Intro

Talk to HK about Q&A forum

Other things to mention:

Lecture notes/recordings on Moodle (note: I actually don’t know where recordings live)

….

**Class overview**

We have a 5 week class, and each week we’re going to be building up our Git skills.

So we will work our way up to being able to work with git on your local computer including installing it, using basic commands and best practices, to learning about GitHub, setting up an account, pulling projects from GitHub to your local computer and pushing them back up, and we’ll talk about collaboration. And we’ll take those skills and put them together for a final project, where we will make a website with git and use something GitHub pages

**Final Project**

The project involves essentially copying a site that’s already made – so for those of you who aren’t familiar with web development, don’t worry, you’re not going to have to learn much code. But for those of you who are familiar, you’re free to do what you like to the code.

Let’s move on to actually learning about git

**What is git & why use it?**

Ok I am going to explain git in general here – with a disclaimer and the first time I learned about git, it was a little confusing. The more you use it, the more it makes sense. Like anything, it takes a bit of practice.

Git is open source, free software that allows you to go back and see earlier versions of your project very easily. You can work on the project – add documents/files, change wording/code in a safe way – by using another version of your project called a branch. That way if the new work you’re doing wreaks havoc on your program, you can easily throw it away and start again.

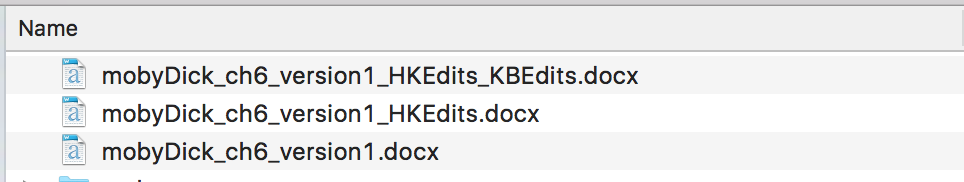
And when you’re working on the project, you add messages and tags that allow you or other people working on the project to get a better sense of the history and organization of the project – it’s all combined in the way git tracks your project, so it’s not like you have to worry about keeping track of this documentation, either. Git does all the tracking for you.

**Git vs. GitHub**

Before we talk more about git, I want to separate out the “GitHub” part of this class. This is something that usually confuses people. Git and GitHub are different things. Git is software you install on your computer or server. GitHub is a web site that hosts projects and files managed by git. You can use git without GitHub, but GitHub is something that only works for projects tracked with git. Also, there are sites out there that do pretty much the same thing as GitHub – like BitBucket. Also, some companies and universities have their own private git server, using something like GitLab. But GitHub is still the most popular, and if you know how to work with it, you’ll know how to work with other places like BitBucket. We’ll talk more about GitHub in another class.

**Back to Git**

Now that we know that we’re just talking about git here, and not trying to wrap our brains around the “GitHub” part of it, lets talk a bit more about git itself. Git is a type of version control. You probably have already used a method of version control –



So in this example, let’s say Heather and I have decided to edit Moby Dick. We’re not using git on it. So Heather makes edits, but she doesn’t want to lose how the original document looked. So she saves a new copy of the project – in this case it’s just the one chapter she’s editing. You might have done this, or something similar, like make a copy of the whole project that have your edits or new work. In either case, we can look at these different files as different *versions* of the project. What version control does is that it takes care of organizing and filing away all those different versions for you, and you can jump to an earlier version if needed.

**What is version control**

I’ve pulled a more formal definition of verison control into this slide – so git is one of those systems that records changes to a file or set of files over time so that you can recall specific versions later.

This quote comes from a great book on git that’s freely available online – it really does explain git in an easy to understand way. We’re not going to go into the history of git in this class, or how git compares to other types of version control, but I highly recommend this book if you are interested. One thing to note about version control software is that people have been using it since the 1970’s, and git was introduced in 2005 (it was created by Linus Torvalds, who also created the Linux kernel). Git has become incredibly popular.

**How Git Helps**

To see why version control/git is so helpful, lets look back at the Moby Dick project. (next slide) Everytime I make a change, I save it to my computer – we’re all familiar with that. But if I have git installed, and have it tracking my project – in this case, the downloaded pages of Moby Dick, I will also save my changes to git – this is called committing, and for this class, we’ll be doing it on the command line.

(next slide) Every time I tell git to track a change, we’re saving an entirely new version of the project. Even if I change just one file, the version we save to git will contain a snapshot of how all the files look at this time.

(next slide) So a really nice – although initially confusing part – about git is that even though you have multiple versions of a project, you really don’t see them all on your computer. Git is tracking everything behind the scenes. Even though technically here we have 3 different versions of Moby Dick – each version has all >800pages – because of the way git works it isn’t really taking up as much space as that on your computer (again, refer to “Pro Git” book about this!) – but you can jump back to see or revert to any of those full versions at any point.

