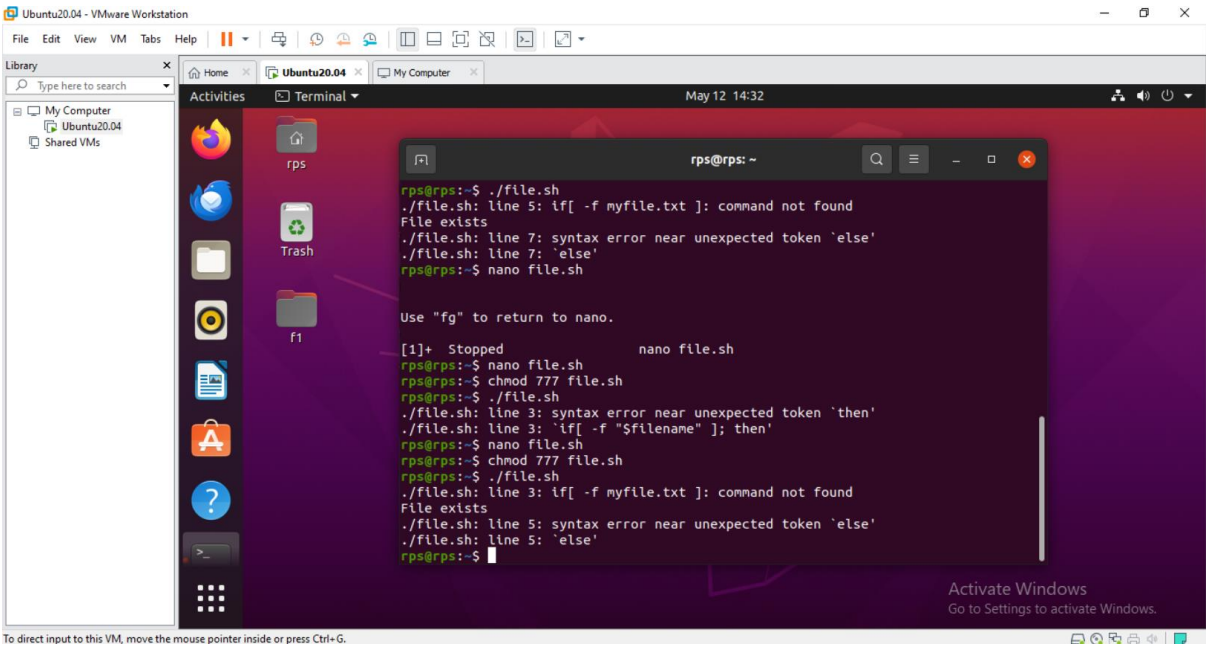
Shell Scripting With Bash Assignments:

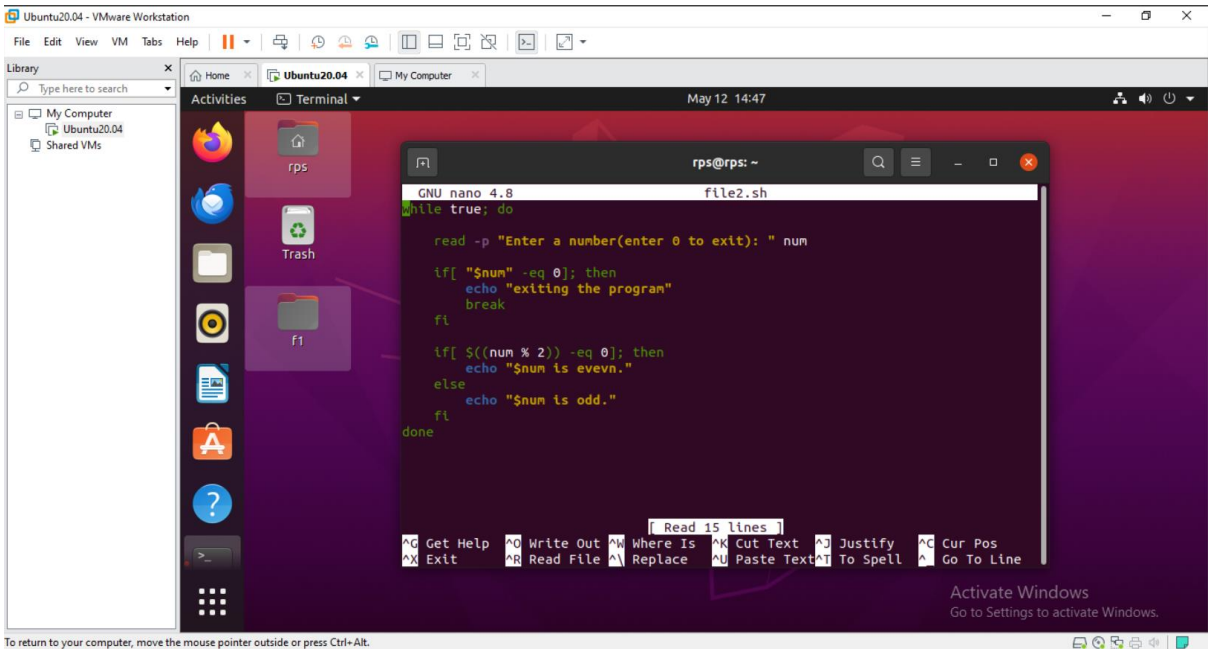
Assignment 1: Ensure the script checks if a specific file (e.g., myfile.txt) exists in the current directory. If it exists, print "File exists", otherwise print "File not found".

