UNIX & SHELL PROGRAMMING

SHORT QUESTIONS

1) **What is difference between $* and $#?**
   - *$** (dollar-asterisk) represents all the command-line arguments passed to a script or function as a single string.
   - *$#* (dollar-pound) represents the number of command-line arguments passed to a script or function.
2) **List out the logical blocks in Unix file system?**
   - Logical Blocks in Unix File System:
     In the context of the Unix file system, "logical blocks" typically refer to fixed-size blocks of data used for disk storage and management. These are not the same as logical control structures in shell scripting. If you're referring to logical structures within the Unix file system, they would include items like:
   - Superblock: Contains metadata about the file system.
   - Inode: Represents a file or directory and stores metadata.
   - Data Blocks: Contain the actual data of files.
3) **Define Major number and Minor number.**
   - Major Number and Minor Numbe*:
     In Unix-like operating systems, major and minor numbers are used to identify and manage device drivers.
   - Major Number: Identifies the type of device or driver.
   - Minor Number: Specifies a particular instance or unit of that device type.
4) **What is the significance of pwd command in Unix?**
   - pwd Command:
     The `pwd` command in Unix stands for "print working directory." It displays the current directory's absolute pathname. It's useful for verifying your current location within the directory structure.
5) **What happens when user executes following commands: cd (without any argument) and cd ..?**
   - cd Command:
   - `cd` without any argument changes the current directory to the user's home directory.
   - `cd ..` moves up one directory level (to the parent directory).
6) **List the input mode commands of vi screen editor.**
   - . *Input Mode Commands of vi*:
     In the vi text editor, some common input mode commands include:
   - `i`: Enter insert mode before the cursor.
   - `I`: Enter insert mode at the beginning of the current line.

- `a`: Enter insert mode after the cursor.
- `A`: Enter insert mode at the end of the current line.

7) What is the use of /etc directory in Unix?

- /etc Directory:
  The `/etc` directory in Unix/Linux contains system-wide configuration files and initialization scripts. It stores configuration settings and information required for the proper functioning of the operating system and installed software.

8) What happens when user executes following commands: cat (without any argument), cat > file1 and cat file1

- cat Commands:
- `cat` (without any argument) displays standard input. It's used to concatenate and display files or standard input.
- `cat > file1` creates a new file named `file1` and allows you to input text into it.
- `cat file1` displays the content of `file1`

9) What is the use of tee command?

- tee Command:
  The `tee` command reads from standard input and writes to standard output and files simultaneously. It's often used in pipelines to both display data on the screen and save it to one or more files.

10) What is the purpose of super block in Unix file system?

- Super Block:
  The superblock in Unix file systems contains critical metadata about the file system, such as the file system type, size, block size, free blocks, and more. It's essential for mounting the file system and performing various administrative tasks on it.

11) What is the use of /bin directory in Unix?

- /bin Directory in Unix:
  The `/bin` directory in Unix contains essential binary executable files that are required for the system's basic operations, even during system recovery. These binaries are necessary for system booting and repair and are typically available in single-user mode. Common commands like `ls`, `cp`, `mv`, and `rm` are found in this directory.

12) What is exit status?

- Exit Status:
  Exit status, also known as the return code or exit code, is a numerical value returned by a Unix command or script to indicate its termination status. A zero (0) exit status typically signifies successful execution, while

non-zero values indicate an error or some form of failure. These codes are useful for scripting to make decisions based on the success or failure of previous commands.

13) **List out various types of file in Unix.**

- Various Types of Files in Unix:
  Unix categorizes files into various types, including:
  Regular Files: Contain data (e.g., text files, binary files).
- Directories: Containers for other files and directories.
- Symbolic Links: Pointers or references to other files or directories.
  Device Files:
- Block Devices: Represent devices that work in fixed-size blocks (e.g., hard drives).
- Character Devices: Represent devices that work with streams of characters (e.g., keyboards).
- Named Pipes (FIFOs): Special files used for inter-process communication.
- Socket*: Used for network communication between processes.
  Special Files: Various system files in `/dev` and others.

14) **What is the difference between wc < file and wc file?**

- Difference between `wc < file` and `wc file`:
- `wc < file`: This takes the contents of "file" as input from standard input. It's like manually redirecting the file's contents to the `wc` command. The file itself is not specified as an argument to `wc`.
- `wc file`: This command takes "file" as an argument and directly operates on the file, providing statistics (word count, line count, etc.) for the specified file. It reads and analyzes the content of "file."

15) **What is the significance of mkdir command in Unix?**

- mkdir Command Significance:
  The `mkdir` command in Unix is used to create directories (folders). It is significant for organizing files and managing the directory structure. By creating directories with meaningful names, users can efficiently store and categorize their files and data.

16) **What is command substitution?**

- Command Substitution:
  Command substitution in Unix allows you to replace a command enclosed in backticks (\`) or `$()` with the command's output. This is useful when you want to capture the output of a command and use it as an argument or input for another command or as part of a variable assignment. For example:
- **Using backticks:**

bashresult=`date`

- **Using `$()`:**

  bash

  result=$(date)

  In both cases, the `date` command's output is captured and stored in the `result` variable**.**

LONG QUESTIONS

1. Write a note on the history of UNIX operating system.

UNIX

➢ Unix (officially trademarked as UNIX, sometimes also written as Unix with small caps) is a computer operating system originally developed in 1969 by a group of AT&T employees at Bell Labs, including Ken Thompson, Dennis Ritchie, Brian Kernighan, Douglas McIlroy, and Joe Ossanna.

History of UNIX :

In late 1960's at AT&T Bell Laboratories the developers were working on the project known as MULTICS (Multiplexed Information and Computing Service).MULTICS was an operating system designed on GE-645 mainframe computers. The purpose of developing the operating was to have an operating system that can be portable and can support multiuser. It was developed with multiuser capability still it was considered to be inadequate, the project was dropped but the researchers working the system, Ken Thompson, Dennis Ritchie, M. D. McIlroy, and J. F. Ossanna, decided to redo the work on a much smaller scale. Ken Thompson still had access to the Multics environment; he wrote simulations for the new file and paging system on it. A team of Bell Labs researchers led by Thompson and Ritchie, including Rudd Canaday, developed a hierarchical file system, the notions of computer processes and device files, a command-line interpreter, and some small utility programs.

In the 1970s Brian Kernighan coined the project name Unics (UniplexedInformation and Computing Service) as a play on Multics. Unics could eventually support multiple simultaneous users, and it was renamed UNIX. UNIX code was written in the assembly language of PDP-7 and was, consequently machine dependent. Ritchie and Thompson worked quietly on UNIX for several years. Ken Thompson then developed a new programming "B" which was subsequently modified by Dennis Ritchie and renamed the "C" language. The whole UNIX system was rewritten in "C" language by 1973 which made is portable language.

The AT&T Company distributed the UNIX to academic and research institutions at a nominal fee. The University of California, Berkeley (UCB), created a UNIX of its own. It was called BDS UNIX (Berkeley Software Distribution). These versions became quite popular worldwide, especially in university and engineering circles.

Berkeleyrewrites the whole operating system in the way they wanted. They created the Standard editor of the UNIX system (vi) and a popular shell(C shell). They created a better file system and they also offered with their standard distribution networkingprotocolsoftware (TCP/IP) that made the Internet possible.

Some other Organizations also developed the other versions of UNIX as given in table.

2. Explain the features of UNIX operating system.

- **Multiuser Capability:**
  UNIX is a multiuser system. In a multiuser system the same computers resources-hard disk, memory etc. are accessible to many users. All the users are given different terminal. All the terminals are connected to the main computer whose resources are availed by all users. So, a user at any of the terminals can use not only the computer, but also any peripherals that may be attached. e.g.: Printer. The following figure depicts the multiuser system. The number of terminals connected to the host machine depends on the number of ports that are present in its controller card. There are several types of terminals that can be connected to the host machine.
- **Dumb Terminals**: These terminals consist of the keyboard and display unit with no memory or disk of its own. These can never acts independent machines If they are to be used they have to be connected to the host machine.
- **Terminal Emulation/Smart Terminal**:PC has its own microprocessor, memory & storage device by attaching this PC to the host machine through a cable & running s/w from this machine, we can emulate to work as it is dumb terminal. However memory & disk are not in use & machine can't carry out any processes on its own.
- **Dial-In terminals:**
  These terminals use telephone lines to connect with host machine. To communicate over telephone line, it is necessary to attach a unit called modem to the terminal as well as to the host machine.
- **Multitasking Capability:**
  UNIX is capable of carrying out multiple jobs at the same time,i.e. a single user can type a program in its editor as well as simultaneously executes

some other command you might have given earlier. The latter job is performed in the background while in the foreground he uses the editor. This is managed by dividing the CPU time intelligently between all processes being carried out. Depending upon the priority the operating system allots small time slots to all the process running in foreground and background. MS-DOS also provides multitasking facility which is known as serial Multiprocessing. In DOS time slicing is done i.e. at a time only one job will run rest of the jobs are stopped temporarily. Where as inUNIXtime slicing is not given, in UNIXpriorities are given and the processes that have same priority are scheduled on a round-robin base.

- **Inter Process Communication:**
  UNIX provides excellent feature that allows users to communicate with fellow users. The communication can be within the network of a single main computer or between two or more networks. The users can easily exchange mail, dataand programs through such networks.
- **Security**:
  UNIX provides outstanding facility for security. It provides security in 3 levels.
  - The first level security is provided by providing passwords (also called Login Level Security). It can be considered as a System Level Security. Username and passwords are assigned to all the users to assure that no other user can access his work.
  - The second level security is at the file level. There are read, write and execute permissions to each file that decide who can access a particular file, who can modify it and who can execute it.
  - The third level security is given by file encryption. This utility encodes user's file into an unreadable format, so that even if the user succeeds in opening the file he will not be able to read it. When user wants to see the contents of the file he can decrypt the file.
- **System Portability:**
  UNIX is written in High level language "C", hence it can easily run on any machine with or without small changes. The code can be changed and compiled on a new machine.
- **Open System:**
  UNIX has an open architecture one can add to the toolkit by simply writing a program & Starting that executable in a separate area in the file. A separate device can also be added by creating file for it. Modification of system is easy because the source code is always available.
- **Programming facility:**

UNIX shell is also a programming language; it is designed for a programmer, not a casual end user. It has all the necessary ingredients, like control structures loops and variables that establish it as a powerful programming language in its own right. These features are used to design shell scripts.

➢ **Documentation**:
UNIX provides an excellent feature of online help. The principle online help facility available is the man command, which is the most important reference for commands and their configuration. It gives detailed information about all the commands.

➢ **Pattern Matching:**
UNIX features very sophisticated pattern matching feature. There are some special characters (like *,?,[..]) through UNIX provides very nice feature oFpattern matching. There is also facility of regular expression that is framed with characters from this set.

3. Explain different level of security provided by UNIX operating system.

➢ **Unix provides three types of securities.**

i. System level
ii. Directory level
iii. File level

- **System level:**
In Unix, every user has been allocated login id and password. When the system administrator opens an account for user, an entry is created in system password file, called /etc/passwd. So to access the resources of the Unix system, user has to log in first. Only the authorized user can access the system.
- **Directory level:**
In Unix, every thing is treated as file. Even directories or devices are also considered as file. There are read, write, and execute permissions to each file, which decide who can access a particular file. Who can modify it and who can execute it.
- **File level:**
Lastly, there is file encryption. Encryption utility encodes your file into an unreadable format, so that even if someone succeeds in opening it, your secret information are safe.

4. Differentiate between multiuser and multitasking operating system

➢ The main difference between a multiuser and multitasking operating system is that multiuser operating system allows multiple users to access

the computer system at the same time, while a multitasking operating system allows multiple programs to run at the same time.
A multiuser operating system allows multiple users to share the same computer system. Each user has their own account and can run their own programs.
Multiuser Oss are typically used in servers, where they allow multiple users to access shared resources such as files, printers, and databases.
A multitasking operating system allows multiple programs to run at the same time on the same computer. The OS divides the cpu time between the programs, giving each programs a small amount of time to run. This allows the programs to appear to be running simultaneously, even though they are actually sharing the CPU.
Here are some examples of multiuser operating system:

1. Unix
2. Linux
3. macOs

Here are some examples of multitasking operating system:

4. windows
5. os x
6. android
7. ios

5. What is shell? Explain types of shell in detail.

➢ **Shell**:

Shell is the most widely used utility program on all Unix systems. It is loaded in memory when a user logs in. The shell establishes an interaction between the user and a kernel. The shell is also known as the command interpreter because it interprets the commands and passes it to the kernel.
Types of shells:
The shell runs like any other program under the Unix system. Hence, one shell program can replace the other, or call another shell program as it would call any other program. Due to this feature, several shells have been developed in response to difference needs of users. Some of the popular shells are listed below:

- **Bourne shell:**
  This is one the most widely used shells in the Unix world. It was developed by Steve Bourne at AT & T Bell Laboratories in the late 1970's. It is the primary Unix command interpreter and comes along with every Unix system. It provides 'dollar prompt$' on Unix

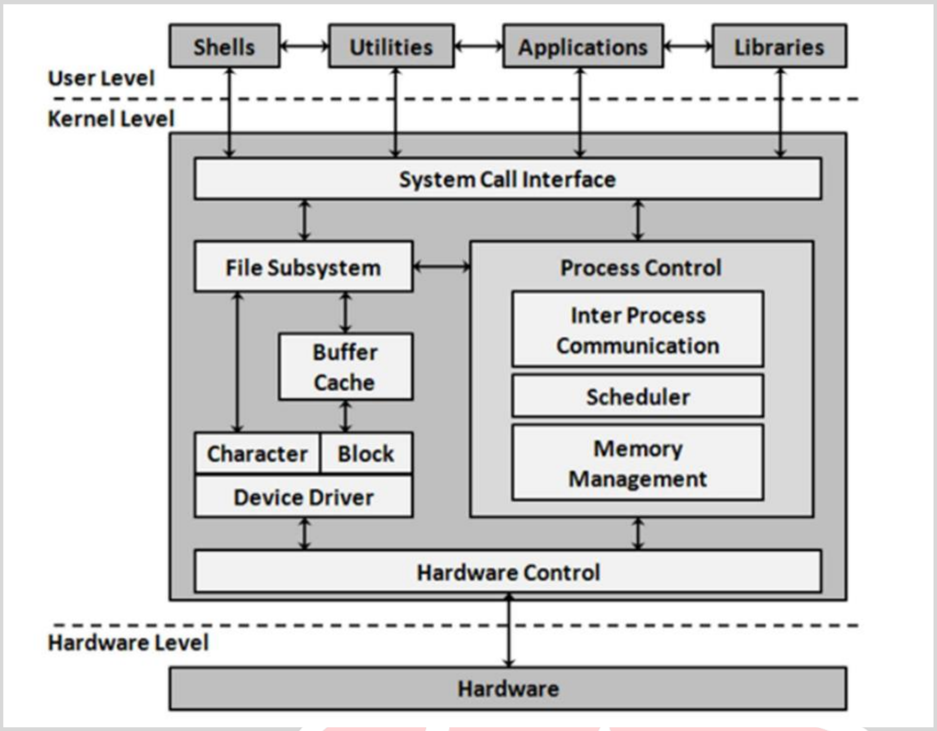
installations as the trademark of the Bourne shell. The Bourne shell is an executable file named sh.

- **C Shell:**
  Bill Joy developed C shell at the university of California. Its commands were supposed to like C statement. It has two advantages over the bourne shell. Firstly,
  it allows aliasing of commands. This is useful when lengthy commands can be renamed by user i.e. user gives short-name of frequently used lengthy command.
  So instead of typing the entire command, user can simply use the short alias at the command line and save typing time also.
- **Korn Shell:**
  It is very powerful shell. It is superset of Bourne shell. It can completely replace the Bourne shell in a system and has several more functions that are build into the shell making it more efficient. The Korn shell includes all the enhancements in the
  C shell, like command history and aliasing, and offers a few more feature itself.
- **Bash Shell:**
  It is an enhanced version of the Bourne shell. Bash stands for bourne again shell. It
  is used in Unix environment.
- **Tcsh Shell:**
  Tcsh is pronounced as tee-cee shell. It is a compatible version of the C shell. It is used in Linux environment.
- **PDKSH Shell:**
  Pdksh stand for public domain korn shell. Linux offers pdksh as a substitute for ksh shell.

6. Explain the architecture of UNIX operating system.

➢ The UNIX architecture can be divided into three levels: User level, Kernel level, Hardware level.

- The system call and library Interface represent the border between user programs and the kernel as shown in the figure. The system calls are partitioned to the system calls that interact with the file sub system and the system calls that interact with the process control subsystem.



- The file subsystem manages the files, allocates files space, administrating the free space, controlling access to files, and retrieving data for users. Processes interact with the file system through system calls. E.g., Open, close, read, write, etc.
- Device drivers are the kernel modules that control the operation of peripheral devices. Block I/O devices are random access storage device to the rest of the system.
- The process control subsystem is responsible for process synchronization, interprocess communication, memory management, process scheduling. The system calls used with process control systems are fork (creating a new process), exec (overlay the image of a program onto the running process), Exit (finish executing a process).
- The memory management module controls the allocation of memory. If at any time the system does not have enough physical memory for all processes, the kernel moves between main memory and secondary memory so that the all processes get a fair chance to execute.
- The scheduler module allocates the CPU to processes. It schedules them to run in turn until they voluntarily relinquish the CPU while awaiting a resource or until the kernel pre-empts them when their recent run time exceeds a time quantum.

- The hardware control is responsible for handling interrupts and for communicating with the machine. Devices such as disks or terminals may interrupt the CPU while a process is executing. The kernel executes the interrupt and then resumes the previously executing process. This way it provides access of hardware devices.

