iOS Dev Accelerator
Week 1 Day 4

- Autolayout
- Git Basics

Autolayout

- Is Awesome
- Autolayout is a system that lets you design dynamic interfaces that respond appropriately to changes in screen size and orientation.
- You setup the rules of your interface and Autolayout takes care of the rest.

Autolayout

- To properly use Autolayout, it must know 2 things about every view in your interface:
  1. How big the object is going to be
  2. Where the object is going to be located
- You accomplish this using constraints.
- Demo

What is Git?

- Git is an open source version control system.
- Git is an application you install on your computer.
- Git itself is not Github, Github is a hosting service for git repositories.
- type git —version into terminal to see if you have git installed and which version you have.

Git Basics

- git init - Creates an empty git repo or reinitializes an existing one.
- After running git init, there will be a hidden .git file in the directory you are in. type ls -a to list all files. You will see the .git file.
- this .git file is a directory that takes snapshots of your project's files every time you commit or save the state of your project.

Git Basics

- Files being monitored (aka tracked) by a .git have 3 possible states:
    1. Committed - data is safely stored in your local database
    2. Modified - you have changed the file but have not committed it to your database yet
    3. Unmodified - this file has no changes since the last snapshot

Git Basics

- the .git files keeps track of everything by tracking 3 sections:
- Working Directory - a single checkout of one version of the project. These files are uncompressed after being pulled out of the object database and placed on disk for use.
- Staging area - one file, stores information about what will go into your next commit.
- Git Directory - where git stores the metadata and object database of your project

Git Workflow

1. You modify files in your working directory.
2. You stage the files, which adds snapshots of them to your staging area.
3. You do a commit, which takes files as they are in the staging area and stores snapshot permanently to your git directory.
4. Profit

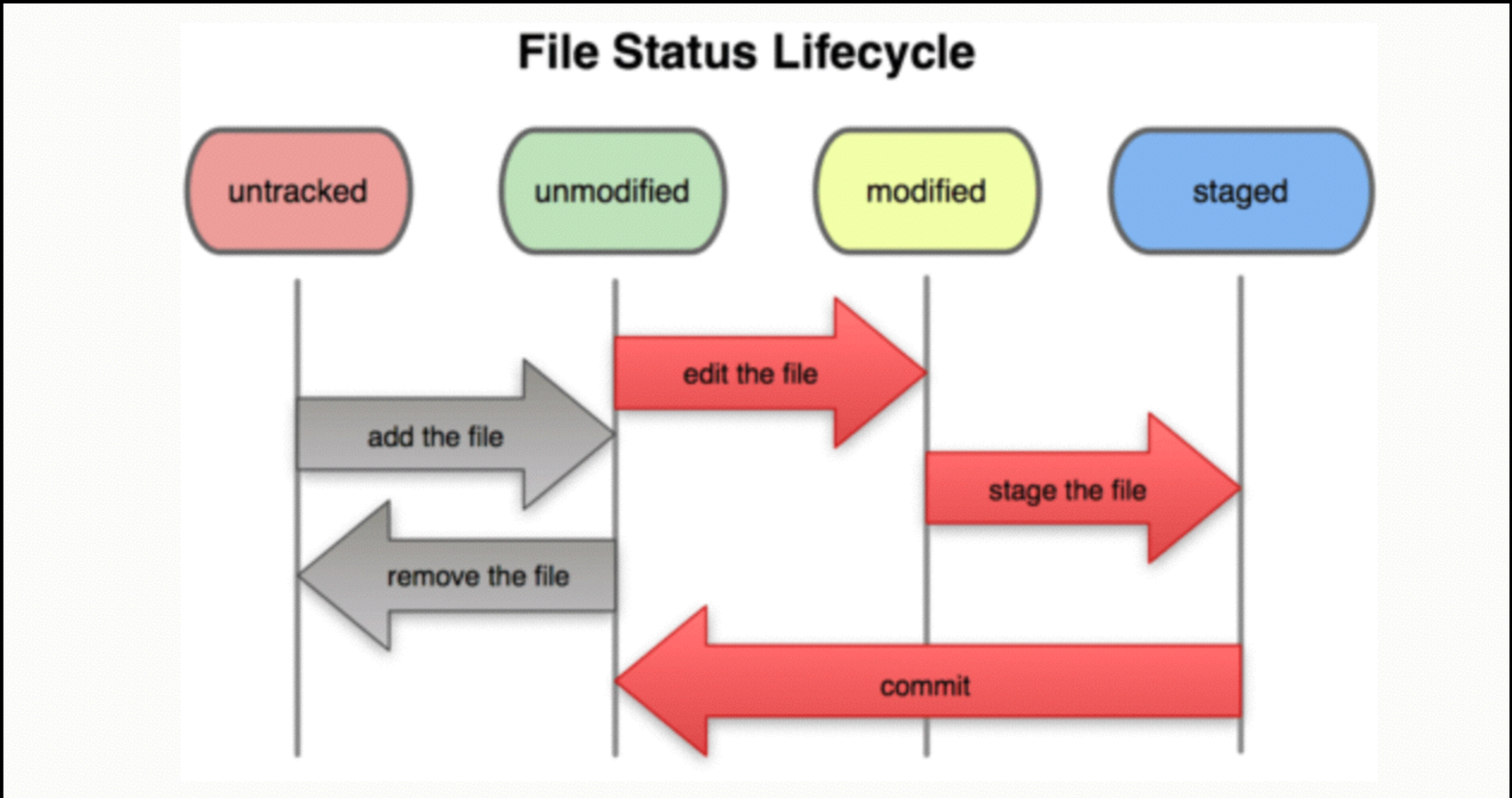
Git Clone

- Use the clone command to get a copy of an existing git repository.
- Every version of every file for the entire history of the project is pulled down with clone.
- git clone ‘url’
- Demo

