日期：2019年3月4日

# git删除远程仓库中的文件

参考：<https://blog.csdn.net/lolgenius/article/details/78527385>

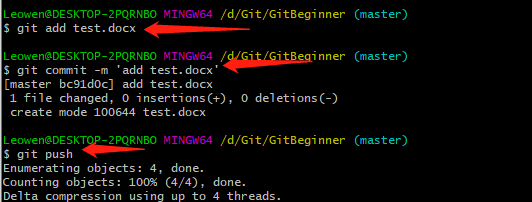
(1) 在本地拉取远程分支 git pull origin master

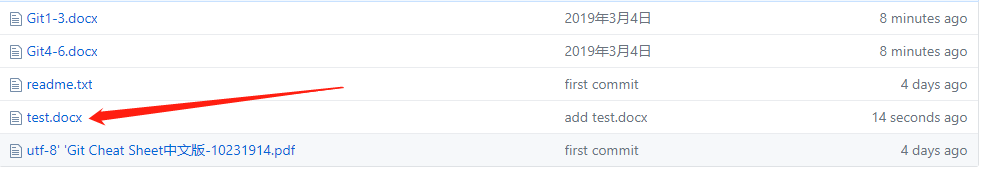
(2) 在本地删除对应的文件：git rm filename（同时在缓存和物理存储中删除文件，慎用），git rm --cache filename（只在缓存中删除对应的文件）

（3）将处理后的工程提交，上传到远程仓库：git commit -m"本地删除远程文件filename"），git push origin master

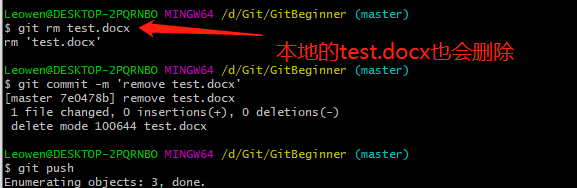
案例：

新建一个test.docx文件，添加并提交

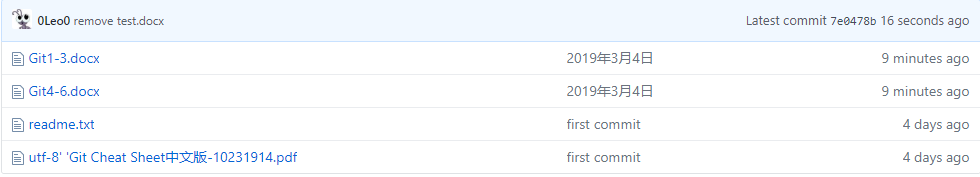




使用git rm test.docx 本地的test.docx也会删除



在push之后，远程的仓库也就没有了该文件



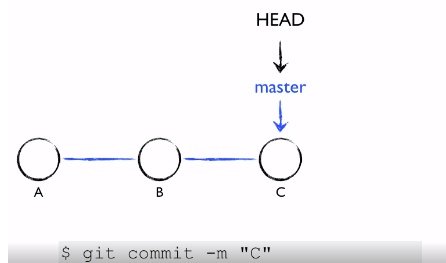
# 创建和合并分支

我们已经知道每次提交，Git都把它们串成一条时间线，这条时间线就是一个分支。截止到目前，只有一条时间线，在Git里，这个分支叫主分支，即**master**分支。**HEAD**严格来说不是指向提交，而是指向**master**，**master**才是指向提交的，所以，**HEAD**指向的就是当前分支。

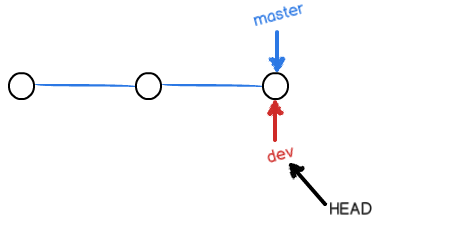
一开始的时候，**master**分支是一条线，Git用**master**指向最新的提交，再用**HEAD**指向**master**，就能确定当前分支，以及当前分支的提交点：



每次提交(git commit)，**master**分支都会向前移动一步，这样，随着你不断提交，**master**分支的线也越来越长：

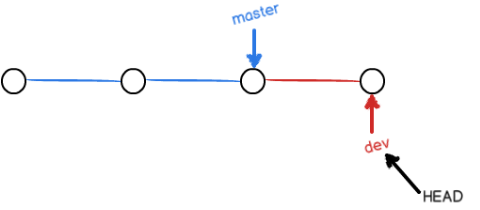


当我们创建新的分支，例如**dev**时，Git新建了一个指针叫**dev**，指向**master**相同的提交，再把**HEAD**指向**dev**，就表示当前分支在**dev**上：