7. Explain the Kernel architecture in detail.

➢ **Kernel**:

- The kernel is the core of the operating system. Kernel is mostly written in C. It is loaded into memory when the system is booted and communicates directly with the hardware. User programs that need to access the hardware use the services of the kernel, which performs the job on the user's behalf. The programs access kernel through set of functions called system calls. The kernel also manages the system memory, Schedules processes, decides their priorities, and performs another task .
- The system call and library Interface represent the border between user programs and the kernel as shown in the figure. The system calls are partitioned to the system calls that interact with the file sub system and the system calls that interact with the process control subsystem
- The file subsystem manages the files, allocates files space, administrating the free space, controlling access to files, and retrieving data for users. Processes interact with the file system through system calls. E.g., Open, close, read, write, etc.
- Device drivers are the kernel modules that control the operation of peripheral devices. Block I/O devices are random access storage device to the rest of the system. The process control subsystem is responsible for process synchronization, interprocess communication, memory management, process scheduling. The system calls used with process control systems are fork (creating a new process), exec
- (overlay the image of a program onto the running process), Exit (finish executing a process).
- The memory management module controls the allocation of memory. If at any time the system does not have enough physical memory for all processes, the kernel moves between main memory and secondary memory so that the all processes get a fair chance to execute.
- The scheduler module allocates the CPU to processes. It schedules them to run in turn until they voluntarily relinquish the CPU while awaiting a resource or until the kernel pre-empts them when their recent run time exceeds a time quantum.The hardware control is responsible for handling interrupts and for communicating

- with the machine. Devices such as disks or terminals may interrupt the CPU while a process is executing. The kernel executes the interrupt and then resumes the previously executing process. This way it provides access of hardware devices.

8. Define shell. Explain feature of shell in detail.

- ➢ The shell is the **command interpreter**. It acts as an **interface between the user and kernel**. When the **user types any command** it is interpreted by **shell** and then given to the **kernel.** There could be several copies of shell running on the system. For each user who logs in the system a different shell is created.
- Interactive Environment:
  The shell allows the user to create a dialog (communication channel) between the
  user and the host UNIX system. This dialog terminates until the user ends the system session.
- Shell scripts:
  It is the shell that has the facility to be programmed; the shell contains commands that can be utilized by the user. Shell scripts are group of UNIX command string together and executed as individual files. The shell is itself a program; accept that is written in "C" language
- Input/Output Redirection:
  Input/Output Redirection is a function of shell that redirects the o/p from program to a destination other than screen. This way you can save the o/p from a command into a file and redirect it to a printer, another terminal on the h/w or even another program. similarly, a shell can be a program that accepts i/p form other than keyboard by redirecting its i/p from another source.
- Programming Language:
  The shell includes features that allow it to be used as programming language. This feature can be used to build shell script that performs complex operations.
- Shell variables:
  The user can control the behaviour of the shell as well as other programs & utilities by storing data in variables.

9. Write a note on UNIX file system.

- ➢ Unix file system is a logical method of organizing and storing large amounts of information in a way that makes it easy to manage. A file is a smallest unit in which the information is stored. Unix file system has several important features.

- All data in Unix is organized into files. All files are organized into directories. These directories are organized into a tree-like structure called the file system. Files in Unix System are organized into multi-level hierarchy structure known as a directory tree. At the very top of the file system is a directory called "root" which is represented by a "/". All other files are "descendants" of root.
- The Unix file system is a hierarchical file system used by Unix-based operating systems to store and organize files and directories.
- It is a tree-like structure that starts with a single directory called the root directory, which is denoted by a forward slash (/) character.
- Unix file system also uses a set of permissions to control access to files and directories. Each file and directory have an owner and a group associated with it, and permissions can be set to allow or restrict access to these entities.
  - / : The slash / character alone denotes the root of the filesystem tree.
  - /bin : Stands for "binaries" and contains certain fundamental utilities, such as ls or cp, which are generally needed by all users.
  - /boot : Contains all the files that are required for successful booting process.
  - /dev : Stands for "devices". Contains file representations of peripheral devices and pseudo-devices.
  - /etc : Contains system-wide configuration files and system databases. Originally also contained "dangerous maintenance utilities" such as
  - init,but these have typically been moved to /sbin or elsewhere.
  - /home : Contains the home directories for the users.
  - /lib : Contains system libraries, and some critical files such as kernel
  - modules or device drivers.

10. Write a note on types of file in UNIX.

➢ The UNIX files system contains several different types of files

- **Ordinary files** – An ordinary file is a file on the system that contains data, text, or program instructions. It is used to store your information, such as some text you have written or an image you have drawn. This is the type of file that you usually work with.
- **Directories** – Directories store both special and ordinary files. For users familiar with Windows or Mac OS, UNIX directories are equivalent to folders. A directory file contains an entry for every file

and subdirectory that it houses. If you have 10 files in a directory, there will be 10 entries in the directory.

- **Special Files** – Used to represent a real physical device such as a printer, tape drive or terminal, used for Input/Output (I/O) operations. Device or special files are used for device Input/Output(I/O) on UNIX and Linux systems. They appear in a file system just like an ordinary file or a directory. On UNIX systems there are two flavours of special files for each device, character special files and block special files.
- **Pipes** – UNIX allows you to link commands together using a pipe. The pipeacts a temporary file which only exists to hold data from one command until it isread by another. A Unix pipe provides a one-way flow of data. The output or result of the first command sequence is used as the input to the second commandsequence. To make a pipe, put a vertical bar (|) on the command line between twocommands.
- **Sockets** – A Unix socket (or Inter-process communication socket) is a specialfile which allows for advanced inter-process communication. A Unix Socket isused in a client-server application framework. In essence, it is a stream of data,very similar to network stream (and network sockets), but all the transactions arelocal to the filesystem.
- **Symbolic Link** – Symbolic link is used for referencing some other file of thefile system. Symbolic link is also known as Soft link. It contains a text form of the path to the file it references. To an end user, symbolic link will appear to haveits own name, but when you try reading or writing data to this file, it will insteadreference these operations to the file it points to. If we delete the soft link itself,the data file would still be there. If we delete the source file or move it to adifferent location, symbolic file will not function properly.

11. Explain the characteristic of UNIX file system.

➢ The characteristic of Unix file system is as below:

1. Hierarchical Organization:

The top-level directory of the hierarchy is traditionally called root (written as a slash /). The tree "grows" downward from the root, with paths connecting the root to each of the other files. At the end of each path is an ordinary file or a directory file. Ordinary files, frequently just called files, are at the ends of paths that cannot support other paths. When you refer to the tree, up is toward the root, and down is away from the root. Directories directly connected by a path are called parents (closer to the root) and children (farther from the root).

2. Case-sensitive:

The Unix file system is case-sensitive, which means that the capitalization of file names matters. For example, the files file.txt and FILE.TXT would be considered to be the same file. The case-sensitivity of the unix file system can be a bit confusing at first, but it can also be helpful. For example, it can be used to prevent accidentally overwrite a file with same name but different capitalization.

3. Permissions:

The Unix file system uses permissions to control who can access a file or directory and what they can do with it. There are three types of permissions:

- Read Permission allows the user to read the contents of the file or directory.
- Write Permission allows the user to modify the contents of the file or directory.
- Execute Permission allows the user to run the file as a program.

Each file and directory has three sets of permissions, one for the owner of the file or directory, one the group that the file or directory belongs to, and one for everyone else. The permissions are represented by three characters, each of which
can be one of the following:

- r for read permission
- w for write permission
- x for execute permission

4. Device files:

Device files are a special type of file in the unix file system that represent physical devices, such as hard drives, printers, and network interfaces. Device files are used to access and control these devices.
There are two general kinds of device file in Unix:

- Character special files: Character special files are used to represent devices that can be read or written one character at a time, such as a keyboard or a serial port.
- Block special files: block special files are used to represent devices that can be read or written in blocks of data, such as a hard drive or a USB drive.

12. Explain the UNIX file system component in detail.

➢ The Unix file system is a hierarchical file system used by Unix-based operating systems to store and organize files and directories. It is a tree-like structure, with the root directory at the top and all other directories and files breaching off from it.

The main components of the Unix file system are:

1) **Files**: Files are the basic unit of storage in the Unix system. They can contain any types of data, such as text, images, programs, and so on.

2) **Directories**: Directories are used to organize files. They can contain other directories, as well as files.

3) **Permission**: permission control who can access a file or directory and what they can do with it. There are three types of permission: read, write and execute.

4) **Hard links**: Hard links are pointers to a file. They allow you to create multiple names for the same file.

5) **Symbolic links**: symbolic links are pointers to another file or directory. They are often used to create shortcuts or aliases.

6) **Special files**: Special files represent devices, such as hard disks, printers, and terminals.

13. Write a note on I-NODE

➢ INODE (short for index node) is a data structure in a Unix-style file system that descries a file-system object such as file or directory. Each inode stores the attributes and disk block locations of the object's data. File-system object attributes may include metadata, as well as owner and permission data. A directory is a list of inodes with thiery assigned names. The list includes an entry for itself, its parent, and each of its children.

The inode is a critical data structure in a unix-sytle file system. It is used to keep track of all the files and directories in the file system, and it provides the information that is needed to access and manipulate these files and directories.

The following are some of the information that is stored in an inode:

- file type (regular file, directory, etc.)
- file size
- file permissions
- owner and group IDs
- last access time
- last modification time
- last change time
- direct block pointer
- single indirect pointer
- double indirect pointer
- triple indirect pointer

The inode number is a unique identifier for each inode. It is used to locate the inode in the file system. The inode number is also used to reference the inode in file system operations.
The number of inodes in a file system is limited. This means that the number of files and directories that can be created in the file system is also limited. Thenumber of inodees in file system is typically specified when the file system iscreated.
Inodes are an important part of the Unix-style file system. They provide the information that is needed to access and manipulate files and directories. The Inode is critical data structure that makes the unix-style file system efficient andreliable.

14. Explain UNIX directory structure in detail.

➢ The Unix directory structure is a hierarchical file system structure, much like an upside-down tree, with the root directory at the base of the file system and all other directories spreading from there.

➢ The root directory is the top-level directory in the file system. It contains all the other directories and files on the system. Some of the most common directories under the root directory include:

- /bin: this directory contains executable programs that are essential for the system to function, such as the ls and cp commands.
- /etc: This directory contains configuration files for the system, such as the passwd file, which stores user account information.
- /home: This directory contains the home directories for all users on the system.
- /lib: This directory contains shared libraries that are used by executable programs.

- /mnt: This directory is used to mount filesystems from other devices, such as USB drives or network shares.
- /opt: This directory is used to install optional software packages.
- /root: This directory is the home directory for the root user, who has full administrative privileges on the system.
  In addition to these common directories, there are many other directories that may be present on a UNIX system, depending on the specific distribution and configuration.
  The UNIX directory structure is a flexible and powerful way to organize files and directories. It allows users to easily find the files they need and to manage their files effectively.
  Here are some of the advantages of the UNIX directory structure:
- It is hierarchical, which makes it easy to organize files and directories into a logical structure.
- It is scalable, so it can be used to store a large number of files and directories.
- It is portable, so it can be used on different types of computers.
- It is secure, with permissions that can be set to control access to files and directories.
- Here are some of the disadvantages of the UNIX directory structure:
- It can be difficult to navigate, especially for large file systems.
- It can be difficult to manage permissions, especially for large file systems.
- It is not as user-friendly as some other file system structures, such as the Windows file system.

15. Explain the structure of /etc/passwd file.

- The /etc/passwd file is a plain text file that contains information about all user accounts on a Linux or Unix system. Each line in the file represents one user account and contains seven fields, separated by colons (:). The fields are:
- **Username**: The username of the account. This is the name that the user willuse to log in to the system.
- **Password**: The password of the account. This field is typically encrypted, but in some older systems, it may be stored in plaintext.
- **User ID (UID):** A unique identifier for the user account. Each user account must have a unique UID.
- **Group ID (GID):** The primary group ID of the user account. This is the group that the user will be a member of by default.
- **Gecos**: A comment field that can be used to store additional information

about the user account, such as the user's full name or office phone number.

- **Home directory**: The absolute path to the user's home directory. This is the directory that the user will be placed in when they log in to the system.
- **Login shell**: The absolute path to the user's login shell. This is the program that the user will run when they log in to the system.
  The /etc/passwd file is owned by the root user and has permissions of 644. This means that it can only be read and written by the root user and by users with sudo privileges.
  The /etc/security/passwd file is a newer file that is used to store the encrypted passwords for all user accounts. The /etc/passwd file only contains a reference to the /etc/security/passwd file for each user account. This allows the encrypted passwords to be stored in a more secure location.
  The /etc/passwd and /etc/security/passwd files are important system files that should be protected from unauthorized access. Only authorized users should be able to read or modify these files.

16. Explain the structure of /etc/shadow file

- The /etc/shadow file is a text file that contains information about the passwords of all users on a Unix system. It is owned by the root user and the shadow group, and has permissions of 0640. The file is divided into nine fields, separated by colons (:):
  - **Username**: The username of the user account.
  - **Encrypted password**: The encrypted password of the user account. The encryption algorithm used is specified by the first character of the field. The following algorithms are supported:
    1: MD5
    2a: Blowfish
    2y: Eksblowfish
    5: SHA-256
    6: SHA-512
  - **Last password change**: The date of the last password change, expressed as the number of days since January 1, 1970 (Unix time).
  - **Minimum password age**: The minimum number of days that must pass
    before a user can change their password.
  - **Maximum password age**: The maximum number of days that a password can be used before it must be changed.
  - **Password inactive days**: The number of days after the password expires before the account is locked.

- **Grace logins**: The number of times a user can log in with an expired password before their account is locked.
- **Account expiry date**: The date on which the account will expire.
- **Reserved**: This field is reserved for future use.

The /etc/shadow file is not directly readable by users, but it can be accessed by programs with the appropriate permissions, such as the passwd command.
The /etc/shadow file is an important security feature that helps to protect the passwords of users on a Linux system. By keeping the passwords encrypted and inaccessible to unauthorized users, the /etc/shadow file helps to prevent unauthorized access to user accounts.

17. Explain booting process and init process in detail.

➢ The booting process is the sequence of events that occurs when a computer is turned on. It starts with the BIOS (Basic Input/Output System), which is a small program that is stored in a chip on the motherboard. The BIOS performs a few basic checks of the hardware, and then loads the bootloader.

- The bootloader is a program that loads the operating system into memory. The bootloader is typically stored in a partition on the hard drive, but it can also be stored on a USB drive or other removable media.
- Once the bootloader has loaded the operating system, the init process is started.
- The init process is responsible for initializing the system and starting up the basic services that the operating system needs to run.
- The init process is typically implemented as a daemon, which is a program that runs in the background and does not require user interaction. The init process is responsible for starting up other daemons, such as the network daemon, the filesystem daemon, and the graphical user interface (GUI) server. The init process also controls the run levels of the system. he run level is a configuration of the system that determines which daemons are running. There are typically several run levels, such as single-user mode, multi-user mode, and graphical mode.

The following are the steps involved in the booting process of a Unix system:

- The BIOS performs a power-on self-test (POST).
- The BIOS loads the bootloader.
- The bootloader loads the kernel.
- The kernel initializes the hardware.
- The kernel loads the initramfs.

- The initramfs mounts the root filesystem.
- The init process is started.
- The init process starts up the basic services.
- The init process enters the default run level.
- The following are the steps involved in the init process of a Unix system:
- The init process reads the /etc/inittab file.
- The init process determines the current run level.
- The init process starts up the daemons that are associated with the current run level.
- The init process monitors the system and restarts any daemons that fail.
- The init process waits for a shutdown signal.

18. Explain File Access permission in UNIX system.

➢ File access permissions in Unix systems control who can read, write, and execute
a file. There are three types of permissions:

- **Read** (r): The ability to read the contents of the file.
- **Write** (w): The ability to modify the contents of the file.
- **Execute** (x): The ability to execute the file as a program.

Each file has three sets of permissions, one for the owner of the file, one for the group that owns the file, and one for everyone else. The permissions are represented by three characters, one for each type of permission. The characters can be either a hyphen (-), which means that the permission is denied, or a letter, which means that the permission is granted. The letters are r, w, and x for read, write, and execute, respectively.
Here are some additional things to keep in mind about file access permissions in Unix systems:

- The owner of a file is the user who created the file.
- The group that owns a file is the group that the owner belongs to.
- The permissions for everyone else are the permissions that are granted to users who do not belong to the group that owns the file.
- The permissions for a file can be inherited by directories and
- subdirectories.
- The permissions for a file can be changed using the chmod command.

19. Explain ls command in detail with all option.

➢ The ls command is a Unix command that lists the contents of a directory. It is one of the most basic and commonly used commands in Unix.
The basic syntax of the ls command is:
ls [options] [files or directories] The options control the output of the ls command. Some of the most common
options are:
- -a : Lists all files, including hidden files. Hidden files are files that start with a dot (.).
- -l : Lists all files in long format. This format shows more information about each file, such as the file permissions, size, and modification time.
- -r : Lists the files in reverse order.
- -t : Lists the files by modification time, with the most recently modified files first.
- -F : Appends a character to each file name to indicate its type. For example,
  directories are marked with a slash (/), executable files are marked with an asterisk (*), and symbolic links are marked with an arrow (->).
  You can also use the ls command to list the contents of directories recursively.
  This means that the ls command will list the contents of all subdirectories, as well as the contents of the current directory. To do this, use the -R option.

