iOS Dev Accelerator
Week 7 Day 2

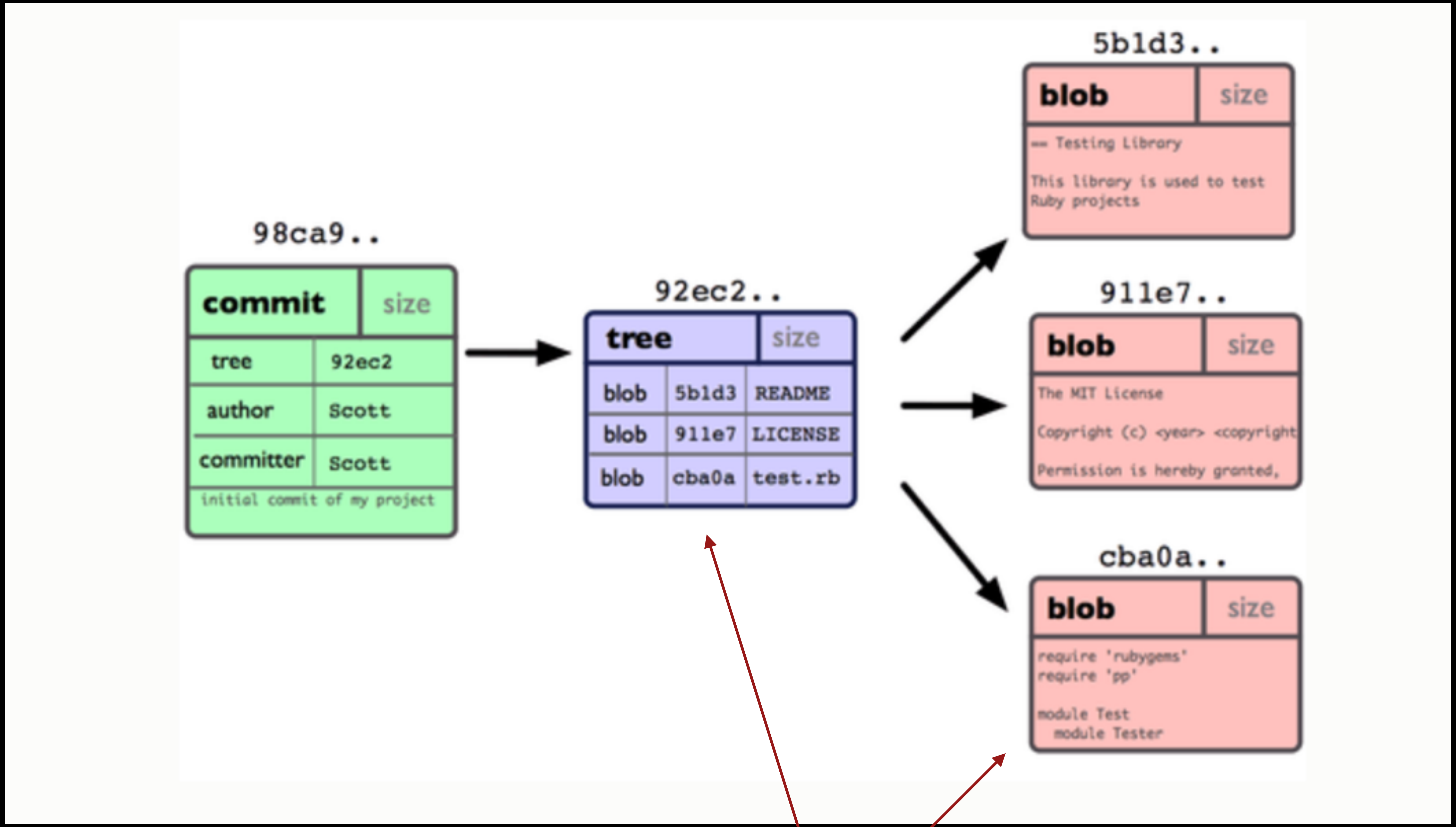
- Advanced Git
- Hash Tables
- Lesser Known Operators
- Operator Overloading

Git Branches

- To understand what a branch is, you need to further understand how git works.
- every commit is actually a commit object.
- that git object has a pointer to the snapshot of the staged files.
- it also has a pointer to its direct parent commit:
  - zero parent pointers if its the first commit
  - one parent pointer if its a regular commit
  - two parent pointers if its a result of a merge

