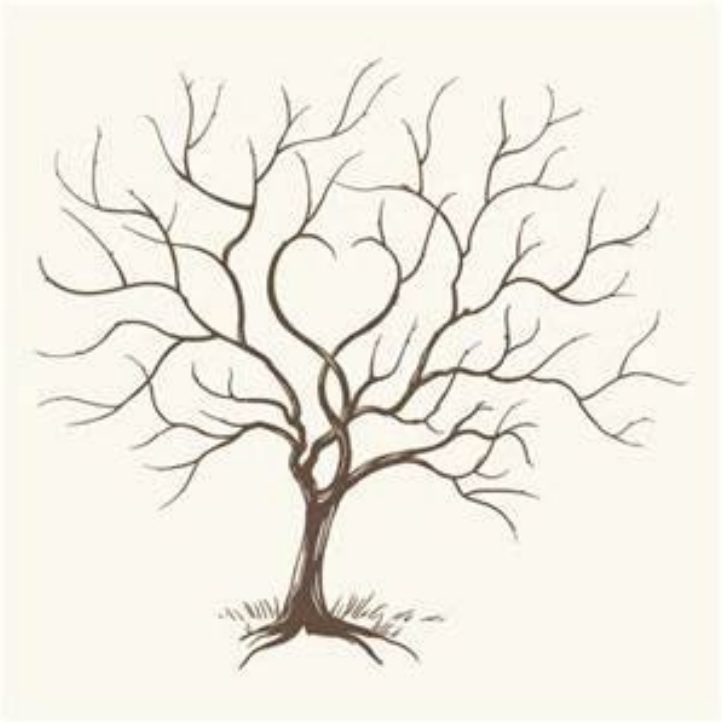
Git Branches

prepared by Roberto A. Bittencourt

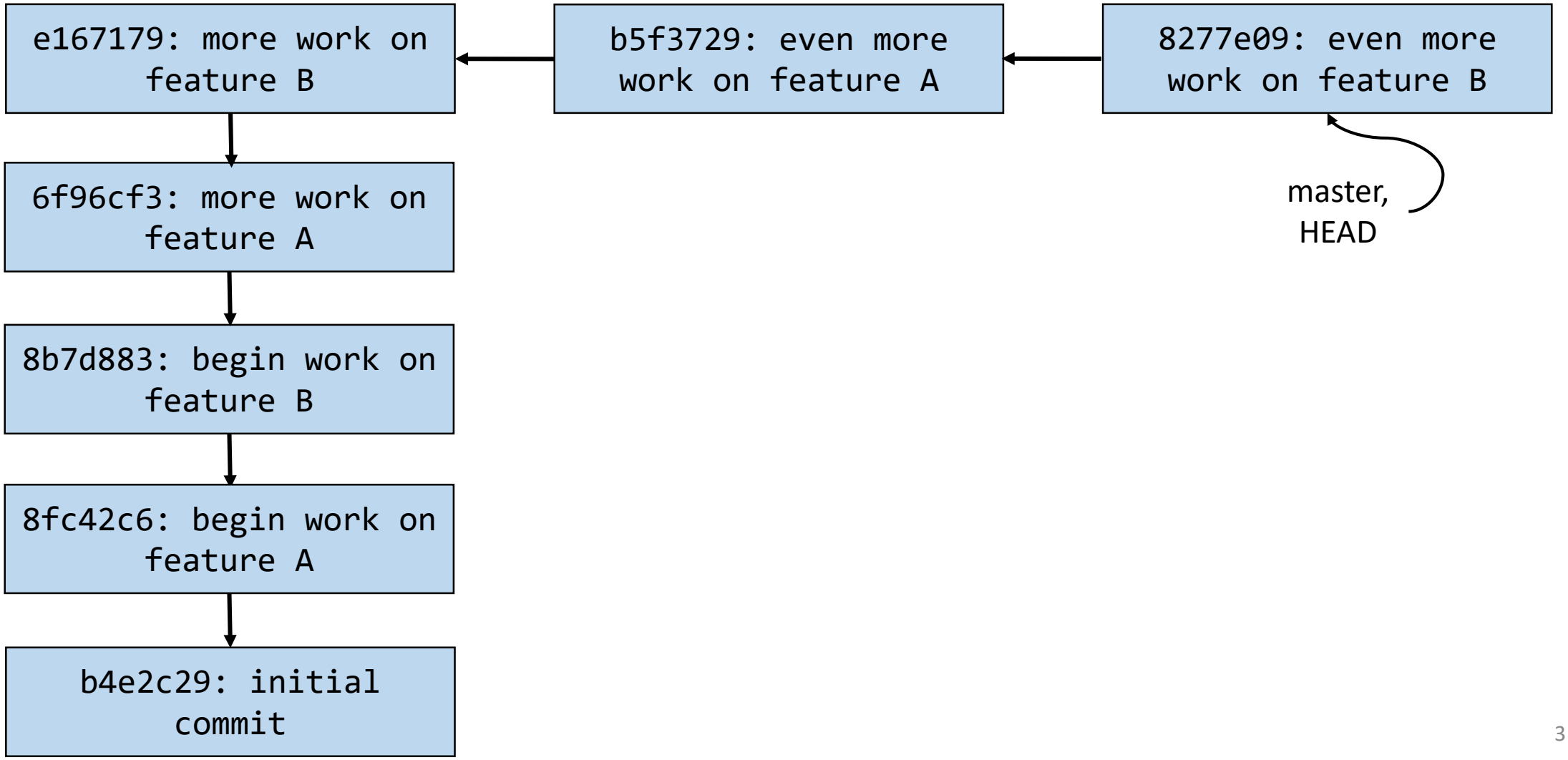
Based on slides by Aaron Perley and Ilan Biala

Branches

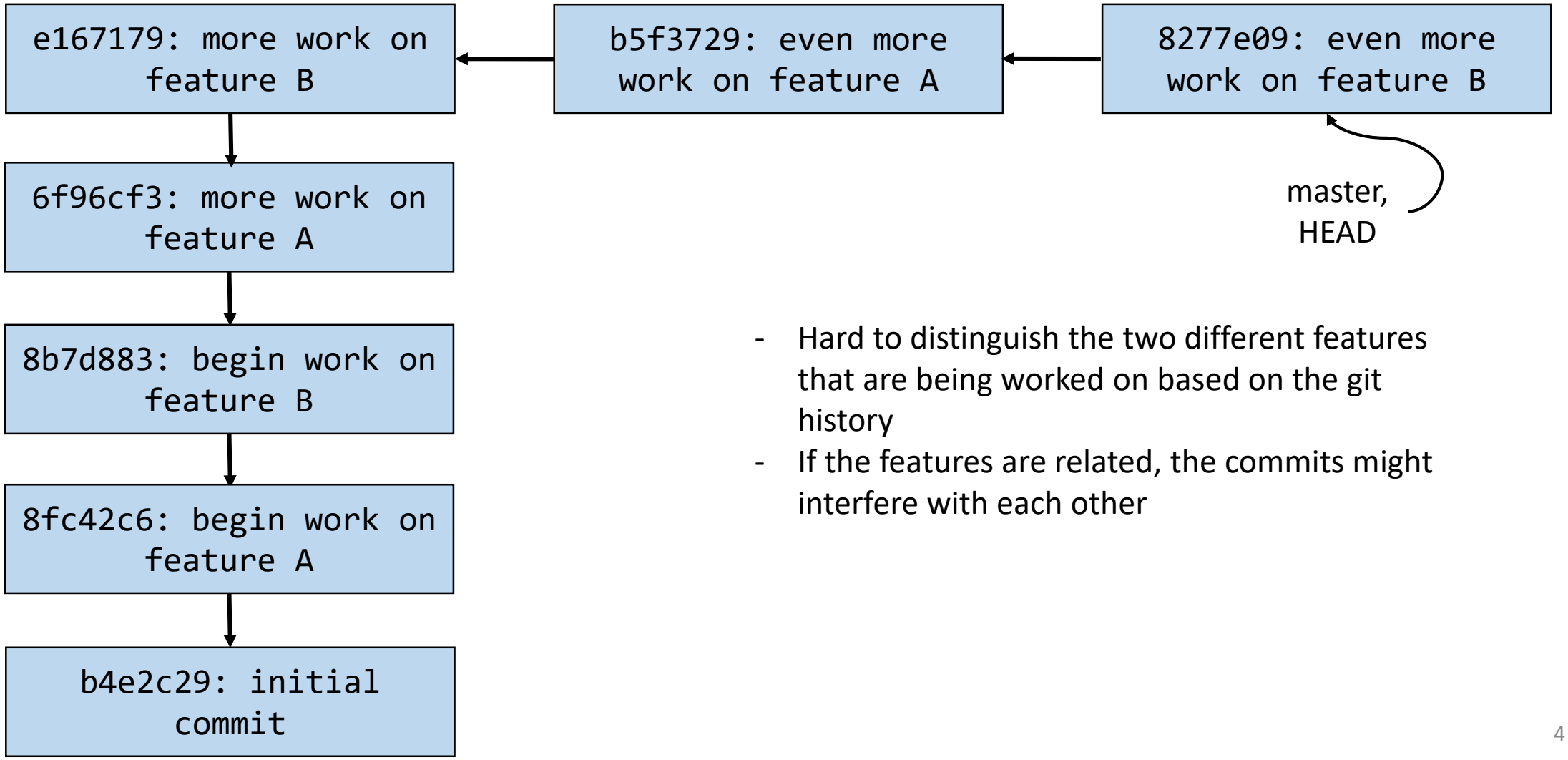


This part of the slides is fully based on the course materials of Aaron Perley and Ilan Biala (CMU)
(Reproduced for educational purposes)

