little languages

lecture 17:

git merging

Start-up the VM!
No lecture directory today.

Workflows

- Different teams will have different workflows for organizing repositories

- Rules will inform:
  - When and why should you establish a branch?
  - When are you allowed to merge a branch back in?
  - How do you *release* a version of a project?
  - How do you *catch up* a repository?

- Often in organizations these rules will be influenced by:
  - Are you passing all tests?
  - Have you done a code review?
  - Will your branch merge cleanly?

- Today we'll explore the most important skill in these workflows: merging.
  - For the full story on how large teams operate, read more here:
  - https://nvie.com/posts/a-successful-git-branching-model/

Starting a new `git` Repository

1. Make a directory for your project:

```
cd $HOME
mkdir git-workflow
cd git-workflow
```

2. Initialize the git repository:

```
git init
```

3. Setting up the project on GitHub, too?

```
git remote add origin git@github.com:<UserName>/<Repository>.git
```

Follow-along: Initial Commits

- Let's establish a file named lyrics.txt

- For the first commit, we'll add the line: *Is this the real life*
  - Commit message: *Start of a great song.*

- For the second commit, we'll add the line: *Is this just fantasy*
  - Commit message: *More of the song.*

Commit Aliases

- HEAD - the current commit you have checked out
- HEAD^ - the parent of HEAD
- HEAD~N - the Nth ancestor of HEAD
  - HEAD~1 is the same as HEAD^
  - HEAD~2 is the grandparent of HEAD
- See last 10 commits:

```
git log --oneline --graph HEAD~10..
```

Branches

- Commonly you'll use feature branches
  - Working on a new feature? Establish a branch!

- To create a branch, in two steps:

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

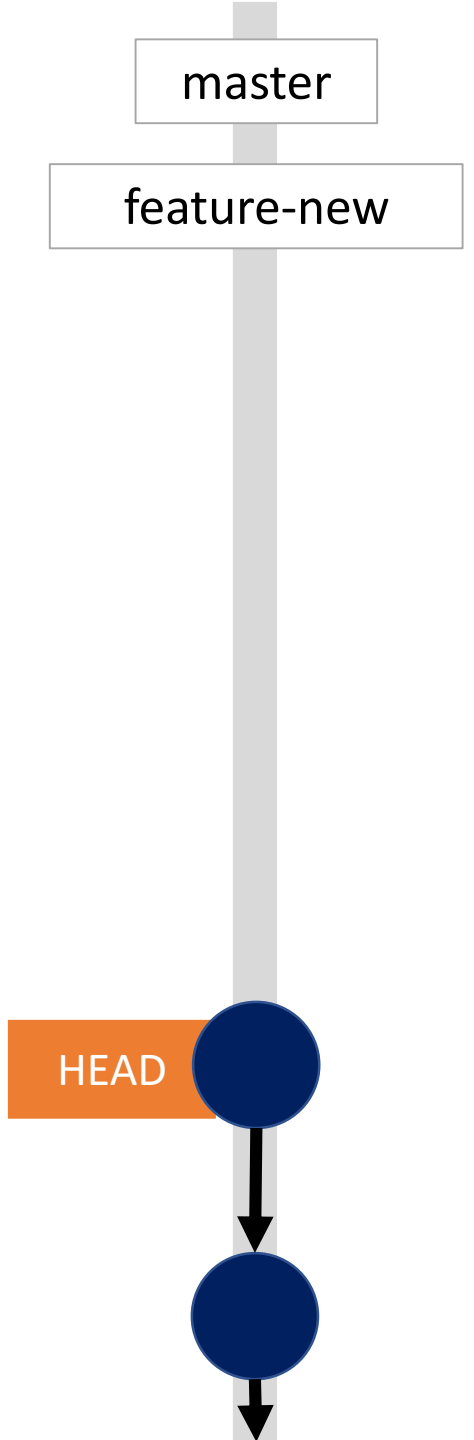
- These steps are usually combined:

```
git checkout -b feature-new
```

- Confirm branch checked out:

```
git branch
```

- Remember, a branch is just a pointer to a commit. So a new branch doesn't diverge until commit(s) are made.



