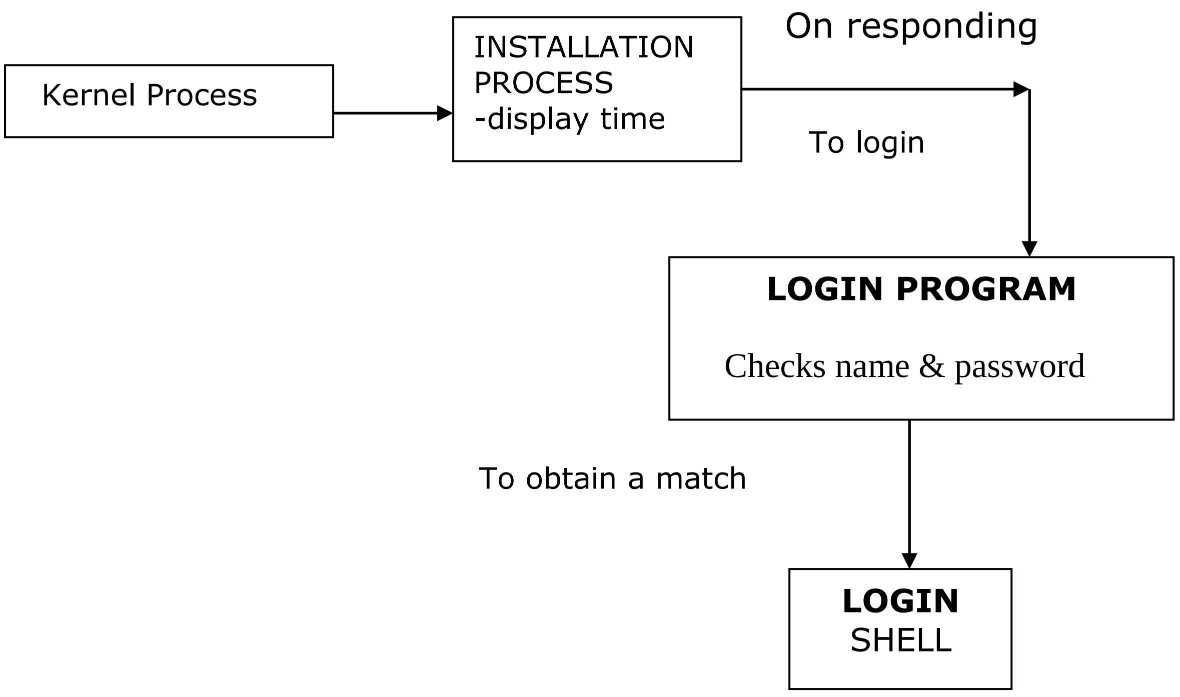
Q)Discuss the login process in UNIX?

On switching in a UNIX environment, the *Kernel* program is first program that is loaded in the computer's memory. From the Hard-disk , this program triggers a chain of initialization processes at the end of which the login prompt appears on the screen.

On entering a login name, the login program is activated, login points the password prompt on he screen and turn off character echoing , so that the password entered is not displayed on the terminal. Login compares the password entered with the /etc/passwd file. If they match, a program named in the same file replaces the login program. The program named is usually BOURNE SHELL or the C SHELL and is called login shell.



LOGIN PROCESS IN A UNIX SYSTEM

Unix Commands

alias COMMAND:

alias command allows you to create a shortcut to a command. As the name indicates, you can set alias name for the commands/paths which is too longer to remember.

SYNTAX:

The Syntax is

```
alias [options] [ AliasName [ =String ] ]
```

OPTIONS:

-a	Removes all alias definitions from the current shell execution environment.
-p	Prints the list of aliases in the form alias name=value on standard output.

EXAMPLE:

1. To create a shortcut temporarily:

   alias lhost='cd /var/www/html'

   This command will set lhost to cd /var/www/html/.
   Now if you type lhost it will take you to the specified folder/directory.
2. To create a shortcut Permanently:
   You can put your aliases into the /home/user/.bashrc file. It is good to add them at the end of the file.
   alias home='cd /var/www/html/hscripts/linux-commands'
   Now if you type home it will take you to the specified folder/directory.
3. To create a shortcut for a command:
   alias c='clear'
   This command will set c to clear.
   Now if you type c it will clear the screen.

banner COMMAND :

SYNTAX:

```
banner <argument>
```

It displays the argument in large banner form on the screen. It does not work in Linux.

Bc COMMAND:

bc command is used for command line calculator. It is similar to basic calculator. By using which we can do basic mathematical calculations.

SYNTAX:

The Syntax is

```
bc [options]
```

OPTIONS:

-c	Compile only. The output is dc commands that are sent to the standard output.
-l	Define the math functions and initialize scale to 20, instead of the default zero.
filename	Name of the file that contains the basic calculator commands to be calculated this is not a necessary command.

bg COMMAND:

bg command is used to place a job in background. User can run a job in the background by adding a "&" symbol at end of the command.

SYNTAX:

The Syntax is

bg [options] [job]

OPTIONS:

-l	Report the process group ID and working directory of the jobs.
-p	Report only the process group ID of the jobs.
-x	Replace any job_id found in command or arguments with the corresponding process group ID, then execute command passing it arguments.
job	Specifies the job that want to run in the background.

cal COMMAND:

cal command is used to display the calendar.

SYNTAX:

The Syntax is

cal [options] [month] [year]

OPTIONS:

-1	Displays single month as output.
-3	Displays prev/current/next month output.
-s	Displays sunday as the first day of the week.
-m	Displays Monday as the first day of the week.
-j	Displays Julian dates (days one-based, numbered from January 1).
-y	Displays a calendar for the current year.

cat COMMAND:

cat linux command concatenates files and print it on the standard output.

SYNTAX:

The Syntax is

cat [OPTIONS] [FILE]...

OPTIONS:

-A	Show all.
-b	Omits line numbers for blank space in the output.
-e	A $ character will be printed at the end of each line prior to a new line.
-E	Displays a $ (dollar sign) at the end of each line.
-n	Line numbers for all the output lines.

-s	If the output has multiple empty lines it replaces it with one empty line.
-T	Displays the tab characters in the output.
-v	Non-printing characters (with the exception of tabs, new-lines and form-feeds) are printed visibly.

cc COMMAND (cc means c compiler) :

it compiles the c program and creates a binary or output file normally its output file is a.out. This can be changed with the help of –o option with cc command.

Syntax cc <c filename>

cd COMMAND:

cd command is used to change the directory.

SYNTAX:

The Syntax is

cd [directory | ~ | ./ | ../ | -]

OPTIONS:

-L	Use the physical directory structure.
-P	Forces symbolic links.

chgrp COMMAND:

chgrp command is used to change the group of the file or directory. This is an admin command. Root user only can change the group of the file or directory.

SYNTAX:

The Syntax is

chgrp [options] newgroup filename/directoryname

OPTIONS:

-R	Change the permission on files that are in the subdirectories of the directory that you are currently in.
-c	Change the permission for each file.
-f	Force. Do not report errors.

EXAMPLE:

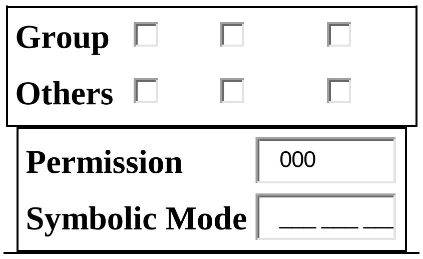
1. chgrp hiox test.txt (The group of 'test.txt' file is root, Change to newgroup hiox.)

chmod COMMAND:

chmod command allows you to alter / Change access rights to files and directories.

File Permission is given for users,group and others as,

	Read	Write	Execute
User	☐	☐	☐



Group	☐	☐	☐
Others	☐	☐	☐

Permission	000
Symbolic Mode	___ ___ ___

SYNTAX:
The Syntax is
chmod [options] [MODE] FileName

File Permission

#	File Permission
0	none
1	execute only
2	write only
3	write and execute
4	read only
5	read and execute
6	read and write
7	set all permissions

OPTIONS:

-c	Displays names of only those files whose permissions are being changed
-f	Suppress most error messages
-R	Change files and directories recursively
-v	Output version information and exit.

EXAMPLE:

1. To make a file readable and writable by the group and others.

   chmod 066 file1.txt

2. To allow everyone to read, write, and execute the file

   chmod 777 file1.txt

<u>chown COMMAND</u>:

chown command is used to change the owner / user of the file or directory. This is an admin command, root user only can change the owner of a file or directory.

SYNTAX:
The Syntax is
chown [options] newowner filename/directoryname

OPTIONS:

-R	Change the permission on files that are in the subdirectories of the directory that you are currently in.
-c	Change the permission for each file.

-f	Prevents chown from displaying error messages when it is unable to change the ownership of a file.

EXAMPLE:

1. chown hiox test.txt

   The owner of the 'test.txt' file is root, Change to new user hiox.

2. chown -R hiox test

   The owner of the 'test' directory is root, With -R option the files and subdirectories user also gets changed.

clear COMMAND:

This command clears the terminal screen.

SYNTAX:

The Syntax is

```
clear
```

cmp COMMAND:

cmp linux command compares two files and tells you which line numbers are different.

SYNTAX:

The Syntax is

```
cmp [options..] file1 file2
```

OPTIONS:

- c	Output differing bytes as characters.
- l	Print the byte number (decimal) and the differing byte values (octal) for each difference.
- s	Prints nothing for differing files, return exit status only.

EXAMPLE:

Compare two files:

```
cmp file1 file2
```

The above cmp command compares file1.php with file2.php and results as follows.

```
file1.php file2.php differ: byte 35, line 3
```

cp COMMAND:

cp command copy files from one location to another. If the destination is an existing file, then the file is overwritten; if the destination is an existing directory, the file is copied into the directory (the directory is not overwritten).

SYNTAX:

The Syntax is

```
cp [OPTIONS]... SOURCE DEST
cp [OPTIONS]... SOURCE... DIRECTORY
cp [OPTIONS]... --target-directory=DIRECTORY SOURCE...
```

OPTIONS:

-a	same as -dpR.
--backup[=CONTROL]	make a backup of each existing destination file
-b	like --backup but does not accept an argument.
-f	if an existing destination file cannot be opened, remove it and try again.
-p	same as --preserve=mode,ownership,timestamps.
--preserve[=ATTR_LIST]	preserve the specified attributes (default: mode,ownership,timestamps) and security contexts, if possible additional attributes: links, all.
--no-preserve=ATTR_LIST	don't preserve the specified attribute.

--parents	append source path to DIRECTORY.