Here are some other useful ls options:
- -d : Lists directories only.
- -i : Lists files by inode number.
- -n : Lists files by numerical user ID (UID) and group ID (GID).
- -h : Displays file sizes in human-readable format.
- -color : Colorizes the output of the ls command.

20. Explain the output of ls command.

➢ The ls command is used to list the contents of a directory. The output of the ls command is divided into several columns, each of which represents a different piece of information about the file or directory. The columns are:
File type: The first character in the column indicates the type of file. The possible values are:
- - (Hyphen): Regular file
- d: Directory
- l: Symbolic link
- b: Block device
- c: Character device

- s: Socket
- f: FIFO

  Permissions: The next 9 characters in the column represent the permissions for the file. The permissions are divided into three sets of three characters, one for each user type: owner, group, and others. Each character in a set can be one of the following:
- r: Read permission
- w: Write permission
- x: Execute permission
- -(Hyphen): No permission
- Number of hard links: The next column shows the number of hard links to
- the file. A hard link is a way of creating multiple names for the same file.
- Owner: The next column shows the name of the owner of the file.
- Group: The next column shows the name of the group that owns the file.
- Size: The next column shows the size of the file in bytes.
- Date and time: The last column shows the date and time when the file was last modified.

21. Explain cat command in detail with its option.

➢ Cat stands for concatenate. It displays the content of one or more files. It is similar to type command of MS-DOS

Syntax:

Cat [option] [file1][file2]…

The above syntax indicates that you can use cat command without any option and filename(s).

there are several purpose of cat command.

- To display the content of file flie1 present in working directory then we issue command at dollar Prompt as follow:
  ```
  $cat file1 <enter>
  This is file1
  ```
- The full path-name can be specified to display a file stored in another directory.
  ```
  $cat /usr/user/file2<enter>
  This is flie2
  ```
- It can also display contents of one or more file.
  ```
  $cat file1 file2<enter>
  This is file1
  This is flie2
  ```
- If you use cat command without any arguments then it will take input from standard input devices(i.e. keyboard) and display them on standard output

devices until user press <ctrl=d>.This character is used to terminate standard input.

```
$cat<enter>
This is line1<enter>
This is line1
This is line2<enter>
This is line2
<ctrl+d>
```

The output shows that all lines display in duplicate. This is because when we apply a cat command at prompt without any argument it waits for input. If we type a line and then press an <enter> key, the inputted line is again display on standard output.

Note: In Unix <ctrl+d> character is used to mark for end -of –end- file. It is also known as input redirection character.

- User can create a file using cat command. To create a file the command is:

```
$cat >file1
This is line1 of file1<enter>
This is line2 of file1<enter>
<ctrl+d>
```

Here the '>' symbol is used for output redirection.

- If the file file1 is already exist then the above command will erase the content of file.

To keep the existing content of the file and want to append more into the line, then
then we can use '>>' symbol.

```
$cat >> file
This is the last line of file1<enter>
<ctrl+d>
```

- Cat can also accept standard input and output.

```
$cat <f1>f1.out
```

Some of the option used with cat command is as follow.

I. –v(visual) : This option display control character and other non-printing character.

Consider a file f1 the contains tab characters at the end of first line and control character (shift+tab) at the end of second line. If we write a command without option it displays contents as follow:

```
$cat f1
This is a tab characters
This is a control characters
```

If we used –v option then the output display control character e.g. shift+tab as ^[[z

$ cat-vf1
This is a tab characters
This is a control characters^[[z

II. –e or –E: It prints a $ mark the end of fine. In other word, it display $ instead of new –
line character

$cat –e f1
This is a tab character
This is a control character ^[[z

III. –t: It print character as ^I and form character as ^I

$cat –t f1
This is a tab character
This is a control character ^[z

IV. –n: it number each line of file(s) and written to standard output

```
$cat –n file1
    1: This is line 1 of file 1
    2: This is line 2 of file 1
    3: This is line 3 of file 1
```

22. Explain rm command in detail with its option.

- **rm**: rm stands for remove. It deletes one or more files/directories. Its equivalent MS-DOS command is Del or erase.
  Syntax:
  rm[option] file(s)/directory-name(s)
  This command is used to remove a file or a directory. Let us see some example of it.
- To remove a file file1 from the current directory then the command is
  $rm file1<enter>
  Here the rm command is silent i.e. it immediately display prompt without any message. It indicates that the file is removed successfully. Otherwise display a message on the screen.
  You can remove more than one file like this:
  $rm file1 file2
  It removes file1 and file2 from the current directory.
  Some of the option used with rm command as follow:
- -i (interactive): This option removes files interactively. It display prompt for removal of file(s). To remove only selected files from the directory then we can apply a command as

```
$rm –I *
File1 : y
File2 : y
...so on
```

To keep file, press 'n' or hitting <enter> key and to remove it press y/Y.

- -r (recursive): It is used to remove non-empty directory, together with all the files and subdirectories it contains.
  The-roption is used with directory name.
  `$rm -r mydir`
  It removes a mydir directory with all files and subdirectories within it.
  Also, you can removes more than one directories using rm command with option-r.
  `$rm-r dl d2d3`
  It removes three directories d1, d2 and d3 within the current directory.
  There is no meaning to use-r option with regular file like this:
  `$rm-r file1`
- -f (force):To remove a file, you must have write permission to that file. If you do not have write permission on the file, you will be prompted (y or n) to override i.e. it display prompt for removal of write-protected file. This option is used to remove a file forcibly without displaying write-protected message.
- Let`s consider a write-protected file f1 and user issued a command like this:

  ```
  $rm f1
  rm: remove write-protected regular file 'f1'? y
  ```

  Here, it display write-protected message for file fl. If user want to suppress this message during removal of write-protected file then issued a command like this:
  `$rm –f f1`

23. Explain cp command in detail with its option.

- ➢ cp stands for copy. It creates a duplicate file(s) having access and modification date-time is similar to current system date-time. Its equivalent MS-DOS command is copy.

  Syntax:
  `cp[option] filename/directory-name filename/directory-name`
  This command required two arguments to copy a file or a directory. The first and second argument is also known as source and destination file/directory respectively.
- ➢ The following command sequences show that how you can create duplicate file of file1 in the same directory with different file name file2.

```
$ls<enter>
File1
$cp file1 file2<enter>
#file copied successfully
$ls<enter>
File1
file2
```

cp command does not display success message when a file is copied successfully. Above command sequences create a duplicate file file2 having same contents and permission as file1 have but the modification and access date-time of file2 is changed to current system date-time.

Files can be copied to and from another sub-directory also.

 Files can be copied to and from another sub-directory also

```
$cp/usr/bca1/file1 /usr/bca2/file2<entert>
```

- If we have copied a file from one directory to another directory without the name of destination file- name then it creates a file in another directory with same name as source file.

  ```
  $cp/usr/bcal/filel/usr/bca2/<enter>
  ```

  Here, it creates a duplicate file file1 in/usr/bca2 directory.

   It also copy one or more files with the same name to another directory.

  ```
  $cp file1 file2 file3 my_dir
  ```

  Here file1, file2 and file3 are copied with the same name into the directory my_dir
- Some of the options used with cp command are as follow:
- -r(recursive):The –r option recursively copy an entire directory structure to another directory.

  e.g. $cp –r olddir newdir

  It recursively copies files/subdirectories to a new directory newdir. If newdir is exist in the current directory then it creates new directory olddir in the directory newdir.
- -i (interactive):Sometimes, a destination file is already exist and user want to show an overwriting prompt for an existing file during copy then-i option is used.

  For example, consider that we have two files f1 and 2 under the current directory and apply a command as follow:

  ```
  $cp f1 f2
  ```

  Then, it copies a file f1 to f2 without any warning. But if we apply a command with-

  i option, it will warn you before overwrite a file like this:

e.g. $cp-i f1 f2
cp: overwrite f2? Y
Answer 'y' overwrites file f2 whereas any other responses ignore the operation -p (preserve):Normally, copy operation changes access and modification datetime of destination file to current system date-time. This option preserves these (access and modification date-time) attributes of file(s).

```
$ls-1f1:ls-luf1
-rw-rw-r-- 1 bharattmtbca 133 Jul 23 12:01 f1#modification date-time
-rw-rw-r-- 1 bharattmtbca 133 Jul 24 14:45 f1 #access date-time
$cp-pf1f2; ls-1f2;ls-luf2
-rw-rw-r-- 1 bharattmtbca 133 Jul 23 12:01 f1 #modification date-time
-rw-rw-r-- 1 bharattmtbca 133 Jul 24 14:45 f1 #access date-time
```

The example shows that the access and modification date-time of file f1 is inherited to file f2.

- -l (link):This option creates a link instead of copying a file. In other words, it creates an alias name of the file. You can create link of file f1 like this:
  $cp-lf1f1.ln $

24. Explain mv command in detail with its option.

- **mv** stands for move. It is used to move an individual file, a list of files, or a directory from one directory to another. It is also used to rename a file/directory.
  Its equivalent MS-DOS command is move. The general syntax is:
  Syntax:
  mv [option] filename(s)/directory-name filename/directory-name
  - In Unix a file can be moved one directory to another directory
    $mv/usr/user1/file1/use/user/file2<enter>
  - A user can move several files at once to the same destination directory by first write name of all the files to be moved and giving the name of the destination last.
    $mv file1 file2 file3 dir_new
  - To rename a file in the current directory, you can also use mv, but with the new filename as the destination.
    $mv old file new file
    It removes old file to new_file.
  - -I (interactive): it warns you overwrite an existing file.it display the filename followed by a question mark, Answer y/Y overwrites the file otherwise ignore operation of that file
  - To overwrite file1 to file2 the command is :

```
$mv –I file1 file2
Mv: overwrite file2? Y
```

- -f (Force):it suppress interactive overwrite message

25. Explain any two file comparison command

- **cmp** :This command compares two files byte by byte. If the two files are identical, the cmp command will not output anything. If the two files are different, the cmp command will output the byte number where the first difference is found.
  For example, to compare the files file1.txt and file2.txt, you would use the following command:
  cmp file1.txt file2.txt
  diff :This command compares two files line by line. It shows the lines that are different in the two files, as well as the lines that are only in one file or the other.
  For example, to compare the files file1.txt and file2.txt, you would use the following command:
  diff file1.txt file2.txt
  The diff command has many options that can be used to control how the comparison is performed. For example, the -b option can be used to compare the files byte by byte, like the **cmp** command.

26. What are the difference between copying a file and moving a file? Explain with its example.

- In Unix-like operating systems, copying and moving files are two distinct operations with different effects:
- **Copying**:
  Copying makes a duplicate of a file or directory into a new location. In Linux, the "cp" command allows the user to copy a file or a directory.
  The following command creates a copy of file1 and names it as file2. If the file2 already exists, it will be overridden with the new content.
  cp file1.txt file2.txt.
- **Moving**: Moving transfers the original files or directories from one location to another.
  It deletes the content from the first location and creates content in a new location. In Linux, the mv command is used to move a file or a directory to a new location.
  Example: $mv file1 file2 file3 dir_new
  1. Copying a File:
  - When you copy a file, you create a duplicate of the original file at the destination location while keeping the original file intact.

- The `cp` command is commonly used for copying files in Unix. For example:

```
cp sourcefile destination
```

2. Moving (Renaming) a File:

- When you move (also known as renaming) a file, you change its location or name within the file system. The file is no longer present in its original location under its original name.
- The `mv` command is used to move or rename files in Unix. For example:

```
mv oldname newname
or
mv sourcefile destination
```

- In summary, copying creates a duplicate of a file, while moving changes the file's location or name within the file system. The original file may or may not exist after a move, depending on whether it was moved to a different directory or simply renamed within the same directory.

27. Explain head and tail command.

- **Head**: The head command is used to view the first few lines of a file. By default, it will display the first 10 lines of a file, but this number can be changed with the '-n' option. The syntax for the head command is as follows –

  ```
  $ head [options] [file(s)]
  ```

  - head command options:
    The head command has several options that can be used to customize its output. Some of the most used options are –
  - -n –the -n option is used to specify the number of lines to display.
    For example, to view the first 20 lines of a file named "example.txt", the command would be:

    ```
    $ head -n 20 example.txt
    ```
  - -q – the -q option is used to suppress header printing when multiple files are used.
  - -v – the -v option is used to always print headers when using multiple files.
- **Tail**: The tail command is used to display the last few lines of a file. Like the head command, tail will display the last 10 lines of a file by default, but this number can be changed with the -n option. The syntax of the tail command is as follows –

  ```
  $ tail [options] [file(s)]
  ```

  - tail command options

The tail command also has several options that can be used to customize its output. Some of the most used options are −

- -n − The -n option is used to specify the number of lines to display. For example, to view the last 20 lines of a file named "example.txt", the command would be:
  ```
  $ tail -n 20 example.txt
  ```
- -f − The '-f' option is used to keep the file open and continue displaying output as the file grows. This option is useful when working with log files.
  ```
  $ tail -f example.log
  ```
- -F − The '-F' option is similar to the '-f' option, but it also controls file truncation.

28. Write a sort command detail with its all option.

➢ **Sort**: sort command can be used for sorting the contents of a file. Besides sorting, it can merge multiple sorted files and store the result in specified output file.

Syntax:

```
sort [option) [file(s) ]
```

Sorting proceeds character-by-character and begins with first character of each line. If first characters of two lines are same then sorting carry on to next character and so forth (Consider the following command.

```
$sort f1 #display sorted content of file f1
```

It sort contents of input file f1 and display it on screen

- Consider the following command
  ```
  $sort f1 f2 f3
  ```
- It concatenates contents of input files f1,f2 and f3 and writes collectively sorted output to standard output.
- To merge the contents of a file with the standard input the command like this:
  ```
  $sort-f1 #'-'indicates standard input file
  ```
  Various option used with sort command are listed below
- -o filename: It writes sorted output in a specified to file filename. sorts three files f1, f2 and f3 and writes sorted content of these files into an output file called mfile then the command is as follow:
  ```
  $sort –o mfile f1 f2 f3 <enter>
  ```
- A user can also redirect sorted output to another file using output redirection.
  ```
  $sort f1 f2 f3 > mfile<enter>
  ```

- –u: It removes duplicate lines from the sorted output i.e. the output contains only unique lines.
  $sort –uf1<enter>
- –m: It merges list of files that have already been sorted.
  $sort-m fl f 2 #files fl andf2 must be sorted
- –t sep: It sorts the file on any field delimited (separated) with sep character (default separator is tab).
  Consider a data file student.dat which consists of five-field: roll no, name, jdate, marks and grade separated by "'" .
- –r: It reverses the sort order.
- –n: It sort data in numeric order. By default, sorting is not numeric
- –c: It checks whether files are already sorted. If they are, it does nothing.
  $sort –c myfile

29. Explain the cut and paste command in detail.

➢ **Cut command**: The cut command is used to extract specified parts of each line in a file. It can be used to extract characters, fields, or bytes from each line. The syntax for the cut command is:
$cut [options] [file]
The options for the cut command are:

- -c: Extract characters. The characters to be extracted are specified as a list, separated by commas.
- -f: Extract fields. The fields to be extracted are specified as a list, separated by commas. The fields are numbered starting from 1.
- -d: Specify the delimiter. The delimiter is the character that separates the fields in each line. The default delimiter is the tab character.
- -s: Ignore empty lines.

For example, the following command will extract the first and third characters from each line in the file file.txt:

30. Explain cmp command in detail.

➢ The basic syntax of the cmp command is:
$cmp file1 file2
This will compare the two files file1 and file2 and print the results to standard output. If the files are identical, cmp will not print anything. If the files are different, cmp will print the byte and line number where the first difference is found.
The cmp command has a number of options that can be used to customize its behavior.
Some of the most useful options are:

- -s: This option suppresses the output of cmp. This can be useful if you only want to know if
- the files are identical or different, and you don't need to see the actual difference.
- -n: This option allows you to limit the number of bytes that cmp will compare. This can be useful if you only need to compare a small part of the files.
- -l: This option tells cmp to report if one file is an initial subsequence of the other. This can be useful for comparing files that are very similar.
- -v: This option causes cmp to print more verbose output. This includes the actual bytes that are different, as well as the byte and line number where the difference occurs.
- For more information on the cmp command, you can consult the man page:

$man cmp

Here are some examples of how to use the cmp command:

- # Compare two files and print the results to standard output
  $cmp file1 file2
- # Compare two files and suppress the output
  $cmp -s file1 file2
- # Compare two files and only compare the first 10 bytes
  $cmp -n 10 file1 file2
- # Compare two files and report if one file is an initial subsequence of the other cmp -l file1 file2
- # Compare two files and print more verbose output
  cmp -v file1 file2

31. Explain uniq and wc command in detail.

➢ **Uniq**: it gets one copy of one each line and write into the standard output. In other words, it reads unique lines from successive repeated lines and writes it to standard output.

Syntax:

Uniq[option]…..[]input file [output file]]

Some of the most useful options for the uniq command include:

- -d(duplicate): This option causes the uniq command to print the duplicate lines as well as the unique lines.
- -u(unique): This option causes the uniq command to only print the unique lines.
- -c(count): This option causes the uniq command to print the number of times each line appears in the file.

- -f(fields): This option allows you to specify the number of fields to compare when checking for duplicates.
- -s(specify): This option allows you to specify the number of characters to compare when checking for duplicates.

