Assignment No. 8 Date:

Objective : To study and implement Unix basic commands such as pwd, ls, cat, redirection and pipes and Unix utilities like tr, sed, grep, egrep, awk

Theory :

The following command is utilised to display the present working directory

1. pwd :

Syntax: pwd

2. ls :

It is a command line utility that lists the content/files listed in a particular Unix directory. ls command with no parameters the command displays the files listed in your current working directory. With ls command, you can add one or more modifiers to get additional information.

- List contents of a directory
- ls is a command to list computer files in Unix

Syntax :
ls [option] [filenames]

Command	**Use**
ls -a	Displays all files including hidden files
ls -l	Long listing- The default output of the ls command shows only the names of the files and directories. The -l option shows files in a long listing format with details like file type, file permissions, Number of hard links to the file, file owner, file group, file size, Date and Time, file name.
ls -i	Displays list of directory contents with inode number
ls -r	Lists directory contents in reverse order.
ls -t	Sort the data by modification date

Here is an example:

```
ls -l /etc/hosts
```

```
-rw-r--r-- 1 root root 317 Dec 4 11:31 /etc/hosts
```

3. cat :

The cat ("concatenate") command is one of the most frequently used command in Unix. cat command allows us to create single or multiple files, view content of file, concatenate files and redirect output in terminal or files.

- Concatenate and display files
- It reads data from the file and gives their content as output.

Syntax:
cat [option...] [file ...]

Command	**Description**
$ cat filename	To view a single file. It will show content of given filename
$ cat file1 file2	View multiple files. Shows the content of file1 and file2.
$ cat -n filename	Shows contents of a file preceding with line numbers
$ cat > new1	Create file named new1
$ cat [sourceFile] > [destFile]	Copy the contents of one file to another file. The content will be copied to the destination file.
$ cat -s filename	Suppress repeated empty lines from the file.
$ cat file1 >> file2	Append the contents of one file to the end of another file
$ cat "file1" "file2" "file3" > "mergedFile"	Merge the contents of multiple files

4. Redirection :
Syntax:
A command can take input, give output, or do both. Redirection helps us redirect these input and output functionalities to the files or folders we want, and we can use special commands or characters to do so.

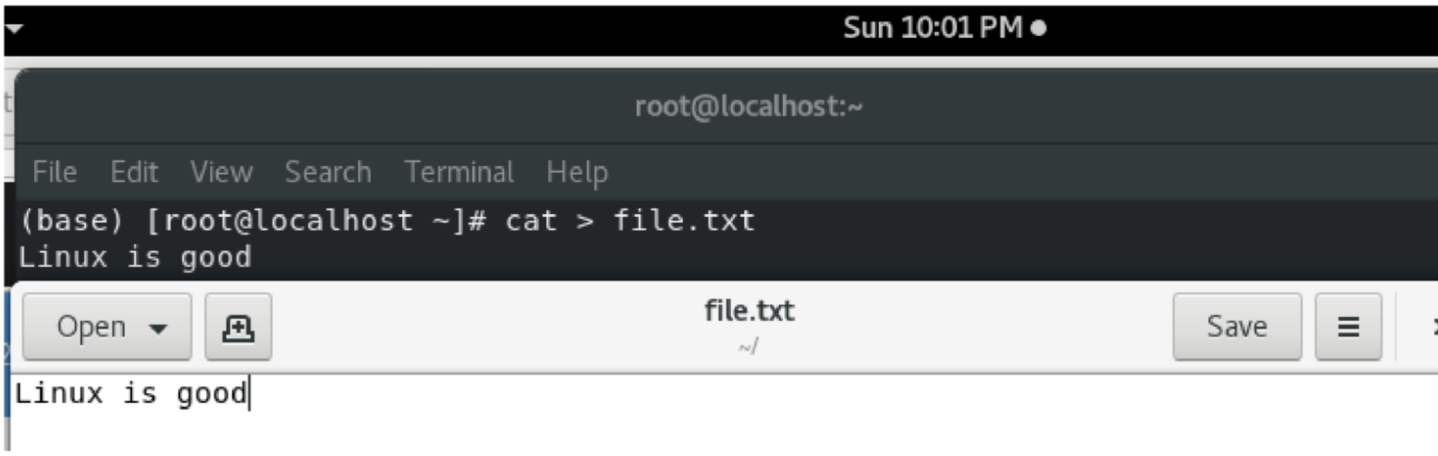
For example, if we run the "date" command, it gives us output on the screen. However, if we want to save the result in a file instead, we can use output redirection. This way, we can store the output of the date command in a file and refer to it later. These redirections can come in handy when we work with multiple and large outputs or inputs since we can use file data directly as input and store results in files.

