Working with git on your computer

Class 2 of 5

Welcome back.

Welcome back to class!

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 – 20 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

Working with git on your computer

- An overview of general commands
- Git commands in context of using git
- How do I undo something?
- Git concepts: commit, HEAD, branch
- Gitignore (*post-class note: we did not get to this in this class*)

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.

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 (UPDATE: there was no time). 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.

Git Bash

- Is a shell
- Similar to Mac Terminal or Windows Command Line
- Allows both git commands and linux-based commands

We're using Git Bash on PC to more easily work with git on the command line. Otherwise, some of the commands we'll talk about may not work on the Windows Command prompt.

Git Bash is a shell. A shell is a program that interprets the commands a user enters and then interacts with the operating system based on those commands.

Commands

- pwd
- ls, ls -al

Documentation tip
Git Bash: <command> --help
Terminal or Linux: man <command> q to exit

DEMO: look to video at minutes 7:34 – 13:40

Common commands: creating files & directories and navigation

- mkdir
- cd
- touch
- cp
- mv -> rename and move

DEMO: look to video at minutes 13:57 – 20:40

Common commands: removing files & directories

- rm
- Rmdir
- rm -r

DEMO: look to video at minutes 20:54 – 24:34

Common commands: misc & shortcuts

- clear
- cat
- cd ~ (home directory)
- Past commands: arrow keys, history !<#>
- Tab completion of file names
- Ctrl + C

DEMO: look to video at minutes 24:58 – 29:53

Git Commands you will see

Common commands:
git status, git add ., git commit, git log

Commands for configuring project:
Git init, git config, git remote *add*

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 commands you'll have to use, and the only commands you'll have to use for this class, I'm sure.

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.

Git status – I put git status first because you will be using this command all the time.

Git Commands you will see

Commands we'll work with with branches and moving back through commits

Git branch, git checkout, git merge, git reset, git diff

Commands for working with GitHub/external repos

Git clone, git push, git pull, git fetch, git stash, git pop, git tag, git remote *-v*

(we will look at these commands in future classes – although some of them, like git stash, pop, and tag we may not have time for)

Vim, Nano & Emacs

- Text editors in the command line
- You may run into these as you use git

Vim

- Common commands:
  - i = insert
  - Escape -> gets you out of input mode
  - :q = quit
  - :w = write/save

Use these commands to work with Vim. (see Demo at)

While 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 go back to an earlier commit?

- Revert:
  - Git reset *hash*
- Safely look at an old commit:
  - Git checkout *hash*

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.

How do I go back to an earlier commit?

- Git reset HEAD
- Git reset HEAD --hard

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.

Look at video for demo at: 45:35 -48:30. Also, Kate made a mistake in this portion of the video – can you spot it? If so post to the forum.

How do I go back to an earlier commit?

When you want to go back to a commit before the last one:

- 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.

So this is one way 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.

