Week-7:

a) Pipe your /etc/passwd file to awk, and print out the home directory of each user.

b) Develop an interactive grep script that asks for a word and a file name and then tells how many lines contain that word.

a) Pipe your /etc/passwd file to awk, and print out the home directory of each user.

```
cat /etc/passwd | awk “{ print $7}”
```

or

```
$ vi home.awk
{
if(match ($0, /^.*home/) > 0)
{
split( $0, user)
split(user[1],homedir, ":")
print homedir[1]
}
}
```

To compile the program use

Sh filename.sh

To return the program

awk command :

```
$ cat /etc/passwd | awk -f home.awk
```

b) Develop an interactive grep script that asks for a word and a file name and then tells how many lines contain that word.

```
$ cat > file.txt
abc
pqr
xyz
abc
abc

$ vi grep.sh
echo "Enter the pattern to be searched: "
read pattern
echo "Enter the file to be used: "
read filename
echo "Searching for $pattern from file $filename"
echo "The selected records are: "
grep "$pattern" $filename
echo "The no. of lines contains the word ( $pattern ) :"
grep -c "$pattern" $filename
```

Output :

$ sh grep.sh

Enter the pattern to be searched:
abc
Enter the file to be used:
File.txt
Searching for abc from file file.txt
The selected records are:
abc
abc
abc
The no. of lines contains the words (abc) :3

awk command

Awk is one of the most powerful tools in Unix used for processing the rows and columns in a file. Awk has built in string functions and associative arrays. Awk supports most of the operators, conditional blocks, and loops available in C language.
Awk is a programming language which allows easy manipulation of structured data and the generation of formatted reports. Awk stands for the names of its authors "**A**ho,**W**einberger, and **K**ernighan"
One of the good things is that you can convert Awk scripts into Perl scripts using a2p utility.
The basic syntax of AWK:
awk 'BEGIN {start_action} {action} END {stop_action}' filename
Here the actions in the begin block are performed before processing the file and the actions in the end block are performed after processing the file. The rest of the actions are performed while processing the file.
Awk Working Methodology

- Awk reads the input files one line at a time.
- For each line, it matches with given pattern in the given order, if matches performs the corresponding action.
- If no pattern matches, no action will be performed.
- In the above syntax, either search pattern or action are optional, But not both.
- If the search pattern is not given, then Awk performs the given actions for each line of the input.
- If the action is not given, print all that lines that matches with the given patterns which is the default action.
- Empty braces with out any action does nothing. It wont perform default printing operation.
- Each statement in Actions should be delimited by semicolon.

Let us create employee.txt file which has the following content, which will be used in the examples mentioned below.

$cat employee.txt
100 Thomas Manager Sales $5,000
200 Jason Developer Technology $5,500
300 Sanjay Sysadmin Technology $7,000
400 Nisha Manager Marketing $9,500
500 Randy DBA Technology $6,000

Example 1. Default behavior of Awk

By default Awk prints every line from the file.

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

100 Thomas Manager Sales $5,000
200 Jason Developer Technology $5,500
300 Sanjay Sysadmin Technology $7,000
400 Nisha Manager Marketing $9,500
500 Randy DBA Technology $6,000
```

In the above example pattern is not given. So the actions are applicable to all the lines. Action print with out any argument prints the whole line by default. So it prints all the lines of the file with out fail. Actions has to be enclosed with in the braces.

Awk Example 2. Print the lines which matches with the pattern.

```
$ awk '/Thomas/>/Nisha/' employee.txt
100 Thomas Manager Sales $5,000
400 Nisha Manager Marketing $9,500
```

In the above example it prints all the line which matches with the 'Thomas' or 'Nisha'. It has two patterns. Awk accepts any number of patterns, but each set (patterns and its corresponding actions) has to be separated by newline.

Awk Example 3. Print only specific field.
Awk has number of built in variables. For each record i.e line, it 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. $0 represents whole line. NF is a built in variable which represents total number of fields in a record.

```
$ awk '{print $2,$5;}' employee.txt

Thomas $5,000
Jason $5,500
Sanjay $7,000
Nisha $9,500
Randy $6,000

$ awk '{print $2,$NF;}' employee.txt

Thomas $5,000
Jason $5,500
Sanjay $7,000
Nisha $9,500
Randy $6,000
```

In the above example $2 and $5 represents Name and Salary respectively. We can get the Salary using $NF also, where $NF represents last field. In the print statement ',' is a

concatenator.

Awk Example 4. Initialization and Final Action
Awk has two important patterns which are specified by the keyword called BEGIN and END.

Syntax:
BEGIN { Actions}
{ACTION} # Action for everyline in a file
END { Actions }
is for comments in Awk
Actions specified in the BEGIN section will be executed before starts reading the lines from the input.
END actions will be performed after completing the reading and processing the lines from the input.

