Day 6   
Employee ID:109134424   
Login ID:rngro   
Name:-Ranga Narayana Srinivas Rao

**1.RegEX Symbols in linux ? List them down with description?**   
Answer:-

Here’s a list of **common RegEx (Regular Expression) symbols used in Linux** along with their **descriptions**. Regular expressions are powerful tools for pattern matching, widely used in commands like grep, sed, awk, and more.

**🔤 Basic Symbols**

| **Symbol** | **Description** |
| --- | --- |
| . | Matches **any single character** except newline (\n) |
| ^ | Matches the **start of a line** |
| $ | Matches the **end of a line** |
| \* | Matches **zero or more** of the previous character |
| + | Matches **one or more** of the previous character |
| ? | Matches **zero or one** of the previous character (makes it optional) |
| \ | Escapes a special character (e.g., \. matches a literal dot) |

**🧩 Character Classes**

| **Symbol** | **Description** |
| --- | --- |
| [abc] | Matches **any one** of the characters: a, b, or c |
| [^abc] | Matches **any character except** a, b, or c |
| [a-z] | Matches any **lowercase letter** from a to z |
| [A-Z] | Matches any **uppercase letter** from A to Z |
| [0-9] | Matches any **digit** from 0 to 9 |
| . | Matches **any single character** |
| [:digit:] | Matches any digit (same as [0-9]) — used with [[:digit:]] |
| [:alpha:] | Matches any alphabetic character |
| [:space:] | Matches any whitespace character (space, tab, newline) |

**🔄 Quantifiers**

| **Symbol** | **Description** |
| --- | --- |
| {n} | Matches exactly **n times** |
| {n,} | Matches **n or more times** |
| {n,m} | Matches **between n and m times** |

**🚦 Grouping and Alternation**

| **Symbol** | **Description** |
| --- | --- |
| () | Groups expressions together |
| ` | ` |

**🧼 Escape Sequences**

| **Symbol** | **Description** |
| --- | --- |
| \d | Matches any digit (same as [0-9]) |
| \D | Matches any non-digit |
| \w | Matches any word character (alphanumeric + underscore) |
| \W | Matches any non-word character |
| \s | Matches any whitespace character |
| \S | Matches any non-whitespace character |

**🛠️ Usage in Commands**

* grep "^Error" → Matches lines starting with "Error"
* grep "[0-9]\{3\}" → Matches exactly 3 digits (escaped braces for basic grep)
* sed -n "/pattern/p" → Prints lines matching "pattern"
* awk '/[a-z]+/' → Matches lines with one or more lowercase letters

**2. What are the imp features of Linux os ?**

Answer:-

**✅ 1. Open Source**

* **Linux is free to use, modify, and distribute.**
* The source code is open to everyone, so developers can customize it to suit their needs.

**✅ 2. Multitasking**

* **Linux can run multiple tasks or programs at the same time.**
* You can download files, listen to music, and write documents all at once.

**✅ 3. Multi-user Capability**

* **Multiple users can use the same Linux system at the same time.**
* Each user gets their own files and settings, and one user doesn’t interfere with others.

**✅ 4. Security**

* **Linux is known for its strong security features.**
* It uses file permissions, user roles, firewalls, and encryption to protect the system.
* Viruses and malware are rare in Linux.

**✅ 5. Stability and Reliability**

* **Linux rarely crashes and can run for years without restarting.**
* It is used on servers and systems that need to stay online all the time.

**✅ 6. Portability**

* **Linux can run on almost any type of hardware.**
* From personal computers to smartphones, tablets, servers, and even supercomputers.

**✅ 7. Shell/Command Line Interface (CLI)**

* **Linux provides a powerful command-line interface (Terminal).**
* You can automate tasks, manage files, and control system behavior with commands.

**✅ 8. Support for Networking**

* **Linux has excellent networking features.**
* It is often used as a server (web server, DNS server, file server, etc.).

**✅ 9. File System Support**

* **Linux supports many file systems**, such as ext4, XFS, Btrfs, FAT32, NTFS, etc.
* This makes it flexible for data storage and sharing.

**✅ 10. Community Support**

* **Linux has a large global community.**
* If you face an issue, you can find help in forums, documentation, or tutorials online.

**✅ 11. Customizable**

* **You can customize almost everything in Linux**, from the desktop environment to system behavior.
* Lightweight versions (like Lubuntu) are available for old systems.

**✅ 12. Performance**

* **Linux is fast and efficient.**
* It uses less system resources compared to other operating systems like Windows.

**3. What is a Kernel in Linux and can you explain its functions?**  
Answer:-

The **kernel** is the **core part of the Linux operating system**.  
It acts as a **bridge between the hardware and software**, managing communication between the user applications and the physical devices (CPU, memory, disk, etc.).

**🔍 Simple Definition:**

**The kernel is like a manager that handles system resources and ensures everything works smoothly between hardware and software.**

**🧠 Main Functions of the Linux Kernel:**

| **Function** | **Description** |
| --- | --- |
| **1. Process Management** | - Handles the creation, scheduling, and termination of processes.- Ensures each program gets time to run on the CPU. |
| **2. Memory Management** | - Manages system memory (RAM).- Allocates memory to programs and frees it when not needed.- Prevents programs from using others’ memory. |
| **3. Device Management** | - Controls hardware devices using drivers (keyboard, mouse, hard drive, etc.).- Ensures software can communicate with devices. |
| **4. File System Management** | - Manages files and directories on storage devices.- Handles permissions, reading, writing, and organization of data. |
| **5. System Call Handling** | - Provides a set of functions (system calls) that programs use to request services from the kernel.- Example: opening a file, creating a process, etc. |
| **6. Interrupt Handling** | - Responds to hardware interrupts (e.g., keyboard press or network input).- Ensures the system reacts quickly to real-time events. |
| **7. Networking** | - Manages network connections, data transfer, and security.- Enables communication between computers. |
| **8. Security & Access Control** | - Ensures that only authorized users and processes can access specific resources.- Manages user permissions and system protection. |

**🛠️ Types of Kernels:**

* **Monolithic Kernel** – All functions run in one large block of code (used in Linux).
* **Microkernel** – Only basic functions are in the kernel; others run in user space.
* **Hybrid Kernel** – Combines features of both monolithic and microkernel.