All this can be done easily on the terminal using some simple commands.

Types of Redirection

1. **Overwrite Redirection:**

Overwrite redirection is useful when you want to store/save the output of a command to a file and replace all the existing content of that file. for example, if you run a command that gives a report, and you want to save the report to the existing file of the previous report you can use overwrite redirection to do this.



```
(base) [root@localhost ~]# cat > file.txt
Linux is good
```

```
Linux is good
```

- ">" standard output
- "<" standard input

Implementation:

So whatever you will write after running this command, will be redirected and copied to the "file.txt". This is standard output redirection.

cat > file.txt

Now, this is standard input redirection, here cat command will take the input from "file.txt" and print it to the terminal screen. The meaning of the cat command is copy and paste.

2. **Append Redirection:**

With the help of this Redirection, you can append the output to the file without compromising the existing data of the file.

- ">>" standard output
- "<<" standard input

3. **Merge Redirection:**

This allows you to redirect the output of a command or a program to a specific file descriptor instead of standard output. the syntax for using this is ">&" operator followed by the file descriptor number.

- "p >& q" Merges output from stream p with stream q
- "p <& q" Merges input from stream p with stream q

Pipes :

Pipes connect the standard output of one command to the standard input of another. Separate the two commands with the pipe symbol (|). Here's an example:

Syntax:

```
$ echo "hello there"
hello there
$ echo "hello there" | sed "s/hello/hi/"

hi there
```

echo "hello there" prints hello there to stdout. But when we pipe it to sed "s/hello/hi/", sed takes that output as its input and replaces "hello" with "hi", then prints out that result to stdout. Shell only sees the final result after it's been processed by sed, and prints that result to the screen.

If sed sends its result to standard out, we can pipe sed to another sed.

```
$ echo "hello there" | sed "s/hello/hi/" | sed "s/there/robots/" hi
robots
```

Above, we've connected echo to sed, then connected that to another sed. Pipes are great for taking output of one command and transforming it using other commands.

Unix Utilities :

tr :
The tr command is a UNIX command-line utility for translating or deleting characters. It supports a range of transformations including uppercase to lowercase, squeezing repeating characters, deleting specific characters, and basic find and replace. It can be used with UNIX pipes to support more complex translation. tr stands for translate.

Syntax :

```
$ tr [OPTION] SET1 [SET2]
```

Options
-c : complements the set of characters in string. It means operations apply to characters not in the given set
-d : delete characters in the first set from the output.
-s : replaces repeated characters listed in the set1 with single occurrence -
t : truncates set1.

1. **How to convert lower case characters to upper case. To convert characters from lower case to upper case, you can either specify a range of characters or use the predefined character classes. $ cat greekfile**

Output:

```
WELCOME TO
```

```
GeeksforGeeks
```

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

Output:

```
WELCOME TO
GEEKSFORGEEKS
```

or

```
$ cat greekfile | tr [:lower:] [:upper:]
```

Output:

```
WELCOME TO
GEEKSFORGEEKS
```

Alternatively, you can provide input for the tr command using redirection:

```
tr [:lower:] [:upper:] <greekfile
```

Output:

```
WELCOME TO
GEEKSFORGEEKS
```

2. **How to translate white-space characters to tabs. The following command translates all the white-space characters to tabs**

```
$ echo "Welcome To GeeksforGeeks" | tr [:space:] "\t"
```

