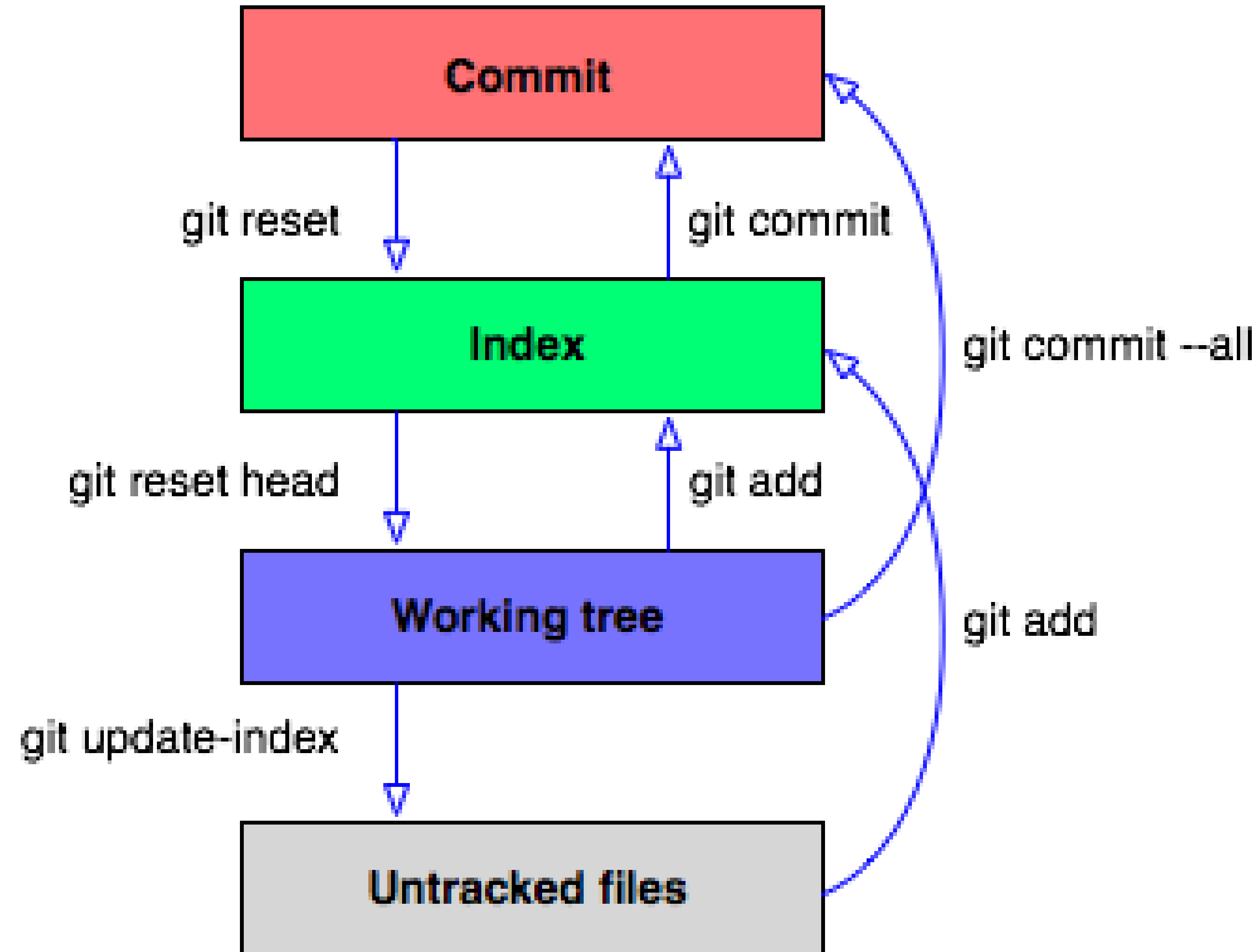
Lecture 4
Branches

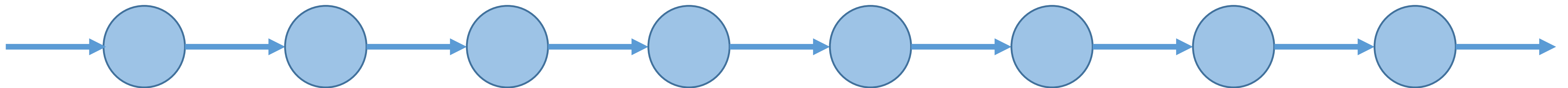
Sign in on the attendance sheet!

