Project: Jeremy's Version Control (JVC)

Software Requirements Specifications

Software Engineer/QA/Functional Analyst: Jeremy Duong

Description: JVC is a version control system that works somewhat like Git but with a much simpler workflow. It allows the user to track changes to a working directory, save changes to the working directory in the form of a "version," and revert the working directory to a previous version.

I. Introduction:

1. Purpose:

The main purpose of the project is for me to gain a better understanding of git, the version control system being used by millions of software developers around the world. My goal is to understand as much as possible the inner working of git by designing and building my own version control system in a similar manner to that by which git was built.

2. Scope:

Git is certainly a reliable version control system (VCS) already used by many. However, I have noticed a common trend that SE and CS students usually get really confused the first time they use git. From my observation, I've noticed that the reason for this confusion is because of the many different commands that git has, including "pull", and "push." These commands are used to interact with a remote repository, not just to keep track of changes, which is the main functionality of a version control system. It often takes multiple semesters to even begin to understand what git actually is and to use it remotely efficiently.

Thus, even though this is my senior project, my vision and hope for the project is that it will become a simple and easy-to-use version control system that will help at least new SE and CS students familiarize themselves with the concept of version control before actually using git.

3. Overview:

As mentioned before, jvc will be a simplified version of git. It will be built using very similar data structures and data storage methods. The user can keep track of changes to the repository using "jvc status", save changes using "jvc save," look at past saves (versions) using "jvc history," and revert to earlier changes using "jvc revert <version-index>."

4. User Profiles:

My client for the project is brother Matt Manley who teaches the lower-level programming classes (CSE 110, CSE 111, CSE 210, etc…) on campus. The target users of the project are new CS and SE students who are taking these low-level classes, who have never used a VCS or git before. Having a simple design and workflow, JVC is intended to make it easier for the students to understand what a VCS is and learning how to use one quickly so that when they start using git, they will have a much easier time understanding the advanced functionalities that git provides.

5. Workflow:

There are 5 main commands corresponding to the 5 functionalities that JVC has: **jvc init**, **jvc status**, **jvc save**, **jvc history**, and **jvc revert**. The project will be accomplished by implementing these 5 main commands one by one. Each one will then be tested while being developed and at then end when it is complete.
The 3 commands “jvc init,” “jvc status,” and “jvc save” will be developed in parallel because they are strongly related in terms of how data storage works.
Once all 5 commands are implemented and tested, the project will be ready for demonstration.

6. Standards:

Coding:

The project will follow the coding standards taught in CS124 and CS165:

- No line of code has more than 80 characters
- Block comments are used through out the program to describe every function

Design:

In terms of user interface, the program will be used in the terminal like git.

Learning Model

The BYU-I Learning Model:

- Prepare
- Teach one another
- Ponder and Prove

Architecture

As of the project’s architecture, keeping in mind the need of CS210 students to learn object-oriented concepts, JVC will be designed to be very object-oriented. This means that each of the 5 functionalities will be abstracted into classes. Helper algorithms will also be abstracted into their own classes such as HashingUnit, BlobCreator, TreeCreator,…

The JVC database will be created inside the designated repository and used to store files and directories contents.

Quality Assurance Characteristics and Metrics

Test repository will be created to test the 5 functionalities.

- **jvc init** is acceptable if it creates a ".jvc" folder with the appropriate starting files
- **jvc status** is acceptable if it can pick up all the changes made after the last save
- **jvc save** is acceptable if it can save all the changes by creating the appropriate TREES and BLOBS.
- **jvc history** is acceptable if it can show all the versions that have been saved for the current repository.
- **jvc revert** is acceptable if it can revert all files to the states that they were in after the specified version, delete new files created after the specified version, and all directories' trees are implemented correctly.

Legal & Security Risks.

As part of fulfilling its functionalities, the program will be constantly reading files content, writing new files, and modifying and deleting existing files. Because of this, there is a risk of the program deleting or corrupting important files. Caution will be taken to ensure this does happen and that the creating, modifying, and deleting of files happen only within the current working directory and only when the current directory is indeed a jvc repository (specified by the existence of the ".jvc" folder).