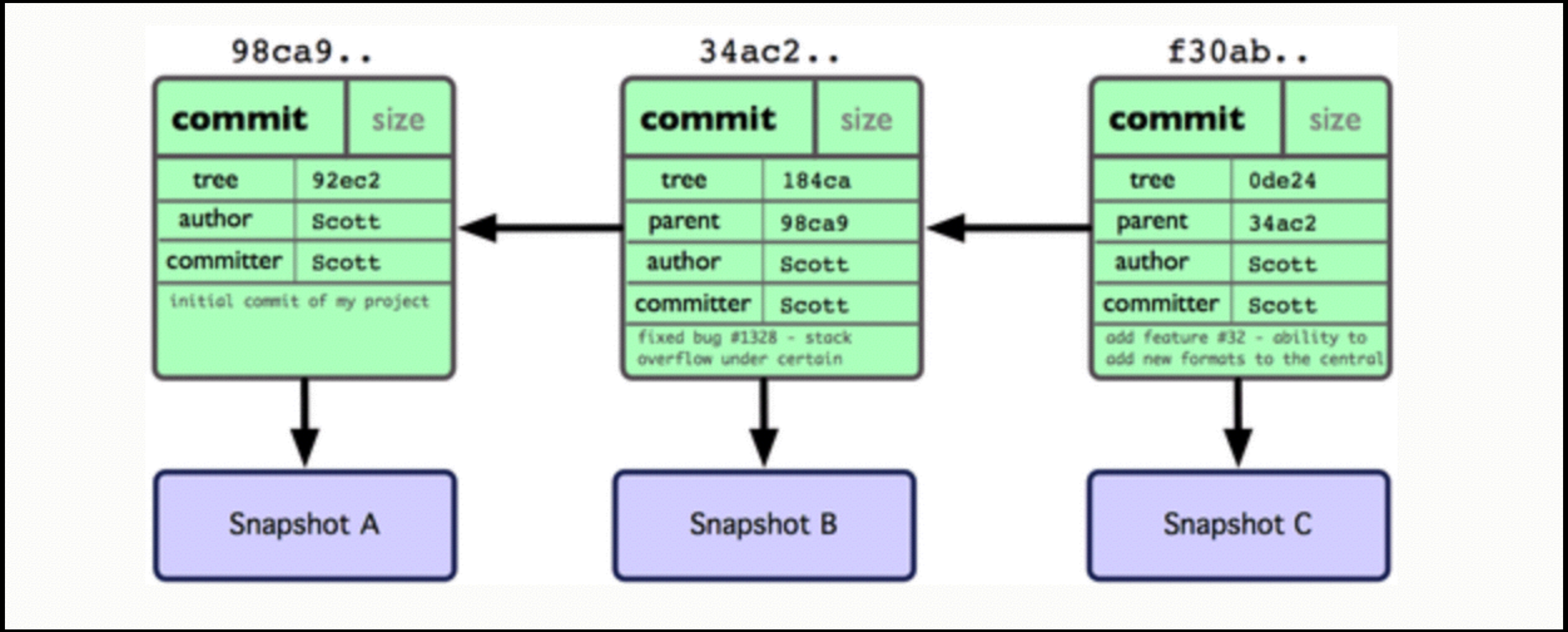
Anatomy of a Branch

- green: commit object containing metadata and pointer to tree
- purple: tree object lists the contents of the directory and specifies which files are stored as which blobs
- red: one blob file for every file in your project



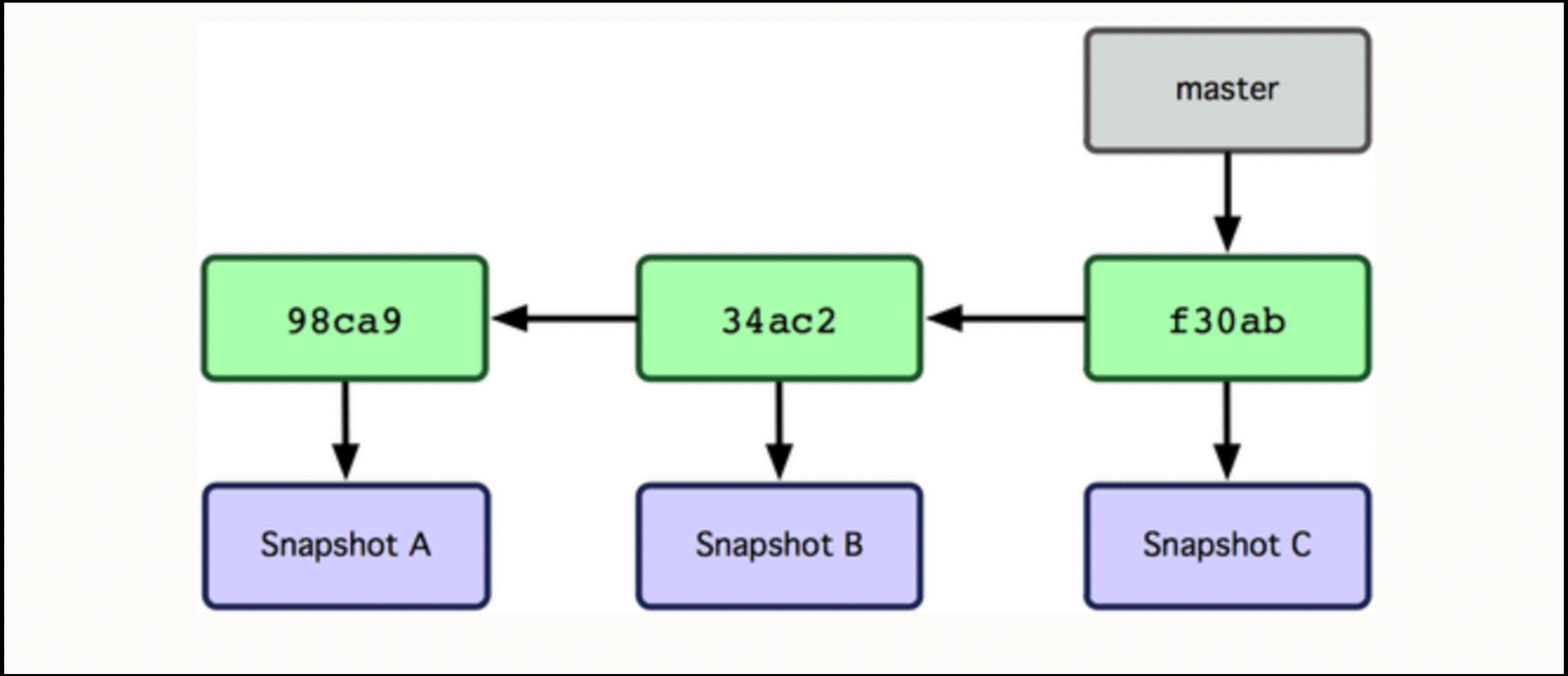
"Snapshot"

