TUTORIAL

How To Create a Pull Request on GitHub

Git Development Open Source

By Lisa Tagliaferri

Last Validated on October 13, 2018 • Originally Published on September 23, 2016 398k

English

Introduction

Free and open source, Git is a distributed version control system that makes collaborative software projects more manageable. Many projects maintain their files in a Git repository, and sites like GitHub have made sharing and contributing to code simple, valuable, and effective.

Open-source projects that are hosted in public repositories benefit from contributions made by the broader developer community through pull requests, which request that a project accept changes you have made to its code repository.

This tutorial will guide you through making a pull request to a Git repository through the command line so that you can contribute to open-source software projects.

Prerequisites

You should have Git installed on your local machine. You can check if Git is installed on your computer and go through the installation process for your operating system by following this guide.

You'll also need to have or create a GitHub account. You can do so through the GitHub website

A **repository**, or **repo** for short, is essentially the main folder of a project. The repository contains all the relevant project files, including documentation, and also stores the revision history for each

Fork the Repository

You can fork a repository on GitHub by navigating with your browser to the GitHub URL of the open-source project you would like to contribute to.

GitHub repository URLs will reference both the username associated with the owner of the repository, as well as the repository name. For example, DigitalOcean Community is the owner of the cloud_haiku project repository, so the GitHub URL for that project is:

```
https://github.com/do-community/cloud_haiku
```

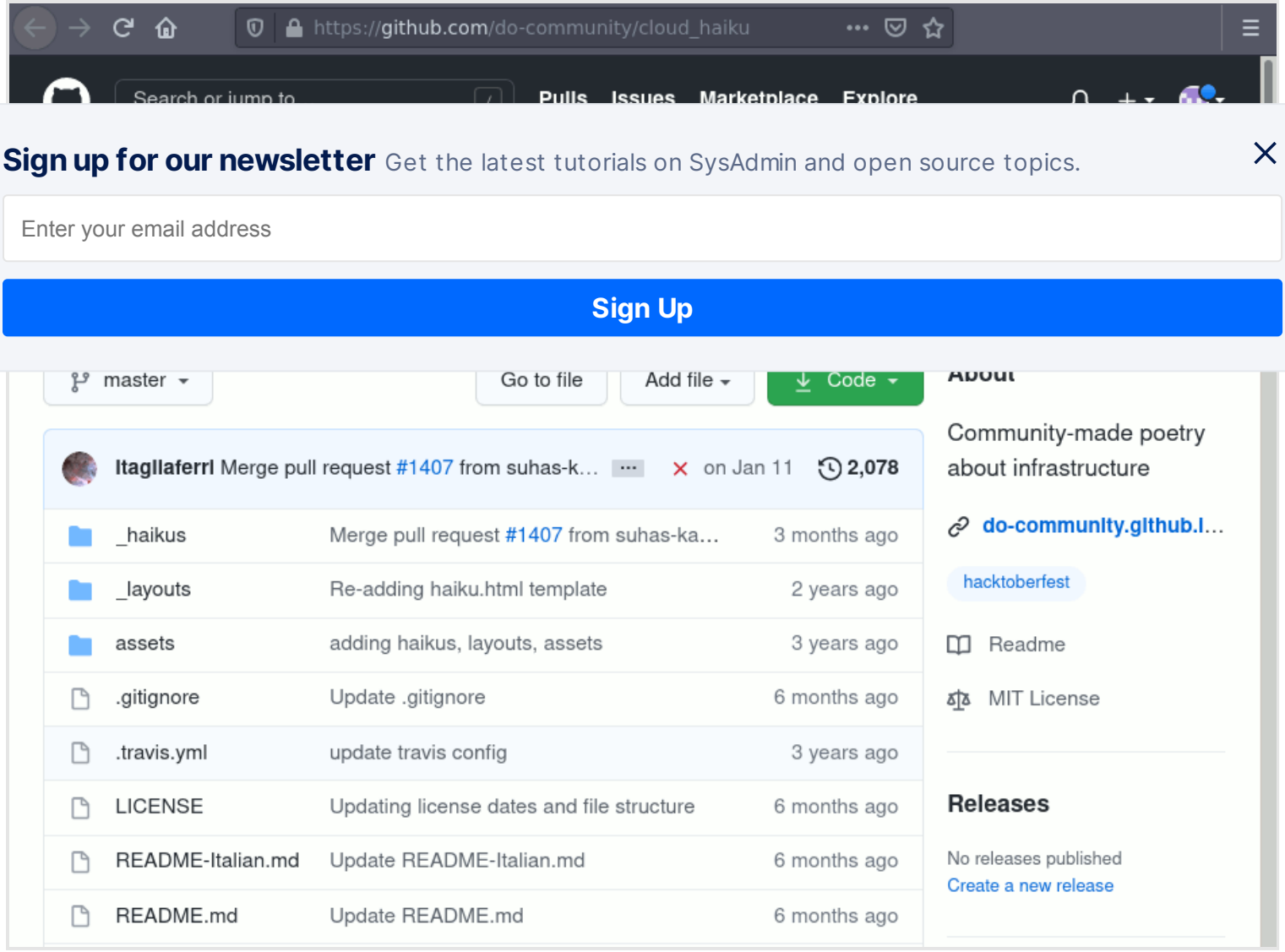
In the above example, **do-community** is the username and **cloud_haiku** is the repository name.

