git Branches

INFO 201

Joel Ross

Winter 2017

Final Projects

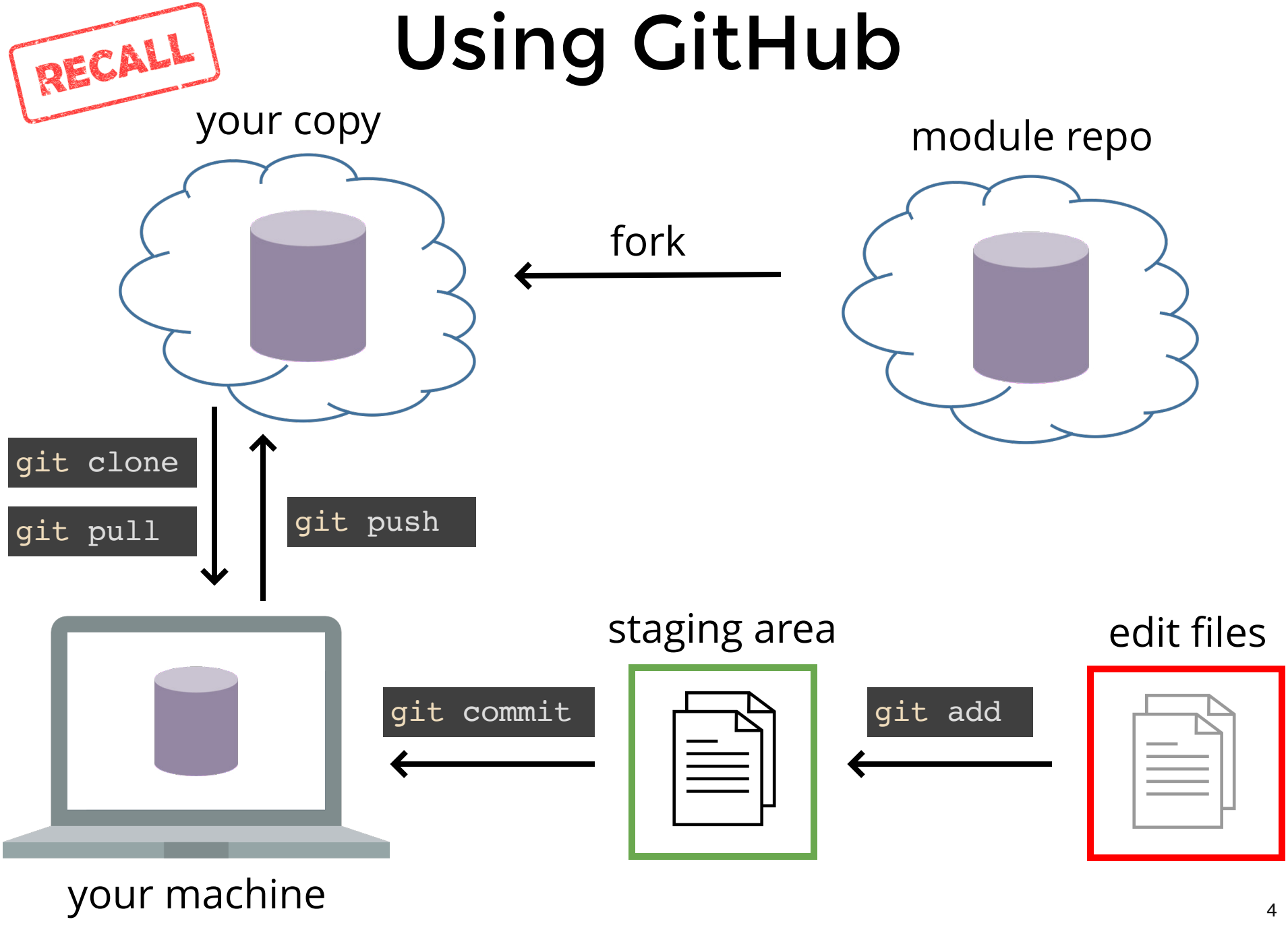
- Description is online
- Group project (groups of 3-4, within section)
  - Groups will be made in section this week: **show up!**
- Proposal due **Thur 02/23**

Today's Objectives

By the end of class, you should be able to

- Use **git branches** to track different versions of your code
- **Merge** changes between branches
- Resolve **merge conflicts**
- Host web sites with **GitHub Pages**

Using GitHub



Code for Today

https://github.com/info201-w17/git-branches

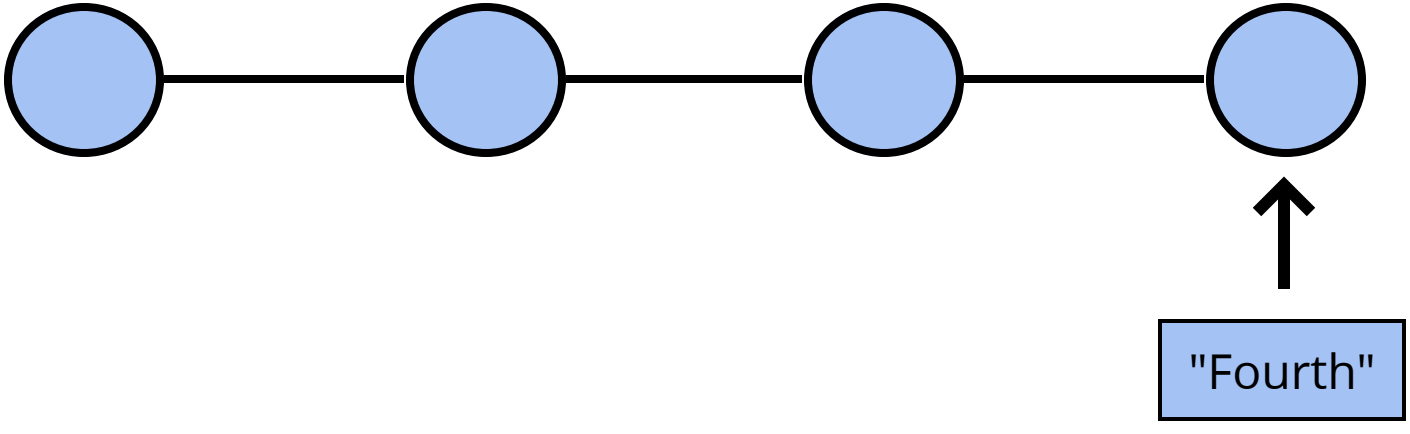
FORK and clone this repo!