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

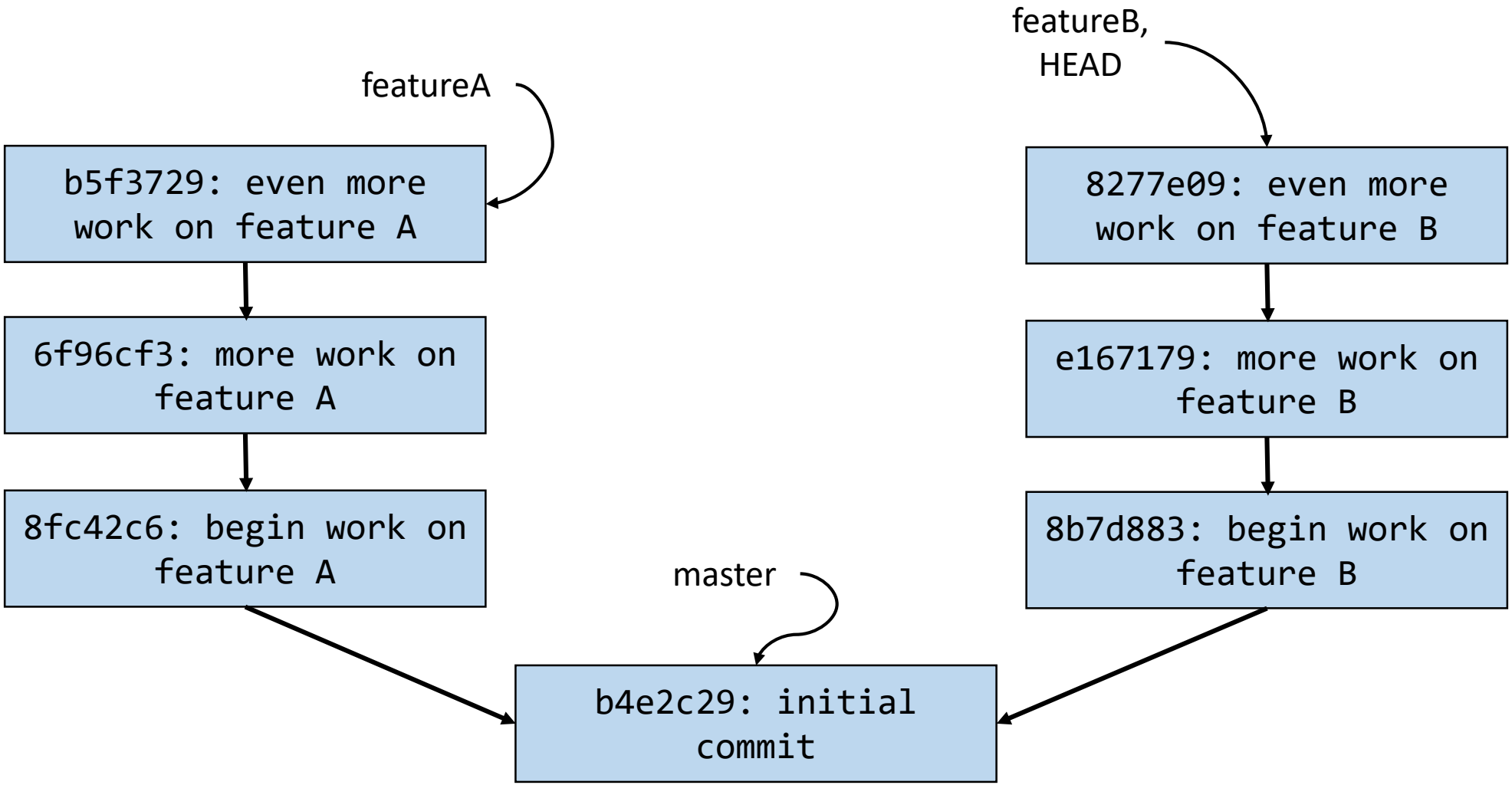


Scenario: You work on two features at once in a project



- 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

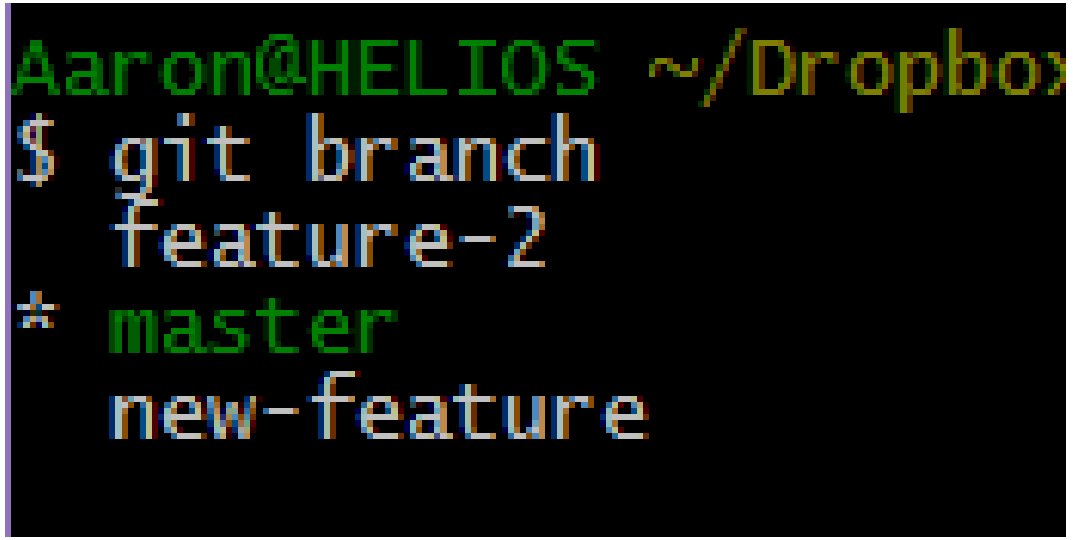
Solution: Non-linear development via branches



git branch

Example use:

`git branch`



- 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
```

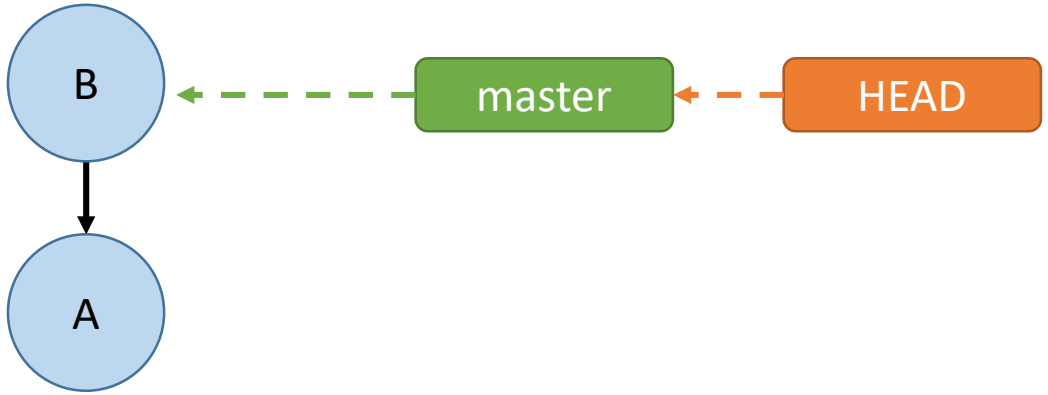
- Switches to the branch named “develop”

Commits are made on whatever branch you're on

1. git commit –m "A"
2. git commit –m "B"

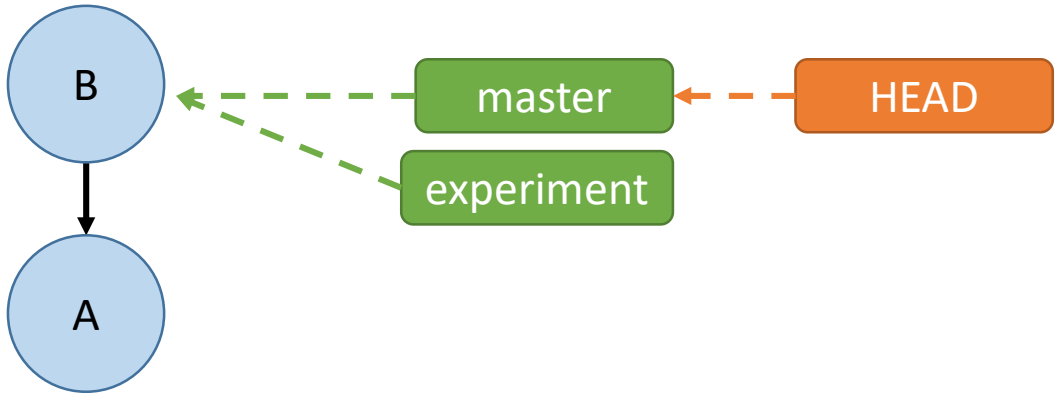
Commits are made on whatever branch you're on

1. git commit –m "A"
2. git commit –m "B"
3. git branch experiment



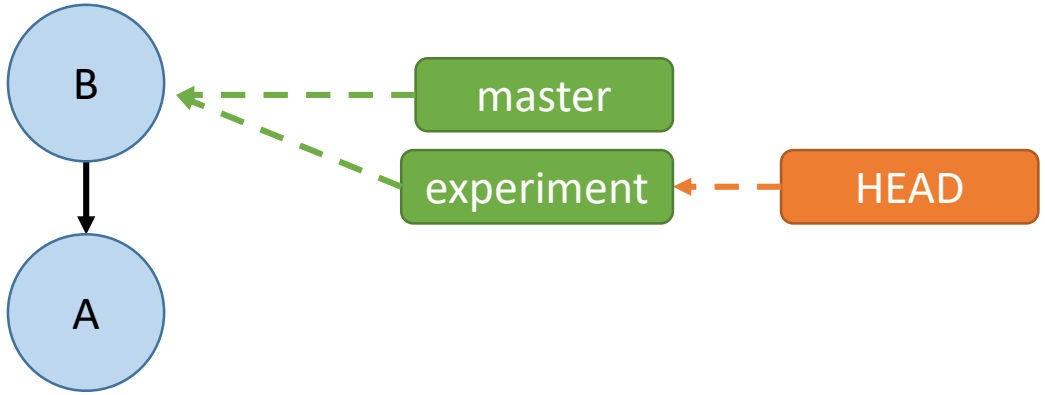
Commits are made on whatever branch you're on