Last Time

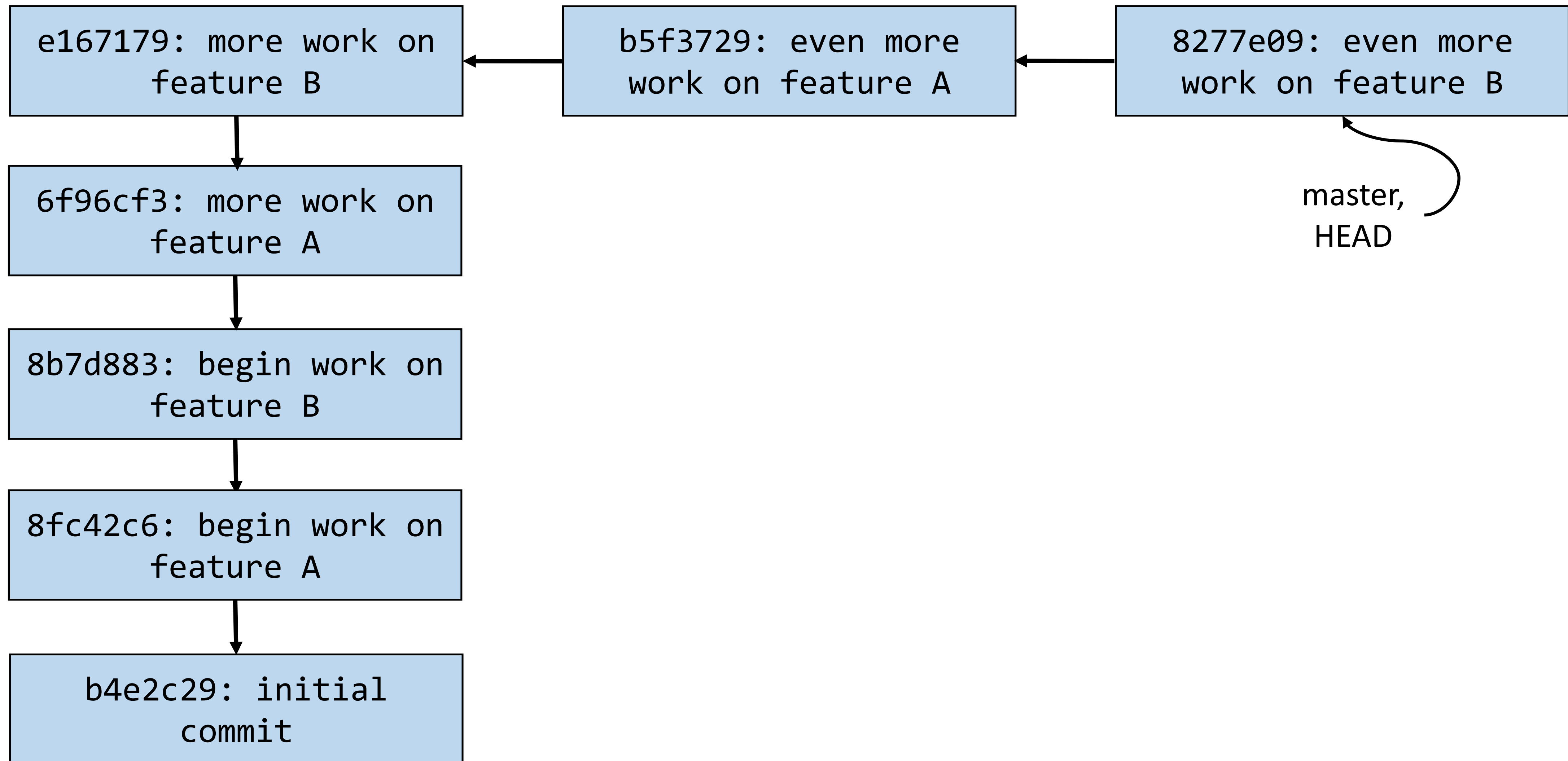


Last Time

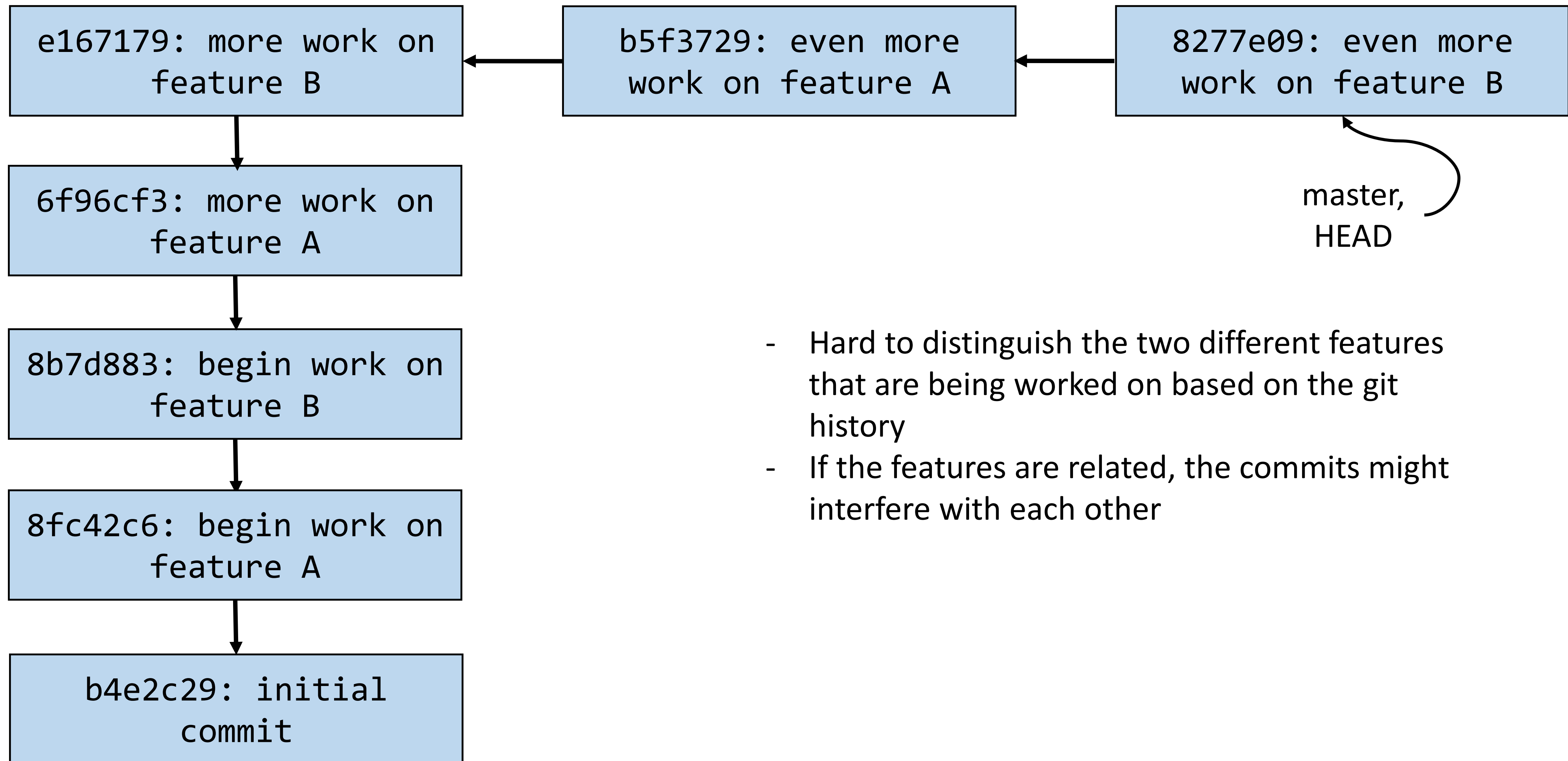
```
.gitignore

git config --global user.name "Subra Suresh"
git config --global user.email subra@cmu.edu
```