RECALL

Commit History

Git history has been a **linear sequence** of commits.

```
git commit -m "Fourth"
```



Why Non-Linear?

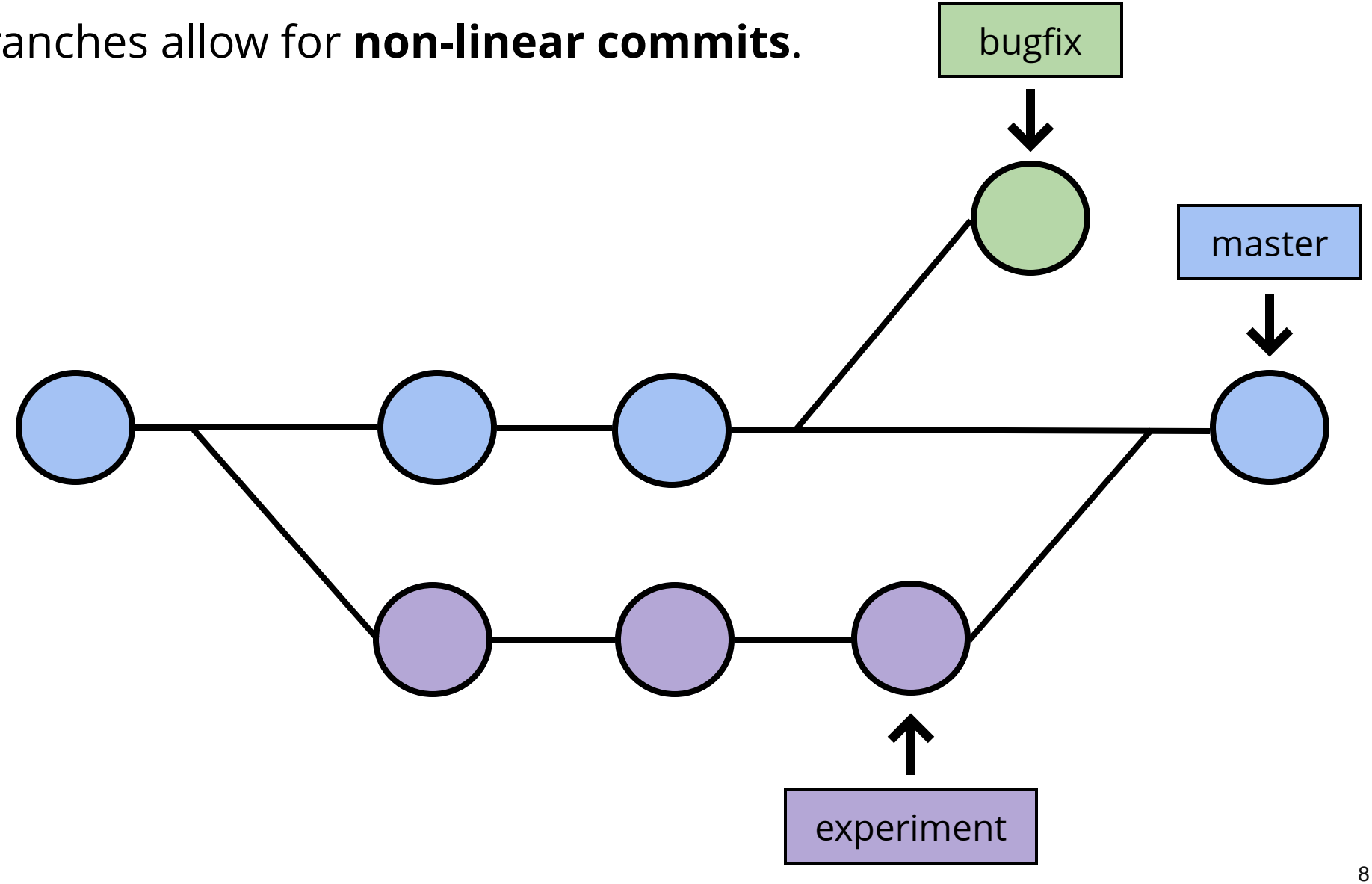
- What if we want to try something new and crazy **without breaking code** that we've already written?

- What if we want to work on two different features **simultaneously**?

- What if we want **multiple people** to work on the same code without stepping on each other's toes?

Branches

Branches allow for **non-linear commits**.



