Lab 1

2)

a. cat variation used to create a new file and append to existing file:

* To create a new file: cat > newfile.txt
* To append to an existing file: cat >> existingfile.txt

b. head and tail with options:

* To display the first 10 lines of a file: head file.txt
* To display the last 10 lines of a file: tail file.txt
* To display the first 5 lines of a file: head -n 5 file.txt
* To display the last 100 bytes of a file: tail -c 100 file.txt

c. cp with options:

* To copy a file from one location to another: cp file1.txt /path/to/destination/
* To copy a file and prompt before overwriting an existing file: cp -i file1.txt /path/to/destination/
* To force copy a file, even if the destination file already exists: cp -f file1.txt /path/to/destination/
* To copy multiple files to a directory: cp file1.txt file2.txt /path/to/directory/

d. mv with options:

* To move a directory to a new location: mv /path/to/dir1 /path/to/dir2
* To move a file and prompt before overwriting an existing file: mv -i file1.txt /path/to/destination/
* To force move a file, even if the destination file already exists: mv -f file1.txt /path/to/destination/
* To move multiple files to a directory: mv file1.txt file2.txt /path/to/directory/

e. rm with options:

* To remove a file: rm file.txt
* To prompt before deleting a file: rm -i file.txt
* To force delete a file, without prompting: rm -f file.txt
* To remove a directory and its contents: rm -r directory/

f. rmdir with options:

* To remove an empty directory: rmdir empty\_directory/
* To force remove a non-empty directory: rmdir -r non\_empty\_directory/
* To force remove a non-empty directory without prompting: rmdir -rf non\_empty\_directory/

g. find with options:

* To find all files with a specific name: find /path/to/directory/ -name "file.txt"
* To find all files of a specific type: find /path/to/directory/ -type f
* To find all directories: find /path/to/directory/ -type d

3)

To list all the file names satisfying the given criteria, we can use the find command with appropriate options. Here is the command:  
  
find /path/to/directory -type f -name "\*.txt" -regex '.\*[0-9].\*' -size +3c ! -name '[aeiou]\*'  
  
Here, /path/to/directory should be replaced with the path to the directory where we want to search for files. The options used in the command are as follows:

* -type f: limits the search to regular files only.
* -name "\*.txt": searches for files with the extension .txt.
* -regex '.\*[0-9].\*': searches for files that contain at least one digit.
* -size +3c: searches for files with a minimum size of 4 bytes (the "+3c" is used to specify a size of at least 3 bytes, since the minimum length is 4).
* ! -name '[aeiou]\*': excludes files that start with a vowel (a, e, i, o, or u).

The above command will list all the file names that satisfy the given criteria in the specified directory and its subdirectories.

Lab 2

1)  
Certainly! Here are some common use cases for the commands I mentioned:

1. **grep**: Used for searching for specific text in files. For example, if you wanted to find all lines in a log file that contain the word "error", you could use **grep "error" logfile.txt**.
2. **sort**: Used for sorting the contents of a file. For example, if you had a file with a list of names and you wanted to sort them in alphabetical order, you could use **sort names.txt**.
3. **wc**: Used for counting the number of lines, words, and characters in a file. For example, if you had a text file and you wanted to know how many lines it contains, you could use **wc -l file.txt**.
4. **cut**: Used for extracting specific columns or fields from a file. For example, if you had a CSV file and you wanted to extract the first and third columns, you could use **cut -d',' -f1,3 file.csv**.
5. **sed**: Used for performing text substitutions and transformations. For example, if you wanted to replace all occurrences of the word "cat" with "dog" in a file, you could use **sed 's/cat/dog/g' file.txt**.
6. **tr**: Used for translating or deleting characters in a file. For example, if you wanted to convert all lowercase letters to uppercase in a file, you could use **tr '[:lower:]' '[:upper:]' < file.txt**.
7. **ps**: Used for listing the currently running processes on a system. For example, if you wanted to see all of the processes that are currently running on your system, you could use **ps -ef**.
8. **kill**: Used for sending a signal to a process to terminate it. For example, if a process had become unresponsive and you wanted to force it to quit, you could use **kill <process id>**.
9. **chmod**: Used for changing the permissions of a file or directory. For example, if you wanted to give the owner of a file read, write, and execute permissions and everyone else read and execute permissions, you could use **chmod 755 file.txt**.
10. **echo**: Used for printing text to the terminal. For example, if you wanted to print a message to the user, you could use **echo "Hello, world!"**.
11. **bc**: Used for performing arithmetic operations. For example, if you wanted to add two numbers together, you could use **echo "2 + 2" | bc**.
12. **vi**: Used for editing text files. For example, if you wanted to make changes to a configuration file, you could use **vi config.txt**.