Scenario: You work on two features at once in a project

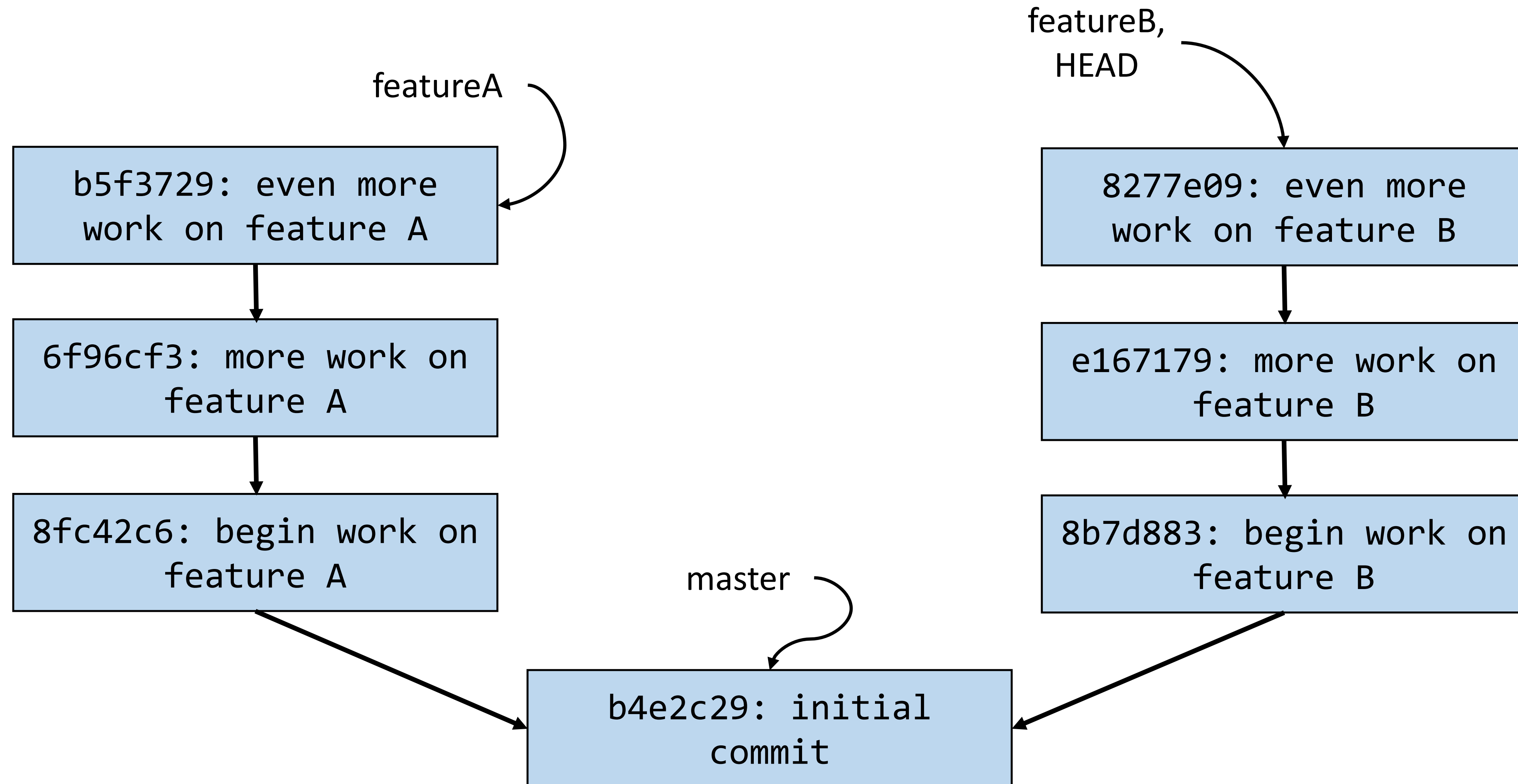


Scenario: You work on two features at once in a project



- 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



git branch

Example use:

```
git branch
```

```
Aaron@HELIOS ~/Dropbox
$ git branch
  feature-2
* master
  new-feature
```

