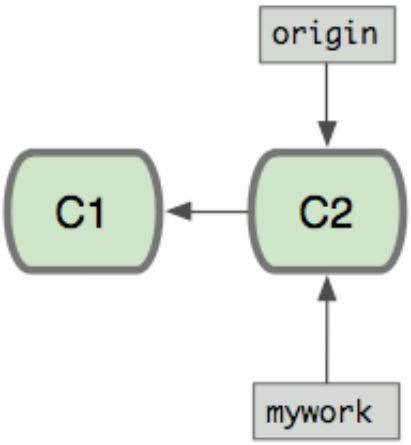
git rebase 用法

假设当前你在基于远程分支”origin”创建了一个”mywork” branch

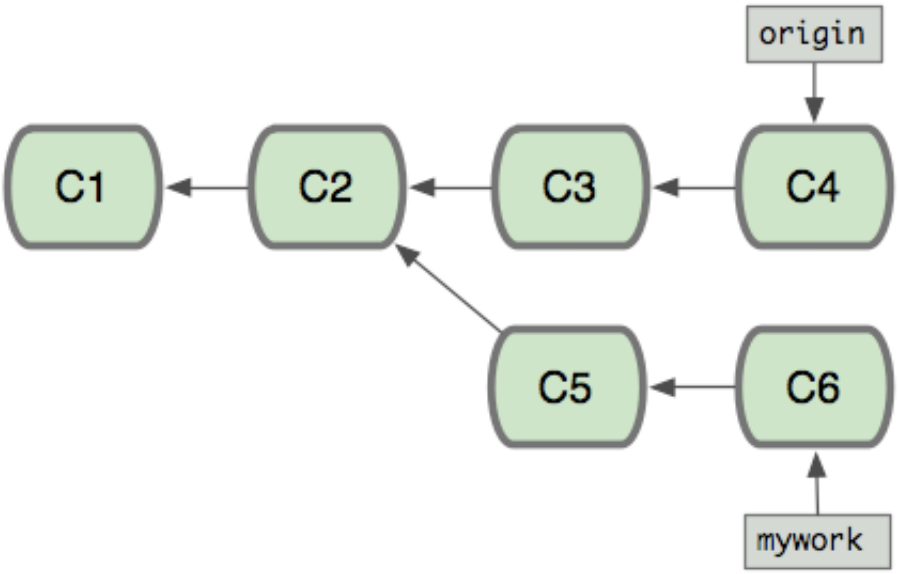
```
git checkout -b mywork origin
```



然后，我们在mywork分支上做了一些修改，并做了提交

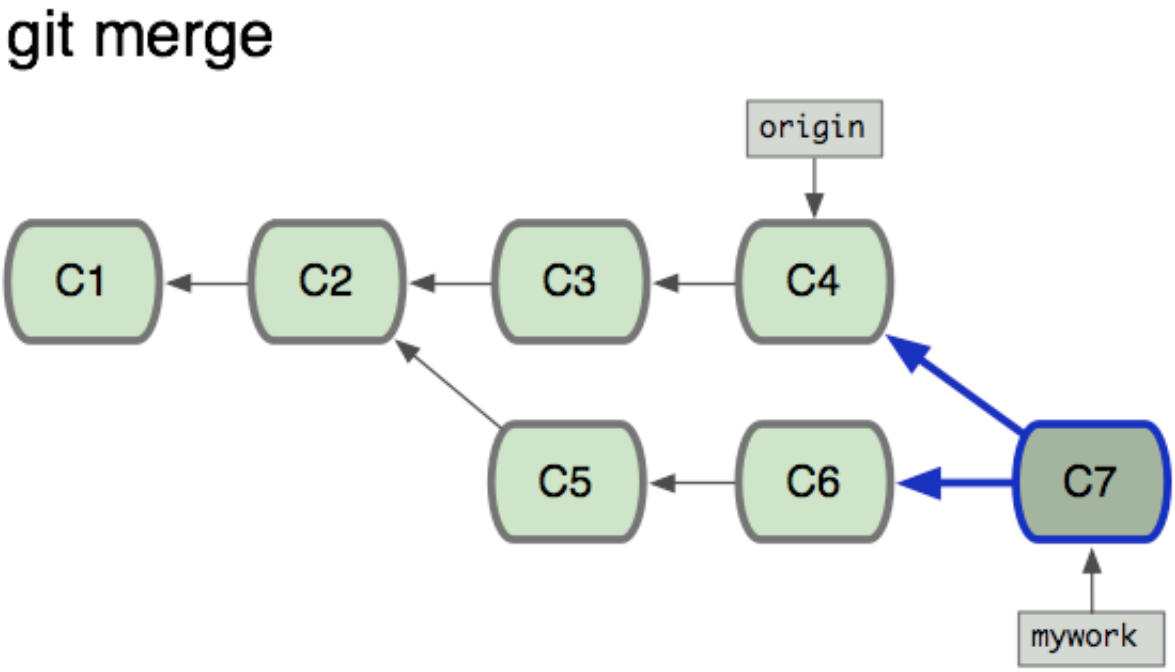
```
vim file
git add file
git commit -m “revised file”
```