Branch Commands

```
git branch
```

List available branches

```
git branch [my_branch]
```

Create a new branch called "my_branch"

```
git checkout [my_branch]
```

Switch to branch "my_branch"

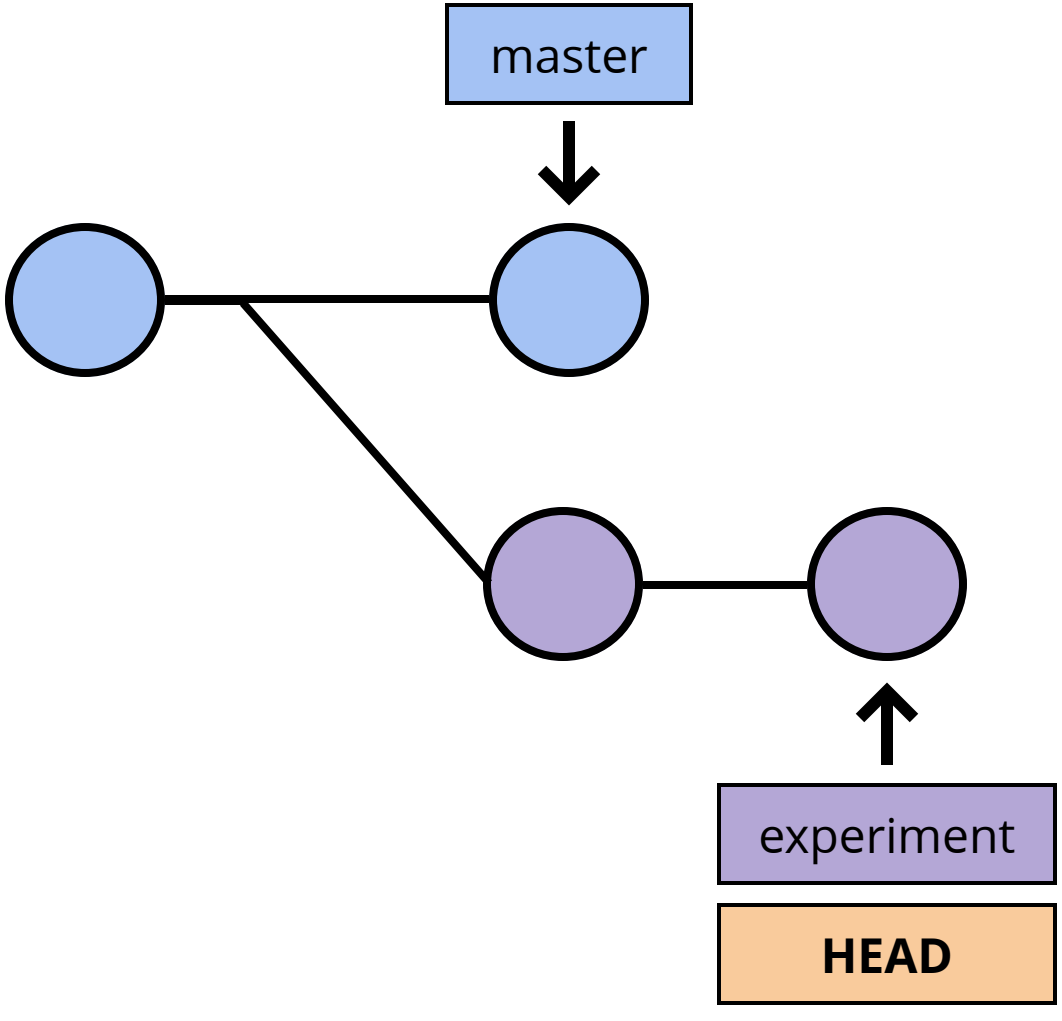
```
git checkout -b [my_branch]
```

Create and switch to branch "my_branch"

```
git branch -d [my_branch]
```

Delete branch "my_branch"