Branches: Follow-along

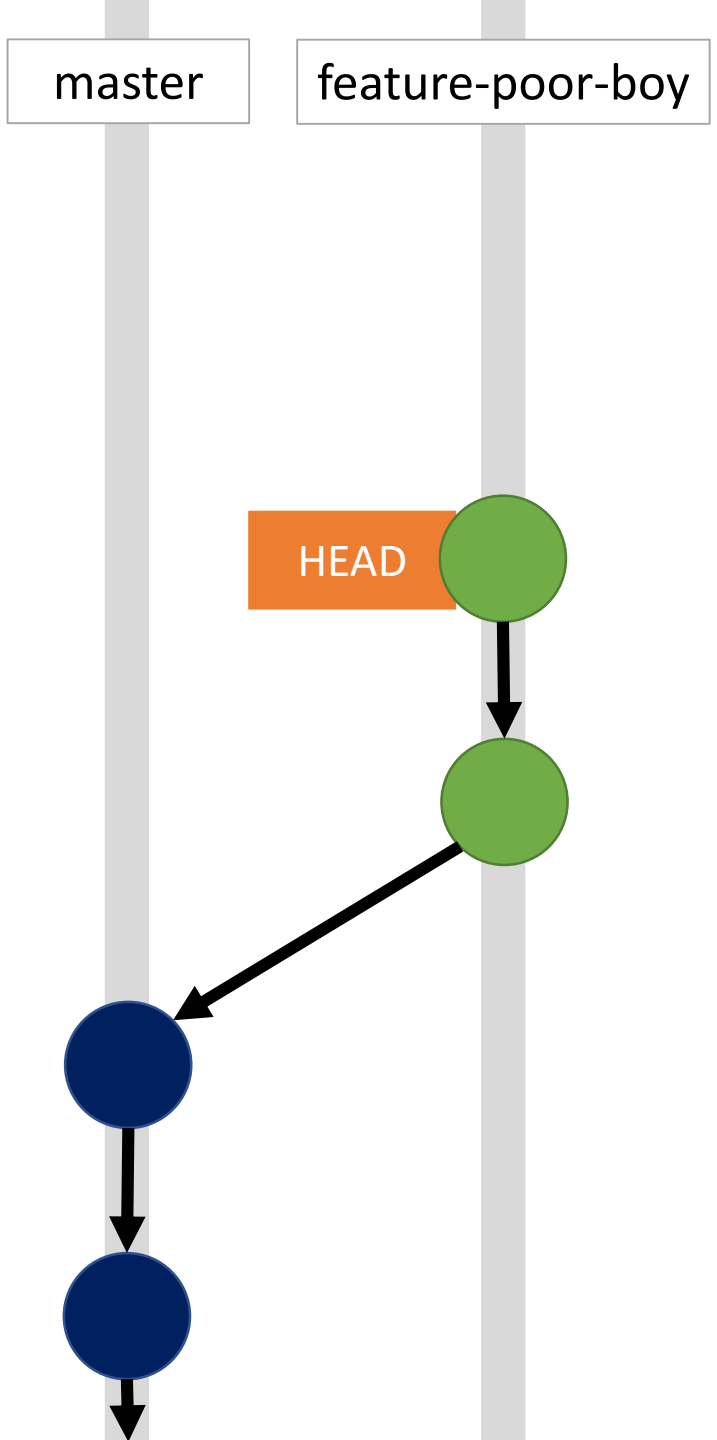
- Let's establish a branch for a new verse
  - `git checkout -b feature-poor-boy`

