Day-06

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".

→Solution :

Code for this :

filename="**NotePaid.txt**"

```
if [ -e "$filename" ]; then
   echo "File exists"
else
   echo "File not found"
fi
```

Explanation :

- filename="**NotePaid.txt**" :This sets the variable filename to the name of the file you want to check.
- if [-e "$filename"]; then: This line checks if the file exists (-e flag).
- echo "File exists": If the file exists, this line prints "File exists".
- else: If the file does not exist,
- echo "File not found": This line prints "File not found".
- fi: This marks the end of the if statement.

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.

→Solution :

Code For This :

```
while true; do
  read -p "Enter a number (0 to quit): " num
  if [ "$num" -eq 0 ]; then
    break
  elif [ "$num" -eq "$((num % 2))" ]; then
    echo "The number $num is odd."
  else
    echo "The number $num is even."
  fi
done
```

Explanation :

- The script enters an infinite loop (while true).
- Inside the loop, it prompts the user to enter a number.
- It reads the number entered by the user using the read command.
- If the number entered is '0', breaks out of the loop.
- Otherwise, it checks whether the number is even or odd using the modulo operator %.
- If the remainder after dividing the number by 2 is 0, it prints that the number is even.
- If the remainder is not 0, it prints that the number is odd.
- The loop continues until the user enters '0'.

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.

→Solution :

Code for This :

```
print_line_count() {
   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
print_line_count "NotePaid.text"
```

Explanation :

- Defines a function called ‘print_line_count’ 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 ‘print_line_count’ function with different filenames specified in the array filenames

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").

→Solution :

Code For This :

```
# Create directory TestDir if it doesn't exist
mkdir -p TestDir

# Loop to create files
for ((i=1; i<=10; i++)); do
   filename="NotePaid$i.txt"
   echo "$filename" > 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 (NotePaid1.txt to NotePaid10.txt).
- Inside the loop, it sets the variable filename to the current filename (NotePaid$i.txt), where $i is the loop variable.
- Writes the filename to the corresponding file using echo.
- The > TestDir/$filename part redirects the output of echo to the file inside the TestDir directory.
- After the loop completes, it prints "Files created successfully in TestDir."
- When you run this script, it will create the TestDir directory if it doesn't exist and create ten files inside it, each containing its filename as its content.

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.

→Solution :

Code For This :

```
# Function to print the number of lines in a file
count_lines() {
  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

# Check if the directory already exists or if there are permission issues
if [ -d "output" ]; then
  echo "Error: Directory already exists."
  exit 1
fi

if [ ! -w "." ]; then
  echo "Error: Lacking permissions to create files."
  exit 1
fi

# Create the output directory
mkdir output
if [ $? -ne 0 ]; then
  echo "Error: Failed to create output directory."
  exit 1
```

```
fi

# Call the count_lines function with different filenames
for filename in myfile.txt anotherfile.txt; do
  if count_lines "input/$filename"; then
    mv "input/$filename" "output/$filename"
  fi
done

# Disable debugging mode if it was enabled
if [ -n "$DEBUG" ]; then
  set +x
fi
```

Explanation :

This script defines a 'count_lines' function that takes a filename as an argument and prints the number of lines in the file. If the file does not exist, the function returns an error code of 1. The script then checks if the output directory already exists or if there are permission issues. If there are any errors, the script prints an error message and exits with a non-zero exit code.

The script then enables debugging mode if the 'DEBUG' environment variable is set. It then creates the output directory and calls the 'count_lines' function with different filenames. If the 'count_lines' function succeeds, the script moves the file from the input directory to the output directory.

Finally, the script disables debugging mode if it was enabled.

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.

→Solution :

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:

Command For This :

```
grep "ERROR" logfile.txt | awk '{print $1" "$2" "$3" "$4" "$5" "$6" "$7" "$8" "$9" "$10" "$11" "$12" "$13" "$14" "$15" "$16" "$17" "$18" "$19" "$20" "$21" "$22" "$23" "$24" "$25" "$26" "$27" "$28" "$29" "$30" "$31" "$32}' | awk '{print $1" "$2" "$3" "substr($0, index($0,$4))}'
```

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.

→Solution :

'Sed' command that can replace all occurrences of "old_text" with "new_text" in a text file and output the result to a new file:

Command For This :

```
# Set the input file, old text, and new text
input_file="input.txt"
old_text="old_text"
new_text="new_text"

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

# Use sed to replace the old text with the new text
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 (i.e., it replaces all occurrences, not just the first one).
- 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'.