TELER: A Streamlined Version Control System

Project Proposal

Hadley Luker Zachary J. Susag

April 23, 2018

1 Problem Outline

During their computer science education, many students utilize version control systems for class assignments, professional work, and personal projects. However, students rarely use the entire set of functionality available from version control systems, either due to its unintuitiveness or its inapplicability to their common use cases. For example, a common workflow for a student using Git entails the following:

1. `git add .`
2. `git commit -m "bad commit message"`
3. `git push`

As you can see, the process to push all changes made in a local repository to a remote takes three separate commands: the addition of changes to a staging area, marking these changes with a commit message, and then pushing the changes to the remote. Many students are not familiar with commands outside of this range, and in fact even versions of these commands with different flags or arguments are unfamiliar. For instance, if a student runs `git commit` without any flags, Git automatically opens Vim by default. Vim's unintuitive keybindings and minimal user interface often cause unfamiliar students to panic and quit the terminal entirely. Other common problems include manually resolving merge conflicts, failing to add all the desired files to the staging area, and difficulty reverting a repository to a specific previous commit.

One conceptual barrier a student must overcome is the staging area. Given the above workflow, the staging area seems superfluous as the main goal of a VCS is to synchronize changed files across systems and to label said changes.

Overall, git dilutes its functionality among numerous, unintuitively named commands. As such, version control systems like git have a steep learning curve before a user could fully benefit from the power they provide.

2 Approach

We propose a streamlined, minimal, distributed version control system, the main emphasis of which is on the core of a typical student's workflow: synchronization with a remote repository

and change tracking. Instead of spreading functionality out among several commands, like git, our proposed VCS, dubbed TELER will have two main commands for pushing and pulling from a remote as well as auxiliary commands for reverting to previous commits, branching, etc.

All of our proposed commands will operate off of reasonable defaults based upon the student's common use case. For pushing to a remote we assume that the user wants to send all changed files to the first added remote, named *origin* in git. This condenses the "adding", "committing" and "pushing" steps in git to a single command in TELER. It should be noted that this approach eliminates the staging area entirely. In order for a user to send specific files instead of all changed files they would need to specify via a flag. For pulling, we make the assumption that the user would like to receive all files from the remote directory, just like git.

By collapsing the "add", "commit", "push" routine into a single step, symmetry is created between the actions of pushing and pulling. We argue this symmetry provides a more natural, intuitive workflow.

To measure how successful TELER is we would like to ideally have a student test the system out and see how natural it is. If it comes intuitively while still providing the essential power that git does, we would consider TELER a success.

3 Problem Solving Strategies

The two problem solving strategies that TELER will use are networking and persistent storage.

3.1 Networking

TELER will be a distributed VCS, meaning that a copy of the full history of the repository will be stored locally. This makes it possible to use TELER without the need of a central server, instead relying on peer-to-peer communications. Upon a push/pull, files will be sent to or received from the network location of the remote. We hope to be able to make this connection by opening a `ssh` socket to the remote repository and transmitting a serialized version of the commit being sent or received. It should also be noted that this serialized version would be encrypted via `ssh`. Essentially, we hope to achieve the same distributed functionality as git.

3.2 Persistent Storage

A crucial part of TELER will be detecting whether a file has been changed since the latest commit. As such, when the user is ready to push changes to a remote TELER will have to recursively scan the directory for new, changed, or removed files from the previously stored commit. Once these files are identified they will need to be added to the shadow directory (`.git` in git) in a manner in which it is easy and fast to find the commit. To achieve this we will store the file under its SHA-1 hash, along with other commit metadata, and subsequently store the compressed file in an objects directory. To minimize space we will compress all files using `zlib`. It will also potentially be necessary to store metadata information of these files such as owners, permissions, and modification times. In essence, we will be creating a incredibly minimal filesystem to efficiently store the history of the repository.

4 Implementation Outline

We identify three major components of TELER: the shadow directory structure, the network transmission protocol, and determining differences between file versions.

For the shadow directory we plan to use a similar structure as git. In git, within the `.git` directory there is a file for each branch which lists the latest commit, configuration information such as remote locations, and all of the objects which represent the full history of the repository. An object is either a tree or a blob, both compressed using `zlib`. A tree represents a directory and can contain trees and blobs. A blob is simply a compressed version of a file present in the directory. The objects will be stored according to the SHA-1 hash of the file to allow for both rapid checking of differences as well as efficient search within the object directory. Note that this structure naturally lends itself to a recursive approach. When a user wishes to revert their directory to a previous commit all that is necessary is to find the commit number and decompress the commit log which will list the head of the tree. From there, all that is necessary is to descend the tree structure and write the decompressed blobs to the working directory. As a result of this organization no file will be stored multiple times within the directory; however, multiple references to a blob may exist. In essence, the shadow directory is simply a directed, acyclic graph in which the nodes are the commits and the edges preserve precedence among said commits.

For the networking portion we will use a similar peer-to-peer networking system as in the chat room lab. On each system TELER will run in the background continuously accepting connections from other instances of TELER on other systems. When a push command is executed a socket to the default remote location, as specified by the user, will be opened. Once a connection has been established the local instance of TELER will send first the metadata information about the latest commit, then the entire tree of blobs concerning that commit. The remote instance of TELER will receive these changes and store them within its shadow directory using the same organization mechanism. Once the changes are received the connection is severed and the remote directory will be changed to reflect the differences. We hope to be able to use the `ssh` protocol to send these changes in an efficient manner.

Lastly, in order to determine whether a change to a file has been made, first TELER must traverse the tree of blobs from the latest commit, generate the hashes of the files in the working directory, and compare them to those of the latest commit. If the hashes are different then the file has been changed and this change should be reflected in the shadow directory when the push command is executed. Along with this component are commit reversion utilities and creating diffs of changed files.

By the end of week one we hope to have fleshed out the shadow directory structure and being able to add commits to the shadow directory. By the end of week two we hope to be able to have TELER efficiently find changed files, create commits automatically, and update the shadow directory as appropriate. The final week we will be adding networking functionality to allow for synchronization across machines.

We expect that the greatest difficulty will be creating this shadow directory. With the amount of file operations along with metadata tracking it is probable that some data will be lost among the first few versions. It also may be necessary to simplify the structure and attempt to enhance it later. Furthermore, the serialization of the commits into a format that is easily sent and received could fail, especially if we combine that with using `ssh`. Figuring out how to have the remote repository determine where it should place the received commit could also prove difficult.