  Here are some examples of how to use the uniq command:
- To print the unique lines in the file myfile.txt, you would use the following command:
  $uniq myfile.txt
- To print the duplicate lines in the file myfile.txt, you would use the following command:
  $uniq -d myfile.txt
- To print the number of times each line appears in the file myfile.txt, you would use the following command:
  $uniq -c myfile.txt
- To print the unique lines in the file myfile.txt, ignoring the first 3 fields, you would use the following command:
  $uniq -f 3 myfile.txt

To print the unique lines in the file myfile.txt, ignoring the first 5 characters, you would use the following command:
$uniq -s 5 myfile.txt

- **WC**: It prints the number of lines, word and characters (bytes) for each input files(s). Moreover, it also prints a total of lines, word and characters if more than one files more than one file is specified.
  Syntax:
  Wc[option][filename(s)]

  Some of the most useful options for the wc command include:
- -l: This option causes the wc command to only print the number of lines in the file.
- -w: This option causes the wc command to only print the number of words in the file.
- -c: This option causes the wc command to only print the number of bytes in the file.
- -m: This option causes the wc command to count the number of characters in the file, including spaces and tabs.
- -L: This option causes the wc command to print the length of the longest line in the file.

  Here are some examples of how to use the wc command:
- To print the number of lines in the file myfile.txt, you would use the following command:

wc -l myfile.txt

- To print the number of words in the file myfile.txt, you would use the following command:
  wc -w myfile.txt
- To print the number of bytes in the file myfile.txt, you would use the following command:
  wc -c myfile.txt
- To print the number of characters in the file myfile.txt, including spaces and tabs, you would use the following command:
  wc -m myfile.txt
- To print the length of the longest line in the file myfile.txt, you would use the following command:
  wc -L myfile.txt

32. Define filter? Explain any three filter of UNIX in detail.

➢ **Filter**: In unix, a filter is a program that takes plain text (either stored in a file or produced by another program) as standard input, transforms it into a meaningful format, and then returns it as standard output. Filters are very small programs that are designed for a specific function, such as converting text to uppercase, removing all punctuation, or finding all lines that match a certain pattern.
Here are three of the most commonly used filters in Unix:

- **grep** : This filter is used to search for a pattern or regular expression in a file. The basic syntax of the grep command is as follows:
  grep PATTERN FILE Where PATTERN is the pattern or regular expression that you want to search for, and FILE isthe name of the file that you want to search.
  For example, the following command would search for all lines that contain the word "hello" in
  the file file.txt:
  grep hello file.txt
- **sort**: This filter is used to sort the lines in a file. The basic syntax of the sort command is as follows:
  sort FILE Where FILE is the name of the file that you want to sort
  The sort command can sort the lines in a file in ascending or descending order, and it can also sort the lines by different fields, such as the first word, the second word, and so on.
  For example, the following command would sort the lines in the file file.txt in ascending order by the first word:
  sort -n -k1 file.txt

- **uniq** : This filter is used to remove duplicate lines from a file. The basic syntax of the uniq command is as follows:
  uniq FILE Where FILE is the name of the file that you want to remove duplicates from The uniq command can also be used to print the number of times each line appears in a file.
  For example, the following command would remove duplicate lines from the file file.txt: uniq file.txt

33. Write a short note on user defined variable in UNIX.

➢ **Variable**:- In computer programming, a variable is a named location in memory that holds a value. The value of a variable can be changed during the execution of a program. Variables are used to store data that needs to be used repeatedly in a program Types of variable: Basically, variables in unix are of two types:

(1)User define variable
(2)System/environmental variable

User define variable: A user-defined variable in Unix is a variable that is created by the user and has a name that is chosen by the user. Userdefined variables are stored in the current shell environment and can be used by any command that is executed in the current shell.

Syntax:

```
variable_name=value
```

Where variable_name is the name of the variable and value is the value that you want to assign to the variable.
For example, the following command defines a variable called name and assigns the value "John Doe" to it:

```
name="John Doe"
```

Once a user-defined variable is defined, you can use it in any command by prefixing the variable name with a dollar sign ($). For example, the following command prints the value of the variable name:

```
echo $name
```

User-defined variables are case-sensitive, so the variable name is different from the variable NAME.
Here are some of the rules for defining user-defined variables in Unix:

- The variable name can be any combination of letters, numbers, and underscores.
- The variable name cannot start with a number.
- The variable name cannot be a reserved word.
- The value of the variable can be any type of data, such as a string, a number, or a list.

User-defined variables can be deleted using the unset command. The syntax for the unset command is as follows:
unset variable name
Here are some examples of how to use user-defined variables in Unix:

- To define a variable called age and assign the value 30 to it:
  age=30
- To print the value of the variable age:
  echo $age
- To define a variable called name and assign the value of the current user to it:
  name=$(whoami)
- To define a variable called list and assign a list of values to it:
  list=(1 2 3 4 5)
- To delete the variable name:
  unset name

34. Write a note on environment variable.

➢ When you start a Unix system, some variables are set at the time of system startup and some are after logging in. These variables are known as environmental variables. They are also known as Unix-defined or System variables. A user can control the Unix system using these variables. The environmental variables are available to the base/parent shell as well as any number of new sub-shell createdunder the base shell. In short, environment variables are inherited by the sub-shellfrom its base shell. The scope of the environment variables are extended to userenvironment, i.e. the sub shell that runs shell script, Unix utilities such as mail, vi etc.
In Unix, most of the environmental variables are in upper-case letter. Such as HOME, PATH, PS1, PS2, IFS, LOGNAME etc...
Let's discuss some of the environment variables in the following section

- **HOME**: variable contains the home directory path directory path as its value. This is the default directory of user and will be placed on it at the logging in.
  $echo $HOME<enter>
  /home/bharat
  Let's discuss some of the environment variables in the following section
- **PATH**: It is a variable that provides a search path to locate any executable command submitted at command line. PATH variable contains list of directories that are searched by the shell to locate a

command. A user can know the search path for executable file or command as follow:

```
$echo $PATH<enter>
/usr/local/bin:/bin:/usr/bin
```

- **IFS**: IFS contains a sequence of characters that are used as word separators during the interpretation of the command line. It contains invisible characters like the space, tab and new-line. A user can view the content of this variable by taking the octal dump as follow
- **PS1**: PSI variable contains the system prompt i.e. the $ symbol. This is the primary prompt of the system. The system prompt may be changed by setting the value of this variable to the desired prompt.
- **LOGNAME**: This variable contains the login/user name of the user. This variable is also set when a user is successful logged in to the Unix system. A user can view his login name by applying following command at sheprompt.
  ```
  $echo $LOGNAME<enter>
  Nirva
  ```
  Here nirva is login name of the current user
- **PWD**: It contains absolute pathname of current directory. The value of this variable is changed as soon as a user switches to another directory. A user can view the value of this variable using echo command as follow:
  ```
  $ echo $PWD
  /home/bharat/y2013
  ```
- **MAIL**: It contains absolute pathname of user's mailbox file. A user can view the value of this variable using echo command as follow:
  ```
  $ echo $MAIL
  /var/spool/mail/bharat
  ```
- **MAILCHECK**: It stores mail-checking interval for incoming mail. The value of this variable is in terms of seconds
  ```
  $echo$MAILCHECK
  ```
  It displays 60 which indicate that after 60 seconds the shell checks the file for the arrival of new mail. If t new mail is arrived, it informs user about new mail by the message as follow:
  ```
  You have new mail in /var/spool/mail/bharat
  ```
  Here, if a user bharat is running a command, he will get new arrival mail message on his terminal on after the command has accomplished its task.

35. Write a note on command line argument and positional parameter in detail.

➢ A command line argument is a value that is passed to a command when it is executed. Arguments are typically specified after the command name, separated by spaces.

- A positional parameter is a special variable in a shell script that stores the value of a command line argument. The positional parameters are numbered starting from 1, with $1 referring to the first argument, $2 referring to the second argument, and so on.
- For example, the following command line:
  ls -l /etc/passwd
  has two command line arguments: -l and /etc/passwd. The positional parameters for this command would be $1= -l and $2=/etc/passwd.
  positional parameters can be used in shell scripts to access the values of
- the command line arguments. For example, the following shell script:
  ```
  #!/bin/bash
  echo "The first argument is $1"
  echo "The second argument is $2"
  ```
  Here are some additional things to keep in mind about command line arguments and positional parameters:
- The order of the command line arguments is important. The first argument is assigned to $1, the second argument is assigned to $2, and so on.
- Command line arguments can be of any type, including strings, numbers, and special characters.
- Command line arguments can be enclosed in quotation marks to prevent the shell from interpreting special characters.
- The shift command can be used to shift the positional parameters by a specified number of positions

36. Write a note on control structure of UNIX.

➢ Control structures in Unix refer to the mechanisms and constructs that allow you to control the flow of execution in shell scripts or command-line operations. These structures are essential for decision-making, looping, and branching within Unix scripts. Here are some of the primary control structures in Unix:

1. if-then-else: The `if` statement is used to make decisions based on the evaluation of a condition. You can use the `then` clause to specify what should happen if the condition is true, and the `else` clause for the false condition. For example:

```
bash
if [ condition ]; then
else
fi
```

2. case-esac: The `case` statement is used for more complex conditional branching based on pattern matching. It allows you to specify multiple conditions and execute different commands for each case. For example:

```
bash
case $variable in
   pattern1)
      ;;
   pattern2)
      ;;
   *)
      ;;
esac
```

3. for loop: The `for` loop is used for iterating over a set of values, such as elements of an array or items in a list. For example:

```
bash
for item in item1 item2 item3; do
   # commands to execute for each item
done
```

4. while loop: The `while` loop is used to execute a set of commands as long as a specified condition is true. For example:

```
bash
while [ condition ]; do
done
```

5. until loop: The `until` loop is similar to the `while` loop but executes commands as long as the condition is false. It continues execution until the condition becomes true. For example:

```
bash
until [ condition ]; do
done
```

6. break and continue: Within loops, you can use the `break` statement to exit the loop prematurely and the `continue` statement to skip the current iteration and move to the next one.

These control structures provide the flexibility needed to create complex and conditional scripts in Unix. They allow you to make decisions, iterate through data, and create responsive and efficient shell scripts or command-line operations. Depending on the specific task at hand, you can choose the control structure that best fits your needs.

37. Write a note on shell interpretation at prompt

➢ The shell is a command-line interpreter that reads commands from the user and executes them. When the user presses Enter after typing a command, the shell performs the following steps:

- Breaks the command into words. The shell separates the command into words by spaces and tabs.
- Expands aliases and variables. The shell expands any aliases or variables that are used in the command.
- Looks for the command in the current directory. If the command is not found in the current directory, the shell looks for it in the directories listed in the $PATH environment variable.
- Executes the command. If the command is found, the shell executes it. The shell interpretation process is repeated until the user types the exit command to exit the shell.

38. Explain the output of ls -l command in detail

➢ The ls -l command lists the contents of the current directory in a long format. The output of the ls -l command is divided into 10 columns:

- **Permissions**: This column shows the permissions for the file or directory. The permissions are represented by a combination of characters, each of which has a specific meaning.
  r: The user has read permission for the file or directory.
  w: The user has write permission for the file or directory.
  x: The user has execute permission for the file or directory.
  -: The user does not have the specified permission.
  @: The file or directory is owned by the root user.
  +: The file or directory has extended attributes.
- **Number of links**: This column shows the number of hard links to the file or
- **directory**. A hard link is a way of creating multiple names for the same file.
  Owner: This column shows the name of the user who owns the file or directory.
- **Group**: This column shows the name of the group that owns the file or directory.
- **Size**: This column shows the size of the file or directory in bytes.
  Date and time: This column shows the date and time that the file or directory was last modified.
- **Filename**: This column shows the name of the file or directory.

The ls -l command can be used to list the contents of any directory. It is a very useful command for quickly getting information about the files and directories in a directory.

Here is an example of the output of the ls -l command:

```
total 16
-rw-r--r-- 1 bard staff 1000 Mar 22 12:00 file1.txt
drwxr-xr-x 2 bard staff 4096 Mar 22 12:00 directory1
```

39. Define process in detail.

➢ In Unix, a process is an instance of a running program. It is a program that is currently being executed by the operating system. Every process has a unique identifier called the process ID (PID). The PID is used by the operating system to track and manage the process. A process has a number of attributes, including:

- **Process ID (PID):** A unique identifier for the process.
- **Parent Process ID (PPID):** The PID of the process that created this process.
- **State**: The current state of the process. The possible states are:
  - **New**: The process has been created but has not yet started running.
  - **Ready**: The process is ready to run but is waiting for the CPU.
  - **Running**: The process is currently running on the CPU.
  - **Waiting**: The process is waiting for an event to occur, such as the completion of an I/O operation.
- **Terminated**: The process has finished executing.
- **CPU time:** The amount of CPU time that the process has used.
- **Memory usage**: The amount of memory that the process is using.
- **Open files:** The files that the process has open.
- **Threads**: The threads that the process has created.

The operating system manages processes by creating and destroying them, scheduling them to run on the CPU, and allocating resources to them. The operating system also provides a number of mechanisms for processes to communicate with each other, such as pipes and sockets.

Here are some additional things to keep in mind about processes in Unix:

- Each process has its own memory space. This means that processes cannot access each other's data directly.

- Processes communicate with each other through shared memory, pipes, or sockets.Processes can be created and destroyed dynamically.
- The operating system schedules processes to run on the CPU based on anumber of factors, such as the priority of the process and the amount of CPU time that the process has used.

40) Explain how user communicate with other user in UNIX

➢ There are many ways for users to communicate with each other in Unix. Some of the most common methods are:

Terminal: Users can communicate with each other directly through the terminal. This can be done using the write command, which sends a message to another user, or the talk command, which allows for a two-way conversation.

Email: Email is a more asynchronous communication method that allows users to send messages to each other even if they are not logged in at the same time. The mail command is used to send and receive email messages.

Chat: There are many chat applications available for Unix that allow users to communicate in real time. Some popular chat applications include IRC, XMPP, and Telegram.

File sharing: Users can share files with each other using a variety of methods, such as the scp command or a cloud storage service like Drop box or Google Drive.

Remote access: The ssh command can be used to connect to a remote system and access its resources, such as files and applications. This can be a useful way for users to collaborate on projects or troubleshoot problems.

1. **Write a note on the history of UNIX operating system.**

In late 1960's at AT&T Bell Laboratories the developers were working on the project known as MULTICS (Multiplexed Information and Computing Service). MULTICS was an operating system designed on GE-645 mainframe computers. The purpose of developing the operating was to have an operating system that can be portable and can support multiuser. It was developed with multiuser capability still it was inadequate; the project was dropped but the researchers working the system, Ken Thompson, Dennis Ritchie, M. D. McIlroy, and J. F. Ossanna, decided to redo the work on a much smaller scale. Ken Thompson still had access to the Multics environment; he wrote simulations for the new file and paging system on it. A team of Bell Labs researchers led by Thompson and Ritchie, including Rudd Canaday, developed a hierarchical file system, the notions of computer processes and device files, a command-line interpreter, and some small utility programs.

In the 1970s Brian Kernighan coined the project name Unics (Uniplexed Information and Computing Service) as a play on Multics. Unics could eventually support multiple simultaneous users, and it was renamed UNIX. UNIX code was written in the assembly language of PDP-7 and was, consequently machine dependent. Ritchie and Thompson worked quietly on UNIX for several years. Ken Thompson then developed a new programming "B" which was subsequently modified by Dennis Ritchie and renamed the "C" language. The whole UNIX system was rewritten in "C" language by 1973 which made is portable language.

2. **Explain the features of UNIX operating system.**

UNIX provides many features as given below:

1. **Multiuser Capability:**

UNIX is a multiuser system. In a multiuser system the same computers resources-hard disk, memory etc. are accessible to many users. All the users are given different terminal. All the terminals are connected to the main computer whose resources are availed by all users. So, a user at any of the terminals can use not only the computer, but also any peripherals that may be attached. e.g.: Printer. The following figure depicts the multiuser system.

The number of terminals connected to the host machine depends on the number of ports that are present in its controller card. There are several types of terminals that can be connected to the host machine.

- Dumb Terminals: These terminals consist of the keyboard and display unit with no memory or disk of its own. These can never act independent machines If they are to be used they have to be connected to the host machine.
- Terminal Emulation/Smart Terminal: PC has its own microprocessor, memory & storage device by attaching this PC to the host machine through a cable & running s/w from this machine, we can emulate to work as it is dumb terminal. However, memory & disk are not in use & machine cannot carry out any processes on its own.
- Dial-In terminals: These terminals use telephone lines to connect with host machine. To communicate over telephone line, it is necessary to attach a unit called modem to the terminal as well as to the host machine.

2. **Multitasking Capability:**

UNIX can carry out multiple jobs at the same time, i.e., a single user can type a program in its editor as well as simultaneously executes some other command you might have given earlier. The latter job is performed in the background while in the foreground he uses the editor. This is managed by dividing the CPU time intelligently between all processes being carried out.

3. **Inter Process Communication:**

UNIX provides excellent feature that allows users to communicate with fellow users. The communication can be within the network of a single main computer or between two or more networks. The users can easily exchange mail, dataand programs through such networks.

4. **Security**:

UNIX provides outstanding facility for security. It provides security in 3 levels.

- The first level security is provided by providing passwords (also called Login Level Security). It can be considered as a System Level Security. Username and passwords are assigned to all the users to assure that no other user can access his work.
- The second level security is at the file level. There are read, write, and execute permissions to each file that decide who can access a particular file, who can modify it and who can execute it.
- The third level security is given by file encryption. This utility encodes user's file into an unreadable format, so that even if the user succeeds in opening the file, he will not be able to read it. When user wants to see the contents of the file, he can decrypt the file.

5. **System Portability:**

UNIX is written in High level language" C" hence it can easily run on any machine with or without small changes. The code can be changed and compiled on a new machine.

6. **Open System:**

UNIX has an open architecture one can add to the toolkit by simply writing a program & starting that executable in a separate area in the file. A separate device can also be added by creating file for it. Modification of system is easy because the source code is always available.

