CS 108 Project - Bash Grader

Varsha Seemala

April 28 , 2024

Contents

1 Objective

The objective of this project is to create a CSV file manager and interpreter using a bash script, enabling users to interact with CSV files containing student examination records.

2 Introduction

CSV files, short for comma-separated values, organize data in a structured format where values are delimited by commas.The directory that we will be working on will have files with data of students for a particular exam and the bash script **submission.sh**. The format of the data is as follows: <Roll_Number,Name,Marks>. Every csv file will have a heading and then rows of data for various students.The directory will also contain **statistics.py** and **examgraph.py** for customization.

3 Basic Idea of the Bash Script

To use the project, users can run the **submission.sh** script with various commands and arguments. The script supports the following commands:

- **combine:** Combines data from multiple CSV files into a single main.csv file. Each CSV file corresponds to an exam, and the main CSV file contains columns for all exams. Students' data is merged into the main file, handling cases where students are absent in certain exams.
- **upload:** Imports new CSV files into the project directory for inclusion in the combined main.csv file.
- **total:** adds a new column named 'total' to the main.csv file, calculating the total marks for each student across all exams.
- **update:** Updates student marks in the main.csv file based on user input, ensuring consistency across individual exam files.
- **A version of Git:** The bash script provides a mini version of git. It stores the version of the working directory in a remote repository provided by the user.
  - **git_init:** Initializes a remote directory for version control, allowing users to manage versions of the project files.
  - **git_commit:** Commits the current version of files to the remote directory, generating a unique hash value for each commit and storing commit messages in a .git_log file.
  - **git_checkout:** Reverts the project directory to a specific commit, identified by either a full hash value or a prefix.
  - **git_branch:** creates a branch at a specific commit in the remote repository and stores the current version of the working repository at that branch.
  - **git_checkout_branch:** reverts the current directory to the version saved in a specific branch.
  - **git_log:** shows all the commits made and their respective commit messages.
- **stats:** This function provides statistics on student performance in each exam and offers an option to display graphical representations of exam performance.
- **command_help:** This function displays the available commands along with their descriptions.

The script is structured using a case statement (case ... esac) to execute the appropriate function based on the command provided as the first argument when running the script. Each function handles specific tasks related to CSV file management, version control, and updating marks.

4 Working of the Bash Script

The script incorporates an efficient renaming system that ensures unique filenames for the moved files. The renaming system follows the following rules:

1. Initialization Check: It checks for the existence of a flag file named .first_run. If it doesn't exist, it provides a welcome message and usage instructions for the script's functionalities.This means that the message will be provided only during the first run of the script.

2. Command help function : The command_help function in the Bash script serves the purpose of providing users with a summary of the available commands along with their descriptions.Any time the user wants to check the functionalities that this script can offer,he can use this function.
   Usage:

   ```
   bash submission.sh command_help
   ```

3. Combine function: The combine function in the provided Bash script merges multiple CSV files into a single main.csv file. It first checks if main.csv already exists and creates it if not. Then, it updates the header with the names of the CSV files being combined. After that, it collects unique roll numbers and names from all CSV files and appends them to main.csv. Finally, it updates the marks of each student in main.csv based on the data in the individual CSV files, handling absent students appropriately.If the total column was already present in main.csv and data of new csv files was added to the main, the script recalculates the total marks for each student and updates the total column accordingly.
   Usage:

   ```
   bash submission.sh combine
   ```

4. Total function: The total function in the provided Bash script calculates the total marks for each student in the main.csv file. Here's a brief overview of its functionality:
   Initialization: The function initializes necessary variables and sets up the field separator (FS) and output field separator (OFS) for parsing CSV files.
   Processing Rows: It processes each row in main.csv, excluding the header. For each row:
   If it's the header row, it checks if the "total" column is present. If not, it adds the "total" column.
   If it's a data row, it calculates the total marks by summing up all marks for that student, excluding "a" for absent marks.
   Updating File: It updates main.csv with the new total marks column. If the "total" column was already present, it updates the existing column; otherwise, it adds a new column.
   Usage:

   ```
   bash submission.sh total
   ```

5. The upload function in the provided Bash script is responsible for copying a specified file or directory to the current repository directory. Argument Check: It checks if the user has provided a file or directory path as an argument.
   Copy Operation: If a valid path is provided, the function copies the specified file or directory to the current directory where the script is located.
   Usage:

```
bash submission.sh upload
```

