HubSpot | Product

HOME

JOBS

BLOG

OPEN SOURCE

HUBSPOT PRODUCT AND ENGINEERING BLOG

An Intro to Git and GitHub for Beginners (Tutorial)

OCT 1, 2015 / BY MEGHAN NELSON

Tweet   Share   Like 153   Share

In August, we hosted a Women Who Code meetup at HubSpot and led a workshop for beginners on using git and GitHub. I first walked through a slide presentation on the basics and background of git and then we broke out into groups to run through a tutorial I created to simulate working on a large, collaborative project. We got feedback after the event that it was a helpful, hands-on introduction. So if you're new to git, too, follow the steps below to get comfortable making changes to the code base, opening up a pull request (PR), and merging code into the master branch. Any important git and GitHub terms are in bold with links to the official git reference materials.

Step 0: Install git and create a GitHub account

The first two things you'll want to do are install git and create a free GitHub account.

Follow the instructions here to install git (if it's not already installed). Note that for this tutorial we will be using git on the command line only. While there are some great git GUIs (graphical user interfaces), I think it's easier to learn git using git-specific commands first and then to try out a git GUI once you're more comfortable with the command.

Once you've done that, create a GitHub account here. (Accounts are free for public repositories, but there's a charge for private repositories.)

Step 1: Create a local git repository

When creating a new project on your local machine using git, you'll first create a new **repository** (or often, **'repo'**, for short).

To use git we'll be using the terminal. If you don't have much experience with the terminal and basic commands, check out this tutorial (especially the 'Navigating the Filesystem' and 'Moving Around' sections).

To begin, open up a terminal and move to where you want to place the project on your local machine using the cd (change directory) command. For example, if you have a 'projects' folder on your desktop, you'd do something like:

```
mnelson:Desktop mnelson$ cd ~/Desktop
mnelson:Desktop mnelson$ mkdir myproject
mnelson:Desktop mnelson$ cd myproject/
```

terminalcd.md hosted with ♡ by **GitHub**   **view raw**

To initialize a git repository in the root of the folder, run the **git init** command:

```
mnelson:myproject mnelson$ git init
Initialized empty Git repository in /Users/mnelson/Desktop/myproject/.git/
```

gitinit.md hosted with ♡ by **GitHub**   **view raw**

Step 2: Add a new file to the repo

Go ahead and add a new file to the project, using any text editor you like or running a `touch` command.

Once you've added or modified files in a folder containing a git repo, git will notice that changes have been made inside the repo. But, git won't officially keep track of the file (that is,

put it in a commit - we'll talk more about commits next) unless you explicitly tell it to.

```
mnelson:myproject mnelson$ touch mnelson.txt
mnelson:myproject mnelson$ ls
mnelson.txt
```

addfile.md hosted with ♡ by **GitHub**   view raw

After creating the new file, you can use the `git status` command to see which files git knows exist.

```
mnelson:myproject mnelson$ git status
On branch master

Initial commit

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        mnelson.txt

nothing added to commit but untracked files present (use "git add" to track)
```

gitstatus.md hosted with ♡ by **GitHub**   view raw

What this basically says is, "Hey, we noticed you created a new file called mnelson.txt, but unless you use the '`git add`' command we aren't going to do anything with it."

An interlude: The staging environment, the commit, and you

One of the most confusing parts when you're first learning git is the concept of the staging environment and how it relates to a commit.

A **commit** is a record of what files you have changed since the last time you made a commit. Essentially, you make changes to your repo (for example, adding a file or modifying one) and then tell git to put those files into a commit.

Commits make up the essence of your project and allow you to go back to the state of a project at any point.

So, how do you tell git which files to put into a commit? This is where the **staging environment** or **index** come in. As seen in Step 2, when you make changes to your repo, git notices that a file has changed but won't do anything with it (like adding it in a commit).

To add a file to a commit, you first need to add it to the staging environment. To do this, you can use the `git add <filename>` command (see Step 3 below).

Once you've used the git add command to add all the files you want to the staging environment, you can then tell git to package them into a commit using the `git commit` command.

Note: The staging environment, also called 'staging', is the new preferred term for this, but you can also see it referred to as the 'index'.

Step 3: Add a file to the staging environment

Add a file to the staging environment using the `git add` command.

