CS2052 Software Development Tools

Course Project Report

Task Manager Program

Student Name	Student ID	Section
AlJood Yasir Alsaleh	445006878	2
Huda Alaa Sleem	44511003	2
Retaj Hussain Alhazmi	445006594	2
Rana Jamal Alsheikh	445003045	2

Supervised by:

Dr. Eman Talal Alharbi

Table of Contents

1. Introduction

1.1 Purpose of the Project:

Managing daily tasks efficiently is a common challenge between many individuals, especially without dedicated software. This project aims to create a simple command-line task manager using a Bash script. The Task Manager we created strives to make users lives easier by helping them organize their time and complete their tasks in a smooth and efficient manner.

1.2 Project Overview:

The Task Manager allows users to add tasks with due dates, view pending tasks sorted by due date, mark tasks as complete, count the number of pending & complete tasks, and view the current date & time all managed through plain text files. The tool also provides the user with many words of motivation when achieving tasks, creating an extremely user-friendly environment.

1.3 Project Motivation:

The motivation behind this project is to demonstrate the power of shell scripting for automating common workflows, while integrating version control with Git to track changes in tasks. As well as providing us artificial intelligence students with new skills and expertise that will benefit us in our educational path and career in the future.

2. Design & Implementation

2.1 Design Strategy:

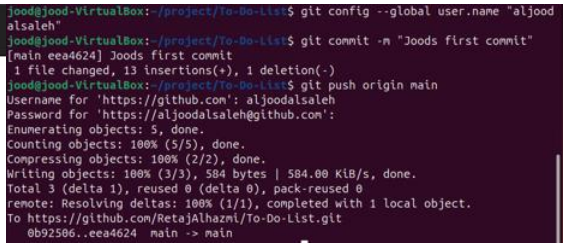
This project consists of a single Bash script (ToDoList.sh) that manages **two** files:
1. pending_tasks.txt: Stores all current, incomplete tasks.
2. completed_tasks.txt: Stores all completed tasks.

The script uses standard Unix commands (sort, sed, grep, nl, wc) and basic Bash programming structures. It ensures simplicity and maintainability while providing practical task management functionality.

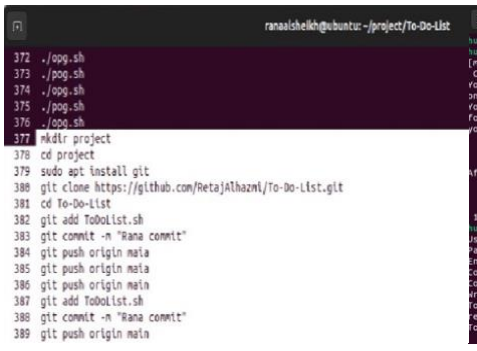
This script was written & developed using **GitHub** (a version control system that facilitates collaboration among program developers), due to its support for structured and efficient workflow in team-based software development.

Use of GitHub among team members:

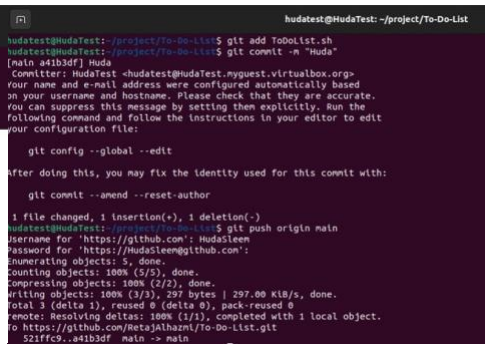
📸 Team leader **Retaj Alhazmi's** terminal screen after creating the repository and writing her code snippet



📸 Team member **AlJood Alsaleh's** terminal screen after adding her code snippet to the main branch