II. Requirements:

A. Main Requirements

1. **jvc init:**
    a. When the command is executed inside a directory, it shall create a folder called ".jvc" which contains the jvc database. The jvc database (the ".jvc" folder) shall initially contain the following:
        - A folder called "head", which contains a file called "master." This file's purpose is to store the index of the latest save made by the user on the master branch. (Note: Branching is a stretch requirement. However, if branching is implemented, there shall be more files in the "head" folder, each named after a branch and contains the index of the latest version of that branch).
        - A folder called "idxSup" (short for index supplier), which contains 2 files: "version" and "tree." The "version" file contains the next usable index to refer to a specific version, and the "tree" file contains the next usable index to refer to a specific tree.

b. If the ".jvc folder" already exists, the command shall display a message "Error: Already a repository" and then do nothing else.

2. **jvc status:**
   a. When the command is executed inside a directory, it shall show all the files within the working repository that have been changed, newly created, or deleted since the last execution of **jvc save.**
   b. If there are no changed, newly created, or deleted files, show the text:

      "No changed, newly created, or deleted files found."

   c. If the command is executed but the ".jvc" folder does not exist, the program shall display the message "Error: Not a jvc repository", then do nothing else.

3. **jvc save:**
   a. When the command is executed, the program shall create new blobs (file content) for all files and create new trees (directories) for all folders that would show up when executing **jvc status.** It shall also create a new version object named by a unique version index. This version object shall contain the index of the new tree representing the root directory, and the save message of that version.
   b. The trees (representing directories) and blobs (representing file contents) created shall be sufficient to recreate the whole working repository if the **jvc revert** command is executed.
   c. Once this is executed, if **jvc save** is executed again, it shall show the message "No unsaved changes found."
   d. The user can include a note for the save simply by including the message after typing "jvc save." For example, for the first save, the user can type:

      ```
      jvc save "saved my initial version"
      ```

   e. If a message is not provided, the program shall use the following as the default message:

      ```
      "Saved version index <version-index>"
      ```

   f. If the command is executed but the ".jvc" folder does not exist, the program shall display the message "Error: Not a jvc repository", then do nothing else.

4. **jvc history:**
   a. When the command is executed, the program shall show a list of all versions saved for the current repository, together with their messages.

b. If the command is executed but the ".jvc" folder does not exist, the program shall display the message "Error: Not a jvc repository", then do nothing else.

5. **jvc revert <version-index>:**
    a. To execute this command, a version index must be provided. If a version index is not provided, the program shall display a message:

    ```
    Usage: jvc revert <version-index>. Use 'jvc history' to look up version indices.
    ```

    b. If a valid version index is provided with the command, the program shall trace the version object which points to the tree object of the root directory, then recursively recreate all files and folders and their states at the time the save happens for that version.
    c. If the version index provided is not found as one of the version indices stored in the jvc database, the program shall display the message

    ```
    Error: Version Not Found! Use 'jvc history' to look up version indices.
    ```

    d. If the command is executed but the ".jvc" folder does not exist, the program shall display the message "Error: Not a jvc repository", then do nothing else.

B. Stretch Requirements

1. jvc branch
    a. This command is used to branch off from the current branch. It should create a new file inside ".jvc/head" called after the branch name provided to keep track of the index of the most recent save for this branch.

    b. More flags and options can be provided to see the current branch, checkout a different branch, or create a new branch.

2. jvc merge
    a. This command should allow the merging of the current branch onto master.
    b. If a conflict happens, the file created will contain the content of both versions. The program will then ask the user to resolve the conflicts in that file and make a new save.

III. Design Overview:

1. Workflow:

a. Before doing anything, the user will need to initialize a folder as a repository using the command **jvc init.**
b. At any point after executing **jvc init,** the user can execute the command **jvc status** to see what changes to the repository has not been saved to a version.
c. The user can then make any change to the repository that he desires. This includes adding new files (add), modifying existing files (mod), or deleting existing files (del).
d. When ever the user is satisfied with the state of the directory, or at any point during the process where there are changes to be saved, the user can execute the command **jvc save** to save all changes made since the last save to a new version.
e. From this point on any time "add," "mod," or "del" happens, the user can execute **jvc save** again to create a new version.

