UNIT-IV

1. Write a note on who command.

1. This command is used to display the users who are logged on the system currently.

General format is,

$ who

Example:

```
$ who
Vinay     tty1     Sep  28   10:12
Anand     tty2     Sep  28   10:18
Sarthak   tty7     Sep  28   11:06
```

The first column of the output represents the user names. The second column represents the corresponding terminal names and the remaining columns represents the time at which the users logged on.

2. Give the syntax and meaning of mkdir command of linux.

2. mkdir

This mkdir (make directory) command is used to make (create) new directories.

General format is,

mkdir [-p] <directory_narnel> <directory_narne2>

The option *-p* is used to create consequences of directories using a single *mkdir* command.

Example:

$ mkdir vinay

This command will create a directory named 'vinay'

3. What are the uses of mv and cp commands?

3. This mv (move) command is used to rename the specified files /directories.

General format is,

mv <source> <destination>

Note that to make move, the user must have both write and execute permissions on the <source>.

The cp (copy) command is used to copy the content of one file into another. If the destination is an existing file, the file is overwritten; if the destination is an existing directory, the file is copied into that directory.

General format is,

cp <source-file> <destination-file>

4. List the logical operators and their meaning in linux.

4.

Logical operator	Meaning
!	NOT operation
-a	AND operation
-o	OR operation

5. Give the meaning and syntax of grep command.

5.

grep

This grep (global search for regular expression) command is used to search for a spec
fied pattern from a specified file and display those lines containing the pattern.

General format is,

```
grep [-options] pattern <filename>
```

(Quote the pattern if the pattern contains Shell special characters.)

where *options* can be,

b	Ignores spaces, tabs
i	Ignores case distinction for matching (do not differentiate capital and sma case letters)
v	Displays only the lines that do not match the specified pattern
c	Displays the total number of occurrences of the pattern in the file
n	Displays the resultant lines along with their line numbers
<number>	Displays the matching lines along with <number> of lines above and b low

6. List the input and output redirection operators of linux.

6. Input redirection operator:

< or <0

Syntax: Command < filename

Ouput redirection operator:

> or > 1

Syntax: Command > filename

7. Explain ls command in linux.

7. ls

This command is used to list the content of the specified directory.
General format is,

ls [-options] <directory_name>

where *options* can be,

a Lists all directory entries including the hidden files
l Lists the files in long format.
r Lists the files in the reverse order
t Lists the files sorted by the last modification time
R Recursively lists all the files and sub-directories as well as the files in the sub-directories.
p Puts a slash after each directory
s Displays the number of storage blocks used by a file.
x Lists contents by lines instead of by columns in sorted order
F Marks executable files with * (

8. Explain logical operators in linux.

8. Logical operators:

!	**NOT operation; It is a unary operator that negates (true to false, false to true) the result of the specified expression**
-a	**AND operation; Returns true if both the expressions are true, else returns false.**
-o	**OR operation; Returns false if both the expressions are false, else returns true.**

Assume ,

a=20 and b=40
AND operation
i.[$a –lt 30 –a $b –gt 50]
is false

Program

```
a=20
b=40
if [ $a –lt 30 –a $b –gt 50 ]
then
echo “you are correct”
else
echo “you are wrong”
fi
```

OR operation

ii. [$a –lt 30 –o $b –gt 50]
is true

9. Explain the pwd command.

9. This pwd (print working directory) command displays the full pathname for the current working directory.

General format is,

$ pwd

Example

```
$ pwd
/home/bmi
```

Your present working directory is */home/bmi*.