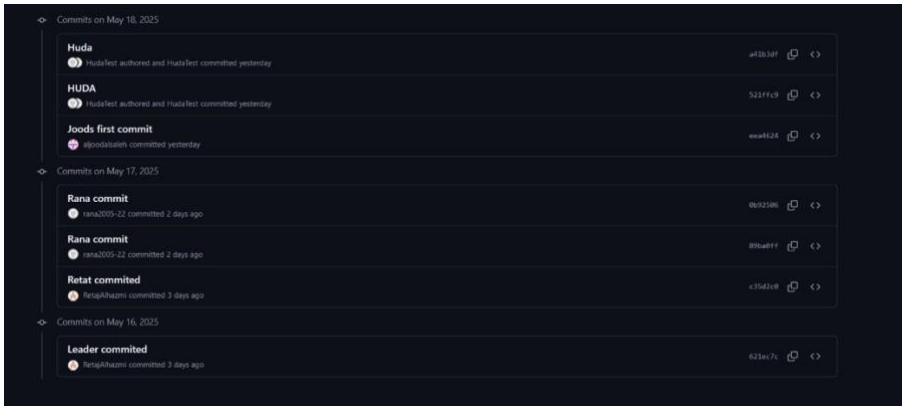


📸 Team member **Rana Alsheikh's** terminal screen after adding her code snippet to the main branch



📸 Team member **Huda Sleem's** terminal screen after adding her code snippet to the main branch

Cont..



📸 Git repository contributors & their commits

2.2 System Functionalities

1. **Add Task:** Prompts the user for a task name and due date, then appends this information to the pending_tasks.txt file. Changes are committed to Git for version tracking.
Code snippet:

```
ubuntu@ubuntu:~/project/To-Do-List$ cat ToDoList.sh
#!/bin/bash

# Task files - Retaj
PENDING_FILE="pending_tasks.txt" # Assign pending_tasks.txt to PENDING_FILE variable (incompleted tasks)
COMPLETED_FILE="completed_tasks.txt" # Assign completed_tasks.txt to COMPLETED_FILE variable (completed tasks)

# 1.Add a new task - Retaj
add_task() {
echo "Enter task name: " # Ask the user to input task name
read task_name # Take task name input from the user
echo "Enter due date (YYYY-MM-DD): " # Ask the user to input due date
read due_date # Take due date input from the user
echo "$task_name-$due_date" >> "$PENDING_FILE" # Put task name and due date variables into pending tasks file
echo "Task added!" # Confirmation message
git add "$PENDING_FILE" # Stage the updated task file for git version control
git commit -m "Added task: $task_name" # Commit the change with a message that includes the task name
}
```

2. **View Tasks:** Displays pending tasks sorted by their due dates using Unix sort.
Code snippet:

```
# 2.View tasks sorted by due date - Retaj
view_tasks() {
if [ -s "$PENDING_FILE" ]; then # Check if the pending tasks file is not empty
echo "Current pending tasks (sorted by due date): " # Display header text
sort -t '-' -k2 "$PENDING_FILE" # Sort tasks by date and print them
else
echo "No pending tasks available." # Inform the user if the file is empty
fi
}
```

3. **Complete Task:** Allows the user to select a task by number to mark as complete. The task is moved from the pending_tasks.txt file to the completed_tasks.txt file and a motivational message is displayed. Changes are committed to Git for version tracking.