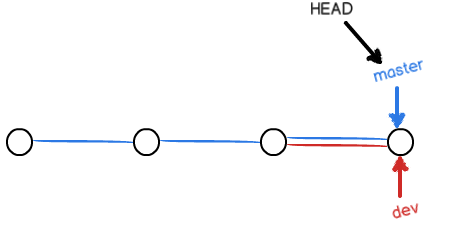


你看，Git创建一个分支很快，因为除了增加一个**dev**指针，改改**HEAD**的指向，工作区的文件都没有任何变化！

不过，从现在开始，对工作区的修改和提交就是针对**dev**分支了，比如新提交一次后，**dev**指针往前移动一步，而**master**指针不变：

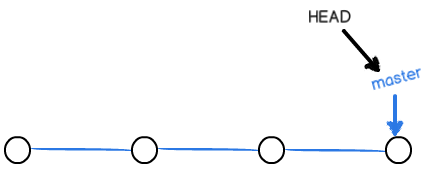


假如我们在**dev**上的工作完成了，就可以把**dev**合并到**master**上。Git怎么合并呢？最简单的方法，就是直接把**master**指向**dev**的当前提交，就完成了合并：



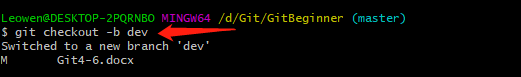
所以Git合并分支也很快！就改改指针，工作区内容也不变！

合并完分支后，甚至可以删除**dev**分支。删除**dev**分支就是把**dev**指针给删掉，删掉后，我们就剩下了一条**master**分支：



**实战：**

首先，我们创建**dev**分支，然后切换到**dev**分支：

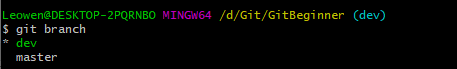


git checkout 命令加上 -b 参数表示创建并切换，相当于下面两条命令

git branch dev

git checkout dev

然后用git branch 命令查看当前分支：

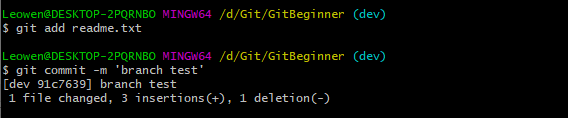


git branch 命令会列出所有分支，当前分支前面会标有一个\*号

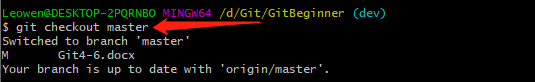
然后，我们就可以在**dev**分支上正常提交，比如对readme.txt做个修改，加上一行：

Creating a new branch is quick.

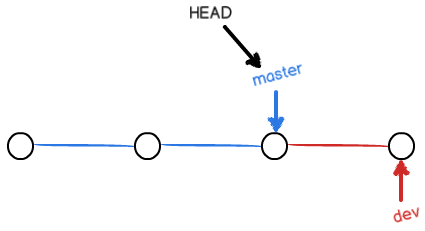
然后提交：

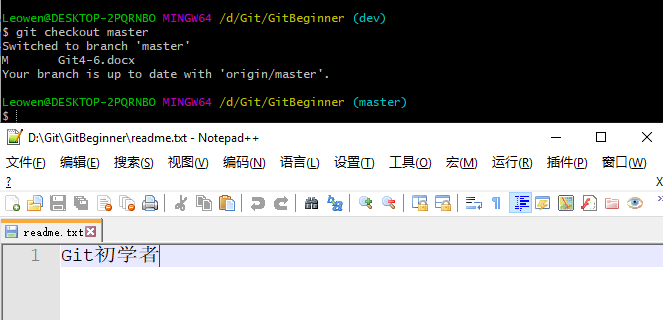


现在，**dev**分支的工作完成，我们就可以切换回**master**分支

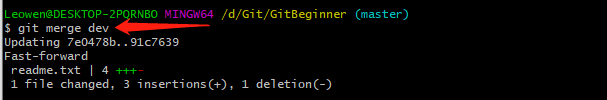


切换回**master**分支后，再查看一个**readme.txt**文件，刚才添加的内容不见了！因为那个提交是在**dev**分支上，而**master**分支此刻的提交点并没有变：

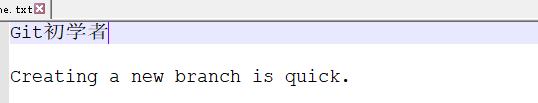




现在，我们把**dev**分支的工作成果合并到**master**分支上：



git merge命令用于合并指定分支到当前分支。合并后，再查看**readme.txt**的内容，就可以看到，和**dev**分支的最新提交是完全一样的。



注意到上面的Fast-forward信息，Git告诉我们，这次合并是“快进模式”，也就是直接把**master**指向**dev**的当前提交，所以合并速度非常快。