(next slide). If I had tracked – in Git speak, “committed” each of those changes individually, this is what my history would look like in git so far. Git log is the command I use to see the history – more on that next week – and you’ll see that there are notes: I left those myself when I committed the changes. This is very helpful in that I can have an idea of what is going on in each version. The number/letter string preceeding each message is called a hash. This identifies the version, and you can use this to refer to that version of Moby Dick, or whatever project you are working on. You can also use tags, but we’ll be referring to hashes first.

(next slide)

So, that was a lot of jargon – again, it becomes clearer through practice, so don’t worry if that was a little confusing. What I do want to emphasize now is that because of the way git works, tracking changes to your project is a lot easier with git than without it. You don’t have to worry about losing a copy of a version of your project or finding documentation. Also, all the backups are built into the git system (this gets a little more complicated if you’re using a database – again, more on that later).

**Collaboration**

Git is incredibly useful to use on projects where you are just working by yourself. But you can imagine, if it’s good at tracking changes, and helping documentation, that it takes a lot of the pain out of collaboration. When Heather and I are revising Moby Dick, I can easily see the changes she’s made, and git will help me merge her changes with mine. And that’s why it’s so helpful for open source projects, because with a server like GitHub in the middle, you don’t even really have to know the person you’re working with, and if someone’s code breaks the project, you can revert back to a previous version, also known as a commit. We will talk more about collaboration and merging in a future class. For now, lets give you a closer overview of how you work with git on your own machine.

**How you work with git: an overview**

First, you have to make sure you have git installed on your computer. You only have to do this one time (per computer), and we’re going to ask you to install git before the next class. There are instructions on Moodle and we’ll be available by email if you have problems (also available in office hours).

Once you have git installed on your computer, you can ask git to track any projects on your computer. Your project could consist of one file or of all the files on your computer – probably helpful to be somewhere in between. You can even track projects that have already been started – but of course you will only ever be able to revert back to the first time you committed.

So to start a git project you just run git init at the root of your project.

--LIVE DEMO-- *> show finder, move over folder to termail. Then run git init*

Once you have initialized your project with “git init”, you will be ready to make your first version, or commit to the project.

When git is tracking a project - it means git is looking for changes to each file or new files to the project. But it doesn’t really get in your way as you’re working. You have to check in with it - by doing commands or by using a Git GUI app - on the command-line, you’ll type in git status. and git will show you the files that have been changed. At any point, you can tell git to save a version of the project - this is called a commit.

*---Live Demo 🡪 git add . /git commit –m*

I’m going to make an initial commit so I can always come back to this original state. Remember that “committing” = making a version of your project. It’s that snapshot of how all these files looked like at this point in time.

Back to slide

(next slide)

Committing is actually a two-step process. The first step is adding the files – really this is staging the files – and the 2nd step is committing. The reason is this: you can commit every single change you make, but that could quickly become an overwhelming version history. Or, you could work all day and commit all these varied changes you made on Moby Dick all in one commit – but that might not be a helpful history to look back on. Deciding when to commit is a bit of an artform – BUT when we practice we will keep it simple. And staging the files helps you to pick and choose which changes you want to add to a particular commit/version.

*Live demo – make a couple of changes, add them separately*

*Make a couple more changes, add them together*

Ok, the final step I have here is of course not mandatory – I have plenty of git repositories on my computer that I never intend to share with anyone – but the fact is that I *could* share if I wanted to, because git – and GitHub – make it so easy. Git was built to make collaboration easier. So it will be very easy to grab Heather’s changes to Moby Dick and merge them in with my project. And same for her to take my changes to the project. And at the end of the class we’ll have you pushing a git repository to GitHub and people can download/use your code if you want, or collaborate on a project.

**First Step**

Now that you have an overview of why people use git, and an overview of how you work with it, the first step is to install it on your computer.

You’ll go to this page (<https://git-scm.com/>) and click on the computer monitor icon (the site will recognize what kind of computer you are using and offer you the right download)



There’s a link to how to install git on Moodle. It takes us out to a github page that will have information for this class – so this is the side of GitHub that hosts web pages for git-tracked projects, not the side of GitHub that shows you all the files and commits – we’ll see that soon.

For PC people, you’ll be using Git Bash instead of the regular terminal in Windows – it comes with git when you install it.

For Mac users, you may already have git installed.

One thing to be aware of is that you may need to have admin permissions on your computer to install git.

**For Next Week**

(If time, show them how to revert to an earlier commit). If more time show them https://try.github.io/