**📌 Summary:**

* The **kernel is the heart of Linux OS**.
* It manages everything from **processes, memory, and devices to files and network communication**.
* Without the kernel, your Linux system wouldn’t be able to function.

**4. What is BASH? Full form with explaination?**Answer:-

**BASH** stands for:

**Bourne Again SHell**

**📘 Explanation:**

BASH is a **command-line shell** and **scripting language** used in **Linux and Unix-based operating systems**.  
It is the **default shell** for most Linux distributions.

**🧾 Full Form Breakdown:**

* **Bourne**: Refers to the original Unix shell called the **Bourne Shell (sh)** developed by **Stephen Bourne**.
* **Again**: Indicates that it's an **improved version** or **replacement** of the original Bourne Shell.
* **Shell**: A program that lets users interact with the operating system through commands.

**🛠️ Key Features of BASH:**

| **Feature** | **Description** |
| --- | --- |
| **Command Execution** | Run Linux commands like ls, cd, pwd, etc. |
| **Scripting** | Write scripts to automate tasks using .sh files. |
| **Variables** | Supports user-defined and system variables. |
| **Loops and Conditions** | Write loops (for, while) and conditions (if, case). |
| **Command History** | Access previously run commands with arrow keys. |
| **Auto-completion** | Press Tab to auto-complete file or command names. |
| **Redirection and Pipes** | Redirect input/output using >, <, ` |
| **Job Control** | Pause, resume, and manage background tasks (&, jobs, fg, bg). |

**💡 Real-World Example:**

You open a **terminal** in Linux and type:

echo "Hello, World!"

BASH processes this command, displays the output, and waits for your next input.

**🧠 Summary:**

* **BASH** = **Bourne Again Shell**
* It's a **powerful shell and scripting tool** used to control Linux systems.
* It helps users **interact with the OS, automate tasks, and write scripts.**

**5. What is the diffrenece between window and linux?**Answer:-

**💻 Windows vs Linux – Detailed but Easy Comparison:**

| **Feature** | **Windows** | **Linux** |
| --- | --- | --- |
| 💰 **Cost** | You must **buy a license** (usually pre-installed) | **Free to download and use**, open source |
| 🎨 **Interface** | Very **graphical and user-friendly**, mostly mouse-based | Also has a GUI, but **command-line (terminal)** is used often |
| 🧱 **Source Code Access** | Closed-source (users can't see or change internal code) | Open-source (users can see, modify, and improve the code) |
| 🔒 **Security** | Targeted more by viruses and hackers | More secure and **less prone to malware** |
| 📦 **Software Installation** | Software comes as .exe or through Microsoft Store | Uses package managers (apt, dnf, etc.) or .deb/.rpm files |
| ⚙️ **Control over System** | Limited — Microsoft controls updates, settings, etc. | Full control — users can modify almost anything |
| 💾 **Usage** | Best for personal use, games, offices (Word, Excel) | Best for **programming, servers, cybersecurity, automation** |
| 🔁 **Updates** | Automatically managed by Microsoft | You choose what and when to update |
| 👨‍🔧 **Used by** | General users, gamers, office workers | Developers, system admins, ethical hackers, students |
| 🧠 **Learning Curve** | Very easy to start using | Needs some learning at first, especially command line |

**🧃 Real-Life Example (with more explanation):**

**🏠 Windows is like a Furnished Apartment:**

* You just move in and **start living**.
* Everything (furniture, wiring, walls) is **pre-arranged**.
* You don’t have permission to **change major things** like breaking a wall or replacing the flooring.
* If something goes wrong, you **call the building company** to fix it.
* **Good for users who want ease and no technical hassle**.

**🏗️ Linux is like Building Your Own House:**

* You start with **an empty plot**.
* You can **build the house your way** — choose the rooms, colors, design.
* You are responsible for **maintenance, changes, upgrades**.
* You need **some knowledge**, but it gives **full freedom and power**.
* **Great for people who like to explore, customize, and control everything**.

**📝 Summary:**

* **Windows = Easy, fixed, user-friendly, but limited in control**
* **Linux = Powerful, flexible, secure, but needs some learning**