Code snippet:

```
# 3.Complete a task (move from pending to complete) - Huda
complete_task() {
if [ ! -s "$PENDING_FILE" ]; then        # Check if the pending tasks file is empty or doesn't exist
echo "No tasks to complete."
return
fi

echo "Pending tasks:"                            # Display the list of pending tasks
nl -w1 -s'. ' "$PENDING_FILE"                    # Number and list each pending task
echo "Enter task number to mark as complete:"    # Prompt the user to enter a task number
read task_number                                 # Read the task number from user input

if echo "$task_number" | grep -qE '^[0-9]+$'; then   # Check if input is a valid positive integer
 completed_line=$(sed -n "${task_number}p" "$PENDING_FILE")  # Get the task line from the file

if [ -z "$completed_line" ]; then               # Check if the task number is invalid
echo "Invalid task number."
return
fi
echo "$completed_line" >> "$COMPLETED_FILE"      # Add the completed task to the completed file
sed -i "${task_number}d" "$PENDING_FILE"         # Remove the task from the pending file

messages=(                                       # Define motivational messages array
"Great job! Keep up the awesome work! 💪"
"Well done! You're one step closer to success! 🚀"
"Task completed! You're doing amazing! 👏"
"Excellent work! Stay focused and keep pushing forward! ✅"
"Proud of your progress! Keep going strong! 🌟"
"Another task down! You're unstoppable! ✨"
"You're crushing it! Keep up the momentum! 💼"
"Success is built one task at a time — and you're doing it! 🏗️"
"Impressive! Keep showing up and getting it done! 🔥"
"Fantastic! You're building great habits! 🌈"
)
random_index=$((RANDOM % ${#messages[@]}))       # Pick a random index from the messages array
echo "Task completed! ${messages[$random_index]}" # Display the selected motivational message

git add "$PENDING_FILE" "$COMPLETED_FILE"        # Stage both updated files
git commit -m "Completed task: $completed_line"  # Commit with a message about the completed task
else
echo "Invalid number."                           # Handle invalid (non-numeric) input
    fi
}
```

4. **Count Tasks:** Shows the number of pending and completed tasks.

Code snippet:

```
# 4.Show number of tasks - Jood
count_tasks() {
    pending_count=0 # Initial value for pending tasks counter
    completed_count=0 # Initial value for completed tasks counter

    if [ -f "$PENDING_FILE" ]; then # If the pending tasks file exists
        pending_count=$(wc -l < "$PENDING_FILE") # Count the number of lines in the file
    fi

    if [ -f "$COMPLETED_FILE" ]; then # If the completed tasks file exists
        completed_count=$(wc -l < "$COMPLETED_FILE") # Count the number of lines in the file
    fi

    echo "Pending tasks: $pending_count" # Print the number of pending tasks
    echo "Completed tasks: $completed_count" # Print the number of completed tasks
}
```