Once you have identified the project you would like to contribute to, you can navigate to the URL, which will be formatted like so:

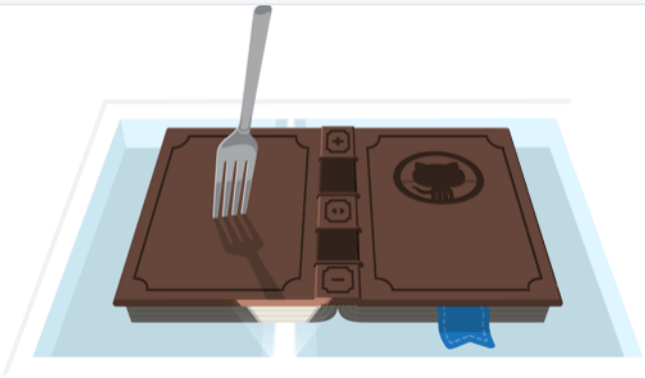
```
https://github.com/username/repository
```

Or you can search for the project using the GitHub search bar.

When you're on the main page for the repository, you'll see a "Fork" button on your upper right-hand side of the page, underneath your user icon:



Click on the fork button to start the forking process. Within your browser window, you'll receive feedback that looks like this:



Once the process is done, your browser will go to a screen similar to the repository image above, except that at the top you will see your username before the repository name, and in the URL it will also say your username before the repository name.

So, in the example above, instead of **do-community / cloud_haiku** at the top of the page, you'll see **your-username / cloud_haiku**, and the new URL will look like this:

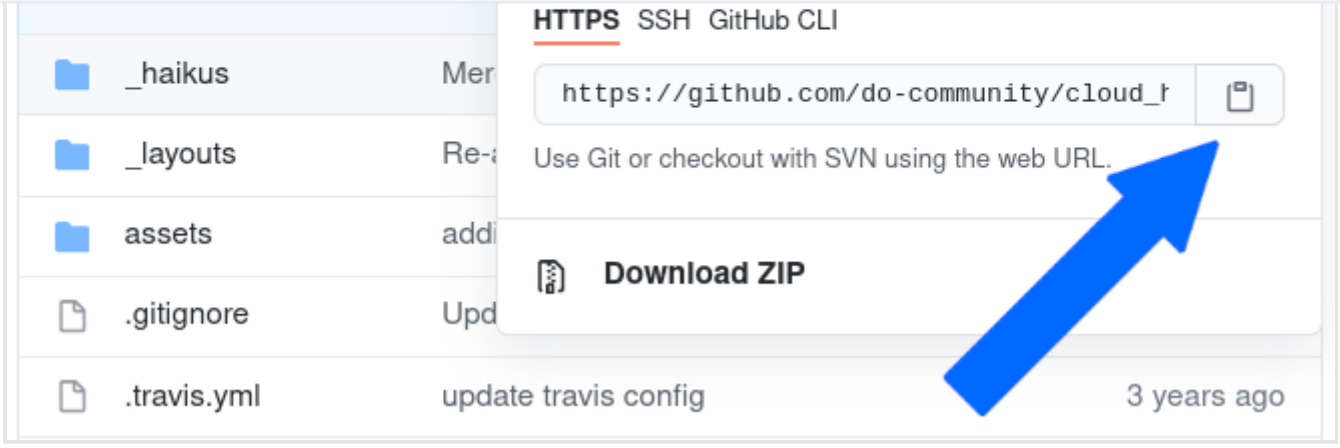
```
https://github.com/your-username/cloud_haiku
```

With the repository forked, you're ready to clone it so that you have a local working copy of the code base.

Clone the Repository

To make your own local copy of the repository you would like to contribute to, let's first open up a terminal window.

You can alternatively copy the URL by using the green "⤓ **Code**" button from your repository page that you just forked from the original repository page. Once you click the button, you'll be able to



Once we have the URL, we're ready to clone the repository. To do this, we'll combine the `git clone` command with the repository URL from the command line in a terminal window:

```
$ git clone https://github.com/your-username/repository.git
```

Now that we have a local copy of the code, we can move on to creating a new branch on which to work with the code.

Create a New Branch

Whenever you work on a collaborative project, you and other programmers contributing to the repository will have different ideas for new features or fixes at once. Some of these new features will not take significant time to implement, but some of them will be ongoing. Because of this, it is important to branch the repository so that you are able to manage the workflow, isolate your code, and control what features make it back to the main branch of the project repository.

The primary branch of a project repository is usually called the **main** branch. A common best practice is to consider anything on the main branch as being deployable for others to use at any

To create our branch, from our terminal window, let's change our directory so that we are working in the directory of the repository. Be sure to use the actual name of the repository (such as

descriptively so that others working on the project understand what you are working on.

```
$ git branch new-branch
```

Now that our new branch is created, we can switch to make sure that we are working on that branch by using the `git checkout` command:

```
$ git checkout new-branch
```

Once you enter the `git checkout` command, you will receive the following output:

```
Output
Switched to branch 'new-branch'
```

Alternatively, you can condense the above two commands, creating and switching to a new branch, with the following command and `-b` flag:

```
$ git checkout -b new-branch
```

If you want to switch back to main, you'll use the `checkout` command with the name of the main branch:

```
$ git checkout main
```

The `checkout` command will allow you to switch between multiple branches, so you can potentially work on multiple features at once.

file to this command:

With our file staged, we'll want to record the changes that we made to the repository with the `git commit` command.

The **commit message** is an important aspect of your code contribution; it helps the other contributors fully understand the change you have made, why you made it, and how significant it is. Additionally, commit messages provide a historical record of the changes for the project at large, helping future contributors along the way.

If we have a very short message, we can record that with the `-m` flag and the message in quotes:

```
$ git commit -m "Fixed documentation typos"
```

But, unless it is a very minor change, we will more than likely want to include a lengthier commit message so that our collaborators are fully up to speed with our contribution. To record this larger message, we will run the `git commit` command which will open the default text editor:

```
$ git commit
```

If you would like to configure your default text editor, you can do so with the `git config` command, and set nano as the default editor, for example:

```
$ git config --global core.editor "nano"
```

Or vim:

```
# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
```


Underneath the introductory comments, you should add the commit message to the text file.