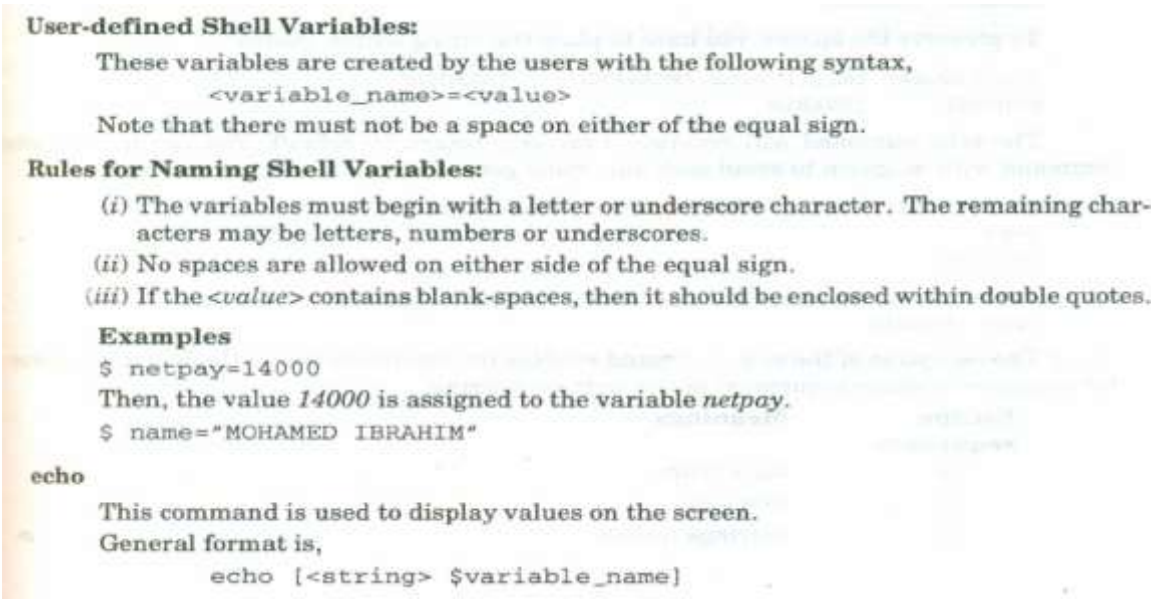
10. List the string operators and their meaning in linux.

10. String operators are =(equal to), !=(not equal to), -z(zero length) and –n(non zero length)

`string1=string2`	Compares two strings; returns true if both are equal, else returns false
`string1!=string2`	Compares two strings; returns true if both are not equal, else returns false
`-z string`	Checks whether the specified string is of zero length or not; returns true if the string is *null*, else returns false
`-n string`	Checks whether the specified string is of non-zero length or not; returns true if the string is not *null*, else returns false

11. How do you use user defined variables in linux?

11.



User-defined Shell Variables:

These variables are created by the users with the following syntax,

```
<variable_name>=<value>
```

Note that there must not be a space on either of the equal sign.

Rules for Naming Shell Variables:

(*i*) The variables must begin with a letter or underscore character. The remaining characters may be letters, numbers or underscores.

(*ii*) No spaces are allowed on either side of the equal sign.

(*iii*) If the *<value>* contains blank-spaces, then it should be enclosed within double quotes.

Examples

```
$ netpay=14000
```

Then, the value *14000* is assigned to the variable *netpay*.

```
$ name="MOHAMED IBRAHIM"
```

echo

This command is used to display values on the screen.

General format is,

```
echo [<string> $variable_name]
```

12. Give the difference between mv and cp command.

12.

mv	**cp**
1. This mv (move) command is used to rename the specified files /directories.	1. The cp (copy) command is used to copy the content of one file into another.
2. General format is, mv <source> <destination>	2. General format is, cp <source-file> <destination-file>

13. Explain the difference between mv and rm in linux.

13. The mv (move) command is used to rename the specified files /directories.

mv	**rm**
1. This mv (move) command is used to rename the specified files /directories.	1. This rm (remove) command is used to remove (delete) a file from the specified directory.
2. General format is, mv <source> <destination>	2. General format is, rm <filename>
3. To make move, the user must have both write and execute permissions on the <source>.	3. To remove a file, you must have *write* permission for the directory that contains the file.