**6. Define the basic components of Linux?**Answer

**🧩 Basic Components of the Linux Operating System:**

| **Component** | **Description** |
| --- | --- |
| **1. Kernel** | 🧠 The **core part** of Linux. It directly interacts with the hardware (CPU, memory, disk, etc.) and manages system resources. Example: It decides which app gets how much memory or CPU time. |
| **2. Shell** | 💬 The **interface between user and kernel**. It takes your commands (typed in Terminal), passes them to the kernel, and shows results. Example: BASH (Bourne Again Shell) is a popular shell in Linux. |
| **3. File System** | 📂 Organizes data and files in a structured way (like folders). Everything in Linux is treated as a **file** — even devices. Example: /home/user, /etc, /bin, etc. |
| **4. System Libraries** | 📚 These are collections of **pre-written code** used by programs to perform tasks (like printing text, opening files). They help programs run **efficiently** without writing code from scratch. Example: GNU C Library (glibc). |
| **5. System Utilities** | 🛠️ Small programs/tools that help you perform tasks like file handling, software installation, monitoring, etc. Example: cp, mv, top, apt, ps, etc. |
| **6. User Applications** | 🧑‍💻 These are the programs used by users. Can be graphical (like web browsers, editors) or command-line tools. Example: Firefox, LibreOffice, VS Code, Terminal commands. |
| **7. Daemons** | 👻 Background services or processes that **run silently** and perform tasks like printing, managing devices, networking, etc. Example: cron (for scheduled tasks), cupsd (for printers). |

**🏗️ Real-Life Example: Linux as a Factory**

| **Factory Part** | **Linux Component** | **Role** |
| --- | --- | --- |
| Factory Manager | Kernel | Controls everything in the factory |
| Receptionist | Shell | Takes requests from visitors (you) and passes them to the manager |
| Factory Files | File System | Organizes all tools, documents, machines, etc. |
| Toolkit | System Libraries | Provides ready-made tools to build things |
| Workers | System Utilities | Do the actual work (move files, install things) |
| Machines | User Applications | Used by people to complete tasks |
| Security Cameras | Daemons | Run in background, watching and maintaining the system |

**7. Is it legal to edit Kernal?**

Answer:-

**✅ Linux Kernel is Open Source and Licensed under GPL (GNU General Public License)**

This means:

* **You are allowed to** view, modify, and share the Linux kernel code.
* You can even use it to create your **own version** of Linux.
* **BUT** if you distribute your modified version to others, you **must also share your source code** under the same GPL license.

**🛠️ What You Can Do Legally:**

* Edit the kernel for **personal use** or **learning**.
* Modify it for your own **custom project** or hardware.
* Contribute improvements to the **Linux community**.

**⚠️ What You Can’t Do:**

* You **cannot take the Linux kernel**, modify it, and **sell it as your own closed-source product**.
* You must **respect the GPL license** — any distributed changes must remain open.

**📘 Real-Life Example:**

Imagine Linux kernel is like a **free recipe** for a cake:

* You can bake it, change ingredients, or add new flavors.
* You can even give your cake to others — **but you must also give them your updated recipe** (open source).

**8.Can you explain LILO?**Answer:-

**LILO** stands for **Linux Loader**.  
It is a **boot loader** used in **Linux operating systems**.

**🚀 What Does LILO Do?**

When you turn on your computer, LILO is the program that:

1. **Loads the Linux kernel** into memory.
2. **Starts the Linux operating system**.
3. Can also let you **choose between different operating systems** (like Windows or Linux) if more than one is installed (this is called dual-booting).

**⚙️ How LILO Works:**

* LILO is installed in the **MBR (Master Boot Record)** or **boot sector**.
* When your computer starts, the **BIOS** hands control to LILO.
* LILO then loads the **chosen OS kernel** and starts the system.

**📌 Key Features of LILO:**

| **Feature** | **Explanation** |
| --- | --- |
| **Multi-boot** | You can choose between multiple operating systems at boot time. |
| **Configurable** | You can edit the LILO config file to set default OS, time delay, etc. |
| **Simple** | Works well for basic boot tasks, especially in older systems. |

**⚠️ Note:**

LILO is **less common today**. It has mostly been replaced by **GRUB (GRand Unified Bootloader)**, which is more powerful and flexible.

**🧁 Real-Life Example:**

Think of LILO as a **menu at a restaurant**:

* You come in (boot the computer).
* The waiter (LILO) gives you the menu (list of OS options).
* You pick what you want to eat (Linux or Windows).
* The waiter (LILO) brings that dish to your table (loads the OS).

**9. What is shell? How many shells are there and what are they ? can you explain.**

Answer:-

A **shell** is a **program that acts as an interface between the user and the operating system** (kernel).

* It allows you to **communicate with the computer** by typing commands.
* When you type a command, the shell **interprets it** and tells the **kernel** to perform the task.
* Shells can be **command-line based** or **graphical**, but in Linux, the term "shell" usually refers to the **command-line shell**.

**💬 Real-Life Example:**

Imagine you're in a restaurant:

* **You (User)** tell the **waiter (Shell)** what you want to eat.
* The waiter communicates your order to the **chef (Kernel)**.
* The chef prepares the food and the waiter brings it back to you.

So, **Shell = Waiter**, helping you talk to the system.

**🛠️ Types of Shells in Linux**

There are **several types of shells**, but here are the most common ones:

| **Shell Name** | **Full Form / Description** | **Command to Use** |
| --- | --- | --- |
| **BASH** | **Bourne Again SHell** – Most popular and default in most Linux systems. Easy to use and powerful. | bash |
| **SH** | **Bourne Shell** – Original shell, simple and old. Basis for others like bash. | sh |
| **CSH** | **C Shell** – C-like syntax; useful for people familiar with C programming. | csh |
| **KSH** | **Korn Shell** – A combination of features from Bourne and C Shell. | ksh |
| **ZSH** | **Z Shell** – Advanced and modern shell with features like auto-correction, themes, plugins. | zsh |
| **Fish** | **Friendly Interactive Shell** – User-friendly with syntax highlighting and autosuggestions. | fish |
| **Tcsh** | Enhanced version of C Shell with more features and editing options. | tcsh |

**📝 Summary:**

* A **shell** is the **middleman** between you and the Linux system.
* It helps you **run commands, manage files, write scripts**, and more.
* There are many types of shells, and you can choose or change your default shell based on your needs and preference.

