

### EXPERIMENT -1

#### Study of Unix/Linux general purpose utility command list: man, who, cat, cd, cp, ps, ls, mv, rm, mkdir, rmdir, echo, more, date, time, kill, history, chmod, chown, finger, pwd, cal, logout, shutdown.

**Man Command**

man command in Linux is used to display the user manual of any command that we can run on the terminal.

#### man options commandname WHO Command

who command displays information about the current status of system.

**Who *options file***

Who as default prints login names of users currently logged in.

Options

* -a use all options.
* -b Report information about last reboot.
* -d report expired processes.
* -H print headings.
* -p report previously spawned processes.
* -u report terminal usage.

#### Cat Command

cat filename

cat is used to display the contents of the file. cat> filename

cat is used to create file

#### Cd Command.

cddirectoryname

Will change directory from current directory to specified directory.

#### Cp Command

cp - copy files and directories

Copy SOURCE to DEST, or multiple SOURCE(s) to DIRECTORY.

#### cp [OPTION]... [-T] SOURCE DEST

**PS command**

ps command is probably the most useful command for systems administrators. It reports information on active processes.

Psoptions

#### Options.

-a Lists all processes in system except processes not attached to terminals.

-e Lists all processes in system.

-f Lists a full listing.

-j print process group ID and session ID.

#### Ls command

ls command is most widely used command and it displays the contents of directory. options

* ls will list all the files in your home directory, this command has many options.
* ls -l will list all the file names, permissions, group, etc in long format.
* ls -a will list all the files including hidden files that start with . .
* ls -lt will list all files names based on the time of creation, newer files bring first.
* ls -Fxwill list files and directory names will be followed by slash.
* ls -Rwilz lists all the files and files in the all the directories, recursively.
* ls -R | more will list all the files and files in all the directories, one page at a time.

#### MV Command

Rename SOURCE to DEST, or move SOURCE(s) to DIRECTORY.

#### rm Command

**mv change file name or directory location**

mv filename1 filename2

**rm** removes each specified file. By default, it does not remove directories.



#### rmoptions File name Mkdir command

**mkdir*dirname*** Makes a new directory

mkdiraditya

will create new directory, i.e. here aditya directory is created.

#### rmdirCommand

To remove a empty directory

**rmdir*dirname*** Removes directory

#### ECHO Command

**More Command**

- display a line of text

The more command is a command line utility for viewing the

contents of a file or files once screen at a time.



#### Date command.

Date displays todays date, to use it type date at prompt.

Sun Dec 7 14:23:08 EST 1997

is similar to what you should see on screen.

#### Time Command

The time command runs the specified program command with the given arguments. When command finishes, time writes a message to standard error giving timing statistics about this program run.

#### Kill Command

The command kill sends the specified signal to the specified process or process group. If no signal is specified, the TERM signal is sent.

kill - terminate a process

#### History

history command is used to view the previously executed command.

#### Chmod command

chmod command is used to change permissions on a file.

cal.txt.

initially when this file will be created the permissions for this file depends upon umask set in your profile files. As you can see this file has



666 or -rw-rw-rw attributes. ls -la cal.txt

-rw-rw-rw- 1 ssb dxidev 135 Dec 3 16:14 cal.txt

In this line above I have -rw-rw-rw- meaning respectively that owner can read and write file, member of the owner's group can read and write this file and anyone else connected to this system can read and write this file., next ssb is owner of this file dxidev is the group of this file, there are 135 bytes in this file, this file was created on December 3 at time16:14 and at the end there is name of this file. Learn to read these permissions in binary, like this for example Decimal 644 which is 110 100 100 in binary meandrw-r--r-- or user can read,write this file, group can read only, everyone else can read only.

#### Chown Command

To change the owner of file system files, directories. Chownowner\_namefile\_name

#### Finger Command

Finger command is a user information lookup command which gives details of all the users logged in. This tool is generally used by system administrators. It provides details like login name, user name, idle time, login time, and in some cases their email address even. This tool is similar to the Pinky tool but the Pinky tool is just the lightweight version of this tool

$finger

#### Pwd command.

pwd command will print your home directory on screen, pwd means print working directory.

/home/satish



#### Cal command

cal command will print the calendar on current month by default. If you want to print calendar of august of 1965. That's eighth month of 1965.

cal 8 1965 will print following results.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| S | August 1965  M Tu | | W Th | | F | S |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 8 | 9 | 10 | 11 | 12 | 13 | 14 |
| 15 | 16 | 17 | 18 | 19 | 20 | 21 |
| 22 | 23 | 24 | 25 | 26 | 27 | 28 |
| 29 | 30 | 31 |  |  |  |  |

#### Logout Command

Logout of a login shell. This command can be used by normal users to end their own session.

#### Shutdown command.

Shutdown command can only be executed by root. To gracefully bring down a system, shutdown command is used.



#### Who Command

[22A95A0518@Linux ~]$ who

|  |  |
| --- | --- |
| 22A95A0518 pts/0 | 2022-09-22 11:26 (172.7.139.250) |
| 22A9LE05074 pts/1 | 2022-09-22 11:31 (172.7.139.250) |
| 21A91A0575 pts/2 | 2022-09-22 11:32 (172.7.139.250) |
| 21A91A0576 pts/3 | 2022-09-22 11:32 (172.7.139.250) |
| 21A91A0578 pts/4 | 2022-09-22 11:30 (172.7.139.250) |
| 21A91A0580 pts/5 | 2022-09-22 11:27 (172.7.139.250) |
| 21A91A0582 pts/6 | 2022-09-22 11:28 (172.7.139.250) |
| 21A91A0583 pts/7 | 2022-09-22 11:29 (172.7.139.250) |
| 21A91A0584 pts/8  **Cat Command** | 2022-09-22 11:30 (172.7.139.250) |

[22A95A0518@Linux ~]$ cat>f1 hello welcome to unix lab

<ctrl+d>

[22A95A0518@Linux ~]$ cat f1 hello welcome to unix lab **Mkdir Command**

[22A95A0518@Linux ~]$ mkdir myfiles

**Cd Command** [22A95A0518@Linux ~]$ cd myfiles [22A95A0518@Linux myfiles]$ cd .. [22A95A0518@Linux ~]$ ls

f1 myfiles

**Remove directory containing files** [22A95A0518@Linux ~]$ mkdir myfiles [22A95A0518@Linux ~]$ cd myfiles [22A95A0518@Linux myfiles]$cat>f4 Hello

<ctrl+d>



[22A95A0518@Linux ~]$rmdir myfiles Failed to remove ’myfiles’:directory not empty [22A95A0518@Linux ~]$rm –r myfiles **Rmdir Command**

[22A95A0518@Linux ~]$ rmdirmyfiles [22A95A0518@Linux ~]$ ls

f1

**Ls Command** [22A95A0518@Linux ~]$ ls f1 myfiles

**Ps Command** [22A95A0518@Linux ~]$ ps PID TTY TIME CMD

31024 pts/0 00:00:00 bash

31368 pts/0 00:00:00 ps

**Cp Command** [22A95A0518@Linux ~]$ cp f1 f2 [22A95A0518@Linux ~]$ cat f1 hello welcome to unix lab [22A95A0518@Linux ~]$ cat f2 hello welcome to unix lab [22A95A0518@Linux ~]$ ls

f1 f2

**Rm Command** [22A95A0518@Linux ~]$ rm f2 [22A95A0518@Linux ~]$ ls

f1

**Mv Command** [22A95A0518@Linux ~]$ mv f1 f3 [22A95A0518@Linux ~]$ ls



f3

[22A95A0518@Linux ~]$ cat f1 cat: f1: No such file or directory [22A95A0518@Linux ~]$ cat f3 hello welcome to unix lab

#### Echo Command

[22A95A0518@Linux ~]$ echo 'hello how are you' hello how are you

#### More Command

[22A95A0518@Linux ~]$ more f1

**Date Command** [22A95A0518@Linux ~]$ date Thu Sep 22 12:01:42 IST 2022

[22A95A0518@Linux ~]$ date +"%d-%m-%y" 22-09-22

[22A95A0518@Linux ~]$ date "+%T" 12:04:57

[22A95A0518@Linux ~]$ date '+%B %d' September 22

#### Cal Command

[22A95A0518@Linux ~]$ cal September 2022

Su Mo Tu We ThFrSa 1 2 3

4 5 6 7 8 9 10

11 12 13 14 15 16 17

18 19 20 21 22 23 24



25 26 27 28 29 30

[22A95A0518@Linux ~]$ cal 12 2022

December 2022

Su Mo Tu We ThFrSa 1 2 3

4 5 6 7 8 9 10

11 12 13 14 15 16 17

18 19 20 21 22 23 24

25 26 27 28 29 30 31

**Time Command**

To measure the time required to run a program called date [22A95A0518@Linux ~]$ time date

