# Using Git for Version Control of FIRST Labview Projects

## What is Version Control?

Version control is keeping multiple versions of your files so that you can revert changes, look at edit history, and back up old versions. If you don’t want to lose your files or “break” them with experimental edits, then you should use some kind of version control. Version control can be as simple as copying and renaming a file before you edit it. However, there are a lot of software tools devoted to version control. You may have heard of some (Subversion, CVS, Mercurial, Git, etc). Modern version control tools also help multiple people work together on one project by allowing each person to have their own “branch” of the project and then helping you merge them together as features are completed and bugs are fixed.

## How Does Version Control Work?

Code is kept in what is called a “repository.” The repository stores your current version of the code, but also the history of all of your edits. After you edit code, you “commit” it to the repository. Each commit is stored as a separate version. The most recent version is called the “head.”

Imagine you have a project that is pretty stable, but you want to try out something experimental. If you want to keep your stable project separate from the experimental code, you can create a branch. The branch, like a tree branch, separates off from its parent (the stable project). Changes made to the branch don’t affect the main project, and the branch has its own version history. You can keep building commits onto the branch, and you can keep building commits onto the parent. You can even make branches of branches. If you want to bring the code from the branch back into its parent, you can merge two branches. Version control tools include software to help you look at the differences between two branches and choose which changes from each branch to keep.

Another reason to use branches is mentioned above. Multiple users working on the same branch will often get in one another’s way. If two users edit the same file at the same time and try to commit their changes, it will create a conflict that they will have to resolve. Conflicts can be resolved by merging the conflicted files, but in most cases, you want your users to work in separate branches. Each user has their own branch, so they don’t conflict with one another. One branch is designated the master branch, and users will merge working code back into the master branch.

Sometimes, a project will reach a milestone with a special significance—maybe a stable version of code to take to competition. At these milestones, you would create a “tag.” A tag is a named version that is kept separate from the rest of your versions. Unlike a branch, you cannot build on a tag.

## What is Git?

Git is a version control tool originally created by Linus Torvalds for version control on his project, the Linux operating system kernel. It is free and open source, meaning anyone can use or modify it. Git is what is called a distributed version control tool, which means that each user stores the whole project history on their own computer, but can sync with other users and a central server over the internet. This contrasts with older version control tools like Subversion, which stored all of the version history on the central server and were dependent on an internet connection. Because Git does not require an internet connection, it is ideal for FIRST teams working on the road at competition.

## Getting Git

Git is free to use and easy to install. Since you’re using Labview, I’m going to assume you’re also using Windows (although Git is available for Mac and Linux as well). To get Git:

1. Go to <http://code.google.com/p/msysgit/downloads/list>
2. Download the one labeled “Full Installer for official Git xxxxx Featured Beta,” where xxxxx is a version number.
3. Run this file to install Git. You can use the default options for everything.

You now have Git working on your system. You can start working with it now, but it only comes with some pretty basic tools, and you’ll have to do a lot of things with the command line. Fortunately, there’s a better way.

## TortoiseGit

TortoiseGit is a tool that integrates Git into the windows file browser. With Tortoise, you can do Git commands just by right-clicking files. To get and set up TortoiseGit:

1. Go to <http://code.google.com/p/tortoisegit/downloads/list>
2. Download the one labeled “Tortoisegit xxxxx 32bit Featured,” where xxxxx is a version number. (If you have a 64-bit Operating System, you can download the 64bit version.)
3. Run this file to install TortoiseGit. You can use the default options for everything.
4. Once you have finished installing, right click somewhere on your desktop. You should have a whole bunch of new Git related menu items. Go to TortoiseGit->Settings.
5. In the Settings window, go to Git->Config. Enter your name and email here (This is how other users know who you are when you share your code.)
6. If you do not have Labview installed, skip this step. In the Settings window, go to External Programs->Merge Tool.
7. Click “Advanced…” In the window that pops up, click “Add…” Fill in .vi as the extension and for External Program, fill in  
   C:\Program Files\National Instruments\Shared\LabVIEW Merge\LVMerge.exe %base %theirs %mine %merged  
   if you have a 32-bit operating system or  
   C:\Program Files (x86)\National Instruments\Shared\LabVIEW Merge\LVMerge.exe %base %theirs %mine %merged  
   if you have a 64-bit operating system. Click Ok to close this window.
8. Click Ok to close the “Advanced Merge Settings” window.
9. Click Ok to save your settings and close the “Settings – TortoiseGit” window.

TortoiseGit is now fully set up.

## GitHub

Git is great because it’s distributed—you don’t need to connect to a central server over the internet. However, you’re probably working with a group. It’s a lot easier for a group to work together if there’s a central hub for them to work from. It’s even better if that central hub has things like a project wiki, a bug/issue tracker, and a way to view all the files and commit history of the project. GitHub is all of these things, plus it’s free.

To get an account on GitHub, go to [https://github.com/signup/free](https://github.com/signup/free%20) and sign up. Once you’re signed in, you can create your own repositories or create an Organization. Send me your GitHub name and I’ll add you to our team’s organization (http://github.com/frc2399).

