

GitHub + working with remote repos

Class 3 of 5

Welcome back.

Welcome back to class – I hope everyone has had a good week.

How did practicing with commands go?

I've put a link to the “answers” up on the Moodle course.

We'll have a couple of different practices for you to do this week in working with GitHub.

Today is all about GitHub, and working with remote repositories.

GitHub + working with remote repos

- GitHub intro
- Set up account/keys
- Clone and/or Fork
- Push to & Pull from GitHub
- Set up an existing repo

We'll take a look at GitHub, I'll show you how to create and place some SSH keys onto your GitHub account so you will be able to quickly push and pull repositories from GitHub.

I'll talk about cloning & forking – which are slightly different ways of making copies of repositories on GitHub

Then we'll look at pushing and pulling which is how you move changes – or receive changes - to your repository from GitHub.

And finally – how take an existing project – like the Moby Dick one I have been working on - and set it up in GitHub.

GitHub

- <https://github.com/>
- Hosts git projects
- Large, popular
- Provides access control
 - Makes sharing & collaboration much easier

Finally, let's talk about GitHub. GitHub is a site that hosts git-tracked projects. It's hugely popular – so many popular open source libraries and projects are hosted here. Node, d3, Ruby on Rails, Bootstrap & FontAwesome, Reactd3, Homebrew. So very heavy hitters are here – and it's open to anyone to have an account and connect with people through sharing code.

So I can look at all the source files here, or maybe just get the files to start a project with. But also the very important thing about GitHub is that if I were a developer who wanted to contribute back to the project, I could. Now, I say that with some caveats – for example, git is hosted on GitHub, but the developers there don't use GitHub as a way to contribute – but for the most part, contributors can make contributions through GitHub, because GitHub allows for access control. You can easily add other people to your project, but there's also a system called forking and pull requests that make people don't have to know each other/trust them ahead of time in order to make contributions.

GitHub is also good for just looking at code or using someone else's project as a base for a new project. I could make clone of Ruby on Rails here, and then start contributing, or just look at the code.

Libraries on GitHub

Websites or web-based applications

- NYPL
- NCSU Libraries
- Omeka, DSpace and Fedora Team

Scripts or code snippets

- Custom css for LibGuides from Virginia Commonwealth University
- Snippets of Javascript for Summon – Grand Valley State
- EZproxy configuration files
- XSLTs

Multiple libraries also have a presence on GitHub – and here obviously I actually mean academic, public libraries, not software libraries like JQuery.

Here are just a few examples – I know in the areas I work in, which is web development, NY public library are doing really interesting things and posting – like these crowd-sourcing NYPL-Spacetime, where they are eliciting knowledge about the cities' history through apps that have map and building info - so the point of NYPL putting the code for this on GitHub I believe is probably to contribute with collaborators but also so other libraries can build on top of what they are doing for their own cities.

I know NC State Libraries are also pretty active... I am sure you all have good examples of work libraries are doing. So we have libraries sharing EZProxy configurations, LibGuide styling; Portland State had some interesting Drupal modules to spin up sort of a Do it Yourself reference site for users.

I have Omeka and Fedora here – I put Fedora team because I'm not sure if the Fedora source files are in GitHub but there are definitely tools and other programs to extend Fedora that are hosted on GitHub.

Let's take a look at Omeka, it's a CMS for online digital collections – I know a lot of libraries use it – I'm not familiar with it – but lets take a look at it.

Git clone

- Git clone *repository_url_here*
- Copies a git project to your machine
- HTTPS vs. SSH

Demo of cloning: 11:30 – 16

There is one thing that you need to do to be able to use git clone, and that's setting up a key on your computer. So there are ways around doing this, but it's going to save you time later, and plus, you will probably have to do something similar with other remote servers you are communicating with – be it through git commands or for other tools or languages – so it's worth doing if just to learn.

You might notice that when I go to clone a project in GitHub, there's some confusion as to whether to do with HTTP or SSH. These are different internet protocols; GitHub gently recommends HTTP because sometimes SSH ports can be blocked on a network. I have never had this problem, and generally SSH is easier once you have it set up because you don't have to keep entering your GitHub user name and password.

Using SSH

Setting up SSH keys: [instructions](#)

- Create an SSH key pair
- Copy the public key
- Paste it into GitHub

I am going to walk you through setting up an SSH key in GitHub – and ask you to set up the key for next week – but we also have instructions (show)

Essentially, you'll use a command that will automatically create a public key and private key. We'll want to copy the public key over to GitHub. The commands are ways to get the contents of the key onto your clipboard – really it's a long unique hash, plus your email address.