Thu Sep 22 12:07:56 IST 2022

real 0m0.002s user 0m0.001s sys 0m0.000s

#### Kill Command

To kill processes simply pass the process id to the kill command. kill 4529

**History Command**

[22A95A0518@Linux ~]$ history

1. #echo$SHELL
2. echo $SHELL
3. cat /etc/shells
4. ls



1. i
2. columwq
3. :wq
4. /bin
5. /sbin
6. /etc
7. c shell:/bin/csh
8. viteja.c

**Pwd Command**

[22A95A0518@Linux ~]$ pwd

/home/22A95A0518

**Chmod Command** [22A95A0518@Linux ~]$ ls -l total 8

-rw-rw-r-- 1 22A95A0518 22A95A0518 618 Sep 22 11:58 f1

-rw-rw-r-- 1 22A95A0518 22A95A0518 26 Sep 22 11:34 f3

[22A95A0518@Linux ~]$ chmod 666 f1 [22A95A0518@Linux ~]$ ls -l

total 8

-rw-rw-rw- 1 22A95A0518 22A95A0518 618 Sep 22 11:58 f1

-rw-rw-r-- 1 22A95A0518 22A95A0518 26 Sep 22 11:34 f3

[22A95A0518@Linux ~]$ chmod 777 f3



[22A95A0518@Linux ~]$ ls -l total 8

-rw-rw-rw- 1 22A95A0518 22A95A0518 618 Sep 22 11:58 f1

-rwxrwxrwx 1 22A95A0518 22A95A0518 26 Sep 22 11:34 f3

**finger**

[22A95A0518@Linux ~]$ finger

Login Name Tty Idle Login Time Office Office Phone Host

|  |  |  |  |
| --- | --- | --- | --- |
| 20A91A0510 | pts/35 | Sep 23 06:09 | (172.7.139.250) |
| 20A91A0519 | pts/38 | 1 Sep 23 06:05 | (172.7.139.250) |
| 20A91A05C9 | pts/48 | Sep 23 06:11 | (172.7.139.250) |
| 21A91A0517 | pts/41 | 1 Sep 23 06:18 | (172.7.139.250) |
| 21A91A0518 | pts/44 | Sep 23 06:09 | (172.7.139.250) |
| 22A95A0518 | pts/21 | 37 Sep 23 06:01 | (172.7.139.250) |
| 22A95A0518 | pts/37 | 1 Sep 23 06:23 | (172.7.139.250) |
| 21A91A0575 | pts/17 | 5 Sep 23 06:01 | (172.7.139.250) |
| 21A91A0578 | pts/3 | Sep 23 06:45 | (172.7.139.250) |
| 21A91A0582 | pts/43 | Sep 23 06:07 | (172.7.139.250) |
| 21A91A0584 | pts/10 | 5 Sep 23 06:00 | (172.7.139.250) |
| 21A91A0585 | pts/4 | 4 Sep 23 06:00 | (172.7.139.250) |
| 21A91A0591 | pts/22 | 5 Sep 23 06:01 | (172.7.139.250) |
| 21A91A0594 | pts/11 | 1 Sep 23 06:01 | (172.7.139.250) |
| 21A91A0599 | pts/30 | Sep 23 06:03 | (172.7.139.250) |



|  |  |  |  |
| --- | --- | --- | --- |
| 21A91A05A5 | pts/26 | Sep 23 06:03 | (172.7.139.250) |
| 21A91A05A7 | pts/40 | 1 Sep 23 06:06 | (172.7.139.250) |
| 21A91A05B1 | pts/16 | 1 Sep 23 06:01 | (172.7.139.250) |
| 21A91A05B2 | pts/51 | Sep 23 06:18 | (172.7.139.250) |
| 21A91A05B3 | pts/9 | 4 Sep 23 06:00 | (172.7.139.250) |



#### Study of Bash shell, Bourne shell and C shell in Unix/Linux operating system

* + A shell is an environment in which we can run our commands, programs, and shell scripts.
  + The shell provides you with an interface to the UNIX system.
  + Shell categories
    - Bourne shell ( sh)
    - Korn shell ( ksh)
    - Bourne Again shell ( bash)
    - POSIX shell ( sh)

To know which shell we are presently in [root@Linux~]#echo $SHELL

To know all shell with paths [root@Linux ~]#cat /etc/shells

/bin/sh

/bin/bash

/sbin/nologin

/bin/tcsh

/bin/csh

/bin/zsh

#### Bourne shell

Steve Bourne created Bourne shell most popular one. This one is used by many UNIX users.

/Bourne shell: /bin/sh shell prompt : $

#### C Shell

Bill Joy Created C shell.

Two advantages over Bourne shell

1. It allows aliasing of commands. It is useful when lengthy commands are renamed by you.
2. C shell has a command history feature. C shell keeps track of all commands issued in the command line.

C shell: /bin/csh Shell prompt :% **Korn shell**

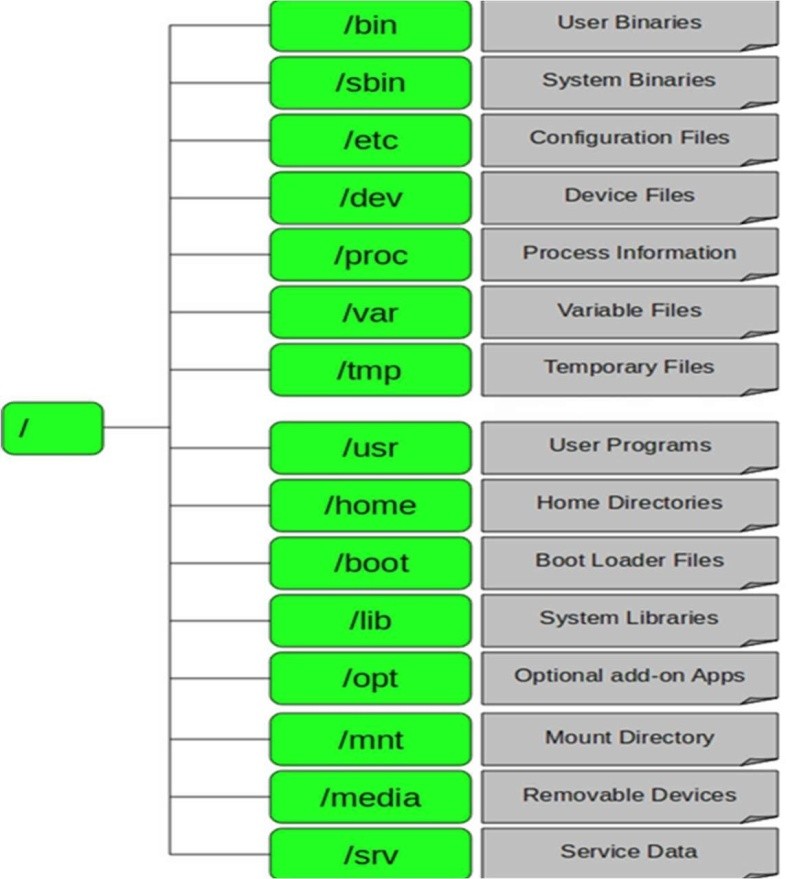
David Korn Created Korn shell at AT&T labs



It is very powerful and is a superset of Bourne shell. Korn shell: /bin/ksh

Shell prompt :$

**Bash** – the Bourne again shell was developed by GNU project .It is based on features of C and K shells.



#### Study of Unix/Linux file system (tree structure).

1. /bin

The /bin directory is for User Binaries. It is where many of the most common Linux commands are stored. Specifically, this is where the single user mode binaries are stored.

Example: ls,cp,vietc

1. /sbin

This directory is almost exactly like the /bin directory, with one exception. The binaries here are primarily used by Administrators for system maintenance.