Git – Working Directory

- Every file in your working directory has only 2 states - tracked and untracked.
- tracked means it was in the last snapshot, and untracked means it wasn’t.
- Once a file is tracked, it can then be one of 3 states: unmodified, modified, staged

Git File Status Lifecycle



Git Basics

- Use the git status command to determine the state of your files
- It also tells you which branch you are on
- use git add to begin tracking new files and to stage files after they have been modified
- use git add to also mark a merge conflict as resolved, more about this later.
- Once you stage all your changes with git add, use git commit to commit those changes.
- git commit -m “commit message” is quicker and cleaner than git commit

Git Commit History

- use git log to see the commits made to this repo in reverse chronological order. The most recent commit shows up first
- use flag -p to show exactly what code was introduced
- -<n> to show n number of last commits (-2 to see last 2 commits)

Git Undo

- if you commit too early, use git commit —amend after you have staged the files you want to commit
- use git reset HEAD <file> to unstage a file
- use git checkout — <file> to revert a file back to the state it was in in the last commit (be careful with this, you lose all the changes you made!)

Git Remotes

- Remote repos are versions of your project that are hosted in the Cloud™ or a private network.
- Run the git remote command to see if you have any remote servers configured.
- Cloning a repo automatically adds that remote repo under the name origin. (git clone <url>)
- Origin is the default name of a remote repo, but you can call it whatever you want

Git Remote Commands

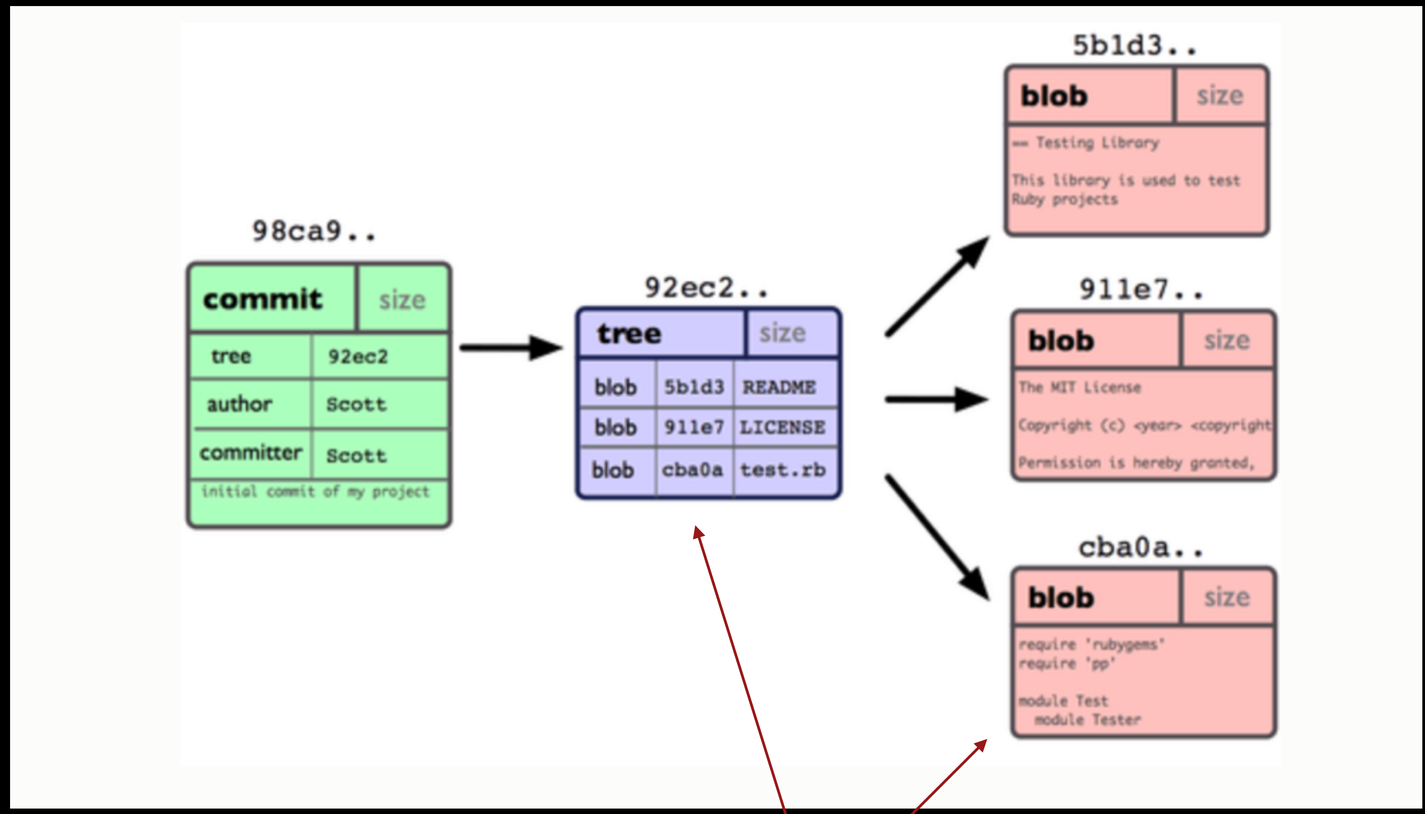
- git remote -v shows the URL for each remote as well
- use git remote add <nickname> <url> to add a remote repo
- after committing, use git push <nickname> <branch> to push your committed changes to your remote
- use git pull to automatically fetch and merge a remote branch into your current branch
- Demo!

