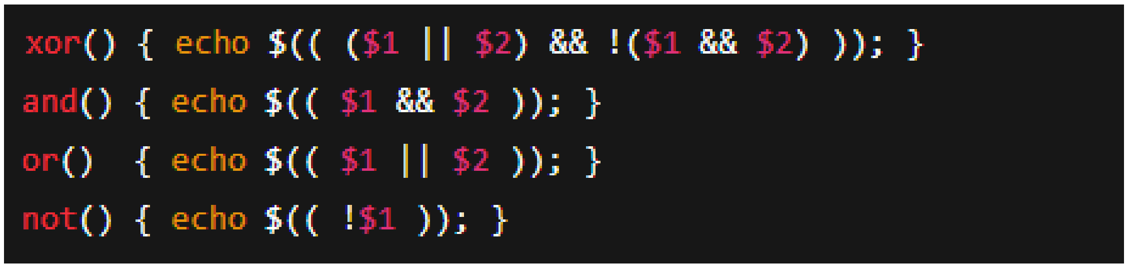
```
$ if ! [ -f "config.txt" ]; then
```

Another Example: Not equal check

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

Boolean Logical Operations:

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



```
xor() { echo $(( ($1 || $2) && !($1 && $2) )); }
and() { echo $(( $1 && $2 )); }
or()  { echo $(( $1 || $2 )); }
not() { echo $(( !$1 )); }
```

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...	Operator	Checks If...
-e	File exists	-w	File is writable
-f	Regular file	-x	File is executable
-d	Directory	file1 -nt file2	file1 is newer than file2
-s	File is not empty		
-r	File is readable	file1 -ot file2	file1 is older than file2

Part 3

if Statements

Basic Syntax:

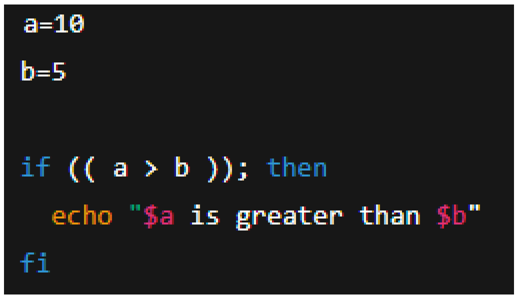
```
$ if CONDITION; then

$ # commands if true

$ fi
```

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

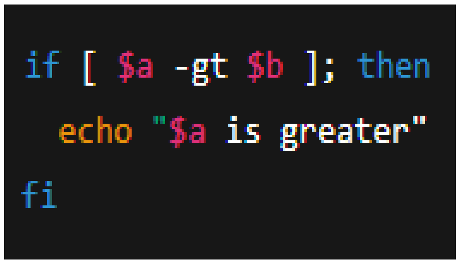
```
$ if CONDITION; then

$ # if true

$ elif OTHER_CONDITION; then

$ # if first false, this is checked

$ else

$ # if all conditions fail

$ fi
```

Accepted CONDITION Types:

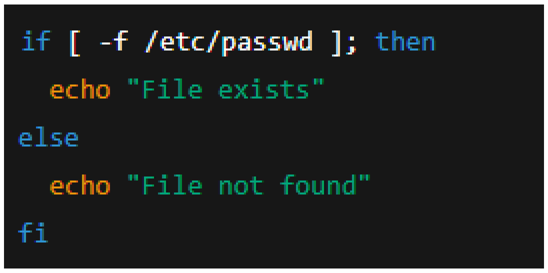
Syntax	Use When...	Example
[ ]	Basic POSIX test	[$a -eq 5]
[[ ]]	Advanced string & logic tests	[[$user == "root"]]
(( ))	Arithmetic expressions	((a > b))
command	Exit status-based checks	if grep "error" log.txt



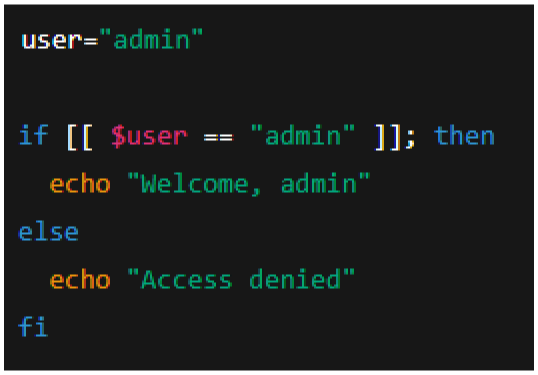
```
a=10
b=5

if (( a > b )); then
  echo "$a is greater than $b"
fi
```



