

(/apps/redirect?  
utm\_source=side-  
banner-click)

# Git笔记与思考四：变基



囧书 (/u/d7847f241ab5) [+ 关注](#)

2017.11.01 15:37\* 字数 1372 阅读 78 评论 0 喜欢 0

(/u/d7847f241ab5)

## 概念

变基(Rebase)也是合代码的一种手段。

变基与合并(Merge)不同的是，他可以修改历史，使用rebase来代替merge合代码的话，得到的历史记录是一条直线提交历史，无分叉，很漂亮。

然而这也是它的缺点，它抹去了分支历史信息，无法追溯。

## 指令

变基指令和合并类似，也是对分支进行操作，所以需要指定一个分支名

```
git rebase 分支名
```

合并时所指定的分支，是被合进来的分支，意思是把别人的东西纳入给自己。而变基的数据流向似是相反，变基指令所指定的分支，是自己将要注入的目的地。变基指令发出时，是把自己接在目标分支的后面，所以变基变的是自己。



## merge和rebase的数据流向

你可能会疑惑，当我要把两个分支合并时，或者变基时，是要站在分支1的角度合分支2，还是反过来？

为此，来做个实验，搞清楚它们的方向。这里我起了个名字叫数据流向，可能起得不恰当，将就着看，意思懂就行，不用执着文字相。

(/apps/redirect?  
utm\_source=side-  
banner-click)

## 公共祖先


创建一个空仓库，默认得到一个master分支，任意放个文件，进行一次初始化提交。

```
git init
touch foo.txt
git add --all
git commit -m "init"
```

然后再创建一个分支feature

```
git branch feature
```



初次提交.png

这时，分支master和分支feature都指向初始提交，这个提交点作为两条分支的公共祖先。

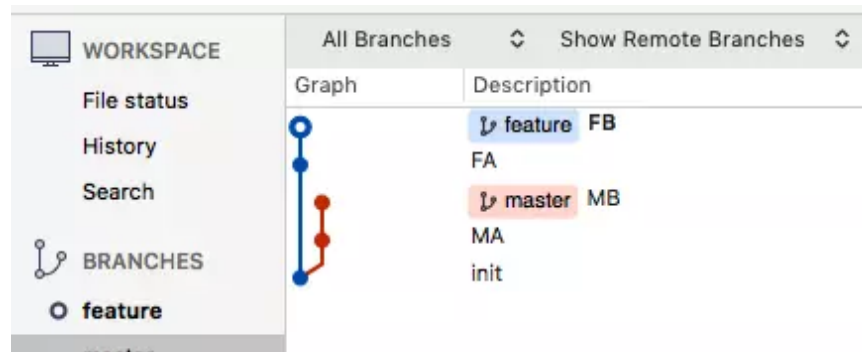
(/apps/redirect?  
utm\_source=side-  
banner-click)

## 并行开发

现在让master分支和feature分支有其各自的修改提交，且提交时间无序，我们来模拟这个情景。

```
### 在master分支提交MA
git checkout master
touch MA.txt
git add --all
git commit -m "MA"
### 在feature分支提交FA
git checkout feature
touch FA.txt
git add --all
git commit -m "FA"
### 在master分支提交MB
git checkout master
touch MB.txt
git add --all
git commit -m "MB"
### 在feature分支提交FB
git checkout feature
touch FB.txt
git add --all
git commit -m "FB"
```





并行开发.png

(/apps/redirect?  
utm\_source=side-  
banner-click)

