Assignment - 1 Day-7 **Ankit Kumar keshri**

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".

```
#!/bin/bash

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

- `#!/bin/bash`: This line specifies the shell to be used to execute the script, in this case, Bash.
- `[ -f "myfile.txt" ]`: This is the condition that checks if the file `myfile.txt` exists in the current directory. The `-f` flag checks if the file exists and is a regular file.
- `echo "File exists"`: If the file exists, this command prints "File exists" to the standard output.
- `echo "File not found"`: If the file does not exist, this command prints "File not found" to the standard output.

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.

```
#!/bin/bash

echo "Enter numbers (enter '0' to exit):"

while true; do
    read -p "Enter a number: " number
```

```
    # Check if the input is '0'
    if [ "$number" -eq 0 ]; then
        echo "Exiting..."
        break
    fi

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

- `#!/bin/bash`: Specifies the shell to be used to execute the script.
- `echo "Enter numbers (enter '0' to exit):"`: Prints a message prompting the user to enter numbers.
- `while true; do`: Starts an infinite loop.
- `read -p "Enter a number: " number`: Prompts the user to enter a number and stores the input in the variable `number`.
- `if [ "$number" -eq 0 ]; then`: Checks if the input is '0'. If it is, the script prints "Exiting..." and breaks out of the loop.
- `if [ "$((number % 2))" -eq 0 ]; then`: Checks if the number is even by calculating the remainder when divided by 2. If the remainder is 0, the number is even; otherwise, it's odd.
- Inside the loop, the script continuously prompts the user for numbers until '0' is entered, printing whether each number is odd or even.

You can save this script to a file (e.g., `check_odd_even.sh`), make it executable with the command `chmod +x check_odd_even.sh`, and then run it with `./check_odd_even.sh`.

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.

```
#!/bin/bash

# Function to count lines in a file
count_lines_in_file() {
    local filename="$1"
    local num_lines=$(wc -l < "$filename")
    echo "Number of lines in $filename: $num_lines"
}

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

- `#!/bin/bash`: Specifies the shell to be used to execute the script.
- `count_lines_in_file() { ... }`: Defines a function named `count_lines_in_file` which takes a filename as an argument.
- `local filename="$1"`: Assigns the value of the first argument (the filename) to the variable `filename`.
- `local num_lines=$(wc -l < "$filename")`: Uses the `wc` command to count the number of lines in the file specified by the filename. The `<` symbol is used for input redirection. The output of `wc -l` is stored in the variable `num_lines`.
- `echo "Number of lines in $filename: $num_lines"`: Prints the filename and the number of lines in the file.

- The script then calls the `count_lines_in_file` function with different filenames: "file1.txt", "file2.txt", and "file3.txt".

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").

```
#!/bin/bash

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

# Navigate to the TestDir directory
cd TestDir || exit

# Create ten files named File1.txt, File2.txt, ..., File10.txt
for ((i = 1; i <= 10; i++)); do
   filename="File$i.txt"
   echo "$filename" > "$filename"
done

echo "Files created successfully."
```

- `#!/bin/bash`: Specifies the shell to be used to execute the script.
- `mkdir -p TestDir`: Creates the directory `TestDir` if it doesn't already exist. The `-p` option ensures that the command doesn't produce an error if the directory already exists.
- `cd TestDir || exit`: Navigates into the `TestDir` directory. If for some reason navigation fails, the script exits.
- `for ((i = 1; i <= 10; i++)); do`: Starts a loop to create ten files.
- `filename="File$i.txt"`: Constructs the filename for each iteration of the loop (e.g., File1.txt, File2.txt, ..., File10.txt).
- `echo "$filename" > "$filename"`: Writes the filename (e.g., "File1.txt") into the corresponding file.
- `echo "Files created successfully."`: Prints a message indicating that the files have been created successfully.

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.

let's outline the modifications you can make to your script to handle errors and add a debugging mode in a Linux environment.
Handling Errors:
Check if the directory already exists before attempting to create it. If it exists, you can either abort the script or prompt the user for further action.
Check for permissions to create files in the specified directory. If the script lacks the necessary permissions, it should gracefully handle the error, informing the user and possibly exiting with an error code.
Debugging Mode:
Add an option to enable debugging mode, which will print additional information during script execution. This can include variable values, intermediate steps, or any other information that helps in understanding the script's behavior.
Here's a basic outline of how you could implement these changes:

```
#!/bin/bash
# Function to print debug messages
debug() {
   if [ "$DEBUG" = true ]; then
      echo "DEBUG: $1"
   fi
}
# Function to handle errors
handle_error() {
   echo "Error: $1"
   exit 1
}
# Parse command line arguments
while getopts ":d" opt; do
   case ${opt} in
      d )
         DEBUG=true
         ;;
      \? )
         echo "Usage: $0 [-d (enable debugging)]"
         exit 1
         ;;
   esac
done
shift $((OPTIND -1))
# Check if directory already exists
```

```
if [ -d "$1" ]; then
    handle_error "Directory already exists. Aborting."
fi
# Check permissions to create files
if [ ! -w "$1" ]; then
    handle_error "Insufficient permissions to create files in the specified directory."
fi
# Create the directory
mkdir -p "$1" || handle_error "Failed to create directory."
debug "Directory created successfully."
# Add more script logic here...
echo "Script execution completed successfully."
```

In this script:
The debug function prints messages if debugging mode is enabled.
The handle_error function prints an error message and exits the script with a non-zero exit code.
Command-line options are parsed using getopts. -d enables debugging mode.
The existence of the directory is checked using [-d "$1"].
Permissions to write to the directory are checked using [! -w "$1"].
Debug messages are printed using debug function calls.
You can expand the script with additional functionality as needed.

To use the script, you would run it like this:
./your_script.sh -d /path/to/directory
This will enable debugging mode and attempt to create the specified directory.

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.

```
#!/bin/bash
# Sample log file
logfile="sample.log"
# Use grep to extract lines containing "ERROR"
error_lines=$(grep "ERROR" "$logfile")
# Use awk to print date, time, and error message of each extracted line
echo "$error_lines" | awk '{print $1, $2, $NF}'
```

In this script:
We specify the path to the sample log file.

grep "ERROR" "$logfile" extracts lines containing "ERROR" from the log file.
awk '{print $1, $2, $NF}' prints the first and second fields (date and time) along with the last field (error message) of each extracted line.
sed 's/^[0-9]\{4\}-[0-9]\{2\}-[0-9]\{2\} [0-9]\{2\}:[0-9]\{2\}:[0-9]\{2\}//' removes the timestamp from each error message. Adjust the regular expression as needed for your timestamp format.
To use the script, save it to a file (e.g., extract_errors.sh), make it executable (chmod +x extract_errors.sh), and run it:

./extract_errors.sh
This script will extract lines containing "ERROR" from the sample log file, print the date, time, and error message of each extracted line using awk, and perform additional data processing on error messages using sed.

```
# Additional data processing with sed
# For example, removing timestamps from error messages
echo "$error_lines" | sed 's/^[0-9]\{4\}-[0-9]\{2\}-[0-9]\{2\} [0-9]\{2\}:[0-9]\{2\}:[0-9]\{2\}//'
```

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.

```
#!/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 input file and old/new text
input_file="$1"
old_text="$2"
new_text="$3"
# Check if input file exists
if [ ! -f "$input_file" ]; then
    echo "Error: Input file '$input_file' not found."
    exit 1
fi
# Perform the replacement using sed and output to a new file
output_file="${input_file}.new"
sed "s/$old_text/$new_text/g" "$input_file" > "$output_file"
echo "Replacement complete. Result written to '$output_file'."
```

To use this script:
Save the script to a file (e.g., replace_text.sh).
Make it executable: chmod +x replace_text.sh.
Run it with three arguments: the input file, old text, and new text:

./replace_text.sh input_file.txt old_text new_text
This script checks if the correct number of arguments is provided, ensures the input file exists, performs the replacement using sed, and outputs the result to a new file.