Lecture 4
More on Commits and Branches
Homework 3 Review
1. Make changes to files
   `vim file1.txt file3.txt`

2. Add changes to the staging area
   `git add file1.txt`

3. Commit changes in staging area
   `git commit -m "fixed bug in file1.txt"`

**List of commits**

- `bb2df1a`
  - file1.txt (v2)
  - file2.txt (v1)
  - file3.txt (v1)

- `782cb4f`
  - file1.txt (v1)
  - file2.txt (v1)
  - file3.txt (v1)

- `ab628cc`
  - file1.txt (v1)
  - file2.txt (v1)
What about new files?

- **Working Directory**
  - newfile.txt (v1)
  - file1.txt (v1)
  - file2.txt (v1)

- **Staging Area**
  - newfile.txt (v1)
  - file1.txt (v1)
  - file2.txt (v1)

- **List of commits**
  - bb2df1a (HEAD)
    - newfile.txt (v1)
    - file1.txt (v1)
    - file2.txt (v1)
  - 782cb4f
    - file1.txt (v1)
    - file2.txt (v1)
  - ab628cc
    - file1.txt (v1)

No difference from an edit, use `git add newfile.txt`. 
What about removing files?

- Working Directory
  - newfile.txt (v1)
  - file1.txt (v1)
  - file2.txt (v1)

- Staging Area
  - file1.txt (v1)
  - file2.txt (v1)

- List of commits
  - bb2df1a
    - file1.txt (v1)
    - file2.txt (v1)
  - 782cb4f
    - newfile.txt (v1)
    - file1.txt (v1)
    - file2.txt (v1)
  - ab628cc
    - file1.txt (v1)

```
git rm newfile.txt (also deletes newfile.txt from working directory!)
```
What if I want to undo changes in the Working Dir?

Working Directory

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

Staging Area

List of commits

bb2df1a
(HEAD)
coolfile.txt (v1)
file1.txt (v1)
file2.txt (v1)

782cb4f
newfile.txt (v1)
file1.txt (v1)
file2.txt (v1)

ab628cc
file1.txt (v1)

git checkout -- coolfile.txt (Note staging area is unaffected)
What if I want to ‘unstage’ a file?

git reset HEAD coolfile.txt (Note WD is unaffected)
What if I want to start over and go back to exactly what the HEAD looks like (in both WD and SA)?

Working Directory
- `git reset --hard HEAD`
- `coolfile.txt (v2)`
- `coolfile.txt (v1)`
- `file1.txt (v2)`
- `file1.txt (v1)`
- `file2.txt (v1)`

Staging Area
- `coolfile.txt (v2)`
- `coolfile.txt (v1)`
- `file1.txt (v2)`
- `file1.txt (v1)`
- `file2.txt (v1)`

List of commits
- `bb2df1e`
- `coolfile.txt (v1)`
- `file1.txt (v1)`
- `file2.txt (v1)`

- `782cb4f`
- `newfile.txt (v1)`
- `file1.txt (v1)`
- `file2.txt (v1)`

- `ab628cc`
- `file1.txt (v1)`

`git reset --hard HEAD (overwrites entire WD!)`
Summary
Last Time

- Branches are pointers to specific commits
- Branches allow us to create commit histories that diverge
- We can merge diverged histories back together
git branch <newbranchname>

Example use:
git branch experiment

• Creates a new branch called “experiment” that points to wherever you are right now (i.e. wherever HEAD is right now)
git checkout <branchname>

Example use:
git checkout experiment

Switches the HEAD to the branch named “develop”
git reset --hard <commit_hash>

Example use:
git reset --hard HEAD~

Moves the current branch to point to a different commit
Naming Commits Relative to the HEAD

<commit-ish>~: The parent of the commit
<commit-ish>~n: The nth parent of the commit

<commit-ish> is anything that is or points to a commit:
• short hash a39dcf5
• branch name
• HEAD
Merging

git merge experiment

“Will replay the changes made on the experiment branch since it diverged from master (i.e. B) until its current commit (D) on top of master, and record the result in a new commit along with the names of the two parent commits.” (from git help merge)
Fast Forward Merges

- Occur when the branch being merged onto is an ancestor of the branch being in.
- No merge commit is made unless --no-ff flag is used.
- Will never cause conflicts!

```bash
git merge experiment
```
Three-Way Merges

- Occur when the branch being merged onto is **not a descendent** of the branch being merged in.
- The branch being merged onto has “moved on” since the split.
- Creates a merge commit, can cause conflicts!

*git merge experiment*

Performs a “three way merge” between B, F, and D
MERGE CONFLICT

`: ( andrew@hydreigon ~/temp3
03:57 PM (master)$ git merge goodidea
Auto-merging D
CONFLICT (add/add): Merge conflict in D
Automatic merge failed; fix conflicts and then commit the result.

`: ( andrew@hydreigon ~/temp3
03:57 PM (master)$ git s
On branch master
You have unmerged paths.
   (fix conflicts and run "git commit")

Changes to be committed:
   new file:   C

Unmerged paths:
   (use "git add <file>..." to mark resolution)
   both added:   D

master, HEAD

G

master, HEAD
goodidea

F

D

E

C

B

A
This file is demo.txt

<<<<<<<<< HEAD
Here is another line. modified in master
=======
Here is another line. modified in goodidea
>>>>>> goodidea
“How to fix a merge conflict”

• Run `git status` to find the files that are in conflict.

• For each of these files, look for lines like “<<<<<<< HEAD” or “>>>>>>> 3de67ca” that indicate a conflict.

• Edit the lines to match what you want them to be.

• After you finish doing this for each conflict in each file, `git add` these conflicted files and run `git commit` to complete the merge.

```bash
$ (andrew@hydreigon ~)/temp3
03:57 PM (master)$ git s
On branch master
You have unmerged paths.
  (fix conflicts and run "git commit")

Changes to be committed:

    new file:   C

Unmerged paths:
  (use "git add <file>..." to mark resolution)

    both added:   D
```
Activity!

Start the homework!