7. **System calls and libraries:**

Unix is written in c language. There are many commands available in Unix that handles specialized function called system calls. These calls are built into the kernel, and all library functions and utilities are written them.

8. **Programming feature:**

Unix is highly programmable; it was designed for programmer, not a casual end user. The Unix shell programming language has all the necessary ingredients like control structures, loops and variables that establish it as a programming language in its own right.

9. **Networking:**

Unix was not originally a networking system. This concept was added to unix system after a split between BSD Unix and AT & T Unix. Both developers incorporated networking into heart of the operating system. Networking allows users at one location to log into system at the other ends and enables user to access the resource of host computer.

3. **Explain different level of security provided by UNIX operating system.**

Unix provides three types of securities.

i. System level
ii. Directory level
iii. File level

- System level:

In Unix, every user has been allocated login id and password. When the system administrator opens an account for user, an entry is created in system password file, called **/etc/passwd**. So to access the resources of the Unix system, user has to log in first. Only the authorized user can access the system.

- Directory level:

In Unix, every thing is treated as file. Even directories or devices are also considered as file. There are read, write, and execute permissions to each file, which decide who can access a particular file. Who can modify it and who can execute it.

- File level:

Lastly, there is file encryption. Encryption utility encodes your file into an unreadable format, so that even if someone succeeds in opening it, your secret information are safe.

4. **Differentiate between multiuser and multitasking operating system.**

The main difference between a multiuser and multitasking operating system is that multiuser operating system allows multiple users to access the computer system at the same time, while a multitasking operating system allows multiple programs to run at the same time.

A multiuser operating system allows multiple users to share the same computer system. Each user has their own account and can run their own programs. Multiuser Oss are typically used in servers, where they allow multiple users to access shared resources such as files, printers, and databases.

A multitasking operating system allows multiple programs to run at the same time on the same computer. The OS divides the cpu time between the programs, giving each programs a small amount of time to run. This allows the programs to appear to be running simultaneously, even though they are actually sharing the CPU.

Here are some examples of multiuser operating system:

- Unix
- Linux
- macOs

Here are some examples of multitasking operating system:

- windows
- os x
- android
- ios

5. **What is shell? Explain types of shell in detail.**

Shell:

Shell is the most widely used utility program on all Unix systems. It is loaded in memory when a user logs in. The shell establishes an interaction between the user and a kernel. The shell is also known as the command interpreter because it interprets the commands and passes it to the kernel.

Types of shells:

The shell runs like any other program under the Unix system. Hence, one shell program can replace the other, or call another shell program as it would call any other program. Due to this feature, several shells have been developed in response to difference needs of users. Some of the popular shells are listed below:

- **Bourne shell:**

This is one the most widely used shells in the Unix world. It was developed by **Steve Bourne** at **AT & T Bell Laboratories** in the **late 1970's**. It is the primary Unix command interpreter and comes along with every Unix system. It provides **'dollar prompt$'** on Unix installations as the trademark of the Bourne shell. The Bourne shell is an executable file named sh.

- **C Shell:**

Bill Joy developed C shell at the university of California. Its commands were supposed to like C statement. It has two advantages over the bourne shell. Firstly, it allows aliasing of commands. This is useful when lengthy commands can be renamed by user i.e. user gives short-name of frequently used lengthy command. So instead of typing the entire command, user can simply use the short alias at the command line and save typing time also.

- **Korn Shell:**

It is very powerful shell. It is superset of Bourne shell. It can completely replace the Bourne shell in a system and has several more functions that are build into the shell making it more efficient. The Korn shell includes all the enhancements in the C shell, like command history and aliasing, and offers a few more feature itself.

- **Bash Shell:**

It is an enhanced version of the Bourne shell. Bash stands for bourne again shell. It is used in Unix environment.

- **Tcsh Shell:**

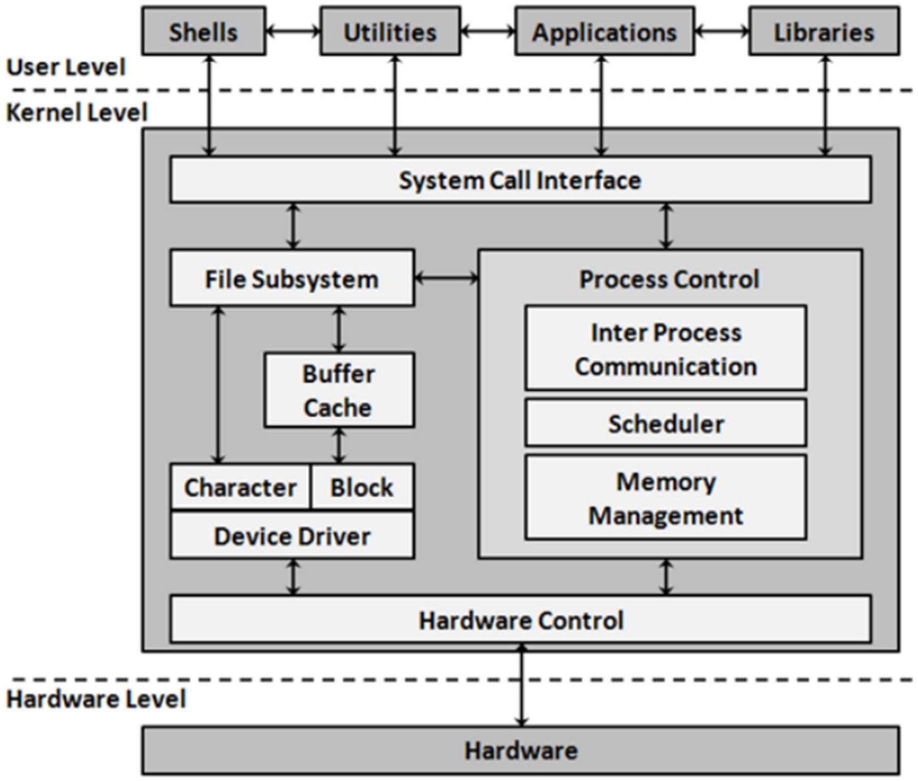
Tcsh is pronounced as tee-cee shell. It is a compatible version of the C shell. It is used in Linux environment.

- **PDKSH Shell:**

Pdksh stand for public domain korn shell. Linux offers pdksh as a substitute for ksh shell.

6. **Explain the architecture of UNIX operating system.**

The UNIX architecture can be divided into three levels: User level, Kernel level, Hardware level.



The system call and library Interface represent the border between user programs and the kernel as shown in the figure. The system calls are partitioned to the system calls that interact with the file sub system and the system calls that interact with the process control subsystem

The file subsystem manages the files, allocates files space, administrating the free space, controlling access to files, and retrieving data for users. Processes interact with the file system through system calls. E.g., Open, close, read, write, etc.

Device drivers are the kernel modules that control the operation of peripheral devices. Block I/O devices are random access storage device to the rest of the system.

The process control subsystem is responsible for process synchronization, inter-process communication, memory management, process scheduling. The system calls used with process control systems are fork (creating a new process), exec (overlay the image of a program onto the running process), Exit (finish executing a process).

The memory management module controls the allocation of memory. If at any time the system does not have enough physical memory for all processes, the kernel moves between main memory and secondary memory so that the all processes get a fair chance to execute.

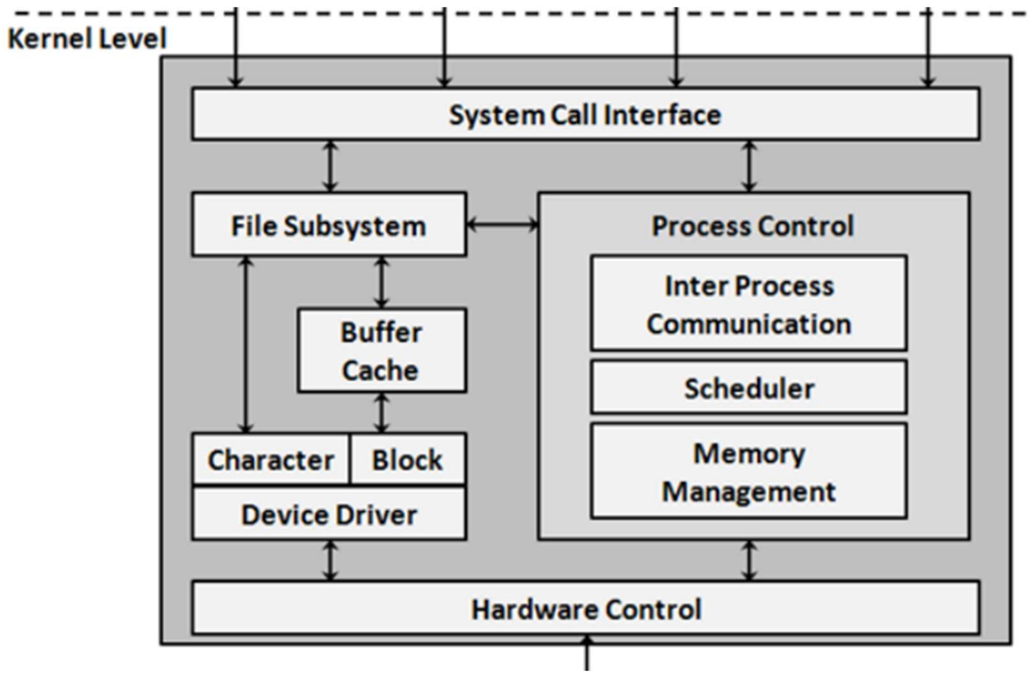
The scheduler module allocates the CPU to processes. It schedules them to run in turn until they voluntarily relinquish the CPU while awaiting a resource or until the kernel pre-empts them when their recent run time exceeds a time quantum.

The hardware control is responsible for handling interrupts and for communicating with the machine. Devices such as disks or terminals may interrupt the CPU while a process is executing. The kernel executes the interrupt and then resumes the previously executing process. This way it provides access of hardware devices.

7. **Explain the Kernel architecture in detail.**

Kernel:

The kernel is the core of the operating system. Kernel is mostly written in C. It is loaded into memory when the system is booted and communicates directly with the hardware. User programs that need to access the hardware use the services of the kernel, which performs the job on the user's behalf. The programs access kernel through set of functions called system calls. The kernel also manages the system memory, Schedules processes, decides their priorities, and performs another task



The file subsystem manages the files, allocates files space, administrating the free space, controlling access to files, and retrieving data for users. Processes interact with the file system through system calls. E.g., Open, close, read, write, etc.

Device drivers are the kernel modules that control the operation of peripheral devices. Block I/O devices are random access storage device to the rest of the system.

The process control subsystem is responsible for process synchronization, inter-process communication, memory management, process scheduling. The system calls used with process control systems are fork (creating a new process), exec (overlay the image of a program onto the running process), Exit (finish executing a process).

The memory management module controls the allocation of memory. If at any time the system does not have enough physical memory for all processes, the kernel moves between main memory and secondary memory so that the all processes get a fair chance to execute.

The scheduler module allocates the CPU to processes. It schedules them to run in turn until they voluntarily relinquish the CPU while awaiting a resource or until the kernel pre-empts them when their recent run time exceeds a time quantum.

8. **Define shell. Explain feature of shell in detail.**

- Interactive Environment:

The shell allows the user to create a dialog (communication channel) between the user and the host UNIX system. This dialog terminates until the user ends the system session.

- Shell scripts:

It is the shell that has the facility to be programmed; the shell contains commands that can be utilized by the user. Shell scripts are group of UNIX command string together and executed as individual files. The shell is itself a program; accept that is written in “C” language

- Input/Output Redirection:

Input/Output Redirection is a function of shell that redirects the o/p from program to a destination other than screen. This way you can save the o/p from a command into a file and redirect it to a printer, another terminal on the h/w or even another program. similarly, a shell can be a program that accepts i/p form other than keyboard by redirecting its i/p from another source.

- Programming Language:

The shell includes features that allow it to be used as programming language. This feature can be used to build shell script that performs complex operations.

- Shell variables:

The user can control the behaviour of the shell as well as other programs & utilities by storing data in variables.

9. **Write a note on Unix file system.**

Unix file system is a logical method of **organizing and storing** large amounts of information in a way that makes it easy to manage. A file is a smallest unit in which the information is stored. Unix file system has several important features. All data in Unix is organized into files. All files are organized into directories. These directories are organized into a tree-like structure called the file system. Files in Unix System are organized into multi-level hierarchy structure known as a directory tree. At the very top of the file system is a directory called "root" which is represented by a "/". All other files are "descendants" of root.

The Unix file system is a hierarchical file system used by Unix-based operating systems to store and organize files and directories. It is a tree-like structure that starts with a single directory called the root directory, which is denoted by a forward slash (/) character.

Unix file system also uses a set of permissions to control access to files and directories. Each file and directory have an owner and a group associated with it, and permissions can be set to allow or restrict access to these entities.

- **/ :** The slash / character alone denotes the root of the filesystem tree.
- **/bin :** Stands for "binaries" and contains certain fundamental utilities, such as ls or cp, which are generally needed by all users.
- **/boot :** Contains all the files that are required for successful booting process.
- **/dev :** Stands for "devices". Contains file representations of peripheral devices and pseudo-devices.

- **/etc :** Contains system-wide configuration files and system databases. Originally also contained “dangerous maintenance utilities” such as init,but these have typically been moved to /sbin or elsewhere.
- **/home :** Contains the home directories for the users.
- **/lib :** Contains system libraries, and some critical files such as kernel modules or device drivers.

10.write a note on types of Unix Files system.

The UNIX files system contains several different types of files

1. Ordinary files – An ordinary file is a file on the system that contains data, text, or program instructions. It is used to store your information, such as some text you have written or an image you have drawn. This is the type of file that you usually work with.

2. Directories – Directories store both special and ordinary files. For users familiar with Windows or Mac OS, UNIX directories are equivalent to folders. A directory file contains an entry for every file and subdirectory that it houses. If you have 10 files in a directory, there will be 10 entries in the directory.

3. Special Files – Used to represent a real physical device such as a printer, tape drive or terminal, used for Input/Output (I/O) operations. **Device or special files** are used for device Input/Output(I/O) on UNIX and Linux systems. They appear in a file system just like an ordinary file or a directory. On UNIX systems there are two flavours of special files for each device, character special files and block special files.

4. Pipes – UNIX allows you to link commands together using a pipe. The pipe acts a temporary file which only exists to hold data from one command until it is read by another. A Unix pipe provides a one-way flow of data. The output or result of the first command sequence is used as the input to the second command sequence. To make a pipe, put a vertical bar (|) on the command line between two commands.

5. Sockets – A Unix socket (or Inter-process communication socket) is a special file which allows for advanced inter-process communication. A Unix Socket is used in a client-server application framework. In essence, it is a stream of data, very similar to network stream (and network sockets), but all the transactions are local to the filesystem.

6. Symbolic Link – Symbolic link is used for referencing some other file of the file system. Symbolic link is also known as Soft link. It contains a text form of the path to the file it references. To an end user, symbolic link will appear to have its own name, but when you try reading or writing data to this file, it will instead reference these operations to the file it points to. If we delete the soft link itself, the data file would still be there. If we delete the source file or move it to a different location, symbolic file will not function properly.

11.Explain the characteristic of UNIX file system.

The characteristic of Unix file system is as below:

1. **Hierarchical Organization:**

The top-level directory of the hierarchy is traditionally called **root** (written as a slash /). The tree "grows" downward from the root, with paths connecting the root to each of the other files. At the end of each path is an ordinary file or a directory file. *Ordinary files*, frequently just called *files*, are at the ends of paths that cannot support other paths. When you refer to the tree, *up* is toward the root, and *down* is away from the root. Directories directly connected by a path are called *parents* (closer to the root) and *children* (farther from the root).

2. **Case-sensitive:**

The Unix file system is case-sensitive, which means that the capitalization of file names matters. For example, the files file.txt and FILE.TXT would be considered to be the same file. The case-sensitivity of the unix file system can be a bit confusing at first, but it can also be helpful. For example, it can be used to prevent accidentally overwrite a file with same name but different capitalization.

3. **Permissions:**

The Unix file system uses permissions to control who can access a file or directory and what they can do with it. There are three types of permissions:

1) Read Permission allows the user to read the contents of the file or directory.
2) Write Permission allows the user to modify the contents of the file or directory.
3) Execute Permission allows the user to run the file as a program.

Each file and directory has three sets of permissions, one for the owner of the file or directory, one the group that the file or directory belongs to, and one for everyone else. The permissions are represented by three characters, each of which can be one of the following:

- r for read permission
- w for write permission
- x for execute permission

4. **Device files:**

Device files are a special type of file in the unix file system that represent physical devices, such as hard drives, printers, and network interfaces. Device files are used to access and control these devices.

There are two general kinds of device file in Unix:

- Character special files: Character special files are used to represent devices that can be read or written one character at a time, such as a keyboard or a serial port.
- Block special files: block special files are used to represent devices that can be read or written in blocks of data, such as a hard drive or a USB drive.

12.Explain the UNIX file system component in detail.

The Unix file system is a hierarchical file system used by Unix-based operating systems to store and organize files and directories. It is a tree-like structure, with the root directory at the top and all other directories and files breaching off from it.

The main components of the Unix file system are:

1) **Files:** Files are the basic unit of storage in the Unix system. They can contain any types of data, such as text, images, programs, and so on.
2) **Directories:** Directories are used to organize files. They can contain other directories, as well as files.
3) **Permission:** permission control who can access a file or directory and what they can do with it. There are three types of permission: read, write and execute.