EXAMPLE:

1. Copy two files:

   cp file1 file2

   The above cp command copies the content of file1.php to file2.php.

chfn command - change your finger information

SYNOPSIS

chfn [-f full-name] [-o office] [-p office-phone] [-h home-phone] [-u] [-v] [username]

DESCRIPTION

chfn is used to change your finger information. This information is stored in the */etc/passwd* file, and is displayed by the **finger** program. The Linux **finger** command will display four pieces of information that can be changed by **chfn** : your real name, your work room and phone, and your home phone.

COMMAND LINE

Any of the four pieces of information can be specified on the command line. If no information is given on the command line, **chfn** enters interactive mode.

INTERACTIVE MODE

In interactive mode, **chfn** will prompt for each field. At a prompt, you can enter the new information, or just press return to leave the field unchanged. Enter the keyword "none" to make the field blank.

OPTIONS

-f, --full-name
Specify your real name.
-o, --office
Specify your office room number.
-p, --office-phone
Specify your office phone number.
-h, --home-phone
Specify your home phone number.
-u, --help
Print a usage message and exit.
-v, --version
Print version information and exit.

date COMMAND:

date command prints the date and time.

SYNTAX:

The Syntax is

```
date  [options] [+format] [date]
```

OPTIONS:

-a	Slowly adjust the time by sss.fff seconds (fff represents fractions of a second). This adjustment can be positive or negative.Only system admin/ super user can adjust the time.
-s date-string	Sets the time and date to the value specfied in the datestring. The datestr may contain the month names, timezones, 'am', 'pm', etc.
-u	Display (or set) the date in Greenwich Mean Time (GMT-universal time).

Format:

%a	Abbreviated weekday(Tue).
%A	Full weekday(Tuesday).
%b	Abbreviated month name(Jan).
%B	Full month name(January).
%c	Country-specific date and time format..
%D	Date in the format %m/%d/%y.
%j	Julian day of year (001-366).
%n	Insert a new line.
%p	String to indicate a.m. or p.m.
%T	Time in the format %H:%M:%S.
%t	Tab space.
%V	Week number in year (01-52); start week on Monday.

EXAMPLE:
date command

date (The above command will print Wed Jul 23 10:52:34 IST 2008)

df COMMAND:

df command is used to report how much free disk space is available for each mount you have. The first column show the name of the disk partition as it appears in the /dev directory. Subsequent columns show total space, blocks allocated and blocks available.

SYNTAX:
The Syntax is
df [options]

OPTIONS:

-a	Include dummy file systems.
-h	Print sizes in human readable format.(e.g., 1K 234M 2G)
-H	Print sizes in human readable format but use powers of 1000 not 1024.
-i	List inode information instead of block usage.
-l	Limit listing to local file systems.
-P	Use the POSIX output format.
-T	Print file system type.

EXAMPLE:

1. df

Output:

```
Filesystem           1K-blocks      Used Available Use% Mounted on
/dev/mapper/VolGroup00-LogVol00
                     150263916  14440324 128067408  11% /
/dev/sda1               101086     10896     84971  12% /boot
tmpfs                   253336         0    253336   0% /dev/shm
```

In the above output:
/dev/mapper/VolGroup00-LogVol00 -> Specifies FileSystem.
/dev/sda1 -> Specifies FileSystem.
tmpfs -> Specifies FileSystem.

Prints default format.

du COMMAND:

du command is used to report how much disk space a file or directory occupies.

SYNTAX:
The Syntax is du [options] directories

OPTIONS:

-a	Displays the usage of space that each file is taking up.
-k	Write the files size in units of 1024 bytes, rather than the default 512-byte units.
-s	Instead of the default output, report only the total sum for each of the specified files.
-L	Process symbolic links by using the file or directory which the symbolic link references, rather than the link itself.
-x	When evaluating file sizes, evaluate only those files that have the same device as the file specified by the file operand.

EXAMPLE:

1. du -a images

Output:

```
12     images/daisy.jpg
20     images/flo.gif
76     images/CHILD.gif
12     images/indigo.gif
152    images/flower.gif
12     images/sunflower.jpg
12     images/tulip-flower-clipart5.gif
12     images/flower.jpg
8      images/thumbnail.aspx
8      images/baby.jpg
12     images/woodpecker.gif
168    images/baby.gif
8      images/thumbnail.jpg
1012   images/house.bmp
12     images/peacock.gif
1544   images
```

Displays the size of each file in the specified directory.

diff COMMAND:

diff command is used to find differences between two files.

SYNTAX:
The Syntax is
diff [options..] from-file to-file

dir COMMAND:

Like DOS it works in Linux environment showing the list of directories and files in a sorted manner in the current directory

Display COMMAND:

display [*options* **...]** *file* **[***options***...]***file*

DESCRIPTION

Display is a machine architecture independent image processing and display program. It can display an image on any workstation screen running an X server. **Display** can read and write **many** of the more popular image formats (e.g. **JPEG**, **TIFF**, **PNM**, **Photo CD**, etc.).

With **display**, you can perform these functions on an image:
load an image from a file
display a sequence of images as a slide show
write the image to a file
delete the image file
copy & paste a region of the image
resize the image

echo COMMAND:

echo command prints the given input string to standard output.

SYNTAX:

The Syntax is

echo [options..] [string]

OPTIONS:

-n	do not output the trailing newline
-e	enable interpretation of the backslash-escaped characters listed below
-E	disable interpretation of those sequences in STRINGs

EXAMPLE:

1. echo command

   ```
   echo "hscripts Hiox India"
   ```

   The above command will print as hscripts Hiox India

exit COMMAND :

Allows you to exit from a program, shell or log you out of a Unix network.

Syntax

exit

Examples

exit - If supported would exit you from the program, shell or log you out of network.

fg COMMAND:

fg command is used to place a job in foreground.

SYNTAX:

The Syntax is

fg [specify job]

OPTIONS:

There is no options for fg command.

EXAMPLE:

1. To move a process in foreground:

Lets start some three jobs and suspend those running process in background.

kmail- start the email client application.

Press ctrl+z to stop the current job.

xmms- music player application.

Press ctrl+z to stop the current job.

sleep 120- a dummy job.

Press ctrl+z to stop the current job.

jobs

The above command will display the jobs in the shell.

```
[1]   Stopped              kmail
[2]-  Stopped              xmms
[3]+  Stopped              sleep 120
fg 1
```

The above command will run the kmail application process in foreground.

file COMMAND:

file command tells you if the object you are looking at is a file or a directory.

SYNTAX:

The Syntax is

file [options] directoryname/filename

OPTIONS:

-c	Check the magic file for format errors. For reasons of efficiency, this validation is normally not carried out.
-h	Do not follow symbolic links.
-m	Use mfile as an alternate magic file.
-f	ffile contains a list of the files to be examined.

EXAMPLE:

1. file *.txt

Output:

```
aprlist.txt:   ASCII English text
cal.txt:       ASCII text
marchlist.txt: ASCII English text
text.txt:      ASCII text
```

Prints the 'txt' files.

find COMMAND:

find command finds one or more files assuming that you know their approximate filenames.

SYNTAX:

The Syntax is

find path [options]

OPTIONS:

-name	It search for the given file, in the current directory or any other subdirectory.

EXAMPLE:

1. find -name 'cal.txt'

   The system would search for any file named 'cal.txt' in the current directory and any subdirectory.

finger COMMAND:

finger command displays the user's login name, real name, terminal name and write status (as a "*" after the terminal name if write permission is denied), idle time, login time, office location and office phone number..

SYNTAX:

The Syntax is

finger [-lmsp] [user ...] [user@host ...]

OPTIONS:

- l	Prints all the information described by -s option and also the user's home directory, home phone number, login shell, mail status, and the contents of the files ".plan",".project",".pgpkey", and ".forward" from the users home directory.
- m	Match arguments only on user name (not first or last name).
- p	Supress the prinitng format of -l, It will not display the contents of ".plan",".project", and ".pgkey" files.
- s	Prints the output in short format.

EXAMPLE:

```
1. To Print the user information in short format:
2. finger -s hiox

3. Login    Name       Tty    Idle  LoginTime   Office OfficePhone
4. HIOX    HIOX INDIA   *:0          Sep 14 09:07
5. HIOX    HIOX INDIA   *pts/0     9  Sep 14 09:08
6. HIOX    HIOX INDIA   *pts/1  1:29  Sep 14 09:12
```

finger command prints the user information as user's login name, real name, terminal name and write status, idle time, login time, office location and office phone number.

ftp COMMAND:

ftp [*options*] [*hostname*]

Transfer files to and from remote network site *hostname*. **ftp** prompts the user for a command. The commands are listed after the options. Some of the commands are toggles, meaning they turn on a feature when it is off and vice versa. Note that versions may have different options.

Options

-d

Enable debugging.

-g

Disable filename globbing.

-i

Turn off interactive prompting.

-n

No autologin upon initial connection.

-v

Verbose. Show all responses from remote server.

ls COMMAND:

ls command lists the files and directories under current working directory.

SYNTAX:

The Syntax is

ls [OPTIONS]... [FILE]

OPTIONS:

-l	Lists all the files, directories and their mode, Number of links, owner of the file, file size, Modified date and time and filename.
-t	Lists in order of last modification time.
-a	Lists all entries including hidden files.
-d	Lists directory files instead of contents.
-p	Puts slash at the end of each directories.
-u	List in order of last access time.
-i	Display inode information.
-ltr	List files order by date.
-lSr	List files order by file size.

EXAMPLE:

1. Display root directory contents:

   ls /

   lists the contents of root directory.

2. Display hidden files and directories:

   ls -a

   lists all entries including hidden files and directories.

man COMMAND:

man command which is short for manual. It provides in depth information about the requested command (or) allows users to search for commands related to a particular keyword.

SYNTAX:
The Syntax is
```
man commandname [options]
```

EXAMPLE:

1. man mkdir

   Display the information about mkdir command.

mkdir COMMAND:
mkdir command is used to create one or more directories.

SYNTAX:
The Syntax is
```
mkdir [options] directories
```

EXAMPLE:
Create directory:
```
mkdir test
```

The above command is used for create the directory 'test'.

more COMMAND:
more command is used to display text in terminal screen. It allows only backward movement.

SYNTAX:
The Syntax is
```
more [options] filename
```

OPTIONS:

-c	Clear screen before displaying.
-e	Exit immediately after writing the last line of the last file in the argument list.
-n	Specify how many lines are printed in the screen for a given file.
+n	Starts up the file from the given number.

EXAMPLE:

1. more -c index.php

   Clear the screen before printing the file .

mv COMMAND:
mv command which is short for move. This command is used for move/rename file from one directory to another. mv command is different from cp command as it completely removes the file from the source and moves to the directory specified, where cp command just copies the content from one file to another.