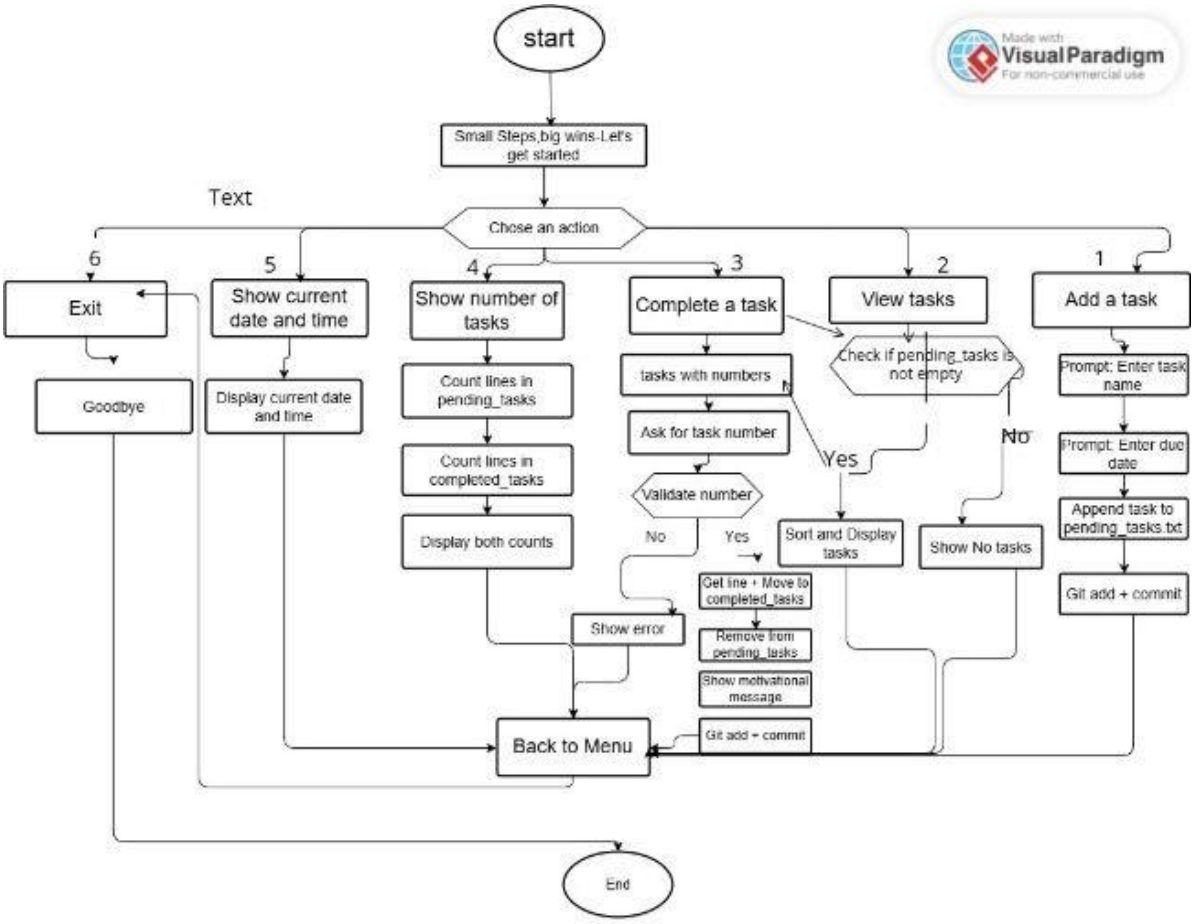
5. **Show Date and Time:** Displays the current system date and time.
Code snippet:

```
# 5.Show current date and time -Jood
show_date_time() {
echo "Current date and time: $(date)" # Print the current date and time using the shell command 'date'
}
```

6. **Main Program Loop:** Displays the main menu where the user selects the desired function.
Code snippet:

```
# Show welcome message - Rana
echo "Small steps, big wins – Let's get started."
# Main program loop - Rana
while true;
do
echo ""  # Print an empty line for spacing and better readability
echo "Choose an action:"  # Display menu prompt
echo "1) Add a task"      # Option 1: Call add_task function
echo "2) View tasks"      # Option 2: Call view_tasks function
echo "3) Complete a task" # Option 3: Call complete_task function
echo "4) Show number of tasks"   # Option 4: Call count_tasks function
echo "5) Show current date and time" # Option 5: Call show_date_time
echo "0) Exit"            # Option 0: Exit the script

read choice               # Read the user's input (menu selection)

# Use a case statement to handle different user choices
case $choice in
    1) add_task ;;        # If the user enters 1, call the add_task function
    2) view_tasks ;;      # If 2, call the view_tasks function
    3) complete_task ;;   # If 3, call the complete_task function
    4) count_tasks ;;     # If 4, call the count_tasks function
    5) show_date_time ;;  # If 5, call the show_date_time function
    0) echo "Goodbye!"; exit ;;  # If 0, print goodbye and exit the loop/script
    *) echo "Invalid choice." ;; # For any other input, show an error message
   esac
done
```

2.3 System Diagram:



3. Testing & Results

3.1 Testing Strategy & Results:

Each feature/function was tested interactively:

1. **Add Task:** Adding multiple tasks with various due dates was successful, and changes were committed to Git for version tracking. **Test results:**

```
Choose an action:
1) Add a task
2) View tasks
3) Complete a task
4) Show number of tasks
5) Show current date and time
0) Exit
1
Enter task name:
study
Enter due date (YYYY-MM-DD):
2025-5-25
Task added!
[main 3d7865a] Added task: study
 1 file changed, 1 insertion(+), 2 deletions(-)

Choose an action:
1) Add a task
2) View tasks
3) Complete a task
4) Show number of tasks
5) Show current date and time
0) Exit
1
Enter task name:
work
Enter due date (YYYY-MM-DD):
2025-5-23
Task added!
[main 6c670dc] Added task: work
 1 file changed, 1 insertion(+)
```

