Github Managing Forks

Christian Külker

2024-03-12

Contents

1 Introduction

There are several ways to manage forks. One way is to manage forks through the web interface. While this has the advantage of no additional configuration in the forked repository, it has the disadvantage of a complex workflow and cannot be applied in all cases. In addition, it requires opening multiple views (browser tabs) of different repositories, which in some cases can lead to a pull request being opened in the wrong repository. Of course,

those requests cannot be deleted, marking the error forever. Therefore, this document describes a different method that uses the command line and the additional configuration of an upstream remote source.

2 Forking/ Merging

Fork/Merge the upstream

Forking (GitHub) or merging (GitLab) the upstream needs to be done before the management. While the naming between GitHub and GitLab is different the concept is similar. So, for now, I will use GitHub as an example: After logging into GitHub via the https://github.com web interface, browse to the project to be forked and use the drop down button 'fork' and choose '+ create a new fork'. Enter a new fork name (repository name) and decide weather you copy only the default branch or not. In some cases it has to be **not** checked, because the default branch is not the development branch. This depends on the repositories policy. Then clone the fork to your local working space. More details can be found in the document GitHub Pull Requests.

3 Management - Too Long To Read - TLTR

Set up a managing fork via upstream (drawback, local repo will be ahead)

```
git remote show
git remote add upstream https://github.com/whoever/whatever.git
git remote show
git remote show upstream
```

Update a fork via 'upstream' (Update only, no local commits expected)

```
git remote show upstream # make sure the result is 'fast-forwardable'
git fetch upstream
git branch -va # make sure if the branches are not changed (e.g. main->dev)
# Often 'git checkout main' or master is used, but the next 2 lines are
 ↪  trying
# to guess the branch in a more general way
export B=$(git branch -va|grep 'origin/HEAD'| \
sed -e 's%.*remotes/origin/HEAD.*->\s\+origin/%%g')
git checkout $B # See if $B is main/develop/...: see remotes/origin/HEAD
 ↪  -> \
origin/BRANCH
# Option 1: (good if no additional unique commits)
git merge upstream/$B # See if $B is main/develop/...
```

```
git push
# Option 2: (my unique commits not in $B gets on top)
git rebase upstream/$B
git push -f origin $B
```

If done via the GUI (“Fetch upstream”), it will also add a merge commit like “Merge branch ‘corona-warn-app:release/2.24.x’ into development” (if the HEAD points to development) and leave the repository +1 commits ahead.

4 Adding A Remote Upstream

To understand the current state of a git repository - any repository, forked or not - you can use the git remote command.

```
git remote show
origin
```

Usually, it just shows the word `origin` if the repository is a fork. If you specify the name `origin`, more information can be displayed.

```
git remote show origin
```

For the repository `cwa-documentation`, for example, it will give:

```
* remote origin
  Fetch URL: git@github.com:ckuelker/cwa-documentation.git
  Push  URL: git@github.com:ckuelker/cwa-documentation.git
  HEAD branch: main
  Remote branches:
    SabineLoss-patch-1                                tracked
    SabineLoss-patch-2                                tracked
    fix/typo-overview-security.md-identification-against tracked
    main                                              tracked
  Local branch configured for 'git pull':
    main merges with remote main
  Local refs configured for 'git push':
    fix/typo-overview-security.md-identification-against pushes to \
fix/typo-overview-security.md-identification-against (up to date)
    main                                              pushes to \
main                                             (up to date)
```

After the configuration details for pushing and fetching, and the current state of where

the repository head is pointing at (main), information about remote and local branches is displayed.

```
git remote add upstream https://github.com/whoever/whatever.git
```

Example:

```
git remote add upstream
 ↪  https://github.com/corona-warn-app/cwa-documentation.git
```

To confirm that a remote entity has been successfully added, the `git remote` command is useful.

```
git remote show
origin
upstream
```

It should show the word `upstream`.

Similar to the `remote origin` information, detailed information about the `remote upstream` can be displayed using the `git remote show upstream` command.

For the `cwa-documentation` repository the upstream information looks as follows:

```
git remote show upstream
* remote upstream
  Fetch URL: https://github.com/corona-warn-app/cwa-documentation.git
  Push  URL: https://github.com/corona-warn-app/cwa-documentation.git
  HEAD branch: main
  Remote branches:
    SabineLoss-patch-1                          tracked
    SabineLoss-patch-2                          tracked
    fix/typo-backend-infrastructure-architecture tracked
    main                                        tracked
    tkowark-patch-1                             tracked
    tune_linting                                tracked
  Local ref configured for 'git push':
    main pushes to main (fast-forwardable)
```

The above shows a current situation. The following shows an out-of-sync situation (output truncated):

```
  ...
  Local branch configured for 'git pull':
    main merges with remote main
```

```
  Local ref configured for 'git push':
    main pushes to main (local out of date)
```

5 Updating the Fork

There is more than one way to update a fork. In trivial cases, the git repository GUI (e.g. https://github.com) provides a method to do this. Since this may change with a new release of the GUI, a more general method is to add a remote upstream, fetch the upstream, checkout main, and merge upstream and main.

