1. First Task - File Management Script

```
# Define the backup directory path
BACKUP_DIR="$HOME/backup"

# Create the backup directory if it doesn't exist
mkdir -p "$BACKUP_DIR"

# Get the current date and time in format: YYYY-MM-DD_HH-MM-SS
TIMESTAMP=$(date +"%Y-%m-%d_%H-%M-%S")

# Copy all .txt files from the current directory to the backup directory,appending the
timestamp to the filename
for file in *.txt; do
    if [[ -f "$file" ]]; then
        BASENAME=$(basename "$file" .txt)
        cp "$file" "$BACKUP_DIR/${BASENAME}_$TIMESTAMP.txt"
    fi
done

echo "Backup completed. Files copied to: $BACKUP_DIR"
```

2. Second Task - System Health Check

```
# Log file
LOG_FILE="system_health.log"

# Get current timestamp
TIMESTAMP=$(date "+%Y-%m-%d %H:%M:%S")

# Get CPU usage (average over 1 second)
CPU_USAGE=$(top -bn1 | grep "Cpu(s)" | awk '{print 100 - $8}')
CPU_USAGE_INT=${CPU_USAGE%.*}  # Convert to integer

# Get total and available memory (in MB)
read TOTAL_MEM AVAILABLE_MEM <<< $(free -m | awk '/^Mem:/ {print $2, $7}')

# Calculate available memory percentage
MEM_PERCENT=$(( AVAILABLE_MEM * 100 / TOTAL_MEM ))

# Log the current status
echo "[$TIMESTAMP] CPU: ${CPU_USAGE}% | Available Memory:
${MEM_PERCENT}%" >> "$LOG_FILE"

# Check thresholds
if [ "$CPU_USAGE_INT" -gt 80 ]; then
    echo "[$TIMESTAMP] WARNING: High CPU usage detected: ${CPU_USAGE}%!"
>> "$LOG_FILE"
fi
```

```
if [ "$MEM_PERCENT" -lt 20 ]; then
    echo "[$TIMESTAMP] WARNING: Low available memory: ${MEM_PERCENT}%!"
>> "$LOG_FILE"
fi

echo "[$TIMESTAMP] System health check completed." >> "$LOG_FILE"
```

3. Third Tas- User Account Management

```
I# nput file with usernames
USER_LIST="user_list.txt"

# Output file for credentials
CREDENTIALS_FILE="credentials.txt"

# Clear or create the credentials file
> "$CREDENTIALS_FILE"

# Loop through each username in the list
while IFS= read -r username || [[ -n "$username" ]]; do
    Skip empty lines
    [[ -z "$username" ]] && continue

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

    # Create user
    useradd -m "$username"

    # Generate random password
    password=$(openssl rand -base64 12)

    # Set the password
    echo "$username:$password" | chpasswd

    # Save credentials
    echo "$username : $password" >> "$CREDENTIALS_FILE"

    echo "Created user: $username"
done < "$USER_LIST"

echo "All users processed. Credentials saved to $CREDENTIALS_FILE."
```

4. Fourth Task - Automated Backup

```
# Prompt user for directory path
```

```
read -p "Enter the full path of the directory to back up: " DIR_PATH

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

# Get base directory name (e.g., /home/user/mydata → mydata)
DIR_NAME=$(basename "$DIR_PATH")

# Get current date
DATE=$(date +"%Y-%m-%d")

# Set output filename
ARCHIVE_NAME="backup_${DIR_NAME}_${DATE}.tar.gz"

# Compress the directory
tar -czf "$ARCHIVE_NAME" -C "$(dirname "$DIR_PATH")" "$DIR_NAME"

# Notify user
echo "Directory '$DIR_PATH' compressed into '$ARCHIVE_NAME'"
```

5. Fifth Task - Simple To-Do List

```
TODO_FILE="todo.txt"

# Ensure the todo file exists
touch "$TODO_FILE"

function show_menu() {
    echo "==== Simple To-Do List ===="
    echo "1. View Tasks"
    echo "2. Add Task"
    echo "3. Remove Task"
    echo "4. Exit"
    echo "============================"
}

function view_tasks() {
    echo "---- Your To-Do List ----"
    if [[ ! -s "$TODO_FILE" ]]; then
        echo "No tasks found."
    else
        nl -w2 -s'. ' "$TODO_FILE"
    fi
    echo "-------------------------"
}
```

```
function add_task() {
    read -p "Enter a new task: " task
    echo "$task" >> "$TODO_FILE"
    echo "Task added."
}

function remove_task() {
    view_tasks
    read -p "Enter the task number to remove: " num
    if [[ "$num" =~ ^[0-9]+$ ]]; then
        sed -i "${num}d" "$TODO_FILE"
        echo "Task removed."
    else
        echo "Invalid input. Please enter a number."
    fi
}

# Main loop
while true; do
    show_menu
    read -p "Choose an option (1-4): " choice
    case "$choice" in
        1) view_tasks ;;
        2) add_task ;;
        3) remove_task ;;
        4) echo "Goodbye!"; break ;;
        *) echo "Invalid option. Please try again." ;;
    esac
done
```

6. Sixth Task - Automated Software Installation

```
# File paths
PACKAGE_FILE="packages.txt"
LOG_FILE="install_log.txt"

# Clear previous log
> "$LOG_FILE"

# Detect package manager
if command -v apt >/dev/null 2>&1; then
    PKG_MGR="apt"
    INSTALL_CMD="sudo apt-get install -y"
elif command -v dnf >/dev/null 2>&1; then
    PKG_MGR="dnf"
    INSTALL_CMD="sudo dnf install -y"
elif command -v yum >/dev/null 2>&1; then
    PKG_MGR="yum"
    INSTALL_CMD="sudo yum install -y"
```

```
else
    echo "No supported package manager found (apt, yum, dnf)." | tee -a
"$LOG_FILE"
    exit 1
fi

echo "Using package manager: $PKG_MGR" | tee -a "$LOG_FILE"

# Read package names and install
while IFS= read -r package || [ -n "$package" ]; do
    if [[ -z "$package" ]]; then
        continue
    fi

    echo "Installing $package..." | tee -a "$LOG_FILE"

    if $INSTALL_CMD "$package" >> "$LOG_FILE" 2>&1; then
        echo "[SUCCESS] $package installed." | tee -a "$LOG_FILE"
    else
        echo "[ERROR] Failed to install $package." | tee -a "$LOG_FILE"
    fi

    echo "-----------------------------------" >> "$LOG_FILE"
done < "$PACKAGE_FILE"

echo "Installation complete. Check $LOG_FILE for details."
```

7. Seventh Task - Text File Processing

```
# Check if filename is passed
if [ $# -ne 1 ]; then
    echo "Usage: $0 <filename>"
    exit 1
fi

FILE="$1"

# Check if file exists and is readable
if [ ! -f "$FILE" ] || [ ! -r "$FILE" ]; then
    echo "Error: File does not exist or is not readable."
    exit 2
fi

# Count lines, words, and characters
LINES=$(wc -l < "$FILE")
WORDS=$(wc -w < "$FILE")
CHARS=$(wc -m < "$FILE")

# Find the longest word
```

```
LONGEST_WORD=$(tr -c '[:alnum:]' '[\n*]' < "$FILE" | awk 'length > max { max =
length; word = $0 } END { print word }')

# Display results
echo "File: $FILE"
echo "Lines     : $LINES"
echo "Words     : $WORDS"
echo "Characters: $CHARS"
echo "Longest word: $LONGEST_WORD"
```