How to Collaborate On GitHub

*by*Jonathan Cutrell

If you don't already know, GitHub is an incredibly effective way to collaborate on development projects. Providing a place for anyone with an internet connection to have an avenue where they can share code with the world for free (not to mention the robust supporting tools for source inspection and easy viewing of commit histories). GitHub has been adopted by many large open-source projects as their primary home for collaboration and contribution.

But how do you join in and contribute to a project? Sure, you know how to use Git to track changes to files and push those files to a server. But there are major benefits to getting involved in larger open-source projects, and GitHub is arguably the best place to start. Today, we will discuss a few rules of the road for collaborating on open source projects, and give you the knowledge and intuition you will need to get involved.

Start Small

> *Don't be afraid to start small*

One of the most important things to understand when getting started with collaboration on open-

source projects is to recognize your role. Often, there are plenty of things you as a developer can do that don't involve being an extremely clever programmer. In fact, fear of being an inadequate programmer is often a reason why people don't get involved in open source projects to begin with. Don't be afraid to start small: instead of trying to fix a major bug or rewriting an entire module, try finding things like documentation inadequacies or cross-device testing and patching, or even simple syntax errors and grammar issues (like this one from GitHub user mzgol).

These kinds of tasks are a good way to get your foot in the door as a contributor to the project without trying to take on more than you can handle. Sign up forCodeTriage to get automated GitHub Issues sent to your inbox. If one hits your inbox that you feel confident you can take on, work on it and send a pull request. (We'll talk about how to do that a bit further down in the post.)

Learn the Ecosystem of the Project

With any collaborative effort, a set of conventions has probably been adopted. This may include a vocabulary set, a way of contributing and formatting commit messages, a certain rhythm of collaborating that the contributors have agreed to, or even syntactic standards that have been established. Before you try to get involved with a project, read all documents related to these things. For instance, GitHub has standardized a `CONTRIBUTING.md` file (check out the guidelines for getting involved with jQuery for a thorough example). These guides are maintained by the people who also maintain the codebase and the `master branch`.

Another way of understanding the ecosystem of a project is to simply look at the existing codebase and the `git log`. Reading through the commit messages and perusing the code style can tell you a lot about a project. Read through the project's documentation, and adopt the vocabulary used so that your contributions maintain continuity and portray a similar voice.

Once you're part of the project's cultural ecosystem, how do you actually contribute code?

The Pull-Request Workflow for Code Contribution

> *The workflow for contributing code can seem daunting at first.*

The workflow for contributing code can seem daunting at first. The most important thing to remember is to follow the patterns and standards outlined by the project you are working on (as we have already discussed). The general workflow that GitHub supports is fairly simple.

1. Fork the target repo to your own account.

2. Clone the repo to your local machine.

3. Check out a new "topic branch" and make changes.

4.Push your topic branch to your fork.

5.Use the diff viewer on GitHub to create a pull request via a discussion.

6.Make any requested changes.

7.The pull request is then merged (usually into the master branch) and the topic branch is deleted from the upstream (target) repo.

Within this workflow, you may see many variations for any given project. For instance, the naming conventions for topic branches vary. Some projects use conventions like `bug_345`, where 345 is the ID # of a GitHub issue that has been filed. Some projects prefer shorter commit messages than others. Here is a series of commands that would complete the workflow above.