Branches



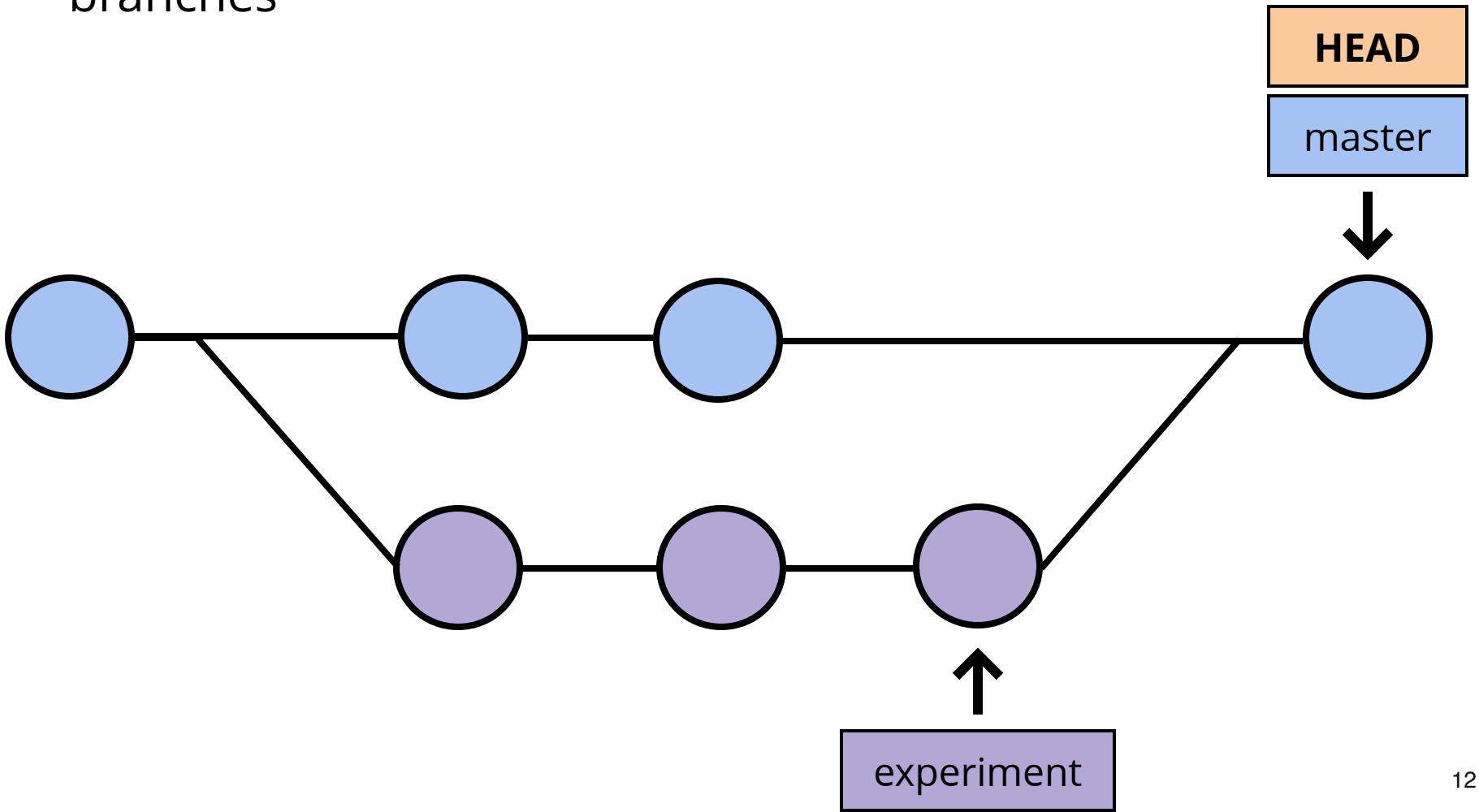

```
git branch experiment

git checkout experiment

git commit

git commit

git checkout master

git commit

git checkout experiment
```

Branch Practice!

1. Open the **README.md** file (in Atom)
2. Create and `checkout` a new branch called **experiment**
3. Add another item *to the end of the list*
4. Commit your change (hint: use `git commit -am "msg"` to add and commit at once!)
5. `checkout` the **master** branch
6. Add yet another item *to the **beginning** of the list*
7. Commit your change
8. Switch between the **experiment** and **master** branches (clicking on Atom in between). See the file contents changing?

Merging

We can `merge` two branches back together, producing a commit that contains the combined changes from both branches



Merging

```
git merge [other_branch] --no-edit
```

Merges changes from *other_branch* **into** the current branch.

A new commit is created on the **current** branch containing the merged content.

Merging Practice

1. Make sure you are on the **master** branch (use `git branch` to check; the current branch has a `*`)
2. Use `git merge` to merge the **experiment** branch into **master** branch.
   - Remember to use the `--no-edit` flag so you don't get dropped into **vi**! If you do, hit `:q` (colon then q) to flee.
3. Check in Atom that the file now contains both sets of changes!

Merging: A Metaphor

Merging Practice II

1. You should be on the **master** branch.
2. Create and `checkout` a new branch called **danger**
3. On the **danger** branch, change the word "kittens" to "puppies". Remember to `commit` your change.
4. `checkout` the **master** branch again.
5. Change the word "kittens" to something else that is pleasant. `commit` your change.
6. Use `git merge` to merge the **danger** branch into **master** branch

7. **DON'T PANIC**


MERGE CONFLICTS
MERGE CONFLICTS EVERYWHERE
memegenerator.net

Merge Conflicts

A merge conflict is when two commits from different **branches** include different changes **to the same code**.

Git does not know which version to keep, so makes *you* choose.

Merge conflicts must be resolved **manually**

Conflicts are expected!

Resolving Conflicts

In order to **resolve** a conflict, you need to edit the file (code) so that you pick which version to keep.
git will add "code" where you need to make a decision:

```
<<<<<<< HEAD
```

the two versions to pick from

```
# This is the code from the "local" version (the branch you merged INTO)
# a.k.a the version from the HEAD commit

message <- "I am an original"
lyric <- "I've got no strings to hold me down"

# There can be multiple lines that conflict, including lines being deleted

=======
```

a divider between the versions

```
# This is the code from the "remote" version (the branch you merged FROM)

message <- "I think I'm a clone now..."

# The lines need not be related in content, they've just changed in a way
# that git can't figure out which to keep!

>>>>>>> f292a3332aedc8df3e8e8cf22ca3debc214c6460
```

end conflict area

Resolving Conflicts

- Use `git status` to see which files have merge conflicts. Note that files may have more than one!

- Delete the `<<<<<<<` and `=======` and `>>>>>>>` !!

- Once you're satisfied that the conflicts are all resolved, `add` and `commit` your changes (the code you "modified" to resolve the conflict):

```
git add .
git commit -m "Merge branch 'other'"
```


BRACE YOURSELVES
MERGE CONFLICTS ARE COMING
memegenerator.net

Option: Rebasing

An alternative to **merging**.

