October/november2013 [1 marks]

1) Display the processes of user ‘bca01’

➔ `ps -u bca01`

2) List the user from /etc/passwd in the alphabetically sorted order.

➔ `cut -d: -f1 /etc/passwd | sort`

3) Count the frequency of users who are logged in from more than one terminal

➔ `who | awk '{ print $1 }' | sort | uniq -d | uniq -c`

4) Display string in upper case of the file f1.txt.

➔ `cat f1.txt | tr '[:lower:]' '[:upper:]'`

5) write a command to print lines which contain ‘Accounts’. **[IMP]**

➔ `grep 'Accounts' filename`

6) Write a command to print lines which do not contain ‘Accounts’**[IMP]**

➔ `grep -v 'Accounts' filename`

7) Write a command to print line 10 to 15**[IMP]**

➔ `sed -n '10,15p' filename`

8) Write a command to substitute ‘doshi’ with ‘desai’

➔ `sed 's/doshi/desai/g' inputfile > outputfile`

9) Write a command to print lines with line numbers which contain “Marketing”.

➔ `grep -n 'Marketing' filename`

10) Write a command to display line which starts with ‘The’.

➔ `grep '^The' filename`

October/november2014

Write command using sed or grep for the following:

11) Display two lines starting from 7th line of file X1.

➔ `sed -n '7,8p' X1`

12) Display the lines that do not contain “UNIX”. **[IMP]**

➔ `grep -v 'UNIX' filename`

13) Display the lines which are not starting with 2 at the beginning. **[MIMP]**

➔ `grep -v '^2' filename`

14) Write a command to replace ‘UIX OS’ on line no 5th to 10th.

➔ `sed '5,10s/UIX OS/NEW OS/g' filename`

15) Write a command to display all file name containing only digits in a filename.

➔ `find /your/directory -type f -regex '.*/[0-9]+'`

Write a command using awk utility:

1) Print odd numbers of words in each line. **[MIMP]**

➔ `awk '{ for (i = 1; i <= NF; i += 2) { printf "%s ", $i } printf "\n" }' filename`

2) Count occurrences of pattern ‘unix’ in file f1

➔ `grep -o 'unix' f1 | wc -l`

3) Display those words whose length greater than 8 characters and consist of alphabet only

➔ `grep -o '\b[[:alpha:]]\{9,\}\b' filename`

4) Print lines which end with 5, 6, 7 from file f1.

➔ `grep '[567]$' f1`

5) Write a script to print 1 to 10 nos.

➔

```
 for ((i=1; i<=10; i++)); do
    echo $i
done
chmod +x print_numbers.sh
```

6) Write a command to print those lines where field2 is computer field 3 > 15000 from sales file.

➔ **awk '$2 == "computer" && $3 > 15000' sales**

October/november2015

1) Display lines 1 to 5 from file x1

➔ **head -n 5 x1**

2) Display last word of each line from a file x1

➔ **awk '{print $NF}' x1**

3) To count number of characters in first line of file x1

➔ **awk 'NR==1 {print length}' x1**

4) Display all line that start with 'let', from a file x1. The letters l, e or t may be in any case.

➔ **grep -i '^let' x1**

Write a command using sed for the following:

1) Substitute 'endif ' with 'fi' on line 10 of file X1.

➔ **sed -i '10s/endif /fi/' X1**

2) Display three lines starting from 5th line of file X1.

➔ **sed -n '5,7p' X1**

3) Display all lines before string "Unix" from file X1.

➔ **sed -n '/Unix/q;p' X1**

4) Display all blank lines between lie 10 and 20 of file X1

➔ **sed -n '10,20{/^$/p}' X1**

October/november2016

write command for following

1) To concatenate two files f1 and f2 including some text to be inserted from keyboard after content of f1.

➔ **cat f1 - <(echo "Text to insert from the keyboard") f2 > combined_file**

2) To display all files having last character as digit.

➔ **find /path/to/directory -type f -name "*[0-9]"**

3) Remove directory tree dir/dir2/dir3 using single command.

➔ **rm -rf dir/dir2/dir3**

4) Write a command to display system processes.

➔ **ps aux**

5) Assign value of 10th positional parameter to a variable x.

➔ **x="$10"**

Perform following commands using grep or sed:

1) Display line 10 to 15 from file X1.

➔ **cat X1 | sed -n '10,15p'**

2) Display line before a line that contains pattern xyz in f1.txt.

➔ **grep -B 1 "xyz" f1.txt**

3) To display those lines in between 25 and 50 having pattern 'unix' in it.

➔ **sed -n '25,50{/unix/p}' yourfile.txt**

4) To remove all leading space from a file f1.

➔ **sed -i 's/^[\t]*//' f1**

November / December – 2016

Write command for following:

16) Write a command to sort a line of file and also remove a repeated line.**[MIMP]**

➔ **sort file.txt | uniq**

17) Write a command to print first seven lines of file.

➔ **head -n 7 yourfile.txt**

18) To count number of characters in last line of line x1. .**[IMP]**

➔ **tail -n 1 x1 | wc -m**

Perform following commands using grep or sed:

19) Display lines having exactly 50 characters of file X1

➔ **grep -E '^.{50}$' X1**

20) Display lines 10 to 15 from file X1

➔ **sed -n '10,15p' X1**

21) Display the directory listing [imp]

➔ **ls**

22) To count number of words in line 40 to 60 of file f1.txt.

➔ **sed -n '40,60p' f1.txt | tr -s ' ' '\n' | wc -w**

Write a command using awk utility:

23) Display those words whose length greater than 10 characters and consist of alphabet only. [imp]

➔ **grep -E '\b[a-zA-Z]{11,}\b' yourfile.txt**

24) Count occurrences of pattern 'Operating System' in file f1[mimp]

➔ **grep -o 'Operating System' f1 | wc -l**

October / November – 2017

25) To delete all special characters from the files f1.txt.

➔ **tr -d '[:punct:]' < f1.txt > f1_no_special_chars.txt**

26) Write a command to print first ten line of file.

➔ **head -n 10 yourfile.txt**

27) Replace all occurrences of 'SYBCA' with 'TYBCA' in 5th line of file f1.txt.

➔ **awk 'NR==5 {gsub("SYBCA", "TYBCA")} 1' f1.txt > temp.txt && mv temp.txt f1.txt**

28) To run a utility x1 at 11:00 AM.

➔ **echo "x1" | at 11:00**

29) Write a command to find the word 'Unix OS' ignoring case

➔ **grep -i 'Unix OS' yourfile.txt**

30) Write a command to display occurrence of string 'BCA'.

➔ **grep -o 'BCA' yourfile.txt**

31) Write a command to display line which start with '1'.

➔ **grep '^1' yourfile.txt**

32) Write a command to print last six lines of file

➔ **tail -n 6 yourfile.txt**

33) Write a command to display line which end with 'India'. [mimp]

➔ **grep 'India$' yourfile.txt**

November / December – 2018

Write command for following:

34) Replace all occurrences of 'SYBCA' with 'TYBCA' in 8th line of file studlist.

➔ **sed '8s/SYBCA/TYBCA/g' studlist > temp.txt && mv temp.txt studlist**

35) Count all occurrences of 'SYBCA' in the file studlist.

➔ **grep -o 'SYBCA' studlist | wc -l**

36) To count number of characters in last line of file studlist

➔ **tail -n 1 student | wc -m**

37) Display the processes of user 'TYBCA101'.

➔ **ps -u TYBCA101**

38) Count number of character in first five lines of file studlist.

➔ **head -n 5 studlist | awk '{ sum += length } END { print sum }'**

Perform following commands on file 'studlst':

39) Write a command to find the 'Operating System' ignoring case.

➔ **grep -i 'Operating System' yourfile.txt**

40) Write a command to display occurrence of string 'Unix'.

➔ **grep -c 'Unix' yourfile.txt**

41) Write a command to display line which start with 'The'.

➔ **grep '^The' yourfile.txt**

October/november2013 2 marks question

October/november2013

1. **Differentiate between $@ and $*.**

➢ @ is quoted, it preserves the original number and structure of the arguments.
* is quoted, it becomes one long quoted string, regardless of the original number and structure of the arguments.

2. **What are the importance of /tmp directory?**

- tmp directory contains temporary files created by the UNIX. These files automatically get deleted when the system is shut down or restarted.

3. What chmod 615 employee command does?

The chmod 615 employee command sets the permissions of the file or directory named employee to 615. This means that:
The owner of the file or directory (the user who ran the command) has read, write, and execute permissions.
Members of the owner's group have read permissions.
Other users have execute permissions only.
This permission setting is typically used for files or directories that contain scripts or other executable files that should be accessible to the owner and members of the owner's group, but not to other users.
Here is a breakdown of the permission bits in the 615 mode:
Owner:
Read (r): The owner can read the contents of the file or directory.
Write (w): The owner can modify the contents of the file or directory.
Execute (x): The owner can run executable files in the directory.
Group:
Read (r): Members of the owner's group can read the contents of the file or directory.
Other:
Execute (x): Other users can run executable files in the directory.
It is important to note that permissions can be modified by other users if they have the necessary permissions. For example, a user who is a member of the owner's group could modify the permissions of the employee file or directory to allow other users to read or write it.

4. List at least two popular shells including their default prompt

- **Bash Shell**: This is the default shell of many Linux distributions. It is compatible with the Bourne shell and has additional features such as command-line editing, job control, and command history. The default prompt for a non-root user is $.
- **C Shell**: This is a shell that incorporates features from the C programming language, such as aliases and expression syntax. The default prompt for a non-root user is hostname %.