6. Git functionalities implemented in the bash script.

   - git_init: Initializes a remote directory for version control.
     Creates a remote directory if it doesn't exist already.
     Stores the path of the remote directory in a .git_remote file.
     Usage:

     ```
     bash submission.sh git_init <address of the remote repo>
     ```

   - git_commit: Saves the current version of all files in the repository to the remote directory.
     Generates a random 16-digit hash value as the commit ID.
     Appends the commit ID and commit message to a .git_log file in the remote directory.
     Prints the names of modified files since the last commit.
     Usage:

     ```
     bash submission.sh git_commit -m "commit_msg"
     ```

   - git_checkout:Reverts the current directory back to a specific commit.
     Accepts either a commit message or a commit hash value as input.
     Copies all files from the specified commit's directory to the current directory.
     Usage:

     ```
     bash submission.sh git_checkout <hash>
     bash submission.sh git_checkout -m "commit_msg"
     ```

   - git_branch: Creates a branch (hidden directory) at a specific commit ID in the remote repository.
     Requires the commit ID and branch name as arguments.
     Creates a directory named with the branch name inside the specified commit's directory.
     Usage:

     ```
     bash submission.sh git_branch <hash> <branch_name>
     ```

   - git_checkout_branch: Reverts the current directory to the version saved in a specific branch.
     Requires the commit ID and branch name as arguments.
     Copies all files from the specified branch's directory to the current directory.
     Usage:

     ```
     bash submission.sh git_checkout_branch <hash> <branch_name>
     ```

   - git_log : The git_log function in the provided Bash script displays a log of all commits made to the remote repository

     ```
     bash submission.sh git_log
     ```

7. The update function in the provided Bash script is responsible for updating the marks of a particular student in the main.csv file and other related CSV files.

   - Input Roll Number: Prompts the user to input the roll number of the student whose marks need to be updated.
   - Search for Student: Searches for the student with the given roll number in the main.csv file.
   - Display Current Marks: Displays the current marks of the student retrieved from main.csv.
   - Prompt for Update: Prompts the user whether they want to update the marks for the student.

- Update Marks: If the user chooses to update the marks, the function prompts the user to input the updated marks for each exam. It then updates the marks for the student in main.csv as well as in other related CSV files.
- Completion Message: Displays a message confirming the successful update of marks if the user chooses to update them. Otherwise, it indicates that no updates were made.
- Usage:

  ```
  bash submission.sh upload
  ```

8. The stats function in the provided Bash script is responsible for generating statistics based on the data in the main.csv file.

   - Check for main.csv: Checks if the main.csv file exists. If not found, it prompts the user to combine and total first.
   - Display Statistics: Generates and displays statistics of the exams. It includes satistical measures like mean, median, and standard deviation.
   - Graphical Representation: Optionally, it prompts the user if they want to see the graphical representation of student performance in each exam.
   - Prompt for Update:Graph Generation: If the user chooses to see the graphical representation, it generates a graphical representation of student performance in the specified exam using a Python script (examgraph.py).
   - Completion: After displaying the statistics or generating graphs, it completes the function execution.
   - Usage:

     ```
     bash submission.sh stats
     ```

5 Error Handling

Here I discuss the error handling mechanisms implemented in the script to ensure proper usage and prevent any potential issues. The script includes the following error checks:

- Ensuring a remote repository is initialized before performing command like git_commit, git_checkout etc

  ```
  if [ ! -f .git_remote ]; then
        echo "Error: Remote directory not initialized. Please run 'git_init' first."
        return 1
    fi
  ```

- creation of remote repository if not already present and checking for invalid number of arguments in git_init

  ```
  if [ ! -d "$remote_dir" ]; then
      echo "Creating a remote directory"
      mkdir $remote_dir
  fi
  if [ $# -ne 1 ]; then
      echo "Usage: bash $0 git_init <remote_directory>"
      return 1
  fi
  ```

- Ensuring a valid rollno is given as an input when update command is run.

```
read -p "Enter student's roll number: " roll_number

# Find the student in main.csv
student=$(grep "$roll_number" main.csv)

# Check if student exists
if [ -z "$student" ]; then
    echo "Error: Student with roll number $roll_number not found."
    return
fi
```

