Version Control and Successful Integration with Robot Code
Manan Shah
Los Altos Community Teams 14-0399

Version Control and Successful Integration with Robot Code

Abstract

Have you ever wished you could back up all of your programs into one simple system - one that is easily accessible from everywhere and includes tools to boost productivity and collaboration - to make your life as a programmer easier? The Los Altos Community Robotics Teams use a wonderful tool just for the aforementioned purpose. At the end of this paper, you will have learned the importance of version control along with valuable information regarding 1) setting up a version control system, 2) deploying key tools needed to organize and maintain a repository, and 3) using invaluable resources to better organize your project and to succeed in future programming challenges.

1 Why Version Control?

Version control is a management system that is used throughout multitudes of projects in order to keep track of code/version history and prevent terrible losses of code. There are three key reasons why using such a system is extremely important to both the organization of any teams' code and the quality of the robot's performance at a competition, each of which I will outline below.

1. **Code Management.** When working as a group, nothing is more important than effectively organizing and distributing tasks for programmers to accomplish. Furthermore, in order to integrate everyone's ideas and write one solid, cohesive program, one must conglomerate numerous and unique interpretations in an attempt to produce a final code version. Instead of a team member working on compiling everyone's ideas into one final document, version control allows for individuals to work on different portions of the same (let's say, `main`) program at the same time. Such a system eliminates the need of putting together different versions of the same program and allows for efficient collaboration between team members.

2. **Backing up Code.** Another key aspect of working as part of a group is managing previous versions of code. Imagine a scenario in which the primary programmer on your team has a terrible computer crash. Suddenly, all of the code that you have been working days and nights for is gone in a flash, without any way to save or back it up. Like documents on a personal computer, programs are very important to keep safe and properly maintained. A coherent version control system helps perform this seemingly difficult task, as it both registers versions of code that you have committed on an external server and logs every stage in the process.

3. **Keeping a Neat and Clean History.** When code breaks, as it often does, programmers (novice or experienced) often frantically hit the Control-Z button in order

to (hopefully) revert the error that was inadvertently created. As you may already notice, this constant process of writing code and attempting to revert it to fix errors is not an optimal procedure, one which can be fixed through the use of a version control system. Indeed, every time progress is made, programmers "commit" their changes to an external server, so that if there is an error in the code in the current version, it is extremely easy to revert to the last working version and continue working from there.

2 Background

While the term *version control* is a broad concept, I will begin by explaining the essence of how such systems save and restore your code. There are three major components in any such system: 1) your working disk, 2) your local repository, and 3) a separate remote repository.

The working disk is simply the current hard-drive that your uncommitted code resides on. Whenever you write code for your Botball robot, every file is saved on your local disk. The problem with this process is that the program is only located on one filesystem, which could cause many issues if that one system happens to break. Version control systems push the code on your working disk to both a local repository and a repository on their own web servers, where it will be safe in case any harm comes to your personal device.

In order to properly stage your commits - that is, keep track of your progress in order to potentially revert to a previous, working state, version control systems make use of a *local repository*. Every time you commit code (save a version or milestone of your project), the system keeps it in the local repository. While the code in this process is still preserved on your hard disk, any changes saved in the local repository are ready for the final stage of versioning whereas the working disk simply holds unfinished programs.

