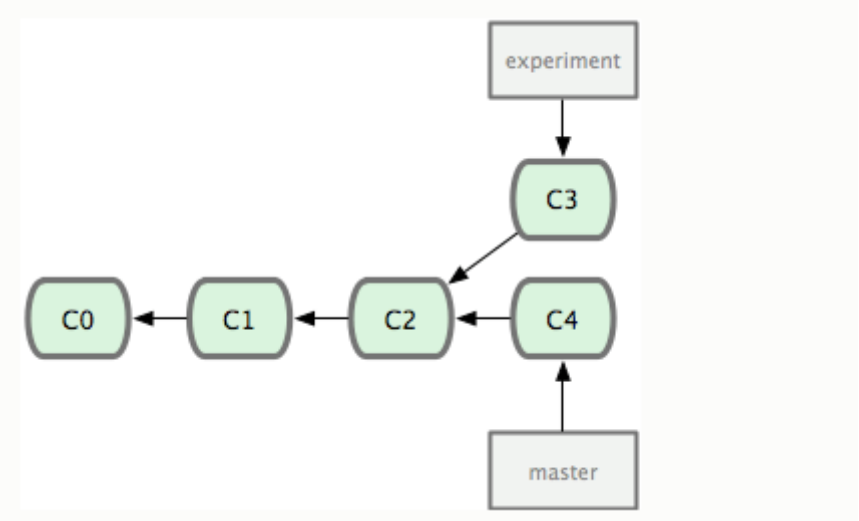
# 四：git 分支的变基

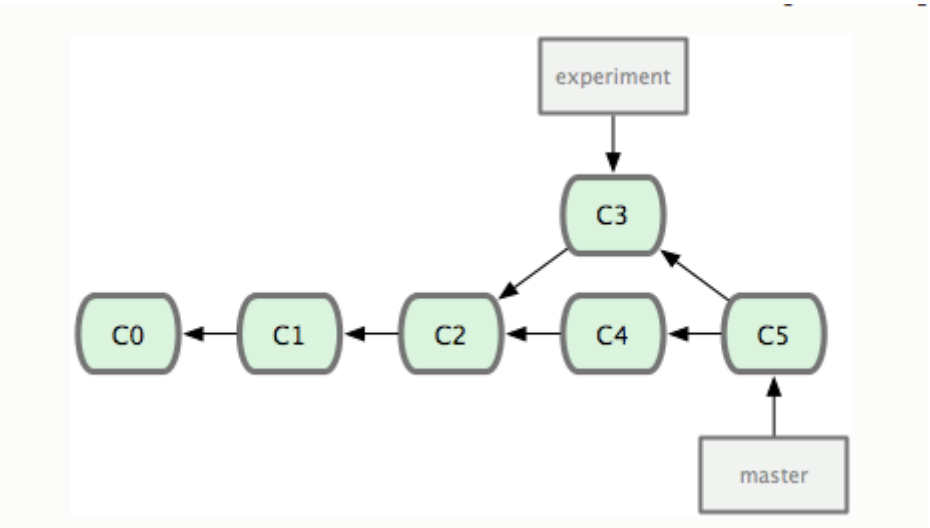
把一个分支的修改整合到另一个分支的办法有两种： merge 和 rabase。 下面介绍什么是变基、如何使用变基、变基有什么作用在什么情况下使用。

- **基本的变基操作**

加入开发过程中分叉到两个分支，且又各自提交了更新。




最容易的整合分支的方法是 merge 命令，它会把两个分支最新的快照 （C3 和 C4）以及最新的共同祖先（C2）进行三方合并，合并的结果是产生一个新的提交对象(C5),如下图：

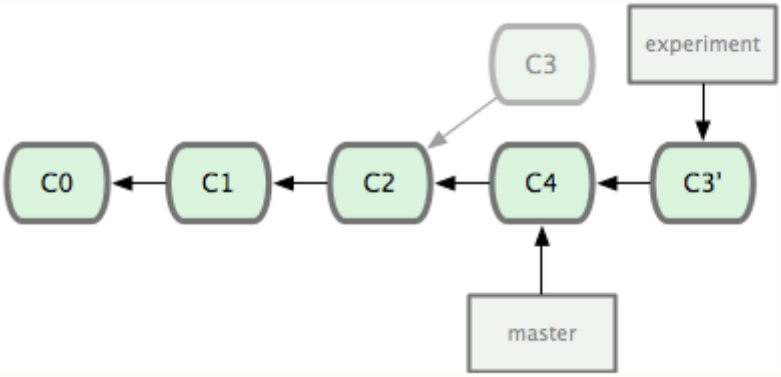



其实，还有另外一个操作，你可以把在 C3 产生的变化补丁在 C4的基础上重新打一遍，在git里这种操作叫做变基（rebase）。有了 rebase 命令，就可以把一个分支里提交的改变移到另一个分支里重放一遍。运行：

```
$ git checkout experiment
$ git rebase master
```

