

Prac_1 Check the output of the following commands:

date, ls, who, cal, ps, wc, cat, uname, pwd, mkdir, rmdir, cd, cp, rm, mv, diff, chmod, grep, sed, head, tail, cut, paste, sort, find, man

Prac_2 Write a script to find the complete path for any file.

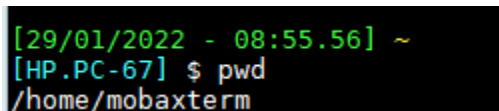
```
clear
echo "Enter File Name : \c "
read fileName

if [ -f $fileName ]
then
str=`find $fileName`
path=`pwd`
echo "Full path of file is $path/$str"

else
echo "file [ $fileName ] not exist in \c "
pwd
fi
```

What is the pwd command example?

Simply type pwd into your terminal, and the command will output the absolute path of your **present working directory**. In this example, the command has indicated that our present working directory is

A terminal window screenshot with a black background. The prompt is [29/01/2022 - 08:55.56] ~. The user has entered the command [HP.PC-67] \$ pwd. The output is /home/mobaxterm.

```
[29/01/2022 - 08:55.56] ~
[HP.PC-67] $ pwd
/home/mobaxterm
```

Find

Find command is use for find the file

```
[29/01/2022 - 09:16.56] ~  
[HP.PC-67] $ find x.txt  
x.txt  
  
[29/01/2022 - 09:17.03] ~  
[HP.PC-67] $ find a.txt  
find: a.txt: No such file or directory  
[29/01/2022 - 09:17.10]
```

Prac_3 Write a shell script to execute following command

echo sorting the file

sort abc.txt > xyz.txt

echo executing two commands

who ; ls

echo -e " this is \n a three-line \n Text message "# use -e option if required

echo The version is `uname -a`

echo Help of cat command

note:

in above program create two file abc.txt xyz.txt

using cat > abc.txt

cat > xyz.txt

save with ctr+z

Prac_4 Write a shell script to execute following command

```
echo 1. How would u display the hidden files
echo 2. How delete directory with files
echo 3. How would user can do interactive copying
echo 4. How would user can do interactive deletion of files
echo 5. Explain two functionalities of mv command with example
echo enter your choice
read ch
case $ch in
1) echo Displaying hidden files
ls .[a-z]* ;;
2) echo Deleting directories with files
rm -R abc.txt ;;
3) echo Interactive copy
cp -i a.txt xyz.txt ;; # file2 should be created first to check interactivity
4) echo Interactive Deletion
rm -i x.txt ;;
5) echo mv command
mv xyz.txt x.txt ;;
*) echo Invalid choice ;;
esac
```

5 Write a shell script to execute following commands

1. Create a file called text and store name, age and address in it.
2. Display the contents of the file text on the screen.
3. Delete the directories mydir and newdir at one shot.
4. Sort a numeric file?
5. Change the permissions for the file newtext to 666.

echo "1. Create a file called text and store name, age and address in it."

echo "2. Display the contents of the file text on the screen."

echo "3. Delete the directories mydir and newdir at one shot."

echo "4. Sort a numeric file"

echo "5. Change the permissions for the file newtext to 666."

echo "enter your choice"

read ch

case \$ch in

1) echo "Create a file called text and store name,age and address in it."

echo "Enter the filename"

read fn

cat > \$fn ;;

2) echo "Display the contents of the file text on the screen."

cat \$fn ;;

3) echo "Delete the directories mydir and newdir at one shot."

rmdir mydir newdir ;;

4) echo "Sort a numeric file"

sort -n filename ;;

5) echo "Change the permissions for the file newtext to 666."

```
chmod 666 newtext ;;
```

```
*) echo "Invalid choice" ;;
```

```
esac
```

Pra-6: Write shell script that accept filename and displays last modification time if file exists.

```
echo "enter filename"
read fn
if [ -e $fn ]
then
ls -l $fn | cut -d " " -f 23
else
echo "file does not exist"
fi
```

NOTE

```
ls -l $fn | cut -d " " -f 23
```

here ls-l display in detail list

```

[18/01/2022 - 12:50.59] ~
[admin.DESKTOP-9HF80PQ] $ vi pra5.sh

[18/01/2022 - 12:55.14] ~
[admin.DESKTOP-9HF80PQ] $ vi pra14.sh

[18/01/2022 - 13:03.58] ~
[admin.DESKTOP-9HF80PQ] $ ls-l
bash: ls-l: command not found

[18/01/2022 - 13:04.00] ~
[admin.DESKTOP-9HF80PQ] $ ls -l
total 7
lrwxrwxrwx  1 admin  Domain U      29 Jan 18 08:29 Desktop -> /drives/C/Users/admin/Desktop
lrwxrwxrwx  1 admin  Domain U       7 Jan 18 08:29 Drives -> /drives
lrwxrwxrwx  1 admin  Domain U     30 Jan 18 08:29 LauncherFolder -> /drives/C/Users/admin/Desktop/
lrwxrwxrwx  1 admin  Domain U     31 Jan 18 08:29 MyDocuments -> /drives/C/Users/admin/Documents
drwxr-xr-x  1 admin  Domain U       0 Jan 18 08:32 abc
-rw-r--r--  1 admin  Domain U       9 Jan 18 08:30 abc.txt
-rw-r--r--  1 admin  Domain U    151 Jan 18 08:58 pr14.sh
-rw-r--r--  1 admin  Domain U    123 Jan 18 09:14 pr6.sh
-rw-r--r--  1 admin  Domain U    138 Jan 18 09:29 pra10.sh
-rw-r--r--  1 admin  Domain U    113 Jan 18 09:33 pra11.sh
-rw-r--r--  1 admin  Domain U    148 Jan 18 13:03 pra14.sh
-rw-r--r--  1 admin  Domain U       0 Jan 18 12:55 pra5.sh
-rw-r--r--  1 admin  Domain U    113 Jan 18 09:38 pra6.sh
drwxr-xr-x  1 admin  Domain U       0 Jan 18 09:30 ri
-rw-r--r--  1 admin  Domain U    117 Jan 18 08:40 tt.sh
-rw-r--r--  1 admin  Domain U       9 Jan 18 09:06 xyz.txt

[18/01/2022 - 13:04.05] ~
[admin.DESKTOP-9HF80PQ] $

```

Cut -d [Delimiter]

Use for extract field from a list/file.