Multiple commits



first commit

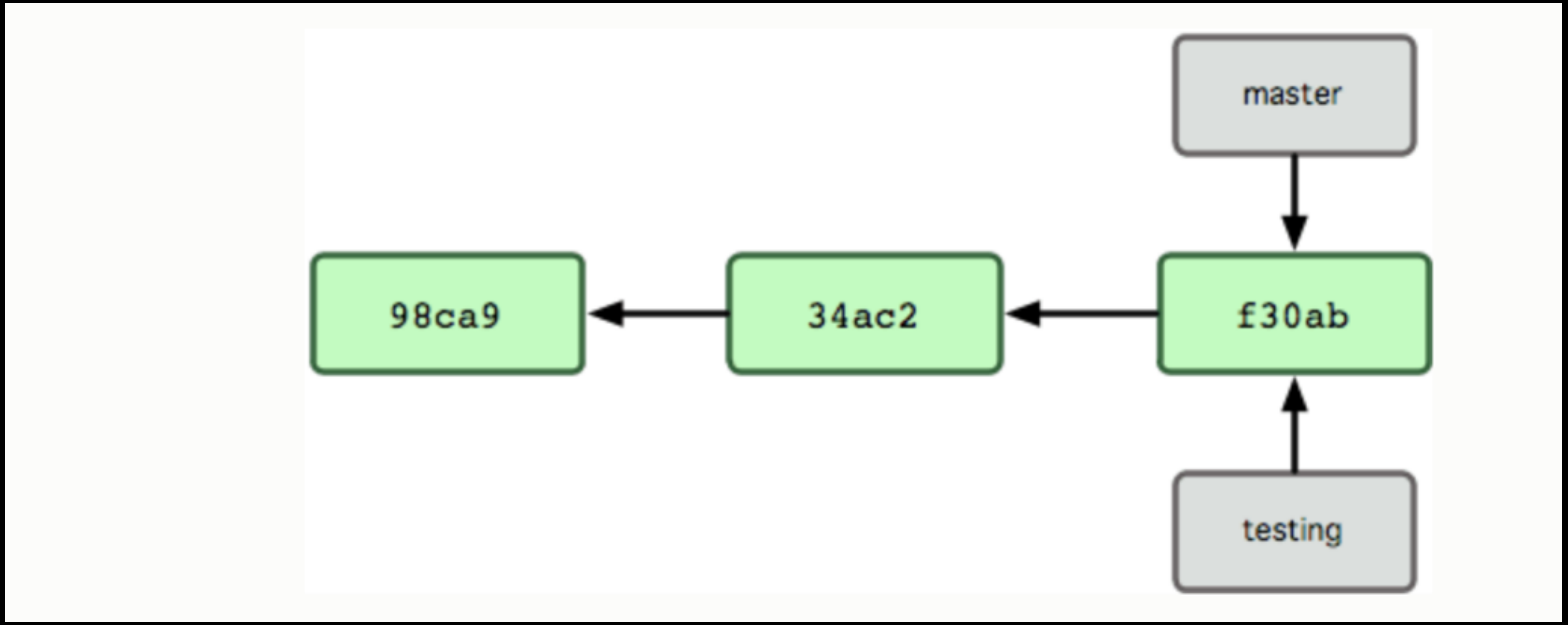
Branch == pointer



A branch is just a lightweight pointer to a commit

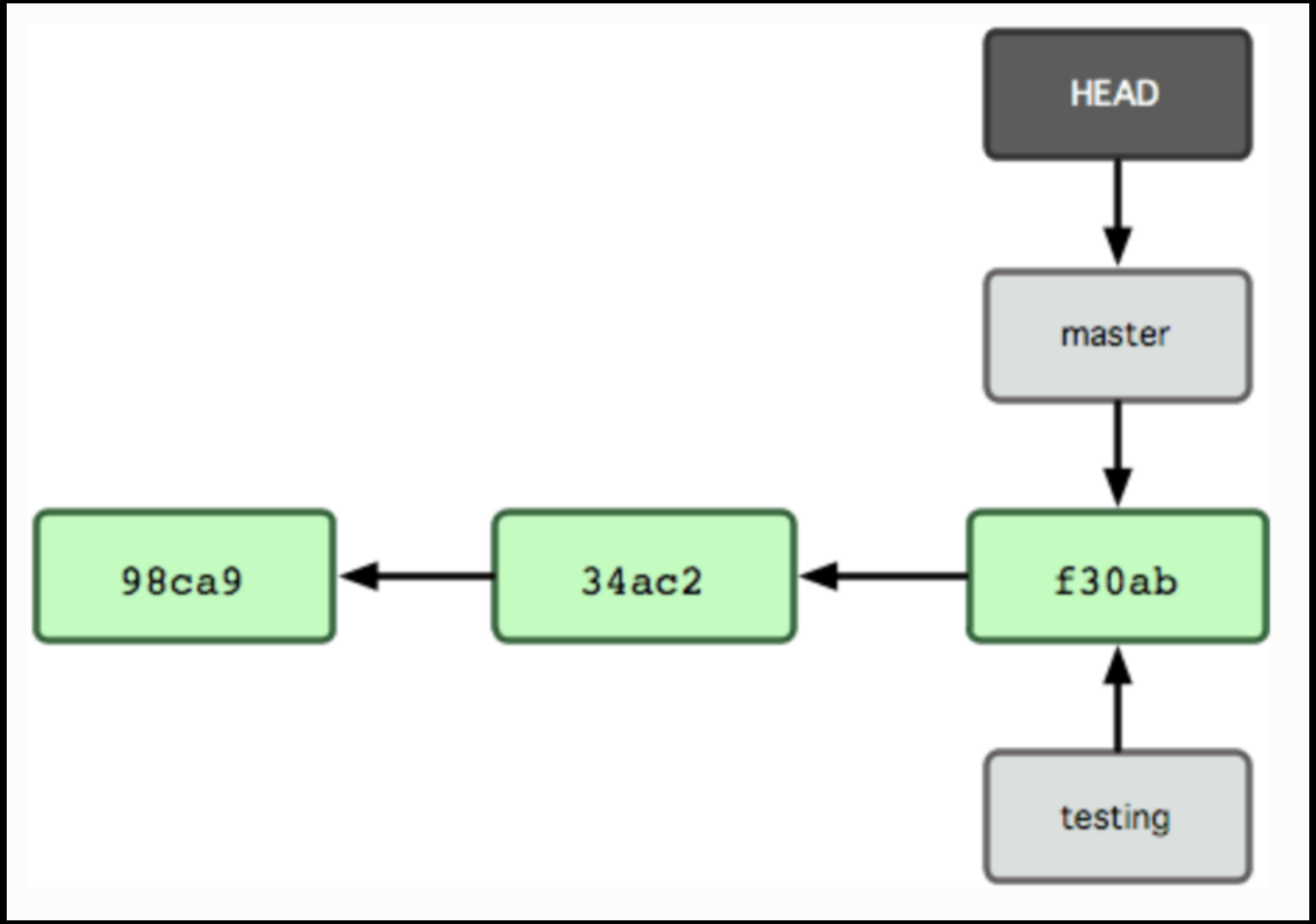
Creating a branch

- Use git branch <name> to create a new branch.
- Your new branch will point to the same commit of the branch you are currently on when you ran the that command