当然，也不是每次合并都能Fast-forward，我们后面会讲其他方式的合并。

合并完成后，就可以放心地删除**dev**分支了：



删除后，查看**branch**，就只剩下**master**分支了：



# 小结：

Git鼓励大量使用分支：

查看分支：git branch

创建分支：git branch <name>

切换分支：git checkout <name>

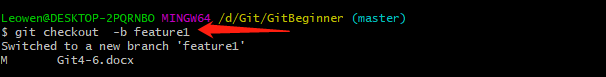
创建+切换分支：git checkout -b <name>

合并某分支到当前分支：git merge <name>

删除分支：git branch -d <name>

# 解决冲突：

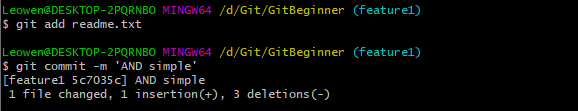
准备新的**feature1**分支，继续我们的新分支开发：



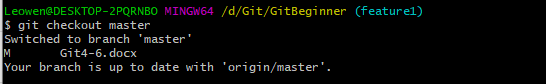
修改**readme.txt**最后一行，改为：

Creating a **new** branch is quick **AND** simple.

在**feature1**分支上提交：



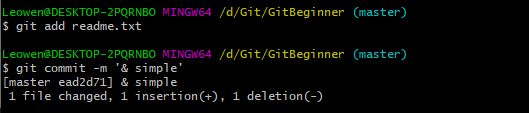
切换到**master**分支：



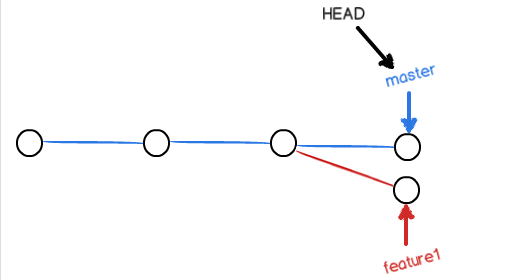
在**master**分支上把**readme.txt**文件的最后一行改为：

Creating a **new** branch is quick & simple.

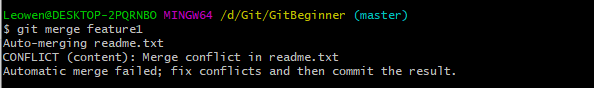
提交：



现在，**master**分支和**feature1**分支各自都分别有新的提交，变成了这样：

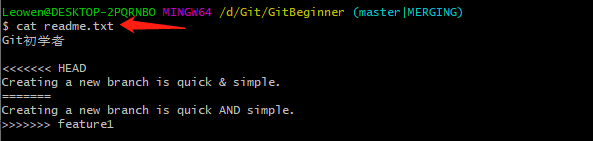


这种情况下，Git无法执行“快速合并”，只能试图把各自的修改合并起来，但这种合并就可能会有冲突，我们试试看：



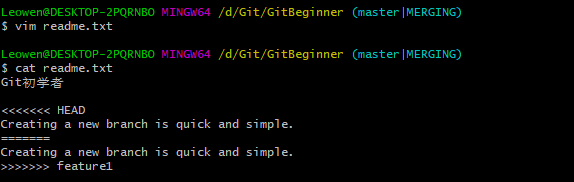
果然冲突了！Git告诉我们，**readme.txt**文件存在冲突，必须手动解决冲突后再提交。使用**git status**也可以告诉我们冲突的文件。

查看**readme.txt**的内容：

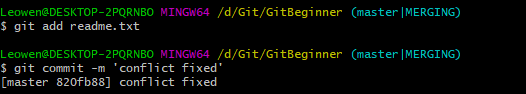


Git用**<<<<<<<**，**=======**，**>>>>>>>**标记出不同分支的内容，我们修改如下后保存：

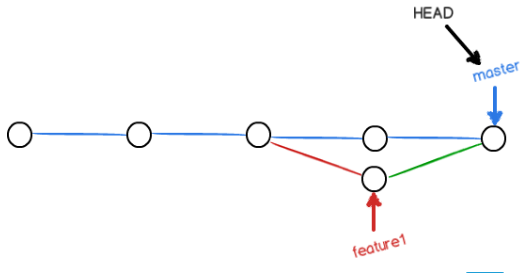
Creating a **new** branch is quick **and** simple.



再提交：

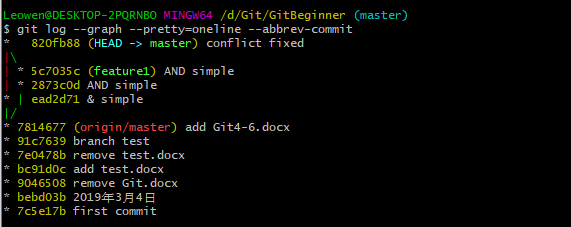


现在，**master**分支和**feature1**分支变成了下图所示：



用带参数的git log也可以看到分支的合并情况：

git log --graph --pretty=oneline --abbrev-commit



最后，删除**feature1**分支：