-f 23 use for particular field and which line you want to see.

Other example

Extracting field from file

Remember, in order to extract a field from a file, we would need a delimiter (i.e. a column separator), based on which the file will be divided into columns and we can extract any of them. In this case, the syntax would be-

```
cut -d [DELIMITER] -f [RANGE] [FILENAME]
```

Here, we are instructing cut command to use a particular delimiter with option -d and then extract certain fields using option -f.

1. Display a specific field from a file

In case of a *csv* file, it is crystal clear that our delimiter will be a comma (.). Now, we need to enlist the names of the employees working in our organization, i.e. field number 2.

Example:

```
$ cut -d ',' -f 2 employees.txt
Employee Name
John Davies
Mary Fernandes
Jacob Williams
Sean Anderson
Nick Jones
Diana Richardson
```

2. Displaying Multiple Fields from a File

Moving forward now, let's display more than one field now. Suppose, we need to include 'Age' and 'Gender' fields also. For this, we must specify the range - again, a start and an end.

```
$ cut -d ',' -f 2-4 employees.txt
Employee Name, Age, Gender
John Davies, 35, M
Mary Fernandes, 29, F
Jacob Williams, 40, M
Sean Anderson, 25, M
Nick Jones, 42, M
Diana Richardson, 29, F
```

7 Write a shell script to display the login names that begin with 's'.

```
who | grep ^s
```

8 Write a shell script to remove the zero sized file from the current directory

```
for i in *  
  
do  
  
if [ ! -s $i ]  
  
then  
  
rm $i  
  
echo " $i removed "  
  
fi  
  
done
```

9 Write a shell script to display the name of all the executable file from the current directory.

```
countx=0  
for i in *  
do  
if [ -x $i ]  
then  
countx=`expr $countx + 1`  
fi  
done  
echo Number of executable files are $countx
```


10. Write a shell script that will display welcome message according to time

```
d=`date +%H`  
if [ $d -lt 12 ]  
then  
echo Good Morning  
elif [ $d -gt 12 -a $d -lt 14 ]  
then  
echo good afternoon  
else  
echo good night  
fi
```

Note

Here date command display date

```
[admin.DESKTOP-9HF8OPQ] $ date  
Tue Jan 18 13:10:40 IST 2022
```

