# UCLA CS35L

Week 8

Wednesday

# Reminders

- Assignment 7 due this Friday (5/22)
- Assignment 8 due next Friday (5/29)

- Week 10 Assignment, first presenters are now next Wednesday
- Reach out to me if:
  - You need to send in a recording due to timezone issues making it hard to present live
  - Your partner has not responded to you about preparing for the presentation/report

- Anonymous feedback for Daniel
  - https://forms.gle/tZwuMbALe825DBVn8

# git diff

```
git diff [--staged]
```

- Show in diff file format, all current changes
- By default, only shows changes in the working directory. To show changes in the staged area, use the optional **--staged** option

- UPDATE – A student helped correct me on the default behavior:
  - CORRECT Answer – git diff shows the diff between the working directory and the staging area
  - What I mistakenly said – git diff shows the diff between working directory and the last  commit

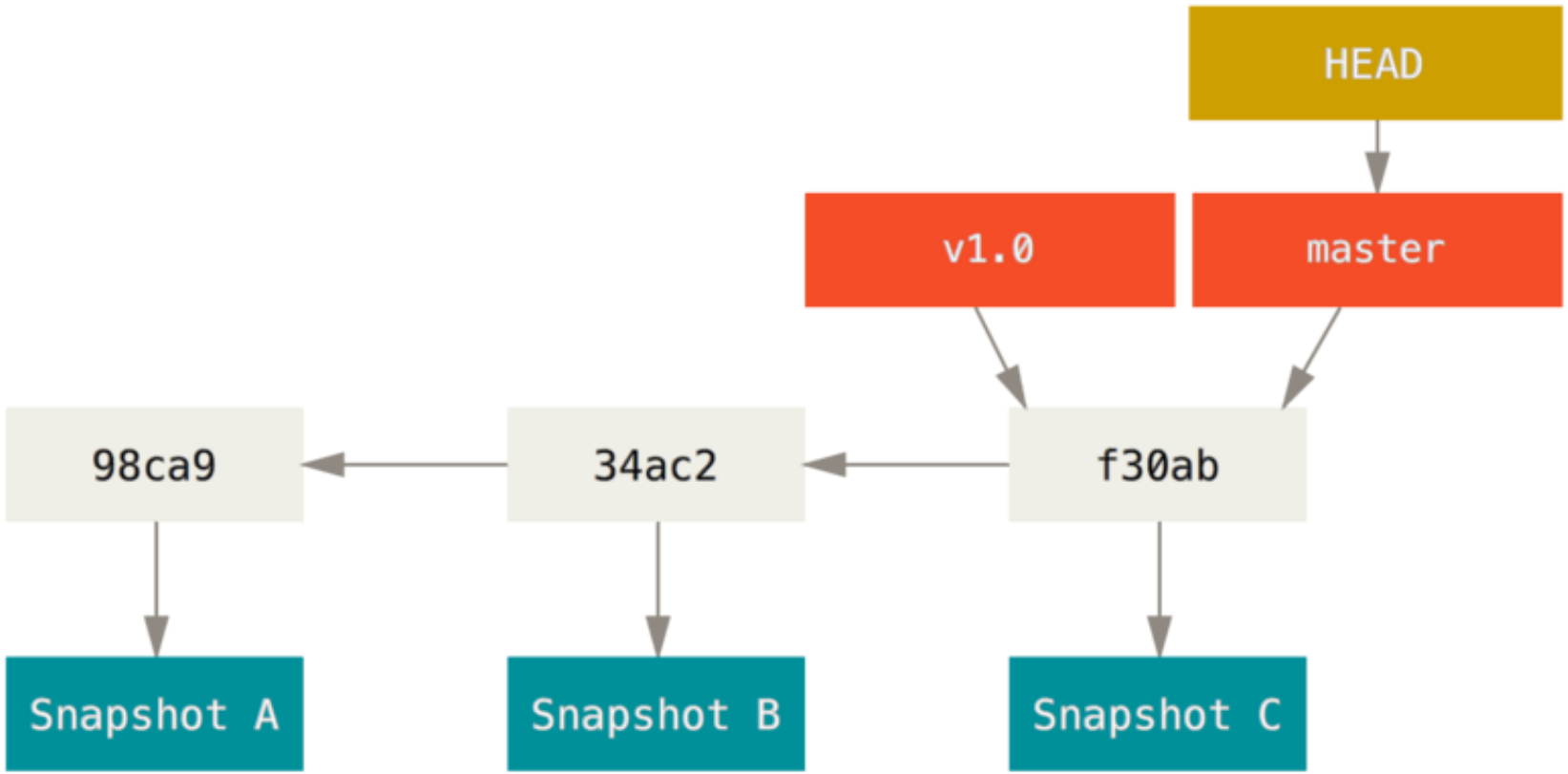
# Branches and Merging

# Branches

- So far we have only been working on one main trunk
- But the big advantage of version control is to work on separate branches (that don’t affect the original) and are merged in when ready.
- Examples
  - There is a “master” main branch of all working code
  - I make a branch to write the sorting feature and add it in
  - Someone else makes a branch to fix a bug with how master reads input

