Git Tutorial

Travis Westura

May 18, 2014

1 Introduction

This guide covers the basics of using Git and GitHub.

2 Creating a Basic Repository

This section covers two ways of starting a project that uses Git: first on your computer and second from GitHub. It then covers linking the two together.

2.1 Creating on Your Computer

From your computer create a new folder and navigate there using the git bash shell. Inside the folder, enter the command

```
git init
```

Alternatively you could use

```
git init <directory_name>
```

To create an initialized directory.

After that go to GitHub and create a repository, preferably with the same name as the folder you just made.

2.2 Creating from GitHub

Go to GitHub and create a repository. Then decide where you want the folder containing your project to be located on your computer, navigate there using the git bash terminal, and use the command

```
git clone <url of repository>
```

2.3 Linking your computer's repository with your GitHub repository

While inside the folder in the git bash terminal, run the command

```
git remote origin add <url of repository>
```

Now let's create a new file and then add it to the repository. To create the file type:

```
touch newfile.txt
```

Then

```
git add newfile.txt
```

Followed by

```
git commit -m "Add newfile.txt"
```

Finally

```
git push origin master
```

Refresh the GitHub webpage and you will see the new file in the repository.

Now lets make a change to the file on the repository and see how we can get this change to be reflected on your computer. First, open the file on your computer and edit it, by typing Hello, World! Then save it and add, commit, and push it. Check the file on GitHub to make sure it is changed.

Now click on the file on GitHub and click edit. Add some text to it, then click the green button below to commit, changing the default commit message if you wish by using the text box. Then use the command

```
git pull origin master
```

Check the text file on your computer, and it should be updated.

3 Dealing with Basic Merge Conflicts

This section deals with an example of multiple people working on one project and resolving branch conflicts. First we will create a project. Then we will make two separate edits to it to create a merge conflict. Then we will resolve this conflict.

3.1 Making Two Local Copies of a Project

We will create a project in GitHub and then clone it into two separate directories on your computer.

First go to GitHub and create a new project, perhaps titled BasicMerge. Using the GitHub click the create a new file here button, create ShoppingList.txt, and use the GitHub text editor to add a few lines of text to the file, perhaps of breakfast foods. For example

```
Shopping List:
Pancakes
Waffles
French Toast
Bread
Butter
Muffins
```

Commit the file and check GitHub to make sure that it is there. We will later be editing this from two separate directories in order to simulate two people working on a project and making conflicting edits.

Now we are going to create two copies of this project on your computer. Create two separate directories somewhere, perhaps titled One and Two. Next open two separate instances of your Git Bash shell. You should be able to do this by double-clicking the GitBash icon to open one shell, and then double-clicking the icon again to open another.

Now navigate to your One directory with one Git shell and to your Two directory with the other Git shell. Now, using the clone command we have already learned, clone your git repository into both of these directories. Recall the command has the form:

```
git clone <url of repository>
```

You should now be able to view a copy of the project in both of these folders.

3.2 Editing the Files

Let's start by making a change in GitHub and seeing it reflected in the two files. Let's change the second line from Pancakes to Chocolate Chip Pancakes using the GitHub editor. Commit this change, and then in both of your Git shells pull from the GitHub repository. Remember to first navigate the bash shell into the correct director and recall the command to pull is:

```
git pull origin master
```

You can now check the files in both directories to see the change reflected in them.

Now let's alter the file in directory one. Suppose we are really excited about waffles, so we then change the line Waffles to WAFFLES!!! Save this change, and we now want to push it to the repository. Recall we need to add, commit, and push:

```
git add ShopplingList.txt
git commit -m "Change Waffles"
git push origin master
```

Refresh GitHub to see this change reflected there as well. Next, we will pull this change to directory two, again using the second Git Bash shell and the command:

```
git pull origin master
```

Notice that the file is now updated in the second directory as well.

3.3 Dealing with a Merge Conflict

Now let's make some changes to the files that conflict with each other. Let's suppose that the person working in One does not want French Toast, but rather Buttered Toast. Meanwhile the person working in TWO prefers Burnt Toast, for some strange reason. Go ahead and make these changes, but don't add, commit, and push them immediately.

Now that the changes are saved go ahead and use add and commit in each shell, but don't push anything just yet. We are going to get our first merge conflict. Now, using the Git shell in one, push the changes to the repository. Check GitHub to see that the changes are made.

Next push using the other Git shell. Notice that something went wrong. You should get an error message such as the following

```
error: failed to push some refs to '<url>'
hint: Updates were rejected because the remote contains work
    that you do not have locally. This is usually caused by
    another repository pushing to the same ref. You may want to
    first integrate the remote changes (e.g., 'git pull ...')
    before pushing again.  See the 'Not about fast-forwards' in
    'git push --help' for details.
```

You can go an read through the note about fast-forwards if you want. Just enter the command

```
git push --help
```

and you will be taken to the website.

Let's take a look at what that error message is telling us. The remote, that is, our GitHub server, contains work that you do not have locally, that is, is not present in the file you are editing. The cause is another repository, the butter lover working in One, pushing to the same ref.

To fix this problem we want to begin by pulling from the repository. So use the pull command from the Git shell working in Two:

```
git pull origin master
```

You will see that the merge failed. Go and look at the shopping list file, and it should look like this:

```
Shopping List:
Chocolate Chip Pancakes
WAFFLES!!!
<<<<<<< HEAD
Burnt Toast
=======
Buttered Toast
>>>>>>> 70830fb399791beedcc403bdede1b02cd2a7e574
Bread
Butter
Muffins
```

The ugly mess of arrows points out (pun intended) where the conflict occurred. The first section, above the equal signs, is the code in your file. The second section, below the equal signs, is the code you just pulled. You fix the merge conflict, you want to edit the text. So suppose you decide that it would be better to have just plain Toast. Delete all of the messy arrows and fix that section of the file. It should not look like this:

```
Shopping List:
Chocolate Chip Pancakes
WAFFLES!!!
Toast
Bread
Butter
Muffins
```

Now we just want to add the file again, commit it once more, and push our updates:

```
git add ShoppingList.txt
git commit -m "Fix merge conflict now use plain old Toast"
git push origin master
```

Now use the Git shell in One to pull once more. Check that everything in One, Two, and GitHub looks the same.