Version control: help you maintain a detailed history of the project as well as the ability to work on different versions of it.

There are two main types of version control system models:

* the *centralized model* - all users connect to a central, master repository
* the *distributed model* - each user has the entire repository on their computer

**Git Key Terms**

**Version Control System / Source Code Manager**

Commit: save the state of your project in Git

**repository** is a directory which contains your project work, as well as a few files (hidden by default on Mac OS X) which are used to communicate with Git. Repositories can exist either locally on your computer or as a remote copy on another computer. A repository is made up of commits.

**Working Directory** is the files that you see in your computer's file system

When project files are open up on a code editor, you're working with files in the Working Directory.

Working Directory different from current working directory (the directory that your shell is "looking at" right now)

**Checkout**: content in the repository has been copied to the Working Directory.

**staging area**: prep table where Git will take the next commit. Files on the Staging Index are poised to be added to the repository.

**SHA** = ID number for each commit.

**branch** = when a new line of development is created that diverges from the main line of development. It can continue without altering the main line.

The HTML file has HTML and CSS changes on the Staging Index and an additional HTML change in the Working Directory.

What will get committed if a commit were made right now?

When a commit is made, only the changes that are in the Staging Index are saved in the repository. (The HTML and CSS changes on the Staging Index)

* *ls - used to list files and directories*
* *mkdir - used to create a new directory*
* *cd - used to change directories*
* *rm - used to remove files and directories*
* git init  sets up all of the necessary files and directories that Git will use to keep track of everything.
  + All of these files are stored in a directory called .git
  + (notice the . at the beginning - that means it'll be a hidden directory on Mac/Linux).
  + This .git directory is the "repo"! This is where git records all of the commits and keeps track of everything!
* Git add <file> add file to staging area, and ready to keep there.
* git clone Instead of having Git create a directory with the same name as the project that's being cloned, just provide a name directly on the command line.:

$ git clone <path-to-repository-to-clone>

* git status: To figure out what's going on with a repository.

Results:

1.On branch master –Git is on the masterbranch. You've got a description of a branch on your terms sheet so this is the "master" branch (which is the default branch).

2.

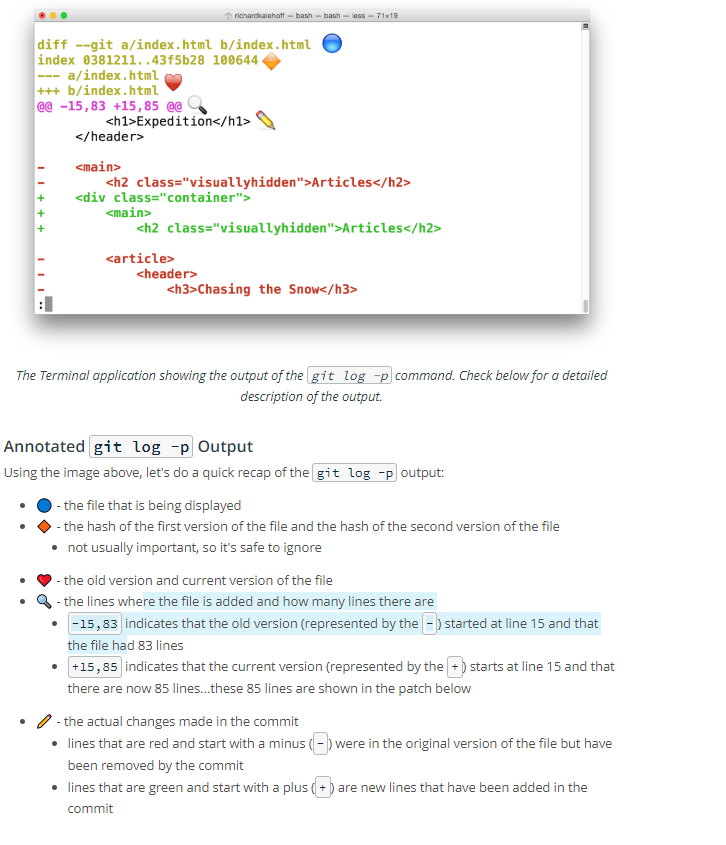
* $ git log
* to scroll **down**, press
  + j or ↓ to move *down* one line at a time
  + d to move by half the page screen
  + f to move by a whole page screen
* to scroll **up**, press
  + k or ↑ to move \_up\_ one line at a time
  + u to move by half the page screen
  + b to move by a whole page screen
* press q to **quit** out of the log (returns to the regular command prompt)

$ git log --oneline

* lists one commit per line
* shows the first 7 characters of the commit's SHA
* shows the commit's message

$ git log --stat

* displays the file(s) that have been modified
* displays the number of lines that have been added/removed
* displays a summary line with the total number of modified files and lines that have been added/removed
* $ git log –p



git show:  shows a specific commit

### The Period .

The period refers to the current directory and can be used as a shortcut to refer to all files and directories (including all nested files and directories!).

$ git add css/app.css js/app.js

*# would become*

$ git add .

git commit  command takes files from the Staging Index and saves them in the repository.

## *Bypass The Editor With The -m Flag*

git diff command can be used to see changes that have been made but haven't been committed, yet.

$ git diff

## Globbing Crash Course

* # - marks line as a comment
* \* - matches 0 or more characters
* ? - matches 1 character
* [abc] - matches a, b, \_or\_ c
* \*\* - matches nested directories - a/\*\*/z matches
  + a/z
  + a/b/z
  + a/b/c/z

So if all of the 50 images are JPEG images in the "samples" folder, we could add the following line to .gitignore to have Git ignore all 50 images.

.gitignore file is used to tell Git about the files that Git should not track. This file should be placed in the same directory that the .git directory is in.

Process to new repo:

Git init

Git add .

Git commit –m “###”

Copy 2 lines from Github (push an existing repository from command line)

Process to update repo:

Git init

Git add

Git push

* Tag = it tag command is used to add a marker on a specific commit. The tag does not move around as new commits are added.
  + add a tag to the most recent commit
  + add a tag to a specific commit *if a SHA is passed*
* The git branch command is used to interact with Git's branches:
* $ git branch

## Create A Branch

use git branch and provide it the name of the branch you want it to create. branch called "sidebar":

git branch sidebar

*to list all branches*

$ git branch

## The git checkout Command

To switch between branches, we need to use Git's checkout command:

$ git checkout sidebar

## Active Branch

fastest way to determine the active branch is to look at the output of the git branch command. An asterisk will appear next to the name of the active branch:

\* footer-fix

## Delete A Branch

branch -d sidebar( name of branch)

can't delete a branch that you're currently on.

* + to delete the sidebarbranch, you'd have to switch to either the master branch or create and switch to a new branch.

To force deletion, you need to use a capital D flag - git branch -D sidebar

Combining branches together: **merging**.

$ git merge <name-of-branch-to-merge-in>

Now, since footer is directly ahead of master, this merge = one of the easiest merges to do.

Merging footer into master will cause a **Fast-forward merge**. A Fast-forward merge will just move the currently checked out branch *forward* until it points to the same commit that the other branch (in this case, footer) is pointing to.

To merge in the footer branch, run:

$ git merge footer

## Regular Merge

 Where two divergent branches are combined. You'll be surprised that to merge in a divergent branch.

To merge in the sidebar branch, make sure you're on the master branch and run:

$ git merge sidebar