**10. What is Swap space ?**

Answer:-

**Swap space** is a portion of the hard drive or SSD that is used as **virtual memory** when your system’s **RAM (main memory)** is full.

**🧠 Simple Explanation:**

* Your computer runs programs in **RAM** (fast memory).
* But if RAM is **completely used**, Linux temporarily moves some inactive data from RAM to **swap space**.
* This helps your system **keep running smoothly** without crashing.

**📘 Real-Life Example:**

Imagine your desk (RAM) is full of books while you're studying.

* You take some books you don’t need right now and put them in your **drawer (swap space)**.
* When you need them again, you pull them back onto the desk.
* This gives you more **working space**, but accessing the drawer is **slower than using the desk**.

**🔍 Where is Swap Located?**

* **Swap Partition**: A separate section of the hard drive reserved only for swapping.
* **Swap File**: A regular file used as swap, easier to manage and resize.

**⚙️ Key Features of Swap Space:**

| **Feature** | **Description** |
| --- | --- |
| **Backup memory** | Acts as a backup when RAM is full. |
| **Supports hibernation** | Saves system state to swap when you hibernate. |
| **Slower than RAM** | Because it uses disk storage, not actual memory. |
| **Optional but recommended** | Especially useful for systems with limited RAM. |

**📝 Summary:**

* **Swap space = backup memory on disk**.
* It helps Linux manage more applications than RAM can hold.
* **Too much swapping** can slow down the system, so enough **physical RAM** is still important.

**11. What is Mount ? how do you mount and unmount file system in Linux?**  
Answer:-

In Linux, **mounting** means **attaching a storage device or filesystem** (like USB, hard drive, CD/DVD) to a **directory** (folder) so that you can access its contents.

**🧠 Simple Explanation:**

* In Linux, you **don’t access drives as C:, D:, etc.**
* Instead, you **mount** them to a folder (like /mnt/usb) so that Linux can read/write files.

**📘 Real-Life Example:**

Imagine a bookshelf (**Linux directory**).  
When you buy a new book (**USB drive**), you can’t read it unless you **place it on a shelf** (**mount it**).  
To stop reading it, you **remove it from the shelf** (**unmount it**).

**🛠️ How to Mount a Filesystem**

1. **Create a mount point (folder):**
2. sudo mkdir /mnt/myusb
3. **Mount the device:**
4. sudo mount /dev/sdb1 /mnt/myusb

🔹 /dev/sdb1 is the device name (can vary).  
🔹 /mnt/myusb is where you want to access the files.

1. **Access files:**  
   Now you can go to /mnt/myusb and use the files on your USB or drive.

**🔄 How to Unmount a Filesystem**

When you're done, **unmount it** safely:

sudo umount /mnt/myusb

🔸 This ensures no data is being written and prevents corruption.

**🧪 To View Mounted Filesystems:**

mount

or

df -h

**📝 Summary:**

| **Task** | **Command** |
| --- | --- |
| Create mount point | sudo mkdir /mnt/myusb |
| Mount device | sudo mount /dev/sdb1 /mnt/myusb |
| Unmount device | sudo umount /mnt/myusb |
| Check mounts | mount or df -h |

**12.What is chmod command ? how to use it?**Answer:-

chmod stands for **change mode**.  
It is used to **change the permissions** (read, write, execute) of files and directories.

**Why is chmod important?**

Linux files have **permissions** that control who can:

* **Read** the file (r)
* **Write** (modify) the file (w)
* **Execute** the file (x)

chmod lets you set or modify these permissions for:

* The **owner** of the file
* The **group** of the file
* **Others** (everyone else)

**Basic Permission Types**

| **Symbol** | **Meaning** |
| --- | --- |
| r | Read |
| w | Write |
| x | Execute |

**Permission Categories**

| **Category** | **Who it means** |
| --- | --- |
| u | User (owner) |
| g | Group |
| o | Others (everyone else) |
| a | All (user, group, others) |

**How to Use chmod**

There are **two ways** to use chmod:

**1. Symbolic mode (using letters)**

Format:

chmod [who][+/-/=][permission] filename

* + adds permission
* - removes permission
* = sets exact permission (removes others)

Example:

chmod u+x script.sh

Means: add execute (x) permission for user (owner) on script.sh.

**2. Numeric (octal) mode**

Each permission has a number value:

| **Permission** | **Value** |
| --- | --- |
| Read (r) | 4 |
| Write (w) | 2 |
| Execute (x) | 1 |

Add values for each category (user, group, others):

Example:

* 7 = 4 + 2 + 1 = read + write + execute
* 6 = 4 + 2 + 0 = read + write
* 5 = 4 + 0 + 1 = read + execute

Command format:

chmod XYZ filename

Where X = user permissions, Y = group, Z = others.

Example:

chmod 755 script.sh

Means:

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

**Check permissions using:**

ls -l filename

**Summary Examples**

| **Command** | **Meaning** |
| --- | --- |
| chmod u+x file | Add execute permission to user |
| chmod go-w file | Remove write permission from group & others |
| chmod 644 file | User: rw-, Group: r--, Others: r-- |
| chmod 755 script.sh | User: rwx, Group: r-x, Others: r-x |

**13.** **Can you add a new user account? Crate a new user in different ways and paste ss?**Answer

**How to Add a New User in Linux: Different Methods**

**1. Using adduser (Friendly Script)**

sudo adduser newusername

* This command will prompt you to enter the new user's password and other optional info.
* It automatically creates the home directory, sets permissions, and more.