Output:

```
Welcome    To    GeeksforGeeks
```

In the previous example we can also use redirection to provide intput for tr. Although this time we will use a here string for that:

```
 tr [:space:] "\t" <<< "Welcome To GeeksforGeeks"
```

Output:

```
Welcome    To    GeeksforGeeks
```

3. **How to translate braces into parenthesis. You can also translate from and to a file. In this example we will translate braces in a file with parenthesis.**

```
$ cat greekfile
```

Output:

```
{WELCOME TO}
GeeksforGeeks
```

```
$ tr "{}" "()" <greekfile >newfile.txt
```

Output:

```
(WELCOME TO)
GeeksforGeeks
```

The above command will read each character from “geekfile.txt”, translate if it is a brace, and write the output to “newfile.txt”.

4. **How to squeeze a sequence of repetitive characters using -s option. To squeeze repetitive occurrences of characters specified in a set use the -s option. This removes repeated instances of characters of the last SET specified. OR we can say that, you can convert multiple continuous spaces with a single space**

```
$ echo "Welcome    To    GeeksforGeeks" | tr -s " "
```

Output:

```
Welcome To GeeksforGeeks
```

And again, accomplish the same task but using a string here:

```
tr -s " " <<< "Welcome           To        GeeksforGeeks"
```

Output:

```
Welcome To GeeksforGeeks
```

5. **How to delete specified characters using -d option. To delete specific characters use the -d option. This option deletes characters in the first set specified.**

```
$ echo "Welcome To GeeksforGeeks" | tr -d W
```

Output:

```
elcome To GeeksforGeeks
```

Or equivalently use:

```
tr -d W <<< "Welcome to GeeksforGeeks"
```

Output:

```
elcome To GeeksforGeeks
```

6. **To remove all the digits from the string, you can use**

```
 $ echo "my ID is 73535" | tr -d [:digit:]
```

or

```
$ tr -d [:digit:] <<< "my ID is 73535"
```

Output:

```
my ID is
```

7. **How to complement the sets using -c option You can complement the SET1 using -c option. For example, to remove all characters except digits, you can use the following.**

```
$ echo "my ID is 73535" | tr -cd [:digit:]
```

or

```
$ tr -cd [:digit:] <<< "my ID is 73535"
```

Output:

```
73535
```

sed :

SED command in UNIX stands for stream editor.

- SED is a powerful text stream editor. Can do insertion, deletion, search and replace(substitution).
- SED command in unix supports regular expression which allows it perform complex pattern matching.

Syntax:

```
sed OPTIONS... [SCRIPT] [INPUTFILE...]
```

Example:
Consider the below text 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.
```

S**ample Commands**

1. **Replacing or substituting string** : Sed command is mostly used to replace the text in a file. The below simple sed command replaces the word “unix” with “linux” in the file.

```
$sed 's/unix/linux/' geekfile.txt
```

Output :

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

Here the “s” specifies the substitution operation. The “/” are delimiters. The “unix” is the search pattern and the “linux” is the replacement string.

By default, the sed command replaces the first occurrence of the pattern in each line and it won’t replace the second, third…occurrence in the line.

2. **Replacing the nth occurrence of a pattern in a line** : Use the /1, /2 etc flags to replace the first, second occurrence of a pattern in a line. The below command replaces the second occurrence of the word “unix” with “linux” in a line.