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

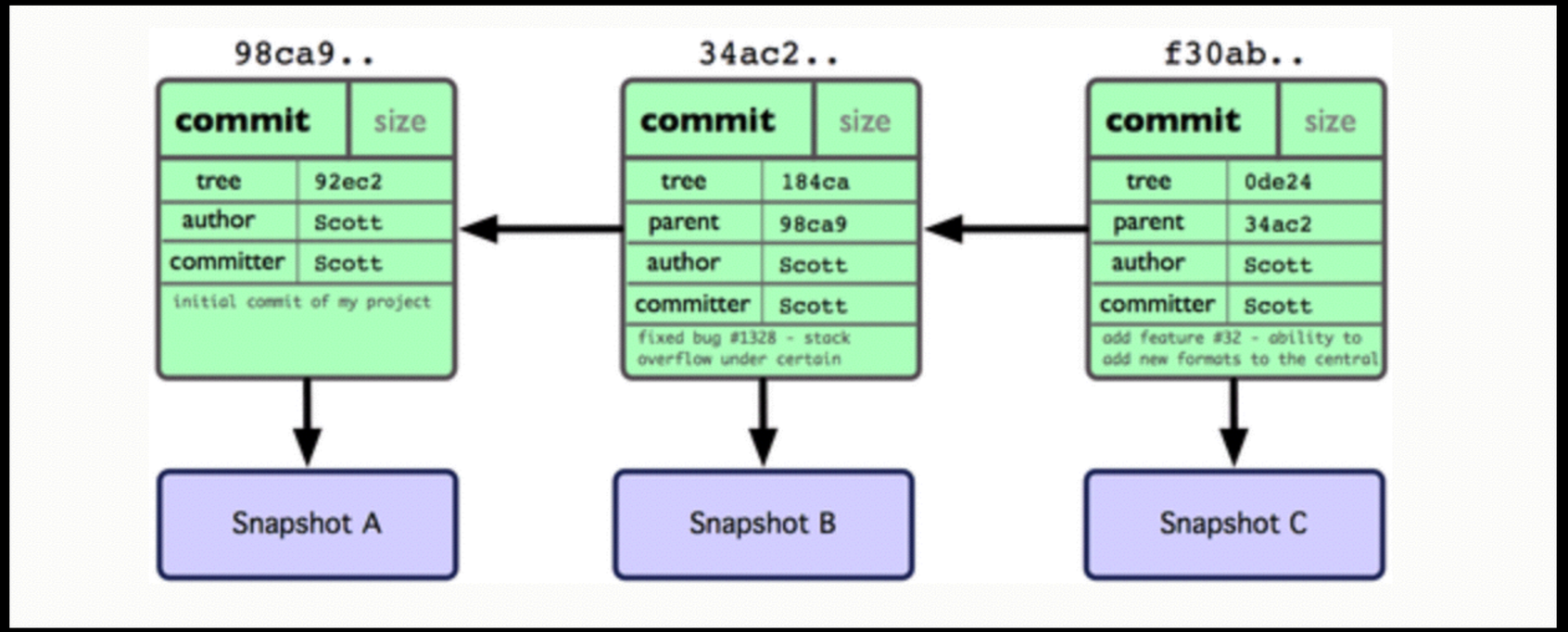
Anatomy of a Branch

- 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



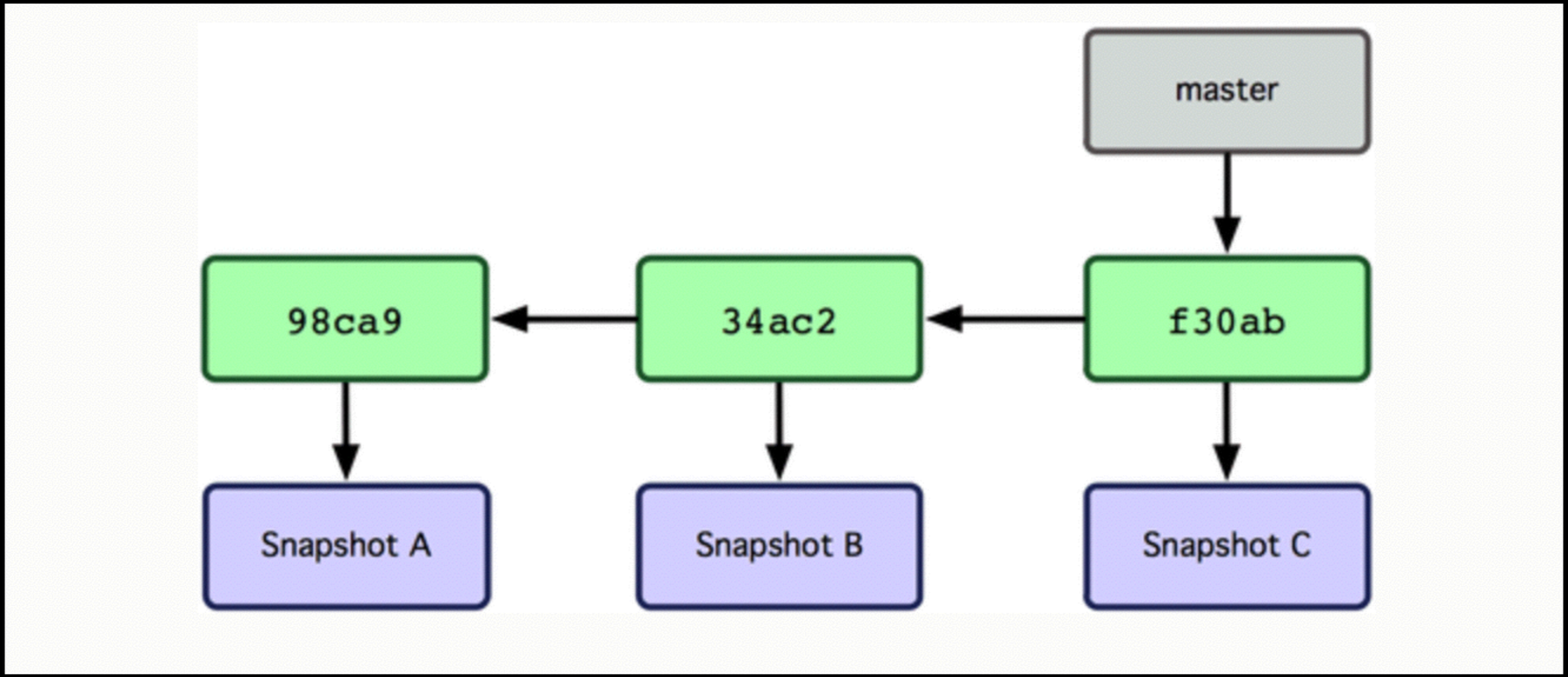
"Snapshot"

