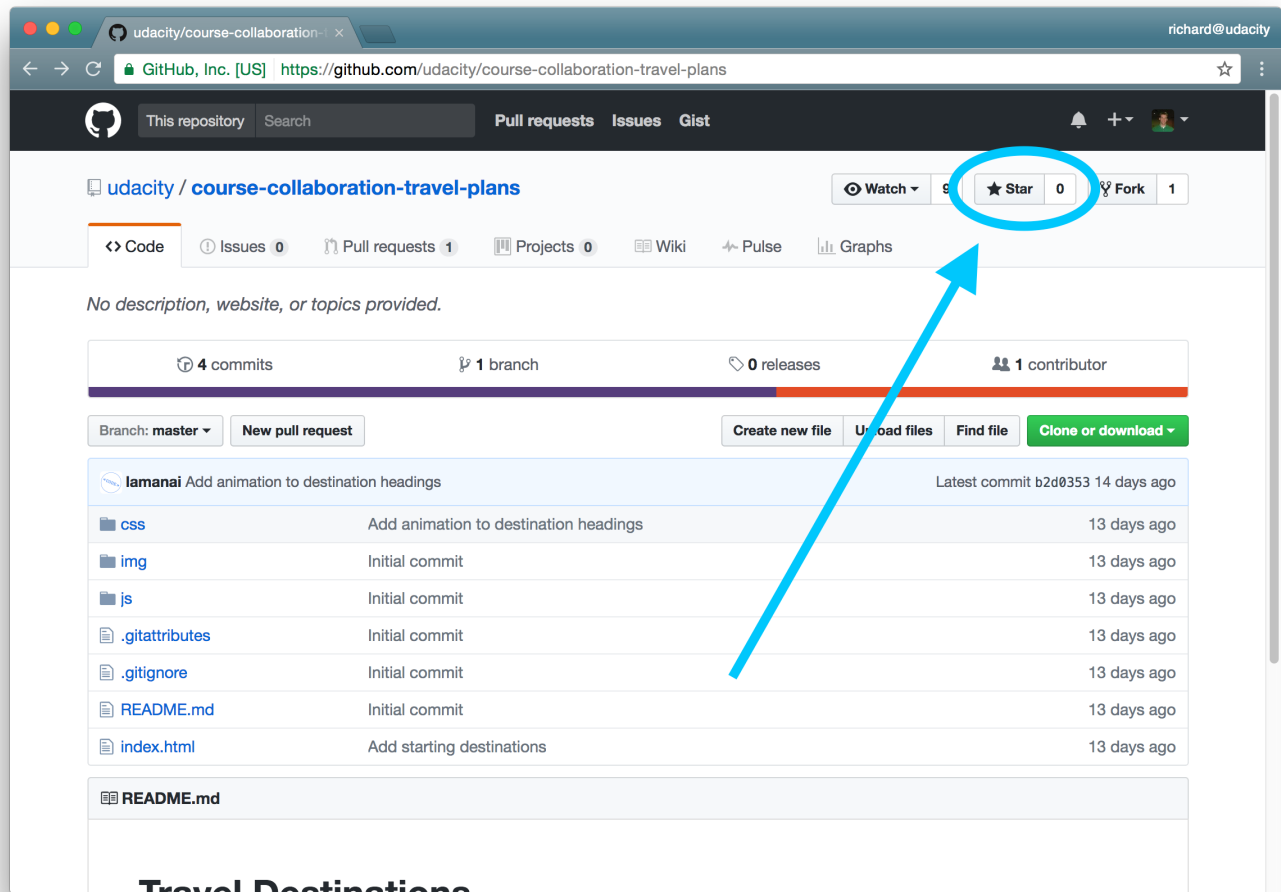


While you're working on a topic branch of changes that you want to make to a repository, that repository will probably be receiving updates of its own from the original authors.

## Stars & Watching

If you want to keep up-to-date with the Repository, GitHub offers a convenient way to keep track of repositories - it lets you star repositories:



*The Star button and level for a repository.*

You can go to <https://github.com/stars> to list out and filter all of the repositories that you have starred.