1. /etc

The configuration files for your programs and operating system are stored in /etc. Example: Java JVM

1. /dev

This is where all of the device files are located. For example, this is the directory that you would call to in order to mount a drive with a command like: mount /dev/sda2 /mnt/backup

1. /proc

The /proc directory is one of the most interesting in the whole Linux File System. It is actually its own virtual file system with a massive amount of text information about system processes.



1. /var

This is where all of the variable files are stored. Most commonly, this is where log files and web server files are stored.

1. /tmp

These are simply temporary files.

1. /usr

Programs installed by single users get stored here.

1. /home

This is where all of the user home directories are except for the root user’s home directory which is /root.

1. /boot

The files that make up the boot loader go in /boot. Everything from boot loader menus, to the actual kernel files are stored here.

1. /lib

All of the binary files that are located in /bin and /sbin are supported by the library files located in /lib.

1. /opt

/opt is short for “optional”. It is the directory where individual vendors can install optional add-on software for the operating system.

1. /mnt

The /mnt directory is the mount point that system administrators can use to mount file systems temporarily.

1. /media

The /media directory serves the same purpose as the /mnt directory except it is specifically for removable devices and can be used by non administrators.

1. /srv

The /srv directory contains server specific service files.



#### Study of .bashrc, /etc/bashrc and Environment variables.

The /etc/bashrc is executed for both interactive and non-interactive shells. /etc/bashrc or

/etc/bash.bashrc is the systemwide bash per-interactive-shell startup file. Is is used system wide functions and aliases. However, environment stuff goes in /etc/profile file.the

/etc/profile is executed only for interactive shells

.bashrc is a shell script that Bash runs whenever it is started interactively. It initializes an interactive shell session.

.bashrc runs on every interactive shell launch.

Following is the partial list of important environment variables :-

DISPLAY : Contains the identifier for the display that X11 programs should use by default.

HOME : Indicates the home directory of the current user: the default argument for the cd built-in command.

IFS : Indicates the Internal Field Separator that is used by the parser for word splitting after expansion.

LANG : LANG expands to the default system locale; LC\_ALL can be used to override this. For example, if its value is pt\_BR, then the language is set to (Brazilian) Portuguese and the locale to Brazil.

LD\_LIBRARY\_PATH : On many Unix systems with a dynamic linker, contains a colonseparated list of directories that the dynamic linker should search for shared objects when building a process image after exec, before searching in any other directories.

PATH : Indicates search path for commands. It is a colon-separated list of directories in which the shell looks for commands.

PWD : Indicates the current working directory as set by the cd command.



RANDOM : Generates a random integer between 0 and 32,767 each time it is referenced.

SHLVL : Increments by one each time an instance of bash is started. This variable is useful for determining whether the built-in exit command ends the current session.

TERM : Refers to the display type

VZ : Refers to Time zone. It can take values like GMT, AST, etc.

UID : Expands to the numeric user ID of the current user, initialized at shell startup.

## Viva Questions

1. **What is the purpose of echo command?**

**A.** It is used to print a statement.

1. **What are the types of shells?**

**A.** bourne shell , korn shell, bourne again shell, c shell etc.

1. **where the device files will be located?**

**A.** Those are located /dev environmental variable.



### EXPERIMENT – 2

#### 2.a)Use the cat command to create a file containing the following data. Call it mytable use tabs to separate the fields.

**Vi Editor**

An editor allows the users to see a portion of a file on the screen and to modify characters and lines by simply typing at the cursor position.

Vi editor

* Vi stands for visual
* It is a full screen editor and allows the user to view and edit the entire document at the same time.
* Vi is case sensitive.
* It has powerful undo features.

Modes of Vi editor

Vi editor works in 3 modes

#### Command mode

In this mode all the keys pressed by the user are interpreted to be editor commands. No text is displayed on the screen even if corresponding keys is pressed on the keyboard.

#### Insert mode

This mode permits to insert a new text, editing and replacement of existing text. When Vi editor is in insert mode the letters typed at the keyboard are echoed on the screen.

#### Escape mode

Commands typed at the command line. Starting with Vi editor

**Syntax:** vi filename Moving the cursor

The cursor movement commands are:

Command Action

H or backspace Left one character l or spacebar Right one character

K or - Up one line

J or + Down one line

I Moves forward a word #b Moves back a word

#e Moves to the last character in the word

F[character] Moves right to the specified character in a line

T[character] Moves right and places it one character before the specified character 0 or ^ Moves to the beginning of the file

#$ Moves to the end of the file Quitting vi



Press zz or ‘:wq’ in command mode.

#### Sort Command

sort -o: if you want to write the **output to a new file**, output.txt, redirects the output like this or you can also use the built-in sort option -o, which allows you to specify an output file.

Syntax:

$ sort inputfile.txt > filename.txt

Using the -o option is functionally the same as redirecting the output to a file.

$ sort -o filename.txt inputfile.txt

|  |  |  |
| --- | --- | --- |
| 1425 | Ravi | 15.65 |
| 4320 | Ramu26.27 | |
| 6830 | Sita 36.15 | |
| 1450 | Raju 21.86 | |

[22A95A0518@Linux~]$cat>mytable

|  |  |  |
| --- | --- | --- |
| 1425 | Ravi | 15.65 |
| 4320 | Ramu26.27 | |
| 6830 | Sita 36.15 | |
| 1450 | Raju 21.87 | |

c)Use the cat command to display the file table [22A95A0518@Linux ~]$ cat mytable

|  |  |  |
| --- | --- | --- |
| 1425 | Ravi | 15.65 |
| 4320 | Ramu26.27 | |
| 6830 | Sita 36.15 | |
| 1450 | Raju 21.86 | |

**c)** Use the vi command to correct any errors in the file, mytable.

[22A95A0518@Linux ~]$ vi mytable 1425 Ravi 15.65



|  |  |  |
| --- | --- | --- |
| 4320 | Ramu | 26.27 |
| 6830 | Sita | 36.15 |
| 1450  ~ | Raju | 21.86 |
| :wq |  |  |

[22A95A0518@Linux~]$vimytable

|  |  |  |
| --- | --- | --- |
| 1425 | Ravi | 15.65 |
| 4320 | Ramu | 26.27 |
| 6830 | Sita | 36.15 |
| 1450 | Raju | 21.86 |
| 5567 | Geeta | 56.23 |
| **~** |  | |
| **~** |
| **~** |
| **~** |
| **~**  :dd |

[22A95A0518@Linux~]$vimytable

|  |  |  |
| --- | --- | --- |
| 1425 | Ravi | 15.65 |
| 4320 | Ramu | 26.27 |
| 6830 | Sita | 36.15 |
| 1450 | Raju | 21.86 |
| 5567 | Geeta | 56.23 |

**c)** Usethesortcommand tosortthefilemytableaccordingtothefirstfield.Callthesorted file my table(same name)

[22A95A0518@Linux~]$sortmytable 1425 Ravi 15.65



|  |  |  |
| --- | --- | --- |
| 1450 | Raju | 21.86 |
| 4320 | Ramu26.27 | |
| 5567 | Geeta56.23 | |
| 6830 | Sita 36.15 | |

[21A9A05D3@Linux~]$sort-rmytable

|  |  |  |
| --- | --- | --- |
| 6830 | Sita | 36.15 |
| 5567 | Geeta | 56.23 |
| 4320 | Ramu | 26.27 |
| 1450 | Raju | 21.86 |
| 1425 | Ravi | 15.65 |

[21A9A05D3@Linux~]$sort–oppmytable

|  |  |  |
| --- | --- | --- |
| 1425 | Ravi | 15.65 |
| 1450 | Raju | 21.86 |
| 4320 | Ramu26.27 | |
| 5567 | Geeta56.23 | |
| 6830 | Sita 36.15 | |

1. Print the file my table [22A95A0518@Linux~]$catmyt

|  |  |  |
| --- | --- | --- |
| able |  | |
| 1425 | Ravi | 15.65 |
| 4320 | Ramu | 26.27 |
| 6830 | Sita | 36.15 |
| 1450 | Raju | 21.86 |
| 6830 | Sita | 36.15 |



## Viva Questions

* 1. **Which command is used to display the file?**

**A.** cat command is used to display the file.

* 1. **which command is used to sort the table?**

**A.** sort command is used to display the file.

* 1. **What is the use of vi editor?**

**A.** It is used to edit the shell files.



### EXPERIMENT-3

1. **Write a C program that makes a copy of a file using standard I/O, and system call**

### Description:

#### Syntax

fd = open (filename, flags);

#### Flags

**O\_RDONLY**: read only **O\_WRONLY**: write only **O\_RDWR**: read and write

**O\_CREAT**: create file if it doesn’t exist **O\_EXCL**: prevent creation if it already exists **Returns:**

* return first unused file descriptor
* return -1 when error

File descriptor is integer that uniquely identifies an open file of the process.

read() function reads cnt bytes of input into the memory area indicated by buf. A successful read() updates the access time for the file.

