| Sr.No. | Topics | Duration(Mins) | Session No(2 Hours) | Session No.(4 Hours) |
|---|---|---|---|---|
| 1 | Overview of Linux | 20 | 1 | 1 |
| 2 | Linux Architecture | 30 | 1 | 1 |
| 3 | Linux Distribution | 20 | 1 | 1 |
| 4 | Basic Linux Commands | 70 | 2 | 1 |
| 5 | File Permission Management | 50 | 2 | 1 |
| 6 | User Creation | 40 | 2 | 1 |
| 7 | Shell Scripts | 80 | 3 | 2 |
| 8 | SSH and VI Utility | 50 | 3 | 2 |
| 9 | Practical | 60 | 4 | 2 |

Linux Fundamentals

1. Overview of Linux

   Information

- Linux is an open-source operating system kernel that serves as the foundation for numerous Linux-based operating systems, known as distributions or distros.
- It was created by Linus Torvalds in 1991 and has since become one of the most popular operating systems in the world.

1. Open Source:
- Linux is an open-source operating system, which means its source code is freely available to the public.
- This allows users to view, modify, and distribute the code, promoting collaboration and innovation.

2. Unix-like System:
- Linux is inspired by the Unix operating system and follows similar design principles.
- It provides a multi-user, multitasking environment with a command-line interface.

3. Distributions:

- Linux is distributed in various flavours known as distributions or distros.
- Examples include Ubuntu, Fedora, CentOS, Debian, and many more.
- Each distribution offers a different set of pre-configured software packages and user interfaces to cater to specific needs and preferences.

4. Command-Line Interface (CLI):

- Linux is renowned for its powerful command-line interface, where users can interact with the system by typing commands.
- The CLI provides extensive control over the operating system and is favored by advanced users and system administrators for its flexibility and automation capabilities.

5. Graphical User Interface (GUI):

- Linux also supports a range of graphical user interfaces, allowing users to interact with the system using windows, icons, and menus.
- Popular GUI environments include GNOME, KDE, XFCE, and Unity.

6. Stability and Reliability:

- Linux is known for its stability and reliability.
- It is widely used in servers, supercomputers, and embedded systems due to its ability to handle high

workloads, support multiple users, and operate continuously for long durations without issues.

7. Security:

- Linux has a strong focus on security.
- With its open-source nature, vulnerabilities can be quickly identified and patched by the community.
- Additionally, Linux offers robust access controls, permissions management, and encryption capabilities, making it highly secure.

8. Software Package Management:

- Linux distributions provide package management systems that allow users to easily install, update, and remove software packages.
- Package managers handle dependencies, ensuring that all required components are installed correctly.

9. Customizability:

- Linux offers a high degree of customizability, allowing users to configure the system to suit their specific needs.
- Users can modify the kernel, choose from a wide range of software applications, and customize the user interface to create a personalized computing environment.

- Linux's versatility, stability, security, and vast array of available software make it a popular choice for various use cases, including desktop computing, server hosting, cloud infrastructure, IoT devices, and more.
- Its open-source nature fosters collaboration and innovation, driving the continuous development and improvement of the Linux ecosystem.

Explain Unix/Linux Operating System :

- The Unix/Linux operating system is a family of multitasking, multiuser computer operating systems that are widely used in both server and desktop environments.
- Unix was originally developed in the 1970s at Bell Labs by Ken Thompson, Dennis Ritchie, and others, while Linux, which is a Unix-like operating system, was created in the early 1990s by Linus Torvalds.

- Key features of Unix/Linux operating systems include:

a. Multitasking and Multiuser:

- Unix/Linux systems are designed to allow multiple processes or users to run concurrently.
- This means that multiple programs or tasks can be executed simultaneously, and multiple users can access and use the system at the same time.

b. Shell:

- Unix/Linux operating systems provide a command-line interface called a shell.
- The shell allows users to interact with the system by typing commands and executing them.
- Common Unix/Linux shells include Bash, Csh, and Ksh.

c. File System:

- Unix/Linux systems use a hierarchical file system that organizes files and directories in a tree-like structure.
- Directories can contain both files and other directories, allowing for organization and easy navigation.

d. Networking Capabilities:

- Unix/Linux systems have built-in networking capabilities, allowing for communication between different systems over local networks or the internet.
- This enables features such as remote logins, file sharing, and network services.

e. Security:

- Unix/Linux systems have robust security features built into their design.

- Access to files, directories, and system resources is controlled through permissions, ensuring that only authorized users or processes can access sensitive information.

f. Portability:

- Unix/Linux operating systems are highly portable and can run on a wide range of hardware architectures.
- This portability has contributed to their popularity and widespread adoption across various devices, from servers to embedded systems and smartphones.

g. Open Source:

- Linux, in particular, is known for being an open-source operating system, meaning that its source code is freely available and can be modified and distributed by anyone.
- This has fostered a large and active community of developers who contribute to its development and support.

- Unix/Linux operating systems have been widely adopted due to their stability, security, and flexibility.
- They are used in various domains, including servers, supercomputers, embedded systems, and personal computers.

- Unix/Linux also serves as the foundation for many other operating systems, including macOS, Android, and iOS.

Evolution of Linux :

- The evolution of Linux can be traced back to the early 1990s when Linus Torvalds, a Finnish computer science student, started developing an operating system kernel as a hobby project.

I. Creation of Linux Kernel (1991):

- In 1991, Linus Torvalds released the first version of the Linux kernel, which was initially developed for personal use.
- He shared the source code with the online community, which led to collaborative development and rapid improvements.

II. GNU/Linux Collaboration (1992):

- The Free Software Foundation's GNU Project had been developing a complete free software operating system called GNU, but it lacked a functional kernel.
- The Linux kernel, combined with GNU software utilities and libraries, formed the basis for the GNU/Linux operating system.

III. Rise of Distributions (mid-1990s):

- As Linux gained popularity, individuals and organizations started creating distributions (distros) that packaged the Linux kernel with additional software, drivers, and user-friendly installation tools.

- Popular early distributions include Slackware, Debian, and Red Hat.

IV. Advancements in Desktop Environments (late 1990s):
   - During the late 1990s, Linux saw significant advancements in graphical desktop environments. Projects like KDE (K Desktop Environment) and GNOME (GNU Network Object Model Environment) provided user-friendly interfaces and productivity tools, making Linux more accessible to a wider audience.

V. Enterprise Adoption (early 2000s):
   - Linux began gaining traction in enterprise environments due to its stability, security, and cost-effectiveness.
   - Companies like IBM, Red Hat, and Novell started offering enterprise-focused distributions and support services, further accelerating its adoption.

VI. Android (2008):
   - Google adopted the Linux kernel as the foundation for its Android operating system.
   - Android's success in the mobile market propelled Linux into widespread use on smartphones and other mobile devices, significantly expanding its user base.

VII. Containerization and Cloud Computing (2010s):
   - The rise of containerization technologies like Docker, along with the growth of cloud computing, further boosted the popularity of Linux.

- Linux became the de facto operating system for cloud platforms, providing the foundation for scalable, distributed applications and infrastructure.

VIII. Linux on Embedded Systems and IoT:

- Linux's versatility and small footprint made it an ideal choice for embedded systems and Internet of Things (IoT) devices.
- Linux-based distributions like Embedded Linux and Yocto Project gained popularity, enabling Linux to power a wide range of embedded devices, including smart appliances, industrial systems, and consumer electronics.

IX. Continuous Development and Innovation:

- Linux continues to evolve through the collaborative efforts of the open-source community.
- The kernel is regularly updated with new features, performance improvements, and security patches.
- Additionally, advancements in areas like containerization, artificial intelligence, and edge computing have contributed to Linux's ongoing growth and relevance.

- Today, Linux is widely used in various domains, including servers, desktops, mobile devices, embedded systems, and cloud infrastructure.
- It remains an open-source project driven by a global community of developers, contributors, and organizations, ensuring its continuous

development and adaptation to changing technology trends.

## 2. Linux Architecture

Information

The architecture of Linux can be divided into four main components:

I. Kernel:

- The Linux kernel is the core of the operating system.
- It interacts directly with the hardware and provides essential services such as process management, memory management, device drivers, file system access, and networking.
- The kernel is responsible for resource allocation and ensuring that different processes can run simultaneously without interfering with each other.

II. System Libraries:

- System libraries are a collection of pre-compiled code and functions that provide APIs (Application Programming Interfaces) for developers to access various system functionalities.