# HEAD Pointer and Branches

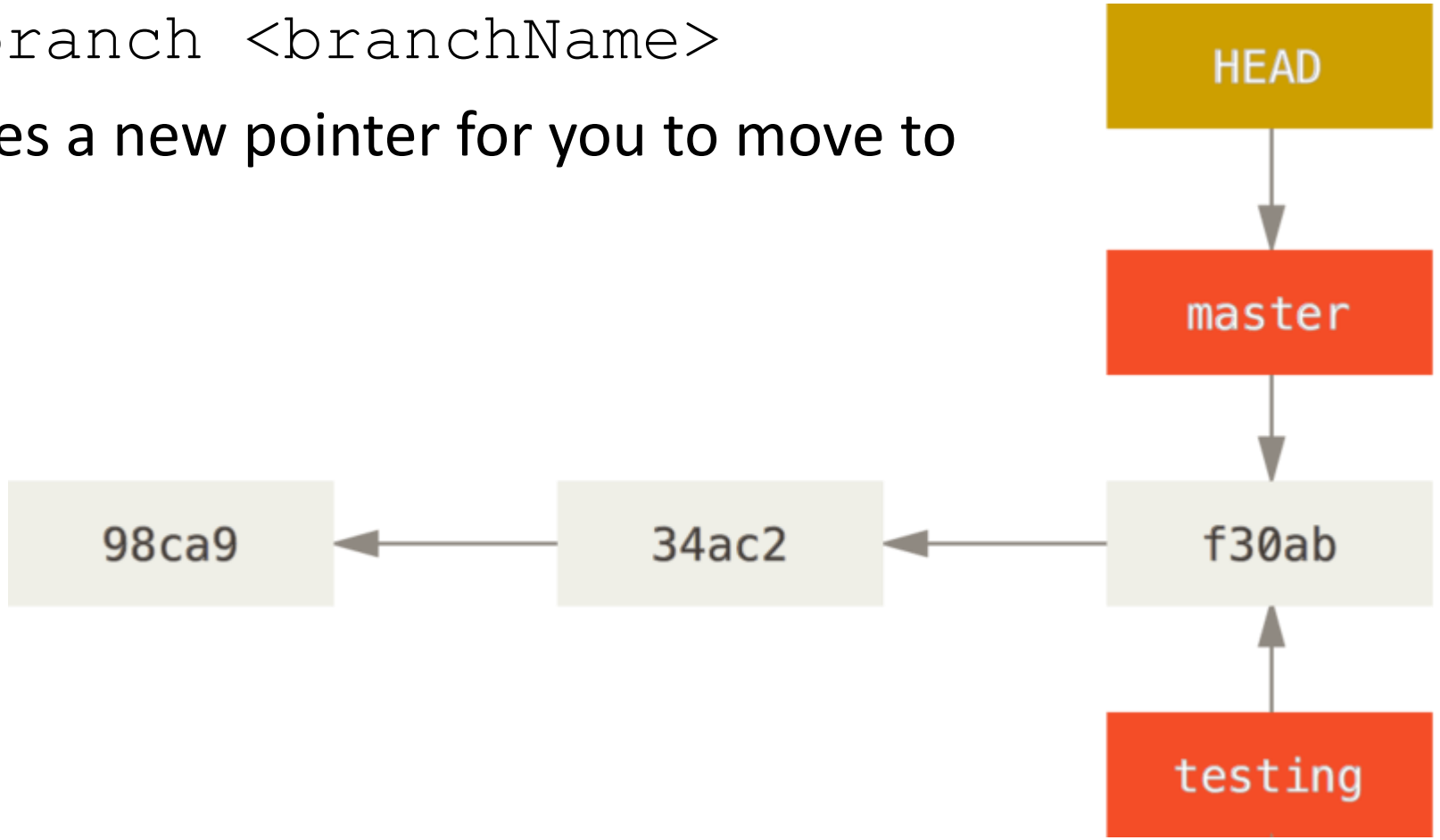
- The HEAD Pointer points to the current commit you are on.
- Each branch contains a pointer to the current commit that it is on.
- master is the first branch usually created, so HEAD starts by pointing at the latest commit in master


HEAD
v1.0
master
98ca9
34ac2
f30ab
Snapshot A
Snapshot B
Snapshot C

# Create a branch

`git branch <branchName>`

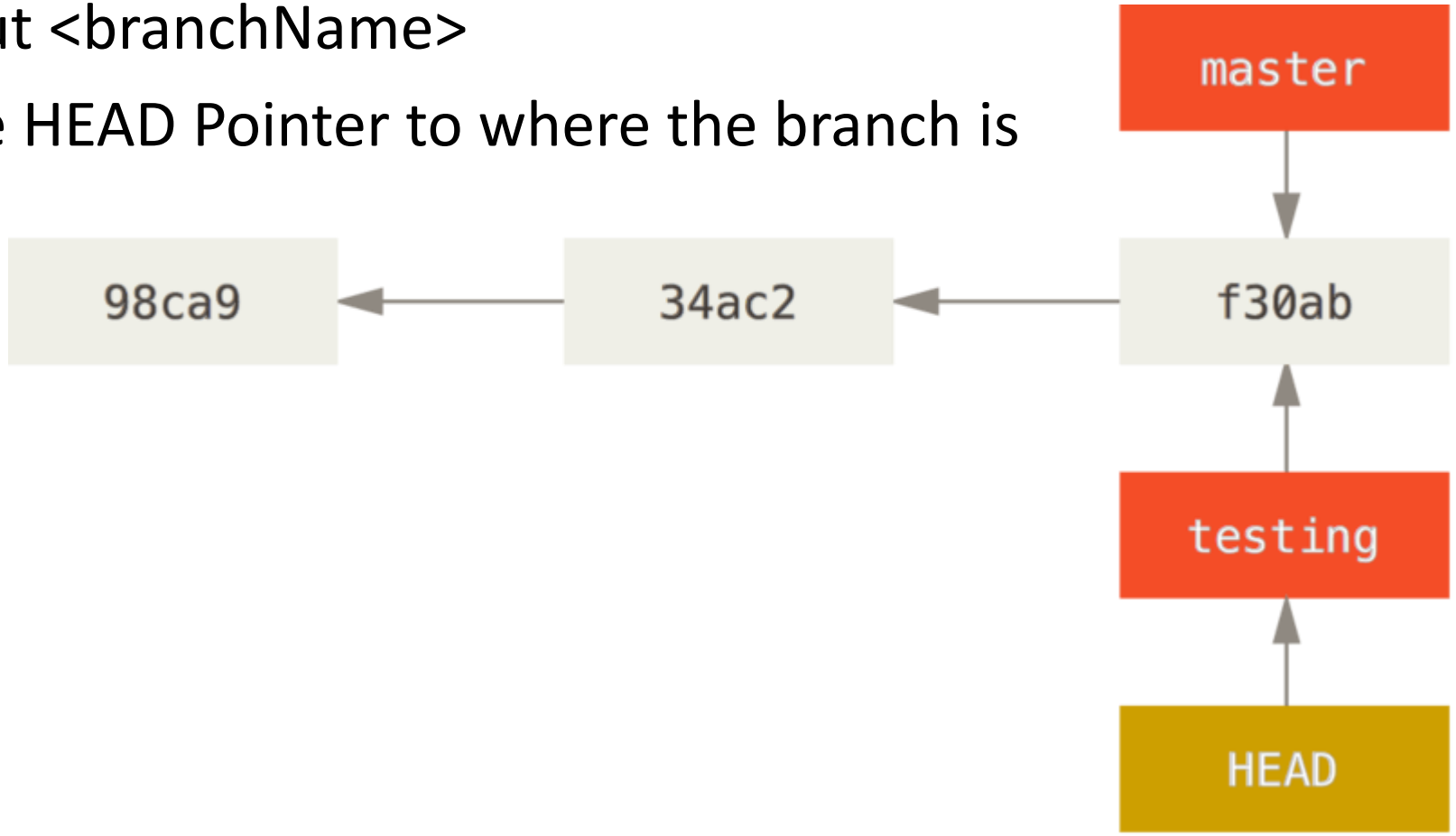
- Creates a new pointer for you to move to