2)

Here are the grep commands to perform the following activities:

1. To select the lines from a file that have exactly two characters:

grep -E '^.{2}$' filename  
  
Here, ^.{2}$ is the regular expression pattern that matches lines with exactly two characters.

1. To select the lines from a file that start with the upper case letter:

grep '^[A-Z]' filename

Here, ^[A-Z] is the regular expression pattern that matches lines that start with an uppercase letter.

1. To select the lines from a file that end with a period:

grep '\.$' filename

Here, \.$ is the regular expression pattern that matches lines that end with a period.

1. To select the lines in a file that have one or more blank spaces:  
   grep ' ' filename  
   Here, the regular expression pattern is simply a space character, which matches lines that have one or more blank spaces.
2. To select the lines in a file and direct them to another file which has digits as one of the characters in that line:

grep '[0-9]' filename > outputfile  
Here, [0-9] is the regular expression pattern that matches lines that have at least one digit, and > outputfile directs the output to a new file named outputfile.

3)  
To create the studentInformation.txt file using the vi editor, you can follow these steps:

1. Open the terminal and type the following command to open the file in the vi editor:

* vi studentInformation.txt
* Press the i key to enter the insert mode.
* Type in the data in the following format:

ruby

1. RegistrationNo:Name:Department:Branch:Section:Sub1:Sub2:Sub3

1234:XYZ:ICT:CCE:A:80:60:70

5678:ABC:CSE:IT:B:70:80:90

9012:PQR:IT:EEE:A:60:70:80

...

1. Press the Esc key to exit the insert mode.
2. Type :wq and press Enter to save and exit the file.

Now, you can use the following grep and awk commands to perform the required tasks:

i) Display the number of students (only count) belonging to the ICT department.

bash

grep 'ICT' studentInformation.txt | wc -l

Here, grep 'ICT' studentInformation.txt matches all the lines in the file that contain the string 'ICT', and wc -l counts the number of matching lines.

ii) Replace all occurrences of IT branch with “Information Technology” and save the output to ITStudents.txt

python

sed 's/IT/Information Technology/g' studentInformation.txt > ITStudents.txt

Here, sed 's/IT/Information Technology/g' studentInformation.txt replaces all occurrences of 'IT' with 'Information Technology' in the studentInformation.txt file, and > ITStudents.txt saves the output to a new file named ITStudents.txt.

iii) Display the average marks of the student with the given registration number "1234" (or any specific existing number).

swift

awk -F: '$1=="1234"{print ($6+$7+$8)/3}' studentInformation.txt

Here, awk -F: '$1=="1234"{print ($6+$7+$8)/3}' studentInformation.txt uses the awk command to match the line that starts with '1234' in the first column and then prints the average of the marks in the 6th, 7th, and 8th columns.

iv) Display the title row in uppercase. The remaining lines should be unchanged.

bash

head -l studentInformation.txt | tr [:lower:] [:upper:]

tail -n +2 studentInformation.txt

Here, head -1 studentInformation.txt extracts the first line (i.e., the title row), and tr [:lower:] [:upper:] converts all lowercase letters to uppercase. tail -n +2 studentInformation.txt extracts all the remaining lines (i.e., from the second line onwards).

3)   
To list all the files containing "MIT" in the current folder, you can use the following command:

bash

ls | grep MIT

To display the lines containing "MIT" being replaced with "Manipal Institute of Technology", you can use the following command:

bash

grep -r "MIT" . | cut -d ":" -f 1 | uniq | xargs sed -i 's/MIT/Manipal Institute of Technology/g'

Here, the grep -r "MIT" . command searches for all lines containing "MIT" in the current directory and its subdirectories, and the cut -d ":" -f 1 command extracts the file names from the output. The uniq command removes duplicate file names, and xargs sed -i 's/MIT/Manipal Institute of Technology/g' replaces all occurrences of "MIT" with "Manipal Institute of Technology" in those files.

