**git merge**是用来合并两个分支的。

git merge b

      # 将b分支合并到当前分支

同样 git rebase b，也是把 b分支合并到当前分支

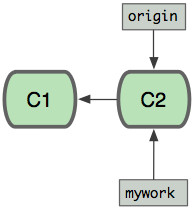
-----------------------------------

他们的 **原理 如下**：

假设你现在基于远程分支"origin"，创建一个叫"mywork"的分支。

$ git checkout -b mywork origin

假设远程分支"origin"已经有了2个提交，如图



现在我们在这个分支做一些修改，然后生成两个提交(commit).

$ vi file.txt

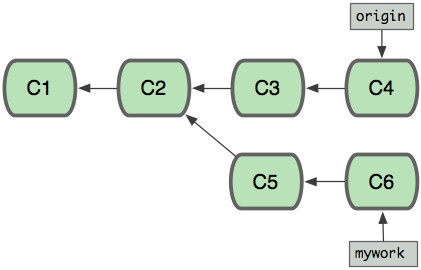
$ git commit

$ vi otherfile.txt

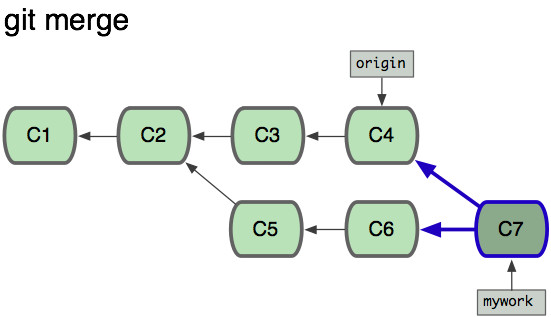
$ git commit

...

但是与此同时，有些人也在"origin"分支上做了一些修改并且做了提交了. 这就意味着"origin"和"mywork"这两个分支各自"前进"了，它们之间"分叉"了。



在这里，你可以用"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):

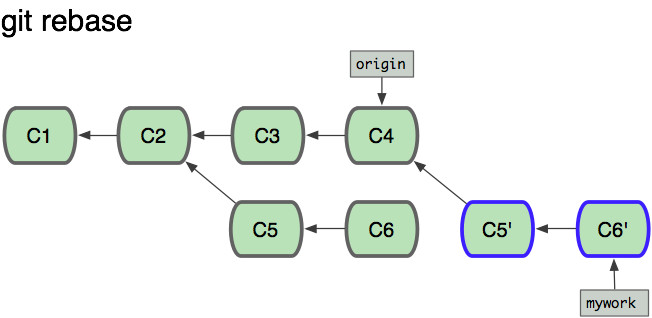


但是，如果你想让"mywork"分支历史看起来像没有经过任何合并一样，你也许可以用 git rebase:

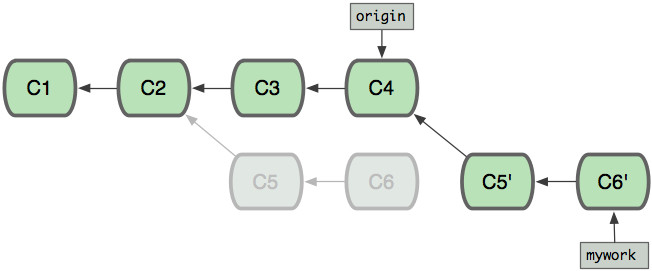
$ git checkout mywork

$ git rebase origin

这些命令会把你的"mywork"分支里的每个提交(commit)取消掉，并且把它们临时 保存为补丁(patch)(这些补丁放到".git/rebase"目录中),然后把"mywork"分支更新 为最新的"origin"分支，最后把保存的这些补丁应用到"mywork"分支上。



当'mywork'分支更新之后，它会指向这些新创建的提交(commit),而那些老的提交会被丢弃。 如果运行垃圾收集命令(pruning garbage collection), 这些被丢弃的提交就会删除. （请查看 git gc)