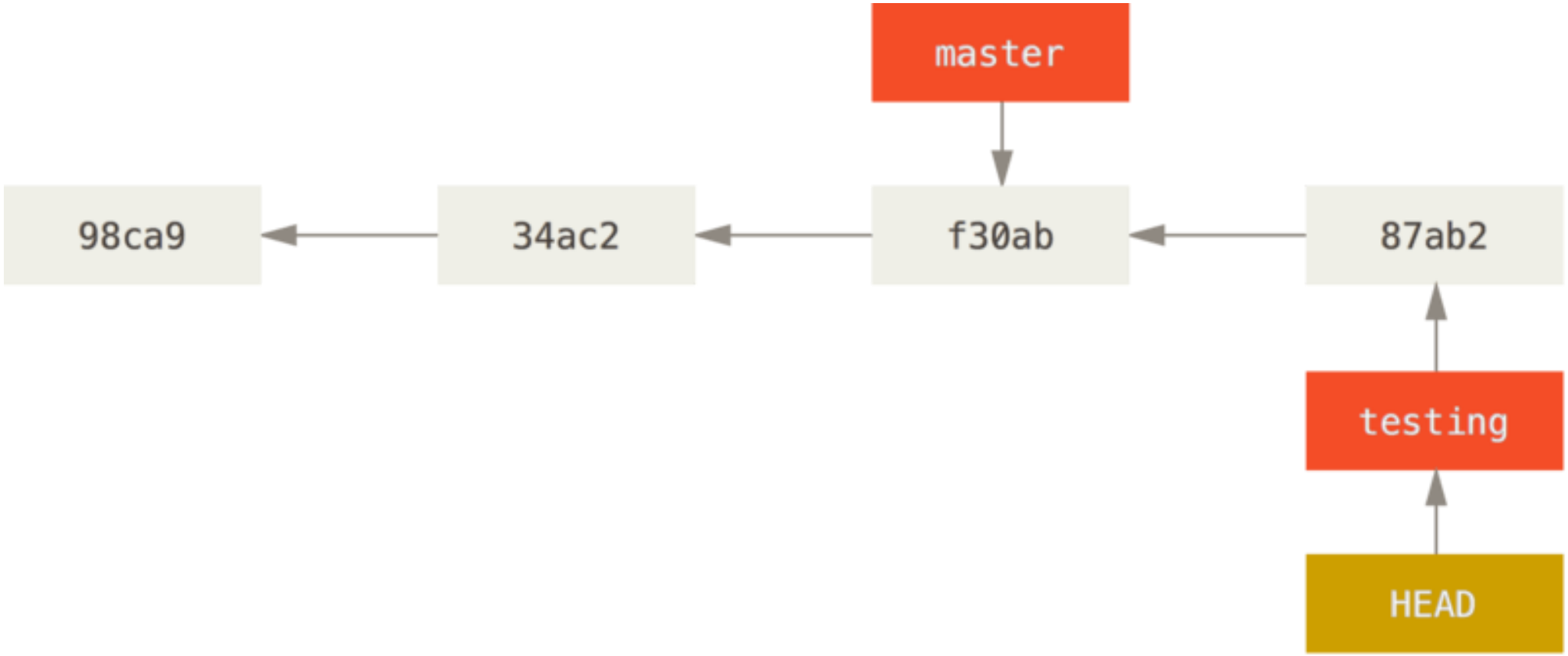
# git checkout – switching branches

git checkout <branchName>

- Move the HEAD Pointer to where the branch is



# Create commits on Branch



# git checkout - Variations

- `git checkout –b <branchName>`
  - Will create the branch if it does not exist
- `git checkout –b <branchName> <remote>/<branch>`
  - Ex. git checkout -b testing origin/testing
  - Will create a local branch, copied from and tracking the remote branch specified

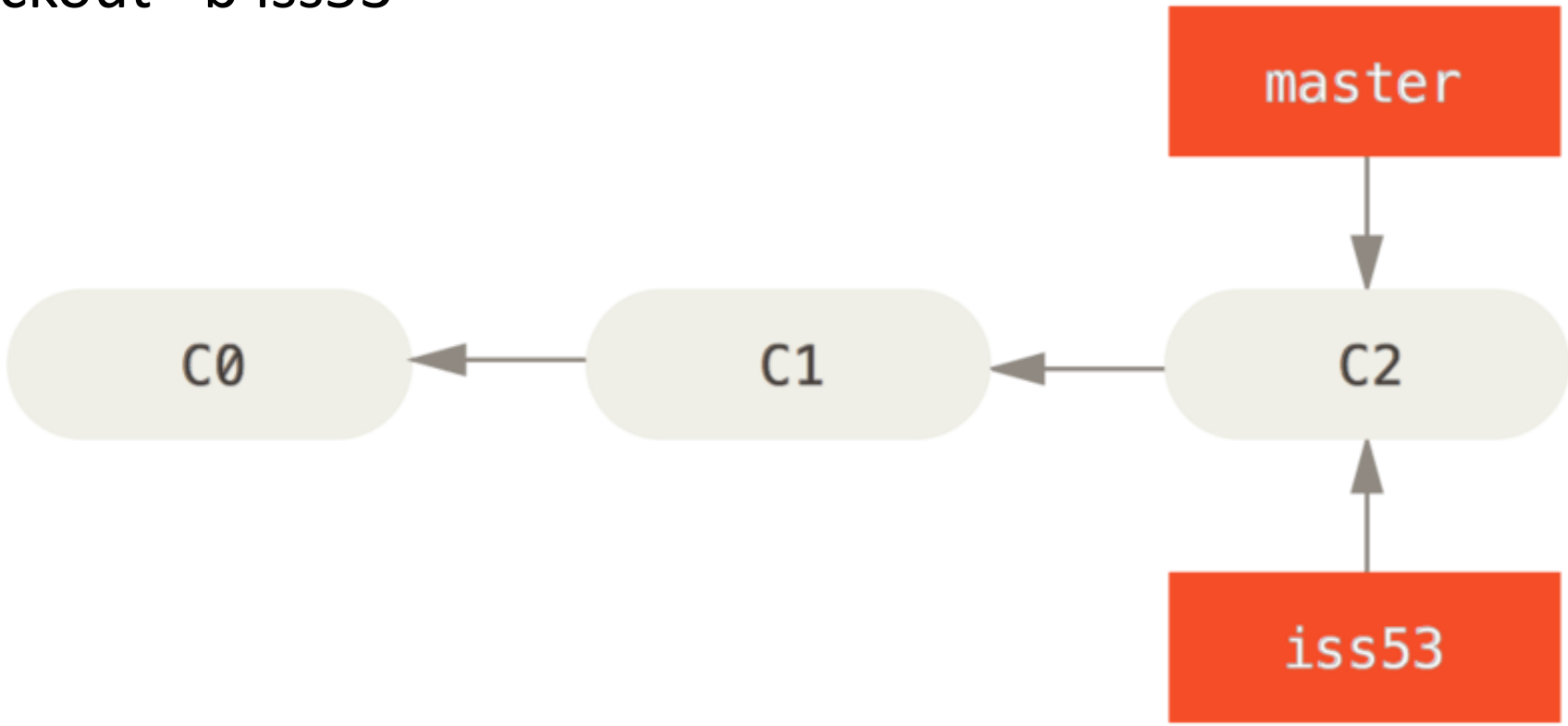
# Merging

`git merge <branchName>`

- At some point, you are done with your branch and want to merge it into the main line of code.
- Command above will merge the named branch into the current branch you are on.
- Two main types of merges
  - Fast-Forward Merge
  - Three-way Merge
- Will look at examples in next slide