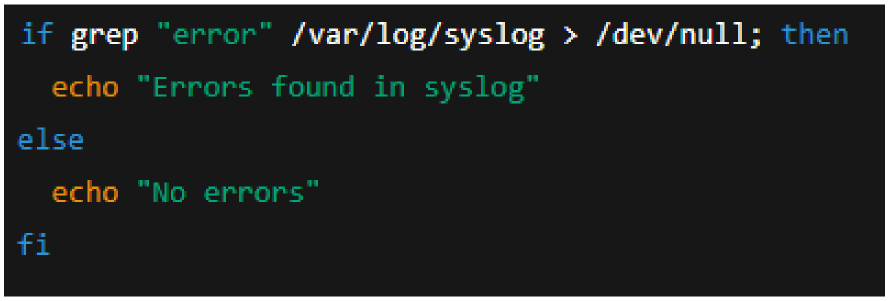
```
if [ $a -gt $b ]; then
  echo "$a is greater"
fi
```



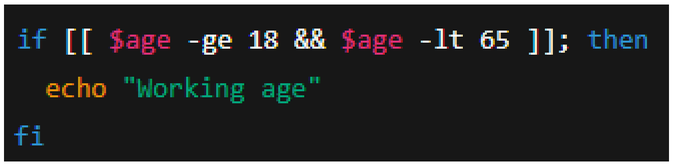
```
if [ -f /etc/passwd ]; then
  echo "File exists"
else
  echo "File not found"
fi
```



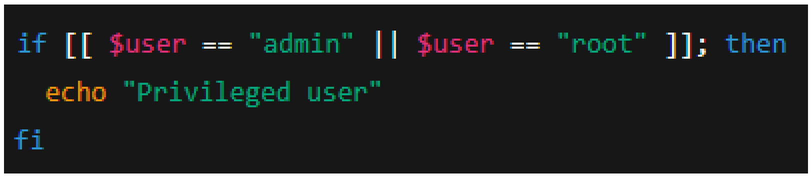
```
user="admin"

if [[ $user == "admin" ]]; then
  echo "Welcome, admin"
else
  echo "Access denied"
fi
```



```
if grep "error" /var/log/syslog > /dev/null; then
  echo "Errors found in syslog"
else
  echo "No errors"
fi
```



```
if [[ $age -ge 18 && $age -lt 65 ]]; then
  echo "Working age"
fi
```



```
if [[ $user == "admin" || $user == "root" ]]; then
  echo "Privileged user"
fi
```

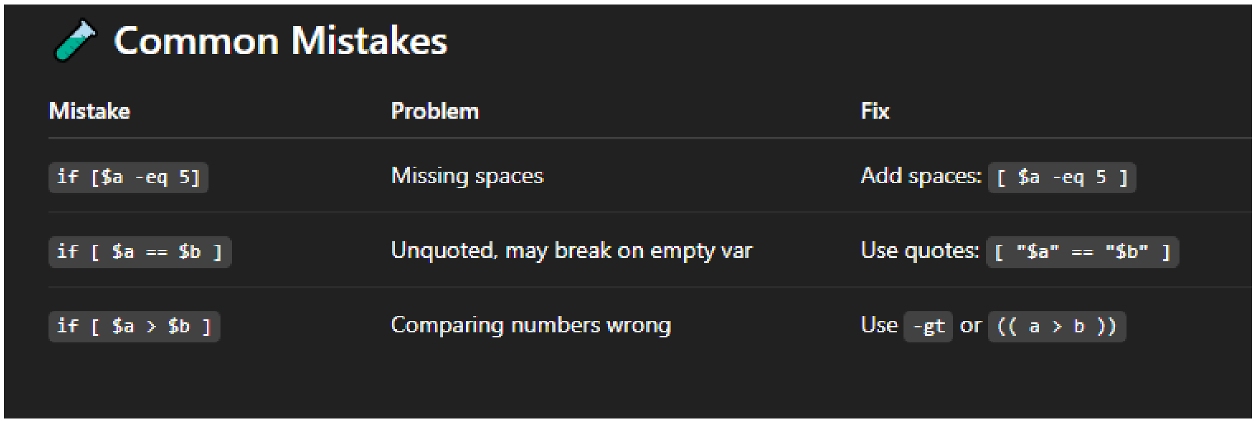
Notes & Gotchas

Don’t forget:

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

```
if [ "$name" == "Obaida" ]; then    # Good
```

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



🧪 Common Mistakes

Mistake	Problem	Fix
`if [$a -eq 5]`	Missing spaces	Add spaces: `[ $a -eq 5 ]`
`if [ $a == $b ]`	Unquoted, may break on empty var	Use quotes: `[ "$a" == "$b" ]`
`if [ $a > $b ]`	Comparing numbers wrong	Use `-gt` or `(( a > b ))`

Part 4

case Statement

Syntax:

```
case $variable in
  pattern1)
    # commands
    ;;
  pattern2)
    # commands
    ;;
  *)
    # default (if nothing matches)
    ;;
esac
```

Examples:

```
echo "Choose an option:"
echo "1) Start"
echo "2) Stop"
echo "3) Restart"

read choice

case $choice in
  1)
    echo "Starting..."
    ;;
  2)
    echo "Stopping..."
    ;;
  3)
    echo "Restarting..."
    ;;
  *)
    echo "Invalid option"
    ;;
esac
```

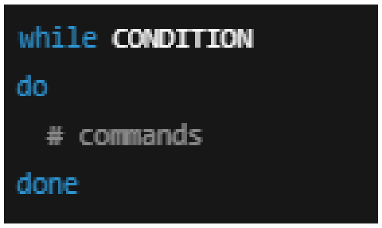
```
file="report.pdf"

case $file in
  *.txt)
    echo "It's a text file"
    ;;
  *.pdf)
    echo "It's a PDF file"
    ;;
  *.jpg|*.png)
    echo "It's an image file"
    ;;
  *)
    echo "Unknown file type"
    ;;
esac
```

```
read -p "Enter a character: " char

case $char in
  [a-z])
    echo "You entered a lowercase letter"
    ;;
  [A-Z])
    echo "You entered an uppercase letter"
    ;;
  [0-9])
    echo "You entered a digit"
    ;;
  *)
    echo "Unknown input"
    ;;
esac
```

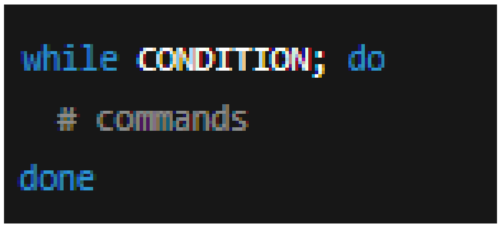
Part 5:

While Loop

Syntax:

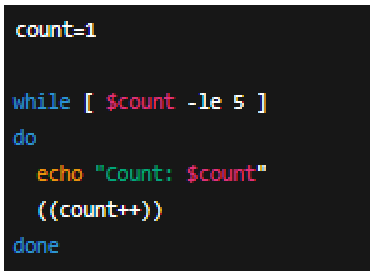


```
while CONDITION
do
  # commands
done
```

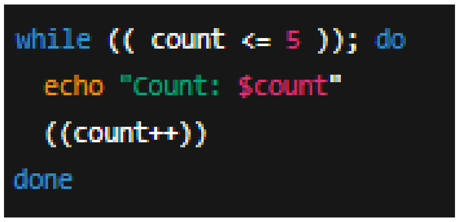


```
while CONDITION; do
  # commands
done
```

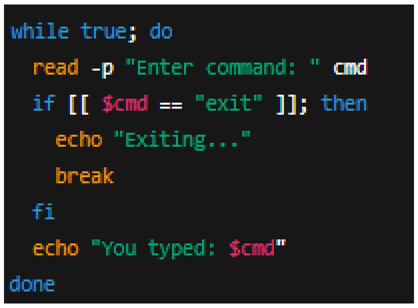
Examples:



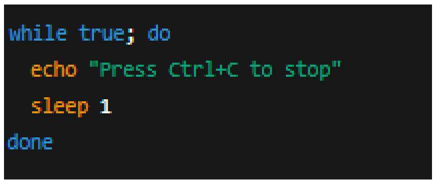
```
count=1

while [ $count -le 5 ]
do
  echo "Count: $count"
  ((count++))
done
```