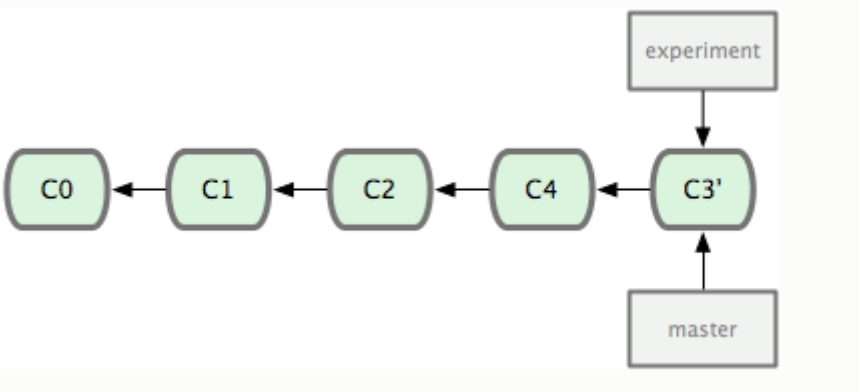
```
First, rewinding head to replay your work on top of it...
Applying: added staged command
```

它的原理是回到两个分支最近的共同祖先，根据当前分支（也就是当前要进行变基的分支 experiment ）的历次提交对象（这里只有一个 C3）,生成一系列补丁文件，然后以基底分支（也就是master分支）的最后一个提交对象（C4）为新的出发点，逐个应用之前准备好的补丁文件，最后会合并形成提交对象（C3'）,从而改写 experiment 的提交历史，使它成为 master 分支的直接下游，如下图：