4)  
  
To display the number of lines, characters, and words of files containing a digit in its name, you can use the following command:

bash

wc `find . -name '\*[0-9]\*'`

Here, the find . -name '\*[0-9]\*' command searches for all files containing a digit in its name in the current directory and its subdirectories, and the output is piped to the wc command, which counts the number of lines, characters, and words in those files.

5)  
  
  
To run the wc command in the background many times, you can use the following command:

bash

for i in {1..10}; do wc & done

This will run the wc command in the background 10 times.

To kill all the processes named wc, you can use the pkill command as follows:

bash

pkill wc

This will send a signal to all the processes named wc to terminate them.

Additional 1)  
To delete the character before the last character in each line of a file using sed, you can use the following command:

rust

sed 's/.\{2\}$//g' filename

Here, s/.\{2\}$//g is the sed substitution command. It matches any character followed by the last character in each line, and replaces it with an empty string, effectively deleting the second-last character.

The option {2} specifies the number of characters to match, and $ indicates the end of the line. The g option is used to apply this substitution globally to all matching patterns in the file.

Replace filename with the name of the file you want to modify. This command will print the modified content to the standard output. If you want to modify the file in place, you can add the -i option:

rust

sed -i 's/.\{2\}$//g' filename

Note that this command will modify the file directly, so be sure to make a backup copy of the file before running the command.

Additional 2)

You can use the grep and wc commands to count the number of lines containing digits in a file. The grep command can be used to search for lines containing digits using a regular expression, and the wc command can be used to count the number of lines in the output of the grep command.

Here's an example command:

perl

grep -c '[0-9]' filename.txt

This command searches for lines containing digits in the filename.txt file using the regular expression [0-9], and the -c option of the grep command counts the number of matching lines. The output of this command will be the number of lines containing digits in the file.

Lab 3)   
  
1. Write a shell script to find whether a given file is the directory or regular file.

bash

#!/bin/bash

echo "Enter file name: "

read file

if [ -f "$file" ]; then

echo "$file is a regular file."

elif [ -d "$file" ]; then

echo "$file is a directory."

else

echo "$file is not a valid file or directory."

fi

1. Write a shell script to list all files (only file names) containing the input pattern (string) in the folder entered by the user.

bash

#!/bin/bash

echo "Enter the directory path: "

read dir

echo "Enter the search pattern: "

read pattern

if [ -d "$dir" ]; then

find "$dir" -type f -name "\*$pattern\*" -printf "%f\n"

else

echo "$dir is not a valid directory."

fi

1. Write a shell script to replace all files with .txt extension with .text in the current directory. This has to be done recursively i.e if the current folder contains a folder “OS” with abc.txt then it has to be changed to abc.text ( Hint: use find, mv )

bash

#!/bin/bash

find . -type f -name "\*.txt" -exec sh -c 'mv "$0" "${0%.txt}.text"' {} \;

1. Write a shell script to calculate the gross salary. GS=Basics + TA + 10% of Basics. Floating point calculations has to be performed.

bash

#!/bin/bash

echo "Enter basic salary: "

read basic

echo "Enter travel allowance: "

read ta

gs=$(echo "scale=2; $basic + $ta + $basic\*0.1" | bc)

echo "Gross salary: $gs”

5. Here's a shell script that achieves the desired task:

bash

#!/bin/bash

# Prompt user to enter file extension and target directory name

echo "Enter file extension (without dot): "

read extension

echo "Enter target directory name: "

read target\_dir

# Create target directory if it doesn't exist

if [ ! -d "$target\_dir" ]; then

mkdir "$target\_dir"

fi

# Copy all files with the given extension to the target directory

for file in \*."$extension"; do

if [ -f "$file" ] && [ ! -d "$file" ]; then

cp "$file" "$target\_dir"

fi

done

This script prompts the user to enter a file extension (without the dot) and a target directory name. It then checks if the target directory exists and creates it if it doesn't. It then loops through all files in the current directory with the given file extension and copies them to the target directory, as long as they are not directories. The check for whether a file is a regular file and not a directory is important to avoid copying any subdirectories that may have the same file extension.