This is how it works:



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.

This is how it works:

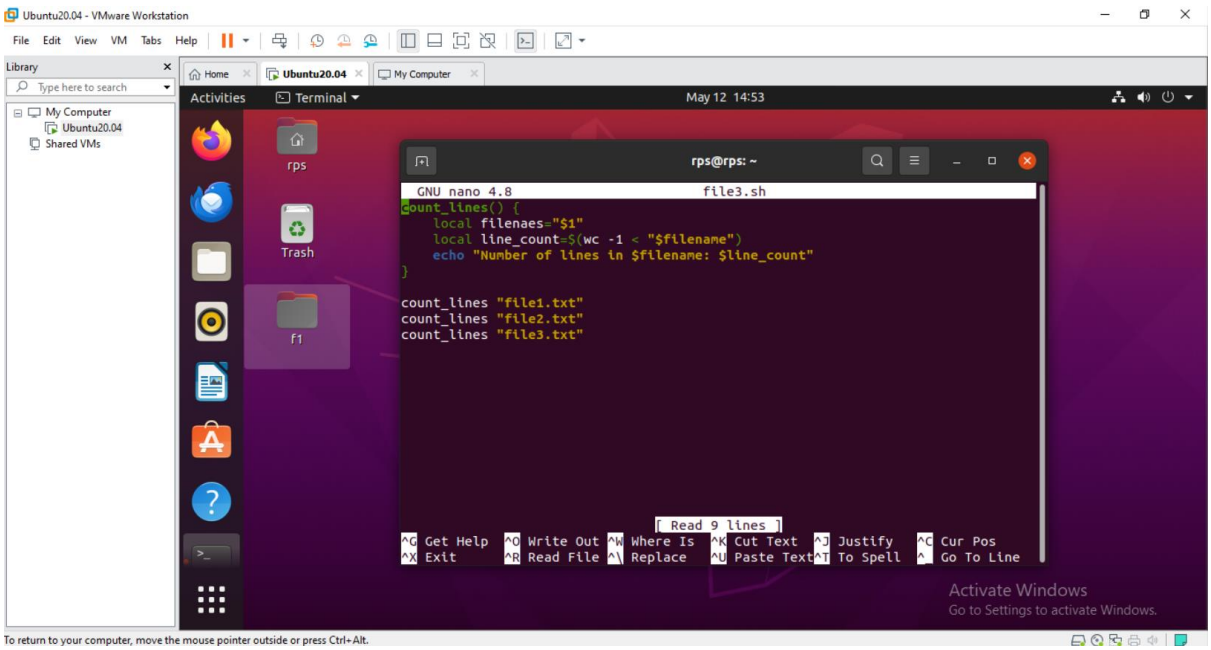


Here's how the script works:

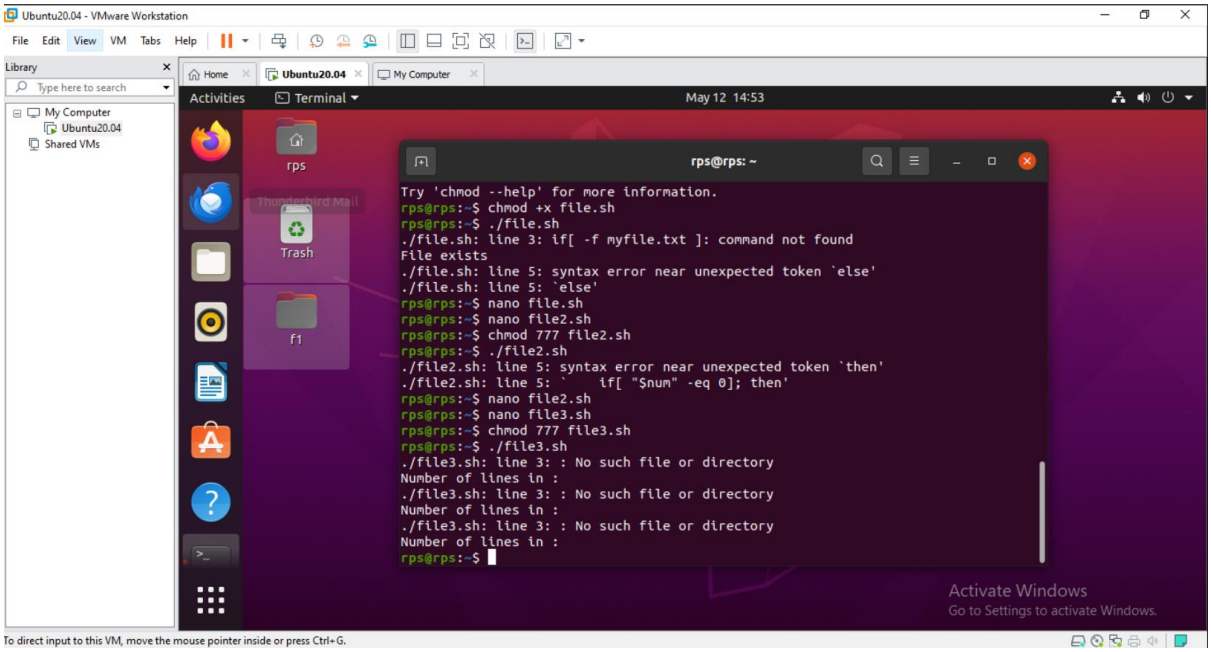
- It enters a while loop that runs indefinitely.
- Inside the loop, it prompts the user to enter a number using **read**.
- It checks if the entered number is 0 using an if statement.
- If the entered number is 0, it prints "Exiting the program." and exits the loop using **break**.
- If the entered number is not 0, it checks if the number is even or odd using the modulo operator **%**.
- It prints the result, indicating whether the number is even or 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.

Input:

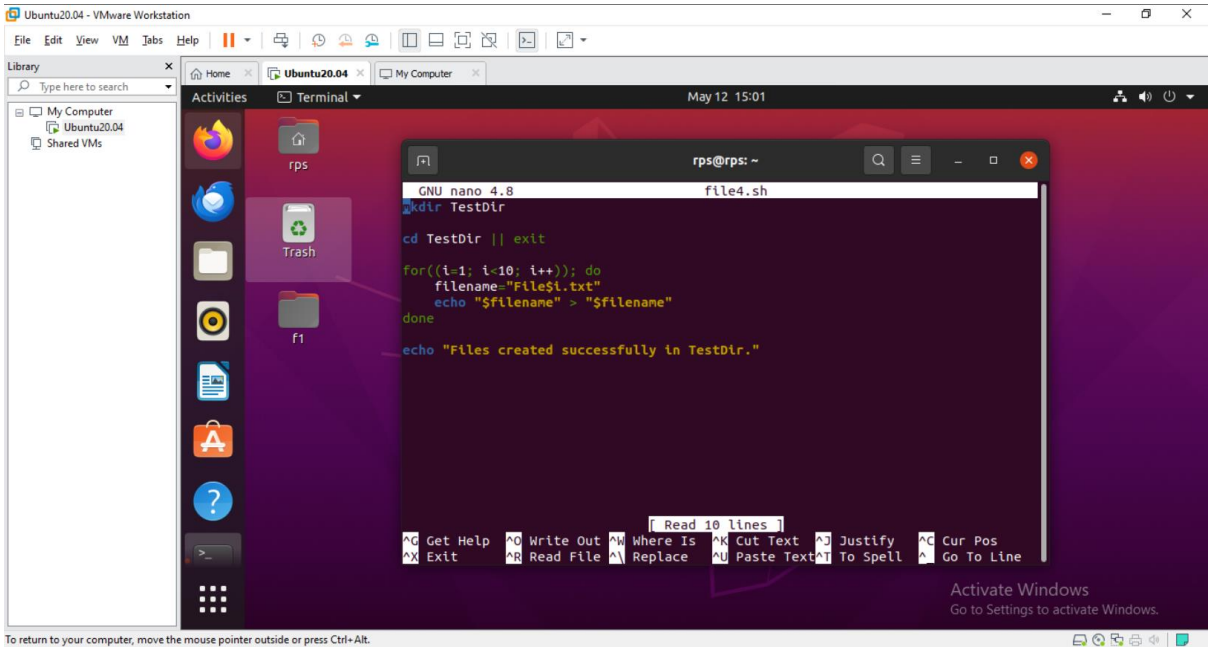


Output:

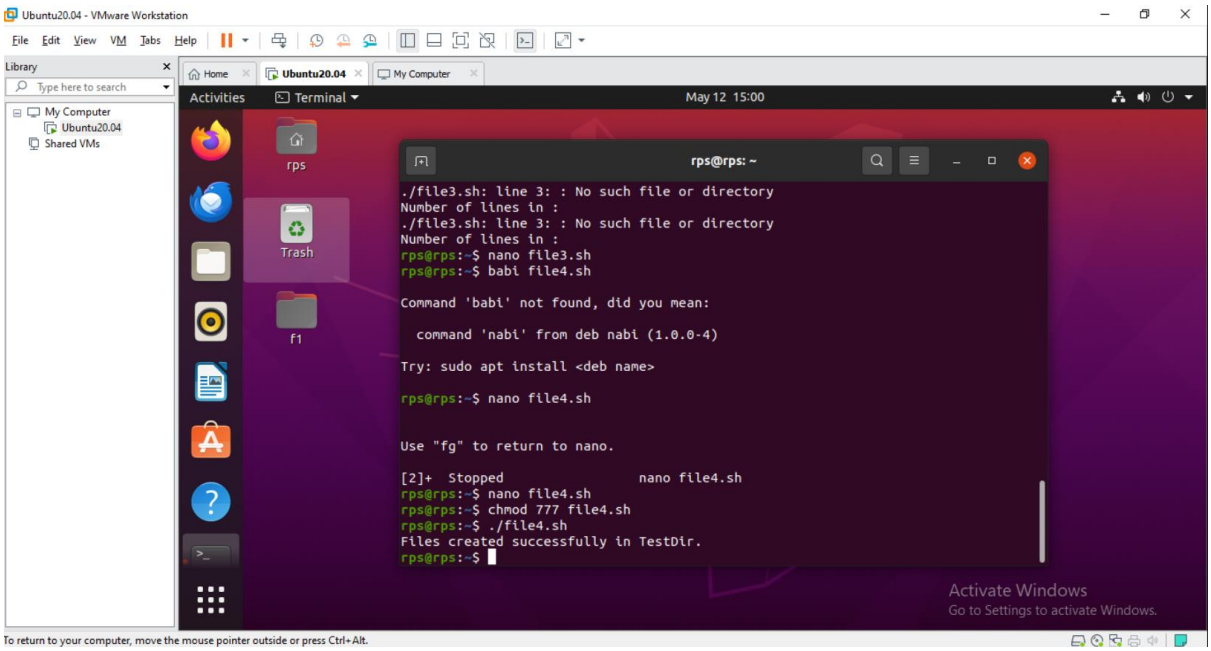


Assignment 4: Write a script that creates a directory named TestDir and inside it, creates ten files named File1.txt, File2.txt, ... File10.txt. Each file should contain its filename as its content (e.g., File1.txt contains "File1.txt").

Input:



Output:



Here's how the script works:

- It first creates a directory named **TestDir** using the **mkdir** command.
- Then it changes into the **TestDir** directory using **cd**.
- Inside the loop, it iterates from 1 to 10 using a **for** loop.
- For each iteration, it constructs the filename (**File1.txt**, **File2.txt**, etc.).
- It uses **echo** to write the filename to the corresponding file.
- After the loop completes, it prints "Files created successfully in TestDir.".

When you run this script, it will create the **TestDir** directory and populate it with ten files named **File1.txt**, **File2.txt**, ... **File10.txt**, 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.