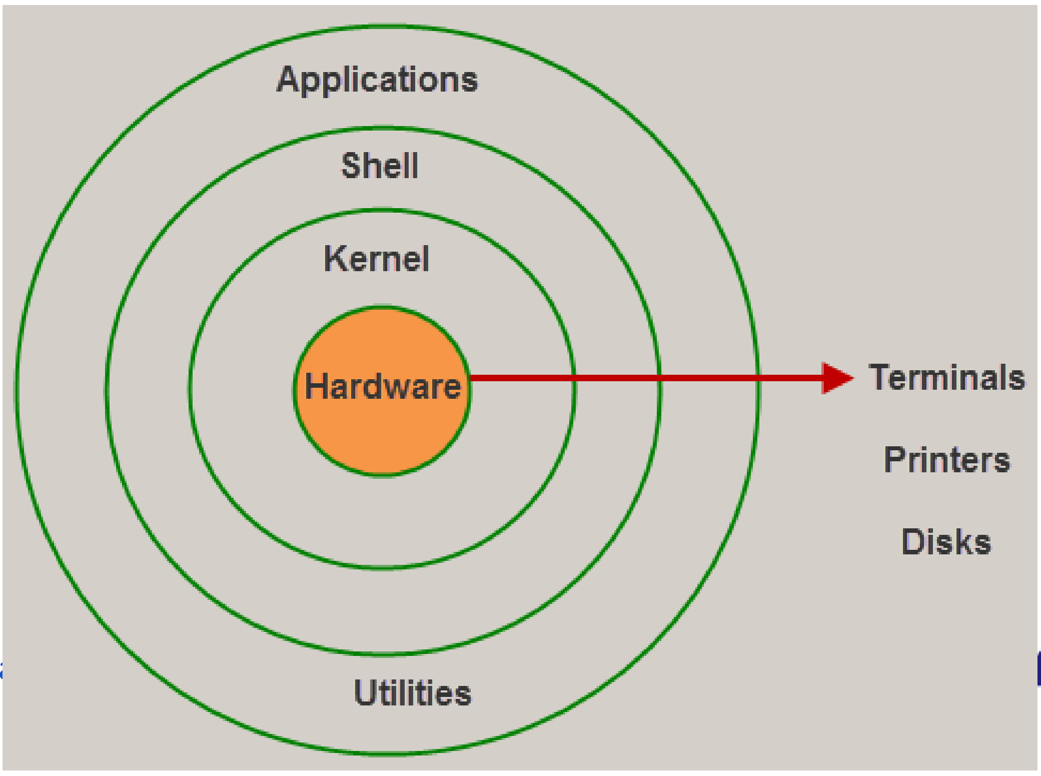
- These libraries include the C library (libc), which provides low-level programming interfaces, as well as other libraries like libpthread for thread management, libm for mathematical operations, and libssl for secure communication.

III. System Utilities:
- System utilities are command-line tools and programs that facilitate system administration and management.
- They include tools for managing processes (such as ps and top), handling file systems (such as ls and cp), configuring network settings (such as ifconfig and ip), managing users and permissions (such as useradd and chmod), and performing system diagnostics (such as dmesg and fsck).

IV. Graphical User Interface (GUI):
- The GUI layer provides a visual interface for users to interact with the system.
- Linux supports various desktop environments, such as GNOME, KDE, Xfce, and LXDE, which provide a graphical desktop environment, window manager, and a set of applications.
- The GUI layer uses the X Window System (X11) or Wayland display server protocols to handle graphics rendering and user input.

- In addition to these main components, Linux also has a wide range of software applications and services that run on top of the architecture.
- These include web servers, databases, programming languages, development tools, office suites, and multimedia applications.
- The open-source nature of Linux allows for flexibility and customization, with the ability to modify and extend the system to meet specific requirements.

- Overall, the architecture of Linux is designed to provide a stable, secure, and efficient operating system that can run on a variety of hardware platforms, from embedded systems to servers and personal computers.
- The modular design and robust kernel enable Linux to handle diverse workloads and adapt to evolving computing needs.



## 3. Linux Distribution Information

Linux distributions, often referred to as distros, are different variations or flavors of the Linux operating system.
They are created by taking the Linux kernel and combining it with various software packages, libraries, utilities, and desktop environments to provide a complete operating system experience.

i. Ubuntu:

- Ubuntu is one of the most widely used Linux distributions, known for its user-friendly interface and focus on simplicity.
- It is based on the Debian distribution and offers regular releases, long-term support (LTS) versions, and a vast software repository.

ii. Fedora:

- Fedora is a community-driven Linux distribution sponsored by Red Hat.

- It focuses on innovation, rapid releases, and staying on the cutting edge of technology.
- Fedora emphasizes open-source software and is often used by developers and enthusiasts.

iii. Debian:

- Debian is one of the oldest and most influential Linux distributions.
- It follows a strict commitment to free software principles and is known for its stability, reliability, and extensive package management system.
- Debian serves as the foundation for many other Linux distributions.

iv. CentOS:

- CentOS is a community-driven distribution that aims to provide a free, enterprise-class, and stable operating system.
- It is based on the source code of Red Hat Enterprise Linux (RHEL) and is known for its long-term support and compatibility with RHEL.

v. Arch Linux:

- Arch Linux is a lightweight and flexible distribution that focuses on simplicity, minimalism, and customization.
- It follows a rolling release model, which means users continuously receive the latest updates and software versions.

vi. openSUSE:

- openSUSE is a community-driven distribution that offers both stable releases and a rolling release version called "Tumbleweed."
- It provides a user-friendly experience, powerful administration tools, and support for various desktop environments.

vii. Linux Mint:

- Linux Mint is a distribution known for its focus on providing a user-friendly experience, similar to the Windows operating system.
- It offers several desktop environment options, including Cinnamon, MATE, and Xfce, and comes with pre-installed multimedia codecs.

viii. Gentoo:

- Gentoo is a highly customizable distribution that emphasizes performance optimization and customization.
- It uses a package management system called Portage, which allows users to compile software from source code to suit their specific hardware and requirements.

- These are just a few examples of the numerous Linux distributions available.
- Each distribution has its own characteristics, package management system, default desktop environment, and target audience.

- Users can choose a distribution based on factors such as ease of use, stability, specific software requirements, or personal preferences.

Drawback of Linux :

- While Linux is a powerful and widely used open-source operating system, it also has some drawbacks and challenges.
- It's important to consider these aspects along with its benefits when choosing an operating system for your needs.
- Here are some drawbacks of Linux:

1. Software and Driver Compatibility:

- Linux may face compatibility issues with certain proprietary software and hardware drivers.
- Some software applications and peripherals are designed primarily for Windows or macOS, which might not have Linux versions or well-supported drivers.

2. Learning Curve:

- For users who are accustomed to Windows or macOS, the transition to Linux might involve a learning curve.
- Command-line operations and the different user interfaces could be unfamiliar, requiring some time to become proficient.

3. Limited Commercial Support for Desktop Users:

- While there is strong community support for Linux, commercial support options can be limited for individual desktop users.
- Enterprises can often find commercial support, but individual users may have fewer options for professional assistance.

4. Gaming Support:

- Though gaming on Linux has improved over the years, many games are still primarily developed for Windows.
- While tools like Steam for Linux have expanded the gaming library, game developers might not prioritize Linux compatibility.

5. Fragmentation:

- Linux distributions (distros) come in various flavors, which can lead to fragmentation.
- Each distro has its own package management, software availability, and user interface, potentially causing confusion for newcomers.

6. Lack of Standardization:

- While standards exist in the Linux community, the lack of strict central control can result in differences in software versions, libraries, and system configurations across different distributions.

7. Desktop Software Availability:

- While Linux offers a plethora of open-source software, specific commercial software packages may not have Linux versions.
- This can be a limitation for users who rely on certain proprietary tools.

8. Lack of Vendor Support:

- Some hardware vendors might not provide Linux drivers or official support for their products, which can lead to compatibility issues or reduced functionality.

9. Command Line:

- While the command line can be powerful and efficient, it might be intimidating for users who are used to graphical interfaces.
- Some tasks are more convenient to perform using a GUI.

10. Enterprise Software:

- While Linux is widely used in server environments, certain enterprise-level software packages, especially those specialized for particular industries, may not have Linux compatibility.

- Despite these drawbacks, many individuals, businesses, and organizations find Linux to be a reliable and versatile operating system that fits their needs.
- Linux's strengths in terms of security, customization, open-source community, and compatibility with web technologies often outweigh its drawbacks for those who choose to adopt it.

## 4. Basic Linux Commands Information

- Here are some basic Linux commands that are commonly used for navigating the file system, managing files and directories, and executing tasks:

a. 'ls': Lists the contents of a directory.
- Example: ls or ls -l (to display detailed information)

b. 'cd': Changes the current directory.
- Example: cd /path/to/directory or cd .. (to go up one level)

c. 'pwd': Prints the current working directory.

d. 'mkdir': Creates a new directory.
- Example: mkdir directory_name

e. 'rm': Removes files or directories.

- Example: rm file_name or rm -r directory_name (to remove a directory)

f. 'cp': Copies files and directories.
- Example: cp file_name destination or cp -r directory_name destination

g. 'mv': Moves or renames files and directories.
- Example: mv file_name new_file_name or mv file_name directory_name

h. 'cat': Displays the contents of a file.
- Example: cat file_name

i. less: Views the contents of a file page by page.
- Example: less file_name

j. grep: Searches for a pattern in files.
- Example: grep pattern file_name

k. 'chmod': Changes the permissions of a file or directory.
- Example: chmod permissions file_name (e.g., chmod +x script.sh to make a script executable)

l. ‘chown’: Changes the ownership of a file or directory.