If you rerun the git status command, you'll see that git has added the file to the staging environment (notice the "Changes to be committed" line).

```
mnelson:myproject mnelson$ git status
On branch master

Initial commit

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

        new file:   mnelson.txt
```

addtostaging.md hosted with ♡ by **GitHub**   view raw

To reiterate, the file has **not** yet been added to a commit, but it's about to be.

Step 4: Create a commit

It's time to create your first commit!

Run the command `git commit -m "Your message about the commit"`

```
mnelson:myproject mnelson$ git commit -m "This is my first commit!"
[master (root-commit) b345d9a] This is my first commit!
 1 file changed, 1 insertion(+)
 create mode 100644 mnelson.txt
```

commit.md hosted with ♡ by **GitHub**   view raw

The message 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. Don't put a message like "asdfadsf" or "foobar". That makes the other people who see your commit sad. Very, very, sad.

Step 5: Create a new branch

Now that you've made a new commit, let's try something a little more advanced.

Say you want to make a new feature but are worried about making changes to the main project while developing the feature. This is where **git branches** come in.

Branches allow you to move back and forth between 'states' of a project. For instance, if you want to add a new page to your website you can create a new branch just for that page without affecting the main part of the project. Once you're done with the page, you can **merge** your changes from your branch into the master branch. When you create a new branch, Git keeps track of which commit your branch 'branched' off of, so it knows the history behind all the files.

Let's say you are on the master branch and want to create a new branch to develop your web page. Here's what you'll do: Run `git checkout -b <my branch name>`. This command will automatically create a new branch and then 'check you out' on it, meaning git will move you to that branch, off of the master branch.

After running the above command, you can use the `git branch` command to confirm that your branch was created:

```
mnelson:myproject mnelson$ git branch
  master
* my-new-branch
```

gitbranch.md hosted with ♡ by **GitHub**   view raw
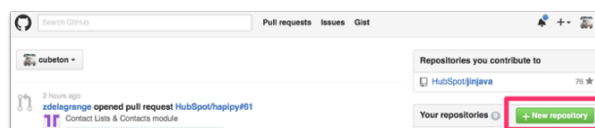
The branch name with the asterisk next to it indicates which branch you're pointed to at that given time.

Now, if you switch back to the master branch and make some more commits, your new branch won't see any of those changes until you **merge** those changes onto your new branch.
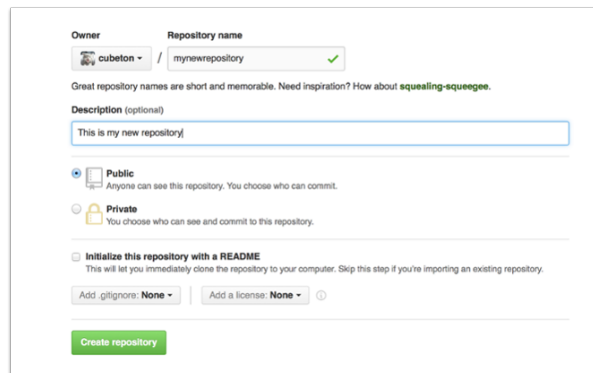
Step 6: Create a new repository on GitHub

If you only want to keep track of your code locally, you don't need to use GitHub. But if you want to work with a team, you can use GitHub to collaboratively modify the project's code.

To create a new repo on GitHub, log in and go to the GitHub home page. You should see a green '+ New repository' button:

After clicking the button, GitHub will ask you to name your repo and provide a brief description:

When you're done filling out the information, press the 'Create repository' button to make your new repo.

GitHub will ask if you want to create a new repo from scratch or if you want to add a repo you have created locally. In this case, since we've already created a new repo locally, we want to push that onto GitHub so follow the **'....or push an existing repository from the command line'** section:

```
mnelson:myproject mnelson$ git remote add origin https://github.com/cubeton/mynewrepository.git
mnelson:myproject mnelson$ git push -u origin master
Counting objects: 3, done.
Writing objects: 100% (3/3), 263 bytes | 0 bytes/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To https://github.com/cubeton/mynewrepository.git
 * [new branch]      master -> master
Branch master set up to track remote branch master from origin.
```

addgithub.md hosted with ♡ by **GitHub**   view raw

 (You'll want to change the URL in the first command line to what GitHub lists in this section since your GitHub username and repo name are different.)

