Git and GitHub for beginners

Any important git-related words are **bolded**

In this tutorial we're going to simulate what it would be like working on a big, collaborative project. This will involve making changes to the code base, opening up a **pull request (PR)** and **merging** your code into the master branch.

Step 0: Send me your GitHub username

To push code changes to a GitHub repository, you must be listed as a collaborator on that repository. Please email me your GitHub username at diego@thebridge.tech so I can add you as a collaborator to the gittutorial repository.

Step 1: Cloning from a remote server to your local machine

The process of downloading a **repo** from a remote server to your local machine is known as **cloning**.

To clone a repo, first you need to copy the repo's URL in github. To do that, go to the clone or download green button and copy the URL.

Move to where you want to place the project on your local machine using the `cd` (change directory) command. For instance if you have a 'projects' folder on your desktop, you'd do something like `cd ~/Desktop/projects`

When you clone a repo, it downloads a brand new folder which contains all the files inside of it (so you don't need to make a specific folder for the project)

Once you're in the location you want to be, in the terminal, use the command:

```
git clone https://github.com/diego-thebridge/git101
```

It may prompt you to log in with your GitHub information.

Step 2: Checking out your project

Now that you have the project on your local machine, you can look at all the files and the changes that have been made to it. There's a couple things you can check out:

a. The history of the project. Using the command `git log` you can see a list of all the changes made in the project. For instance here is a project that only has one commit so far:

b. What branches exist in the project Using the command `git branch` you can see what branches a project contains

In the picture above you can see that this one project has three branches in it. The master branch (which is basically always there), and 'my-new-branch' and 'my-second-cool-feature'.

Step 3: Creating a new branch

You can create a new branch to base your changes off of. What this will do is make a pointer to the current commit you're out, an start basing your code from those changes.

Let's say you are on the master branch (what you should be automatically), and want to create a new feature. What you do is run `git checkout -b <my branch name>`.

Now, if someone makes more changes to the master branch after you've made our branch, your new branch won't know about the changes unless you **merge** those changes into your branch.

So let's try it out! Run `git checkout -b <your name>` What this does is say, "Hey git, I want to create a new branch (that's what the '-b' flag does) and then I want you to move me onto that branch (that's what the 'checkout' does)."

Afterwards you can use the `git branch` command to confirm that your branch really was created!

You should notice that you're now pointed towards your new branch as well.

Step 4: Adding a new file to the repo

Go ahead and add a new file to the project. You can use any text editor you like. Name it something like yourname.txt. Once you add or modify files in a folder that contains a git repo, git notices it. But it won't add the file to a commit unless you tell it to.

After creating the new file, you can use the `git status` command to see what files it knows about.

What this is saying above is, 'Hey, we noticed you created a new file mnelson.txt, but unless you use the ```git add`` command we aren't going to touch it.'

Step 5: Creating a commit

Now we're going to tell git that we do want it to care about the file!

Use the command `git add <your file name>`

What this does is put your file in the staging or index environment. It's saying, 'Hey, we're about to make a commit with this specific file'. This gives you the ability to not commit every single file you've changed.

If you run the `git status` command you'll see it looks a little different. Now it's saying that that given file is getting ready to be commited. To clarify, it has **not** yet been put into a commit, but it will be soon!

Now we'll make the commit! Run the command `git commit -m "Your message about the commit"`

The message you put at the end of the commit should be something related to what the commit contains - maybe it's a new feature, maybe it's a bug fix, maybe it's just fixing a typo. It shouldn't be something like "asdfadsf" or "foobar". That makes the other people who see your commit sad.

Step 6: Pushing a branch to the GitHub

Now we'll **push** the commit in your branch to GitHub. This allows other people to see your changes. If they're approved, then they'll be merged into the master branch.

To push into your new branch on GitHub, the command name is `git push origin yourbranchname`

GitHub will automatically create the branch for you on the server.

You may be wondering what that "origin" word is in the command above. What happens when you clone a remote repository to your local machine, git creates an **alias** for you. In nearly all cases this alias is **origin**. It's essentially a shorthand for that remote repository's URL. This means to push your changes to the remote repository, you could've used either the command: `git push git@github.com:git/git.git yourbranchname` or `git push origin yourbranchname`

You can check now on GitHub to see that the branch has been created. Go to the fron page of the repository: https://github.com/diegogl/git101

You may see a little note saying a branch with your name has just been pushed into the repository. You can also click the 'branches' link to see your branch listed there.

Now click the green button seen to the bottom right of that picture. We're going to make a **pull request**!

Step 7: Creating a Pull Request (PR)

A pull request is a way of alerting a repository's owners that you want to make some changes to their code. It allows them to review the code and make sure it looks good.

This is what the PR page looks like before you've submitted it:

This is what it looks like once you've submitted the PR request:

You may see that big green button at the bottom that says 'Merge pull request'. Hitting this button means you'll merge your changes into the master branch.

It's not always the case that the button will be green. In some cases you'll be faced with a **merge conflict** - this is when there's a change in one file that is different in another file, and git can't figure out which version to use. You'll have to manually go through and tell git which version to use.

As a side note, it's not nice to push a change directly to the master branch. In most cases you probably won't have the authorization to do that anyways.

Step 8: Merging a PR

Go ahead and press that green 'Merge pull request' button. This will cause your changes to go into the master branch.

When done, it's always nice to delete your branch (too many branches can get hard to follow), so hit that grey 'Delete branch' button as well.

You can double check that your commits actually got merged. Go to the first page of the repo (https://github.com/diegodl/git101) and click on the 'Commits' link.

From there you'll see a list of all the commits in that branch. You can see the one that I just merged right up top.

You can also see on the right hand side the hash code (SHA-1) of the commit. That's essentially a unique identifier for that specific commit. It's useful because it gives you the ability to refer to specific commits, and if you need to roll-back a change you made, you can use the `git revert (hash code number)` command.

Step 9: Almost done! Time to get your changes on GitHub back to your computer

Right now the repo on GitHub looks a little different than what you have on your local machine. For instance, the commit you made in your branch and merged into the master branch doesn't exist in the master branch on your local machine.

In order to get the most recent changes that you or others have merged on GitHub, use the `git pull origin master` command.

This shows you all the files that have changed and by how much.

Now we can use the `git log` command again and see all the new commits that happen.

(Note, you may need to switch branches back to the master branch. You can do that using the `git checkout master` command).

Step 10: Basking in your git glory

Great job!

Basado en : Tutorial on git and GitHub for beginners, designed for the Women Who Code meetup held on August 12, 2015 at HubSpot.