2. Resources:

a. The internet: A lot of google search will be conducted to figure out how git works as the basis of the project, as well as to figure out certain important C++ syntaxes.
b. My PC and Laptop: What I need to write code.
c. Git and Github: To store any progress I made on the project…
d. Appropriate C++ compiler: The compiler needed for the project is C++17 and newer. Any compiler before that will not work since the library used to manage directories will not be recognized.

3. Data at rest:

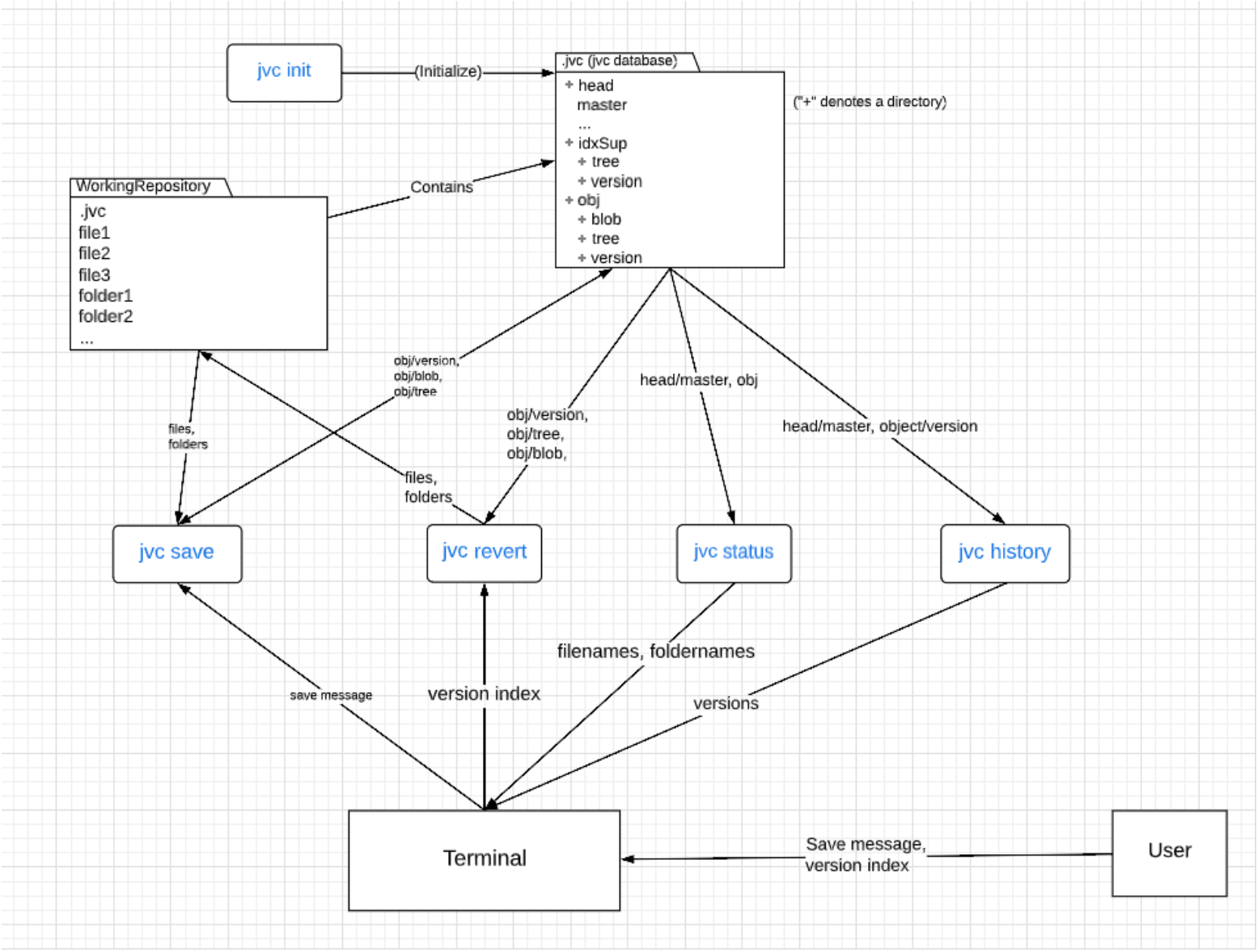
All data about file contents, sub directories, versions, branches, etc… important to the managing of the jvc repository and the functioning of jvc on such repository will be stored inside the ".jvc" folder created when executing the **jvc init** command.