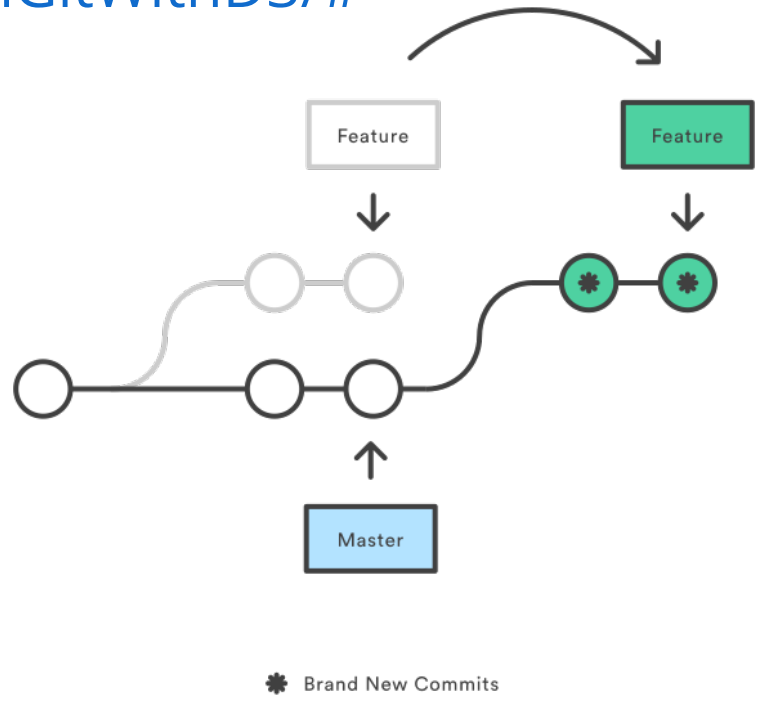
Rather than creating a new commit that is the "merger" of the two branches, takes the commits from one branch and tacks them on to the end of the other.

http://www.wei-wang.com/ExplainGitWithD3/#

This **changes history**.

My advice:

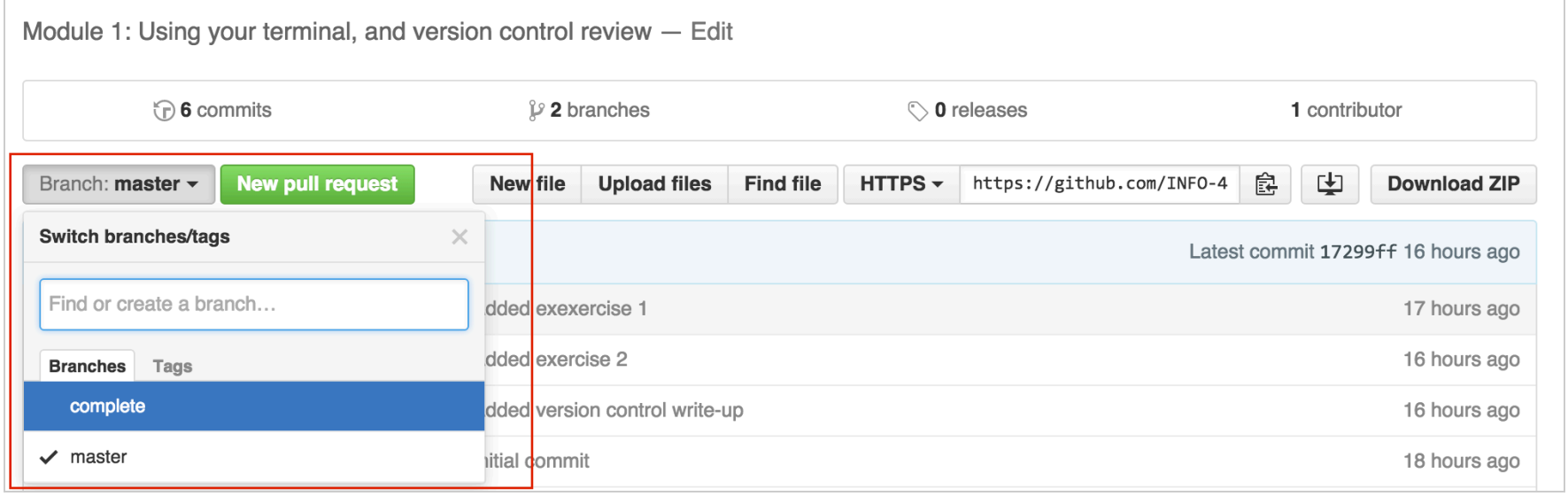
do not rebase



https://www.atlassian.com/git/tutorials/rewriting-history/git-rebase

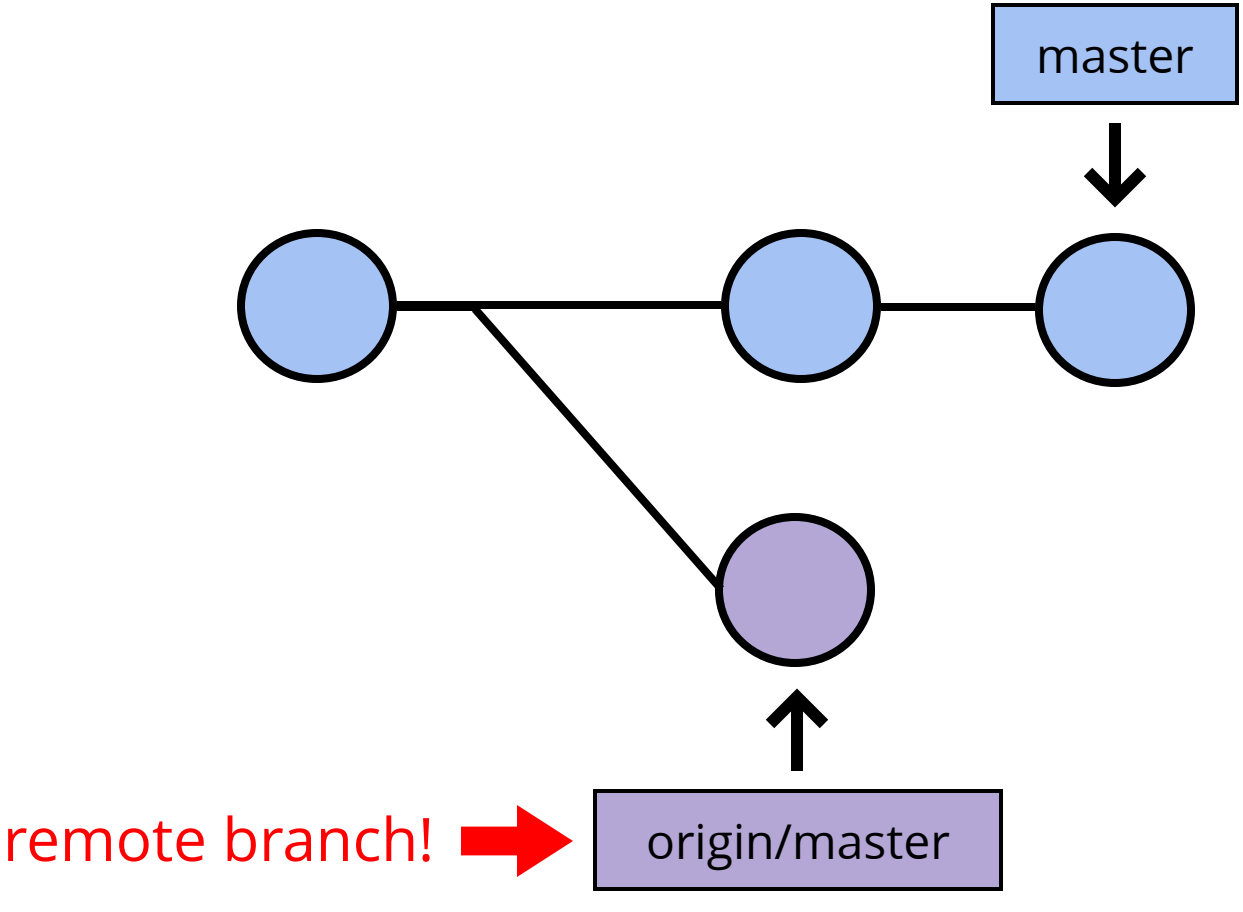
GitHub and Branches

Because GitHub just hosts normal repositories, GitHub has branches as well! These ***can*** (but need not) correspond with the branches on your local machine.



Remote Branches

Other linked repositories (**remotes**, like the one on Github that you cloned from) can simply be seen as different branches that happen to live on another machine.



Remote Branch Cmds

```
git branch -a
```

List **all** branches (including remote ones)

```
git fetch
```

Import branches from into local repo
Are still listed as "remote" branches that need to be *merged*

```
git pull [remote branch] --no-edit
```

Shortcut for `git fetch` then `git merge`

Remote Branch Cmds

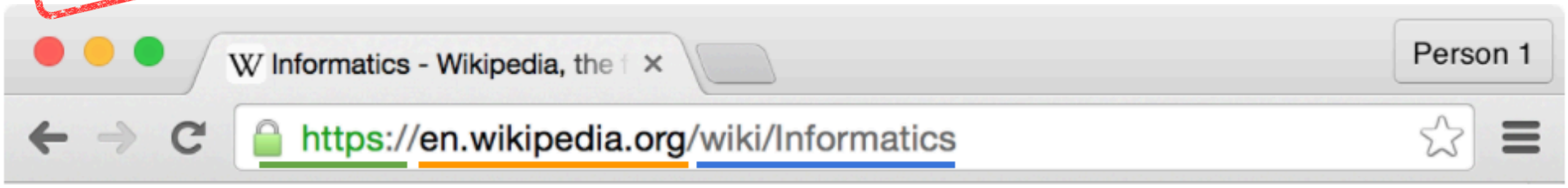
```
git push [remote branch]
```

Upload commits to remote
Essentially has the remote branch
merge (rebase) your changes.

```
git push [remote] --all
```

Push **all** branches

RECALL HTTP Requests

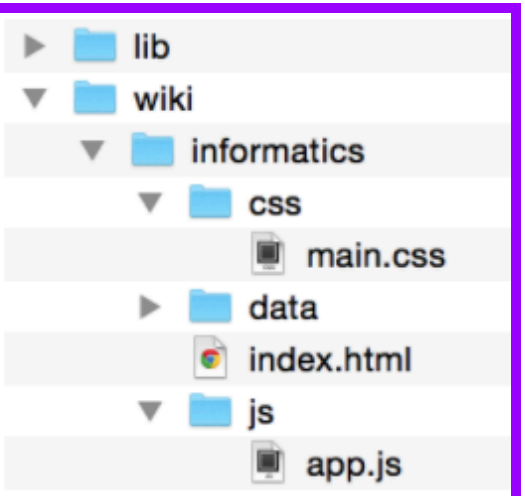


protocol (how to ask for data) | host (who has the data) | path (what data you want)

Request

"Hi Wikipedia, I'd like you to send me the wiki/Informatics data!"