- Example: chown user:group file_name

m. ‘sudo’: Executes a command with superuser (root) privileges.

- Example: sudo command_name

n. ‘man’: Displays the manual page of a command.

- Example: man command_name (e.g., man ls to view the manual for the ls command)

- These are just a few basic Linux commands to get started.
- There are many more commands available for various purposes, and each command often has additional options and arguments.
- You can explore the man pages or online resources to learn more about specific commands and their usage.

## 5. File Permission Management

Information

- File permission management in Linux is essential for controlling access to files and directories.
- Permissions determine who can read, write, or execute a file.

a. Permission Types:

- There are three basic permission types for files and directories: read (r), write (w), and execute (x).
- Read permission allows viewing the content of a file or the list of files in a directory.
- Write permission allows modifying a file or creating/deleting files in a directory.
- Execute permission allows executing a file or accessing files within a directory.

b. Permission Groups:

- Permissions are assigned to three groups: owner, group, and others.
- The owner is the user who created the file, the group represents a collection of users, and others include any user who is not the owner or in the group.

c. Symbolic Representation:

- File permissions can be represented symbolically using a combination of letters (r, w, x) and symbols (+, -, =).
- For example, rwx represents read, write, and execute permissions, while - represents no permission.
- The symbolic representation follows the order of owner-group-others, such as rwxr--r-- means the owner has read, write, and execute permissions, the group has read-only permission, and others have read-only permission.

d. Numeric Representation:

- File permissions can also be represented numerically using a three-digit number.
- Each digit represents the permission for owner-group-others, respectively.
- Each permission (r, w, x) is assigned a value: read (4), write (2), and execute (1).
- The sum of the values determines the permission level.

- For example, 644 means the owner has read and write permissions (4+2=6), and the group and others have read-only permission (4).

e. Changing Permissions:

- File permissions can be changed using the 'chmod' command.
- The 'chmod' command allows modifying permissions by specifying the desired permissions using symbolic or numeric representation.
- For example, chmod +x script.sh adds execute permission to the file script.sh, and chmod 755 directory sets read, write, and execute permissions for the owner, and read and execute permissions for the group and others on the directory.

f. Recursive Permission Changes:

- The 'chmod' command can be used with the -R option to change permissions recursively for all files and directories within a directory.
- This is useful when you want to apply the same permissions to all files and subdirectories.

g. Ownership Management:

- File ownership can be managed using the chown and chgrp commands.
- chown allows changing the owner of a file, while chgrp allows changing the group ownership.

- These commands are often used with the -R option for recursive changes.

- File permission management is crucial for maintaining the security and integrity of files and directories in a Linux system.
- By setting appropriate permissions, you can control access to sensitive files and ensure that only authorized users have the necessary privileges to read, write, or execute them.

6. User Creation

Information

- To create a user in a Linux system, you can follow these steps:

a. Open a terminal:

- Launch a terminal or connect to the Linux system via SSH.

b. Switch to root/superuser:

- To create a user, you'll need administrative privileges.
- Switch to the root user or use the ‘sudo’ command before each command to run them with root privileges.

c. Run the ‘useradd’ command:

- Use the useradd command followed by the username to create a new user.
- For example, to create a user named "john":

```
useradd john
```

- By default, this will create a new user with the same username as their home directory located in the /home directory.

d. Set a password for the user:

- Use the ‘passwd’ command to set a password for the new user.
- For example:

```
passwd john
```

- You will be prompted to enter and confirm the password for the user.

e. Optional:

i.Add additional user information:

- You can add additional information about the user, such as their full name, phone number, or description, using the ‘usermod’ command.
- For example, to add a full name for the user "john":

```
usermod -c "John Doe" john
```

ii. Verify the user creation:

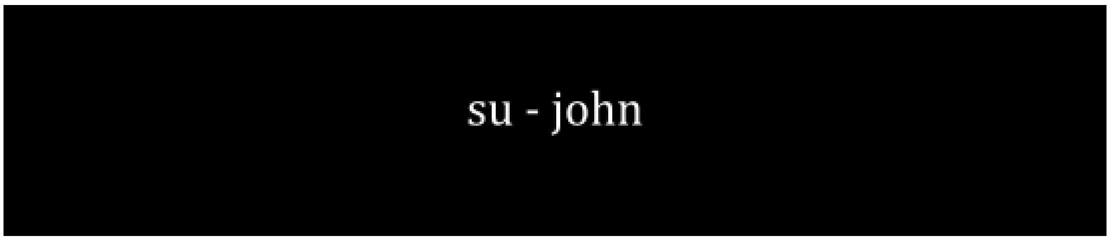
- You can use the id command to verify that the user has been created and assigned a unique user ID (UID) and group ID (GID).
- For example:

```
id john
```

- This will display information about the user, including the UID, GID, and associated groups.

iii. Switch to the new user:

- You can switch to the newly created user using the ‘su’ command.
- For example:



```
su - john
```

- This will switch the current terminal session to the user "john," allowing you to perform actions as that user.

- That's it! You have successfully created a new user in the Linux system.
- The user can now log in with their username and password and perform actions based on the permissions assigned to them.

## 7. Shell Script

### Information

- Shell scripts are plain text files containing a sequence of commands that can be executed by the shell (command-line interpreter) in a Unix-like operating system.
- They allow you to automate tasks, perform system administration tasks, and run a series of commands without manual intervention.

### a. Script File Extension:

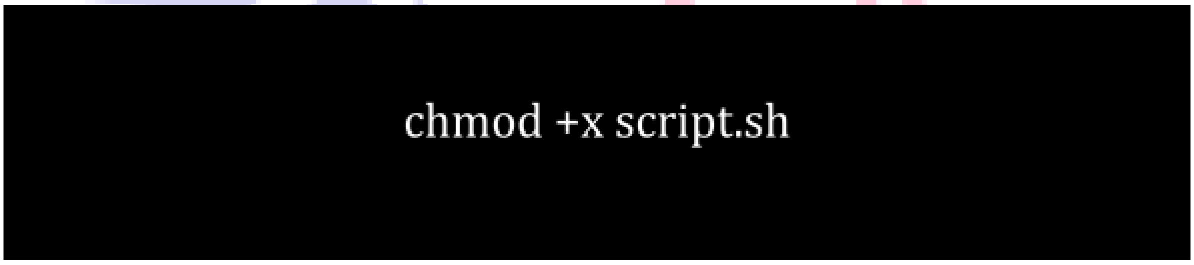
- Shell scripts typically have a file extension such as '.sh' to indicate that they are shell scripts.
- However, this extension is not mandatory, and any plain text file with the appropriate script contents can be executed as a shell script.

b. Shebang Line:

- The first line of a shell script starts with a shebang (#!) followed by the path to the shell interpreter to be used.
- For example, '#!/bin/bash' specifies that the Bash shell should be used to interpret the script.

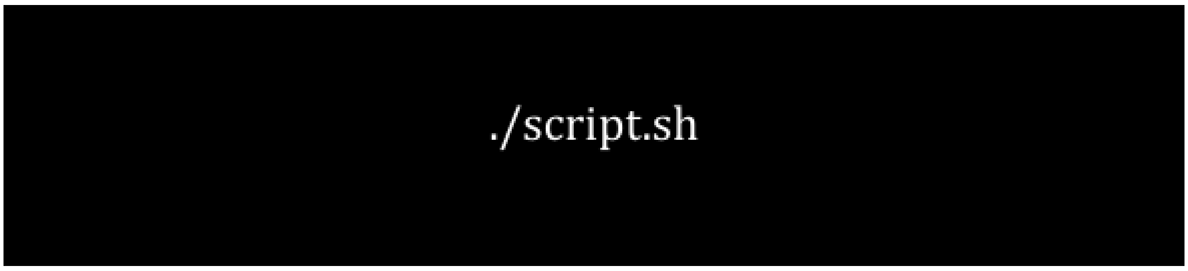
c. Script Execution:

- To execute a shell script, you need to make it executable first.
- You can use the 'chmod' command to set the execute permission.
- For example:



```
chmod +x script.sh
```

- After making it executable, you can run the script using its file name:



```
./script.sh
```

d. Comments:

- Comments in shell scripts provide explanations and documentation.
- They start with the ‘#’ symbol and are ignored by the shell.
- You can use comments to add notes, describe the purpose of the script, or provide usage instructions.

e. Variables:

- Shell scripts allow you to define and use variables.
- Variables can store values that can be accessed and manipulated within the script.
- Variables are typically assigned values using the ‘=’ operator.
- For example:

```
name="John"
age=30
```

- You can reference variables by prefixing them with the $ symbol.
- For example

```
echo "My name is $name and I am $age years old."
```

f. Control Flow:

- Shell scripts support various control flow structures, such as conditionals (if-else statements), loops (for and while), and function definitions.
- These structures allow you to make decisions, repeat actions, and organize your script's logic.

g. Command Substitution:

- Shell scripts can execute commands and capture their output using command substitution.
- Command substitution is done by enclosing the command within $() or backticks ( ).
- For example:

```
current_date=$(date +%Y-%m-%d)
```

  - This captures the current date in the current_date variable.

h. Input and Output:

  - Shell scripts can read input from the user or files using the read command or command-line arguments.

