# SHELL SCRIPT LEARNING

Peter Vaughan

MAY 7, 2025
NSCC ONLINE LEARNING
E-Campus

## Contents

## Name Manager System Project

**Objective:** Build a bash script that interacts with a CSV file (names.csv) to store, view, search, and delete names. Use Git for Windows or similar to run this script.

Install Git for Windows or similar. Run the [ScriptName].sh by right-clicking the file and selecting the correct program under "Open With".

## Skills Covered

There should a range of skills covered in this assignment. While the program can be simple, the script should cover the following skills:

1. Shell Scripting Fundamentals
2. File I/O
3. User Input Handling
4. String and Pattern Matching
5. Conditional Logic and Flow Control
6. Working with CSV Data
7. File Handling
8. Basic error handling (e.g., preventing empty input or duplicate entries)

## Sample Bash Script to Save to a CSV File

How this program works:

- Initializes `sample_data.csv` with headers (`id,name,age`) if it doesn't exist.
- Prompts the user for data.
- Validates the input.
- Appends the new record to the CSV file.

```
CSV_FILE="sample_data.csv"

# Initialize the CSV file with a header if it doesn't exist
if [[ ! -f "$CSV_FILE" ]]; then
  echo "id,name,age" > "$CSV_FILE"
fi

# Function to add a new row to the CSV file
add_record() {
  read -p "Enter ID: " id
  read -p "Enter Name: " name
  read -p "Enter Age: " age

  # Check for empty inputs
  if [[ -z "$id" || -z "$name" || -z "$age" ]]; then
    echo "Error: All fields are required."
    return
  fi

  # Optional: Check if ID already exists
  if tail -n +2 "$CSV_FILE" | grep -q "^$id,"; then
    echo "Error: ID '$id' already exists."
    return
  fi

  # Append to the CSV
  echo "$id,$name,$age" >> "$CSV_FILE"
  echo "Record added successfully."
}

# Call the function
add_record
```

## Sample Script: Edit a Record in a CSV File (Based on ID)

Editing data in a .csv file using a Bash script involves these steps:

1. **Read the current data**.
2. **Find the row you want to edit** (based on some unique field like ID or full name).
3. **Modify that row**.
4. **Write the updated content back to the file.**
5. **ID is not need for assignment. This is an example for conceptual purposes.**

```
#!/bin/bash

CSV_FILE="sample_data.csv"

# Ensure file exists
if [[ ! -f "$CSV_FILE" ]]; then
   echo "id,name,age" > "$CSV_FILE"
fi

edit_record() {
   read -p "Enter ID to edit: " edit_id

   # Check if ID exists
   if ! grep -q "^$edit_id," "$CSV_FILE"; then
      echo "Record with ID '$edit_id' not found."
      return
   fi

   # Prompt for new details
   read -p "Enter new name: " new_name
   read -p "Enter new age: " new_age

   # Create a temp file and write updated contents
   {
      # Read file line-by-line
      while IFS=, read -r id name age; do
         if [[ "$id" == "$edit_id" ]]; then
            echo "$id,$new_name,$new_age"  # modified line
         else
            echo "$id,$name,$age"        # unchanged line
      fi
    done < "$CSV_FILE"
  } > temp.csv

  mv temp.csv "$CSV_FILE"
  echo "Record with ID '$edit_id' updated."
}

# Call the function
edit_record
```

## Submission Expectation

- GitHub Link with the following:
  - Code
  - README.md

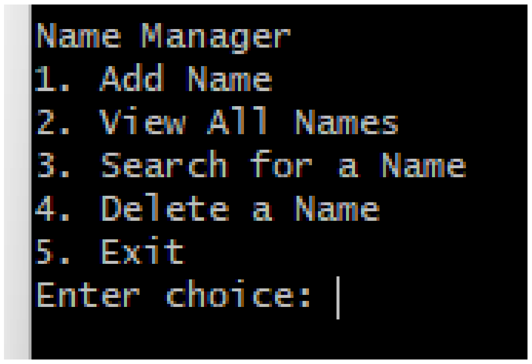
- README.md (on GitHub):
  - Project title and description
  - Setup instructions
  - How to run/demo the project

