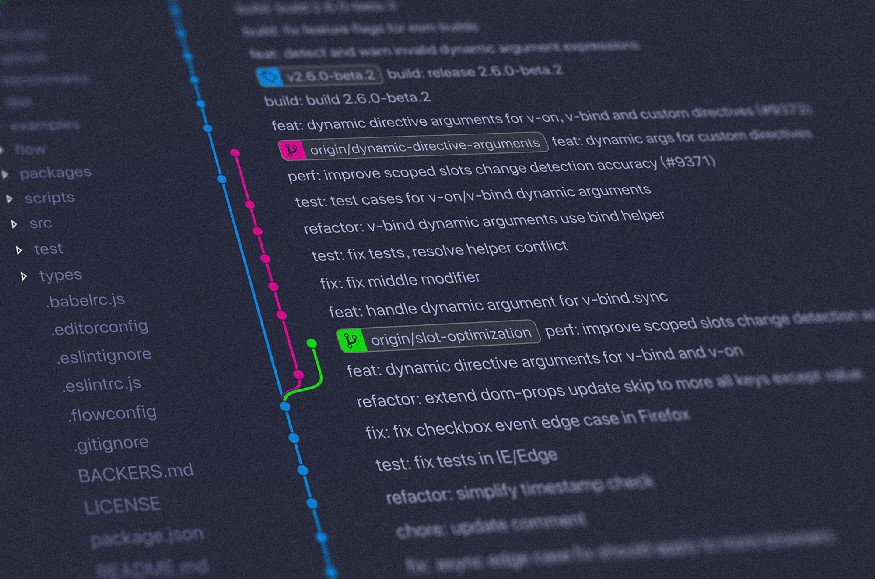
**Understanding Refs, branches and dangling commit in Git?**



Just a free stock image

Have you ever wondered how “HEAD” works on git? how branches are assigned, switched and created? what is detached HEAD state? Just bear with me for some minutes and you won’t bother about it in the future now (if you are a genius and never make mistakes).

If you understand this article you can fully understand the concept of reset, rebase and revert.

**Some knowledge you should already know**

* what are commits? (just the feeling, not the definition)

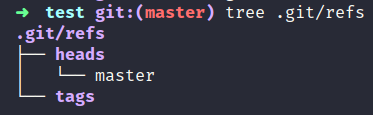
(Skip if you know already) Commits are just the changes that are saved in a project, every time you change a file it will keep a track of those changes by assigning it a unique hash (a random 40 characters that are almost unique), just like checkpoint you save in a game and can go at any of the checkpoints and start your game from that checkpoint.

**Refs or References**

*Refs or references are the pointers to commits.*

that’s it. But let’s look at it from the other side:

Every git enabled project have .git folder where all the dirty work is done under the hood, and it is where all the data about refs are stored.



refs

So if you see the directory structure of .git/refs where all the references are stored, you will find there are two types of refs

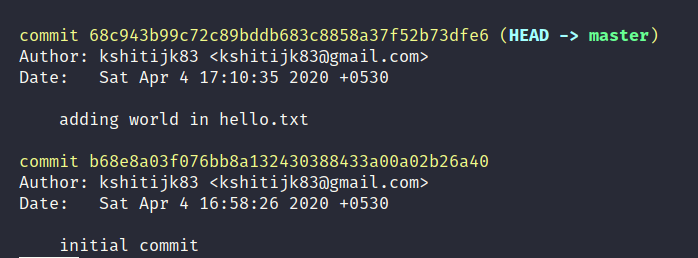
* branches (heads)
* tags (this i will skip as it’s easy if you know refs)

his “heads” folder has references called branches, here there’s currently have only branch: master .

**So, what is a branch?**

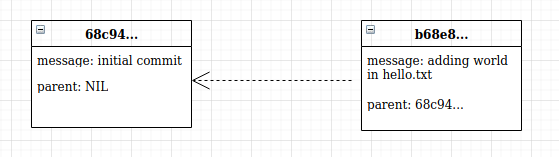
*A branch is just a pointer to a particular commit.*

Let’s understand it deeply. I am assuming you know how to commit changes in a project (git add .&&git commit -m "your message" ). I have made a git initiated project and added a file name “*hello.txt”,*and commit it with the different messages as shown:



git log

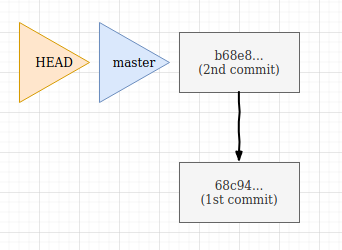
And these two commits are connected as the latest commit will have the parent commit as the previous one before it. That means:



parent of 2nd commit is the first one

A **branch** is simply the movable pointer to one of these commits. The default branch name in Git is called master. As you start making commits, you’re given a master branch that points to the last commit you made. Every time you commit, the master branch pointer moves forward automatically.

So, currently master points to this commit:



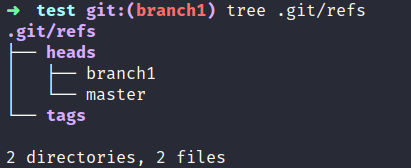
state 1

**So what’s a HEAD?**

HEAD is how a git knows what branch you’re currently working on. It’s a pointer that points to the name of the current branch. It moves automatically when you checkout a new branch or make a commit to a branch. if it tops over your head let’s see it practically.