4) **Hard links:** Hard links are pointers to a file. They allow you to create multiple names for the same file.
5) **Symbolic links:** symbolic links are pointers to another file or directory. They are often used to create shortcuts or aliases.
6) **Special files:** Special files represent devices, such as hard disks, printers, and terminals.

13. Write a note on I-NODE

INODE (short for index node) is a data structure in a Unix-style file system that descries a file-system object such as file or directory. Each inode stores the attributes and disk block locations of the object's data. File-system object attributes may include metadata, as well as owner and permission data. A directory is a list of inodes with thiery assigned names. The list includes an entry for itself, its parent, and each of its children.

The inode is a critical data structure in a unix-sytle file system. It is used to keep track of all the files and directories in the file system, and it provides the information that is needed to access and manipulate these files and directories.

The following are some of the information that is stored in an inode:

- file type (regular file, directory, etc.)
- file size
- file permissions
- owner and group IDs
- last access time
- last modification time
- last change time
- direct block pointer
- single indirect pointer
- double indirect pointer
- triple indirect pointer

The inode number is a unique identifier for each inode. It is used to locate the inode in the file system. The inode number is also used to reference the inode in file system operations.

The number of inodes in a file system is limited. This means that the number of files and directories that can be created in the file system is also limited. The number of inodees in file system is typically specified when the file system is created.

Inodes are an important part of the Unix-style file system. They provide the information that is needed to access and manipulate files and directories. The Inode is critical data structure that makes the unix-style file system efficient and reliable.

14.Explain UNIX directory structure in detail.

The Unix directory structure is a hierarchical file system structure, much like an upside-down tree, with the root directory at the base of the file system and all other directories spreading from there.

The root directory is the top-level directory in the file system. It contains all the other directories and files on the system. Some of the most common directories under the root directory include:

- **/bin**: this directory contains executable programs that are essential for the system to function, such as the ls and cp commands.
- /etc: This directory contains configuration files for the system, such as the passwd file, which stores user account information.
- /home: This directory contains the home directories for all users on the system.
- /lib: This directory contains shared libraries that are used by executable programs.
- /mnt: This directory is used to mount filesystems from other devices, such as USB drives or network shares.
- /opt: This directory is used to install optional software packages.
- /root: This directory is the home directory for the root user, who has full administrative privileges on the system.

In addition to these common directories, there are many other directories that may be present on a UNIX system, depending on the specific distribution and configuration.

The UNIX directory structure is a flexible and powerful way to organize files and directories. It allows users to easily find the files they need and to manage their files effectively.

Here are some of the advantages of the UNIX directory structure:

- It is hierarchical, which makes it easy to organize files and directories into a logical structure.
- It is scalable, so it can be used to store a large number of files and directories.
- It is portable, so it can be used on different types of computers.
- It is secure, with permissions that can be set to control access to files and directories.

Here are some of the disadvantages of the UNIX directory structure:

- It can be difficult to navigate, especially for large file systems.
- It can be difficult to manage permissions, especially for large file systems.
- It is not as user-friendly as some other file system structures, such as the Windows file system.

15. Explain the structure of /etc/passwd file.

The /etc/passwd file is a plain text file that contains information about all user accounts on a Linux or Unix system. Each line in the file represents one user account and contains seven fields, separated by colons (:). The fields are:

- Username: The username of the account. This is the name that the user will use to log in to the system.
- Password: The password of the account. This field is typically encrypted, but in some older systems, it may be stored in plaintext.
- User ID (UID): A unique identifier for the user account. Each user account must have a unique UID.
- Group ID (GID): The primary group ID of the user account. This is the group that the user will be a member of by default.

- Gecos: A comment field that can be used to store additional information about the user account, such as the user's full name or office phone number.
- Home directory: The absolute path to the user's home directory. This is the directory that the user will be placed in when they log in to the system.
- Login shell: The absolute path to the user's login shell. This is the program that the user will run when they log in to the system.

The /etc/passwd file is owned by the root user and has permissions of 644. This means that it can only be read and written by the root user and by users with sudo privileges.

The /etc/security/passwd file is a newer file that is used to store the encrypted passwords for all user accounts. The /etc/passwd file only contains a reference to the /etc/security/passwd file for each user account. This allows the encrypted passwords to be stored in a more secure location.

The /etc/passwd and /etc/security/passwd files are important system files that should be protected from unauthorized access. Only authorized users should be able to read or modify these files.

16.Explain the structure of /etc/shadow file.

The /etc/shadow file is a text file that contains information about the passwords of all users on a Unix system. It is owned by the root user and the shadow group, and has permissions of 0640. The file is divided into nine fields, separated by colons (:):

1. Username: The username of the user account.
2. Encrypted password: The encrypted password of the user account. The encryption algorithm used is specified by the first character of the field. The following algorithms are supported:

   1: MD5

   2a: Blowfish

   2y: Eksblowfish

5: SHA-256

6: SHA-512

3. Last password change: The date of the last password change, expressed as the number of days since January 1, 1970 (Unix time).
4. Minimum password age: The minimum number of days that must pass before a user can change their password.
5. Maximum password age: The maximum number of days that a password can be used before it must be changed.
6. Password inactive days: The number of days after the password expires before the account is locked.
7. Grace logins: The number of times a user can log in with an expired password before their account is locked.
8. Account expiry date: The date on which the account will expire.
9. Reserved: This field is reserved for future use.

The /etc/shadow file is not directly readable by users, but it can be accessed by programs with the appropriate permissions, such as the passwd command.

The /etc/shadow file is an important security feature that helps to protect the passwords of users on a Linux system. By keeping the passwords encrypted and inaccessible to unauthorized users, the /etc/shadow file helps to prevent unauthorized access to user accounts.

17.Explain booting process and init process in detail.

The booting process is the sequence of events that occurs when a computer is turned on. It starts with the BIOS (Basic Input/Output System), which is a small program that is stored in a chip on the motherboard. The BIOS performs a few basic checks of the hardware, and then loads the bootloader.

The bootloader is a program that loads the operating system into memory. The bootloader is typically stored in a partition on the hard drive, but it can also be stored on a USB drive or other removable media.

Once the bootloader has loaded the operating system, the init process is started. The init process is responsible for initializing the system and starting up the basic services that the operating system needs to run.

The init process is typically implemented as a daemon, which is a program that runs in the background and does not require user interaction. The init process is responsible for starting up other daemons, such as the network daemon, the filesystem daemon, and the graphical user interface (GUI) server.

The init process also controls the run levels of the system. he run level is a configuration of the system that determines which daemons are running. There are typically several run levels, such as single-user mode, multi-user mode, and graphical mode.

The following are the steps involved in the booting process of a Unix system:

- The BIOS performs a power-on self-test (POST).
- The BIOS loads the bootloader.
- The bootloader loads the kernel.
- The kernel initializes the hardware.
- The kernel loads the initramfs.
- The initramfs mounts the root filesystem.
- The init process is started.
- The init process starts up the basic services.
- The init process enters the default run level.

The following are the steps involved in the init process of a Unix system:

- The init process reads the /etc/inittab file.
- The init process determines the current run level.
- The init process starts up the daemons that are associated with the current run level.
- The init process monitors the system and restarts any daemons that fail.
- The init process waits for a shutdown signal.

18.Explain File Access permission in UNIX system.

File access permissions in Unix systems control who can read, write, and execute a file. There are three types of permissions:

- Read (r): The ability to read the contents of the file.
- Write (w): The ability to modify the contents of the file.
- Execute (x): The ability to execute the file as a program.

Each file has three sets of permissions, one for the owner of the file, one for the group that owns the file, and one for everyone else. The permissions are represented by three characters, one for each type of permission. The characters can be either a hyphen (-), which means that the permission is denied, or a letter, which means that the permission is granted. The letters are r, w, and x for read, write, and execute, respectively.

Here are some additional things to keep in mind about file access permissions in Unix systems:

- The owner of a file is the user who created the file.
- The group that owns a file is the group that the owner belongs to.
- The permissions for everyone else are the permissions that are granted to users who do not belong to the group that owns the file.
- The permissions for a file can be inherited by directories and subdirectories.
- The permissions for a file can be changed using the chmod command.

19.Explain ls command in detail with all option.

The ls command is a Unix command that lists the contents of a directory. It is one of the most basic and commonly used commands in Unix.

The basic syntax of the ls command is:

ls [options] [files or directories]

The options control the output of the ls command. Some of the most common options are:

-a : Lists all files, including hidden files. Hidden files are files that start with a dot (.).

-l : Lists all files in long format. This format shows more information about each file, such as the file permissions, size, and modification time.

-r : Lists the files in reverse order.

-t : Lists the files by modification time, with the most recently modified files first.

-F : Appends a character to each file name to indicate its type. For example, directories are marked with a slash (/), executable files are marked with an asterisk (*), and symbolic links are marked with an arrow (->).

You can also use the ls command to list the contents of directories recursively. This means that the ls command will list the contents of all subdirectories, as well as the contents of the current directory. To do this, use the -R option.

Here are some other useful ls options:

- -d : Lists directories only.
- -i : Lists files by inode number.
- -n : Lists files by numerical user ID (UID) and group ID (GID).
- -h : Displays file sizes in human-readable format.
- -color : Colorizes the output of the ls command.

20.Explain the output of ls command.

The ls command is used to list the contents of a directory. The output of the ls command is divided into several columns, each of which represents a different piece of information about the file or directory. The columns are:

- File type: The first character in the column indicates the type of file. The possible values are:
    - - (Hyphen): Regular file
    - d: Directory
    - l: Symbolic link
    - b: Block device
    - c: Character device
    - s: Socket
    - f: FIFO
- Permissions: The next 9 characters in the column represent the permissions for the file. The permissions are divided into three sets of three characters, one for each user type: owner, group, and others. Each character in a set can be one of the following:
    - r: Read permission
    - w: Write permission
    - x: Execute permission
    - -(Hyphen): No permission
- Number of hard links: The next column shows the number of hard links to the file. A hard link is a way of creating multiple names for the same file.
- Owner: The next column shows the name of the owner of the file.
- Group: The next column shows the name of the group that owns the file.
- Size: The next column shows the size of the file in bytes.
- Date and time: The last column shows the date and time when the file was last modified.

21. Explain cat command in detail with its option.

Ans: Cat stands for concatenate. It displays the content of one or more files. It is similar to type command of MS-DOS

Syntax:

Cat [option] [file1][file2]…

The above syntax indicates that you can use cat command without any option and filename(s). there are several purpose of cat command.

- To display the content of file flie1 present in working directory then we issue command at dollar
  Prompt as follow:

```
$cat file1 <enter>
This is file1
$
```

- The full path-name can be specified to display a file stored in another directory.

```
$cat /usr/user/file2<enter>
This is flie2
$
```

- It can also display contents of one or more file.

```
$cat file1 file2<enter>
This is file1
This is flie2
$
```

- If you use cat command without any arguments then it will take input from standard input devices(i.e. keyboard) and display them on standard output devices until user press <ctrl=d>.This character is used to terminate standard input.

```
$cat<enter>
This is line1<enter>
This is line1
This is line2<enter>
This is line2
<ctrl+d>
$
```

The output shows that all lines display in duplicate. This is because when we apply a cat command at prompt without any argument it waits for input. If we type a line and then press an <enter> key, the inputted line is again display on standard output.

<u>Note:</u> In Unix <ctrl+d> character is used to mark for end -of –end- file. It is also known as input redirection character**.**

- User can create a file using cat command. To create a file the command is:

```
$cat >file1
This is line1 of file1<enter>
This is line2 of file1<enter>
<ctrl+d>
$
```

Here the '>' symbol is used for output redirection.

- If the file file1 is already exist then the above command will erase the content of file. To keep the existing content of the file and want to append more into the line, then then we can use '>>' symbol.

```
$cat >> file
This is the last line of file1<enter>
<ctrl+d>
$
```

- Cat can also accept standard input and output.

```
$cat <f1>f1.out
```

- **Some of the option used with cat command is as follow.**

I. **<u>–v(visual)</u> :** This option display control character and other non-printing character. Consider a file f1 the contains tab characters at the end of first line and control character (shift+tab) at the end of second line. If we write a command without option it displays contents as follow:

```
$cat f1
This is a tab characters
This is a control characters
$
```

If we used –v option then the output display control character e.g. shift+tab as ^[[z

```
$ cat-vf1

This is a tab characters
This is a control characters^[[z
$
```

II. **<u>–e or –E</u>:** It prints a $ mark the end of fine. In other word, it display $ instead of new – line character

```
$cat –e f1
This is a tab character
This is a control character ^[[z
```

III. **–t:** It print character as ^I and form character as ^I

```
$cat –t f1
This is a tab character
This is a control character ^[z
```

IV. **–n:** it number each line of file(s) and written to standard output

```
$cat –n file1
1: This is line 1 of file 1
2: This is line 2 of file 1
3: This is line 3 of file 1
$
```

22. Explain rm command in detail with its option.

rm: rm stands for remove. It deletes one or more files/directories. Its equivalent MS-DOS command is Del or erase.

Syntax:

rm[option] file(s)/directory-name(s)

This command is used to remove a file or a directory. Let us see some example of it.

- To remove a file file1 from the current directory then the command is

```
$rm file1<enter>
$
```

Here the rm command is silent i.e. it immediately display prompt without any message. It indicates that the file is removed successfully. Otherwise display a message on the screen.

- You can remove more than one file like this:

```
$rm file1 file2
```

It removes file1 and file2 from the current directory.

- **Some of the option used with rm command as follow**

- **-i (interactive):** This option removes files interactively. It display prompt for removal of file(s). To remove only selected files from the directory then we can apply a command as

  ```
  $rm –I *
  File1 : y
  File2 : y
  …so on
  $
  ```

  To keep file, press 'n' or hitting <enter> key and to remove it press y/Y.

- **-r (recursive):** It is used to remove non-empty directory, together with all the files and subdirectories it contains.

- The-roption is used with directory name.

  ```
  $rm -r mydir
  $
  ```

  It removes a mydir directory with all files and subdirectories within it.

- Also, you can removes more than one directories using rm command with option-r.

  ```
  $rm-r dl d2d3
  $
  ```

  It removes three directories d1, d2 and d3 within the current directory.

- There is no meaning to use-r option with regular file like this:

  ```
  $rm-r file1
  $
  ```

- **-f (force):**To remove a file, you must have write permission to that file. If you do not have write permission on the file, you will be prompted (y or n) to override i.e. it display prompt for removal of write-protected file. This option is used to remove a file forcibly without displaying write-protected message.

- Let`s consider a write-protected file f1 and user issued a command like this:

  ```
  $rm f1
  rm: remove write-protected regular file 'f1'? y
  ```

```
$
```

Here, it display write-protected message for file fl. If user want to suppress this message during removal of write-protected file then issued a command like this:

```
$rm –f f1

$
```

23. Explain cp command in detail with its option.

Ans: cp stands for copy. It creates a duplicate file(s) having access and modification date-time is similar to current system date-time. Its equivalent MS-DOS command is copy.

Syntax:

cp[option] filename/directory-name filename/directory-name

This command required two arguments to copy a file or a directory. The first and second argument is also known as source and destination file/directory respectively.

- The following command sequences show that how you can create duplicate file of file1 in the same directory with different file name file2.

```
$ls<enter>
File1
$cp file1 file2<enter>
#file copied successfully
$ls<enter>
File1
file2
```

cp command does not display success message when a file is copied successfully. Above command sequences create a duplicate file file2 having same contents and permission as file1 have but the modification and access date-time of file2 is changed to current system date-time. Files can be copied to and from another sub-directory also.

- Files can be copied to and from another sub-directory also

```
$cp/usr/bca1/file1 /usr/bca2/file2<entert>
$
```

- If we have copied a file from one directory to another directory without the name of destination file- name then it creates a file in another directory with same name as source file.

```
$cp/usr/bcal/filel/usr/bca2/<enter>
```

  Here, it creates a duplicate file file1 in/usr/bca2 directory.

- It also copy one or more files with the same name to another directory.

```
$cp file1 file2 file3 my_dir
$
```

  Here file1, file2 and file3 are copied with the same name into the directory my_dir

Some of the options used with cp command are as follow:

- **-r(recursive):**The –r option recursively copy an entire directory structure to another directory.

```
e.g. $cp –r olddir newdir
$
```

  It recursively copies files/subdirectories to a new directory newdir. If newdir is exist in the current directory then it creates new directory olddir in the directory newdir.

- **-i (interactive):**Sometimes, a destination file is already exist and user want to show an overwriting prompt for an existing file during copy then-i option is used.

  For example, consider that we have two files f1 and 2 under the current directory and apply a command as follow:

```
$cp f1  f2
$
```

  Then, it copies a file f1 to f2 without any warning. But if we apply a command-with-i option, it will warn you before overwrite a file like this:

```
e.g.     $cp-i f1 f2
             cp: overwrite f2? Y
                 $
```

  Answer ‘y’ overwrites file f2 whereas any other responses ignore the operation

- **-p (preserve):**Normally, copy operation changes access and modification date-time of destination file to current system date-time. This option preserves these (access and modification date-time) attributes of file(s).

```
$ls-1f1:ls-luf1
-rw-rw-r-- 1 bharattmtbca 133 Jul 23 12:01 f1#modification date-time
-rw-rw-r-- 1 bharattmtbca 133 Jul 24 14:45 f1      #access date-time

$cp-pf1f2; ls-1f2;ls-luf2

-rw-rw-r-- 1 bharattmtbca 133 Jul 23 12:01 f1 #modification date-time
-rw-rw-r-- 1 bharattmtbca 133 Jul 24 14:45 f1      #access date-time
```

The example shows that the access and modification date-time of file f1 is inherited to file f2.