4. Data on the wire:

According to the main requirements established, it is not necessary for data of a working repository to be sent outside of itself.
The only time data is picked up from the ".jvc" database and processed is when one of the main 5 commands mentioned previously are executed.

5. Data state:

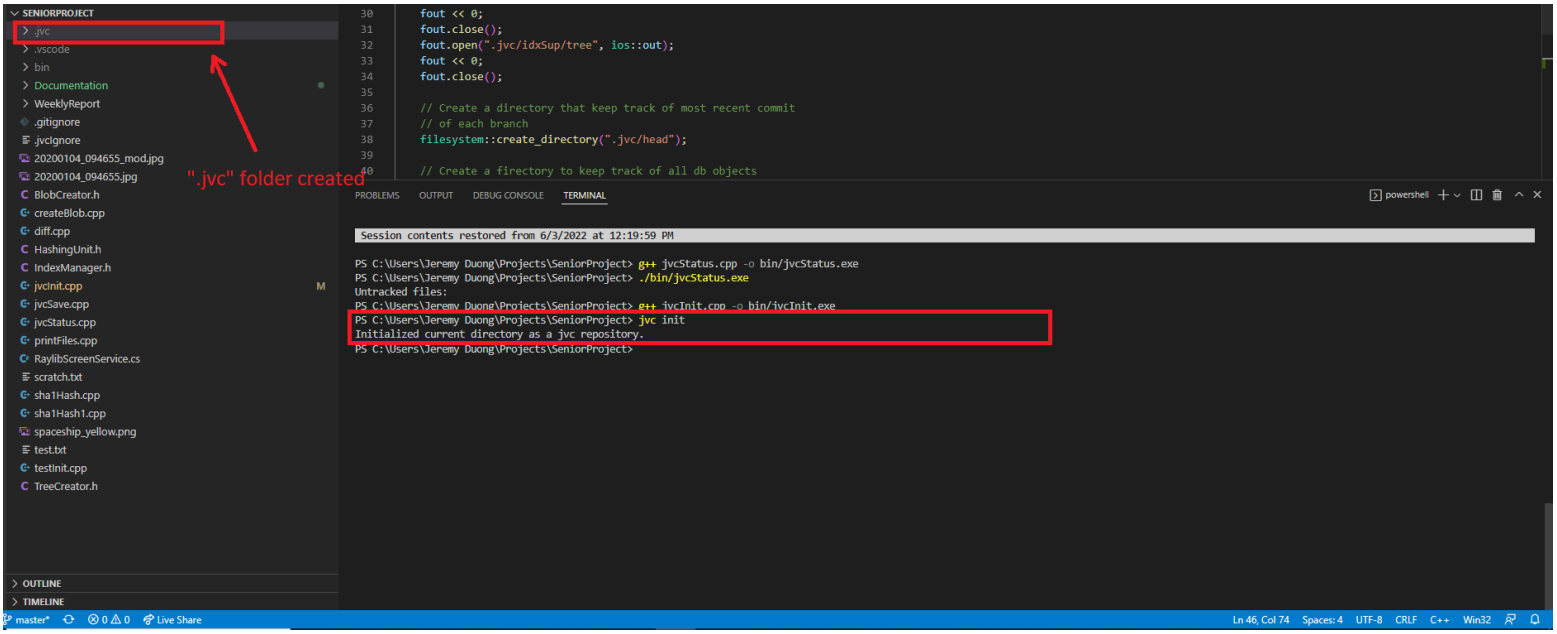
Data shall flow between the working directory, the 5 main functionalities, the .jvc database, and the user according to the following flowchart (The 5 main functionalities are colored Blue):



