Implementing a Software Revision Control System for MASS

Matthew Sell, Winter 2014

Table of Contents

Summary of Work

It has been recognized by Professor Fukuda that a means of managing source code revisions and tracking change history for the MASS project is necessary as more people perform modifications and submit enhancements. Traditionally, a software revision control system is used to satisfy these basic requirements. The goal, as outlined by Professor Fukuda, was to implement a revision control system using existing and/or easily obtainable resources, and to provide documentation and training in an effort to keep this system operational in the future.

Requirements and Deliverables

The requirements and deliverables, as outlined by Professor Fukuda, were the following:

- Use the “Git” revision control system, as it seems to be adopted by many development organizations worldwide and is gaining popularity at a fast rate
- Create one or more repositories that will be available to instructors and students at UWB, but not available to the general public
- Create a repository to be used for practice and training purposes
- Procedures and/or tools must be provided so that access to the repository can be controlled by Professor Fukuda
- Create a presentation that will outline the need for software revision control, how “Git” satisfies those needs, and basic use of Git as it relates to the MASS project

- Create a presentation and provide a demonstration of how Git is used to accomplish various common software development tasks
- Recommend client applications to be used with the Git repositories
- Document repository creation methods

Implementation

Repositories

The greatest determining factor in deciding a location for the repositories was the requirement that access be provided to students and instructors at UWB but not available to the general public. This requirement eliminated several possible Git repositories such as “Github”. For ease of managing permissions, there was a choice between using a tool to manage repository permissions (such as “Atlassian Stash”, or “Gitosis”) or using an existing mechanism, such as network directory access privileges. The decision was to use network privileges and a filesystem-based repository (using SSH for access); this configuration did not require the installation of additional software yet still provided the ability to establish access privileges.

To determine how many repositories would be required and what partitioning scheme would be used, the research group was asked for recommendations. The general consensus was to partition repositories by programming language/framework (Java, C++, and CUDA) with a training repository to be created as well. Four repositories were to be created, with only the training repository containing sample files for practice; the other three repositories were to be left “empty”, populated with source code once Professor Fukuda was satisfied the sub-projects were ready for inclusion in the repositories.

These four repositories were then created, and in the root directory a simple text document was added that provided the command-line statements used to create the repositories.

It should be noted that sample applications are to be included with the respective implementation/repository. For example, sample applications developed in Java are to be placed into the Java repository.

Client Applications

There are many methods for interacting with a Git repository, and it was necessary to evaluate several different tools and provide recommendations and documentation for using the suggested tools. After performing "hands-on" evaluations of four different tools, I recommended the following three:

- Git, command-line: For use with Linux. Although a command-prompt version is available for Windows, it is my recommendation to use graphical-based Git tools for simplicity.
- Eclipse Egit: For use with the Eclipse IDE (Java, C++). Egit is a good tool for performing simple operations while developing within the Eclipse IDE. Advanced functionality (conflict resolution, viewing branches) is available using different menu options from the well-known "Team" menu item. This advanced functionality was not discovered until late in the project and was not evaluated.
- Atlassian Sourcetree: For use with Windows and MacOS. Sourcetree is a more complex tool and has a "busy" user interface, but provides easy access to advanced functionality. Sourcetree also provides a nice presentation of branching and merging history. This tool is recommended in conjunction with Egit; Egit is suggested for daily development activities in Eclipse, and Sourcetree for branching, merging, and conflict resolution.

The above listed applications were demonstrated to the research group and a presentation prepared that outlines downloading, installation, and operating instructions for each.

Remaining Tasks

I attempted to provide detailed instructions on branching, merging, and conflict resolution using the various Git clients, however there was not enough time left in the quarter to produce instructions and a working demonstration. The branching model in Git is a very powerful feature and it will be advantageous for the group to have documentation and a working demonstration before we can utilize this feature.

Future Considerations

I recommend that we explore using Git branching to create long-running branches for major release versions, an integration branch for merging work from contributors, and individual branches for each active developer and/or feature branches. This branching scheme is comparable to typical industry practices and would provide a more robust development environment by isolating potentially breaking changes from the working/released mainline. This recommendation should only be acted upon once sufficient documentation has been created to allow productive use of the Git branching model.