**Example:**

sudo adduser ranga

**Expected output:**

Adding user `ranga' ...

Adding new group `ranga' (1001) ...

Adding new user `ranga' (1001) with group `ranga' ...

Creating home directory `/home/ranga' ...

Copying files from `/etc/skel' ...

Enter new UNIX password:

Retype new UNIX password:

passwd: password updated successfully

Changing the user information for ranga

Enter the new value, or press ENTER for the default

Full Name []:

Room Number []:

Work Phone []:

Home Phone []:

Other []:

Is the information correct? [Y/n]

**2. Using useradd (More Manual)**

sudo useradd -m -s /bin/bash newusername

sudo passwd newusername

* -m creates a home directory /home/newusername.
* -s /bin/bash sets the default shell.
* passwd sets the user password.

**Example:**

sudo useradd -m -s /bin/bash ranga

sudo passwd ranga

**Expected output when setting password:**

Enter new UNIX password:

Retype new UNIX password:

passwd: password updated successfully

**3. Add user and add to a group at the same time**

sudo useradd -m -G groupname newusername

sudo passwd newusername

* -G adds the user to an additional group.

Example:

sudo useradd -m -G sudo ranga

sudo passwd ranga

This creates user ranga with a home directory and adds them to the sudo group.

**How to verify user creation**

Check user info:

id newusername

Example:

id ranga

Expected output:

uid=1001(ranga) gid=1001(ranga) groups=1001(ranga),27(sudo)

Check if home directory exists:

ls -ld /home/ranga

**14. Can you change the password of a user?How do you do that? Plz share ss?**Answer:-

**1. Change your own password:**

Just type:

passwd

It will ask for your **current password**, then prompt for the **new password** twice.

**2. Change another user’s password (as root or sudo user):**

sudo passwd username

Replace username with the actual username.

Example:

sudo passwd ranga

You will be prompted to enter the new password twice for that user.

**What the terminal looks like:**

Here’s an example terminal session for changing another user’s password (let’s say user is ranga):

$ sudo passwd ranga

Enter new UNIX password:

Retype new UNIX password:

passwd: password updated successfully

**15.What is diff between Process and Thread?**

Answer:-:

**🖥️ What is a Process?**

* A **process** is a **program in execution** — like when you open a browser, the browser runs as a process.
* It has its own **memory space**, **system resources**, and **program counter**.
* Each process is independent and isolated from others.

**🔄 What is a Thread?**

* A **thread** is a **smaller unit of a process**.
* A process can have **one or more threads** running inside it.
* Threads share the same **memory space** and resources of the parent process but execute independently.
* Threads allow a program to do **multiple tasks at the same time** (parallelism).

**🧩 Analogy to Understand Process & Thread**

* **Process** = A book (complete program)
* **Thread** = A page in that book (smaller part of the program working on a task)

Imagine you’re reading a book (process). You can focus on one page at a time (thread). If you have friends reading different pages simultaneously, it’s like multiple threads working inside one process.

**🔍 Key Differences:**

| **Feature** | **Process** | **Thread** |
| --- | --- | --- |
| Memory | Has its own separate memory | Shares memory with other threads |
| Communication | Needs complex methods (IPC) | Easy because of shared memory |
| Overhead | Higher (more resources needed) | Lower (lightweight) |
| Execution | Independent | Runs inside a process |

**📝 Summary:**

* A **process** is a full program running on your system.
* A **thread** is a smaller unit inside that process that can run tasks concurrently.
* Linux uses both processes and threads to run programs efficiently.

**16. Linux Grep commands?**Answer:-

**🔍 grep Command in Linux/Unix**

**📌 Basic Syntax:**

grep [options] pattern [file...]

**⚙️ Common grep Options:**

| **Option** | **Description** |
| --- | --- |
| -i | Case-insensitive match |
| -v | Invert match (show lines **not** containing the pattern) |
| -n | Show line numbers |
| -c | Count the number of matching lines |
| -l | List only filenames with matches |
| -h | Hide filenames in multi-file search |
| -o | Print only matched parts |
| -w | Match whole word only |
| -e | Match multiple patterns |
| -f | Take patterns from a file |
| -A n | Print n lines **after** the match |
| -B n | Print n lines **before** the match |
| -C n | Print n lines **before and after** the match |
| -R | Recursively search directories |
| -E | Use extended regular expressions (like egrep) |

**📁 Sample File (hellofile.txt):**

unix is great os. unix was developed in Bell labs.

learn operating system.

Unix linux which one you choose.

uNix is easy to learn.unix is a multiuser os.Learn unix .unix is a powerful.

**✅ Practical Examples:**

1. **Case-insensitive search**:

grep -i "unix" hellofile.txt

1. **Count matching lines**:

grep -c "unix" hellofile.txt

1. **List filenames with matches**:

grep -l "unix" \*.txt

1. **Match whole word only**:

grep -w "unix" hellofile.txt

1. **Print only the matched word**:

grep -o "unix" hellofile.txt

1. **Show line numbers with matches**:

grep -n "unix" hellofile.txt

1. **Show lines not containing the pattern**:

grep -v "unix" hellofile.txt

1. **Match lines starting with a string**:

grep "^unix" hellofile.txt

1. **Match lines ending with a string**:

grep "os$" hellofile.txt

1. **Match multiple patterns**:

grep -e "Agarwal" -e "Aggarwal" -e "Agrawal" hellofile.txt

1. **Read patterns from another file**:

grep -f pattern.txt hellofile.txt

1. **Print nearby lines**:

grep -A1 "learn" hellofile.txt # After

grep -B1 "learn" hellofile.txt # Before

grep -C1 "learn" hellofile.txt # Both before and after

1. **Recursive search in directory**:

grep -iR "hellos" /home/hellos

**17.Linux AWK commands..**Answer:-

**1. Print Entire File**

awk '{ print }' linux\_one/file1.txt

**2. Print Specific Columns (1st & 3rd)**

awk '{ print $1, $3 }' linux\_one/file1.txt

**3. Filter Lines Containing “error”**

awk '/error/ { print }' linux\_one/file1.txt

**4. Sum Values in a Column (e.g., 2nd column)**

awk '{ sum += $2 } END { print sum }' linux\_one/file1.txt

**5. Extract First 3 Characters of 1st Column**

awk '{ print substr($1, 1, 3) }' linux\_one/file1.txt

**6. Join Two Files by Common Column**

Example: file1.txt and file2.txt in linux\_one

awk 'NR==FNR {a[$1]=$2; next} { print $0, a[$1] }' linux\_one/file1.txt linux\_one/file2.txt

**7. Print First Column**

awk '{ print $1 }' linux\_one/file1.txt

**8. Calculate Average of a Column**

awk '{ sum += $1 } END { print sum / NR }' linux\_one/file1.txt

**9. Replace String in File**

awk '{ gsub("oldstring", "newstring") } 1' linux\_one/file1.txt

**10. Print Last Field**

awk '{ print $NF }' linux\_one/file1.txt

**11. Use Regular Expression Match**

awk '/pattern/ { print }' linux\_one/file1.txt

**12. Use Variables**

awk '{ total += $1 } END { print "Total:", total }' linux\_one/file1.txt

**13. Use Built-in Variables (e.g., comma-separated file)**

awk 'BEGIN { FS=","; OFS=" | " } { print $1, $2 }' linux\_one/file1.txt

**14. If-Else Condition**

awk '{ if ($1 > 50) print "Pass"; else print "Fail" }' linux\_one/file1.txt

**15. Define & Use a Function**

awk 'function square(x) { return x\*x } { print square($1) }' linux\_one/file1.txt

**18. How to check file access permission in Linux?**

Answer:- To **check file access permissions in Linux**, you can use the ls -l command. This command displays detailed information about files and directories, including their **permissions**.

**Syntax:**

ls -l filename

**Example:**

ls -l linux\_one/file1.txt

**Output Explained:**

-rw-r--r-- 1 user group 2048 May 29 10:00 file1.txt

The first part -rw-r--r-- shows the **file permissions**:

* - → File type (- for regular file, d for directory)
* rw- → Owner has **read (r)** and **write (w)** permission
* r-- → Group has **read (r)** permission only
* r-- → Others have **read (r)** permission only

**Permission Breakdown:**

| **Symbol** | **Meaning** |
| --- | --- |
| r | Read |
| w | Write |
| x | Execute |
| - | No permission |

**To list permissions of all files in a directory:**

ls -l linux\_one/

**19. What are the default permissions for a new file ?Plz find out for**

**Owner   → ?**

**Group → ?**

**All and others → ?**

Answer:-

By default, when a new file is created in Linux, the **permissions** are:

* **Owner → Read and Write (rw-)**
* **Group → Read (r--)**
* **Others → Read (r--)**

So the default permission looks like:

-rw-r--r--

Note: These defaults are affected by the **umask** setting, which subtracts permissions from the system defaults (666 for files).

**20. What is the command to change the permisssion to read only for the owner, group and all other users?**Answer:- To change the permission of a file to **read-only** for the **owner**, **group**, and **others**, use the following command:

chmod 444 filename

**Explanation:**

* 4 = Read permission
* 444 = Read for owner, group, and others
* This removes write and execute permissions for everyone.

**21. Can you change the file permissions to match the following:**

* **owner: Read and Write**
* **group: Read**
* **other: no permissions (None)**

Answer:- Yes, to set the permissions as:

* **Owner**: Read and Write → 6
* **Group**: Read → 4
* **Others**: No permission → 0

Use the following chmod command:

chmod 640 filename

This will give:

* Owner: rw-
* Group: r--
* Others: ---

**22. What was the command for changing teh file permissions to -rw-r-----?**Answer:- The command to change the file permissions to -rw-r----- is:

chmod 640 filename

This sets:

* **Owner**: read & write (rw-)
* **Group**: read only (r--)
* **Others**: no permissions (---)

**23. Change chmod.exercises permissions to -rwxr-x--x**

**Change the file permissions to match the following:**

**owner: Read, Write and Execute**

**group: Read and Execute**

**other: Execute**Answer:- **Owner** → Read, Write, Execute (rwx)

* **Group** → Read, Execute (r-x)
* **Others** → Execute only (--x)

That matches the permission string: -rwxr-x--x

**✅ Here’s the command you should run:**

chmod 751 linux\_one/file1.txt

**🔹 Explanation:**

* 7 → Owner: 4 (read) + 2 (write) + 1 (execute) = rwx
* 5 → Group: 4 (read) + 0 (write) + 1 (execute) = r-x
* 1 → Others: 0 (read) + 0 (write) + 1 (execute) = --x

Now, if you run:

ls -l linux\_one/file1.txt

You should see:

-rwxr-x--x

**24. What was the command for changing the file permissions to -rwxr-x--x**

Answer:-

The command to change the file permissions to **-rwxr-x--x** is:

chmod 751 filename

In your case, if the file is file1.txt inside the folder linux\_one, the command will be:

chmod 751 linux\_one/file1.txt

**25. what will this command do? chown -c master file1.txt**Answer:-

The command:

chown -c master file1.txt

does the following:

* **chown**: Changes the ownership of a file.
* **-c** (or --changes): Reports (prints) only when a change is actually made.
* **master**: The new owner username to assign to the file.
* **file1.txt**: The file whose ownership will be changed.