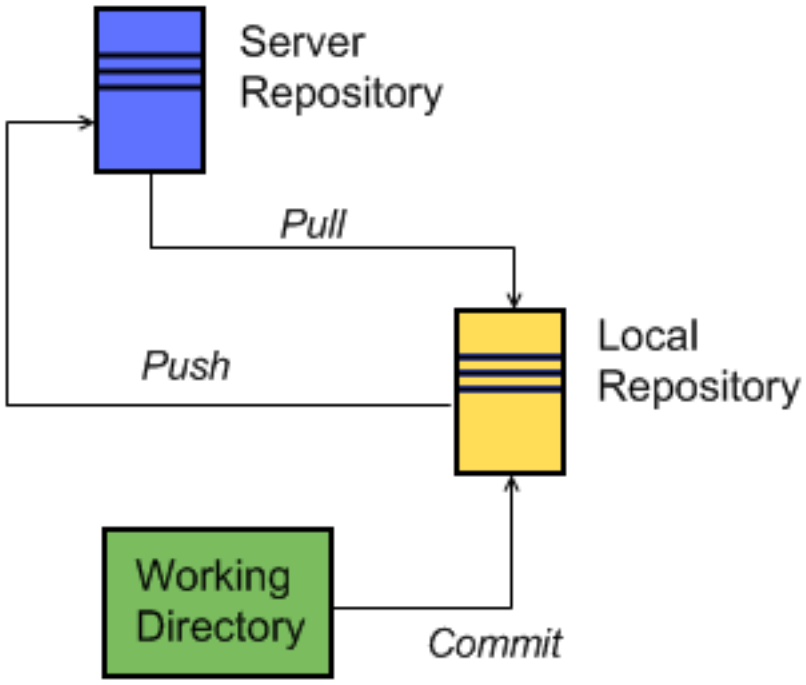
After deciding that your work is complete and ready to be shared with the rest of your team, code is `pushed` to a remote repository, from which it can be synchronized with those of your teammates. The primary advantage of such a system lies in the fact that your code is now stored on a separate filesystem. If any problems occur on the local computer, they can readily be restored from the server at the click of a button. Furthermore, as an extension of the remoteness of the system, version control allows multiple users to commit their changes to different parts of the same file, performing the merge automatically and providing an enhanced final version. Changes can always be reverted, if necessary.

Although this may seem like a complicated process, the steps are actually quite simple. Committing, pushing, and pulling are all key to the version control system and code organization.

3 Definitions and the Basics

After reading section one, you may be wondering *what* exactly committing, pushing, and pulling code is. Let me illustrate by means of a simple diagram. As can be seen from the

pictorial representation below, the three central locations - the server/remote repository, local repository, and your personal working directory - are tightly knit in order to back up and restore your work.



[1]

Committing a file indicates that you have added it to your local repository. In this process, you include a **snapshot** of an important change you have made. Version control systems have an index in which they store your file changes, and committing allows you to maintain a helpful history of useful file changes that have been completed.

Pushing a file implies that you would like to deploy the committed code "snapshots" to a remote repository - *i.e.* one that any individual with appropriate permissions can access. This is a valuable way to share code among your team.

Pulling a file indicates that you would like to update your working directory/local repository with the latest file changes made to the server. As this may conflict with the current changes that you are making to the document at hand, it may be required to manually pick-and-choose changes such that the correct version of code is saved.

The essential workflow requires a programmer to initially complete working on a certain version of code (on the hard disk). The programmer then adds the file to the local repository, where it is stored under the version control system. After this step is complete, s/he then commits the file(s) to show a snapshot of work that has been completed. Finally, the programmer pushes the changes to an external server, where they are recorded and "versioned" to prevent any harms from befalling the team.

4 Getting Started

Throughout this paper, I will be using a version control system known as `Git`. This system is heavily integrated with the GitHub website, which allows for both a traditional commit-push-pull system as well as handy project tools that are sure to benefit any BotBall team. In this part of the paper, I will be discussing how to get started with the basics of version control, and the most common mistakes made with these systems.

4.1 Downloading

Let's begin by downloading a Git client. If you are using a Windows or Mac operating system, the GitHub application can make your workflow much simpler, but if you're running on Solaris, Ubuntu or any other Linux Distribution, you can directly download Git from their homepage.

Ensure that `git` is added to your system PATH if you are on a PC before proceeding.

4.2 Setting Up

The next step after downloading Git is to set up an account on GitHub, which is a powerful integration tool that synchronizes with the git version control system and provides valuable tools to enhance your project. Go to the GitHub webpage (located at https://www.github.com) and register for an account. Your username must contain alphanumeric characters and a valid email address must also be provided.