14. Write a short note on vi editor.

14. The vi editor(visual full screen editor) created by Bill Joy at the University of California at Berkely. This editor can be invoked by typing vi at the $ prompt. The syntax is,

vi [<filename>]

VI MODES

The vi editor works on *three modes* as follows:

INSERT MODE:

- The text should be entered in this mode. And any key press in this mode is treated as text.
- We can enter into this mode from command mode by pressing any of the keys:

  `i, I, a, A, o, O, r, R, s, S.`

COMMAND MODE:

- It is the default mode when we start up vi editor.
- All the commands on vi editor (cursor movement, text manipulation, etc.) should be used in this mode.

- We can enter into this mode from *Insert mode* by pressing the *[Esc]* key, and from *Ex mode* by pressing *[Enter]* key.

EX MODE:

- The ex mode commands (saving files, find, replace, etc.,) can be entered at the last line of the screen in this mode.
- We can enter into this mode from command mode by pressing *[:]* key.

Note that one cannot enter to *ex mode* directly from *input mode* and vice versa.

The following are some of the commands that should be used in *command mode*.

15. Define a kernel.

15. Kernel is the heart of the UNIX system. It is loaded into memory whenever the system is booted. It interacts with the actual hardware in machine language. It manages the resources such as files, memory, processor etc. and it also does the functions such as keep tracking of a program in execution, Allocating the processor time to each process.

16. What is the purpose of break command in linux.

16. This command is used to exit the enclosing loop (for, while, until) or case command. The optional parameter n is an integer value that represents the number of levels to break when the loop commands are nested. This

method is very convenient to exit a deeply nested looping structure when some type of error has occurred.

General format is,

```
break [n]
```

where **n** represents the number of levels to break.

Long questions

1. Explain the looping constructs in linux operating system with examples.

1.

ITERATIVE STATEMENTS

These statements are used to execute a sequence of commands repeatedly based on some conditions.

while loop:

General format is,

```
while <conitional_command>
do
  <commands>
done
```

The *<conditional_command>* is executed for each cycle of the loop, if it returns a *zero* exit status (success), then the commands between *do* and *done* are executed. This process continues until the *<conditional_command>* yields a *non-zero* exit status (failure).

Example

```
a=1
while [ $a -le 5 ]
do
        echo "value of a=" $a
        a='expr $a + 1'
done
```

until loop

General format is,

```
until <conditional_command>
do
 <commands>
done
```

This loop is similar to the *while* loop except that it continues as long as the <contional_command> fails (returns a non-zero exit status).

Example:

```
a=1
until [ $a -ge 3 ]
do
        echo "value of a=" $a
        a='expr $a + 1'
done
```

for loop

General format is,

```
for <variable_name> in <list_of_values>
do
 <commands>
done
```

The *<variable_name>* has a value from *<list_of_values>* in each cycle of the loop. And the loop will be executed until the *<list_of_values>* becomes empty.

Examples

```
(i) for i in 1 2 3 4 5
    do
     echo $i
    done
```

The output of this Shell script will be,

```
1
2
3
4
5
```

2. Write a note on positional parameters.

2.

Positional Parameters:

Most of the Linux commands accept arguments at the command line.

For example,

```
$cp source target
```

Here, *source* and *target* are command line arguments that are sent for the '*cp*' command. Like this, we can also send arguments (*positional parameters*) to a Shell scripts at the command line while it is executed. The argument values can be utilized in the Shell scripts. If an argument has blank spaces, then double quote it. There are some special Shell variables carrying values regarding positional parameters and command execution. The are given below,

$0	Refers to the name of the command (Shell script name)
$1	Refers to the first argument
$2	Refer to the second argument and so on.
$*	Refers all the arguments (positional parameters)
$#	Refers the total number of arguments
$?	Returns the exit status of the last executed command (If a command is executed successfully without giving any error message, then exit status of that command is *zero*; else the exit status of that command is a *non-zero* number.)
$!	Returns the Process IDentification number (PID) of the last background command (command ended with &)
$$	Returns the Process IDentification number (PID) of the current Shell

3. Write a shell program to accept an integer, find the sum of digits and reverse it.

3. Accept an integer and find its reverse. Also check for palindrome.

```
echo "enter the number"
read n
num=$n
rev=0
while [ $n -ne 0 ]
do
r=`expr $n % 10`
rev=`expr $rev \* 10 + $r`
n=`expr $n / 10`
done
echo "the reverse is" $rev
if [ $num -eq $rev ]
then
echo "number is palindrome"
else
echo "number is not palindrome"
fi
```

4. Give the syntax of if statement and explain with an example.

4.