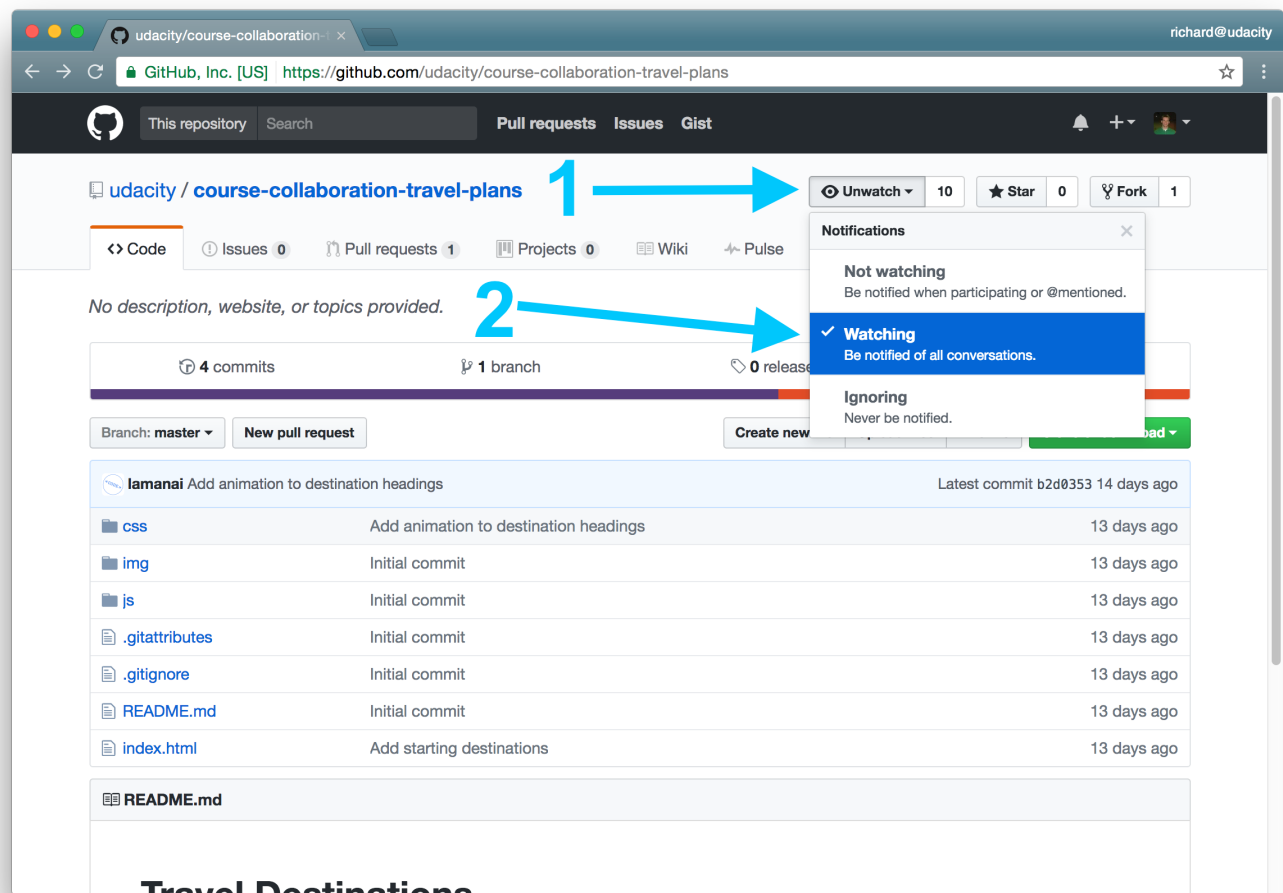
Starring is helpful if you want to keep track of certain repositories. But it's not entirely helpful if you need to actively keep up with a repositories development because you have to manually go to the stars page to view the repositories and see if they've changed.

Starring can be a useful feature to help you keep track of repositories you're interested in. But stars have also turned into a means of measuring a repo's popularity.

If you'd rather not increase a repository's stars, then check out "watching" a repository. Let's look at that right now!

## Watching A Repository

If you need to keep up with a project's changes and want to be notified of when things change, GitHub offers a "Watch" feature:



*The Notification settings. "Watching" a repository will alert you to all activity.*

If you're working on a repository quite often, then I'd suggest setting the watch setting to "Watching". This way GitHub will notify you whenever anything happens with the repository like people pushing changes to the repository, new issues being created, or comments being added to existing issues.

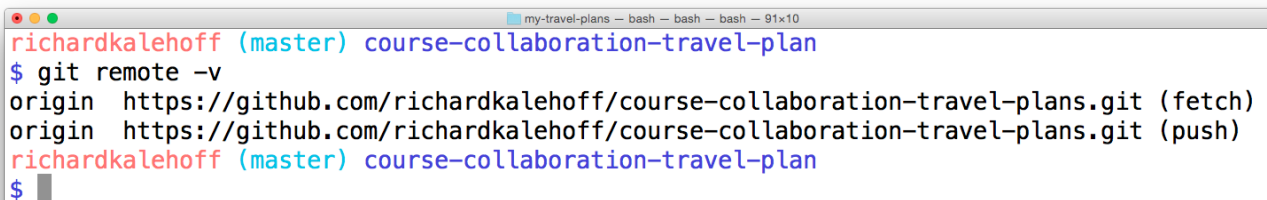
## Including Upstream Changes

Now that you know about watching your repository let say that you're watching it and you get notified that some commits have been pushed to the original, source repository. How do you go about getting those changes into your fork of the repository? If you want to keep doing development on your fork then you'd need your fork to stay in sync with the source repository as much as possible.