Multiple commits



first commit

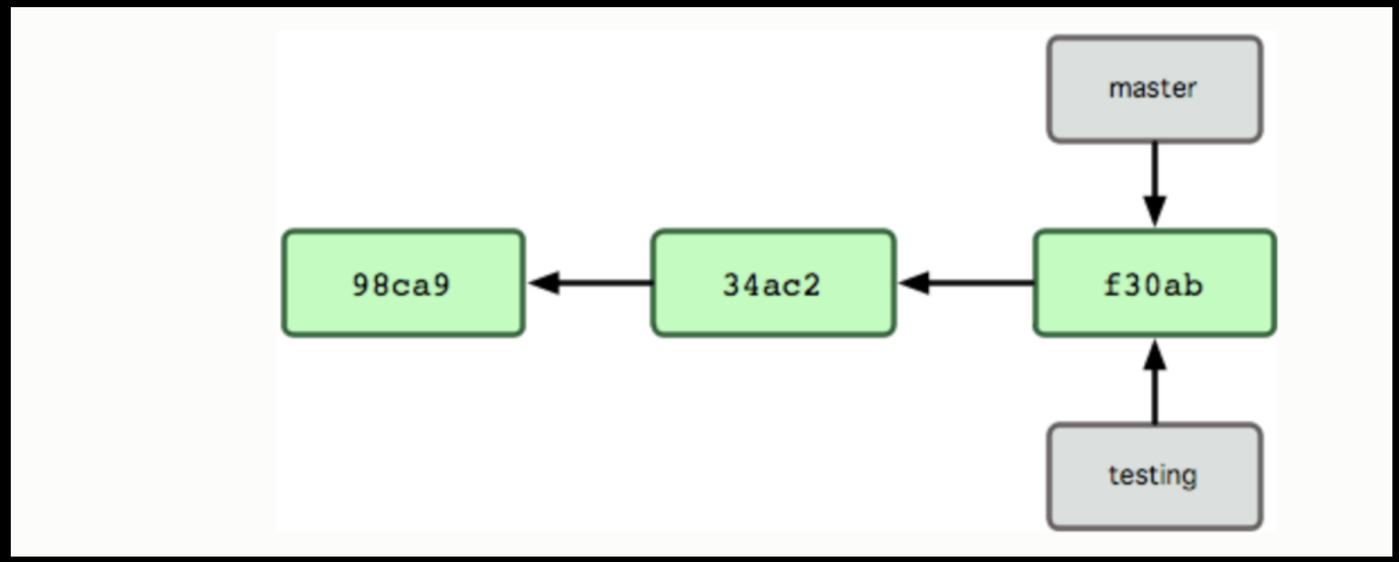
Branch == pointer



A branch is just a lightweight pointer to a commit

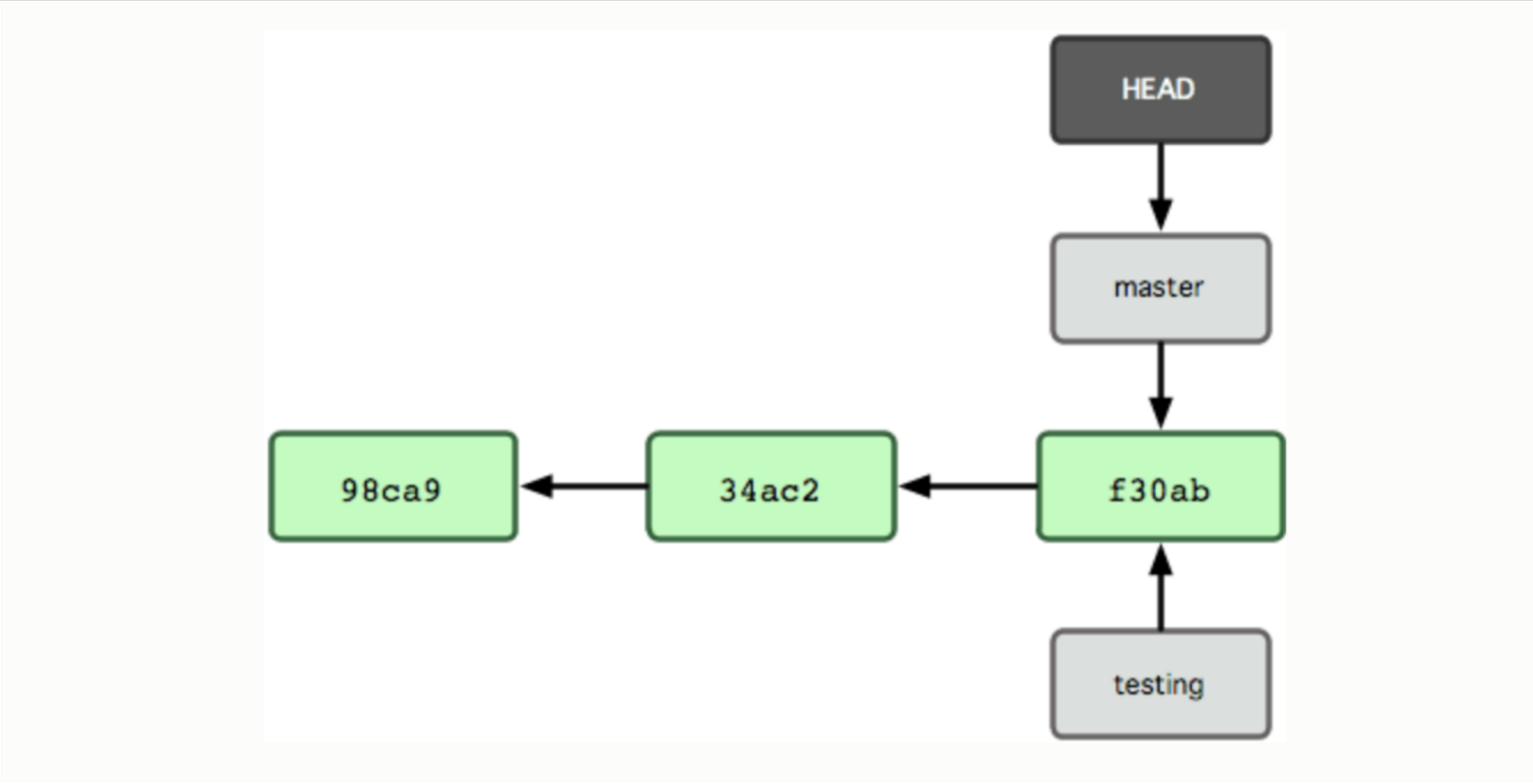