CONTROL STATEMENTS

Shell provides some control statements to execute the commands based on certain condition.

if

(format 1)

```
if (conditional_command)
then
   <commands>
fi
```

Example:

```
a=10
b=20
if [ $a == $b ]
    then   echo "a is equal to b“
fi
```

if

(format 2)

```
if (conditional_command)
then
   <commands>
else
   <commands>
fi
```

Example:

```
a=10
b=20
if [ $a == $b ]
        then
         echo "a is equal to b"
else
  echo "a is not equal to b"
fi
```

if

(format 3)

```
if (conditional_command)
then
   <commands>
elif (conditional_command)
then
   <commands>
```

```
.
.
.
else
    <commands>
fi
```

```
a=10
b=20
if [ $a == $b ]
    then
    echo "a is equal to b“
            elif [ $a -gt $b ]
    then
            echo "a is greater than b“
            elif [ $a -lt $b ]
    then
            echo "a is less than b“
    else
    echo "No condition is satisfied“
fi
```

5. Explain different versions of cat command with examples.

5. i) cat command is used to display the contents of the specified file

$ cat filename

Eg. cat college

It will display the contents of file named college

ii) cat command can be used with redirection operator(>) to create new files.

$ cat > filename
<type the text>
^d (press ctrl+d key to end)

Eg. $ cat > vinay

Hello dear students
Take care and do study

Wear mask everywhere
^d
This will create a file named vinay.

iii) cat command can append the contents of one file to the end of another file.

$ cat file1 >> file2

Eg. cat hello >> hi

This will append the contents of hello file to the end of hi file

6. Write a note on shift command.

6. Shift

The Bourne Shell allows positional parameters $1 through $9 in the Shell scripts at a time. If the command line contains more than nine arguments, the "shift" command can be used to handle the other parameters. This command is used to shift the positional parameters one step forward. As the result, the second argument ($2) becomes the first ($1), and then $3 becomes$2 and so on. In every shift, the value of the first argument is discarded.

$ set welcome to you all students
$ echo $1 $2 $3 $4 $5
welcome to you all students
$ shift
$ echo $1 $2 $3 $4 $5
to you all students
$ shift
$ echo $1 $2 $3 $4 $5
you all students

7. Explain the following commands in linux operating system with an example.
i) cut ii) ls iii) chmod

7. **i) cut**

This command is used to cut the columns / fields of a specified file (Like the head and tail commands cut the lines - rows).

General format is,

```
cut [-options] <filename>
```

where options can be,

c <columns>	Cuts the columns specified in *<columns>*. You must separate the *column numbers* by using commas
f <fields>	Cuts the fields specified in *<fields>*. You must separate field numbers by using commas

ii) ls

This command is used to list the content of the specified directory.

General format is,

ls [-options] <directory_name>

where *options* can be,

a Lists all directory entries including the hidden files

l Lists the files in long format (filenames along with file type, file permissions, number of links, owner of the file, file size, file creation/modification time, number of links for a file). The number oflinks for a file refers more than one name for a file, does not mean that there are more copies of that file.

This *ls -l* option displays also the year only when the file was last modified more than a year back. Otherwise, it only displays the date without year.

r Lists the files in the reverse order

t Lists the files sorted by the last modification time

R Recursively lists all the files and sub-directories as well as the files in the sub-directories.

p Puts a slash after each directory

s Displays the number of storage blocks used by a file.

x Lists contents by lines instead of by columns in sorted order

F Marks executable files with * (i.e the file having executable permission to the user) and directories with /

iii) chmod

The chmod (change mode) command is used to change the file permissions for an existing file. We can use anyone of the following notations to change file permissions.