5. What do you mean by command substitution?

- Command substitution is a technique that allows you to use the output of a command as an argument or input for another command. For example, if you want to print the current date and time, you can use the command date to get the output.

6. What is the purpose of ^ in regular expression?

- The caret symbol (^) in regular expression has two main purposes: to match the start of a line or string, and to negate a character set when used inside square brackets.

7. Devise a pipeline sequence which lists five largest files in the current working directory?

- One possible pipeline sequence that can list the five largest files in the current working directory is:
- ls -lS | head -n 6

8. List the characteristics of Unix file system.[imp]

- (i)Hierarchical Structure
- (ii)File Permission
- (iii)CaseSensetive
- (iv) Symbolic Link
- v)Inodes
- vi)Hard Link

9. The booting process of UNIX.

- The booting process of UNIX is the sequence of steps that the operating system performs when it is powered on or restarted.

10. Explain the modes of vi editor. [mimp] (j2l)

- The modes of vi editor are the different ways of interacting with the editor. Each mode has a specific purpose and a set of commands that can be used. The modes of vi editor are: i) command mode , ii) insert mode, iii) last line mode.

11. Explain following commands:

(i)touch :- it is used to create, change and modify the timestamps of a file.
(ii) Uniq :- uniq is the tool that helps to detect the adjacent duplicate lines and also deletes the duplicate lines.
(iii) Wall :- The wall command is a Unix command-line utility that displays a message on the terminals of all logged-in users. The messages can be either typed on the terminal or the contents of a file. wall stands for write all, to send a message only to a specific user use the write command
(iv)Nice :- The option -n specifies the nice value, which is an integer between -20 and 20. The default nice value is 0.
Ex:- nice -n -10 ./programName

Write a command using awk for the following

12. Write awk script to display the user login ids, their home directories and login shells from the "/etc/passwd" file.

- One possible awk script to display the user login ids, their home directories and login shells from the "/etc/passwd" file is:

```
#!/bin/awk -f
BEGIN {
  FS = ":" # set the field separator to :
  print "User\tHome\tShell" # print the header
}
{
  print $1 "\t" $6 "\t" $7 # print the first, sixth and seventh fields
}
```

This script will read the /etc/passwd file line by line and split each line into fields based on the colon delimiter. Then it will print the user name, home directory and login shell for each line, separated by tabs. The output will look something like this:

```
User        Home      Shell
root        /root     /bin/bash
daemon      /usr/sbin           /usr/sbin/nologin
bin /bin    /usr/sbin/nologin
sys /dev    /usr/sbin/nologin
sync        /bin      /bin/sync
games       /usr/games          /usr/sbin/nologin
...
```

13. Switch the first two fields in each line of a text and put the result in a new file.

- One possible way to switch the first two fields in each line of a text and put the result in a new file is to use the awk command.

  To switch the first two fields in each line of a text, you can use the following awk command:

  ```
  awk ' { t = $1; $1 = $2; $2 = t; print; } ' input_file > output_file
  ```

  This command will read the input_file line by line and split each line into fields based on the default field separator, which is whitespace. Then it will swap the values of the first and second fields by using a temporary variable t. Finally, it will print the modified line to the output_file by using the redirection operator >.
  For example, if your input_file contains the following lines:

  ```
  Alice 23
  Bob 19
  Charlie 25
  ```

  The output_file will contain:

  ```
  23 Alice
  19 Bob
  25 Charlie
  ```

14. To only print lines wherein the first field had a numeric value of less than 20.

- Awk is a powerful text processing tool that can manipulate data based on patterns and actions.To print lines with the first field less than 20, you can use the following awk command:

  ```
  awk '$1 < 20 {print}' input_file > output_file
  ```

  This command will read the input_file line by line and split each line into fields based on the default field separator, which is whitespace. Then it will check if the value of the first field is less than 20, and if so, it will print the whole line to the output_file by using the redirection operator >.
  For example, if your input_file contains the following lines:

  ```
  10 Alice
  15 Bob
  20 Charlie
  ```

25 David
30 Eve
The output_file will contain:
10 Alice
15 Bob

October/november2014

1. **What is the exit status of a command? What is its normal value and where is the value** stored? (j2l)

➢ The exit status of a command is a numerical value that indicates whether the command completed successfully or encountered an error. By convention, an exit status of zero means that the command was successful, and a non-zero value means that the command failed. The exit status of the last executed command is stored in a special variable called $?

2. **Differentiate between while and until loop?** (j2l)

➢ The main difference between while and until loop is that they have opposite conditions for executing the loop body. A while loop executes the loop body as long as the condition is true, and stops when it is false. An until loop executes the loop body as long as the condition is false, and stops when it is true.

For example, suppose you have a variable x that is initially 0, and you want to increment it by 1 until it reaches 10. You can use either a while or an until loop to achieve this, but the conditions will be different. Here is how you can write the loops in bash:

3. **What is job? How can we suspend a foreground and background job?**

➢ A job can run either in the foreground or in the background. A foreground job is the one that occupies the current terminal and receives input from the keyboard and output to the screen.

4. **What is the difference between scheduling processes using batch command and using at command?** (j2l)

➢ The at command executes commands at a specified time, while the batch command executes commands when the system load is low

➢ The at command allows you to specify the exact date and time for running a command or a

script, either in absolute or relative terms

5. **What information contain by $? And $# variable?** (j2l)

➢ The ?and# variables are special variables in Unix and Linux that store information about the commands and arguments in a shell script.

The $? variable represents the exit status of the previous command. Exit status is a numerical value returned by every command upon its completion. As a rule, most commands return an exit status of 0 if they were successful, and 1 if they were unsuccessful

6. **What happen when cd command is run without arguments? (j2l)**

- When the cd command is run without arguments, it changes the current working directory to the user's home directory. The home directory is the default directory that the user is placed in after logging into the system. The home directory is usually located under /home/username, where username is the name of the user. The home directory can also be referred to by the tilde (~) character

7. **What does /etc/passwd contains? (j2l)**

- The /etc/passwd file is a text file that contains information about the user accounts on the system. It has one entry per line, and each entry has seven fields separated by colons. The fields are:- Username, Password, User ID (UID), Group ID (GID), Login shell.

8. **What do you mean by a daemon? List out them. How will you kill a daemon? [mimp] (j2l)**

- A daemon is a program that runs in the background and performs some tasks without user intervention. Daemons are usually started at boot time and stopped at shutdown time. Daemons are also known as services in some operating systems.
- Httpd , sshd , crond, syslogd, ntpd.
- To kill a daemon, you can use the kill command with the process ID (PID) of the daemon, or the killall command with the name of the daemon

9. **Explain following commands: [imp]**

   (i)umask :- The umask command in Unix is a command that allows you to set or view the file mode creation mask, which determines the default permissions for newly created files and directories. The umask command works by subtracting the umask value from the maximum permissions for files (666) or directories (777), and applying the result to the new files or directories

   (ii) bc :- The bc command in Unix is a command that allows you to perform arithmetic calculations and execute mathematical expressions in the terminal. It is short for basic calculator, and it supports various features such as arbitrary precision, variables, functions, operators, and conditional statements

   (iii) touch :- The touch command in Unix is a command that allows you to create new files or update the timestamps of existing files. Timestamps are the dates and times that indicate when a file was created, modified, or accessed.

Answer the following using AWK utility

10. **Explain system variables.**

- System variables in awk are predefined variables that store information about the input, output, and processing of the awk program. They can be used to control the behavior of the awk program or to access some useful data

11. **Explain Hash Arrays.**

- Hash arrays are a type of data structure that store key-value pairs in an efficient way. They use a hash function to map each key to an index in an array, where the corresponding value is stored. Hash arrays allow fast access, insertion, and deletion of key-value pairs,

12. explain any four built-in functions.

➢ Some examples of built-in functions in Unix are:

- **echo**: This function prints a message or a variable value to the standard output or a file. It is often used to display information or test scripts. For example, echo "Hello World" will print Hello World on the screen.
- **cd**: This function changes the current working directory to a specified one. It is used to navigate through the file system. For example, cd /home/user will change the directory to /home/user.
- **test**: This function evaluates a conditional expression and returns a true or false value. It is used to perform various checks on files, strings, or numbers. For example, test -f file.txt will return true if file.txt is a regular file, and false otherwise.
- **printf**: This function formats and prints data to the standard output or a file. It is similar to echo, but allows more control over the output format. For example, printf "%s %d\n" "Hello" 42 will print Hello 42 on a new line.

October/november2015

1. **Explain the purpose of super block.** [imp] (j2l)

➢ The purpose of a superblock is to store important information about a file system, such as its type, size, status, and other metadata structures. A superblock is essential for mounting and accessing a file system, as it tells the operating system how to read and write data on it. A superblock also helps to recover a file system in case of corruption or damage, by providing backup copies of itself in different locations

2. **Which command of vi editor is used to display line number before each line?**

➢ The command of vi editor that is used to display line number before each line is set number or set nu. This command can be entered in the command mode by pressing the colon sign and then typing the command. For example, :set number will show the line numbers on the left side of the screen. To turn off the line numbering, you can use the command set nonumber or set nonu.

3. **What is the function of mv command**? [mimp]

➢ The mv command in Linux is used to **move files or directories** from one location to another within a file system. It can also be used to **rename files or directories**.
The syntax for the mv command is as follows:
mv [options (s)] [source_file_name (s)] [Destination_file_name]

4. **What is major and minor numbers?**

➢ In Linux, **major and minor numbers** are used to identify devices such as hard disks, input/output devices, etc
The minor number is used only by the driver specified by the major number; other parts of the kernel don’t use it, and merely pass it along to the driver.

