1)Write a shell script to create a file. Follow the instructions

(i) Input a page profile to yourself, copy it into other existing file;

(ii) Start printing file at certain line

(iii) Print all the difference between two file, copy the two files.

(iv) Print lines matching certain word pattern.

(i) Input a page profile to yourself, copy it into other existing file;

Solution:-

```
echo"create a file in /user/class/batch in directory"
mkdir –p user/class/b1
echo "Display present working DIR"
cd user/class/b1
pwd
echo "Enter a file name"
read file1
echo "Enter contains in $file1"
cat  > $file1
echo "Enter existing file name"
read  file2
echo "Display copy of contains $file1 to $file2"
cp  $file1   $file2
cat $file2
```

(ii) Start printing file at certain line

Solution:-

```
echo"create a file in /user/class/batch in directory"
mkdir –p user/class/b1
echo "Display present working DIR"
cd user/class/b1
pwd
```

(iii)

```
echo "Enter a file name"
read file1

  echo "Enter contains in $file1"
  cat  > $file1
  echo "Start Printing at 5 line "
  tail +5 $file1
```

(iii) Print all the difference between two file, copy the two files at $USER/CSC/2007 directory.

Solution:-

```
echo "enter first file name"
read file1
echo "enter second file name"
read file2
echo "enter third file name"
read file3
echo "Enter contains to $file1"
cat> $ file1
echo "Enter contains to $file2"
cat> $ file2
echo "Display difference between $file1 and $file2  copy to $file3"
diff –a $file1 $file2 > $file3
cat $file3
```

iv)Print lines matching certain word pattern.

Solution :-

```
#mkdir IT
#cd IT
#vim assignmentno4.4.sh
echo “create a file “
read file1
echo “inputs contains in file $file1”
cat> $file1
echo “Enter word we findout “
read f
grep –ni $f $file1
```

2)Write shell script for-

· Showing the count of users logged in

· Printing Column list of files in your home directory.

· Listing your job with below normal priority

· Continue running your job after logging out

(i) Showing the count of users logged in,

```
sol-> echo " Show all users login"
who
echo " count all login name"
who |wc –l
```

(ii)Printing Column list of files in your home directory

```
sol -> echo " Printing 3-colomn in a Home directory"
ls – l | cut -17-24,39 - 42,56 -
```

(iii)Listing your job with below normal priority

```
Sol-> echo "list of normal priority "
ps – al
echo – al | cut – c26-29, 70 -
```

(iv)Continue running your job after logging out.

```
# nohup command-with-options &
```

note:- Nohup stands for no hang up

3) Write a shell script to count lines, words & characters in its input. (do not use wc)

3. SOURCE CODE:

```
read –p "create file name "
fname
echo "input the contains of file "
cat> $ fname
clear
echo " Display all record "
cat $fname
echo " show file line , word ,char"
gawk  '{ nc+=length ($0)+1nw +=NF}
END '{print " line =" NF , "\n word ="nw, "\n char ="nc}' $fname
```

Output

```
Create and enter file name
Display all records
11
22
33
```

4) Write a shell script to compute GCD & LCM of two numbers.

Write a shell script to compute GCD of two numbers.

```
echo Enter two numbers with space in between
read a b
m=$a
if [ $b -lt $m ]
then
m=$b
fi
while [ $m -ne 0 ]
do
x=`expr $a % $m`
y=`expr $b % $m`
if [ $x -eq 0 -a $y -eq 0 ]
then
echogcd of $a and $b is $m
break
fi
m=`expr $m - 1`
done
```

Write a shell script to compute LCM of two numbers.

```
echo "Enter first no"
read a
echo "Enter 2nd no"
read b
p= 'expr $a \* $b'
while [$b -ne 0]
do
r= 'expr $a % $b'
a=$b
b=$r
done
LCM = 'expr $p / $a'
echo "LCM = $LCM"
```

5) Write a shell script to find whether a given number is prime

```
i=2
rem=1
echo "Enter a number"
read num
if [ $num -lt 2 ]
then
echo -e "$num is not prime\n"
exit 0
fi
while [ $i -le `expr $num / 2` -a $rem -ne 0 ]
do
rem=`expr $num % $i`
i=`expr $i + 1`
done
if [ $rem -ne 0 ]
then
echo -e "$num is prime\n"
else
echo -e "$num is not prime\n"
fi
```

6)Shell script to perform database operations for student data like view, add and delete records in Unix / Linux.