Here are the solutions to the two problems:

1. Shell script to modify occurrences of "ex:" to "Example:" in files

bash

#!/bin/bash

for file in \*

do

if [ -f "$file" ]; then

sed -i -E 's/^[Ee]x:|\. [Ee]x:/Example:/g' "$file"

fi

done

Explanation:

* The script loops over each file in the current directory using a for loop
* The if statement checks if the file is a regular file using the -f flag
* The sed command is used to modify the text in the file. The -i flag specifies that the modification should be done in-place, and the -E flag is used for extended regular expressions
* The regular expression ^[Ee]x:|\. [Ee]x: matches "ex:" at the start of a line or after a period. The g at the end specifies that all occurrences should be replaced

1. Shell script to delete even numbered lines in a text file

bash

#!/bin/bash

echo "Enter file name: "

read filename

sed -i.bak -n '1p;n;p' "$filename"

echo "Even numbered lines deleted."

Explanation:

* The script prompts the user to enter a file name
* The sed command is used to delete even numbered lines in the file. The -i.bak flag creates a backup of the original file, and the -n flag disables automatic printing of lines
* The command 1p;n;p prints the first line, skips the second line, and prints the third line. This prints every odd numbered line and deletes every even numbered line.
* The script displays a message to indicate that even numbered lines have been deleted.

Shell script to check whether a number is prime or not:

bash

#!/bin/bash

echo "Enter a number:"

read n

if [[ $n -lt 2 ]]; then

echo "$n is not a prime number"

exit 0

fi

isPrime=true

for (( i=2; i<$n; i++ )); do

if [[ $(( $n % $i )) -eq 0 ]]; then

isPrime=false

break

fi

done

if [[ $isPrime = true ]]; then

echo "$n is a prime number"

else

echo "$n is not a prime number"

fi

1. Shell script to find the factorial of a number:

bash

#!/bin/bash

echo "Enter a number:"

read n

if [[ $n -lt 0 ]]; then

echo "Factorial of negative numbers is undefined"

exit 0

fi

fact=1

for (( i=1; i<=$n; i++ )); do

fact=$(( $fact \* $i ))

done

echo "Factorial of $n is $fact"

1. Shell script to write even numbered lines to a file with name evenfile and odd numbered lines to a file called oddfile:

bash

#!/bin/bash

echo "Enter file name:"

read filename

if [[ ! -f $filename ]]; then

echo "File not found"

exit 0

fi

evenfile=evenfile

oddfile=oddfile

rm -f $evenfile $oddfile

i=1

while read line; do

if [[ $(( $i % 2 )) -eq 0 ]]; then

echo $line >> $evenfile

else

echo $line >> $oddfile

fi

i=$(( $i + 1 ))

done < $filename

echo "Even numbered lines written to file: $evenfile"

echo "Odd numbered lines written to file: $oddfile"

Lab 4  
Shell script to make a duplicate copy of a specified file through command line:

bash

#!/bin/bash

if [ $# -ne 1 ]; then

echo "Usage: $0 <filename>"

exit 1

fi

if [ ! -f "$1" ]; then

echo "Error: File not found"

exit 1

fi

echo "Copying file $1 to $1\_copy"

cp "$1" "$1\_copy"

echo "File copied successfully"

1. Shell script to remove all files that are passed as command line arguments interactively:

bash

#!/bin/bash

if [ $# -eq 0 ]; then

echo "Usage: $0 <filename1> <filename2> ... <filenameN>"

exit 1

fi

for file in "$@"; do

if [ -f "$file" ]; then

read -p "Are you sure you want to delete $file? [y/N]: " choice

case $choice in

[Yy]\* ) rm "$file"; echo "$file deleted successfully";;