2. **View Tasks:** Viewing tasks listed the correct pending tasks and the tasks appeared correctly sorted. **Test results:**

```
Choose an action:
1) Add a task
2) View tasks
3) Complete a task
4) Show number of tasks
5) Show current date and time
0) Exit
2
Current pending tasks (sorted by due date):
work-2025-5-23
study-2025-5-25
```

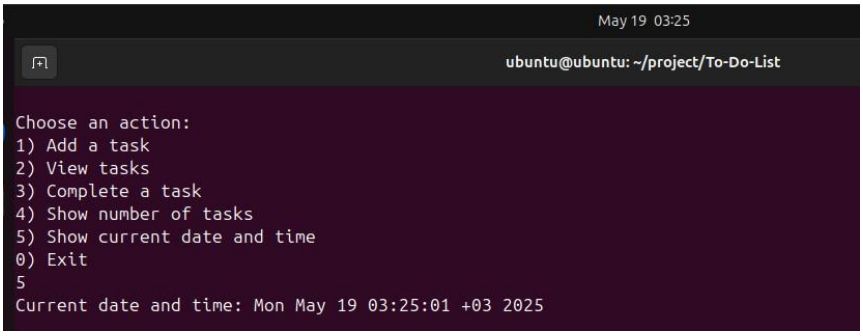
3. **Complete Task:** Completing a task removed it from pending and added it to completed, with a randomly selected motivational message shown. Changes were committed to Git for version tracking. **Test results:**

```
Choose an action:
1) Add a task
2) View tasks
3) Complete a task
4) Show number of tasks
5) Show current date and time
0) Exit
3
Pending tasks:
1. study-2025-5-25
2. work-2025-5-23
3. clean-2025-5-27
Enter task number to mark as complete:
1
Task completed! Proud of your progress! Keep going strong! 🌟
[main 4cd29cf] Completed task: study-2025-5-25
 2 files changed, 2 deletions(-)

Choose an action:
1) Add a task
2) View tasks
3) Complete a task
4) Show number of tasks
5) Show current date and time
0) Exit
3
Pending tasks:
1. work-2025-5-23
2. clean-2025-5-27
Enter task number to mark as complete:
2
Task completed! Success is built one task at a time — and you're doing it! 🏗️
[main 898dea3] Completed task: clean-2025-5-27
 2 files changed, 1 insertion(+), 1 deletion(-)
```

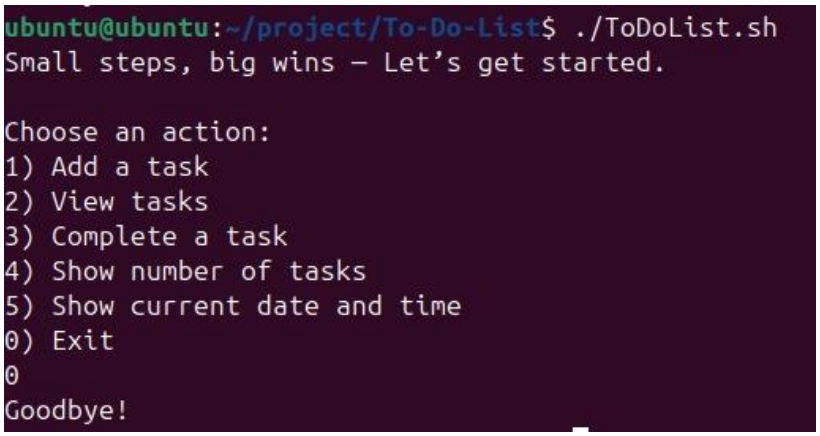
4. **Count Tasks:** The count of pending and completed tasks updated correctly. **Test results:**

```
Choose an action:
1) Add a task
2) View tasks
3) Complete a task
4) Show number of tasks
5) Show current date and time
0) Exit
4
Pending tasks: 1
Completed tasks: 2
```

5. **Show Date and Time:** The current date and time displayed accurately. **Test results:**



6. **Main Program Loop:** The user menu displayed correctly. **Test results:**



3.2 Error Handling:

Error handling was tested by entering invalid function & task numbers, with appropriate user feedback & error messages.