- 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>`

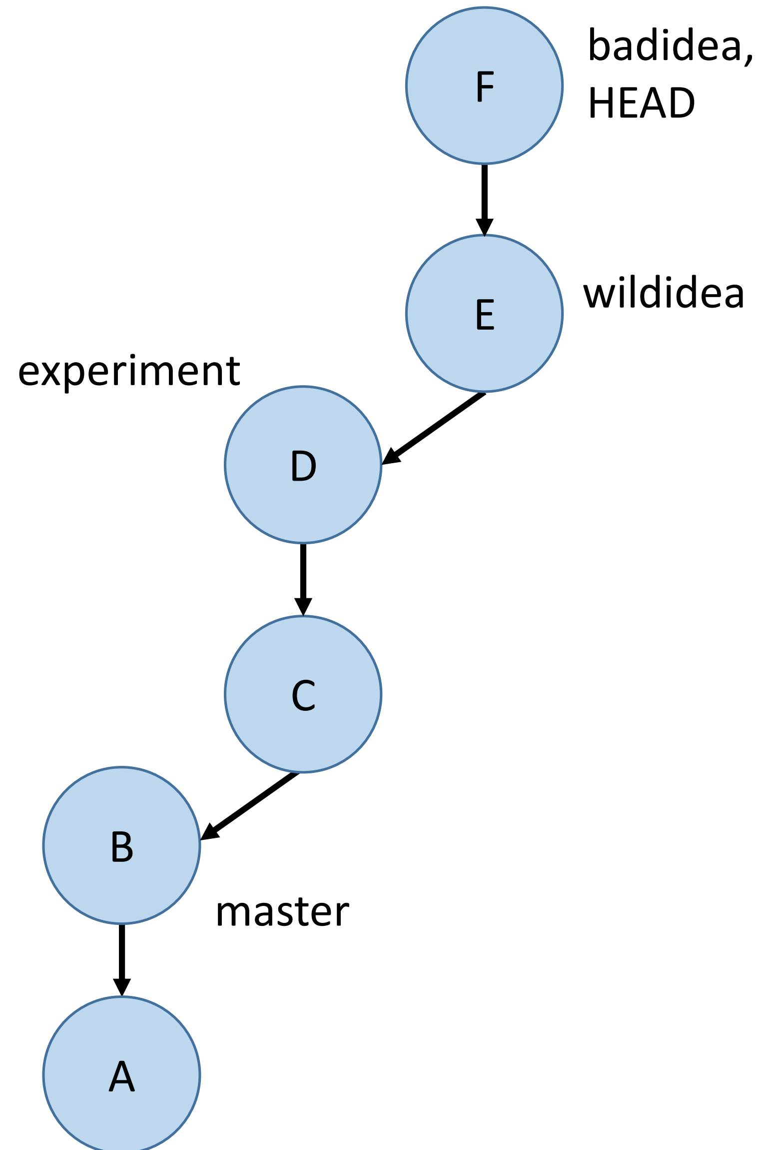
Example use:

```
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

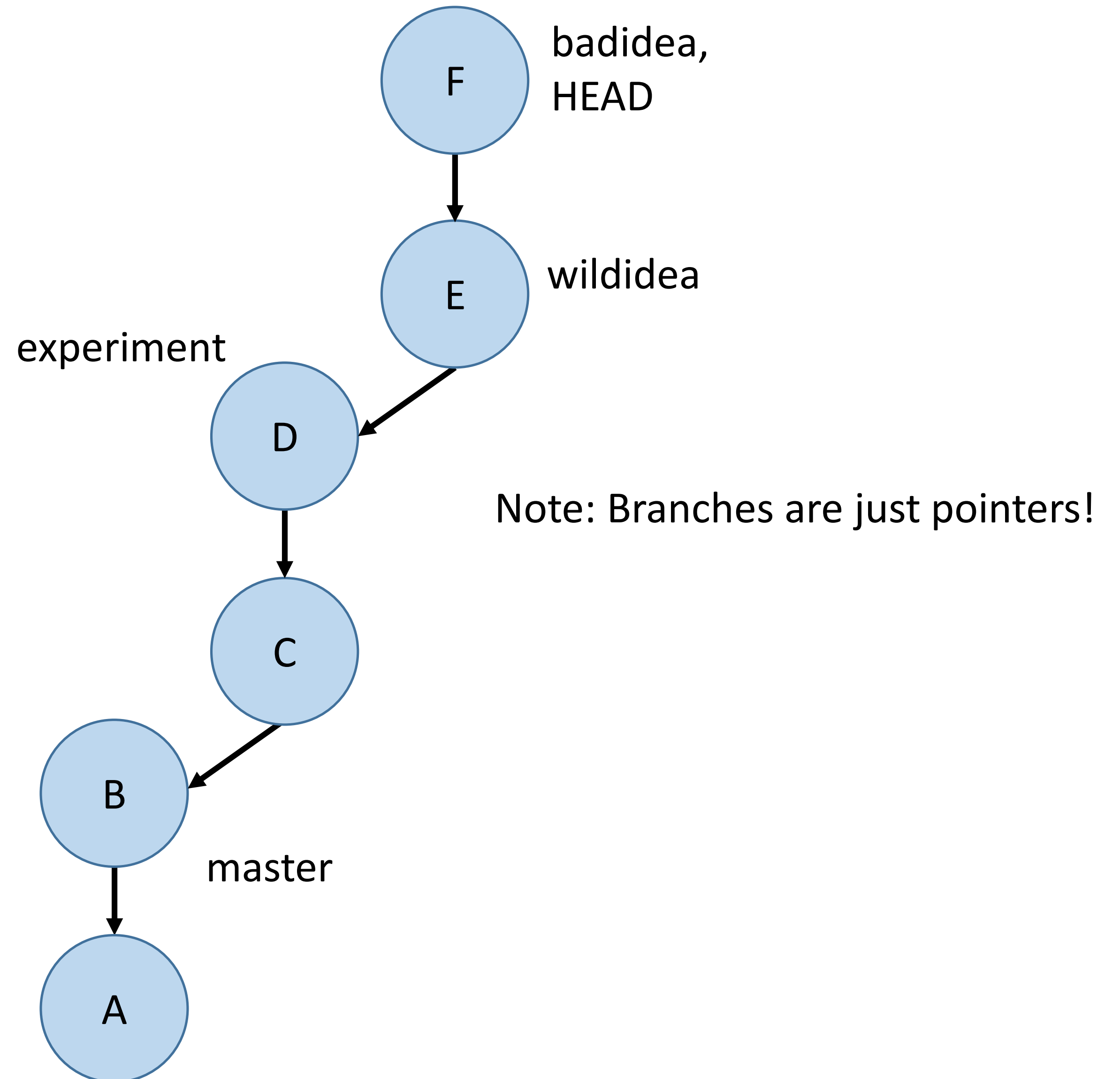
Commits are made on whatever branch you’re on

- git commit –m “A”
- git commit –m “B”
- git branch experiment
- git checkout experiment
- git commit –m “C”
- git commit –m “D”
- git branch wildidea
- git checkout wildidea
- git commit –m “E”
- git branch badidea
- git checkout badidea
- git commit –m “F”



Commits are made on whatever branch you're on

- git commit –m "A"
- git commit –m "B"
- git branch experiment
- git checkout experiment
- git commit –m "C"
- git commit –m "D"
- git branch wildidea
- git checkout wildidea
- git commit –m "E"
- git branch badidea
- git checkout badidea
- git commit –m "F"

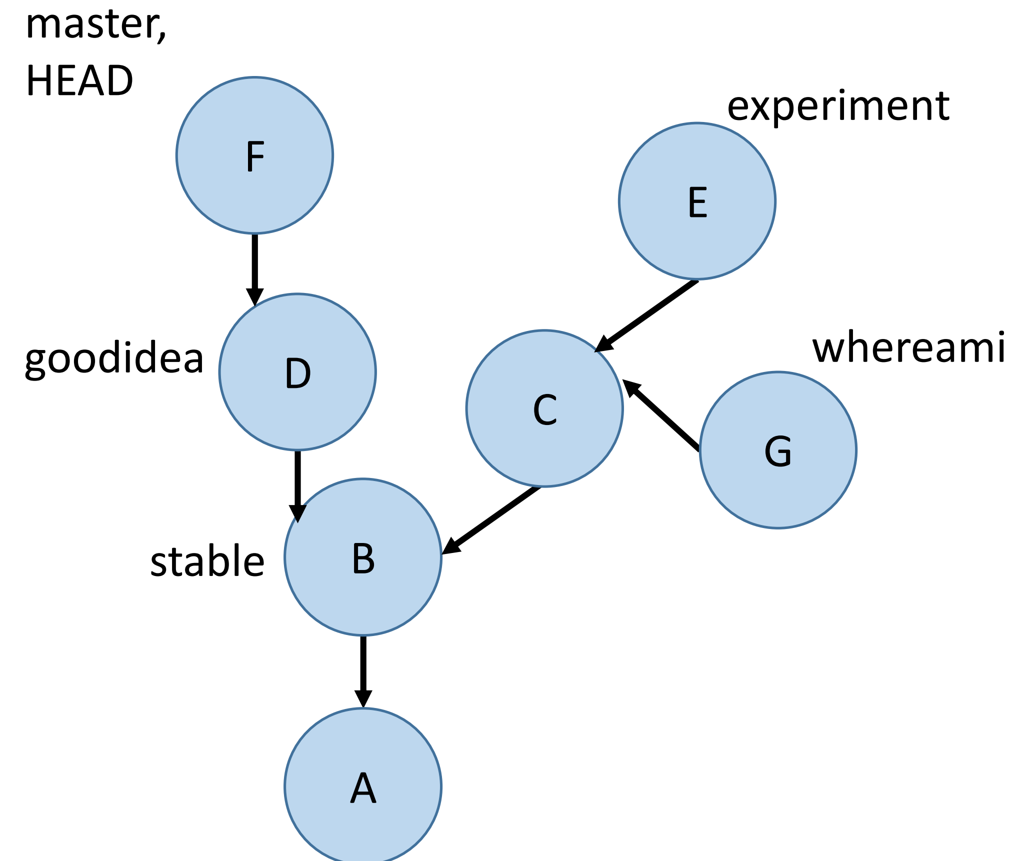


Exercise: What [directed, acyclic] graph do the following git commands produce?

- git commit –m “A”
- git commit –m “B”
- git branch stable
- git branch experiment
- git checkout experiment
- git commit –m “C”
- git checkout master
- git commit –m “D”
- git branch goodidea
- git checkout experiment
- git branch whereami
- git commit –m “E”
- git checkout goodidea
- git checkout master
- git commit –m “F”
- git checkout whereami
- git commit –m “G”
- git checkout master

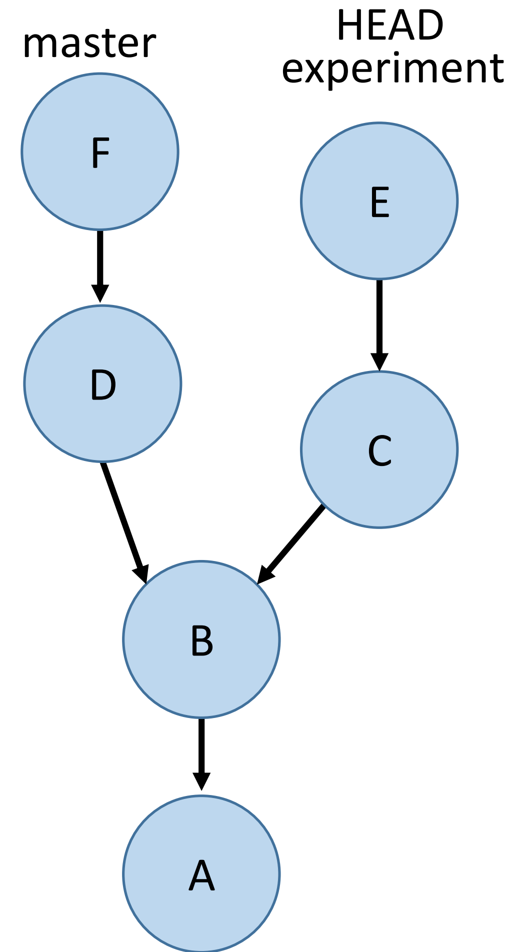
Exercise: What [directed, acyclic] graph do the following git commands produce?

- git commit –m “A”
- git commit –m “B”
- git branch stable
- git branch experiment
- git checkout experiment
- git commit –m “C”
- git checkout master
- git commit –m “D”
- git branch goodidea
- git checkout experiment
- git branch whereami
- git commit –m “E”
- git checkout goodidea
- git checkout master
- git commit –m “F”
- git checkout whereami
- git commit –m “G”
- git checkout master



What branch am I on if I checkout some commit's hash?