To write a useful commit message, you should include a summary on the first line that is around 50 characters long. Under this, and broken up into digestible sections, you should include a description that states the reason you made this change, how the code works, and additional information that will contextualize and clarify it for others to review the work when merging it. Try to be as helpful and proactive as possible to ensure that those maintaining the project are able to fully understand your contribution.

Once you have saved and exited the commit message text file, you can verify what git will be committing with the following command:

```
$ git status
```

Depending on the changes that you have made, you will receive output that looks something like this:

```
Output
On branch new-branch
Your branch is ahead of 'origin/new-branch' by 1 commit.
  (use "git push" to publish your local commits)
nothing to commit, working directory clean
```

At this point you can use the `git push` command to push the changes to the current branch of your forked repository:

```
$ git push --set-upstream origin new-branch
```

```
Counting objects: 3, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (2/2), done.
```


You can now navigate to the forked repository on your GitHub webpage and toggle to the branch you just pushed to see the changes you have made in-browser.

At this point, it is possible to make a pull request to the original repository, but if you have not already done so, you'll want to make sure that your local repository is up-to-date with the upstream repository.

Update Local Repository

While you are working on a project alongside other contributors, it is important for you to keep your local repository up-to-date with the project as you don't want to make a pull request for code that will cause conflicts. To keep your local copy of the code base updated, you'll need to sync changes.

We'll first go over configuring a remote for the fork, then syncing the fork.

Configure a Remote for the Fork

Remote repositories make it possible for you to collaborate with others on a Git project. Each remote repository is a version of the project that is hosted on the Internet or a network you have access to. Each remote repository should be accessible to you as either read-only or read-write, depending on your user privileges.

In order to be able to sync changes you make in a fork with the original repository you're working with, you need to configure a remote that references the upstream repository. You should set up the remote to the upstream repository only once.

Let's first check which remote servers you have configured. The `git remote` command will list whatever remote repository you have already specified, so if you cloned your repository as we did above, you'll at least see the origin repository, which is the default name given by Git for the cloned

```
$ git remote -v
```


If you have previously set up more than one remote, the `git remote -v` command will provide a list of all of them.

Next, we'll specify a new remote upstream repository for us to sync with the fork. This will be the original repository that we forked from. We'll do this with the `git remote add` command.

```
$ git remote add upstream https://github.com/original-owner-username/original-repositor
```

In this example, `upstream` is the shortname we have supplied for the remote repository since in terms of Git, "upstream" refers to the repository that we cloned from. If we want to add a remote pointer to the repository of a collaborator, we may want to provide that collaborator's username or a shortened nickname for the shortname.

We can verify that our remote pointer to the upstream repository was properly added by using the `git remote -v` command again from the repository directory:

```
$ git remote -v
```

```
Output
origin    https://github.com/your-username/forked-repository.git (fetch)
origin    https://github.com/your-username/forked-repository.git (push)
upstream    https://github.com/original-owner-username/original-repository.git (fetch)
upstream    https://github.com/original-owner-username/original-repository.git (push)
```

Now you can refer to `upstream` on the command line instead of writing the entire URL, and you are ready to sync your fork with the original repository.

To sync our fork, from the directory of our local repository in a terminal window, we'll use the `git fetch` command to fetch the branches along with their respective commits from the upstream

be different, and may include a few lines on counting, compressing, and unpacking objects. Your output will end similarly to the following lines, but may vary depending on how many branches are part of the project:

```
Output
From https://github.com/original-owner-username/original-repository
 * [new branch]      main     -> upstream/main
```

Now, commits to the main branch will be stored in a local branch called `upstream/main`.

Let's switch to the local main branch of our repository:

```
$ git checkout main
```

```
Output
Switched to branch 'main'
```

We'll now merge any changes that were made in the original repository's main branch, that we will access through our local upstream/main branch, with our local main branch:

```
$ git merge upstream/main
```

The output here will vary, but it will begin with `Updating` if changes have been made, or `Already up-to-date.` if no changes have been made since you forked the repository.

