Github Managing Forks

Christian Külker

2022-06-25

Contents

Managing forks can be done in different ways. One way is to manage forks via 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 can not be applied to all cases. In addition it requires to open several views (open browser tabs) of different repositories which in some cases can lead to open a pull request at the wrong repository. And of course those requests can not be deleted and mark the error for eternity. Because of this an other method that uses the command line and the additional configuration of an upstream remote origin will be described in this document.

1 To Long To Read - TLTR

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

```
git remote show upstream # make sure result is 'fast-forwardable'
git fetch upstream
git branch -va # make sure if the branches not changed (e.g. main->dev)
export B=$(git branch -va|grep 'origin/HEAD'|sed -e
 ↪  's%.*remotes/origin/HEAD.*->\s\+origin/%%g')
git checkout $B # See if $B is main/develop/...: see remotes/origin/HEAD
 ↪  -> origin/BRANCH
git merge upstream/$B # See if $B is main/develop/...
git push
```

If done via 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.

2 Adding A Remote Upstream

To understand the current situation of a git repository - any repository, regardless if it is a fork or not - the git remote command can be used.

```
git remote show
origin
```

Usually it just shows the word `origin` in case the repository is a fork. When specifying the name `origin` more information can be displayed.

```
git remote show origin
```

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

```
1  * remote origin
2    Fetch URL: git@github.com:ckuelker/cwa-documentation.git
3    Push  URL: git@github.com:ckuelker/cwa-documentation.git
4    HEAD branch: main
5    Remote branches:
6      SabineLoss-patch-1                                tracked
7      SabineLoss-patch-2                                tracked
8      fix/typo-overview-security.md-identification-against tracked
```

```
 9      main                                                  tracked
10    Local branch configured for 'git pull':
11      main merges with remote main
12    Local refs configured for 'git push':
13      fix/typo-overview-security.md-identification-against pushes to \
14  fix/typo-overview-security.md-identification-against (up to date)
15      main                                                  pushes to \
16  main                                                 (up to date)
```

After the configuration details of pushing and fetching and the current status where the head of the repository is pointing at (main), information about remote and local branches are 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 was added successfully the `git remote` command is useful.

```
git remote show
origin
upstream
```

It should show the word `upstream`.

Similar the information of the `remote origin` also for the `remote upstream` detailed information can be displayed via 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 an up to date situation. The following show 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)
```

3 Updating the fork

There are more than one approach to update a fork. In trivial cases the GUI of the git repository provides (for example `github.com`) a method to do so. As this can change with a new release of the GUI a more generic method is to add a remote upstream, fetch the upstream, checkout main and merge upstream and main.

Keeping the fork up-to-date is not absolutely necessary. If working on anything more than just a tiny quick fix, 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:

```
$ 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: fetch latest commits and branches.

```
$ git fetch upstream
```

View all branches, including those from upstream

```
$ git branch -va
```

Or 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')
```

Checkout the main branch and merge the upstream repository main branch. Sometimes the main branch is called ‘main’, sometimes other names like ‘development’

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

In case there are no unique commits on the local main branch, git will simply perform a fast-forward. However, if changes have been made to the source (upstream) repository (this probably should not be done) conflicts may occur. Be careful to changes made by upstream. If changes had been done to the upstream, you should consider 2 ways. One is to just follow upstream. If this is the case only update the fork if your pull request was successfully integrated into the upstream repository. Or second is to develop new software on the basis of the fork. Then synchronizing the fork should not be done in a very controlled way. If you find yourself in the second category and are forced to make many, large and complex synchronizations to your fork, you should consider to abandon your fork and contribute to the upstream repository.

```
$ git push
```

4 Additional Steps

You are done, if you just forked the other repository. The next steps depends on the question (1) what you have done to the fork: history and (2) what is the intended use: usage.

The history question (1) can be divided into:

1. Nothing (no changes, just a fork)
2. Changes (changes in the work tree: nothing 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

5 Further Reading

- [How do I update a `github` forked repository]
- Syncing a fork

6 History

Version	Date	Notes
0.1.1	2022-06-25	Examples, update TLTR, shell->bash, commands, main
0.1.0	2020-09-24	Initial release

7 Disclaimer of Warranty

THERE IS NO WARRANTY FOR THIS INFORMATION, DOCUMENTS AND PROGRAMS, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE INFORMATION, DOCUMENT OR THE PROGRAM “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE INFORMATION, DOCUMENTS AND PROGRAMS IS WITH YOU. SHOULD THE INFORMATION, DOCUMENTS OR PROGRAMS PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

8 Limitation of Liability

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE INFORMATION, DOCUMENTS OR PROGRAMS AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE INFORMATION, DOCUMENTS OR PROGRAMS (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE INFORMATION, DOCUMENTS OR PROGRAMS TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.