1. Add the following line to lyrics.txt:
   `I'm just a poor boy`

2. Make a commit with the message:
   `Starting the great poor boy verse.`

3. Add the following line to lyrics.txt:
   `and nobody loves me`

4. Make a commit with the message:
   `It keeps getting better.`



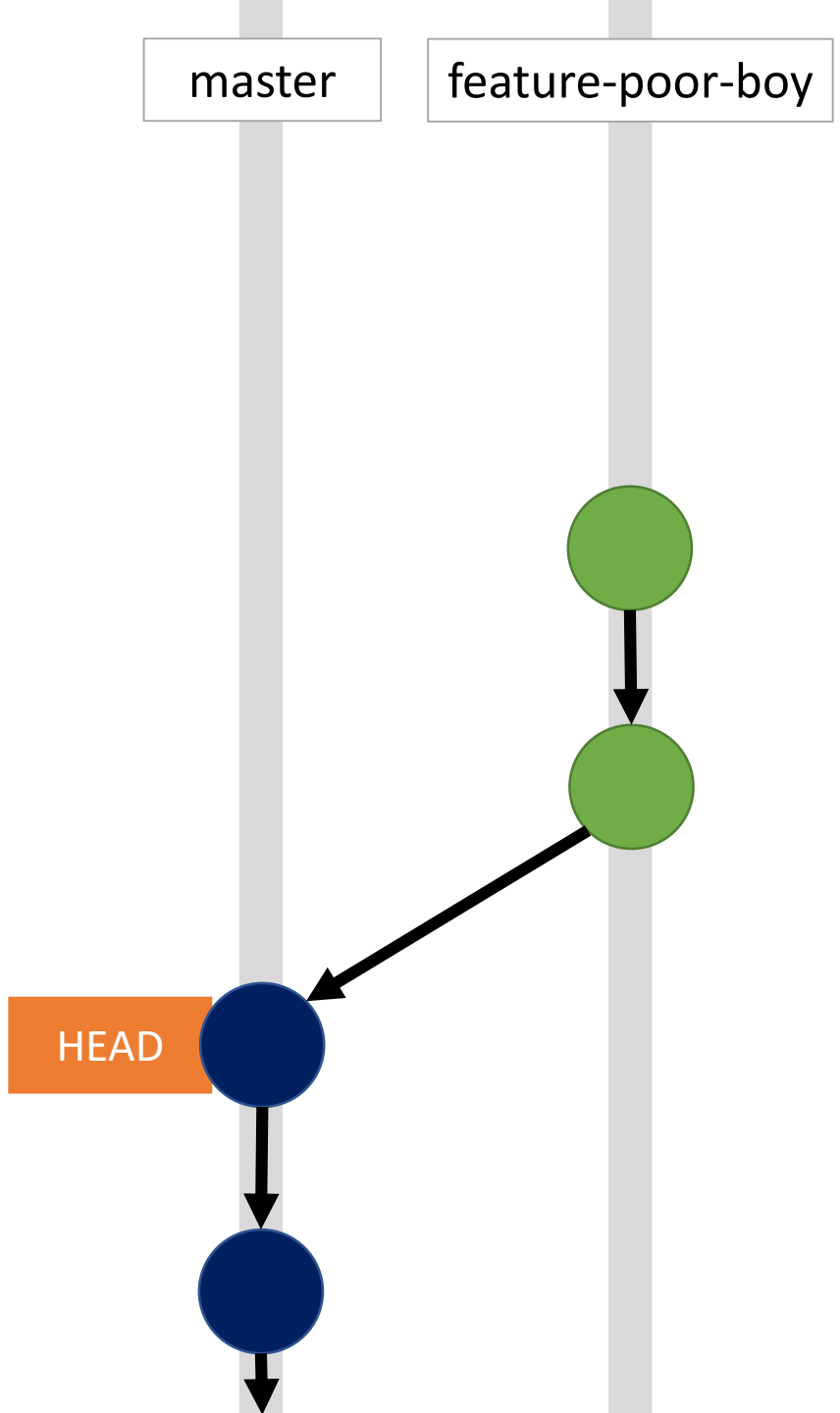
Branches: Changing Branches

- To change to a different branch, check it out:

```
git checkout master
```

- This changes the files in your working directory to exactly match their contents of the branch (commit) checked out.

- If there were uncommitted changes you would risk losing in this process, git would require you to deal with them first.



Divergent Histories

- Let's add another commit to master.

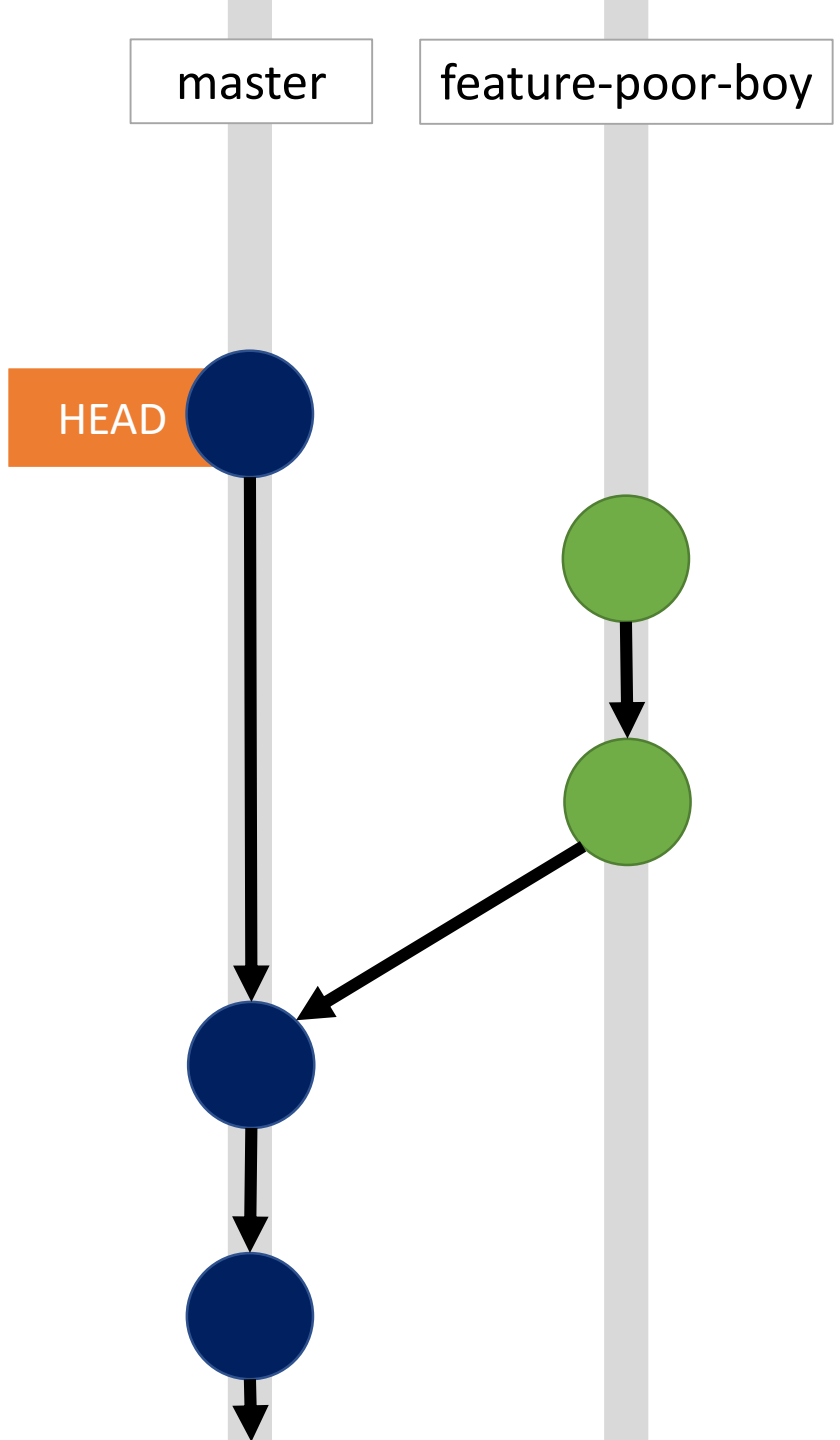
- Add another line to the first verse:

  ```
  Caught in a landslide
  ```

