LINUX.

EXERCISE-1:

1. **Difference between daemon and service with example.**

   Service:

   In Linux, a service is a process that performs specific functions for the system, Services can be managed using system management tools like system. They can be started, stopped, and restarted manually or automatically at boot time based on configuration.

   Example: A web server like Apache can be run as a service. You can start it with a command like
   sudo systemctl start apache2,
   and it can also be configured to start automatically on boot.

   Daemon:

   A daemon is a background process that typically runs without direct user interaction. It usually starts at system boot and remains active, waiting for requests or events

   Example: The SSH daemon (sshd) runs in the background to handle secure shell connections. It starts when the system boots up and listens on a specific port (usually port 22) for incoming SSH requests.

   Key Differences:
   starting point:
   Service: Can be manually started or stopped, and may not always run continuously.
   Daemon: Usually starts at boot time and runs continuously in the background.

   Purpose:
   Service: Often tied to user applications and can be invoked on demand.
   Daemon: Generally performs ongoing system tasks and listens for events or requests.

   Management:
   Service: Managed using commands like systemctl (for systemd), which allows you to check status, start, stop, or restart.
   Daemon: Services can be a part of applications or specific tasks, while daemons are more focused on running system tasks continuously.

   Commands:
   systemctl list-units --type=service | grep sshd
   ps -aux | grep sshd

   ? in terminal indicating that the process is not attached to any terminal.

2. **How to create a hierarchy of directories (/opt/rock/break) using mkdir command?**

   Cmd: mkdir –p /opt/rock/break

   mkdir: to create a directories
   -p : option used to create a parent directory if needed instead sending an error.

Other options:
-m: to set permission while creating a directory
mkdir xyz –m 777 | mkdir xyz –m u=rwx,g=rw,o=x

-v : verbose(prints the message about the transactions)

3. Different types of shells ?
shell:
A shell is a type of computer program called a command-line interpreter that lets Linux and Unix users control their operating systems with commands.

Simply put, the shell is a program that takes commands from the keyboard and gives them to the operating system to perform any operation. In the old days, it was the only user interface available on a Unix-like system such as Linux. Nowadays, we have graphical user interfaces (GUIs) in addition to command line interfaces (CLIs) such as the shell.

to access shell we need - Terminal
It's a program called a terminal emulator. This is a program that opens a window and lets you interact with the shell. There are a bunch of different terminal emulators we can use. Some Linux distributions install several. These might include gnome-terminal, konsole, xterm, rxvt, kvt, nxterm, and eterm.

The prompt, $, which is called the command prompt, is issued by the shell. While the prompt is displayed, you can type a command.

Shell is broadly classified into two categories –

- **Graphical shell**
- **Command Line Shell**

Default shell: The default shell on most Linux systems is Bash (Bourne-Again Shell).

Types of shell:
1. sh (Bourne Shell)
Overview: The original shell developed for Unix.
Features: Simpler syntax compared to Bash.
Limited features (e.g., no command line editing).
Use Cases: Mostly used for scripting; many scripts written in sh are compatible with other shells.

2. Bash (Bourne Again Shell)
Overview: The most widely used shell in Linux distributions.
Features:
Command line editing.
Command history.
Tab completion.
Scripting capabilities with advanced features.
Use Cases: Default shell for many Linux distributions; great for both interactive use and scripting.

3. C Shell (csh)
Overview: A shell that uses C-like syntax.
Features: Aliases and built-in arithmetic expressions.
Job control features.
Scripting uses a syntax that resembles the C programming language.
Use Cases: Useful for users who prefer C-style syntax, but less commonly used for scripting than Bash.

4. Korn Shell (ksh)
Overview: Combines features of the Bourne shell and C shell.
Features: Command line editing.
Array variables and advanced scripting features.
Job control.

Use Cases: Suitable for interactive use and scripting, especially in enterprise environments.

5. Z Shell (zsh)
Overview: A powerful shell with features from Bash, ksh, and csh.
Features:
Advanced tab completion.
Better globbing and file management.
Highly customizable with plugins and themes (especially with frameworks like Oh My Zsh).
Use Cases: Popular among developers and power users for its features and customization.

6. Fish (Friendly Interactive Shell)
Overview: A modern shell focused on user-friendliness.
Features:
Syntax highlighting and autosuggestions.
Simple and clean syntax without the need for extensive configuration.
Use Cases: Ideal for users looking for an easy-to-use shell with a focus on interactivity.
Web-based configuration
Man page completions
Auto-suggestions

echo $SHELL : Command to know the current SHELL

cat /etc/shells : way to see all the installed shell

<u>chsh (change shell) command:</u>

chsh -s /path/to/shell
! After running this command, you will need to log out and log back in for the change to take effect.

For temporary switch just type shell name:
sh

4. **What is the use of tar command ? List out the uses.**
the Linux tar command used to create compressed or uncompressed Archive files.
<u>Syntax:</u>
tar -optins archive_file_name files_to_be_archived

Options Description:
-c Creates an archive by bundling files and directories together.
-x Extracts files and directories from an existing archive.
-f Specifies the filename of the archive to be created or extracted.
-t Displays or lists the files and directories contained within an archive.
-u Archives and adds new files or directories to an existing archive.
-v Displays verbose information, providing detailed output during the archiving or extraction process.
-A Concatenates multiple archive files into a single archive.
-z Uses gzip compression when creating a tar file, resulting in a compressed archive with the '.tar.gz' extension.
-j Uses bzip2 compression when creating a tar file, resulting in a compressed archive with the '.tar.bz2' extension.
-r Updates or adds files or directories to an already existing archive without recreating the entire archive.
-d/(—diff) displays difference b/w compressed file with original file

<u>uses of the tar command:</u>

- **Creating archives**
- **Extracting archives**
- **Viewing contents of an archive**
- **Compressing archives (using gzip or bzip2)**
- **Extracting compressed archives**

- **Appending files to an archive**
- **Removing files from an archive**
- **Extracting specific files from an archive**
- **Using wildcards to include files**
- **Preserving file permissions when extracting**

5. **What is the use of SSH command.**
   With the help of ssh command we can connect to to remote system.

   The ssh (Secure Shell) command is used to securely connect to remote systems over a network. It provides a way to access another computer or server remotely, enabling secure communication between the client and the remote machine. SSH is widely used for managing and administering remote servers, transferring files securely, and performing various remote tasks.

   Main Use of ssh Command:
   - **Remote Login: It allows you to log into and execute commands on a remote machine securely over a network.**
   - **Secure Data Transfer: SSH can also be used for secure file transfer with tools like scp (secure copy) or sftp (secure FTP), both of which rely on SSH.**
   - **Remote Execution: You can run commands on a remote system without logging into it interactively.**

   Syntax: ssh [user@]hostname [command]
   Eg: ssh root@192.168.1.100 'ls –l'

6. **How to create a user with password ?**
   To add a new user in Linux, use the useradd command followed by the username. For example, to add a user named "newuser":
   Eg: sudo useradd newuser

   After creating the user, set a password for the new user:
   sudo passwd newuser.

7. **How to view the content of How a file in reverse order?**
- **tac command**
- **cat -n f1 | sort -nr | cut -f 2-**

8. **How many types of files are available in linux and what are they?**

File type:	Description:
Regular files	Contain data of various content types such as text, script, image, videos, etc.
Directory files	Contain the name and address of other files.
Block or character special files	Represent device files such as hard drives, monitors, etc.
Link files	Point or mirror other files
Socket files	Provide inter-process communication
Named pipe files	Allow processes to send data to other processes or receive data from other processes.

9. **How to search for a particular string in a text file on Linux?**
   Grep command:
   grep -w "search_string" filename

   -w: match exact word

10. **How to search for a particular file in the Linux file system?**
    The find command in Linux is for searching for files and directories within a directory hierarchy.
    Syntax:
    find [path] [options/expression]

Options:
-name "pattern"
-iname "pattern"
-type f: Search for regular files.
-size n: Find files of size n (e.g., +1M, -500k, 2G).
-mtime n: Modified time (days ago).
-n: Modified within the last n days.
+n: Modified more than n days ago.
-user username: Find files owned by a specific user.
-group groupname: Find files owned by a specific group.
-perm mode: Find files with specific permissions (e.g., -perm 755).
-exec command {} ;

Exercise -2

1. **What is the difference between hardlink and soft link?**

Comparison Parameters	Hard link	Soft link
Inode number	Files that are hard linked take the same inode number.	Files that are soft linked take a different inode number.
Directories	Hard links are not allowed for directories. (Only a superuser* can do it)	Soft links can be used for linking directories.
File system	It cannot be used across file systems.	It can be used across file systems.
Data	Data present in the original file will still be available in the hard links.	Soft links only point to the file name, it does not retain data of the file.
Original file's deletion	If the original file is removed, the link will still work as it accesses the data the original was having access to.	If the original file is removed, the link will not work as it doesn't access the original file's data.

2. **What are the important bootable files in linux?**
Here are some of the most important bootable files:

/boot/grub/grub.conf: This file contains the configuration for the GRUB bootloader, which is responsible for loading the Linux kernel.

/boot/vmlinuz-*: This is the compressed Linux kernel image. The * corresponds to the kernel version.
Initial RAM Disk:

/boot/initrd.img-* or /boot/initramfs-*: These files contain the initial ramdisk used to load essential drivers and mount the root filesystem before the actual kernel is ready.
Bootloader:

/boot/efi/: For systems using UEFI, this directory contains the bootloader files, including EFI/BOOT/BOOTX64.EFI.
Systemd Boot Files (if applicable):

/boot/systemd/: If using systemd-boot (formerly gummiboot), this directory contains configuration files for booting.
Configuration Files:

/etc/fstab: This file contains information about disk drives and partitions and their mount points, crucial for the boot process.

