GitHub Lab

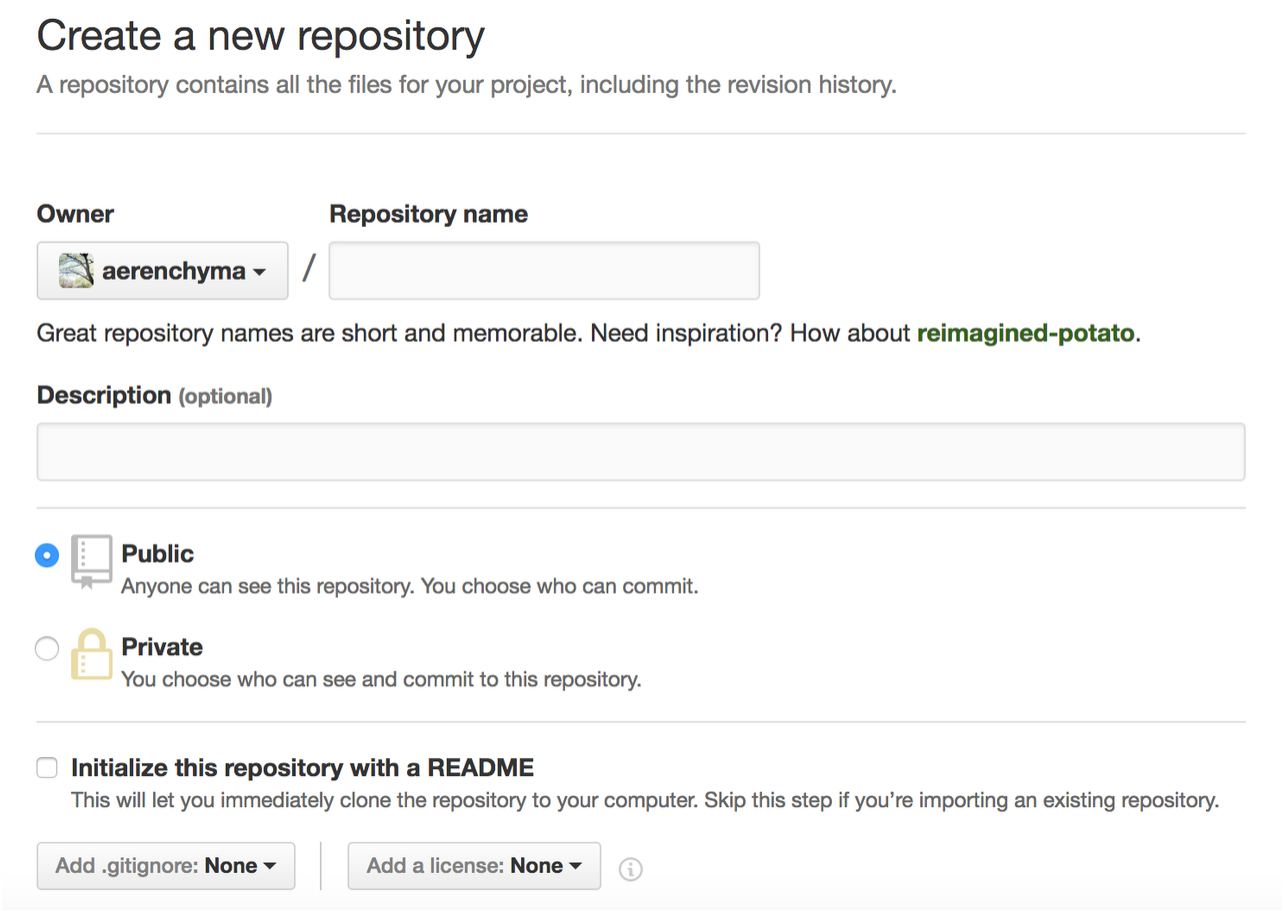
Cloning your repository to GitHub

"OK, I have this project with a Git repository that I really like, but I want to be able to work on it on my computer at work, not just my computer at home. What do I do?"