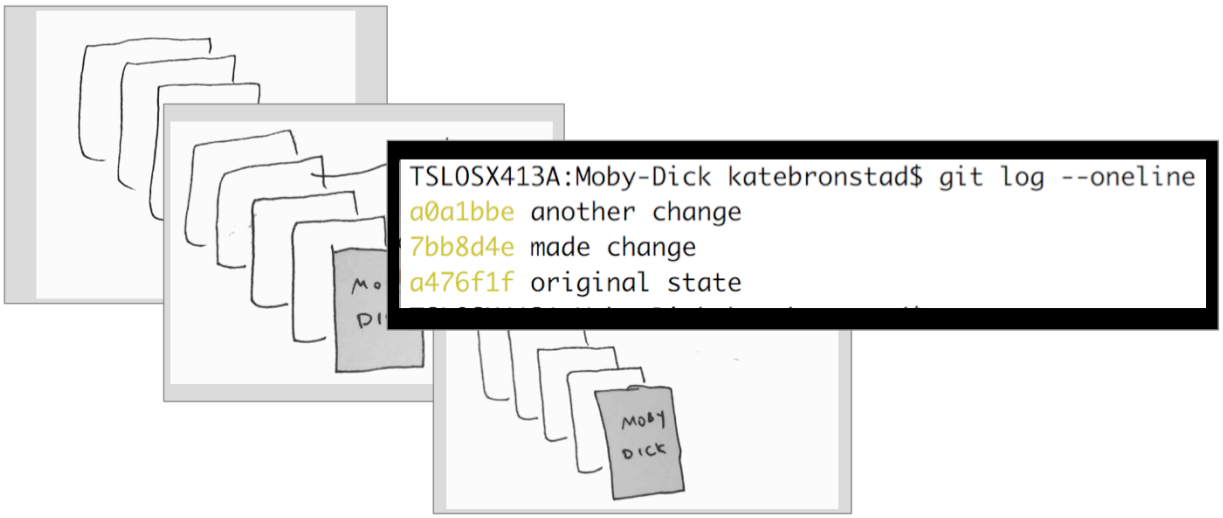
Git Concepts

A Commit

HEAD

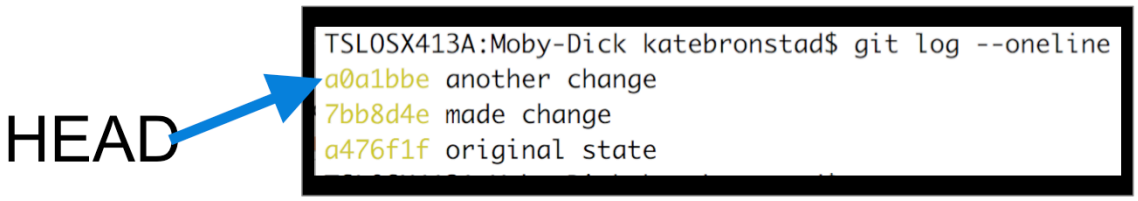
Branches

Git Concepts: Commits



Review of commits: Each commit is a version of your project. We can go back to this commit and see how all the files looked at this point in time.

Git Concepts: HEAD



HEAD is a reference/nickname for the most recent commit (instead of referring to the commit by it's unique hash, we can refer to it by just typing HEAD)

Git Concepts: HEAD

Git reset HEAD~1 --hard or Git reset 7bb8d4e --hard

```
TSLOSX413A:Moby-Dick katebronstad$ git log --oneline
a0a1bbe another change
7bb8d4e made change
a476f1f original state
```

We can move HEAD by resetting it to a previous commit. You can use the unique has to refer to this previous HASH or just say “HEAD~1”. Unless you have a really good reason, you don’t want to reset past more than 1 commit, because that complicates the project history.

Git Concepts: HEAD

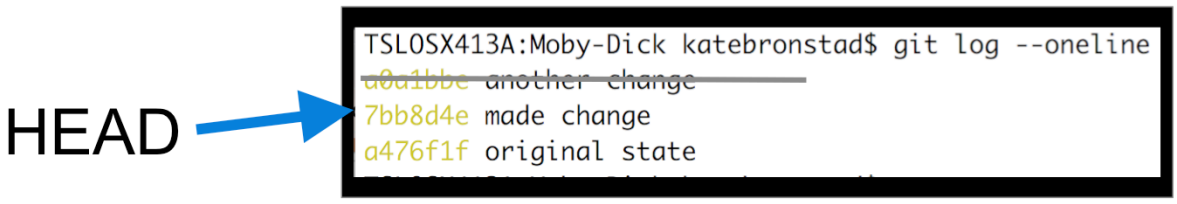


Illustration of how when we reset the HEAD, HEAD now points to what was once the 2nd most recent commit. Everything in the previous commit has been deleted (so, caution using this!)

Git Concepts: Branches

```
TSLOSX413A:Moby-Dick katebronstad$ git log --oneline
a0a1bbe another change
7bb8d4e made change
a476f1f original state
```

We start with a default branch: master.

Whenever we initialize a git project, we're automatically on a branch called master.

Git Concepts: Branches

Master branch

```
TSLOSX413A:Moby-Dick katebronstad$ git log --oneline
a0a1bbe another change
7bb8d4e made change
a476f1f original state
```

New branch: another_moby_dick

```
TSLOSX413A:Moby-Dick katebronstad$ git log --oneline
a0a1bbe another change
7bb8d4e made change
a476f1f original state
```

Git branch another_moby_dick
Git checkout another_moby_dick
OR
Git checkout -b another_moby_dick

We can create new branches. Branches are essentially a copy of the branch we are currently on. So if we were to make a branch on master, we would have a branch that's an exact copy. We can move to this branch by using the command "checkout". Once we are on the new branch, we can do new work. Master will not have this new work unless we merge the branch over.

Git Concepts: Branches

Master branch

```
TSLOSX413A:Moby-Dick katebronstad$ git log --oneline
a0a1bbe another change
7bb8d4e made change
a476f1f original state
```

New branch: another_moby_dick

```
TSLOSX413A:chapter katebronstad$ git log --oneline
2169b55 write new ending
a0a1bbe another change
7bb8d4e made change
a476f1f original state
```

The great thing about branches is that you can do very experimental work and not have to worry about messing up the work you've already done; working with git helps with this anyway, but this is even more helpful when you are collaborating with someone. You don't have to worry about your experimental work merging into the project until you are ready. It gives you a safe space to develop and test your code.

Git Concepts: Branches

Master branch

```
TSLOSX413A:Moby-Dick katebronstad$ git log --oneline
a0a1bbe another change
7bb8d4e made change
a476f1f original state
```

New branch: another_moby_dick

```
TSLOSX413A:chapter katebronstad$ git log --oneline
8d38584 destroy everything
2169b55 write new ending
a0a1bbe another change
7bb8d4e made change
a476f1f original state
```

If you are unhappy with your code, you can always delete this branch and start again.

Git Concepts: Branches

Create a branch to look back at earlier commits

Master branch

```
TSLOSX413A:Moby-Dick katebronstad$ git log --oneline
a0a1bbe another change
7bb8d4e made change
a476f1f original state
```

New branch: earlier_version

```
TSLOSX413A:chapter katebronstad$ git log --oneline
a476f1f original state
```

Git checkout -b earlier_version
Git reset a475f1f **--hard**

In the context of what we talked about earlier, about looking back at an earlier commit in your project history, branches are the safest way to look back at code in an earlier commit. To do this, you would checkout a new branch (the command git checkout –b *new_branch_name* both creates a branch and checks it out at the same time). Then you can reset the HEAD to the earlier commit and have a look at the code.

You'll get rid of the project history in between – BUT all that is still available in the master branch.

There are ways to checkout a file from an earlier commit and we can show you that later.

Gitignore

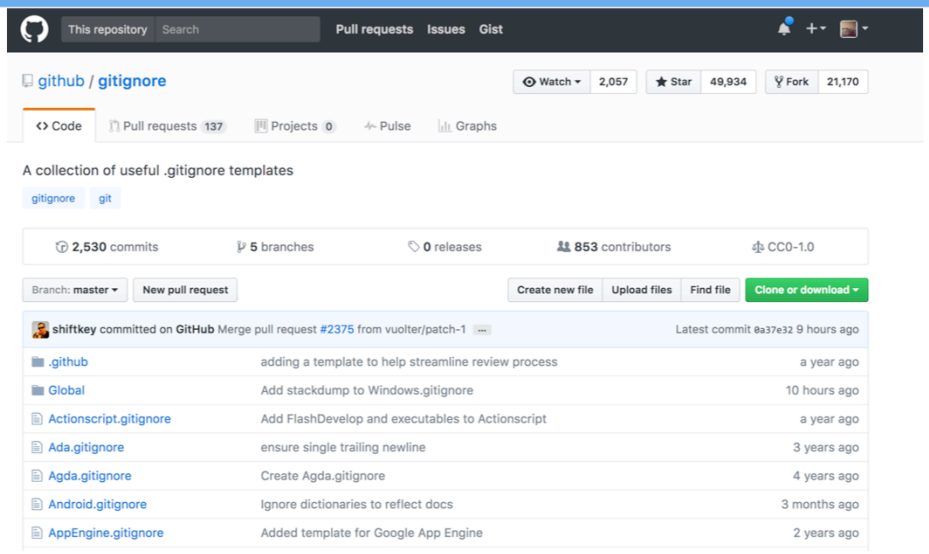
A file, not a command

Git Ignore files you don't want to track

- privacy: user information, confidential information, including config files
- Files used in only in local development or that get installed by package managers (ex. NPM, bower)
- Images

NOTE: We did not get to .gitignore in class and will talk about this in a later class.

Gitignore


This repository
Search
Pull requests
Issues
Gist
github / gitignore
Watch 2,057
Star 49,934
Fork 21,170
Code
Pull requests 137
Projects 0
Pulse
Graphs
A collection of useful .gitignore templates
gitignore
git
2,530 commits
5 branches
0 releases
853 contributors
CC0-1.0
Branch: master
New pull request
Create new file
Upload files
Find file
Clone or download
shiftkey committed on GitHub Merge pull request #2375 from vuolter/patch-1
Latest commit 0a37e32 9 hours ago
.github adding a template to help streamline review process a year ago
Global Add stackdump to Windows.gitignore 10 hours ago
Actionscript.gitignore Add FlashDevelop and executables to Actionscript a year ago
Ada.gitignore ensure single trailing newline 3 years ago
Agda.gitignore Create Agda.gitignore 4 years ago
Android.gitignore Ignore dictionaries to reflect docs 3 months ago
AppEngine.gitignore Added template for Google App Engine 2 years ago

Example Gitignore file

WordPress .gitignore

19 lines (16 sloc) | 300 Bytes

```
*.log
wp-config.php
wp-content/advanced-cache.php
wp-content/backup-db/
wp-content/backups/
wp-content/blogs.dir/
wp-content/cache/
wp-content/upgrade/
wp-content/uploads/
wp-content/wp-cache-config.php
wp-content/plugins/hello.php

/.htaccess
/license.txt
/readme.html
/sitemap.xml
/sitemap.xml.gz

```

For next week

Set up a directory using command line

Add multiple files. Make a commit with multiple files. Edit the files and add a commit.

Revert back to an earlier stage

For next week: advanced

Make a gitignore file that ignores images.
Test by putting images in your project & running git status to see if git tries to track them.

Next week: GitHub

Preview GitHub

Set up account

Connect your computer with GitHub