Kernel Modules:
/lib/modules/: Contains the kernel modules for the currently running kernel, which are loaded during boot.
These files and directories play critical roles in initializing the operating system, loading the kernel, and preparing the system for operation. The exact paths and files may vary slightly between different Linux distributions.

3. **Explain the use of aspell & tr commands with an example?**

Aspell: aspell command is used as a spell checker in Linux. Generally, it will scan the given files or anything from standard input then it check for misspellings. Finally, it allows the user to correct the words interactively.

Syntax: aspell check [options] filename

example: aspell -c sample.txt

-c Check a file for specific spelling errors.

You can use aspell to check individual words directly in the terminal. For instance, running:

aspell –a

Note: As soon as you run this command, it will wait for user input. Type a word in this mode, press enter, and you'll see aspell offering spelling suggestions on stdout.

Tr(Translate):

syntax: tr <'old'> <'new'>

eg: cat exm.txt | tr 'prcu' 'PRCU'

Running tr without any options replaces each of the characters specified in SET1 with the characters from SET2 that have the same position.

tr -s : The 'tr -s' command squeezes the occurence of multiple characters into one.

tr -d: The 'tr -d' command is used to delete characters.

tr -c: complement/inverst operation

command | tr -d <letter>

to convert lower case characters to upper case.

$ cat greekfile | tr [a-z] [A-Z]

$ cat greekfile | tr [:lower:] [:upper:]

remove all the digits from the string, you can use

$ echo "my ID is 73535" | tr -d [:digit:]

From & To File:

$ tr "{}" "()" <greekfile >newfile.txt

4. **touch command is used to create an empty file. Is it only the use of touch commands?**

The main purpose of the touch command is to create, update and modify the timestamps of a file.

Options	Description
-t	create a file using a specified time. [YYMMDDHHMM]
-a	This option changes the access time only.
-c	Suppresses file creation if the file does not exist.
-m	This option changes the modification time only.
-d	Sets the access and modification times using the specified STRING. {YYYY-MM-DD HH:MM:SS}
-r	Uses the access and modification times from the reference file.

5. **How to list all the available partitions in linux?**

1. lsblk Command: The lsblk command lists all block devices, including partitions and their mount points.

2. fdisk Command: You can use fdisk with the -l option to list all partitions on all disks.

3. /proc/partitions File: You can check the /proc/partitions file, which contains information about all partitions.

6. **How to view disk space in linux?**

df -h Command: The df (disk free) command displays the amount of disk space used and available on file systems.

7. **What is GRUB and what are the features of GRUB?**
    - **GRUB, or the GRand Unified Bootloader, is a boot loader package designed to support multiple operating systems on a computer. It is commonly used in Linux environments but can also boot other operating systems.**
    - **Multi-OS Booting: GRUB can boot multiple operating systems installed on a single machine, allowing users to choose which OS to load at startup.**
    - **Support for Various Filesystems: It supports a wide range of filesystems, including ext4, btrfs, NTFS, FAT, and more, enabling it to read boot information from different types of disk partitions.**
    - **Configuration Flexibility: GRUB's configuration is highly customizable. Users can modify settings in the grub.cfg file to change the boot menu appearance, default OS, and other parameters.**
    - **Graphical User Interface: GRUB provides a graphical interface that allows users to select operating systems easily, as well as command-line support for advanced users.**
    - **Kernel Loading Options: It can load Linux kernels with various options, including kernel parameters and initial RAM disk images, enhancing system boot capabilities.**
    - **Booting from Network: GRUB can boot operating systems from a network location using protocols like PXE (Preboot Execution Environment).**
    - **Command-line Interface: For troubleshooting, GRUB offers a command-line interface that allows advanced users to manually boot operating systems or fix issues.**
    - **Support for Recovery: GRUB can be used to boot into recovery modes, enabling users to troubleshoot and repair systems that are not starting properly.**
    - **Password Protection: GRUB allows you to set up password protection for certain boot options, enhancing security for sensitive environments.**

8. **What is the use of the uniq command?**
   Linux uniq command is used to remove all the repeated lines from a file. Also, it can be used to display a count of any word, only repeated lines,
   -c, --count: it prefixes the lines by the number of occurrences.
   -d, --repeated: it is used to print duplicate lines, one for each group.
   -D: It is used to print all the duplicate lines.
   -i, --ignore-case: It is used to ignore the differences while comparing.
   -s, --skip-chars=N: It is used to avoid the comparison of the first N characters.
   -u, --unique: it is used to print unique lines.

9. **What are the standard I/O redirectors?**
   common standard I/O redirectors:

   1. Standard Output (stdout) Redirector:
   >: Redirects the output of a command to a file, overwriting the file if it exists.
   command > file.txt

   >>: Appends the output of a command to the end of a file, preserving existing content.
   Command >> file.txt

   2. Standard Input (stdin) Redirector:
   <: Redirects input from a file to a command.
   command < file.txt

   3. Standard Error (stderr) Redirector:
   2>: Redirects standard error output to a file, overwriting the file if it exists.
   command 2> error.txt

2>>: Appends standard error output to a file.
command 2>> error.txt

4. Combining stdout and stderr:
&>: Redirects both standard output and standard error to the same file (available in Bash).
command &> output.txt

2>&1: Redirects standard error to standard output, allowing you to capture both in the same stream.
command > output.txt 2>&1

5. Piping:
|: Takes the output of one command and uses it as the input for another command.
command1 | command2

10. **What should be the minimum password length of a non root user and root user?**
    The minimum password length for a root user in Linux is typically recommended to be at least 12 characters. For non-root users, a minimum of 8 characters is generally advised. However, longer passwords are always better for security.

    To change the minimum password length in Linux, you can modify the pam_pwquality module settings. Here's how:
    Open the configuration file:

    sudo nano /etc/security/pwquality.conf

    Edit or add the following line:
    minlen = 12
    (Replace 12 with your desired minimum length.)

    Save and exit the file.

    To ensure changes take effect, you may need to check or modify /etc/pam.d/common-password (Debian-based) or /etc/pam.d/system-auth (Red Hat-based) to ensure pam_pwquality.so is included.

    Restart any relevant services or the system if necessary.

11. **Create a file with only one character and Count no of characters using the "wc -c" command. What is it showing and why?**

    <u>Hidden Newline:</u> Some text editors automatically append a newline character at the end of the file when saving. So if you enter one character and save, the editor might add a newline, resulting in a total of two bytes.

    If the file includes a newline character (for example, if you press Enter after typing the character), the command will output This is because the newline character (\n) also counts as a byte, so you have one byte for the character and one for the newline, totaling two bytes.

    you can use a command like cat -A filename to visualize all characters, including hidden ones like newline.
    ($ indicates the end of a line).

12. **How can we change the ownership of a file?**
    To change the ownership of a file in a Unix-like operating system, you can use the chown command.
    <u>syntax:</u> chown [new_owner]:[new_group] filename

    Change Owner Only: chown newuser filename

    Change Owner and Group: chown newuser:newgroup filename

    Recursive Change: To change ownership of a directory and all its contents, use the -R option:
    sudo chown -R newuser:newgroup directoryname.

13. **Write the command to search for a particular string say "user" in all files in / directory recursively?**
    grep -rl "user" /

14. **What is the use of lsof command?**
    The lsof (List Open Files) command in Linux is used to display information about files that are currently open by processes. It can be used to check which files are being accessed by which processes, including regular files, directories, block devices, and network sockets.
    Syntax: lsof [options] [file|PID|cmd]
    Eg: lsof
    Example Output:

```
COMMAND   PID   USER   FD   TYPE DEVICE SIZE/OFF   NODE NAME
bash    1234 user   cwd   DIR  8,1    4096    2345 /home/user
```

    Useful Options:
    -u : List files opened by a specific user. [lsof -u username]
    -p : List files opened by a specific process (PID). [lsof -p <PID>]
    -i : List files opened by network connections. [lsof -i udp|port_no]
    +D : List all files in a specific directory.[lsof +D /path/to/directory]
    -n : Show IP addresses instead of hostnames (useful for network connections).
    -t : Only display process IDs (PID) without additional information.
    -c : Filter open files by command name. [lsof -c firefox]

15. **Assign the following permissions to a file called "test". Write the command in both symbolic and numeric notations.**
    a. **user read**
    b. **group read,execute,write**
    c. **other execute,read,write**

    numeric method:
    chmod 477 test

    symbolic method:
    chmod u=r,g=rwx,o=rwx test
    chmod u=r,go+rwx test

Exercise -3

1. **What is the difference between ext2 and ext3 file systems and also list advantages and disadvantages?**
   the key differences, along with their advantages and disadvantages:

   Differences:
   Journaling:
   - **ext2: Does not support journaling, which means it does not keep a log of changes. In the event of a crash or power failure, data integrity can be at risk.**
   - **ext3: Supports journaling, which logs changes before they are written to the main file system, providing greater data integrity and quicker recovery after a crash.**

   Performance:
   - **ext2: Generally faster for write operations in some scenarios, especially on small files, since there is no overhead of journaling.**
   - **ext3: Slightly slower for write operations due to the journaling overhead, but performance is often acceptable for most applications.**

   File System Size:
   Both ext2 and ext3 support similar maximum file and file system sizes, but ext3 can be more efficient in managing larger volumes due to its journaling feature.