Step 7: Push a branch to GitHub

Now we'll **push** the commit in your branch to your new GitHub repo. This allows other people to see the changes you've made. If they're approved by the repository's owner, the changes can then be merged into the master branch.

To push changes onto a new branch on GitHub, you'll want to run `git push` origin `yourbranchname`. GitHub will automatically create the branch for you on the remote repository:

```
mnelson:myproject mnelson$ git push origin my-new-branch
Counting objects: 3, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 313 bytes | 0 bytes/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To https://github.com/cubeton/mynewrepository.git
 * [new branch]      my-new-branch -> my-new-branch
```

addnewbranchgithub.md hosted with ♡ by **GitHub**   view raw

You might be wondering what that "origin" word means in the command above. What happens is that when you clone a remote repository to your local machine, git creates an **alias** for you. In nearly all cases this alias is called "**origin**." It's essentially shorthand for the remote repository's URL. So, 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`

(If this is your first time using GitHub locally, it might prompt you to log in with your GitHub username and password.)

If you refresh the GitHub page, you'll see 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.
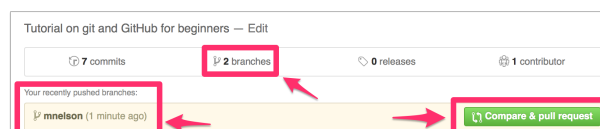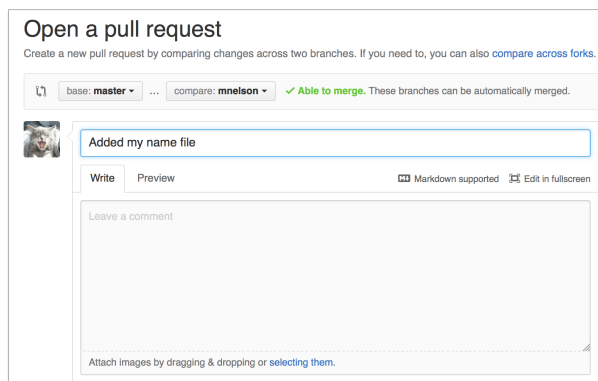

Now click the green button in the screenshot above. We're going to make a **pull request**!
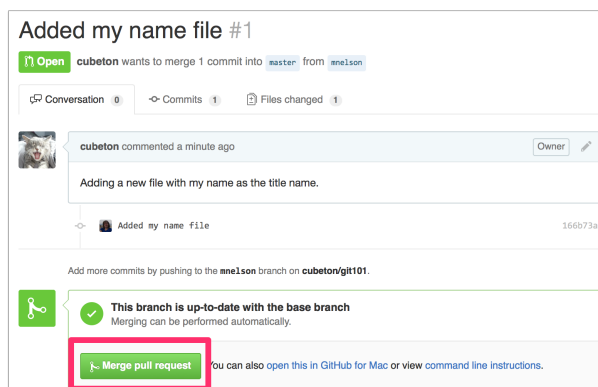
Step 8: Create a Pull Request (PR)

A pull request (or PR) is a way to alert a repo'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 before putting your changes on the master branch.

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

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

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

Note that this button won't always be green. In some cases it'll be grey, which means you're faced with a **merge conflict**. This is when there is a change in one file that conflicts with a change in another file and git can't figure out which version to use. You'll have to manually go in and tell git which version to use.

Sometimes you'll be a co-owner or the sole owner of a repo, in which case you may not need to create a PR to merge your changes. However, it's still a good idea to make one so you can keep a more complete history of your updates and to make sure you always create a new branch when making changes.

Step 9: Merge a PR

Go ahead and click the green 'Merge pull request' button. This will merge your changes into the master branch.
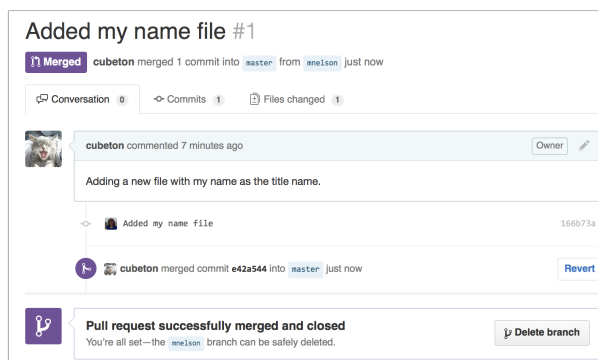
When you're done, I recommend deleting your branch (too many branches can become messy), so hit that grey 'Delete branch' button as well.