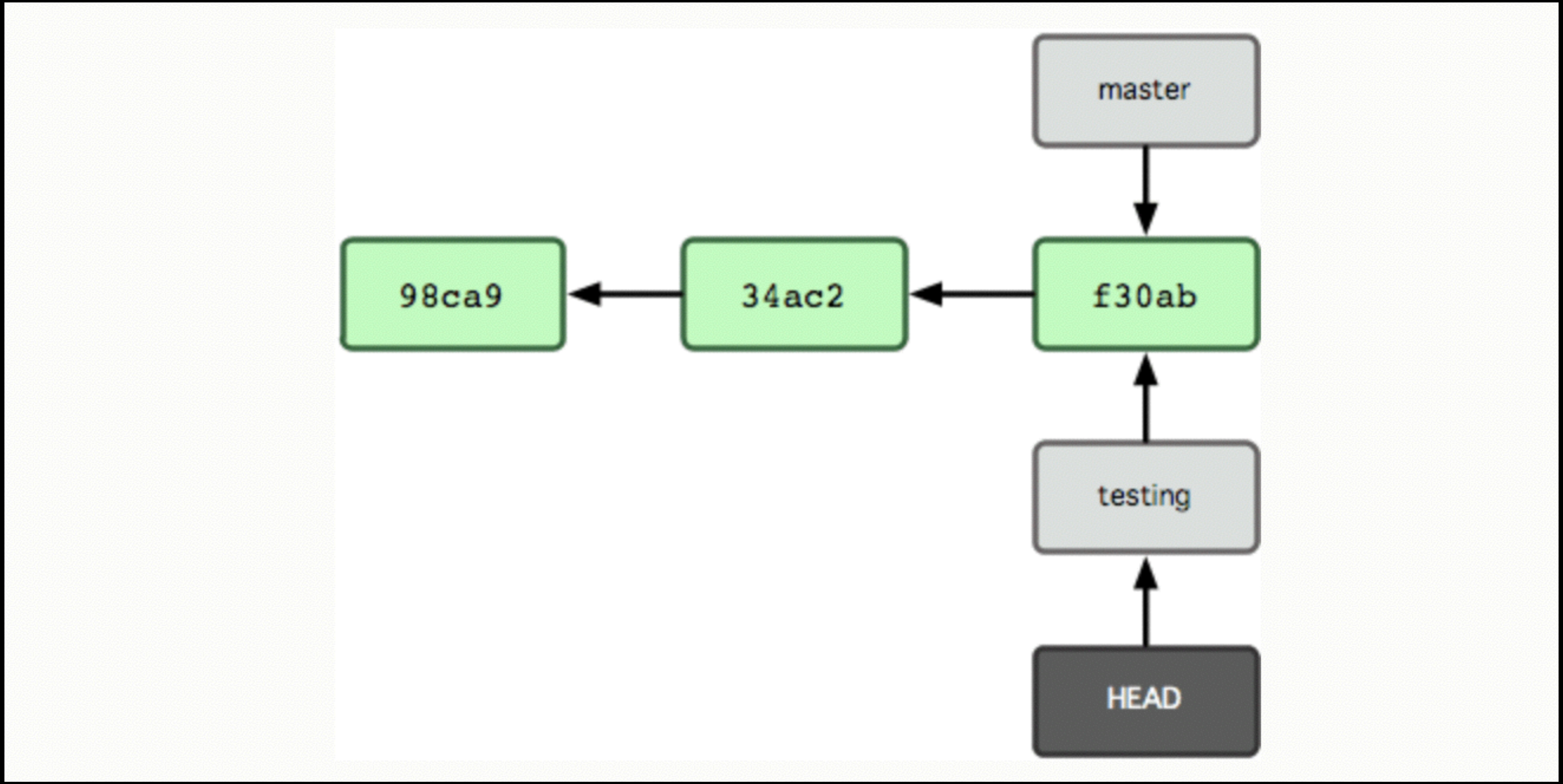
HEAD pointer

- Git keeps a special pointer called HEAD, which points to the current local branch you are on.
- The branch command does not switch you to your new branch, it just creates it. So we are still on master.



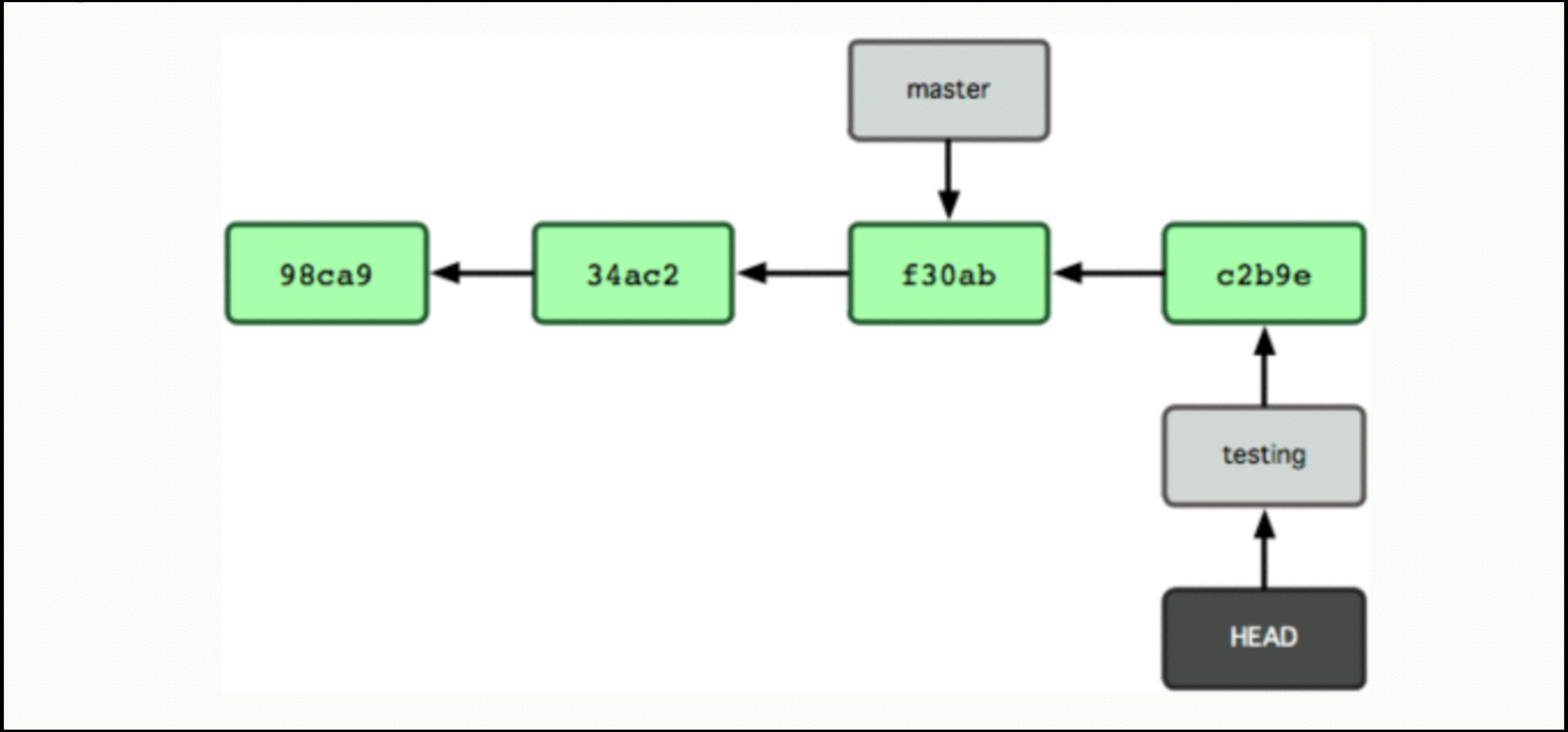
Git Checkout

- Running git checkout <branch name> switches HEAD to point to branch you specify