# 1. Create a branch

- git checkout –b iss53

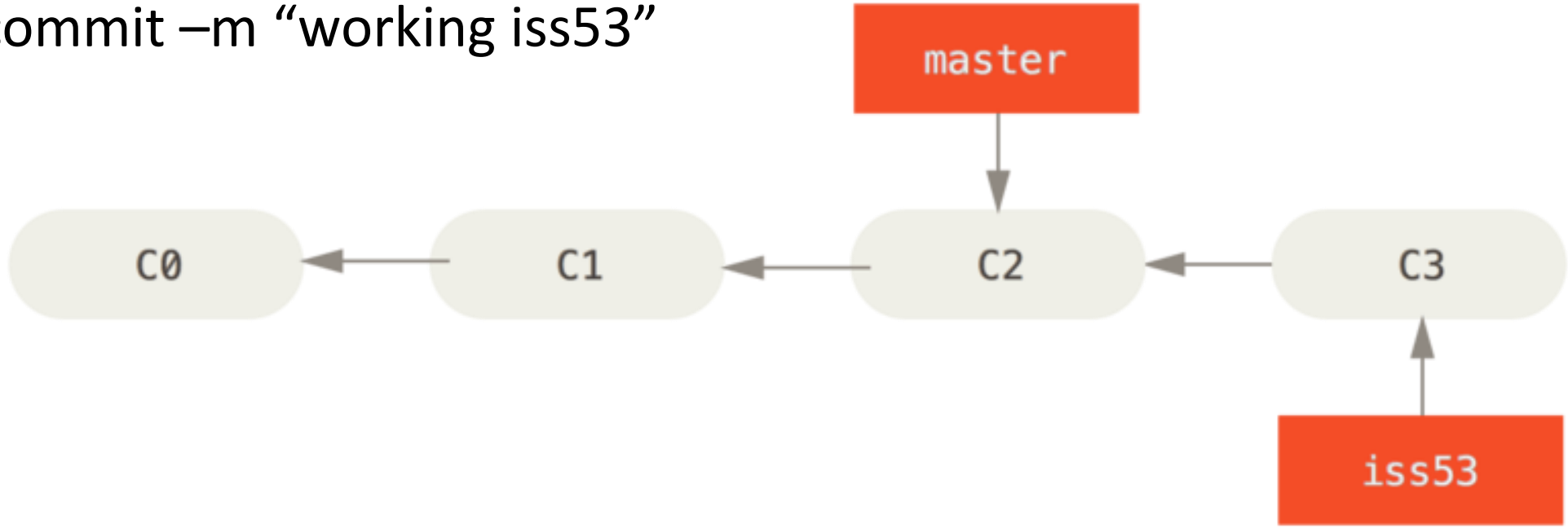


# 2. Add a commit on that branch

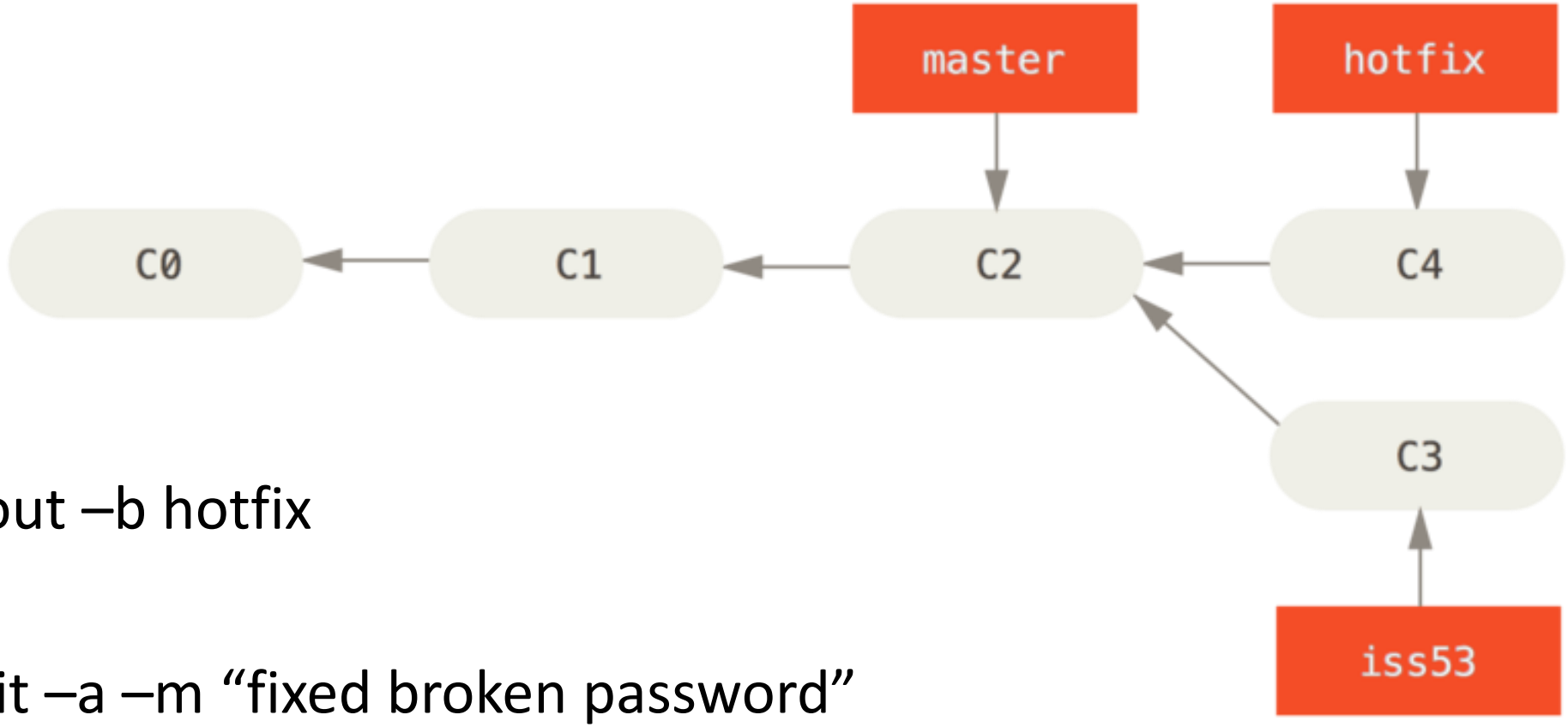
//change a file

git add file1

git commit –m “working iss53”