Web Service



Response

GitHub Pages

A GitHub service that **hosts** web pages (`.html` files) found in a repository's **`gh-pages`** branch.

```
# Make sure you are on the `master` branch
git branch

# Checkout a new gh-pages branch from here. This branch will
# have the same commits as `master` to this point
git checkout -b gh-pages

# Upload web site to GitHub
git push -u origin gh-pages
```

Visit published page at:

https://GITHUB-USERNAME.github.io/REPO-NAME

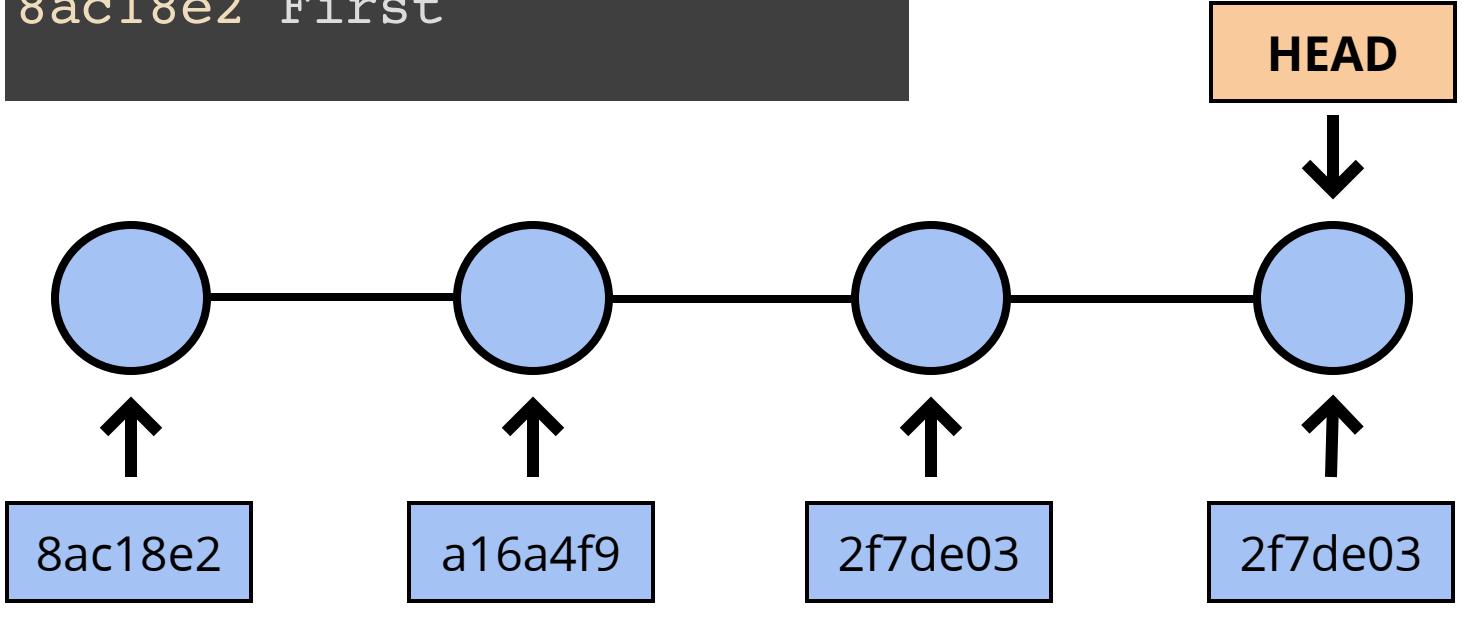
https://help.github.com/articles/user-organization-and-project-pages/

