Github Tips and Tricks

On this Page

See Also

- github help
- GIT Tips and Tricks

Tips

Keeping your fork up-to-date

(see: Fork a Repo)

Forking a repo on github is pretty easy, there's a button for that. However they do not provide a UI for keeping your fork up-to-date with the parent (upstream) repository.

To keep your fork up-to-date, you'll need to act as an intermediary, shuttling the updates from the upstream repo to your fork. This is handled through git's fetch, merge, push, and remote features.

```
# assumes you have already created a fork of a repo

# create a clone of the fork specified by "forkURL" and call it forkedRepo
git clone "forkURL" forkedRepo

# set a remote for the upstream repository
git remote add upstream "upstreamURL"

# retrieve the updates from the upstream repository
git fetch upstream

# make sure that you know what you're getting from upstream and what is in your master
git log upstream/master ^master
git log ^upstream/master master

# if everything is good, you can merge upstream into your clone
git merge upstream/master

# now you're ready to update your fork
git push origin master
```

Issuing a Pull Request

(see: Sending Pull Requests)

Managing a Pull Request

(see: Sending Pull Requests)

Advanced Diffs

With some URL hackery you can perform githubs diffs with tags, commits, branches, and even between repos.

You can replace item1 with a branch, tag, or commit hash. item2 should be a branch name.
You can also diff between repos by using the syntax "repo:branchName" to replace item1 or item2

```
https://github.com/fluid-project/infusion/compare/item1...item2
```

Maintaining commit history when moving work from a personal repo to a main project repo

Sometimes, a project begins life in a personal repo on GitHub and we later want to move that work to a repo under an organization. One option is to transfer ownership. This section outlines an alternative approach that involves merging work from the original repo into a fork of the new main, organization, repo. You may need to do this, for example, if the organization repo has already been created with some content.

The steps below will use the `simonbates/nexus` repo as an example.

1. Create an upstream repo in the organization; example: `gpii/nexus`
    - **Important**: Make sure this upstream repo is initialized with a placeholder README.md, or it will not be cloneable and forkable with the steps below.
2. On GitHub, rename the original repo to avoid confusion with the upstream repo, and to enable clean forking of the new upstream repo
    - For example, rename `simonbates/nexus` to `simonbates/simon-nexus`
3. Delete any local clones of the origin repo (or rename and update the `origin` remote – any local clones will still point to the previous GitHub repo URL)
4. On GitHub, fork the upstream repo
    - For example, fork `gpii/nexus` to `simonbates/nexus`
5. Clone the fork onto your computer
    - For example, `simonbates/nexus`
6. Add the original repo as a remote in the cloned fork
    - For example, in the local clone of `simonbates/nexus`, add `simonbates/simon-nexus` as a remote
7. Merge the work from the original repo into the new clone
    - For example, in the local clone of `simonbates/nexus`, merge `simon-nexus/master`
    - When merging, add the `--allow-unrelated-histories` flag if you are using git version 2.9 or later. Otherwise you will get an error message on "refusing to merge unrelated histories".
        - See: https://stackoverflow.com/questions/37937984/git-refusing-to-merge-unrelated-histories
8. Things to check that may need updating:
    a. Repo references in package.json or license statements - these should be updated to be references to the new organizational repo
9. Push the changes from the local clone up to GitHub and make a pull request