```
$sed 's/unix/linux/2' geekfile.txt
```

Output:

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

3. **Deleting lines from a particular file** : SED command can also be used for deleting lines from a particular file. SED command is used for performing deletion operation without even opening the file

Examples:

1. To Delete a particular line say n in this example

Syntax:
$ sed 'nd' filename.txt Example:
$ sed '5d' filename.txt

2. To Delete a last line

Syntax:
$ sed '$d' filename.txt

3. To Delete line from range x to y

Syntax:
$ sed 'x,yd' filename.txt Example:
$ sed '3,6d' filename.txt

4. To Delete pattern matching line

Syntax:
$ sed '/pattern/d' filename.txt
Example:

```
$ sed '/abc/d' filename.txt
```

grep :

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 printout).

Syntax: grep [options] pattern [files]
Here,

[options]: These are command-line flags that modify the behavior of grep.

[pattern]: This is the regular expression you want to search for.

[file]: This is the name of the file(s) you want to search within. You can specify multiple files for simultaneous searching.

Options in grep Command

Options	**Description**
-c	This prints only a count of the lines that match a pattern
-l	Displays list of a filenames only.

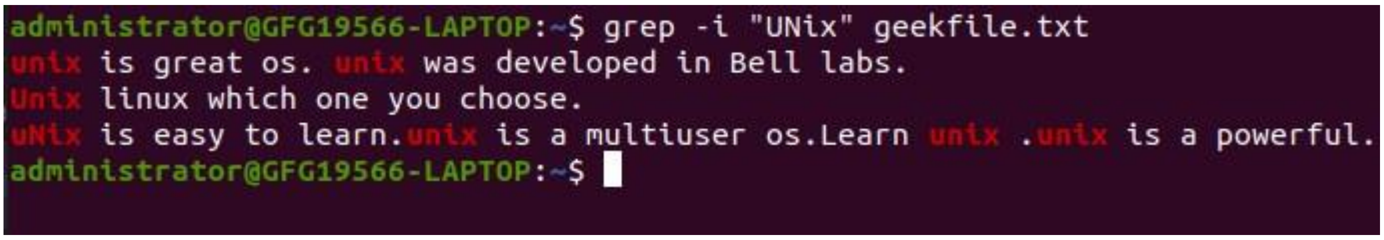
Options	**Description**
-n	Display the matched lines and their line numbers.
-v	This prints out all the lines that do not matches the pattern
-e exp	Specifies expression with this option. Can use multiple times.
-f file	Takes patterns from file, one per line.
-w	Match whole word
-o	Print only the matched parts of a matching line, with each such part on a separate output line.

Example of grep Command in Linux

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:



```
administrator@GFG19566-LAPTOP:~$ grep -i "UNix" geekfile.txt
unix is great os. unix was developed in Bell labs.
Unix linux which one you choose.
uNix is easy to learn.unix is a multiuser os.Learn unix .unix is a powerful.
administrator@GFG19566-LAPTOP:~$ 
```

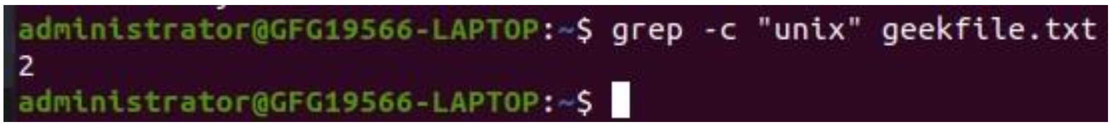
Case insensitive search

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:



```
administrator@GFG19566-LAPTOP:~$ grep -c "unix" geekfile.txt
2
administrator@GFG19566-LAPTOP:~$ 
```

Displaying the count number of the matches

3. **Display the file names that matches the pattern**

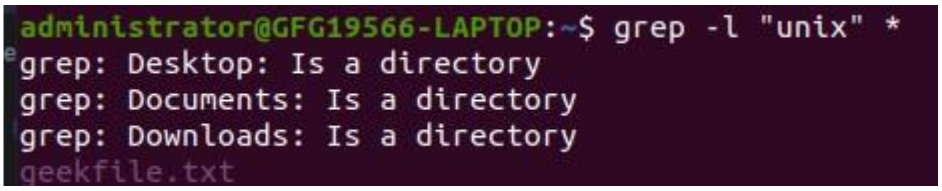
We can just display the files that contains the given string/pattern.

grep -l "unix" *

or grep -l "unix" f1.txt f2.txt f3.xt

f4.txt

Output:



```
administrator@GFG19566-LAPTOP:~$ grep -l "unix" *
grep: Desktop: Is a directory
grep: Documents: Is a directory
grep: Downloads: Is a directory
geekfile.txt
```

egrep :
On Unix-like operating systems, the egrep command searches for a text pattern, using extended regular expressions to perform the match. Running egrep is equivalent to running grep with the -E option.
Syntax: egrep [*options*] *PATTERN* [*FILE*...]
Options

-A *NUM*, --after-context=*NUM*	Print *NUM* lines of trailing context after matching lines. Places a line containing -- between contiguous groups of matches.
-a, --text	Process a binary file as if it were text; this is equivalent to the --binary-files=text option.

--binary-files=*TYPE*	If the first few bytes of a file indicate that the file contains binary data, assume that the file is of type *TYPE*. By default, *TYPE* is binary, and grep normally outputs either a one-line message saying that a binary file matches, or no message if there is no match.
--colour[=*WHEN*], --color[=*WHEN*]	Surround the matching string with the marker find in GREP_COLOR environment variable. *WHEN* may be 'never', 'always', or 'auto'
-c, --count	Suppress normal output; instead print a count of matching lines for each input file. With the -v, --invert-match option (see below), count non-matching lines.
-D *ACTION*, --devices=*ACTION*	If an input file is a device, FIFO or socket, use *ACTION* to process it. By default, *ACTION* is read, which means that devices are read as if they were ordinary files. If *ACTION* is skip, devices are silently skipped.
-e *PATTERN*, --regexp=*PATTERN*	Use *PATTERN* as the pattern; useful to protect patterns beginning with "-".
-F, --fixed-strings	Interpret *PATTERN* as a list of fixed strings, separated by newlines, that may be matched.
-f *FILE*, --file=*FILE*	Obtain patterns from *FILE*, one per line. The empty file contains zero patterns, and therefore matches nothing.

awk :

The awk command is a Linux tool and programming language that allows users to process and manipulate data and produce formatted reports. The tool supports various operations for advanced text processing and facilitates expressing complex data selections.

Syntax:

```
awk [options] 'selection_criteria {action}' input-file > output-file
```

The available options are:

Option	Description
-F [separator]	Used to specify a file separator. The default separator is a blank space.
-f [filename]	Used to specify the file containing the awk script. Reads the awk program source from the specified file, instead of the first command-line argument.
-v	Used to assign a variable.

The awk command's main purpose is to make information retrieval and text manipulation easy to perform in Linux. This Linux command works by scanning a set of input lines in order and searches for lines matching the patterns specified by the user.

For each pattern, users can specify an action to perform on each line that matches the specified pattern. Thus, using awk, users can easily process complex log files and output a readable report.

AWK Statements

The command provides basic control flow statements (if-else, while, for, break) and also allows users to group statements using braces {}.

- **if-else**

The if-else statement works by evaluating the condition specified in the parentheses and, if the condition is true, the statement following the if statement is executed. The else part is optional.

For example:

```
awk -F ',' '{if($2==$3){print $1","$2","$3} else {print "No Duplicates"}}' answers.txt
```

- **while**

The while statement repeatedly executes a target statement as long as the specified condition is true. That means that it operates like the one in the C programming language. If the condition is true, the body of the loop is executed. If the condition is false, awk continues with the execution.

For example, the following statement instructs awk to print all input fields one per line:

```
 awk '{i=0; while(i<=NF) { print i ":"$i; i++;}}' employees.txt
```

- **for**

The for statement also works like that of C, allowing users to create a loop that needs to execute a specific number of times.

For example:

```
awk 'BEGIN{for(i=1; i<=10; i++) print "The square of", i, "is", i*i;}'
```

break

The break statement immediately exits from an enclosing while or for. To begin the next iteration, use the continue statement.

The next statement instructs awk to skip to the next record and begin scanning for patterns from the top. The exit statement instructs awk that the input has ended.

Following is an example of the break statement:

```
awk 'BEGIN{x=1; while(1) {print "Example"; if ( x==5 ) break; x++; }}'
```