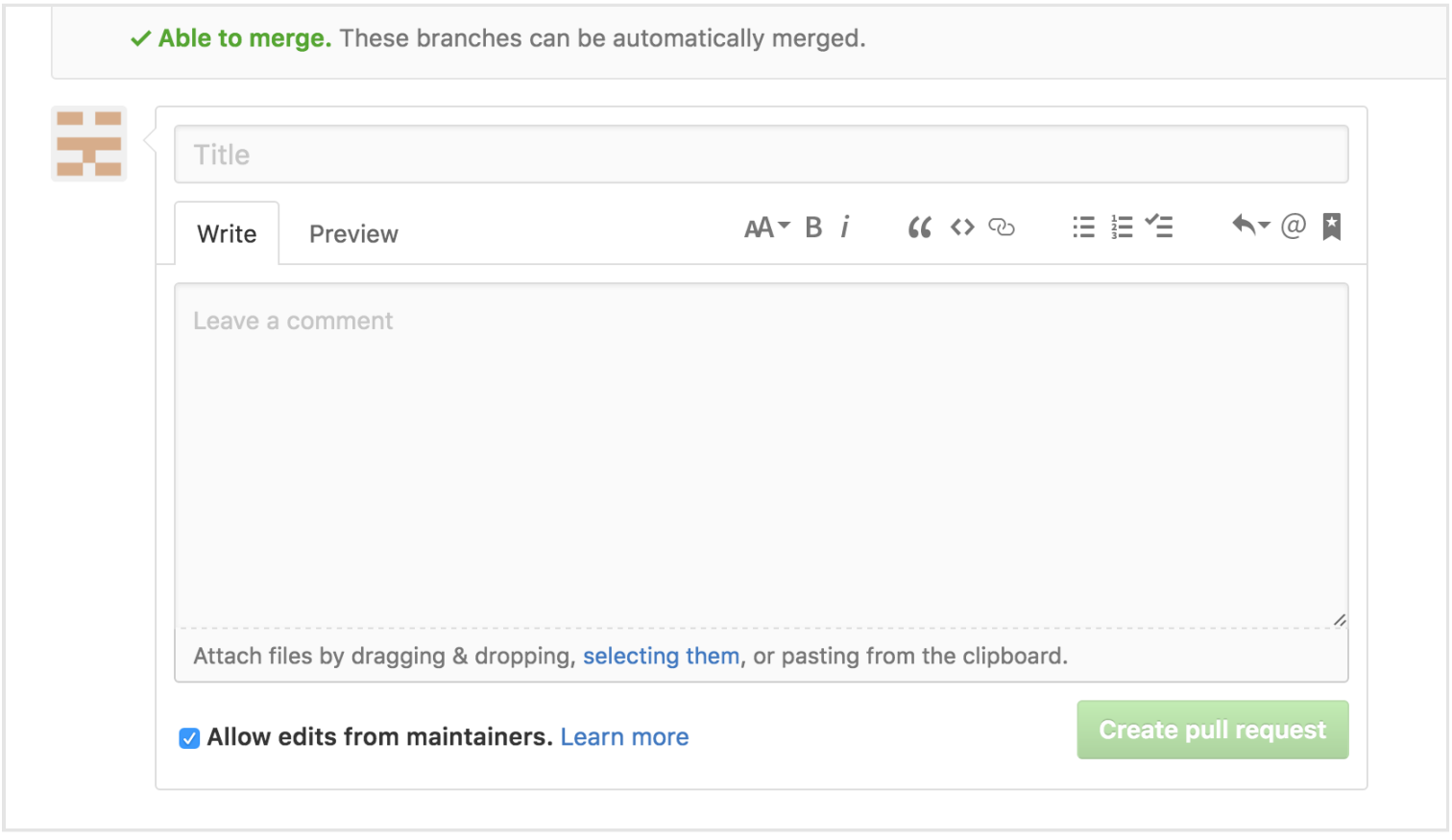
Your fork's main branch is now in sync with the upstream repository, and any local changes you

Create Pull Request

At this point, you are ready to make a pull request to the original repository.