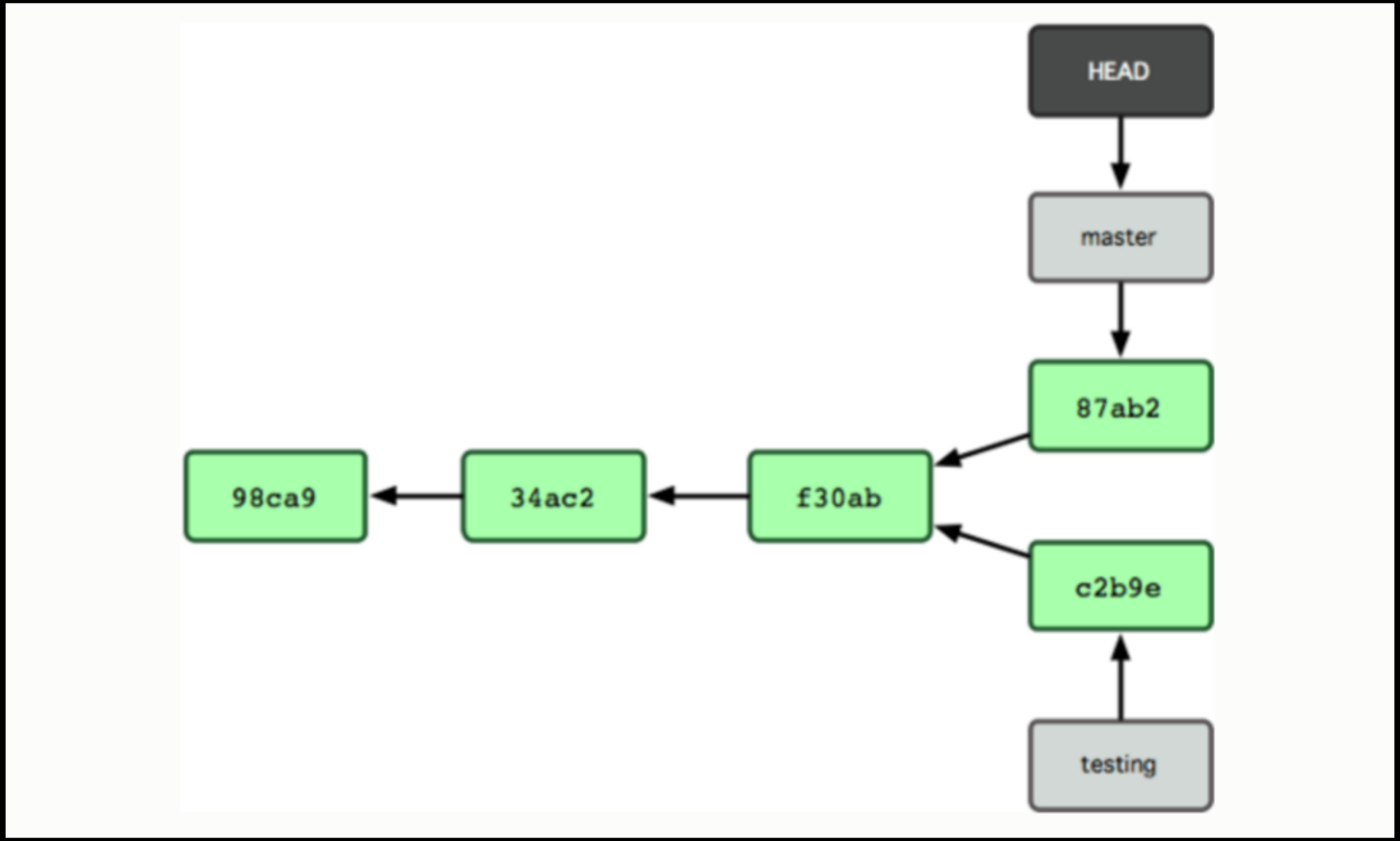
Moving forward

- If you commit now in your testing branch, testing moves forward while master stays put.