- This is similar to the solution you are currently in. You have a project with a history in a Git repo *on your computer*, but it's only accessible on your computer. Git, and GitHub, and the tools that Git gives you, can help you with this.
- If you create a new (empty) repository on the GitHub website and use the proper Git commands and URLs to `push` your repository to GitHub, you'll be able to access it from any other computer that has Git! (We'll talk about how in a moment.)
- "GitHub repository" — place online (on the GitHub website) that holds a copy/clone of a **Git repository** that you have on a computer.

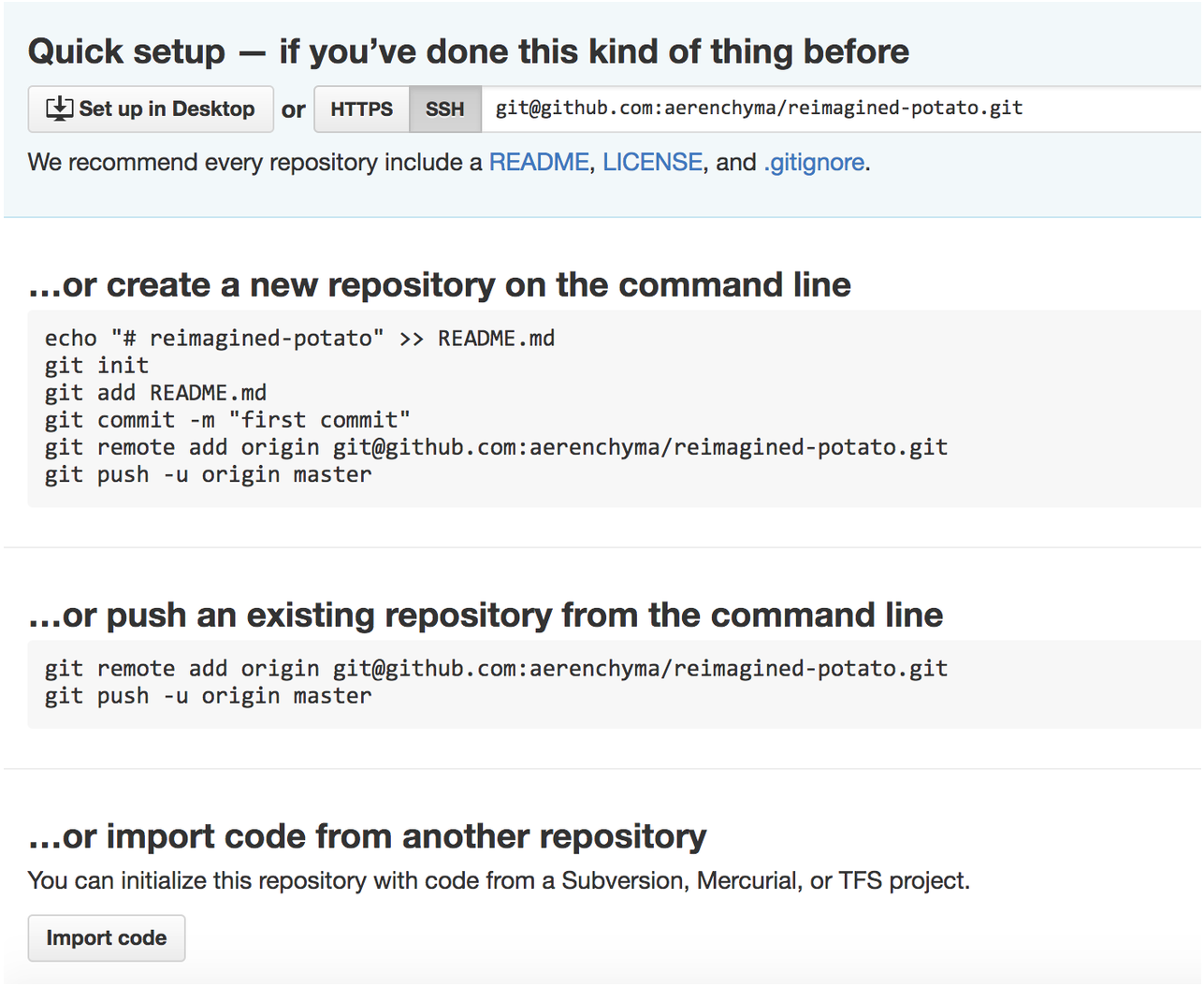
Follow these steps:

- If you don't have one already, create an account on github.com. If you create one using your umich email address, you will be able to get educational license where you can have private repositories, but for the purposes of this class, that's not necessary.
- Once you have crated an account, create a new repository on your GitHub account that will be specifically for this project (the toy project you created above): To do so, go to https://github.com/new once you're logged in to GitHub or click the New Repo button. It should look something like this:



Fill in any unique name and brief description (toy-project_<your initials> might be a good name, it'll be clear and meaningful for you). Keep the repository public. **DO NOT click the checkboxes to add a readme, gitignore, or license right now.** We're going to follow a specific step-by-step process to get used to git and github, and these will force us to follow a slightly more confusing process.

Then click the green **Create Repository** button at the bottom of the screen. You should be taken to a screen that looks like this:



GitHub gives you a lot of choices about what to do next. But they can be a bit confusing if you don't know what they mean.

Their second header pertains to our current situation. We have a repository, but we want to clone it to GitHub.

- Open up your command prompt and make sure you have `cd` -ed to that directory

- `git status` to check out what the status of things and make sure you have committed all the changes and your working tree is clean.

Still inside the same directory where your git repo is, type:

- `git remote add origin` + **whatever your link is (GitHub will tell you on this screen!)**

    - For example, for mine above, since I chose to call my GitHub repo **507_git**, and I'm signed in to my GitHub account, it shows me that I should type (or even just copy right from their screen up there ^ Note that I switched to the http form of the address, so I don't have to set up ssh right now):
    - `git remote add origin https://github.com/klasnja/507_git.git`

This means, translated to English:

> *I'm designating a remote place to store my code and connecting it to this git repository on my computer.*
>
> *I'm calling the remote place `origin`, which is the default name for a remote repository where code lives. This is my code, and my git repo, and my GitHub repo and my GitHub account, so I'm calling it **origin** as convention dictates.*
>
> *This is the special GitHub url that specifies where the **remote place** is located on the internet. It's now connected to my git repo on my computer that I created earlier.*

Finally, you want to *push* the code that you have in your git repository, along with all the history and commits that go with it, TO GitHub (so you can later access it somewhere else).

- `git push -u origin master`

This means:

> *Push all of the data currently in this git repository to the remote place called **origin** (on the main "master" branch, which is the only one we are going to work with right now). the **-u** means you're prompting for a logy in. It may prompt you for your GitHub username and password.*

You should see something like this:

```
whitehead:test-repo klasnja$ git remote add origin https://github.com/klasnja/507_git.git
whitehead:test-repo klasnja$ git push -u origin master
Username for 'https://github.com': klasnja@umich.edu
Password for 'https://klasnja@umich.edu@github.com':
Enumerating objects: 10, done.
Counting objects: 100% (10/10), done.
Delta compression using up to 4 threads
Compressing objects: 100% (7/7), done.
Writing objects: 100% (10/10), 817 bytes | 817.00 KiB/s, done.
Total 10 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1), done.
To https://github.com/klasnja/507_git.git
 * [new branch]      master -> master
Branch 'master' set up to track remote branch 'master' from 'origin'.
whitehead:test-repo klasnja$
```

That means it was successful! It's online! Let's take a look at it: https://github.com/klasnja/507_git

Our stuff is in the cloud!

Now you can work on the code from different computers, share code with others, and rest easy that your project isn't going to disappear from the earth when you spill a big cup of tea on your laptop (as I did a little while ago).

Pulling changes from GitHub

Imagine something has changed—you worked on the file at work and now you're at home on a different computer. We can simulate this by editing a file or creating a new file directly on Github. I'll just create a new file.

Now we want to update your local repository with the changes from the cloud.

- First, let's check if there are any differences between what's in the cloud and what we have on our computer. To do so, go back to the command line and do the following:
  - type `git remote update` (this will update what git on your computer knows about the remote repo)
  - type `git status` (now you see that your local repo is behind the repo on GitHub)
- To bring them back in sync, type `git pull`.

Take a look at your project directory. What happened?

You got the changes from the repo!