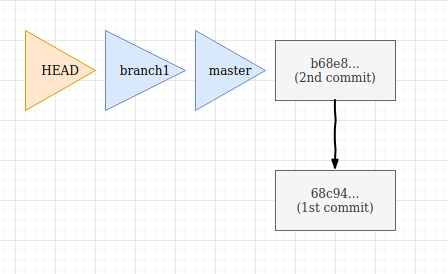
Now suppose our current project is in *state 1 as shown in the above diagram, let’s make a new branch*

git checkout -b branch1



what are the branches we have currently

now we have two branches in .git/refs/heads. and our current branch is branch1 .



state 2

branch1 is created which points to the same commits where master was pointing at (last commit). And now HEAD is pointing at branch1.

If you want to see where HEAD points at currently you can check by seeing what the “HEAD” file contains in the .git folder.

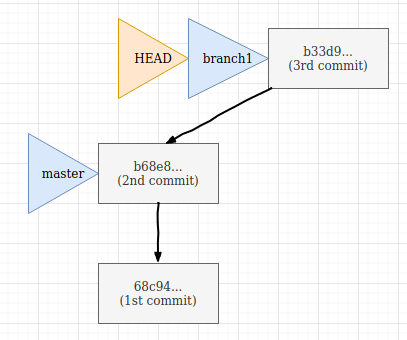
cat .git/HEAD



head points to branch1 in refs folder where all the references are stored.

Now, what happens if i make another commit working while working on branch1 ?

echo "change on branch1">hello.txt  
git add .  
git commit -m "commit 3"



HEAD will move with the current branch it points to, i.e., branch1.

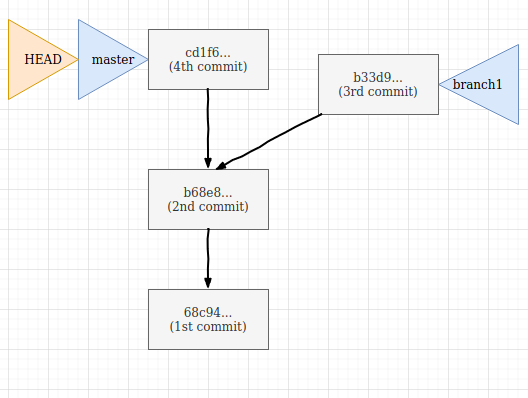
so if i move back to master, HEAD also moves and points to master, so we can say:

*HEAD is a pointer that generally points to the current branch we are working on, hence proved!! (yeah, right!).*

After moving back to master, you again commit something and that will be on master branch like this:

git checkout master // switched to master  
echo "changing the master branch file">hello.txt  
git add . && git commit -m "change content in master branch"

Now this will happen:



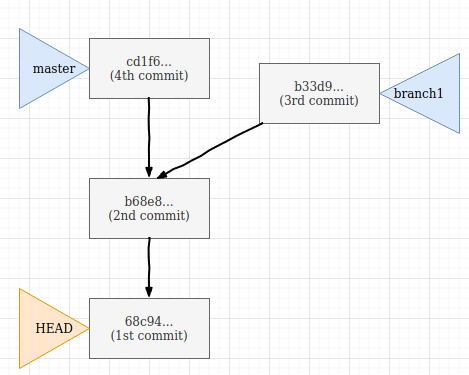
but if you want to see the contents of a specific commit? That’s a bonus for you if you have reached here(thanks!).

**Headless/Detached HEAD state?**

If you want to go and see what were the contents of the 1st commit, then you will checkout that specific commit.

What git will do, it will point HEAD to that commit

git checkout 68c94  
> // You are in 'detached HEAD' state...



state 5

now the contents when the first commit was made are in the “hello.txt” file because we are back in time!!!. you can see where HEAD points now

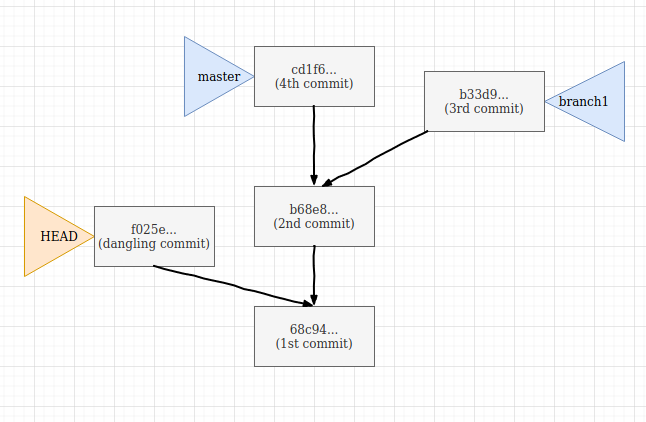


head points to that commit

You can go back to master by this:

git checkout master

But if you want to change something in Detached head state and made a commit



state 6

Now HEAD will points to that commit, but as this commit doesn’t have any branch referenced to it and if we go back to master branch, this commit will be garbage collected and hence is called a **Dangling commit**.