- Add a commit with the message:

  ```
  This song is getting good
  ```

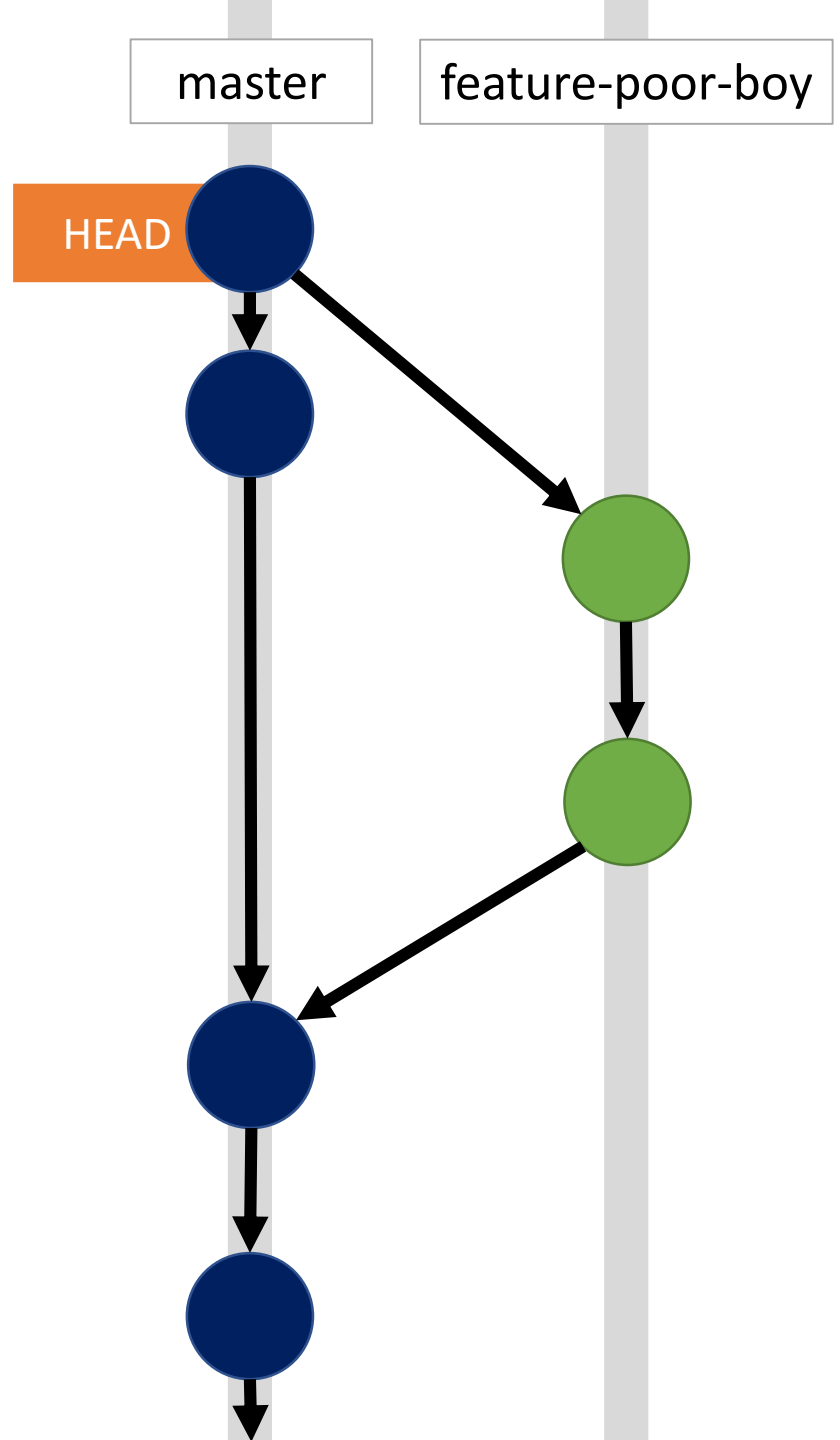
- Notice now our histories are divergent!
  - Visualize: `git log --oneline --graph --all`
  - This happens frequently in team projects. Team members make progress and the project advances while you're working on a feature.



Merging Feature Branches

- When merging feature branches it's a best practice to establish a "merge commit"
  - In doing so, the presence of the feature branch's commits *separate* from the master branch's is retained.
- In cases like the one we're in it's unavoidable.
- In other cases, as shown in a previous lecture, when there are no commits on the branch you're merging into (master), *fast-forwarding* avoids a merge commit.
  - Fast-forwarding makes it look as though intermediate commits were all a part of the master branch.
- To merge, checkout the branch you're merging into, then:

```
git merge --no-ff feature-poor-boy
```

