GIT

Level2

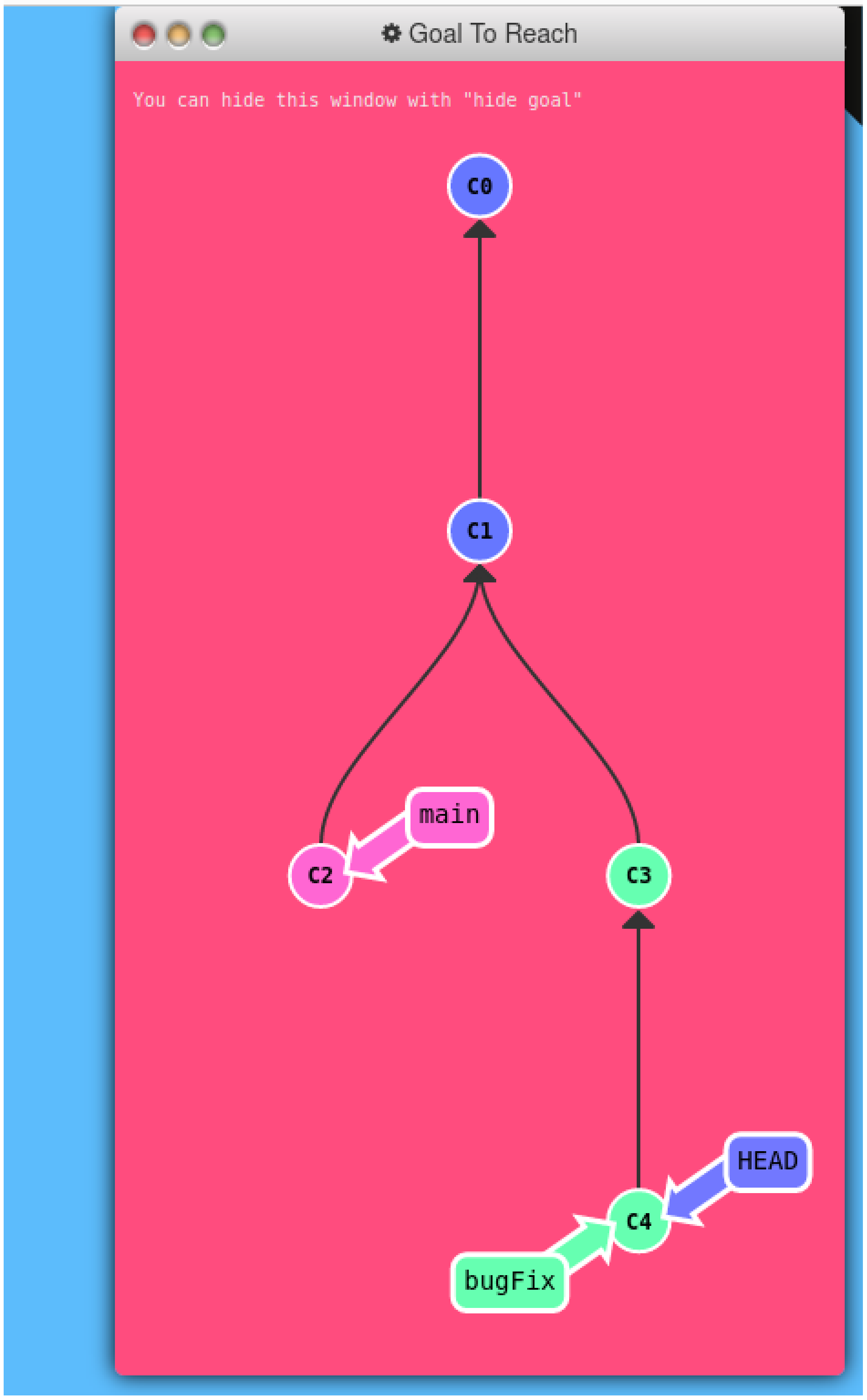
Task 1

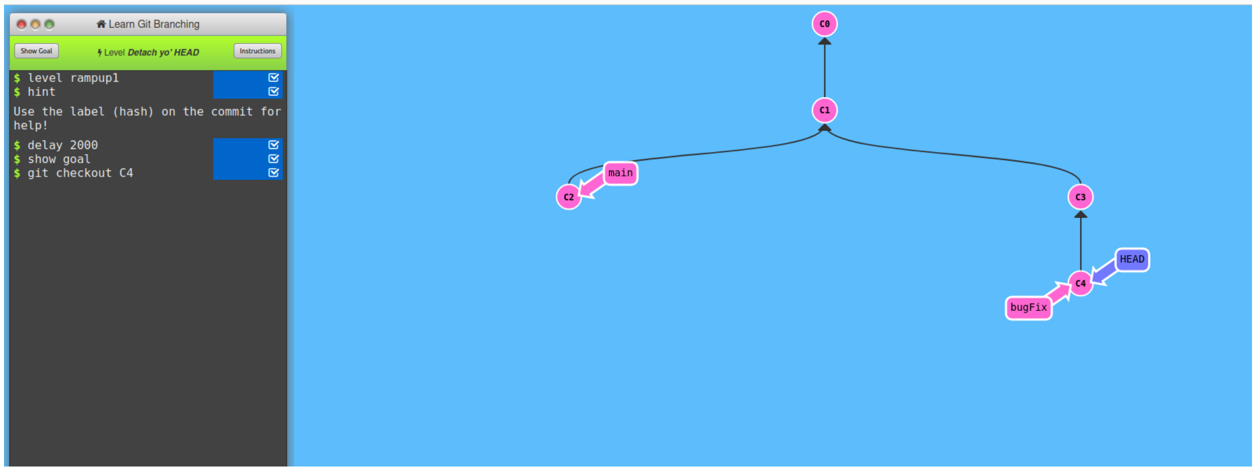
First we have to talk about "HEAD". HEAD is the symbolic name for the currently checked out commit -- it's essentially what commit you're working on top of.

HEAD always points to the most recent commit which is reflected in the working tree. Most git commands which make changes to the working tree will start by changing HEAD.

Normally HEAD points to a branch name (like bugFix). When you commit, the status of bugFix is altered and this change is visible through HEAD.

In this task we have to show the head pointer pointing bugFix branch.