Here is the code for the above question:

```
#!/bin/bash

# Debug mode function
debug_mode() {
  if [ "$debug" = true ]; then
    echo "Debug Mode Enabled"
    set -x  # Enable debugging output
  fi
}

# Function to create directory and files
create_files() {
  # Check if directory already exists
  if [ -d "$dir" ]; then
    echo "Error: Directory '$dir' already exists."
    exit 1
```

```
    fi

    # Create the directory
    mkdir -p "$dir" || {
        echo "Error: Failed to create directory '$dir'. Check permissions."
        exit 1
    }

    # Change to the directory
    cd "$dir" || {
        echo "Error: Failed to change directory to '$dir'. Check permissions."
        exit 1
    }

    # Loop to create ten files
    for ((i=1; i<=10; i++)); do
        filename="File$i.txt"
        # Create the file with the filename as its content
        echo "$filename" > "$filename" || {
            echo "Error: Failed to create file '$filename'. Check permissions."
            exit 1
        }
    done

    echo "Files created successfully in $dir."
}

# Main script

# Enable or disable debug mode
debug=false
if [ "$1" = "--debug" ]; then
```

```
  debug=true
fi
debug_mode

# Specify the directory name
dir="TestDir"

# Call the function to create directory and files
create_files
```

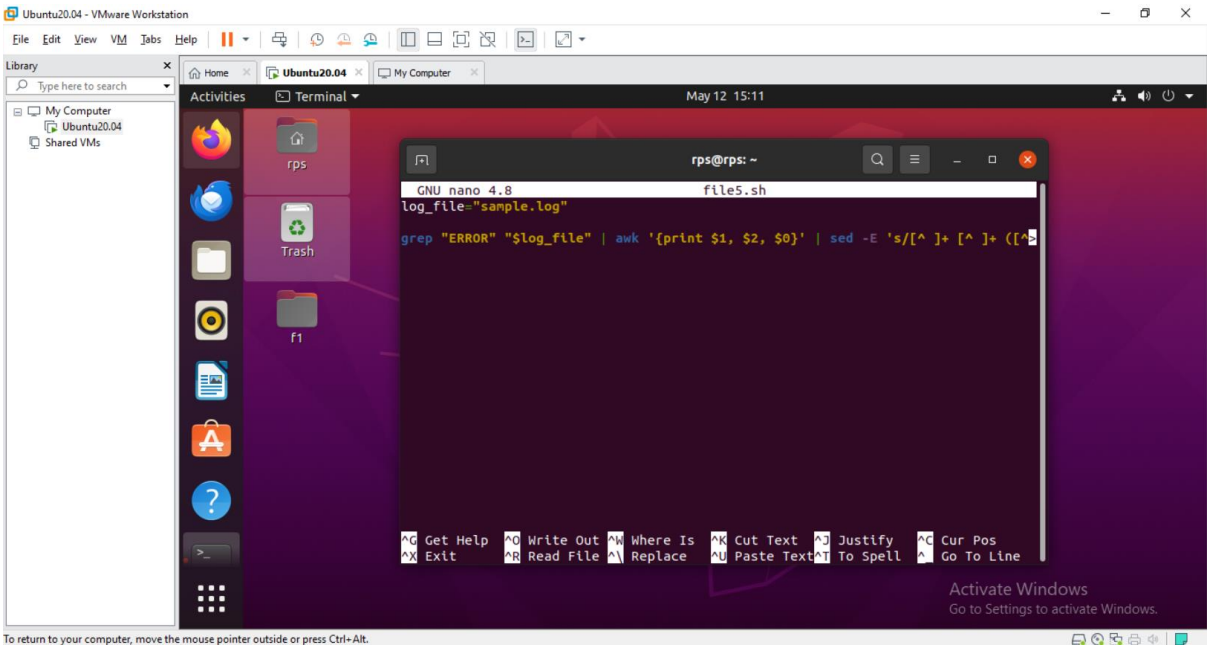
1. I added a **debug_mode** function to enable or disable debug mode based on the presence of the **--debug** flag.
2. Inside the **create_files** function:
    - I added error handling to check if the directory already exists before creating it.
    - I added error handling for creating the directory and changing into it.
    - I added error handling for creating each file.
3. In the main script:
    - I added a conditional to check if the **--debug** flag is provided. If so, debug mode is enabled.
    - I call the **debug_mode** function to enable debug mode if necessary.

When you run this script, it will handle errors such as the directory already existing or lacking permissions to create files. Additionally, you can enable debug mode by passing **--debug** as a command-line argument to get additional information during execution.