#### Syntax

size\_t read (int fd, void\* buf, size\_t cnt);

**fd:** file descriptor

**buf:** buffer to read data from

**cnt:** length of buffer

#### Returns: How many bytes were actually read

* return Number of bytes read on success
* return 0 on reaching end of file
* return -1 on error

**write:** Writes cnt bytes from buf to the file. If cnt is zero, write() simply returns 0 without attempting any other action.

#### Syntax

size\_t write (int fd, void\* buf, size\_t cnt);

**fd:** file descriptor

**buf:** buffer to write data to

**cnt:** length of buffer

#### Returns: How many bytes were actually written

* return Number of bytes written on success
* return 0 on reaching end of file
* return -1 on error



### Algorithm:

**STEP 1:** START

**STEP 2:** Declare 2 integer variables f1 ,f2 and one character array , 1 long variable n;

**STEP 3:** IF (((f1 = open(argv[1], O\_RDONLY)) == -1 || ((f2=open(argv[2], O\_CREAT | O\_WRONLY | O\_TRUNC, 0700))== 1)))

If the above condition satisfies display there is a problem in file then exit.

**STEP 4:** WHILE ((n=read(f1, buff, 50))>0)

**STEP 5**: IF (write(f2, buff, n)!=n) then Print “There is a problem in writing”

**STEP 6:** IF n=-1 then display “There is a problem in writing”

**STEP 7:** close the file f2

**STEP 8:** STOP

### Program:

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

int main(int argc, char \*argv[])

{

int f1, f2; char buff[50]; long int n;

if(((f1 = open(argv[1], O\_RDONLY)) == -1 || ((f2=open(argv[2], O\_CREAT | O\_WRONLY | O\_TRUNC, 0700))== 1)))

{

perror("problem in file"); exit(1);

}

while((n=read(f1, buff, 50))>0) if(write(f2, buff, n)!=n)

{

perror("problem in writing"); exit(3);

}

if(n==-1)

{

perror("problem in reading");



exit(2);

}

close(f2);

}

### Output:

[22A95A0518@Linux ~]$ cat>f1 This is unix lab [22A95A0518@Linux ~]$ cat>f2 [22A95A0518@Linux ~]$ cc 3.a.c

[22A95A0518@Linux ~]$ ./a.out f1 f2 [22A95A0518@Linux ~]$ cat f2

This is unix lab



1. **Write a C program to emulate the UNIX ls –l command.**

### Description:

Fork system call is used for creating a new process, which is called ***child process***, which runs concurrently with the process that makes the fork() call (parent process). It takes no parameters and returns an integer value.

#### Syntax

fork()

***Negative Value***: creation of a child process was unsuccessful.

***Zero***: Returned to the newly created child process.

***Positive value***: Returned to parent or caller.

The execlp() function replaces the current process image with a new process image specified by file.

#### Syntax

#include <unistd.h>

int execlp( const char \* file, const char \* arg0, const char \* arg1,

…

const char \* argn, NULL );

The wait() system call **suspends execution of the current process until one of its children terminates**.

### Algorithm:

**STEP 1:** START

**STEP 2:** Include the needed header files

**STEP 3**: Declare pid which stores the process id

**STEP 4:** Create a child using fork() ans assign the returned value into pid

**STEP 5:** IF pid <0 PRINT “fork failed” then exit

**STEP 6:** ELSE IF pid=0 execute ls –l command using execlp

**STEP 7:** ELSE PRINT “Wait for child complete”

**STEP 8**: STOP



### Program:

#include <stdio.h> #include <unistd.h> #include <sys/types.h> #include <sys/wait.h> #include <stdlib.h>

int main()

{

int pid; //process id

pid = fork(); //create another process if ( pid< 0 )

{ //fail

printf(“\nFork failed\n”); exit (-1);

}

else if ( pid == 0 )

{ //child

execlp( “/bin/ls”, “ls”, “-l”, NULL ); //execute ls

}

else

{ //parent

wait (NULL); //wait for child printf(“\nchild complete\n”); exit (0);

}

}

### Output:

[22A95A0518@Linux ~]$ vi 3b.c [22A95A0518@Linux ~]$ cc 3b.c



[22A95A0518@Linux ~]$ ./a.out total 176

-rw-rw-r-- 1 22A95A0518 22A95A0518 316 Dec 9 06:39 2

-rw-rw-r-- 1 22A95A0518 22A95A0518 0 Dec 19 07:47 3.a

-rw-rw-r-- 1 22A95A0518 22A95A0518 537 Dec 19 07:49 3.a.c

-rw-rw-r-- 1 22A95A0518 22A95A0518 537 Dec 19 07:48 3.a.sh

-rw-rw-r-- 1 22A95A0518 22A95A0518 429 Dec 19 08:03 3b.c

drwxrwxr-x 2 22A95A0518 22A95A0518 4096 Dec 9 06:49 ab

-rwxrwxr-x 1 22A95A0518 22A95A0518 5118 Dec 19 08:03 a.out

-rw-rw-r-- 1 22A95A0518 22A95A0518 158 Oct 28 06:52 arg.sh

-rw-rw-r-- 1 22A95A0518 22A95A0518 69 Oct 21 07:01 arith.sh

-rw-rw-r-- 1 22A95A0518 22A95A0518 249 Oct 28 07:49 binary.sh

-rw-rw-r-- 1 22A95A0518 22A95A0518 779 Oct 28 07:49 bin.sh

-rw-rw-r-- 1 22A95A0518 22A95A0518 137 Dec 9 07:47 check.sh

-rw-rw-r-- 1 22A95A0518 22A95A0518 408 Dec 9 07:29 copy.c

-rw-rw-r-- 1 22A95A0518 22A95A0518 103 Dec 2 06:12 c.sh

drwxrwxr-x 2 22A95A0518 22A95A0518 4096 Dec 9 07:07 d1

-rw 1 22A95A0518 22A95A0518 1984 Nov 18 07:20 dead.letter

-rw-rw-r-- 1 22A95A0518 22A95A0518 262 Dec 9 07:07 eleven.c

-rw-rw-r-- 1 22A95A0518 22A95A0518 0 Dec 9 06:33 eleven.cc

-rw-rw-r-- 1 22A95A0518 22A95A0518 32 Nov 18 07:04 equal

-rw-rw-r-- 1 22A95A0518 22A95A0518 17 Dec 19 08:00 f1



1. **Write a C program that illustrates how to execute two commands concurrently with a command pipe. Ex: - ls –l | sort.**

### Description:

Pipe() system call

a pipe is a connection between two processes, such that the standard output from one process becomes the standard input of the other process.

Syntax

#### int pipe(int fds[2]);

**Parameters :**

**fd[0]** will be the fd(file descriptor) for the read end of pipe.

**fd[1]** will be the fd for the write end of pipe.

**Returns :** 0 on Success.

**-1** on error.

The dup() system call creates a copy of a file descriptor.