Goal To Reach
You can hide this window with "hide goal"
C0
C1
C2
main
C3
C4
HEAD
bugFix

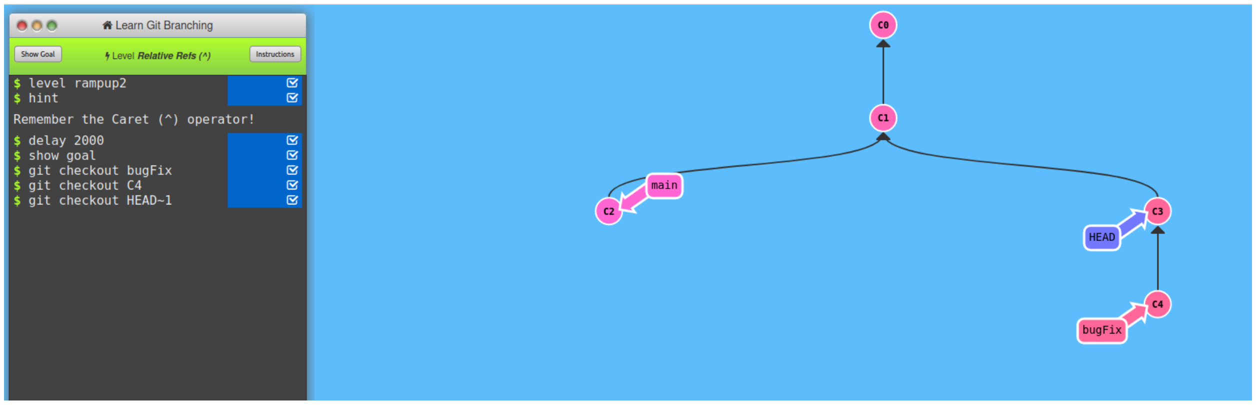


```
$ level rampup1
$ hint
Use the label (hash) on the commit for
help!
$ delay 2000
$ show goal
$ git checkout C4
```

```
git checkout C4
```

Task 2

In this task we have to move the head pointer to 1 step above the bugFix branch on which the head is currently pointing.



```
$ level rampup2
$ hint
Remember the Caret (^) operator!
$ delay 2000
$ show goal
$ git checkout bugFix
$ git checkout C4
$ git checkout HEAD~1
```

```
git checkout bugFix
git checkout C4
git checkout HEAD~1
```