1. git commit –m "A"
2. git commit –m "B"
3. git branch experiment
4. git checkout experiment



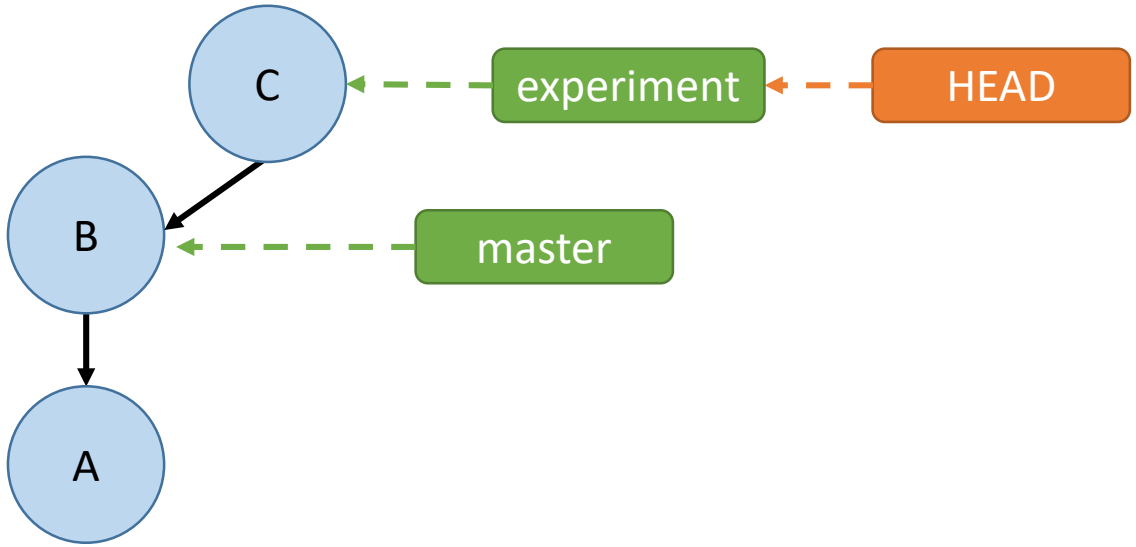
Commits are made on whatever branch you're on

1. git commit –m "A"
2. git commit –m "B"
3. git branch experiment
4. git checkout experiment
5. git commit –m "C"



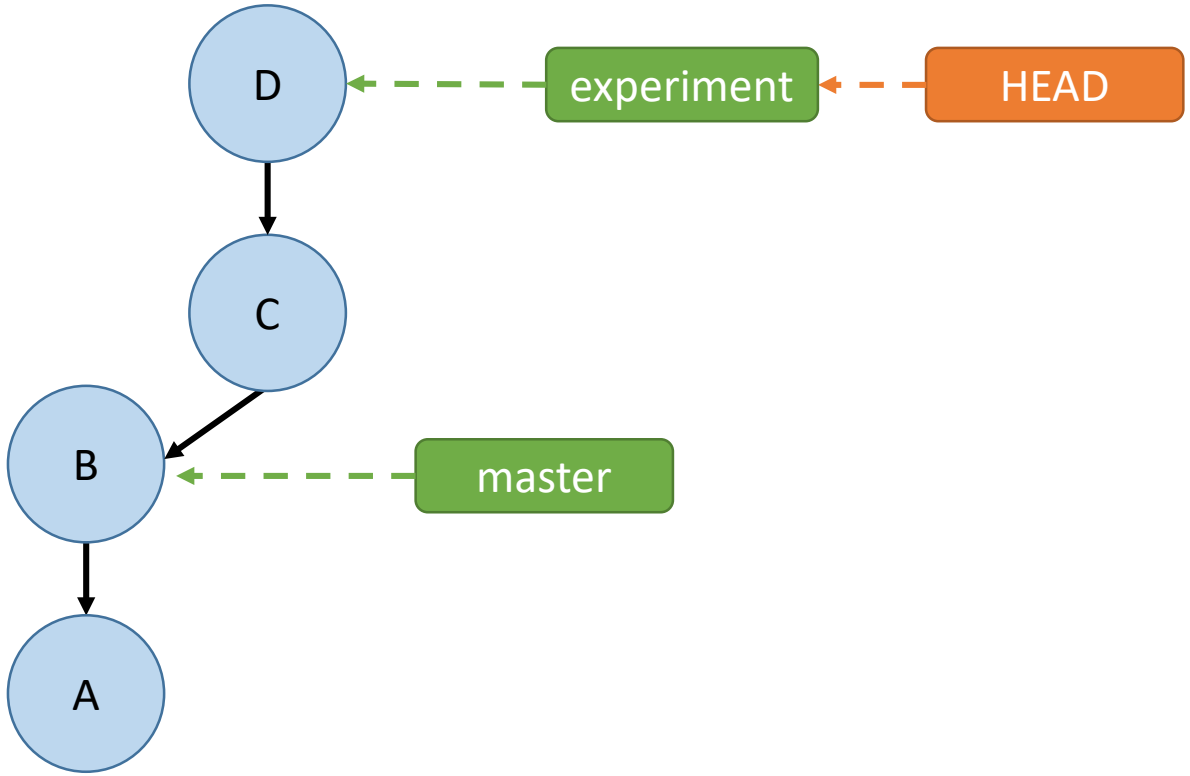
Commits are made on whatever branch you're on

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"



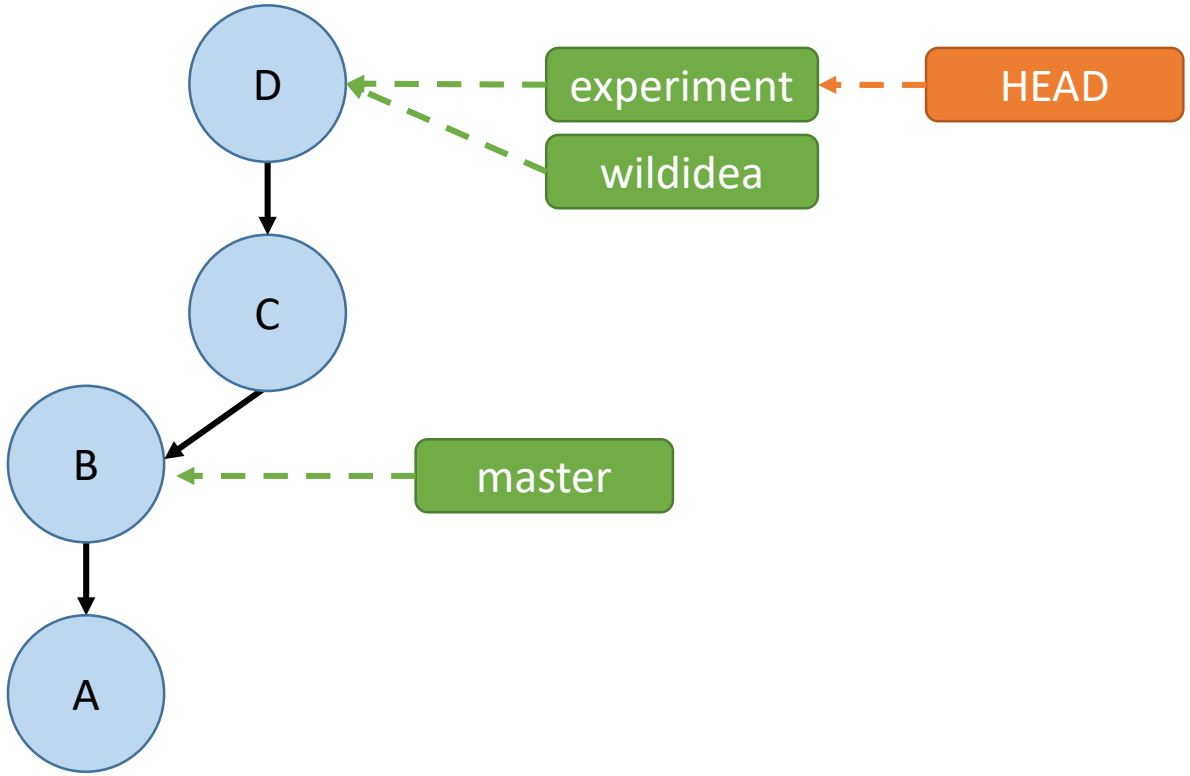
