Tools for Open Source

Git, GitHub, and more

Héctor M. de la Rosa Prado

Universidad Nacional Autónoma de México

April 26, 2019

Table of Contents

The Birth of Git

- In this course we will learn one of the greatest tools for open source development.

- In this course we will learn one of the greatest tools for open source development. Git

- In this course we will learn one of the greatest tools for open source development. Git
- We will also look at how we can contribute to open source helping in projects and making our own

- In this course we will learn one of the greatest tools for open source development. Git
- We will also look at how we can contribute to open source helping in projects and making our ownas well as giving an idea of how one should code in the open source community.

- Whenever someone mentions open source, the first thing that come to mind is the Linux Operating System.
- At this time, Linus worked on many projects developing Linux, this made it difficult to stay on track of his work.

- Git is a Version Control System created by Linus Torvalds to help him manage all of his projects.
- Since then, it has become the most popular Version Control System used today.

- Git is a Version Control System created by Linus Torvalds to help him manage all of his projects.
- Since then, it has become the most popular Version Control System used today.

Definition

Version Control (or Revision Control) is a tool that helps us manage the changes in a project.

Git Basics

The basic git commands we will be using will be the following:

Local	Branching	Remote
init	checkout	push
add	branch	pull
commit	merge	clone
log	reset	remote

The basic git commands we will be using will be the following:

Local	Branching	Remote
init	checkout	push
add	branch	pull
commit	merge	clone
log	reset	remote

There we will just learn the basic tools to get you started, but there are more advanced techniques that you can learn in a more advanced course.

Let's start by making a small toy project.

- Make a file called "HelloWorld.py" which askes for a name in input and prints a small greeting to that person.
- Once you are done, go to the bash terminal and enter git init.
- You should get the following message to know that everything went well: Initialized empty Git repository in 'FOLDER'

Clearly we don't want an empty project, so we need to add the file HelloWorld.py to the repository.

- Add a file or changes using 'git add HelloWorld.py'
- You can also add multiple changes by seperating with spaces
- Or add all changes by using 'git add .'
- To make sure your changes were add you can use 'git status'

An important not is, this file still isn't in our repository, all we did was 'stage' the changes, Now we have to make a commit.

- You can make a commit with 'git commit', but you also need to put a message related to your commit.
- This is why most people just 'git commit -m "message"', This will save you a bit of time and is much less confusing
- Save your new file and use the message 'Initial Commit'
- You can check your save history with 'git log'

Local Branching

So far, all we have is a faster terminal version of google docs for code. Although this is nice, git wouldn't be so popular with just these features.

What really makes git shine is how you can manage a large project with various people working on it at the same time.

Part of this is thanks to local and remote branches.

First lets start with local branching. We can see what branches a repo has with 'git branch'. For now we only have one which is named master. Lets make two new branches named 'branch1' and 'branch2'

- To make a new branch, all we have to do is 'git branch NEWBRANCH'
- If we check again we now have two branches. To move to the new branch we use 'git checkout NEWBRANCH'.
- For now they are the same, but later we can make some differences to them.
- These two steps can be joined with 'git checkout -b NEWBRANCH', the -b flag builds the branch before moving to it.

While in the new branches, make two files, one named sin.py and the secound named cos.py. Here you can make a function sinFunction(x:int) that returns the sin function of x and cosFunction(x:int) that return the cos function of x.

- In branch1 make these functions with numpy, and in branch2, use the math library.
- After that, go back to master and in HelloWorld.py import these two files and print sinFunction(0.5) and cosFunction(0.5).
- Make sure to commit all three of these branches with the message you like.

As you may have noticed, the new branch turns to a copy of what we already had in our previous branch. If we wanted to reset to a previous commit, we will need to reset.

- Resetting is simple, look up the commit hash code you wish to return to with 'git log' and type 'git reset f074bbb527b4adf9b93bcc931c4be132e77ed51b' (this is an example).
- When you reset, you have three options. hard mixed or soft.

- A soft reset returns your state but leaves the changes staged. So if you look at your files after a soft reset, nothing would have changed, but if you use 'git status' all of these changes will be recorded. You can redo these changes just by doing a commit.
- A mixed reset will leave the changes like a soft reset, but they will not be staged, meaning you would have to add then commit the changes before redoing the commit.
- A hard reset removes every change you had at the time. This means that all of your files will be exactly the same as when you had that exact commit. The changes you mad made will disappear from that branch and there is no way of getting it back. BE CAREFUL!

