GitLab/Git 101

Author: Adam Campbell

All the relevant files for the MicroCART project are saved on ISU's GitLab (git.ece.iastate.edu). GitLab provides us a centralized Git repository, much like GitHub does.

Git

Git, like SVN, is a version control system, and as such takes care of maintaining the files and backups, handling pushes/pulls, and providing a powerful and robust way to go back in history if something breaks. Unlike SVN (which needs a central server that everything is pushed to/pulled from), Git is *distributed*, so it's not necessary to have a central server at all. In fact, the central server, if any, is nothing more than just another client that's in a publicly known and accessible place. In Git, every client (basically every computer) clones a local copy of the repository, and then maintains it. This is a major difference from SVN and can cause some headaches if you forget about it.

Git maintains local repositories. This means that your personal laptop, the ground station computer, and anywhere else that has a copy of the repository, not only has a *copy* of the repository, but has a fully-fledged Git repo. This brings up the difference between **commit** and **push** in Git-world. In SVN, pushing is when you take the changes you've made on your local file system and *push* them to the central remote server, updating the repo for everyone. In Git, because there's a local repo as well as a centralized remote repo, a distinction is made. **Commit** is when you take changes and apply them to your *local repo*, and **push** is when you take your *local repo* and apply changes in that to the *remote repo (centralized server, like Gitlab).* You can push without committing, since push doesn't care about local filesystem changes; it only takes your local repo and imposes it upon the remote repo. **Pull** functions largely the same as in SVN, wherein changes from the remote repo are brought in and applied to the local repo.

The basic workflow for using Git with our setup is as follows:

1. First off, you need a **Git client**. Common ones are TortoiseGit (https://code.google.com/p/tortoisegit/) and GitExtenstions (https://code.google.com/p/gitextensions/). A Git client is a program that you use to connect to the centralized repo and clone/pull/push/commit/etc. It doesn't really matter which one you use.
2. Next, **clone** the Git repo. Your mileage may vary, but there's usually some kind of "git clone" option in your Git client. Clone from the repo's clone URL: https://git.ece.iastate.edu/awjc/microcart.git. This will copy all the stuff in the remote repo and

make a local repo on your local filesystem. (Note that you may have to change the *push URL* to the SSH version [su_gitlab@git.ece.iastate.edu:awjc/microcart.git] in order to push/pull).

3. Now that cloning is complete, you're set up to use the repo. **Do your work**, you glorious little worker you, and when you're ready to share your masterpiece with the world:
4. **Commit**. This is done in the Git client, and is the process by which you apply your changes formally, in a big chunk called a "commit", to your ***local git repo (NOT the remote repo).*** You can commit all your stuff at once, or do it in lots of small commits. Don't be shy to commit something, even if it's incomplete. Committing is a way of preserving your work so you don't lose it, and so it's easier to manage if you need to undo something. Committing does not affect the remote repo in any way, and other won't be able to see your changes until you:
5. **Push.** This takes all your commits and applies them in the same order to the remote repo. Assuming there are no conflicts (that is, no one has pushed, since the last time you pulled, some changes that change things that you are trying to push right now), then the push will complete and the remote repository will now reflect your changes (that is, future cloners and pullers will get your changes). Speaking of which, when you want to receive someone else's work in your own local repo, you will:
6. **Pull**. This is when you take the changes that have been pushed to the repo by others, and pull them into your own repo. That is, you take any commits that have been applied to the repo by others since the last time you pulled, and you apply those commits to your own local repository. You may not be able to pull successfully, depending on the state of your local filesystem. In general, it's a bad idea to pull if you have outstanding modifications on your file system. You'll need to commit your changes or stash them (which just undoes them temporarily, but you can pop the stash and redo them later) in order to pull in the new changes.
7. **Steps 3, 4, 5, & 6** will happen somewhat independently over the course of development. Just do your work like normal, commit when you want to save something on your local repo, push when you want to publish your changes to the remote repo for others to see, and pull when you want to bring in others' changes into your local repo.

Our Repo

In our particular repo, we are using only a single master branch, for simplicity's sake. Local repos are stored as usual on everyone's laptops and the computers in the labs (any machine that accesses the central repo creates its own local repo). The central repo is saved on Gitlab, specifically at a clone URL of https://git.ece.iastate.edu:awjc/microcart.git.

The repo has a *tasks* folder, in which all of the tasks we've worked on are stored. Each task has its own folder, which should be self-contained. The tasks related to hardware/software on the board typically contain a *sw* folder, where the XSDK (software) projects are stored, and a *system* folder, where the XPS (hardware) files are stored.