- **-l (link):**This option creates a link instead of copying a file. In other words, it creates an alias name of the file. You can create link of file f1 like this:

```
$cp-lf1f1.ln
$
```

NOTE: cp command, if the source contains multiple files or directory, then the destination must be a directory

24. Explain mv command in detail with its option.

Ans:-mv stands for move. It is used to move an individual file, a list of files, or a directory from one directory to another. It is also used to rename a file/directory. Its equivalent MS-DOS command is move. The general syntax is:

Syntax:
mv [option] filename(s)/directory-name filename/directory-name

- In Unix a file can be moved one directory to another directory

```
$mv/usr/user1/file1/use/user/file2<enter>
```

- A user can move several files at once to the same destination directory by first write name of all the files to be moved and giving the name of the destination last.

$mv file1 file2 file3 dir_new

- To rename a file in the current directory, you can also use mv, but with the new filename as the destination.

```
$mv old file new file
        $
```

- It removes old file to new_file.

❖ **-I (interactive):** it warns you overwrite an existing file.it display the filename followed by a question mark, Answer y/Y overwrites the file otherwise ignore operation of that file

✓ To overwrite file1 to file2 the command is :

```
$mv –I file1 file2
Mv: overwrite file2?  Y
```

❖ **-f (Force):**it suppress interactive overwrite message

NOTES: Moving files is very faster than copying a file in UNIX. Using my command, the actual contents of a file are not moved. It really moves an entry in an inode table that tells the system what directory the data is in. So the size of the file being moved has no bearing on the time taken by the mv command.

25. Explain any two file comparison command

❖ **cmp :**This command compares two files byte by byte. If the two files are identical, the cmp command will not output anything. If the two files are different, the cmp command will output the byte number where the first difference is found.

For example, to compare the files file1.txt and file2.txt, you would use the following command:

```
cmp file1.txt file2.txt
```

- **diff :**This command compares two files line by line. It shows the lines that are different in the two files, as well as the lines that are only in one file or the other.

For example, to compare the files file1.txt and file2.txt, you would use the following command:

```
diff file1.txt file2.txt
```

The diff command has many options that can be used to control how the comparison is performed. For example, the -b option can be used to compare the files byte by byte, like the cmp command.

26. What are the difference between copying a file and moving a file? Explain with its example.

Copying:

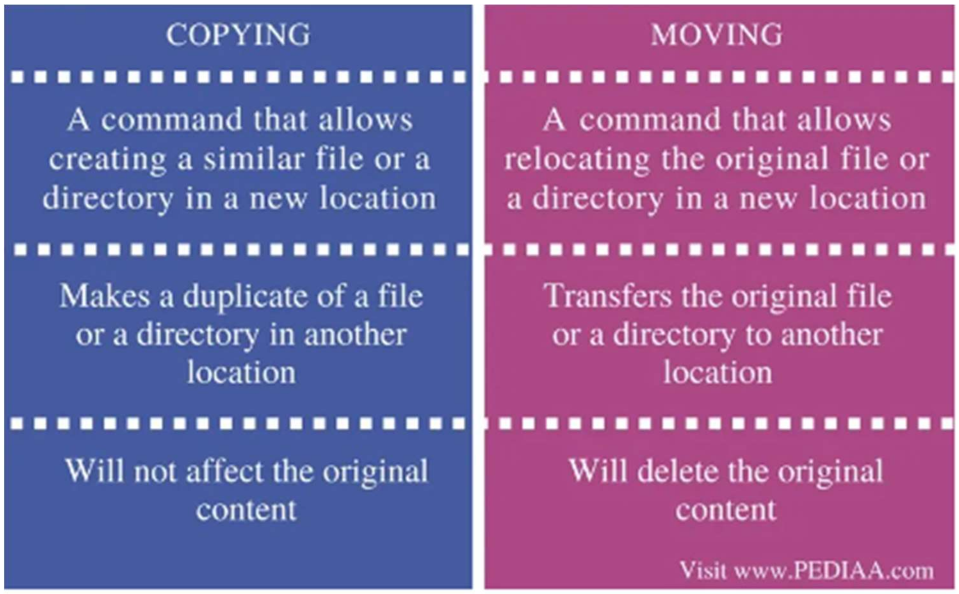
Copying makes a duplicate of a file or directory into a new location. In Linux, the **"cp" command** allows the user to copy a file or a directory.

The following command creates a copy of file1 and names it as file2. If the file2 already exists, it will be overridden with the new content.

```
cp file1.txt  file2.txt.
```

Moving: Moving transfers the original files or directories from one location to another. It deletes the content from the first location and creates content in a new location. In Linux, the **mv command** is used to move a file or a directory to a new location.

Example: `$mv file1 file2 file3 dir_new`



COPYING	MOVING
A command that allows creating a similar file or a directory in a new location	A command that allows relocating the original file or a directory in a new location
Makes a duplicate of a file or a directory in another location	Transfers the original file or a directory to another location
Will not affect the original content	Will delete the original content

Visit www.PEDIAA.com

27. Explain head and tail command.

1. Head: The head command is used to view the first few lines of a file. By default, it will display the first 10 lines of a file, but this number can be changed with the ‘-n’ option. The syntax for the head command is as follows −

```
$ head [options] [file(s)]
```

❖ head command options:

The head command has several options that can be used to customize its output. Some of the most used options are −

• **-n** −the -n option is used to specify the number of lines to display.

For example, to view the first 20 lines of a file named "example.txt", the command would be:

```
$ head -n 20 example.txt
```

- **-q** − the -q option is used to suppress header printing when multiple files are used.
- **-v** − the -v option is used to always print headers when using multiple files.

2.Tail: The **tail** command is used to display the last few lines of a file. Like the head command, tail will display the last 10 lines of a file by default, but this number can be changed with the **-n** option. The syntax of the tail command is as follows −

```
$ tail [options] [file(s)]
```

❖ tail command options

The **tail** command also has several options that can be used to customize its output. Some of the most used options are −

-n − The -n option is used to specify the number of lines to display.

For example, to view the last 20 lines of a file named "example.txt", the command would be:

```
$ tail -n 20 example.txt
```

-f − The '-f' option is used to keep the file open and continue displaying output as the file grows. This option is useful when working with log files.

```
$ tail -f example.log
```

-F − The '-F' option is similar to the '-f' option, but it also controls file **truncation**.

28. Write a sort command detail with its all option.

Sort: sort command can be used for sorting the contents of a file. Besides sorting, it can merge multiple sorted files and store the result in specified output file.

Syntax:

```
sort [option) [file(s) ]
```

Sorting proceeds character-by-character and begins with first character of each line. If first characters of two lines are same then sorting carry on to next character and so forth

- ✓ (Consider the following command.

```
$sort f1        #display sorted content of file f1
```

  It sort contents of input file f1 and display it on screen

- ✓ Consider the following command

```
$sort f1 f2 f3
```

  It concatenates contents of input files f1,f2 and f3 and writes collectively sorted output to standard output.

- ✓ To merge the contents of a file with the standard input the command like this:

```
$sort-f1       #'-'indicates standard input file
```

❖ **Various option used with sort command are listed below**

- **-o filename:** It writes sorted output in a specified to file filename.

  - ✓ sorts three files f1, f2 and f3 and writes sorted content of these files into an output file called mfile then the command is as follow:

```
$sort –o mfile f1 f2 f3 <enter>
```

  - ✓ A user can also redirect sorted output to another file using output redirection.

```
$sort f1 f2 f3 > mfile<enter>
```

- **–u:** It removes duplicate lines from the sorted output i.e. the output contains only unique lines.

```
$sort –uf1<enter>
```

- **–m:** It merges list of files that have already been sorted.

  $sort-m fl f 2 #files fl andf2 must be sorted

- **–t sep:** It sorts the file on any field delimited (separated) with sep character (default separator is tab).

Consider a data file student.dat which consists of five-field: roll no, name, jdate, marks and grade separated by "" .

- **–r:** It reverses the sort order.

- **–n**: It sort data in numeric order. By default, sorting is not numeric

- **–c:** It checks whether files are already sorted. If they are, it does nothing.

  $sort –c myfile

29. Explain the cut and paste command in detail.

Cut command: The cut command is used to extract specified parts of each line in a file. It can be used to extract characters, fields, or bytes from each line. The syntax for the cut command is:

$**cut [options] [file]**

❖ **The options for the cut command are:**

- **-c:** Extract characters. The characters to be extracted are specified as a list, separated by commas.
- **-f:** Extract fields. The fields to be extracted are specified as a list, separated by commas. The fields are numbered starting from 1.
- **-d:** Specify the delimiter. The delimiter is the character that separates the fields in each line. The default delimiter is the tab character.
- **-s:** Ignore empty lines.

For example, the following command will extract the first and third characters from each line in the file file.txt:

```
$cut -c 1,3 file.txt
```

This will output the following:

```
a b
c d
e f
```

The following command will extract the first and third fields from each line in the file file.txt:

```
$cut -f 1,3 file.txt
```

This will output the following:

```
a b
c d
e f
```

The following command will extract the characters from the 5th to the 10th from each line in the file file.txt:

```
$cut -c 5-10 file.txt
```

This will output the following:

```
bcde
def
fgh
```

The following command will ignore empty lines in the file file.txt and extract the first and third characters from each line:

```
$cut -c 1,3 -s file.txt
```

This will output the following:

```
a b
c d
e f
```

- ❖ **Paste command:** The paste command is used to join two or more files horizontally. The syntax for the paste command is:

```
$paste [options] [file1] [file2]...
```

- ❖ **The options for the paste command are:**

- **-d:** Specify the delimiter. The delimiter is the character that will be used to separate the lines from each file. The default delimiter is the tab character.
- **-s:** Suppress leading and trailing blanks.

For example, the following command will join the files file1.txt and file2.txt horizontally, using the tab character as the delimiter:

```
$paste file1.txt file2.txt
```

This will output the following:

```
a b c
d e f
```

The following command will join the files file1.txt and file2.txt horizontally, using the comma as the delimiter and suppressing leading and trailing blanks:

```
$paste -d , -s file1.txt file2.txt
```

This will output the following:

```
a,b,c
d,e,f
```

30. Explain cmp command in detail.

The basic syntax of the cmp command is:

```
$cmp file1 file2
```

This will compare the two files file1 and file2 and print the results to standard output. If the files are identical, cmp will not print anything. If the files are different, cmp will print the byte and line number where the first difference is found.

The cmp command has a number of options that can be used to customize its behavior. Some of the most useful options are:

- **-s:** This option suppresses the output of cmp. This can be useful if you only want to know if the files are identical or different, and you don't need to see the actual difference.

- **-n:** This option allows you to limit the number of bytes that cmp will compare. This can be useful if you only need to compare a small part of the files.
- **-l:** This option tells cmp to report if one file is an initial subsequence of the other. This can be useful for comparing files that are very similar.
- **-v:** This option causes cmp to print more verbose output. This includes the actual bytes that are different, as well as the byte and line number where the difference occurs.

- ❖ **For more information on the cmp command, you can consult the man page:**

$man cmp

Here are some examples of how to use the cmp command:

# Compare two files and print the results to standard output

$cmp file1 file2

# Compare two files and suppress the output

$cmp -s file1 file2

# Compare two files and only compare the first 10 bytes

$cmp -n 10 file1 file2

# Compare two files and report if one file is an initial subsequence of the other
cmp -l file1 file2

# Compare two files and print more verbose output
cmp -v file1 file2

31. Explain uniq and wc command in detail.

1. **Uniq:** it gets one copy of one each line and write into the standard output. In other words, it reads unique lines from successive repeated lines and writes it to standard output.

Syntex:

Uniq[option]…..[]input file [output file]]

Some of the most useful options for the uniq command include:

- **-d(duplicate):** This option causes the uniq command to print the duplicate lines as well as the unique lines.
- **-u(unique):** This option causes the uniq command to only print the unique lines.
- **-c(count):** This option causes the uniq command to print the number of times each line appears in the file.
- **-f(fields):** This option allows you to specify the number of fields to compare when checking for duplicates.
- **-s(specify):** This option allows you to specify the number of characters to compare when checking for duplicates.

❖ **Here are some examples of how to use the uniq command**:

- To print the unique lines in the file myfile.txt, you would use the following command:

```
$uniq myfile.txt
```

- To print the duplicate lines in the file myfile.txt, you would use the following command:

```
$uniq -d myfile.txt
```

- To print the number of times each line appears in the file myfile.txt, you would use the following command:

```
$uniq -c myfile.txt
```

- To print the unique lines in the file myfile.txt, ignoring the first 3 fields, you would use the following command:

```
$uniq -f 3 myfile.txt
```

- To print the unique lines in the file myfile.txt, ignoring the first 5 characters, you would use the following command:

```
$uniq -s 5 myfile.txt
```

2. **WC:** It prints the number of lines, word and characters (bytes) for each input files**(s)**. Moreover, it also prints a total of lines, word and characters if more than one files more than one file is specified.

Syantx:

```
Wc[option][filename(s)]
```

❖ Some of the most useful options for the wc command include:

- **-l:** This option causes the wc command to only print the number of lines in the file.
- **-w:** This option causes the wc command to only print the number of words in the file.
- **-c:** This option causes the wc command to only print the number of bytes in the file.
- **-m:** This option causes the wc command to count the number of characters in the file, including spaces and tabs.
- **-L:** This option causes the wc command to print the length of the longest line in the file.

❖ Here are some examples of how to use the wc command:

- To print the number of lines in the file myfile.txt, you would use the following command:

```
wc -l myfile.txt
```

- To print the number of words in the file myfile.txt, you would use the following command:

```
wc -w myfile.txt
```

- To print the number of bytes in the file myfile.txt, you would use the following command:

```
wc -c myfile.txt
```

- To print the number of characters in the file myfile.txt, including spaces and tabs, you would use the following command:

```
wc -m myfile.txt
```

- To print the length of the longest line in the file myfile.txt, you would use the following command:

```
wc -L myfile.txt
```

32. Define filter? Explain any three filter of UNIX in detail.

- **Filter:** In unix, a filter is a program that takes plain text (either stored in a file or produced by another program) as standard input, transforms it into a meaningful format, and then returns it as standard output. Filters are very small programs that are designed for a specific function, such as converting text to uppercase, removing all punctuation, or finding all lines that match a certain pattern.

❖ **Here are three of the most commonly used filters in Unix:**

- **grep :** This filter is used to search for a pattern or regular expression in a file. The basic syntax of the grep command is as follows:

```
grep PATTERN FILE
```

Where PATTERN is the pattern or regular expression that you want to search for, and FILE is the name of the file that you want to search.

For example, the following command would search for all lines that contain the word "hello" in the file file.txt:

```
grep hello file.txt
```

- **sort:** This filter is used to sort the lines in a file. The basic syntax of the sort command is as follows:

  ```
  sort FILE
  ```

  Where FILE is the name of the file that you want to sort

  The sort command can sort the lines in a file in ascending or descending order, and it can also sort the lines by different fields, such as the first word, the second word, and so on.

  For example, the following command would sort the lines in the file file.txt in ascending order by the first word:

  ```
  sort -n -k1 file.txt
  ```

- **uniq :** This filter is used to remove duplicate lines from a file. The basic syntax of the uniq command is as follows:

  ```
  uniq FILE
  ```

  Where FILE is the name of the file that you want to remove duplicates from

  The uniq command can also be used to print the number of times each line appears in a file.

  For example, the following command would remove duplicate lines from the file file.txt:

  ```
  uniq file.txt
  ```

33. Write a short note on user defined variable in UNIX.

Variable:- In computer programming, a variable is a named location in memory that holds a value. The value of a variable can be changed during the execution of a program. Variables are used to store data that needs to be used repeatedly in a program

Types of variable: Basically, variables in unix are of two types:

(1)User define variable

(2)System/environmental variable

(1)User define variable: A user-defined variable in Unix is a variable that is created by the user and has a name that is chosen by the user. User-defined variables are stored in the current shell environment and can be used by any command that is executed in the current shell.

Syntax:

```
variable_name=value
```

Where variable_name is the name of the variable and value is the value that you want to assign to the variable.

For example, the following command defines a variable called name and assigns the value "John Doe" to it:

```
name="John Doe"
```

Once a user-defined variable is defined, you can use it in any command by prefixing the variable name with a dollar sign ($). For example, the following command prints the value of the variable name:

```
echo $name
```

User-defined variables are case-sensitive, so the variable name is different from the variable NAME.

Here are some of the rules for defining user-defined variables in Unix:

- The variable name can be any combination of letters, numbers, and underscores.
- The variable name cannot start with a number.
- The variable name cannot be a reserved word.
- The value of the variable can be any type of data, such as a string, a number, or a list.

User-defined variables can be deleted using the unset command. The syntax for the unset command is as follows:

unset variable_name

❖ **Here are some examples of how to use user-defined variables in Unix:**

- To define a variable called age and assign the value 30 to it:

  age=30

- To print the value of the variable age:

  echo $age

- To define a variable called name and assign the value of the current user to it:

  name=$(whoami)

- To define a variable called list and assign a list of values to it:

  list=(1 2 3 4 5)

- To delete the variable name:

  unset name

34. Write a note on environment variable.

When you start a Unix system, some variables are set at the time of system startup and some are after logging in. These variables are known as environmental variables. They are also known as Unix-defined or System variables. A user can control the Unix system using these variables. The environmental variables are available to the base/parent shell as well as any number of new sub-shell created under the base shell. In short, environment variables are inherited by the sub-shell from its base shell. The scope of the environment variables are extended to user environment, i.e. the sub shell that runs shell script, Unix utilities such as mail, vi etc**.**

In Unix, most of the environmental variables are in upper-case letter. Such as HOME, PATH, PS1, PS2, IFS, LOGNAME etc...