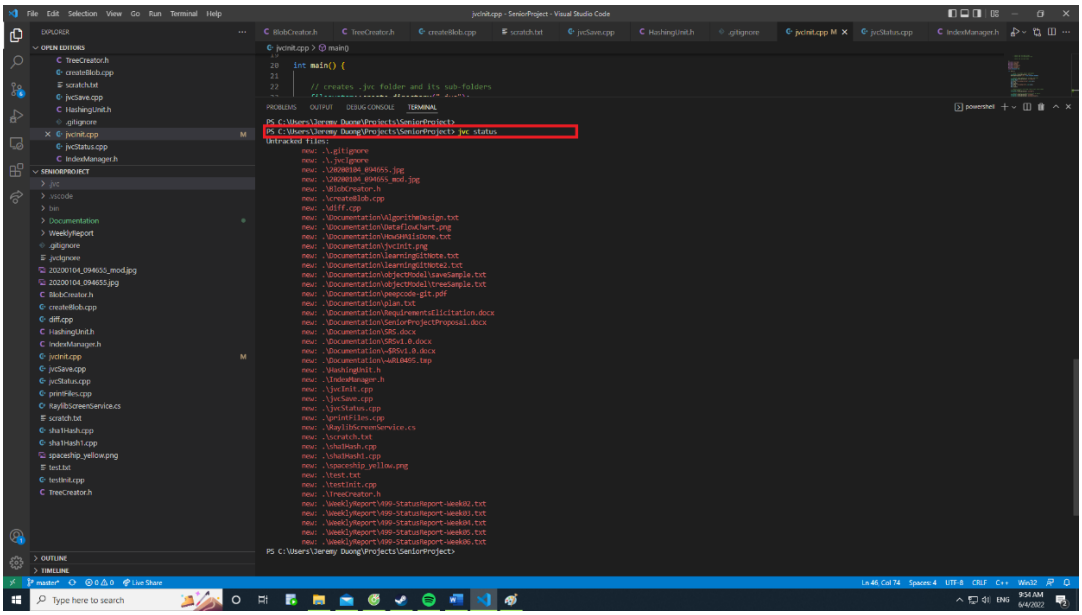
6. HMI (Human Machine Interface):

The user will in interact with the working repository through jvc, using the terminal. The following are pictures depict the user's experience when running the various jvc commands:

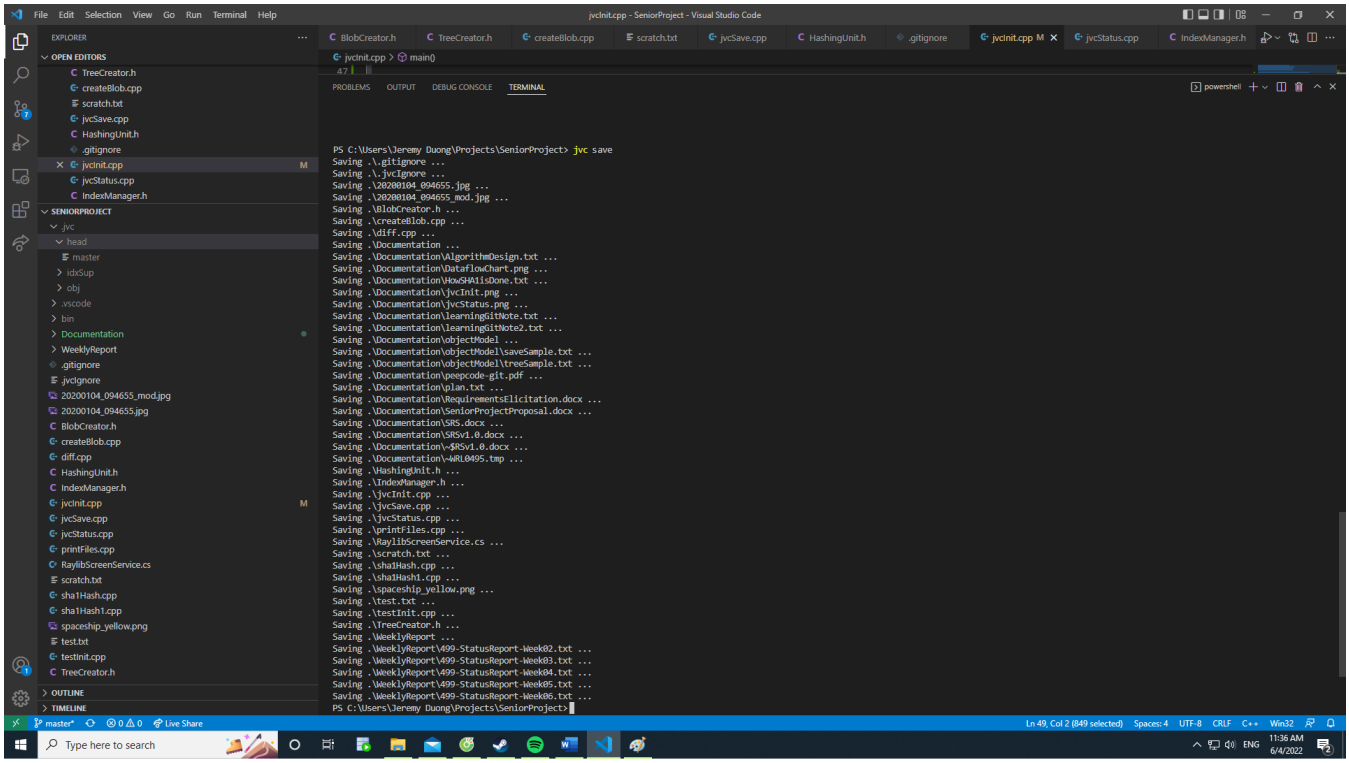
a. **jvc init:**



b. **jvc status:**



c. **jvc save:**



d. **jvc history:**

```
PS C:\Users\Jeremy Duong\Projects\SeniorProject> jvc history
Version <5>
        Fixed a bug with user interface
Version <4>
        Implemented database connection
Version <3>
        Deleted a feature
…
Press Enter to see more. Enter ‘q’ to quit: |

PS C:\Users\Jeremy Duong\Projects\SeniorProject>
PS C:\Users\Jeremy Duong\Projects\SeniorProject>
```

e. **jvc revert**:

```
PS C:\Users\Jeremy Duong\Projects\SeniorProject> jvc revert 3
Successfully reverted to version <3>:
        Deleted a feature
PS C:\Users\Jeremy Duong\Projects\SeniorProject>
PS C:\Users\Jeremy Duong\Projects\SeniorProject>
```

IV. Verifications:

1. Demonstrations:

a. **jvc init**:
   - Create a new folder and designate it as the working repository
   - Enter the folder using the terminal
   - Run the command **jvc init** in the terminal
   - **Verify:** Confirm that the ".jvc" folder is created with all the appropriate starter files and folders mentioned in Requirements.

b. **jvc status** and **jvc save**:
   - Add a file called "firstText.txt" in the repository created in the demo for jvc init. Put the content "This is the first text content" in the file created
   - Run the command **jvc status** in said repository
   - **Verify:** "firstText.txt" shows up as a new file
   - Add another file called "secondText.txt" in the repository. Put the content "This is the content for the second text" in this file
   - Run the command **jvc status** in the repository again
   - **Verify:** both "firstText.txt" and "secondText.txt" show up as new files
   - Run **jvc save** with the message:

     "Initial version. Adding firstText.txt and secondText.txt"

   - Run **jvc status** again
   - **Verify:** The program displays the text: "No modified, newly created, or deleted file found"
   - **Verify:** There is a new file under ".jvc/obj/version" called "0", in which there is a pointer to the tree object which contain pointers to all blobs and other trees inside the repository. The tree reflects the exact state of the repository at the time that save happens.

c. **jvc status** and **jvc save** (2nd demo)**:**
   - Continue from the previous demo
   - Edit the file "firstText.txt" and give it the new content: "This is the new content of the first txt"
   - Run **jvc status**
   - **Verify:** firstText.txt shows up as a modified file
   - Run **jvc save** with the message "Second version. Modified a file."
   - Run **jvc status** again