Creating a branch

- 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



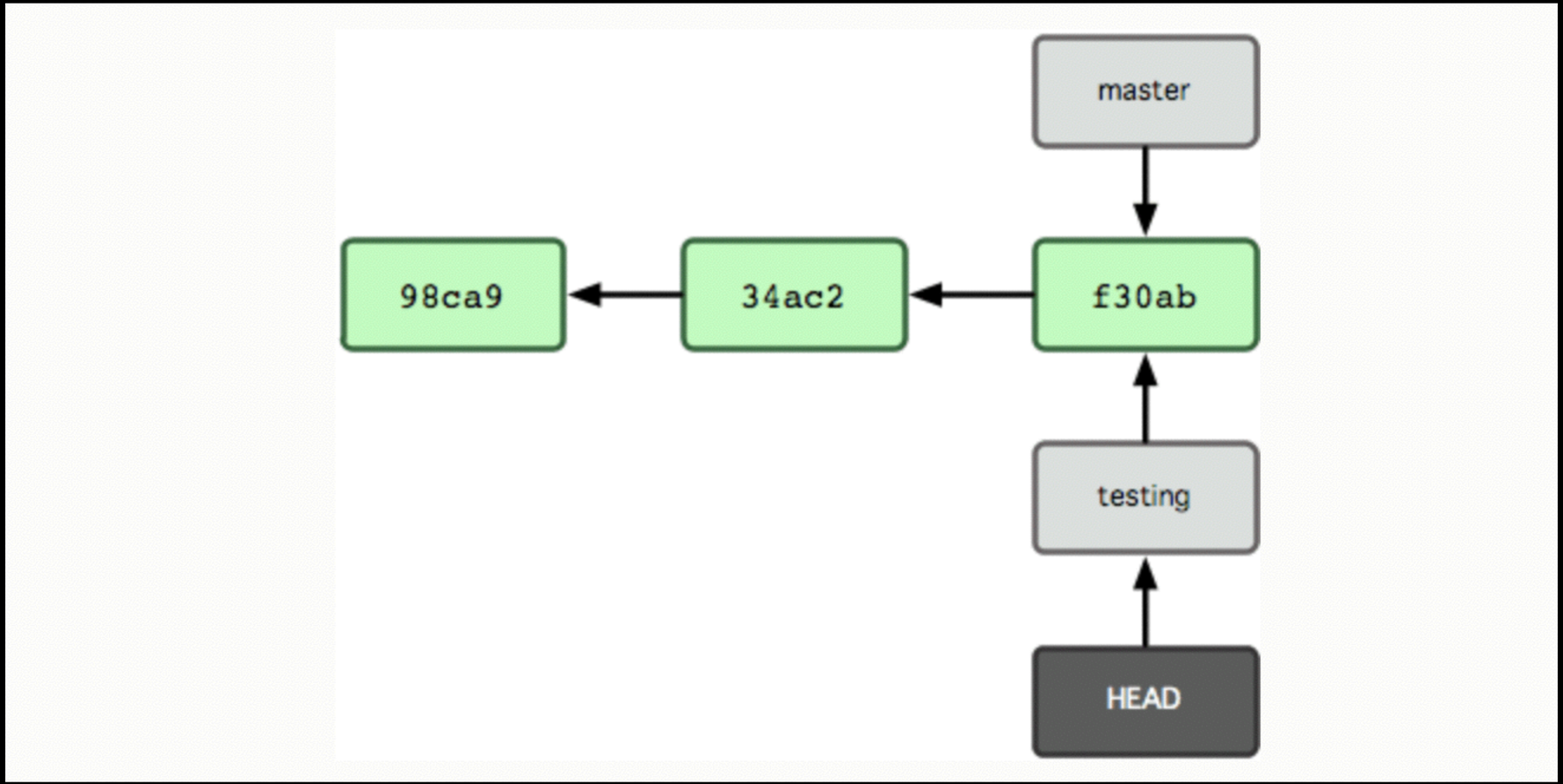
HEAD pointer

- 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.