5. **Explain the use of tee command. [imp] (j2l)**

➢ The **tee** command in Linux is used to **read standard input and write it to both standard output and one or more files**. It is named after the T-splitter used in plumbing, which breaks the output of a program so that it can be both displayed and saved in a file .

6. **Define redirection. List out the symbols used for redirection [mimp] (j2l)**

   In Linux, **redirection** is a process of changing the input or output of a command from the default source or destination to another source or destination

Symbol	Description
>	Redirects standard output to a file.
>>	Appends standard output to a file.
<	Redirects standard input from a file.
2>	Redirects standard error to a file.
2>>	Appends standard error to a file.
&>	Redirects both standard output and standard error to a file.
&>>	Appends both standard output and standard error to a file.
<<	Allows input to be taken from the script itself, rather than from an external source.

7. **Explain following commands:**

   (i) date :- The date command in Unix is used to display the current date and time, and also to set the system date and time . By default, the date command displays the date in the time zone on which Unix/Linux operating system is configured.

   (iii) bc :- The bc command in Unix is a command that allows you to perform arithmetic calculations and execute mathematical expressions in the terminal. It is short for basic calculator, and it supports various features such as arbitrary precision, variables, functions, operators, and conditional statements

October/november2016

1. **What does /etc/passwd contains?**

➢ The /etc/passwd file is a text file that contains information about the user accounts on the system. It has one entry per line, and each entry has seven fields separated by colons. The fields are:- Username, Password, User ID (UID), Group ID (GID), Login shell.

2. **What is the use of sleep command?**

➢ The sleep command in Linux is used to **pause the execution of a command** for a specified amount of time . This is useful when you want to delay the execution of a command or script for a certain amount of time.

3. **When it is appropriate to schedule a process using at command?**

➢ The at command in Linux is used to schedule a command or script to run at a specified time in the future . It is appropriate to use the at command when you want to run a command or script at a specific time, such as running a backup script at midnight or sending an email at a later time.

4. **What is ppid? Which command is used to access it?**

- ➢ In Linux, **PPID** stands for **Parent Process ID**.
- ➢ It is the process ID of the parent process that spawned the current process .
  To access the PPID of a process, you can use the ps command in Linux. The following command will display the PPID of a process with a given PID:- ps -o ppid= -p PID

5. **What is shell? Write the functions of shell.** **(j2l)**

- ➢ A **shell** is a command-line interface that allows users to interact with the operating system by typing commands
- ➢ The following are some of the functions of a shell:
  - ✓ **Command Execution:**
  - ✓ **Scripting**
  - ✓ **File Management**
  - ✓ **Input/Output Redirection**
  - ✓ **Piping**
  - ✓ **Environment Variables**

6. **What is the difference between cat f1.txt and cat>f1.txt?**

- ➢ Cat f1.txt is use for run the f1.txt file on terminal
- ➢ Cat>f1.txt is use for insert the data in f1.txt file

7. **Explain following commands**

   i) cut :- The cut command in Unix is used to **extract sections of text** from each line of a file and write the result to standard output

   ii) pr :- The pr command in Unix is used to **prepare a file for printing** by adding suitable footers, headers, and the formatted text

   iii) tr :- The tr command in Unix is a command-line utility for **translating or deleting characters**. It can be used to perform a range of transformations, including converting uppercase to lowercase, squeezing repeating characters, deleting specific characters, and basic find and replace

8. **Explain file comparison using test command with example**

- ➢ The test command in Linux is used to test the validity of a command or expression. It checks whether the command/expression is true or false. It is used to check the type of file and the permissions related to a file. The test command returns 0 as a successful exit status if the command/expression is true, and returns 1 if the command/expression is false .

9. **Explain escape and quote mechanism with an example.**

- ➢ **Escape characters** are used to remove the special meaning of a character in a command or expression. The backslash (\) is the escape character in Linux. Any character immediately following the backslash loses its special meaning, and any letter following the backslash gains its special meaning

10. **Explain links in unix.**

- In Unix, a **link** is a pointer to a file or directory. Links allow more than one file name to refer to the same file, elsewhere. There are two types of links: **hard links** and **symbolic links**.

11. **Display those words whose length are greater than 10 characters and consist of alphabet** only. **[imp]**

- To display words whose length is greater than 10 characters and consist of alphabets only, you can use the grep command. Here is an example command that you can use:

```
grep -E '\\b[a-zA-Z]{11,}\\b' <filename>
```

12. Print odd numbers of words in each line. [imp]

- To print odd numbers of words in each line, you can use the awk command. Here is an example command that you can use:

```
awk '{for(i=1;i<=NF;i+=2) printf("%s ",$i);printf("\n")}' <filename>
```

13. Count occurrences of pattern 'Operating System' in file f1 [imp]

- To count the number of occurrences of the pattern 'Operating System' in file f1, you can use the grep command. Here is an example command that you can use:

```
grep -o 'Operating System' f1 | wc -l
```

November / December – 2016

1. **List out functions of kernel. (j2l)**

- The **kernel** is a central component of an operating system that manages the operations of computer hardware and memory

2. **What is the purpose of /bin directory of Unix file system? [imp] (j2l)**

- The /bin directory in the Unix file system contains essential user binaries (programs) that must be present when the system is mounted in single-user mode . It contains executable files, Linux commands that are used in single-user mode, and common commands that are used by all users, such as cat, cp, cd, and ls . The /bin directory also contains several useful commands that are of use to both the system administrator as well as non-privileged users

3. **Explain -u, -n, -r options of sort. [imp] (j2l)**

- -u: This option is used to remove duplicate lines from the output. If there are multiple lines that are identical, only one copy of the line will be printed .
- -n: This option is used to sort the file numerically. It is used when we want to sort a file with numeric data present inside .
- -r: This option is used to sort the input file in reverse order i.e. descending order by default .

4. **What will be the file and directory permission when the command umask 777 is used? (j2l)**

- File have all permission 1)read 2)write 3)execute.

5. **Explain following commands:**

   i) Is :- The ls command in Unix is used to list the contents of a directory
   ii) Kill :- The kill command in Unix is used to terminate a process manually.
   iii) Write :- The write command in Unix is used to send a message to another user

6. **Explain test command with example**

- The test command in Unix is used to test the validity of a command. It checks whether the command/expression is true or false. It is used to check the type of file and the permissions related to a file. The test command returns 0 as a successful exit status if the command/expression is true, and returns 1 if the command/expression is false . Here is an example of how to use the test command:

  test -f /etc/passwd && echo "File exists"

7. **Explain for loop with an example**

- A **for loop** is a control flow statement that allows code to be executed repeatedly based on a set of conditions. It is used to iterate over a sequence of values, such as an array, list, or range of numbers. Here is an example of how to use the for loop in Unix:
  for i in {1..5}; do echo "Number: $i"; done
  This command will print the numbers 1 through 5 on the terminal window.

8. **Explain sed command.**

- The sed command in Unix is a powerful text editor that can perform various functions on files, such as searching, finding and replacing, insertion, or deletion. The sed command reads a file line by line and applies the specified commands to each line.
- It is used to edit files without opening them, which is much quicker than first opening the file in an editor and then changing it

Write a command using awk utility:

1. Display those words whose length is greater than 5 characters and consist of digits only

Ans:

grep -Eo '\b[[:digit:]]{6,}\b'

This command uses the following options:

- -E: Enables extended regular expressions.
- -o: Prints only the matched part of the line.
- \b: Matches a word boundary.
- [[:digit:]]{6,}: Matches a sequence of 6 or more digits.

Example:

Create a file with some words, including some that are greater than 5 characters and consist of digits only.

cat > words.txt << EOF

```
hello
world
123456
 789012
EOF
```

Display those words whose length is greater than 5 characters and consist of digits only.

grep -Eo '\b[[:digit:]]{6,}\b' words.txt

Output:

```
123456
789012
```

Long question [7 marks]

October/november2013

1) **What does it mean by a link? Give the difference between soft link and hard link? Also write disadvantages when using hard link (pageno : 139)** **[mimp] (j2l)**

Ans:

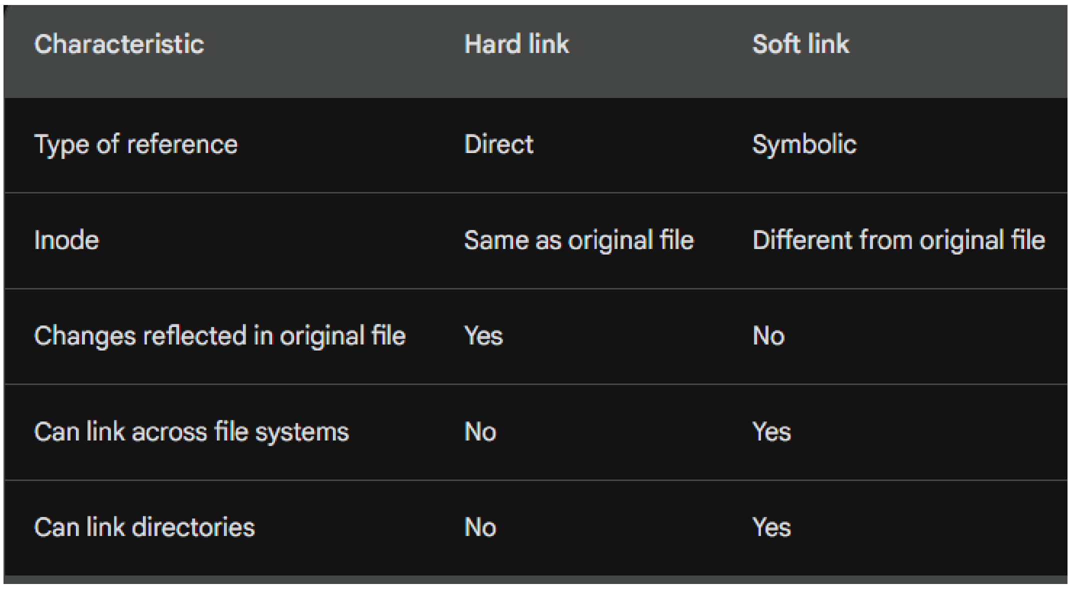
A link in Unix is a special file that points to another file or directory. There are two types of links: hard links and soft links.

Hard link: A hard link is a direct reference to a file. It shares the same inode as the original file, which means that it is essentially the same file with a different name. Any changes made to a hard link are also reflected in the original file.

Soft link:

A soft link is a symbolic reference to a file. It contains the path to the original file, but it does not share the same inode. Any changes made to a soft link are not reflected in the original file.

Differences between hard links and soft links



Characteristic	Hard link	Soft link
Type of reference	Direct	Symbolic
Inode	Same as original file	Different from original file
Changes reflected in original file	Yes	No
Can link across file systems	No	Yes
Can link directories	No	Yes

Disadvantages of using hard links in Unix

- Hard links can only be created between files on the same file system.
- Hard links cannot link directories.
- If the original file is deleted, all hard links to that file will also be broken.
- Hard links can make it difficult to track which file is the original file and which files are hard links

Examples

To create a hard link to a file, use the ln command:

ln file_name hard_link_name

To create a soft link to a file, use the ln -s command:

ln -s file_name soft_link_name

October/november2014

2) Write a note on different role played by shell during the program execution.

Ans:

The shell plays a number of important roles during program execution. These roles include:

- Loading the program into memory: The shell locates the program file on disk and loads it into memory.
- Setting up the program environment: The shell sets up the environment for the program, including variables and other resources.
- Starting the program: The shell starts the program by passing control to the program's entry point.
- Redirecting input and output: The shell can redirect the program's input and output to different files or devices.
- Communicating with the program: The shell can communicate with the program by sending signals and receiving exit codes.

- Terminating the program: The shell terminates the program when it is finished executing.

In addition to these roles, the shell also provides a number of other features that can be used to control program execution, such as:

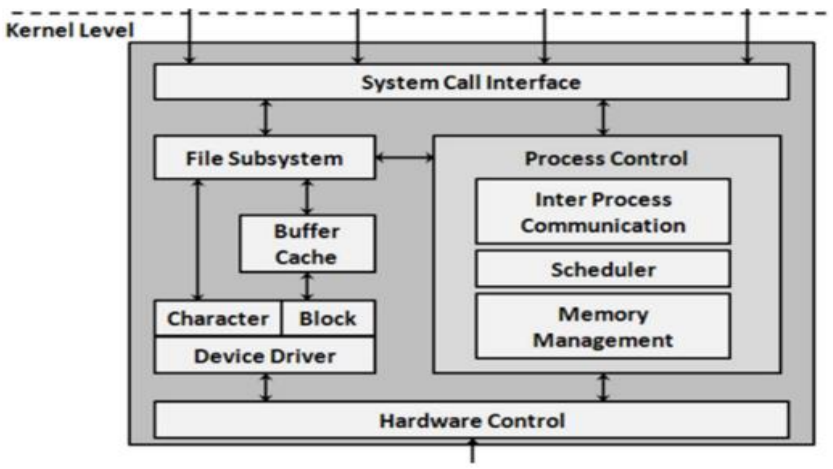
- Command-line arguments: The shell can be used to pass command-line arguments to programs.
- Shell scripts: The shell can be used to write shell scripts, which are programs that can be used to automate tasks.
- Job control: The shell can be used to control multiple running programs, such as starting, stopping, and suspending them.

3) Explain UNIX Kernel structure architecture.

Ans:

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, 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.

4) **What is the purpose of boot block? What it consist of? (j2l)**

Ans:

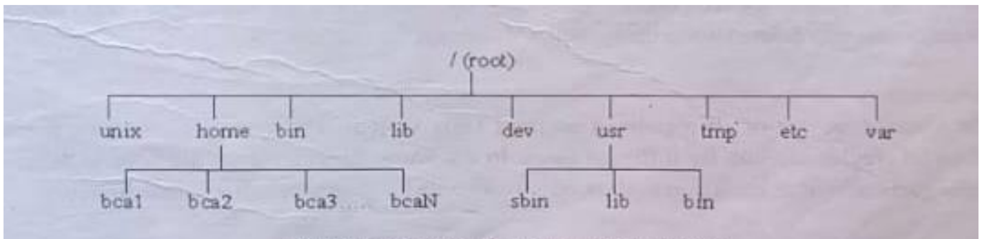
BOOT BLOCK: Boot block (or Block-0) is the first block of the file system. This block is normally reserved for booting process. It is also known as MBR (Master Boot Record). The Boot block contains small boot strapping program. when the system is booted the system BIOS checks for existence of hard disk and loads the entire code of boot block in to the memory. It then handovers control to the boot strapping program. This program loads the kernel in to memory. Though all the files and directories have boot block, the bootstrap program of the boot block of root file system is loaded in to memory. For other file systems, this block is simply kept empty.

October/november2015

5) **Explain the directory structure of Unix. [mimp] (j2l)**

Ans:

A file system has a directory structure containing various components of Unix system. Each file system has its own directory tree headed by root.
The Unix file system has a hierarchical structure which resembles an upsidedown tree. Thus, at first level of file system there is a directory known as root directory.
The root directory is denoted by slash (/). Under the root directory, there are several other sub-directories called bin, lib, usr, etc, tmp and dev. The root directory is a parent of all these directories. Each of these sub-directories contains several files and directories called sub- sub-directories.



[Fig (2.a): directory structure of file system]

- **/ :** 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.
- **/var :** The variable part of the file system. It contains all your jobs, mail queues and incoming mail

6) **Explain booting sequence with init process [mimp] (j2l)**

Ans:

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.

7) **Explain the modes of vi editor** **[mimp]**

October/november2016

1) **Explain features of Kernel.**

   Ans: The kernel is the central part of the Unix operating system. It is responsible for managing the system's resources and providing services to user applications. The kernel runs in a privileged mode, which means that it has full access to all of the system's resources. User applications run in a less privileged mode, and can only access system resources through the kernel.

The main functions of the Unix kernel are:

- **Process management:** The kernel creates and manages processes, which are the basic units of execution in Unix. The kernel is responsible for scheduling processes to run on the CPU, and for allocating memory and other resources to them.
- **Memory management:** The kernel manages the system's memory, allocating it to processes and ensuring that they do not interfere with each other. The kernel also provides services such as caching and virtual memory.

- **File system management:** The kernel manages the system's file system, providing services such as file creation, deletion, and modification. The kernel also ensures that files are accessed in a safe and consistent manner.
- **Device management:** The kernel manages the system's devices, such as the disk, network card, and display. The kernel provides services such as device drivers and interrupt handling.
- **System calls:** The kernel provides system calls, which are interfaces that allow user applications to access system resources and services. For example, a user application might make a system call to read a file from the disk or to send a packet over the network.

In addition to these main functions, the Unix kernel also provides a number of other services, such as security, networking, and accounting.

Here are some specific examples of how the Unix kernel is used:

- When you start a program, the kernel loads it into memory and creates a new process for it.
- When you read a file, the kernel locates the file on disk and transfers the contents of the file to memory.
- When you send a network packet, the kernel breaks the packet up into smaller pieces and sends them over the network interface.
- When you press a key on your keyboard, the kernel generates an interrupt and then calls the appropriate device driver to handle the interrupt.

2) Explain booting process and importance of init process.

Ans:

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.

November / December – 2016

1) Explain features of shell. [mimp] (j2l)

Ans:

- **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.

2) Explain content of I-node block. [imp] (j2l)

Ans:

NODE BLOCK (Index/Information Node block):

In UNIX all the entities are considered to be files. I-Nodes are used to describe the file characteristic & the location of data block, which store the contents of the file. The information related to all these files (not the contents) is stored in an I-node table on the disk. For each file there is an I-node entry in the table. Each entry is made up of 64 byte and contains the relevant details are:

• File Type (ordinary, directory, device file)
• Number of links (No. of alias)

- The numeric user-id of the owner.
- The numeric group-id (GUID - Global User Id) of the owner.
- File mode (access permission)
- Size of file.
- Date and time of last modification of file data.
- Date and time of last access of the file data.
- Date and time of last change of I-node.
- Address of blocks where the file is physically present. I-node is accessed by a number known as I-node number. This number is unique for every file in the file system.

October / November – 2017

3) **Explain content of I- node table. [imp]**

Ans:

The I-node table in Unix is a data structure that stores information about all of the files and directories on a file system. Each inode in the table contains a unique identifier, as well as metadata about the corresponding file or directory, such as:

- File type (regular file, directory, symbolic link, etc.)
- File size
- File permissions (read, write, execute)
- Owner and group IDs
- Access and modification times
- Number of hard links to the file
- Location of the file's data blocks on disk

The I-node table is essential for the proper functioning of the Unix file system. When a user opens a file or directory, the kernel looks up the corresponding inode in the I-node table to get the information it needs to access the file's data.