Step 1: Forking

Fork the repo on GitHub.com

GitHub, Inc. [US] https://github.com/jcutrell/jquery

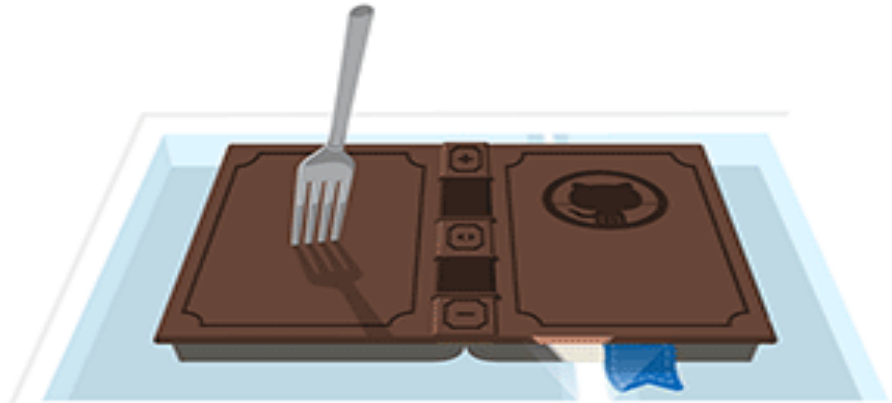
This repository | Search or type a command | Explore | Gist | Blog | Help | jcutrell

jcutrell / jquery
forked from jquery/jquery

Unwatch | Star 0

Forking jquery/jquery

We're forking a repository just for you. It should only take a few seconds. Refresh at will



Step 2: Cloning

If you don't already know, GitHub is an incredibly effective way to collaborate on development projects. Providing a place for anyone with an internet connection to have an avenue where they can share code with the world for free (not to mention the robust supporting tools for source inspection and easy viewing of commit histories). GitHub has been adopted by many large open-source projects as their primary home for collaboration and contribution.

But how do you join in and contribute to a project? Sure, you know how to use Git to track changes to files and push those files to a server. But there are major benefits to getting involved in larger open-source projects, and GitHub is arguably the best place to start. Today, we will discuss a few rules of the road for collaborating on open source projects, and give you the knowledge and intuition you will need to get involved.

Start Small

> *Don't be afraid to start small*

One of the most important things to understand when getting started with collaboration on open-source projects is to recognize your role. Often, there are plenty of things you as a developer can do that don't involve being an extremely clever programmer. In fact, fear of being an inadequate programmer is often a reason why people don't get involved in open source projects to begin with. Don't be afraid to start small: instead of trying to fix a major bug or rewriting an entire module, try finding things like documentation inadequacies or cross-device testing and patching, or even simple syntax errors and grammar issues (like this one from GitHub user mzgol).

These kinds of tasks are a good way to get your foot in the door as a contributor to the project without trying to take on more than you can handle. Sign up for CodeTriage to get automated GitHub Issues sent to your inbox. If one hits your inbox that you feel confident you can take on, work on it and send a pull request. (We'll talk about how to do that a bit further down in the post.)

Learn the Ecosystem of the Project

With any collaborative effort, a set of conventions has probably been adopted. This may include a vocabulary set, a way of contributing and formatting commit messages, a certain rhythm of collaborating that the contributors have agreed to, or even syntactic standards that have been established. Before you try to get involved with a project, *read all documents related to these things*. For instance, GitHub has standardized a `CONTRIBUTING.md` file (check out the guidelines for getting involved with jQuery for a thorough example). These guides are maintained by

the people who also maintain the codebase and the `master branch`.

Another way of understanding the ecosystem of a project is to simply look at the existing codebase and the `git log`. Reading through the commit messages and perusing the code style can tell you a lot about a project. Read through the project's documentation, and adopt the vocabulary used so that your contributions maintain continuity and portray a similar voice.

Once you're part of the project's cultural ecosystem, how do you actually *contribute code*?

The Pull-Request Workflow for Code Contribution

> *The workflow for contributing code can seem daunting at first.*

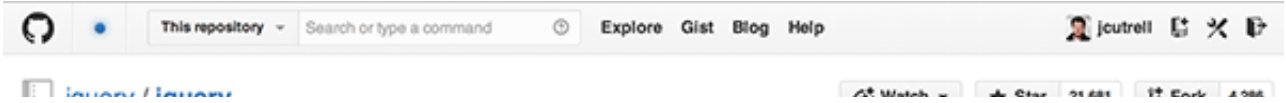
The workflow for contributing code can seem daunting at first. The most important thing to remember is to follow the patterns and standards outlined by the project you are working on (as we have already discussed). The general workflow that GitHub supports is fairly simple.

