**Senior Project Reflection**

**Name: Jeremy Duong.**

1. **Briefly describe your project.**

JVC (Jeremy’s Version Control) is a version control system that I’ve built from the ground up using C++ standard library. It is built with very similar object model as Git with only a few differences. Compared to git, JVC is designed to be much friendlier to Computer Science and Software Engineering students who are new to the concept of a version control system. The vocabs used to refer to certain objects and actions in the JVC scheme is also more intuitive and easier to understand for new users.

Git Repository: <https://github.com/JereMIbq1995/JeremyVersionControl.git>

1. **How did you meet, exceed, or fall short of your original expectations?**

At the start of the project, I wanted to set high expectations for my project, so much that I feel a little uncomfortable about my ability to meet all of them. However, I’ve been able to meet all of the expectations I set for myself. All the core requirements are completed. The main 5 functionalities of JVC, including **init, status, save, history,** and **revert** have all been completely implemented and well functional as expected.

In terms of my effort spent to complete the above functionalities, I notice that the amount of work I needed to do was way less than I projected. The project only took a total of 107 hours, while to the expected time of completion was 200 hours. So, I used almost half as much time as projection.

Given the complexity of the project, what I have been able to accomplish, and the amount of time it took to do so, I would comfortably say that the result certainly exceeded my expectations.

1. **What changes were made to your project during the process?**

During the process of completing the project, there have certainly been some changes as compared to the original version of the project. Most of the changes sprung from the fact that I was trying to use the Git object model and then realized I could do something simpler.

For example, in the Git object model, there is a hash associated with every blob, tree, and version object. The purpose of having a hashed name for a tree based on its content is so that Git can save space and never have 2 trees with duplicated content. However, this approach takes a toll on the performance since every time a tree is created, its content needs to be hashed. As of the version objects, the purpose of having a hash name for every version is also to differentiate actual changes of the content, and also to facilitate collaboration between different contributors of the same repository. However, given the simplicity I try to achieve for JVC, I only see the need to hash blobs, but NOT trees and versions. The identification of such objects are provided by an index supplier which simply tells me the next available integer to be used for the purpose of naming them. Because of this change, I was able to greatly simplify the algorithm and speed up the progress of the project.

Another change is in my vision of the functionalities the project. I originally wanted the project to have an additional branching functionality, executed using the “**jvc branch”** command. However, as I developed JVC, I realized that branching can be achieved by reverting to a recent but older version, and then execute a save from that version. I found that this method of branching is even more intuitive to the user and closer to the goal of achieving simplicity that JVC had at the beginning of development. As such, I decided not to implement **jvc branch**. This direction would make JVC unique compared to Git in the way it handles collaboration and branching, if I choose to further develop it in the future.

1. **What were the most important/interesting lessons learned about the new computer science topics you learned about?**

In terms of computer science lessons, I learned a lot about a variety of topics. Among the most important ones are: recursions and how an understanding of trees can help improve your understanding of how directories are handled, files reading/writing using binary, and algorithm efficiency when hashing must be done.

My project uses recursion in every single functionality, except for **jvc init.** The **status** functionality must traverse through the current directory, their subdirectories, the subdirectories of all such subdirectories, and so on… in order to find all the changes to the directory compared to the snapshot of the directory taken by the last save command. The **save** functionality must also traverse through every single file and subdirectory in order to create a snapshot of the repository. The **revert** functionality needs to traverse through every file and folder of the current snapshot, compare it to that of the version that the repository is to be reverted to, and make the modification, addition, and deletion needed to revert the repository to a previous version correctly. The **history** functionality also needs to recursively read the parent versions of the current version until it reaches the initial version. All of these traversals can only be done with an understanding that directories are essentially trees, and that one of the most intuitive way to traverse a tree is through recursion.

Another lesson of computer science that I must understand in order to complete the project is the fact that every file is just a set of 1’s and 0’s. Yes, this is a very classic lesson about computers. But, with an understanding of this, a person can realize that if he copies the exact sequence of 1’s and 0’s from one file to another and then names it the same way the original file was named, he achieve the exact copy of the original file. This understanding is fundamental to reading and writing contents of files to and from BLOBs, which allows the revert functionality, which is also just a copying or comparing of bits, to be done correctly.