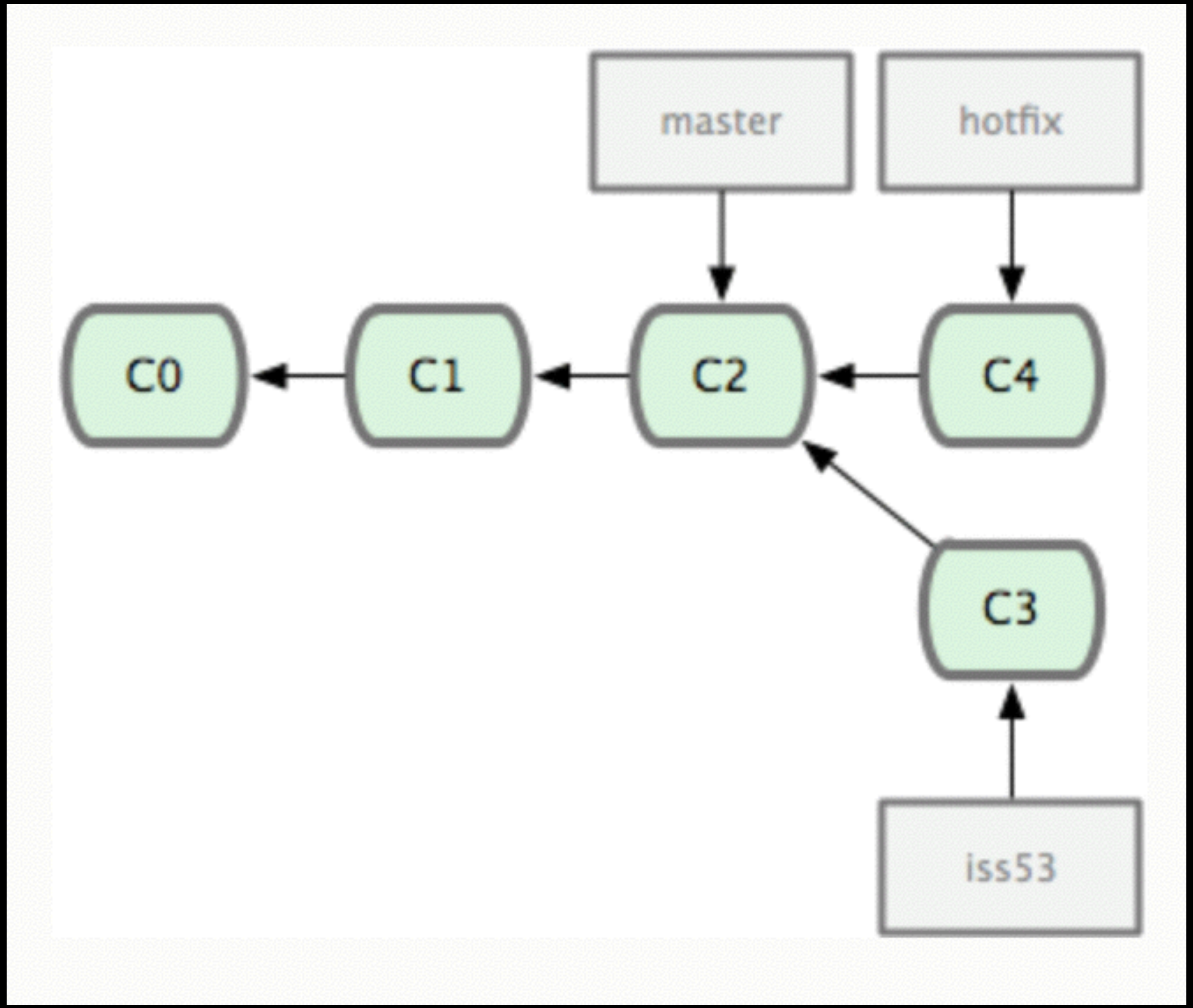
Divergence

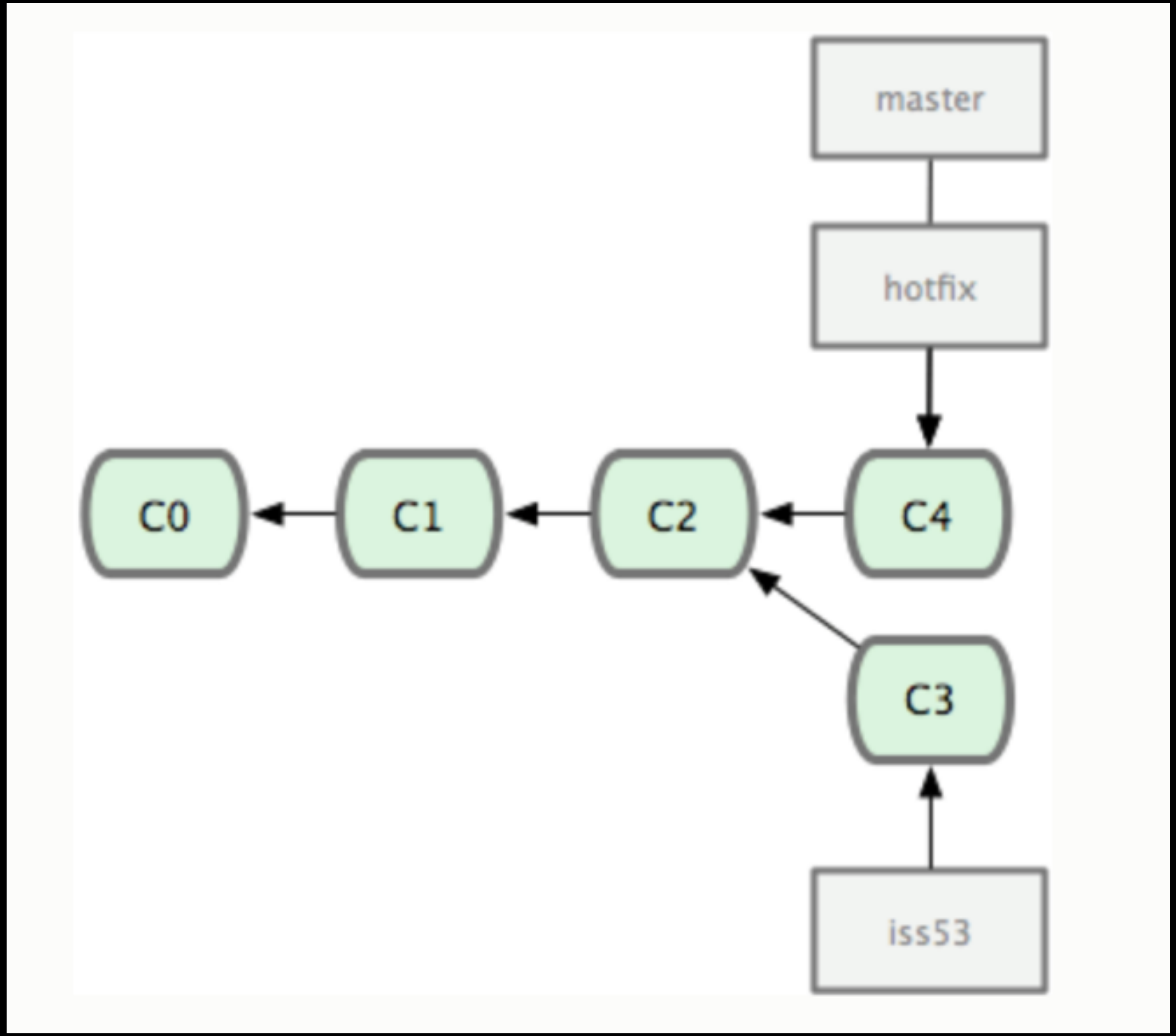
- Now if you checkout master and make commits, we have a divergence.