   Compatibility:
   - **ext2: Can be used as a non-journaling file system, making it compatible with a wider range of operating systems.**

- **ext3: Backward compatible with ext2, meaning ext3 can be mounted as ext2, but not vice versa.**

Advantages and Disadvantages:
ext2
Advantages:

- **Simpler and has less overhead due to no journaling.**
- **Potentially faster for certain workloads, especially on small files.**
- **Easier to recover data using certain tools since it has no journal to manage.**

Disadvantages:

- **Higher risk of data corruption in case of power loss or system crashes.**
- **Longer recovery times after a crash, as the file system needs to be checked thoroughly.**

ext3
Advantages:

- **Better data integrity due to journaling, reducing the risk of corruption.**
- **Faster recovery times after crashes, as it only needs to replay the journal.**
- **Can be converted from ext2 to ext3 without data loss.**

Disadvantages:

- **Slightly slower performance in write-intensive scenarios due to the overhead of journaling.**
- **More complex than ext2, which can be a drawback in certain environments.**

2. **How to view kernel version and Operating system name in linux?**
   Using uname Command
   <u>Kernel Version:</u>
   uname -r
   <u>Operating System Name:</u>
   uname –o

3. **How to create swap space in linux?**
   <u>1. Create Swap Space Using a Swap File:</u>
   Step 1: Create the Swap File:
   sudo fallocate -l 2G /swapfile
   Alternatively, if fallocate is not available, you can use dd:
   sudo dd if=/dev/zero of=/swapfile bs=1M count=2048

   Step 2: Set the Correct Permissions:
   sudo chmod 600 /swapfile

   Step 3: Make the Swap File:
   sudo mkswap /swapfile

   Step 4: Enable the Swap File:
   sudo swapon /swapfile

   Step 5: Make the Swap Permanent:
   sudo nano /etc/fstab
   Add the following line at the end:
   /swapfile none swap sw 0 0

   Step 6: Verify the Swap Space:
   sudo swapon –show

   <u>2. Create Swap Space Using a Swap Partition:</u>
   Step 1: Create a New Partition
   sudo fdisk /dev/sda
   - **Create a new partition by pressing n.**

- **Set the partition type to swap (type 82 in fdisk).**
- **Write the changes with w.**

Now repeat the steps from step3 – step6.

3.Create Swap Space Using LVM:

1. Create Physical Volume (PV)
sudo pvcreate /dev/sda

2.Create Volume Group (VG)
sudo vgcreate vg01 /dev/sda1

3. Create Logical Volume (LV) for Swap
sudo lvcreate -L 2G -n swap_lv vg01
In this command:
-L 2G: Specifies the size of the logical volume (2GB).
-n swap_lv: Names the logical volume swap_lv.
vg01: Specifies the volume group.

4. Create Swap on the Logical Volume:
sudo mkswap /dev/vg01/swap_lv

5. Enable the Swap Space:
sudo swapon /dev/vg01/swap_lv

6. Make the Swap Permanent:
sudo nano /etc/fstab:
/dev/vg01/swap_lv none swap sw 0 0
Save and exit.

7. Verify the Swap Space:
sudo swapon --show

4. **What are the different modes available in the vi editor?**

Vi Command Mode :
When vi starts up, it is in Command Mode. This mode is where vi interprets any characters we type as commands and thus does not display them in the window. This mode allows us to move through a file, and delete, copy, or paste a piece of text. Enter into Command Mode from any other mode, requires pressing the [Esc] key. If we press [Esc] when we are already in Command Mode, then vi will beep or flash the screen.

Vi Insert mode:
This mode enables you to insert text into the file. Everything that's typed in this mode is interpreted as input and finally, it is put in the file. The vi always starts in command mode. To enter text, you must be in insert mode. To come in insert mode, you simply type i. To get out of insert mode, press the Esc key, which will put you back into command mode.

Vi Escape Mode:
Line Mode is invoked by typing a colon [:], while vi is in Command Mode. The cursor will jump to the last line of the screen and vi will wait for a command. This mode enables you to perform tasks such as saving files and executing commands.

5. **How to view free memory in linux?**
free Command: The free command provides a quick overview of memory usage, including free and used memory.

Eg: free
For a more human-readable format (with sizes in KB, MB, or GB):
free –h

6. **What is the use of cpio and restore commands?**

1. cpio Command:

The cpio (copy in/out) command is used for creating and extracting archives of files. It works in a pipeline, reading a list of files to archive (or extract) from standard input or a file list, then performing the corre
sponding action on those files.
similar to creating a .tar file. However, cpio is often used in combination with other commands, such as find, to create more flexible archives

Basic Syntax and example of cpio:

 # Creating an archive

```
find /path/to/directory -type f | cpio -o > archive.cpio
find /etc -type f | cpio -o > etc_files.cpio
```

to crate a tar archive:

```
find /etc –type f | cpio –ov –H tar > archive.tar
```

 # Extracting an archive

```
cpio -idv < archive.cpio
cpio -idv < etc_files.cpio
```

#Listing Files in cpio Archive

```
cpio -tv < archive.cpio
```

 # Copying files from one directory to another

```
find /source/directory -type f | cpio -pdm /destination/directory
find /home/user/old_dir -type f | cpio -pdm /home/user/new_dir
```

options:
-o: This option tells cpio to create an archive.
-i: This option tells cpio to extract files from an archive.
-p: Copy files from one directory to another.
-v: Verbose mode, showing the files being processed.
-m: Preserve the modification times of the files.
-d: Create directories as needed (useful when extracting).

2.restore command:

The restore command in Linux is used to restore files from a backup archive that was created by the cpio command (typically using cpio -o to create the archive). It allows you to recover files from backups that are stored in a cpio-compatible format, which could be stored on a tape, disk, or any storage medium.

Syntax:

```
restore [options] < archive.cpio
```

Options:
-i: Interactive mode — allows you to select files interactively for restoration.
-x: Extract mode — extracts files from the archive.
-t: List the contents of the archive (without extracting).
-v: Verbose mode — shows detailed information while extracting.

Exapmle:

1. List the Contents of a cpio Archive:

```
restore -t < archive.cpio
```

Extract Files from the cpio Archive:

```
restore -x < archive.cpio
```

Additional Notes:

Backup format: The restore command is designed to work with cpio archives that use the -o option for creating backups. If you have a .tar archive, you would typically use the tar command, not restore.

File Permissions: When restoring files, restore attempts to preserve the file permissions, ownership, and timestamps as they were when the files were originally archived.

<u>why do we have restore command when we have a cpio -i option to extract:</u>
The key reason why both cpio and restore exist is historical and functional: while both are used for extracting files, the restore command offers additional features and capabilities that are specifically geared toward backup and recovery operations. the restore command is more specialized for backup use cases.

7. **What is the use of the pidof command?**
   The pidof command in Linux is used to find the process ID (PID) of a running program or process.
   <u>Syntax:</u> pidof [options] program_name/cmd_name
   <u>eg:</u> pidof sshd

8. **What are the differences between cron and at ?**
   - **CRON is for running task at a regular base (every hour, day, first of the month etc), cron job is used to schedule the job. It is used for maintain the daily routing work.**
   - **crontab allows the user to submit, edit or delete entries to cron. A crontab file is a user file that holds the scheduling information.**
   - **AT on the other hand, is a one-shot. At a certain time (tomorrow at 14:00) a job is started. Once. At schedule the task at only once.**

9. **What is the use of gzip and bzip2 commands?**

bzip	Bzip2
gzip is a widely used compression tool that uses the DEFLATE algorithm.	bzip2 is another compression tool that uses the Burrows-Wheeler Transform and Huffman coding algorithms.
It compresses each file individually, and adds a .gz extension to the file name.	It compresses each file individually, and adds a .bz2 extension to the file name.
gzip offers a good balance between speed and compression ratio, and is compatible with many platforms and programs.	bzip2 typically achieves a higher compression ratio than gzip, especially for text files, but it is also slower and more CPU-intensive.
It doesn't provide any inbuilt functionality or associate program to recover the damaged .gz files.	It doesn't provide any inbuilt functionality or associate program to recover the damaged .gz files.

10. **How to get only info related information in centralized logging?**
    In Linux, when you're using centralized logging systems (like Syslog, Rsyslog, Journalctl, or a custom log management solution) to aggregate logs, you typically want to filter out logs related to the INFO level from various sources. Here are some common approaches for achieving this, depending on the tools you're using for centralized logging in Linux.

    <u>1. Using journalctl (for systemd logs):</u>
    If your system uses systemd, logs are typically stored in journal logs, and you can filter them using journalctl. For example, to show only logs with INFO level:
    journalctl -p info
    -p: This option specifies the priority level.
    info: This filters the logs to show only those with the INFO priority level. You can also use other priority levels like err, warning, debug, etc.

    <u>If you want to see logs for a specific service, you can include the service name like so:</u>
    journalctl -u <service_name> -p info

    <u>For example, to get INFO logs for nginx:</u>
    journalctl -u nginx -p info

2.Using grep to Filter Logs from Files:

If your logs are stored in text files (like /var/log/syslog, /var/log/messages, or application-specific log files), you can use grep to filter out logs related to INFO.

For example, to show only INFO logs from the /var/log/syslog:
grep "INFO" /var/log/syslog
This will return only the lines containing the INFO string.

To filter for INFO logs from a specific service or application, you can further narrow it down by including the service name in the grep command.