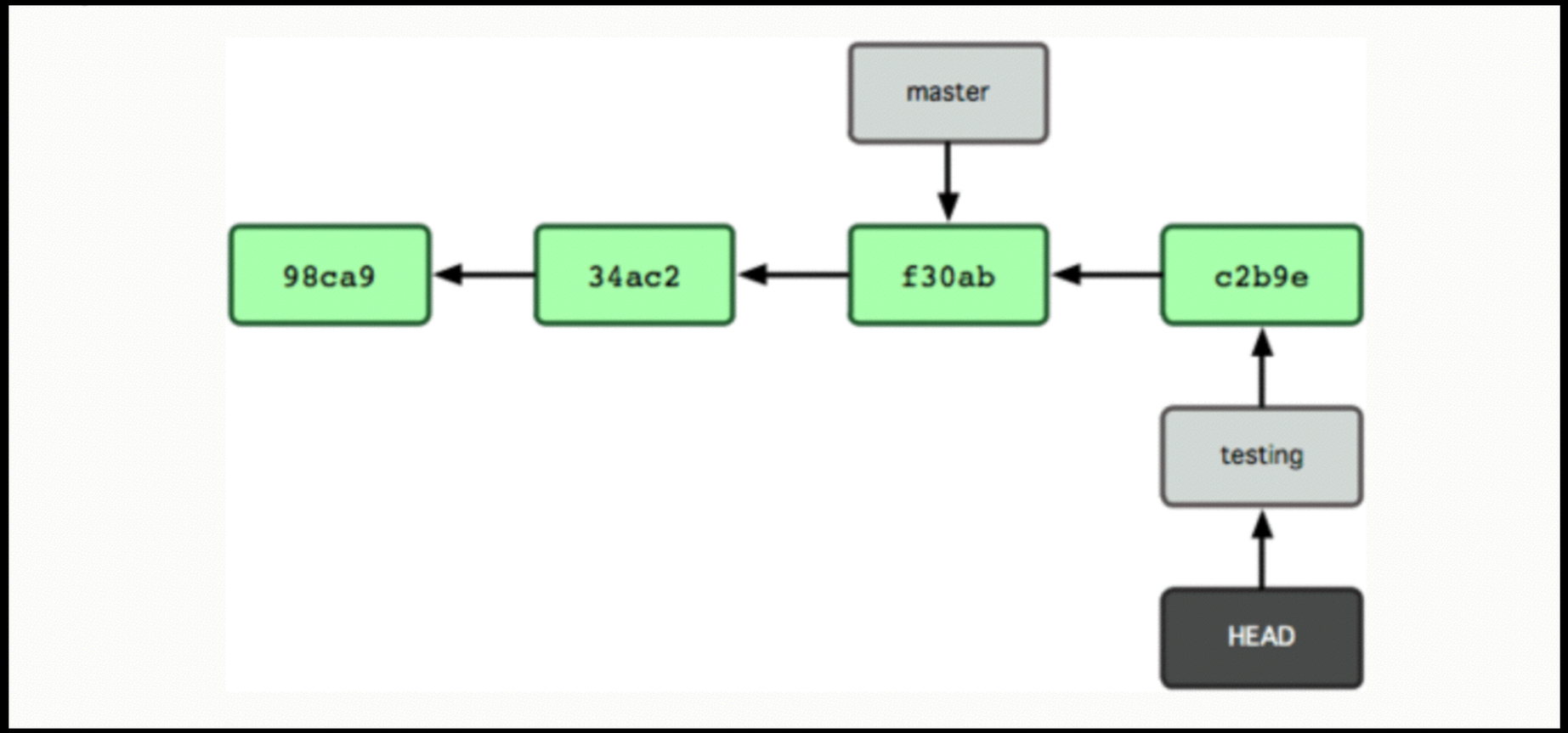
Git Checkout

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



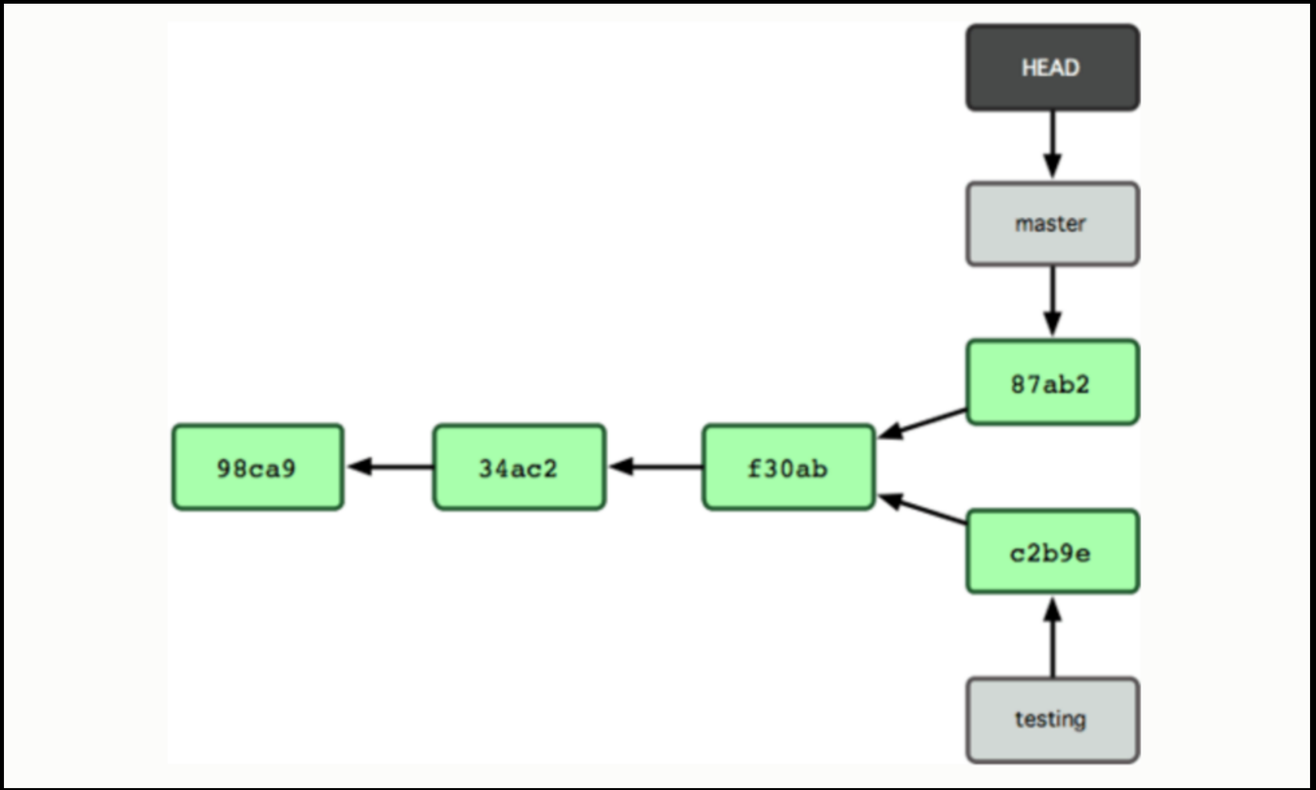