Merging

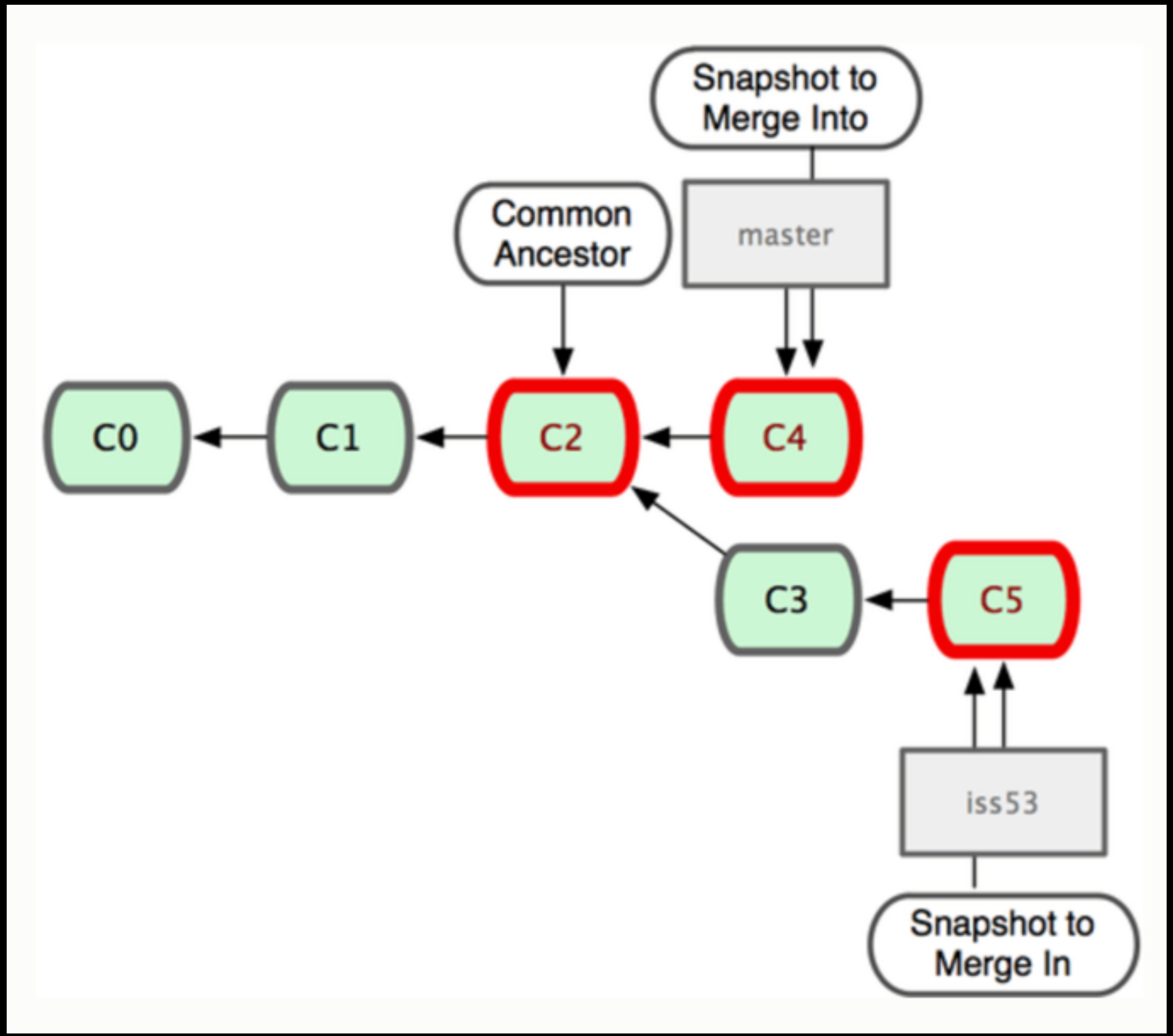
- Use the merge command to merge two branches pointing to different commits.
- A fast forward merge will not create a new commit:

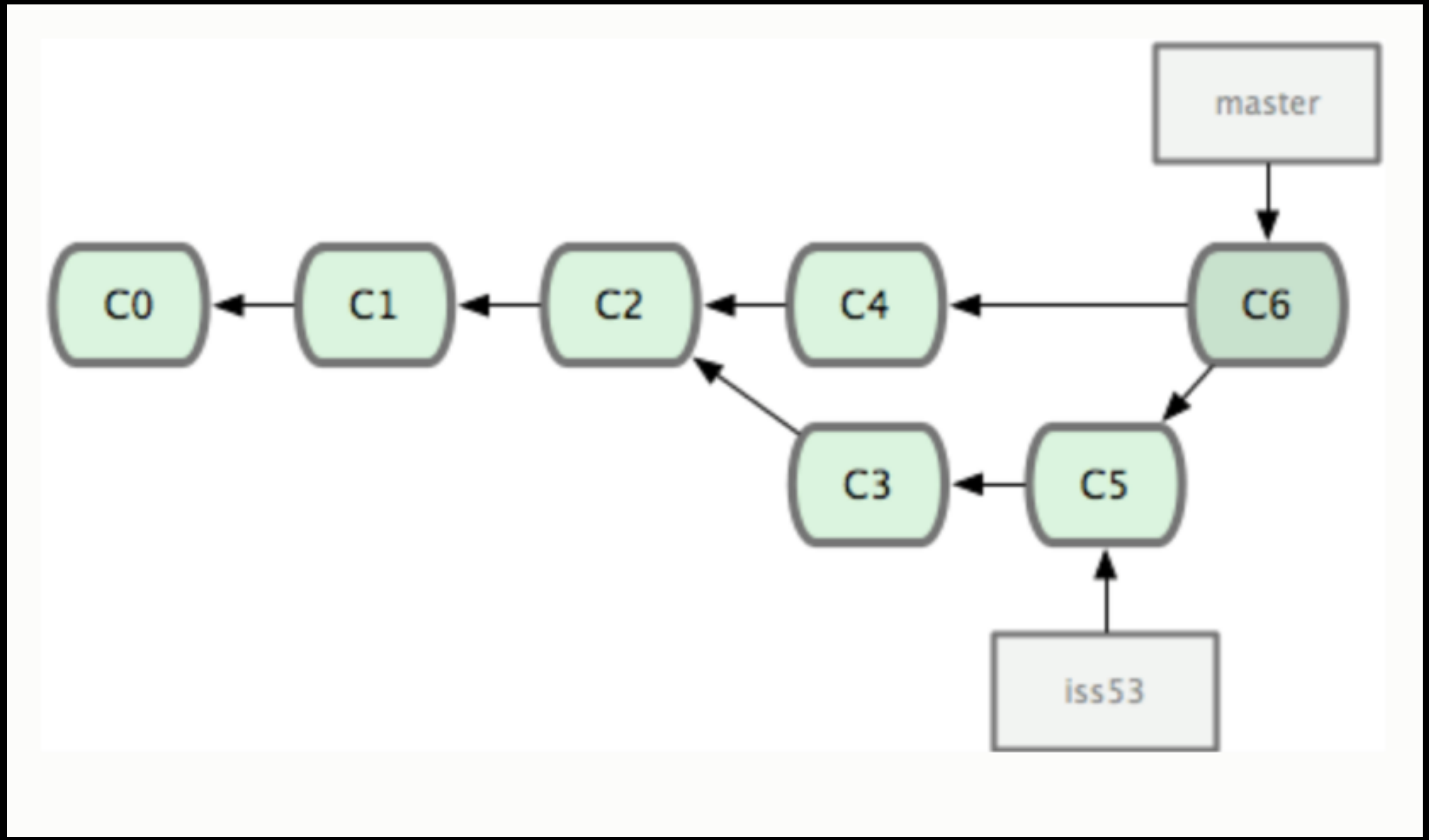




Merging

- A recursive merge happens when the two branches merging are not direct ancestors.
- A recursive merge will create a brand new commit as a result of the merge.