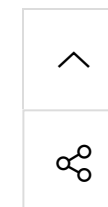
现在前置工作准备好了，可以把项目复制4份，以便进行几种合并结果的比较。

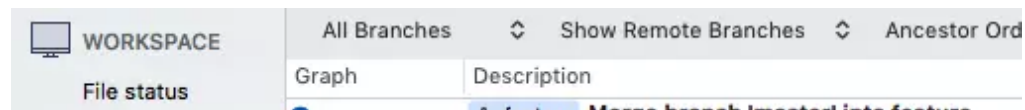
## 四种合代码方式

接下来，我们尝试四种合代码方式，分别是：

- merge master
- merge feature
- rebase master
- rebase feature

对复制的4份项目，分别执行指令，得到如下结果：

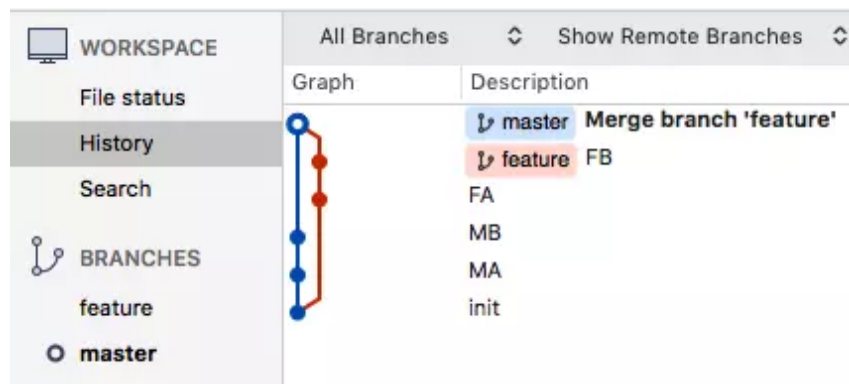




merge master.png

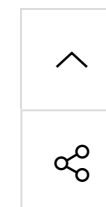
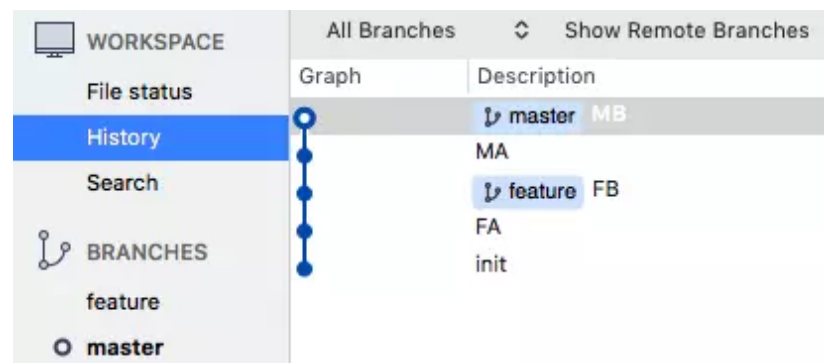
`git merge master` : 在feature分支上发出合并指令, 这样会把master分支的提交合到feature自己身上, 然后再创建一次合并提交。

(/apps/redirect?  
utm\_source=side-  
banner-click)



merge feature.png

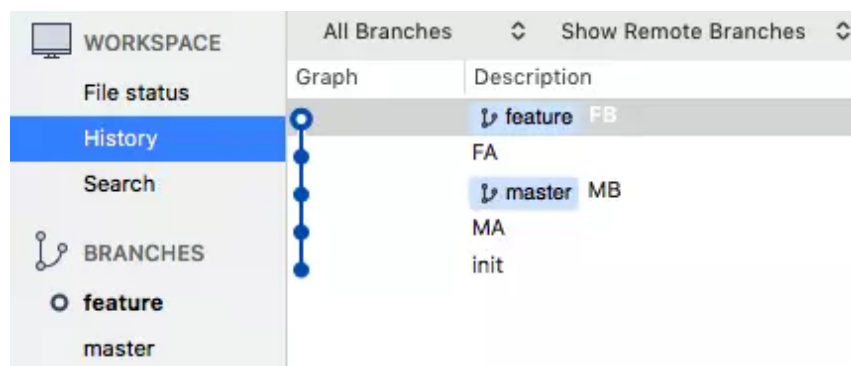
`git merge feature` : 在master分支上发出合并指令, 这样会把feature分支的提交合到master自己身上, 然后再创建一次合并提交。



rebase feature.png

`git rebase feature`：在master分支上发出变基指令，这样会把master分支异于feature分支的提交接在feature之后。

(/apps/redirect?  
utm\_source=side-  
banner-click)



rebase master.png

`git rebase master`：在feature分支上发出变基指令，这样会把feature分支异于master分支的提交接在master之后。

比较几种合代码的情况，如果要进行merge操作的话，最好是在主分支上执行merge，把次分支的代码合进来；如果要进行rebase操作的话，最好是在次分支上进行，把自己变基合入主分支。

## 变基的工作原理

变基其实是复制要被变基的分支上的提交，然后在别的分支上把提交依次重演出来。注意这是复制，而不是移动。也就是说，旧有分支的提交还是可以通过hash值找回来的，在没被 gc 清理的前提下。

由于变基的原理是复制，这导致产生新的提交。在上一小节的实验中，比如 `rebase master` 操作，从现实时间上来说，MB 提交是迟于 FA 提交中，但变基结果是 FA 在 MB 之后，为什么？

因为变基后的 FA 已经不是原feature分支上的 FA 了，它是在变基过程中新产生的一个复制结果，其提交信息也已经被改变。

再者，变基完成后，从版本库历史上，也已经看不出哪些提交是属于哪个分支的了。可以说，变基修改了历史。

(/apps/redirect?  
utm\_source=side-  
banner-click)

## 变基的冲突

在合并时会遇到的冲突情况，在变基时也一样不能避免。

由于变基过程，是一个一个新复制的提交在另一条分支上重演出来，所以可能会出现多次冲突的情况。

在提交重演的时候，每遇到一次冲突，变基过程就会暂停下来，这时，你需要手动处理冲突的文件，处理完后add到暂存区，然后使用如下指令让变基继续。

```
git rebase --continue
```

当然，如果你有很多个提交在重演时都冲突的话，意味着你需要多做几次continue...

## 移植分支

普通变基的指令是这样的：`git rebase master`

移植分支的指令是这样的：`git rebase master --onto 另一分支名`

怎么理解？

如果不加 `--onto` 选项，其实是把自己移植到了目标分支master上，如果加了 `--onto 分支`



名，就会把原本要接到目标master的提交，拐了个弯，接到了指定的另一个分支那里去。

就这么回事^ ^。

(/apps/redirect?  
utm\_source=side-  
banner-click)

小礼物走一走，来简书关注我

赞赏支持

📖 笔记 (/nb/5867702)

举报文章 © 著作权归作者所有



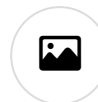
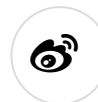
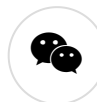
囧书 (/u/d7847f241ab5) ♂

写了 48177 字，被 412 人关注，获得了 537 个喜欢

(/u/d7847f241ab5)

+ 关注

喜欢



更多分享