For example, to get INFO logs from the nginx log file:
grep "INFO" /var/log/nginx/access.log

11. **What is the difference between yum and rpm?**

RPM and YUM are both package managers for Red Hat Linux distributions, but they have some key differences:

#Packaging format: RPM is a packaging format, while YUM is a command used to install packages.

#Standalone: RPM is a standalone package manager, while YUM is not. YUM uses the RPM python library for most of its operations.

#Package dependencies: RPM cannot resolve package dependencies, but YUM can.

#Automatic updates: YUM automatically upgrades packages.

#Multiple versions: YUM can install multiple versions of a package.

#Reverting versions: YUM allows users to revert to previous versions of a package.

#Ease of use: YUM is better for day-to-day use because it keeps the system clean and updated.

#Default package manager: RPM is the default extension for files used by the program, while YUM is the default package manager for CentOS.

12. **What is the use of the id command?**
The id command in Linux is used to display user and group information for the current user or for a specified user. It provides details such as the user's UID (User ID), GID (Group ID), and the groups to which the user belongs.

Syntax: id [OPTION] [USER]
Eg: id username
example output: uid=1000(user) gid=1000(user) groups=1000(user),27(sudo),30(dip)

13. **How to change the default runlevel?**

Systemctl set-defult multiuser.target

14. **What is the difference between adduser and useradd ?**

Feature	useradd	adduser
Command Type	Low-level system command	High-level script (usually a wrapper for useradd)
Interactivity	Non-interactive (requires options)	Interactive (asks for user input)
Default Behavior	Requires explicit options (e.g., -m for home directory, -s for shell)	Automatically creates home directory, sets shell, and prompts for password

Configuration Files	**Directly modifies /etc/passwd, /etc/shadow, etc.**	**Same, but with additional logic to handle default settings**
Availability	**Available on all Linux distributions**	**Primarily available on Debian-based systems (Ubuntu, etc.)**
Additional information	**doesnot ask for additional information such as full name, room number, work phone, and home phone.**	**may ask for additional information such as full name, room number, work phone, and home phone.**
Password Setup	**Does not set password; needs passwd command**	**Prompts for a password during user creation**
Usage	**Primarily used for scripting and system-level tasks**	**Primarily for creating users interactively in an easy way**

15. What is MBR?

MBR stands for Master Boot Record, and it is a critical part of the booting process in many computer systems, especially those using traditional BIOS-based systems (prior to UEFI). The MBR is a small but essential section of a storage device (like a hard drive or SSD) that contains information about the disk's partitions and is responsible for booting the operating system.

Components of the MBR:

- **Bootloader Code (446 bytes): The first 446 bytes of the MBR contain the bootloader, a small program that loads the operating system's boot code. This is the part responsible for starting the boot process.**
- **Partition Table (64 bytes): The next 64 bytes contain the partition table, which describes the layout of the disk, such as how the disk is divided into partitions (e.g., primary partitions, extended partitions, and logical partitions). The partition table supports up to four primary partitions.**
- **Boot Signature (2 bytes): The final 2 bytes in the MBR are a signature (0x55AA), which indicates that the MBR is valid.**

Points About MBR:

- **MBR (Master Boot Record) is located in the first sector of a hard drive or storage device (512 bytes in size).**
- **It contains a bootloader, a partition table, and a boot signature.**
- **The bootloader is responsible for starting the operating system or handing over control to another bootloader.**
- **MBR can support up to 4 primary partitions, and it is limited to disk sizes of 2 TB due to its 32-bit partitioning scheme.**
- **Legacy BIOS systems use MBR for booting and partitioning, but newer systems with UEFI generally use the GPT partitioning scheme instead.**

Limitations of MBR:

- **MBR supports up to four primary partitions on a disk. However, to overcome this limitation, one of these primary partitions can be an extended partition that can hold multiple logical partitions.**
- **The MBR has a 32-bit partition table, which means it can support disks with a size up to 2 TB (terabytes). Larger disks (above 2 TB) require a different partitioning scheme, such as GPT (GUID Partition Table).**
- **MBR is also limited to legacy BIOS-based systems. Modern systems using UEFI (Unified Extensible Firmware Interface) often use GPT instead of MBR due to its support for larger disks and more partitions.**

16. How to find files of a particular user in / file system?

Find Files Owned by a Particular User:

```
sudo find / -user username
```

17. How to find files which have the user as test and group as root?

```
find /path/to/search -user test -group root
```

18. What are the different types of files available to install applications on Linux?

Package Files (Binary Packages)

Debian-based Distributions (e.g., Ubuntu, Debian)

.deb files:

Purpose: .deb files are used in Debian-based distributions, including Ubuntu.

sudo apt install ./package.deb
Red Hat-based Distributions (e.g., RHEL, CentOS, Fedora)
.rpm files:

Purpose: .rpm (Red Hat Package Manager) files are used by Red Hat-based distributions, including CentOS and Fedora.
sudo yum install package.rpm

19. **Where does the process information is stored?**

In Linux, process information is primarily stored in the /proc directory. The /proc directory is a virtual filesystem (often called procfs) that provides detailed information about the system and its processes, running kernel, and other system parameters.

Important System Files in /proc/:

/proc/ Directory (Process Information)
/proc/[pid]/status: Human-readable information about a process (e.g., process state, memory usage).
/proc/[pid]/stat: Compact statistics about a process (e.g., PID, PPID, CPU time, memory).
/proc/[pid]/cmdline: Command line used to start the process.
/proc/[pid]/fd/: Directory containing symbolic links to files opened by the process.
/proc/[pid]/environ: Environment variables for the process.
/proc/[pid]/maps: Memory map of the process (addresses of all memory regions used).
/proc/[pid]/exe: A symbolic link to the executable of the process.

20. **How to view the background running process of a terminal?**
Using the jobs Command
The jobs command is used to list the background jobs associated with the current shell session (i.e., the processes started in the current terminal). It shows you jobs that have been started with & (background) or suspended and can be resumed.

Example:
Jobs
Example output:
[1] 12345 Running some_command &
[2] 12346 Stopped another_command
This will display a list of all background jobs running in the current shell, showing their job number, status (running, stopped, etc.), and the command that started them.

Using the ps Command:
show all processes (ps -aux): This command lists all processes running on the system, including background processes.

21. **What are nice and renice commands?**

The nice and renice commands allow users to adjust the scheduling priority of processes, influencing how the Linux kernel allocates CPU time among them.

'nice' Command: This command is used to start a new process with a specific priority, known as the "nice value." A higher nice value lowers the process's priority, while a lower (negative) nice value increases it. Processes with higher priority receive more CPU time.
Syntax: nice -n [value] [command]
The value can be between -20 (highest priority) and 19 (lowest priority).
the default is 0, which is the normal priority level.

Eg: nice -n -10 command_name
nice -n 10 top

'renice' Command: Unlike nice, which sets the priority when starting a process, renice modifies the priority of an already running process. This flexibility allows system administrators to manage process priorities based on the current system load dynamically.
Syntax: renice -n [value] -p [pid]
The value can be between -20 (highest priority) and 19 (lowest priority).

Eg: renice -n -5 -p 1234

#To check the nice value of a process:

ps -el | grep terminal

Important Notes:

- The nice value ranges from -20 to 19
- -20: Highest priority (process gets more CPU time).
- 0: Default priority (normal priority).
- 19: Lowest priority (process gets the least CPU time).
- Default Nice Value: Most processes, unless modified, run with a default nice value of 0. Root users can assign a nice value in the range of -20 to 19. Non-root users can assign nice values in the range of 0 to 19.

22. **What is the signal related to the keystroke ctrl+z ?**
    The keystroke Ctrl+Z in a terminal sends the SIGTSTP signal to the currently running process. This signal is used to suspend a process temporarily, effectively pausing its execution and placing it in the background. The process remains in a stopped state, and you can resume its execution later using commands like fg (to bring it back to the foreground) or bg (to continue it in the background).

    SIGTSTP (Signal 20): This is the signal number 20 in most Unix-like systems. The signal name stands for Terminal Stop.

23. **How to find the processes running on a particular port?**
    Using lsof (List Open Files)
    lsof is a powerful command that lists all open files and the processes that opened them. Since in Unix-like systems, network connections are considered files, you can use lsof to find out which process is using a particular port.

    Syntax: sudo lsof -i :<port_number>
    Eg: To find which process is using port 8080:
    sudo lsof -i :8080

24. **What is the use of signal 1 ?**
    Signal 1 in Unix-like systems is known as SIGHUP, which stands for "Hangup". It is a signal that was originally sent to a process when its controlling terminal (the terminal from which it was started) was closed or the session was terminated.
    Uses of SIGHUP (Signal 1)

    Terminal Hangup (Original Purpose):
    The SIGHUP signal was originally designed to notify processes when the terminal that was used to start them was closed (i.e., when the controlling terminal was "hung up"). This could happen if the user logged out, closed their terminal window, or if a network connection was dropped.

    Reloading Configuration Files:
    In modern usage, many daemon processes (background services) use SIGHUP to trigger a reload of their configuration files without fully restarting the process. This allows services to apply new settings without losing their current state or disrupting ongoing operations.
    To send SIGHUP to a process with a known PID:
    kill -1 <PID>.