Commits are made on whatever branch you're on

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



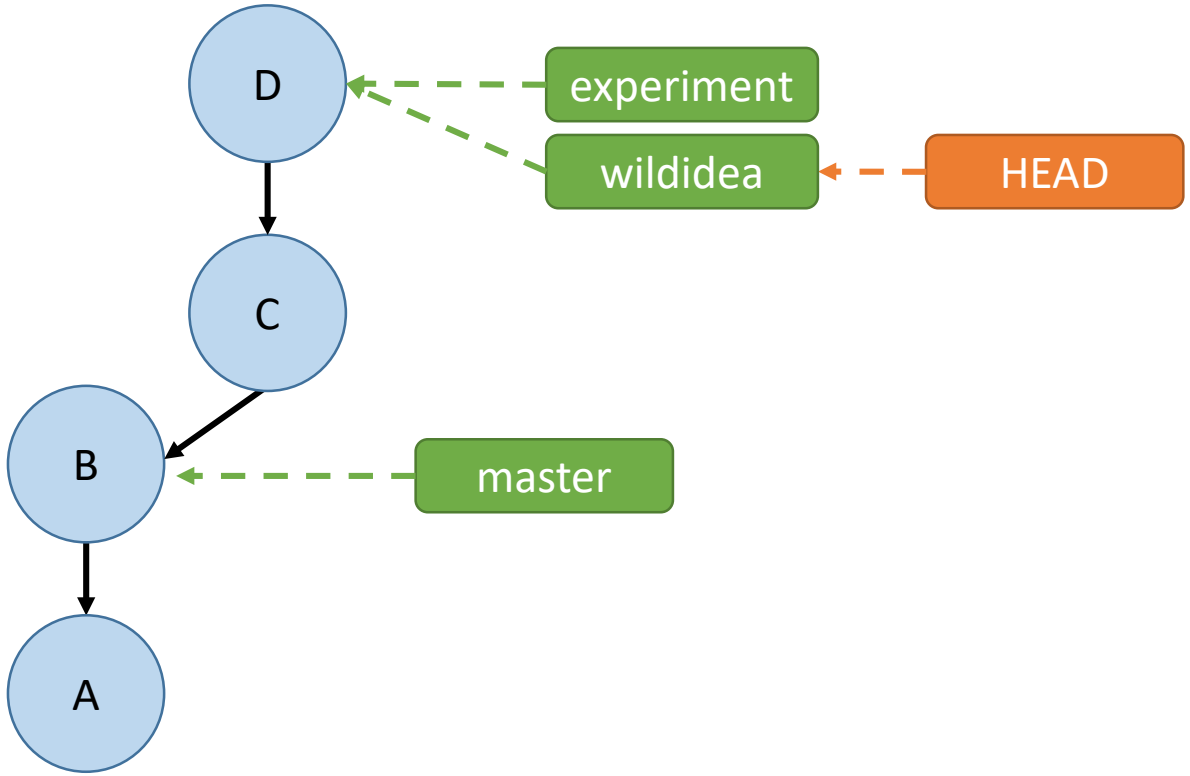
Commits are made on whatever branch you're on

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



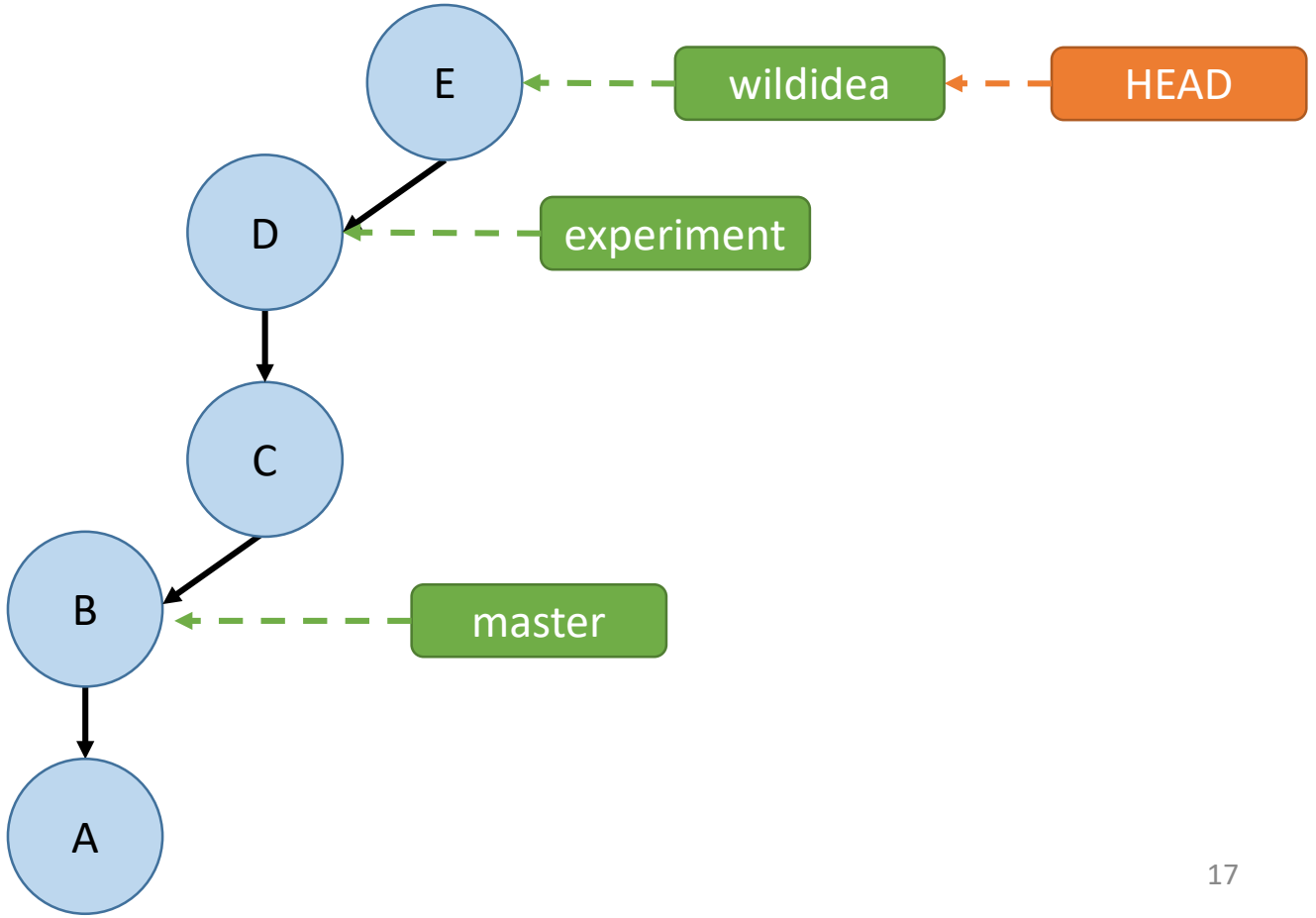
Commits are made on whatever branch you're on

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"



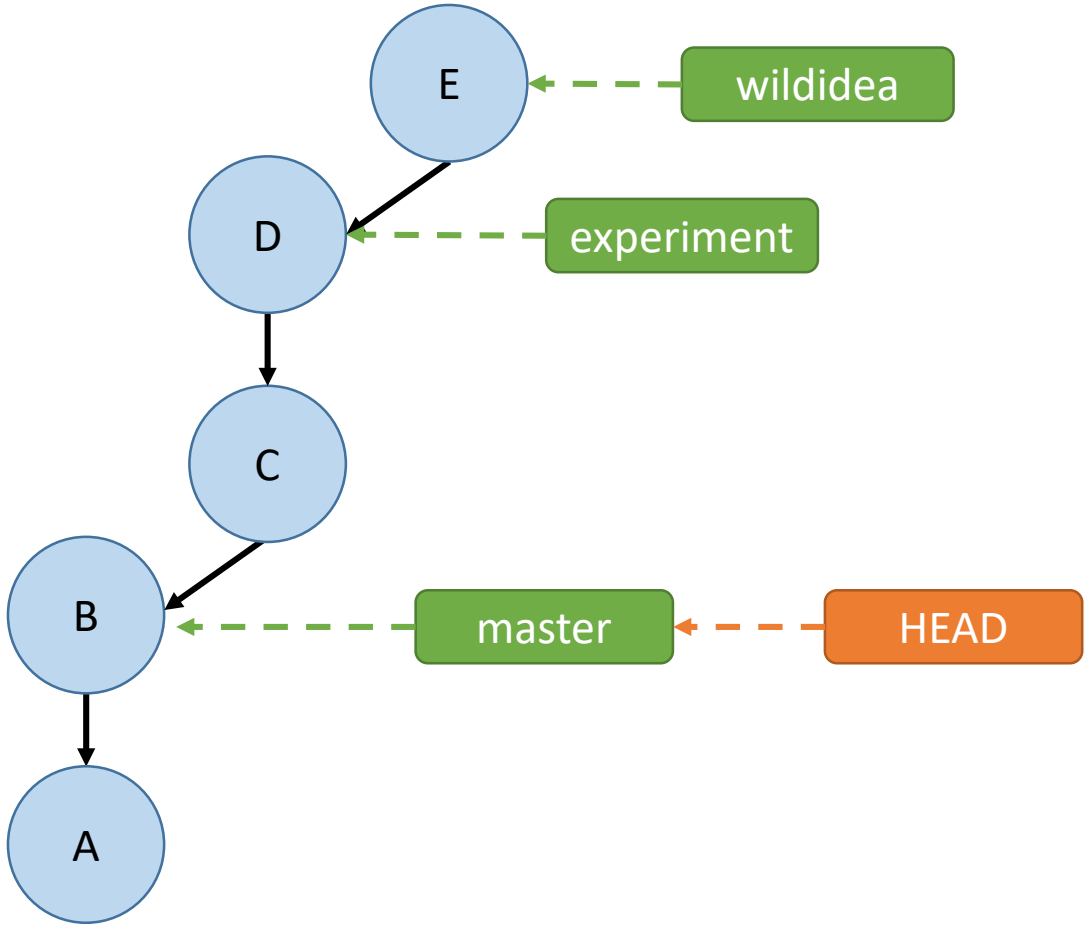
Commits are made on whatever branch you're on