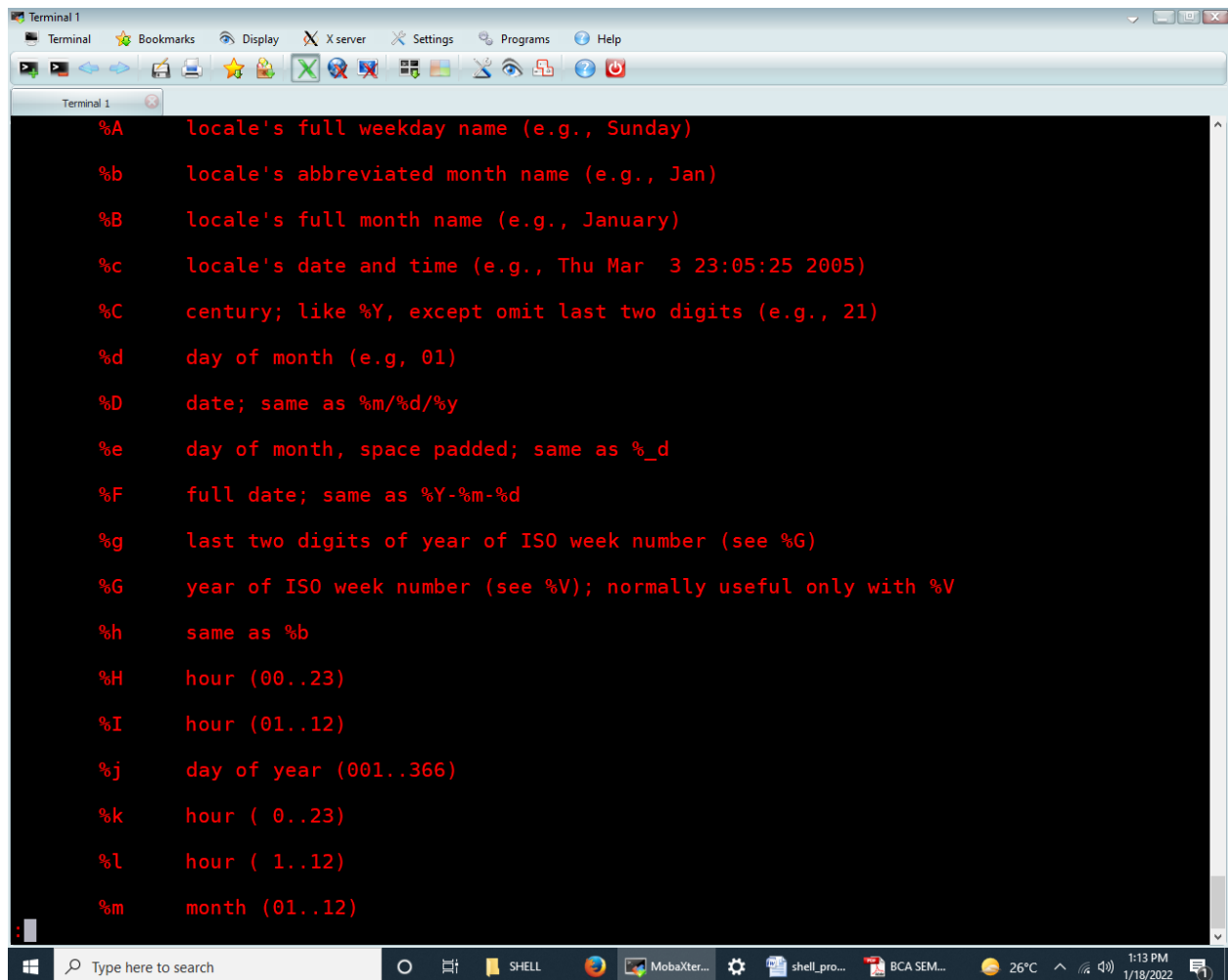
NAME

date - print or set the system date and time

SYNOPSIS

date [OPTION]... [+FORMAT]

date [-u|--utc|--universal] [MMDDhhmm[[CC]YY][.ss]]



11. Write a shell script to find number of ordinary files and directory files

```
countd=0
countf=0
for i in *
do
    if [ -d $i ]
    then
        countd=`expr $countd + 1`
    fi
    if [ -f $i ]
```

```
then
countf=`expr $countf + 1`
fi
done
echo Number of directories are $countd
echo Number of Ordinary files are $countf
```

12 Write a shell script that takes a filename from the command line and checks whether the file is an ordinary file or not.

- If it is an ordinary file then it should display the contents of the file.
- If it is not an ordinary file then script should display the message: "File does not exist or is not ordinary, cannot display."

```
echo "enter filename"
read $1
if [ -f $1 ]
then
echo Its an ordinary file
else
echo File does not exist or is not ordinary file
fi
```

13 Write a shell script that takes a filename from the user and checks whether it is a directory file or not.

```
echo enter the filename
read fn
if [ -f $fn ]
then
echo Its a directory Is $fn
else
echo Its not a directory
```

fi

Note

-f is used for file

-d used for directory

14 Write a shell script that takes a filename as an argument and checks if the file exists and is executable.

echo enter file name

read fn

if [-e \$fn -a -x \$fn]

then

echo file exists and is executable

else

echo file does not exist or is not executable

fi

Note

-e \$fn -a -x \$fn

-e used for exist

\$fn used for file name

-a means and

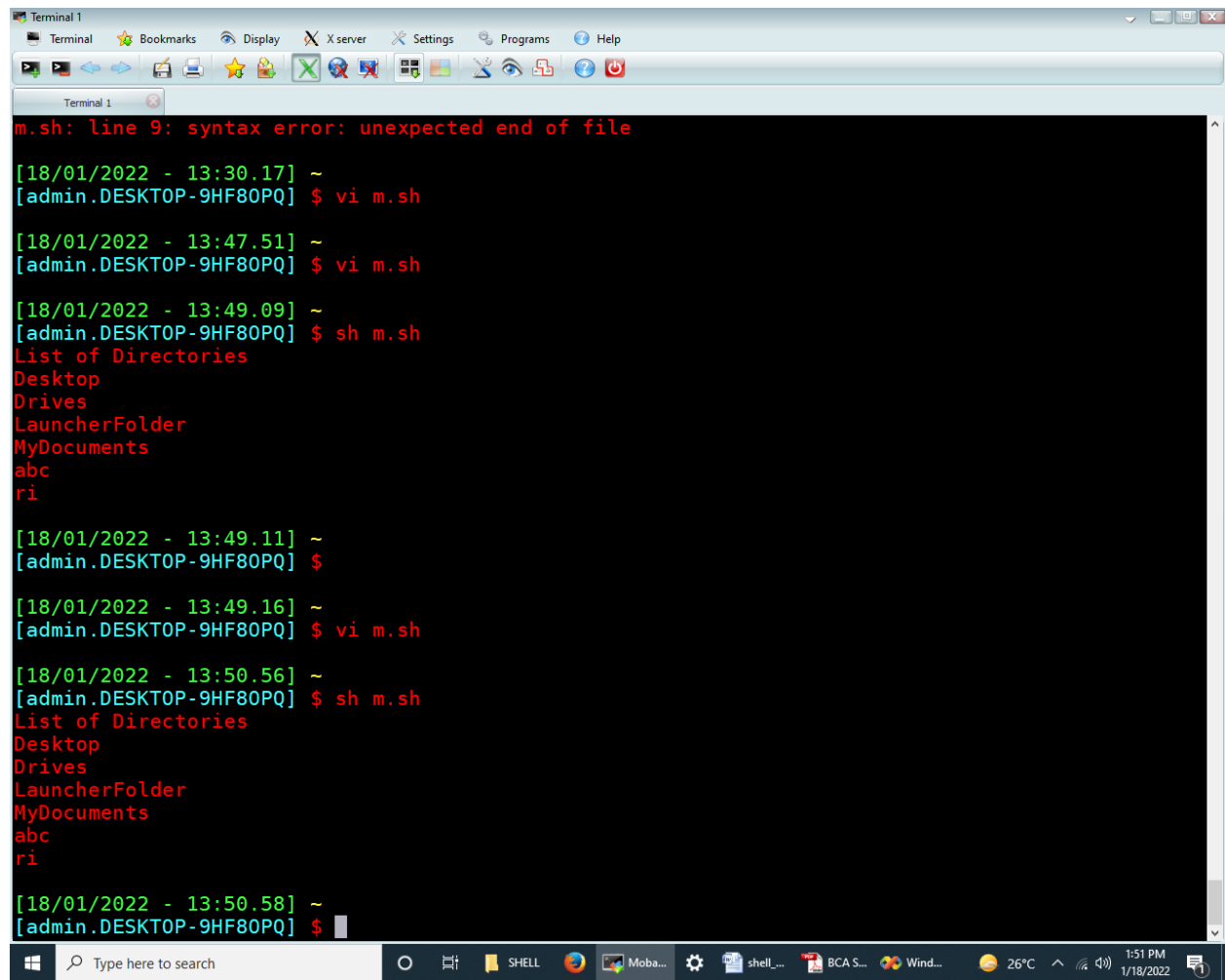
-x means executable

15 Write a shell script that displays all subdirectories in current working directory.

echo List of Directories

```
for i in *  
do  
if [ -d $i ]  
then  
echo $i  
fi  
done
```

output



```
m.sh: line 9: syntax error: unexpected end of file  
  
[18/01/2022 - 13:30.17] ~  
[admin.DESKTOP-9HF80PQ] $ vi m.sh  
  
[18/01/2022 - 13:47.51] ~  
[admin.DESKTOP-9HF80PQ] $ vi m.sh  
  
[18/01/2022 - 13:49.09] ~  
[admin.DESKTOP-9HF80PQ] $ sh m.sh  
List of Directories  
Desktop  
Drives  
LauncherFolder  
MyDocuments  
abc  
ri  
  
[18/01/2022 - 13:49.11] ~  
[admin.DESKTOP-9HF80PQ] $  
  
[18/01/2022 - 13:49.16] ~  
[admin.DESKTOP-9HF80PQ] $ vi m.sh  
  
[18/01/2022 - 13:50.56] ~  
[admin.DESKTOP-9HF80PQ] $ sh m.sh  
List of Directories  
Desktop  
Drives  
LauncherFolder  
MyDocuments  
abc  
ri  
  
[18/01/2022 - 13:50.58] ~  
[admin.DESKTOP-9HF80PQ] $
```

16 Write a shell script that calculates the number of ordinary and directory files in your current working directory.

```
countd=0
countf=0
for i in *
do
    if [ -d $i ]
    then
        countd=`expr $countd + 1`
    fi
    if [ -f $i ]
    then
        countf=`expr $countf + 1`
    fi
done
echo Number of directories are $countd
echo Number of Ordinary files are $countf
```

17 Write a shell script that accepts 2 filenames and checks if both exists; if both exist then append the content of the second file into the first file

```
echo enter the first filename
read fn1
echo enter the second filename
read fn2
if [ -f $fn1 -a -f $fn2 ]
then
    echo Both file exists
    cat $fn2 >> $fn1
else
    echo Files does not exist
fi
```

Note:

In above program create two file

```
[HP.PC-67] $ cat > m.txt
hi
h r u

[8]+ Stopped /bin/busybox.exe cat > m.txt

[29/01/2022 - 09:55.51] ~
[HP.PC-67] $ cat > n.txt
welcome

[9]+ Stopped /bin/busybox.exe cat > n.txt

[29/01/2022 - 09:56.02] ~
[HP.PC-67] $ cat m.txt >> n.txt

[29/01/2022 - 09:56.14] ~
[HP.PC-67] $ cat m.txt
hi
h r u

[29/01/2022 - 09:56.18] ~
[HP.PC-67] $ cat n.txt
welcome
hi
h r u

[29/01/2022 - 09:56.22] ~
[HP.PC-67] $
```

You can also contain copies to new file using follow command .in below command we merge two file and contain display new file.

```
[29/01/2022 - 09:56.54] ~  
[HP.PC-67] $ cat m.txt n.txt >y.txt  
  
[29/01/2022 - 09:57.16] ~  
[HP.PC-67] $ cat y.txt  
hi  
h r u  
welcome  
hi  
h r u  
  
[29/01/2022 - 09:57.20] ~  
[HP.PC-67] $
```

18 Write a shell script that takes the name of two files as arguments and performs the following:

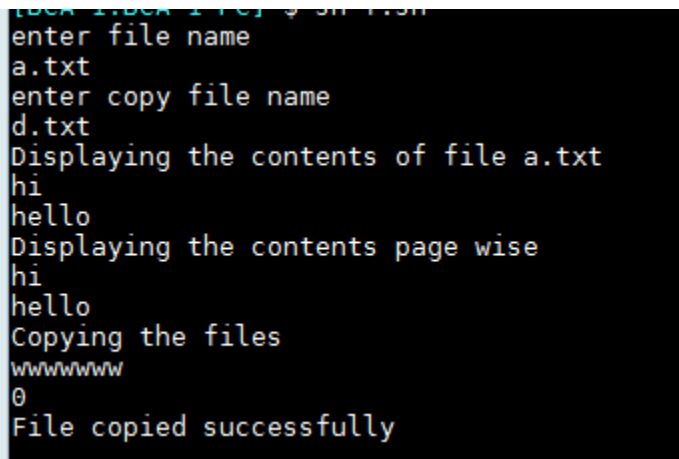
- i. Displays the message : “Displaying the contents of file :(first argument)” and displays the contents page wise.
- ii. Copies the contents of the first argument to second argument.
- iii. Finally displays the message: “File copied successfully.”³

```
echo enter file name  
read n  
echo enter copy file name  
read file  
echo Displaying the contents of file $n  
cat $n  
echo Displaying the contents page wise  
cat $n | more
```



```
echo Copying the files
cp $n $file
c=`echo $?`
if [ $c -eq 0 ]
then
echo File copied successfully
else
echo Files not copied successfully
fi
```

Note:



```
(BCA 1-BCA 1-FC) $ ./sh 1.sh
enter file name
a.txt
enter copy file name
d.txt
Displaying the contents of file a.txt
hi
hello
Displaying the contents page wise
hi
hello
Copying the files
wwwwwww
0
File copied successfully
```

Echo \$?