```
Aaron@HELIOS ~/Dropbox/Dropbox Documents/98174/testing (master)
$ 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
```



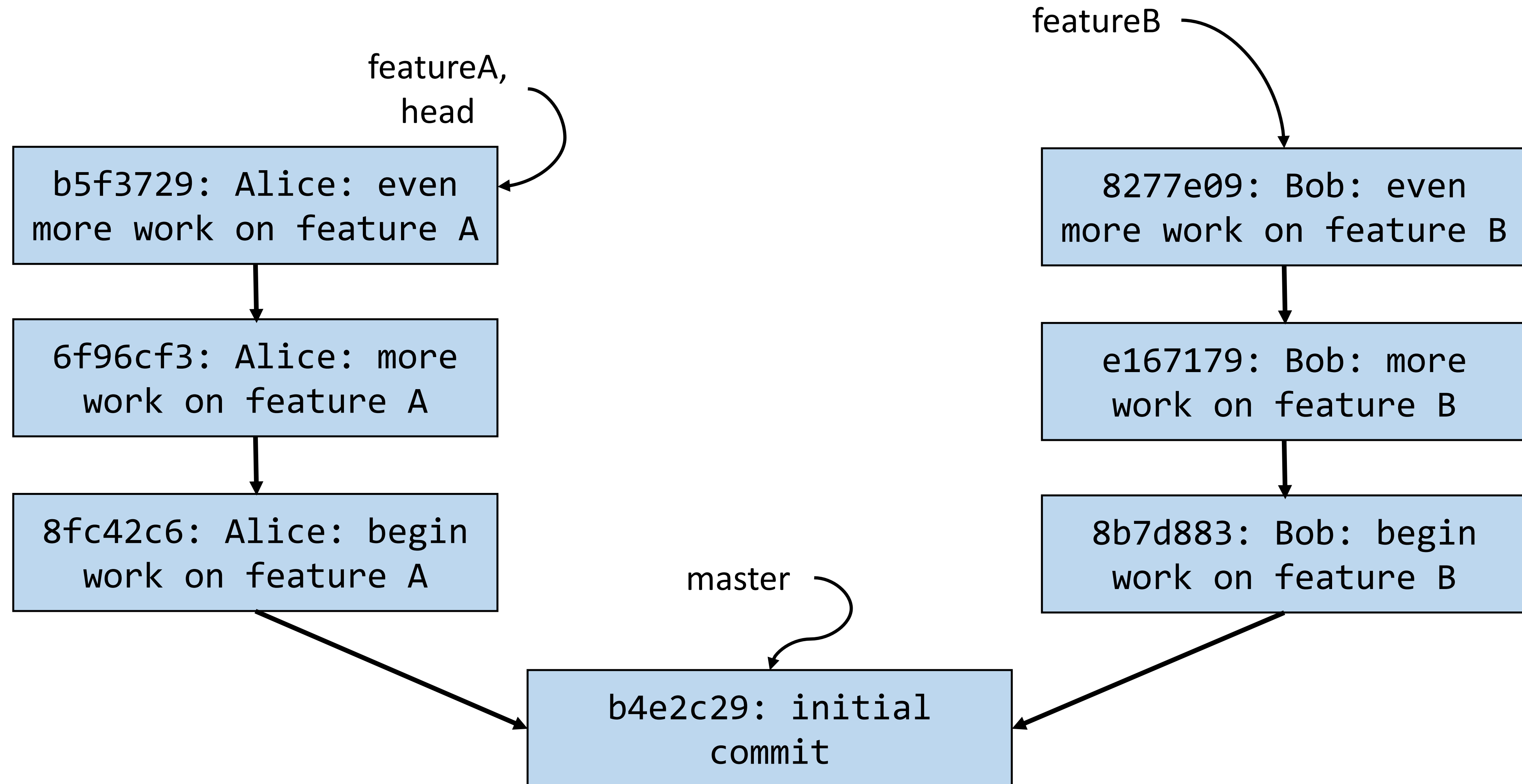
How to start a new branch from this commit?

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

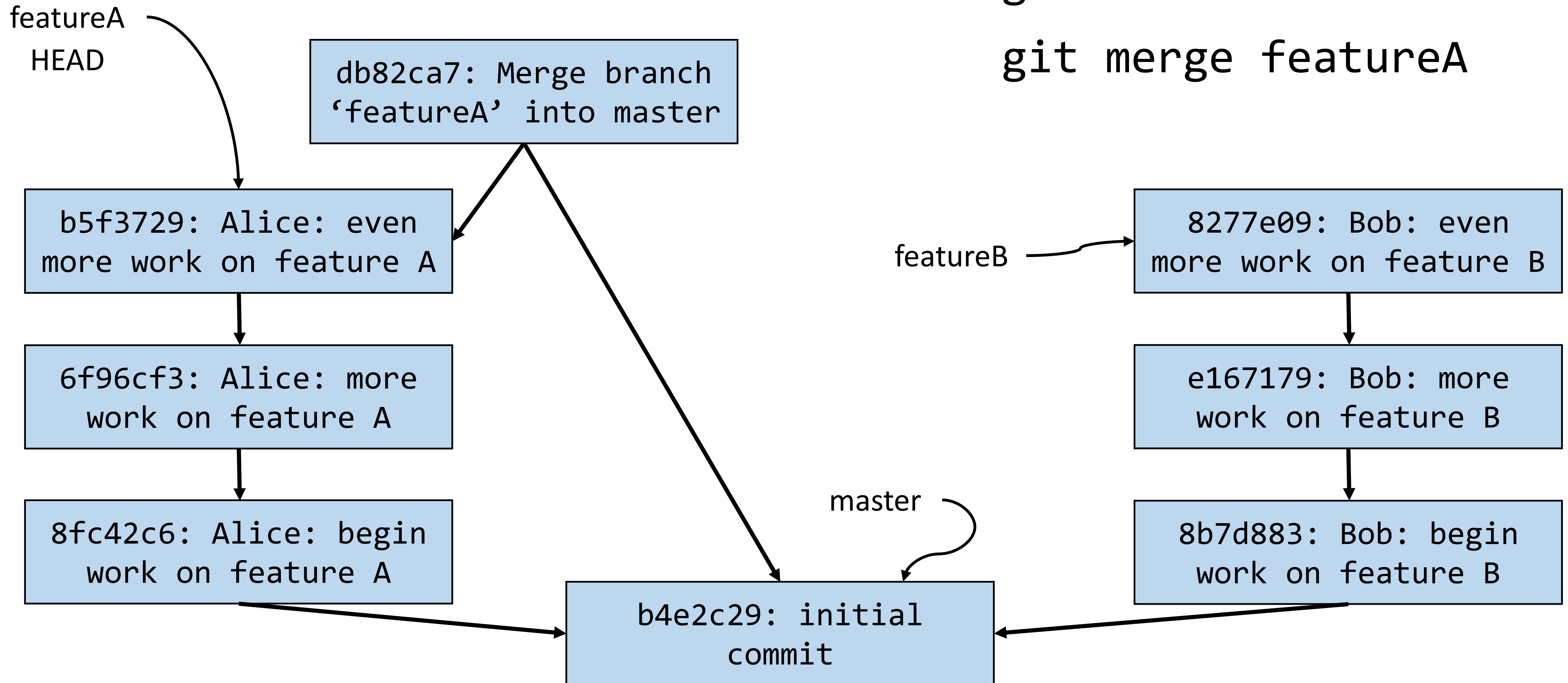
How to get back to experiment?

