Shell scripting with bash

Assignment-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".

Here is a Bash script that checks if a specific file exists in the current directory and prints a message accordingly:

Base script:

```
#!/bin/bash

file="demo.txt"

if [ -f "$file" ];
then
        echo "exist"
else
        echo "not exist"
fi
~
~
~
~
~
~
~
~
~
~
```

output:

```
rps@rps-virtual-machine:~/myscript$ ./check_file.sh
bash: ./check_file.sh: Permission denied
rps@rps-virtual-machine:~/myscript$ chmod +x check_file.sh
rps@rps-virtual-machine:~/myscript$ ./check_file.sh
not exist
rps@rps-virtual-machine:~/myscript$
```

Explanation:

1. We define the name of the file we want to check for existence in the **file** variable.
2. We use an if statement with the **[-f "$file"]** condition to check if the file exists. The -f flag checks if the file is a regular file (not a directory or special file).
3. If the condition is true (the file exists), we print a message saying the **exist** in the current directory.
4. If the condition is false (the file does not exist), we print a message saying the **not exist** in the current directory.

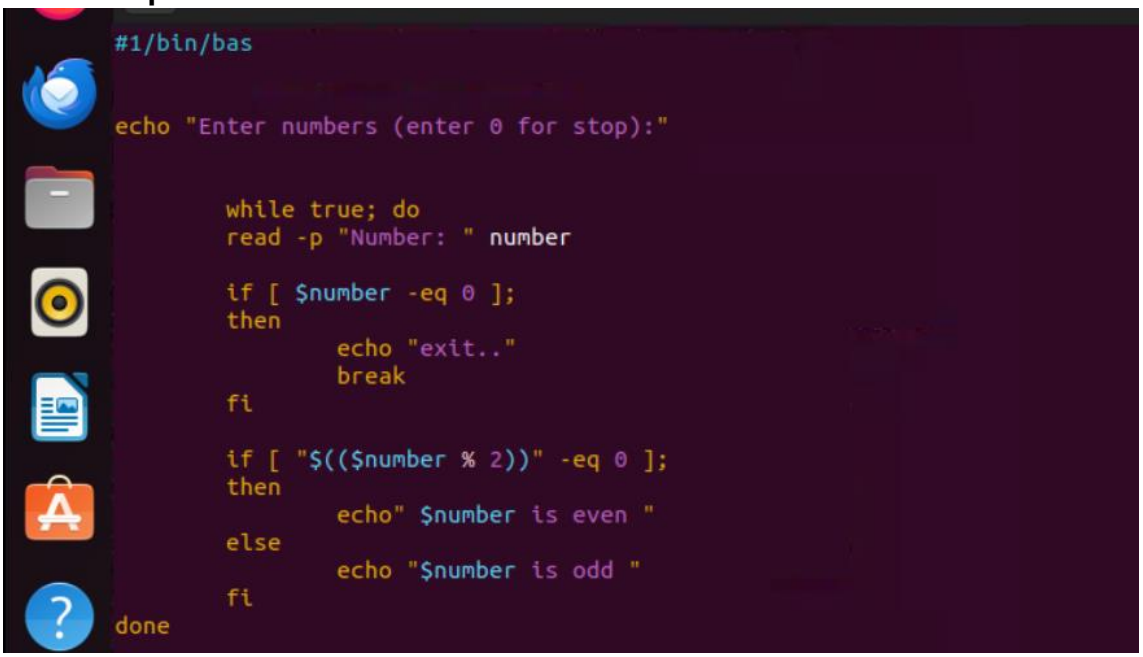
   To use this script:
   1. Save the script to a file, for example,**vi check_file.sh**.
   2. Make the script executable with **chmod +x check_file.sh**.
   3. Run the script with **./check_file.sh**.

Shell scripting with bash

Assignment-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.