现在回到master 分支，进行一次快进合并

```
$ git checkout master
$ git merge experiment
```

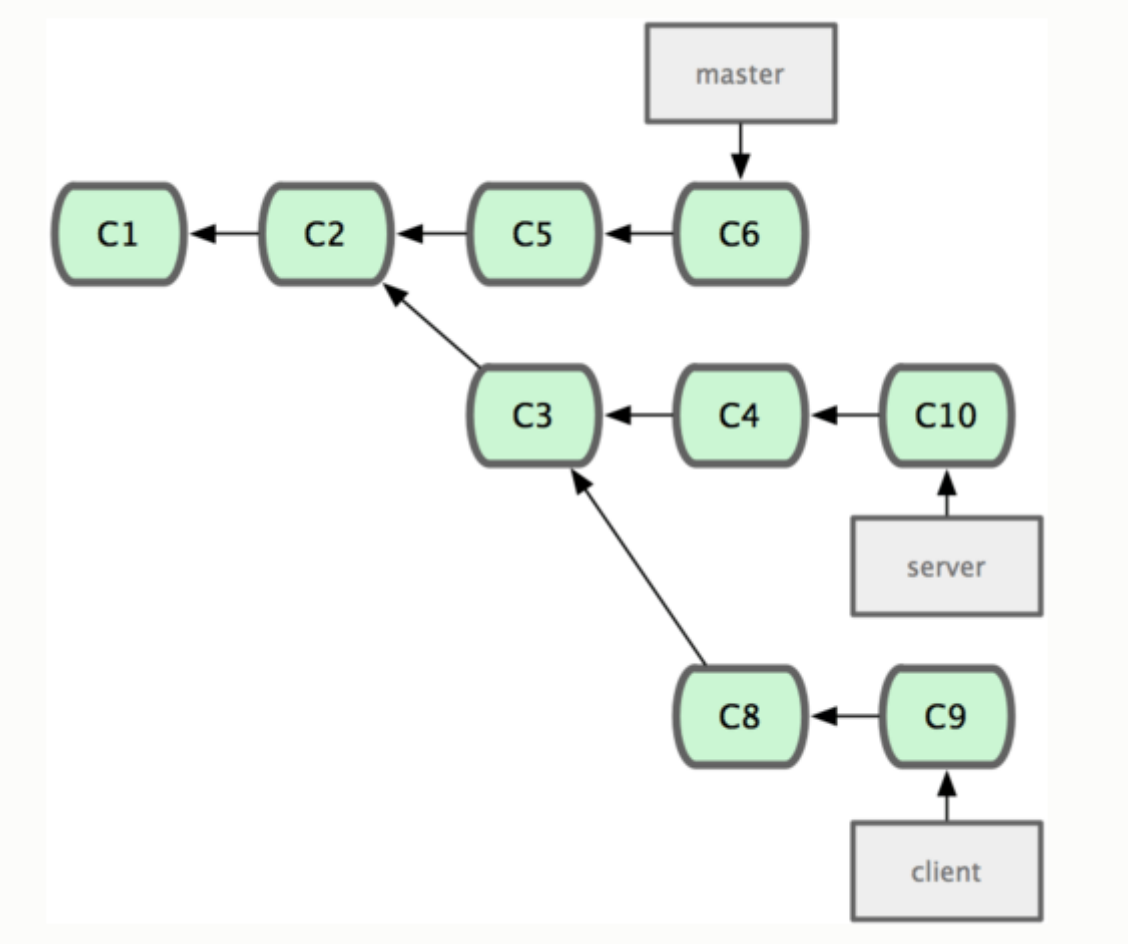



现在的C3'对应的快照其实是和普通三方合并 C5 的快照一模一样的，而使用变基能产生一个更为整洁的提交历史，一个变基过的提交记录，仿佛所有修改都是在一根线上先后进行的，尽管实际它们原本是同步发生的。

一般我们使用变基的目的，是想要得到一个能在远程分支上干净应用的补丁 — 比如某些项目你不是维护者，但想帮点忙的话，最好用变基：先在自己的一个分支里进行开发，当准备向主项目提交补丁的时候，根据最新的 origin/master 进行一次变基操作然后再提交，这样维护者就不需要做任何整合工作（译注：实际上是把解决分支补丁同最新主干代码之间冲突的责任，化转为由提交补丁的人来解决。），只需根据你提供的仓库地址作一次快进合并，或者直接采纳你提交的补丁。

- **有趣的变基**

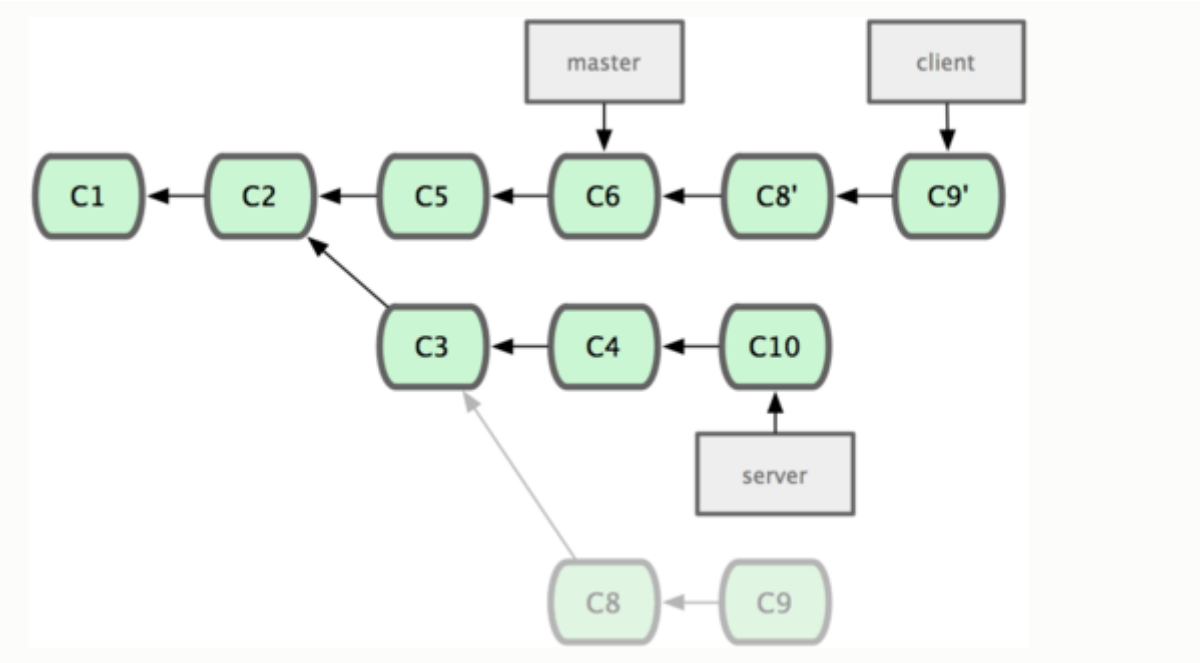
变基也可以放到其他分支进行，并不一定非得根据分化之前的分支。如下图：我们为了给服务器端代码添加功能创建了特性分支 server，然后提交了 C3 C4。然后又在C3 的位置增加了一个client 分支来对客户端代码进行一些相应修改，所以提交了C8 C9.最后又回到 server 分支提交了C10.