2. SOURCE CODE:

```
clear
i="y"
echo "Enter name of database "
read db
while [ $i = "y" ]
do
clear
echo "1.View the Data Base "
echo "2.View Specific Records "
echo "3.Add Records "
echo "4.Delete Records "
echo "5.Exit "
echo "Enter your choice "
readch
case $ch in
1)cat $db;;
2)echo "Enter id "
read id
grep -i "$id" $db;;
3)echo "Enter new std id "
ireadtid
echo "Enter new name:"
readtnm
echo "Enter designation "
read des
echo "Enter college name"
read college
echo "$tid    $tnm    $des    $college">>$db;;
4)echo "Enter Id"
read id
        # set -a
    #    sed '/$id/d' $db>dbs1
grep -v "$id" $db>dbs1
echo "Record is deleted"
cat dbs1;;
5)exit;;
```

```
*)echo "Invalid choice ";;
esac
echo "Do u want to continue ?"

read i
if [ $i != "y" ]
then
exit
Fi
done
```

AWK command in Unix/Linux

1. AWK Operations:

(a) Scans a file line by line

(b) Splits each input line into fields

(c) Compares input line/fields to pattern

(d) Performs action(s) on matched lines

2. Useful For:

(a) Transform data files

(b) Produce formatted reports

3. Programming Constructs:

(a) Format output lines

(b) Arithmetic and string operations

(c) Conditionals and loops

Syntax:

```
awk options 'selection _criteria {action }' input-file > output-file
```

Options:

```
-f program-file : Reads the AWK program source from the file
                  program-file, instead of from the
                  first command line argument.
-F fs            : Use fs for the input field separator
```

Example:

Consider the following text file as the input file for all cases below:

```
$cat > employee.txt
```

```
ajay manager account 45000
sunil clerk account 25000
varun manager sales 50000
amit manager account 47000
tarun peon sales 15000
deepak clerk sales 23000
sunil peon sales 13000
satvik director purchase 80000
```

1. Default behavior of Awk: By default Awk prints every line of data from the specified file.

```
$ awk '{print}' employee.txt
```

Output:

```
ajay manager account 45000
sunil clerk account 25000
varun manager sales 50000
amit manager account 47000
tarun peon sales 15000
deepak clerk sales 23000
sunil peon sales 13000
satvik director purchase 80000
```

2. Print the lines which match the given pattern.

```
$ awk '/manager/ {print}' employee.txt
```

Output:

```
ajay manager account 45000
varun manager sales 50000
amit manager account 47000
```

3. Splitting a Line Into Fields : For each record i.e line, the awk command splits the record delimited by whitespace character by default and stores it in the $n variables. If the line has 4 words, it will be stored in $1, $2, $3 and $4 respectively. Also, $0 represents the whole line.

```
$ awk '{print $1,$4}' employee.txt
```

Output:

```
ajay 45000
sunil 25000
varun 50000
amit 47000
tarun 15000
deepak 23000
sunil 13000
satvik 80000
```

Built-In Variables In Awk

NR: NR command keeps a current count of the number of input records.

NF: NF command keeps a count of the number of fields within the current input record.
FS: FS command contains the field separator character which is used to divide fields on the input line. The default is “white space”, meaning space and tab characters. FS can be reassigned to another character (typically in BEGIN) to change the field separator.

RS: RS command stores the current record separator character. Since, by default, an input line is the input record, the default record separator character is a newline.
OFS: OFS command stores the output field separator, which separates the fields when Awk prints them. The default is a blank space. Whenever print has several parameters separated with commas, it will print the value of OFS in between each parameter.
ORS: ORS command stores the output record separator, which separates the output lines when Awk prints them. The default is a newline character. print automatically outputs the contents of ORS at the end of whatever it is given to print.