**So, this command changes the owner of file1.txt to the user master and outputs a message only if the ownership was changed. If the owner was already master, it won't print anything.**

**26. Can you define what is  a process ?**Answer:-

Yes! 👍

A **process** is a **running instance of a program**. When you start a program in Linux (like opening a text editor, running a script, or executing a command), the system creates a **process** to handle that task.

Each process has its own:

* **Process ID (PID)** – a unique number to identify it.
* **Memory** – it uses system memory (RAM) while running.
* **User** – the person who started the process.
* **State** – whether it's running, sleeping, stopped, or finished.

In simple words, a **process is what happens when a program is actively doing something** in your system.

You can view running processes using commands like:

ps

top

htop

**27. Can you list all the running processes?**

Answer:- Yes! To list all the running processes in Linux, you can use the following command:

ps -e

Or a more detailed version:

ps -ef

**Explanation:**

* ps – shows current running processes.
* -e – lists all processes.
* -f – provides full-format listing (user, PID, parent PID, etc.).

This command will display a list of all the running processes on your system.

**28. What will ps -f command do ? plz try n check .. ss required**.  
Answer:-  
The command ps -f displays a **full-format listing** of the currently running processes for the user. Here's what the output typically includes:

UID PID PPID C STIME TTY TIME CMD

sandbox 1 0 0 11:32 ? 00:00:00 tini -- python3 -m uvicorn ...

sandbox 3 1 28 11:32 ? 00:00:08 python3 -m uvicorn ...

sandbox 12 3 19 11:33 ? 00:00:04 /usr/local/bin/python3 ...

sandbox 56 3 20 11:33 ? 00:00:03 /usr/local/bin/python3 ...

sandbox 77 3 95 11:33 ? 00:00:01 /usr/local/bin/python3 ...

sandbox 85 12 99 11:33 ? 00:00:00 ps -f

**Columns explanation:**

* UID: User ID of the process owner.
* PID: Process ID.
* PPID: Parent Process ID.
* C: CPU utilization.
* STIME: Start time of the process.
* TTY: Terminal associated.
* TIME: CPU time consumed.
* CMD: Command that started the process.

This command is useful for seeing parent-child relationships and command details of processes.

**29.What is command to check foreground process and background process?**Answer:-  
To check **foreground** and **background** processes in Linux, you can use the following commands:

**✅ To check background jobs (processes you started in background):**

jobs

* Shows jobs that are running or stopped in the background of the current shell.

**✅ To check all foreground and background processes:**

ps

or for more detailed view:

ps -aux

* Lists **all** running processes including foreground and background across all users.

**✅ To check only your processes:**

ps -u your\_username

**✅ To see jobs with status (running/stopped):**

ps -ef

**30. Can you createa  a variable name with your name in it ?**Answer:-

Yes! You can create a variable in Linux with any valid name, including one that contains your name.

✅ Example:

If your name is Ranga, you can create a variable like this:

Ranga\_name="Hello, this is Ranga!"

To display the variable:

echo $Ranga\_name

This will output:

Hello, this is Ranga!

**31. Can you make the above name variable read only..**Answer:-  
Yes! You can make a variable **read-only** in Linux using the readonly command.

**✅ Example:**

Ranga\_name="Hello, this is Ranga!"

readonly Ranga\_name

Now, if you try to change the value:

Ranga\_name="Trying to change"

You’ll get an error like:

bash: Ranga\_name: readonly variable

This protects the variable from being modified later in the script or session.

**32.**

**Now will unset or delete the variables**

**Use the below command and check**

**Unset Name**

**Now check for**

**echo $Name   —> this should not print anything.. Plz try also specify the reason**Answer:-  
Sure! Here's how you can **unset or delete** a variable in Linux:

**✅ Command:**

unset Ranga\_name

**🔍 Now check:**

echo $Ranga\_name

**🧾 Output:**

It will not print anything (i.e., a **blank line**).

**📌 Reason:**

The unset command **removes the variable from memory**. After this, the shell no longer recognizes Ranga\_name as a variable, so echo $Ranga\_name returns nothing.

Note: If the variable was marked as readonly, you **cannot unset it**. In that case, you'll see an error like:

bash: unset: Ranga\_name: cannot unset: readonly variable

Let me know if you'd like to try with a different variable or need help handling readonly ones!

**33.Read n Know that**

**Variable Types**

**When a shell is running, three main types of variables are present ?**Answer

**🧪 1. Local Variables**

* These are defined in the current shell session.
* They are **not passed** to child processes.
* Typically used for temporary storage during script execution or shell use.

**Example:**

name="Ranga"

**🌍 2. Environment Variables**

* These are available system-wide or in the current shell and all its **child processes**.
* Used for configuration and application settings (like PATH, HOME, etc.).

**Example:**

export name="Ranga"

**🧱 3. Shell Variables**

* These are set and used by the shell itself (like BASH\_VERSION, UID, PWD, etc.).
* Some are predefined and control the behavior of the shell.