Here is an example of a simple I-node table:

Inode	**File type**	**File size**	**Owner**	**Group**	**Permissions**	**Data blocks**
1	**Regular file**	**1024 bytes**	**root**	**root**	**644**	**1, 2, 3**
2	**Directory**	**4096 bytes**	**root**	**root**	**755**	**4, 5, 6**
3	**Symbolic link**	**12 bytes**	**root**	**root**	**644**	**/home/user**

In this example, the inode with number 1 represents a regular file called myfile.txt. The inode contains the information that the file is 1024 bytes in size, is owned by the user root, and has the permissions 644 (readable and writable by the owner and group, but not executable by anyone). The file's data is stored in the disk blocks 1, 2, and 3.

4) Explain the features of Unix OS. [mimp]

Ans:

1. **Multiuser Capability:**
   UNIX is a multiuser system. In a multiuser system the same computers resourceshard 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, data and 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. **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.

7. **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.

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.

November / December – 2018

1) Explain the functions of Kernel.

Ans:

The kernel is the central part of the Unix operating system. It is responsible for managing the system's resources and providing services to user applications. The kernel runs in a privileged mode, which means that it has full access to all of the system's resources. User applications run in a less privileged mode, and can only access system resources through the kernel.

The main functions of the Unix kernel are:

- **Process management:** The kernel creates and manages processes, which are the basic units of execution in Unix. The kernel is responsible for scheduling processes to run on the CPU, and for allocating memory and other resources to them.
- **Memory management:** The kernel manages the system's memory, allocating it to processes and ensuring that they do not interfere with each other. The kernel also provides services such as caching and virtual memory.
- **File system management:** The kernel manages the system's file system, providing services such as file creation, deletion, and modification. The kernel also ensures that files are accessed in a safe and consistent manner.
- **Device management:** The kernel manages the system's devices, such as the disk, network card, and display. The kernel provides services such as device drivers and interrupt handling.
- **System calls:** The kernel provides system calls, which are interfaces that allow user applications to access system resources and services. For example, a user application might make a system call to read a file from the disk or to send a packet over the network.

In addition to these main functions, the Unix kernel also provides a number of other services, such as security, networking, and accounting.

Here are some specific examples of how the Unix kernel is used:

- When you start a program, the kernel loads it into memory and creates a new process for it.
- When you read a file, the kernel locates the file on disk and transfers the contents of the file to memory.
- When you send a network packet, the kernel breaks the packet up into smaller pieces and sends them over the network interface.
- When you press a key on your keyboard, the kernel generates an interrupt and then calls the appropriate device driver to handle the interrupt.

8) **Explain command line interpretation of the shell. [mimp] (j2l)**

Ans:

Modes of vi :There are three basic mode of vi editor such as

(i) command mode,

(ii) (ii) insert mode and

(iii) (iii) last-line or ex (execute) mode. Let us understand all the modes one by one and the commands used in each mode.

(i) Command mode

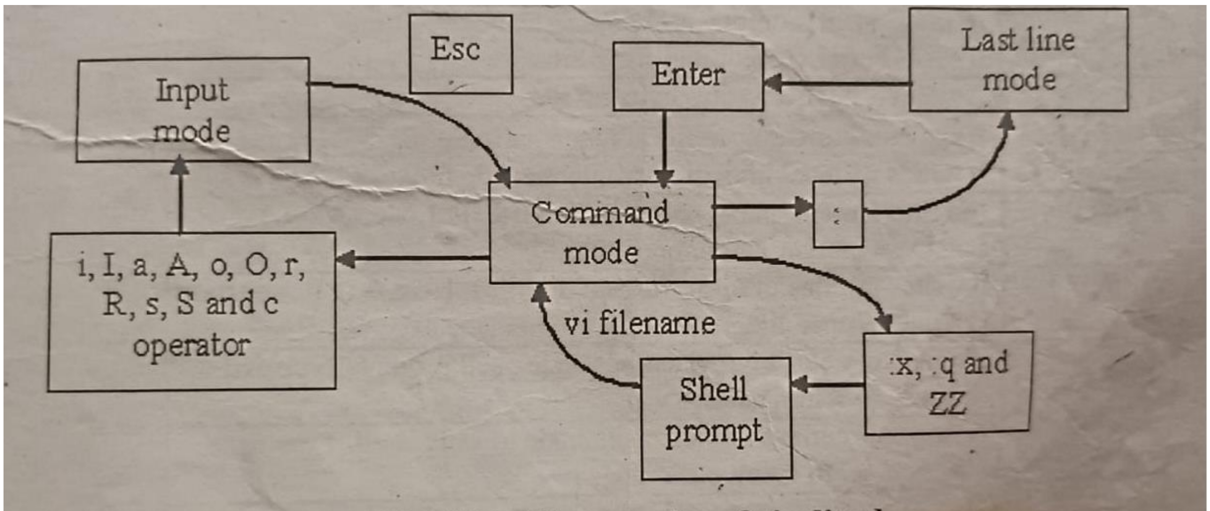
By default, any file opened in vi-editor is in command mode. This is the default mode of vi editor. Most of the letters, or short sequences of letters (without explicitly pressing <enter>) typed by user in this mode will be interpreted as commands. When a user is in command mode and press Esc-key then the internal speaker of a terminal will beep.

(ii) Insert mode

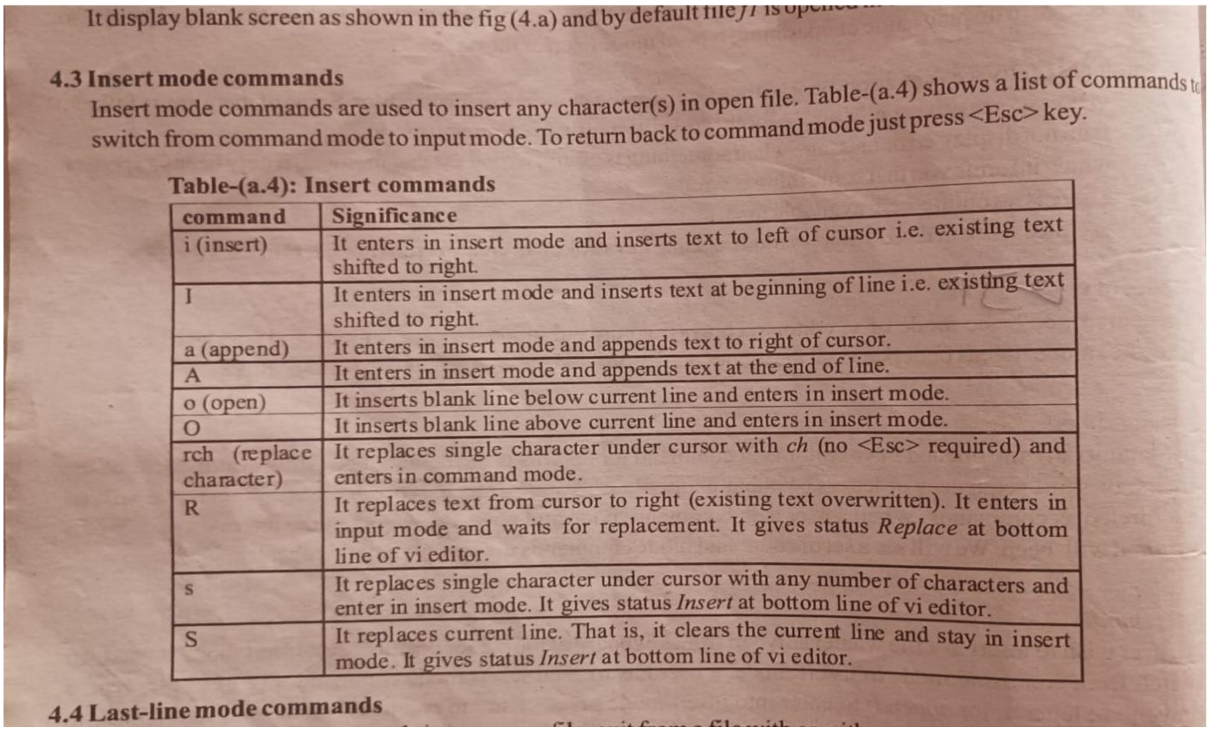
If a user wish to insert in a file whatever typed from keyboard at the cursor position then insert mode is used. To switch from command mode to insert mode, a user has to used insert mode command. For example, type a (lowercase letter a, for append) to enter into insert mode press <Esc> to end insert mode, and return back to command mode.

(iii) Last Line mode (ex mode)

Last line mode is used for line oriented commands. In command mode, if user type a colon (:) then the cursor moves to the bottom line of the screen, with a colon prompt. Type a line mode command and then press <enter> Each time you use a line mode command, you must type a colon to enter in line mode, then type the command at the colon prompt and then press <enter> when you finish typing the command.