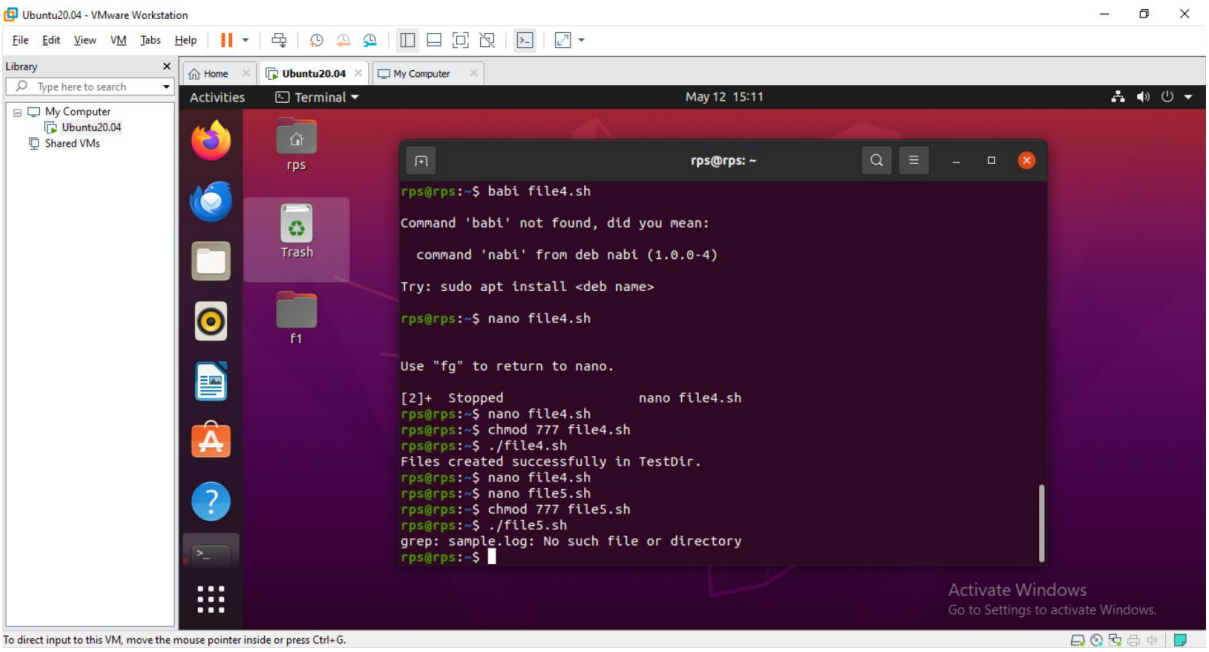
Assignment 6: Given a sample log file, write a script using grep to extract all lines containing "ERROR". Use awk to print the date, time, and error message of each extracted line.

Data Processing with sed

Input:



Output:



Explanation:

1. **grep "ERROR" "$log_file"**: This command extracts all lines containing "ERROR" from the sample log file specified by **log_file**.
2. **awk '{print $1, $2, $0}'**: This command is used to print the date, time, and the entire line (**$0**) of each extracted line. **$1** and **$2** represent the first two fields (date and time) in the log file.
3. **sed -E 's/^[^]+ [^]+ ([^]+) ([^]+) (.+)$/\1 \2 \3/'**: This **sed** command processes each line to extract only the date, time, and error message. It uses a regular expression to capture the date and time in the first two fields and the error message in the remaining part of the line. The captured groups are then rearranged in the desired format using backreferences (**\1**, **\2**, **\3**).

When you run this script, it will extract all lines containing "ERROR" from the sample log file and print the date, time, and error message of each extracted line. The output will be formatted as **date time error_message**.

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's a bash script that takes a text file as input and replaces all occurrences of "old_text" with "new_text" using sed, then outputs the result to a new file:

bash

#!/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="$3" # Perform the replacement using sed and output the result to a new file sed "s/$old_text/$new_text/g" "$input_file" > "${input_file%.txt}_replaced.txt" echo "Replacement complete. Output saved to ${input_file%.txt}_replaced.txt"

Explanation:

1. The script checks if the correct number of arguments is provided. If not, it displays a usage message and exits.
2. It assigns the input file path ($1), old text ($2), and new text ($3) to variables.
3. It uses sed to perform the replacement (s/old_text/new_text/g) on the input file and redirects the output to a new file.
4. The output file name is generated by appending "_replaced.txt" to the original file name using parameter expansion (${input_file%.txt}_replaced.txt).
5. Finally, it prints a message indicating that the replacement is complete and specifies the path to the output file.

To use this script, save it to a file (e.g., replace_text.sh), make it executable (chmod +x replace_text.sh), and then run it with the input file path, old text, and new text as arguments. For example:

bash

./replace_text.sh input.txt old_text new_text

This will replace all occurrences of "old_text" with "new_text" in the input.txt file and save the result to a new file named input_replaced.txt. Adjust the arguments as needed for your specific use case.