假设接下来的一次软件发布中，我们决定把客户端的修改并到主线中，而暂缓并入服务器端软件的修改。这个时候我们就可以把基于 client 分支的改变而非server分支的改变（即 C8 C9）,跳过 server 分支 直接放到 master 分支中重演一遍，这时需要使用 git rebase 的 --onto 选项指定新的基底分支 master

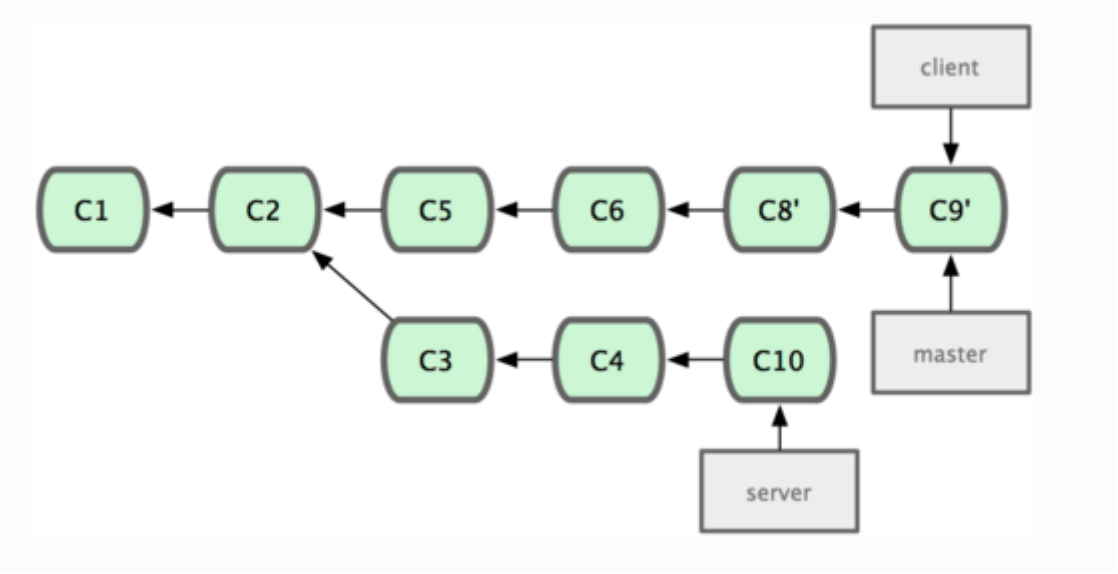
```
$ git rebase --onto master server client
```

这个命令是说： 取出 client分支，找到 client 分支和server分支的共同祖先之后的变化（即 C8,C9）然后把他们在master重演一遍。




现在可以快进 master 分支了

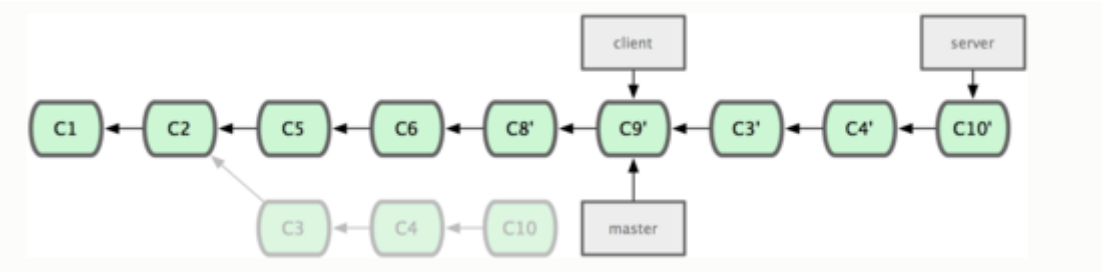
```
$ git checkout master
$ git merge client
```