[Fig (4.b): Three modes of vi editor]



It display blank screen as shown in the fig (4.a) and by default file *f1* is

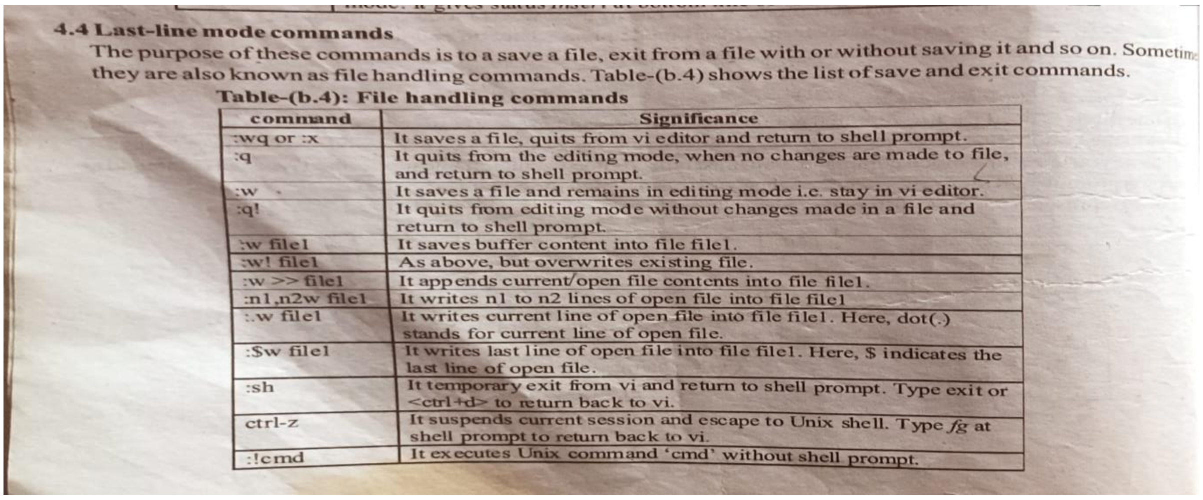
4.3 Insert mode commands

Insert mode commands are used to insert any character(s) in open file. Table-(a.4) shows a list of commands to switch from command mode to input mode. To return back to command mode just press <Esc> key.

Table-(a.4): Insert commands

command	Significance
i (insert)	It enters in insert mode and inserts text to left of cursor i.e. existing text shifted to right.
I	It enters in insert mode and inserts text at beginning of line i.e. existing text shifted to right.
a (append)	It enters in insert mode and appends text to right of cursor.
A	It enters in insert mode and appends text at the end of line.
o (open)	It inserts blank line below current line and enters in insert mode.
O	It inserts blank line above current line and enters in insert mode.
rch (replace character)	It replaces single character under cursor with *ch* (no <Esc> required) and enters in command mode.
R	It replaces text from cursor to right (existing text overwritten). It enters in input mode and waits for replacement. It gives status *Replace* at bottom line of vi editor.
s	It replaces single character under cursor with any number of characters and enter in insert mode. It gives status *Insert* at bottom line of vi editor.
S	It replaces current line. That is, it clears the current line and stay in insert mode. It gives status *Insert* at bottom line of vi editor.

4.4 Last-line mode commands



4.4 Last-line mode commands

The purpose of these commands is to a save a file, exit from a file with or without saving it and so on. Sometim they are also known as file handling commands. Table-(b.4) shows the list of save and exit commands.

Table-(b.4): File handling commands

command	Significance
:wq or :x	It saves a file, quits from vi editor and return to shell prompt.
:q	It quits from the editing mode, when no changes are made to file, and return to shell prompt.
:w	It saves a file and remains in editing mode i.e. stay in vi editor.
:q!	It quits from editing mode without changes made in a file and return to shell prompt.
:w file1	It saves buffer content into file file1.
:w! file1	As above, but overwrites existing file.
:w >> file1	It appends current/open file contents into file file1.
:n1,n2w file1	It writes n1 to n2 lines of open file into file file1
:.w file1	It writes current line of open file into file file1. Here, dot(.) stands for current line of open file.
:$w file1	It writes last line of open file into file file1. Here, $ indicates the last line of open file.
:sh	It temporary exit from vi and return to shell prompt. Type exit or <ctrl+d> to return back to vi.
ctrl-z	It suspends current session and escape to Unix shell. Type *fg* at shell prompt to return back to vi.
:!cmd	It executes Unix command 'cmd' without shell prompt.

[5marks]

October/november2013

1. Discuss Relative and Absolute File Access Permissions with example **[mimp]**

Ans:

File Access Permissions

In Unix, all files and directories have permission. There are three file level and directory level permission: read write and execute. Symbolically, there are represented as follow:

Read permission :r

Write permission :W

Execute permission :X

In UNIX, File permissions are assigned numerical octal values from 0 to 7. Individually, however they have the following octal values:

Read permission :4

Write permission:2

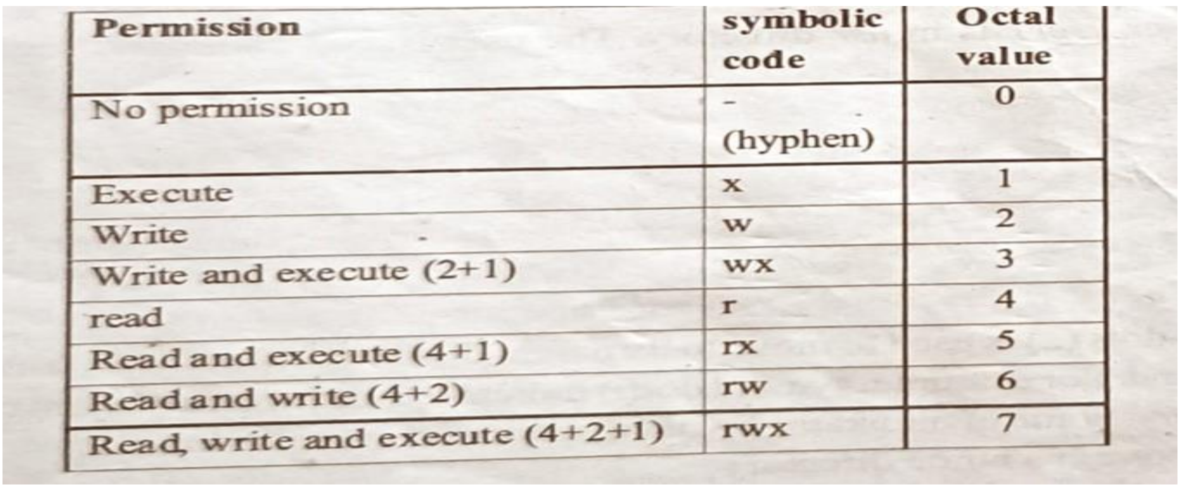
Execute permission :1

Thus, for a file that has a permission field like-rwxrwxrwx (symbolic notation), the permissions are said to be read, write and execute for the owner, group and others. In octal notation, it would be written as 777.

Similarly, a file with permissions-rw-rw-rw- is said to have permission read and write to owner, group user and. other user. In octal notation, it would be written as 666.

A zero (0) value in the permission field is treated as a complete removal of the read, write and execute permission for all (i.e. owner, group and other users).

Table-(a.3) shows permission in symbolic form and its octal value.



Permission	symbolic code	Octal value
No permission	- (hyphen)	0
Execute	x	1
Write	w	2
Write and execute (2+1)	wx	3
read	r	4
Read and execute (4+1)	rx	5
Read and write (4+2)	rw	6
Read, write and execute (4+2+1)	rwx	7

2. Write a script to perform mathematical operations using menu

ANs:

```
# switch case program
```

```
echo "enter the first number"
read f
echo "Enter the second number"
read s
echo "Menu"
echo "1. Addition"
echo "2. Subraction"
echo "3. Multiplication"
echo "4. Division"
echo "enter your choice"
read choice

case $choice in
1)
ans=$(($f + $s))
;;
2)
ans=$(($f - $s))
;;
3)
ans=$(($f * $s))
;;
4)
ans=$(($f / $s))
;;
esac
echo "Your answer is: "
echo $ans
```

3. Write a shell script to check whether the given file is empty or not. **[imp]**

Ans:

```
#!/bin/bash

# Get the filename from the user
echo "Enter the filename to check: "
read filename

# Check if the file exists
if [[ ! -e "$filename" ]]; then
echo "The file does not exist."
exit 1
fi

# Check if the file is empty
if [[ -s "$filename" ]]; then
echo "The file is not empty."
else
```

```
echo "The file is empty."
fi
```

4. **Write note on ls command with options and example.[mimp] (j2l)**

Ans:

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

5.Explain any two communication commands(j2l)

Ans:

two common communication commands in Unix are:

- wall
- write

Wall:

wall allows you to send a message to all users who are currently logged into the system. This is a useful command for system administrators to send out announcements or warnings to all users.

To use the wall command, simply type the following command and then enter your message:

```
wall <message>
```

Write:

write allows you to send a private message to another user who is currently logged into the system.

To use the write command, simply type the following command followed by the username of the user you want to send the message **to:**

```
write <username> <message>
```

October/november2014

1. **write note on grep utility. [mimp] (j2l)**

Ans:

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.

2. **Write a shell script to reverse a number**. (j2l)

Ans:

```
# Get the number to reverse from the user
echo "Enter a number to reverse: "
read number

# Initialize the reversed number variable
```

```
reversed_number=0

# While the number is greater than zero, keep reversing it
while [[ $number -gt 0 ]]; do
  # Get the remainder of the number when divided by 10
  remainder=$((number % 10))

  # Add the remainder to the reversed number variable
  reversed_number=$((reversed_number * 10 + remainder))

  # Divide the number by 10 to get the next digit
  number=$((number / 10))
done

# Print the reversed number
echo "The reversed number is: $reversed_number"
```

3. **Write a shell script to find smallest of three numbers that are read from keyboard**

Ans:

```
# Read the three numbers from the keyboard
echo "Enter the first number: "
read num1
echo "Enter the second number: "
read num2
echo "Enter the third number: "
read num3

# Find the smallest number
smallest_number=$num1
if [[ $num2 -lt $smallest_number ]]; then
  smallest_number=$num2
fi
if [[ $num3 -lt $smallest_number ]]; then
  smallest_number=$num3
fi

# Print the smallest number
echo "The smallest number is: $smallest_number"
```

4. **Explain find command.[imp] (j2l)**

Ans:

October/november2015

5. **Write a shell script to check the entered no is palindrome or not.**

Ans:

the find command in Unix is a powerful tool for searching for files and directories. It can be used to search based on a variety of criteria, including filename, file size, modification date, and permissions.

