Day 7

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".

```bash
#!/bin/bash

# Define the file name
filename="myfile.txt"

# Check if the file exists
if [ -f "$filename" ]; then
    echo "File exists"
else
    echo "File not found"
fi
```

Save this script to a file, for example, `check_file.sh`, and then make it executable and run it as follows:

```
chmod +x check_file.sh
./check_file.sh
```

This script uses a conditional statement to check if `myfile.txt` exists in the current directory. If it does, it prints "File exists"; otherwise, it prints "File not found".

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.

```
While true: do

        echo “Enter the number [0 for exit] “
        read=number ;

        if [“$number” –eq 0] then
                echo “Exiting”
                break
        if  [ $((number %2)) –eq 0]; then
                echo “&number is even”
        else
                echo “$number is odd”
        fi
done
```

Explanation

1. **`#!/bin/bash`**: Specifies that the script should be run using the Bash shell.

2. `while true; do`: Starts an infinite loop.
3. `read -p "Enter a number (0 to exit): " number`: Prompts the user to enter a number and stores it in the variable `number`.
4. `if [ "$number" -eq 0 ]; then`: Checks if the entered number is '0'.

- `echo "Exiting..."`: Prints an exit message.
- `break`: Exits the loop if the entered number is '0'.

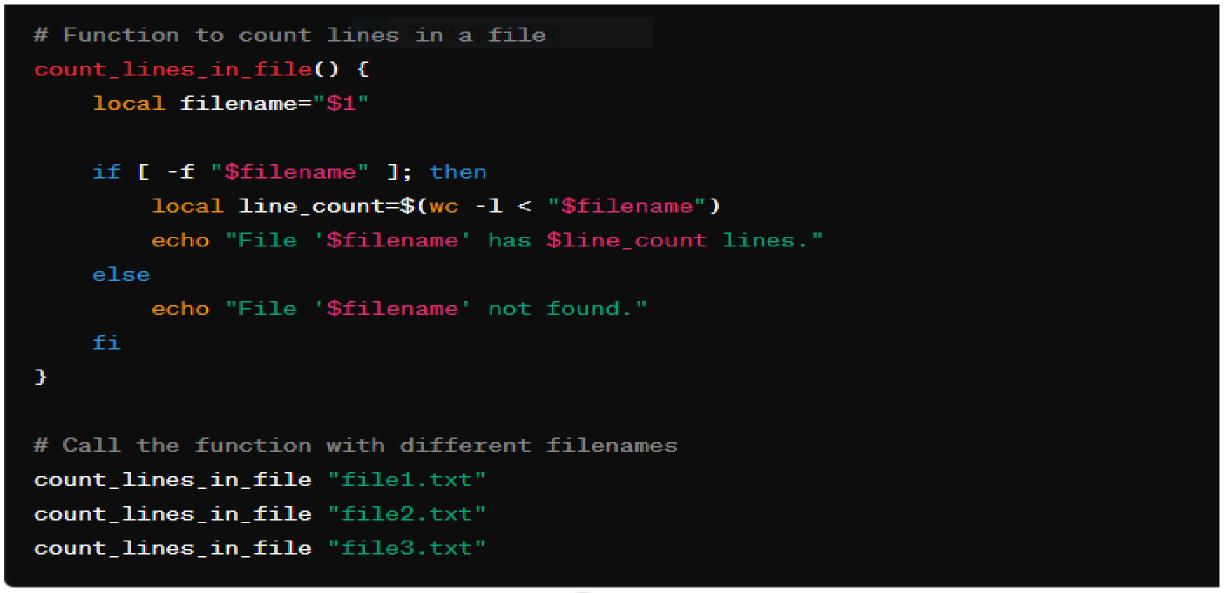
5. `if [ $((number % 2)) -eq 0 ]; then`: Checks if the number is even by calculating the remainder of the number divided by 2.