Lastly, one problem that I ran into while completing the project is answering the question: “how fast can I hash the content of a file while I have so many files to hash?” The SHA1 hash algorithm requires that a fixed amount of iteration (80 iterations if I’m not mistaken) must be done for every 512-bits block of file content. This means that if I were to hash every single file when I run ‘**jvc status**’ for the purpose of detecting changes, it can certainly take up a lot of time to do so. Luckily, I realized that detecting of changes can be done simply with a bit comparison instead of hashing like I used to think. Because of this realization, I was able to implement **jvc status** to operate with very reasonable speed. However, the ‘**jvc save**’ still needs to do this extensive hashing for many files in order to create blobs. The solution to make it faster is to do the hashing only for files whose contents have been changed or for files that are newly added. This greatly reduces the run time as compared to the hashing-everything-aproach.

1. **What were the most important/interesting lessons learned about the project itself?**

I think the most interesting realization I have about the project is that it certainly does not have to be handled the same way Git is handled. Certainly, there are lessons to be learned and apply in terms of storage and algorithm optimizations, but I can still make JVC whatever I want it to be. Yes, the original vision is to make it like Git. But, once the foundation for the basic functionalities and storage schemes are thought out and properly set up, I can start thinking about solutions to collaboration and branching that work for JVC, but might be completely different from the ones used by Git.

For example, as mentioned above, branching is one of the main functionalities implemented by Git. However, with the JVC is set up and with its goal to make things simple for new users, branching can just be done with a revert and a save.

Another thing I also learned is that if I so desire to make a jvchub (like github) to store my repository remotely, all I would need to communicate to that remote repository to make sure all my changes are updated is the “.jvc” database. As long as the “.jvc” database is in sync between the local and the remote, we should be golden.

1. **What were the most important/interesting lessons learned about yourself?**

To be honest, I was actually very impressed with myself and the progress I have been able to make. When I first set out to make my own git, regardless of how determined I was to understand Git, I certainly had a lot of doubts. The amount of research I knew I had to do and the complexity of the algorithms and storage schemes I needed to use were altogether a big mountain for me to climb.

However, my big lesson is that all big things start small. It worked the same way when Linus Torvalds invented Git. Originally, I imagine it was something as small as the JVC I have now. Before that, it was probably something even smaller. But, how do you eat an elephant? One bite at a time. How do you climb a mountain? One step at a time. With JVC, I started from the ground up answering simple questions such as “How do I hash a string using SHA1 hash?”, “How do I hash the content of a file?”, “How do I turn a file into a BLOB?”, “How do I turn a directory into a TREE?”, etc… Each of these small questions has its own answer, but when all put together, they create JVC.

Truly, the pattern of creating something great is always the same: “*by small and simple things are great things brought to pass*” (Book of Mormon, Alma 37:6)

This made me realize that I myself am a “small and simple” thing because of the vast sea of knowledge of computer science and other topics that I have not yet comprehended. But know that I can still do many great things if I diligently put enough effort in mastering the small and simple knowledge and skills that I have.

1. **Final Thoughts**

Understanding Git and how it works on the inside has always been in my mind ever since I started using it. As such, being able to create something like JVC is definitely something that I’m most proud of and have learned the most from. It’s one of these things that you create not just to get a grade but purely for the joy of seeing it works. It’s a joy to see a functionality works the way I expected it to work in a big test folder even with a very complex recursion algorithm sitting inside of it. It’s a joy (and a pain sometimes) to track the JVC source code using JVC itself, to see 26 different versions shown by “jvc history”, and then revert version 26 to 0 and back to 26 with no error (after so many errors that I got before). I’m grateful for the opportunity to create JVC and the things I’ve learned while creating it, especially to learn that I can do hard things, knowing that there are even harder things for me to do in the future.