如果此时，有人在”origin”分支上做了一些修改并且提交，这样mywork分支和origin分支分别各自前进，所以就会出现“分叉”的情况



这时，你可以用git pull命令把“origin”分支上的修改拉下来并且和你的修改合并，结果看起来就像一个新的“合

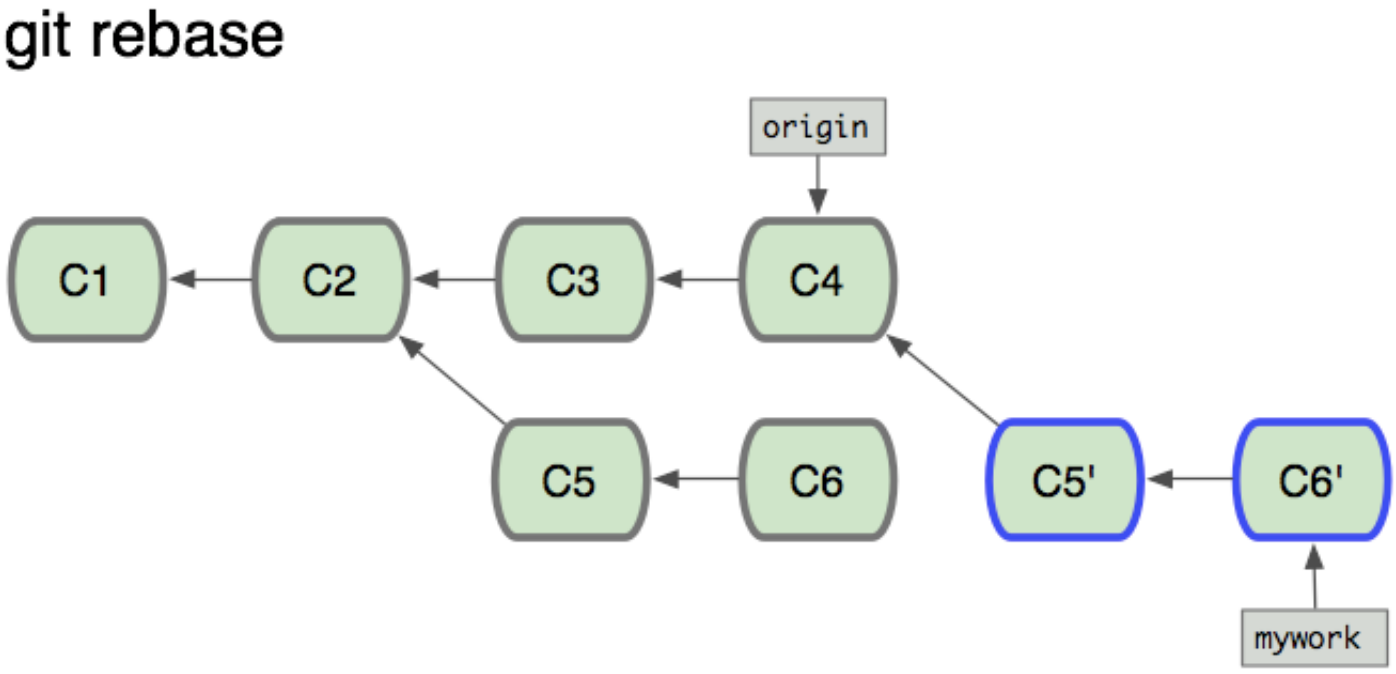
并的提交”(merge commit)：



但是，如果你想要让“mywork”分支的log看起来像没有经过任何合并一样，你可以用git rebase:

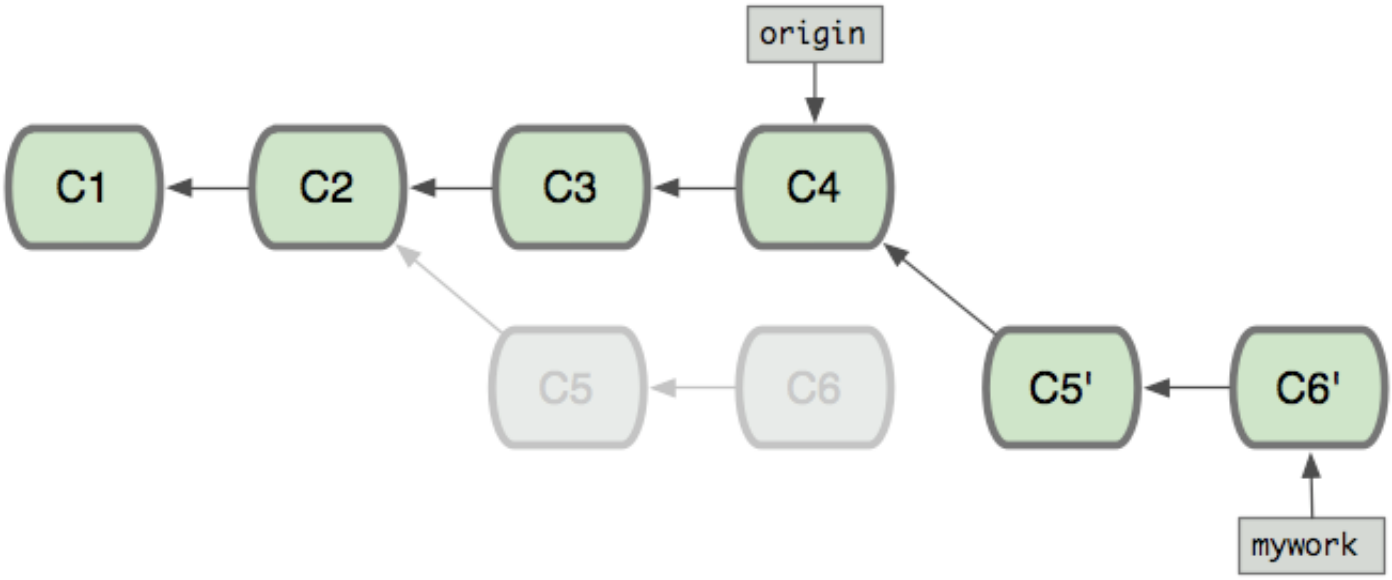
```
git checkout mywork
git rebase origin
```

这些命令会把你的”mywork”分支里的每个commit取消掉，并且把他们临时保存为patch(这些patch放在.git/rebase目录里)，然后把mywork 分支更新到最新的”origin”分支，最后把保存的这些分支patch应用到“mywork”分支上。