1. **Entering invalid function number**
   Test results:

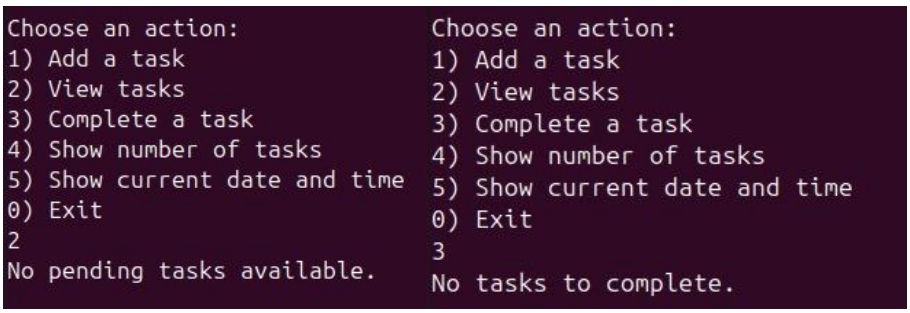
```
Choose an action:
1) Add a task
2) View tasks
3) Complete a task
4) Show number of tasks
5) Show current date and time
0) Exit
8
Invalid choice.

Choose an action:
1) Add a task
2) View tasks
3) Complete a task
4) Show number of tasks
5) Show current date and time
0) Exit
hi
Invalid choice.
```

2. **Entering invalid task number**
   Test results:

```
Choose an action:
1) Add a task
2) View tasks
3) Complete a task
4) Show number of tasks
5) Show current date and time
0) Exit
3
Pending tasks:
1. clean-2025-5-25
Enter task number to mark as complete:
2
Invalid task number.
```

```
Choose an action:
1) Add a task
2) View tasks
3) Complete a task
4) Show number of tasks
5) Show current date and time
0) Exit
3
Pending tasks:
1. clean-2025-5-25
Enter task number to mark as complete:
hi
Invalid number.
```

3. **Requesting functions with empty .txt files (no tasks)**
   Test results:



3.3 Results Analysis:

The system has proven to be reliable & error-free after functional testing confirmed that all core features perform reliably and as expected. Performance remained stable even with multiple tasks, showcasing the system's ability to scale within typical usage bounds. User interaction via the command-line interface was straightforward, and error handling was sufficient to guide users in case of incorrect inputs. Overall, the system meets its design objectives for lightweight, dependency-free task management on Unix-like systems.

4. Conclusion

4.1 Project Achievements

This project successfully delivered a basic yet effective shell-script-based task manager. It demonstrated how simple scripting and version control can be combined to create useful tools that people can apply to their daily lives. This project also benefited us immensely as students, as it provided us with lots of new information and experience regarding bash scripting & version control, which we will definitely apply to our futuristic work.

4.2 Project Limitations

Although we have successfully achieved many goals in our project, there is still much room for improvement. Especially since it's a very simple program with limited functions & abilities.
Some of the program limitations include:

- No support for editing tasks once added.
- Tasks are stored in plain text without advanced data structures.
- Program doesn't provide a graphical user interface.

4.3 Future Work

Future improvements could include adding support for task prioritization, as well as implementing search and filter functions to enhance usability with larger task lists. Integration with external tools such as calendar applications could further extend its usefulness. Additionally, improving the user interface with color-coded output or customized words of motivation would enhance the overall user experience while maintaining the script's lightweight and portable nature.

جامعة أم القرى
UMM AL-QURA UNIVERSITY