现在我们决定把 server 分支的变化也包含进来，我们可以直接把 server 分支变基到 master, 而不用像上面的步骤先手工切换到 server 分支再执行变基操作。 直接运行下面命令 ， 该命令会先取出特性分支server，然后在master上重演。

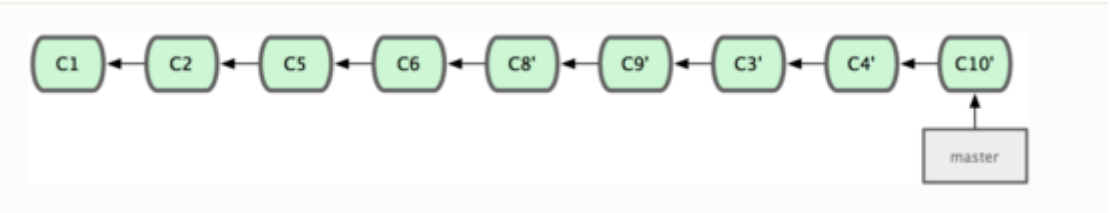
```
git rebase [主分支] [特性分支]
$ git rebase master server
```

于是 server 的进度应用到了 master 的基础上




然后快进主干分支 master

现在 client 和 server 分支的变化都已经集成到了主干分支，可以删掉它们了，最终我们的提交历史如下图：