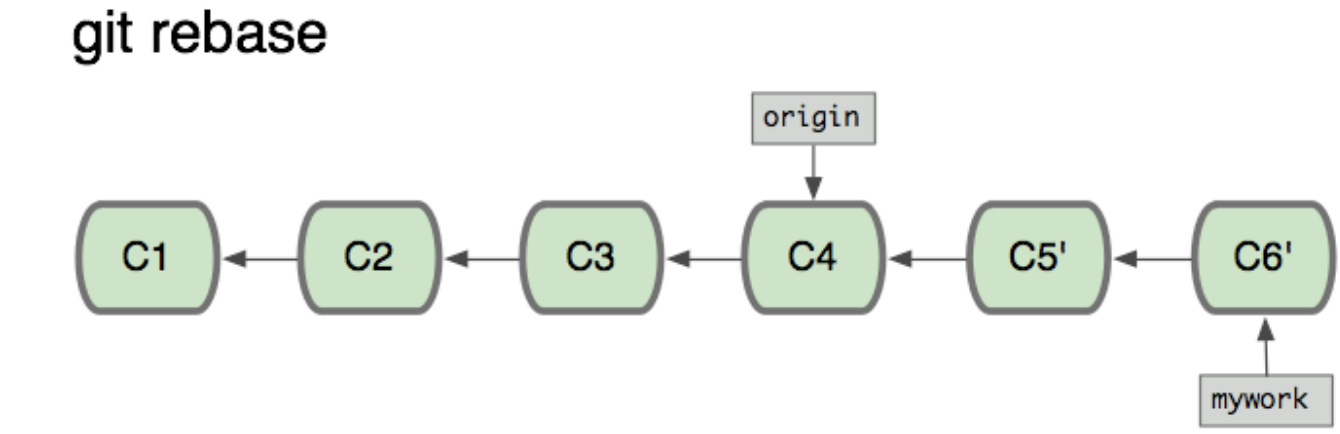


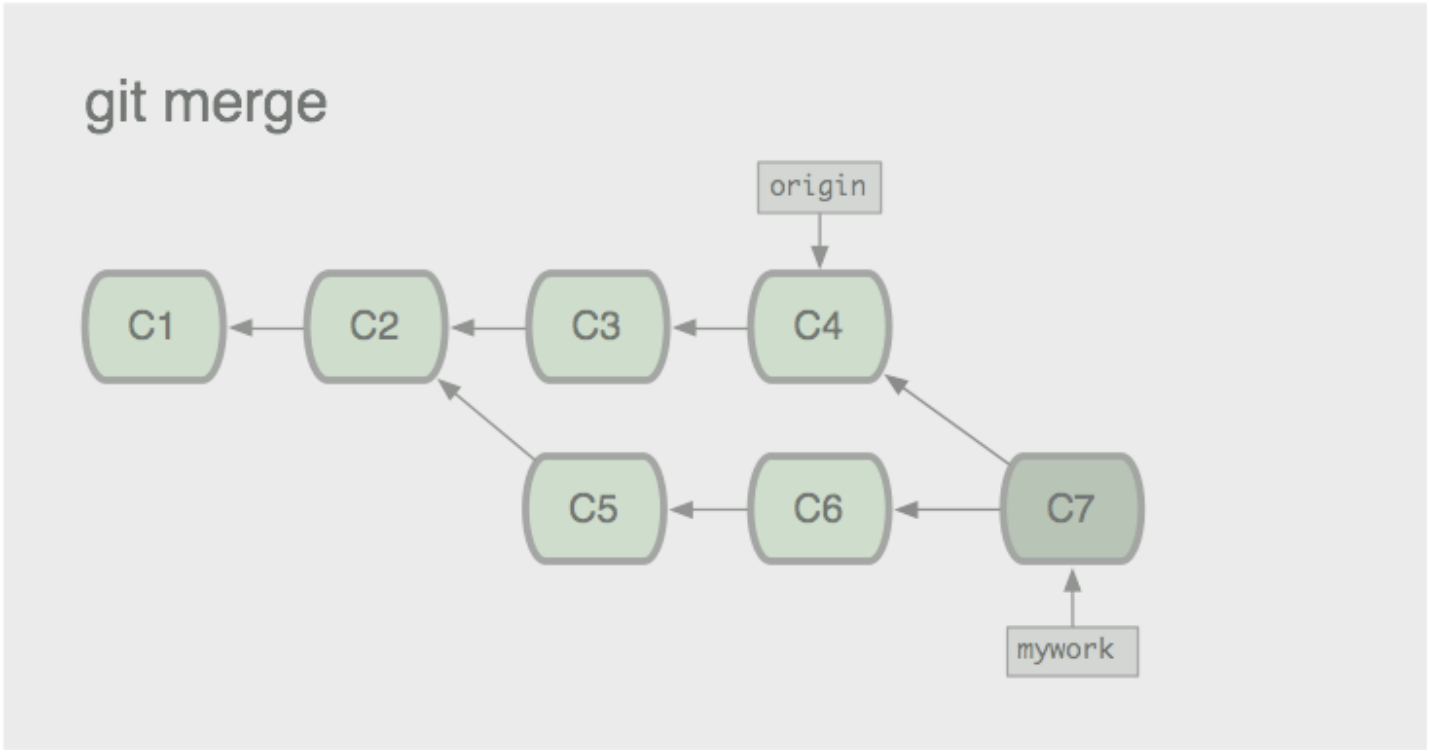
当“mywork”分支更新后，它会指向这些新创建的commit,而那些老的commit会被丢弃。如果运行垃圾收集命

令，这些被丢弃的提交就会被删除，详情查看git gc



现在我们可以看一下用merge和用rebase所产生的历史的区别:





在rebase的过程中，也许会出现冲突(conflict). 在这种情况，Git会停止rebase并会让你去解决 冲突；在解决完冲突后，用"git-add"命令去更新这些内容的索引(index), 然后，你无需执行 git-commit,只要执行:

```
git rebase --continue
```

这样git会继续应用(apply)余下的补丁。

在任何时候，你可以用--abort参数来终止rebase的行动，并且"mywork" 分支会回到rebase开始前的状态。

```
git rebase --abort
```