```
git checkout experiment
```

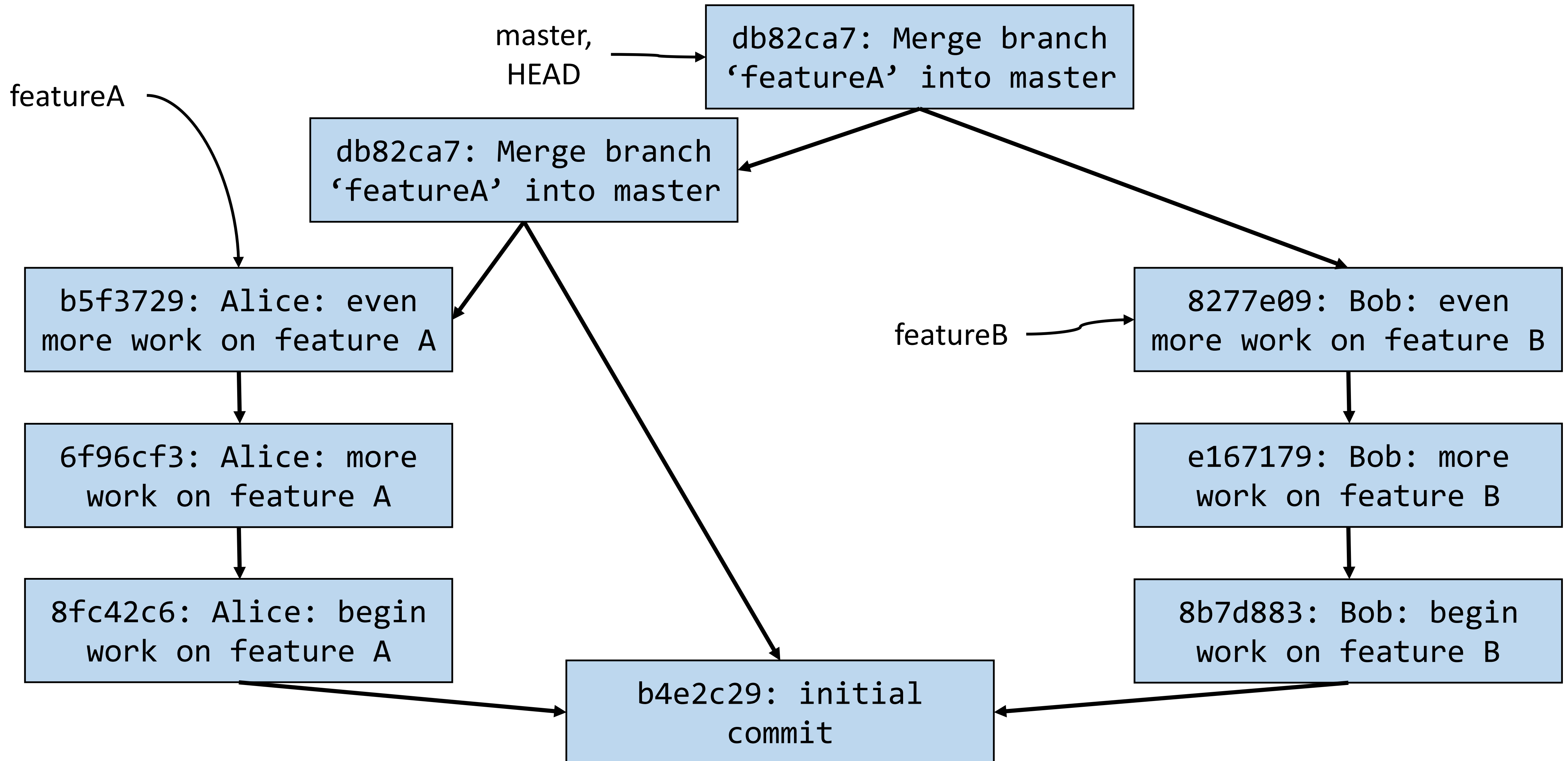
How do we bring branches back together?



How do we bring branches back together?



How do we bring branches back together?



git merge <branch_to_merge_in>

Example use:

```
git merge featureA
```

- Makes a new commit on the CURRENT branch that brings in changes from featureA

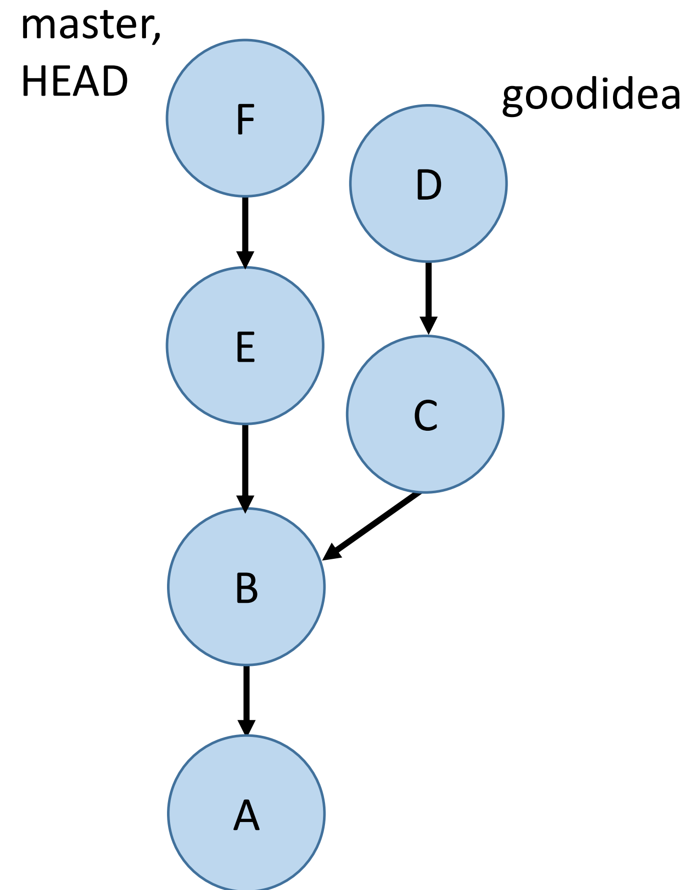
How does git know how to merge changes from another branch into yours?

How does git know how to merge changes from another branch into yours?

- It doesn’t.

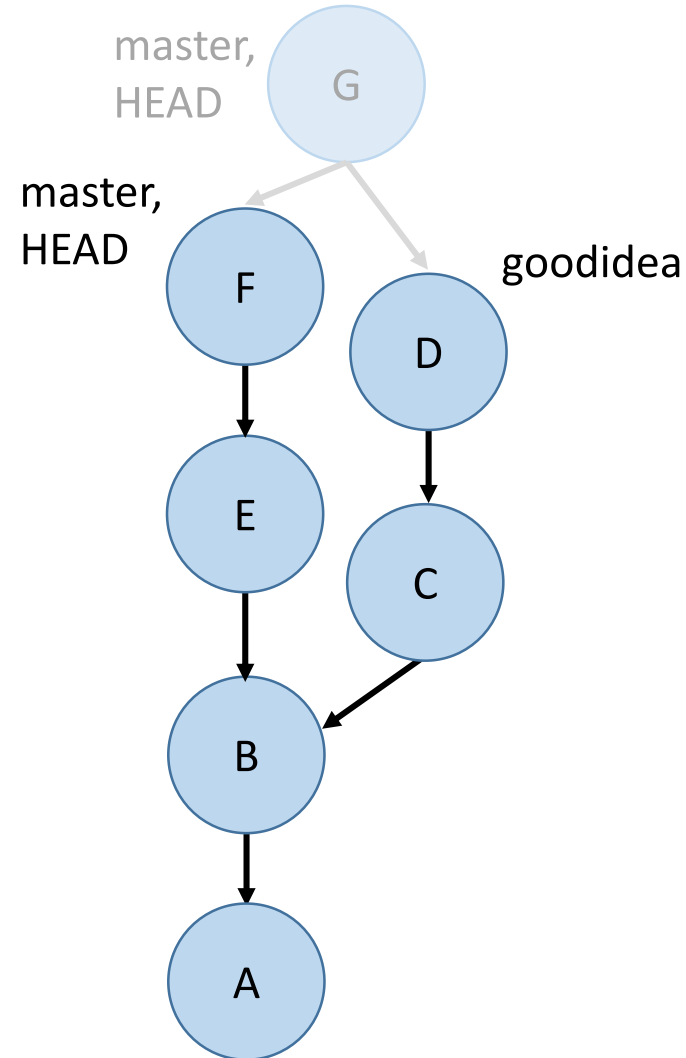
Most cases: Merging with possible conflicts

- 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

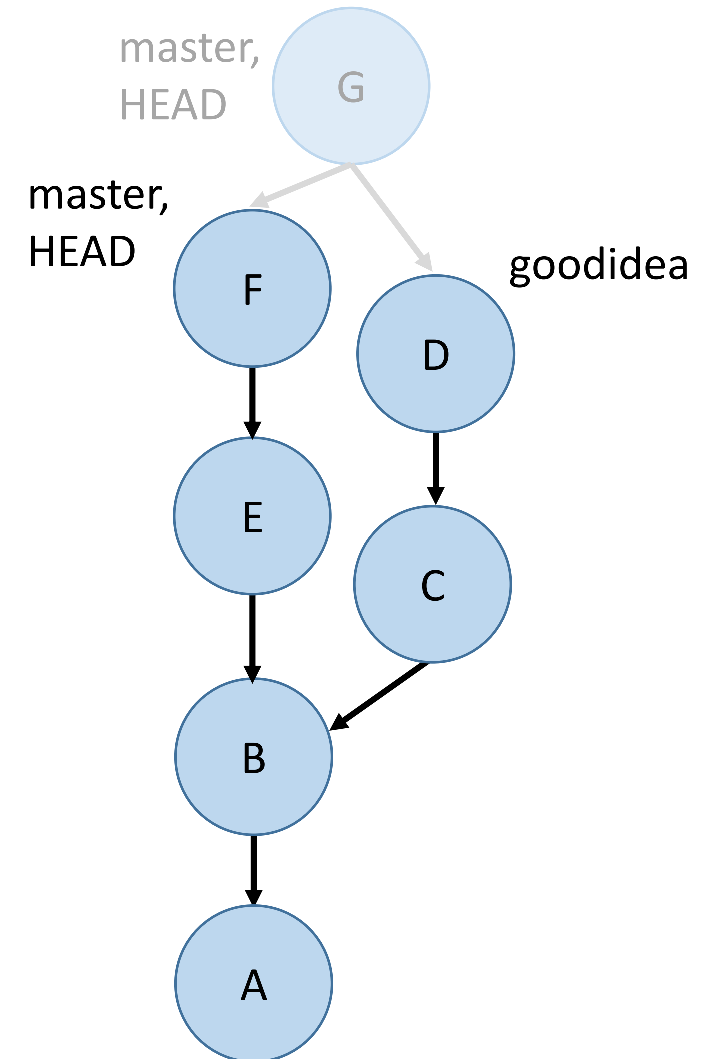
- Let's say I'm on master (as denoted by HEAD) and I want to merge goodidea into master.
- git merge goodidea
- At this point, if bringing in all the changes from goodidea do not conflict with the files in master, then a new commit is created (you'll have to specify a commit message) and we're done.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



MERGE CONFLICT

```
This file is demo.txt

<<<<<<< HEAD
Here is another line. modified in master
=======
Here is another line. modified in goodidea
>>>>>>> goodidea
```

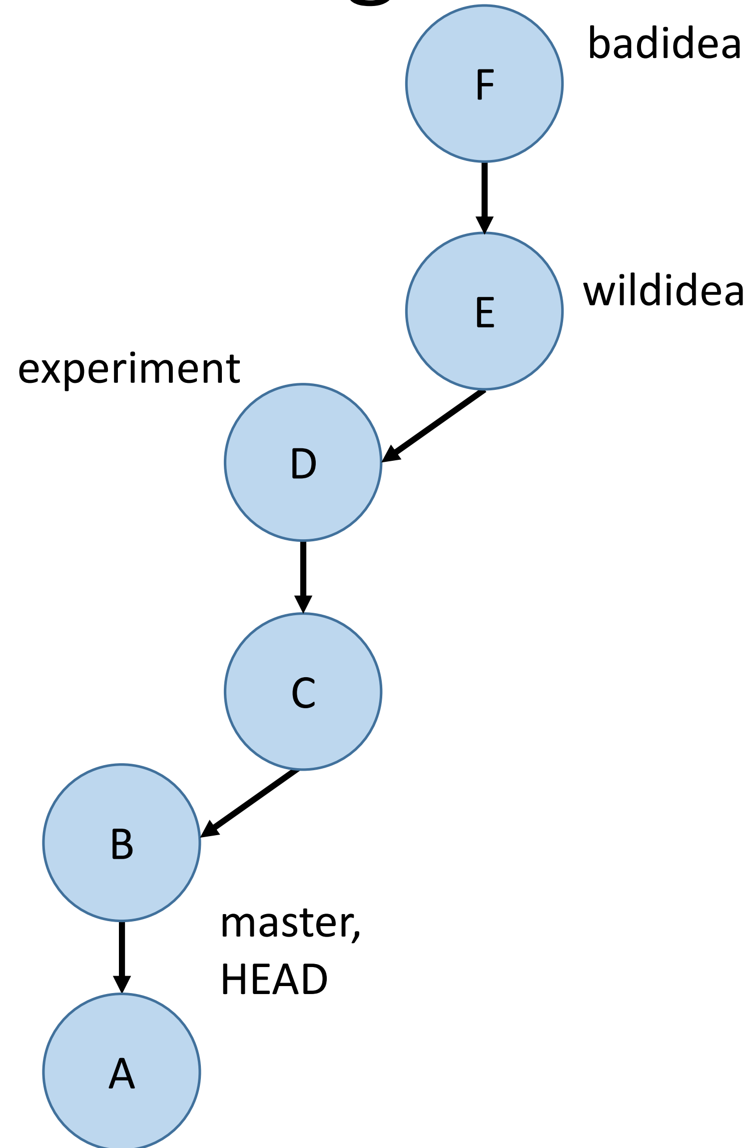
“How to fix a merge conflict”

- 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.