25. **Write a crontab to backup my system (Using gzip) for every month, on Mondays whose dates are between 15-21 at 4 a.m.?**

    0 4 15-21 * 1 tar -czf /path/to/backups/backup_$(date +\%Y-\%m-\%d_\%H-\%M-\%S).tar.gz /home/user/backupfiles

26. **Write a crontab which executes the "ping" and the "ls" commands every 12am and 12pm on the 1st day of every 2nd month.**
It also has to put the output of the commands into a file /var/log/test.
0 0,12 1 2,4,6,8,10,12 * ping -c 4 8.8.8.8 >> /var/log/test 2>&1
0 0,12 1 2,4,6,8,10,12 * ls -l /some/directory >> /var/log/test 2>&1

27. **Where the user, password and group information is stored?**
In Linux and other Unix-like operating systems, user, password, and group information is stored in several key files, which are typically located in the /etc directory.

In Linux and other Unix-like operating systems, user, password, and group information is stored in several key files, which are typically located in the /etc directory. Here's an overview of where this information is stored and the contents of each file:

1. User Information: /etc/passwd
The /etc/passwd file contains information about each user on the system, including the user name, user ID (UID), group ID (GID), home directory, login shell, and more.

Structure of /etc/passwd:
Each line in the file represents a user and contains the following fields, separated by colons (:):

username:x:UID:GID:GECOS:home_directory:login_shell
username: The name of the user (e.g., alice, root).
x: This placeholder indicates that the password is stored elsewhere (in /etc/shadow for security).
UID: The user ID, a unique number assigned to the user (e.g., 1000).
GID: The group ID, which represents the user's primary group.
GECOS: A field that typically contains user information such as full name or contact info.
home_directory: The path to the user's home directory (e.g., /home/alice).
login_shell: The shell that is launched when the user logs in (e.g., /bin/bash).

Example of /etc/passwd entry:
alice:x:1001:1001:Alice User:/home/alice:/bin/bash
root:x:0:0:root:/root:/bin/bash

In Linux and other Unix-like operating systems, user, password, and group information is stored in several key files, which are typically located in the /etc directory. Here's an overview of where this information is stored and the contents of each file:

1. User Information: /etc/passwd
The /etc/passwd file contains information about each user on the system, including the user name, user ID (UID), group ID (GID), home directory, login shell, and more.

Structure of /etc/passwd:
Each line in the file represents a user and contains the following fields, separated by colons (:):
username:x:UID:GID:GECOS:home_directory:login_shell
username: The name of the user (e.g., alice, root).
x: This placeholder indicates that the password is stored elsewhere (in /etc/shadow for security).
UID: The user ID, a unique number assigned to the user (e.g., 1000).
GID: The group ID, which represents the user's primary group.
GECOS: A field that typically contains user information such as full name or contact info.
home_directory: The path to the user's home directory (e.g., /home/alice).
login_shell: The shell that is launched when the user logs in (e.g., /bin/bash).
Example of /etc/passwd entry:
ruby
Copy code
alice:x:1001:1001:Alice User:/home/alice:/bin/bash
root:x:0:0:root:/root:/bin/bash

2. Password Information: /etc/shadow
The /etc/shadow file is where the actual user passwords are stored in a hashed format for security purposes. This file is only accessible to the root user for security reasons.

Structure of /etc/shadow:
Each line corresponds to a user, with fields separated by colons (:):
username:password_hash:last_change:min_age:max_age:warning:inactive:expire_date
username: The user's name (same as in /etc/passwd).
password_hash: The hashed password (if the password is empty, it indicates the user has no password).
last_change: The date of the last password change (in days since January 1, 1970).
min_age: The minimum number of days between password changes.
max_age: The maximum number of days the password is valid.
warning: The number of days before the password expires that the user is warned.
inactive: The number of days after password expiration that the account is disabled.
expire_date: The date when the account will expire.
Example of /etc/shadow entry:
alice:6D2lR123F3t2sI5kQk1shFvHlK24LRfCZekV0C0v3zlhIgTtvf.YDb1S1Ojs6JqF0M6Fg7Q0A0AfV3pS9N6BBR1:18256:0:99999:7:::
root:$6$0hTp8Hdy9Q9NkFzYsPt6u1z2wE3zjpV1Jhz4z9Fq1Cz47i4Hngn0bZfbXhzTkN.k3UqwoRP5lETy6Nhb0A.Pd0:18335:0:99999:7:::

3. Group Information: /etc/group
The /etc/group file contains information about the groups on the system, such as the group name, group password (if any), group ID (GID), and a list of members.

Structure of /etc/group:
Each line in the file has the following format:
group_name:x:GID:user_list
group_name: The name of the group (e.g., users, admin).
x: This placeholder indicates that the group password is stored elsewhere (in the /etc/gshadow file, though this is rarely used).
GID: The group ID, which is a unique number assigned to the group.
user_list: A comma-separated list of users who are members of the group (e.g., alice,bob).
Example of /etc/group entry:
users:x:1001:alice,bob
admins:x:1002:root,alice

Exercise -4

1. **How to find files which have a user as test or group as root?**
   find /path/to/search -user test -group root

2. **How to find files which have 644 permissions?**
   find /path/to/search -type f -perm 644

3. **What is the use of the mount command?**

   The mount command in Linux is used to attach a file system (like a disk, partition, or network share) to a specific location in the directory tree. This location is called a mount point, and it allows users and applications to access the contents of the mounted file system as if it were part of the main directory structure.

   View Mounted File Systems: Running the mount command without any arguments displays a list of all currently mounted file systems, along with their mount points and file system types.
   $mount

Mounting a File System: The primary use of the mount command is to mount a file system to a directory (mount point), allowing you to access the files and directories contained in that file system. For example:
sudo mount /dev/sda1 /mnt
Unmounting a File System: The mount command itself doesn't unmount file systems, but it works in tandem with the umount command, which is used to unmount file systems. For example:
sudo umount /mnt

Mounting Different Types of File Systems:
sudo mount -t ntfs /dev/sdb1 /mnt
sudo mount -t nfs server:/exported/path /mnt

4. **What is the maximum length of a filename? Prove it**
   The maximum length of a filename in Linux (and most Unix-like operating systems) is determined by the file system used. For EXT-based file systems, such as ext4, the maximum filename length is:

   255 characters.

   touch $(printf 'a%.0s' {1..255})
   This command creates a file with a name consisting of 255 characters (a repeated 255 times).

   or
   getconf NAME_MAX /
   getconf is a command-line utility in Unix-like systems used to query system configuration variables, such as limits and system-specific constants.
   NAME_MAX is a system constant that defines the maximum length of a filename

5. **What is su and sudo?**
   <u>su (Substitute User):</u>
   Purpose: The su command allows a user to switch to another user account and execute commands as that user. By default, when used without any options, it switches to the root user (superuser).

   <u>Syntax</u>: su username
   Eg : su Irshaq
   Su //switches to root user

   <u>sudo (SuperUser Do):</u>
   Purpose: The sudo command allows a user to execute a single command with superuser (root) privileges without having to log into a new shell as root. It uses the current user's password, rather than the root user's password.
   The system uses the sudoers file (/etc/sudoers) to control who can execute which commands and whether a password is required.
   <u>Syntax</u>: sudo command
   Eg: sudo yum install nfs

6. **What is the difference between "su" and "su -" commands?**
   <u>su:</u>
   - **Does not load root's environment.**
   - **The current environment is maintained, which can be useful if you want to execute commands with root privileges without changing the entire environment.**
   - **does not invoke login scripts like .bash_profile**
   - Your **current user's home directory** remains ($HOME does not change).

   <u>su - :</u>
   - **Loads the target user's login environment.**
   - **Starts a login shell, as if the user logged in directly.**
   - **Root's environment (or the specified user's environment) is set, making the environment more consistent with that of a real login.**
   - **does invoke login scripts like .bash_profile**
   - Changes to the **target user's home directory** (usually root's /root).

7. **How to mount a pen drive on linux?**
   To mount a pen drive (USB drive) on Linux, you can follow these steps:

   1. Insert the Pen Drive

   2. Identify the Pen Drive
   lsblk
   sdb 8:16 1 16G 0 disk
   └─sdb1 8:17 1 16G 0 part /media/usb

   3.Create a Mount Point:
   sudo mkdir /mnt/usb

   4.Mount the Pen Drive:
   sudo mount /dev/sdb1 /mnt/usb

8. **How to list the currently logged in users in your system in Linux?**
   who Command
   The who command provides information about the users currently logged in to the system, including their usernames, terminal, login time, and IP addresses

   $who
   user1 tty1 2024-11-07 08:00 (:0)
   user2 pts/0 2024-11-07 08:15 (:0)

   The w command
   the w command provides more detailed information about the currently logged-in users, including their login time, idle time, what they're doing, and more.
   08:20:15 up 2:13, 2 users, load average: 0.08, 0.07, 0.02
   USER TTY FROM LOGIN@ IDLE JCPU PCPU WHAT
   user1 tty1 :0 08:00 2.00s 0.05s 0.00s -bash
   user2 pts/0 :0 08:15 3.00s 0.01s 0.00s w

   users Command
   The users command provides a simple list of the usernames
   Eg: $users
   user1 user2

   USER: The username of the logged-in user.
   TTY: The terminal or session the user is connected to.
   FROM: The remote host or IP address (if applicable).
   LOGIN@: The login time.
   IDLE: The idle time (how long the user has been inactive).
   JCPU: The total CPU time used by all processes in the user's session.
   PCPU: The CPU time used by the user's current process.
   WHAT: The command the user is currently running (e.g., bash, w).

