Git

Introduction

Git is a powerful and useful tool, but if used improperly it can be a clusterfuck for managing code. This page is designed to serve as a basic introduction to git, outline how Robosub is using git, and links to helpful information. How we use git is very much tied to the overall software development workflow, so I recommend reading that as well.

Getting Started

I highly recommend you start out by going over our intro tutorial to git, which describes a few high-level concepts that are important to understand before you start using git. For reference, you can find the official git documentation here.

Branches

The workflow we use consists of a few types of branches.

1. `dev` - This branch can be assumed to always be stable, meaning the code in these branches should always compile, and should pass all unit/module tests. User branches are merged into dev after passing a code review.

2. `master` - In contrast to dev, this branch is considered to be super-stable, meaning it meets all the requirements of dev, and in addition the code has been verified via sucessful pool test. Dev leads master, and master catches up to dev whenever the code in dev is considered super-stable.

2. `user/feature branches` - These types of branches are the main ones most people will be directly interacting with. When naming your branches, it's recommended to prefix the name with your name. For example, if I was working on thruster code, I would name my branch “james/thruster”.

Rebase/Integration Method

We use the Rebase/Integration method as our workflow and policy for merging in new code. This method is an alternative to the more common “branch and merge” method. The simple short version of this policy is as follows:

1. Users always work on their own branch
2. When users feel their code is stable, they rebase onto the devbranch
3. A code review will be set up, possibly resulting in software tweaks (back to step 1)
4. An integrator will fast-forward the dev branch up to the user’s branch

Integrators are a small group of people that control what code ends up going into the dev branch and

keep the git repo clean. This group should only be a few people, possibly even just one person for each repository. Team leads might be a good person for this position.

The day-to-day workflow for users is as follows:

1. fetch the latest changes
2. rebase current branch on the integration branch, fix any merge conflicts
3. start coding, make commits on own branch
4. always commit when done working for the day

Advantages

- code has a very linear history
- naturally encourages people to stay up to date with the integration branch
- merge conflicts are easier and safer to resolve
- stable code is controlled by a select few
- No extra commits just showing a merge

Disadvantages

- The parallel development history is lost
- requires integrators to stay on top of things

Commit Often, then Squash Later

Users often go long periods without commiting their work, because they feel like small changes are not worth committing. They wait until a substantial amount of changes have occurred, which can range from a few days to a few weeks. This is bad! This results in others not being aware of work that the user is doing, and also risks work getting lost. Users should at a **minimum** commit whatever they have done at the end of their current work session that day.

Some may complain that this will result in a cluttered git history, however the solution for this is simple. When code is ready to be put into an integration branch, the user should perform an interactive rebase, and at that time they can squash together multiple commits into a single commit. By doing this a user can have 15 commits on their own branch (useful when they are developing), but when they believe their added feature is good and stable they can squash-rebase onto dev, and have single commit representing all the changes for the added feature. This results in a very clean and descriptive history when looking at the dev branch.

What Files to Commit?

As a general rule of thumb, the repo should only contain source files. Anything that is generated from those source files (like binaries, build files, and logs) should not be committed, nor should data files. Never, ever add binary files like pdfs, executables, zipfiles, movies, music etc. These will get deleted!

Quick reference

- `git fetch` retrieve the latest changes from the server
- `git rebase <other branch>` rebase your current branch on top of \<other branch>, which typically should be dev.
- `git checkout <file/directory name>` reset all unstaged changes to \<file/directory name>

From:
http://robosub-vm.eecs.wsu.edu/wiki/ - **Palouse RoboSub Technical Documentation**

Permanent link:
http://robosub-vm.eecs.wsu.edu/wiki/cs/git/start?rev=1471666075

Last update: **2016/08/19 21:07**