```
:( 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
```

Special Case: Fast-forward merges

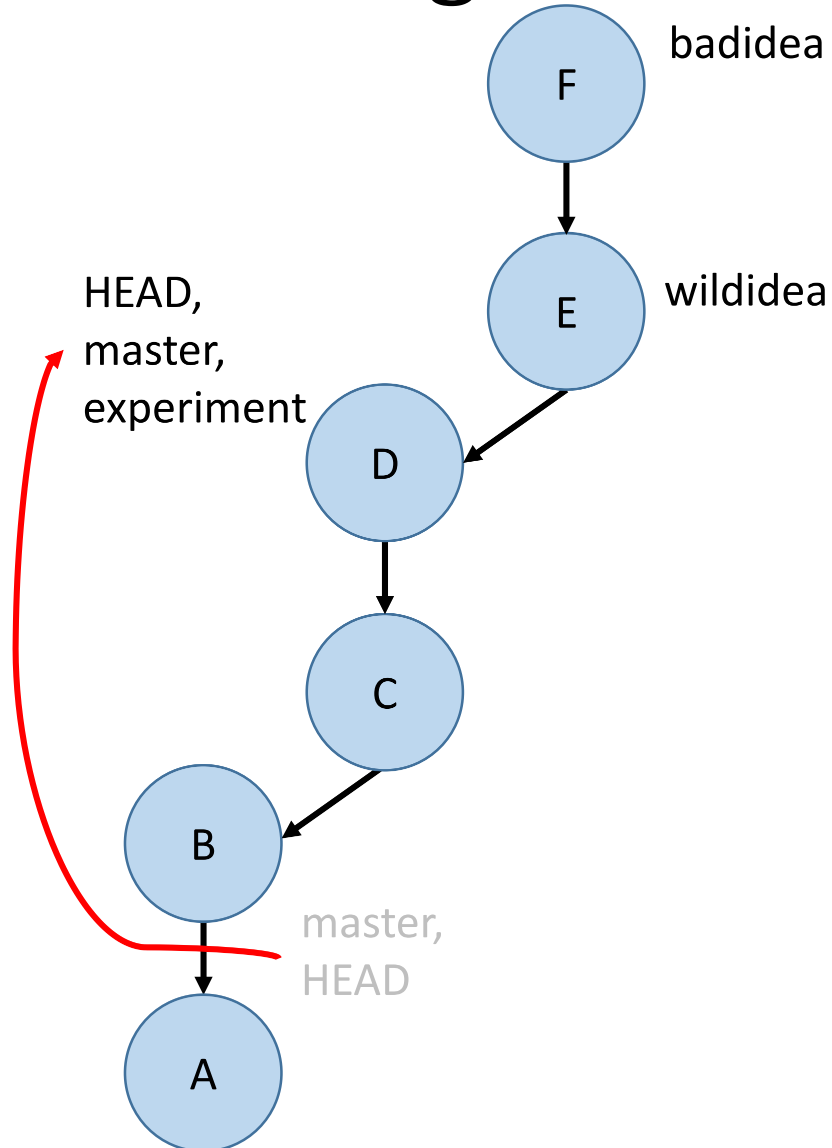
git merge experiment



Special Case: Fast-forward merges

git merge experiment

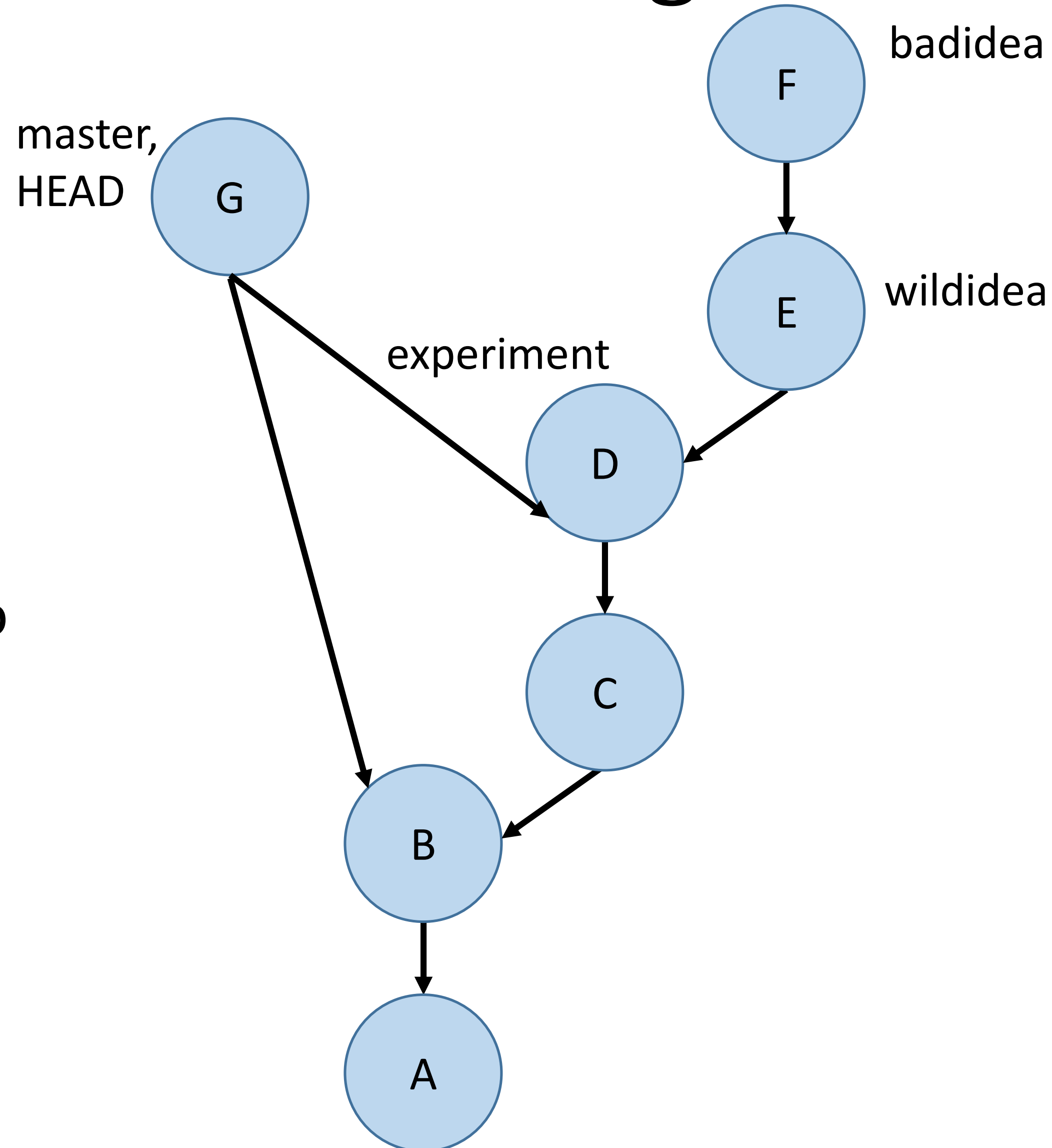
Git doesn't bother creating another commit to combine the changes because this kind of merge is guaranteed to not have conflicts.



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?