4.3 Creating a Repository

You may either create a repository on the GitHub website, from the command line, or from the application itself. Let's begin by setting up a `Hello World` repository from the website. Clicking on the new repository button on the top right of your homepage should redirect you to a settings tab where you will select the name of your repository as well as a brief description.

For those of you more familiar with the git command line, you may create a repository and commit/push a README file using git commands in order to enhance speed and use more powerful tools than those that are available from the application.

Remember to choose a useful name for your repository (for example, robotcode2014 as opposed to thebestsupercodersteam2014).

4.4 Committing and Pushing

We have now reached a point where we have created a repository to store robot code online. Our next step is to `clone` the repository to our computer. Once again, open your git shell and now run the following command:

```
#Fill in {your SSH/HTTPS URL} with the clone URL github provides
# on the repository website
git clone {your SSH/HTTPS URL}
```

You now have a blank folder on your computer, with the same name as your repository, from which you can `push` and `pull` code.

4.4.1 Committing

As you may recall, the three main actions you can perform in a version control environment are committing, pushing, and pulling code. The first of these three steps is `committing`, a step in which you "send" a file from your working directory to your local repository. Let's begin by creating `helloworld.c` and saving it in the directory that you created through the previous set of commands.

Now that we have our sample code saved in the folder, which henceforth we will call the working directory, let's commit it to the local repository. Open your git shell, navigate to the specified directory, and run the following commands:

```
#initializes the repository
git init

#adds helloworld.c to our list of files to commit
git add helloworld.c

#commits helloworld.c to your local repository from your working directory
# -m "text here" is the commit message (identifying what you have done)
git commit -m "add helloworld.c"
```

4.4.2 Pushing

Next, let's push the committed `helloworld.c` to the server, and complete this process. In the git shell, run

```
#sets the remote repository to your newly created repository (note the .git extension)
git remote add origin https://github.com/username/repository name.git

#pushes your commits (namely "add helloworld.c") to the remote repository
git push -u origin master
```

and you're done! Check your online repository to see the file added and your changes made.

5 Summary and Additional Information

You've now learned the basics of version control! Of course, there are still numerous commands to learn and explore. A last major concept that I'd like to touch on is called `merging` commits. Essentially, when both you and a team member edit the same lines of the same file, the version control system does not know which edits to accept and which to reject. As the code is saved on a remote repository, when the version control system attempts to save conflicting versions of the file, problems occur. In this case, you may try running the

`git merge`

command, which lets git automatically merge the conflicting commits. However, when this is not the case, it will be necessary for you to manually fix the conflicting files (which will be marked by `git`) and push the changes. The conflicted area of your file will look something like the following:

```
#include <stdio.h>

int main(void) {
  <<<<<<< HEAD
  conflict - your teammate's version
  =======
  puts("Hello World!\n");
  >>>>>>> branch-x
}
```

This message signifies that your version of code differs from that on the server at a specific point (*i.e.* whereas you wrote `"Hello World"` in your working directory, your teammate committed a different version to the server). In order to correctly fix this error, simply remove the conflict markers (the `<<<HEAD`, `>>>branch-x`, and `===`) and add the version of code that seems most appropriate. Then, `git add` and `commit` the updated version, and you're good to go!

Summing up, version control systems are useful tools that allow for backing up programming files in a similar fashion that one backs up school/personal files on a hard disk. The three primary components used in such systems are *committing, pushing, and pulling* code. Each step of the process allows for a "snapshot" of current, working changes to be added to a repository, such that whenever

Hopefully this paper taught you the basics of version control, how to set up your personal environment, and keep track of changes to code. Please feel free to email me at `manan.shah.777@gmail.com` if you run into any problems or if you have additional questions.

Enjoy keeping your code backed up and using version control during next year's season!

References

[1] The Button Experiment,
http://thebuttonexperiment.com/Blog/wp-content/uploads/2013/06/diagram.png