Now that we have `hello` tracked, let's get ready to share it by

Signing up for github.com

1. Sign up on the form on the right: github.com
2. If it asks about a plan, pick the free one.
3. You're all signed up!

Github hosts our repositories -- it makes it so that we can reach our code online. We can put our git repo there by first making a place for the repository.

Make a repository online

1. Click on the plus sign by your username.
2. Click on `New Repository` from the drop-down.
3. Let's name our repository `hello`, just like our directory name. Matching the name of the repo online with our repo locally helps us stay organized.
4. Click 'Create Repository'.

Connect the `hello` repository on our computer to it's new online home.

1. Once you've created `hello` online, github will give you commands you can use to link the `hello` on your computer to the one we just created online. It looks something like this:

```
git remote add origin https://github.com/pandafulmanda/hello.git
git push -u origin master
```

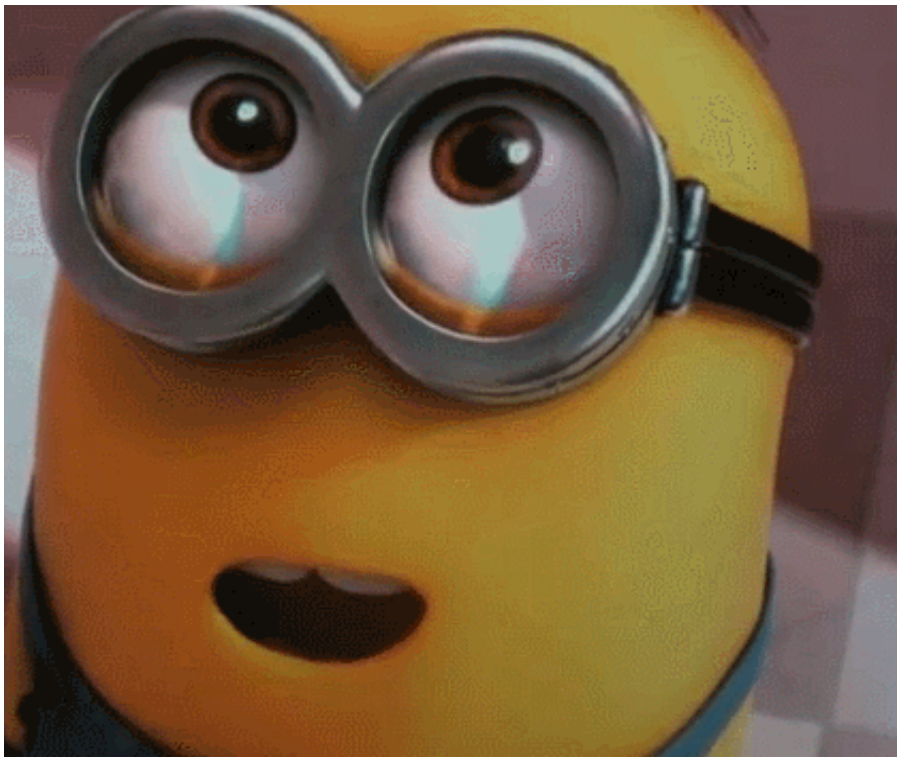
You can click on the clipboard button on the right to copy the code to your clipboard.

2. Press `Enter` to run the code. Now, if we refresh the page for the `hello` repo online, we'll see our code!

   So what did we do in that last step?

   i. `git remote add origin blahblahblah/hello.git` tells our git repo that it has a remote home by the name of origin at blahblahblah/hello.git

   ii. `git push -u origin master` tells our git to push up to that remote home the master version of our code!



Let's try this again with our site project!

Making our site project a git repository

One way to share our site's code and see it online is to make it a git repository and put it on Github. Let's start with making our site into a git repo (short for repository).

Like we did with our `hello` directory,

1. **c**hange **d**irectory into the project directory.
2. Tell **git** to **init**ialize in the directory.
3. Tell **git** to **add** all the files in the directory to the stage.

