Lecture 2
Making Simple Commits

Sign in on the attendance sheet!

credit: https://xkcd.com/1296/
Review of Last Lecture

• git init – creates a git repo in the current directory

• git clone <git url> – copies the remote git repo into the current directory

• git log [ --oneline ] – lists all commits in the git repo, starting with the most recent one

• git help <command>, git <command> --help, man git <command> – brings up the man help page for the git command
Leftover topics from last lecture

• The .git folder
The .git folder

- Every git repository has a .git directory in the toplevel project directory
- This is where all git commit objects and metadata are stored
- **Don’t delete it!** Doing so deletes the repository
- Folders starting with a dot are hidden on UNIX
Today: The Git Commit Workflow

- Review: `git log`
- `git diff`
- `git status`
- `git add`
- `git commit`
- `git show`
From Last Time: git log

Also try git log --oneline:
What is 2eae45f?

• Commits are uniquely represented by SHA-1 hashes
• The first 6-7 characters of a hash are usually enough to identify it uniquely from all the other commits in the repository
• This is called the short hash
Okay, so what is a commit?

1. A **snapshot** of all the files in a project at a particular time.

2. A **checkpoint** in your project you can come back to or refer to.

Anything else?

3. The **changes** a commit makes over the previous commit
This is an example of how git diff works!

Here is a new line of text!
Git diff is my favorit command!

Git diff is my favorite command!
Here is a new line of text!
This is an example of how git diff works!

+Here is a new line of text!

-Git diff is my favorit command!

+Git diff is my favorite command!
+Here is another line of text!
Commits: Revisited

• Editing a file takes its state from 1 particular snapshot to the next
• When we edit a file, we can see it as a set of changes (a “diff”) from the snapshotted state of that file
• Commits bundle up sets of changes to a list of files
git show <commit hash>
The Git Commit Workflow: Edit

Make changes to files
vim file1.txt file3.txt

Working Directory

file1.txt (v2)
file2.txt (v1)
file3.txt (v2)

List of Changes

In file1.txt: add the line “here is a new line!” between lines 3 and 4
In file3.txt: delete line 27
The Git Commit Workflow: Add

**Working Directory**

- file1.txt (v2)
- file2.txt (v1)
- file3.txt (v2)

**List of Changes**

- In file1.txt: add the line “here is a new line!” between lines 3 and 4
- In file3.txt: delete line 27

Add the current differences

```bash
git add file1.txt file3.txt
```
The Git Commit Workflow: Commit

List of Changes

Staging Area

- In file1.txt: add the line “here is a new line!” between lines 3 and 4
- In file3.txt: delete line 27

Commit the currently staged differences

```
git commit -m "fixed bug in file1 and file3"
```
git commit

Example use:
git commit
(or)
git commit –m “commit message goes here”

• Creates a commit out of a snapshot of the staging area, and updates HEAD.
Aside: commit HEAD

- The “most recent commit” has a special name: HEAD
Good commit messages

• Good:
  Build: Don't install jsdom3 on Node.js 0.10 & 0.12 by default

• Bad:
  bugfix lol get rekt

http://whatthecommit.com
git status

Shows files differing between the staging area and the working directory (i.e. unstaged changes), the staging area and HEAD (i.e. changes ready to commit), and untracked files.
git diff

Example use:
(show unstaged changes)
git diff

(show staged changes)
git diff --cached

• Shows unstaged changes or staged changes
git show

Example use:
git show [commit hash (default is HEAD)]

• Shows the changes in the specified commit
Activity: Practicing Making Commits

• Make a new folder, and create a new git repository inside.
• Create a file called “me.txt”. Inside, write your name and hometown.
• Make a commit with this new file.
• Make a new file called “neighbors.txt”.
• Now, find 3 people sitting near you. For each person,
  • Find out their name and hometown, and put it in neighbors.txt.
  • Check the output of git status and git diff and verify it makes sense.
  • git add neighbors.txt
  • Check the output of git status and git diff and verify it makes sense.
  • Commit the change.
  • Check the output of git show and verify it makes sense.
How Git Add+Commit Actually Works

Step 1: Store a file in the git database
• Hash the file
• Store a copy of the file in .git/objects/<hash>
How Git Add+Commit Actually Works

Step 2: Store the state of a directory tree in the git database
How Git Add+Commit Actually Works

Step 3: Store a commit object with a reference to the top-level tree in the git database

tree c982effc
author Aaron Perley
committer Aaron Perley
Commit Message!

blob d670460b4
blob a83bcf5c
tree 8c3d1256