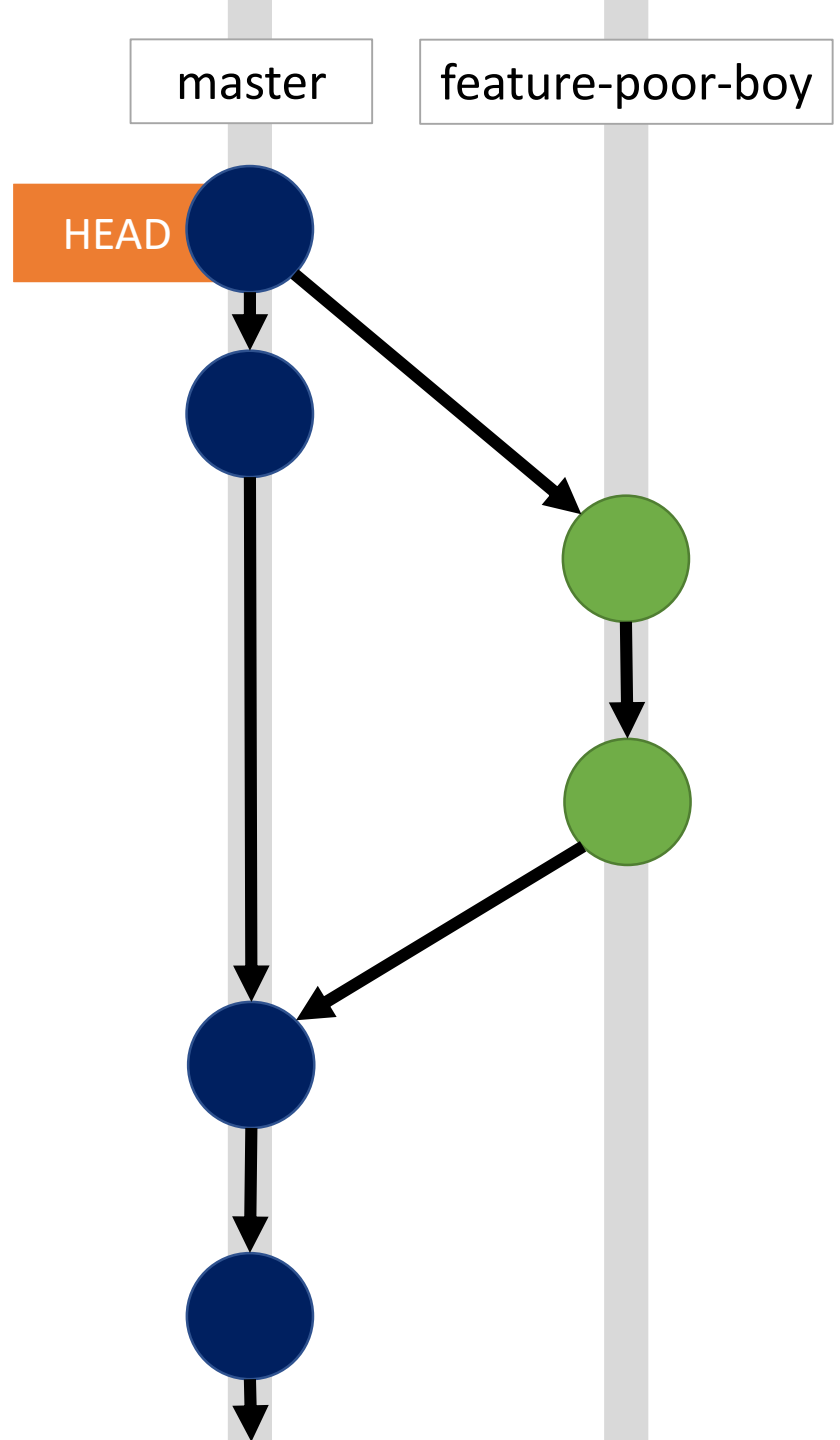


Merge Commits

- Notice a merge commit has two parents!

