Lecture 4
Branches

Sign in on the attendance sheet!
Last Time
Scenario: You work on two features at once in a project

- b4e2c29: initial commit
- 6f96cf3: more work on feature A
- 8b7d883: begin work on feature B
- 8fc42c6: begin work on feature A
- e167179: more work on feature B
- b5f3729: even more work on feature A
- 8277e09: even more work on feature B

Branches:
- master
- HEAD
Scenario: You work on two features at once in a project

- Hard to distinguish the two different features that are being worked on based on the git history
- If the features are related, the commits might interfere with each other
Solution: Non-linear development via branches

- 8b7d883: begin work on feature B
- 8fc42c6: begin work on feature A
- 6f96cf3: more work on feature A
- 8277e09: even more work on feature B
- e167179: more work on feature B
- b5f3729: even more work on feature A

Diagram:

- Master -> b4e2c29: initial commit
- featureA
- featureB, HEAD
git branch

Example use:

git branch

- Lists all the local branches in the current repository and marks which branch you’re currently on
  - Where are “you”? Well, you’re always at HEAD. Usually, you’re also at a branch as well.

- The default branch in a repository is called “master”
git branch <newbranchname>

Example use:
git branch develop

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

Example use:
git checkout develop

- Switches to the branch named “develop”
- Instead of a branch name, you can also put a commit hash
- Very different from “git checkout <commitname> <filename>” (from last week)! That checkouts a single file, this checkouts the entire branch, including all of its files
Commits are made on whatever branch you’re on

1. `git commit -m "A"`
Commits are made on whatever branch you’re on

1. `git commit -m "A"`
2. `git commit -m "B"`
Commits are made on whatever branch you’re on

1. `git commit -m “A”`
2. `git commit -m “B”`
3. `git branch experiment`
Commits are made on whatever branch you’re on

1. `git commit -m "A"
2. `git commit -m "B"
3. `git branch experiment`
4. `git checkout experiment`
Commits are made on whatever branch you’re on

1. `git commit -m “A”`
2. `git commit -m “B”`
3. `git branch experiment`
4. `git checkout experiment`
5. `git commit -m “C”`
Commits are made on whatever branch you’re on

1. `git commit --m “A”`
2. `git commit --m “B”`
3. `git branch experiment`
4. `git checkout experiment`
5. `git commit --m “C”`
6. `git commit --m “D”`
Commits are made on whatever branch you’re on

1. `git commit -m “A”`
2. `git commit -m “B”`
3. `git branch experiment`
4. `git checkout experiment`
5. `git commit -m “C”`
6. `git commit -m “D”`
7. `git branch wildidea`
Commits are made on whatever branch you’re on

1. `git commit -m "A"`
2. `git commit -m "B"
3. `git branch experiment`
4. `git checkout experiment`
5. `git commit -m "C"
6. `git commit -m "D"
7. `git branch wildidea`
8. `git checkout wildidea`
Commits are made on whatever branch you’re on

1. `git commit –m “A”`
2. `git commit –m “B”`
3. `git branch experiment`
4. `git checkout experiment`
5. `git commit –m “C”`
6. `git commit –m “D”`
7. `git branch wildidea`
8. `git checkout wildidea`
9. `git commit –m “E”`
Commits are made on whatever branch you’re on

1. `git commit -m “A”`
2. `git commit -m “B”`
3. `git branch experiment`
4. `git checkout experiment`
5. `git commit -m “C”`
6. `git commit -m “D”`
7. `git branch wildidea`
8. `git checkout wildidea`
9. `git commit -m “E”`
10. `git checkout master`
Commits are made on whatever branch you’re on

1. `git commit –m “A”`
2. `git commit –m “B”`
3. `git branch experiment`
4. `git checkout experiment`
5. `git commit –m “C”`
6. `git commit –m “D”`
7. `git branch wildidea`
8. `git checkout wildidea`
9. `git commit –m “E”`
10. `git checkout master`
11. `git commit –m “F”`
Exercise: What [directed, acyclic] graph do the following git commands produce?