# 3. Need a new urgent branch for hotfix




git checkout –b hotfix

//fix file

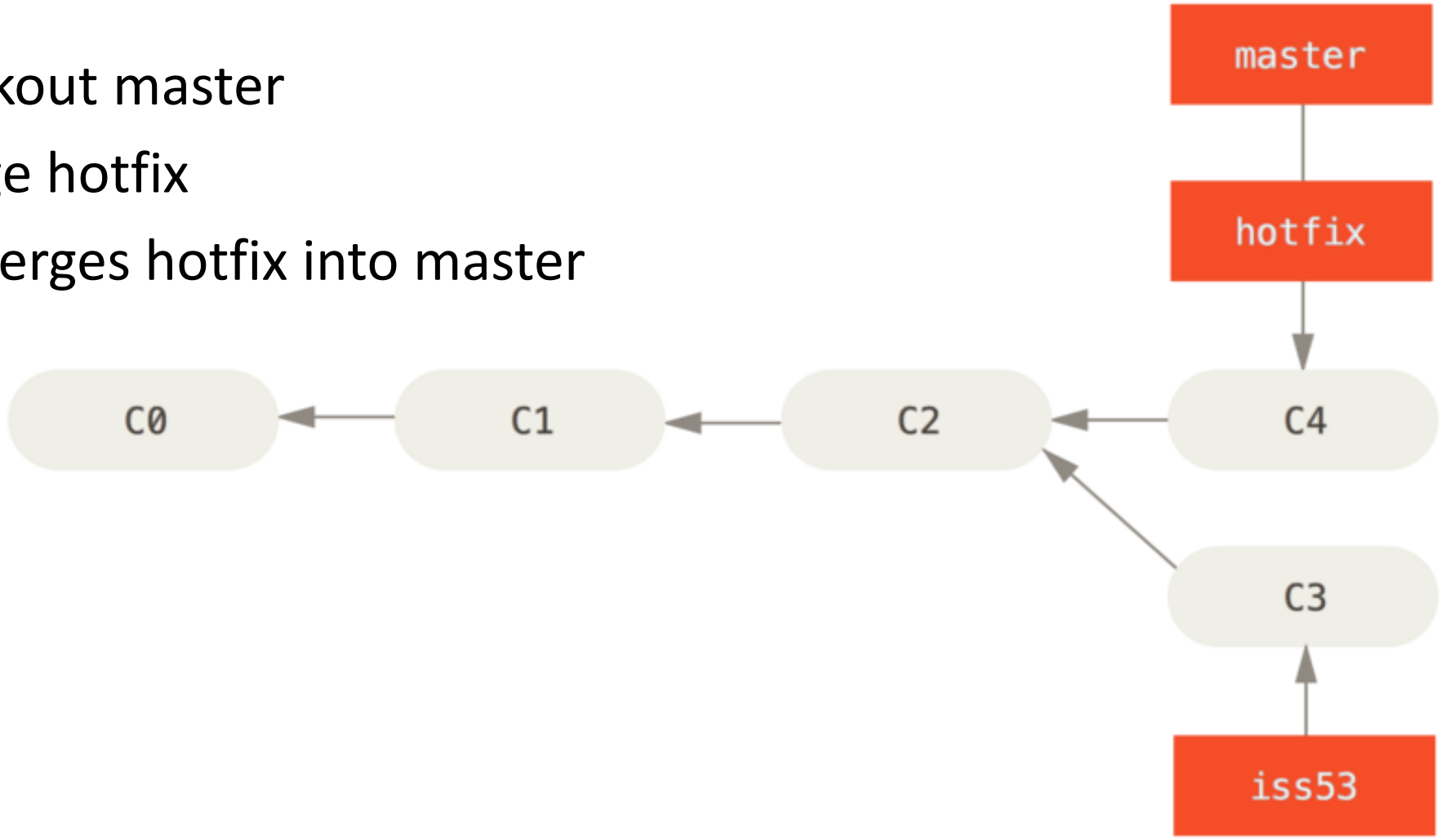
git commit –a –m “fixed broken password”