- **Verify:** The program displays the text: “No modified, newly created, or deleted file found”
- **Verify:** There is a new file under “.jvc/obj/version” called “1”, in which there is a pointer to the tree object which contain pointers to all blobs and other trees inside the repository. The tree reflects the exact state of the repository at the time that save happens. This means that the new blob created has the new content of firstText.txt, and that it’s correctly pointed to by all of its parent trees.

d. **jvc status** and **jvc save** (3rd demo)**:**
   - Continue from the previous demo
   - Delete the file “secondText.txt”
   - Run **jvc status**
   - **Verify:** secondText.txt shows up as a deleted file
   - Run **jvc save** with the message “Third version. Deleted secondText.txt”
   - Run **jvc status** again
   - **Verify:** The program displays the text: “No modified, newly created, or deleted file found”
   - **Verify:** There is a new file under “.jvc/obj/version” called “2”, in which there is a pointer to the tree object which contain pointers to all blobs and other trees inside the repository. The tree reflects the exact state of the repository at the time that save happens. This means that the root tree no longer points to any file named “secondText.txt”.

e. **jvc history**:
   - Continue from the previous demo
   - Run the command **jvc history.**
   - **Verify:** The following will show up in the terminal:

```
PS C:\Users\Jeremy Duong\Projects\SeniorProject> jvc history
Version <2>
        Third version. Deleted secondText.txt
Version <1>
        Second version. Modified a file.
Version <0>
        Initial version. Adding firstText.txt and secondText.txt

PS C:\Users\Jeremy Duong\Projects\SeniorProject>
PS C:\Users\Jeremy Duong\Projects\SeniorProject>
```

f. **jvc revert**:
   - Continue from the previous demo
   - Run the command **jvc revert** with the version index "0":

     ```
     jvc revert 0
     ```

   - **Verify:** the repository is reverted to the exact state right after version 0 was saved. That is, there are 2 files:
     - "firstText.txt" with the content "This is the first text content"
     - "secondText" with the content "This is the content for the second text"

2. Testing:

Requirement 1.a: Verify that jvc init creates the ".jvc" folder with the correct starter files:

- Create a new folder and designate it as the working repository
- Enter the folder using the terminal
- Run the command **jvc init** in the terminal
- **Verify:**
  - The ".jvc" folder is created.
  - Inside the ".jvc" folder, there are 3 subfolders:
    - head, which contains:
      - the file "master"
    - idxSup, which contains:
      - the file "tree"
      - the file "version"
    - obj, which contains:
      - empty subfolder "blob"
      - empty subfolder "tree"
      - empty subfolder "version"

Requirement 1.b: Verify that jvc init prints an error if the ".jvc" folder already exists:

- Continue the test for requirement 1.a
- Run command **jvc init** again
- **Verify:**
  - The message "Error: Already a repository"
  - All files inside the ".jvc" folder stays the same (not corrupted)

Requirement 2.a.1: Verify that jvc status shows newly create file:

- Create a new folder and designate it as a repository using jvc init
- Add a file called “firstText.txt” in the repository created. Put the content “This is the first text content” in the file created
- Run the command **jvc status**
- **Verify:** “firstText.txt” shows up as a new file
- Add another file called “secondText.txt” in the repository. Put the content “This is the content for the second text” in this file
- Run the command **jvc status** again
- **Verify:** both “firstText.txt” and “secondText.txt” show up as new files

Requirement 2.a.2: Verify that jvc status shows modified file:

- Continue from the test for requirement 2.a.1
- Edit the file “firstText.txt” and give it the new content: “This is the new content of the first txt”
- Run the command **jvc status**
- **Verify:**
    - “firstText.txt” shows up
    - “firstText.txt” shows up as “modified”

Requirement 2.a.3: Verify that jvc status shows deleted file:

- Continue from the test for requirement 2.a.2
- Delete the file “secondText.txt”
- Run the command **jvc status**
- **Verify:**
    - “secondText.txt” shows up
    - “secondText.txt” shows up as a deleted file

Requirement 2.b: Verify that jvc status shows the correct message when there is no change since the last save:

- Continue from the test for requirement 2.a.2
- Immediately run **jvc status** again
- **Verify:** The message “No modified, newly created, or deleted file found” is shown

Requirement 2.c: Verify that jvc status shows the correct error message when operated in a folder that is not a jvc repository:

- Create a new folder WITHOUT designating it as a jvc repository with “jvc init”
- Run **jvc status** inside the folder just created
- **Verify:** The message “Error: Not a jvc repository” shows up.