You can modify the branch on the next screen. On either site you can select the appropriate repository from the drop-down menu and the appropriate branch.

Once you have chosen, for example, the main branch of the original repository on the left-hand side, and the `new-branch` of your forked repository of the right-hand side, you should see a screen that looks like this:



GitHub will alert you that you are able to merge the two branches because there is no competing

At this point, you have successfully sent a pull request to an open-source software repository. Following this, you should make sure to update and rebase your code while you are waiting to have

If you're interested in learning more about Git and collaborating on open source, you can read our tutorial series entitled *An Introduction to Open Source*. If you're already familiar with Git, and would like a cheat sheet, you can refer to "How To Use Git: A Reference Guide."

Next in series: How To Rebase and Update a Pull Request →

About the authors

Lisa Tagliaferri

Community and Developer Education

Tutorial Series

An Introduction to Open Source

Ask a question

Search for more help

RELATED

How To Use Enums in TypeScript

Tutorial

How To Create User Interactions with Events in Vue

Tutorial

Comments

18 Comments

ltagliaferri January 17, 2018

0

Hi there. This is the entire recommended process to ensure best practices when making a pull request on GitHub as at the time of writing. Because a majority of the process is done on the command line, that line functions to set expectations for readers — some of whom may be less familiar with the command line — letting them decide whether or not to continue with the tutorial. For additional guidance on pull requests on GitHub, you may want to check out GitHub's documentation.

Reply Report

siva4devops February 20, 2018

2

Hi,

Nice explanation and I have one doubt.

You have created new branch called new-branch from your forked repo and you started your development activity on "new-branch" branch, after completion of your activity pushed it to GitHub, so now you have two branched one is master and new-branch. I haven't seen step saying about merging new-branch changes.

Reply Report

ltagliaferri February 20, 2018

0

Thanks for your comment. This is intended for creating pull requests on open source repositories, so it is up to the repo maintainers to decide whether or not to accept your pull request, or to ask for revisions prior to acceptance.

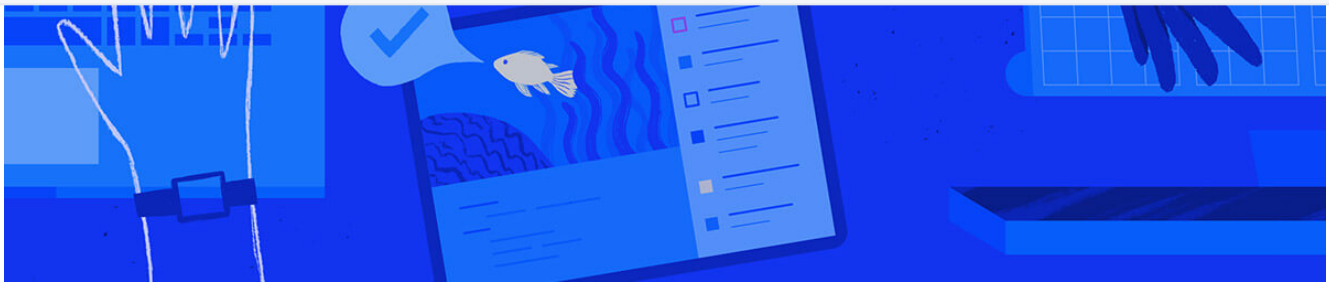
For guidance on doing the merging yourself for a repo that you maintain, you can refer to GitHub's article on merging pull requests.

Reply Report

ltagliaferri May 22, 2018

0

When you need to check against the original repo, you should rebase and update the pull request, you



How To Rebase and Update a Pull Request

by Lisa Tagliaferri

This tutorial will guide you through some of the next steps you may need to take after you submit a pull request to an open-source software project.