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



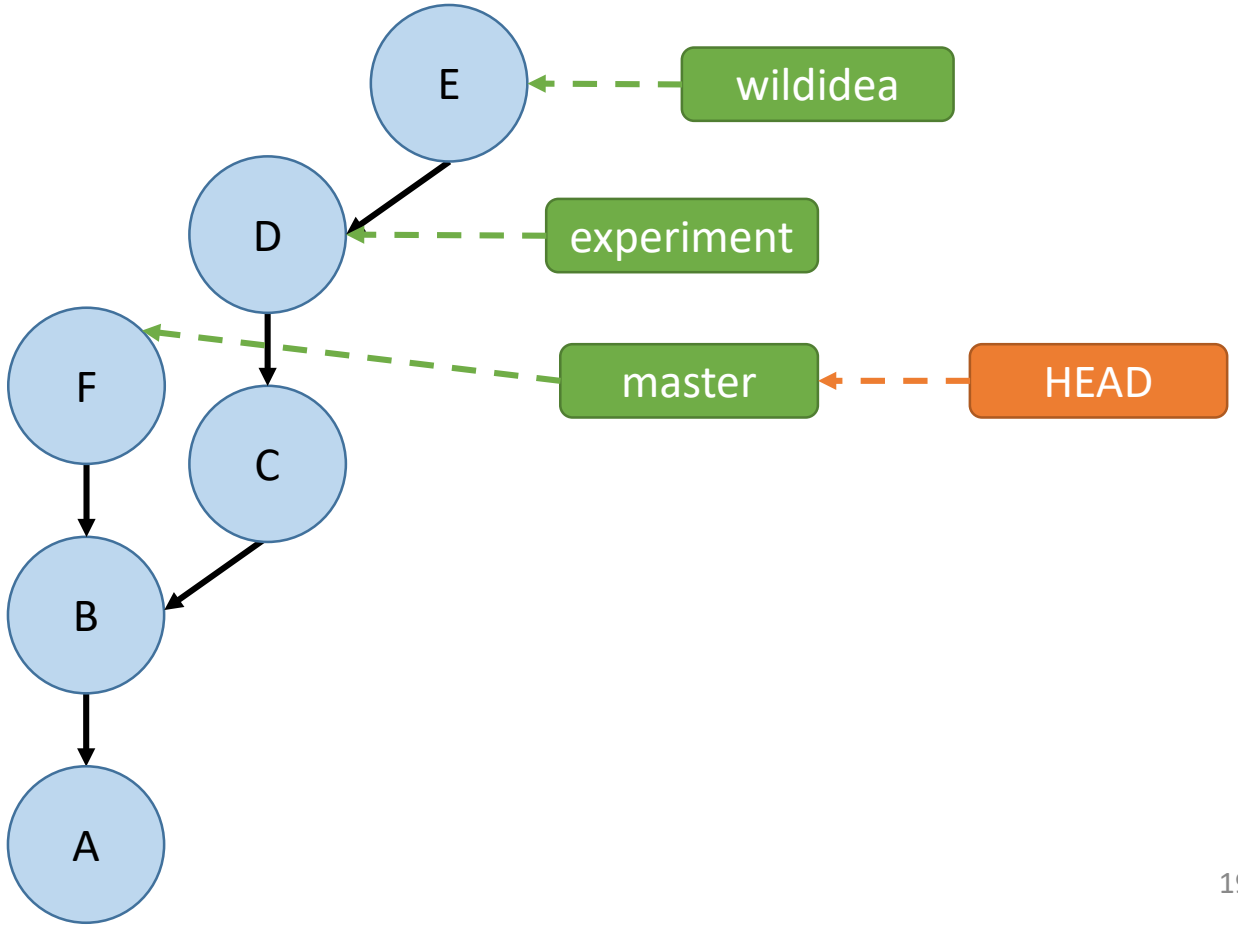
Commits are made on whatever branch you're on

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"

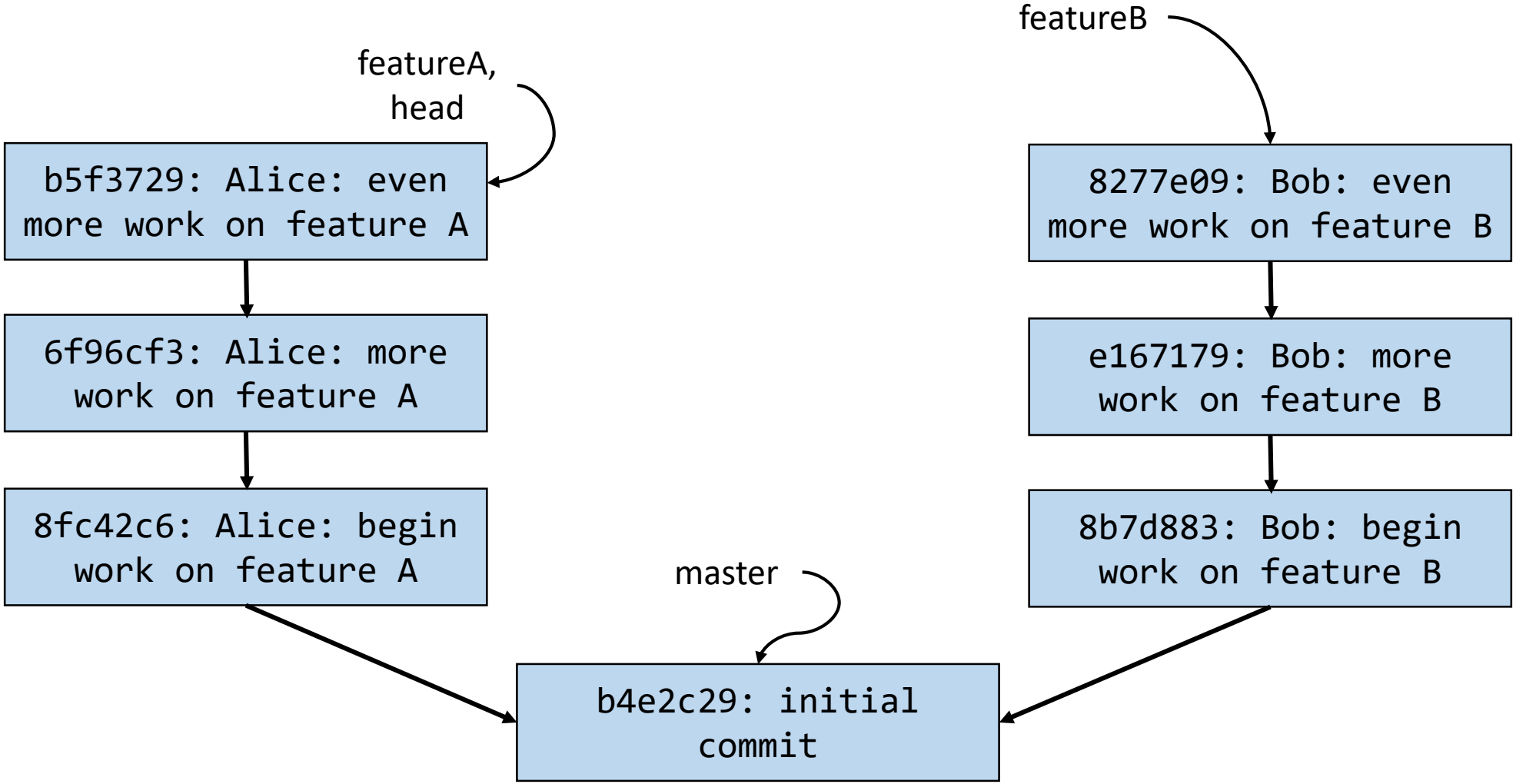


Commits are made on whatever branch you're on

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"

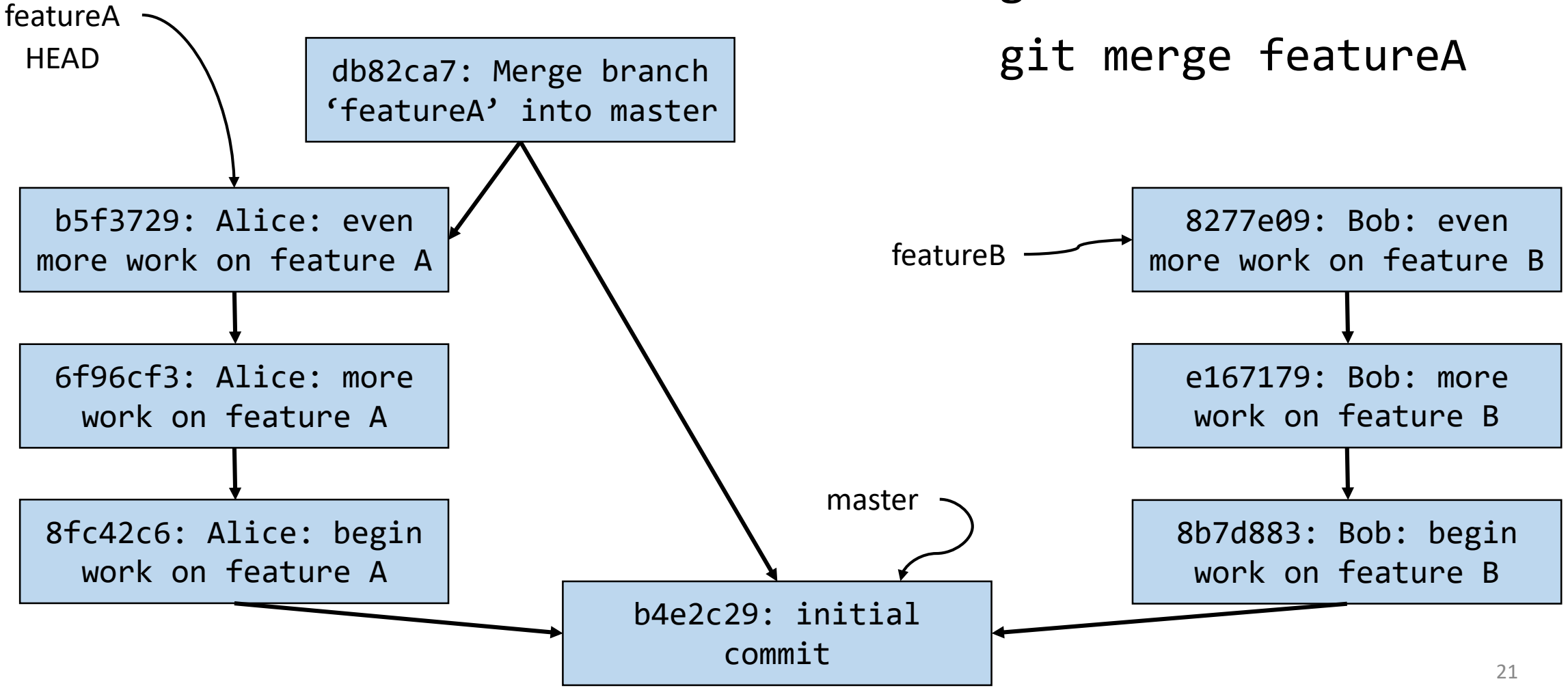


How do we bring branches back together?

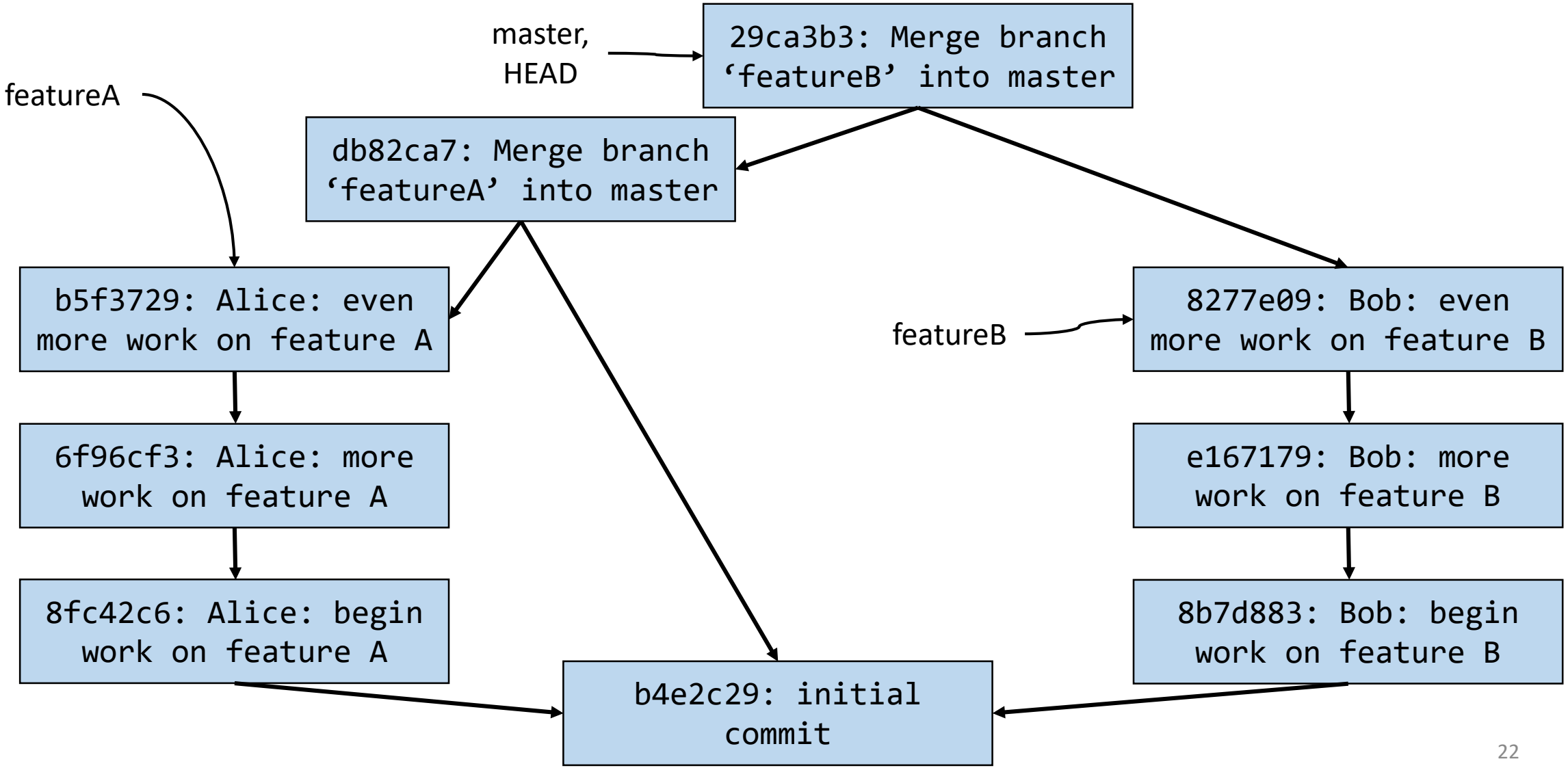


How do we bring branches back together?