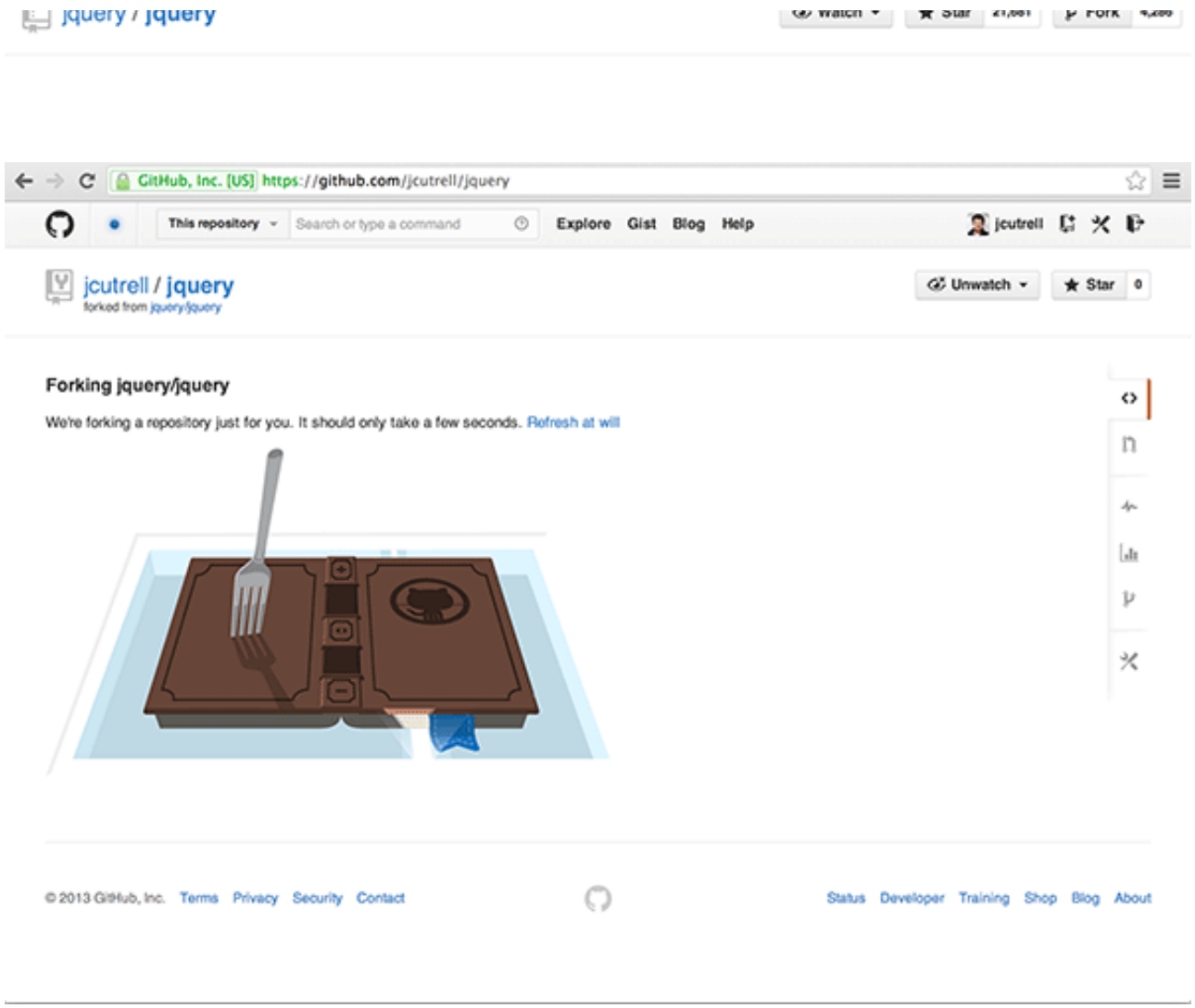
1. Fork the target repo to your own account.
2. Clone the repo to your local machine.
3. Check out a new "topic branch" and make changes.
4. Push your topic branch to your fork.
5. Use the diff viewer on GitHub to create a pull request via a discussion.
6. Make any requested changes.
7. The pull request is then merged (usually into the master branch) and the topic branch is deleted from the upstream (target) repo.

