git

git ‘r done

Today's Agenda

- Code Review Recap
- Git Lecture
- Git Exercise
- CSS refresher
- AJAX refresher
- HTML, CSS, AJAX assignment

git: a history

Designed by Linus Torvalds, the creator of Linux, Git is version control software that manages changes to a project, without overwriting any part of the project.

It was created mostly because the Linux community needed a version control tool after BitKeeper was no longer free.

Why use version control?

Say you're working on a web page, and so is your teammate.

With git, you and your teammate can both save copies of what you're working on to git, without worrying about overriding each other's work.

You can merge your two versions together later, either automagically or manually – depending on the location of the changes.

Okay, so?

If something goes wrong, you can just revert to a previous version, as git saves 'snapshots' of all your work.

It's like Game Saves in a super hard video game – everytime you time, you just reload your last save point. Realize you're missing a crucial item? No worries! Just go back to another save file, back further in your progress.

GitHub

- is a web site (and software program) that puts a pretty interface on top of git
- you can use the GitHub software to manage your projects, and then share and view that project on GitHub.com
- you're still using git, so it's best if you have a solid understanding of what is happening in git before you put GitHub on a resume

Terminology

Let’s do this!!!

(You probably know most of these)

repository (repo)

- a directory or storage space where your project lives
- you can keep any kind of files in a repo
- this is quite actually where the magic happens

commit

- create a snapshot of your project at this specific point in its life
- commits allow you to reevaluate and revert your project to previous states, as a commit is a saved state
- these are your “save files”

master

- the one True Copy of a repo
- the Ultimate Version of your project
- should always be as stable as possible

branch

- a full copy of a repo's master, to do with what you want
- a branch is a safe place for you to work on your project, without fear of messing it up beyond repair
- If working with multiple people on a project, it's best that each person work in their own branch, then merge their changes together on master

HEAD

- this is where your git is currently pointing – the current state of your system
- You can move your HEAD manually up and down your repo's timeline and across branches
- When you run commands, they will be applied on-top of your HEAD.

git Commands

These are the things you type into git to tell it what to do.

Cause you’re a bossy pants.

git help

lists the 21 most common git commands

it is likely more helpful than this list

git status

the rock of your relationship with git

- It shows the current status of your repo, including what branch you're on, if you have any files that are untracked by git in your repo, what is pending in your commit, etc.

git add

brings the files passed to the command to git's attention, telling git that it should start tracking those files in this repo

git commit

the most IMPORTANT command

- has git takes a snapshot of your repo
- if you don't provide a -m argument, git will create a commit template message for you to edit, describing the changes you made in this snapshot, so it is easier to leverage in the future

git branch

creates a new copy of the current repo and stores it as the name you provide; the name of your new branch should follow the command

git checkout

literally "checks out" the supplied branch in the argument, allowing you to make changes with that copy of the repo

- This command will change the files on your system to match the state of the checked out branch.

git remote

shows the remote branches associated with the current repo, and where they’re set to push or pull files to or from

- git remote -v gives you the specifics of your remote branches

git push

push the current local copy of your repo to the remote copy (this typically means onto GitHub for us)

- you can supply a remote branch name to push to a remote other than the default (master)

git fetch

get the latest version of your repo's remote with your local copy of the repo

- you can specify which remote branch to get from into your current repo

git merge

tell git to merge the passed branch with your current branch

git diff

shows the differences in a merge conflict

git pull

this command combines a git fetch with a git merge, pulling and merging the supplied branch into the current one

git merging

This is the part of version control that – let's be honest – no one particularly enjoys doing.

What happens when you merge?

Git merges the current state of the repo you're on with the repo you're merging in.

If the development history is the same for the two repos, git will simply apply the latest changes on-top of the older, and fast-forward the HEAD.

merging, continued

If the development history is different between the repos you're merging, git will find a common ancestor of the two repos, and then do a three-way merge between them and this ancestor.

This three-way will be stored in it's own special snapshot, called a merge commit, and your HEAD will be set to point to this.

Merge Conflicts

If you change the same part of the same file differently – even one character differently – git won't be able to automagically merge your repos together.

Git will not create a merge commit and will instead pause whatever it is doing and wait for you to resolve the conflict.

These conflicting files will be shown as "unmerged" in your git status until you resolve the conflict(s).

Resolving Merge Conflicts

Git will add conflict-resolution markup to your conflicting files to show you the areas it barfed on.

You'll have to manually open each of the conflicting files and remove the conflicts by picking which version you want (or a combo thereof), then saving and committing those changes yourself.

Merge Markup

<<<<<<<<<< < HEAD:file.extension

Everything in this uppersection of the markup is the stuff from your HEAD.

==========

Everything down here is from whatever you're merging in.

>>>>>>>>>> > otherBranch:otherFile.extension

Merge Exercise

There are various tools that you can use to help you resolve merge conflicts, but you should know how to do it manually, just in case.