Cloning a git repository to a second computer

To work on the same repository on a second computer, you'd do the following:

1. `cd` into a directory where you want to create a clone of your repository
2. `git clone https://github.com/klasnja/507_git` (or whatever the URL was for your GitHub repo). *this will create a directory called "507_git", create a copy of*

the full repo in that directory (under .git/), and check out a working copy of current version of all tracked files, so you can work on them.

Exercise:
Now change something in first_file.py on your computer and get those changes into GitHub. What will you need to do to accomplish this?

Collaboration Scenario 1: Collaboratively working on a repo

A key reason why Git is useful, beyond keeping track of your own files, is that it helps . you collaborate. We'll do this by manually adding collaborators to a repo. This gives your collaborators "push rights," allowing both you and them to work on the same code base.

we'll also, along the way, see how we create a repo by starting with a repo on GitHub.

The the following with a neighbor. One person will be A, the other person B.

- A: create a GitHub repo
- A: add B as a collaborator
- A: clone it locally
- A: create file hello.py that prints your name:
  - "My name is ___"
- A: add, commit, push

- B: clone the repo locally
- B: edit hello.py so that it also prints your name
  - "My name is __"
- B: add, commit, push

- A: **pull** the changes (`git pull`)
- A + B: check the git log

Conflicts

- Both A + B, at the same time, add a line under the one where your name prints:

    - print out: “I was born in ____”
- It’s a race: add, commit, push!
    - What happened to the person who pushed second?
    - That’s a conflict!

Resolving Conflicts

- If you have a conflict, you need to do this:

```
git fetch
git diff master origin/master
git merge
```

```
git fetch
```

fetches changes from the remote (origin) and keeps them in a “staging area”

```
git diff master origin/master
```

compares the changes between your local repository (master — which is the default “branch”, more on that later)(note that you have to have “committed” the changes, it won’t compare with untracked changes)

and

the remote repository (origin/master) — origin is the name of the remote repo, master is the name of the branch on that repo

At this point you have some choices, you could get rid of your local changes, you could decide that your version is the best one and overwrite the remote with yours, or you could try to merge the two.

We’re going to look at the “merge” option. (Not covering the others, but Google will happily tell you how to do them.)

Automatic merge vs Manual merge

- sometimes, git can figure it out (if changes are cleanly separated)
- Often, git can’t figure it out, but it tries to help you figure it out

Branching

```
git branch <my branch name>
git checkout <my branch name>
```

Creates a new “branch” with that name, then switches your local machine to work on that branch. While you are “switched” to that branch, all of your adds, commits, and pushes will apply only to that branch, and not to ‘master’. For example:

```
git branch feature-x
git checkout feature-x
```

Actually you could have done this all at once:

```
git checkout -b feature-x
```

Then you work on feature-x for a while, and when it’s done, you merge the changes back into the master. All the merge stuff we saw before applies—sometimes is works automatically, sometimes you have a bit of work to do to reconcile the changes. Here’s how you merge:

```
git checkout master
git merge feature-x
```

This switches you back to the master branch, and then merges the changes from the feature-x branch into the master branch.

Now we’re done with the feature-x branch. So you can delete it.

```
git branch -d feature-x
```

Exercise: Extending our example

- Start with a merged, clean, nice repo that both partners have.
- A will keep working on the “master” branch
- B will create a new branch called “about-me”

```
git branch about-me
git checkout about-me
git status
```
What does status tell you?

Both make the following changes:
- Add favorite food
- Add favorite type of music
- Add astrological sign

When done, we'll need to merge.

Both of you: add, commit, push
B will have to specify where to push as follows:
```
git push -u origin about-me
```

Both of you: do **git pull**
Why didn't a conflict happen this time?

Both of you: check out the other branch (need to make sure all your changes are committed locally first)
```
git checkout master
git checkout about-me
```

Look at the contents of the file. What happened now?
Switch back, what happened now?

A: you will "merge the branches"
A: make sure you're in the master branch (git status, git checkout if needed)
A: do the merge!
```
git merge about-me
```

Look at the file—it (probably) has errors. You need to fix them!
After fixing, add, commit, push - now the master branch is up to date with changes from the branch.

B: switch to master and pull. Verify that you're in sync with A.

The key takeaway, for individual projects

[git init] or [git clone]
forever:
    ((Create and edit files))
    git add
    git commit -m "a real message"
    git push
... and throw in some git status and git logs every once in a while