It is not absolutely necessary to keep the fork current. If you are working on anything more than a tiny, quick fix, make sure you keep your fork up to date by tracking the original upstream repo you forked from. To do this, you'll need to add a remote:

```
$ git remote add upstream
 ↪  https://github.com/UPSTREAM-USER/ORIGINAL-PROJECT.git
```

Verify the new remote named 'upstream' with `git remote -v`. See this example for `cwa-documentation.git`.

```
git remote -v
origin  git@github.com:ckuelker/cwa-documentation.git (fetch)
origin  git@github.com:ckuelker/cwa-documentation.git (push)
upstream        https://github.com/corona-warn-app/cwa-documentation.git
 ↪  (fetch)
upstream        https://github.com/corona-warn-app/cwa-documentation.git
 ↪  (push)
```

Update the repository with the latest changes from upstream by fetching the latest commits and branches.

```
$ git fetch upstream
```

View all branches, including those from upstream

```
$ git branch -va
```

Alternatively, if you would like to automate the search.

```
export B=$(git branch -va|grep 'origin/HEAD'|\
sed -e 's%.*remotes/origin/HEAD.*->\s\+origin/%%g')
```

Option 1 (merge): If you do not have any unique commits this option is probably the

best. Check out the main branch and merge it into the upstream repository's main branch. Sometimes the main branch is called "main", sometimes other names such as "development".

```
$ git checkout $B
$ git merge upstream/$B
```

If there are no unique commits on the local main branch, Git will simply fast-forward. However, if changes have been made to the source (upstream) repository (which you probably should not do), conflicts may occur. Be careful with upstream changes. If changes have been made upstream, you should consider two options. One is to simply follow upstream. In this case, only update the fork when your pull request has been successfully merged into the upstream repository. Or second is to develop new software based on the fork. In this case, synchronizing the fork should be done in a less controlled way. If you find yourself in the second category, and are forced to do many, large, and complex synchronizations to your fork, you should consider abandoning your fork and contributing to the upstream repository.

```
$ git push
```

Option 2 (rebase): For making further pull requests that are as clean as possible, especially if you have additional commits, it's probably better to rebase. General problems of using rebase are remaining. This is not advised if other people are using your fork (other than merging a pull request).

```
$ git rebase upstream/$B
```

If you've rebased your branch onto upstream/$B, you may need to force the push in order to push it to your own forked repository on GitHub. You'd do that with:

```
git push -f origin $B
```

6 Additional Steps

You are done if you just forked the other repository. The next steps depend on (1) what you did with the fork: history, and (2) what you intend to do with it: usage.

The history question (1) can be divided into:

1. Nothing (no changes, just a fork)
2. Changes (changes in the work tree: nothing is added or committed)
3. Add changes
4. Add changes and committed

5. Add changes and committed and created a pull request
6. Add changes and committed and created a pull request that was accepted

The use question (2) can be divided into:

1. Users should be able to use a fork with a pristine commit history
2. Users should be able to use a fork with a different commit history

7 Further Reading

- Option 1: Syncing a fork
- Option 2: How do I update a GitHub forked repository

8 Example: Make a Second Pull Request

This section assumes that you want to make a second pull request and that the main branch is called 'main'. Just to be sure this is what you did for your first pull request.

```
git clone git@github.com:USER/forked-repository.git
cd forked-repository
git checkout -b fix1
# After making a change to 'changed1.file'
git commit -m "Fix X1 (because of Y1)" changed1.file
git push origin fix1
```

1. Make a pull request in the GitHub GUI
2. Wait until it is approved and merged to the upstream repository

Manage the fork after the accepted pull request.

```
git remote add upstream https://github.com/whoever/whatever.git
git fetch upstream
git checkout main
git merge upstream/main
git push
```

Now the fork should be up to date. We make another commit.

```
git checkout -b fix2
# After making a change to 'changed2.file'
git commit -m "Fix X2 (because of Y2)" changed2.file
git push origin fix2
```

1. Make a second pull request in the GitHub GUI
2. Wait until it is approved and merged to the upstream repository again

Then we manage the fork after the accepted pull request again, but without adding a remote upstream.

```
git fetch upstream
git checkout main
git merge upstream/main
git push
```

9 History

Version	Date	Notes
0.1.4	2024-03-12	Options (merge, rebase); example of 2nd pull request
0.1.3	2024-02-27	Add remarks about forking in general; formatting
0.1.2	2023-05-09	Improve writing
0.1.1	2022-06-25	Examples, update TLTR, shell->bash, commands, main
0.1.0	2020-09-24	Initial release

10 Disclaimer of Warranty

11 Limitation of Liability