- Photos
  - Screenshots of your application in action
  - Optional: photos of setup/hardware (if applicable)

- Documentation (typically a PDF or DOCX):
  - Project overview and goals
  - System architecture / design diagrams
  - Implementation details (including technologies used)
  - The IDE and how to run the code
  - Testing strategy and results – including testing units
  - Any challenges faced and how they were solved
  - Future work or improvements

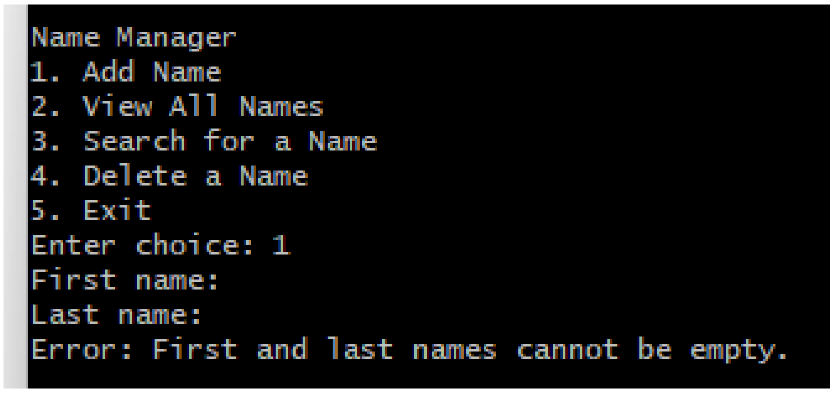
## Output

Install Git for Windows or similar – this is the IDE. Run the name_manager.sh by right-clicking the file and selecting the correct program under “Open With”. When the script is running, a csv file is created called names.csv that stores the names.



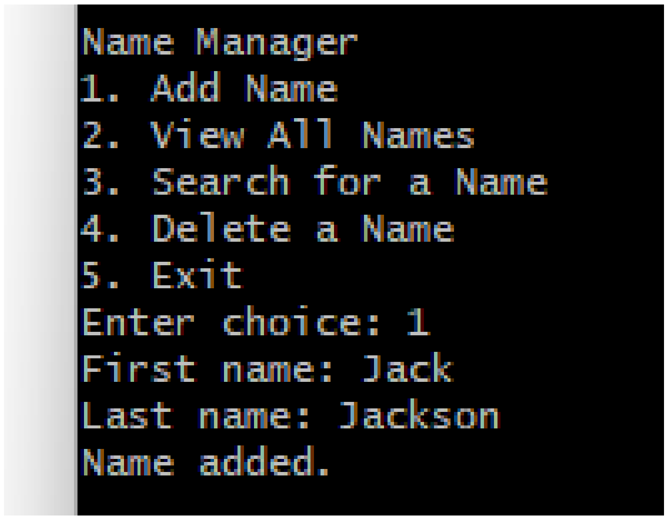
```
Name Manager
1. Add Name
2. View All Names
3. Search for a Name
4. Delete a Name
5. Exit
Enter choice: |
```

Users can add names to the csv file. There is error-handling such as not being allowed to enter blank names. Error handling could be expanded for more features such as excluding numbers and some special characters.



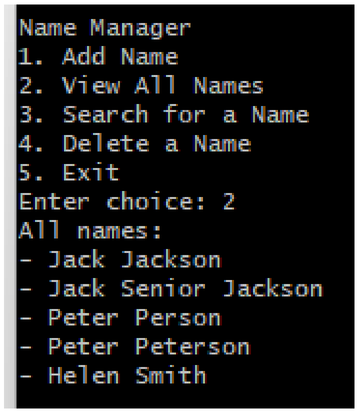
```
Name Manager
1. Add Name
2. View All Names
3. Search for a Name
4. Delete a Name
5. Exit
Enter choice: 1
First name:
Last name:
Error: First and last names cannot be empty.
```

I was having issues connecting the csv file sometimes. I would spend more time investigating this if I had time. If it was my choice, I would redesign this and use a different database storage system.



