Class 2 Lecture Notes

Welcome back to class!

I hope you have all had a good week in between when we spoke last.

Thank you to everyone who introduced yourself on the forum – I hope you’ve all had a chance to both introduce yourself and also read a bit about your classmates. There are actually 42 of you now.

We have a great group - we have systems librarians, archivists, web administrators, people with metadata expertise, lot of people that already work with git, some people who are making it a point to learn it to help in collaboration at work. Some people are interested in learning git to contribute back to open source projects, which is also really cool.

Something we’ve also learned from the survey (and thanks again to those who filled it out) is that there is a wide range of experience and knowledge in git. Some people seem fairly experienced, others not as much. We also learned that more people than not wanted to spent class time learning command line. There is a handful of people who didn’t want to spend class time on that – and that’s totally understandable – so what we are going to do is in a moment I am going to let Heather spend about 10 – 15 minutes talking about command line, showing you a demo. For more resources, I’ve put up a few links on our page (see Moodle)

And while we’re on Moodle – I want to remind you again about our office hours. No one came to either mine or Heather’s! So you might remember from last week – we asked you to install git. Did everyone do this? Is it possible that everyone was able to install git and no one had problems? That would be be awesome but it doesn’t sound realistic. You don’t have to come to office hours, of course, but we want to make sure – did anyone try to attend but wasn’t able? Let us know. Also, you can post to the forum and email us.

But as a reminder our office hour info will always be here on this Moodle page.

Also – on our survey we asked people what their operating systems were – I’m interested in hearing from the people who were running Linux RedHat & Ubuntu – did your git install go smoothly? Let us know.

(back to slides)

Today we will be talking about using git on your computer. I know a lot of you are interested in talking about branching and merging – we may get to that today – at least to an introduction in branching – but I want to make sure we have the very basics of using git down first, and trust me, I find branching & merging incredibly important feature of git so we will definitely be spending time on it, that’s why I do want to spend a whole class on it later, but it is something that we may need to touch on for the basics.

So today, Heather will go over some general command line stuff, then I will talk about some basic, very important git commands within the context of working on a project – and the next things just really matter about the time – but we’ll talk about how to undo things, which leads us to some concepts to start/continue building up a understanding for. And if there’s time we’ll talk about git ignore. For next week, we’ll have you set up a project on your local computer and track it with git. There will be more instructions at the end.

For now, let’s start with Heather talking about the command line…

**Commands you will see**

Ok these are the commands we’ll look at today. I’ve put them on this slide and the next, and besides some more advanced ones these are mostly the only ones you’ll have to use, and the only ones you’ll have to use for this class, I’m sure.

So the best way to know about these commands is to understand what they are for and when to use them, but it’s not like you have to memorize them – you can always google to see how to do things, but again, we’re just building up your understanding here.

I’m not going to go over all these commands today – I’m going to walk you through some common commands on a demo,.

And for the practice we’ll have you do before next week, we’ll only have you practice the ones that I demo today. And, also, there are git command resources on the same page that I showed you the command line resources – including a cheat sheet that some people may want to print for handy reference. One of the reasons that I wanted to organize the commands in this way is that if you look at the cheat sheet or other git command references, they are going to mention things like git clone, push, pull, etc, and you may wonder why we haven’t talked about them yet – really, it’s not about building up to using them, it’s just about how we’re arranging this class.

Common commands:

DEMO

Git add – this stages the file, tells git you intend to commit the file. Git commit is the command you use to make a commit, and remember that this comes with a message to yourself, I will go over how to do that. And remember that a commit is a snapshot, a version of the project. And it takes a snapshot of all the files at that place in time, and you can go back to this version. This version is called a commit. So commit is both an action, and a place in time to go back to. You’ll get more used to this as we use git more.

Git log is the command to show you your history – all those previous commits that you or another developer made.

This next set of commands – you won’t use them as much, maybe once per project

(next slide)

Commands used with branches… and remotes

We might get to these today

Won’t get to the remote – so that will be for next week.

DEMO

*Go to ~/Documents/litaDemo/MobyDick*

**Git status**

Git status – I put git status first because you will be using this command all the time. Remember that git runs in the background, especially if you are using command line, so this will be your way to check in with git – you’re asking, if git sees any changes.

*Make a change*

Git add . (show git add filename)

Git commit -m

See what a difference is

*Git diff (hash) (hash)* \*

Can do branch branch

there is an easier way to refer to the most recent commit, it’s by calling it HEAD.

*Git diff HEAD HEAD~1*

We can use HEAD as a point of reference. You can refer back to earlier commits by noting how far away they are from the most recent commit (or just use the hash)

*Git diff HEAD hash*

Something I need to talk to you about is how to deal with text editing on the command line. I said before that if you don’t use the –m shortcut when you are running the git commit command, you’ll have to do something a little different. So this is what I mean:

Git add

Git commit (use with vim)

I

Escape

:w

:q

(Back to slides)

I have the commands for these in a slide, also there is a command cheat sheet in the resources section

So it’s true that an easy way to avoid using vim is to use the –m flag on commit, but you will also see it in the future when you are dealing with merging. Also, I’ve been using sublime to open up and edit my documents on the command line, but I had to set that up through a symlink on my computer – if you don’t have a text editor set up to do this, you may have to use vim to do quick edits. This is especially true if you are working on remote servers that don’t have a user interface besides the command line. It’s worth getting used to using this type of text editor – its not a git thing, it’s a command line thing. Some similar text editors are Nano and Emacs. They use different commands – highly google-able when you are confronted with something like that.

HOW DO I GET BACK TO AN EARLIER COMMIT?

Aka version/snap shot

What happens if you make a change that you actually don’t want to commit. If you want to revert back to an earlier version, you can do this by using the reset command.

If you just want to simply look back on something, and not lose the project history that you have so far, you would need to checkout a commit. This is basically making a branch of your project – so before I talk about this, I want to talk a bit more about what a branch is. For now, let’s just talking about reverting. You’ve made an error, or you just want to start all over again. The way to do this is to reset.

(next slide)

The most common way to do this is to reset the HEAD. Remember that HEAD is the most recent commit. So let’s say you made a change. And you just want to go back to the most recent commit.

DEMO

*Make a change, save to your computer*

*Don’t add or commit it, just reset HEAD –hard*

So you may wonder what would happen if I just used git reset HEAD – it’s going to leave those changes I made there, they are just not staged.

DEMO

*Make a change, save, add, commit.*

*Git reset HEAD~1 –hard*

You generally only reset your commits to HEAD or 1 before HEAD. If some change you’ve made is deep within your project history, it’s just best to go to those files, make a change, and commit. You don’t want to lose all that stuff in between, unless there is something seriously wrong.

OK, those are ways to go back, but really git is much more flexible when you are using branches to either look back at earlier work, or start a new feature that might get complicated. So, let’s just review some concepts before we talk about branches more – and lets get a brief introduction to what branches are.

Back to slides

What is a commit

What is head

Branches – no demo yet - make a new branch, check it out, make changes, can destroy, or merge back in.

(new slide)

Making a branch is also a safe way to look back at previous development – can check out a new branch and reset it to an earlier commit and not lose all the project history in between.

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

For next week:

Practice on your computer with command line, setting up a repository, and making commits. Practice resetting to an earlier commit.

IF TIME:  
Intro to GitHub.

Account

We’ll have you go through the process of getting your computer to talk to GitHub. It has to know who you are before you push and pull. So we will walk you through this.