- Ensuring that main.csv file exists in the directory when stats command is run.

```
if [[ ! -f main.csv ]]; then
    echo "main.csv not found.Please do combine and total first."
    return 1
fi
```

- Ensuring proper arguments for git_commit

```
if [ $# -lt 2 ]; then
    echo "Usage: bash $0 git_commit -m <commit_message>"
    return 1
fi
```

- Checking whether the argument(complete hash or prefix of the hash) provided by the user for git_checkout matches any previous commit id or previous commit message. If it does not match the function gives an error message and returns.

```
matching_commits=$(grep -oE "^$commit_ref.*" "$remote_dir/.git_log" | wc -l)
    if [ "$matching_commits" -ne 1 ]; then
        echo "Error: Ambiguous commit reference. Please provide a unique commit message o
        return 1
    fi

if [[ ${array[0]} == -m ]]; then
        # Extract commit message from the arguments
        commit_msg=$(echo "$args" | sed 's/-m \(.*\)/\1/')

        # Check if any commit has the provided commit message
        matching_commitmsg=$(grep -oE ".*$commit_msg$" "$remote_dir/.git_log" | wc -l)
        num=$((matching_commitmsg))
        if [ "$num" == 0 ]; then
            echo "Error: No matching commit message."
            return 1
        fi
        if [[ $num -gt 1 ]]; then
            echo "More than one commit has the same commit message. Please proceed throug
            return 1
        fi
```

- Checking whether proper arguments are provided by the user for git_branch and git_checkout_branch. Since both have commit_id as one of the argument it checks whether it is a valid commit_id.For git_checkout_branch branchname is provided as a second argument. The Script

checks whether the branch exists at that particular commit. If it does not it returns an error message.

```
matching_commits=$(grep -oE "^$commit_ref.*" "$remote_dir/.git_log" | wc -l)
    if [ "$matching_commits" -ne 1 ]; then
        echo "Error: Ambiguous commit reference. Please provide a unique commit message c
        return 1
    fi

if [[ ${array[0]} == -m ]]; then
        # Extract commit message from the arguments
        commit_msg=$(echo "$args" | sed 's/-m \(.*\)/\1/')

        # Check if any commit has the provided commit message
        matching_commitmsg=$(grep -oE ".*$commit_msg$" "$remote_dir/.git_log" | wc -l)
        num=$((matching_commitmsg))
        if [ "$num" == 0 ]; then
            echo "Error: No matching commit message."
            return 1
        fi
        if [[ $num -gt 1 ]]; then
            echo "More than one commit has the same commit message. Please proceed throug
            return 1
        fi

branch_name=${array[1]}
 matching_names=$(grep -oE "$branch_name" "$remote_dir/.git_branches" | wc -l)
 if [ "$matching_names" -ne 1 ]; then
        echo "Branch not found. Please check branch name"
        return 1
    fi
```

The script provides friendly error messages to guide the user in resolving any issues and ensuring a seamless experience.

6 Customization

This section highlights the customization options available in the script.

- **git_branch:** introduced a command called git_branch to create a branch at a particular commit.To use the command the user should first checkout to that particular commit_id , make changes in the working directory and run the command git_branch .This would create a branch and would help the user to save temporary changes that they do not want in the commit history.
- **git_checkout_branch:** This command helps to revert the working directory to the version of a branch at a particular commit_id.The user can then work on the changes that he has made at that commit.
- **stats:** The stats function provides valuable insights into student performance by computing statistical measures and optionally visualizing the data. It enhances the utility of the grading script by offering a deeper analysis of exam results. Additionally, its interactive nature allows users to choose whether they want to view graphical representations, making it more user-friendly.For the statistics and graphical representation python libraries pandas,numpy and matplotlib are used.

- **User friendly:** Introduced a function called command_help to give the user a gist of the functionalities of the script without having to read the entire bash script.The command git_log prints the commit ids and their messages onto the terminal so that the user can refer to the list when using git_checkout etc.Friendly real time input messages while using command like update and stats.

7 Conclusion

In summary, this Bash script offers a robust solution for handling CSV files. It integrates a version tracking system similar to Git but without directly relying on Git itself. These Git-like features ensure that all file versions are preserved securely, without having to worry about unintended modifications. With its diverse functionality and customizations, the script streamlines CSV file management, interpretation, and improves workflow efficiency.

8 Demonstration

https://screenrec.com/share/XHl49sFM0T