SYNTAX:
The Syntax is
```
mv [-f] [-i] oldname newname
```

EXAMPLE:

1. To Rename / Move a file:

```
mv file1.txt file2.txt
```

This command renames file1.txt as file2.txt

2. To Move multiple files/More files into another directory

```
mv file1.txt tmp/file2.txt newdir
```

This command moves the files file1.txt from the current directory and file2.txt from the tmp folder/directory to newdir.

passwd COMMAND:

passwd command is used to change your password.

SYNTAX:

The Syntax is

```
passwd [options]
```

EXAMPLE:

1. passwd

   Entering just passwd would allow you to change the password. After entering passwd you will receive the following three prompts:

   ```
   Current Password:
   New Password:
   Confirm New Password:
   ```

   Each of these prompts must be entered correctly for the password to be successfully changed.

pwd COMMAND:

pwd - Print Working Directory. pwd command prints the full filename of the current working directory.

SYNTAX:

The Syntax is pwd [options]

EXAMPLE:

Displays the current working directory.

```
pwd
```

If you are working in home directory then, pwd command displays the current working directory as /home.

rm COMMAND:

rm linux command is used to remove/delete the file from the directory.

SYNTAX:

The Syntax is

```
rm [options..] [file | directory]
```

EXAMPLE:

To Remove / Delete a file:

```
rm file1.txt
```

Here rm command will remove/delete the file file1.txt.

rmdir COMMAND:

rmdir command is used to delete/remove a directory and its subdirectories.

SYNTAX:

The Syntax is

```
rmdir [options..] Directory
```

EXAMPLE:

To delete/remove a directory

```
rmdir tmp
```

rmdir command will remove/delete the directory tmp if the directory is empty.

Vi COMMAND :

This "**vi**" tutorial is intended for those who wish to master and advance their skills beyond the basic features of the basic editor. It covers buffers, "**vi**" command line instructions, interfacing with UNIX commands, and ctags. The **vim** editor is an enhanced version of **vi**. The improvements are clearly noticed in the handling of tags.

The advantage of learning **vi** and learning it well is that one will find **vi** on all Unix based systems and it does not consume an inordinate amount of system resources. Vi works great over slow network ppp modem connections and on systems of limited resources. One can completely utilize vi without departing a single finger from the keyboard.

who COMMAND:

who command can list the names of users currently logged in, their terminal, the time they have been logged in, and the name of the host from which they have logged in.

SYNTAX:

The Syntax is

```
who [options] [file]
```

OPTIONS:

am i	Print the username of the invoking user, The 'am' and 'i' must be space separated.

EXAMPLE:

1. who -uH

   Output:

   ```
   NAME     LINE         TIME         IDLE          PID COMMENT
   hiox     ttyp3        Jul 10 11:08   .           4578
   ```

   This sample output was produced at 11 a.m. The "." indiacates activity within the last minute.

2. who am i

   who am i command prints the user name.

SIMPLE FILTERS

Filters are the commands which accept data from standard input manipulate it and write the results to standard output. Filters are the central tools of the UNIX tool kit, and each filter performs a simple function. Some commands use delimiter, pipe (|) or colon (:). Many filters work well with delimited fields, and some simply won't work without them. The piping mechanism allows the standard output of one filter serve as standard input of another. The filters can read data from standard input when used without a filename as argument, and from the file otherwise

The Simple Database

Several UNIX commands are provided for text editing and shell programming. (emp.lst) - each line of this file has six fields separated by five delimiters. The details of an employee are stored in one single line. This text file designed in fixed format and containing a personnel database. There are 15 lines, where each field is separated by the delimiter |.

```
$ cat emp.lst

2233 | a.k.shukla | g.m | sales | 12/12/52 | 6000
9876 | jai sharma | director | production | 12/03/50 | 7000
5678 | sumit chakrobarty | d.g.m. | marketing | 19/04/43 | 6000
2365 | barun sengupta | director | personnel | 11/05/47 | 7800
5423 | n.k.gupta | chairman | admin | 30/08/56 | 5400
1006 | chanchal singhvi | director | sales | 03/09/38 | 6700
6213 | karuna ganguly | g.m. | accounts | 05/06/62 | 6300
1265 | s.n. dasgupta | manager | sales | 12/09/63 | 5600
4290 | jayant choudhury | executive | production | 07/09/50 | 6000
2476 | anil aggarwal | manager | sales | 01/05/59 | 5000
6521 | lalit chowdury | directir | marketing | 26/09/45 | 8200
3212 | shyam saksena | d.g.m. | accounts | 12/12/55 | 6000
3564 | sudhir agarwal | executive | personnel | 06/07/47 | 7500
2345 | j. b. sexena | g.m. | marketing | 12/03/45 | 8000
0110 | v.k.agrawal | g.m.| marketing | 31/12/40 | 9000
```

pr : paginating files

We know that,

```
cat dept.lst

01|accounts|6213
02|progs|5423
03|marketing|6521
04|personnel|2365
05|production|9876
06|sales|1006
```

pr command adds suitable headers, footers and formatted text. pr adds five lines of margin at the top and bottom. The header shows the date and time of last modification of the file along with the filename and page number.

```
pr dept.lst
```

```
May 06 10:38 1997 dept.lst page 1

01:accounts:6213
02:progs:5423
03:marketing:6521
04:personnel:2365
05:production:9876
06:sales:1006
```

…blank lines…

pr options

The different options for pr command are:

- -k prints k (integer) columns
- -t to suppress the header and footer
- -h to have a header of user’s choice
- -d double spaces input
- -n will number each line and helps in debugging
- -on offsets the lines by n spaces and increases left margin of page

pr +10 chap01

starts printing from page 10

pr -l 54 chap01

this option sets the page length to 54

head – displaying the beginning of the file

The command displays the top of the file. It displays the first 10 lines of the file, when used without an option.

```
head emp.lst
```

-n to specify a line count
head -n 3 emp.lst

will display the first three lines of the file.

tail – displaying the end of a file

This command displays the end of the file. It displays the last 10 lines of the file, when used without an option.

```
tail emp.lst
```

-n to specify a line count

```
tail -n 3 emp.lst
```

displays the last three lines of the file. We can also address lines from the beginning of the file instead of the end. The +count option allows to do that, where count represents the line number from where the selection should begin.

```
        tail +11 emp.lst
                Will display 11th line onwards
```

Different options for tail are:

- Monitoring the file growth (-f)
- Extracting bytes rather than lines (-c)

Use tail –f when we are running a program that continuously writes to a file, and we want to see how the file is growing. We have to terminate this command with the interrupt key.

cut – slitting a file vertically

It is used for slitting the file vertically. head -n 5 emp.lst | tee shortlist will select the first five lines of emp.lst and saves it to *shortlist.* We can cut by using -c option with a list of column numbers, delimited by a comma (cutting columns).

```
cut -c 6-22,24-32 shortlist
```

```
cut -c -3,6-22,28-34,55- shortlist
```

The expression 55- indicates column number 55 to end of line. Similarly, -3 is the same as 1-3.

Most files don't contain fixed length lines, so we have to cut fields rather than columns (cutting fields).

-d for the field delimiter
-f for the field list

```
cut -d \ | -f 2,3 shortlist | tee cutlist1
```

will display the second and third columns of *shortlist* and saves the output in *cutlist1.* here | is escaped to prevent it as pipeline character

- To print the remaining fields, we have

```
                cut –d \ | -f 1,4- shortlist > cutlist2
```

paste – pasting files

When we cut with *cut,* it can be pasted back with the *paste* command, *vertically* rather than horizontally. We can view two files side by side by pasting them. In the previous topic, cut was used to create the two files cutlist1 and cutlist2 containing two cut-out portions of the same file.

```
                paste cutlist1 cutlist2
```

We can specify one or more delimiters with -d

```
paste -d “|” cutlist1 cutlist2
```

Where each field will be separated by the delimiter |. Even though paste uses at least two files for concatenating lines, the data for one file can be supplied through the standard input.

Joining lines (-s)

Let us consider that the file *address book* contains the details of three persons

```
cat addressbook

paste -s addressbook   -to print in one single line

paste -s -d ”| | \n” addressbook   -are used in a circular manner
```

sort : ordering a file

Sorting is the ordering of data in ascending or descending sequence. The sort command orders a file and by default, the entire line is sorted

```
sort shortlist
```

This default sorting sequence can be altered by using certain options. We can also sort one or more keys (fileds) or use a different ordering rule.

sort options

The important sort options are:

-tchar	uses delimiter *char* to identify fields
-k n	sorts on nth field
-k m,n	starts sort on mth field and ends sort on nth field
-k m.n	starts sort on nth column of mth field
-u	removes repeated lines
-n	sorts numerically
-r	reverses sort order
-f	folds lowercase to equivalent uppercase
-m list	merges sorted files in list
-c	checks if file is sorted
-o flname	places output in file flname

```
sort –t“|” –k 2 shortlist
```

sorts the second field (name)

```
sort –t”|” –r –k 2 shortlist          or

sort –t”|” –k 2r shortlist
```

sort order can be revered with this –r option.

```
sort –t”|” –k 3,3 –k 2,2 shortlist
```

sorting on secondary key is also possible as shown above.

sort –t”|” –k 5.7,5.8 shortlist

we can also specify a character position with in a field to be the beginning of sort as shown above (sorting on columns).

sort –n numfile

when sort acts on numericals, strange things can happen. When we sort a file containing only numbers, we get a curious result. This can be overridden by –n (numeric) option.

cut –d “|” –f3 emp.lst | sort –u | tee desigx.lst

Removing repeated lines can be possible using –u option as shown above. If we cut out the designation filed from emp.lst, we can pipe it to sort to find out the unique designations that occur in the file.