#### Syntax:

**int dup(int oldfd);**

**oldfd:** old file descriptor whose copy is to be created.

### Algorithm:

**STEP 1:** START

**STEP 2:** Create an integer array pfds and character array buff **STEP 3:** IF (pipe(pfds)==-1) Print “Pipe failed” then exit **STEP 4:** IF (!fork()) Then close 1 duplicate pfds[1]

**STEP 5:** Else Print “ parent reading from pipe”

**STEP 6**: Then print Buff

**STEP 7:** while(read(pfds[0],buf,80)) Repeat STEP 6

**STEP 8**: STOP

### Program:

#include <stdio.h> #include <unistd.h> #include <sys/types.h> #include <stdlib.h>

int main()



{

int pfds[2]; char buf[30];

if(pipe(pfds)==-1)

{

perror("pipe failed"); exit(1);

}

if(!fork())

{

close(1);

dup(pfds[1]; system (“ls –l”);

}

else

{

printf("parent reading from pipe \n"); while(read(pfds[0],buf,80)) printf("%s \n" ,buf);

}

}

### Output:

[22A95A0518@Linux ~]$ cc command.c [22A95A0518@Linux ~]$ ./a.out

total 200

-rwxrwxr-x 1 22A95A0518 22A95A0518 5493 Dec 2 08:29 a.out

-rw-rw-r-- 4

total 200

-rwxrwxr-x 1 22A95A0518 22A95A0518 5493 Dec 2 08:29 a.out

-rw-rw-r-- 4

total 200

-rwxrwxr-x 1 22A95A0518 22A95A0518 5493 Dec 2 08:29 a.out

-rw-rw-r-- 4

total 200

-rwxrwxr-x 1 22A95A0518 22A95A0518 5493 Dec 2 08:29 a.out

-rw-rw-r-- 4^Z

[5] + Stopped ./a.out



## Viva Questions

* 1. **What is the purpose of open command?**

**A.** It is used to open a file.

* 1. **which command is used to display the list of files?**

**A.** ls –l is used to display the list of files.

* 1. **What is the purpose of pipe command?**

**A.**  It is used to combine two commands . The first command output is input for second command.



#### EXPERIMENT -4

* + 1. **Write a shell script that takes a command –line argument and reports on whether it is directory, a file or something else.**

**Description:**

The **if...elif...fi** statement is the control statement that allows Shell to make correct decision out of several conditions.

Syntax

if [ expression 1 ] then

Statement(s) to be executed if expression 1 is true elif [ expression 2 ]

then

Statement(s) to be executed if expression 2 is true elif [ expression 3 ]

then

Statement(s) to be executed if expression 3 is true else

Statement(s) to be executed if no expression is true

fi

The “if’ keyword is followed by the condition you want to check. In this if-else-if conditional statement, the expressions are evaluated from top to bottom.

### Algorithm:

**STEP 1:** START

**STEP 2:** PRINT “Enter file name: ”

**STEP 3**: READ file name into variable str

**STEP 4:** IF test -f $str Then display a message indicating it is a file

**STEP 5:** ELSE IF test -d $str Then display a message indicating it is a directory

**STEP 6:** ELSE Print “File not exists”

**STEP 8:** END

### Program:

echo " enter file" read str

if test -f $str



then echo "file exists n it is an ordinary file" elif test -d $str

then echo "directory file" else

echo "not exists" fi

### Output:

[root@localhost ~]# ls

4a.sh bench.py d1 f1 hello.c

[root@localhost~]#sh 4a.sh Enter File:

f1

file exists n it is an ordinary file [root@localhost~]#sh 4a.sh Enter File:

d1

directory file



* + 1. **Write a shell script that accepts one or more file name as arguments and converts all of them to uppercase provided they exist in the current directory.**

### Description:

Command-line arguments are parameters that are passed to a script while executing them in the shell. The shell has special variables reserved to point to the arguments which we pass through a shell script. Bash saves these variables numerically ($1, $2, $3, … $n).

Here, the first command-line argument in our shell script is $1, the second $2 and the third is

$3. This goes on till the 9th argument. The variable $0 stores the name of the script or the command itself.

#### For loop

The **for** loop operates on lists of items. It repeats a set of commands for every item in a list. for var in word1 word2 ... wordN

do

Statement(s) to be executed for every word. done

Here *var* is the name of a variable and word1 to wordN are sequences of characters separated by spaces (words). Each time the for loop executes, the value of the variable var is set to the next word in the list of words, word1 to wordN.

### Algorithm:

**STEP 1:** START

**STEP 2:** Print “Enter File Name:”

**STEP 3:** Read file name as ‘filename’

**STEP 4:** if (!-f$fileName) print “FileName doesn’t exist”

**STEP 5:** Convert uppercase to lowercase using tr tr '[a-z]' '[A-Z]' < $fileName

**STEP 6**: STOP

### Program:

# get filename

echo -n "Enter File Name : " read fileName

# make sure file exits for reading if [ ! -f $fileName ]



then

echo "Filename $fileName does not exists" exit 1

fi

# convert uppercase to lowercase using tr command

tr '[a-z]' '[A-Z]' < $fileName

### Output:

[root@localhost~]#sh 4b.sh Enter File Name:f1

THIS IS UNIX LABORATORY

## Viva Questions

1. **What is meant by command line arguments?**

**A.** The values which are passed during the execution of the program .

1. **What is the purpose of tr command?**

**A.** It is used to translate characters.

1. **List some conditional statements?**

**A.** simple if , if – else , else – if ladder , case statements

,



#### EXPERIMENT – 5

1. **Write a shell script that determines the period for which a specified user is working on the system**

**Description:**

Grep global regular expression print

It matches the given string/pattern even if it is found as a substring in a file. The -w option to grep makes it match only the whole words.

grep [options][pattern][file\_name]

To cut by character use the -c option. This can be a list of numbers separated comma or a range of numbers separated by hyphen(-).

Syntax: cut –c [starting\_position\_of\_character,ending\_position\_of\_character] filename

### Algorithm:

**STEP 1:** START

**STEP 2:** PRINT “Enter the Username:”

**STEP 3:** READ usr

**STEP 4:** Assign tuser=`who | tr -s " " | head -1 | cut -d " " -f1`

**STEP 5:** if (tuser = usr)

Then

Assign tm=`who | tr -s " " | head -1 | cut -d " " -f4` Assign uhr=`echo $tm | cut -d ":" -f1`

Assign umin=`echo $tm | cut -d ":" -f2` Assign shr=`date "+%H`

Assign smin=`date "+%M”`

**STEP 6:** if(smin<umin) then Assign shr=shr – 1 Assign smin=smin + 60 End if

**STEP 7:** Assign h=shr - uhr

**STEP 8**:m=smin – umin



STEP 9:Print “User Name:” usr Print “Login period:” h,m

**STEP 10:** else Print “Invalid User” End if

**STEP 11:** STOP

### Program:

echo -e "enter the user name :\c" read usr

tuser=`who | tr -s " " | head -1 | cut -d " " -f1` if [ "$tuser" = "$usr" ]

then

tm=`who | tr -s " " | head -1 | cut -d " " -f4` uhr=`echo $tm | cut -d ":" -f1` umin=`echo $tm | cut -d ":" -f2`

shr=`date "+%H"` smin=`date "+%M"` if [ $smin -lt $umin ] then

shr=`expr $shr - 1` smin=`expr $smin + 60` fi

h=`expr $shr - $uhr` m=`expr $smin - $umin` echo "user name : $usr" echo "login period : $h : $m" else

echo "Invalid User" fi

### Output:

$ sh working.sh

enter the user name :22A95A0518 user name : 22A95A0518

login period : 0 : 51

$ sh working.sh

enter the user name :22A95A0519 Invalid User



1. **Write a shell script that accepts a file name starting and ending line numbers as arguments and displays all the lines between the given line numbers.**

### Description:

Sed stands for Stream Editor.Sed helps in operations like selecting the text, substituting text, modifying an original file, adding lines to text, or deleting lines from the text.

The **-n** option disables automatic printing, while the substitute command **p** instructs **sed** to print lines where substitution occurs.

Sed OPTIONS [SCRIPT] [Inputfile]

Pipe is **used to combine two or more commands**, and in this, the output of one command acts as input to another command, and this command's output may act as input to the next command and so on. It can also be visualized as a temporary connection between two or more commands.

The output redirection operator is the > (greater than) symbol, and the general syntax looks as follows:

command >output\_file ex:-ls>my\_files

Notice there is no output appearing after the command, only the return of the prompt. This is because all output from this command was *redirected* to the file my\_files.