You can double check that your commits were merged by clicking on the 'Commits' link on the first page of your new repo.
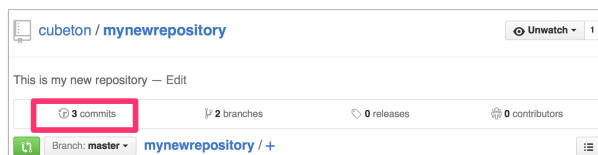
This will show you a list of all the commits in that branch. You can see the one I just merged right up top (Merge pull request #2).

You can also see the **hash code** of the commit on the right hand side. A hash code is a unique identifier for that specific commit. It's useful for referring to specific commits and when undoing changes (use the `git revert` `<hash code number>` command to backtrack).

Step 10: Get changes on GitHub back to your computer
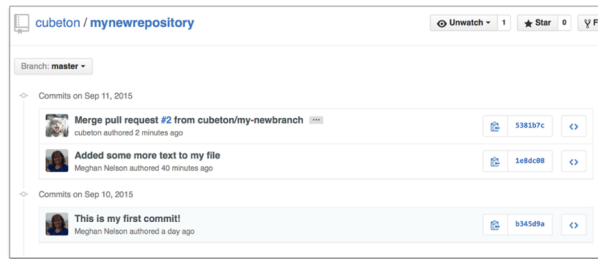
Right now, the repo on GitHub looks a little different than what you have on your local machine. For example, the commit you made in your branch and merged into the master branch doesn't exist in the master branch on your local machine.

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

```
mnelson:myproject mnelson$ git pull origin master
remote: Counting objects: 1, done.
remote: Total 1 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (1/1), done.
From https://github.com/cubeton/mynewrepository
 * branch            master      -> FETCH_HEAD
   b345d9a..5381b7c  master      -> origin/master
Merge made by the 'recursive' strategy.
 mnelson.txt | 1 +
 1 file changed, 1 insertion(+)
```

pulloriginmaster.md hosted with ♡ by **GitHub**   view raw

This shows you all the files that have changed and how they've changed.

Now we can use the `git log` command again to see all new commits.

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

```
mnelson:myproject mnelson$ git log
commit 3e270876db0e5ffd3e9bfc5edede89b64b83812c
Merge: 4f1cb17 5381b7c
Author: Meghan Nelson <mnelson@hubspot.com>
Date:   Fri Sep 11 17:48:11 2015 -0400

    Merge branch 'master' of https://github.com/cubeton/mynewrepository

commit 4f1cb1798b6e6890da797f98383e6337df577c2a
Author: Meghan Nelson <mnelson@hubspot.com>
Date:   Fri Sep 11 17:48:00 2015 -0400

    added a new file

commit 5381b7c53212ca92151c743b4ed7dde07d9be3ce
Merge: b345d9a 1e8dc08
Author: Meghan Nelson <meghan@meghan.net>
Date:   Fri Sep 11 17:43:22 2015 -0400

    Merge pull request #2 from cubeton/my-newbranch

    Added some more text to my file

commit 1e8dc0830b4db8c93efd80479ea886264768520c
Author: Meghan Nelson <mnelson@hubspot.com>
Date:   Fri Sep 11 17:06:05 2015 -0400

    Added some more text to my file

commit b345d9a25353037afdeaa9fcaf9f330effd157f1
Author: Meghan Nelson <mnelson@hubspot.com>
Date:   Thu Sep 10 17:42:15 2015 -0400

    This is my first commit!
```

gitlogaftermerge.md hosted with ♡ by **GitHub**   view raw

Step 11: Bask in your git glory

You've successfully made a PR and merged your code to the master branch. Congratulations! If you'd like to dive a little deeper, check out the files in this Git101 folder for even more tips and tricks on using git and GitHub.

I also recommend finding some time to work with your team on simulating a smaller group project like we did here. Have your team make a new folder with your team name, and add

some files with text to it. Then, try pushing those changes to this remote repo. That way, your team can start making changes to files they didn't originally create and practice using the PR feature. And, use the git blame and git history tools on GitHub to get familiar with tracking which changes have been made in a file and who made those changes.

The more you use git, the more comfortable you'll... git with it. (I couldn't resist.)