Echoes (prints) **the exit value for the previous command**. If it failed it will be different than zero (0). \$ cd ~ \$ echo \$? > 0 \$ cd //whatever/ > bash: cd: //whatever/: No such file or directory \$ echo \$? > 1. Programs exit with a status code.

```

[BCA-1.BCA-1-PC] $ sh 1.sh
enter file name
f.txt
enter copy file name
a.txt
Displaying the contents of file f.txt
cat: can't open 'f.txt': No such file or directory
Displaying the contents page wise
cat: can't open 'f.txt': No such file or directory
Copying the files
cp: can't stat 'f.txt': No such file or directory
wwwwwww
1
Files not copied successfully
[31/01/2022 - 13:53.29] ~

```

19 Write a shell script to display the following menu and acts accordingly:

- i. Calendar of the current month and year.
- ii. Display “Good Morning/Good Afternoon/Good Evening” according to the current login time.
- iii. User name, Users home directory.
- iv. Terminal name, Terminal type.
- v. Machine name.
- vi. No. of users who are currently logged in; List of users who are currently logged in.

echo 1. Calendar of the current month and year.

echo 2. Display Good Morning/Good Afternoon/Good Evening according to the current login time.

echo 3. User name, Users home directory.

echo 4. Terminal name, Terminal type.

echo 5 Machine name.

echo 6. No. of users who are currently logged in List of users who are currently logged in.

echo enter your choice

read ch

case \$ch in

1) echo Calendar of current month is

cal ;;

2) d=`date +"%H"`

```
if [ $d -lt 12 ]
then
echo Good Morning
elif [ $d -gt 12 -a $d -lt 16 ]
then
echo Good Afternoon
else
echo Good Evening
fi
;;
3) echo Username is $USER
echo Users Home directory is $HOME ;;
4) echo Terminal details
tty;;
5) echo Machine name is
uname -m ;;
6) echo The number of users logged in are
who | wc -l
*) echo Invalid choice
esac
```

20 Write a shell script that displays the following menu and acts accordingly

1. Concatenates two strings
2. Renames a file
3. Deletes a file.
4. Copy the file to specific location

```
echo 1. Concatenates two strings
echo 2. Renames a file
echo 3. Deletes a file.
echo 4. Copy the file to specific location
echo enter your choice
read ch
```

```
case $ch in
1) echo enter first string
read str1
echo enter second string
read str2
echo The concated strings are $str1$str2 ;;
2) echo enter the old filename
read ofn
echo enter the new filename
read nfn
mv $ofn $nfn
echo file renamed ;;
3) echo enter the filename
read fn
rm $fn
echo file deleted ;;
4) echo enter the filename
read fn
cp $fn \usr\home\dir\ $fn #you can change the specific path
echo file copied ;;
*) echo invalid choice ;;
esac
```

21 Write a shell script to change the suffix of all your *.txt files to .dat.

```
for file in *.txt
do
echo ${file%.*}.doc;
done
```

22 Write a shell script to accept a directory-name and display its contents. If input is not given then HOME directory's contents should be listed. (Make use of command line argument)

```
if [ $# ]
```

```
then
```

```
ls $1
```

```
else
```

```
ls $HOME
```

```
fi
```

\$# : Stands for **counts of the number of the positional parameters passed to a script**. example test.sh 1 2 3 4 5 6 7 , there are 7 parameters as passed so the count of the \$# is

\$0 is the name of the script itself (script.sh) **\$1** is the **first argument (filename1)** **\$2** is the second argument (dir1) **\$9** is the ninth argument.

23 Write a shell script to get all files of home directory and rename them if their names start with c.

```
Newname = oldname111
```

```
[BCA-1.BCA-1-PC] $ ls |grep "^c"
```

```
c.txt
```

```
city.txt
```

24 Write a shell script that takes two filename as arguments. It should check whether the contents of two files are same or not, if they are same then second file should be deleted.

```
echo enter the first filename

read fn1

echo enter the second filename

read fn2

cmp $fn1 $fn2

c=`echo $?`

if [ $c -eq 0 ]

then

echo both files are same

rm $fn2

else

echo both files are not same

fi
```

25 Write a shell script that accepts two directory names from the command line and copies

all the files of one directory to another. The script should do the following

- If the source directory does not exist, flash a error message
- If destination directory does not exist create it
- Once both exist copy all the files from source directory to

destination directory

```
if [ -d $1 -a -d $2 ]
then
    cp -R $1 $2

else
    mkdir $2
    echo "Directory created $2"
    cp -R $1 $2

fi
```

26 Write a shell script that displays the following menu

- List home directory
- Date
- Print working directory
- Users logged in

Read the proper choice. Execute corresponding command. Check for invalid choice.

```
echo 1.List home directory
```

```
echo 2.Date
```

```
echo 3. Print working directory
```

```
echo 4. Users logged in
```

echo enter your choice

read ch

case \$ch in

1) echo Home directory is \$HOME ;;

2) echo Todays date is `date` ;;

3) echo Present working directory is `pwd` ;;

4) echo No of users logged in are

who | wc -l ;;