The basic syntax for the find command is as follows:

find <starting_directory> <options> <expression>

- **<starting_directory>** is the directory where the search should begin. If no starting directory is specified, the current directory will be used.
- **<options>** are used to modify the search behavior. Some common options include:
-
    - **-name <filename>:** Search for files with the specified filename.
    - **-type <file_type>:** Search for files of the specified type, such as regular files, directories, or symbolic links.
    - -**size <file_size>:** Search for files with the specified size, in bytes.
    - **-mtime <number>:** Search for files that were modified within the specified number of days.
    -
- **<expression>** is a Boolean expression that is used to filter the search results. For example, the expression -name "*.txt" would search for all text files in the current directory.

The find command is a very versatile tool, and it can be used for a variety of tasks, such as:

- Finding lost files
- Cleaning up unwanted files
- Managing permissions on files and directories
- Automating tasks related to files and directories

Here are some more examples of how to use the find command:

```
# Find all files with the extension `.txt` in the current directory
find . -name "*.txt"

# Find all files that were modified within the past 24 hours
find . -mtime -1

# Find all files that are larger than 100MB
find . -size +100M

# Find all directories in the current directory and its subdirectories
find . -type d
```

6. **Write a shell script to test that the file is a readable file or not**. **(j2l)**

Ans:

```
# Get the filename from the user
```

```
echo "Enter the filename to test: "
read filename

# Check if the file exists
if [[ ! -e "$filename" ]]; then
  echo "The file does not exist."
  exit 1
fi

# Check if the file is readable
if [[ -r "$filename" ]]; then
  echo "The file is readable."
else
  echo "The file is not readable."
Fi
```

7. **Explain Insert, Append and change command of sed [mimp](j2l)**

Ans:

The insert, append, and change commands in sed are used to insert, append, and change text in a file.

Insert

The insert command inserts the specified text before the current line in the pattern space. The pattern space is the line of text that sed is currently processing.

To use the insert command, you simply type the following command:

```
i text
```

where text is the text that you want to insert.

Append

The append command appends the specified text to the end of the current line in the pattern space.

To use the append command, you simply type the following command:

```
a text
```

where text is the text that you want to append.

Change

The change command replaces the entire current line in the pattern space with the specified text.

To use the change command, you simply type the following command:

```
c text
```

where text is the text that you want to replace the current line with.

Here are some examples of how to use the insert, append, and change commands:

# Insert the line "This is a new line." before the first line of the file "myfile.txt"

```
sed '1i This is a new line.' myfile.txt
```

# Append the line "This is another new line." to the end of the file "myfile.txt"

```
sed 'a This is another new line.' myfile.txt
```

Change the first line of the file "myfile.txt" to "This is the changed line."

```
sed '1c This is the changed line.' myfile.txt
```

The insert, append, and change commands are very useful for editing text files. They can be used to insert, append, and change text on a single line or on multiple lines.
It is important to note that the insert, append, and change commands can only be used on the current line in the pattern space. To edit multiple lines, you can use the -n option to prevent sed from printing the output. You can then use the p command to print the modified lines.

```
sed -n '1i This is a new line.
a This is another new line.
p' myfile.txt
```

8. **Explain any two communications between users' commands in unix.**

Ans:

two common communications between users' commands in Unix are:

- Piping
- Redirection

Piping:

Piping allows you to connect the output of one command to the input of another command. This is useful for chaining together multiple commands to perform a complex task.
To pipe the output of one command to the input of another command, you use the pipe operator (|). For example, the following command will pipe the output of the ls command to the input of the grep command:

```
ls | grep "txt$"
```

Redirection:

Redirection allows you to redirect the input or output of a command to a file. This is useful for saving the output of a command for later use or for providing input to a command from a file.
To redirect the input of a command to a file, you use the less-than (<) operator. For example, the following command will redirect the input of the sort command to the file input.txt:

```
sort < input.txt
```

9. **Write note on pattern matching feature of awk utility giving example.**

October/november2016

1. **Write a shell script to enter the records in a file from keyboard and display the file. Make proper validation**.

Ans:

```
# Create a file to store the records
touch records.txt

# Prompt the user to enter the records
echo "Enter the records:"

# Loop until the user enters a blank line
while read line; do

  # Validate the record
  if [[ -z "$line" ]]; then
    break
  fi

  # Validate the record format
  if [[ ! "$line" =~ /^[a-zA-Z0-9]+\s+[a-zA-Z0-9]+$/ ]]; then
    echo "Invalid record format. Please enter the record in the
following format: name surname"
    continue
  fi

  # Append the record to the file
  echo "$line" >> records.txt

done

# Display the file contents
echo "The records are:"
cat records.txt
```

2. **Write a shell script to test the length of string is zero or not. [imp]**

Ans:

```
# Get the string from the user
echo "Enter the string: "
read string

# Test the length of the string
if [[ ${#string} -eq 0 ]]; then
  echo "The string is empty."
else
  echo "The string is not empty."
Fi
```

3. **Explain break and continue statement with example**

Ans:

Break statement

The break statement is used to exit a loop. It can be used in any type of loop, including for loops, while loops, and until loops.

When the break statement is executed, the loop is exited and control is passed to the next statement in the program.

Example:

```
                  for i in 1 2 3 4 5; do
            if [[ $i -eq 3 ]]; then
             break
            fi
            echo "$i"
           done
```

Output:

```
1
2
```

In the above example, the break statement is used to exit the for loop when the value of i is equal to 3. As a result, the loop only iterates twice and the numbers 1 and 2 are printed to the console.

Continue statement

The continue statement is used to skip the remaining statements in a loop iteration and start the next iteration. It can be used in any type of loop, including for loops, while loops, and until loops.

When the continue statement is executed, the current iteration of the loop is skipped and control is passed to the beginning of the loop for the next iteration**.**

Example:

```
for i in 1 2 3 4 5; do
  if [[ $i -eq 3 ]]; then
    continue
  fi
  echo "$i"
done
```

in the above example, the continue statement is used to skip the iteration of the loop where the value of i is equal to 3. As a result, the number 3 is not printed to the console.

November / December – 2016

1. **Write a shell script that accepts two decimal numbers from keyboard and display their sum in octal form. Make proper validation.** **[imp] (j2l)**

Ans:

```
# Prompt the user to enter the two numbers
echo "Enter the first number: "
read num1
```

```
echo "Enter the second number: "
read num2

# Validate the numbers
if [[ ! "$num1" =~ /^[0-9]+$/ ]]; then
  echo "Invalid number: $num1"
  exit 1
fi

if [[ ! "$num2" =~ /^[0-9]+$/ ]]; then
  echo "Invalid number: $num2"
  exit 1
fi

# Calculate the sum of the two numbers
sum=$(($num1 + $num2))

# Convert the sum to octal
octal_sum=$(printf "%o" $sum)

# Display the sum in octal form
echo "The sum of $num1 and $num2 in octal form is: $octal_sum"
```

2. Write a shell script to test the file is an executable file or not[mimp] (j2l)

ANS:

```
#Get the file name from the user
echo "Enter the file name:"
read file_name

# Check if the file exists
if [ ! -f "$file_name" ]; then
  echo "File does not exist."
  exit 1
fi
# Check if the file is executable
if [ -x "$file_name" ]; then
  echo "File is executable."
else
  echo "File is not executable."
fi
```

3. Explain command for changing file permissions and ownership. (j2l)

Ans:

The two commands for changing file permissions and ownership in Unix are:

- **chmod (change mode)**
- **chown (change owner)**

chmod (change mode)

The chmod command is used to change the permissions of a file or directory. Permissions are divided into three categories:

- Read (r): The ability to read the contents of the file or directory.
- Write (w): The ability to write to the file or directory.
- Execute (x): The ability to execute the file or directory.
-

Each of these permissions can be assigned to three different types of users:

- User (u): The owner of the file or directory.
- Group (g): The group that the file or directory belongs to.
- Others (o): All other users.

The chmod command uses a symbolic notation to specify the permissions that should be assigned to a file or directory. The following table shows the symbols that can be used to specify permissions:

Symbol	Permission
r	Read
w	Write
x	`Execute
+	Add the specified permission
-	Remove the specified permission
=	Set the specified permission to the specified value

To change the permissions of a file or directory using the chmod command, you must specify the following:

- The path to the file or directory.
- The permissions that you want to assign.
- The type of user (u, g, or o) that you want to assign the permissions to.

For example, the following command would assign the read, write, and execute permissions to the user and the group for the file my_file.txt:

```
chmod u+rwx,g+rwx my_file.txt
```

chown (change owner)

The chown command is used to change the ownership of a file or directory. To change the ownership of a file or directory using the chown command, you must specify the following:

- The path to the file or directory.
- The user or group that you want to assign ownership to.

For example, the following command would change the ownership of the file my_file.txt to the user john:

```
chown john my_file.txt
```

4. **explain at and batch command.** **[mimp] (j2l)**

Ans:

The at and batch commands in Unix are used to schedule commands to run at a later time. The at command is used to schedule commands to run once, while the batch command is used to schedule commands to run periodically.

at command

The at command can be used to schedule a command to run at a specific time.