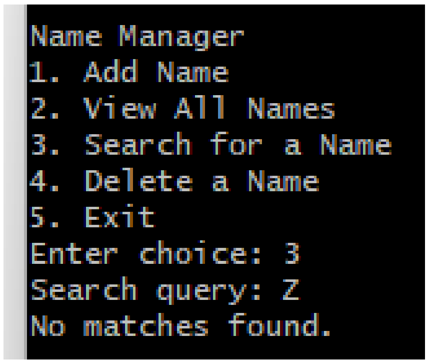
```
Name Manager
1. Add Name
2. View All Names
3. Search for a Name
4. Delete a Name
5. Exit
Enter choice: 1
First name: Jack
Last name: Jackson
Name added.
```

Users can view all the names the user info. This is a small sample size for testing purposes. In school, we did not learn too much about csv files as they are not used as much anymore. There are more secure ways to store data. However, there is many ways to store info. This is the simple approach and it is not the focus of the assignment. I would have to d research on this before suggesting a database.

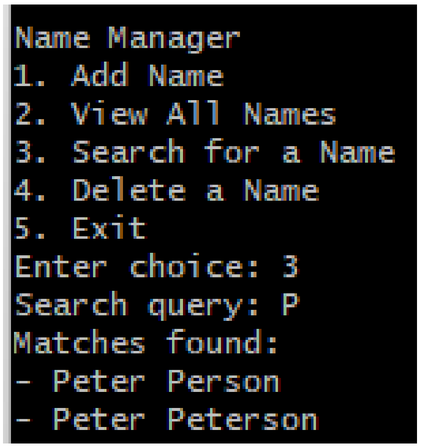


```
Name Manager
1. Add Name
2. View All Names
3. Search for a Name
4. Delete a Name
5. Exit
Enter choice: 2
All names:
- Jack Jackson
- Jack Senior Jackson
- Peter Person
- Peter Peterson
- Helen Smith
```

This is the name search with some basic error-handling. This is a basic concept as per the assignment. This could be more comprehensive search. Users can have a broad search such as this and an exact search as well. There are other more comprehensive search functions. However, this is per the assignment.



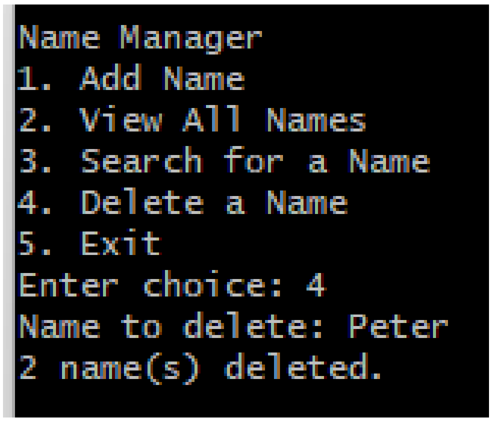
```
Name Manager
1. Add Name
2. View All Names
3. Search for a Name
4. Delete a Name
5. Exit
Enter choice: 3
Search query: Z
No matches found.
```