### Algorithm:

STEP 1:START

STEP 2: PRINT “Enter the File Name” STEP 3: READ fname

STEP 4: PRINT “Enter the Starting line number” STEP 5: READ s

STEP 6: PRINT “Enter the Ending line number ” STEP 7: READ n

STEP 8: sed -n $s,$n\p $fname | cat > newline STEP 9: DISPLAY FILE newline

cat newline STEP 10: STOP



### Program:

echo "enter the filename" read fname

echo "enter the starting line number" read s

echo "enter the ending line number" read n

sed -n $s,$n\p $fname | cat > newline cat newline

**Output:**

[root@localhost ~]# sh 5b.sh Enter the filename:

file

Enter the starting line number:

2

Enter the ending line number:

5

hello swaroop rohith ram sita

## Viva Questions

* 1. **Which command is used for searching a given stirng?**

**A.** GREP command is used for searching a given stirng.

* 1. **What is the purpose of sed command?**

**A.** It is used to retrieve particular portion of lines from the file.

* 1. **What are the options of grep command?**

**A.** –n , -i , -e , -b , -l etc…



#### EXPERIMENT – 6

**Write a shell script that computes the gross salary of a employee according to the following rules:**

#### If basic salary is < 1500 then HRA =10% of the basic and DA =90% of the basic.

1. **If basic salary is >=1500 then HRA =Rs500 and DA=98% of the basic. The basic salary is entered interactively through the key board.**

**Description:**

The **if...else...fi** statement is the next form of control statement that allows Shell to execute statements in a controlled way and make the right choice.

**S**yntax

if [ expression ] then

Statement(s) to be executed if expression is true else

Statement(s) to be executed if expression is not true Fi

The Shell *expression* is evaluated in the above syntax. If the resulting value is *true*,

given *statement(s)* are executed. If the *expression* is *false*, then no statement will be executed.

#### 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.

But we can also perform mathematical operations in the Linux terminal using **bc** without entering the interactive shell. For this process, we’ll need to utilize the **echo** command and **|** pipes. This will echo any text we type and pass it to the **bc** command.

Ex:$ echo 1+1|bc

### Algorithm:

STEP 1: START

STEP2: PRINT "Enter basic Salary:" STEP 3: READ bsal

STEP 4: if (bsal<1500)

THEN EVALUATE gsal=((bsal+((bsal/100)\*10)+(bsal/100)\*90))



PRINT "The gross Salary:" PRINT gsal

END IF

STEP 5:if( bsal>1500)

THEN EVALUATE gsal=((bsal+500)+(bsal/100)\*98)) PRINT "The gross Salary:"

PRINT gsal END IF

STEP 6: STOP

### PROGRAM:

echo "enter the basic salary:" read bsal

if [ $bsal -lt1500 ] then

gsal=$((bsal+((bsal/100)\*10)+(bsal/100)\*90)) echo "The gross salary : $gsal"

fi

if [ $bsal -ge1500 ] then

gsal=$(((bsal+500)+(bsal/100)\*98)) echo "the gross salary : $gsal"

fi

### Output:

[root@localhost ~]# sh 6.shEnter the basic salary:

1500

The gross salary:3470

## Viva Questions

* 1. **How the basic calculator can be used in the program?**

**A.** It can be used in echo statement like echo 2+4|bc.

* 1. **How the if statement works?**

**A.** First condition will be checked , if it is true block will be executed otherwise false block

will be executed.

* 1. **How the gross salary is calculated?**

**A.** gross salary = bsa + hra.



#### EXPERIMENT – 7

1. **Write a shell script that accepts two integers as its arguments and computes the value of first number raised to the power of the second number.**

**Description:**

$# Stores the number of command-line arguments that were passed to the shell program.

-ne is not equal to

$ echo “$1^$2 | bc”

To Compute $1 to the power $2

exit command is used *to exit the shell where it is currently running*.

### Algorithm:

STEP 1: START

STEP 2: PRINT "Enter the integer value: " STEP 3 : READ VALUE TO int1

STEP4: PRINT "Enrer the power of that integer:" STEP 5: READ VALUE TO int2

STEP 6: ASSIGN pv=int1

i=1

STEP 7: pv= pv\* int1 i= i+1

STEP 8: REPEAT STEP 7 UNTILL i<int2

STEP 9: PRINT "The value of first number to the power of second number is : " STEP 10: PRINT pv

STEP 11: STOP

### PROGRAM:

echo "Enter the integer value :" read int1

echo "Enter the power of that integer:" read int2

pv=$int1



i=1

while [ $i -lt $int2 ] do

pv=`expr $pv \\* $int1` i=`expr $i + 1 `

done

echo "The value of first number to the power of the second number :" echo "$pv"

### Output:

[root@localhost ~]# sh 7a.c Enter the integer value:

2

Enter the power of that integer:

3

The value of first number to the power of the second number:



1. **Write a shell script which will display Armstrong number from given arguments.**

### Description:

**Armstrong number** is *a number that is equal to the sum of cubes of its digits*. For example 0, 1, 153, 370, 371 and 407 are the Armstrong numbers.

$\* Stores all the arguments that were entered on the command line ($1 $2 ...).

The **while** loop enables you to execute a set of commands repeatedly until some condition occurs.

while command do

Statement(s) to be executed if command is true done

Here the Shell command is evaluated. If the resulting value is true, given statement(s) are executed. If command is false then no statement will be executed and the program will jump to the next line after the done statement.

### Algorithm:

STEP 1: START

STEP 2: PRINT "Enter the number:" STEP 3: READ n

STEP 4: ASSIGN t=n, s=0, b=0, c=10 STEP 5: CALCULATE

r=n % c i=r \* r \* r

s=s + i n=n / c

STEP 6: REPEAT STEP 5 UNTIL n>b STEP 7: PRINT s

STEP 8: if (s=t)

THEN PRINT "ARMSTRONG NUMBER"

ELSE THEN PRINT "NOT A ARMSTRONG NUMBER" END IF

STEP 9: ASSIGN result=n

PRINT result STEP 10: STOP



### Program:

echo "Enter the number" read n

function ams

{

t=$n s=0 b=0 c=10

while [ $n -gt $b ] do

r=$((n % c))

i=$((r \* r \* r))

s=$((s + i))

n=$((n / c)) done

echo $s

if [ $s == $t ] then

echo "Amstrong number" else

echo "Not an Armstrong number" fi

}

result=`ams $n` echo "$result"

### Output:

[root@localhost ~]# sh 7b.c Enter the number:

153

Armstrong Number

## Viva Questions

* 1. **What is the syntax of while loop?**

**A.** while command

do

Statement(s) to be executed if command is true

done



* 1. **What is the purpose of expr command?**

**A.** It is used to evaluate a expression.

* 1. **What is meant by an Armstrong number?**

**A.** Armstrong number is a number that is equal to the sum of cubes of its digits.



#### EXPERIMENT – 8

**Write an interactive file-handling shell program. Let it offer the user the choice of copying, removing, renaming, or linking files. Once the user has made a choice, have the program ask the user for the necessary information, such as the file name, new name and so on**

**Description:**

The basic syntax of the **case...esac** statement is to give an expression to evaluate and to execute several different statements based on the value of the expression.

The interpreter checks each case against the value of the expression until a match is found. If nothing matches, a default condition will be used.

case word in pattern1)

Statement(s) to be executed if pattern1 matches

;;

pattern2)

Statement(s) to be executed if pattern2 matches

;;

pattern3)

Statement(s) to be executed if pattern3 matches

;;

\*)

Default condition to be executed

;;

esac

**ln command** is a Unix command for linking files to each other.

It creates new files with the names you specify, and refer them to already existing files syntax:

ln sourcefile destinationfile

### Algorithm:

**STEP 1:** START

**STEP 2:** Display the operations to be performed **STEP 3:** Then print “Enter your choice: ” **STEP 4:** read Choice from user

**STEP 5**: if choice is 1 Then list files using ls **STEP 6:** if choice is 2 Then copy files using cp **STEP 7:** if choice is 3 Then remove file using rm



**STEP 8:** if choice is 4 Then rename files using mv **STEP 9:** if choice is 5 Then link files using ln **STEP 10:** Else display invalid choice

**STEP 11**: Repeat the above steps until user enter [CTRL+C] TO EXIT"

**STEP 12:** STOP

### Program:

while true do

echo "\*\*\*\*\*\*\*MENU\*\*\*\*\*\*\*\*\*" echo "

1. List of files.
2. Copying files.
3. Removing files.
4. Renaming files.
5. Linking files.

press [CTRL+C] TO EXIT"

echo "enter your choice " read ch

case "$ch" in

1 ) echo "The list of file names." ls -l || echo "These are file";;

2) echo "Enter the old filename." read ofile

echo "Enter the new file name." read nfile

cp $ofile $nfile&& echo "Copied sucessfully." || echo "Copied is not possible." ;; 3 ) echo "Enter the file name to remove."

read rfile

rm -f $rfile&& echo "Successfully removed." ;;



1. ) echo "Enter the old file name." read ofile

echo "Enter the new file name." read nfile

mv $ofile $nfile&& echo "The file $ofile name renamed to $nfile." || echo "You cann't Rename the file.".;;

1. ) echo "Enter the original filename." read ofile

echo "Enter the new filename to link a file." read lfile

ln $ofile $lfile&& echo "Creat the linking file Sccessfully." || echo "You cann't Linking the file.";; \* )

echo "Invalid option."