**Example:**

echo $BASH\_VERSION

echo $PWD

**34. CAn u try to add a list of your friends names in an array and try to printout**

**Ex:**

**NAME[0]="Ram"**

**NAME[1]="Sita"**

**NAME[2]="Tina"**

**NAME[3]="Veena"**

**NAME[4]="Tim"**

**echo "First Index: ${NAME[0]}"**

**echo "Second Index: ${NAME[1]}"**Answer:-

Sure! Here's how you can create an array of friends' names and print them using Bash in Linux:

# Creating an array of friends' names

FRIENDS[0]="Ravi"

FRIENDS[1]="Kiran"

FRIENDS[2]="Suma"

FRIENDS[3]="Amit"

FRIENDS[4]="Divya"

# Printing individual elements

echo "First Friend: ${FRIENDS[0]}"

echo "Second Friend: ${FRIENDS[1]}"

echo "Third Friend: ${FRIENDS[2]}"

echo "Fourth Friend: ${FRIENDS[3]}"

echo "Fifth Friend: ${FRIENDS[4]}"

If you want to print **all the names** at once:

echo "All Friends: ${FRIENDS[@]}"

**35. Can you print all the list at once in an array.. Try the below cmds and check**

**Echo “${array\_name[\*]}”**

**Echo “${array\_name[@]}”**

**Opetrators**

* **Arithmetic Operators**
* **Relational Operators**
* **Boolean Operators**
* **String Operators**
* **File Test Operators**

**If else**

**if...fi statement**

**if...else...fi statement**

**if...elif...else...fi statement**

**case...esac statement**

**The while loop**

**The for loop**

**The until loop**

**The select loop**

Answer:-  
Absolutely! Let's go step by step.

**✅ Printing all elements of an array at once in Linux Bash**

Assume you have an array like this:

FRIENDS=("Ravi" "Kiran" "Suma" "Amit" "Divya")

To print **all elements** at once:

echo "${FRIENDS[\*]}"

echo "${FRIENDS[@]}"

**✅ Output (Both commands give):**

Ravi Kiran Suma Amit Divya

**✅ Shell Operators Overview**

**➤ Arithmetic Operators (used with let, expr, (( )))**

| **Operator** | **Description** |
| --- | --- |
| + | Addition |
| - | Subtraction |
| \* | Multiplication |
| / | Division |
| % | Modulus |

**➤ Relational Operators (used in [ ... ])**

| **Operator** | **Meaning** |
| --- | --- |
| -eq | Equal |
| -ne | Not Equal |
| -gt | Greater Than |
| -lt | Less Than |
| -ge | Greater Than or Equal |
| -le | Less Than or Equal |

**➤ Boolean Operators**

| **Operator** | **Description** |
| --- | --- |
| ! | Logical NOT |
| -o | Logical OR |
| -a | Logical AND |

**➤ String Operators**

| **Operator** | **Description** |
| --- | --- |
| = or == | Equal |
| != | Not equal |
| -z | String is null (zero length) |
| -n | String is not null |

**➤ File Test Operators**

| **Operator** | **Tests if file...** |
| --- | --- |
| -e | exists |
| -f | is a regular file |
| -d | is a directory |
| -r | is readable |
| -w | is writable |
| -x | is executable |

**✅ Conditional Statements**

**if...fi statement:**

if [ condition ]

then

command

fi

**if...else...fi:**

if [ condition ]

then

command1

else

command2

fi

**if...elif...else...fi:**

if [ condition1 ]

then

command1

elif [ condition2 ]

then

command2

else

command3

fi

**case...esac:**

read -p "Enter a number: " num

case $num in

1) echo "One" ;;

2) echo "Two" ;;

\*) echo "Invalid" ;;

esac

**✅ Loops in Shell**

**➤ while loop**

count=1

while [ $count -le 5 ]

do

echo "Count: $count"

((count++))

done

**➤ for loop**

for name in "Ravi" "Kiran" "Suma"

do

echo "Friend: $name"

done

**➤ until loop**

x=1

until [ $x -gt 5 ]

do

echo "Number: $x"

((x++))

done

**➤ select loop (used for menus)**

select fruit in Apple Banana Mango Exit

do

case $fruit in

Exit) break ;;

\*) echo "You selected $fruit" ;;

esac

done

**36. Plz let me know whats the output of the below snippet:**

**a=0**

**while [ "$a" -lt 10 ]    # this is loop1**

**do**

**b="$a"**

**while [ "$b" -ge 0 ]  # this is loop2**

**do**

**echo -n "$b "**

**b=`expr $b - 1`**

**done**

**echo**

**a=`expr $a + 1`**Answer:-

Let's analyze the snippet step-by-step and then I'll tell you the output.

a=0

while [ "$a" -lt 10 ] # loop1: a from 0 to 9

do

b="$a"

while [ "$b" -ge 0 ] # loop2: b counts down from a to 0

do

echo -n "$b "

b=`expr $b - 1`

done

echo

a=`expr $a + 1`

done

**What happens?**

* Outer loop (a) runs from 0 to 9.
* For each a, inner loop (b) counts down from a to 0, printing numbers on the same line.
* After the inner loop finishes, it prints a newline.

**Output:**

0

1 0

2 1 0

3 2 1 0

4 3 2 1 0

5 4 3 2 1 0

6 5 4 3 2 1 0

7 6 5 4 3 2 1 0

8 7 6 5 4 3 2 1 0

9 8 7 6 5 4 3 2 1 0

Explanation:

* For a=0, inner loop prints 0
* For a=1, inner loop prints 1 0
* For a=2, inner loop prints 2 1 0
* ...
* For a=9, inner loop prints 9 8 7 6 5 4 3 2 1 0