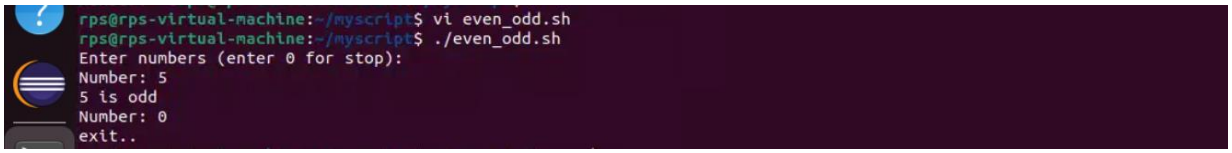
Here's a **Bash script** that reads **numbers** from the user until they enter '0' and prints whether each number is **odd** or **even**:

Bash script:



```
#1/bin/bas

echo "Enter numbers (enter 0 for stop):"

        while true; do
        read -p "Number: " number

        if [ $number -eq 0 ];
        then
                echo "exit.."
                break
        fi

        if [ "$(($number % 2))" -eq 0 ];
        then
                echo" $number is even "
        else
                echo "$number is odd "
        fi
done
```

Output:



```
rps@rps-virtual-machine:~/myscript$ vi even_odd.sh
rps@rps-virtual-machine:~/myscript$ ./even_odd.sh
Enter numbers (enter 0 for stop):
Number: 5
5 is odd
Number: 0
exit..
```

Here's how the script works:

1. The while loop runs **indefinitely** until the user enters **'0'**.
2. The read command prompts the user to enter a **number**.
3. If the entered number is **'0'**, the script prints a message **"Exit.."** and exits the loop using **break**.
4. The script checks if the number is **even** or **odd** by using the **modulo operator %** to check if the remainder of dividing the number by 2 is 0. If it is, the number is **even**; otherwise, it's **odd**.

To run the script,

1. Save the script to a file, for example, **vi even_odd.sh**.
2. Make the script executable with **chmod +x even_odd.sh**.
3. Run the script with **./even_odd.sh.**

Shell scripting with bash

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

Here's a Bash script that defines a function to count the number of lines in a file and calls the function with different filenames:

```
#1/bin/bash

count_lines() {

 filename=$1

 if [ -f "$filename" ]; then
         lines=$(wc -l < "$filename")
         echo "number of lines in $filename: $lines"
 else
         echo "File $filename not found."
 fi
}

count_lines "file1.txt"
count_lines "file2.txt"

~
```

Output:

```
rps@rps-virtual-machine:~/myscript$ chmod +x count_lines.sh
rps@rps-virtual-machine:~/myscript$ ./count_lines.sh
File file1.txt not found.
rps@rps-virtual-machine:~/myscript$ vi count_lines.sh
rps@rps-virtual-machine:~/myscript$ vi file1.txt
rps@rps-virtual-machine:~/myscript$ vi file2.txt
rps@rps-virtual-machine:~/myscript$ ./count_lines.sh
number of lines in file1.txt: 1
rps@rps-virtual-machine:~/myscript$ vi count_lines.sh
rps@rps-virtual-machine:~/myscript$ ./count_lines.sh
number of lines in file1.txt: 1
number of lines in file2.txt: 1
```

Here's how the script works:

1. The **count_lines** function is defined. It takes a filename as an argument ($1).
2. Inside the function, the **wc** command is used to count the number of lines in the file.
3. The **-l** option counts the number of lines, and the **<** redirects the file content to **wc**. The result is stored in the **lines** variable.
4. The function prints a **message** indicating the filename and the number of lines in the file.
5. The script calls the **count_lines** function three times, passing different filenames as arguments.

To run the script,

1. save it to a file (e.g., **vi count_lines.sh**)
2. make it executable with **chmod +x count_lines.sh**.
3. Then, run the script with **./count_lines.sh**.

Shell scripting with bash

Before running the script, make sure the specified files (**file1.txt, file2.txt**) exist in the same directory as the script. If the files don't exist or have different names, update the filenames accordingly.

Assignment 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"").