Reply Report

joshgibson82 May 8, 2018

1

Wonderful Guide! Thanks so much for writing!

Reply Report

ltagliaferri May 9, 2018

0

Thanks for the comment, glad you found it useful!

Reply Report

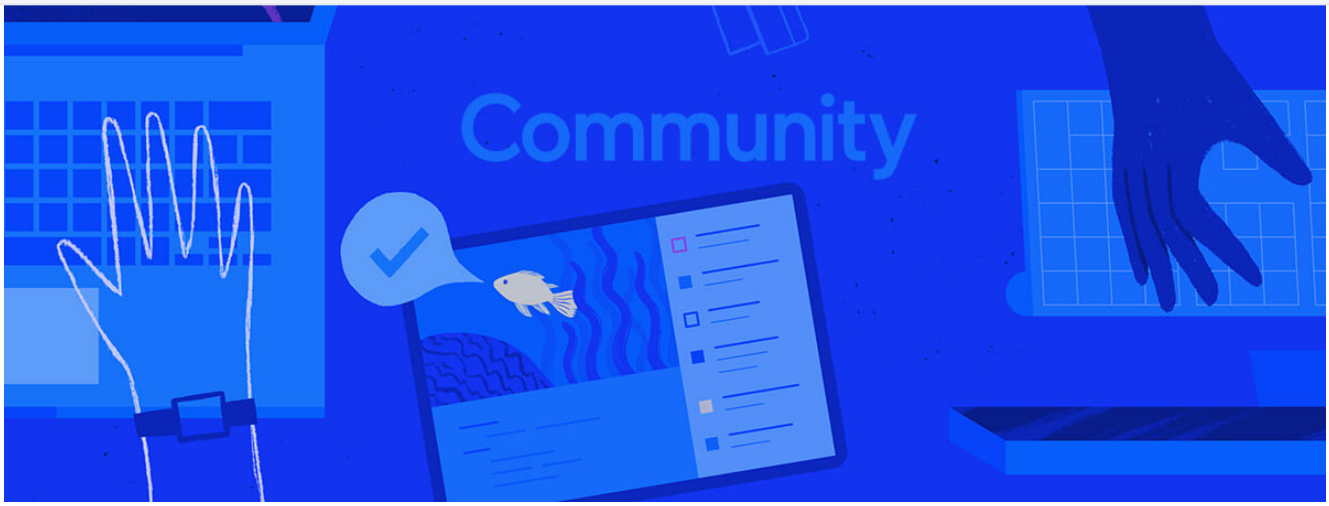
ernestochaveschaves May 18, 2018

3

Hey, great guide.

I do have one question though.
So, we forked the open-source-repo into our master. Then, we branched our master and created branch-feature, made changes to it.
After that, to sync our master branch with the open-source-repo, we fetched and merged into our own master.

Thanks for your comment — the “Create Pull Request” section details how to create a PR on the original repo from your forked repo as long as code is not competing. The way it is approached in this tutorial is using the in-browser GUI interface. You can look at GitHub’s official documentation on pull



How To Create a Pull Request on GitHub

by Lisa Tagliaferri

This tutorial will guide you through making a pull request to a Git repository through the command line so that you can contribute to open-source software projects.

Reply Report

ernestochaveschaves May 18, 2018

Hm OK. Not really what I was asking. I’ll give the docs another try. Thanks.

Reply Report

ltagliaferri May 18, 2018

In the last step, the PR should be pointing to the original open-source repo if that is your intended repo.