9. **How to know the uptime of your linux system?**
   uptime Command
   The most straightforward way to check the system uptime is by using the uptime command:
   $uptime
   08:25:31 up 2 days, 3:10, 3 users, load average: 0.01, 0.05, 0.01

Explanation:

08:25:31: Current system time.
up 2 days, 3:10: The system has been up for 2 days and 3 hours, 10 minutes.
3 users: The number of users currently logged into the system.
load average: 0.01, 0.05, 0.01: The system load averages for the past 1, 5, and 15 minutes.

Or
$top commnd

10. **How to list only directories?**
    ls -d */: List only directories in the current directory.
    find . -type d: Find and list all directories starting from the current directory.
    tree -d: List directories in a tree format (if tree is installed).
    echo */: Quick list of directories using a wildcard.

11. **List all the shortcuts related to vi?**
    Vi editor shortcuts:

Switch Between Modes:

- **i : Enter Insert Mode before the cursor position.**
- **I : Enter Insert Mode at the beginning of the line.**
- **a : Enter Insert Mode after the cursor position.**
- **A : Enter Insert Mode at the end of the line.**
- **o : Open a new line below the current line and enter Insert Mode.**
- **O : Open a new line above the current line and enter Insert Mode.**
- **Esc : Exit Insert Mode and return to Command Mode.**

Movement Commands:

- **0 (zero) : Move to the beginning of the current line.**
- **^ : Move to the first non-blank character of the line.**
- **$: Move to the end of the line.**
- **gg : Go to the first line of the file.**
- **G : Go to the last line of the file.**
- **:n : Go to line n (replace n with the line number).**
- **Ctrl + f : Move one page forward.**
- **Ctrl + b : Move one page backward.**

Text Editing Commands:

- **dd : Delete the current line.**
- **yy : Yank (copy) the current line.**
- **p : Paste the yanked/deleted text after the cursor.**
- **u : Undo the last change.**

Search and Replace:

- **/pattern : Search forward for the specified pattern.**
- **:%s/old/new/g : Replace all occurrences of old with new throughout the file.**

File Operations:

- **:w : Save (write) the file.**
- **:w filename : Save the file as filename.**
- **:q : Quit (exit) vi.**
- **:q! : Quit without saving changes.**
- **:x or ZZ : Save and quit the file.**
- **:wq : Save and quit the file.**

12. **What is the use of diff command?**
The diff command is a powerful tool for comparing files in Linux and Unix-like systems. It is commonly used for tasks such as checking for changes between versions of files, comparing configuration files,

Syntax: diff [options] file1 file2
Example: diff file1.txt file2.txt

The default output format uses a set of line numbers and symbols like +, -, and c to indicate differences:
-: Lines that are in file1 but not in file2 (lines to be deleted).
+: Lines that are in file2 but not in file1 (lines to be added).
c: Indicates a change between the files.

Eg:
$ diff file1.txt file2.txt
2c2
< Hello world

> Hello there
4a5
> This is a new line

2c2: Line 2 in file1.txt is different from line 2 in file2.txt (change).
< Hello world: Line in file1.txt.
> Hello there: Line in file2.txt.
4a5: Line 4 of file1.txt is followed by an additional line 5 in file2.txt (added line).
> This is a new line: A new line in file2.txt.

13. **In which file all the shell information is available?**
Files Containing Shell Information:

- **/etc/passwd: Contains the default shell for each user.**
- **/etc/shells: Lists all valid login shells.**
- **~/.bashrc: User-specific configurations for interactive shells (Bash).**
- **~/.bash_profile or ~/.profile: User-specific configurations for login shells (Bash).**
- **/etc/profile: Global settings for login shells.**
- **/etc/bash.bashrc: Global settings for non-login shells (Bash).**
- **~/.zshrc, ~/.zprofile, ~/.config/fish/config.fish, etc.: Configuration files for other shells (Zsh, Fish).**

14. **What does ~ represent?**
~: Refers to the home directory of the **current user**.

15. **What does? and .. represents?**
. : Refers to the current directory.
Eg: ls .
.. : Refers to the parent directory (one level up in the directory tree).
Eg: cd ..

16. **What is the use of | operator?**
The | (pipe) operator allows you to connect the output of one command to the input of another, creating a chain of commands. This enables powerful command-line operations, like filtering, counting, sorting, or transforming data in a stream without creating temporary files.
Syntax: command1 | command2
Eg: ls | wc –l

17. **How can I store the output of a command to a file?**
>: Redirects the output to a file, overwriting the file if it exists.
command > filename
>>: Redirects the output to a file, appending the output to the file if it exists.
command >> filename

tee: Allows you to store the output to a file while simultaneously displaying it on the terminal.
command | tee filename

18. **What is the use of the tee command?**
The tee command is a powerful tool for viewing and saving output simultaneously, especially useful in situations where you need to log output for later review while still seeing it in real-time. It's commonly used in scripting, debugging, and monitoring tasks, as well as for log management.

Syntax: command | tee [options] file
Eg: ls -l | tee output.txt
Options:
-a: Append to the file, instead of overwriting.

19. **What is the difference between locate, find and slocate?**

- **find is the most flexible and powerful tool, as it searches in real-time and supports many different filtering options, but it can be slow for large directories or filesystems.**
- **locate is fast because it uses a pre-built database, but it may miss newly created or modified files unless the database is updated regularly using updatedb.**
- **slocate is similar to locate, but it has added security features, ensuring that users only see files they have permission to access, making it more secure than locate.**

Exercise -5

1. **What does scp command do?**
The scp (Secure Copy Protocol) command in Linux is used to securely copy files or directories between two systems over a network. It uses SSH (Secure Shell) for data transfer, ensuring that the files are encrypted during the transfer to maintain privacy and security.

Format for Specifying Remote Locations:
username@remote_host:/path/to/remote/file
eg:user@192.168.1.100:/home/user/

Copying a file from local to remote:
scp localfile.txt user@remotehost:/remote/directory/

Copying a file from remote to local:
scp user@remotehost:/remote/file.txt /local/directory/

Copying between two remote systems:
scp user1@remote1:/path/to/file user2@remote2:/path/to/destination

eg: scp /home/user/file.txt

2. **How to lock a user?**
sudo passwd -l username
The -l option stands for "lock."

Locking a user account with passwd -l adds an exclamation mark (!) to the beginning of the user's encrypted password in the /etc/shadow file.

irshaq:!6Zp6m...:17845:0:99999:7:::

Unlock a User:
sudo passwd -u username

3. **Explain the entries in /etc/passwd and /etc/shadow files?**

   /etc/passwd File Example:

   ```
   irshaq:x:1000:1000:Trainee:/home/irshaq:/bin/bash
   ```

   - **The username (irshaq).**
   - **x. The encrypted password stored in the /etc/shadow file.**
   - **UID (1000). The user ID (UID)**
   - **GID (1000). The Group ID (GID) refers to the user's primary group. Secondary groups are listed in the /etc/groups file.**
   - **Comment: trainee the comment field containing additional information about the user.**
   - **The home directory (/home/irshaq).**
   - **The default shell (bin/bash).**

   /etc/shadow File example.

   ```
   irshaq:$6$YTJ7JKnfsB4esnbS$5XvmYk2.GXVWhDo2TYGN2hCitD/wU9Kov.uZD8xsnleuf1r0ARX3qodIKiDsdoQA444
   b8IMPMOnUWDmVJVkeg1:19446:0:99999:7:::n/bash
   ```

   - **Username. User account and login name that exist in the system.**
   - **Encrypted password. Password using the format $type$salt$hashed and eight to 12 characters long.**
   - **Last password change. Date since Jan. 1, 1970, when the password was last changed.**
   - **Minimum password age. The minimum number of days that must elapse before the password can be changed by the user.**
   - **Maximum password age. The number of days after which the password must be changed.**
   - **Warning period. The number of days before the password expires, during which time the user gets a warning to change the password.**
   - **Inactivity period. The number of days post-expiration -- since Jan. 1, 1970 -- before the user's account is disabled.**
   - **Expiration date. The date on which the account was disabled.**
   - **Unused. This field is left empty and reserved for future use.**

4. **How to check the dependencies of a particular package?**
   yum deplist <package-name>
   rpm -qR <package-name>
   dnf repoquery --requires <package-name>

5. **How to check the version of files installed by a package?**
   rpm -qi <package-name>
   dnf info <package-name>
   yum info <package-name>

6. **How to view the exit status of the previous command executed?**

   Echo $?

7. **What is the use of NFS service?**

- **the key uses of the NFS (Network File System) service:**

- **Remote File Access: Allows access to remote files as if they were on the local system.**

- **File Sharing: Enables file sharing across multiple systems in a network.**

- **Centralized Data Management: Provides a central location for storing and managing files, reducing redundancy.**

- **Home Directory Sharing: Centralizes user home directories for easy access from any machine on the network.**
- **Backup and Restore: Used for centralized backups by storing data on a central server.**
- **Simplified File Access in Distributed Systems: Enables efficient file access for distributed applications and systems.**
- **Cost-effective Storage: Reduces local storage requirements by accessing data from a centralized server.**
- **Distributed File Systems: Supports distributed environments where multiple clients need access to the same files.**

8. **What is NFS and on Which port does NFS run? What are the configuration files related to NFS?**

- **Port 2049: The primary port used by NFS for client-server communication (for NFSv3 and NFSv4).**
- **Port 111: Used by rpcbind (portmapper) to help NFS clients locate the NFS service.**
- **Configuration Files:**
- **/etc/exports: Defines which directories are shared by the NFS server and with which clients.**
- **/etc/fstab: Configures persistent NFS mounts on the client system.**
- **/etc/exports.d/: Optional directory for managing NFS exports in separate files.**
- **/etc/nfs.conf: Contains configuration options for NFS services.**
- **/etc/hosts.allow and /etc/hosts.deny: Used for access control to NFS services.**
- **/etc/sysconfig/nfs: Configuration file for NFS service on Red Hat-based systems.**

9. **What is the use of chmod and chown commands?**

<u>chmod (Change Mode):</u>
Purpose: Modify file permissions.
Syntax: chmod <permissions> <file/directory>
Permissions: Can be set using symbolic (rwx) or numeric (0-7) modes.
Example: chmod 755 file.txt (Owner: rwx, Group: rx, Others: rx).