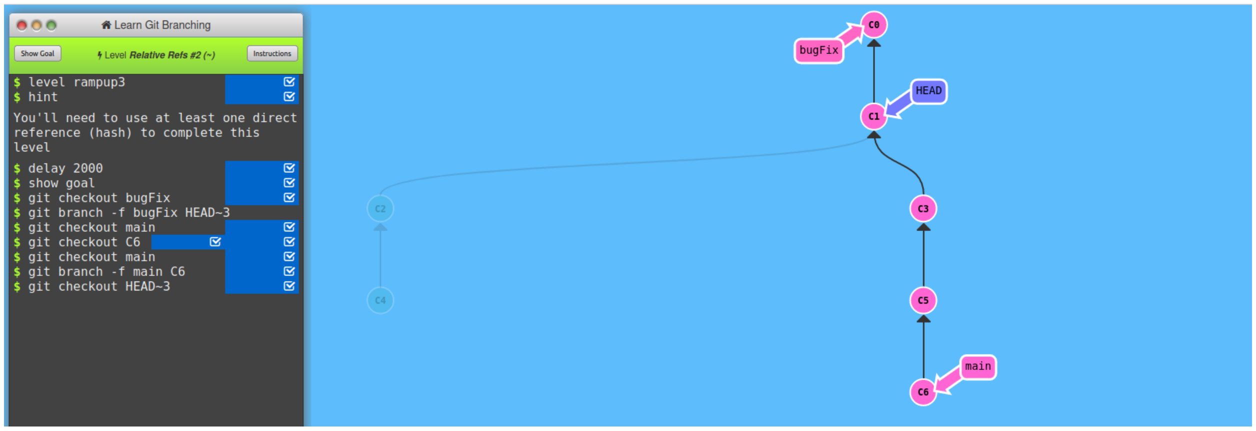
Task 3

~ operator and branch forcing.

We can move the HEAD pointer to the specified commit via ~ operator

The `git branch -f` (or `git branch --force`) command in Git allows you to forcefully reset a branch to point to a specific commit. This can be useful when you want to update a branch to match another commit, even if it means overwriting the existing branch history.

git branch -f branch-name commit-hash



```
$ level rampup3
$ hint
You'll need to use at least one direct
reference (hash) to complete this
level
$ delay 2000
$ show goal
$ git checkout bugFix
$ git branch -f bugFix HEAD~3
$ git checkout main
$ git checkout C6
$ git checkout main
$ git branch -f main C6
$ git checkout HEAD~3
```

```
git chekout bugFix
git branch -f bugfix HEAD~3
git checkout main
git branch -f main C6
git checkout HEAD~3
```

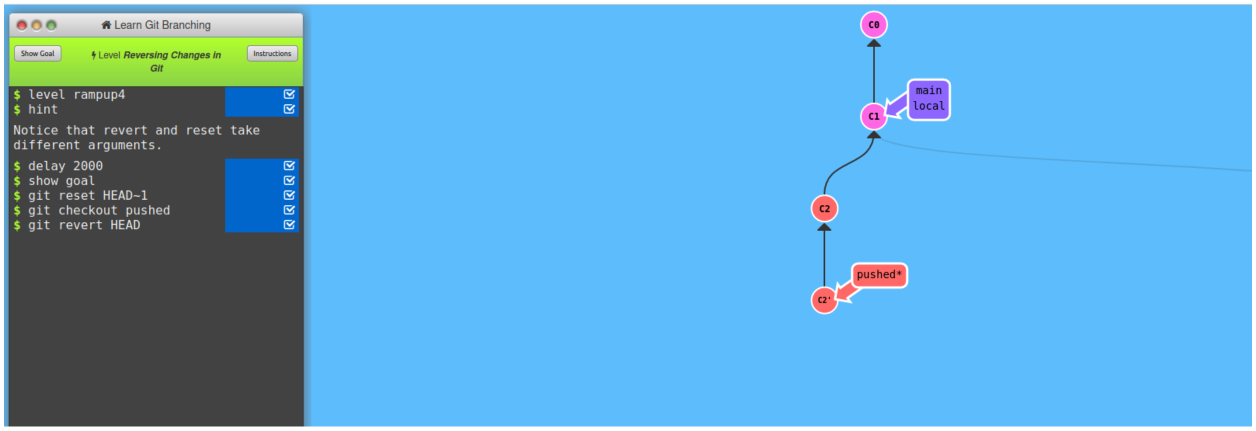
Task 4

`git reset` reverses changes by moving a branch reference backwards in time to an older commit. In this sense you can think of it as "rewriting history;"
`git reset` will move a branch backwards as if the commit had never been made in the first place.

While resetting works great for local branches on your own machine, its method of "rewriting history" doesn't work for remote branches that others are using.

In order to reverse changes and *share* those reversed changes with others, we need to use `git revert`.

In this task we have to use reverse and revert command.



```
$ level rampup4
$ hint
Notice that revert and reset take
different arguments.
$ delay 2000
$ show goal
$ git reset HEAD~1
$ git checkout pushed
$ git revert HEAD
```

```
git reset HEAD~1
git checkout pushed
git revert HEAD
```