GitHub Standard Fork & Pull Request Workflow

Whether you're trying to give back to the open source community or collaborating on your own projects, knowing how to properly fork and generate pull requests is essential. Unfortunately, it's quite easy to make mistakes or not know what you should do when you're initially learning the process. I know that I certainly had considerable initial trouble with it, and I found a lot of the information on GitHub and around the internet to be rather piecemeal and incomplete - part of the process described here, another there, common hang-ups in a different place, and so on.

In an attempt to collate this information for myself and others, this short tutorial is what I've found to be fairly standard procedure for creating a fork, doing your work, issuing a pull request, and merging that pull request back into the original project.

Creating a Fork

Just head over to the GitHub page and click the "Fork" button. It's just that simple. Once you've done that, you can use your favourite git client to clone your repo or just head straight to the command line:

```
# Clone your fork to your local machine
git clone https://github.com/USERNAME/PROJECT-NAME
```

Keeping Your Fork Up to Date

While this isn't an absolutely necessary step, if you plan on doing anything more than just a tiny quick fix, you'll want to make sure you keep your fork up to date by tracking the original "upstream" repo that you forked. To do this, you'll need to add a remote:

```
cd PROJECT-NAME
# Add 'upstream' repo to list of remotes
git remote add upstream
https://github.com/UPSTREAM-USER/ORIGINAL-PROJECT.git

# Verify the new remote named 'upstream'
git remote -v
```

Whenever you want to update your fork with the latest upstream changes, you'll need to first fetch the upstream repo's branches and latest commits to bring them into your repository:

```
# Fetch from upstream remote
git fetch upstream

# View all branches, including those from upstream
git branch -va
```

Now, checkout your own master branch and merge the upstream repo's master branch:

```
# Checkout your master branch and merge upstream
git checkout master
git merge upstream/master
```

If there are no unique commits on the local master branch, git will simply perform a fast-forward. However, if you have been making changes on master (in the vast majority of cases you probably shouldn't be - see the next section, you may have to deal with conflicts. When doing so, be careful to respect the changes made upstream.

Now, your local master branch is up-to-date with everything modified upstream.

Doing Your Work

Create a Branch

Whenever you begin work on a new feature or bugfix, it's important that you create a new branch. Not only is it proper git workflow, but it also keeps your changes organized and separated from the master branch so that you can easily submit and manage multiple pull requests for every task you complete.

To create a new branch and start working on it:

```
# Checkout the master branch - you want your new branch to come from master
git checkout master

# Create a new branch named newfeature (give your branch its own simple
informative name)
git branch newfeature

# Switch to your new branch
git checkout newfeature
```

Now, go to town hacking away and making whatever changes you want to.

Submitting a Pull Request

Cleaning Up Your Work

Prior to submitting your pull request, you might want to do a few things to clean up your branch and make it as simple as possible for the original repo's maintainer to test, accept, and merge your work.

If any commits have been made to the upstream master branch, you should rebase your development branch so that merging it will be a simple fast-forward that won't require any conflict resolution work.

```
# Fetch upstream master and merge with your repo's master branch
git fetch upstream
git checkout master
git merge upstream/master

# If there were any new commits, rebase your development branch
git checkout newfeature
```

```
git rebase master
```

Now, it may be desirable to squash some of your smaller commits down into a small number of larger more cohesive commits. You can do this with an interactive rebase:

```
# Rebase all commits on your development branch
git checkout
git rebase -i master
```

This will open up a text editor where you can specify which commits to squash.

Submitting

Once you've committed and pushed all of your changes to GitHub, go to the page for your fork on GitHub, select your development branch, and click the pull request button. If you need to make any adjustments to your pull request, just push the updates to GitHub. Your pull request will automatically track the changes on your development branch and update.

Accepting and Merging a Pull Request

Take note that unlike the previous sections which were written from the perspective of someone that created a fork and generated a pull request, this section is written from the perspective of the original repository owner who is handling an incoming pull request. Thus, where the "forker" was referring to the original repository as `upstream`, we're now looking at it as the owner of that original repository and the standard `origin` remote.

Checking Out and Testing Pull Requests

Open up the `.git/config` file and add a new line under `[remote "origin"]`:

```
fetch = +refs/pull/*/head:refs/pull/origin/*
```

Now you can fetch and checkout any pull request so that you can test them:

```
# Fetch all pull request branches
git fetch origin

# Checkout out a given pull request branch based on its number
git checkout -b 999 pull/origin/999
```

Keep in mind that these branches will be read only and you won't be able to push any changes.

Automatically Merging a Pull Request

In cases where the merge would be a simple fast-forward, you can automatically do the merge by just clicking the button on the pull request page on GitHub.

Manually Merging a Pull Request

To do the merge manually, you'll need to checkout the target branch in the source repo, pull directly from the fork, and then merge and push.

```
# Checkout the branch you're merging to in the target repo
git checkout master

# Pull the development branch from the fork repo where the pull request
development was done.
git pull https://github.com/forkuser/forkedrepo.git newfeature

# Merge the development branch
git merge newfeature

# Push master with the new feature merged into it
git push origin master
```

Now that you're done with the development branch, you're free to delete it.

```
git branch -d newfeature
```

Copyright

Additional Reading

- Atlassian - Merging vs. Rebasing

Sources

- GitHub - Fork a Repo
- GitHub - Syncing a Fork
- GitHub - Checking Out a Pull Request

https://gist.github.com/Chaser324/ce0505fbed06b947d962