Making Changes to Tagged Commits
Software Engineering for Scientist

Consider the following repo with three tags: v1.0, v2.0, and v3.0

```
$ git log --oneline --graph
*   b31d246 (HEAD -> master, tag: v3.0, origin/master, origin/HEAD) Merge pull
request #4 from cu-swe4s-fall-2020/add_bsearch
|\
| * fb05b69 (origin/add_bsearch, add_bsearch) add binary search
|/
*   9a3256e (tag: v2.0) Merge pull request #3 from cu-swe4s-fall-2020/add_lsearch
|\
| * 857e976 (origin/add_lsearch, add_lsearch) add linear search
|/
*   e08dac8 (tag: v1.0) Merge pull request #2 from cu-swe4s-fall-2020/fixgetcol
|\
| * b0e0ed9 (origin/fixgetcol, fixgetcol) fix fix get_column
|/
* 5a6349a add code skeleton
* 8c04b00 Initial commit
```

First, let's discuss some of the information displayed above.

- Every line that begins with a start (*) corresponds to a commit that advanced the repo.
- Every commit has an identifier (e.g., `e08dac8`).
- A tag gives a name to a commit. The `v1.0` tag points to the `e08dac8` commit.
- You can see commits made to branches off of the main repo (e.g., `b0e0ed9` in the `origin/fixgetcol` branch), and the pull requests that merged those branches back into the master branch.
- Just as tags point to prior commits, the HEAD (usually) points to the latest commit. To branch off of an old commit we will move the HEAD back.

Suppose we need to fix a bug that was in v2.0. First we jump back to that point in the repo by checking out that tag.

```
$ git checkout v2.0
Note: checking out 'v2.0'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by performing another checkout.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -b with the checkout command again. Example:

  git checkout -b <new-branch-name>

HEAD is now at 9a3256e Merge pull request #3 from cu-swe4s-fall-2020/add_lsearch
```

It is equivalent to checkout the commit by id.

```
$ git checkout 9a3256e
```

If we inspect the repo we can see how HEAD has moved back to the commit.

```
$ git log --oneline --graph
*   9a3256e (HEAD tag: v2.0) Merge pull request #3 from cu-swe4s-fall-2020/add_lsearch
|\
| * 857e976 (origin/add_lsearch, add_lsearch) add linear search
|/
*   e08dac8 (tag: v1.0) Merge pull request #2 from cu-swe4s-fall-2020/fixgetcol
|\
| * b0e0ed9 (origin/fixgetcol, fixgetcol) fix fix get_column
|/
* 5a6349a add code skeleton
* 8c04b00 Initial commit
```

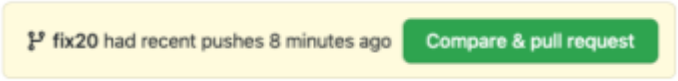
As the message states above, to make changes we need to create a branch.

```
$ git checkout -b fix20
Switched to a new branch 'fix20'
```

Suppose the bug is in the linear search function in my_utils.py. We fix it, commit those changes to the local branch, and push the branch to the remote repo.

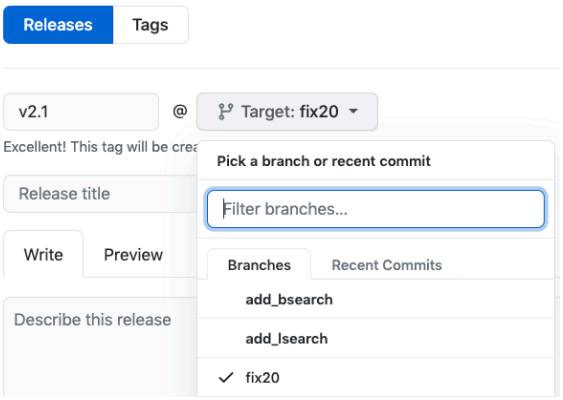
```
$ git add my_utils.py
$ git commit -m "return None if not found"
[fix20 d2b63d1] return None if not found
 1 file changed, 1 insertion(+), 1 deletion(-)
$ git push origin fix20
```

To create a release switch to GitHub. You will see a message from GitHub urging you to merge the branch.



Don't. If you try you will see this warning. Heed this warning.

Instead go to releases and draft a new release, and select the fix20 as the target branch to apply the tag



You can also do all of this from the command line by tagging this commit and then push it to the remote repo

```
$ git tag -a v2.1 d2b63d1 -m "Fix return value for linear search"
$ git push origin v2.1
```

On GitHub this tag will look different because it is not a “release.” Tags are a native git feature. Releases are a GitHub construct that displays tags in a particular way. Since we created the tag on the command line, it will look different than a release. To get the look of a release, you can create a new release from an existing tag.