- They can also display output to the terminal using the echo command or redirect output to files.

i. File Operations:

- Shell scripts can perform file operations such as creating, deleting, renaming, and copying files and directories using commands like ‘mkdir’, ‘rm’, ‘mv’, and ‘cp’.

j. Error Handling:

- Shell scripts can handle errors and respond accordingly using conditional statements or by checking the exit codes of commands using the ‘$?’ variable.

- Shell scripts provide a powerful and flexible way to automate tasks and manage system administration in Linux.
- With the ability to combine commands, control flow structures, variables, and input/output operations, you can create complex scripts to accomplish various tasks efficiently.

Types of Shell Script :

- There are several types of shell scripts based on the shell interpreter used to execute them.
- The most common types include:

i.Bash Scripts:

- Bash (Bourne Again SHell) is the default shell in most Linux distributions.
- Bash scripts are written in the Bash scripting language and typically have the '.sh' file extension.
- They are widely used for automation, system administration, and general-purpose scripting.

ii. sh Scripts:

- The sh shell, or Bourne shell, is one of the original Unix shells.
- sh scripts are compatible with various Unix-like operating systems and provide basic scripting capabilities.
- They may have the '.sh' file extension, similar to Bash scripts.

iii. Korn Shell (ksh) Scripts:

- Korn Shell is an extended version of the Bourne shell with additional features and improvements.
- ksh scripts are written in the Korn Shell scripting language and can be used on Unix-like systems.
- They often have the '.ksh' or '.sh' file extension.

iv. C Shell (csh) Scripts:

- The C Shell is another popular Unix shell known for its C-like syntax.
- csh scripts are written in the C Shell scripting language and are mainly used for interactive shell scripting.
- They typically have the '.csh' or '.sh' file extension.

v. Zsh Scripts:

- Zsh (Z Shell) is a powerful and highly customizable shell with features from Bash, Korn Shell, and others.
- Zsh scripts are written in the Zsh scripting language and provide advanced scripting capabilities.
- They often have the '.zsh' or '.sh' file extension.

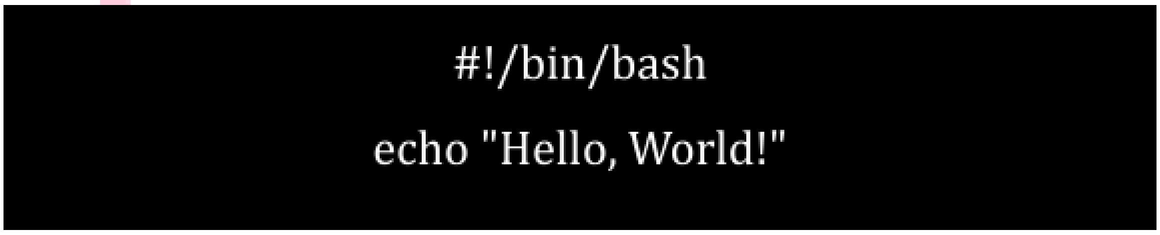
vi. Dash Scripts:

- Dash (Debian Almquist Shell) is a minimalistic shell designed for efficient execution and compatibility with POSIX standards.
- Dash scripts are written in the Dash scripting language and are commonly used for scripting on Debian-based systems.
- They may have the '.sh' file extension.

- It's worth noting that many modern Linux distributions use Bash as the default shell and support a wide range of scripting languages.
- However, depending on the specific requirements and compatibility of the target system, you may choose a particular shell or scripting language for your scripts.

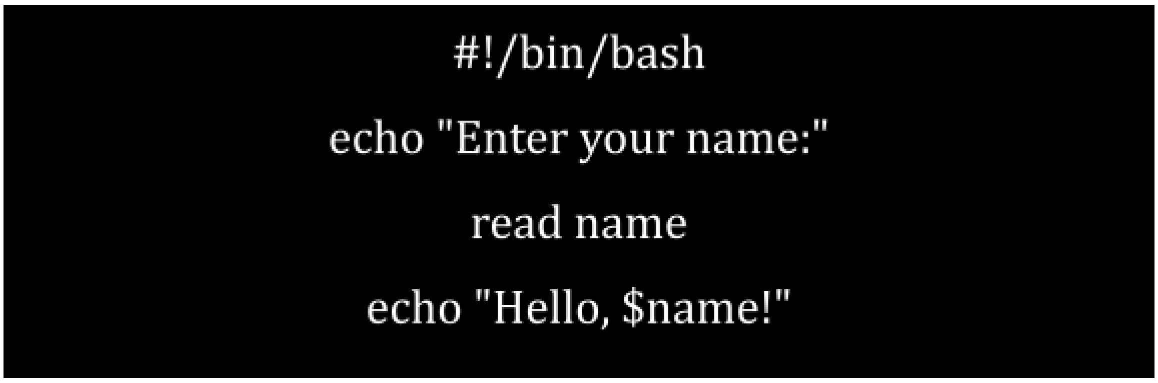
Writing various types of Shell Script Programs:

a. Hello World:



```
#!/bin/bash
echo "Hello, World!"
```

b. User Input and Output:



```
#!/bin/bash
echo "Enter your name:"
read name
echo "Hello, $name!"
```

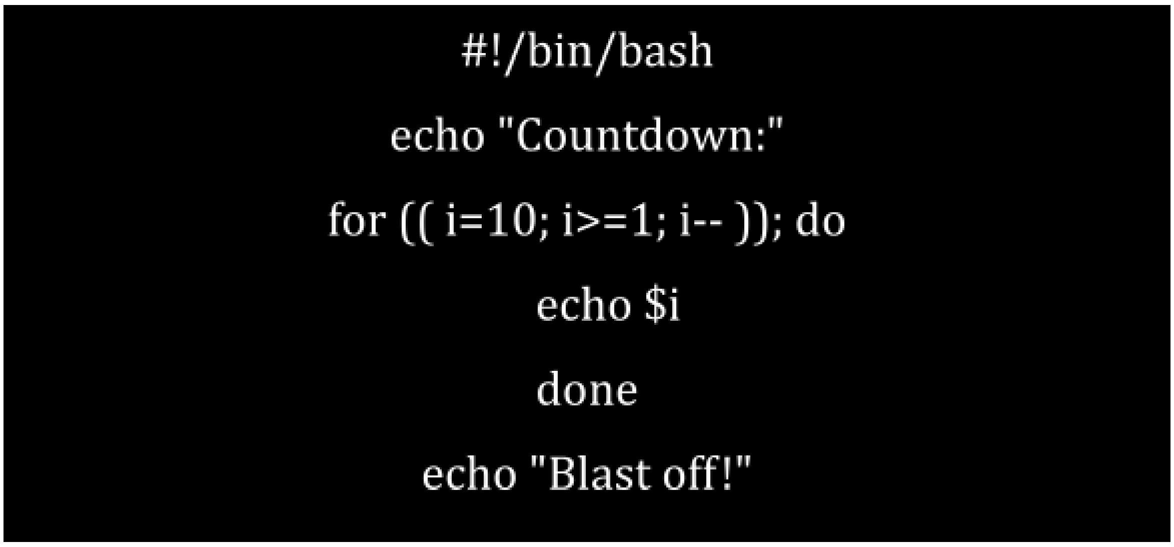
c. File Manipulation :

```
#!/bin/bash
echo "Enter a file name:"
read filename
touch $filename
echo "File created: $filename"
```

d. Arithmetic Calculation:

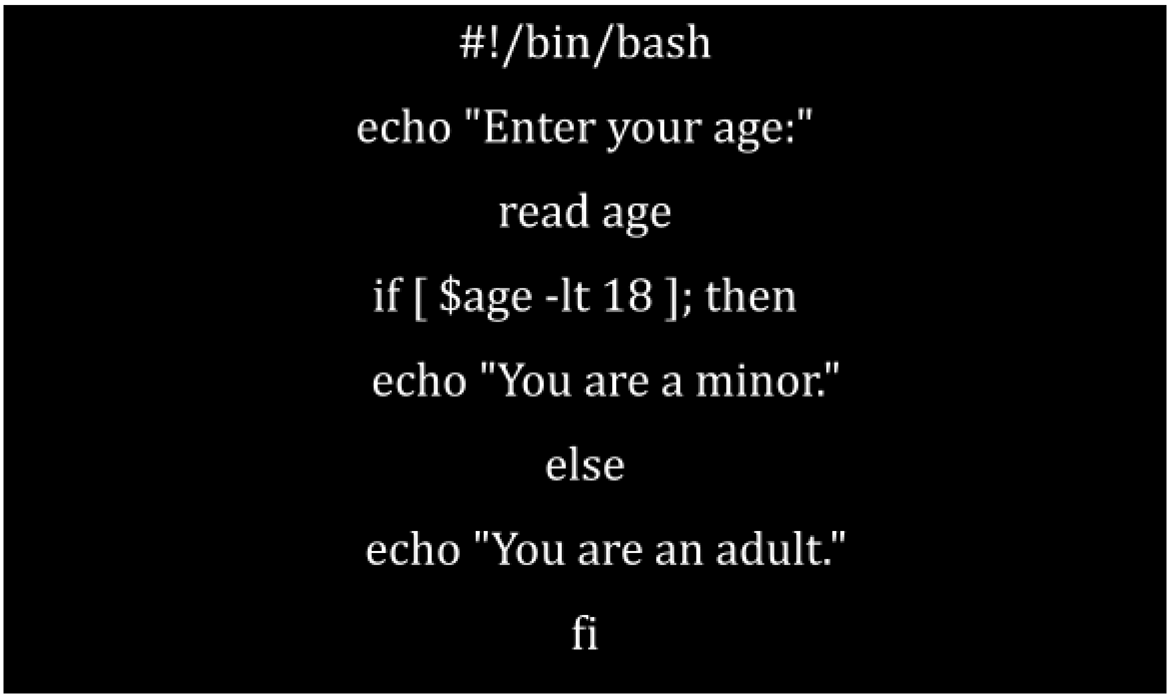
```
#!/bin/bash
echo "Enter two numbers:"
read num1
read num2
sum=$((num1 + num2))
echo "Sum: $sum"
```

e. Looping :



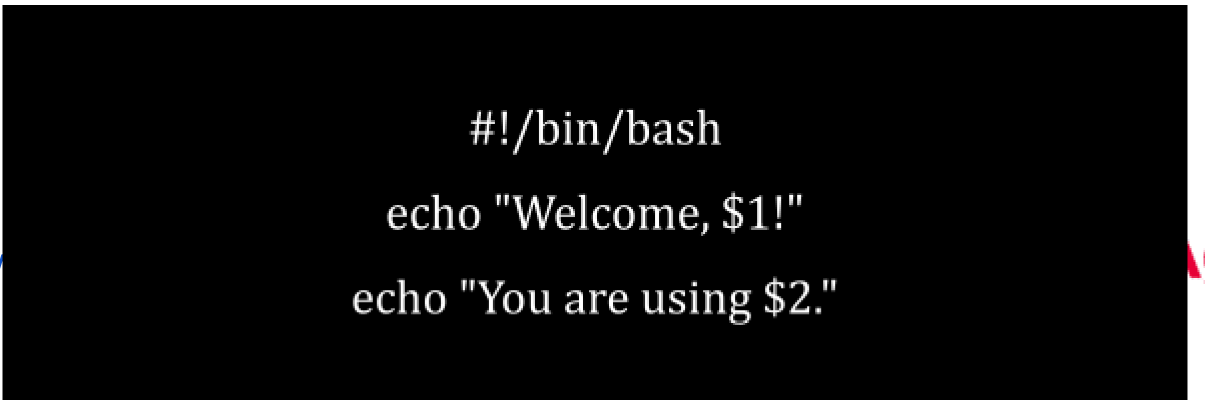
```
#!/bin/bash
echo "Countdown:"
for (( i=10; i>=1; i-- )); do
echo $i
done
echo "Blast off!"
```

f. Conditional :



```
#!/bin/bash
echo "Enter your age:"
read age
if [ $age -lt 18 ]; then
echo "You are a minor."
else
echo "You are an adult."
fi
```

g. Command Line Arguments :



```
#!/bin/bash
echo "Welcome, $1!"
echo "You are using $2."
```

h. Functions :

```
#!/bin/bash
greet() {
echo "Hello, $1!"
}
greet "John"
```

```
#!/bin/bash
echo "Enter a message:"
read message
echo $message > output.txt
echo "Message written to file."
```

j. Error Handling

```
#!/bin/bash
command_that_may_fail
if [ $? -eq 0 ]; then
echo "Command executed successfully."
```

- These are just a few examples to get you started.
- You can explore more complex shell script programs by combining different concepts, such as string manipulation, regular expressions, conditional loops, and interacting with system commands.
- The possibilities are endless, and shell scripting provides a powerful toolset for automating tasks and managing systems.

Conditional Statement:

- Conditional statements in programming allow you to make decisions and execute different blocks of code based on certain conditions.
- In shell scripting, there are several conditional statements available.
- The most commonly used ones are:

a. if statement:

- The if statement allows you to perform a specific action if a condition is true.
- The basic syntax is:

```
if [ condition ]; then
# code to execute if the condition is true
fi
```

- Example:

```
age=25
if [ $age -gt 18 ]; then
echo "You are an adult."
fi
```

b. if-else statement:

- The if-else statement allows you to execute one block of code if a condition is true and another block if the condition is false.
- The syntax is:

```
if [ condition ]; then
# code to execute if the condition is true
else
# code to execute if the condition is false
fi
```

- Example :

```
age=15
if [ $age -gt 18 ]; then
echo "You are an adult."
else
echo "You are a minor."
fi
```

c. elif statement:

- The elif statement allows you to check additional conditions when the initial condition is false.
- It is used in combination with the if statement and can have multiple elif blocks.
- The syntax is:

```
if [ condition1 ]; then
# code to execute if condition1 is true
elif [ condition2 ]; then
# code to execute if condition2 is true
else
# code to execute if all conditions are false
fi
```

- Example :

```
age=65
if [ $age -lt 18 ]; then
echo "You are a minor."
elif [ $age -ge 18 ] && [ $age -lt 60 ]; then
echo "You are an adult."
else
echo "You are a senior citizen."
fi
```

- These conditional statements allow you to control the flow of your shell script based on specific conditions.

- You can use comparison operators (‘-eq’, ‘-ne’, ‘-lt’, ‘-gt’, ‘-le’, ‘-ge’) to evaluate conditions, combine conditions using logical operators (‘&&’, ‘||’), and use variables or command output within the conditions.

Looping Statement :

- In shell scripting, looping statements allow you to repeat a set of commands or actions multiple times.
- There are mainly two types of loops available: for loop and while loop.

a. for Loop:

- The for loop iterates over a list of values or elements and executes a set of commands for each iteration.
- The basic syntax of a for loop is:

```
for variable in list
do
# commands to execute
done
```

Example :

```
#!/bin/bash
for i in 1 2 3 4 5
do
echo "Value: $i"
done
```

Output :

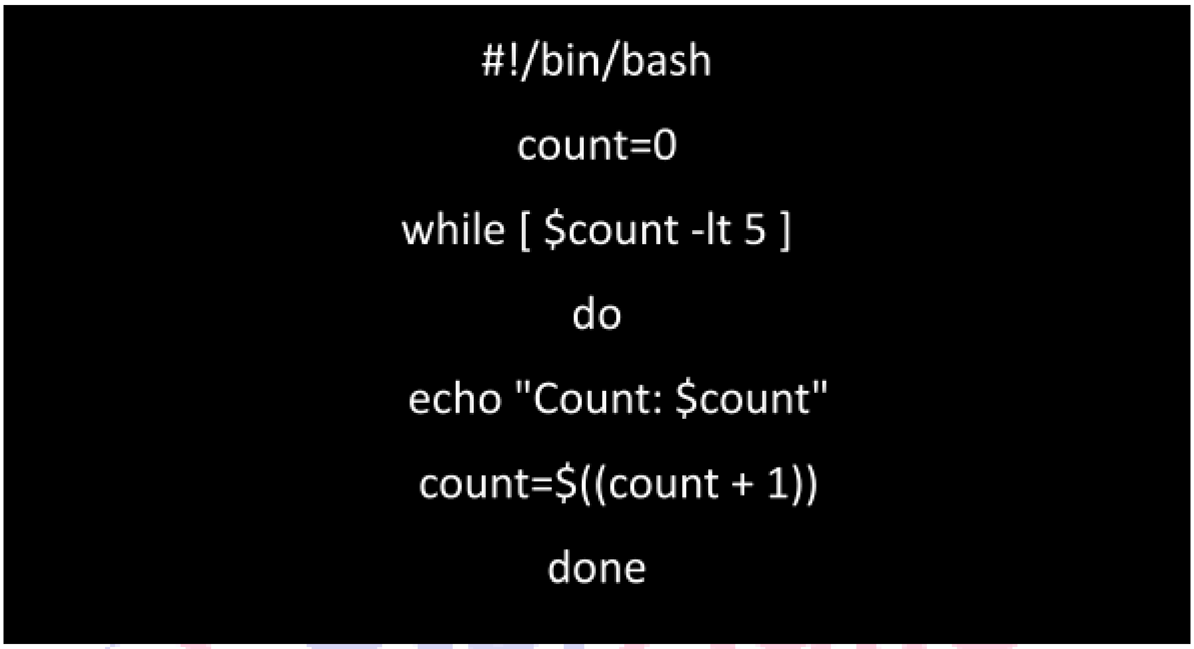
```
Value: 1
Value: 2
Value: 3
Value: 4
Value: 5
```

b. while Loop:

- The while loop executes a set of commands repeatedly as long as a condition is true.
- The basic syntax of a while loop is:

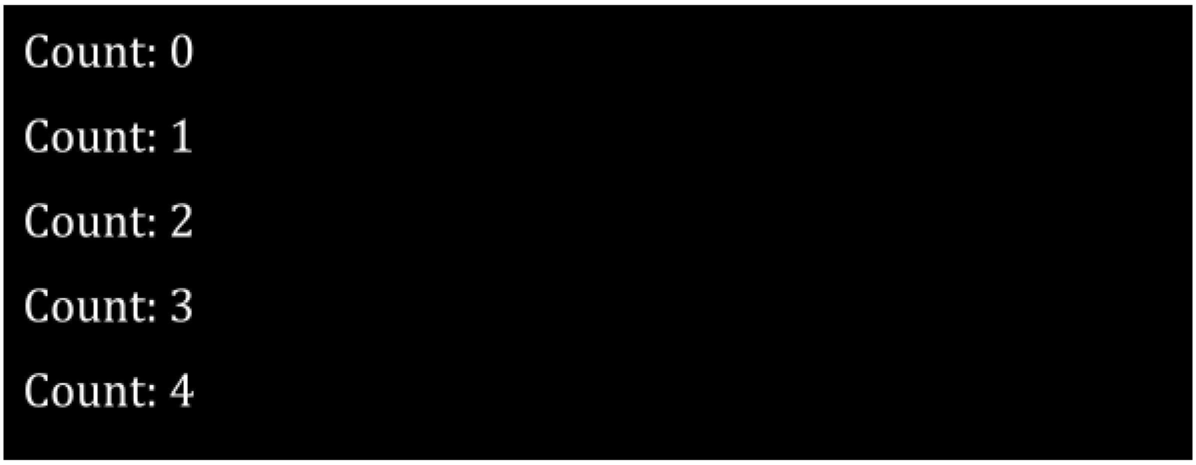
```
while condition
do
# commands to execute
done
```

Example :



```
#!/bin/bash
count=0
while [ $count -lt 5 ]
do
echo "Count: $count"
count=$((count + 1))
done
```

Output :



```
Count: 0
Count: 1
Count: 2
Count: 3
Count: 4
```

- In the above example, the while loop continues executing the commands as long as the condition '$count –lt' 5 (count is less than 5) is true.
- The loop increments the count variable by 1 in each iteration.

- Both 'for' and 'while' loops can be combined with various control flow statements and conditions to perform complex operations and handle dynamic scenarios.
- You can also use the 'break' statement to exit a loop prematurely or the 'continue' statement to skip the rest of the current iteration and move to the next one.

Shell Script Features :

- Shell scripting, as part of Unix-like operating systems, offers several features that make it a powerful tool for automating tasks and managing system configurations.

a. Shell Environment:

- Shell scripts run within a shell environment, which provides access to various system resources, commands, utilities, and variables.

- It allows you to interact with the operating system and execute commands as if you were typing them directly into the terminal.

b. Script Execution:

- Shell scripts can be executed by invoking the shell interpreter and passing the script file as an argument.
- They can also be made executable by setting the execute permission and run directly like any other executable program.

c. Command Execution:

- Shell scripts can execute system commands, utilities, and external programs.
- They allow you to leverage the vast collection of existing tools and utilities available in the Unix/Linux ecosystem.
- You can run commands, capture their output, and manipulate the data as needed.

d. Variable Support:

- Shell scripts support variables, which can hold values of different types, including strings and integers.
- Variables allow you to store data, perform calculations, and pass information between different parts of the script.
- Variable names are case-sensitive by default.

e. Control Flow:

- Shell scripts support various control flow structures, including conditional statements (if-else), loops (for, while), and case statements (case).
- These structures enable you to make decisions, repeat actions, and control the flow of execution based on conditions and input.

f. Input and Output:

- Shell scripts can read input from the user or from files using standard input (stdin).
- They can also display output to the terminal using standard output (stdout).
- Input/output redirection and piping allow you to manipulate and redirect the flow of data between commands.

g. Command-Line Arguments:

- Shell scripts can accept command-line arguments, allowing users to pass parameters and values when executing the script.
- These arguments can be accessed within the script using special variables ($1, $2, etc.) representing the positional parameters.

h. Error Handling:

- Shell scripts provide error handling capabilities, including exit status codes, error messages, and error handling techniques.
- You can check the success or failure of commands using the '$?' variable and use conditional statements to handle errors or take appropriate actions.

i. String Manipulation:

- Shell scripts support various string manipulation operations, including concatenation, substring extraction, searching, replacing, and pattern matching using regular expressions.
- String manipulation allows you to process and manipulate textual data efficiently.

j. File Operations:

- Shell scripts provide numerous file operations, such as creating, deleting, renaming, copying, and moving files and directories.
- They can also read and write file content, change file permissions, and perform other file-related tasks.

- These features make shell scripting a versatile and flexible tool for automating repetitive tasks, system administration, and creating complex workflows.
- Shell scripts are widely used in various domains, including system administration, DevOps, data processing, automation, and more.

## 8. SSH and VI utility

Information

SSH (Secure Shell) and Vi are both widely used utilities in the Unix/Linux environment.

a. SSH (Secure Shell):

- SSH is a cryptographic network protocol that provides a secure way to access and manage remote systems over an unsecured network.

- It enables secure remote login, remote command execution, and secure file transfers.
- SSH encrypts the communication between the client and server, preventing unauthorized access and protecting the integrity of the data transmitted.
- SSH utility allows you to establish a secure connection to a remote server using the SSH protocol.
- Once connected, you can execute commands on the remote server, transfer files securely, and even create secure tunnels for other network services.
- The most common usage of SSH is to log in to a remote server using a username and password or public/private key authentication.

b. Vi (Visual Editor):

- Vi is a text editor that comes preinstalled on most Unix/Linux systems.
- It is a command-line-based editor known for its power and efficiency in editing text files.
- Vi operates in different modes: command mode and insert mode.
- In command mode, you can navigate through the file, search and replace text, and execute various editing commands.
- In insert mode, you can enter and edit text.

- Vi is highly customizable and has a wide range of commands for performing tasks like copying, pasting, deleting, and formatting text.
- Although Vi has a steep learning curve, mastering it allows you to edit files quickly and efficiently from the command line.
- Vi has different variants, such as Vim (Vi Improved), which offers additional features and enhancements.

- Vi can be used to edit configuration files, scripts, source code, and any text-based files on the command line or in a terminal environment.

- Note: If you're new to Vi, it may be initially challenging to use.
- However, with practice and familiarity, it becomes a powerful tool for editing text efficiently in a Unix/Linux environment.

Shell script execution establishing SSH connection to EC2 using SSH or PuTTY :

- To establish an SSH connection to an EC2 instance using either SSH or PuTTY, you can follow these steps:

a. Using SSH (for Linux/Mac):

- Open a terminal or command prompt on your local machine.

- Locate the private key file (.pem) associated with your EC2 instance.
- If you don't have one, create a new key pair and download the '.pem' file.
- Set the permissions of the private key file to be read-only by the owner:

```
chmod 400 your-key.pem
```

- Connect to the EC2 instance using the SSH command, providing the path to the private key file and the public IP/DNS of the EC2 instance:

```
ssh -i your-key.pem ec2-user@public-ip-or-dns
```

- Replace your-key.pem with the actual path to your private key file and public-ip-or-dns with the public IP address or DNS name of your EC2 instance.

b. Using PuTTY (for Windows):

- Download and install PuTTY and PuTTYgen if you haven't already.

- Launch PuTTYgen and load your private key file (.pem) by clicking "Load" and selecting the file.
- PuTTYgen will convert the key into PuTTY's format. Click "Save private key" and save the converted key as a .ppk file.
- Open PuTTY and enter the public IP/DNS of your EC2 instance in the "Host Name (or IP address)" field.
- Under "Connection" -> "SSH" -> "Auth", browse and select the .ppk file you generated in step 3.
- Click "Open" to start the SSH connection.
- Both SSH and PuTTY will establish a secure SSH connection to your EC2 instance, allowing you to interact with the remote server via the command line.

- Note: Ensure that the security group associated with your EC2 instance allows incoming SSH traffic (port 22) from your IP address or a specific IP range.