- `echo "$number is even"`: Prints that the number is even if the remainder is 0.
- `else`: Executes if the number is not even.
- `echo "$number is odd"`: Prints that the number is odd if the remainder is not 0.

6. `done`: Ends the loop.

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.



```
# Function to count lines in a file
count_lines_in_file() {
    local filename="$1"

    if [ -f "$filename" ]; then
        local line_count=$(wc -l < "$filename")
        echo "File '$filename' has $line_count lines."
    else
        echo "File '$filename' not found."
    fi
}

# Call the function with different filenames
count_lines_in_file "file1.txt"
count_lines_in_file "file2.txt"
count_lines_in_file "file3.txt"
```

Chmod +x count_lines.sh

./count_lines.sh

This script defines a function `count_lines_in_file` that takes a filename as an argument, checks if the file exists, counts the number of lines using `wc -l`, and prints the result.

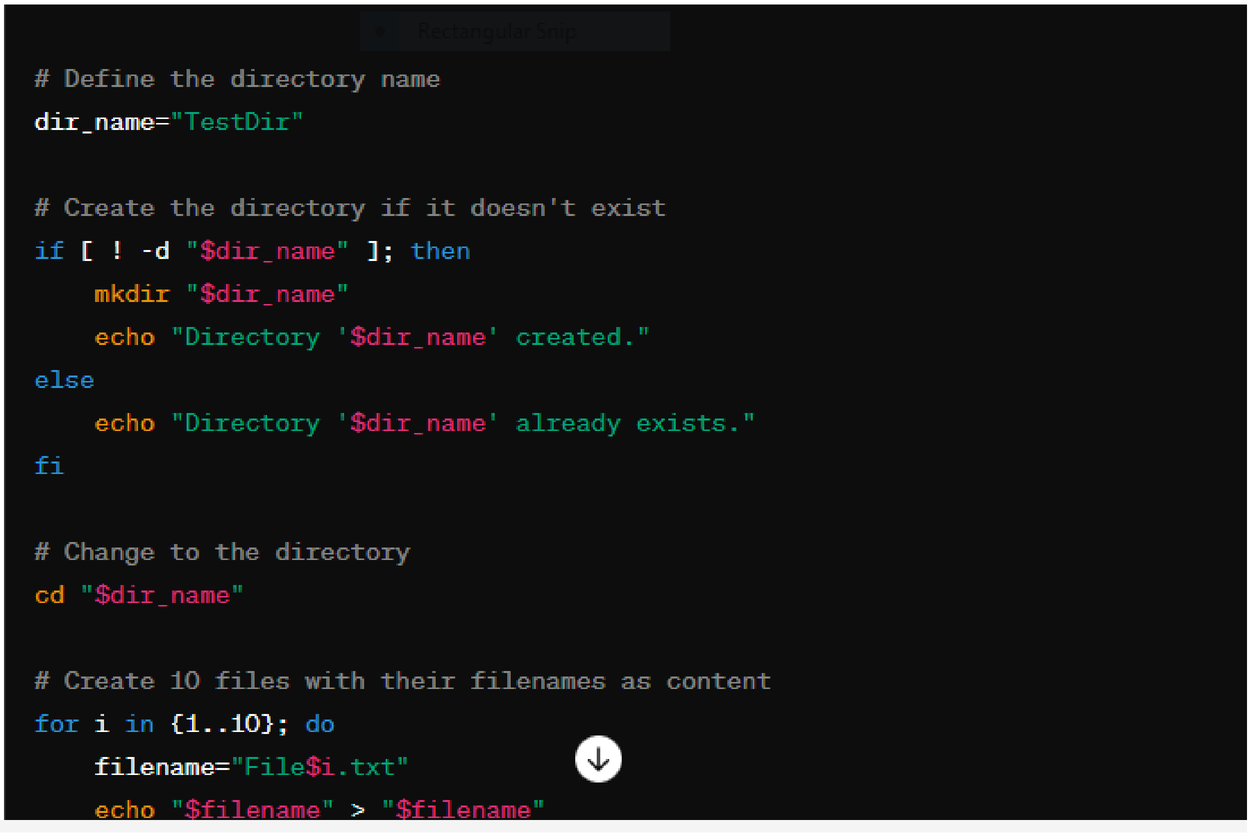
The script then calls this function with different filenames (`file1.txt`, `file2.txt`, `file3.txt`). Modify the filenames as needed to test with different files.