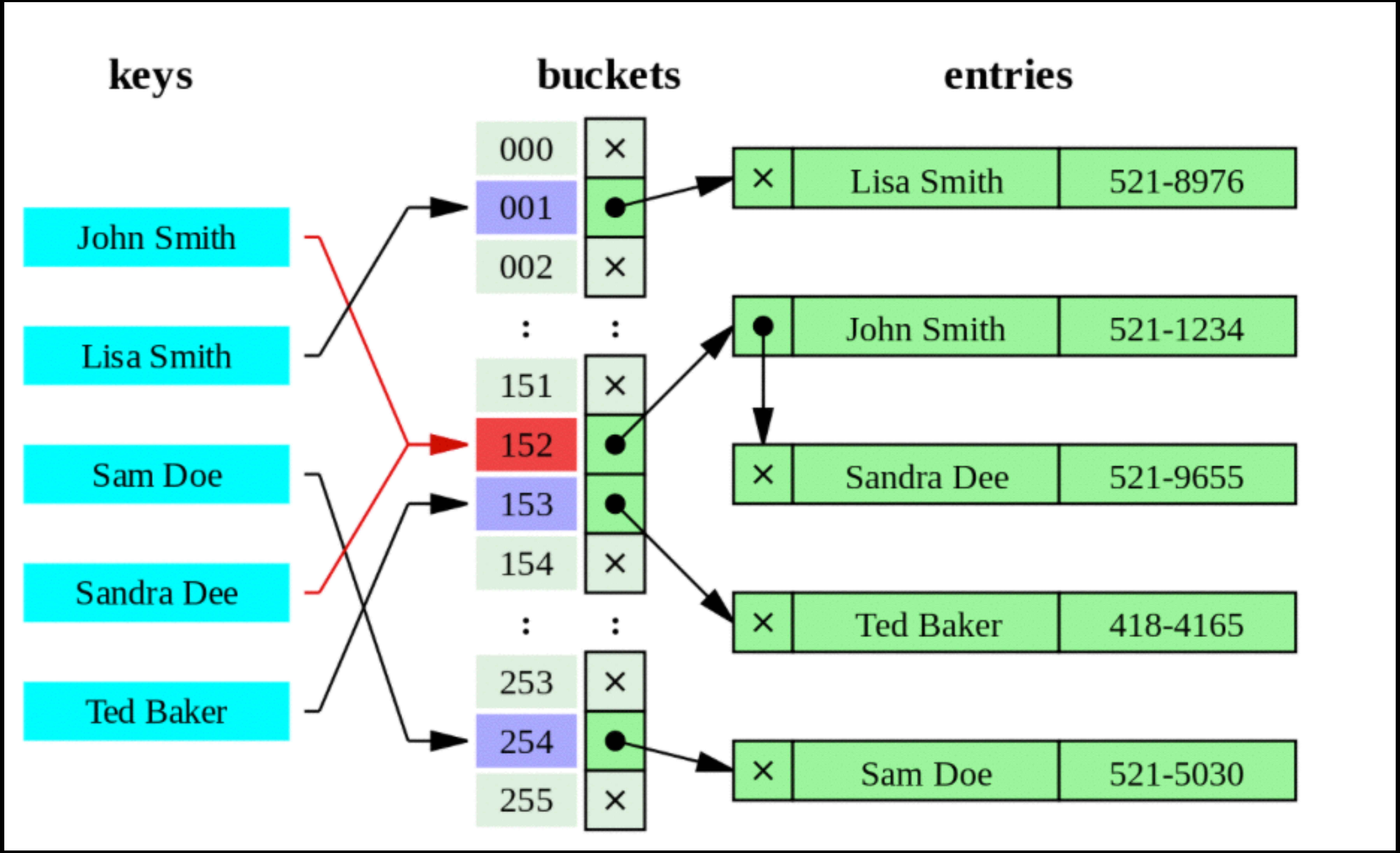
Git Fork Workflow

- A fork is just a clone on the server side.
- Once you fork, clone your fork to your machine and make your commits.
- When you are ready to push your commits, first do a pull from the 'upstream' remote to make sure you have the latest code and merge into your changes. (you will need to add a remote repo that points back to the original repo)
- Then push your changes to your fork and initiate a pull request.

Hash Tables

- A hash table is a data structure that can map keys to values.
- The key component to a hash table is a hash function. Given a key, the hash function computes an index to store the values in a backing array.
- On average, hash tables have an O(1) constant time look up. It’s amazing!
- Ideally the hash function will assign each key to unique index, but usually in practice you have collisions. In this case we use our old friend linked list to help us out.

Hash Tables


keys
buckets
entries
John Smith
Lisa Smith
Sam Doe
Sandra Dee
Ted Baker
000
001
002
151
152
153
154
253
254
255
Lisa Smith
521-8976
John Smith
521-1234
Sandra Dee
521-9655
Ted Baker
418-4165
Sam Doe
521-5030

Modulus Operator

- We are going to use the modulus operator in our simple hash function.
- The modulus operator finds the remainder of division of one number by another.
- so 10 % 3 is 1 because 3 goes into 10 3 times, and then a 1 is left.

Ternary Conditional Operator

```
var rowCount : Int!
if array.count > 5 {
    rowCount = array.count
} else {
    rowCount = 5
}
```

```
var rowCount = array.count > 5 ? array.count : 5
```

Nil-Coalescing Operator

- The *nil coalescing operator* (`a ?? b`) unwraps an optional a if it contains a value, or returns a default value b if a is `nil`.
- The expression a is always of an optional type.
- The expression b must match the type that is stored inside a.
- The nil coalescing operator is shorthand for the code below:

```
a != nil ? a! : b
```

Operator Overloading

- Overload operators on your own types or pre-existing types or create custom operators
- @Prefix (-a) and@ Postfix (a++)
- @Infix (a+b), (a==b),(a!=b)
- @assignment for (+=)

Operator Overloading

```
func * (left: String, right: Int) -> String {
    if right <= 0 {
        return ""
    }

    var result = left
    for _ in 1..<right {
        result += left
    }

    return result
}

"a" * 6  // -> “aaaaaa”
```