Task 1: File Management Script

- **Write a Bash script that:**
    1. Creates a directory named "backup" in the user's home directory.
    2. Copies all .txt files from the current directory into the "backup" directory.
    3. Appends the current date and time to the filenames of the copied files.

```
#!/bin/bash
# Task 1: File Management Script
# Step 1: Create a "backup" directory in the user's home directory if it doesn't exist
backup_dir="$HOME/backup"
if [ ! -d "$backup_dir" ]; then
   mkdir "$backup_dir"
   echo "Directory $backup_dir created."
else
   echo "Directory $backup_dir already exists."
fi
# Step 2: Copy all .txt files from the current directory to the "backup" directory
for file in *.txt; do
   if [ -f "$file" ]; then  # Check if there are any .txt files
      # Step 3: Append the current date and time to the filenames of the copied files
      current_date_time=$(date +"%Y-%m-%d_%H-%M-%S")
      filename=$(basename "$file")
      cp "$file" "$backup_dir/${filename%.txt}_$current_date_time.txt"
      echo "$file has been copied to $backup_dir with the new name ${filename%.txt}_$current_date_time.txt"
   else
      echo "No .txt files found in the current directory."
      break
   fi
done
```

Task 2: System Health Check

- **Create a script that:**
    1. Checks the system's CPU and memory usage.
    2. Reports if the CPU usage is above 80% or if the available memory is below 20%.
    3. Logs the results to a file named system_health.log.

```
#!/bin/bash
# Task 2: System Health Check Script
# Log file location
log_file="system_health.log"
# Get CPU usage (in percentage)
# "top -bn1" gives a snapshot of CPU usage, then "grep 'Cpu(s)'" extracts CPU data, and "awk" parses the usage.
cpu_usage=$(top -bn1 | grep "Cpu(s)" | awk '{print 100 - $8}')
cpu_threshold=80.0
# Get available memory (in percentage)
# "free -m" provides memory info, then "awk" calculates the available memory percentage.
total_memory=$(free -m | awk '/^Mem:/{print $2}')
used_memory=$(free -m | awk '/^Mem:/{print $3}')
memory_usage=$(echo "scale=2; ($used_memory/$total_memory)*100" | bc)
memory_threshold=20.0
# Current date and time
current_time=$(date +"%Y-%m-%d %H:%M:%S")
# Initialize log entry
log_entry="$current_time - CPU: $cpu_usage%, Memory Usage: $memory_usage%"
# CPU usage check
if (( $(echo "$cpu_usage > $cpu_threshold" | bc -l) )); then
   log_entry="$log_entry | ALERT: CPU usage above threshold ($cpu_threshold%)"
fi
```

```
# Memory usage check
available_memory=$(echo "100 - $memory_usage" | bc)
if (( $(echo "$available_memory < $memory_threshold" | bc -l) )); then
   log_entry="$log_entry | ALERT: Available memory below threshold ($memory_threshold%)"
fi

# Append log entry to the log file
echo "$log_entry" >> "$log_file"

# Output to console
echo "System Health Check complete. Log entry added to $log_file."
```

Task 3: User Account Management

- Write a script that:
    1. Reads a list of usernames from a file (e.g., user_list.txt).
    2. Creates a new user for each username.
    3. Generates a random password for each user and saves the username and password to a file named credentials.txt.

```
#!/bin/bash

# Task 3: User Account Management Script

# Input file containing list of usernames
user_list="user_list.txt"

# Output file to store usernames and passwords
credentials_file="credentials.txt"

# Check if user_list.txt exists
```

```
if [ ! -f "$user_list" ]; then
    echo "Error: $user_list not found."
    exit 1
fi

# Function to generate a random password
generate_password() {
    # Generate a random 12-character password (with letters, numbers, and special characters)
    echo "$(tr -dc 'A-Za-z0-9!@#$%&*()_+=' < /dev/urandom | head -c 12)"
}

# Initialize credentials file
echo "Username | Password" > "$credentials_file"
echo "--------------------" >> "$credentials_file"

# Loop through each username in the user_list.txt file
while IFS= read -r username; do
    # Check if the username is non-empty
    if [ -z "$username" ]; then
        echo "Skipping empty username."
        continue
    fi

    # Check if the user already exists
    if id "$username" &>/dev/null; then
        echo "User $username already exists. Skipping..."
        continue
    fi

    # Create the new user without a home directory
    sudo useradd -m "$username"
```

```
    if [ $? -eq 0 ]; then
        echo "User $username created successfully."
    else
        echo "Failed to create user $username."
        continue
    fi

    # Generate a random password for the new user
    password=$(generate_password)

    # Set the user's password
    echo "$username:$password" | sudo chpasswd

    # Save the username and password to the credentials file
    echo "$username | $password" >> "$credentials_file"
    echo "Credentials for $username saved to $credentials_file."

done < "$user_list"

echo "User account creation process completed."
```

Task 4: Automated Backup

- Create a script that:
    1. Takes a directory path as input from the user.
    2. Compresses the directory into a .tar.gz file.
    3. Saves the compressed file with a name that includes the current date (e.g., backup_2023-08-20.tar.gz).

```
#!/bin/bash

# Task 4: Automated Backup Script

# Step 1: Ask the user for the directory to back up
read -p "Enter the directory path you want to back up: " directory

# Check if the directory exists
if [ ! -d "$directory" ]; then
   echo "Error: Directory $directory does not exist."
   exit 1
fi

# Step 2: Generate a name for the backup file
# Use the current date in the format YYYY-MM-DD
current_date=$(date +"%Y-%m-%d")
backup_filename="backup_${current_date}.tar.gz"

# Step 3: Compress the directory into a .tar.gz file
tar -czf "$backup_filename" -C "$(dirname "$directory")" "$(basename "$directory")"

# Step 4: Notify the user about the backup location
if [ $? -eq 0 ]; then
```

```
    echo "Backup of $directory completed successfully."
    echo "Backup file: $backup_filename"
else
    echo "Error occurred while creating the backup."
Fi
```