Assignment 4:

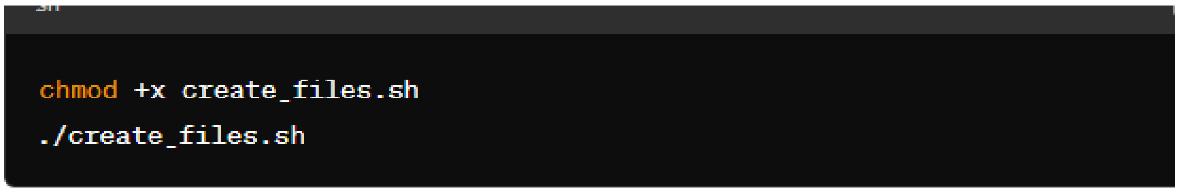
Write a script that creates a directory named TestDir and inside it, creates ten files named File1.txt, File2.txt, ... File10.txt. Each file should contain its filename as its content (e.g., File1.txt contains "File1.txt").

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.

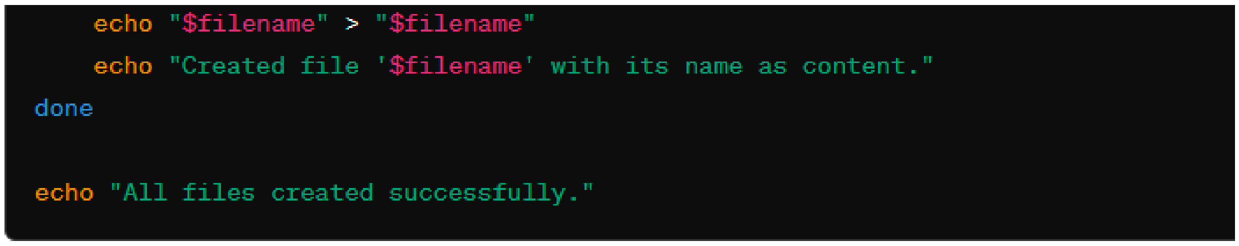
Data Processing with sed



```bash
# Define the directory name
dir_name="TestDir"

# Create the directory if it doesn't exist
if [ ! -d "$dir_name" ]; then
    mkdir "$dir_name"
    echo "Directory '$dir_name' created."
else
    echo "Directory '$dir_name' already exists."
fi

# Change to the directory
cd "$dir_name"

# Create 10 files with their filenames as content
for i in {1..10}; do
    filename="File$i.txt"
    echo "$filename" > "$filename"
```



```sh
chmod +x create_files.sh
./create_files.sh
```



```bash
    echo "$filename" > "$filename"
    echo "Created file '$filename' with its name as content."
done

echo "All files created successfully."
```

This script performs the following steps:

1. Defines the directory name as `TestDir`.
2. Checks if the directory exists; if not, it creates the directory.
3. Changes into the `TestDir` directory.

4. Uses a `for` loop to create ten files (`File1.txt` to `File10.txt`), writing each filename into its respective file.
5. Prints a message for each file created and a final message indicating successful creation of all files.