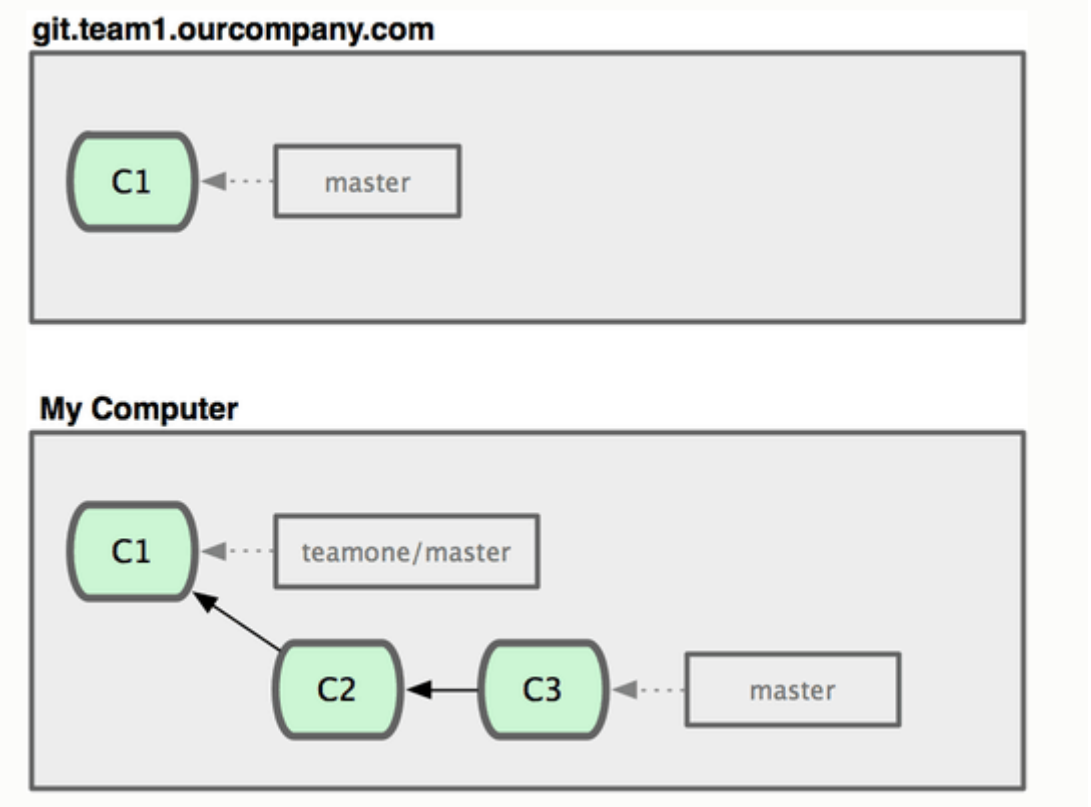



- **变基的风险**

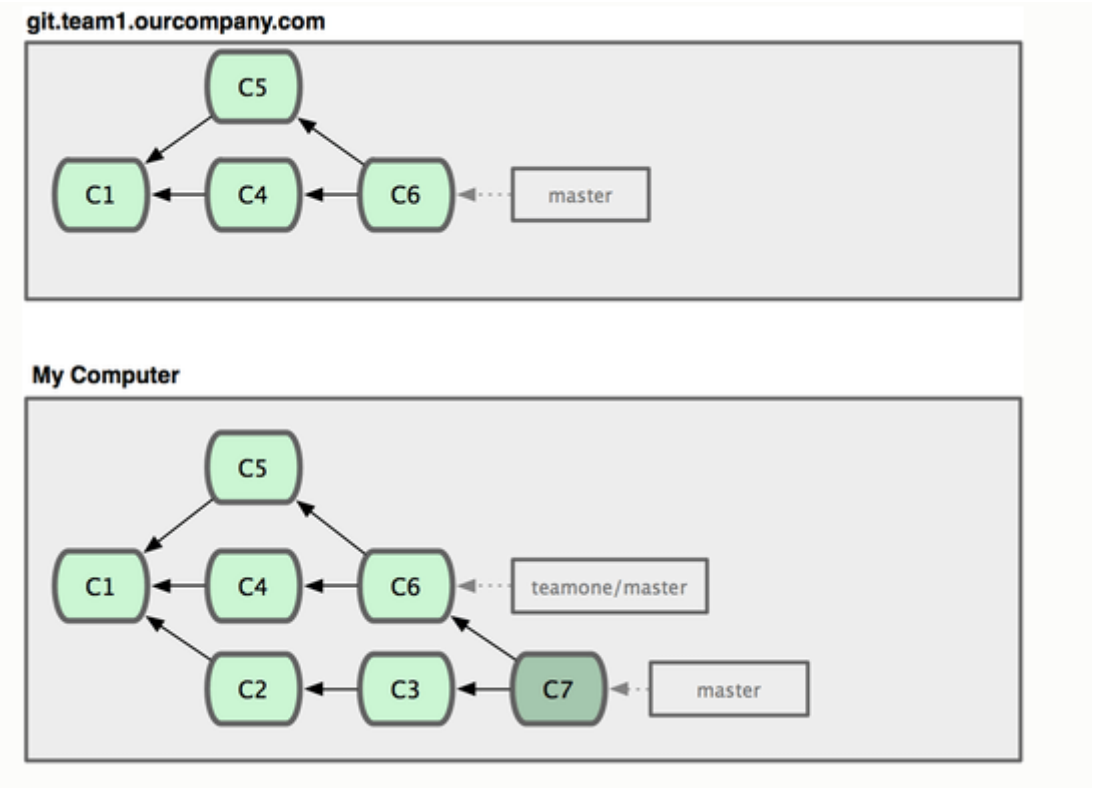
变基要遵循一条准则： **一旦分支中的提交对象发布到公共仓库，就千万不要对该分支进行变基操作。**

在进行变基的时候，实际上抛弃了一些现存的提交对象而创造了一些类似但不同的新的提交对象。如果你把原来的提交对象发布出去，并且其他人下载更新后在其基础上开展工作，而后你又用 git rebase 抛弃这些提交对象，把重演后的提交对象发布出去的话，你的合作者就不得不重新合并他们的工作，这样你再次从他们那里获取内容时，提交历史会变得一团糟。