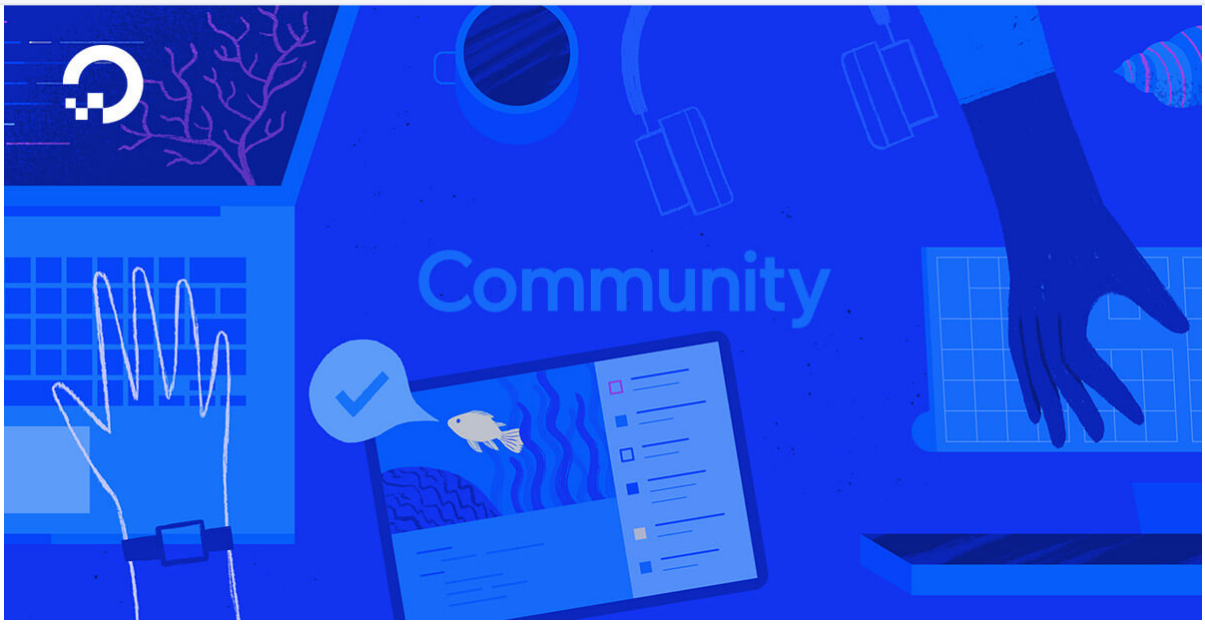
Reply Report

ernestochaveschaves May 18, 2018

Yes, I got that. But I think my question is very similar to the one made by @siva4devops

It is basically, in what step did we update our new branch with any changes coming

I think in your most recent comment you're referring to rebasing and updating a pull request. We have a tutorial that addresses that topic as part of this series, here is the link: https://www.digitalocean.com/community/tutorials/how-to-rebase-and-update-a-pull



How To Create a Pull Request on GitHub

by Lisa Tagliaferri

This tutorial will guide you through making a pull request to a Git repository through the command line so that you can contribute to open-source software projects.

Reply Report

jcbrown602438e40d29b3f9cc7 October 3, 2018

0

Thank you SO much for this guide! I'm learning a lot from the comments as well!

Reply Report

jimmyolano April 19, 2019

0

All is right! Thanks for help me to contribute to another libre software project, I am waiting for pull request approval; thanks again.

Reply Report

Hi Lisa,

It's a good tutorial but the line

Some of the newest versions of Git, they've introduced a new command to deal with branches navigation, wich is `git-switch` command. It would be nice to see this guide with this update too!

Reply Report

HOLLIE'S HUB FOR GOOD

Working on improving health

Sign up for our newsletter Get the latest tutorials on SysAdmin and open source topics.

Enter your email address

Sign Up

BECOME A CONTRIBUTOR

You get paid; we donate to tech nonprofits.

Featured on Community Kubernetes Course Learn Python 3 Machine Learning in Python Getting started with Go Intro to Kubernetes

DigitalOcean Products Virtual Machines Managed Databases Managed Kubernetes Block Storage Object Storage Marketplace VPC Load Balancers

Welcome to the developer cloud

DigitalOcean makes it simple to launch in the cloud and scale up as you grow – whether you're running one virtual machine or ten thousand.

Learn More