Vi editor and its usage :

- Vi is a powerful and widely used text editor in the Unix/Linux environment.
- It is a modal editor, meaning it has different modes for different operations.

a. Opening a File:

- To open a file with Vi, simply type vi followed by the filename:

```
vi filename
```

- If the file exists, Vi will open it for editing.
- If the file doesn't exist, Vi will create a new file with the specified name.

b. Modes in Vi:

- Vi has different modes:

i.Normal Mode:

- The default mode when you open Vi.
- In this mode, you can navigate through the file, issue commands, and perform various operations.

ii. Insert Mode:

- In this mode, you can enter and edit text.
- To switch to Insert Mode, press the ‘I’ key.

iii. Command Mode:

- This mode allows you to issue commands to perform file operations, save changes, exit Vi, etc.
- To enter Command Mode, press the Esc key.

c. Basic Navigation:

i.Move the cursor:

- In Normal Mode, you can use the arrow keys or the h, j, k, l keys to move the cursor left, down, up, or right respectively.

ii. Page navigation:

- Use Ctrl + F to move forward one page and Ctrl + B to move backward one page.

iii. Line navigation:

- Use 0 to move to the beginning of a line and $ to move to the end of a line.

d. Editing Text:

i.Insert text:

- Switch to Insert Mode by pressing i. Then, type the text you want to insert.

ii. Delete text:

- In Normal Mode, position the cursor at the desired location and press x to delete the character under the cursor.
- Use 'dd' to delete the whole line.

iii. Copy and paste:

- Position the cursor at the start of the text you want to copy.
- Press ‘yy’ to copy the line, or ‘yw’ to copy a word.
- Move the cursor to the desired location and press p to paste the copied text.

e. Saving and Exiting:

i.Save changes:

- In Command Mode, type ‘:w’ and press Enter to save the changes made to the file.

ii. Exit Vi:

- In Command Mode, type ‘:q’ and press Enter to exit Vi.

iii. Save and Exit:

- To save changes and exit Vi, use ‘:wq’ or ‘:x’.

- These are just a few basic commands and operations in Vi.
- Vi has many more features and commands for advanced editing, searching, and customization.
- It may take some time to become comfortable with Vi, but once mastered, it becomes a powerful and efficient text editor.

Practical :

1. Creation of User

- To create a user in a Unix/Linux system, you can use the ‘useradd’ or ‘adduser’ command, depending on the specific distribution you are using.

- Here's a general guide on creating a user:

- Open a terminal or log in to the system as a user with administrative privileges (such as root or a user with sudo access).
- Determine the command to use based on your Linux distribution:
- For Red Hat-based distributions (e.g., CentOS, Fedora):

```
useradd username
```

- For Debian-based distributions (e.g., Ubuntu):

```
adduser username
```

c. Replace username with the desired username for the new user.

- For example, if you want to create a user named "john", you would use:

```
useradd john
```

Or

```
adduser john
```

d. Provide a password for the new user when prompted. Note that the password will not be displayed on the screen as you type it.

e. Follow the prompts to provide additional information about the user, such as full name, phone number, etc.

- This step is optional and depends on your specific configuration.

f. Once the user is created, you can log in as that user using the specified username and password.

- Note: The exact behaviour and options for user creation may vary slightly between different Linux distributions.
- It is recommended to refer to the documentation specific to your distribution for precise instructions and additional options.
- Creating a user typically involves more advanced considerations, such as setting user permissions, assigning groups, and configuring user-specific settings.
- These aspects can be further explored based on your specific requirements and the configuration of your Linux system.

2. Establishing SSH connection to the server

- To establish an SSH connection to a server, you can use the ssh command.
- Here's a step-by-step guide:

a. Open a terminal or command prompt on your local machine.
b. Determine the IP address or domain name of the server you want to connect to.
c. Open the terminal and enter the following command:

```
ssh username@server_address
```

- Replace ‘username’ with your username on the server, and ‘server_address’ with the IP address or domain name of the server.

d. If it's your first time connecting to the server, you may see a warning about the authenticity of the host.
- Verify that the fingerprint matches the expected fingerprint of the server, and type "yes" to proceed.

e. Enter your password when prompted.
- Note that the password will not be displayed on the screen as you type it.

f. Once you've entered the correct password, you will be logged in to the server via SSH.

- If the server uses a different SSH port (not the default port 22), you can specify the port number using the ‘-p’ option:

```
ssh -p port_number username@server_address
```

- Replace ‘port_number’ with the actual port number used by the server.

- If you have an SSH key pair configured for authentication, you can use the ‘-i’ option to specify the private key file:

```
ssh -i /path/to/private_key username@server_address
```

- Replace ‘/path/to/private_key’ with the actual path to your private key file.

- Remember to replace ‘username’ and ‘server_address’ with the appropriate values for your setup.

- By establishing an SSH connection, you can securely access and manage remote servers for various purposes, such as system administration, file transfers, and executing remote commands.

3. File Creation and Manipulation using VI editor

- To create and manipulate files using the Vi editor, you can follow these steps:

a. Open the Vi editor:

```
vi filename
```

- Replace filename with the name of the file you want to create or edit.

b. Vi opens in command mode by default.
- To switch to insert mode and start entering text, press i on your keyboard.

c. Start typing the content of your file.

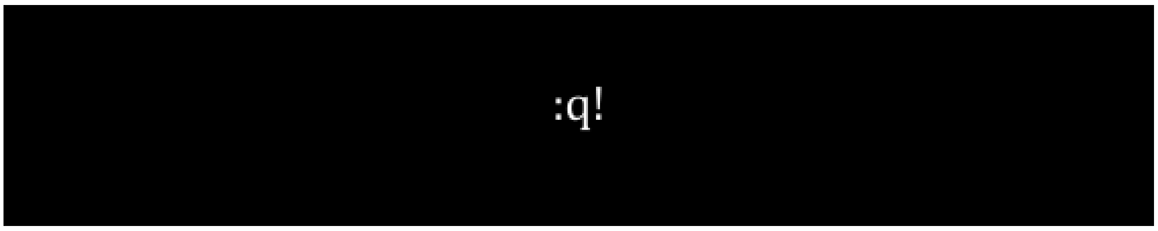
d. To save the changes and return to command mode, press Esc.

e. To save the file and exit Vi, enter the following command in command mode:

```
:wq
```

- This command will write (save) the changes and quit the editor.

f. If you want to exit Vi without saving any changes, enter the following command in command mode



```
:q!
```

- This command will quit the editor without saving.

g. While editing, you can perform various manipulations using Vi commands:

- Delete a character: Position the cursor on the character and press x.
- Delete a line: Position the cursor on the line and press dd.
- Copy a line: Position the cursor on the line and press yy.
- Paste a copied line: Position the cursor where you want to paste and press p.
- Undo the last change: Press u.
- Search for a specific word: Press / followed by the word and press Enter.

h. To navigate through the file, you can use the following commands:

- Move the cursor up: Press k.
- Move the cursor down: Press j.
- Move the cursor to the beginning of the line: Press 0.
- Move the cursor to the end of the line: Press $.

- These are just a few basic commands to create and manipulate files using the Vi editor.
- Vi offers many more features and commands for advanced editing, searching, and customization.
- It may take some time to become proficient with Vi, but it is a powerful text editor once you get familiar with its commands and modes.

4. Managing Permissions

- Managing permissions in Unix/Linux systems is an important aspect of file and directory management.
- Permissions control access to files and directories for users and groups.
- Here are some common commands and concepts related to managing permissions:

a. Viewing Permissions:

- 'ls –l': Use the 'ls' command with the '-l' option to display detailed information about files and directories, including permissions.

b. Permission Representation:

i. Symbolic Representation:

- Permissions are represented by a series of letters: r for read, w for write, and x for execute.
- Each letter can be either present or absent, indicating the permission status for the user, group, and others.
- For example, rw-r--r-- means read and write permissions for the user, and read-only permissions for the group and others.

ii. Numeric Representation:

- Permissions can also be represented numerically using octal values.
- Each permission is assigned a numeric value: read (4), write (2), and execute (1).
- These values are added together to represent the permissions. For example, 644 is equivalent to rw-r--r--.

c. Changing Permissions:

i.‘chmod’:

- The ‘chmod’ command is used to change permissions.
- It accepts both symbolic and numeric representations.

ii. Symbolic Method:

- Use symbolic representation with the +, -, or = symbols to add, remove, or set permissions.

For example:

- ‘chmod +x script.sh’ adds execute permission to the file for the user, group, and others.
- ‘chmod o-r file.txt’ removes read permission for others.
- ‘chmod g=rw file.txt’ sets read and write permissions for the group.

iii. Numeric Method:

- Use the numeric representation to set permissions explicitly.
- For example:
- ‘chmod 755 script.sh’ sets read, write, and execute permissions for the user, and read and execute permissions for the group and others.

d. Ownership:

- ‘chown’: The ‘chown’ command is used to change the owner of a file or directory.
- For example, ‘chown newowner file.txt’ changes the owner of ‘file.txt’ to ‘newowner’.
- ‘chgrp’: The ‘chgrp’ command is used to change the group ownership of a file or directory.
- For example, ‘chgrp newgroup file.txt’ changes the group ownership of ‘file.txt’ to ‘newgroup’.

- These commands help you manage permissions and ownership of files and directories, ensuring proper access control and security in Unix/Linux systems.
- It's important to understand the implications of changing permissions and to exercise caution when modifying them to prevent unauthorized access or accidental misconfiguration.