Within this workflow, you may see many variations for any given project. For instance, the naming conventions for topic branches vary. Some projects use conventions like `bug_345`, where 345 is the ID # of a GitHub issue that has been filed. Some projects prefer shorter commit messages than others. Here is a series of commands that would complete the workflow above.

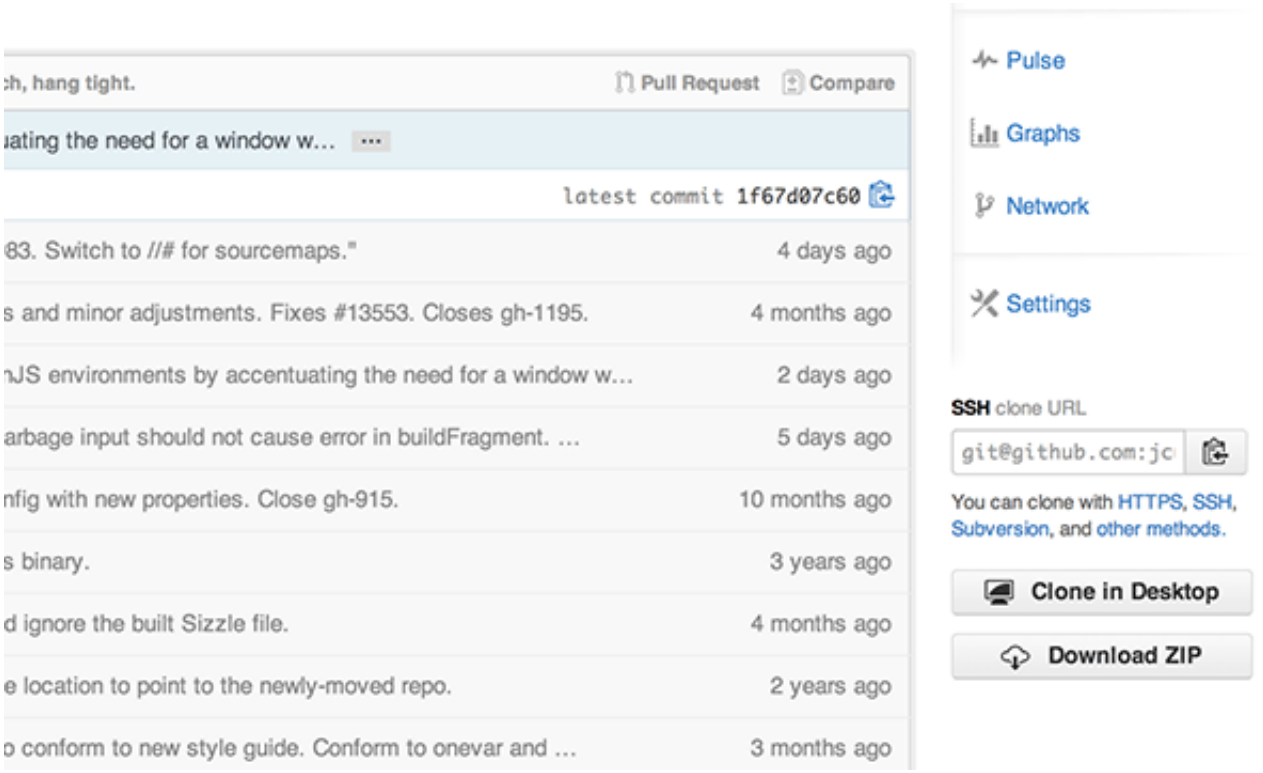
Step 1: Forking

Fork the repo on GitHub.com





Clone the repo using the URL in the right sidebar:



Step 3: Adding the Upstream Remote

Change into the cloned directory and then at this point, you can add the upstream remote:

This will now allow you to pull in changes from the source locally and merge them, like so:

Step 4: Checking Out a Topic Branch

However, before you make your own changes, checkout a topic branch:

Step 5: Committing

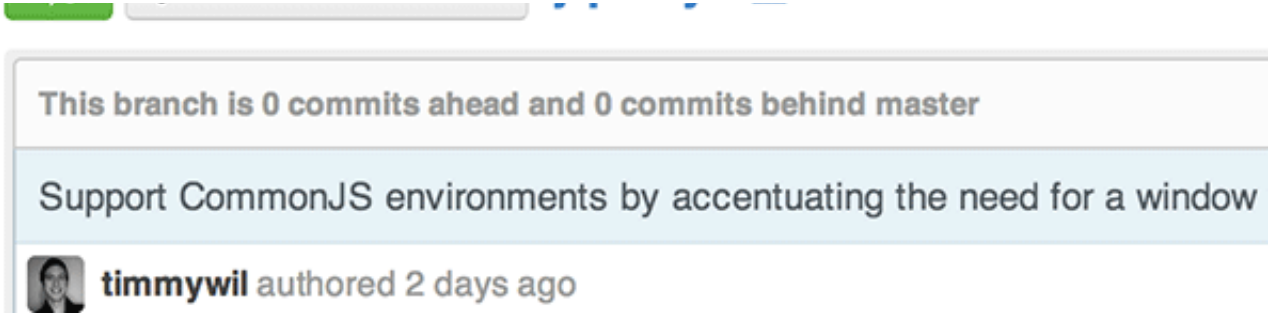
Now, you can make your changes, and create a commit that tracks *just those changes*.

Step 6: Pushing

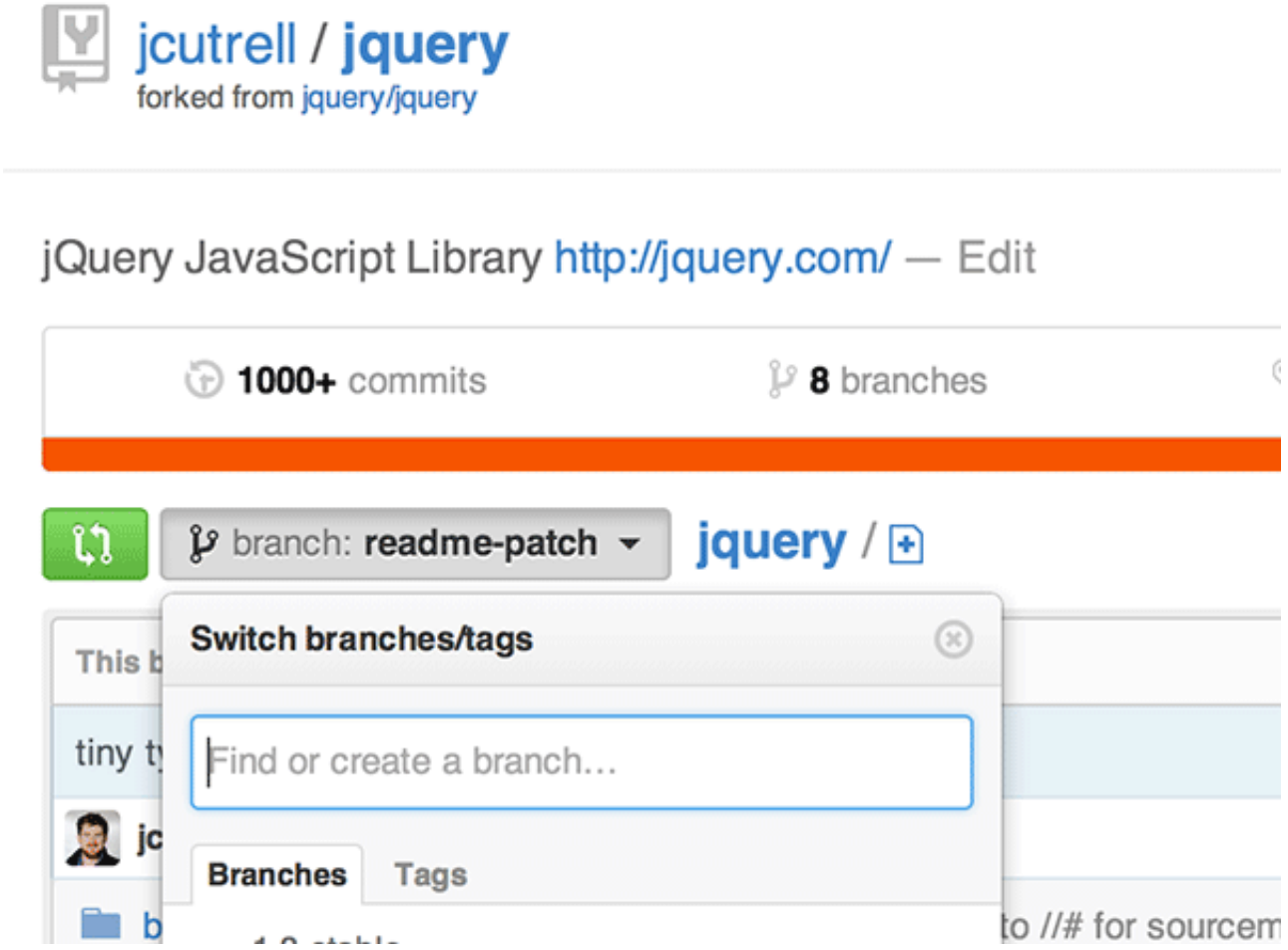
Next, you'll push this topic branch to your own fork of the project.