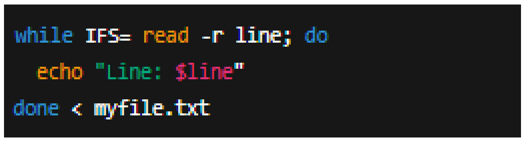
```
while (( count <= 5 )); do
  echo "Count: $count"
  ((count++))
done
```



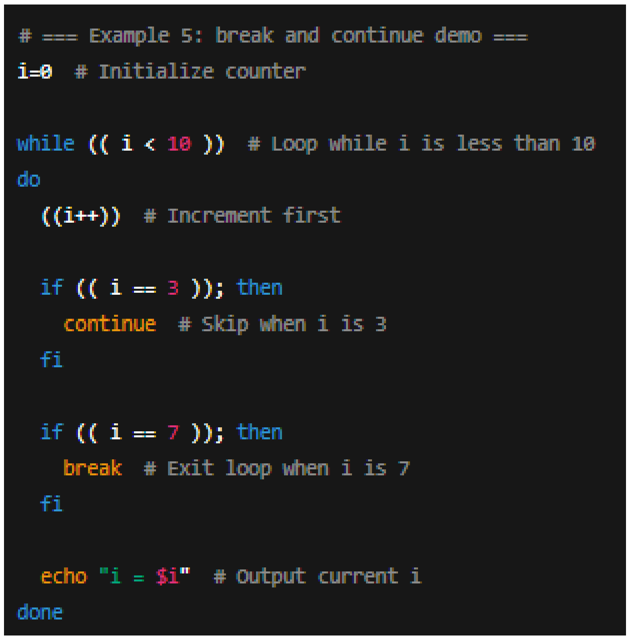
```
while true; do
  read -p "Enter command: " cmd
  if [[ $cmd == "exit" ]]; then
    echo "Exiting..."
    break
  fi
  echo "You typed: $cmd"
done
```



```
while true; do
  echo "Press Ctrl+C to stop"
  sleep 1
done
```



```
while IFS= read -r line; do
  echo "Line: $line"
done < myfile.txt
```



```
# === Example 5: break and continue demo ===
i=0  # Initialize counter

while (( i < 10 ))  # Loop while i is less than 10
do
  ((i++))  # Increment first

  if (( i == 3 )); then
    continue  # Skip when i is 3
  fi

  if (( i == 7 )); then
    break  # Exit loop when i is 7
  fi

  echo "i = $i"  # Output current i
done
```

Part 6

for Loops

Syntax:

```
for item in list
do
  # commands using $item
done
```

Examples:

```
# === Example 1: Loop Through Words ===
for fruit in apple banana orange
do
  echo "I like $fruit"
done
```

```
# === Example 2: Loop Using seq ===
for i in $(seq 1 5)  # Generates: 1 2 3 4 5
do
  echo "Number: $i"
done
```

```
# === Example 3: C-Style For Loop ===
for (( i=1; i<=5; i++ ))
do
  echo "i = $i"
done
```

```
# === Example 4: Loop Over Files ===
for file in /etc/*.conf
do
  echo "Found config: $file"
done
```

```
# === Example 5: Loop Over Command Output ===
for user in $(cut -d: -f1 /etc/passwd)
do
  echo "User: $user"
done
```

```
# === Example 6: Nested Loop (Multiplication Table) ===
for i in {1..3}
do
  for j in {1..3}
  do
    echo -n "$(( i * j )) "  # Print product without newline
  done
  echo  # Print newline after each inner loop
done
```

```
for i in {1..5}; do echo $i; done
```

```
for i in $(seq 1 2 9); do echo $i; done  # 1 3 5 7 9
```

```
# === Example 7: Using break and continue ===
for i in {1..10}
do
  if (( i == 3 )); then
    continue  # Skip number 3
  fi
  if (( i == 7 )); then
    break     # Stop loop at 7
  fi
  echo "i = $i"
done
```