Task 5: Simple To-Do List

- Create a Bash script that:
    1. Implements a simple command-line to-do list.
    2. Allows the user to add tasks, view tasks, and remove tasks.
    3. Saves the tasks to a file (e.g., todo.txt).

```
#!/bin/bash

# Task 5: Simple To-Do List Script

# To-do list file
todo_file="todo.txt"

# Create the file if it doesn't exist
if [ ! -f "$todo_file" ]; then
    touch "$todo_file"
fi

# Function to display menu options
display_menu() {
    echo "To-Do List Menu:"
    echo "1. Add a new task"
    echo "2. View all tasks"
    echo "3. Remove a task"
```

```
    echo "4. Exit"
}

# Function to add a task
add_task() {
    read -p "Enter a new task: " task
    if [ -n "$task" ]; then
        echo "$task" >> "$todo_file"
        echo "Task added: $task"
    else
        echo "Task cannot be empty!"
    fi
}

# Function to view all tasks
view_tasks() {
    if [ -s "$todo_file" ]; then
        echo "Your To-Do List:"
        nl "$todo_file"  # Display tasks with line numbers
    else
        echo "Your To-Do List is empty."
    fi
}

# Function to remove a task
remove_task() {
    view_tasks
    if [ -s "$todo_file" ]; then
        read -p "Enter the task number to remove: " task_number
        if [[ "$task_number" =~ ^[0-9]+$ ]]; then
            sed -i "${task_number}d" "$todo_file"
```

```
            echo "Task $task_number removed."
        else
            echo "Invalid task number."
        fi
    fi
}

# Main loop
while true; do
    display_menu
    read -p "Choose an option (1-4): " choice

    case $choice in
        1)
            add_task
            ;;
        2)
            view_tasks
            ;;
        3)
            remove_task
            ;;
        4)
            echo "Exiting the To-Do List."
            exit 0
            ;;
        *)
            echo "Invalid option. Please choose 1, 2, 3, or 4."
            ;;
    esac
done
```

Task 6: Automated Software Installation

- Write a script that:
    1. Reads a list of software package names from a file (e.g., packages.txt).
    2. Installs each package using the appropriate package manager (apt, yum, etc.).
    3. Logs the installation status of each package.

```
#!/bin/bash

# Task 6: Automated Software Installation Script

# Input file containing a list of package names
package_file="packages.txt"

# Log file to store installation statuses
log_file="installation_log.txt"

# Detect the package manager (apt for Debian/Ubuntu, yum for CentOS/RHEL)
if command -v apt &> /dev/null; then
   package_manager="apt"
elif command -v yum &> /dev/null; then
   package_manager="yum"
else
   echo "Error: Neither apt nor yum package manager found."
   exit 1
fi

# Function to install a package
install_package() {
   local package=$1
   echo "Installing $package..."
```

```
    if [ "$package_manager" == "apt" ]; then
        sudo apt update -y >> "$log_file" 2>&1
        sudo apt install -y "$package" >> "$log_file" 2>&1
    elif [ "$package_manager" == "yum" ]; then
        sudo yum install -y "$package" >> "$log_file" 2>&1
    fi

    # Check if the package was successfully installed
    if [ $? -eq 0 ]; then
        echo "$package installation successful." >> "$log_file"
        echo "Package $package installed successfully."
    else
        echo "$package installation failed." >> "$log_file"
        echo "Failed to install package $package."
    fi
}

# Check if packages.txt exists
if [ ! -f "$package_file" ]; then
    echo "Error: $package_file not found."
    exit 1
fi

# Initialize log file
echo "Installation Log - $(date)" > "$log_file"
echo "----------------------------------------" >> "$log_file"

# Loop through the package names in packages.txt and install each one
while IFS= read -r package; do
    if [ -n "$package" ]; then  # Check if the line is non-empty
        install_package "$package"
```

```
    else
        echo "Skipping empty line."
    fi
done < "$package_file"

echo "Software installation process completed."
```

Task 7: Text File Processing

- Create a script that:
    1. Takes a text file as input.
    2. Counts and displays the number of lines, words, and characters in the file.
    3. Finds and displays the longest word in the file(assignment_pw).

```
#!/bin/bash

# Task 7: Text File Processing Script

# Step 1: Ask the user for the input file
read -p "Enter the path to the text file: " file

# Check if the file exists
if [ ! -f "$file" ]; then
    echo "Error: File $file does not exist."
    exit 1
fi

# Step 2: Count the number of lines, words, and characters in the file
line_count=$(wc -l < "$file")
word_count=$(wc -w < "$file")
char_count=$(wc -m < "$file")
```

```
# Step 3: Find the longest word in the file

# We'll use `tr` to replace all non-alphabet characters with spaces, and then `awk` to find the longest word

longest_word=$(tr -cs 'A-Za-z' '\n' < "$file" | awk '{ if (length > max) { max = length; longest = $0 } } END { print longest }')


# Display the results

echo "File: $file"

echo "Number of lines: $line_count"

echo "Number of words: $word_count"

echo "Number of characters: $char_count"

echo "Longest word: $longest_word"
```