1. git commit –m “A”
2. git commit –m “B”
3. git branch stable
4. git branch experiment
5. git checkout experiment
6. git commit –m “C”
7. git checkout master
8. git commit –m “D”
9. git branch goodidea
10. git checkout experiment
11. git branch whereami
12. git commit –m “E”
13. git checkout goodidea
14. git checkout master
15. git commit –m “F”
16. git checkout whereami
17. git commit –m “G”
18. git checkout master
What branch am I on if I checkout some commit’s hash?

```
$ git checkout cc7a315
Note: checking out 'cc7a315'.
You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by performing another checkout.
If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -b with the checkout command again. Example:
  git checkout -b new_branch_name
HEAD is now at cc7a315... Add demo.txt
```
How to start a new branch from this commit?

`git branch new-feature`
`git checkout new-feature`

How to get back to experiment?
`git checkout experiment`
How do we bring branches back together?

- b5f3729: Alice: even more work on feature A
- 6f96cf3: Alice: more work on feature A
- 8fc42c6: Alice: begin work on feature A
- 8277e09: Bob: even more work on feature B
- e167179: Bob: more work on feature B
- 8b7d883: Bob: begin work on feature B
How do we bring branches back together?

```
8b7d883: Bob: begin work on feature B
```

```
8fc42c6: Alice: begin work on feature A
```

```
6f96cf3: Alice: more work on feature A
```

```
e167179: Bob: more work on feature B
```

```
8277e09: Bob: even more work on feature B
```

```
8b7d883: Bob: begin work on feature B
```

```
db82ca7: Merge branch 'featureA' into master
```

```
HEAD
```

```
master
```

```
featureA
```

```
featureB
```

```
b4e2c29: initial commit
```

```
git checkout master
git merge featureA
```
How do we bring branches back together?

Initial commit

b4e2c29: initial commit

featureA

8fc42c6: Alice: begin work on feature A

6f96cf3: Alice: more work on feature A

b5f3729: Alice: even more work on feature A

featureB

8b7d883: Bob: begin work on feature B

e167179: Bob: more work on feature B

8277e09: Bob: even more work on feature B

db82ca7: Merge branch ‘featureA’ into master

29ca3b3: Merge branch ‘featureB’ into master

master, HEAD
git merge <branch_to_merge_in>

Example use:
git merge featureA

• Makes a new merge commit on the CURRENT branch that brings in changes from featureA
How does git know how to merge changes from another branch into yours?
How does git know how to merge changes from another branch into yours?

• It doesn’t.
Most cases: Merging with possible conflicts

• Let’s say I’m on master (as denoted by HEAD) and I want to merge goodidea into master.
• git merge goodidea
Most cases: Merging with possible conflicts

• Let’s say I’m on master (as denoted by HEAD) and I want to merge goodidea into master.
• `git merge goodidea`
• At this point, if bringing in all the changes from goodidea do not conflict with the files in master, then a new commit is created (you’ll have to specify a commit message) and we’re done.
• Otherwise... `git` just goes halfway and stops.
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

D

F

E

C

B

A

goodidea
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.
Special Case: Fast-forward merges

git merge experiment

master, HEAD

experiment

bad idea

wild idea
Special Case: Fast-forward merges

Git doesn’t bother creating another commit to combine the changes because this kind of merge is guaranteed to not have conflicts.
Special Case: Fast-forward merges

Some people like creating a new commit anyway to document the fact that the merge occurred. To do so, do:

git merge --no-ff
Summary

• git branch – list all branches
• git branch <branchname> - make a new branch
• git checkout <branchname> - switch to another branch or commit
• git merge <branchname> - make a commit merging *in* a branch
Activity!

In pairs:
1. Create a git repository
2. Make a new file called file1.txt, add some lines to it, and commit it
3. Create two branches called branch1 and branch2
4. Edit the same line in the text file and make a commit in each branch
5. Merge both branches back to master (merging the second branch back will require resolving the conflicts).
6. What do we call the merge that occurred when merging the first branch back to master?