Echo " Enter correct choice." esac

done

### Output:

\*\*\*\*\*\*\*\*\*MENU\*\*\*\*\*\*\* 1.List of files.

2.copying files. 3.Removing files. 4.Renaming file. 5.Linking file.

press [CTRL+c] to EXIT enter your choices

1

the llist of file names. total 20

-rw-r--r-- 1 root root 114 Dec 26 2020 bench.py



-rw-r--r-- 1 root root 185 Sep 9 2018 hello.c

-rw-r--r-- 1 root root 14 Dec 30 21:49 rohith

-rw-r--r-- 1 root root 960 Dec 30 21:48 rohith.sh

-rw-r--r-- 1 root root 14 Dec 30 21:50 samuel

\*\*\*\*\*\*\*\*\*MENU\*\*\*\*\*\*\* 1.List of files.

2.copying files. 3.Removing files. 4.Renaming file. 5.Linking file.

press [CTRL+c] to EXIT enter your choices

2

Enter the old filename. rohith samuel

Copied sucessfully.

\*\*\*\*\*\*\*\*\*MENU\*\*\*\*\*\*\* 1.List of files.

2.copying files. 3.Removing files. 4.Renaming file. 5.Linking file.

press [CTRL+c] to EXIT enter your choices

3

Enter the file name to remove. shankar

successfully removed

\*\*\*\*\*\*\*\*\*MENU\*\*\*\*\*\*\* 1.List of files.



2.copying files. 3.Removing files. 4.Renaming file. 5.Linking file.

press [CTRL+c] to EXIT enter your choices

4

Enter the old file name. rohith

Enter the new file name. swaroop

The file name remaned to swaroop .

\*\*\*\*\*\*\*\*\*MENU\*\*\*\*\*\*\* 1.List of files.

1. copying files. 3.Removing files. 4.Renaming file. 5.Linking file.

press [CTRL+c] to EXIT

## Viva Questions

* 1. **Which command is used to copy a file?**

**A.** cp command is used to copy a file.

* 1. **Which command is used to link two files?**

**A.** ln command is used to link two files.

* 1. **What is the syntax of case statement?**

**A.** case word in

pattern1)

Statement(s) to be executed if pattern1 matches

;;

pattern2)

Statement(s) to be executed if pattern2 matches

;;

esac



#### EXPERIMENT – 9

* + 1. **Write shell script that takes a login name as command – line argument and reports when that person logs in**

#### Description:

who|grep “21A91A0578”>/dev/null

who gets the list of logged in users, grep “21A91A0578” filters the list returned by who for entries that contain “21A91A0578”. This will return true if the entry was found. **/dev/null** in Linux is a null device file. This will discard anything written to it, and will return **EOF** on reading.

The **‘>**‘ symbol is used for output (STDOUT) redirection. The **‘<**‘ symbol is used for input(STDIN) redirection “>>” appends output to an existing file

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

**Algorithm:**

**STEP 1:** START

**STEP 2:** IF $# -lt 1 then display improper usage and correct usage is: $0 username exist

**STEP 3:** who|grep “$logname”>/dev/null

**STEP 4:** WHILE TRUE

**STEP 5:** IF $?=0 then

Print $logname has logged in Append data to sh01log.txt Append date to sh01log.txt Print hi in mseg.txt

Append $logname to mseg.txt Print have a good day in mseg.txt Display sh01log.txt

**STEP 6:** ELSE SLEEP 60

**STEP 7:** STOP

### Program:

if [ $# -lt 1 ] then

echo "improper usage"



echo "correct usage is: $0 username exist" fi

logname=$1 while true do

who|grep "$logname">/dev/null if [ $? = 0 ]

then

echo "$logname has logged in" echo "$logname">>sh01log.txt date >>sh01log.txt

echo "Hi" > mesg.txt

echo "$logname" >> mesg.txt

echo "Have a Good Day" >> mesg.txt mail"$logname" < mesg.txt cat sh01log.txt

exit else sleep 60 fi

done

### Output:

[22A95A0518@Linux ~]$ rm mesg.txt 22A95A0518@Linux ~]$ rm sh01log.txt [22A95A0518@Linux ~]$ sh log.sh 21A91A05B1 21A91A05B1 has logged in 21A91A05B1

Fri Nov 18 06:41:20 IST 2022



**b) Write a shell script which receives two file names as arguments. It should check whether the two file contents are same or not. If they are same then second file should be deleted**.

### Description:

The “cmp” command in Linux is used to compare the contents of the two files hence reporting whether the two files are identical or different.

The syntax of the “cmp” command is as follows:

cmp [option] File1 File2

File1 and File2 represent the file names of the two files to be compared.

if [-s filename] is a conditional statement in shell programming which checks if the file exists and it is non empty i.e size of the file is not zero.

### Algorithm:

**STEP 1:** START

**STEP 2:** Display “Enter file 1:” **STEP 3:** Read file1 name into file 1 **STEP 4:** Display “Enter file 2:” **STEP 5:** Read file2 name into file 2

**STEP 6:** Compare file1 and file 2 using diff

**STEP 7:** IF compare returns 0 then Remove second file using rm command

**STEP 8:** Else Print “Keep quite”

**STEP 9:** STOP

### Program:

echo "Enter file 1:" read file1

echo "Enter file 2:" read file2

diff $file1 $file2 >/dev/null



if [ `echo $?` -eq 0 ] then

echo "remove the second file" rm $file2

else

echo "Keep quite" fi

### Output:

[root@localhost ~]# cat>f1 Hi

[root@localhost ~]# cat>f2 Hi

[root@localhost ~]# sh compare.sh Enter file 1:

f1

Enter file 2:

f2

remove the second file [root@localhost ~]# cat f2

cat: f2: No such file or directory

## Viva Questions

* + 1. **What is the purpose of who command?**

**A.** It is used to display the list of users who are currently logged in.

* + 1. **What does $? represents?**

**A.** The $? variable represents the exit status of the previous command.

* + 1. **Which command is used to delete a file?**

**A.** rmcommand is used to delete a file.



# EXPERIMENT – 10

### Description:

fstat is a system call that is used to determine information about a file based on its file descriptor.

#### Syntax:

int fstat(int fd, struct stat \*buf);

**int fd**The file descriptor of the file that is being inquired.

**struct stat \*buf** A structure where data about the file will be stored. Returns a 0 if successful otherwise negative value on failure.

All of these system calls return a stat structure, which contains the following fields:

struct stat {

dev\_tst\_dev; /\* ID of device containing file \*/ ino\_tst\_ino; /\* inode number \*/ mode\_tst\_mode; /\* protection \*/ nlink\_tst\_nlink; /\* number of hard links \*/ uid\_tst\_uid; /\* user ID of owner \*/ gid\_tst\_gid; /\* group ID of owner \*/ dev\_tst\_rdev; /\* device ID \*/

off\_tst\_size; /\* total size, in bytes \*/ blksize\_tst\_blksize; /\* blocksize for filesystem I/O \*/ blkcnt\_tst\_blocks; /\* number of 512B blocks allocated \*/

};

The st\_dev field describes the device on which this file resides. The st\_rdev field describes the device that this file (inode) represents. The st\_size field gives the size of the file in bytes.

The st\_blocks field indicates the number of blocks allocated to the file, 512-byte units. The st\_blksizefield gives the "preferred" blocksize for efficient filesystem I/O. The st\_mode describes the Status of the file.

### Algorithm:

**STEP 1:** START

**STEP 2:** Declare an integer variable fd

**STEP 3:** create a structure variable buf

**STEP 4:** open a file f5 and return value into fd

**STEP 5:** if(fd!=-1) and if(fstat(fd,&buf)==0) print mode of file, size of file,device name inode of file, no.of links, owner of file, block size of the file , time of last modified.