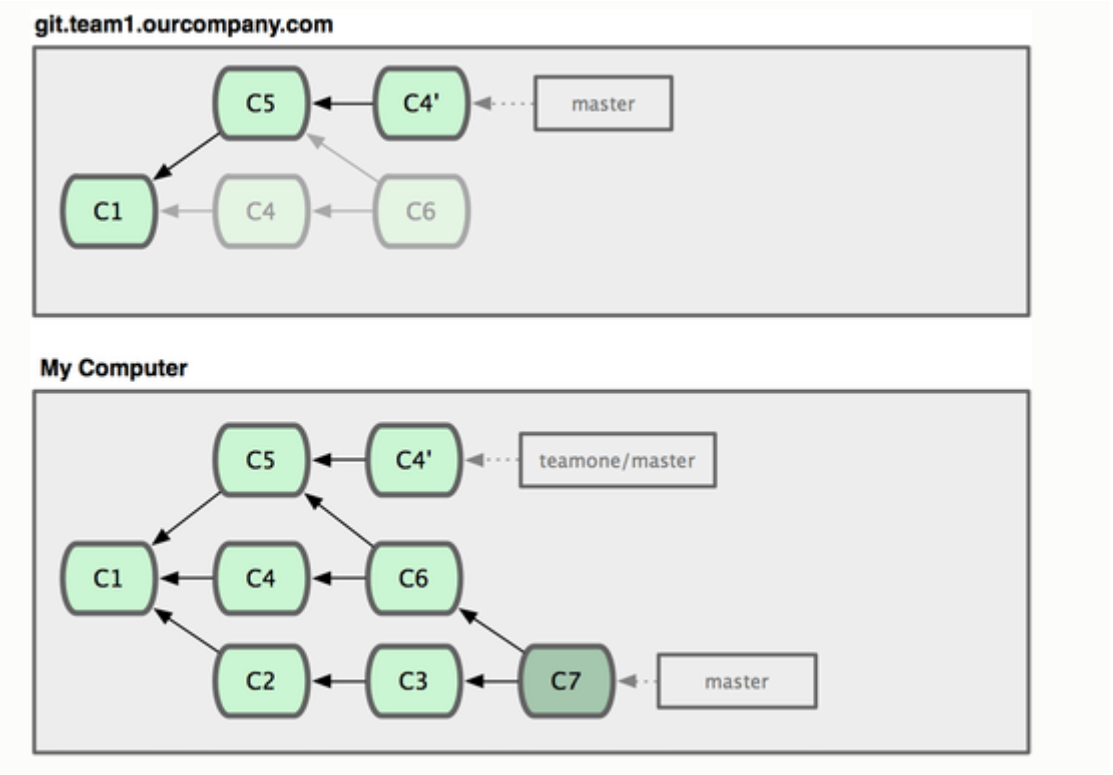
举个例子： 假设你从一个中央服务器克隆并且在它的基础上搞了一些开发，提交历史如下：



现在，某人在 C1 的基础上做了些改变，并合并了他自己的分支得到结果 C6,推送到中央服务器，当你抓取并合并这些数据到你本地开发分支中后，会得到合并结果 C7,历史提交记录如下：