*) echo Invalid choice ;;

esac

27 Write a shell script that displays all hidden files in current directory.

ls .[a-z]*

28 Write a shell script that Combine two files in the third file horizontally and vertically.

echo enter the first filename

read fn1

echo enter the second filename

read fn2

echo Combining two files horizontally

cat \$fn2 >> \$fn1

echo Combining two files vertically

paste \$fn1 \$fn2

note

1. Pasting columns

In its most basic use case, the paste command takes N input files and joins them line by line on the output:

```
sh$ printf "%s\n" {a..e} | tee letters
```

```
a
b
c
d
e
```

```
sh$ printf "%s\n" {1..5} | tee digits
```

```
1
2
3
4
5
```

```
sh$ paste letters digits
```

```
a 1
b 2
c 3
d 4
e 5
```

```
paste -s vertical represent
```

```
a b c d e
1 2 3 4 5
```

29 Write a shell script to delete all the spaces from a given file.

```
echo enter the filename
```

```
read datafile
```

```
cat $datafile | tr -d '[:space:]' > newfile
```

```
echo enter the filename
```

```
read datafile
```

```
cat $datafile | tr -d '[:r:]' > newfile
```

30 Write a shell script to find a given date fall on a weekday or a weekend.

```
d=`date +%u`  
if [ $d -eq 7 ]  
then  
echo "it is weekend"  
else  
echo "it is a weekday"  
fi
```

31 Write a shell script to search for a given word in all the files given as the arguments on the command line.

```
echo "Enter the word"
read w
for i in $@
do
grep $w $i
done
```

32 Write a shell script that display last modified file in the current directory.

```
ls -lt | head -2 | tail -1
```

33 Write a script to display the permissions of the particular file.

```
echo enter the filename
```

```
read fn
```

```
ls -l $fn | cut -c 2-10
```

Cut Command in Unix with Examples

The cut command extracts a given number of characters or columns from a file. For cutting a certain number of columns it is important to specify the delimiter. A delimiter specifies how the columns are separated in a text file

Example: Number of spaces, tabs or other special characters.

Syntax:

```
cut [options] [file]
```

The cut command supports a number of options for processing different record formats. For fixed width fields, the -c option is used.

```
$ cut -c 5-10 file1
```

This command will extract characters 5 to 10 from each line.

For delimiter separated fields, the -d option is used. The default delimiter is the tab character.

```
$ cut -d “,” -f 2,6 file1
```

This command will extract the second and sixth field from each line, using the ‘,’ character as the delimiter.

Example:

Assume the contents of the data.txt file are:

```
Employee_id;Employee_name;Department_name;Salary
10001;Employee1;Electrical;20000
10002; Employee2; Mechanical;30000
10003;Employee3;Electrical;25000
10004; Employee4; Civil;40000
```

And the following command is run on this file:

```
$ cut -c 5 data.txt
```

The output will be:

```
o
1
2
3
4
```

If the following command is run on the original file:

```
$ cut -c 7-15 data.txt
```

The output will be:

```
ee_id; Emp
```

```
Employee1
```

```
Employee2
```

```
Employee3
```

Employee4

If the following command is run on the original file:

```
$ cut -d “,” -f 1-3 data.txt
```

The output will be:

Employee_id;Employee_name;Department_name

10001;Employee1;Electrical

10002; Employee2; Mechanical

10003;Employee3;Electrical

10004; Employee4; Civil

35,37- Write a shell script to display the following menu for a particular file:
echo 1. Display all the words of a file in ascending order.

echo 2. Display a file in descending order.

echo 3. Toggle all the characters in the file.

echo 4. Display type of the file.

echo enter your choice

read ch

echo enter the filename

read fn

case \$ch in

1) sort \$fn ;;

2) `sort -r $fn ;;`

3) `cat $fn | tr [a-z] [A-Z] ;;`

4) `file $fn ;;`

*) `echo invalid choice`

Note: Ascending order command is **sort**

Descending order command is **sort-r**

Toggle all character(capital all letter) **tr** [a-z][A-Z][a-z]

tr (short for **translate**) is a useful command line utility that translates and/or deletes characters from stdin input, and writes to stdout. It is a useful program for manipulating text on the command line.

```
$ cat linux.txt

linux is my life
linux has changed my life
linux is best and everthing to me..:)
```

```
$ cat domains.txt | tr [:lower:] [:upper:]

LINUX IS MY LIFE
LINUX HAS CHANGED MY LIFE
LINUX IS BEST AND EVERTHING TO ME..:)
```

2. Alternatively, you can use the following command to change all lower case letters to upper case in a file as shown.

```
$ cat linux.txt | tr [a-z] [A-Z]

LINUX IS MY LIFE
LINUX HAS CHANGED MY LIFE
LINUX IS BEST AND EVERTHING TO ME..:)
```

3. To save the results written to **stdout** in a file for later processing, use the shell's output redirection feature (>) as shown.

```
$ cat linux.txt | tr [a-z] [A-Z] >output.txt
$ cat output.txt

LINUX IS MY LIFE
LINUX HAS CHANGED MY LIFE
LINUX IS BEST AND EVERTHING TO ME..:)
```

4. In regards to the redirection, you can send input to **tr** using the input redirection and redirect the output to a file using the same command, as shown.

```
$ tr [a-z] [A-Z] < linux.txt >output.txt
```

5. Another useful feature is, you can use the **-d** flag to delete characters, for example to remove the spaces in the domain names using the following command.

```
$ cat domains.txt

www. tecmint. com
www. fossmint. com
www. linuxsay. com
$ cat domains.txt | tr -d ' '

www.tecmint.com
www.fossmint.com
www.linuxsay.com
```

6. If there are repeated characters in a sequence (for instance double spaces) in the text you are processing, you can use the **-s** option to squeeze the characters leaving only one occurrence of it.

```
$ cat domains.txt

www.tecmint....com
www.fossmint.com
www.linuxsay.com
$ cat domains.txt | tr -s ' '

www.tecmint.com
www.fossmint.com
www.linuxsay.com
```