Other sort options are:

sort –o sortedlist –k 3 shortlist

sort –o shortlist shortlist

sort –c shortlist

sort –t “|” –c –k 2 shortlist

sort –m foo1 foo2 foo3

uniq command – locate repeated and nonrepeated lines

When we concatenate or merge files, we will face the problem of duplicate entries creeping in. we saw how sort removes them with the –u option. UNIX offers a special tool to handle these lines – the uniq command. Consider a sorted dept.lst that includes repeated lines:

cat dept.lst

displays all lines with duplicates. Where as,

uniq dept.lst

simply fetches one copy of each line and writes it to the standard output. Since uniq requires a sorted file as input, the general procedure is to sort a file and pipe its output to uniq. The following pipeline also produces the same output, except that the output is saved in a file:

sort dept.lst | uniq – uniqlist

Different uniq options are :

Selecting the nonrepeated lines (-u)

cut –d “|” –f3 emp.lst | sort | uniq –u

Selecting the duplicate lines (-d)

```
cut –d "|" –f3 emp.lst | sort | uniq –d
```

Counting frequency of occurrence (-c)

```
cut –d "|" –f3 emp.lst | sort | uniq –c
```

tr command – translating characters

The tr filter manipulates the individual characters in a line. It translates characters using one or two compact expressions.

tr options expn1 expn2 standard input

It takes input only from standard input, it doesn't take a filename as argument. By default, it translates each character in expression1 to its mapped counterpart in expression2. The first character in the first expression is replaced with the first character in the second expression, and similarly for the other characters.

```
tr '|/' '~-' < emp.lst | head –n 3

exp1='|/' ; exp2='~-'

tr "$exp1" "$exp2" < emp.lst
```

Changing case of text is possible from lower to upper for first three lines of the file.

```
head –n 3 emp.lst | tr '[a-z]' '[A-Z]'
```

Different **tr options** are:
Deleting charecters (-d)

```
tr –d '|/' < emp.lst | head –n 3
```

Compressing multiple consecutive charecters (-s)

```
tr –s ' ' < emp.lst | head –n 3
```

Complementing values of expression (-c)

```
tr –cd '|/' < emp.lst
```

Using ASCII octal values and escape sequences

```
tr '|' '\012' < emp.lst | head –n 6
```

FILTERS USING REGULAR EXPRESSIONS – grep and awk

We often need to search a file for a pattern, either to see the lines containing (or not containing) it or to have it replaced with something else. This chapter discusses two important filters that are specially suited for these tasks – grep and sed. grep takes care of all search requirements we may have. sed goes further and can even manipulate the individual characters in a line. In fact sed can de several things, some of then quite well.

grep – searching for a pattern

It scans the file / input for a pattern and displays lines containing the pattern, the line numbers or filenames where the pattern occurs. It's a command from a special family in UNIX for handling search requirements.

grep *options pattern filename(s)*

```
grep “sales” emp.lst
```

will display lines containing sales from the file emp.lst. Patterns with and without quotes is possible. It's generally safe to quote the pattern. Quote is mandatory when pattern involves more than one word. It returns the prompt in case the pattern can't be located.

```
grep president emp.lst
```

When grep is used with multiple filenames, it displays the filenames along with the output.

```
grep “director” emp1.lst emp2.lst
```

Where it shows filename followed by the contents

grep options

grep is one of the most important UNIX commands, and we must know the options that POSIX requires grep to support. Linux supports all of these options.

-i	ignores case for matching
-v	doesn't display lines matching expression
-n	displays line numbers along with lines
-c	displays count of number of occurrences
-l	displays list of filenames only
-e exp	specifies expression with this option
-x	matches pattern with entire line
-f file	takes pattrens from file, one per line
-E	treats pattren as an extended RE
-F	matches multiple fixed strings

```
grep -i ‘agarwal’ emp.lst

grep -v ‘director’ emp.lst > otherlist
```

wc -l otherlist will display 11 otherlist grep –n ‘marketing’ emp.lst

grep –c ‘director’ emp.lst

grep –c ‘director’ emp*.lst

will print filenames prefixed to the line count

grep –l ‘manager’ *.lst

will display filenames *only*

grep –e ‘Agarwal’ –e ‘aggarwal’ –e ‘agrawal’ emp.lst

will print matching multiple patterns

grep –f pattern.lst emp.lst

all the above three patterns are stored in a separate file *pattern.lst*

Basic Regular Expressions (BRE) – An Introduction

It is tedious to specify each pattern separately with the -e option. grep uses an expression of a different type to match a group of similar patterns. If an expression uses meta characters, it is termed a regular expression. Some of the characters used by regular expression are also meaningful to the shell.

BRE character subset

The basic regular expression character subset uses an elaborate meta character set, overshadowing the shell’s wild-cards, and can perform amazing matches.

*****	Zero or more occurrences
g*	nothing or g, gg, ggg, etc.
.	A single character
.*	nothing or any number of characters
[pqr]	a single character p, q or r
[c1-c2]	a single character within the ASCII range represented by c1 and c2

The character class

grep supports basic regular expressions (BRE) by default and extended regular expressions (ERE) with the –E option. A regular expression allows a group of characters enclosed within a pair of [ ], in which the match is performed for a single character in the group.

grep “[aA]g[ar][ar]wal” emp.lst

A single pattern has matched two similar strings. The pattern [a-zA-Z0-9] matches a single alphanumeric character. When we use range, make sure that the character on the left of the hyphen has a lower ASCII value than the one on the right. Negating a class (^) (caret) can be used to negate the character class. When the character class begins with this character, all characters other than the ones grouped in the class are matched.

The *

The asterisk refers to the immediately preceding character. * indicates zero or more occurrences of the previous character.

g* nothing or g, gg, ggg, etc.

grep "[aA]gg*[ar][ar]wal" emp.lst

Notice that we don't require to use –e option three times to get the same output!!!!!

The dot

A dot matches a single character. The shell uses ? Character to indicate that.

.* signifies any number of characters or none

grep "j.*saxena" emp.lst

Specifying Pattern Locations (^ and $)

Most of the regular expression characters are used for matching patterns, but there are two that can match a pattern at the beginning or end of a line. Anchoring a pattern is often necessary when it can occur in more than one place in a line, and we are interested in its occurance only at a particular location.

^	for matching at the beginning of a line
$	for matching at the end of a line

grep "^2" emp.lst

Selects lines where emp_id starting with 2

grep "7…$" emp.lst

Selects lines where emp_salary ranges between 7000 to 7999

grep "^[^2]" emp.lst

Selects lines where emp_id doesn't start with 2

When meta characters lose their meaning

It is possible that some of these special characters actually exist as part of the text. Sometimes, we need to escape these characters. For example, when looking for a pattern g*, we have to use \
To look for [, we use \[
To look for .*, we use \.*

Extended Regular Expression (ERE) and grep

If current version of grep doesn't support ERE, then use egrep but without the –E option. -E option treats pattern as an ERE.

+	matches one or more occurrences of the previous character

?	Matches zero or one occurrence of the previous character

b+ matches b, bb, bbb, etc.

b? matches either a single instance of b or nothing

These characters restrict the scope of match as compared to the *

```
grep –E “[aA]gg?arwal” emp.lst
```

```
# ?include +<stdio.h>
```

The ERE set

ch+	matches one or more occurrences of character ch
ch?	Matches zero or one occurrence of character ch
exp1|exp2	matches exp1 or exp2
(x1|x2)x3	matches x1x3 or x2x3

Matching multiple patterns (|, (and))

```
grep –E ‘sengupta|dasgupta’ emp.lst
```

We can locate both without using –e option twice, or

```
grep –E ‘(sen|das)gupta’ emp.lst
```

awk

awk is a programmable, pattern-matching, and processing tool available in UNIX. It works equally well with text and numbers. It derives its name from the first letter of the last name of its three authors namely Alfred V. Aho, Peter J. Weinberger and Brian W. Kernighan.

awk is not just a command, but a programming language too. In other words, awk utility is a pattern scanning and processing language. It searches one or more files to see if they contain lines that match specified patterns and then perform associated actions, such as writing the line to the standard output or incrementing a counter each time it finds a match.

Syntax:

```
awk option 'selection_criteria {action}' file(s)
```

Here, selection_criteria filters input and selects lines for the action component to act upon. The selection_criteria is enclosed within single quotes and the action within the curly braces. Both the selection_criteria and action forms an awk program.

Example: `$ awk '/manager/ { print }' emp.lst`

Output:

```
9876    Jai Sharma    Manager    Productions
2356    Rohit         Manager    Sales
5683    Rakesh        Manager    Marketing
```

```
[priyanka@localhost awk]$ cat emp.dbf
0001|Sandeep Agarwalla     |Education  |Student         |5000
0005|Sanjay Agarwal        |Education  |Student         |5000
0007|Deepak Agarwal        |Businessman|                |  --
0054|Devadatta Bhattacharya|Education  |Teacher         |6500
0018|Tamali Neogi          |Education  |Teacher         |5500
0981|Sukla Bhattacharya    |Education  |Reception       |4500
0221|Dilip Roy             |Sales      |Clerk           |2500
0111|Hamanta Mukherji      |Purchase   |Manager         |6400
2000|Mainak Roy            |Purchase   |Dy Manager      |5500
0987|Arunava Sharkhel      |Sales      |Maneger         |4500
0911|Bikash Agarwal        |Sales      |Dy Maneger      |4500
0211|Arunangshu Sarkar     |Hardware   |Student         |4500
0101|Anish Basu            |Software   |Student         |5200
0102|Arnab Roy             |Marketing  |Manager         |4700
1111|Kaustav Choudhury     |Marketing  |Dy Manager      |4500
1211|Banani Ghose          |Education  |Student         |8000
1288|Anirban Mahata        |Hardware   |Student         |3000
1200|Angshuman Chatterjee  |Marketing  |General Manager |7500
2000|Anshumita Das         |Sales      | General Manager|3000
```

```
[priyanka@localhost awk]$ awk '/Education/ {print}' emp.dbf
```

```
0001|Sandeep Agarwalla     |Education  |Student         |5000
0005|Sanjay Agarwal        |Education  |Student         |5000
0054|Devadatta Bhattacharya|Education  |Teacher         |6500
0018|Tamali Neogi          |Education  |Teacher         |5500
0981|Sukla Bhattacharya    |Education  |Reception       |4500
1211|Banani Ghose          |Education  |Student         |8000

[priyanka@localhost awk]$ awk -F "|" '/Sales/ {print $2,$3,$5}' emp.dbf
Dilip Roy             Sales       2500
Arunava Sharkhel      Sales       4500
Bikash Agarwal        Sales       4500
Anshumita Das         Sales       3000

[priyanka@localhost awk]$ awk -F "|" '/Sales/ {print $2"|",$3"|",$5}' emp.dbf
Dilip Roy            | Sales      | 2500
Arunava Sharkhel     | Sales      | 4500
Bikash Agarwal       | Sales      | 4500
Anshumita Das        | Sales      | 3000

[priyanka@localhost awk]$ awk -F "|" 'NR == 3 || NR == 6 { print NR,$2,$3,$5}' emp.dbf
3 Deepak Agarwal         Businessman   --
6 Sukla Bhattacharya     Education   4500

[priyanka@localhost awk]$ awk -F "|" 'NR == 3 , NR == 6 { print NR,$2,$3,$5}' emp.dbf
3 Deepak Agarwal         Businessman   --
4 Devadatta Bhattacharya Education   6500
5 Tamali Neogi           Education   5500
6 Sukla Bhattacharya     Education   4500


[priyanka@localhost awk]$ awk -F "|" '$3 == "Education  " || $3=="Sales      "' emp.dbf
0001|Sandeep Agarwalla     |Education  |Student         |5000
0005|Sanjay Agarwal        |Education  |Student         |5000
0054|Devadatta Bhattacharya|Education  |Teacher         |6500
0018|Tamali Neogi          |Education  |Teacher         |5500
0981|Sukla Bhattacharya    |Education  |Reception       |4500
0221|Dilip Roy             |Sales      |Clerk           |2500
0987|Arunava Sharkhel      |Sales      |Maneger         |4500
0911|Bikash Agarwal        |Sales      |Dy Maneger      |4500
1211|Banani Ghose          |Education  |Student         |8000
2000|Anshumita Das         |Sales      | General Manager|3000
```

Note : Spaces does matter in searching expressions.

