# **Assignment 05:**

# **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.**

## 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.

#!/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" ].

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 06:**

# **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

# 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.

# **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