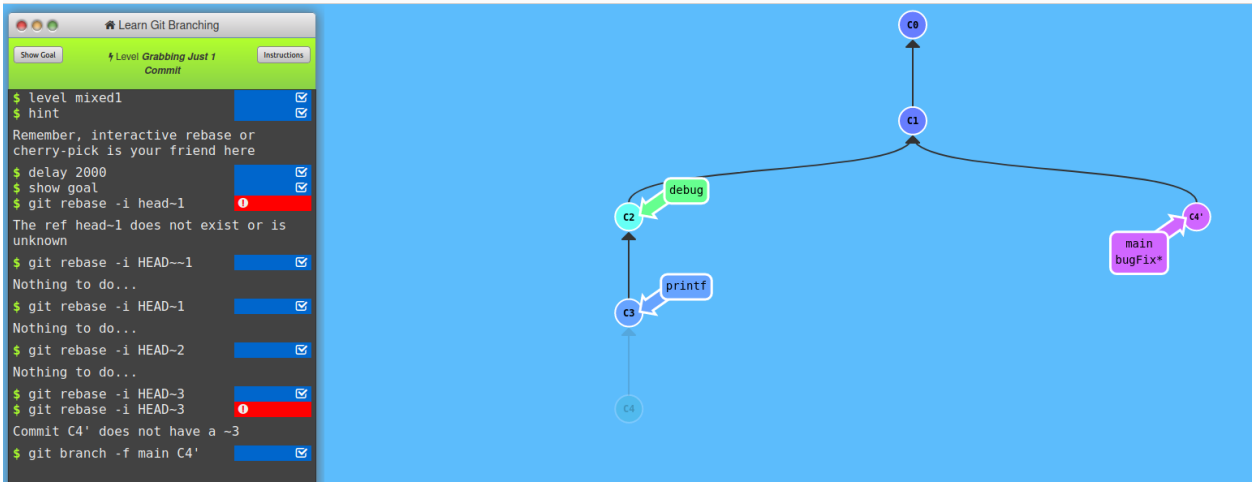
GIT:Level4

Task 1:

In this task we have to rebase the commit and forcefully move the branch to the certain commit to achieve the goal.




```
git rebase -i HEAD~3
pick commit C2 and C3 and omit C4
git branch -f main C4'
```

Task 2:

In this task we have to use git commit —ammen command. The `git commit --amend` command is used to modify the most recent commit. This can be useful if you want to change the commit message, add new changes, or modify the content of the commit.

Juggling Commits

Here's another situation that happens quite commonly. You have some changes (`newImage`) and another set of changes (`caption`) that are related, so they are stacked on top of each other in your repository (aka one after another).

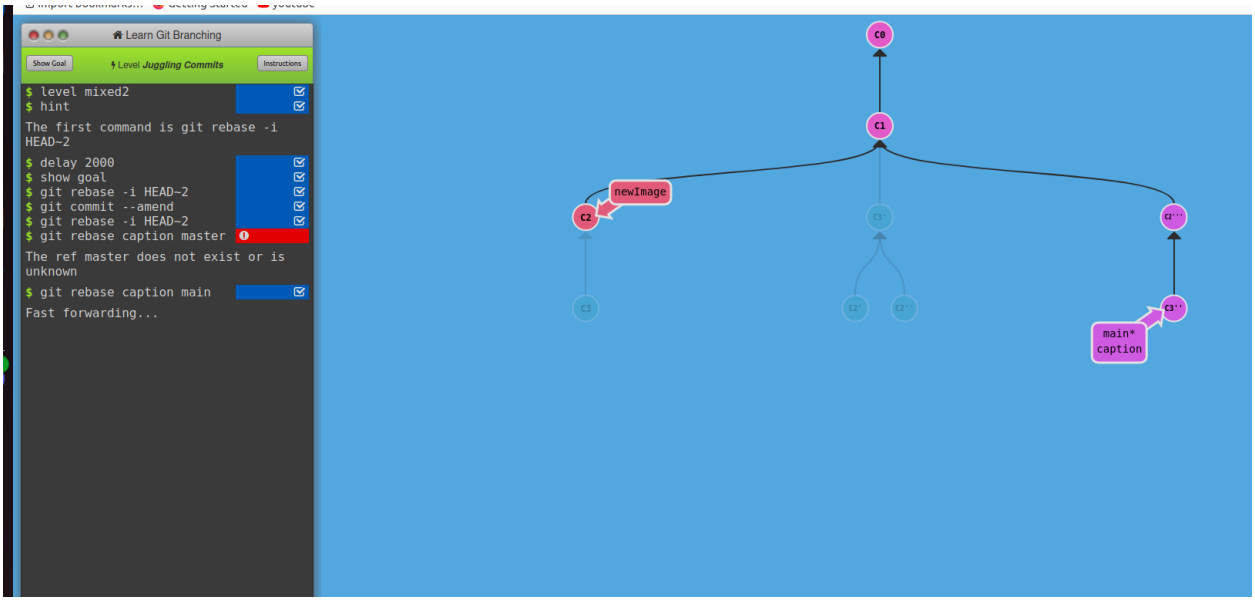
The tricky thing is that sometimes you need to make a small modification to an earlier commit. In this case, design wants us to change the dimensions of `newImage` slightly, even though that commit is way back in our history!!