Part 7

Functions

Syntax

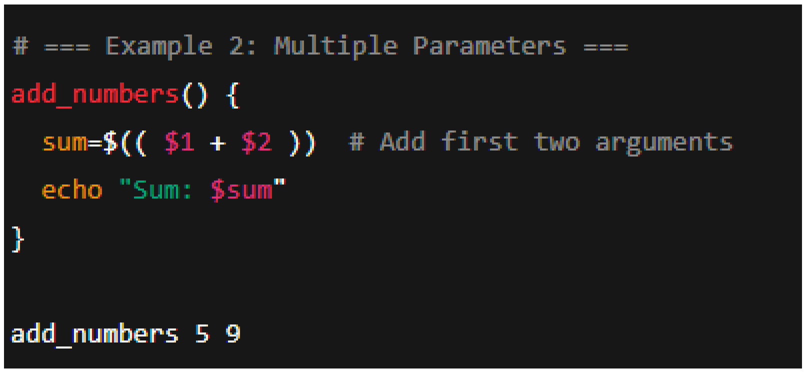
```
# Option 1 (preferred)
function name {
  # commands
}

# Option 2
name() {
  # commands
}
```

Passing a variable syntax

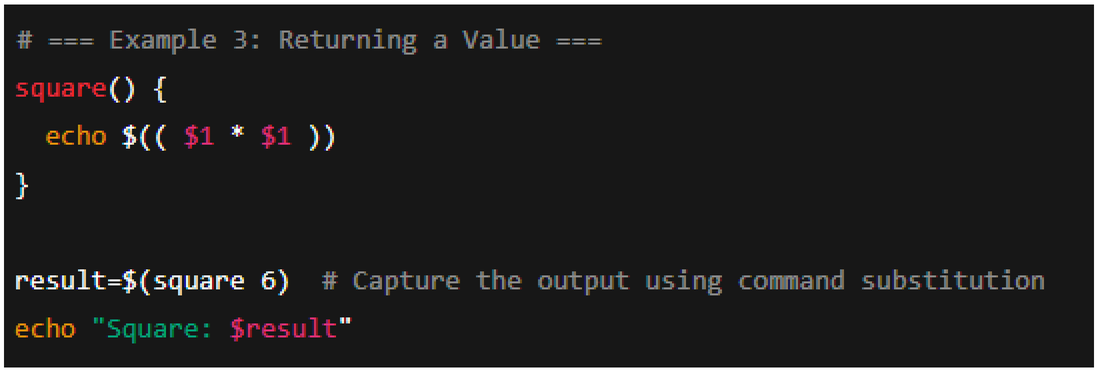
```
my_function() {
  echo "Hello, $1"  # $1 refers to the first argument passed
}

name="Obaida"
my_function "$name"  # Pass variable to function
```

Examples:

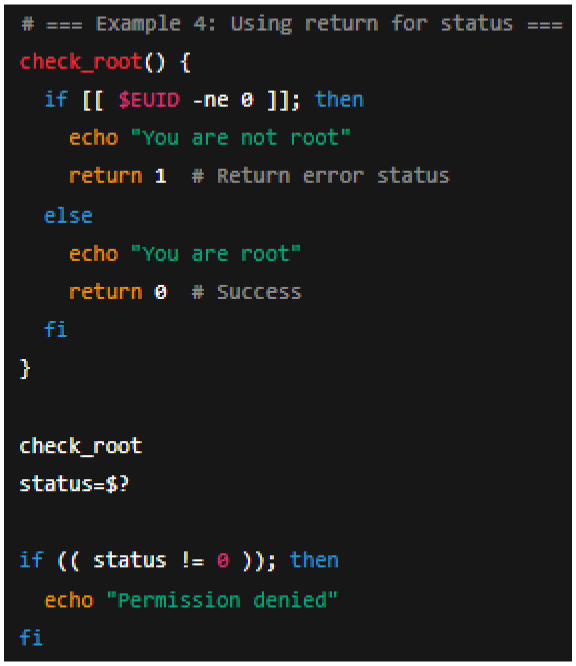
```
# === Example 1: Basic Function ===
greet() {
  echo "Hello, $1!"  # $1 is the first argument passed to the function
}

greet "Obaida"  # Call the function
```