```
$ awk 'BEGIN {print "Name\tDesignation\tDepartment\tSalary";}
> {print $2,"\t",$3,"\t",$4,"\t",$NF;}
> END{print "Report Generated\n--------------";
> }' employee.txt
```

Name Designation Department Salary
Thomas Manager Sales $5,000
Jason Developer Technology $5,500
Sanjay Sysadmin Technology $7,000
Nisha Manager Marketing $9,500
Randy DBA Technology $6,000
Report Generated

In the above example, it prints headline and last file for the reports.

Awk Example 5. Find the employees who has employee id greater than 200

```
$ awk '$1 >200' employee.txt
```

300 Sanjay Sysadmin Technology $7,000
400 Nisha Manager Marketing $9,500
500 Randy DBA Technology $6,000

In the above example, first field ($1) is employee id. So if $1 is greater than 200, then just do the default print action to print the whole line.

Awk Example 6. Print the list of employees in Technology department

Now department name is available as a fourth field, so need to check if $4 matches with the string “Technology”, if yes print the line.

```
$ awk '$4 ~/Technology/' employee.txt
```

200 Jason Developer Technology $5,500
300 Sanjay Sysadmin Technology $7,000
500 Randy DBA Technology $6,000
Operator ~ is for comparing with the regular expressions. If it matches the default action i.e

print whole line will be performed.

Awk Example 7. Print number of employees in Technology department

The below example, checks if the department is Technology, if it is yes, in the Action, just increment the count variable, which was initialized with zero in the BEGIN section.

```
$ awk 'BEGIN { count=0;}
$4 ~ /Technology/ { count++; }
END { print "Number of employees in Technology Dept =",count;}' employee.txt
```

grep command

The ***grep*** command allows you to search one file or multiple files for lines that contain a pattern. Exit status is 0 if matches were found, 1 if no matches were found, and 2 if errors occurred.

SYNTAX

The syntax for the ***grep*** command is:

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

OPTIONS

Option	Description
-b	Display the block number at the beginning of each line.
-c	Display the number of matched lines.
-h	Display the matched lines, but do not display the filenames.
-i	Ignore case sensitivity.
-l	Display the filenames, but do not display the matched lines.
-n	Display the matched lines and their line numbers.
-s	Silent mode.
-v	Display all lines that do NOT match.
-w	Match whole word.

EXAMPLE

grep chope /etc/passwd

Search **/etc/passwd** for user **chope**.

grep -r "computerhope" /www/

Recursively search the directory **/www/**, and all subdirectories, for any lines of any files which contain the string "**computerhope**".

grep -w "hope" myfile.txt

Search the file **myfile.txt** for lines containing the word "**hope**". Only lines containing the distinct word "hope" will be matched. Lines in which "hope" is *part* of a word will *not* be matched.

grep -cw "hope" myfile.txt

Same as previous command, but displays a count of how many lines were matched, rather than the matching lines themselves.
grep -cvw "hope" myfile.txt
Inverse of previous command: displays a count of the lines in **myfile.txt** which do *not*contain the word "hope".
grep -l "hope" /www/*
Display the filenames (but not the matching lines themselves) of any files in **/www/**(but not its subdirectories) whose contents include the string "**hope**".

How do I use grep command to search a file?

Search /etc/passwd file for pravin user, enter:

$ grep pravin /etc/passwd
Sample outputs:
pravin:x:1000:1000:,,,:/home/pravin:/bin/bash
You can force grep to ignore word case i.e match pravin, Pravin, PRAVIN and all other combination
with the -i option:

$ grep -i "pravin" /etc/passwd

Use grep recursively

You can search recursively i.e. read all files under each directory for a string "192.168.1.5"

$ grep -r "192.168.1.5" /etc/

OR

$ grep -R "192.168.1.5" /etc/

The syntax is as follows:
grep 'word' filename
grep 'word' file1 file2 file3
grep 'string1 string2' filename
cat otherfile | **grep** 'something'
command | **grep** 'something'
command option1 | **grep** 'data'
grep --color 'data' filename