Before you can access repositories, you need to create a pair of keys to verify your identity when you connect to a repository on GitHub. One key is your private key, and the other is your public key. As the names suggest, one is for you to keep, and the other is for you to share. To create a keypair:

1. In your Start menu, open find TortoiseGit->PuttyGen.
2. Click Generate.
3. Wiggle your mouse around the window until it’s finished. Don’t close the window yet.
4. Go to https://github.com/account and click on SSH Public Keys.
5. Click on “Add another Public Key”
6. Give your key a title. This should help you remember what PC the key is from.
7. Copy the Public key from the box in the PuttyGen window and paste it into the Key box on the GitHub page.
8. Click “Add Key” on the GitHub page.
9. In the PuttyGen window, give your key a passphrase—a password you can remember; you’ll have to enter this password when you sync with the GitHub repository.
10. Save your public key and your private key. I like to put them in my user directory, in a folder called ssh. Just put them somewhere you won’t lose them.

## Setting up a Local Repository by Cloning the GitHub Repository

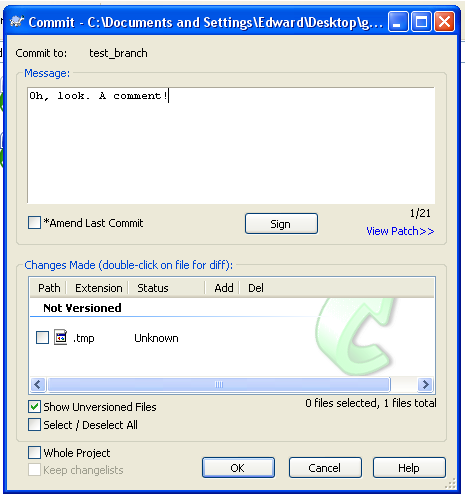
The first thing you’ll have to do is create a local repository. This is where all your code will live on your computer.

1. Create a folder where you want your code.
2. Right click your new folder and choose “Git Clone…”
3. For the URL, choose git@github.com:frc2399/2011-Robot-Code.git
4. Make sure “Load Putty Key” is checked. This is the private key you saved before. Find it with the “…” button.
5. Click Ok and enter your password. Say yes to the big long popup that asks if you want to cache a key.
6. You now have all of the files from the GitHub (and all of the history of those files) on your machine.

## Using Git/TortoiseGit

### Committing Your Changes

Most of the time, you’ll be working with your local repository. You should commit changes often so you don’t lose any work. To commit your changes to your local repository, just right click on the folder you created before and choose “Git Commit -> [branch]” ([branch] will be the name of whatever branch you’re working in). You’re required to put a comment for every commit. This helps describe what changes you made, which is a good way to help document your code.



When you make new files, you’ll have to add them to the repository when you commit them. New files are listed in the commit window under the heading “Not Versioned” (see above). Check the boxes for the files you want to add. If you don’t want a file to be part of your repository, then don’t check its box.

### Making Branches

If you want to start a new branch, right click in your folder and choose TortoiseGit->Create Branch… Name your branch and click Ok. You’ll need to create this branch on GitHub too, so right click and choose TortoiseGit->Push. This will push your new branch to the repository.

### Switching Branches/Getting New Branches from GitHub

If you’re working from more than one branch, you’ll need to switch back and forth between them. Right clicking in your folder and choose TortoiseGit->Checkout/Switch… Pick what branch you want to use. The choices include all your local branches and all the branches on GitHub (these will all start with “remotes/”). If you download a new branch from GitHub, TortoiseGit creates a new branch in your local repository.

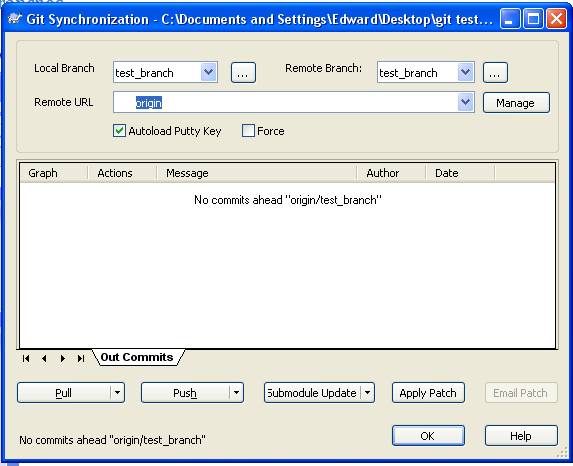
Remember to commit before switching branches! If you don’t, it won’t let you switch.

### Syncing With the GitHub Repository

In Git, you can “push” your changes to another repository or “pull” changes into your repository. For the most part, you’ll be pushing changes to your own branch. You can also pull changes if the version on GitHub is ahead of yours (like if you want to check out someone else’s branch).

To push and pull changes, use TortoiseGit’s syncing tool. You can open the Sync tool in the right click menu (it’s called “GitSync…”). The local branch should be the branch on your computer you want to sync. The remote branch should be the same branch on the remote (GitHub) repository. You may have to click the “…” to find the branch you want. Once you select both branches, you’ll get a summary of the commits that don’t match up between the two repositories. To push your changes to GitHub, click Push. To pull changes on GitHub into your repository, click Pull.

Remember that only changes you have *already* committed will be pushed!



### Merging

## Sources and Resources

* *Revision Control with Git for FRC Teams* by Austin Wright  
  Basically a longer, more in-depth version of this document.  
  <https://docs.google.com/Doc?id=dcz67k4q_42f7hzcwdg&pli=1>
* *Using LVMerge Utility with TortoiseSVN* by Jesse Batsche  
  Written for TortoiseSVN, but works for TortoiseGit. Lets you use National Instruments’ merge tools with TortoiseGit.  
  <http://www.dmcinfo.com/blog.aspx/articleType/ArticleView/articleId/136/Using-LVmerge-LabVIEW-Merge-Tool-with-TortoiseSVN.aspx>