And then once you have the key copy, you go to GitHub and put it in your settings. Remember this key is going to identify you and your computer – so you might want to name your key in a way that reminds you which computer it's coming from.

Fork a repository

- Fork = clone within Github
- Can issue “pull requests” to original project
- To work on a forked project, you clone your forked repository to your local machine

There's an extra step you might take when cloning a repository from GitHub, and that is something called “forking”

Forking isn't a git command, or anything in git itself, really. It's a software term regarding making a copy of a project – here in GitHub, it means that when you fork a project, GitHub will give you a copy of a repository in GitHub itself.

This way you can contribute back to a project if you aren't one of the main contributors, because you can issue something called a pull request through GitHub – GitHub controls the access – and the person who is in charge of the code on the other end can look at what you've done and decide whether or not to merge it into their project or not.

Fork a repository

- Demo
- Fork + clone

DEMO: 16 – 28:48

Why Fork?


- Fork when you intend to work on the repository & now your workflow will include GitHub
- Clone when you are only interested at looking at your files locally

When do you fork, or when do you just clone? I think if you just want to have a look at the code and don't intend on collaborating on the project, or if you don't think you will even make it a project yourself on GitHub, then use clone.

Use fork if you do think you will be using this repository in GitHub, because so many things are already set up for you. Remember our Moby Dick project? If we started this on our local computer and we have to do a few more steps to get it to GitHub – it's super simple, but forking does save us some steps if we're starting from a GitHub project to begin with.

GitHub Push & Pull

- Git Pull:

- GitHub  Local repository
- Fetch + merge = pull

- Git Push

- GitHub  Local repository

Once we've got our repository on our local machines, we're gonna want to be able to move our work back and forth to GitHub. Push and pull are two big commands here. Pull – you're going to pull down any new changes from GitHub itself – pull itself is actually short hand for 2 commands: fetch and merge, which will talk more about next week.

Push means to send the changes you have made locally to the GitHub repository. Now you saw that we have a remote automatically added with GitHub. We can have more than one remote that we push to and pull from.

GitHub Push & Pull

DEMO

Push and pull demo at 31:20 – 36:30

Kate's mistake, Round 2: Mistake made at minute 34. Post on the forum if you know what's going on! (Hint: it has to do with branches, we will talk about this in the next class)

Keep a fork up to date

- Add a remote
- Git remote add *upstream* <http://someurlhere>
- Git fetch *upstream*
- Git merge *upstream/master*

If you've forked a branch, and if you're contributing back to the original project, you'll want to keep up to date. We can do this by adding another remote. So when we run the command `git remote` now, we see have the remote for origin – the default one – but we can set one up for the original branch by declaring a new remote. And it's a convention to name the alias in this situation "upstream"

```
Git remote add upstream url
Git remote -v
```

```
Git fetch upstream
If not on the branch for merge, check it out
Git merge upstream/master
```

This merge "upstream/master" is taking the changes on this special upstream/master branch that refers to . More about branches and merging next week.

Move an existing project to GitHub

- Set up empty repository in GitHub
- Git remote add *origin* <http://someurlhere>
- Git push -u origin master
 - Origin = remote alias
 - Master = branch

There are times when you start a project on your local machine and you want to move it to GitHub. All you need to do is create an empty repository on GitHub, and then add the remote address on your local project.

This assumes, of course, that you've already set the local project up in git and have made commits – I don't think you can push a project until there is at least one commit.

Show demo with moby dick

Make sure you don't add a README

```
Git add remote origin your_empty_github_repo_url_here
Git push -u origin master
```

The `-u` sets up a tracking reference to the remote branch, and you can just say `git push`, `git pull` in the future, instead of saying `git pull origin master`. Or `git push remote_alias branch_name`. Doesn't always need to be `origin master` although by default it will probably start out that way. So for now I will at least keep using these alias names for push and pull. To be honest I still do that with my repositories because I want to be very aware of what I'm doing... Totally up to you.

Also on GitHub

- Issue Tracker
 - Gist
 - Wiki
 - [Code of conduct](#)

See around 45 min for discussion of issue tracker, wikis, and gists. I didn't have time to talk about code of conduct – you can add a code of conduct to your project through a template.

For next class

- Mid-class survey
- Set up GitHub account + SSH keys
- Practice fork + cloning

See <http://hklish01.github.io/gettingtoknowgit/class3.html> for instructions.