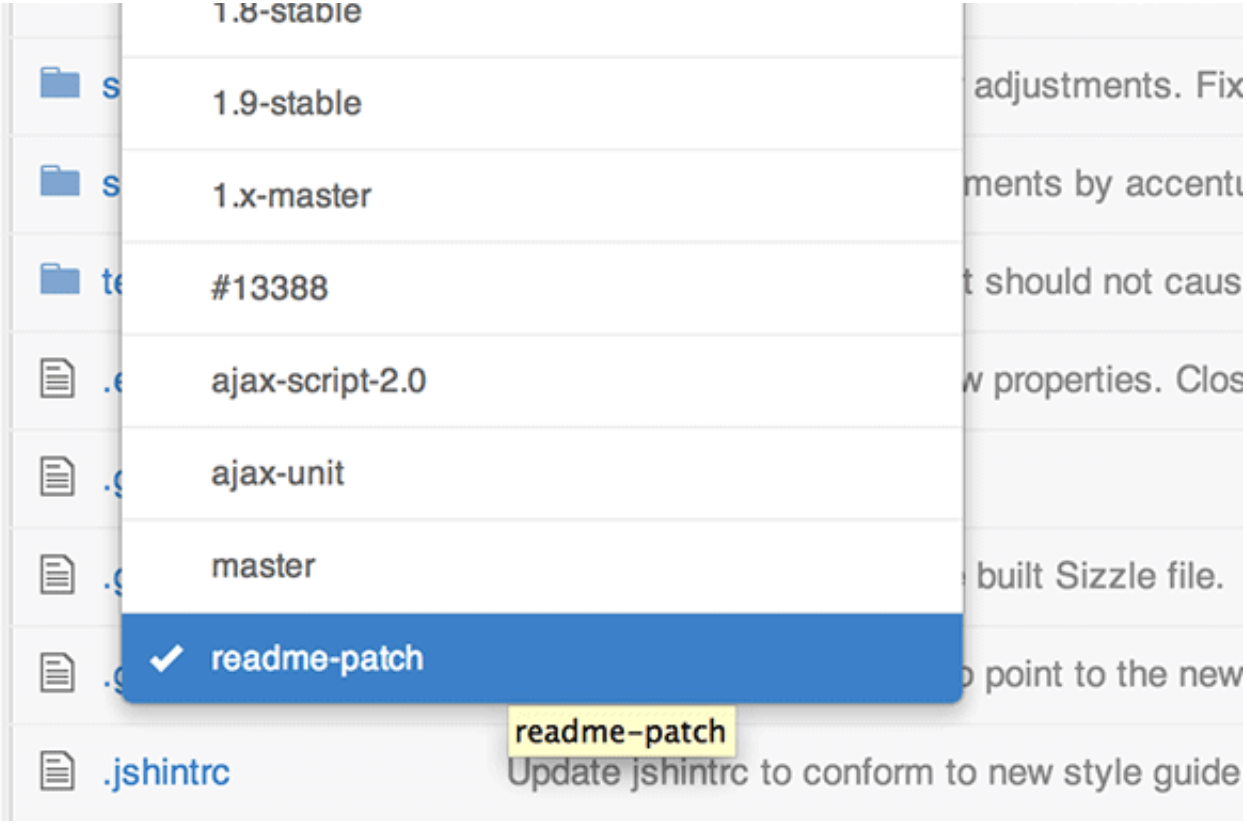
Step 7: Creating a Pull Request

Finally, you will create a pull request. First, go to your fork of the repo. You might see a "**your recently pushed branches**", and if so, you can choose "**Compare and Pull Request**". Otherwise, you can select your branch from the dropdown, and subsequently click "**Pull Request**" or "**Compare**" at the top right of the repo section.



Creating a pull request via the **Compare and Pull Request** button.





Creating a pull request via the branch dropdown menu.

Either of these will take you to a page where you can create a pull request and comment on the request. This page also includes a visualization of the changes you made. This makes it easy for the project administrator to see what you have done and make easier decisions about whether it is appropriate to merge your commit. If they have questions, they can ask them in the comments; they may also ask you to clean up your pull request and resubmit, and subsequently close the pull request.

Note that it is incredibly important that you show the administrators of a project full respect; after all, you can always use your forked version of the code, and if they chose not to pull in your changes, it is because they have the position to do so. Remember, according to Github Employee Zach Holman's take in "How GitHub Uses GitHub to Build GitHub", pull requests are conversations. This is how they should be treated; instead of expecting your commit to be accepted, you should expect only that it will open conversation about the code that you wrote.

GitHub Issues + Pull Requests = Project Management Zen

GitHub offers GitHub Issues, which is a robust way of creating documented, interactive, automated conversations about bugs or features for any given project. While Issues can be disabled, they are enabled by default. There are a lot of awesome features that Issues has built-in, but one of the most important features is its integration with pull requests. A user can reference an issue in their commit message by simply including the issue's numerical ID in the commit message. For instance:

This commit message would automatically mark issue #3 as closed when its associated pull request is accepted. This kind of automation makes GitHub a wonderful tool for development project management.