Written by Meghan Nelson

SUBSCRIBE FOR UPDATES

Email

HubSpot Product needs the contact information you provide to us to contact you about our products and services. You may unsubscribe from these communications at anytime. For information on how to unsubscribe, as well as our privacy practices and commitment to protecting your privacy, check out our Privacy Policy.

SUBSCRIBE

TOPICS

PRODUCT ››

ENGINEERING ››

DESIGN ››
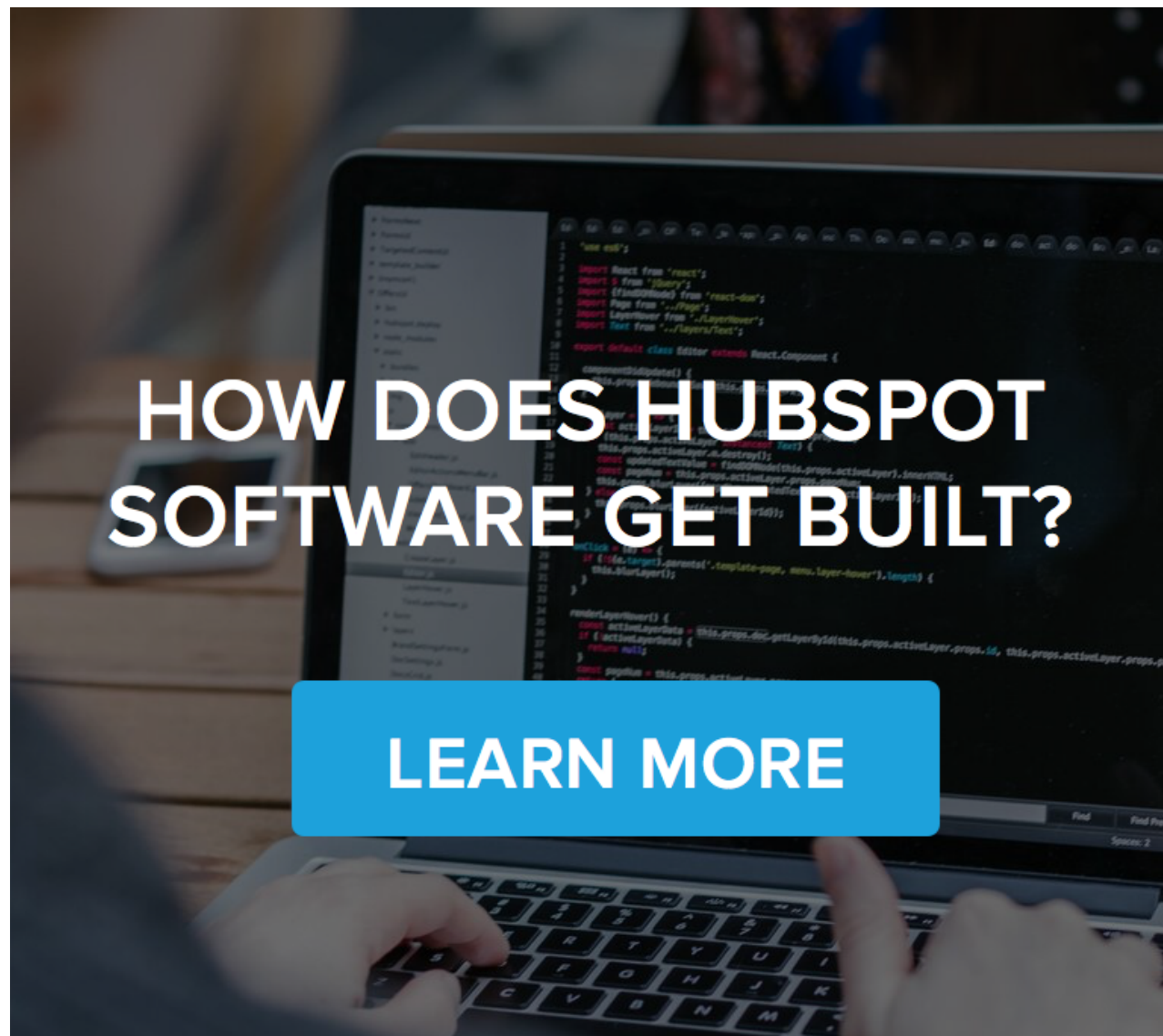
PRODUCT MANAGEMENT ››

UX ››

CULTURE ››

CAREER GROWTH ››

Sam 2015/10/1 下午9:34:59

Very informative post! Thanks!