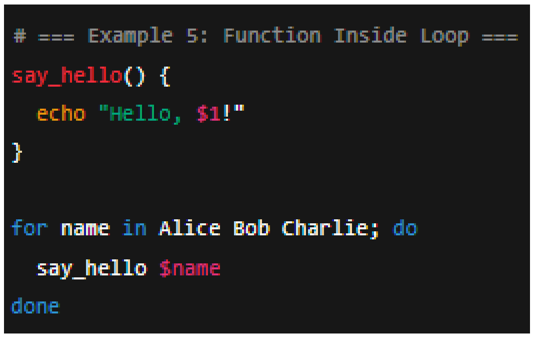
```bash
# === Example 2: Multiple Parameters ===
add_numbers() {
  sum=$(( $1 + $2 ))  # Add first two arguments
  echo "Sum: $sum"
}

add_numbers 5 9
```



```bash
# === Example 3: Returning a Value ===
square() {
  echo $(( $1 * $1 ))
}

result=$(square 6)  # Capture the output using command substitution
echo "Square: $result"
```

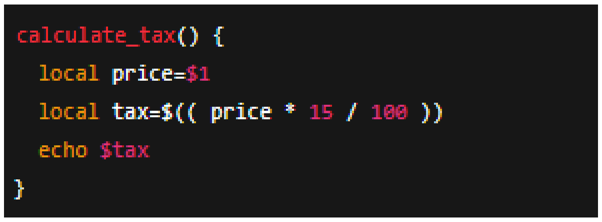


```bash
# === Example 4: Using return for status ===
check_root() {
  if [[ $EUID -ne 0 ]]; then
    echo "You are not root"
    return 1  # Return error status
  else
    echo "You are root"
    return 0  # Success
  fi
}

check_root
status=$?

if (( status != 0 )); then
  echo "Permission denied"
fi
```



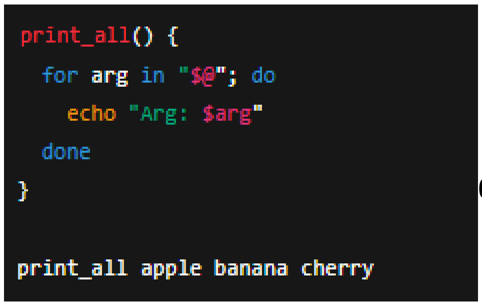
```bash
# === Example 5: Function Inside Loop ===
say_hello() {
  echo "Hello, $1!"
}

for name in Alice Bob Charlie; do
  say_hello $name
done
```

Local variables inside function



```bash
calculate_tax() {
  local price=$1
  local tax=$(( price * 15 / 100 ))
  echo $tax
}
```

Access All Arguments in a Loop:



```bash
print_all() {
  for arg in "$@"; do
    echo "Arg: $arg"
  done
}

print_all apple banana cherry
```

Part 8

Arguments

```
# === Example 1: Arguments in a Bash Script ===
#!/bin/bash

echo "Script name: $0"     # $0 is the name of the script
echo "First arg: $1"       # $1 is the first CLI argument
echo "Second arg: $2"      # $2 is the second CLI argument
echo "Total arguments: $#"

echo "All arguments using \$@:"
for arg in "$@"; do        # "$@" preserves spaces in each argument
  echo "- $arg"
done
```

Key and value argument syntax using getopts

```
#!/bin/bash

# Default values
count=1
country="Unknown"

while getopts "c:n:" opt; do
  case $opt in
    c)
      count=$OPTARG
      ;;
    n)
      country=$OPTARG
      ;;
    *)
      echo "Usage: $0 -c count -n country"
      exit 1
      ;;
  esac
done

echo "Count is: $count"
echo "Country is: $country"
```

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

```bash
#!/bin/bash
# This script prompts the user to enter input and reads one line

echo "Please enter your name:"
read name  # Reads a line from keyboard (stdin)

echo "Hello, $name!"
```

Pipe

```bash
#!/bin/bash
# This script reads input piped from another command and prints each line

echo "Reading input from a pipe..."

while IFS= read -r line; do
  echo "Received line: $line"
done
```

```bash
cat data.txt | ./script_pipe.sh
```

<

```bash
#!/bin/bash
# This script reads lines from stdin and prints each line with a prefix

echo "Reading input via < redirection..."

while IFS= read -r line; do
  echo "Line read: $line"
done

echo "Finished reading input."
```

```bash
./script_redirect.sh < data.txt
```

Stdout will be skipped

Stder: is the standard error stream.