1. remove only files:
   ls -l * | grep -v drwx | awk '{print "rm "$9}' | sh
   or with awk alone:
   ls -l|awk '$1!~/^drwx/{print $9}'|xargs rm
   Be careful when trying this out in your home directory. We remove files!
2. remove only directories
   ls -l | grep '^d' | awk '{print "rm -r "$9}' | sh or
   ls -p | grep /$ | wk '{print "rm -r "$1}'

How to Run Shell Scripts

There are two ways you can execute your shell scripts. Once you have created a script file:

Method 1
Pass the file as an argument to the shell that you want to interpret your script.

Step 1 : create the script using **vi, ex or ed**

For example, the script file **show** has the following lines

```
echo Here is the date and time
date
```

Step 2 : To run the script, pass the **filename** as an argument to the sh (shell)

```
$ sh show
Here is the date and time
Sat jun 03 13:40:15 PST 2006
```

Method 2:
Make your script executable using the chmod command.

When we create a file, by default it is created with read and write permission turned on and execute permission turned off. A file can be made executable using chmod.

Step 1 : create the script using **vi, ex or ed**

For example, the script file **show** has the following lines

```
echo Here is the date and time
date
```

Step 2 : Make the file executable

```
$ chmod u+x script_file
$ chmod u+x show
```

Q)Write a shell script to find the EVEN & ODD numbers in a given list of numbers?

```
clear
echo -e "Enter how many inputs you want :- \c"
read a
c=1
while test $c -le $a;do
  echo -e "Enter no $c :- \c"
  read b
  if test `expr $b % 2` -eq 0;then
        echo $b>>e
  else
        echo $b>>o
  fi
  c=`expr $c + 1`
done
echo -e "\nList of Odd numbers"
cat o
echo -e "\nList of Even numbers"
cat e
rm e o
```

Output

```
Enter how many inputs you want :- 8
Enter no 1 :- 32
Enter no 2 :- 9
Enter no 3 :- 14
Enter no 4 :- 2
Enter no 5 :- 5
Enter no 6 :- 17
Enter no 7 :- 11
Enter no 8 :- 3

List of Odd numbers
9
5
17
11
3

List of Even numbers
32
14
2
```

Q)Write a MENU driven program in the following :-

1. **Contents of the \etc\passwd.**
2. **List of users who have currently logged in.**
3. **Print working directory**
4. **Exit**

```
ch=0
while : ; do
        clear
        echo -e "1. Contents of /etc/passwd
2. List of current users who has logged in
3. Path and name of working directory
4. Exit\n
Enter your choice (1-4) : \c"
        read ch
        case $ch in

                1) cat /etc/passwd
                   echo -e "\nPress any key to continue...........\c"
                   read w;;

                2) who -H
                   echo -e "\nPress any key to continue...........\c"
                   read w;;

                3) pwd
                   echo -e "\nPress any key to continue...........\c"
                   read w;;

                4) break;;

                *) echo "Invalid input!!!!!!!!!!!"
                   echo -e "\nPress any key to continue..........
                read w;;
        esac
done
```

Output

```
1. Contents of /etc/passwd
2. List of current users who has logged in
3. Path and name of working directory
4. Exit

Enter your choice (1-4) : 1

root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:
daemon:x:2:2:daemon:/sbin:
adm:x:3:4:adm:/var/adm:
lp:x:4:7:lp:/var/spool/lpd:
sync:x:5:0:sync:/sbin:/bin/sync
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
halt:x:7:0:halt:/sbin:/sbin/halt
mail:x:8:12:mail:/var/spool/mail:
news:x:9:13:news:/var/spool/news:
uucp:x:10:14:uucp:/var/spool/uucp:
operator:x:11:0:operator:/root:
games:x:12:100:games:/usr/games:
gopher:x:13:30:gopher:/usr/lib/gopher-data:
ftp:x:14:50:FTP User:/var/ftp:
nobody:x:99:99:Nobody:/:
named:x:25:25:Named:/var/named:/bin/false
xfs:x:43:43:X Font Server:/etc/X11/fs:/bin/false
gdm:x:42:42::/home/gdm:/bin/bash
rpcuser:x:29:29:RPC Service User:/var/lib/nfs:/bin/false
rpc:x:32:32:Portmapper RPC user:/:/bin/false
mailnull:x:47:47::/var/spool/mqueue:/dev/null
jk:x:500:500:Jiban Kumar Sahu:/home/jk:/bin/bash
sawan:x:501:500:Sawan Kumar:/home/sawan:/bin/bash
subarna:x:502:501:subarna saha:/home/subarna:/bin/bash
tirtha:x:503:502:tirtha roy chaudhury:/home/tirtha:/bin/bash
sandeep:x:504:502:sandeep agarwalla:/home/sandeep:/bin/bash

Press any key to continue...........



1. Contents of /etc/passwd
2. List of current users who has logged in
3. Path and name of working directory
4. Exit

Enter your choice (1-4) : 2


USER     LINE     LOGIN-TIME   FROM
sandeep  pts/0    May 22 09:30
sandeep  pts/2    May 22 08:33
```

```
Press any key to continue...........

1. Contents of /etc/passwd
2. List of current users who has logged in
3. Path and name of working directory
4. Exit

Enter your choice (1-4) : 3

/home/sandeep

Press any key to continue...........

1. Contents of /etc/passwd
2. List of current users who has logged in
3. Path and name of working directory
4. Exit

Enter your choice (1-4) : 7
Invalid input!!!!!!!!!!

Press any key to continue...........

1. Contents of /etc/passwd
2. List of current users who has logged in
3. Path and name of working directory
4. Exit

Enter your choice (1-4) : 4
[sandeep@server sandeep]$
```

Q)Write a shell script that takes a command line argument that inputs a number n and a word. The word would then be printed “n” times (one word per line).

```
clear
c=1
echo -e "                Printing words $1 times\n\n"
while test $c -le $1;do
        echo $2
        c=`expr $c + 1`
done
```

<u>Output</u>

```
[sandeep@server sandeep]$ sh string 10 UNIX

     Printing words 10 times


UNIX
UNIX
UNIX
UNIX
UNIX
UNIX
UNIX
UNIX
UNIX
UNIX
```

Q)Write a shell script to print the multiplication table of a given number?

```
clear
echo -e "Enter the number for multiplication table :- \c"
read n
echo
c=1
while test $c -le 20;do
        if test $c -lt 10; then
                echo "$n X $c  = `expr $n \* $c`"
        else
                echo "$n X $c = `expr $n \* $c`"
        fi
        c=`expr $c + 1`
done
```

Output

```
Enter the number for multiplication table :- 18

18 X 1  = 18
18 X 2  = 36
18 X 3  = 54
18 X 4  = 72
18 X 5  = 90
18 X 6  = 108
18 X 7  = 126
18 X 8  = 144
18 X 9  = 162
18 X 10 = 180
18 X 11 = 198
18 X 12 = 216
18 X 13 = 234
18 X 14 = 252
18 X 15 = 270
18 X 16 = 288
18 X 17 = 306
18 X 18 = 324
18 X 19 = 342
18 X 20 = 360
```

Q)Write a shell script which executed as soon as the user login to his/her account displaying the message “Good Morning” || “Good Afternoon” || “Good Evening” depending upon the time at which the user login?

```
clear
echo -e "Now the time is : - `date  +"%T"`"
hour=`date +"%T" | cut -d ":" -f1`
if [ $hour -lt 12 ] && [ $hour -ge 0 ];then
        echo "Good Morning !"
elif [ $hour -ge 12 ] && [ $hour -lt 16 ];then
        echo "Good Afternoon !"
else
        echo "Good Evening !"
fi
```

The Output is shown below shows the time the user as soon as the user logs to his/her account.The shell script above is written in a file named time and placed in the file .bash_profile which gets executed when the user login.

```
Now the time is : -  17:22:48
Good Evening !
[sandeep@server sandeep]$
```

Q)Write a shell script to convert the content of file to upper case. The file name should be given as argument from command line?

```
clear
if test -f $1 ; then
        tr '[a-z]'  '[A-Z]' < $1 > temp
        mv temp $1
fi
```

<u>Output</u>

<u>Contents before program executed</u>

```
[sandeep@server sandeep]$ cat ff
the quick brown fox jumps over the lazy dog
the quick brown fox jumps over the lazy dog
```

<u>Contents after program executed</u>

```
[sandeep@server sandeep]$ cat ff
THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG
THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG
```

Q)Write a shell script whick will receive any name of file as argument. The script should check whether argument supplied is a file or directory. If the argument is a file then it should print the number of lines, words and character present in the file?

```
clear
if test -f $1 ; then
echo "Number of lines `wc -l <$1`"
echo "Number of words `wc -w <$1` "
echo "Number of character `wc -c <$1` "
elif test -d $1;then
echo "This is a directory file"
else
        echo "'$1' File is not existing !"
fi
```

<u>Output</u>

<u>The contents of the file</u>

```
[sandeep@server sandeep]$ cat l1
the quick
brown fox
jumps over thje lazy
dpog
```

<u>The program execution</u>

```
[sandeep@server sandeep]$ sh dd l1
Number of lines       4
Number of words       9
Number of character      46
```

Q)Write a shell script to rename all the files in the directory as extension .sh. Directories in that directory do not get renamed.

```
for file in * ; do
        x=`ls -ld $file | cut -c1`
        if test $x != 'd' ; then
                mv $file ${file}.sh
        fi
done
```

<u>Ouput</u>

<u>The contents of the directory</u>

```
[sandeep@server sandeep]$ ls
capital   dd     ddcclear  devc  f1    l1    string
chext     ddc    delete    dir1      fact  menu  tm
contents  ddcas  dev       even.pif  ff    mul
```

<u>The program execution</u>

```
[sandeep@server sandeep]$ sh chext
```

<u>The contents of directory after execution</u>

```
[sandeep@server sandeep]$ ls
capital.sh   ddc.sh       dev.sh   f1.sh    menu.sh
chext.sh     ddcas.sh     devc.sh  fact.sh  mul.sh
contents.sh  ddcclear.sh  dir1        ff.sh    string.sh
dd.sh        delete.sh    even.sh  l1.sh    tm.sh
```

*dir1 is a directory which doesn’t gets renamed.