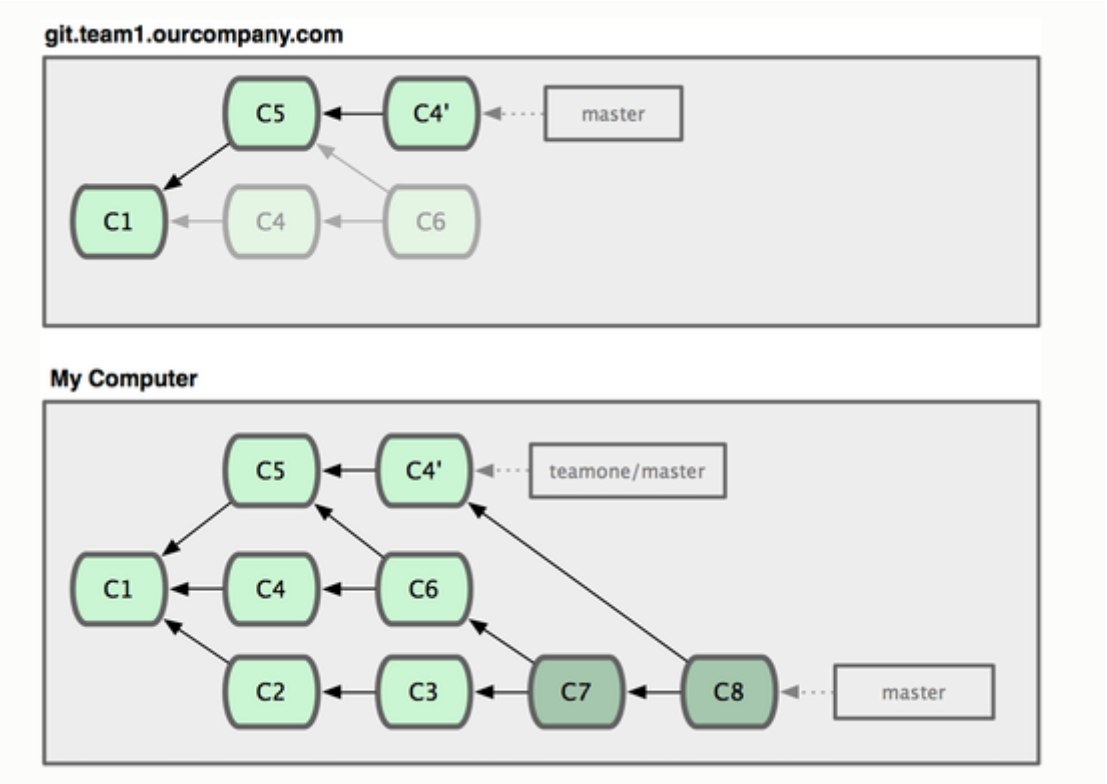



接下来，那个推送 C6上来的人决定使用变基取代之前的合并操作，然后又用 git push --force 覆盖了服务器上的历史，得到 C4'。而之后当你从服务器上下载最新的提交后，会得到下面这种：




下载更新后需要合并，但是此时变基产生的提交对象 C4' 的 SHA-1 校验和和之前的 C4 完全不同，所以Git会把它们当做新的提交对象处理，而实际上此刻你的 C7 里早已包含了C4 的修改内容，于是

合并操作会把 C7 和 C4'合并为 C8




此时你的提交历史里就会同时包含 C4 和 C4'，两者有着不同的 SHA-1 校验值，如果用 git log 会看到两个提交拥有相同的作者日期和说明。而更糟的是当你把这样的历史推送到服务器，会再次吧这些变基后的提交引入服务器，影响其他人。

- **git fetch 和 git pull 的区别**

git 从远程分支获取最新的版本到本地有这样两个命令：

- git fetch: 相当于是从远程获取最新版本到本地，不会自动merge

```
$ git fetch origin master
$ git log -p master..origin/master
$ git merge origin/master
```

解释下以上三个命令：

- 第一个命令的全写应该是 `git fetch [远程仓库名] [远程分支名]:[本地分支名]` 上面这条命令从远程的 origin 的master 分支下载最新的版本到 origin/master 分支上。
- 然后比较本地master分支和origin/master 分支的差别
- 最后进行合并

以上过程可以用更清晰的过程

```
$ git fetch origin master:test
$ git diff test
$ git merge test
```

从远程获取最新的版本到本地的test分支上，之后再进行比较合并。

- git pull 相当于是从远程获取最新版本并merge 到本地 完整命令形式为 `git pull [远程仓库名] [远程分支名]:[本地分支名]`

```
$ git pull origin master
```

上述命令相当于 git fetch 和 git merge 实际使用中，git fetch 更安全些，因为在merge前，我们可以查看更新情况，决定是否合并。

> git pull 不带任何参数的时候，是拉取的当前分支(执行过 git push -u )

- **git push 常见用法**

git push 的一般形式为 git push [远程仓库名] [本地分支名]:[远程分支名] 例如： git push origin master:refs/for/master 是将本地的master 分支推送到远程仓库 origin 上对应的 master 分支。

- git push origin master

如果远程分支被省略，则表示将本地分支推送到与之存在追踪关系的远程分支（通常两者同名），如果该远程分支不存在，则会被创建。

- git push origin :refs/for/master

如果省略本地分支，相当于推送了一个空的分支到远程，也就是删除远程分支，等同于 git push origin --delete master

- git push origin

如果当前分支与远程分支存在追踪关系，则本地分支和远程分支都可以省略，将当前分支推送到 origin 主机的对应分支。

- git push

如果当前分支只有一个远程分支，则主机名也可以省略，可以使用 git branch -r 查看远程分支名。

- git push 其他命令
    - git push -u orgin master 如果当前分支与多个分支存在追踪关系，则可以使用 -u 指定一个默认主机，这样可以直接使用 git push ，默认只推送当前分支，这种叫做 simple 方式，还有一种叫做 matching 方式，会推送所有对应的远程分支的本地分支。 如果想更改设置，可以使用git config命令。git config --global push.default matching OR git config --global push.default simple；可以使用git config -l 查看配置

    - git push --alll origin 这种情况就是不管是否存在对应的远程分支，将本地的所有分支都推送到远程主机。
    - git push --force origin （git push 的时候需要本地先 git pull 更新到和服务器一致，如果本地版本库比远程的低则会提示git pull 更新，可以使用这个命令强制提交）
    - git push origin --tags 推送标签

- refs/for // refs/for 的意义在于我们提交代码到服务器之后是需要经过code review 之后才能进行merge的，而refs/heads 不需要

- **git pull 和 git pull --rebase 的区别**