Here's a script that creates a directory named "TestDir" and inside it, creates ten files named "File1.txt" to "File10.txt" with their respective filenames as content:

Bash script:

```
#!/bin/bash

mkdir TestDir

cd TestDir

for i in {1..10}; do
        echo "File$i.txt" > File$i.txt
done
echo "Files are created"

~
```

Output:

```
rps@rps-virtual-machine:~/myscript$ vi create_files.sh
rps@rps-virtual-machine:~/myscript$ chmod +x create_files.sh
rps@rps-virtual-machine:~/myscript$ ./create_files.sh
Files are created
```

Here's how the script works:

- The script creates a directory named **"TestDir"** using the **mkdir** command.
- It then changes to the **"TestDir"** directory using the **cd** command.
- Inside the **"TestDir"** directory, it uses a for loop to iterate from 1 to 10.
- using the **echo** command and the redirection operator **>**, it creates a file named **"File$i.txt" (e.g., File1.txt, File2.txt, ..., File10.txt).**
- Finally, it prints a message **Files are created**.

To run the script:

- Save the script to a file **(e.g., vi create_files.sh).**
- Make the script executable using **chmod +x create_files.sh**.
- Run the script using **./create_files.sh**.

Shell scripting with bash

Assignment 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.

Here's the modified script that handles errors such as the directory already existing or lacking permissions to create files, and includes debugging mode:

Bash:

```
#!/bin/bash
demoFun(){
   if [ ! -f  "$1"  ]; then
      echo "File not found: $1"
      return 1
   fi
   wc -l < "$1"
   return 0
}
# Enable debugging mode if the debug environment variable is set
if [ -n "$DEBUG" ]; then
   set -x
fi
if [ -d "$1" ]; then
   echo "Error: Directory '$1' 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

# Call the functions

main(){

   demoFun "input/$filename"

}


# Disable debugging mode if it was enabled

if [ -n "$DEBUG" ]; then

   set +x

fi
```

output:

The script runs like this:

- The **demoFun** function is defined in the script. Inside the function, it checks if the specified file exists using the **-f** option with **[! -f "$1"].** If the file doesn't exist, it prints an error message and returns with a non-zero status code (1).

- If the file exists, it uses **wc -l < "$1"** to count the number of lines in the file and outputs the result. The function returns with a zero status code (0) to indicate successful execution.

- The script checks if the **DEBUG** environment variable is set using **[-n "$DEBUG"].** If it is set, it enables debugging mode with **set -x** and checks if a directory name was provided as an argument **($1)**. If the directory already exists, it prints an error message and exits with a non-zero status code (1).

- It checks if the current directory is writable using **[! -w "."].** If not, it prints an error message and exits with a non-zero status code **(1)**. Else, it creates the specified directory using **mkdir "$1**".

- It checks the exit status of the **mkdir** command using **$?.** If the directory was created successfully, it prints a success message. Otherwise, it prints an error message and exits with a non-zero status code (1).

- The **main** function is defined, which calls **demoFun** with a specific file path **("input/$filename")**. If debugging mode was enabled earlier, it disables it with **set +x**.

To run the script:

- Save the script to a file (e.g., **vi debugging.sh**).
- Make the script executable with **chmod +x debugging.sh**.
- enable debugging mode with **export DEBUG=1**.
- Run the script with the desired directory name **./debugging.sh sample.**

Assignment 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.

Assignment 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.

Here is a bash script that takes a text file, replaces all occurrences of a specified old text with a new text, and outputs the result to a new file, along with a message indicating the operation was successful:

Bash Script:

```
#!/bin/bash

# Define the input file
input_file="input.txt"

# Define the old text
old_text="old_text"

# Define the new text
new_text="new_text"

# Define the output file
output_file="output.txt"

# Perform the replacement
sed "s/${old_text}/${new_text}/g" ${input_file} > ${output_file}

# Print a success message
echo "Text replacement successful. Output saved to ${output_file}."
```

Shell scripting with bash

Running the Script

1. Save the script in a file with a **.sh** extension, for example, replace_text.sh.
2. Run the command **chmod +x replace_text.sh** to make the script executable.
3. Run the Script: Run the script by typing **./replace_text.sh**.

How the Script Works

1. Setting Variables: The script sets the input file, old text, new text, and output file as variables.
2. Replacing Text: The script 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 makes the substitution global.
3. Redirecting Output: The output of the sed command is redirected to the output file using the > symbol.