Program to find the greatest three numbers

```
clear
echo -e "Enter first variable : \c"
read a
echo -e "Enter second variable : \c"
read b
echo -e "Enter third variable : \c"
read c
if test $a -gt $b ; then
        if test $a -gt $c ; then
                echo " `echo $a ` is maximum "
        else
                echo " `echo $c ` is maximum "
        fi
else
        if test $b -gt $c ; then
                echo " `echo $b ` is maximum "
        else
                echo " `echo $c ` is maximum "
        fi
fi
```

Program to find the year entered is leap year or not?

```
clear
echo -e "Enter an year : \c";
read year
if test $year -gt 0 ; then
        x=`expr $year % 4`
        y=`expr $year % 100`
        z=`expr $year % 400`
        if test $x -eq 0
        then
                if test $y -eq 0
                then
                        if test $z -eq 0
                        then
                                cal 2 $year
                        else
                                echo "`echo $year` is not a leap-year"
                        fi
                else
                        cal 2 $year
                fi
        else
                echo "`echo $year` is not a leap-year"
        fi
else
        echo "Bad input!!!!!!!!!!!!!!!!!!"
fi
```

Write program to test whether a given number is prime or not?

```
clear
echo -e "Enter a number = : \c"
read n
q=`expr $n - 1`
x=2
while test $x -le $q ; do
        m=`expr $n % $x`
        if test $m -eq 0 ; then
                echo "The number $n is not prime"
                break
        fi
        x=`expr $x + 1`
done
if test $m -ne 0 ; then
        echo "The number $n is prime"
fi
```

Program to find the factorial of a number?

```
clear
echo -e "Enter a number : \c"
read n
x=1
s=1
while test $x -le $n ; do
        s=`expr $s \* $x`
        x=`expr $x + 1`
done
echo "The factorial of $n = $s"
```

Write a program to print the fibbonacii sereies?

```
clear
echo -e "Enter number of terms = \c"
read n
p=0
q=1
echo "$p"
echo "$q"
m=`expr $n - 2`
x=1
while test $x -le $m ; do
        s=`expr $p + $q`
        echo "$s"
        p=`expr $q`
        q=`expr $s`
        x=`expr $x + 1`
done
```

Write a script which receives two file names as arguments and check if their contents are same.

```
clear
if test $# -eq 2;then
        if test -f $1;then
                if test -f $2;then
                        sort $1 > tp1
                        sort $2 > tp2
                        d=`comm -3 tp1 tp2 | wc -c`
                        if [ $d -eq 0 ];then
                                echo " Files are equal"
                        else
                                echo "Files are not equal"
                        fi
                else
                        echo "$2 is not a file"
                fi
        else
                echo "$1 is not a file"
        fi
else
        echo "Invalid arguments"
fi
```

Output

Contents before program executed

```
[root@localhost Sandeep]# cat temp1
the quick brown fox jumps over the lazy dog
the quick brown fox jumps over the lazy dog

[root@localhost Sandeep]# cat temp2
I am learning unix well.I love to do shell programs.
All the students of MCA love the unix teacher.
```

Contents after program executed

```
[root@localhost Sandeep]# sh compare.sh temp1 temp2
Files are not equal

[root@localhost Sandeep]# sh compare.sh temp1 temp1
Files are equal
[root@localhost Sandeep]# sh compare.sh f1 temp1
f1 is not a file
```

Program To Illustrate Fork System Call

```
#include<stdio.h>
main()
{
            int childid;
            int parentid;
            int processid;
            processid=getpid();
            childid=fork();
            if(childid<0)
                 {
                        printf("Fork Failed \n");
                        exit(1);
                  }
            else if(childid==0)
                {
                        printf("Iam the Child:\n\t\t  Childid= %d \t Prentid=%dProcessid=%d \n “,
                        ldid,getppid(),getpid());
                 }
            else
                 {
                        printf("Iam the Parent:\n\t\t  Childid=%d Parentid=%d
                        Processid=%d\n",childid,getppid(),getpid());
                   }
 }
```

Program to implement signal using sigint

```
#include<stdio.h>
#include<signal.h>

void abc();

main()

{
signal(SIGINT,abc);
while(1)
  {
     printf("hello worldf\n");
     sleep(1);
   }
}

void abc()
{

printf("\n you have a signal");

}
```

Program to implement signal using alarm signal

```
#include<stdio.h>
#include<signal.h>
#include<stdlib.h>


main()
{
   void abc();

 signal(SIGALRM,abc);

while(1)
   {
  alarm(60);/*60 means 60 seconds */
   pause();
    }
}
void abc()
{
system("cat f1");

/*f1 is the where all the appointments are written*/

return;
}
```

Write a program to create a child process. The parent process takes the name of a file and sends it to the child process and the child process finds the number of lines, words and the characters which it returns to the parent process. The parent process then displays them on the screen. Use pipes for IPC.

```
#include<stdio.h>
#include<conio.h>

main()
{
        File *fp;
        int p1[2],p2[2].pid,c=0,I,nol,now,noc;
        char flnm[30],rcvflnm[30],out[80],ch,in[80];

        pipe(p1);
        pipe(p2);

        pid=fork();

        /* parent */
        if(pid!=0)
        {
                close(p1[1]);
                close(p2[0]);

                Printf("enter the name of the file:");
                Scanf("%s",flnm);
                Write(p2[1],flnm,20);
                read(p1[0],in,80);
                Printf("%s",in);
                Printf("\n");
}
/*child*/
else
{
                close(p1[0]);
                close(p2[1]);

                read(p2[0],rcvflnm,30);

                fp=fopen(rcvflnm,"r");
                if(fp==NULL)
                {
                        printf("cannot open file");
                        exit(0);
                }
                  nol=0;     /*number of lines*/
                  now=0;    /*number of words*/
                  noc=0;      /*number of chracters*/
                  while(1)
                {
                        ch=fgetc(fp);
                        if(ch==EOF)
                                break;
                              noc++;
                        if(ch==` ` ||ch == `\t`)
                                now++;
                        if(ch==`\n`)
                        nol++;
                }
                printf(out,"no.of chars"%d, no.of words:%d,no.of lines:%d ",noc,now,nol);

                write(p1[1],out,80);
                fclose(fp);
 }
 exit(0);
}
```

Write a Client server program to send message from client to server using FIFOS (Names Pipes)

```
/* Client  (Write)*/

#include<fcntl.h>
#include<stdio.h>
#include<errno.h>
#include<string.h>

main(int argc, char *argv[])
{
        int fd,j,nwrite;
        char msgbuf[64];

        if((fd=open("myfifo",O_WRONLY))<0)
                perror("fifo open failed");

        for(j=1;j<argc;j++)
        {
                strcpy(msgbuf,argv[j]);
                if((nwrite=write(fd,msgbuf,64))<=0)
                        perror("message write failed");
        }
}

/* Server (Read) , Execute Read first*/

#include<fcntl.h>
#include<stdio.h>

main()
{
        int fd;
        char msgbuf[64];

        if(mknod("myfifo",010666,0)<0)
                perror("myfifo failed");

        if((fd=open("myfifo",O_RDWR))<0)
                perror("Fifo open failed");

        for(;;)
        {
                if(read(fd,msgbuf,64)>0)
                        printf("Message received %s\n",msgbuf);
        }
}
```

Program to create Message Queues.

```
#include<sys/types.h>
#include<sys/ipc.h>
#include<sys/msg.h>
#include<stdio.h>

main()
{
        int i,msqid;
        key_t key=11;

        msqid=msgget(key,IPC_CREAT|IPC_EXCL);

        if(msqid<0)
                perror("Msgget failed");
        else
                printf("MQ created with key %d\n",msqid);
}
```

Program to create Message Queues with permission.

```
#include<sys/types.h>
#include<sys/ipc.h>
#include<sys/msg.h>
#include<stdio.h>

main()
{
        int i,msqid;
        key_t key=15;

        msqid=msgget(key,IPC_CREAT|0644);

        if(msqid<0)
                perror("Msgget failed");
        else
                printf("MQ created with key %d\n",msqid);
}
```

Program to create Mutiple Message Queues.

```
#include<sys/ipc.h>
#include<sys/msg.h>
#include<stdio.h>

main()
{
	int msqid;
	key_t i;

	for(i=21;i<50;i++)
	{
		msqid=msgget(i,IPC_CREAT|0666);

		if(msqid<0)
			perror("Msgget failed");
		else
			printf("MQ created with key %d\n",msqid);
	}
}
```

Program to create Message Queues to make communication between a child and a parent.

```
#include<sys/types.h>
#include<sys/ipc.h>
#include<sys/msg.h>
#include<stdio.h>
#include<string.h>
#include<stdlib.h>

main()
{
        int qid,len;
        struct
        {
                long mtype;
                char mtext[225];
        }message,buff;

        qid=msgget((key_t)11,IPC_CREAT|0666);
        if(qid==-1)
        {
                perror("msgget failed");
                exit(1);
        }

        strcpy(message.mtext,"Hello World\n");
        message.mtype=1;

        len=strlen(message.mtext);
        if(msgsnd(qid,&message,len,0)==-1)
        {
                perror("msgsnd failed");
                exit(1);
        }
        if(msgrcv(qid,&buff,len,0,0)==-1)
        {
                perror("msgrcv failed");
                exit(1);
        }
        printf("Message received %s \n",buff.mtext);
}
```

Program to create Message Queues to make communication between two processes.

/* Process 1 */

```
#include<sys/types.h>
#include<sys/ipc.h>
#include<sys/msg.h>
#include<stdio.h>
#include<string.h>
#include<stdlib.h>

main()
{
        int qid,len;
        struct
        {
                long mtype;
                char mtext[225];
        }message,buff;

        qid=msgget((key_t)15,IPC_CREAT|0666);
        if(qid==-1)
        {
                perror("msgget failed");
                exit(1);
        }
        strcpy(message.mtext,"GOOD MORNING\n");
        message.mtype=1;

        if(msgsnd(qid,&message,21,0)==-1)
        {
                perror("msgsnd failed");
                exit(1);
        }
        strcpy(message.mtext,"GOOD AFTERNOON\n");
        message.mtype=2;
        if(msgsnd(qid,&message,21,0)==-1)
        {
                perror("msgsnd failed");
                exit(1);
        }
        strcpy(message.mtext,"GOOD EVENING\n");
        message.mtype=3;
        if(msgsnd(qid,&message,21,0)==-1)
        {
                perror("msgsnd failed");
                exit(1);
        }
        printf("Message received %s \n",buff.mtext);
}
```

/* Process 2 */

```
#include<sys/types.h>
#include<sys/ipc.h>
#include<sys/msg.h>
#include<stdio.h>
#include<string.h>
#include<stdlib.h>

main()
{
        int qid,i;
        struct
        {
                long mtype;
                char mtext[225];
        }buff;

        qid=msgget((key_t)15,IPC_CREAT|0666);
        if(qid==-1)
        {
                perror("msgget failed");
                exit(1);
        }

        for(i=0;i<3;i++)
        {
                if(msgrcv(qid,&buff,21,0,0)==-1)
                {
                        perror("Msgrcv failed");
                        exit(1);
                }
                printf("Message received %s \n",buff.mtext);
        }
}
```