# 4. Complete Fast-Forward Merge

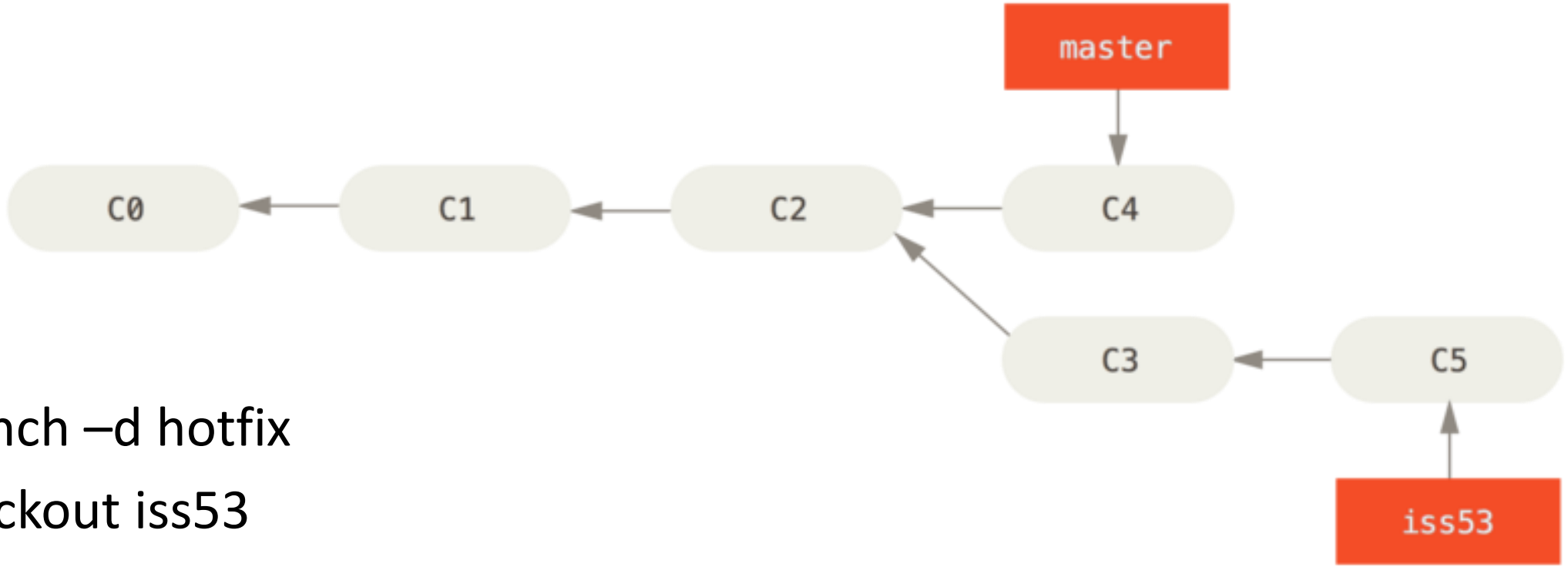
git checkout master

git merge hotfix

//this merges hotfix into master



# 5. Go back to working on iss53




git branch –d hotfix

git checkout iss53

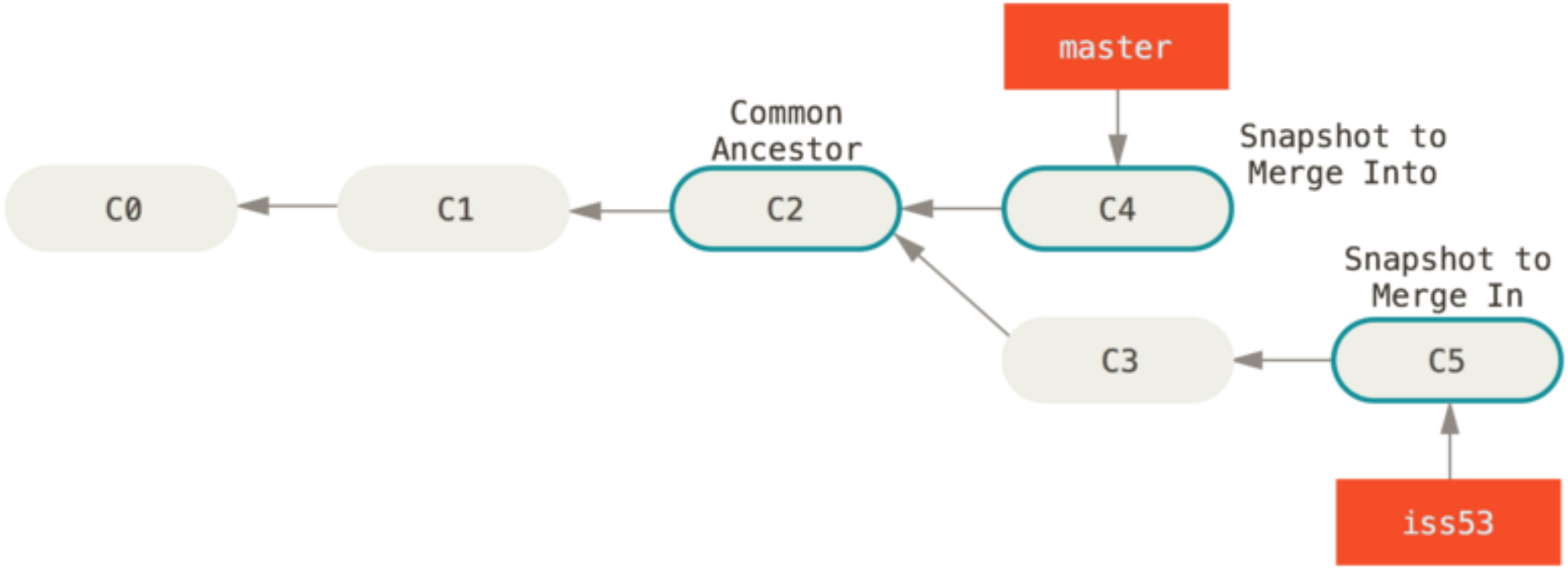
//make some changes

git commit –a –m “changed naming”

