1. Write a shell script that demonstrates the use of cat, ls, more, ps, chmod, and grep commands on files and processes in the current directory.

#!/bin/bash

echo "----- Listing files in the current directory -----"

ls -l

echo ""

# Create a sample file

echo "Creating sample file 'example.txt'..."

echo -e "Line 1\nLine 2\nLine 3\nSearch this line\nAnother line" > example.txt

echo "----- Displaying contents of example.txt using cat -----"

cat example.txt

echo ""

echo "----- Viewing contents of example.txt using more -----"

more example.txt

echo ""

echo "----- Searching for the word 'Search' in example.txt using grep -----"

grep "Search" example.txt

echo ""

echo "----- Showing currently running processes using ps -----"

ps aux | head -n 10

echo ""

echo "----- Searching for 'bash' processes using grep and ps -----"

ps aux | grep "bash"

echo ""

echo "----- Changing permissions of example.txt using chmod (read-only) -----"

chmod 444 example.txt

ls -l example.txt

echo ""

echo "Cleaning up..."

rm example.txt

echo "Done."

1. Write a shell script that takes a filename as input and checks:
   1. If the file exists
   2. If it is a regular file, directory, or symbolic link
   3. Its read, write, and execute permissions using the test command.

#!/bin/bash

# Prompt for filename

read -p "Enter the filename: " filename

echo "Checking file: $filename"

echo "------------------------------"

# a. Check if the file exists

if [ -e "$filename" ]; then

    echo "The file exists."

    # b. Check file type

    if [ -f "$filename" ]; then

        echo "It is a regular file."

    elif [ -d "$filename" ]; then

        echo "It is a directory."

    elif [ -L "$filename" ]; then

        echo "It is a symbolic link."

    else

        echo "It is another type of file."

    fi

    # c. Check permissions

    [ -r "$filename" ] && echo "Read permission: Yes" || echo "Read permission: No"

    [ -w "$filename" ] && echo "Write permission: Yes" || echo "Write permission: No"

    [ -x "$filename" ] && echo "Execute permission: Yes" || echo "Execute permission: No"

else

    echo "The file does NOT exist."

fi

1. Write a shell script using if, if-else, and if-elif constructs to:
   1. Check if a file is empty
   2. If not, display the number of lines
   3. If it contains more than 10 lines, move it to a directory named long\_files

#!/bin/bash

# Prompt for filename

read -p "Enter the filename: " filename

# a. Check if file exists

if [ ! -e "$filename" ]; then

    echo "File does not exist."

    exit 1

fi

# b. Check if the file is empty

if [ ! -s "$filename" ]; then

    echo "File exists but is empty."

else

    # File is not empty; count lines

    lines=$(wc -l < "$filename")

    echo "File has $lines lines."

    # c. Check if the number of lines is greater than 10

    if [ "$lines" -gt 10 ]; then

        # Create target directory if it doesn't exist

        [ ! -d "long\_files" ] && mkdir long\_files

        # Move file

        mv "$filename" long\_files/

        echo "File moved to 'long\_files/' directory."

    elif [ "$lines" -le 10 ]; then

        echo "File has 10 or fewer lines. Not moved."

    fi

fi

1. Write a shell script using a for or while loop to list all files in a directory and display whether each file is readable, writable, and executable.

#!/bin/bash

# Prompt for directory name

read -p "Enter the directory path: " dir

# Check if the directory exists

if [ ! -d "$dir" ]; then

    echo "✘ Directory does not exist."

    exit 1

fi

echo "Checking permissions of files in: $dir"

echo "---------------------------------------"

# Loop through each item in the directory

for file in "$dir"/\*; do

    if [ -f "$file" ]; then

        echo "File: $(basename "$file")"

        # Check readability

        [ -r "$file" ] && echo "Readable" || echo "Not Readable"

        # Check writability

        [ -w "$file" ] && echo "Writable" || echo "Not Writable"

        # Check executability

        [ -x "$file" ] && echo "Executable" || echo "Not Executable"

        echo "---------------------------------------"

    fi

done

1. Write a shell script that displays a menu using a case statement with the following options:
   1. View current date and time
   2. List logged-in users
   3. Display disk usage
   4. Exit

#!/bin/bash

while true; do

    echo "====== MENU ======"

    echo "a. View current date and time"

    echo "b. List logged-in users"

    echo "c. Display disk usage"

    echo "d. Exit"

    echo "=================="

    read -p "Enter your choice [a/b/c/d]: " choice

    case $choice in

        a|A)

            echo "Current Date and Time:"

            date

            ;;

        b|B)

            echo "Logged-in Users:"

            who

            ;;

        c|C)

            echo "Disk Usage:"

            df -h

            ;;

        d|D)

            echo "Exiting"

            exit 0

            ;;

        \*)

            echo "Invalid choice. Please select a, b, c, or d."