Let's see how we can get these changes from the remote repository into our repository.

Incase Lam starts making changes to her project that I won't have in my fork of her project, I'll add her project as an additional remote so that I can stay in sync with her.

In my local repository, I already have one remote repository which is `origin` remote.

A terminal window titled 'my-travel-plans - bash - bash - bash - 91x10' showing a user named 'richardkalehoff' in the 'master' branch of a repository named 'course-collaboration-travel-plan'. The user runs the command '\$ git remote -v', which outputs 'origin https://github.com/richardkalehoff/course-collaboration-travel-plans.git (fetch)' and 'origin https://github.com/richardkalehoff/course-collaboration-travel-plans.git (push)'. The prompt '\$' is followed by a cursor.

```
richardkalehoff (master) course-collaboration-travel-plan
$ git remote -v
origin https://github.com/richardkalehoff/course-collaboration-travel-plans.git (fetch)
origin https://github.com/richardkalehoff/course-collaboration-travel-plans.git (push)
richardkalehoff (master) course-collaboration-travel-plan
$
```

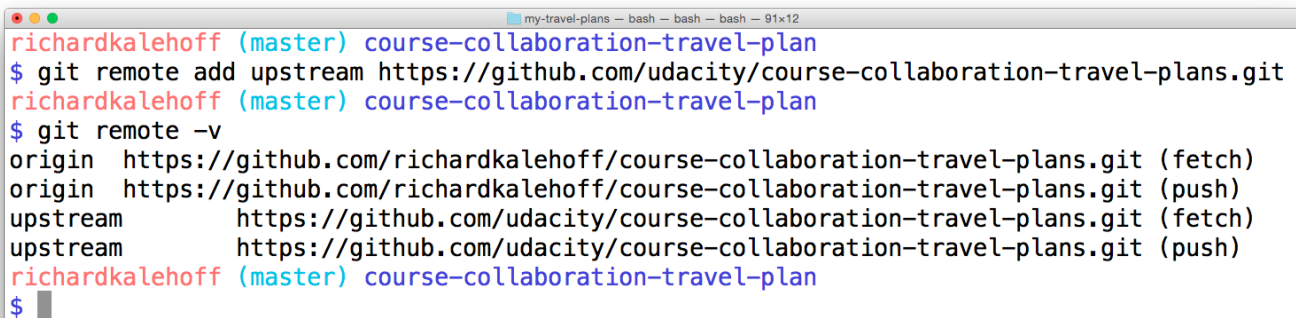
The terminal application showing the existing connect to the remote repository. This is my remote repository and has the shortname `origin`.

Remember that the word `origin` is just the *default* name that's used when you `git clone` a remote repository for the first time. We're going to use the `git remote` command to *add a new* shortname and URL to this list. This will give us a connection to the source repository.

```
$ git remote add upstream https://github.com/udacity/course-collaboration-tr
```

Notice that I've used the name `upstream` as the shortname to reference the source repository. As with the `origin` shortname, the word `upstream` here is not special in any way; It's just a regular word. This could have been any word... like the word "banana". But the word "upstream" is typically used to refer to the source repository.

Let's check out what the list of remotes looks like now after adding this new remote:

A terminal window titled 'my-travel-plans - bash - bash - bash - 91x12' showing a user named 'richardkalehoff' in the 'master' branch of a repository named 'course-collaboration-travel-plan'. The user runs the command '\$ git remote add upstream https://github.com/udacity/course-collaboration-travel-plans.git'. Then they run '\$ git remote -v', which displays the following output:

```
origin https://github.com/richardkalehoff/course-collaboration-travel-plans.git (fetch)
origin https://github.com/richardkalehoff/course-collaboration-travel-plans.git (push)
upstream https://github.com/udacity/course-collaboration-travel-plans.git (fetch)
upstream https://github.com/udacity/course-collaboration-travel-plans.git (push)
```

The terminal application show both information about both remotes - `origin` and `upstream`.

## Origin vs Upstream Clarification

One thing that can be a tiny bit confusing right now is the difference between the `origin` and `upstream`. What might be confusing is that `origin` does *not* refer to the source repository (also known as the "*original*" repository) that we forked from. Instead, it's pointing to our forked repository. So even though it has the word `origin` is not actually the original repository.

Remember that the names `origin` and `upstream` are just the default or de facto names that are used. If it's clearer for you to name your `origin` remote `mine` and the `upstream` remote `source-repo`, then by all means, go ahead and rename them. What you name your remote repositories in your local repository does not affect the source repository at all.

```
richardkalehoff (master) course-collaboration-travel-plan
$ git remote rename origin mine
richardkalehoff (master) course-collaboration-travel-plan
$ git remote rename upstream source-repo
richardkalehoff (master) course-collaboration-travel-plan
$ git remote -v
mine      https://github.com/richardkalehoff/course-collaboration-travel-plans.git (fetch)
mine      https://github.com/richardkalehoff/course-collaboration-travel-plans.git (push)
source-repo https://github.com/udacity/course-collaboration-travel-plans.git (fetch)
source-repo https://github.com/udacity/course-collaboration-travel-plans.git (push)
richardkalehoff (master) course-collaboration-travel-plan
$
```

Using the `git remote rename` command to rename `origin` to `mine` and `upstream` to `source-repo`.

## ⚠️ Resetting Remote Names ⚠️

The image above demos the renaming of the remotes, but I have returned them to their default/defacto names of `origin` and `upstream` with the following commands:

```
$ git remote rename mine origin
$ git remote rename source-repo upstream
```

## Retrieving Upstream Changes

Now to get the changes from upstream remote repository, all we have to do is run a `git fetch` and use the `upstream` shorthand rather than the `origin` shorthand:

```
$ git fetch upstream master
```

```
richardkalehoff (master) course-collaboration-travel-plans
$ git fetch upstream master
From https://github.com/udacity/course-collaboration-travel-plans
 * branch          master      -> FETCH_HEAD
 * [new branch]     master      -> upstream/master
richardkalehoff (master) course-collaboration-travel-plans
$
```

The terminal application showing the results of doing `git fetch upstream master`. A new branch is added to the local repository.

#### QUESTION 1 OF 2

Now that you've added a connection to the new `upstream` remote repository, if you run `git fetch upstream master` will that update *your forked repository* on GitHub?

Yes

No

SUBMIT

Now that we've fetched all of the changes from the upstream remote repository, let's do a log to see what new information we have in our local repository. I'm using the following `git log` command to make sure I display *all* commits from *all* branches (including remote and tracking branches!):

```
$ git log --oneline --graph --decorate --all
```

```
my-travel-plans — bash — less — 94x15
* 52e493f (upstream/master) Add Mountain View to visit Udacity HQ!
| * bb3b625 (origin/include-richards-destinations, include-richards-destinations) Add destination to Scotland
| * af476f5 Add destination to Paris
| * 6d0ae70 Add destination to Florida
|/
* 1c12194 (HEAD -> master, origin/master, origin/HEAD) Add animation to destination headings
* 9cb2c78 Style destinations
* dcbe7bd Add starting destinations
* 44d48a6 Initial commit
(END)
```

The terminal application showing the log of my local repository after fetching the `upstream` remote's changes.

It can be a bit difficult to read with the wrapping of the commit messages but you should be able to see that there is now an `upstream/master` remote branch that is ahead of the local `master` branch. `upstream/master` is on commit `52e493f` while the `master` branch is on commit `1c12194`.

We can use the `upstream/master` branch to keep track of where the source repository's master branch is. We can now get any changes that are made to the source repository's `master` branch by just running `git fetch upstream master`.

Using `git fetch upstream master` pulled in the changes from the `master` branch on the `upstream` remote repository.

What single command would we use if we want to fetch the `upstream/master` changes *and* merge them into the `master` branch?

Enter your response here

SUBMIT

To push these new changes from the Lam's repository, we don't want to run

`git push origin upstream/master` because `upstream/master` is not a local branch. To get these changes into my forked version of her project, I could merge `upstream/master` into an existing branch (like the local `master` branch) and push that.

```
# to make sure I'm on the correct branch for merging
$ git checkout master

# merge in Lam's changes
$ git merge upstream/master

# send Lam's changes to *my* remote
$ git push origin master
```

## Recap

When working with a project that you've forked. The original project's maintainer will continue adding changes to their project. You'll want to keep your fork of their project in sync with theirs so that you can include any changes they make.

To get commits from a source repository into your forked repository on GitHub you need to:

- get the cloneable URL of the source repository
- create a new remote with the `git remote add` command
  - use the shorthand `upstream` to point to the source repository
  - provide the URL of the source repository
- fetch the new `upstream` remote
- merge the `upstream`'s branch into a local branch
- push the newly updated local branch to your `origin` repo