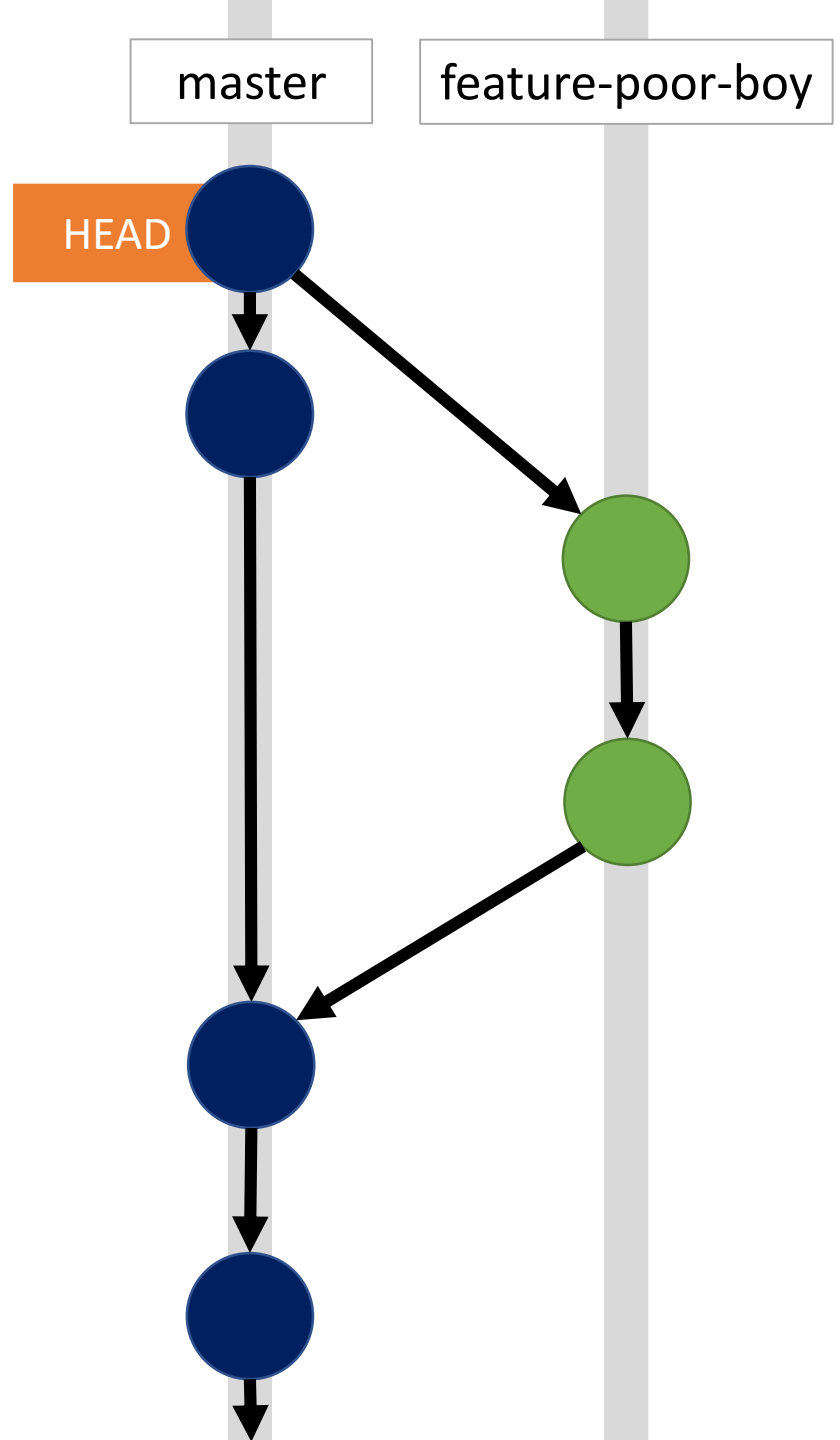
- Even though git has "branches" commit histories are not usually trees, they're graphs.

- More specifically, git repositories are
  - *directed*
    - each commit points to parent(s)
    - parents do not have references to children
  - *acyclic*
    - you cannot create a self-referential or cyclical history
    - there is a path from the current commit back to the start of the project
  - *graphs*



Conflicts

- What happens when two branches have modified the same parts of a file in divergent commits and you attempt to merge the branches?
- A conflict!
- Let's make one...



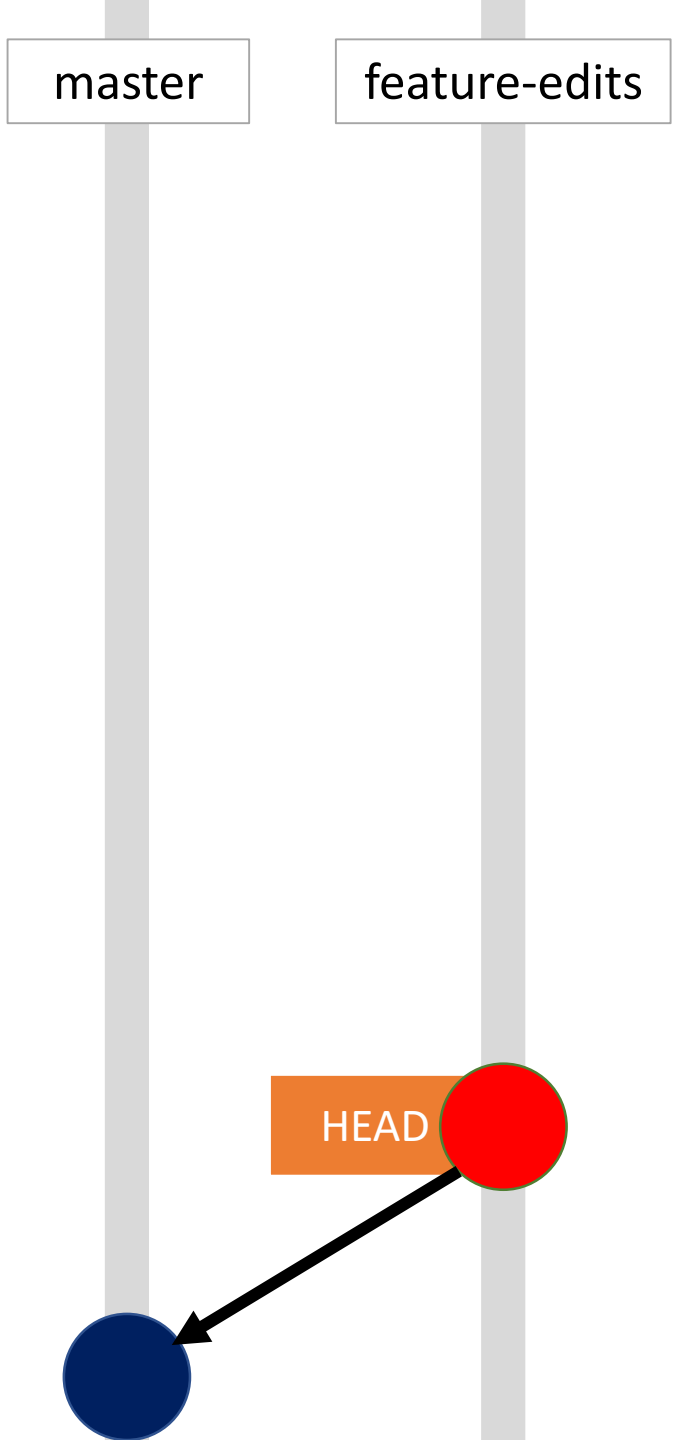
Creating a conflict (1/3)