We will overcome this difficulty by doing the following:

- We will re-order the commits so the one we want to change is on top with `git rebase -i`
- We will `git commit --amend` to make the slight modification
- Then we will re-order the commits back to how they were previously with `git rebase -i`
- Finally, we will move main to this updated part of the tree to finish the level (via the method of your choosing)

There are many ways to accomplish this overall goal (I see you eye-ing cherry-pick), and we will see more of them later, but for now let's focus on this technique. Lastly, pay attention to the goal state here -- since we move the commits twice, they both get an apostrophe appended. One more apostrophe is added for the commit we amend, which gives us the final form of the tree

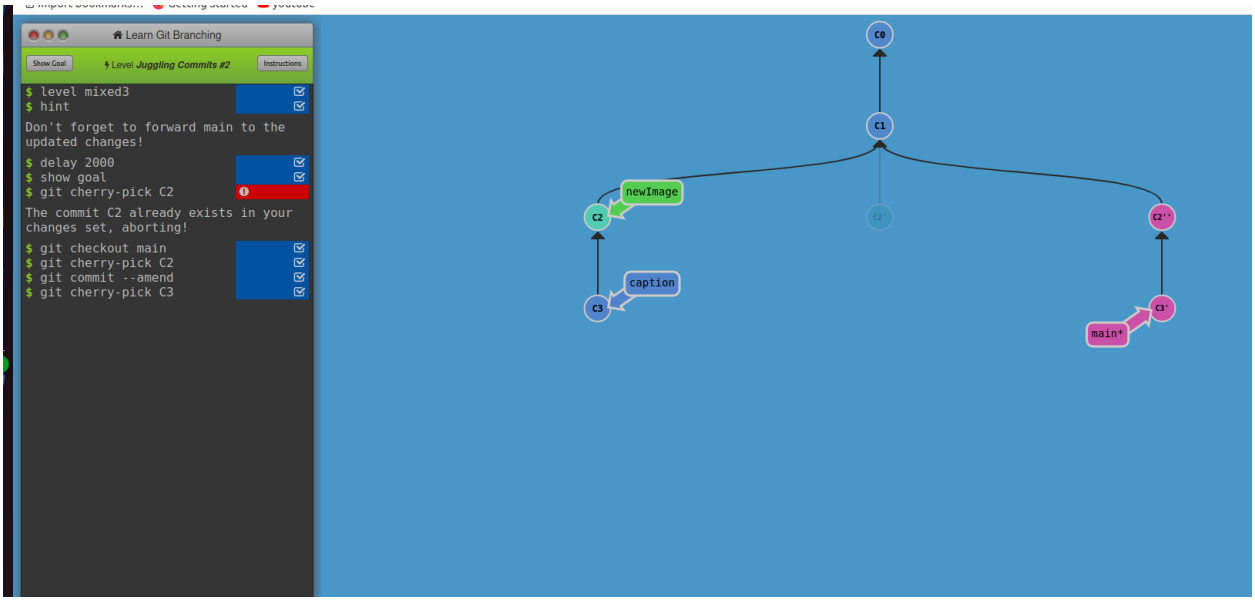
That being said, I can compare levels now based on structure and relative apostrophe differences. As long as your tree's `main` branch has the same structure and relative apostrophe differences, I'll give full credit.




```
git rebase -i HEAD~2
rearrange
git commit --amend
git rebase -i HEAD~2
rearrange
git rebase caption master
```