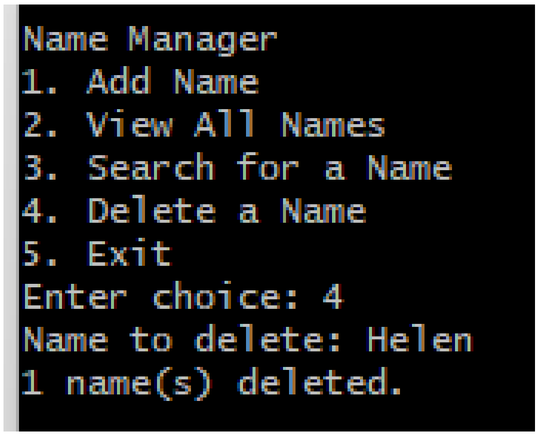
```
Name Manager
1. Add Name
2. View All Names
3. Search for a Name
4. Delete a Name
5. Exit
Enter choice: 3
Search query: P
Matches found:
- Peter Person
- Peter Peterson
```

Time complexity could be a discussion with bigger scripts. This discussion could be part of the database discussion, too. Storing to certain databases could slow down the system lots or have other unintended issues.

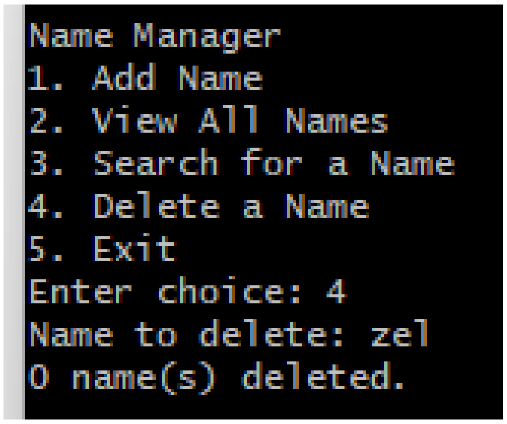
Deleting is a very broad concept. If a character or string of characters is including in the name in the csv file, then the name will be deleted. For this assignment, this is acceptable. However, in practical use, users would want more options on how to delete the info. Should we limit to a maximum of one deletion? If we delete by a unique ID, this will help minimize delete the wrong person. We can include many options or approaches here.



```
Name Manager
1. Add Name
2. View All Names
3. Search for a Name
4. Delete a Name
5. Exit
Enter choice: 4
Name to delete: Peter
2 name(s) deleted.
```



```
Name Manager
1. Add Name
2. View All Names
3. Search for a Name
4. Delete a Name
5. Exit
Enter choice: 4
Name to delete: Helen
1 name(s) deleted.
```



```
Name Manager
1. Add Name
2. View All Names
3. Search for a Name
4. Delete a Name
5. Exit
Enter choice: 4
Name to delete: zel
0 name(s) deleted.
```

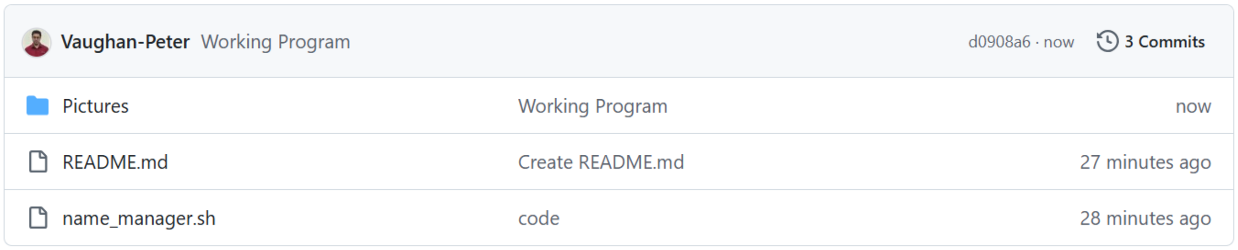
## Submission

GitHub Link: https://github.com/Vaughan-Peter/Shell-SelfLearning/tree/main

ReadMe File: Complete in the GitHub Repo

Documentation: Done

Pictures Folder



Vaughan-Peter Working Program d0908a6 · now 3 Commits

| | | |
|---|---|---|
| Pictures | Working Program | now |
| README.md | Create README.md | 27 minutes ago |
| name_manager.sh | code | 28 minutes ago |