- Let's create and checkout a new branch:

```
git checkout -b feature-edits
```

- Let's add a question mark to the first two lines:
  - Is this the real life?
  - Is this just fantasy?

- Make a commit with message:
  - git commit -m 'Add question marks'



Creating a conflict (2/3)

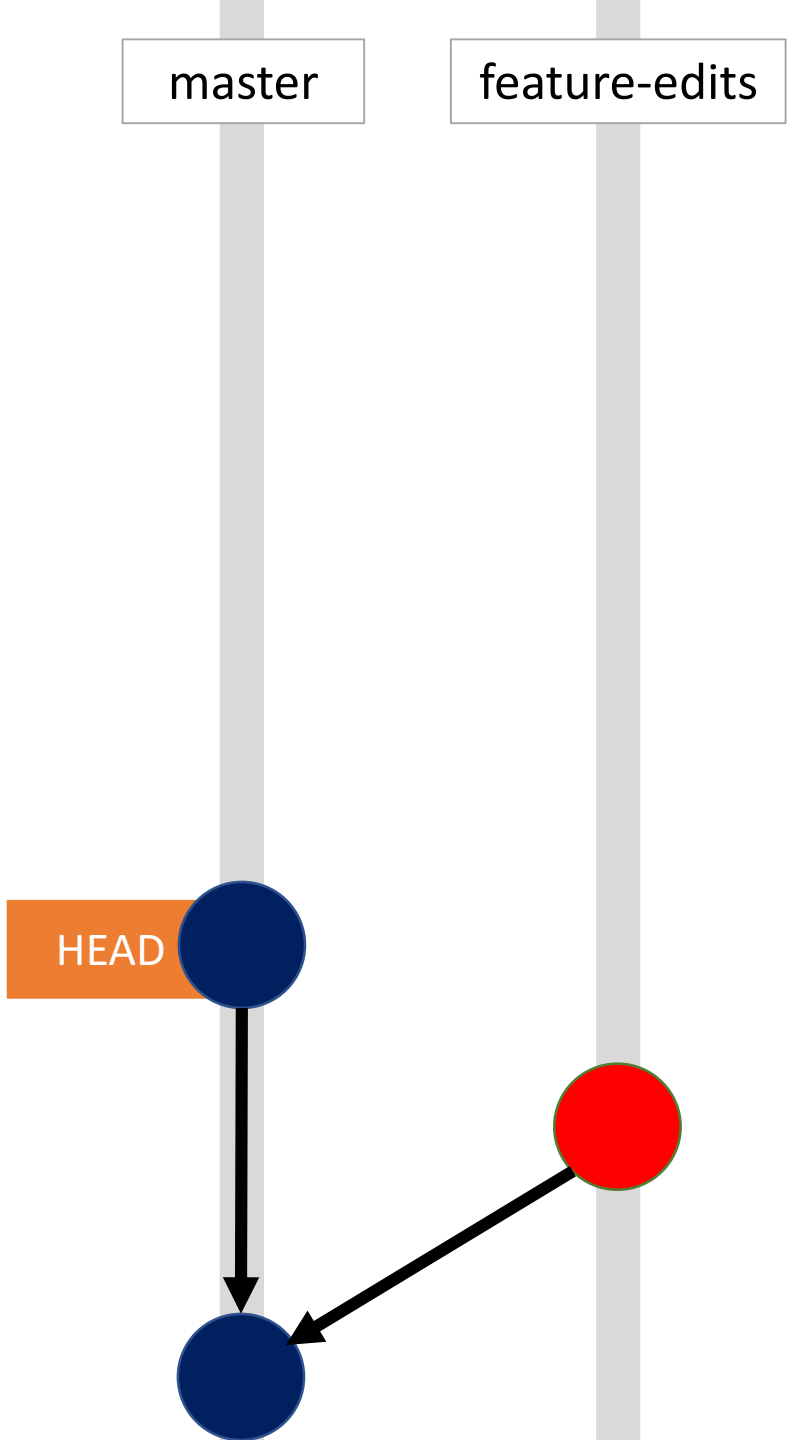
- Let's switch back over to master

```
git checkout master
```

- Let's make the first two lines a single line:
  - Is this the real life / Is this just fantasy

- Make a commit with message:

```
git commit -m 'Yas Queen'
```



Creating a conflict (2/3)

