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.

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.

**About GitHub**

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**

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.

Demo: *Git clone* [*git@github.com:omeka/Omeka.git*](mailto:git@github.com:omeka/Omeka.git)

*(show the download/zip options) – once you know how to use git clone it’s the easiest way.*

*show git log*

*back to GitHub, show commits, browse files.*

So 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.

(show git commit slide)

Ok, this slide is mostly for reference. Git clone is a command to bring a project down to your local machine from a remote.

So 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.

http://jr0cket.co.uk/2016/05/ssh-or-https-that-is-the-github-question.html

So 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 (show) 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**

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.

I will show you what this looks like:

Here is a repository I might want to contribute to. You’ll notice it’s not necessarily a project in in of itself but really built for building a list to easily share and add to from people who don’t really know each other.

https://github.com/devfreebooks/devfreebooks.github.io

Point out that there are ~15 contributors. Actually a lot – but I am not one of them. I could contact the person who maintains this site and ask him to become a contributor. But that’s a lot to ask – they would have to trust me, because whatever I push could hurt the site.

But if I fork this, I can make a pull request through GitHub, and GitHub will notify the maintainers here, and the maintainers can look at my code and decide what to do with it – if they like it, they can merge it in, but they can take their time to test it.

So, if I want to contribute back to this – or even, say, take what they’ve done, and make a site/repository about, lets say learning data visualization – I’m going to fork it, so I will have a local copy to work with, and a GitHub repository that’s already set up to acknowledge the connection between my local copy and this repository.

So, let’s fork this repository – you’ll see that I get this overview of code in my repository, and I can look back at the other people’s commits. I want to add to it – I can do this on GitHub, but realistically, you’ll want to bring it down to your local machine to work on – so we can say git clone

(clone to local)

show remote –v

This “remote” command shows you a list of any external repositories you might be working with. Whenever we work with remotes, we use this alias to communicate with it – here, the alias is “origin”, and it’s one that GitHub set up by default – so instead of always having to write out the URL, we can just use this alias – and can add as many as we need.

*show log of older commits*

**Why Fork?**

So 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**

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 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 remore that we push to and pull from.

DEMO

*Git pull* (already up to date)

Make a change, commit

*Git push origin master*\* (if fails may have wrong branch)

So how do we keep up to date with the main project? We need to pull down from the main site often. Branching helps with this.

**Keep a fork up to date**

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**

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 url

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**

**Git ignore**

Sometimes there are files you don’t want to track. This could be because you have confidential info – like configuration files, which may have passwords – or because they are huge, clunky application files that aren’t needed for the program, and are only used for local development to compile files – things like sass cache directories that get created whenever sass is compiled into CSS, or files that get downloaded with package managers like with NPM install.

(Next slide)

The good news is that there are templates for gitignore files out there on places like GitHub – so depending on the language or framework you are using someone may have already created a gitignore file you can edit for you own purposes, but they have an indication of where the sensitive documents are, or where are the files that get created that aren’t worth tracking.

Basically a .gitignore file is a list of exceptions. By default, git is going to track a change to ANY file it sees under your git repo. The .gitignore file tells git, don’t even bother with any change you see in this directory, or has this naming convention, etc