7. The `-c` option tells `tr` to use the complement in the given of **SET**. In this example, we want to delete all the letters and only leave the **UID**.

```
$ echo "My UID is $UID" | tr -cd "[:digit:]\n"
OR
$ echo "My UID is $UID" | tr -d "a-zA-Z"
```

8. Here is an example of breaking a single line of words (sentence) into multiple lines, where each word appears in a separate line.

```
$ echo "My UID is $UID"

My UID is 1000

$ echo "My UID is $UID" | tr " " "\n"

My
UID
is
1000
```

9. Related to the previous example, you can also translate multiple lines of words into a single sentence as shown.

```
$ cat uid.txt

My
UID
is
1000

$ tr "\n" " " < uid.txt

My UID is 1000
```

10. It is also possible to translate just a single character, for instance a space into a " : " character, as follows.

```
$ echo "Tecmint.com =>Linux-HowTos,Guides,Tutorials" | tr "  
" ":"
```

```
Tecmint.com:=>Linux-HowTos,Guides,Tutorials
```

There are several sequence characters you can use with **tr**, for more information, see the **tr** man page.

```
$ man tr
```

36 Write a shell script to check whether the named user is currently logged in or not.

```
echo "enter the username"
```

```
read un
```

```
c=`who | grep -c $un`
```

```
if [ $c -gt 0 ]
```

```
then
```

```
    echo "User is currently logged in "
```

```
else
```

```
    echo "User is not currently logged in"
```

```
fi
```

```
"enter the username"  
admin  
"User is not currently logged in"
```

38 Write a shell script to find total no. Of users and finds out how many of them are

currently logged in.

```
echo "The number of users in the system are"
```

```
cat etc/passwd | wc -l
```

```
echo "The number of uses currently logged in are "
```

```
who | wc -l
```

39 Write a shell script that displays the directory information in the following format-

Filename Size Date Protection Owner

```
echo Enter the filename
```

```
read fn
```

```
echo Filename      Size      Date      Protection      Owner
```

```
echo `ls -l $fn | cut -d ' ' -f23` `ls -l $fn | cut -d ' ' -f5` `ls  
-l $fn | cut -d ' ' -f20,` `ls -l $fn | cut -d ' ' -f1` `ls -l $fn |  
cut -d ' ' -f10`
```

40 Write a shell script to display five largest files from the current directory

```
ls -ls | head -1 | tail -1
```

Or

```
ls -l | tail -1
```

or

ls -lt | head -2

or

ls -lt

\$ ls -lt

total 5

```
-rw-r--r--  1 BCA-1  Domain U    104 Mar 14 09:15 m.sh
-rw-r--r--  1 BCA-1  Domain U     21 Mar 14 08:56 city.txt
-rw-r--r--  1 BCA-1  Domain U   611 Mar 14 08:40 t.sh
-rw-r--r--  1 BCA-1  Domain U    27 Mar 14 08:12 names
-rw-r--r--  1 BCA-1  Domain U    27 Mar 14 08:08 c.txt
-rw-r--r--  1 BCA-1  Domain U   106 Mar 14 07:55 a.txt
lrwxrwxrwx  1 BCA-1  Domain U    29 Mar 14 07:51 Desktop ->
/drives/C/Users/BCA-1/Desktop
lrwxrwxrwx  1 BCA-1  Domain U     7 Mar 14 07:51 Drives -> /drives
lrwxrwxrwx  1 BCA-1  Domain U   30 Mar 14 07:51 LauncherFolder ->
/drives/C/Users/BCA-1/Desktop/
lrwxrwxrwx  1 BCA-1  Domain U   31 Mar 14 07:51 MyDocuments ->
/drives/C/Users/BCA-1/Documents
```

[BCA-1.BCA-1-PC] \$ ls -lt | tail -1

```
lrwxrwxrwx  1 BCA-1  Domain U   31 Mar 14 07:51 MyDocuments ->
/drives/C/Users/BCA-1/Documents
```

[BCA-1.BCA-1-PC] \$ ls -lt | head -2

total 5

```
-rw-r--r--  1 BCA-1  Domain U    104 Mar 14 09:15 m.sh
```

The *head* Command

The syntax of the *head* command is pretty straightforward:

```
head [OPTIONS] FILES
```

Let's prepare a file (*numbers_en.txt*) as the input example to understand the command better:

```
$ cat numbers_en.txt
one           : 1
two           : 2
three         : 3
four          : 4
...
ninety-seven  : 97
ninety-eight  : 98
ninety-nine   : 99
one hundred   : 100
```

The file contains English words of numbers from 1 to 100. Thus, the file has 100 lines.

The *head* command will, by default, write the first ten lines of the input file to the standard output:

```
$ head numbers_en.txt
one           : 1
two           : 2
three         : 3
four          : 4
five          : 5
six           : 6
seven         : 7
eight         : 8
nine          : 9
ten           : 10
```

Output a Specific Number of Lines

With the *-n* option, we can let the *head* command output the first *n* lines instead of the default 10.

For example, if we want to have the first seven lines printed to standard out, we'd use *-n 7*:

```
$ head -n 7 numbers_en.txt
one           : 1
two           : 2
three         : 3
four          : 4
five          : 5
six           : 6
seven         : 7
```

If we pass the *-n* option together with a number following the *-*, for example *-n -x*, the *head* command will print all lines but the last *x* lines of the file.

For instance, if we want to ignore the last 97 lines from the file, we'd do `-n -97`:

```
$ head -n -97 numbers_en.txt
one      : 1
two      : 2
three    : 3
```

The *tail* Command

The syntax of using the *tail* command is quite straightforward, too:

```
tail [OPTIONS] FILES
```

The *tail* command will by default write the last ten lines of the input file to the standard output:

```
$ tail numbers_en.txt
ninety-one : 91
ninety-two : 92
ninety-three : 93
ninety-four : 94
ninety-five : 95
ninety-six : 96
ninety-seven : 97
ninety-eight : 98
ninety-nine : 99
one hundred : 100
```

Output a Specific Number of Lines

With the `-n` option, we can let the *tail* command output the last *n* lines instead of the default 10.