Program to create 25 semaphores

```
#include<stdio.h>
#include<sys/types.h>
#include<sys/ipc.h>
#include<stdlib.h>

main()
{
	int semid;
	semid=semget(0x20,25,0666|IPC_CREAT);

	if(semid>0)
		printf("1st semget suceeded\n");
	else
	{
		perror("1st Senget");
		exit(0);
	}
}
```

Program to get the value of Semaphore

```
#include<stdio.h>
#include<sys/types.h>
#include<sys/sem.h>
#include<errno.h>

main()
{
	int semid,retval;
	semid=semget(0x20,1,0666|IPC_CREAT);
	retval=semctl(semid,0,GETVAL,0);
	printf("Value returned id %d\n",retval);
}
```

Program to set and get the value of Semaphore

```
#include<stdio.h>
#include<sys/types.h>
#include<sys/sem.h>
#include<errno.h>

main()
{
        int semid,retval;
        semid=semget(0x30,1,0666|IPC_CREAT);
        semctl(semid,0,SETVAL,1);
        retval=semctl(semid,0,GETVAL,0);
        printf("Value of the semaphores after setting is %d \n",retval);


        semctl(semid,0,SETVAL,2);
        retval=semctl(semid,0,GETVAL,0);
        printf("Value of the semaphores after setting is %d \n",retval);

}
```

Program to create binary semaphore

```
#include<stdio.h>
#include<sys/types.h>
#include<sys/ipc.h>
#include<sys/sem.h>

main()
{
        int semid,pid;

        struct sembuf sop;
        semid=semget(0x30,1,IPC_CREAT|0666);
        pid=fork();

        if(pid==0)
        {
                sleep(10);
                printf("Child before semop\n");
                sop.sem_num=0;
                sop.sem_op=0;
                sop.sem_flg=0;

                semop(semid,&sop,1);
                printf("Child over\n");
        }
        else
        {
                printf("Parent before 1st semctl\n");
                semctl(semid,0,SETVAL,1);
                printf("Parent Sleeping\n");
           sleep(15);
                printf("Parent before second semctl \n");
                semctl(semid,0,SETVAL,0);
                printf("Parent Over");
        }
}
```

Write a Program to create a Shared Memory, Create a child process to to reading and writing from shared memory.

```
#include<sys/types.h>
#include<sys/ipc.h>
#include<sys/shm.h>

main()
{
        char *ptr;
        int shmid,pid;

        shmid=shmget(key_t)1,20,IPC_CREAT|0666)
        ptr=(char *)shmat(shmid,(char *)0,0);

        pid=fork();

        if(pid==0)
                strcpy(ptr,"Hello World");
        else
        {
                wait(0);
                parent("Parent reads %s\n",ptr);
        }
}
```

Client Server Program for Connection Oriented Network using Sockets

/* CLIENT */

```
#include<stdio.h>
#include<sys/socket.h>
#include<arpa/inet.h>
#include<netinet/in.h>
#include<sys/types.h>
#include<stdlib.h>
#include<string.h>

#define MAX 80

main(int argc, char *argv[])
{
	int sockfd,s,i,n;
	struct sockaddr_in serv_addr;
	char buff1[MAX],buff2[MAX];

	sockfd=socket(AF_INET,SOCK_STREAM,0);
	bzero((char *)&serv_addr,sizeof(serv_addr));
	serv_addr.sin_family=AF_INET;
	serv_addr.sin_addr.s_addr=inet_addr("192.168.1.1");
	serv_addr.sin_port=htons(atoi(argv[1]));
	s=connect(sockfd,(struct sockaddr *) & serv_addr,sizeof(serv_addr));

	if(s<0)
	{
		printf("Error\n");
		exit(1);
	}
	for(;;)
	{
		write(1,"Enter Message:",15);
		n=read(0,buff1,MAX);
		send(sockfd,buff1,n,0);
		n=recv(sockfd,buff2,20,0);
		write(1,"Client received",18);
		write(1,buff2,n);
	}
	close(sockfd);
	exit(0);
}
```

/*SERVER*/

```
#include<stdio.h>
#include<sys/socket.h>
#include<arpa/inet.h>
```

```
#include<netinet/in.h>
#include<sys/types.h>
#include<stdlib.h>
#include<string.h>

main(int argc, char *argv[])
{
        int sockfd,newsockfd,n,i,cli_len,pid,MAX=80;
        struct sockaddr_in serv_addr,cli_addr;
        char buff[MAX];

        sockfd=socket(AF_INET,SOCK_STREAM,0);
        bzero((char *)&serv_addr,sizeof(serv_addr));
        serv_addr.sin_family=AF_INET;
        serv_addr.sin_addr.s_addr=htonl(INADDR_ANY);
        serv_addr.sin_port=htons(atoi(argv[1]));

        bind(sockfd,(struct sockaddr*)& serv_addr,sizeof(serv_addr));
        listen(sockfd,5);

        printf("server is waiting....\n");
        for(;;)
        {
                cli_len=sizeof(cli_addr);
                newsockfd=accept(sockfd,(struct sockaddr*)& cli_addr,&cli_len);
                if(newsockfd<0)
                {
                        write(1,"server!....accept error:\n",24);
                }
                pid=fork();

                if(pid==0)
                {
                        close(sockfd);
                        for(i=0;i<10;i++)
                        {
                                write(1,"\nserver recived:",18);
                                n=recv(newsockfd,buff,MAX,0);
                                write(1,buff,n);
                                write(1,"\nEnter Message:",15);
                                read(0,buff,MAX);
                                send(newsockfd,buff,MAX,0);
                        }
                        close(newsockfd);
                        exit(0);
                }
        }
}
```

Client Server Program for Connectionless Network using Sockets

```
/*CLIENT */
#include<stdio.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>
```

```
#include<netdb.h>
short portno;

int main(int argc,char *argv[])
{
        int sockfd,numbytes,addr_len;
        struct sockaddr_in servaddr,cliaddr;
        char buff1[100];
        if(argc!=4)
        {       printf("usage:client<portno><hostname><message>\”);
                exit(1);
        }
        if((sockfd=socket(AF_INET,SOCK_DGRAM,0))=-1)
        {
                printf("server socket");
                exit(1);
        }
        cliaddr.sin_family=AF_INET;
        cliaddr.sin_port=htons(0);
        cliaddr.sin_addr.s_addr=htonl(0l);
        if(bind(sockfd,(struct sockaddr*)&cliaddr,sizeof(struct sockaddr))=-1)
        {
                perror("bind error");
                exit(1);
        }
        portno=atoi(argv[1]);
        servaddr.sin_family=AF_INET;
        servaddr.sin_port=htons(portno);
        servaddr.sin_addr.s_addr=inet_addr(argv[2]);
        numbytes=sendto(sockfd,argv[3],strlen(argv[3]),0,(struct sockaddr*)&servaddr,sizeof(servaddr));
        if(numbytes<0)
        {       printf("\n client:send to error\n");
                exit(1);
        }
        printf("talker:\n");
        printf("send %d number of bytes to %s \n",numbytes, net_ntoa(servaddr.sin_addr));
        exit(0);
}
```

/*SERVER*/

```
#include<stdio.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>
short portno;

int main(int argc,char *argv[])
{
        int sockfd,numbytes,addr_len;
        struct sockaddr_in servaddr,cliaddr;
        char buff1[100];
        if(argc!=2)
        {
                printf("usage:sever<portno>");
        exit(1);
        }

        if((sockfd=socket(AF_INET,SOCK_DGRAM,0))==-1)
        {
                printf("server socket");
                exit(1);
        }
        portno=atoi(argv[1]);
        servaddr.sin_family=AF-INET;
        servaddr.sin_port=htons(portno);
        servaddr.sin_addr.s_addr=htonl(INADDR_ANY);

        if(bind(sockfd,(struct sockaddr*)&servaddr,sizeof(struct sockaddr))<0)
        {
                perror("bind error");
                exit(1);
        }
        addr_len=sizeof(cliaddr);

        if((numbytes=recvfrom(sockfd,buf,sizeof(buff),0,(struct sockaddr*)&cliaddr,&(addr_len)))<0)
        {
                perror("recvfrom");
                exit(1);
        }
        printf("listener:\n");
        printf("go packet from %s\n",inet_ntoa(cliaddr.sin_addr));
        printf("\npacket is %d bytes long \n",numbytes);
        printf("packet contains:%s \n",buff);
        exit(0);
}
```

SOCKET PROGRAM TO PERFORM TCP TIME CLIENT

```
#include<stdio.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>
short portno;
main(int argc,char* argv[])
{
        int sock,i;
        int read_frm_stdin,read_frm_sock;
        char buffer[512];
        struct sockaddr_in servaddr,cliaddr;
        if(argc!=3)
        {
                printf("CLIENT:Usage: client<portno><server_name>\n");
                exit(1);
        }
        if((sock=socket(AF_INET,SOCK_STREAM,0))<0)
        {
                perror("CLIENT: socket");
                exit(1);
        }
        servaddr.sin_family=AF_INET;
        portno=atoi(argv[1]);
        servaddr.sin_port=htons(portno);
        sevaddr.sin_addr.s_addr=inet_addr(argv[2]);
        if(connect(sock,(struct sockaddr*)&servaddr,sizeof(servaddr))<0)
        {
                perror("CLIENT: connect");
                exit(1);
        }
        read_frm_sock=read(sock,buffer,sizeof(bufffer));
        if(read_frm_sock<0)
        {
                perror("CLIENT: read sock");
                exit(1);
        }
        /*writing on to the standard output*/
        if(write(1,buffer,read_frm_sock)!=read_frm_sock)
        {
                perror("write stdout");
                exit(1);
        }
        exit(0);
}
```

SOCKET PROGRAM TO PERFORM TCP TIME SERVER

```
#include<stdio.h>
```

```
#include<time.h>
#include<string.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>
short portno;

main(int argc,char *argv[])
{       int sockmain,cli,i;
        char buffer[512];
        struct sockaddr_in servaddr,cliaddr;
        long t;
        char *st;
        if(argc!=2)
        {       printf("USAGE:server<portno>\n");
                exit(1);
        }
        if((sockmain=socket(AF_INET,SOCK_STREAM,0))<0)
        {       perror("socket");
                exit(1);
        }
        servaddr.sin_family=AF_INET;
        servaddr.sin_port=htons(atoi(argv[1]));
        servaddr.sin_addr.s_addr=htonl(0L);

        if(bind(sockmain,(struct sockaddr *)&servaddr,sizeof(servaddr))<0)
        {       perror("bind");
                exit(1);
        }

        if(listen(sockmain,5)<0)
        {       perror("listen"); exit(1);
        }
        for(;;)
        {       i=sizeof(cliaddr);
                if((sockcli=accept(sockmain,(struct sockaddr *)&cliaddr,&i))<0)
                {       perror("accept");       exit(1);
                }
                t=time(&t);
                st=(char*)ctime(&t);
                strcpy(buffer,st);
                i=strlen(st);
                if(write(sockcliaddr,buffer,i)!=i)
                {       perror("write socket");
                        exit(1);
                }
        }
}
```