<u>chown (Change Owner):</u>
Purpose: Change the owner and/or group of a file or directory.
Syntax: chown <owner>:<group> <file/directory>
Example: chown john:admins file.txt (Owner: john, Group: admins).
Both chmod and chown are powerful commands that give you control over access and ownership of files and directories, helping you manage file security in a Linux system.

Exercise -6

1. **What is the use of the netstat command?**

The netstat command is a network utility used to display information about network connections,
netstat is a powerful tool for examining the state of the network on a machine, troubleshooting connectivity issues, and monitoring network activity.

<u>netstat Options:</u>

- **-a : Show all connections (both listening and non-listening).**
- **-t : Show TCP connections.**
- **-u : Show UDP connections.**
- **-l : Show only listening sockets.**
- **-n : Show numerical addresses instead of resolving hostnames.**
- **-p : Show the process ID (PID) and the name of the program using the connection.**
- **-r : Show the routing table.**
- **-s : Show network statistics (packets, errors, etc.).**

2. **What is the use of profiles?**

profiles refer to configuration files or settings that determine the environment and behavior for users or processes. These profiles are often used to customize and manage user settings, system-wide configurations, and process environments. Profiles can be associated with user environments, shell settings, system behaviors, and network configurations.

System-Wide Profiles:

These profiles are applied globally to all users and affect system-wide configurations. These profiles are typically set in system-wide configuration files that configure the behavior of the shell, environment variables, and system-wide settings.

Common Files:

- **/etc/profile: This file is sourced for login shells. It can set global environment variables, configure user paths, and set up default shell settings for all users.**
- **/etc/bash.bashrc: This file is sourced for interactive non-login shells. It contains system-wide shell configurations, aliases, and other settings.**
- **/etc/environment: This file is used to set global environment variables that apply to all users and processes.**
- **/etc/login.defs: This file contains default settings for user logins, such as password expiration and minimum password length.**
- **/etc/profile.d/: This directory contains additional shell scripts that are sourced from /etc/profile. Administrators can place scripts here to apply settings globally, such as adding custom environment variables or configuring third-party software.**

User Profiles (User Configuration Files):

In Linux, user profiles typically refer to configuration files located in the user's home directory. These files define environment variables, user-specific settings, and shell behavior for each user account.

Common Files:

- **~/.bash_profile or ~/.profile: These files are executed when a user logs into a shell (for login shells). They are often used to set environment variables (like PATH, EDITOR, etc.), configure the shell prompt, and run startup scripts.**
- **~/.bashrc: This file is executed every time a user opens a new terminal (for non-login shells). It is commonly used to set aliases, functions, and other shell-related settings.**
- **~/.bash_logout: This file is executed when the user logs out of a session and can be used for cleanup tasks.**

They help set up:

- **User-specific settings: Environment variables, aliases, shell customizations.**
- **System-wide settings: Global configurations affecting all users (e.g., paths, user limits, shell behavior).**

example:

```
alias ll='ls -l'
export EDITOR=nano
HISTSIZE=1000
```

3. **What is the use of environment variables?**

Purpose of Environment Variables:

- **Store system or user-specific configuration data.**
- **Control the behavior of processes and applications.**
- **Allow customization of system and user preferences.**

Common Uses:

- **Defining system paths (PATH, HOME, SHELL).**
- **Configuring user-specific settings (EDITOR, LANG).**
- **information about the current user, the session, and the system (USER, UID, SHELL)**
- **Storing sensitive information for applications (e.g., AWS_ACCESS_KEY_ID).**
- **Managing process behavior and security.**

How to Set and View:

- **Temporarily: export VAR_NAME=value**
- **View: echo $VAR_NAME**
- **Make permanent: Edit configuration files like ~/.bashrc, /etc/environment.**

By leveraging environment variables, you can configure your system, manage user preferences, and streamline the behavior of both system processes and applications.

4. **What are the types of environment variables available and give some examples?**
common types of environment variables in Linux:

1. User-Specific Environment Variables:
These variables are set for individual users and are typically stored in files like ~/.bashrc, ~/.bash_profile, ~/.profile, or ~/.zshrc.
Examples:
- **HOME: User's home directory.**
- **USER: The name of the current logged-in user.**
- **SHELL: The current shell being used (e.g., /bin/bash).**
- **PATH: Directories where executable programs are located.**

2. System-Wide Environment Variables:
These variables are set for all users on the system and are typically defined in system files like /etc/profile, /etc/environment, or /etc/bash.bashrc.
Examples:
- **PATH: A colon-separated list of directories in which the shell looks for executable files.**
- **LANG: The default language/locale setting (e.g., en_US.UTF-8).**
- **TZ: Timezone setting.**
- **HOME: Default home directory for all users.**

3. Shell-Specific Environment Variables:
These are variables specific to a particular shell session. They are often set within the shell configuration files and control aspects of shell behavior.
Examples:
- **PS1: The primary prompt string (e.g., \u@\h:\w\$).**
- **BASH_VERSION: The version of the Bash shell.**
- **HISTSIZE: The number of commands to save in the history file.**

4. System Environment Variables:
These variables are set by the system or administrators to configure important system-wide settings and processes.
Examples:
- **ROOT: The root directory of the file system.**
- **LD_LIBRARY_PATH: Directories where the dynamic linker looks for shared libraries.**
- **CFLAGS: Compiler flags used by programs when building from source.**

5. Application-Specific Environment Variables:
These variables are used by specific software applications or services to configure their operation.
Examples:
- **JAVA_HOME: The directory where the Java runtime environment is installed.**
- **PYTHONPATH: A list of directories to be searched for Python modules.**
- **NODE_ENV: The environment in which a Node.js app is running (e.g., development, production).**

6. Session-Specific Environment Variables:
These variables exist only for the duration of a user's session and are typically set upon login. They are often inherited from parent processes and can be customized per session.
Examples:
- **DISPLAY: The display to use for graphical programs (e.g., :0 for local display).**
- **SSH_CONNECTION: Information about the SSH connection.**
- **XDG_SESSION_ID: An identifier for the user session, used in desktop environments like GNOME or KDE.**

7. Exported Variables:
Variables that are "exported" are passed from the shell to all child processes, making them available to any applications or scripts run from the shell. This can be done using the export command.

Example:

- **export VAR=value: Sets and exports VAR to all child processes.**

8. Temporary (Non-Persistent) Environment Variables:

These variables are set during the session (or until the terminal or process is closed) and do not persist across reboots or new shell sessions.

Examples:

- **Setting a variable directly in the shell: VAR=value**
- **These will only exist for the current shell session and will not be saved after logging out or closing the terminal.**

9. Environment Variables for Network Configuration:

These are used to configure networking and related services on the system.

Examples:

- **http_proxy: Set the proxy server for HTTP requests.**
- **ftp_proxy: Set the proxy server for FTP requests.**
- **NO_PROXY: A list of domain names to bypass the proxy server for.**

10. Filesystem Environment Variables:

These variables deal with filesystem paths and locations.

Examples:

- **PWD: Current working directory.**
- **OLDPWD: The previous working directory.**
- **LOGNAME: The name of the user currently logged in.**

5. **What are the configuration files related to profils?**
   - **~/.bashrc**
   - **~/.bash_profile**
   - **~/.profile**
   - **~/.bash_logout**
   - **System-Wide Profile Files:**
   - **/etc/profile**
   - **/etc/.bashrc**
   - **/etc/login.defs**
   - **/etc/profile.d/**
   - **/etc/skel/**

6. **What is the use of /etc/profile file?**
   /etc/profile is a system-wide configuration file for login shells that sets environment variables and shell settings for all users.
   - **It is executed when a user logs into the system via a terminal or remotely (e.g., SSH).**

7. **What is the use of /etc/bashrc file?**
   /etc/bashrc is a system-wide configuration file for interactive non-login shells. It defines global aliases, functions, and
   - **shell settings that apply to all users when they open a new terminal session.**

8. **What is the use of .bash_profile file?**
   .bash_profile is a user-specific configuration file for login shells that sets up environment variables, startup programs, and other shell settings.
   - **It is executed when a user logs into the system via a terminal or remotely (e.g., SSH).**

9. **What is the use of .bashrc file?**
   .bashrc is a configuration file for Bash shells that sets up user-specific environment variables, aliases, and shell settings for interactive non-login shells.
   - **It is executed each time a new terminal session starts.**

10. What is the use of DISPLAY variable?

The DISPLAY environment variable in Linux is used to specify the X Window System display server to which graphical applications should send their output. This variable helps graphical applications determine where to render their graphical user interface (GUI).

Purpose of the DISPLAY Variable:

When you run a graphical application, it needs to know which display server (i.e., screen) to use to show its window. The DISPLAY variable provides this information by pointing to the appropriate X server and display.

The general format of the DISPLAY variable is:

<hostname>:<display_number>.<screen_number>

Default Display: :0 (the first screen on the local machine).

<hostname>: The name of the machine running the X server. it will default to localhost else the hostname.