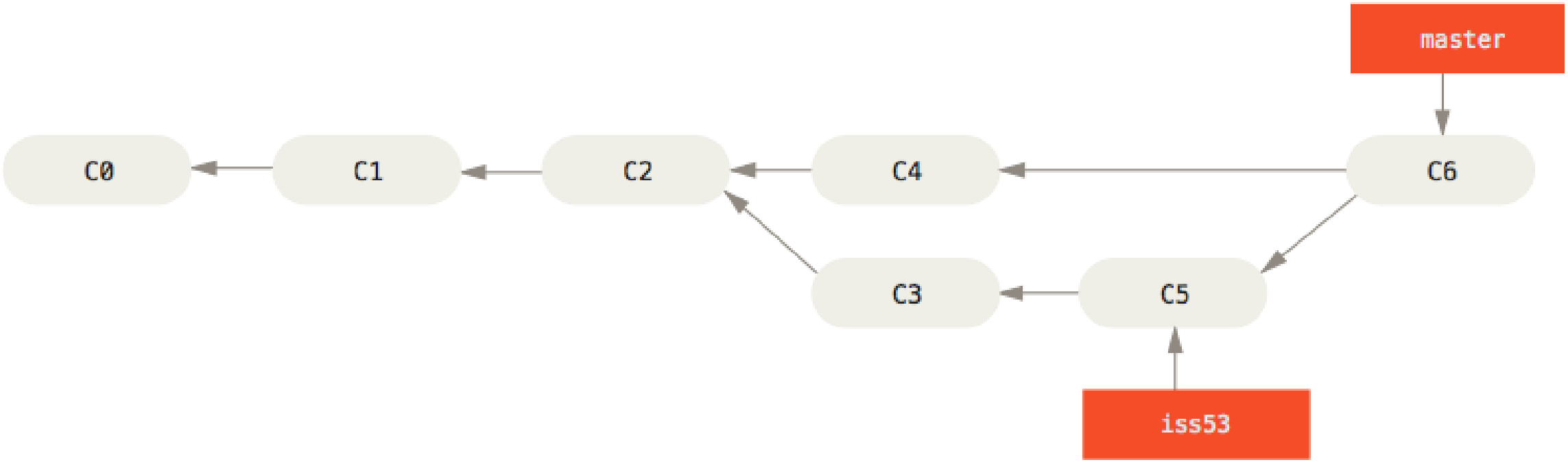
# 6. Merge iss53 into Master

git checkout master

git merge iss53



# 7. All merges completed



# Git merge conflicts

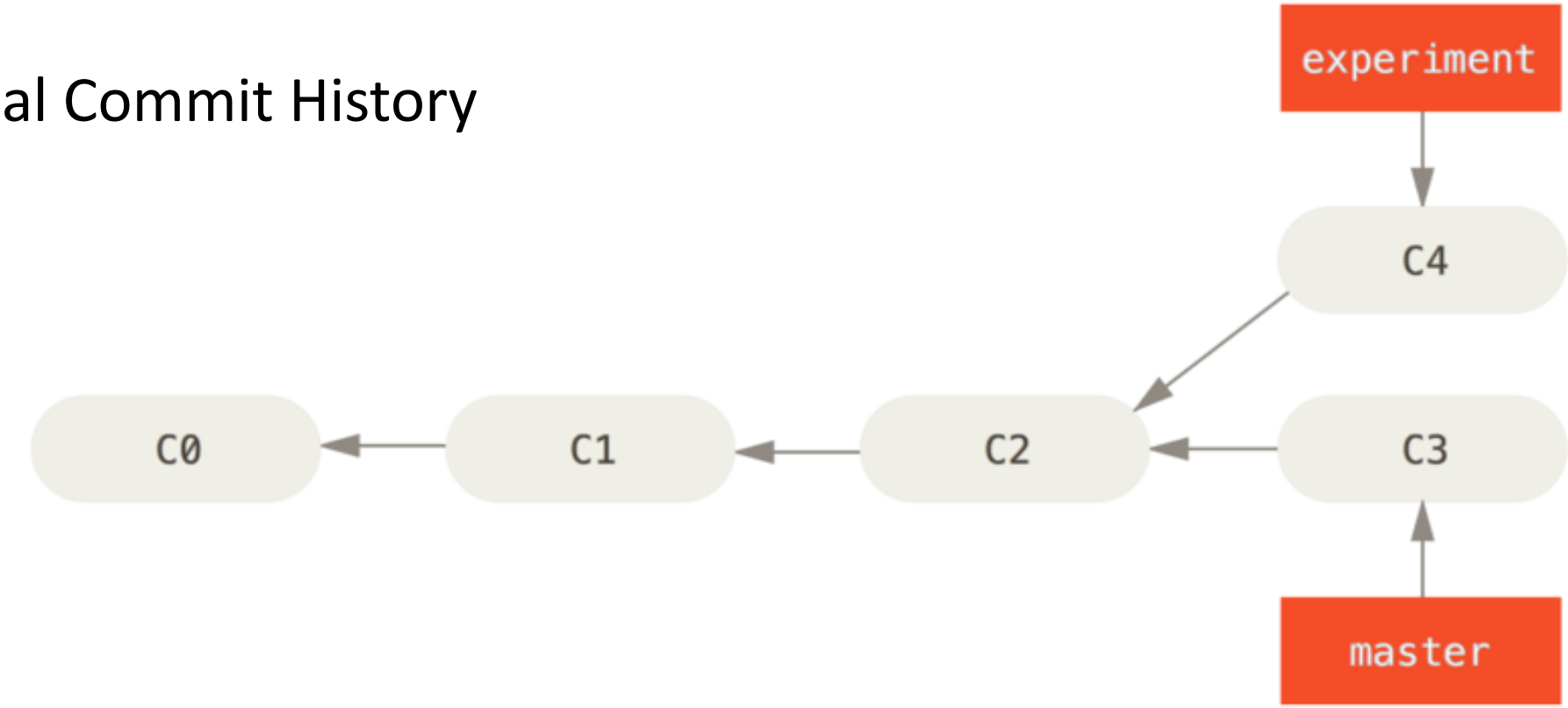
- Not every merge goes perfectly
- If hotfix and iss53 both modified the same part of the file, what is the final part we should keep? Git doesn’t know….
- In case of conflict, the merge will stop and tell you to resolve
  - Either go change the files manually
  - Use a visual tool like `git mergetool`

# Merge vs Rebase

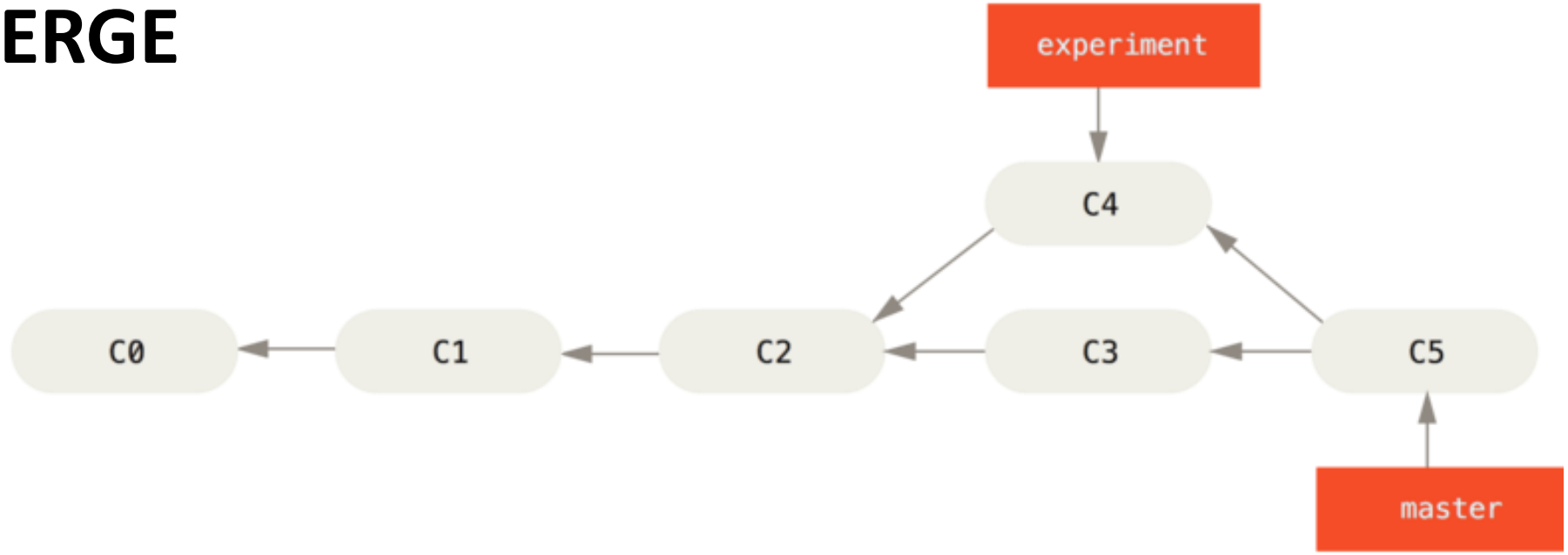
- Both designed to integrate two branches together
  - Merge takes the contents from the branch and tries to integrate into master
  - Rebase makes a new commit after master, and moves the branch to that
- Rebase will make for a “cleaner” more linear commit history, but does so by rewriting the history.
- Rebase addresses conflicts one at a time instead of all at once like merge
- General Tips
  - Rebase on local non-published work
  - Merge on shared, published code. So that way you don’t alter the commit history that other people may rely on.