Seek Out Secondary Channels of Collaboration

> *Often, large open-source projects benefit from many different kinds of collaborative work.*

Don't get caught up thinking that the only way you can contribute is through pull requests. Often, large open-source projects benefit from many different kinds of collaborative work. For instance, a project like Ruby on Rails was notorious for its *community*; this community would answer questions on forums and in IRC chatrooms to help build knowledge about the framework, and also would help drive the future direction of the framework by talking about ideas and uncovering bugs.

These channels of collaboration are usually opened up as support environments as mentioned before, such as forums and chatrooms. There may also be email chains, meetups, or conference calls that help define the project's direction and create a lively, productive community around the project. Without this kind of community, pull requests are far less effective.

Most of All, It's About Your Attitude

Remember, open source is driven by people who have the attitude that sharing knowledge and building collaborative intelligence is a worthwhile endeavor. Your involvement in these projects will be most effective if you approach a given project with the inquisitive attitude that asks "how can I help?" rather than a closed attitude that says "I'm going to help however I want." People in the open source world want to work with people who are genuinely driven to help others.

Conclusion

If you are interested in getting involved in an open source project, great! Remember, if you approach the project with the right attitude and start

small, you could see your name on pull requests merged into code that is distributed to people all over the world and used every day. Take the time to learn about the project and the people who are involved with the project. Develop a genuine interest in helping the project become better. The power of GitHub and the open-source world is continuing to grow every day; start collaborating with other developers, and you can be a part of that world!

code.tutsplus.com

http://code.tutsplus.com/tutorials/how-to-collaborate-on-github--net-34267

http://goo.gl/ffgS