```
git checkout master
git merge featureA
```



How do we bring branches back together?



`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

- On GitHub, you merge by means of handling a pull request

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

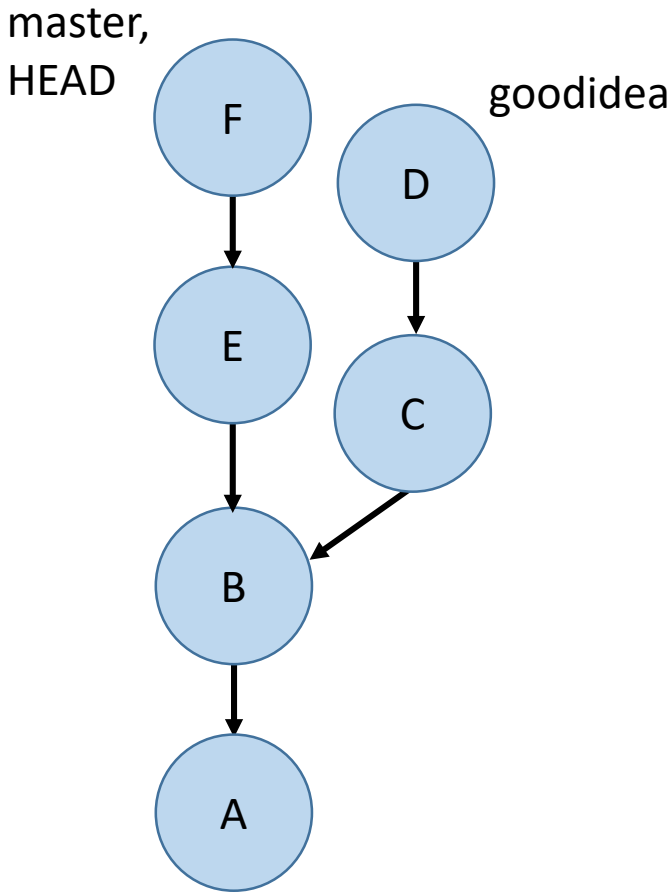
- Any guesses?

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

- It doesn’t.

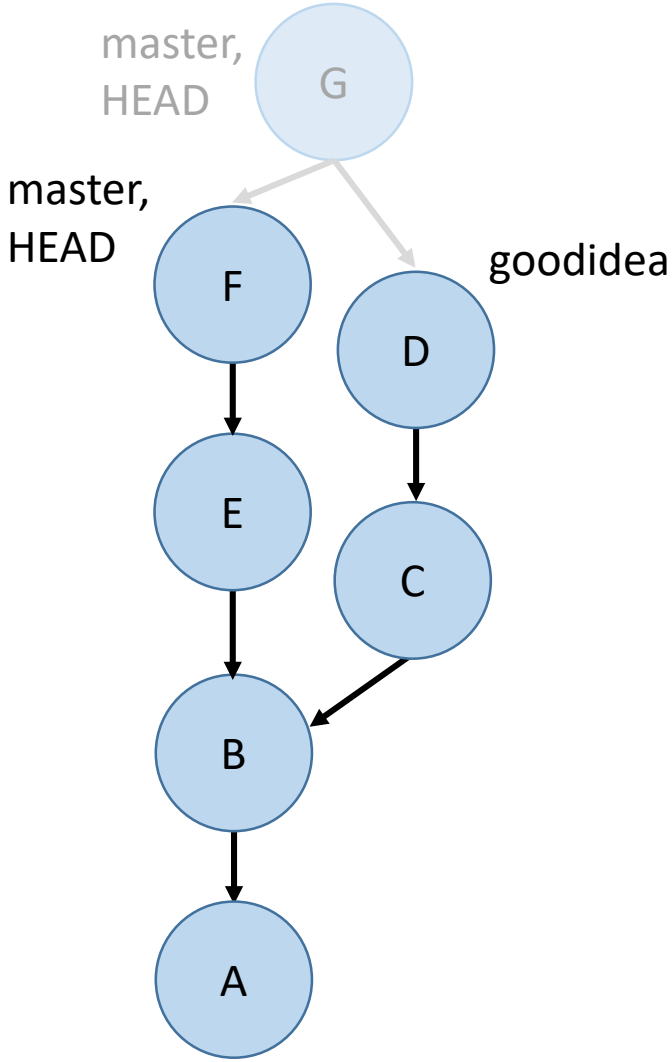
Most cases: Merging with possible conflicts

- 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

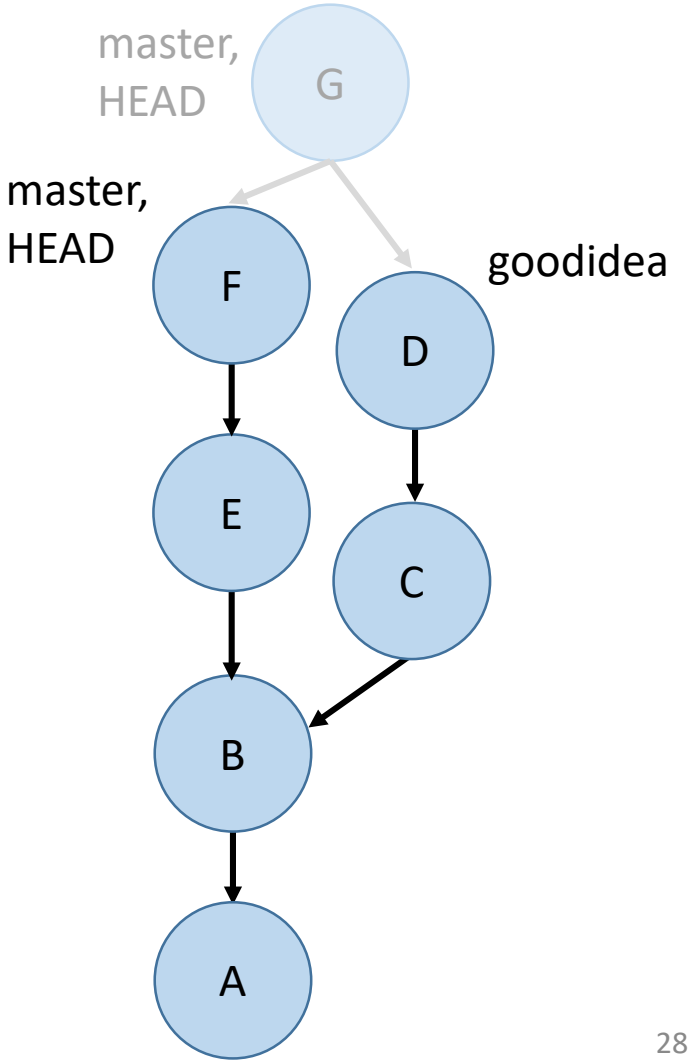
- Let's say I'm on master (as denoted by HEAD) and I want to merge goodidea into master.
- git merge goodidea