**STEP 6:** ELSE print “Error in fstat()”

**STEP 7:** ELSE Print “Errror in open system call”

**STEP 8:** STOP



### Program:

#include<stdio.h> #include<unistd.h> #include<sys/stat.h> #include<fcntl.h> void main()

{

int fd;

struct stat buf; fd=open("f5.txt",O\_RDONLY|O\_CREAT,600); if(fd!=-1)

{

if(fstat(fd,&buf)==0)

{

printf("mode of file is %u",buf.st\_mode); printf("\n size of the file is %u",buf.st\_size); printf("\n device name %u",buf.st\_dev); printf("\n inode of file is %u",buf.st\_ino); printf("\n no. of links are %u",buf.st\_nlink); printf("\n owner of a file is %u",buf.st\_uid); printf("\n no.of blocks is %u",buf.st\_blocks); printf("\n group owner is %u",buf.st\_gid);

printf("\n blocks size of the file is %u",buf.st\_blksize); printf("\n time of last modified is %u",buf.st\_ctime);

}

else

printf("error in fstat() syscall");

}

else

printf("error in open() sys call");

}

### Output:

[22A95A0518@Linux ~]$ cc file.c [22A95A0518@Linux ~]$ ./a.out mode of file is 33368

size of the file is 0 device name 2054 inode of file is 176704 no. of links are 1

owner of a file is 10341 no.of blocks is 0

group owner is 10342



blocks size of the file is 4096

time of last modified is 1669944828 [22A95A0518@Linux ~]$ ls -l f5.txt

---x-wx--T 1 22A95A0518 22A95A0518 0 Dec 2 07:03 f5.txt

**Viva Questions**

1. **What are the types of files?**

**A.** text file , shell file , c file etc…

1. **Write a brief about the file permission?**

**A.** There are read , write and execute commands for a file.

1. **What is the purpose of fstat()?**

**A.** The **fstat()** function gets status information about the object specified by the open descriptor descriptor and stores the information in the area of memory indicated by the buffer argument.



# EXPERIMENT – 11

**Write a C program which supports that child process inherits environment variables, command line arguments, opened’ files.**

**Description:**

The bzero() function places *n* zero-valued bytes in the area pointed to by *s*.

#### Syntax:

#include <string.h>

void bzero(void \*s, size\_t n);

The *strcat()* function appends a copy of the string pointed to by *s2* (including the terminating null byte) to the end of the string pointed to by *s1.* The *strcat()* function returns *s1*; no return value is reserved to indicate an error.

Syntax:

#include <string.h>

char \*strcat(char \**s1*, const char \**s2*);

The system() function ***passes string to the sh shell command for execution***.

### Algorithm:

**STEP 1:** START

**STEP 2:** Declare a character array d with size 50

**STEP 3:** IF (argc==2) Then call bzero function by passing d, size of d as arguments

**STEP 4:** Then concatinate d ,ls and d,I and d,argv[1]

**STEP 5:** Else Print “Invalid inputs”

**STEP 6:** STOP

**Program:** #include<stdio.h> #include<stdlib.h> #include<string.h> main(int argc,char \*argv[])

{

char d[50]; if(argc==2)

{

bzero(d,sizeof(d));

strcat(d,"ls ");

strcat(d,"-i");

strcat(d,argv[1]); system(d);

}

else

{

printf("\nInvalid No. of inputs");

}

}

### Output:

[22A95A0518@Linux ~]$mkdir d1 [22A95A0518@Linux ~]$ cd d1 [22A95A0518@Linux d1]$ cat>s1 hi

[22A95A0518@Linux d1]$ cd .. [22A95A0518@Linux ~]$ cc 11.c [22A95A0518@Linux ~]$ ./a.out d1 172873 .

[22A95A0518@Linux ~]$ ./a.out swaroop

163179 .zshrc 170816 swap.sh 168710 program.c 168618 mytable 169834 file4

176634 f3 181517 d1/ 169570 armstrong.sh 178096 11.c

171752 wor.sh 174634 stu.sh 168841 program1.sh 172875 .mozilla/ 169861 file3

177126 f2 171660 countries 171739 arg.sh 2 ../

171525 work.sh 171041 stu.data



176909 f1 173727 .ccache/

169555 prime.sh 168534 mm

168706 a.out 17287

169897 file2



**Viva Questions**

1. **What are the environment variables?**

**A.**UNIX environment variables are variables that apply to both the current shell and to any subshells that it creates.

1. **what is the purpose of bzero() function?**

**A.** The bzero() function places n zero-valued bytes in the area pointed to by s.

1. **Which function is used for the concatenation of two strings?**

**A.** strcat() used for the concatenation of two strings.



### EXPERIMENT – 12

**Write a shell script to display factorial value from given argument list.**

### Description:

Functions enable you to break down the overall functionality of a script into smaller, logical subsections, which can then be called upon to perform their individual tasks when needed.

Using functions to perform repetitive tasks is an excellent way to create **code reuse**. Creating Functions

To declare a function, simply use the following syntax −

function\_name ()

{

list of commands

}

The name of your function is **function\_name**, and that's what you will use to call it from elsewhere in your scripts. The function name must be followed by parentheses, followed by a list of commands enclosed within braces.

### Algorithm:

1. Get a number
2. Use for loop or while loop to compute the factorial by using the below formula

3. fact(n) = n \* n-1 \* n-2 \* .. 1

1. Display the result.

### Program:

echo "Enter a number" read num

fact=1

while [ $num -gt1 ] do

fact=$((fact \* num)) #fact = fact \* num num=$((num - 1)) #num = num - 1 done



echo $fact

### Output:

Enter a number 5

120

## Viva Questions

* 1. **What does $\* represents?**

**A.** It represents the array of command line arguments.

* 1. **What is meant by the factorial number?**

**A.** The product of all positive integers less than or equal to a given positive integer and denoted by that integer and an exclamation point**.**

* 1. **What is the syntax of function?**

**A.** function\_name ()

{

list of commands

}



### AUGUMENTED PROGRAMS

**EXPERIMENT - 13**

**Write a shell script to display reverse number from given argument list.**

### Algorithm:

**STEP 1**: START

**STEP 2:** Read an inter n

**STEP 3:** initialize num=0

**STEP 4:** obtain remainder assign it into k

**STEP 5:** num= (num\*10)+k;

**STEP 6:** n=n/10;

**STEP 7:** Repeat from step 4 until n >0

**STEP 8:** STOP

### Program:

echo enter n read n num=0

while [ $n -gt0 ] do

num=$(expr $num \\* 10) k=$(expr $n % 10) num=$(expr $num + $k) n=$(expr $n / 10)

done

echo number is $num

### Output:

enter n 456

number is 654

## Viva Questions

1. **List the looping statements?**

**A.** while loop , for loop.

1. **What does $# represents?**

**A.** The count of the total number of command line arguments passed.

1. **How to compare two numbers?**

**A.** By using operators like –gt , -lt , -eq etc.



### EXPERIMENT – 14

**Write a shell script which will display Fibonacci series up to a given number of arguments.**

### Algorithm:

**STEP 1:** START

**STEP 2:** Display “Enter a number:”

**STEP 3:** read n

**STEP 4:** initialize a=0, b=1

**STEP 5:** print a value

**STEP 6:** add a+b and assign it to fn **STEP 7:** Assign b to a and fn to b **STEP 8:** Increement i

**STEP 9:** Repeat from step 5 until <n

**STEP 10:** STOP

### PROGRAM:

echo “Enter a number:” read n

a=0 b=1

echo "The Fibonacci series is : " for (( i=0; i<n; i++ ))

do

echo -n "$a " fn=$((a + b)) a=$b

b=$fn done

### Output:

Enter a number: 8

Fibonacci Series is :



0

1

1

2

3

5

8

## Viva Questions

1. **Given an example for Fibonacci series.**

**A.** 0 1 1 2 3…………

1. **How to compute Fibonacci series?**

**A.** The sequence follows the rule that each number is equal to the sum of the preceding two numbers.

1. **what is the syntax of for loop?**

**A.** for var in word1 word2 ... wordN

do

Statement(s) to be executed for every word.

done