REPLY TO SAM

June 2015/12/13 上午8:14:56

Thanks for the tutorial, but I have one question. I've done to step 8 (Step 8: Create a Pull Request (PR), but the github page could not find any changes, it says "master and my-branch-name are identical." I tried several times but still couldn't find the reason. I don't know where did I do wrong, anyone has the same problem?

REPLY TO JUNE

Meghan Nelson 2015/12/14 上午10:04:26

Hi June! I'm glad you're going through the tutorial. It sounds like what's happening is you've created your new branch on GitHub, but you haven't made changes (commits) on that branch or pushed your local changes (commits) onto GitHub.

What you can do is:

1. Make sure you're working on the correct branch (use the 'git branch' command). There should be an asterisk next to your branch name you're currently working on.
2. Make sure you've made commits on your branch. You can use the 'git status' command to see if there are any unsaved changes (You want it to say 'On branch my-branch-name nothing to commit, working directory clean'). You can use the 'git log' command to make sure that you have made commits on that branch. You'll need to make at least one commit for there to be differences to appear.
3. Finally once you're sure you've made a commit on that branch, try redoing the 'git push origin my-branch-name' command and refresh the page on GitHub. You should then your changes listed in your branch on GitHub.

I hope this helps, let me know if you run into any problems!

REPLY TO MEGHAN NELSON

June 2015/12/14 下午5:24:09

Hi Meghan,
Thanks for the answer. Yesterday I tried it again and finally could make the git push command!
I'm completely new in coding, have just studied it for a few months, and finding something really easy to follow like your tutorial is such a great happiness.
Thanks so much!

REPLY TO JUNE

Michael Owen 2015/12/30 上午4:50:48

Thanks for the answer. Yesterday I tried it again and finally could make the git push command!
I'm completely new in coding, have just studied it for a few months, and finding something

really easy to follow like your tutorial is such a great happiness.

Thanks so much!

Thanks for the answer. Yesterday I tried it again and finally could make the git push command! I'm completely new in coding, have just studied it for a few months, and finding something really easy to follow like your tutorial is such a great happiness.

Thanks so much!

REPLY TO MICHAEL OWEN

Kristin Day 2016/5/31 上午6:05:20

This was so helpful! Thank you so much for posting!!

REPLY TO KRISTIN DAY

Virendra More 2016/8/14 上午6:19:51

Can I configure remote repository with corporate domain user accounts?

REPLY TO VIRENDRA MORE

Jesse Lind 2019/11/29 下午1:21:52

Hello. Great post--SO HELPFUL!!! :) I was floundering until I read it. You have the gift of teaching! Anyways...I followed the above tutorial and merged the branches on gitHub; however, once I ran `git pull origin master` from the command line, the extra branch (`test-branch`) that was merged into master on gitHub is still present on my local machine. Is that expected behavior? I figured the `git pull` command would cause my local machine to mirror what's in my gitHub repo.

REPLY TO JESSE LIND

Pankaj Sharma 2019/12/3 上午12:55:59

Simply awesome . very simple and beautiful.

REPLY TO PANKAJ SHARMA

Mn Mm, 2019/12/14 上午9:54:44

m

REPLY TO MN MM,

Amalendu Kar 2019/12/19 上午11:17:06

Thanks, awesome tutorial

REPLY TO AMALENDU KAR

Carter 2019/12/29 下午9:22:36

Thank you so much! Guides like this are a lifesaver for absolute beginners like me.

REPLY TO CARTER

Joseph McCarron 2020/1/5 下午5:38:32

The reason why people a having the problem with the identical file thing is that after creating the branch the file was not changed so it was still identical to the master branch

REPLY TO JOSEPH MCCARRON

Ndip Des 2020/1/6 下午11:57:00

how can a view the side i just push in the browser

REPLY TO NDIP DES

First Name*

Last Name

Email*

Website URL

Comment*

We're committed to your privacy. HubSpot uses the information you provide to us to contact you about our relevant content, products, and services. You may unsubscribe from these communications at any time. For more information, check out our Privacy Policy.

protected by **reCAPTCHA**
Privacy - Terms

SUBMIT COMMENT

What is HubSpot | Our Story | Our Products | Culture Code

Facebook | Twitter | Instagram