Requirement 3.a and 3.b: Verify that jvc creates blob, tree, and version objects correctly, and that the blob, tree, and version objects created are enough to recreate the state of the repository using jvc revert

- Continue from the test for requirement 2.a.2
- Run **jvc save** with the message "Initial version. Adding firstText.txt and secondText.txt"
- **Verify:**
    - There is a new file under ".jvc/obj/version" called "0", in which there is a pointer to a tree object named "0"
    - There is a new file under ".jvc/obj/tree" called "0" which contains information about 2 blobs, including their hashed names and the files that they represent ("firstText.txt" and "secondText.txt)
    - There are 2 blobs created under ".jvc/obj/blob" with the hash name specified inside tree "0" previously mentioned
    - The content of the 2 blobs created are the same as the content of "firstText.txt" and "secondText.txt" respectively

Requirement 3.c: Verify that jvc save displays the correct message when there are no changes detected

- Continue from the test for requirement 3.a and 3.b
- Run **jvc save** again
- **Verify:** The message "No unsaved changes found" is displayed.

Requirement 3.d: Verify that "jvc save" saves the version message correctly

- Continue from the test for requirement 3.a and 3.b
- **Verify:** Under ".jvc/obj/version", look at the file "0" and confirm that the message "Initial version. Adding firstText.txt and secondText.txt" is found there.

Requirement 3.e: Verify that "jvc save" uses the default message when no version message is provided with the save.

- Continue from the test for requirement 2.a.2
- Run **jvc save** without any message
- **Verify:**
    - All verifications of the test for requirement 3.a and 3.b
    - Under ".jvc/obj/version", look at the file "0" and confirm that the default message "`Saved version index 0`" is found there.

Requirement 3.f: Verify that "jvc save" displays the correct error message when executed in a non jvc repository folder.

- Create a new folder WITHOUT designating it as a jvc repository with "jvc init"
- Run **jvc save** inside the folder just created
- **Verify:** The message "Error: Not a jvc repository" shows up.

Requirement 4.a: Verify that "jvc history" displays the correct versions history.

(Refer to demonstration **e** under the Demonstration section. Ensure all verifications in that demo are confirmed)

Requirement 4.b: Verify that "jvc history" displays the correct error message when executed in a non jvc repository folder.

- Create a new folder WITHOUT designating it as a jvc repository with "jvc init"
- Run **jvc history** inside the folder just created
- **Verify:** The message "Error: Not a jvc repository" shows up.

Requirement 5.a: Verify that "jvc revert" displays the correct error message when a version index is not provided.

- Execute demonstration **d** under the Demonstration section WITHOUT providing a version index. That is, only execute the command:

```
jvc revert
```

- **Verify:**
  - The following message is displayed to the terminal:

```
Usage: jvc revert <version-index>. Use 'jvc history' to look up version indices.
```

  - Ensure that the state of all files up to the time right before the command is executed in demonstration **d** are the still the same

Requirement 5.b: Verify that "jvc revert" actually reverts the repository to the state specified by the version index.

(Refer to demonstration **d** under the Demonstration section. Ensure all verifications in that demo are confirmed)

Requirement 5.c: Verify that "jvc revert" displays the correct error message when an invalid version index is provided.

- Execute demonstration **d**, BUT with version index 4
- **Verify:**
  - The following message is displayed on the terminal:

    ```
    Error: Version Not Found! Use 'jvc history' to look up version indices.
    ```

  - Ensure that the state of all files up to the time right before the command is executed in demonstration **d** are the still the same

Requirement 5.d: Verify that "jvc revert" displays the correct error message when executed in a directory that is not a jvc repository.

- Create a new folder WITHOUT designating it as a jvc repository with "jvc init"
- Run **jvc revert** inside the folder just created
- **Verify:** The message "Error: Not a jvc repository" shows up.

V. Citations:

(No citation was used)

_____The End_____