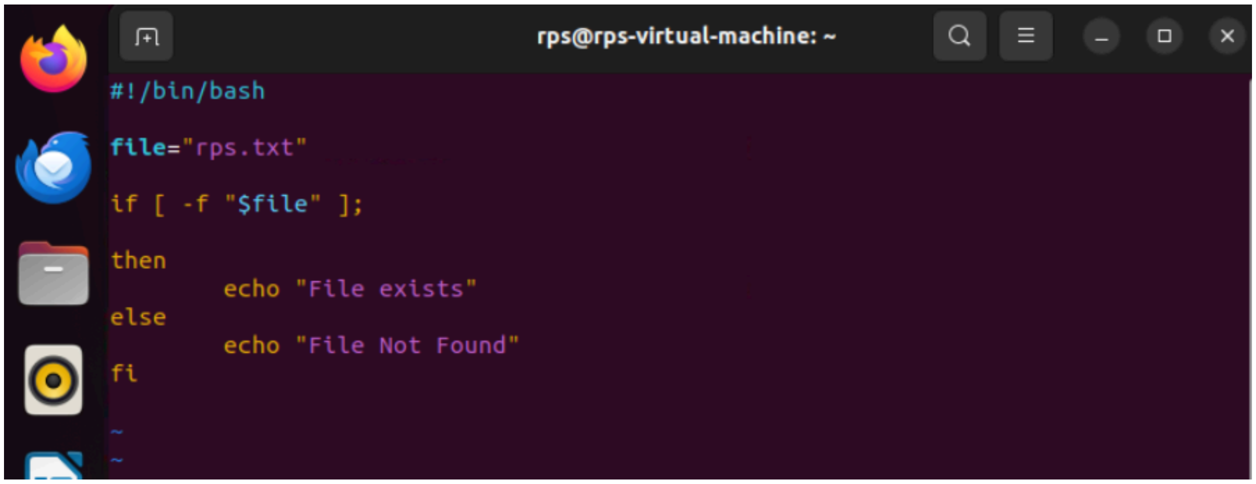
ASSIGNMENT - LINUX OPERATING SYSTEM

1. Ensure the script checks if a specific file(e.g, myfile.txt) exists in the current directory if it exists print “File exists”, otherwise print “File Not Found”.

Bash Script that checks if a specific file exists in the current directory and print a message accordingly:

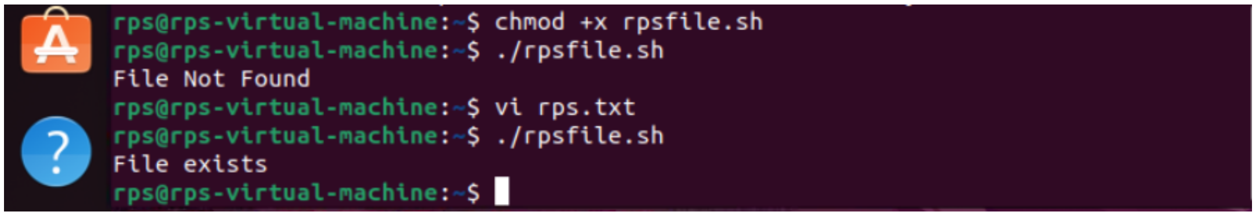
Bash Script



```
#!/bin/bash
file="rps.txt"
if [ -f "$file" ];
then
        echo "File exists"
else
        echo "File Not Found"
fi
~
~
```

```
#!/bin/bash

file="rps.txt"

if [ -f  "$file" ];  then

        echo "File exists"

else

        echo "File not found"

fi
```

