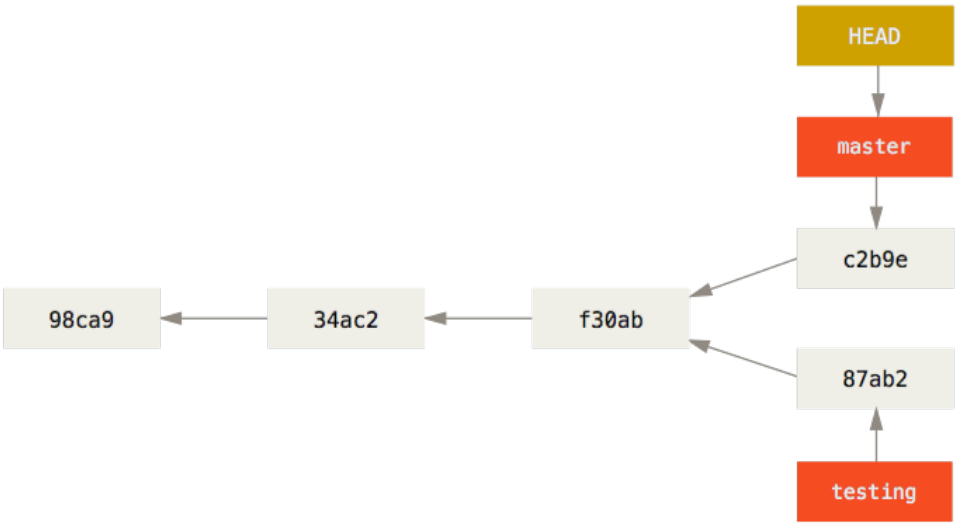
02 Git

April 28, 2022

1 Git Basics: Going Back in Time, Creating and Managing Git Branches

What good is version control software, if you can't go back to different versions of your repository? In this section, we'll talk about git's model of the timeline *("branches of commits")* and how it labels the currently-active commit *("the HEAD")*. We'll move the head to different commits *("checking-out" a commit)*, add multiple timelines to the project *("branching")*, and merge the changes from those different timelines together back into a single timeline *("merging branches")*.



1.1 Terminal Commands

- **`git branch`**: Display the list of names of branches in this repository
- **`git branch newbranchname`**: Create a new branch with "newbranchname" at the current commit
- **`git branch -d branchname`**: Delete a branch
- **`git log`**: Checks the log of all past commits. **To exit the log, press the `q` key.**
  - **`git log -1`**: Shortcut, only shows the last one commit.

- `git checkout branchname`: Make your workspace match the commit at branchname.

- `git checkout HEAD~1`: Make your workspace match the commit one step back in time.
- `git merge branchname`: Merges branchname into the current branch

1.2 Exercise

Building off the Git repository we created in the last exercise, let's checkout different commits in the log, create new branches with new changes, and merge them together. By the end, we'll even recover from deleting a whole file by going back in time to before the file was deleted!

Sequence

Current State: At this point, you should have a Readme file and a main.py Python file, with a few commits created.

1: What branch names are there in your repository?

2: Create a new branch called "temporary", then check the list of branches. Which branch are you currently working on?

3: Check the log. Which commit is the HEAD pointed at? Are the ends of any branches there as well?

4: Checkout the "temporary" branch, then look at the list of branches. Which branch are you currently working on?

Current State: We now have two branches, "master/main" and "temporary". Their names don't mean anything, nor do the fact that they are branched, because they are all currently set at the same exact commit in our project. Let's change that by making a commit!

5: Delete the **main.py** file, add the changes, then commit the change with the message "deleted python file". Check the status and the log and verify the change was committed

6: Take a look at your log. What commit is the HEAD at? What commit is the end of the "temporary" branch at? What commit is the end of the "master" branch at?

Current State: Now our branches end at different commits, with the currently-active one moving along with the head and the unactive one staying in place.

7: Checkout the master branch, then check the log. Where is the HEAD now? Do you see any differences in your files/folders?

8: Checkout the "temporary" branch, then check the log. Where is the HEAD now? Do you see any differences in your files/folders?

9: *Discussion*: What is the "checkout" command doing?

Current State: Thanks to our named branches, we can now go back and forth between two versions of our project just be checking out the branch!

10: Let's merge the two branches, bringing the changes from one branch into the other. Merge the "temporary" branch into the "master" branch. *(Note: you must be currently on the branch that you want to change).* Check the log, are the ends of the two branches at the same commit? Check the files. Are they as you expect them to be?

11: Now that the two branches are basically the same, let's delete one of them. Delete the temporary branch, then confirm that it's gone.

Current State: Now we should only have one branch left, the "master/main" branch, and the HEAD should be at the most recent commit. Does this mean that we can no longer go back in time? No, the HEAD can still be checked out to any commit.

12 (extra). Oops, I forgot to mention that we actually wanted that main.py file, we just wanted it called "run.py" :-D . Let's move the HEAD back a commit and copy-paste it (don't delete it) to a new file called run.py. Check the status; no changes to existing files should be registered. Then go back to the end of the master branch, add the run.py file, and commit it!

13 (extra). What if we want to go back to the very first commit? The most useful thing here is to get the commit's unique "hash", that complex string of letters and numbers. You don't need the whole thing; just copy the first six letters or so, then checkout to that hash. Git will go straight to the commit!

14: Once you're done exploring the history, checkout the master branch so you're looking at the most recent commit

Current State: We have one branch, a new **run.py** file, and we're looking at the most recent commit.

[ ]: