# Handling version conflicts in git

because it's "a bit tricky"…

By Nicholas Rutherford
nick.rutherford@gmail.com
https://github.com/nruth

# Setting up git

- http://help.github.com/set-up-git-redirect
- Linux is easy
- Mac users should use homebrew… or macports if you haven't upgraded yet
- Windows users should get an Ubuntu virtual machine - I have no idea whether it works at all with Windows

# A nice reference site or 3

- http://gitref.org/
- http://progit.org
- http://help.github.com
- and `git help <command>`

# Basics

- For a good intro look at http://progit.org/book/ch1-3.html
- Each commit is a "filesystem snapshot" of the directory you are version controlling, no duplicates (shared references and gc like in dynamic memory allocation)
- You work on your PC, locally, until you run a different command to share changes with other people - tentative changes & eventual consistency

# Follow-along Example

# Start a new project

- mkdir my_work; cd my_work
- git init
- touch my_first_file
- git add my_first_file
- git commit -m “my first commit”

# Make some changes

- So far you have a project with 1 commit containing 1 empty file.
- Add some text to the file
- try the `git status` and `git diff` commands
- add your changes to the staging area with `git add <filepath>` - can be a directory, extended filename, or .
- if you get stuck try `git help add`

# Staging area

- Changes are staged before commit
- This lets you control the granuarity of your commits. This varies by project, but typically you should have 1 commit focussed on 1 program feature, bug fix, or small code change
- Don't commit "saving my work" and including the last 3 hours of changes.
- This is bad practice, and others will not like

# Selective staging

- `git status` will now show that you have a staged file
- create another empty file, but don't add it to staging
- run `git status` again
- Note that only the original file is staged

# Commit from staging

- `git commit` (will launch a text editor for the commit message)
- or `git commit -m “your commit message”`
- now try git status again
- also look at `git log` and `git whatchanged`

# Remote repository

git ˈhub /ˈgɪt
ˌhʌb/

“GitHub is the best way to collaborate with others. Fork, send pull requests and manage all your public and private git repositories.”

- https://github.com/signup/free
- students can get free upgrades: https://github.com/edu (private repositories)

# Create a new github repository

- http://help.github.com/create-a-repo/
- Use your existing project
- it's ok to add the readme file like they say, just skip the init stuff

# Push / Pull / Fetch

- Push sends updates from your local repository to the remote "reference" (repository address) you provide. Here it will be your repository hosted on github.
- It could also be many other places, e.g. sending updates to a deployed webserver on a VPS, or even your local filesystem
- Transmission protocols include ssh, https, local filesystem, and a few other things

# Push / Pull / Fetch

- Fetch downloads updates (commits you haven't seen yet) from a remote repository
- It doesn't apply them, it keeps them in another tree branch
- You use merge to combine these updates with your own branch
- Don't worry too much about branches for now. Just think of them as parallel copies of your project with different commits.

# Push / Pull / Fetch

- Pull = fetch + merge
- I don't like this command, because I don't like automatic merges
- But most tutorials will show you it, and it's probably fine for most people.
- If you fetch and merge I *think* you will get used to the ideas of remotes, remote branches, branches, etc a lot faster than if you use pull.

# Branching

http://progit.org/book/ch3-0.html

I could draw some pictures

But I was supposed to talk about conflict resolution

# Conflict Resolution

# What's a version conflict?

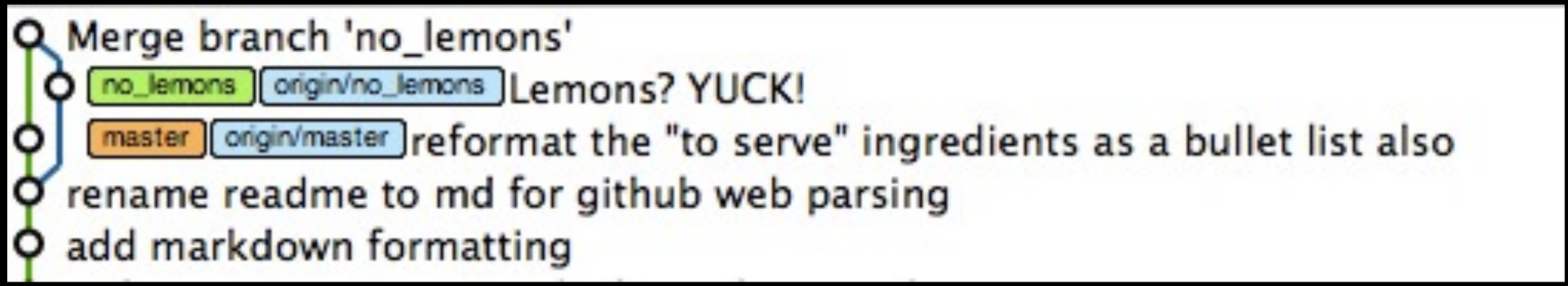
- For DS students this is easy: think optimistic replication w/ operational transfer
- when concurrent commits modify similar lines of source code and git can't work out how to resolve the problem it will give up and ask you to do it

# Demo project

- https://github.com/nruth/git-conflict-resolution-demo
- clone a local copy of this
- explore the files on github to see how markdown formatting looks when turned into HTML

# Demo project

- The README.md explains how to trigger and resolve the merge conflict




# The tree will look a bit like this after a merge commit

- Note that each original commit is preserved, with the same SHA hash - no history is lost

- and the merge commit shows the actual changes made to the filesystem snapshot as a result of merging the branches

- If you ever experiment with “rebasing” you’ll see that this is quite different