TELER: A Streamlined Version Control System

Project Report

Hadley Luker Zachary J. Susag

May 15, 2018

1 Project Overview

During their education, most computer science students must learn to use a version control system (VCS). Although these systems see wide use in research and industry for software development and production, in educational contexts they typically service a narrower use case: saving, submitting, and tracking changes in programming assignments. This limited use case causes many students to become overwhelmed when confronted with the numerous and powerful commands available to modern VCSes such as Git and Mercurial. For example, the typical workflow of saving changes to a remote repository in Git proceeds as follows:

1. `git add .`
2. `git commit -m "Fixed bug."`
3. `git push`

These three commands are practically an idiom that students are habitually trained into, and deviations from this pattern are rare and incite confusion. For example, if the `-m` is omitted from the end of `git commit`, the terminal will run the Vim text editor by default in order for the user to write a commit message. Due to Vim's unintuitive keybindings and minimal user interface, students confronted with this sudden change in environment may make errors in the editor or find themselves unable to exit and subsequently close the entire hosting terminal window.

For these reasons, we decided to develop TELER: a streamlined version control system for the student use case. Our system exposes exactly five commands: repository initialization, pushing changes with a required commit-like summary to a remote, pulling changes from a remote, reverting changes to a specific version, and viewing version history. This interface condenses the common add-commit-push / pull command sequence at the expense of requiring all files to be pushed or pulled at once. However, this has positive effects: it encourages users to push frequently and document important changes as they are made.

Evaluation summary.

2 Design & Implementation

The design of our program is centered around two main commands and three components: the command line interface, the shadow directory which contains all changes recorded by TELER, and the auxiliary data structures and compression that enable TELER to be more time and space efficient.

2.1 Design

The use of textscteler depends mostly on two key commands, `push` and `pull`.

2.1.1 textttpush

The act of pushing in a TELER repository saves new changes in a space-efficient manner within the TELER shadow directory, described later. The program first loads the previous commit into memory and compares the hashes of the files it finds to those in the current working directory at those positions. If they differ, TELER discards the changes *** i forgot at *** this point ***

2.1.2 textttpull

2.2 Shadow Directory

2.3 Command-Line Interface

The command-line interface is a simple argument parser made using GNU C's `argp` interface. This interface automatically generates help and syntax information for display on the command line through the assignment of certain global variables and "option vectors", a structure which organizes and configures command line arguments. However, our implementation is slightly modified from the GNU standard of C argument morphology. The GNU practice is to precede all arguments on the command line with a number of hyphens: one for single-character arguments (e.g. `-v`) and two for long arguments (e.g. `--verbose`). We chose to let the first argument be hyphen-less. (For differentation's sake, we refer to this "first argument" as the "command".) This reflects more closely the practices used by Git for their command line interface, which we suspect will be more familiar to students. This decision caused us to parse the first argument through an if–else if chain and instead use `argp`'s parser function to parse arguments given to the command itself.

2.4 Data Structures

2.4.1 Hash Table

The hash table indexes all files within the repository, in addition to certain directory information and commits. Any data structure could feasibly be used for this, but since its primary purpose is for lookup, a hash table was chosen.

2.4.2 Tree

The tree organizes hash table entries by their file system structure. It enables `teler` to save or reconstruct the version of the repository that was saved in the latest commit.

2.5 Compression

As TELER saves every version of a file that was added in a change, it becomes important to utilize every means of space-reduction we have available to us. Therefore, all files and commits within a TELER repository are compressed. We utilize the `zlib` library to accomplish this. *** How it do and how we use ***

3 Evaluation

To evaluate TELER we compared its speed in generating commits through TELER PUSH with Git. For hardware we used an Intel Core i5-3320M at 3.3GHz with 8GB of RAM with a 320GB spinning hard disk. The workstation we used to evaluate TELER was running Arch Linux with kernal version x86_64 Linux 4.16.8-1-ARCH.We collected the running times of Git and TELER using the `time` command.

To fairly assess each program we generated a random directory with a maximum depth per directory of 3, up to 4 first-level directories, and up to 6 maximum children per directory. We then recorded the time it took for each program to make a commit of the entire directory. This was done ten times with a new subdirectory of the same specifications being made, increasing the size of the total directory each time. During this process we also recorded the total size of the directory in bytes.