- At this point, if bringing in all the changes from goodidea do not conflict with the files in master, then a new commit is created (you'll have to specify a commit message) and we're done.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

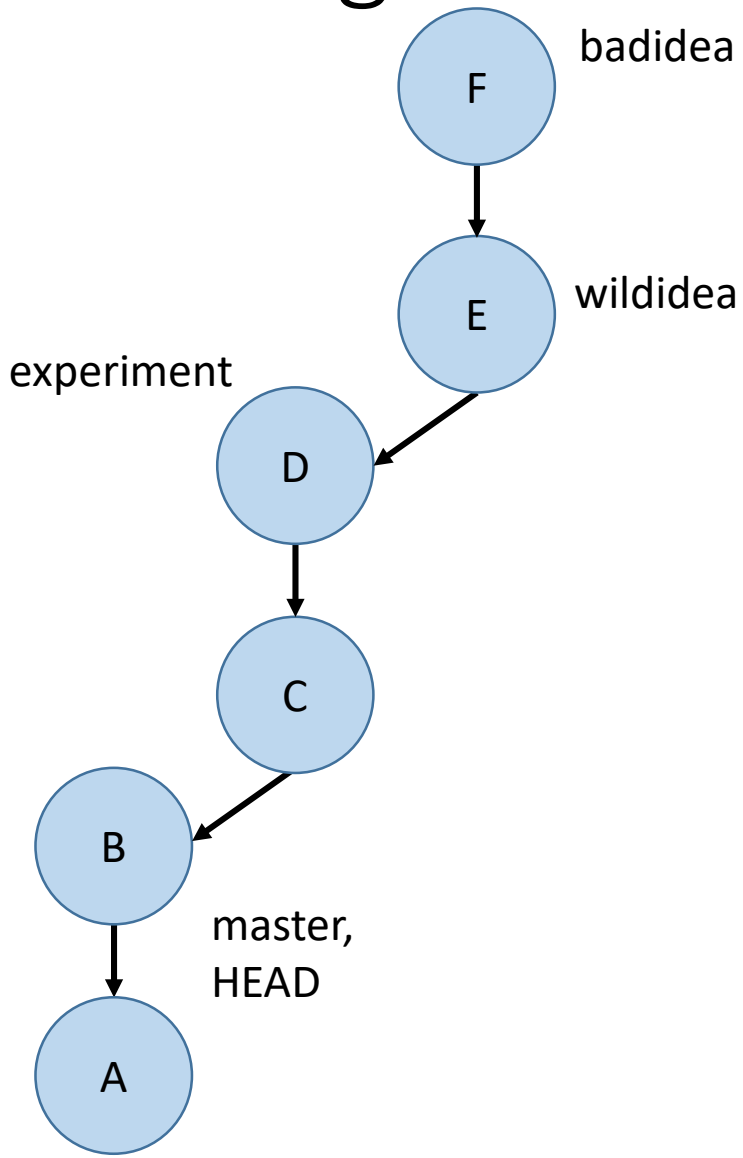
“How to fix a merge conflict”

- 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

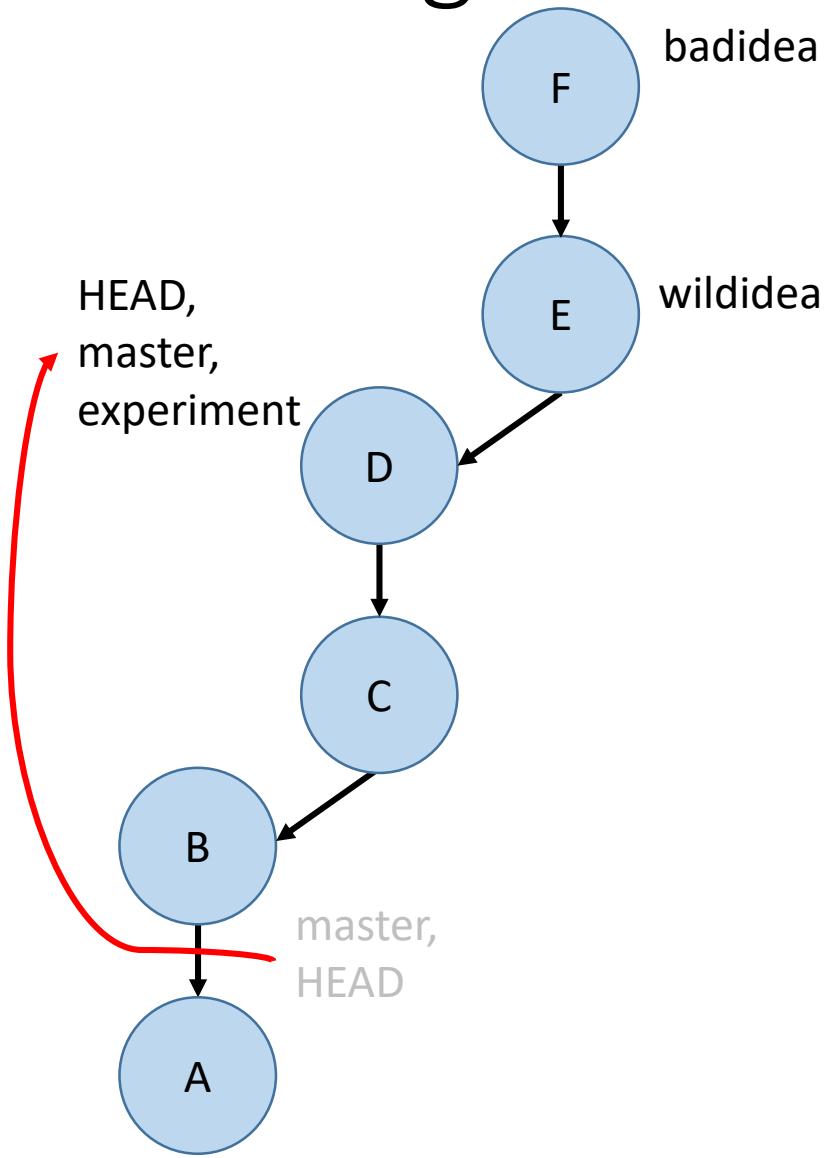
git merge experiment



Special Case: Fast-forward merges

git merge experiment

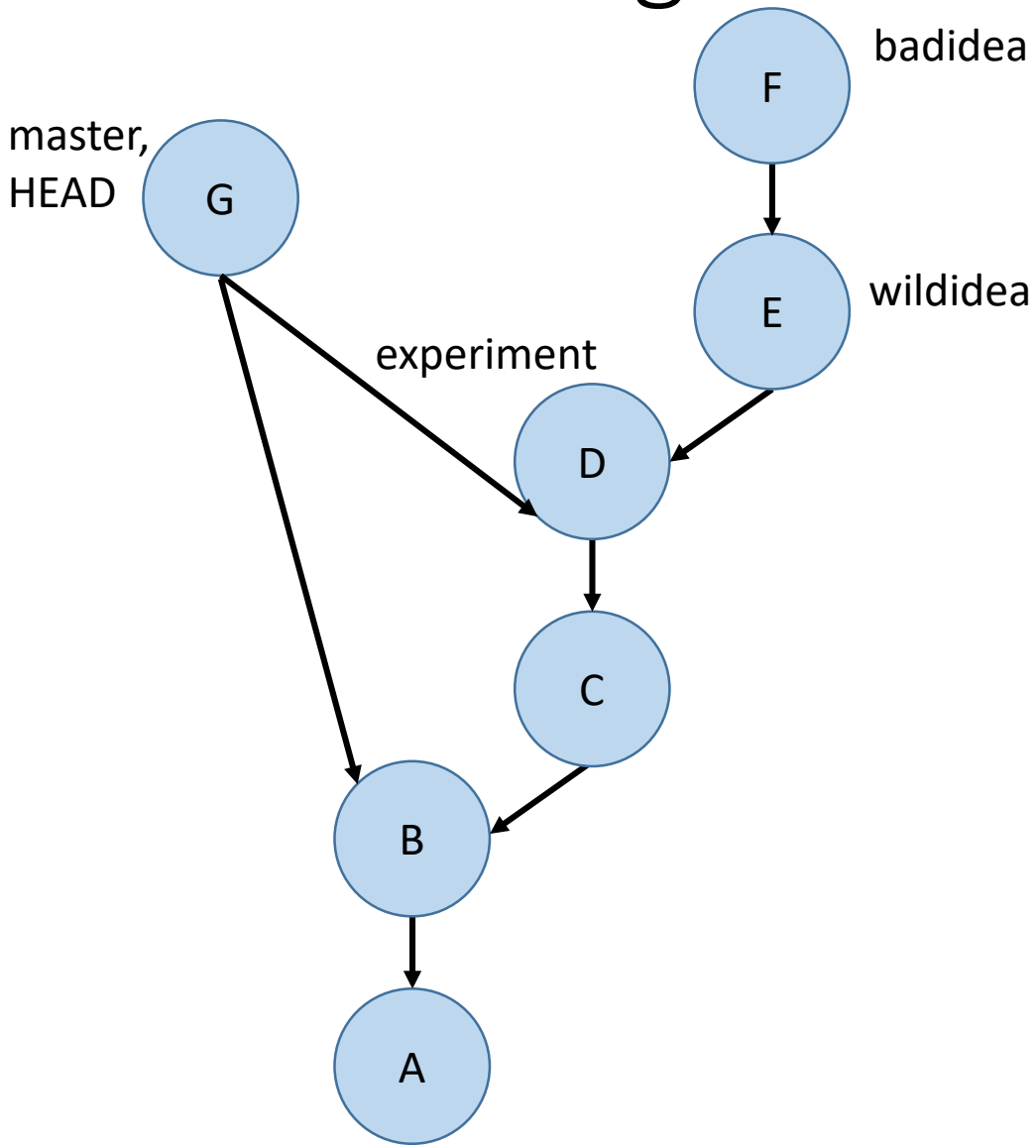
Git doesn't bother creating another commit to combine the changes because this kind of merge is guaranteed to not have conflicts.



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