PHP PROGRAMS

/* Finding Largest of two numbers */

```
<?php
print("Enter 1st number :");
```

```
$i=fgets(STDIN);
print("Enter 2nd number:");
$j=fgets(STDIN);

if($i>$j)
        print("Fist");
else
        print("Second");
?>
```

/* Sum of two numbers */

```
<?php
printf("Hello\n");
$i=fgets(STDIN);
$j=fgets(STDIN);
$sum=$i + $j;
echo("Enter a number :$sum");
?>
```

# Eletricity Bill Calculation
0 – 100 units Rs2/- per unit
101 – 200 units Rs3/- per unit
201 – 500 units Rs5/- per unit
>500 units Rs7/- per unit

```
<?php
print("Enter no of units ");
$n=fgets(STDIN);
if($n<=100)
{       $t=$n*2;
}
elseif($n<=200)
{       $t=200 + ($n – 100)*3;
}
elseif($n<=500)
{       $t=500 + ($n – 200)*5;
}
else
{
        $t=2000 + ($n – 200)*7;
}
print("total charges =$t");
print("\n");
?>
```

/* Switch case program */

```
<?php
print("   1. Addition of two number\n");
print(" 2. Subtraction \n");
print(" 3. Factorial of number \n");
print("\n\t\tEnter your option");
$n=fgets(STDIN);

switch($n)
{
        case 1: print("Enter first number : ");
                $x1=fgets(STDIN);
                print("Enter second number : ");
                $x2=fgets(STDIN);
                $sum=$x1 + $x2;
                print("\n The sum = $sum");
                break;

        case 2: print("Enter first number : ");
                $x1=fgets(STDIN);
                print("Enter second number : ");
                $x2=fgets(STDIN);
                $sub=$x1 - $x2;
                print("\n The sub = $sub");
                break;


        case 3: print("Enter first number : ");
                $x1=fgets(STDIN);
                $f=1;
                for($i=1;$i<=$x1;$i++)
                        $f=$f * $i;
                print("The factorial : $f");
                break;
}
?>
```

/* File print program */

```
<?php
$f=fopen("f1",r) or exit("File does not exist");
while(!feof($f))
{
        print(fgets($f)."\n");
}
fclose($f);
?>
```

Python Programs

STRINGS:-

```
string1="unixprogramming language"
print(string1[5:15])#programmin
print(type(string1))#<class 'str'>
string2=string1#copy of string1 to string2
print(string2)#unixprogramming language
str1=string2.replace("unix","linux")#replace unixto linux
print(str1)#linuxprogramming language
print(len(string1))#25
print(max(string1))#x
print(string1.capitalize())#Unix programming language
print(string1.title())#Unix Programming Language
print(string1.upper())#UNIX PROGRAMMING LANGUAGE
print(string1.lower())#unixprogramming language
print(string1.split())#['unix', 'programming', 'language']
print(string1*3)#unixprogramming languageunixprogramming languageunixprogramming language
print(string1+" mca")#unixprogramming language mca
```

LISTS:-

```
list1=[1,3,'a',1,'j',1]
print(list1[5])#j
list1.append(6)#adding element to list
print(list1)#['1', '3', 'a', '1', 'j', 6]
list2=list1.copy()#copy list1 to list2
print(list2)#['1', '3', 'a', '1', 'j', 6]
print(list1.count(1))#3
list1.extend(list2)#adding two lists
print(list1)#[1, 3, 'a', 1, 'j', 1, 6, 1, 3, 'a', 1, 'j', 1, 6]
print(list1.index('a'))#2
print(list1.pop())#6
list1.remove('j')#remove particular element
print(list1)#[1, 3, 'a', 1, 1, 6, 1, 3, 'a', 1, 'j', 1]
list1.reverse()#print in reverse order
print(list1)#[1, 'j', 1, 'a', 3, 1, 6, 1, 1, 'a', 3, 1]
list1.clear()#clear the list
print(list1)#[]
del list1#delete whole list with object also
```

TUPLES:-

```
tuple1=(1,2,3,1,2,'r','a',1)
print(tuple1[0:])#(1, 2, 3, 1, 2, 'r', 'a', 1)
print(tuple1.count(1))#3
print(tuple1.index('r'))#5
print(len(tuple1))#8
tuple2=(1,9,5,6,74,3,8,7)
print(tuple(sorted(tuple2)))#(1, 3, 5, 6, 7, 8, 9, 74)
print(min(tuple2))#1
```

SETS:-

```
s1={1,2,4,5,6,2,3,4,1,10,22,55}
print(s1)#{1, 2, 3, 4, 5, 6, 10, 22, 55}
s2=set()
s2.add(10)#ONLY ADDING FOR EMPTY SET
s2.add(20)
s2.add(90)
s2.add(10)
print(s2)#{10, 20, 90}
s3=[99,33,44,88,55]
s2.update(s3)#ADDING LIST TO SET
print(s2)#{33, 99, 10, 44, 20, 55, 88, 90}
s4=s2.copy()#COPY
print(s4)#{33, 99, 10, 44, 20, 55, 88, 90}
print(s4.pop())#REMOVE RANDOM ELEMENT
a={1,2,3,4,5,6,7,8,101,111}
```

```
a.remove(2)#REMOVE ONLY ONE ELEMENT
print(a)#{1, 3, 4, 5, 6, 7, 8, 101, 111}
a.discard(5)#REMOVE ELEMENT
a.discard(100)
print(a)#{1, 3, 4, 6, 7, 8, 101, 111}
print(s1|a)#{1, 2, 3, 4, 5, 6, 7, 8, 101, 10, 111, 22, 55}
print(s1&a)#{1, 3, 4, 6}
print(s1-a)#{2, 5, 10, 22, 55}
print(s1^a)#{2, 101, 7, 8, 5, 10, 111, 22, 55}
print(100 in a)#False
print(100 not in a)#True
sets={ x for x in range(10)}
print(sets)#{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
```

Dictionary:-

```
dict={1:'rajesh',2:'gadde',3:"karthik",4:'shanth'}
print(dict)#{1: 'rajesh', 2: 'gadde', 3: 'karthik', 4: 'shanth'}
dict[5]='shiva'#dict[KEY]=VALUE
dict[6]='sai'#adding to dict
dict[7]='prasanth'
print(dict)#{1: 'rajesh', 2: 'gadde', 3: 'karthik', 4: 'shanth', 5: 'shiva', 6: 'sai', 7: 'prasanth'}
print(dict[4])#shanth
dict[2]='vamsi'#updating a dict
print(dict)#{1: 'rajesh', 2: 'vamsi', 3: 'karthik', 4: 'shanth', 5: 'shiva', 6: 'sai', 7: 'prasanth'}
del dict[7]#delitingitem from dict
print(dict)#{1: 'rajesh', 2: 'vamsi', 3: 'karthik', 4: 'shanth', 5: 'shiva', 6: 'sai'}
#d.clear()#clear the dictand shows empty dict
#del d#deletewhole object
print(len(dict))#6
#print(d.pop(3))#remove 3rd item
```

```
#print(d.popitem())#remove random item
print(dict.keys())#dict_keys([1, 2, 3, 4, 5, 6])
print(dict.values())#dict_values(['rajesh', 'vamsi', 'karthik', 'shanth', 'shiva', 'sai'])
print(dict.items())#dict_items([(1, 'rajesh'), (2, 'vamsi'), (3, 'karthik'), (4, 'shanth'), (5, 'shiva'), (6, 'sai')])
#print(d.copy())
print(dict.setdefault(1,'mahesh'))#1 is not there take to one thposition
print(dict)#{1: 'rajesh', 2: 'vamsi', 3: 'karthik', 4: 'shanth', 5: 'shiva', 6: 'sai'}
print(dict.setdefault(10,'mahesh'))#take value to 10th position
print(dict)#{1: 'rajesh', 2: 'vamsi', 3: 'karthik', 4: 'shanth', 5: 'shiva', 6: 'sai', 10: 'mahesh'}
d={100:'r',101:'l',102:'k',103:'s'}
dict.update(d)
print(dict)#{1: 'rajesh', 2: 'vamsi', 3: 'karthik', 4: 'shanth', 5: 'shiva', 6: 'sai', 10: 'mahesh', 100: 'r', 101: 'l', 102:
'k', 103: 's'}
```

Conditionals:-

```
s1=int(input("enter s1 marks:"))
s2=int(input("enter s2 marks:"))
s3=int(input("enter s3 marks:"))
avg=(s1+s2+s3)/3
if avg>90 and avg<=100:
print("A grade")
elifavg>80 and avg<=90:
print("B grade")
elifavg>70 and avg<=80:
print("C grade")
elifavg>60 and avg<=70:
print("D grade")
elifavg>40 and avg<=60:
print("E grade")
else:
print("fails in the exam")
""“ Output:-enter s1 marks:98
enter s2 marks:56
enter s3 marks:98
B grade"""
```

Loops:-

```
for x in range(5):
print("MCA")
y=1
""" OUTPUT:-
MCA
MCA
MCA
MCA
MCA"""
while y<=7:
print("second year")
y+=1
"""OUTPUT:-
second year
second year
second year
second year
second year
second year second year”””
```

Pre-definedFunctions:-
•Print()

•Count()

•Len()

•Index()

•Max()

•Min()

•Range()

•Update()

•Sorted() and etc.

User-defined functions:-

```
def fun1():
print("welcome to python")
def fun2():
print("It is very simple")
def fun3():
print("and very easy")
fun1()
fun2()
fun3()
```

Output:-

```
welcome to python
It is very simple
and very easy
```

Pre-defined Exception:-

```
try:
a=int(input(" enter a integer:"))
b=int(input("enter another integer:"))
c=a/b
print(c)
except Exception as e:
print("exception is:",e)
else:
print("excutesuccesfully")
finally:
print("thanks")
```

Output:-

```
enter a integer:8270
enter another integer:0
exception is: division by zero
thanks
```

User-defined Exception:-

```
try:
class AgeError(Exception):
def __init__(self,msg):
self.msg=msg
age=int(input("enter age: "))
if age<=18:
raise AgeError("your age below 18 years")
elifage>=60:
raise AgeError("your age above 60 years")
else:
print("your age between 18 and 60 years")
except Exception as e:
print(e)
else:
print("thanks giving information")
finally:
print("welcome.....")
```