The syntax for the at command is as follows:

```
at [time]
```

where time is the time at which you want the command to run. The time can be specified in absolute or relative terms. For example, to schedule a command to run at 10:00 AM tomorrow, you would use the following command:

```
at tomorrow 10:00 AM
```

To schedule a command to run in 5 minutes, you would use the following command:

```
at now + 5 minutes
```

batch command

The batch command can be used to schedule a command to run periodically. The syntax for the batch command is as follows:

```
batch [interval] [command]
```

where interval is the interval at which you want the command to run and command is the command that you want to run. The interval can be specified in seconds, minutes, hours, or days. For example, to schedule a command to run every 10 minutes, you would use the following command:

```
batch 10 minutes my_command.sh
```

October / November – 2017

1. **Write a script to enter records in a studlist.txt file. The fields are Rollno, Name and Marks.**

Ans:

```
# Create the studlist.txt file if it doesn't exist
touch studlist.txt

# Prompt the user to enter the records
echo "Enter the records:"

# Read the records from the user
read -p "Rollno: " rollno
read -p "Name: " name
read -p "Marks: " marks

# Append the record to the studlist.txt file
```

```
echo "$rollno $name $marks" >> studlist.txt

# Repeat the process until the user enters a blank line
while read -p "Continue (y/n)? " cont; do
  if [ "$cont" != "y" ]; then
    break
  fi

  echo "Enter the next record:"
  read -p "Rollno: " rollno
  read -p "Name: " name
  read -p "Marks: " marks

  echo "$rollno $name $marks" >> studlist.txt
done

echo "The records have been saved to the file studlist.txt."
```

2. **write a shell script to check whether entered file exists or not. [imp] (j2l)**

Ans:

```
# Ask the user to enter a file name
echo "Enter a file name:"
read file_name

# Check if the file exists
if [ -f "$file_name" ]; then
   echo "File exists."
else
   echo "File does not exist."
Fi
```

November / December – 2018

1. **Which command is useful to change the file permission? Explain it giving by example. [imp]**

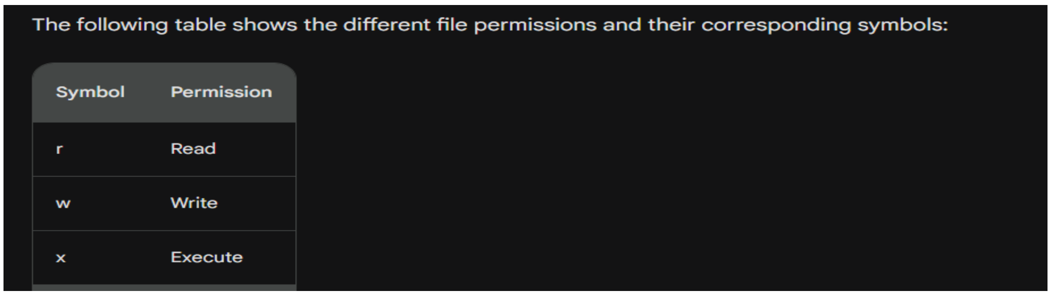
Ans:

The chmod command is used to change the file permissions in Unix. The file permissions determine who has access to a file and what they can do with it.

Example:

To change the permissions of the file myfile.txt so that the owner has read, write, and execute permissions, and the group and others have read permissions, you would use the following command

```
chmod 755 myfile.txt
```


The following table shows the different file permissions and their corresponding symbols:

Symbol	Permission
r	Read
w	Write
x	Execute

2. **Write a shell script to validate the name of a person accepted through the 5 keyboard so that it does not exceed 10 chars of length.**

Ans:

```
# Prompt the user to enter the name
echo "Enter the name: "
read name

# Validate the name length
if [[ ${#name} -gt 10 ]]; then
  echo "The name must not exceed 10 characters in length."
  exit 1
fi

# The name is valid
echo "The name is valid."
```

1. **Explain expr command giving example for string manipulation.** **[mimp] (j2l)**

Ans:
The expr command is a built-in shell command in Unix and Linux that is used to evaluate expressions. It can be used for both arithmetic and string manipulation.
Examples for string manipulation:

- Get the length of a string:
  expr length "string"

- Extract a substring from a string:
  expr substr "string" 2 3
- Match a pattern in a string:
  expr match "string" "str"
- Replace a substring in a string:
  expr replace "string" "str" "new"

# Get the length of the string

```
str="Hello, world!"
str_len=$(expr length "$str")

# Extract the first 5 characters from the string
first_5_chars=$(expr substr "$str" 1 5)

# Match the pattern "world" in the string
world_match=$(expr match "$str" "world")

# Replace the substring "world" with "universe" in the string
replaced_str=$(expr replace "$str" "world" "universe")

# Print the results
echo "String length: $str_len"
echo "First 5 characters: $first_5_chars"
echo "World match: $world_match"
echo "Replaced string: $replaced_str"
```

output

```
String length: 13
First 5 characters: Hello
World match: world
Replaced string: Hello, universe!
```

❖ **Explain pipes in unix with example(j2l)**

Ans:

Pipes in Unix are a way to connect the output of one command to the input of another command. This allows you to chain together multiple commands to perform complex tasks without having to save the output of each command to a temporary file.

To create a pipe, you use the | character between two commands. The output of the first command will be sent to the input of the second command. You can pipe together as many commands as you want.

Here is an example of using a pipe:

```
ls -l | sort –r
```

This command will list all of the files in the current directory in long format, and then sort them in reverse order. The ls command is the first command in the pipe, and the sort command is the second command. The output of the ls command is piped to the input of the sort command.

Another example:

```
grep 'hello' file.txt | wc –l
```

This command will count the number of lines in the file file.txt that contain the word "hello". The grep command is the first command in the pipe, and the wc command is the second command. The output of the grep command is piped to the input of the wc command.

- **Explain sed command and its sub command(j2l)**

Ans:

The sed command is a stream editor that can be used to perform various text transformations on files. It is a very powerful tool, but it can also be a bit complex to learn.
sed works by reading a file line by line and applying a series of commands to each line. The commands can be used to search for and replace text, delete lines, insert lines, and more.
sed subcommands are used to perform specific operations on the pattern space, which is the line of text that is currently being processed. Some of the most common sed subcommands include

- `s`: Substitute text.
- `a`: Append text to the end of the line.
- `i`: Insert text before the line.
- `d`: Delete the line.
- `p`: Print the line.

sed can also be used to perform regular expressions, which are patterns that can be used to match complex sequences of characters.

Here is an example of a simple sed command:

```
sed 's/hello/world/' file.txt
```

sed can be used to perform a wide variety of tasks, such as:

- Searching for and replacing text in files.
- Removing unwanted lines from files.
- Formatting text files.
- Extracting data from text files.
- Converting text files to other formats.

- **Explain positional parameter of the shell(j2l)**

Ans:

Positional parameters are special variables in the Unix shell that contain the arguments that are passed to the shell script when it is invoked. The positional parameters are numbered from 0 to 9, with $0 containing the name of the script itself and $1 containing the first argument, and so on.
Positional parameters can be used to access the arguments that are passed to a shell script from the command line. This can be useful for writing shell scripts that can be customized by the user.
For example, the following shell script will print the first argument that is passed to it:

```
echo "The first argument is: $1"
```

- **Explain features of unix(j2l)**

Ans:

Here are some of the key features of Unix:

- **Multitasking:** Unix can run multiple programs at the same time. This allows users to be more productive and to get more done.
- **Multiuser:** Unix can support multiple users at the same time. This makes it ideal for use in servers and other multi-user systems.
- **Hierarchical file system:** Unix uses a hierarchical file system to organize files and directories. This makes it easy to find and manage files.
- **Powerful command-line interface**: Unix has a powerful command-line interface (CLI) that allows users to interact with the system and to perform a wide variety of tasks.
- **Portability**: Unix is portable, meaning that it can be run on a variety of different hardware platforms. This makes it a good choice for developing software that needs to be portable.
- **Open source:** Many versions of Unix are open source, meaning that the source code is freely available for anyone to modify and redistribute. This makes it easy to customize Unix to meet the specific needs of users and organizations.

In addition to these key features, Unix also offers a wide range of other features, such as networking support, security features, and a variety of built-in tools and utilities.
Here are some of the benefits of using Unix:

- **Reliability and stability:** Unix is known for its reliability and stability. It is a good choice for systems that need to be up and running 24/7.
- **Security:** Unix has a number of security features that make it difficult for unauthorized users to access the system.
- **Flexibility and customization:** Unix is a very flexible and customizable operating system. It can be easily customized to meet the specific needs of users and organizations.
- **Performance:** Unix is known for its good performance. It is a good choice for systems that need to be able to handle a high workload.

❖ **Explain while loop with example(j2l)**

Ans:

A while loop in Unix is a control flow statement that allows you to execute a block of code repeatedly until a certain condition is met. While loops are very useful for tasks such as reading files line by line, processing data in batches, and performing iterative operations.

The syntax for a while loop in Unix is as follows:

```
while <condition>; do
  <commands>
done
```

The condition is a Boolean expression that is evaluated before each iteration of the loop. If the condition evaluates to true, the commands block is executed. If the condition evaluates to false, the loop terminates.

Here is an example of a while loop in Unix:

```
# Read the contents of a file line by line.
while read line; do
  echo $line
done < file.txt
```