Examples:

Use of NR built-in variables (Display Line Number)

```
$ awk '{print NR,$0}' employee.txt
```

Output:

```
1 ajay manager account 45000
2 sunil clerk account 25000
3 varun manager sales 50000
4 amit manager account 47000
5 tarun peon sales 15000
6 deepak clerk sales 23000
7 sunil peon sales 13000
8 satvik director purchase 80000
```

Use of NF built-in variables (Display Last Field)

```
$ awk '{print $1,$NF}' employee.txt
```

Output:

```
ajay 45000
sunil 25000
varun 50000
amit 47000
tarun 15000
deepak 23000
sunil 13000
satvik 80000
```

Another use of NR built-in variables (Display Line From 3 to 6)

```
$ awk 'NR==3, NR==6 {print NR,$0}' employee.txt
```

Output:

```
3 varun manager sales 50000
4 amit manager account 47000
5 tarun peon sales 15000
6 deepak clerk sales 23000
```

For the given text file:

```
$cat > geeksforgeeks.txt

A    B    C
Tarun    A12    1
Man    B6    2
Praveen    M42    3
```

1) To print the first item along with the row number(NR) separated with " – " from each line in geeksforgeeks.txt:

```
$ awk '{print NR "- " $1 }' geeksforgeeks.txt
```

```
1 - A
2 - Tarun
3 - Manav
4 - Praveen
```

2) To return the second row/item from geeksforgeeks.txt:

The question should be:- To return the second column/item from geeksforgeeks.txt:

```
$ awk '{print $2}' geeksforgeeks.txt
```

```
A12
B6
M42
```

3) To print any non empty line if present

```
$ awk 'NF < 0' geeksforgeeks.txt
```

4) To find the length of the longest line present in the file:

```
$ awk '{ if (length($0) > max) max = length($0) } END { print max }' geeksforgee
```

```
13
```

5) To count the lines in a file:

```
$ awk 'END { print NR }' geeksforgeeks.txt
```

```
3
```

6) Printing lines with more than 10 characters:

```
$ awk 'length($0) > 10' geeksforgeeks.txt
```

```
Tarun    A12    1
Praveen    M42    3
```

7) To find/check for any string in any specific column:

```
$ awk '{ if($3 == "B6") print $0;}' geeksforgeeks.txt
```

8) To print the squares of first numbers from 1 to n say 6:

```
$ awk 'BEGIN { for(i=1;i<=6;i++) print "square of", i, "is",i*i; }'
```

```
square of 1 is 1
square of 2 is 4
square of 3 is 9
square of 4 is 16
square of 5 is 25
square of 6 is 36
```

grep command in Unix/Linux

The grep filter searches a file for a particular pattern of characters, and displays all lines that contain that pattern. The pattern that is searched in the file is referred to as the regular expression (grep stands for global search for regular expression and print out).

Syntax:

```
grep [options] pattern [files]
```

```
Options Description
-c : This prints only a count of the lines that match a pattern
-h : Display the matched lines, but do not display the filenames.
-i : Ignores, case for matching
-l : Displays list of a filenames only.
-n : Display the matched lines and their line numbers.
-v : This prints out all the lines that do not matches the pattern
-e exp : Specifies expression with this option. Can use multiple times.
-f file : Takes patterns from file, one per line.
-E : Treats pattern as an extended regular expression (ERE)
-w : Match whole word
-o : Print only the matched parts of a matching line,
 with each such part on a separate output line.

-A n : Prints searched line and nlines after the result.
-B n : Prints searched line and n line before the result.
-C n : Prints searched line and n lines after before the result.
```

Sample Commands

Consider the below file as an input.

```
$cat > geekfile.txt
```

```
unix is great os. unix is opensource. unix is free os.
learn operating system.
Unix linux which one you choose.
uNix is easy to learn.unix is a multiuser os.Learn unix .unix is a powerful.
```

1. Case insensitive search : The -i option enables to search for a string case insensitively in the given file. It matches the words like “UNIX”, “Unix”, “unix”.

```
$grep -i "UNix" geekfile.txt
```

Output:

```
unix is great os. unix is opensource. unix is free os.
Unix linux which one you choose.
uNix is easy to learn.unix is a multiuser os.Learn unix .unix is a powerful.
```

2. Displaying the count of number of matches : We can find the number of lines that matches the given string/pattern

```
$grep -c "unix" geekfile.txt
```

Output:

```
2
```

3. Display the file names that matches the pattern : We can just display the files that contains the given string/pattern.

```
$grep -l "unix" *

or

$grep -l "unix" f1.txt f2.txt f3.xt f4.txt
```

Output:

```
geekfile.txt
```

4. Checking for the whole words in a file : By default, grep 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 -w "unix" geekfile.txt
```

Output:

```
unix is great os. unix is opensource. unix is free os.
uNix is easy to learn.unix is a multiuser os.Learn unix .unix is a powerful.
```

5. Displaying only the matched pattern : By default, grep displays the entire line which has the matched string. We can make the grep to display only the matched string by using the -o option.