Let's discuss some of the environment variables in the following section

- **HOME:** variable contains the home directory path directory path as its value. This is the default directory of user and will be placed on it at the logging in.

```
$echo $HOME<enter>

/home/bharat

$
```

- Let's discuss some of the environment variables in the following section

- **PATH:** It is a variable that provides a search path to locate any executable command submitted at command line. PATH variable contains list of directories that are searched by the shell to locate a command. A user can know the search path for executable file or command as follow:

```
$echo $PATH<enter>

/usr/local/bin:/bin:/usr/bin
```

- **IFS:** IFS contains a sequence of characters that are used as word separators during the interpretation of the command line. It contains invisible characters like the space, tab and new-line. A user can view the content of this variable by taking the octal dump as follow:

- **PS1:** PSI variable contains the system prompt i.e. the $ symbol. This is the primary prompt of the system. The system prompt may be changed by setting the value of this variable to the desired prompt.

- **LOGNAME:** This variable contains the login/user name of the user. This variable is also set when a user is successful logged in to the Unix system. A user can view his login name by applying following command at she prompt.

```
$echo $LOGNAME<enter>

Nirva

$
```

Here nirva is login name of the current user

- ❖ **PWD:** It contains absolute pathname of current directory. The value of this variable is changed as soon as a user switches to another directory. A user can view the value of this variable using echo command as follow:

```
$ echo $PWD

/home/bharat/y2013

$
```

- ❖ **MAIL:** It contains absolute pathname of user's mailbox file. A user can view the value of this variable using echo command as follow:

```
$ echo $MAIL

/var/spool/mail/bharat
```

```
$
```

- **MAILCHECK:** It stores mail-checking interval for incoming mail. The value of this variable is in terms of seconds

```
$echo$MAILCHECK
60
$
```

It displays 60 which indicate that after 60 seconds the shell checks the file for the arrival of new mail. If t new mail is arrived, it informs user about new mail by the message as follow:

You have new mail in /var/spool/mail/bharat

Here, if a user bharat is running a command, he will get new arrival mail message on his terminal on after the command has accomplished its task.

35. Write a note on command line argument and positional parameter in detail.

- A command line argument is a value that is passed to a command when it is executed. Arguments are typically specified after the command name, separated by spaces.
- A positional parameter is a special variable in a shell script that stores the value of a command line argument. The positional parameters are numbered starting from 1, with $1 referring to the first argument, $2 referring to the second argument, and so on.

For example, the following command line:

```
ls -l /etc/passwd
```

has two command line arguments: -l and /etc/passwd. The positional parameters for this command would be $1= -l and $2=/etc/passwd.

positional parameters can be used in shell scripts to access the values of the command line arguments. For example, the following shell script:

```
#!/bin/bash

echo "The first argument is $1"

echo "The second argument is $2"
```

Here are some additional things to keep in mind about command line arguments and positional parameters:

- The order of the command line arguments is important. The first argument is assigned to $1, the second argument is assigned to $2, and so on.
- Command line arguments can be of any type, including strings, numbers, and special characters.
- Command line arguments can be enclosed in quotation marks to prevent the shell from interpreting special characters.
- The shift command can be used to shift the positional parameters by a specified number of positions

36. Write a note on control structure of UNIX.

Control structure Control structure alters the flow of execution of shell script. The Unix supports two types of control structure:

✓ **Decision making control structure**

✓ **Loop control structure.**

Decision making control structure allows the computer to take the decision depending upon the condition and executes the statement(s) or skips some statements. The Loop control structure helps computer to execute a group of statement repeatedly for specified number of time

- **Decision making control structure**

  The shell offers two decision making control structure:

  (a) if statement

(b) case-esac statement

(a) **if statement:** It takes decisions, depending on the fulfillment of a certain condition. There are three types of if statement.

✓ if-then-fi statement

✔ if-then-else-fi statement

✓ if-then-elif-else-fi statement

- **if-then-fi statement**: It is a one-way decision making statement. The general syntax of this statement is:

  Syntax:

  ```
  if control command

  then

  statement(s)

  fi
  ```

- **if-then-else-fi statement:** It is a two-way decision making statement. The general syntax of this statement is:

  Syntax:

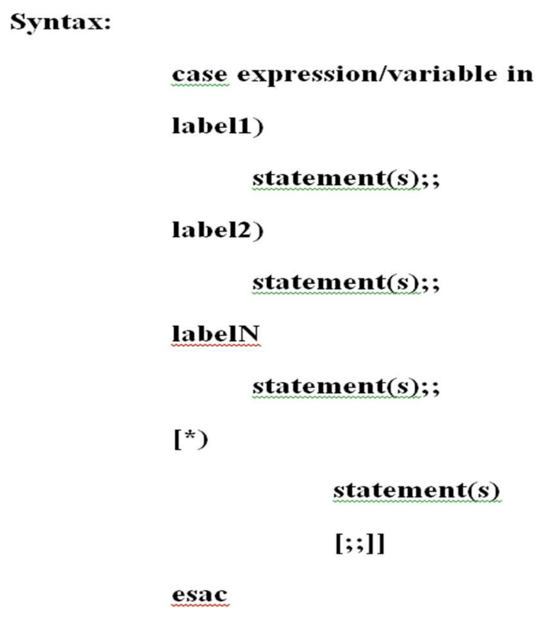
  ```
  if control command

  then

  statement(s)

  else

  statement(s)

  fi
  ```

- **if-then-elif-else-fi statement:** This is the third form of if construct. A user can use this construct instead of multilevel if-then-else statements. The syntax for this statement is:

Syntax:

```
if control-command1
then
        statement(s)
elif control-command2
then
        statement(s)
else
        statement(s)
fi
```

(b) The case statement: The case statement is a better organizational tool than a multilevel if-then-else-fi statement. The case statement will enable you to match several values against one variable. It is useful in menu-driven script. The syntax for the case statement is as follows:



Syntax:

```
case expression/variable in
label1)
        statement(s);;
label2)
        statement(s);;
labelN
        statement(s);;
[*)
                statement(s)
                [;;]]
esac
```

✓ **Loop control structure:** A loop involves repeating some portion of the program either a specified number of times or until a particular condition is being satisfied. There are three methods by which we can repeat a part of program. They are

- **while loop**
- **until loop**
- **for loop**

(a) while loop: To repeat a part of program fixed number of times then while loop is used. The general form of while loop is as follow:

Syntax:

```
while control-command

do

        statement(s)

done
```

✓ **For example, you can display 1 to 10 as follow:**

```
 n=1

while [$n-le 10]

do

        echo $n n=`expr $n+1
        done
```

b) until loop: It is a complement loop of while that means until loop is executed till the exit status of control command is non- zero (false) otherwise control transfer to the next statement following the done. The general form of until loop is as follow:

Syntax:

```
until control-command
do
        statement(s)
done
```

✓ **For example, you can display 1 to 10 as follow:**

```
 n=1
 until [$n -gt 10]
do
        echo $n
        n =`expr $n+1
done
```

(c) for loop: Unlike while and until, for does not test exit status of control statement. The working of this loop is different than the other two loops.

The general form of for loop is**:**

```
for control-variable in value1 value2....valueN
do
```

```
        statement(s)
    done
```

✓ **To display 1 to 4 using for as follow: for i in 1234 do**

```
    For I in 1 2 3 4
    do
        echo $i              Output: 1
    done                             2
                                     3
                                     4
```

37. Write a note on shell interpretation at prompt

The shell is a command-line interpreter that reads commands from the user and executes them. When the user presses Enter after typing a command, the shell performs the following steps:

- ✓ **Breaks the command into words.** The shell separates the command into words by spaces and tabs.
- ✓ **Expands aliases and variables**. The shell expands any aliases or variables that are used in the command.
- ✓ **Looks for the command in the current directory**. If the command is not found in the current directory, the shell looks for it in the directories listed in the $PATH environment variable.
- ✓ **Executes the command**. If the command is found, the shell executes it.

The shell interpretation process is repeated until the user types the exit command to exit the shell.

- ✓ The shell interprets the command line as a single line**.** This means that you cannot break the command line into multiple lines and expect the shell to execute it correctly.
- ✓ The shell does not execute any commands that are preceded by a # character. This is used to comment out commands.
- ✓ The shell interprets special characters, such as |, &&, and ||. These special characters are used to control the flow of execution of the commands

38. Explain the output of ls -l command in detail

The ls -l command lists the contents of the current directory in a long format. The output of the ls -l command is divided into 10 columns:

- ❖ **Permissions:** This column shows the permissions for the file or directory. The permissions are represented by a combination of characters, each of which has a specific meaning.

- **r:** The user has read permission for the file or directory**.**
- **w:** The user has write permission for the file or directory.
- **x:** The user has execute permission for the file or directory**.**
- **-:** The user does not have the specified permission.
- **@:** The file or directory is owned by the root user.
- **+:** The file or directory has extended attributes.

- ❖ **Number of links:** This column shows the number of hard links to the file or directory. A hard link is a way of creating multiple names for the same file.

- ❖ **Owner:** This column shows the name of the user who owns the file or directory**.**
- ❖ **Group:** This column shows the name of the group that owns the file or directory**.**
- ❖ **Size:** This column shows the size of the file or directory in bytes.
- ❖ **Date and time:** This column shows the date and time that the file or directory was last modified**.**

❖ **Filename:** This column shows the name of the file or directory.

The ls -l command can be used to list the contents of any directory. It is a very useful command for quickly getting information about the files and directories in a directory.

Here is an example of the output of the ls -l command:

```
total 16

-rw-r--r-- 1 bard staff 1000 Mar 22 12:00 file1.txt

drwxr-xr-x 2 bard staff 4096 Mar 22 12:00 directory1
```

In this example, the first line shows the total size of all the files and directories in the current directory. The next two lines show the details of the two files in the current directory. The first file, file1.txt, is a regular file that is owned by the user bard and the group staff. The file has a size of 1000 bytes and was last modified on March 22, 2023 at 12:00 PM. The second file, directory1, is a directory that is owned by the user bard and the group staff. The directory has a size of 4096 bytes and was last modified on March 22, 2023 at 12:00 PM.

39. Define process in detail.

In Unix, a process is an instance of a running program. It is a program that is currently being executed by the operating system. Every process has a unique identifier called the process ID (PID). The PID is used by the operating system to track and manage the process.

A process has a number of attributes, including:

- **Process ID (PID):** A unique identifier for the process.
- **Parent Process ID (PPID):** The PID of the process that created this process.
- **State**: The current state of the process. The possible states are:
- **New:** The process has been created but has not yet started running.
- **Ready:** The process is ready to run but is waiting for the CPU.
- **Running:** The process is currently running on the CPU.
- **Waiting:** The process is waiting for an event to occur, such as the completion of an I/O operation.
- **Terminated:** The process has finished executing.
- **CPU time:** The amount of CPU time that the process has used.
- **Memory usage:** The amount of memory that the process is using.
- **Open files**: The files that the process has open.

- **Threads:** The threads that the process has created.

The operating system manages processes by creating and destroying them, scheduling them to run on the CPU, and allocating resources to them. The operating system also provides a number of mechanisms for processes to communicate with each other, such as pipes and sockets.

❖ **Here are some additional things to keep in mind about processes in Unix:**

- Each process has its own memory space. This means that processes cannot access each other's data directly.
- Processes communicate with each other through shared memory, pipes, or sockets.
- Processes can be created and destroyed dynamically.
- The operating system schedules processes to run on the CPU based on a number of factors, such as the priority of the process and the amount of CPU time that the process has used.

40. Explain the Grep,sed utility

Grep and sed are two powerful text processing utilities in Unix. Grep is used to search for patterns in text, while sed is used to edit text.

❖ **Grep:**The grep command searches a file or set of files for lines that match a specified pattern. The pattern can be a regular expression, which is a special syntax for describing patterns in text.

The basic syntax for the grep command is as follows:

```
grep pattern [file ...]
```

For example, the following command would search the file file.txt for lines that contain the word "hello":

```
grep hello file.txt
```

The grep command can also be used to search multiple files. To do this, you can specify the names of the files after the pattern. For example, the following command would search the files file1.txt, file2.txt, and file3.txt for lines that contain the word "hello":

```
grep hello file1.txt file2.txt file3.txt
```

The grep command has a number of options that can be used to control its behavior. For example, the -n option prints the line number of each matching line, and the -i option ignores case when matching patterns.

- ❖ **Sed:** The sed command is used to edit text. It can be used to search for and replace text, insert and delete text, and perform other text transformations.

The basic syntax for the sed command is as follows:

```
sed [options] 'command' file
```

The command is a sed script that specifies the text transformations that should be performed. The sed script is a series of instructions that are separated by semicolons.

For example, the following sed script would search for the word "hello" and replace it with the word "goodbye":

```
sed 's/hello/goodbye/g' file
```

The sed command has a number of options that can be used to control its behavior. For example, the -n option suppresses the output of the sed script, and the -i option edits the file in place.

Grep and sed are powerful tools that can be used to search, edit, and transform text. They are essential tools for any Unix user.

- ❖ Here are some additional things to keep in mind about grep and sed:

- Grep is a command-line tool, while sed is a stream editor. This means that grep reads the entire file into memory before it starts searching, while sed reads the file line by line and performs the edits as it goes.
- Grep is a simpler tool than sed. It is easier to learn and use, but it is less powerful.
- Sed is a more powerful tool than grep. It can be used to perform complex text transformations, but it is more difficult to learn and use.

41. Explain the awk utility

awk name comes from the initial letter of three authors: Alfred V. Aho, Peter J. Weinberger and Brian W. Kernigham.

The awk is a powerful programming language designed as utility. Some extent, its behavior is similar to sed utility. It reads the input file, line by line and performs an action on a part of the line or on the entire line. It is a programming language that permits easy manipulation of structured data and generation of formatted reports from the text file(s). It also accepts regular expression (RE) for pattern matching, has C language type programming constructs, variables and several built-in functions.

- ❖ **Some of the features of an awk utility are as follow:**

- It has an ability to view a text file in terms of records and fields.
- It makes use of variable assignment, arithmetic and string operators to manipulate records.
- It uses common programming constructs, such as loops, conditions and iterations.
- It has an ability to generate formatted reports
- It uses regular expression for pattern matching and filtering data.
- It allows the user to specify instructions in a script.

- It is stream oriented; reading input from text file(s) or from standard input files one at a time and directing the result to standard output to a file.

- **Structure of awk**

The general form of this utility is as follow: Syntax:

awk option 'instruction' filename(s)

An instruction part of awk program has three sections.

(i)BEGIN section

(ii) Processing section

(iii) END section.

Therefore, the detailed structure of awk program is as follow:

```
awk'  BEGIN{action}

Selection criteria {action}

END {action}' [filename(s)]
```

At a time either of the section is compulsory that is any one of the section is required in awk utility.

awk scans each record of each input file. For each record only selection criteria or an action or a combination of both is available. If a selection criterion is specified then it tests the input record to determine whether or not the action should be applied to it. If the selection criteria is not specified then for each record, an action will be performed. The action consists of statements, functions and expressions. The pattern should always be enclosed within slashes

42. Discuss the communication command in UNIX

There are a number of communication commands in Unix. Some of the most common ones are:

- write
- talk.
- wall
- mail
- ssh

.

1. write: The write command is used to send a message to another user who is logged into the same system. The syntax for the write command is as follows:

```
write username
```

Where username is the name of the user that you want to send the message to.

For example, the following command would send a message to the user johndoe:

```
write johndoe
```

The write command will only work if the user you are trying to write to is logged into the same system.

2. talk: The talk command is used to establish a real-time chat session with another user who is logged into the same system. The syntax for the talk command is as follows:

```
talk username
```

Where username is the name of the user that you want to chat with.

For example, the following command would establish a chat session with the user johndoe:

```
talk johndoe
```

The talk command will only work if the user you are trying to talk to is logged into the same system.

3. wall: The wall command is used to send a message to all users who are logged into the system. The syntax for the wall command is as follows:

```
wall message
```

Where message is the message that you want to send.

For example, the following command would send a message to all users who are logged into the system:

The wall command can be used to send important messages to all users, such as system alerts or news announcements.

4. mail: The mail command is used to send an email message to another user. The syntax for the mail command is as follows:

```
mail username
```

Where username is the name of the user that you want to send the email to

For example, the following command would send an email message to the user johndoe:

```
mail johndoe
```

The mail command will create a new email message in your mailbox. You can then edit the message and send it to the recipient.

5. ssh: The ssh command is used to establish a secure shell connection to another computer. The syntax for the ssh command is as follows:

```
ssh username@hostname
```

Where username is the name of the user that you want to connect to, and hostname is the name of the computer that you want to connect to.

For example, the following command would establish a secure shell connection to the computer example.com as the user johndoe:

```
ssh johndoe@example.com
```

The ssh command can be used to connect to remote computers and execute commands on them. It can also be used to transfer files between computers.

43.Explain how user communicate with other user in UNIX

There are many ways for users to communicate with each other in Unix. Some of the most common methods are:

- **Terminal:** Users can communicate with each other directly through the terminal. This can be done using the write command, which sends a message to another user, or the talk command, which allows for a two-way conversation.
- **Email**: Email is a more asynchronous communication method that allows users to send messages to each other even if they are not logged in at the same time. The mail command is used to send and receive email messages.
- **Chat:** There are many chat applications available for Unix that allow users to communicate in real time. Some popular chat applications include IRC, XMPP, and Telegram.
- **File sharing:** Users can share files with each other using a variety of methods, such as the scp command or a cloud storage service like Drop box or Google Drive.
- **Remote access:** The ssh command can be used to connect to a remote system and access its resources, such as files and applications. This can be a useful way for users to collaborate on projects or troubleshoot problems.