Github Pages Practice

Commit IDs

Each commit has a unique **commit id** that refers to it

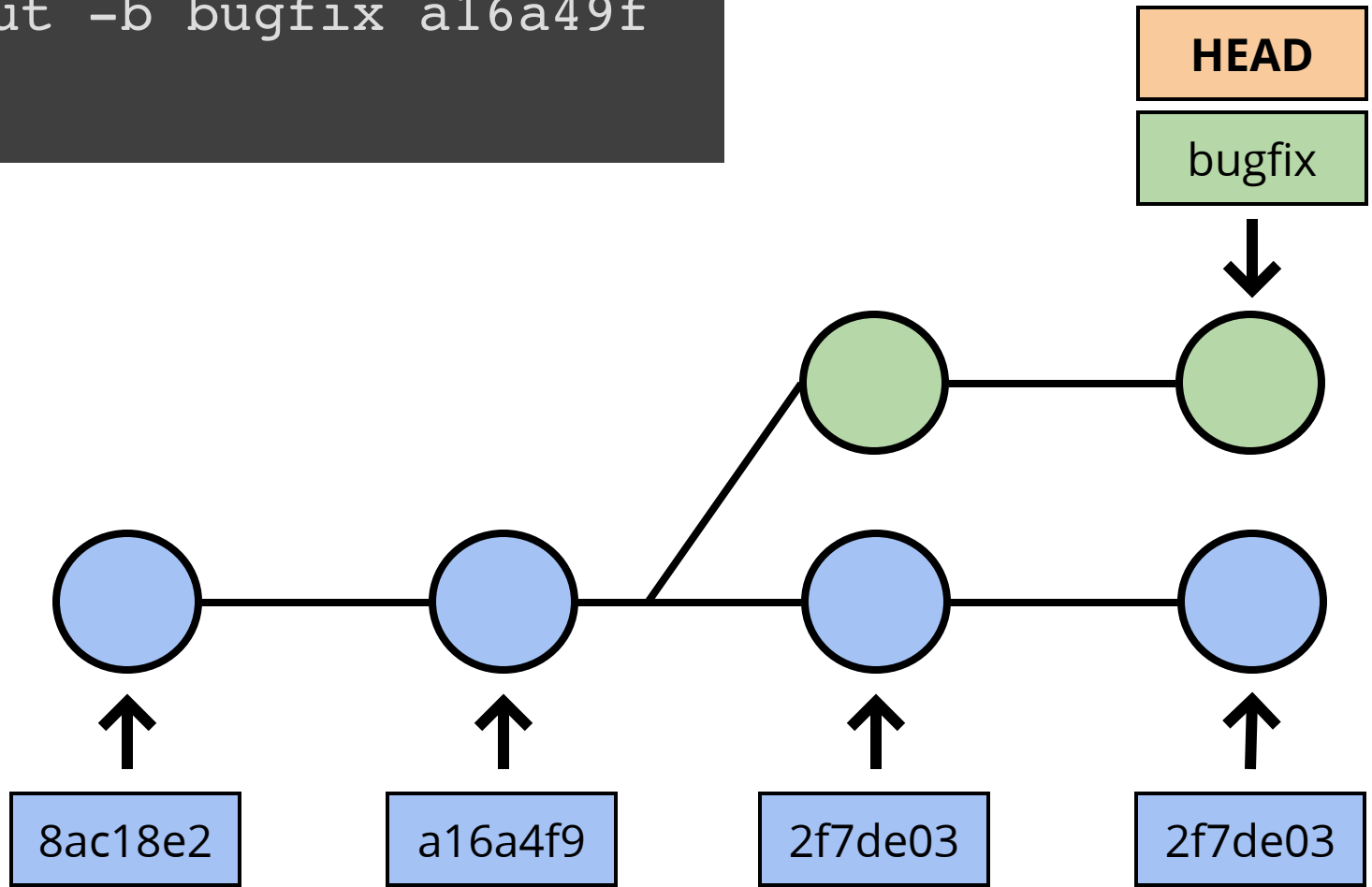
```
$ git log --oneline
2f7de03 Fourth
8417290 Third
a16a4f9 Second
8ac18e2 First
```



Undoing with Checkout

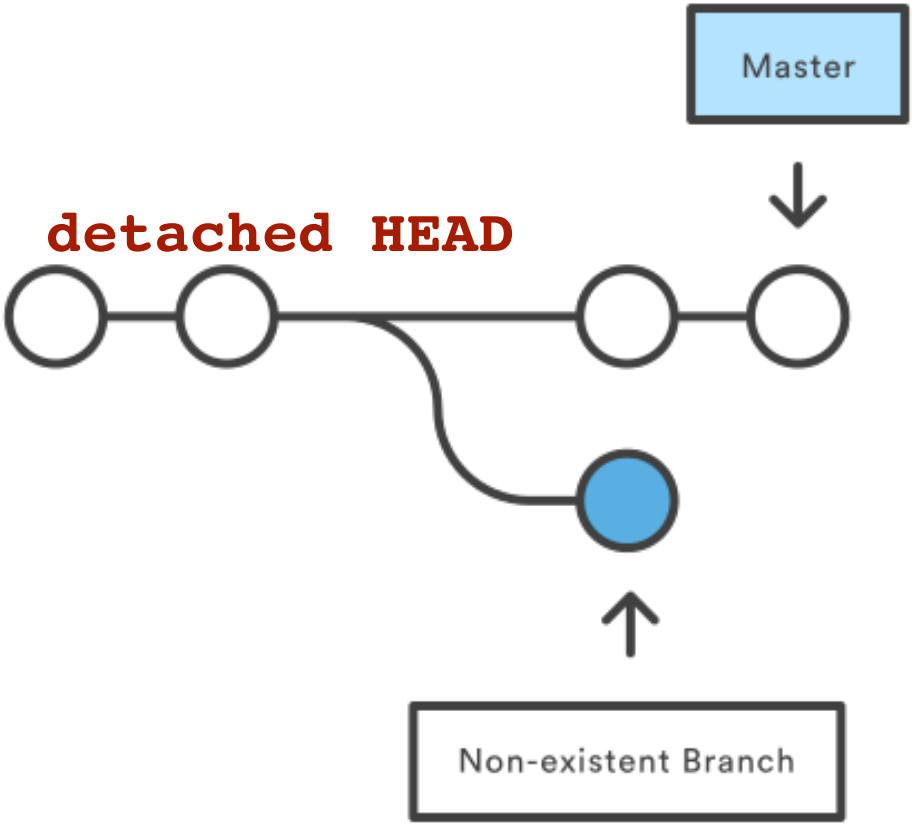
We can use `checkout` to switch not only to the commit named by a branch, but to *any* commit in order to "undo" our work.

```
git checkout -b bugfix a16a49f
git commit
```



Detached Head

If you don't create a **new** branch when checking out an old commit, you'll enter **detached HEAD** state. You can't commit from here, because there is no branch for that commit to be attached to! `checkout master` to go back.



Undoing Things

```
git checkout [commit] [file]
```

Replace file with previous version

```
git checkout -b [branch_name] [commit]
```

Go back to previous commit for development

```
git revert [commit]
```

Change files to undo commit and remove the changes it made (adding a new commit, preserving history)

Branching Questions?

Action Items!

- Be comfortable with **module 14**
- Assignment 6 due ***Thursday before class***

Section: form project teams

Thursday: Collaborating with Git