Output



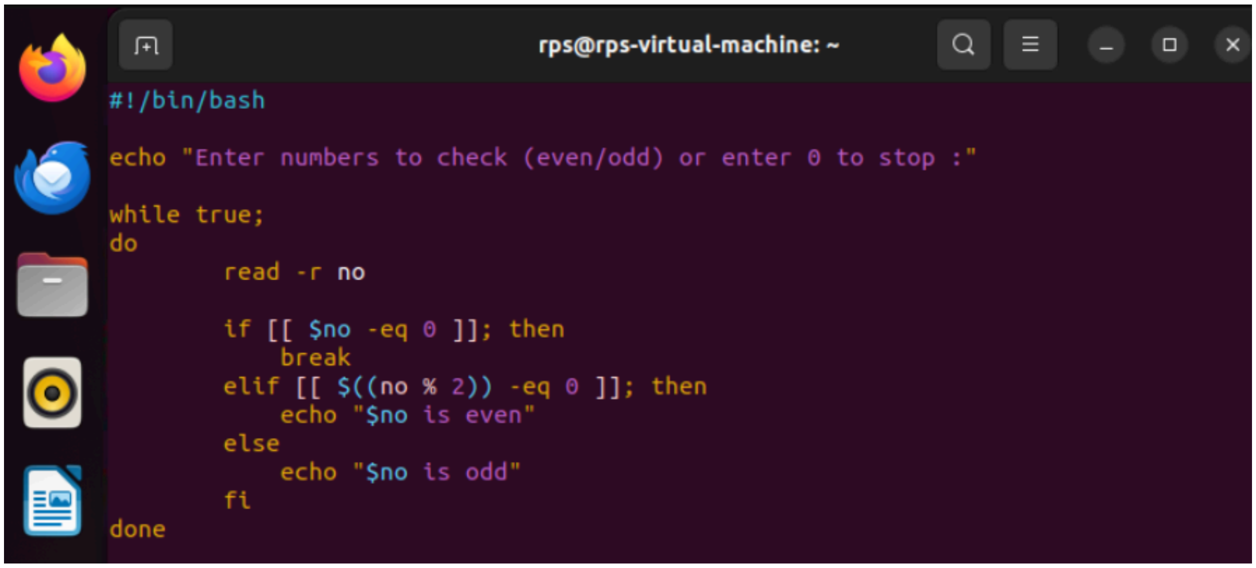
```
rps@rps-virtual-machine:~$ chmod +x rpsfile.sh
rps@rps-virtual-machine:~$ ./rpsfile.sh
File Not Found
rps@rps-virtual-machine:~$ vi rps.txt
rps@rps-virtual-machine:~$ ./rpsfile.sh
File exists
rps@rps-virtual-machine:~$
```

ASSIGNMENT - LINUX OPERATING SYSTEM

2. Write a script that reads numbers from the user until they enter ‘0’. The script should also print whether each number is odd or even.

Bash Script that checks even or odd number given by user until user enter ‘0’ and print a message accordingly:

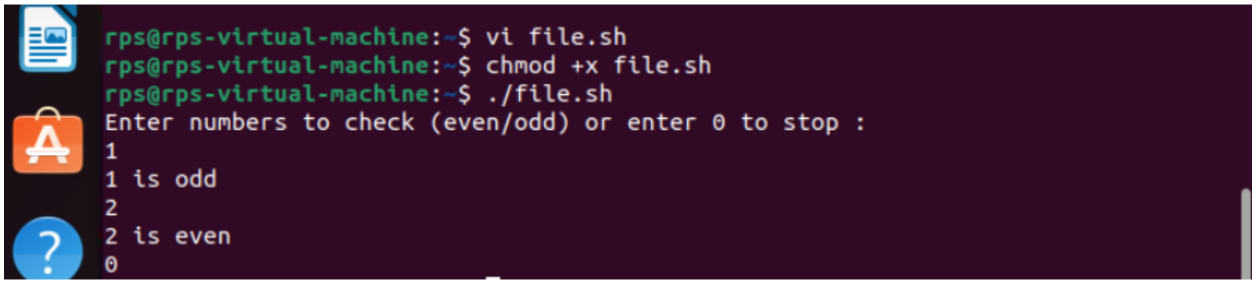
Bash Script



```
#!/bin/bash

echo "Enter numbers to check (even/odd) or enter 0 to stop :"

while true;
do
        read -r no

        if [[ $no -eq 0 ]]; then
            break
        elif [[ $((no % 2)) -eq 0 ]]; then
            echo "$no is even"
        else
            echo "$no is odd"
        fi
done
```

```
#!/bin/bash

 echo "Enter numbers to check (even/odd) or enter 0 to stop :"

while true; do

        read -r no

        if  [ $no -eq 0 ]; then

            break

        elif [ $((no % 2)) -eq 0 ]; then

        echo "$no is even"

        else

        echo "$no is odd"

        fi

done
```

Output



```
rps@rps-virtual-machine:~$ vi file.sh
rps@rps-virtual-machine:~$ chmod +x file.sh
rps@rps-virtual-machine:~$ ./file.sh
Enter numbers to check (even/odd) or enter 0 to stop :
1
1 is odd
2
2 is even
0
```

3. Create a function that takes a filename as an argument and print the number of lines in the file. Call this function from your script with different filenames.