This example shows how to get the last seven lines from the input file:

```
$ tail -n 7 numbers_en.txt
ninety-four : 94
ninety-five : 95
ninety-six : 96
ninety-seven : 97
ninety-eight : 98
ninety-nine : 99
one hundred : 100
```

If we pass the `-n` option together with a number following the “+”, for example “`-n +x`”, the *tail* command will print starting with the *x*-th line till the end of the file.

Let's print from 95th line till the end of the *numbers_en.txt* file:

```
$ tail -n +95 numbers_en.txt
ninety-five      : 95
ninety-six       : 96
ninety-seven     : 97
ninety-eight     : 98
ninety-nine      : 99
one hundred      : 100
```

Output a Specific Number of Bytes

Similar to the *head* command, if we pass *-c x* option to the *tail* command, it will output only the last *x* bytes from the input file.

Let's get the last number *100* from the input file:

```
$ tail -c 4 numbers_en.txt
100
```

41 Write a shell script that toggles contents of the file

```
echo "Enter the filename"
```

```
read fn
```

```
cat $fn | tr [a-z] [A-Z] ;;
```

42 Write a shell script that report whether your friend has currently logged in or not.

If he has logged in then shell script should send a message to his terminal suggesting a dinner tonight. If you do have write permission to his terminal suggesting a dinner tonight. If you do have write permission to his terminal or if he hasn't logged in then such a message should be mailed to him about your dinner proposal.

```
echo "Enter the username"
```

```
read un
```

```
c=`who | grep -c $un`
```

```
if [ $c -gt 0 ]
```

```
then
```

```
    echo "User is currently logged in "
```

```
else
```

```
    echo "User is not currently logged in"
```

```
fi
```

44 Write a shell script to accept any character using command line and list all the files

starting with that character in the current directory

```
ls | grep ^$1
```

```
[BCA-1.BCA-1-PC] $ sh g.sh c  
c.txt  
city.txt
```

45 Create a file called student containing roll-no, name and marks.

- a. Display the contents of the file sorted by marks in descending order
- b. Display the names of students in alphabetical order ignoring the case.
- c. Display students according to their roll nos.
- d. Sort file according to the second field and save it to file 'names'.
- e. Display the list of students who scored between 70 and 80.

echo 1. Display the contents of the file sorted by marks in descending order

echo 2. Display the names of students in alphabetical order ignoring the case.


```
echo 3. Display students according to their roll nos.
echo 4. Sort file according to the second field and save it to file names.
echo 5. Display the list of students who scored between 70 and 80
echo enter filename
read fn
echo enter your choice
read ch
case $ch in

1) sort -k3 -r $fn ;;
2) sort -k2 -i $fn ;;
3) sort $fn ;;
4) sort -k2 $fn > names ;;
5) awk '{ if ( $3 >=70 && $3 <=80 ) print $3 }' $fn ;;
*) echo Invalid Choice ;;

esac
```

The basic function of *awk* is to search files for lines (or other units of text) that contain certain patterns. When a line matches one of the

patterns, *awk* performs specified actions on that line. *awk* keeps processing input lines in this way until it reaches the end of the input files.

Programs in *awk* are different from programs in most other languages, because *awk* programs are *data-driven*; that is, you describe the data you want to work with and then what to do when you find it. Most other languages are *procedural*; you have to describe, in great detail, every step the program is to take. When working with procedural languages, it is usually much harder to clearly describe the data your program will process. For this reason, *awk* programs are often refreshingly easy to read and write.

When you run *awk*, you specify an *awk program* that tells *awk* what to do. The program consists of a series of *rules*. (It may also contain *function*

definitions, an advanced feature that we will ignore for now. See the [Section 8.2](#) in [Chapter 8](#).) Each rule specifies one pattern to search for and one action to perform upon finding the pattern.

Syntactically, a rule consists of a pattern followed by an action. The action is enclosed in curly braces to separate it from the pattern. Newlines usually separate rules. Therefore, an *awk* program looks like this:

```
        pattern { action }
pattern { action }
...
```

```
1001 John sena 40000
1002 Jafar Iqbal  60000
1003 Meher Nigar  30000
1004 Jonny Liver  70000
```

The following *awk* command will read data from **employee.txt** file line by line and print the first field after formatting. Here, “%10s\n” means that the output will be 10 characters long. If the value of the output is less than 10 characters then the spaces will be added at the front of the value.

```
$ awk '{ printf "%10s\n", $1 }' employee.txt
```

Output

```
1001
1002
1003
1004
```

Simple if example:

The following command will read the content of the **items.txt** file and check the 3rd field value in each line. If the value is empty then it will print an error message with the line number.

```
$ awk '{ if ($3 == "") print "Price field is missing in line " NR }' items.txt
```

if-else example:

The following command will print the item price if the 3rd field exists in the line, otherwise, it will print an error message.

```
$ awk '{ if ($3 == "") print "Price field is missing"
else print "item price is " $3 }' items.txt
```

if-else-if example:

When the following command will execute from the terminal then it will take input from the user. The input value will be compared with each if condition until the condition is true. If any condition becomes true then it will print the corresponding grade. If the input value does not match with any condition then it will print fail.

```
$ awk 'BEGIN { print "Enter the mark:"
getline mark < "-"
if (mark >= 90) print "A+"
else if( mark >= 80) print "A"
else if( mark >= 70) print "B+"
else print "Fail" }'
```