- Symbolic notation
- Octal notation

Symbolic mode:

General format is,

```
chmod user_symbols set/deny_symbol access_symbols <filename(s)>
```

where *user_symbols* can be,

u	User
g	User group
o	Others

where *set / deny_ symbol* can be,

+	Assign the permissions
-	Remove the permissions
=	Assign absolute permission

where *access_symbols* can be,

r	Readable
w	Writeable
x	Executable

Examples

```
$ chmod u+x file1
```

This command adds *execute* permission to the user for executing the file – *file1*.

```
$chmod g+x file1
```

This command assigns *execute* permission to *usergroup* to execute the file – *file1* in addition to the existing permissions of that file. Note that the file holds its old permissions other than the changed *execute* permission i.e., the already existing permissions are not removed by default

8. Differentiate between cp and mv commands in linux.

8. Question no 12 in part A

9. Explain any three directory related commands in linux.

9. i). rmdir

This rmdir (remove directory) command is used to remove (delete) the specified directories.
A directory should be empty before removing it.

General format is,

rmdir [-p] <directory_narnel> <directory_narne2>

The option -p is used to remove consequences of directories using a single *rmdir*command.

Example

```
$ rmdir ibr
```

This command will remove the directory *ibr*, which is the subdirectory of the current directory.

```
$ rmdir ibr/ib/i
```

This command will remove the directory *i* only.

```
$ rmdir -p ibr/ib/i
```

This command will remove the directories *i*, *ib* and *ibr* consequently.

ii). cd

This cd (change directory) command is used to change the current working directory to a specified directory.

General format is,

cd <directory_name>

Examples

```
$ cd  /home/ibr
```

Then, the directory */home/ibr* becomes as the current working directory.

```
$ cd ..
```

This command lets you bring the parent directory as current directory. Here, .. represents the parent directory.

iii). pwd

This pwd (print working directory) command displays the full pathname for the current working directory.

General format is,

pwd

Example

```
$ pwd
/home/bmi
```

Your present working directory is */home/bmi*.

(Note: Any three of your choice)

10. Write a shell script to print first 'n' Fibonacci numbers.

10. Accept 'n' and display the first 'n' Fibonacci numbers.

```
echo "enter the number"
read n
f1=0
f2=1
echo "the fibonacci numbers are"
echo $f1
echo $f2
i=2
while [ $i -lt $n ]
do
f3=`expr $f1 + $f2`
echo $f3
f1=$f2
f2=$f3
i=`expr $i + 1`
done
```

11. Give the syntax and explain the case statement with an example.

11.

case statement:

This is a multi-branch control statement.

General format is,

```
case value in
pattern1)  <commands> ;;
pattern2) <commands> ;;
.
.
.
patternN) <commands> ;;
esac
```

This statement compares the *value* to the *patterns* from the top to bottom, and performs the commands associated with the matching pattern. The commands for each pattern must be terminated by double semicolon.

Example

```
echo "1. Directory listing          ;  d. date "
echo "p. print working directory    ;  q. quit"
echo "Enter your choice"
read choice
```

Example

```
echo "1. Directory listing          ;  d. date "
echo "p. print working directory    ;  q. quit"
echo "Enter your choice"
read choice
```

12. Explain chmod command with an example.

12. chmod

The chmod (change mode) command is used to change the file permissions for an existing file. We can use anyone of the following notations to change file permissions.

- Symbolic notation
- Octal notation

Symbolic mode:

General format is,

```
chmod user_symbols set/deny_symbol access_symbols <filename(s)>
```

where *user_symbols* can be,

u	User
g	User group
o	Others

where *set / deny_ symbol* can be,

+	Assign the permissions
-	Remove the permissions
=	Assign absolute permission

where *access_symbols* can be,

r	Readable
w	Writeable
x	Executable

Examples

```
$ chmod u+x file1
```

This command adds *execute* permission to the user for executing the file – *file1*.

```
$chmod g+x file1
```

This command assigns *execute* permission to *usergroup* to execute the file – *file1* in addition to the existing permissions of that file. Note that the file holds its old permissions other than the changed *execute* permission i.e., the already existing permissions are not removed by default

Octal notation:

This mode uses a three-digit number to change the file permissions. In this number, the first digit represents *user* permissions, the second digit represents *usergroup* permissions and the third digit represents *others* permissions.