If we keep our branches seperated, they don't really do much, this is why we need to be able to join changes from one branch to another.

- Go back to master and use the following code to merge master with branch1: git merge branch1 -m 'math+hello –commit'
- This last flag saves us time commiting, or you can commit after the same way you did before.
- Now we have sen.py and cos.py in our master branch. If we look, branch1 still exists, but it hasn't been changed.
- This is great, now we can use branches to work on different features at the same time without having multiple feature bugs confusing our team.
- But there is one more detail...

If we have two branches working on the same file and decide to merge, git is smart enough to work out how to manage them... well most of the time. Sometimes if we modify the same line in both branches, we can get what is called a conflict. Thankfully git makes it easy enough to manage these.

- Now we will merge master with branch2, since we made changes to sen.py in both of them, we should get some conflicts.
- If we do it like before, we will get an error. We have to fix the conflicts. After git tries to merge, it will add both changes to the file.

- If there is a conflict it will mark the current branch with HEAD and the merging branch with its name, and the code will be in between.
- To solve the conflicts simply erase the code you didn't want and add the one you did. If you wanted both you can leave them both in the file.
- Leave numpy in one file and math in another, then finished the merge by staging and commiting these changes.

Remote Branches

The bread and butter of git has been the ability to share your code in pretty much the most efective way.

To do this there are websites like GitHub, BitBucket and GitLab that allow us to upload our projects in the internet. The most famous is GitHub, so it is what we will use here.

Go online and make your free account. If you want to make private repository you would need to pay or apply for a student account. For this you will need a university email address.

Now we have to learn how to add a project to a service like GitHub. For this we will use our sin/cos project. We can manage our remotes with git remote, to add a remote branch use
'git remote add origin https://github.com/holyfiddlex/temp.git'
'git remote add origin', makes a new remote called origin. and the site tells you where the remote can be found.

If we look at our GitHub, there haven't been any changes, that's because the remote git hasn't recorded any change.

- The first time you upload your changes you have to use 'git push -u origin master'
- The -u flag sets 'origin master' as the default push settings, meaning you push the master branch to the remote 'origin'
- Now you can check your git
- From now own you can use 'git push' to upload/download changes to and from origin. Make another change in the README and push.

Now we also need to know how to get involved in another project. For this you will need a friend.

- Go to you friends account and find his/her project for this tutorial.
- Copy the url and add '.git' at the end, it should look like this: 'https://github.com/USERNAME/PROJECTNAME.git'
- On a different folder (Not inside your project folder) use 'git clone https://github.com/USERNAME/PROJECTNAME.git'
- You now have your partners project in your computer, and you can make changes. But...
- Make a change and try to push it with 'git push', there will be an error

The problem we have is that your partner still doesn't have permissions to modify your project. This means we will have to change the permissions in the github project.

- Method 1: In your Project GitHub page, go to settings and then collaborators, you will need to input your password. Add your friend to collaborators.
- Your partner will need to accept an invite sent to their email. After that, they will have permissions to make changes.
- Method 2: Click on your friends project and click 'fork'. This will make a replica of the project in your account. Clone the forked project and upload the changes there (perferable in a different branch).
- After you have made all of the changes, you can make a Pull Request. This can get a bit more complicated and for now we will use Method 1.

Now, if one of our partners make a change in the project and uploads it to GitHub, how to I get those changes?

- Since we are on the same remote and we already configured the default from origin to master, we can just use 'git pull'
- If you have multiple remotes or if you want to pull to a branch that isn't master, you would have to use 'git pull remote branch'
- Pull the change that your partner made into you project.

Like I mentioned before, there are also Pull requests, and on top of that, there is Merge Requests but these will be left for another more advanced course. For now these are the basics.

GitHub Projects
(and the end)

There are many projects in GitHub, I recommend exploring the page and looking for places where you can make a change. This is the end of the tutorial, but feel free to:

- Start a project alone or with your friends. It can be a personal or school project.
- Help others by joining open source projects.
- Learn more about Git and open source project management!