5. Basic commands execution

- Sure! Here are some basic commands that can be executed in a Unix/Linux terminal:

a. 'ls': Lists files and directories in the current directory.

- Example: ls

b. 'cd': Changes the current directory.

- Example: cd /path/to/directory

c. 'pwd': Prints the current working directory.

- Example: pwd

d. 'mkdir': Creates a new directory.

- Example: mkdir new_directory

e. 'rm': Removes files and directories.

- Example: rm filename (removes a file)

- Example: rm -r directory (removes a directory and its contents recursively)

f. ‘cp’: Copies files and directories.
- Example: cp source_file destination_file (copies a file)
- Example: cp -r source_directory destination_directory (copies a directory recursively)

g. ‘mv’: Moves or renames files and directories.
- Example: mv old_filename new_filename (renames a file)
- Example: mv file destination_directory (moves a file to a new directory)

h. ‘cat’: Displays the content of a file.
- Example: cat filename

i. ‘echo’: Prints text or variables to the terminal.
- Example: echo "Hello, World!"

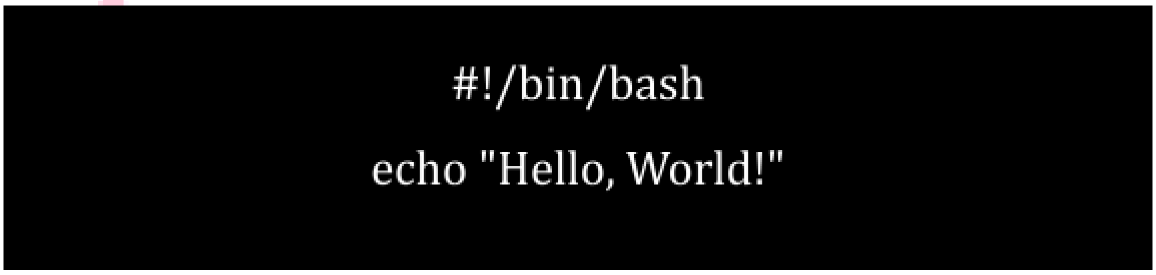
j. ‘grep’: Searches for a pattern in files.
- Example: grep "pattern" filename

- These are just a few basic commands, but there are many more commands available in Unix/Linux systems for various purposes.
- Each command has its own set of options and arguments that can be used to modify its behaviour.
- You can explore the man command (e.g., man ls) to access the manual pages and learn more about each command's usage and options.

6. Writing Shell scripting programs

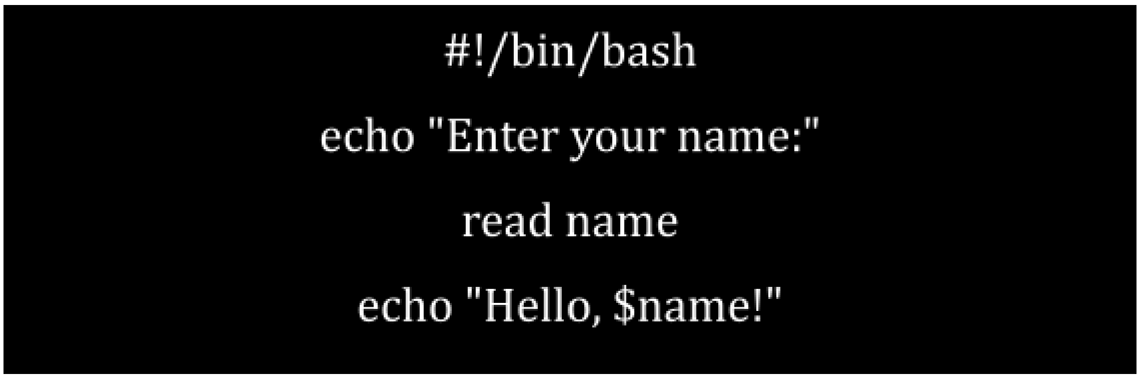
- Certainly! Here are a few examples of shell script programs:

a. Hello World:



```
#!/bin/bash
echo "Hello, World!"
```

b. User Input and Output



```
#!/bin/bash
echo "Enter your name:"
read name
echo "Hello, $name!"
```

c. Arithmetic Calculation

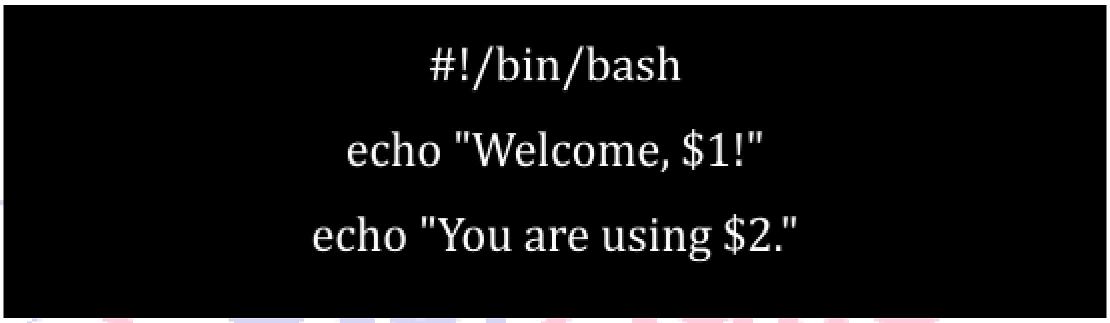
```
#!/bin/bash
echo "Enter two numbers:"
read num1
read num2
sum=$((num1 + num2))
echo "Sum: $sum"
```

```
#!/bin/bash
echo "Enter a number:"
read number
if [ $number -gt 0 ]; then
echo "Number is positive"
else
echo "Number is not positive"
fi
```

e. Looping

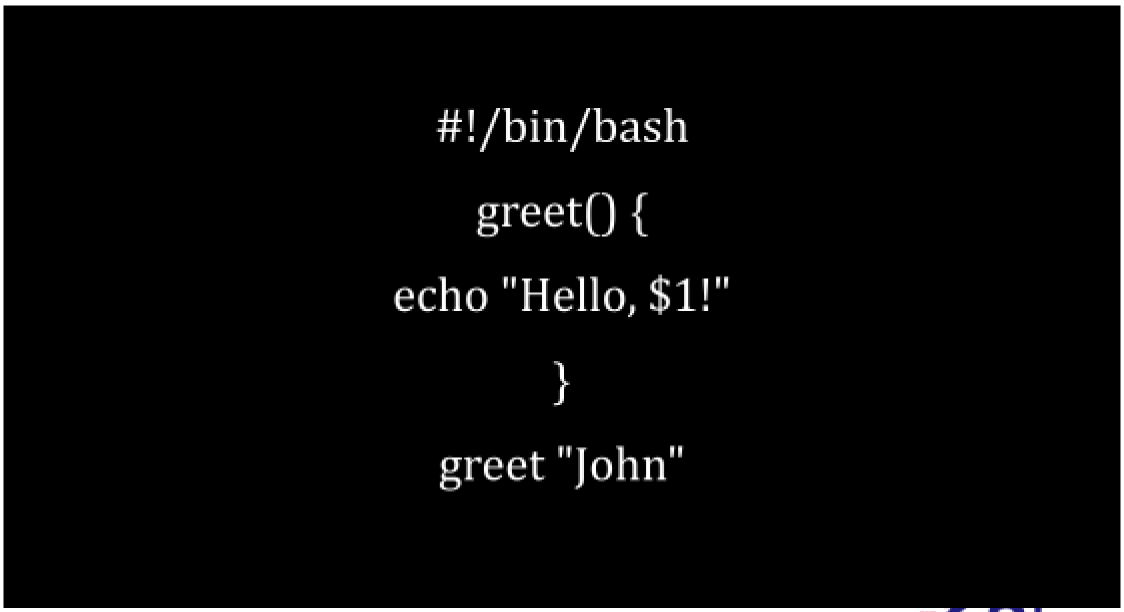
```
#!/bin/bash
echo "Countdown:"
for (( i=10; i>=1; i-- )); do
echo $i
```

f. Command Line Arguments



```
#!/bin/bash
echo "Welcome, $1!"
echo "You are using $2."
```

g. Functions :



```
#!/bin/bash
greet() {
echo "Hello, $1!"
}
greet "John"
```

h. File read and write

```
#!/bin/bash
echo "Enter a message:"
read message
echo $message > output.txt
echo "Message written to file."
```

i.

```
#!/bin/bash
command_that_may_fail
if [ $? -eq 0 ]; then
echo "Command executed successfully."
else
echo "Command failed."
fi
```

j. File Manipulation:

```
#!/bin/bash
echo "Enter a directory name:"
read directory
if [ -d $directory ]; then
echo "Directory already exists."
else
```

- These are just a few examples to get you started. You can explore more complex shell script programs by combining different concepts, such as string manipulation, regular expressions, conditional loops, and interacting with system commands.
- The possibilities are endless, and shell scripting provides a powerful toolset for automating tasks and managing systems.