Why use stderr?

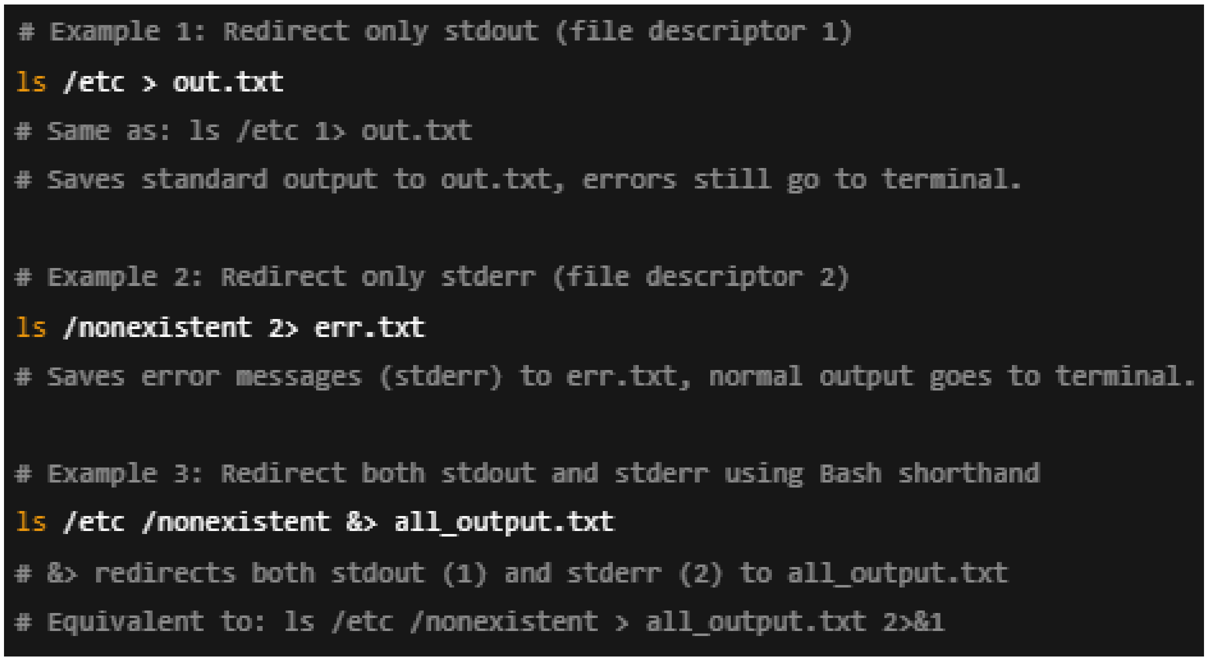
- **Separating errors from normal output allows:**
- **Logging errors separately.**
- **Keeping program output clean.**
- **Handling errors programmatically.**

Directing the output of a command:

1 is for the standard output

2 for the standard error

& for Both



```
# Example 1: Redirect only stdout (file descriptor 1)
ls /etc > out.txt
# Same as: ls /etc 1> out.txt
# Saves standard output to out.txt, errors still go to terminal.

# Example 2: Redirect only stderr (file descriptor 2)
ls /nonexistent 2> err.txt
# Saves error messages (stderr) to err.txt, normal output goes to terminal.

# Example 3: Redirect both stdout and stderr using Bash shorthand
ls /etc /nonexistent &> all_output.txt
# &> redirects both stdout (1) and stderr (2) to all_output.txt
# Equivalent to: ls /etc /nonexistent > all_output.txt 2>&1
```

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:

```
exit 0     # success
exit 1     # failure
```

```
#!/bin/bash
# === Example 2: Custom exit status ===

echo "Running checks..."

if [ "$1" == "pass" ]; then
  echo "Passed."
  exit 0  # success
else
  echo "Failed."
  exit 1  # failure
fi
```

```
#!/bin/bash
# === Example 1: Using $? to check status ===

echo "Listing /etc"
ls /etc

echo "Exit code of ls: $?"  # Should be 0

echo "Listing missing folder"
ls /not_there

echo "Exit code of failed ls: $?"  # Should be non-zero (e.g., 2)
```

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