Moving forward

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



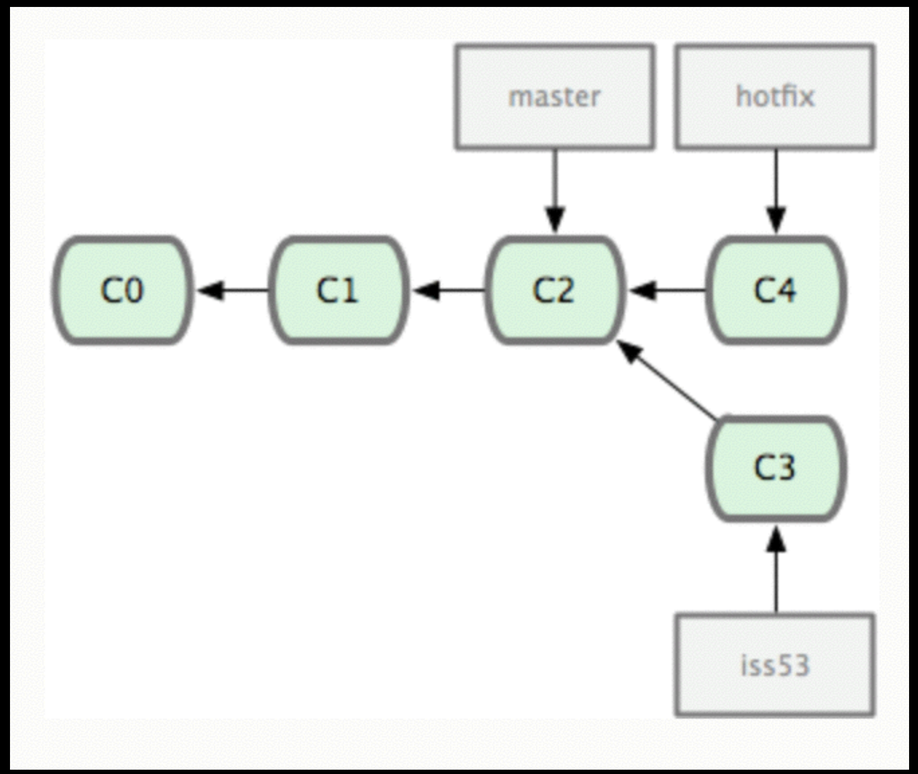
Fork in the road