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.

```
#!/bin/bash


# Enable debugging mode if the DEBUG variable is set to 1

DEBUG=1


# Function to print debug messages

debug() {

  if [ "$DEBUG" -eq 1 ]; then

    echo "DEBUG: $1"

  fi

}


# Define the directory name
```

```
dir_name="TestDir"

# Create the directory if it doesn't exist
if [ -d "$dir_name" ]; then
   echo "Directory '$dir_name' already exists."
else
   if mkdir "$dir_name" 2>/dev/null; then
      echo "Directory '$dir_name' created."
   else
      echo "Error: Could not create directory '$dir_name'. Check your
permissions."
      exit 1
   fi
fi

# Change to the directory
if cd "$dir_name"; then
   debug "Changed to directory '$dir_name'."
else
   echo "Error: Could not change to directory '$dir_name'."
   exit 1
fi

# Create 10 files with their filenames as content
for i in {1..10}; do
```

```
    filename="File$i.txt"
    if echo "$filename" > "$filename" 2>/dev/null; then
        echo "Created file '$filename' with its name as content."
        debug "File content: $(cat "$filename")"
    else
        echo "Error: Could not create file '$filename'. Check your permissions."
    fi
done

echo "All files created successfully."
```

1. **Debugging Mode**: The `DEBUG` variable is set to `1` to enable debugging mode. The `debug` function prints debug messages when this mode is enabled.
2. **Error Handling**:

- Before creating the directory, the script checks if it already exists.
- If the directory cannot be created (e.g., due to permission issues), the script prints an error message and exits.
- When changing to the `TestDir` directory, the script checks if the `cd` command was successful. If not, it prints an error message and exits.
- When creating each file, the script checks if the file was created successfully. If not, it prints an error message.

You can adjust the `DEBUG` variable to `0` to disable debugging mode.

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

```
#!/bin/bash

# Define the log file name
logfile="sample.log"

# Check if the log file exists
if [ ! -f "$logfile" ]; then
   echo "Log file '$logfile' not found."
   exit 1
fi

# Extract lines containing "ERROR" using grep and process them with awk and sed
grep "ERROR" "$logfile" | awk '{print $1, $2, $3, $4, $5, $6}' | sed 's/ERROR//2'

chmod +x extract_errors.sh
./extract_errors.sh
```

1. **Check if the log file exists**: The script first checks if the specified log file exists. If not, it prints an error message and exits.
2. **Extract lines containing "ERROR" using `grep`**: The `grep "ERROR" "$logfile"` command extracts all lines that contain the word "ERROR" from the log file.
3. **Process the extracted lines with `awk`**: The `awk '{print $1, $2, $3, $4, $5, $6}'` command is used to print the date (fields 1 and 2), the time (field 3), and the error message (fields 4, 5, and 6).
4. **Remove the word "ERROR" using `sed`**: The `sed 's/ERROR//2'` command is used to remove the word "ERROR" from the output of `awk`.

Make sure to adjust the log file path (`sample.log`) as needed.

```
#!/bin/bash

# Check if the correct number of arguments is provided
if [ "$#" -ne 3 ]; then
   echo "Usage: $0 input_file old_text new_text"
   exit 1
fi

# Assign arguments to variables
input_file="$1"
old_text="$2"
new_text="$3"
output_file="output.txt"

# Check if the input file exists
if [ ! -f "$input_file" ]; then
```

```
    echo "Error: Input file '$input_file' not found."
    exit 1
fi

# Use sed to replace old_text with new_text and write the result to the output file
sed "s/$old_text/$new_text/g" "$input_file" > "$output_file"

# Confirm the operation
echo "Replaced all occurrences of '$old_text' with '$new_text' in '$input_file'."
echo "The result has been saved to '$output_file'."
```

1. Save the script to a file, for example, **`replace_text.sh`**.
2. Make the script executable:

   sh

   Copy code

   ```
   chmod
   ```

3. Run the script with the required arguments:

   sh

   Copy code

   ```
                                  "old_text" "new_text"
   ```

   This command will replace all occurrences of **`"old_text"`** with **`"new_text"`** in **`input.txt`** and save the result to **`output.txt`**.