Task 3

In this task we have to use git cherry-pitk command to achive the following goal.

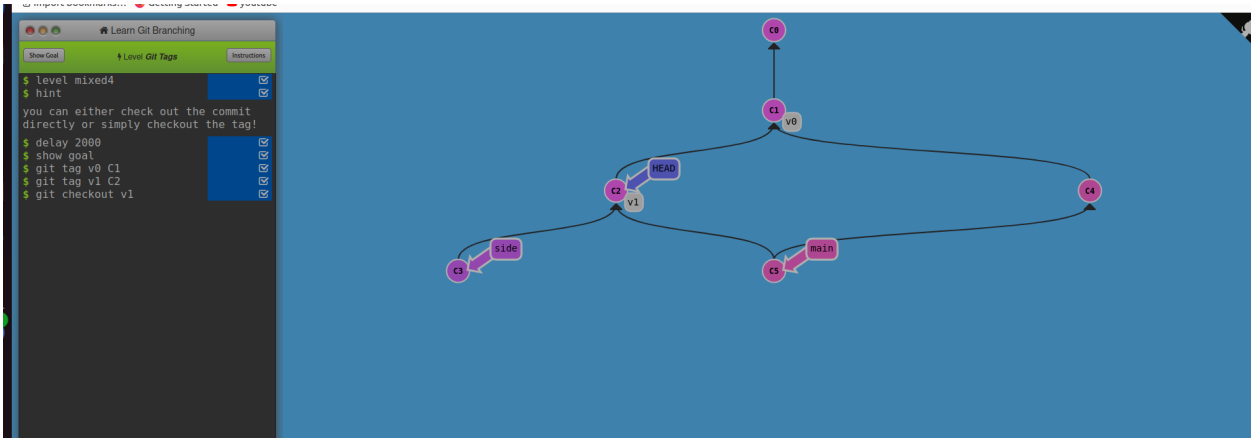


cherry-pick command move the commit to that branch onto which the head is pointing. In this initially the head is pointing to caption branch. So to achieve the goal move the head pointer to the main branch.

```
git checkout main
git cherry-pick C2
git commit --amend
git cherry-pick C3
```

Task 4

In this task we have to use git tag.



```
git tag v0 C1
git tag v1 C2
git checkout v1
```

Task 5

Git Describe

Because tags serve as such great "anchors" in the codebase, git has a command to *describe* where you are relative to the closest "anchor" (aka tag). And that command is called `git describe`!

Git describe can help you get your bearings after you've moved many commits backwards or forwards in history; this can happen after you've completed a git bisect (a debugging search) or when sitting down at the computer of a coworker who just got back from vacation.

Git describe takes the form of:

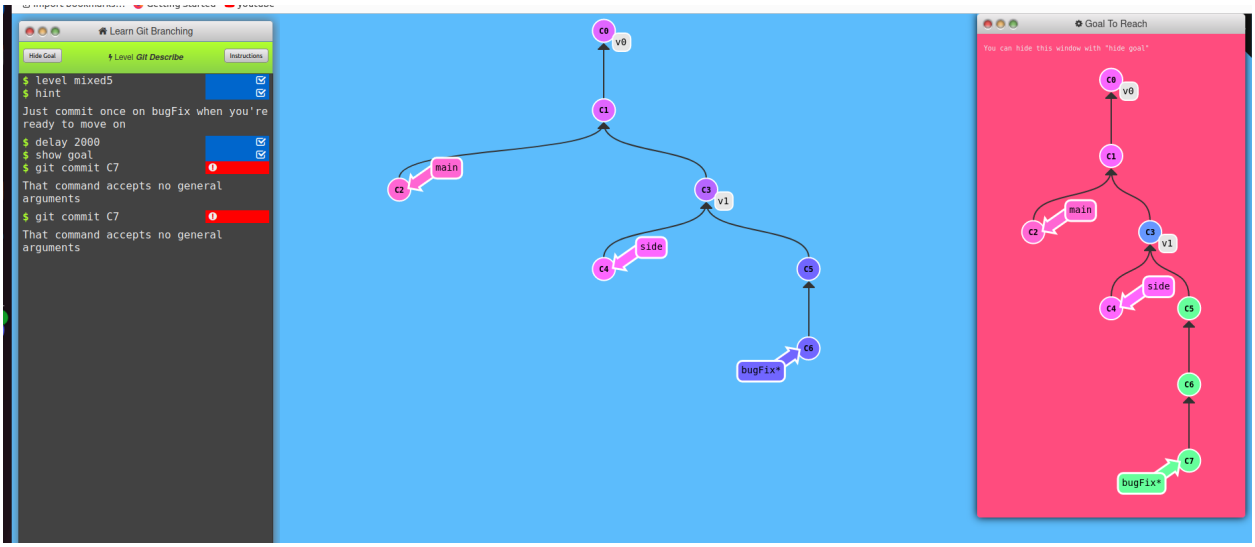
`git describe <ref>`

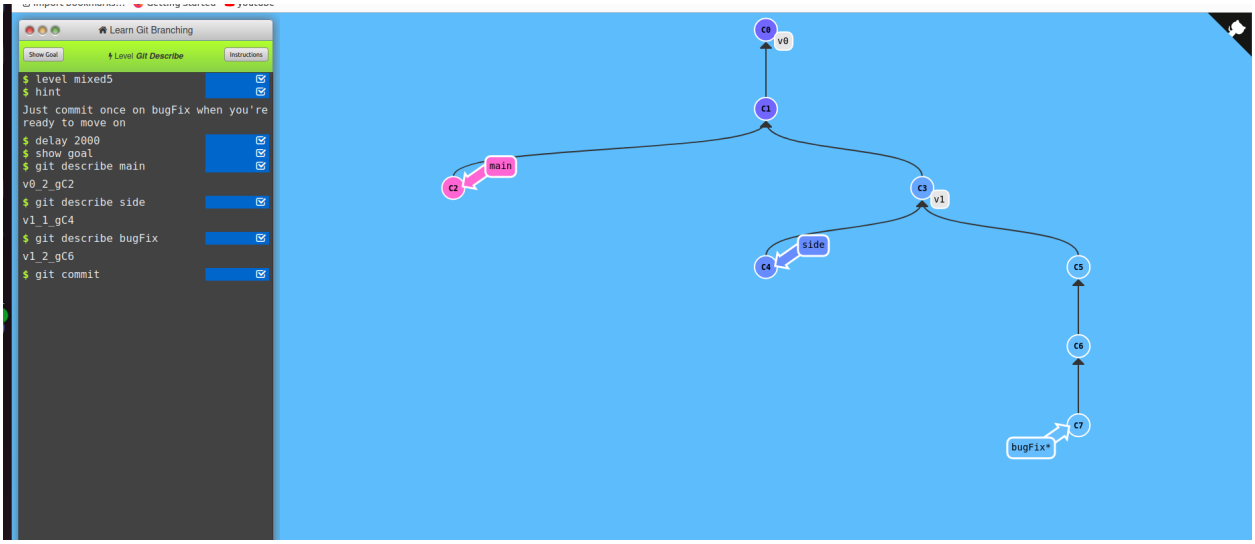
Where `<ref>` is anything git can resolve into a commit. If you don't specify a ref, git just uses where you're checked out right now (`HEAD`).

The output of the command looks like:

`<tag>_<numCommits>_g<hash>`

Where `tag` is the closest ancestor tag in history, `numCommits` is how many commits away that tag is, and `<hash>` is the hash of the commit being described.






```
git describe main
git describe side
git describe bugFix
git commit
```