   You can tell it to add all by doing:

   ```
   git add .
   ```

   The "." means everything in the current directory.
4. Tell **git** to **commit** the changes with a **m**essage describing what we are doing.

Now, we have the code for our site tracked. We can share it on Github by:

1. Creating a new repo to match it on github.
2. Add the remote and push by using the code from the github page
3. Viola!

Hosting our site on github

Something cool we can do, is tell github to host our site online for us. We can do this by:

1. Make a new branch locally called `gh-pages`

   ```
   git branch gh-pages
   ```
2. Checkout the `gh-pages` branch

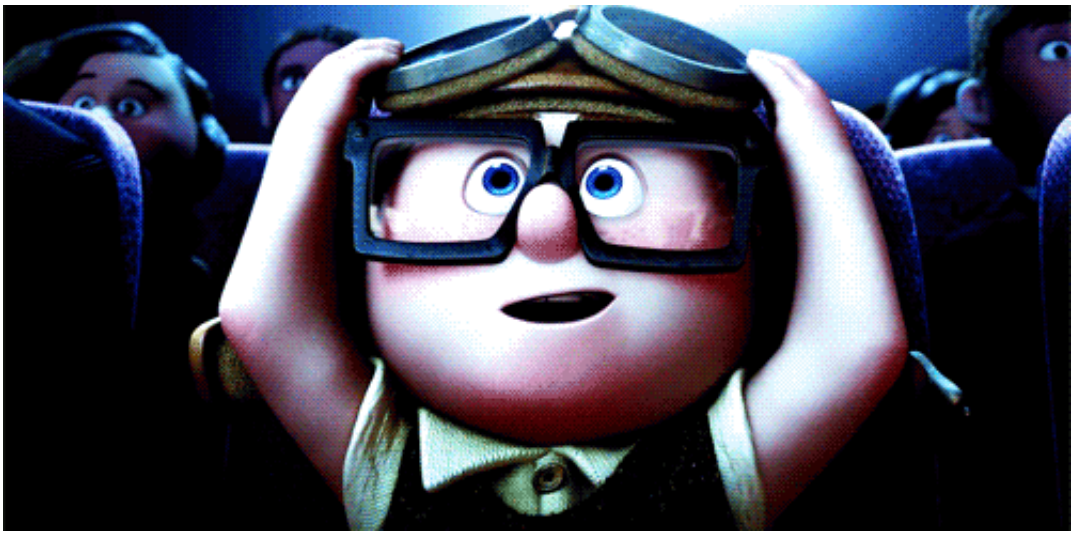
   ```
   git checkout gh-pages
   ```

   Remember when we used the `checkout` command before to change timelines? Here's we are making a new copy of our code, like making a new parallel universe based on our `master` version.

3. Push the `gh-pages` to github so that there is a `gh-pages` branch/parallel universe on github as well.
4. Github will then process our code, and host it at an address in this format:

```
my-username.github.io/my-repo-name
```

That's it! You're live and online!



Making Updates

1. Right now, we are in the `gh-pages` parallel universe. We want to make updates through our `master` universe for now. This means we need to switch ourselves back into `master`.

```
git checkout master
```

2. Make updates to your site and save your code.
3. Track your change as before:

```
git add .
git commit -m "your descriptive message here"
```

4. Push these changes to our `remote`.

```
git push origin master
```

We can leave out the `-u` flag here. That was a special flag we use for a new online repo.

5. Now if you refresh, your code should be updated!

Merging changes into another branch

How do we make our hosted site update as well? We need to combine changes from our `master` branch into our `gh-pages` branch and push the update online.

1. First, change into `gh-pages`

```
git checkout gh-pages
```

2. Then, we can tell git to merge what is in `master` into where we are, `gh-pages`

```
git merge master
```

3. The terminal will tell us that `master is merged into`gh-pages`. That means the tracked changes we made on`master`have now been brought into`gh-pages` as well.

4. Push the updated `gh-pages` to our remote `gh-pages`

```
git push origin gh-pages
```

5. Our site is updated!