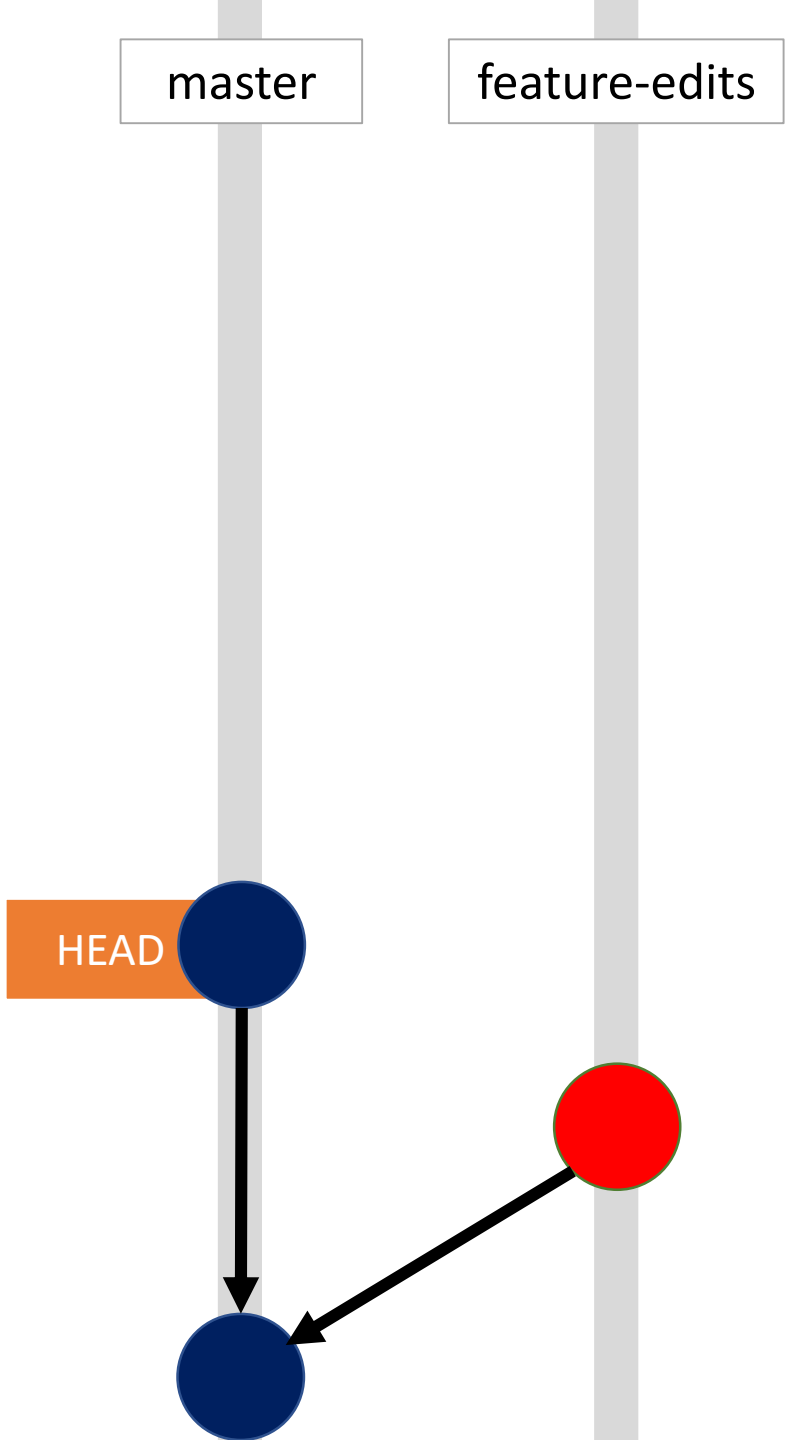
- Now let's merge edits onto master

```
git merge --no-ff edits
```

- Uh oh...
  - Auto-merging lyrics.txt
  - CONFLICT (content): Merge conflict in lyrics.txt
  - Automatic merge failed; fix conflicts and then commit the result.

- To see which files conflict, check status:

```
git status
```



Merging with Conflicts

- Two options:
  1. Abort Merge - **`git merge --abort`**
  2. Fix Conflict and Make Commit

- Opening lyrics.txt, you'll see the conflicting lines:

```
1 <<<<<<< HEAD
2 Is this the real life / Is this just fantasy
3 =======
4 Is this the real life?
5 Is this just fantasy?
6 >>>>>>> feature-edits
7 Caught in a landslide
```

master

feature-edits

- You decide what to keep and what to delete, make the changes, & save.
- Add the conflicting file to stage, make a commit, and you're merged!

Next Problem Set Series: **grep**

- For the next problem set you can choose to work solo *or* in pairs.
  - No groups larger than 2 will be permitted.
  - We encourage pairs but will require substantive contributions from both of you!

- Sequence:
  1. Tokenize and Parse a Regular Expression to print its Expr Tree
  2. Read in files line-by-line and (initially) print them to the screen
  3. Construct a non-deterministic finite state machine (NFA) from the Expr Tree
  4. Simulate the NFA by feeding it lines of text and printing lines with a match

- There will be much less hand-holding this sequence. You will be responsible for establishing the architecture of the program (being informed by the structure used in bc/dc is totally ok, but they're not solving the same problems).