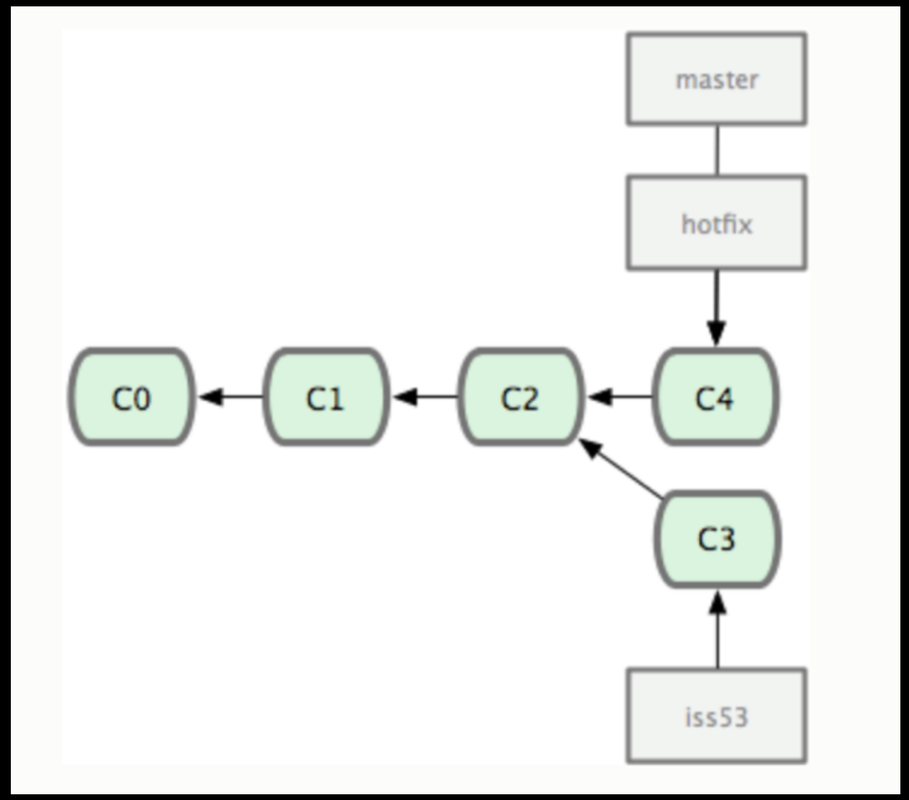
- Now if you checkout master and make commits, we have a divergence.



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.