二、解决冲突

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

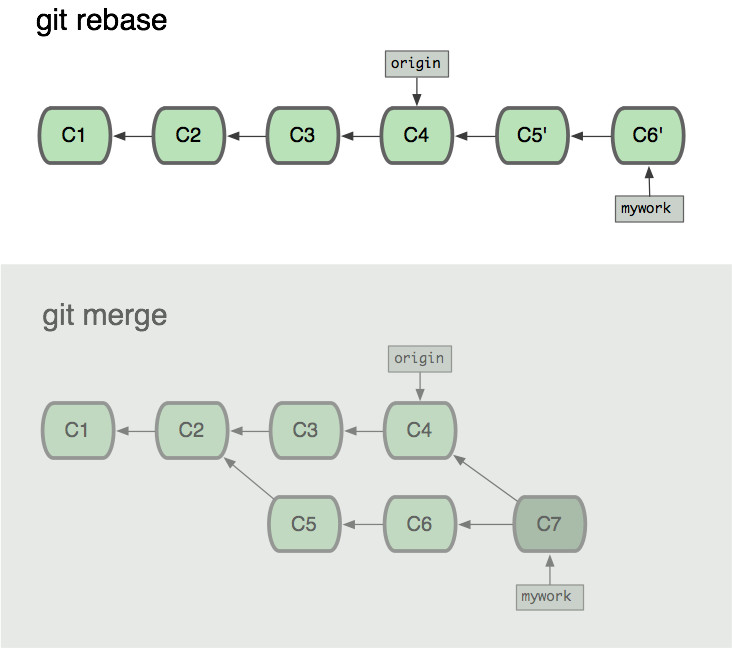
这样git会继续应用(apply)余下的补丁。

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

三、git rebase和git merge的区别

现在我们可以看一下用合并(merge)和用rebase所产生的历史的区别：



当我们使用Git log来参看commit时，其commit的顺序也有所不同。

假设C3提交于9:00AM,C5提交于10:00AM,C4提交于11:00AM，C6提交于12:00AM,

对于使用git merge来合并所看到的commit的顺序（从新到旧）是：C7 ,C6,C4,C5,C3,C2,C1

对于使用git rebase来合并所看到的commit的顺序（从新到旧）是：C7 ,C6‘,C5',C4,C3,C2,C1

 因为C6'提交只是C6提交的克隆，C5'提交只是C5提交的克隆，

从用户的角度看使用git rebase来合并后所看到的commit的顺序（从新到旧）是：C7 ,C6,C5,C4,C3,C2,C1

Git merge会新生成一个commit，但是没有changeid，需要用git commit –amend增加一个changeid

Git rebase 直接指向一个最后一个commit

**内容冲突**

两个用户修改了同一个文件的同一块区域，git会报告内容冲突。我们常见的都是这种，后面的解决办法也主要针对这种冲突。

# 内容冲突的解决办法

## 发现冲突

一般来讲，出现冲突时都会有“CONFLICT”字样：

$ git pull  
Auto-merging **test**.txt  
CONFLICT (content): Merge conflict **in** **test**.txt  
Automatic merge failed; fix conflicts and **then** commit the result.

**编辑冲突的方法**

**直接编辑冲突文件**

冲突产生后，文件系统中冲突了的文件（这里是test.txt）里面的内容会显示为类似下面这样：

a123  
<<<<<<< HEAD  
b789  
=======  
b45678910  
>>>>>>> 6853e5ff961e684d3a6c02d4d06183b5ff330dcc  
c

其中：冲突标记<<<<<<< （7个<）与=======之间的内容是我的修改，=======与>>>>>>>之间的内容是别人的修改。

此时，还没有任何其它垃圾文件产生。

最简单的编辑冲突的办法，就是直接编辑冲突了的文件（test.txt），把冲突标记删掉，把冲突解决正确。

git revert 是撤销某次操作，此次操作之前的commit都会被保留

git reset 是撤销某次提交，但是此次之后的修改都会被退回到暂存区

具体一个例子，假设有三个commit， git st:

commit3: add test3.c

commit2: add test2.c

commit1: add test1.c

当执行git revert HEAD~1时， commit2被撤销了

git log可以看到：

commit1：add test1.c

commit3：add test3.c

git status 没有任何变化

如果换做执行git reset --soft(默认) HEAD~1后，运行git log

commit2: add test2.c

commit1: add test1.c

运行git status， 则test3.c处于暂存区，准备提交。

如果换做执行git reset --hard HEAD^1后，

显示：HEAD is now at commit2，运行git log

commit2: add test2.c

commit1: add test1.c

运行git st， 没有任何变化

另外：

git revert 是撤消该commit，作为一个新的commit。

git revert HEAD~1 撤销 上一个commit的操作（HEAD~n指向的commit），作为一个新的commit。使用git status命令，没有任何变化

git reset --soft HEAD~1 HEAD指向HEAD~1，即现在处于上一个commit。git status，会显示被撤销的操作，并且处于暂存区，准备提交

git reset --hard HEAD~1，同上，只不过git status不会显示被撤销的操作，即git status没有任何变化

git reset --mixed HEAD~1默认方式，可以不加 “--mixed”，回到前一个提交，只是git log发生了变化，文件的更改等信息都被保留了，可以用git status查看。不在暂存区

**git commit --amend -m"修改 提交 说明"**

**将本次和上次的修改合在一起，作为一个提交。（之前的提交修改错了就不要用这个命令，错误的提交还会存在的）**

Git 中的tag指向一次commit的id，通常用来给开发分支做一个标记，如标记一个版本号。

**打标签**

git tag -a v1.01 SHA1 -m "Relase version 1.01"

注解：git tag 是打标签的命令，-a 是添加标签，其后要跟新标签号，-m 及后面的字符串是对该标签的注释。

**提交标签到远程仓库**  
git push origin -tags  
注解：就像git push origin master 把本地修改提交到远程仓库一样，-tags可以把本地的打的标签全部提交到远程仓库。  
**删除标签**  
git tag -d v1.01  
注解：-d 表示删除，后面跟要删除的tag名字  
**删除远程标签**  
git push origin :refs/tags/v1.01  
注解：就像git push origin :branch\_1 可以删除远程仓库的分支branch\_1一样， 冒号前为空表示删除远程仓库的tag。  
**查看标签**  
git tag  
或者

git tag -l

切换到标签如：v1.0.2

git checkout v1.0.2

一般切换到标签后，可以用git log看到HEAD指向的commit的SHA1值，然后用这个SHA1值新建一个分支，进行修改。

把在分支1中的做的修改，提交到分支2中。

Git reflog 可以查看所有分支的所有操作记录（包括（包括commit和reset的操作），包括已经被删除的commit记录

git cherry-pick用于把另一个本地分支的commit修改应用到当前分支。

**实际问题**

  在本地 master 分支上做了一个commit ( 38361a68138140827b31b72f8bbfd88b3705d77a ) ， 如何把它放到 本地 old\_cc 分支上？

办法之一： 使用 cherry-pick.  根据git 文档：

Apply the changes introduced by some existing commits

就是对已经存在的commit 进行apply (可以理解为再次提交）

**简单用法**：

git cherry-pick <commit id>

例如：

$ git checkout old\_cc

$ git cherry-pick 38361a68

1. 如果顺利，就会正常提交。结果：

Finished one cherry-pick.

# On branch old\_cc

# Your branch is ahead of 'origin/old\_cc' by 3 commits.

2. 如果在cherry-pick 的过程中出现了冲突

Automatic cherry-pick failed.  After resolving the conflicts,

mark the corrected paths with 'git add <paths>' or 'git rm <paths>'

and commit the result with:

        git commit -c 15a2b6c61927e5aed6718de89ad9dafba939a90b

就跟普通的冲突一样，手工解决：

执行git status 看哪些文件出现冲突

$ git status

both modified:      app/models/user.rb

接着手动解决冲突的文件，然后通过git add把改到添加到索引，最后执行git commit提交修改。

$ vim app/models/user.rb

$ git add app/models/user.rb

git commit -c <原commit号>

==================

Git branch <branch-name> SHA1

例如：git branch temp 6cf4841fd911680d124ec7209775493a726f2003

这只是在本地建立的分支，可以进行任意的修改