```
$ grep -o "unix" geekfile.txt
```

Output:

```
unix
unix
unix
unix
unix
unix
```

6. Show line number while displaying the output using grep -n : To show the line number of file with the line matched.

```
$ grep -n "unix" geekfile.txt
```

Output:

```
1:unix is great os. unix is opensource. unix is free os.
4:uNix is easy to learn.unix is a multiuser os.Learn unix .unix is a powerful.
```

7. Inverting the pattern match : You can display the lines that are not matched with the specified search string pattern using the -v option.

```
$ grep -v "unix" geekfile.txt
```

Output:

```
learn operating system.
Unix linux which one you choose.
```

8. Matching the lines that start with a string : The ^ regular expression pattern specifies the start of a line. This can be used in grep to match the lines which start with the given string or pattern.

```
$ grep "^unix" geekfile.txt
```

Output:

```
unix is great os. unix is opensource. unix is free os.
```

9. Matching the lines that end with a string : The $ regular expression pattern specifies the end of a line. This can be used in grep to match the lines which end with the given string or pattern.

```
$ grep "os$" geekfile.txt
```

10.Specifies expression with -e option. Can use multiple times :

```
$grep -e "Agarwal" -e "Aggarwal" -e "Agrawal" geekfile.txt
```

11. -f file option Takes patterns from file, one per line.

```
$cat pattern.txt

Agarwal
Aggarwal
Agrawal
```

```
$grep -f pattern.txt  geekfile.txt
```

12. Print n specific lines from a file: -A prints the searched line and n lines after the result, -B prints the searched line and n lines before the result, and -C prints the searched line and n lines after and before the result.

Syntax:

```
$grep -A[NumberOfLines(n)] [search] [file]

$grep -B[NumberOfLines(n)] [search] [file]

$grep -C[NumberOfLines(n)] [search] [file]
```

Example:

```
$grep -A1 learn geekfile.txt
```

Output:

```
learn operating system.
Unix linux which one you choose.
--
uNix is easy to learn.unix is a multiuser os.Learn unix .unix is a powerful.

(Prints the searched line along with the next n lines (here n = 1 (A1).)
(Will print each and every occurrence of the found line, separating each output
(Output pattern remains the same for -B and -C respectively)
```

13. Search recursively for a pattern in the directory: -R prints the searched pattern in the given directory recursively in all the files.

Syntax

```
$grep -R [Search] [directory]
```

Example :

```
$grep -iR geeks /home/geeks
```

Output:

```
./geeks2.txt:Well Hello Geeks
./geeks1.txt:I am a big time geek
---------------------------------
-i to search for a string case insensitively
-R to recursively check all the files in the directory.
```

- cd -- change directory
- pwd -- print working directory
- ls -- list all files and sub-directories in a directory
- stat -- display information about a file
- rm -- remove (i.e. delete) a file
- cp -- copy a file
- cat, more, less -- list the contents of a file
- chmod -- change file mode, i.e. file permissions
- ln -- create a link
- wc -- count words
- mkdir -- create a new directory
- head -- output only the first lines of a file
- tail -- output only the last lines of a file
- grep -- find a word in one or more files
- ps -- process status (lists running procceses, often run as `ps aux` for the most information)
- cut -- extract a column from a file
- sort -- sort a file alphabetically
- uniq -- remove adjacent duplicate lines

- # Intro to Pipes

- One extremely useful feature of shells are pipes. Pipes allow the pipeling of different shell commands by sending the output of one command to the input of another command. Pipes are created through the "|" symbol.
- For example, to count the number of lines produced by "ls -l" you can use the following pipeline:
- **`ls -l | wc`**
- You can also pipe more than two commands. For example, to display a sorted list of the last 10 lines of output of "ls -l" do:
- **`ls | tail -n 10 | sort`**