# Merge vs Rebase Example

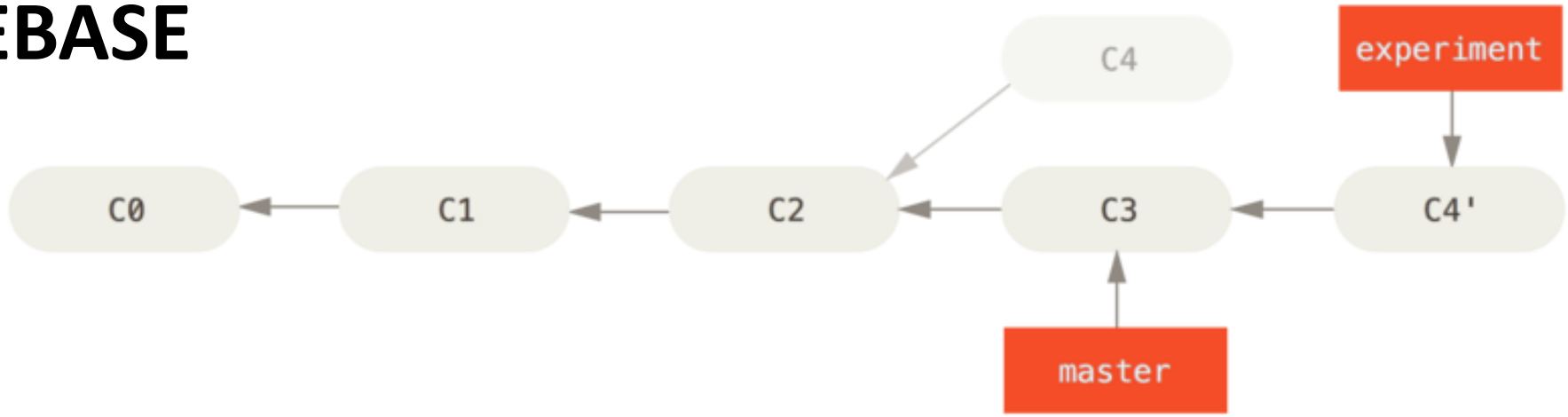
Original Commit History



# MERGE




# REBASE



# Try at home - Git Workflow with Branches

- Create git repo on Github and clone to Server/computer
- To make a change, first create a branch and switch to it
  - git checkout –b <branchName>
- Work on feature incrementally and use Add/Commit
- When feature is ready, switch back to master and merge in
  - git checkout master
  - git merge <branchName>

# Helpful Git Resources

- Git E-Book
  - https://git-scm.com/book/en/v2
- Many Youtube crash course videos for the basics, I like this one
  - https://www.youtube.com/watch?v=SWYqp7iY_Tc&t=1689s

# Other Git Info

# gitignore

- .gitignore file at the top of your working directory
- Specify any files that you want git to ignore by default
  - But you can forcibly add them if you want them.
- Example

```
meirovit$ cat .gitignore
#ignore .c files
*.c
```

# git restore

- Restores files to their last commit state

# git format-patch

git format-patch [numCommits] [CommitID] --stdout

- Used to generate a patch file relevant to that specific commit
- Examples
  - git format-patch -1 <someCommitID> --stdout > patchFile
  - git format-patch -1 --stdout > patchFile

# Applying a patch generated by git

git am < patchFile

- You can use git to apply patches generated by ‘git format-patch’

# Emacs Integration

Homework has a few different ways to use Emacs

- vc-revert (C-x v u)
  - Used to undo changes.
  - For lab 7 – think about which files were patched that you want to restore back
- Reverting selective hunks
  - Open version history (C-x v =)
  - Revert Hunk (C-u C-c C-a)

# Git Internals

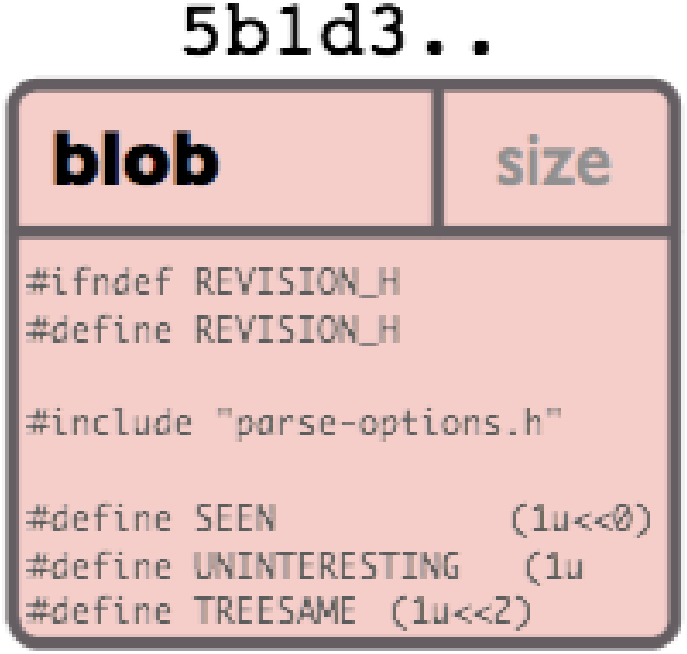
# Git Objects

- Blobs
- Trees
- Commits

# Blobs

- File contents – just a “blob” of binary data
  - If the contents are the same, then it is the same “blob”
- Can examine its contents with

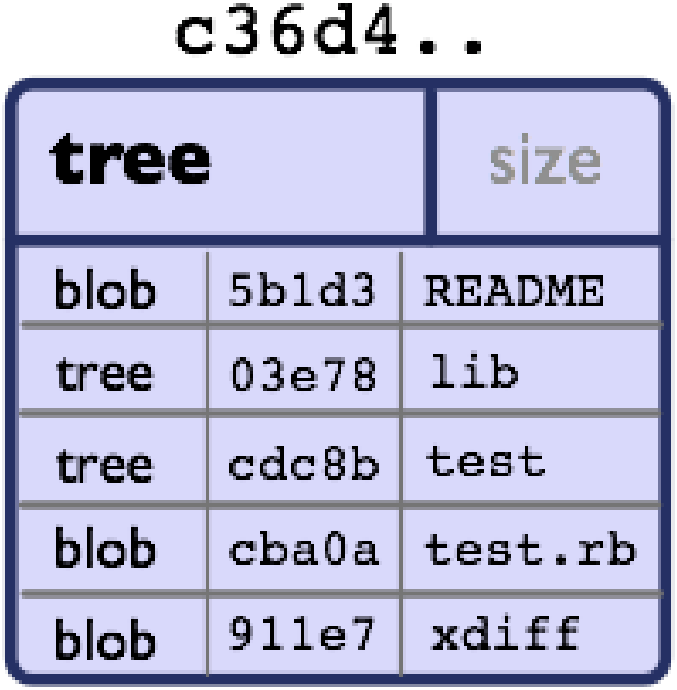
  **`git show <hash>`**



# Trees

- An object that contains pointers to other blobs and trees
- Represents the contents of a directory
- Can examine its contents with

```
git ls-tree <hash>
```



# Commit

- Links the physical state of the directory with metadata. Contains:
  - Tree
  - Parent commit
  - Author
  - Committer
  - Message

- Can examine details with either

```
git show --pretty=raw <hash>
git log
```

# Overall Picture

- Commit points to overall snapshot and top-level directory (tree)
- Tree contains pointers to the file and any sub-directory trees.
- Blob contains file contents