<display_number>: The display number, which is a unique number that identifies a specific X server instance. If you're running only one graphical session, this is typically 0, but you could have multiple X servers running (e.g., for remote connections or different sessions).

<screen_number>: The screen number within that display. Most systems use 0, indicating the primary screen, but additional screens could have other numbers (like 1, 2, etc.), especially in multi-monitor setups.

11. What are the differences between system defined and user defined variables?

Characteristic	System Environment Variables	User Environment Variables
Definition	Predefined by the system to configure system-wide settings	Defined by users for personal session configurations
Scope	System-wide (affects all users)	User-specific (affects only the defining user)
Configuration	Set by system administrators or the operating system	Set by individual users
Persistence	Persistent across sessions and reboots	Persistent only for the current session (unless defined in startup files)
Modification Rights	Generally read-only for regular users, only root can modify	Users can modify freely
Examples	PATH, HOME, USER, SHELL, HOSTNAME	EDITOR=nano, MY_PROJECT_PATH=/home/user/projects
Functionality	Critical for system operations and system-wide configurations	Customizes user's environment or application settings
Access	Accessible by all users and processes on the system	Accessible only to the defining user and their processes
Location	/etc/environment, /etc/profile	~/.bashrc, ~/.profile

12. Which environment variable stores CPU architecture?

HOSTTYPE: The environment variable that stores the CPU architecture in Linux is typically HOSTTYPE

It might store values like x86_64 for a 64-bit Intel or AMD processor, i686 for a 32-bit Intel or AMD processor, or other architectures like armv7l for ARM-based CPUs.

command:

echo $HOSTTYPE

13. What is the use of LD_LIBRARY_PATH env variable?

The LD_LIBRARY_PATH environment variable in Linux is used to specify a list of directories where the system should look for dynamic libraries (shared libraries) at runtime. It essentially tells the dynamic linker (ld.so) where to find the libraries needed by programs during execution.

Purpose of LD_LIBRARY_PATH:

Dynamic Library Search Path:

When a program is executed, it may depend on shared libraries (e.g., .so files in Linux). The LD_LIBRARY_PATH variable provides a way to specify additional directories where these shared libraries can be found, aside from the standard library directories like /lib or /usr/lib.

Override Default Search Paths:
By setting LD_LIBRARY_PATH, users or system administrators can override the default library search path and include custom locations where libraries are stored. This is useful when you have libraries in non-standard locations or need to use specific versions of libraries without modifying system-wide settings.

Development and Debugging:
During development or debugging, you may want to link to a specific version of a library that is not in the default library directories. By adjusting LD_LIBRARY_PATH, you can ensure that the correct version of the library is loaded.

Running Custom or Third-Party Software:
When running third-party software that requires libraries in non-standard directories, you can set LD_LIBRARY_PATH to include those directories, ensuring the software runs correctly.

Format of LD_LIBRARY_PATH:

LD_LIBRARY_PATH=/home/user/mylibs:/usr/local/lib:/opt/libs

- **/home/user/mylibs: A custom directory where libraries are stored.**
- **/usr/local/lib: A standard directory for locally installed libraries.**
- **/opt/libs: Another custom library directory.**

The linker will search these directories in the order specified when trying to find the required libraries.

14. **What are the different types of manual pages available?**
In Linux and other Unix-like operating systems, manual pages (or man pages) provide detailed documentation on commands, system calls, configuration files, libraries, and other system components.
They are categorized into several types, each serving a different purpose. The man pages are divided into sections, and each section covers a specific category of information.

Here are the main types of manual pages available, categorized by their section:
The manual is split into 8 sections,

Section	Title	Description	Examples	Usage	Example Command
1	User Commands	Documentation for user-level commands that can be run from the shell (terminal).	ls, cp, grep, cat, chmod, mkdir, find	Commonly used by all users to perform tasks in the terminal.	man 1 ls
2	System Calls	Documents system calls that provide the interface between user programs and the Linux kernel.	open, read, write, fork, execve, ioctl	Primarily used by developers for low-level system programming.	man 2 open
3	Library Functions	Contains documentation for C library functions and other programming libraries.	printf, malloc, free, strcpy, strcmp	Used by developers to understand and utilize functions in system libraries.	man 3 printf
4	Special Files	Describes special files (e.g., device files) in /dev that represent devices and system resources.	/dev/null, /dev/sda, /dev/tty	Useful for interacting with hardware or virtual devices through the command line.	man 4 termios
5	File Formats and Conventions	Provides information about file formats and system conventions (e.g., configuration files).	/etc/passwd, /etc/fstab, /etc/hostname	Helps users understand system configuration files and their formats.	man 5 passwd

6	Games and Screensavers	Documentation for games and screensavers on the system.	nethack, xclock, fortune	Of interest to users who enjoy games or want to learn about fun utilities.	man 6 nethack
7	Miscellaneous Information	Covers a wide range of topics including macros, conventions, standards, and protocols.	man, roff, groff, tar file formats	Useful for understanding system-wide conventions or general knowledge (e.g., man-page formatting).	man 7 man
8	System Administration Commands	Documentation for commands related to system administration and maintenance (e.g., requires root).	reboot, shutdown, useradd, systemctl, ifconfig	Used by system administrators to manage and maintain the system.	man 8 shutdown
9	Kernel Routines	Documents functions and routines related to the Linux kernel.	Kernel API functions, internal kernel calls	Relevant for kernel developers or users working with kernel modules or kernel development.	man 9 init

options for the man command:

-a Show all available manual pages for a command or topic. [man –a printf]
-c Check the man page file for integrity. [man -c]
-f Display the name of the manual page for a specified topic. [man -f grep]
-h, --help Display help message.
-k Search the man database for a keyword.
-K Search all the man pages on the system for a keyword. [man -k copy]
-w Display the location of the man page file. [man -w ls]

15. What How to match empty lines using sed?

Sed '/^$/' FileName

16. What is the use of wild cards?

In Linux, wildcards are integral tools for pattern matching, enabling efficient handling of files and directories. These special characters allow users to simplify operations such as searching, listing, or manipulating multiple files with minimal effort. Below, I outline the primary uses of each wildcard in detail:

Character	Description	Example
*	Matches any number of characters. You can use the asterisk (*) anywhere in a character string.	wh* finds what, white, and why, but not awhile or watch.
?	Matches a single alphabet in a specific position.	b?ll finds ball, bell, and bill.
[ ]	Matches characters within the brackets.	b[ae]ll finds ball and bell, but not bill.
!	Excludes characters inside the brackets.	b[!ae]ll finds bill and bull, but not ball or bell. Like "[!a]*" finds all items that do not begin with the letter a.
-	Matches a range of characters. Remember to specify the characters in ascending order (A to Z, not Z to A).	b[a-c]d finds bad, bbd, and bcd.
#	Matches any single numeric character.	1#3 finds 103, 113, and 123.

17. **Define awk?**
    awk is a powerful programming language and command-line utility used primarily for pattern scanning and processing. It is commonly used for working with text files, particularly for tasks like searching, extracting, transforming, and reporting on data that is structured in a line-by-line or column-based format, such as CSV files or log files.
    Key features of awk:

- **Pattern matching: awk processes text based on patterns or conditions. You can specify patterns (regular expressions or conditions) that must be met for an action to be performed.**
- **Field-based processing: awk treats each line of text as a series of fields (typically delimited by spaces or tabs), and actions can be performed on individual fields (columns).**
- **Built-in variables: awk has built-in variables such as $0 (the entire line), $1 (the first field), $2 (the second field), and so on, which can be used in actions.**
- **Text processing and manipulation: awk is great for modifying text, summing numerical data, formatting output, and other operations.**

18. **How to delete the last line using sed?**

    Sed '4 d' filename

19. **Write a sed command to transform one set of characters in mass to another set which we do using tr?**

    echo "hello world" | sed 's/[a-z]/\U&/g'

20. **What is the operator used to print line numbers in sed?**

    Sed '=' FileName

21. **How can we refer to fields in awk script?**
    In an awk script, fields are referred to using the $ symbol followed by the field number. By default, awk splits input lines into fields based on whitespace (spaces or tabs). The fields are numbered starting from 1.

    Syntax to refer to fields:
    $1: First field
    $2: Second field
    $3: Third field
    $NF: Last field
    $0: The entire line (all fields combined)

22. **How can we concatenate strings in awk?**
    awk '{print $1 $2}' filename

23. **Which Awk variable is used to access environment variables?**
    In awk, the ENVIRON array is used to access environment variables. The ENVIRON array allows you to read the values of environment variables within an awk script or command.

    Syntax: awk ENVIRON["VARIABLE_NAME"]

    Example:If you want to access the environment variable HOME, you can use the ENVIRON array like this:
    awk 'BEGIN { print ENVIRON["HOME"] }'
    This will print the value of the HOME environment variable.

    Example 2: Accessing multiple environment variables:

```
awk 'BEGIN {
   print "Home Directory: " ENVIRON["HOME"];
   print "User: " ENVIRON["USER"]
}'
```

    output :
    Home Directory: /home/username
    User: username

24. What is the awk function to read lines from standard input?
to read lines from standard input is the getline function.
Syntan: getline var
getline reads the next input line and assigns it to the variable var.
If no variable is provided, it assigns the line to the default variable $0 (the entire line).

awk '{ getline x; print x }' filename

Reading into $0 (default)
awk '{ getline; print $0 }' filename

25. How to change field separators that awk uses?
FS(Field separator)

Awk FS, {print $3} FileName