\* ) echo "File deletion aborted";;

esac

else

echo "Error: File not found: $file"

fi

done

1. Program to sort the strings that are passed as command line arguments:

bash

#!/bin/bash

if [ $# -eq 0 ]; then

echo "Usage: $0 <string1> <string2> ... <stringN>"

exit 1

fi

sort\_strings() {

IFS=$'\n' sorted=($(sort <<<"${strings[\*]}"))

echo "Sorted strings:"

for string in "${sorted[@]}"; do

echo "$string"

done

}

declare -a strings=("$@")

sort\_strings

1. Wordcount script that takes -linecount, -wordcount, -charcount options and performs accordingly, on the input file that is passed as command line argument (using case statement):

bash

#!/bin/bash

if [ $# -ne 2 ]; then

echo "Usage: $0 [-linecount|-wordcount|-charcount] <filename>"

exit 1

fi

if [ ! -f "$2" ]; then

echo "Error: File not found"

exit 1

fi

case "$1" in

-linecount )

echo "Line count: $(wc -l < "$2")"

;;

-wordcount )

echo "Word count: $(wc -w < "$2")"

;;

-charcount )

echo "Character count: $(wc -c < "$2")"

;;

\* )

echo "Invalid option: $1"

echo "Usage: $0 [-linecount|-wordcount|-charcount] <filename>"

exit 1

;;

esac

Note: These scripts may need to be modified to work in different operating systems or shell environments.

5)  
Here's an example shell script that implements the menu driven program to search and delete patterns in a file:

#!/bin/bash

if [ $# -lt 2 ]; then

echo "Usage: $0 <input\_file> <pattern1> [pattern2] [pattern3] ..."

exit 1

fi

input\_file=$1

shift

PS3="Please choose an option: "

options=("Search" "Delete" "Exit")

select opt in "${options[@]}"

do

case $opt in

"Search")

echo "Enter search pattern:"

read search\_pattern

echo "Lines containing '$search\_pattern':"

grep "$search\_pattern" "$input\_file"

;;

"Delete")

echo "Enter pattern to delete:"

read delete\_pattern

sed -i "/$delete\_pattern/d" "$input\_file"

echo "Pattern '$delete\_pattern' deleted."

;;

"Exit")

echo "Exiting..."

exit 0

;;

\*)

echo "Invalid option: $REPLY"

;;

esac

done

Here's how to use the program:

$ ./menu.sh input.txt foo bar baz

Please choose an option:

1) Search

2) Delete

3) Exit

#? 1

Enter search pattern:

foo

Lines containing 'foo':

This line contains foo.

$ ./menu.sh input.txt foo bar baz

Please choose an option:

1) Search

2) Delete

3) Exit

#? 2

Enter pattern to delete:

bar

Pattern 'bar' deleted.

$ cat input.txt

This line contains foo.

This line contains baz.

$ ./menu.sh input.txt foo bar baz

Please choose an option:

1) Search

2) Delete

3) Exit

#? 3

Exiting...

Additional Programs   
 1. Shell script to input a file and display permissions of the owner group and others:

bash

#!/bin/bash

if [ $# -eq 0 ]; then

echo "Please provide a file name as argument"

else

file=$1

if [ ! -f $file ]; then

echo "File does not exist"

else

echo "File: $file"

echo "Owner: $(stat -c '%U' $file)"

echo "Group: $(stat -c '%G' $file)"

echo "Permissions: $(stat -c '%A' $file)"

fi

fi

1. Shell script to display all files that are created between the input years range:

bash

#!/bin/bash

if [ $# -ne 2 ]; then

echo "Usage: $0 <start year>-<end year>"

exit 1

fi

start=$(date -d "$1" +"%s")

end=$(date -d "$2" +"%s")

if [ $start -gt $end ]; then

echo "Start year must be less than or equal to end year"

exit 1

fi

echo "Files created between $1 and $2:"

find . -type f -newermt $1 ! -newermt $2 -ls

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

bash

#!/bin/bash

if [ $# -ne 3 ]; then

echo "Usage: $0 <file> <start line> <end line>"

exit 1

fi

file=$1

start=$2

end=$3

if [ ! -f $file ]; then

echo "File does not exist"

exit 1

fi

if [ $start -gt $end ]; then

echo "Start line number must be less than or equal to end line number"

exit 1

fi

awk "NR>=$start && NR<=$end" $file