            ;;

    esac

    echo ""  # Add a blank line for spacing

done

1. Create a shell script that accepts two filenames as command-line arguments, compares them using positional parameters, and displays which file has more lines using command substitution.

#!/bin/bash

# Check if two arguments are provided

if [ $# -ne 2 ]; then

    echo "Usage: $0 <file1> <file2>"

    exit 1

fi

file1=$1

file2=$2

# Check if both files exist

if [ ! -f "$file1" ]; then

    echo "File '$file1' does not exist or is not a regular file."

    exit 1

fi

if [ ! -f "$file2" ]; then

    echo "File '$file2' does not exist or is not a regular file."

    exit 1

fi

# Get line counts using command substitution

lines1=$(wc -l < "$file1")

lines2=$(wc -l < "$file2")

echo "'$file1' has $lines1 lines."

echo "'$file2' has $lines2 lines."

# Compare the line counts

if [ "$lines1" -gt "$lines2" ]; then

    echo "'$file1' has more lines."

elif [ "$lines2" -gt "$lines1" ]; then

    echo "'$file2' has more lines."

else

    echo "Both files have the same number of lines."

fi

1. Write a script that declares and uses normal, read-only, and exported variables. Enable tracing with set -x and set +x to debug the variable values step by step.

#!/bin/bash

echo "===== Variable Demo Script ====="

# Enable tracing

set -x

# 1. Normal variable

name="Alice"

echo "Normal Variable: name = $name"

# 2. Read-only variable

readonly id="A101"

echo "Read-only Variable: id = $id"

# 3. Exported variable

export department="Engineering"

echo "Exported Variable: department = $department"

# Disable tracing

set +x

echo ""

echo "Now demonstrating access:"

echo "Name: $name"

echo "ID: $id"

echo "Department: $department"

# Attempt to modify read-only variable (will cause an error)

echo ""

echo "Trying to modify read-only variable 'id'..."

set -x

id="B202"  # This line will produce an error

set +x

1. Write a shell script that:
   1. Accepts a username as input
   2. Checks if the user exists using finger or grep with /etc/passwd
   3. Logs the result to a file named user\_check.log

#!/bin/bash

# a. Accept username as input

read -p "Enter the username to check: " username

# b. Check if user exists using grep with /etc/passwd

if grep -q "^$username:" /etc/passwd; then

    echo "User '$username' exists."

    result="[$(date)] User '$username' exists."

else

    echo "User '$username' does NOT exist."

    result="[$(date)] User '$username' does NOT exist."

fi

# c. Log the result to user\_check.log

echo "$result" >> user\_check.log

1. Write a shell script that compiles and executes a simple C program (e.g., to calculate factorial), accepts input from the user, and displays the result.

#!/bin/bash

# Name of the C source and executable

c\_file="factorial.c"

exe\_file="factorial"

# Create the C program

cat > $c\_file << 'EOF'

#include <stdio.h>

int main() {

    int n, i;

    unsigned long long fact = 1;

    printf("Enter a positive integer: ");

    if (scanf("%d", &n) != 1 || n < 0) {

        printf("Invalid input! Please enter a positive integer.\n");

        return 1;

    }

    for(i = 1; i <= n; ++i) {

        fact \*= i;

    }

    printf("Factorial of %d = %llu\n", n, fact);

    return 0;

}

EOF

# Compile the C program

gcc $c\_file -o $exe\_file

if [ $? -ne 0 ]; then

    echo "Compilation failed."

    exit 1

fi

# Run the executable

./$exe\_file

1. Write a shell script that:
   1. Accepts a directory name as input
   2. Checks if the directory exists
   3. Creates a compressed backup (in .tar.gz format) with the current date and time in the filename
   4. Stores the backup in a backup directory (create it if it doesn’t exist)

#!/bin/bash

# a. Accept directory name as input

read -p "Enter the directory to backup: " dir

# b. Check if directory exists

if [ ! -d "$dir" ]; then

    echo "Directory '$dir' does not exist."

    exit 1

fi

# Create backup directory if it doesn't exist

backup\_dir="backup"

if [ ! -d "$backup\_dir" ]; then

    mkdir "$backup\_dir"

    echo "Created backup directory: $backup\_dir"

fi

# c. Create compressed backup with date/time in filename

timestamp=$(date +"%Y%m%d\_%H%M%S")

backup\_file="${backup\_dir}/${dir##\*/}\_backup\_$timestamp.tar.gz"

tar -czf "$backup\_file" "$dir"

if [ $? -eq 0 ]; then

    echo "Backup successful: $backup\_file"

else

    echo "Backup failed."

fi