General format is,

```
chmod three_digit_number <filename1> [<filename2> ...]
```

Digits and their meanings,

0	No permissions
4	Read
2	Write
1	Execute

We can also sum the numbers for mixing of permissions.

3	Write and Execute
5	Read and Execute
6	Read and Write
7	Read, Write and Execute

13. Explain the following commands with example.
    i) sort ii) grep

13 i)

1. sort

This command sorts the contents of a given file based on ASCII values of characters.
General format is,

```
sort [options] <filename>
```

where *options* can be,

-m <filelist>	Merges sorted files specified in <filelist>
-o <filename>	Stores output in the specified <filename>
-r	Sorts the content in reverse order (reverse alphabetical order)
-u	Removes duplicate lines and display sorted content
-n	Numeric sort
-t"char"	Uses the specified "*char*" as delimiter to identify fields
-c	Checks if the file is sorted or not
+pos	Starts sort after skipping the posth field

2. grep

This grep (global search for regular expression) command is used to search for a specified pattern from a specified file and display those lines containing the pattern.
General format is,

```
grep [-options] pattern <filename>
```

(Quote the pattern if the pattern contains Shell special characters.)
where *options* can be,

b	Ignores spaces, tabs
i	Ignores case distinction for matching (do not differentiate capital and small case letters)
v	Displays only the lines that do not match the specified pattern
c	Displays the total number of occurrences of the pattern in the file
n	Displays the resultant lines along with their line numbers
<number>	Displays the matching lines along with <number> of lines above and below

14. Write a note on vi editor.

14. Question no 14 Part A

15. Explain the use of " "(double quotes) and \ (back slash) in linux shell script commands,with an example.

15. " "(double quotes):

The double quotes turn off the special meaning of all the enclosed characters(including #, *, ?, &, |, :, (,), [,], ^) expect $, ` and \.
Example:

```
$ name=vinay
$ echo "your name is $name"
your  name is vinay
```

\ (back slash) :
The back slash turn off the special meaning of a single character that immediately follows it.
Example:
$ name=vinay
$ echo “your name is \$name”
your name is $name

16. Write a shell script to accept a number and print whether it is prime.

16. Accept ‘n’ and check whether it is a prime or not.

```
echo "enter the value of n"
read n
if [ $n -eq 1 ]
then
echo "the number is neither prime or unprime"
else
p=1
i=2
while [ $i -lt $n ]
        do
        r=`expr $n % $i`
                if [ $r -eq 0 ]
                then
                p=0
                break
                fi
        i=`expr $i + 1`
        done
if [ $p -eq 1 ]
then
echo "number is prime"
else
echo "number is not prime"
fi
fi
```

17. Explain the relational operators used in linux operating system.

17.

Numerical Comparison:

General format is,

```
test number1 operator number2
```

where the *operator* can be,

-eq	Returns true if number1 is equal to number2; else returns false
-ne	Returns true if number1 is not equal to number2; else returns false
-gt	Returns true if number1 is greater than number2; else returns false
-lt	Returns true if the number1 is less than number2; else returns false
-ge	Returns true if the number1 is greater than or equal to number2; else returns false
-le	Returns true if number1 is less than or equal to number2; else returns false

18. Explain all directory and file permissions with an example.

18. File Access Permission

Linux treats everything as files. There are three types of files in Linux as follows,

- Ordinary file
- Directory file
- Special file (Device file)

There are three types of modes for accessing these files as follows,

- Read mode (r)
- Write mode (w)
- Execute mode (x)

Linux separates its users into three groups for security and convenience as follows,

- User (u)
- User group *(g)*
- Others (o)

The file permission is displayed in 10 characters width as follows:

BIT POSITION:	1	234	567	8910
MEANING:	file or	permission	permission	permission

directory to users to usergroup to others

r	Readable
w	Writeable
x	Executable
-	Denial of permission

If a file has "**-rwxrw-r--**" as file permission, then the file is not a directory and it can be readable, writeable & executable for its *owner* (user), readable & writeable for its *usergroup* and only readable for *others*.