Using Git to Enhance Productivity

Git, like Subversion, is an example of a Version Control System, or *VCS*. As one of the oldest systems still in use, knowing how to utilize Subversion is of the utmost importance. Despite this, or perhaps because of its prevalence, a number of competitors have recently been developed. The majority of these are known as Distributed Version Control Systems, or *DVCS*. Examples of such systems include Git, Mercurial and Bazaar.

On a superficial level, the main difference between a centralized VCS and a DVCS is where the code history is stored. In a Subversion-controlled project, the history is stored on a central server. Google Code is a common example of a source code repository, or *repo*. When you have a copy of the project, all you have is the latest version, also known as the *working copy*. In a DVCS, you have a complete copy of the code history; this gives you more power over the *revision tree*. For instance, you can experiment with your code without disrupting your partners' work. You are also able to commit changes without having an Internet connection; this is a major restriction of Subversion, and can inhibit productive work being done on your project.

Getting Git

Just as Subclipse integrates Subversion into the Eclipse workflow, EGit gives Eclipse the power to manipulate Git repositories. To install EGit, navigate to "Help > Install New Software" and enter "http://download.eclipse.org/egit/updates". The next step is to set up a Git-compatible repository. The recommended location for your repo is GitHub; not only is GitHub the most well-known Git repository, they have a large amount of documentation covering the tool and even offer transparent Subversion integration! This way, you can use the advanced features of Git while other students not as comfortable with yet another tool can continue to use Subclipse. If you had to complete the Subversion lab, using EGit will feel similar; despite the familiar interface, however, EGit give you a large array of new tools to use.

Unique Features of Git

Covering the vast array of features Git has to offer is beyond the scope of this tutorial. Therefore, we shall focus on a few unique tricks that you can easily utilize to help optimize your coding style.

Working Copies

As mentioned above, a DVCS stores the complete copy of the history locally so you may modify it without affecting your partners. Using this requires no work on your part; you are free to modify your code and try new ideas without worry. A common issue in Subversion that can be avoided due to this design is when one member changes the type of an instance variable. Doing this has a snowball effect: constructors, getters and setters must be rewritten; these changes cause any classes dependent on this class to break as well. In Git, the member could begin a major revamp of the code without

affecting another's current work. When they have finished altering the code, they push the changes to their partners by clicking "Team > Push to upstream."

Branches

There are a number of DVCS implementations being used right now. The main feature that distinguishes Git from Mercurial and others is its branching feature. Most version control systems offer branches; however, the concept as it is applied to Git is far different. When you create a branch in Subversion, it is relatively permanent. A complete copy of the code is created and will maintain a separate history from that point forward. This is useful in a production environment; in the context of this class, however, it is not necessary. A branch in Git, by contrast, is purposefully temporary. A Git branch is best explained by example. Assume you want to create a new intermediate JFrame to display between existing ones; at the same time, however, you need to fix a few bugs in your database. A Git-oriented workflow would have three branches: "master", "new-frame" and "database-fixes". If you want to take a break from working on the new window, simply click "Team > Switch To..." to work on the database. The user interface will work the same way it worked before you started on your new JFrame. So even if you didn't finish your JFrame, you can work on a different part! Once you have finished your new JFrame, how do you make it permanent? Just click "Team > Merge".

Other Resources

A large amount of documentation exists on Git. Some more famous ones are below:

- Official Git website: Useful for people who use command-line Git
- GitHub help: Help with GitHub and general Git help
- EGit manual: Instructions on how to use EGit, the Eclipse plugin for Git