Bash Script that takes filename as an argument and run the whole function when it is call and print a message accordingly:

Bash Script

```
#!/bin/bash

 myFunction() {

 filename="$1"

 if [ -f "$filename" ]; then

 lines=$(wc -l < "$filename")

echo "The file '$filename' has $lines lines."

else echo "Error: File '$filename' not found."

fi

 }

# Call the function with filenames

myFunction "L1"

myFunction "L2"
```

Explanation :

- Defines a function called ‘myFunction’ which takes a filename as an argument.
- Checks if the file specified by the filename exists (-f flag).
- If the file exists, it counts the number of lines in the file using wc -l.
- Prints the number of lines in the file.
- If the file is not found, it prints an error message.
- Calls the ‘myFunction’ function with different filenames specified in the array filenames.

4. Write a script that creates a directory named TestDir and inside it, creates ten files named File1.txt, File2.txt, ... File10.txt. Each file should contain its filename as its content (e.g., File1.txt contains "File1.txt").

Bash Script that creates one directory and inside it, creates 10 files and print a message accordingly:

Bash Script

```
#!/bin/bash

mkdir -p TestDir

for ((i=1; i<=10; i++)); do

    filename="Push$i.txt"

    echo "$filename contains" > "TestDir/$filename"


done
    echo "Files created successfully in TestDir."
```

Explanation :

- Creates a directory named TestDir using mkdir -p, which ensures that the directory is created only if it doesn't already exist.
- Uses a loop to create ten files (Push1.txt to Push2.txt).
- Inside the loop, it sets the variable filename to the current filename (Push$i.txt), where $i is the loop variable.
- Writes the filename to the corresponding file using echo.
- After the loop completes, it prints "Files created successfully in TestDir."

5. Modify the script to handle errors, such as the directory already existing or lacking

permissions to create files. Add a debugging mode that prints additional information when enabled.

Bash script that creates a directory and handles errors, as well as includes a debugging mode and print a message accordingly:

Bash Script

```
#!/bin/bash

myFunction() {

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

            echo "File not found: $1"

            return 1

 fi

 wc -l < "$1"

             return 0

}

# Enable debugging mode if the DEBUG environment variable is set

 if [ -n "$MYDEBUG" ]; then

 set -x

 fi

  if [ -d "$1" ]; then

    echo "Error: Directory already exists!"

    exit 1

 fi

  if [ ! -w "." ]; then

    echo "Error: Lacking permissions to create files."

    exit 1

 fi

  mkdir  "$1"

  if [ $? -eq 0 ]; then
```

```
       echo "Directory created: $1"
  else
        echo "Error: Failed to create output directory!"
  exit 1
fi
}
main()
# Call the functions
 if myFunction "input/$filename"; then
mv "input/$filename" "output/$filename"
 fi
done
# Disable debugging mode if it was enabled
 if [ -n "$MYDEBUG" ]; then
set +x
 fi
```

6. Given a sample log file, write a script using grep to extract all lines containing "ERROR". Use awk to print the date, time, and error message of each extracted line. Data Processing with sed

To extract all lines containing "ERROR" from a log file using 'grep', you can use the following command:
Command For This :

```
grep "ERROR" logfile.txt
```

This will print all lines in 'logfile.txt' that contain the string "ERROR".

To further process the output 'using awk' to print the date, time, and error message of each extracted line, you can use the following command:

Bash Script

```
grep "ERROR" logfile.txt | awk '{print $1" "$2" "$3" "$4" "$5" "$6" "$7" "$8" "$9" "$10}' |
awk '{print $1" "$2" "$3" "substr($0, index($0,$4))}'
```

Explanation :

This command first extracts all lines containing "ERROR" using ‘grep’. It then pipes the output to ‘awk’, which prints the first 32 fields of each line. Finally, it pipes the output to another ‘awk’ command, which prints the date, time, and error message of each line by extracting the substring starting from the fourth field.

Note that the number of fields in the log file may vary, so you may need to adjust the number of fields printed by the first ‘awk’ command accordingly.

7. Create a script that takes a text file and replaces all occurrences of "old_text" with "new_text". Use sed to perform this operation and output the result to a new file.

Bash Script

Bash script that takes a text file and replace all occurrences and output the result to a new file and print a message accordingly:

```
#!/bin/bash

input_file="input.txt"

old_text="old_text"

new_text="new_text"

output_file="output.txt"

sed "s/${old_text}/${new_text}/g" ${input_file} > ${output_file}
```

Explanation :

- It sets the input file, old text, and new text as variables.
- It sets the output file as a variable.
- It uses ‘sed’ to replace all occurrences of the old text with the new text in the input file. The ‘s’ command is used for substitution, and the ‘g’ flag at the end makes the substitution global .
- The output of the ‘sed’ command is redirected to the output file using the ‘>’ symbol.
- To use this script, simply replace the ‘input_file’, ‘old_text’, and ‘new_text’ variables with your own values, and run the script. The output will be written to the ‘output_file’.