# Git Fetch VS Git Pull

前言

在我们使用git的时候用的更新代码是git fetch，git pull这两条指令。但是有没有小伙伴去思考过这两者的区别呢？有经验的人总是说最好用git fetch+git merge，不建议用git pull。也有人说git pull=git fetch+git merge，真的是这样吗？为什么呢？既然如此为什么git还要提供这两种方式呢？

**1. 相同点**

首先在作用上他们的功能是大致相同的，都是起到了更新代码的作用。

**2. 不同点**

先补充一些git里面相关的一些知识：

首先我们要说简单说git的运行机制。git分为本地仓库和远程仓库，我们一般情况都是写完代码，commit到本地仓库（生成本地仓的commit ID，代表当前提交代码的版本号），然后push到远程仓库（记录这个版本号），这个流程大家都熟悉。

我们本地的git文件夹里面对应也存储了git本地仓库master分支的commit ID 和 跟踪的远程分支orign/master的commit ID（可以有多个远程仓库）。那什么是跟踪的远程分支呢，打开git文件夹可以看到如下文件：

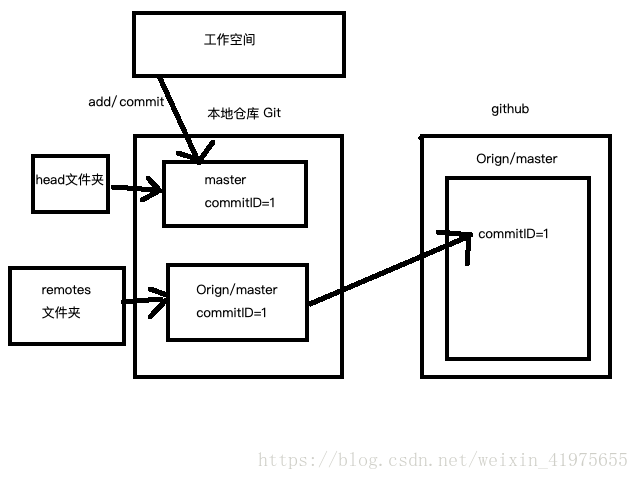
.git/refs/head/[本地分支]

.git/refs/remotes/[正在跟踪的分支]

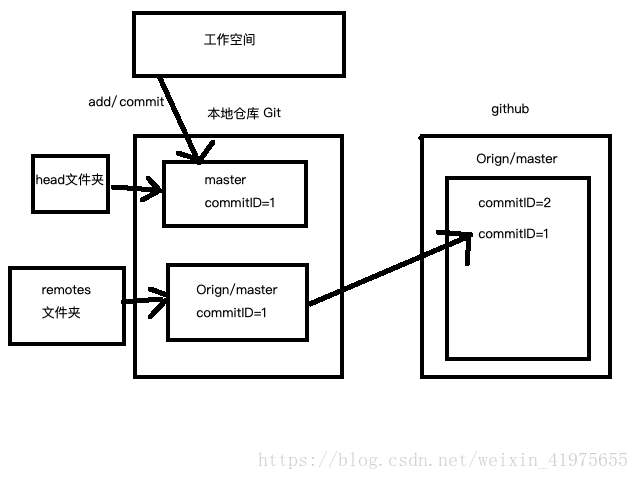
其中head就是本地分支，remotes是跟踪的远程分支，这个类型的分支在某种类型上是十分相似的，他们都是表示提交的SHA1校验和（就是commitID）。

但是，不管他们是如何的相似，他们还是有一个重大的区别：

更改远端跟踪分支只能用git fetch，或者是git push后作为副产品（side-effect）来改变。我们无法直接对远程跟踪分支操作，我们必须先切回本地分支然后创建一个新的commit提交。



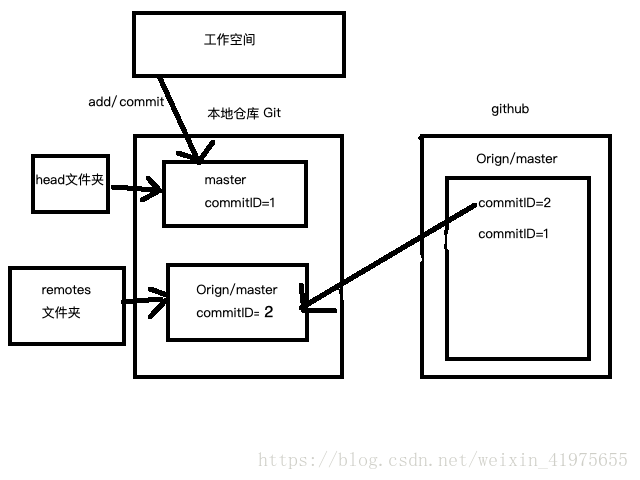
首先假设我们本地仓库的 master 分支上 commit ID =1 ，orign/mastter中的commit ID =1 ;这时候远程仓库有人更新了github orgin库中master分支上的代码，新的代码版本号commit ID =2 ,那么在github上 orign/master的commitID=2，然后我们要更新代码。



**1. git fetch**

使用git fetch更新代码，本地的库中master的commit ID不变，还是等于1。但是与git上面关联的那个origin/master的commit ID变成了2。这时候我们本地相当于存储了两个代码的版本号，我们还要通过merge去合并这两个不同的代码版本，如果这两个版本都修改了同一处的代码，这时候merge就会出现冲突，然后我们解决冲突之后就生成了一个新的代码版本。

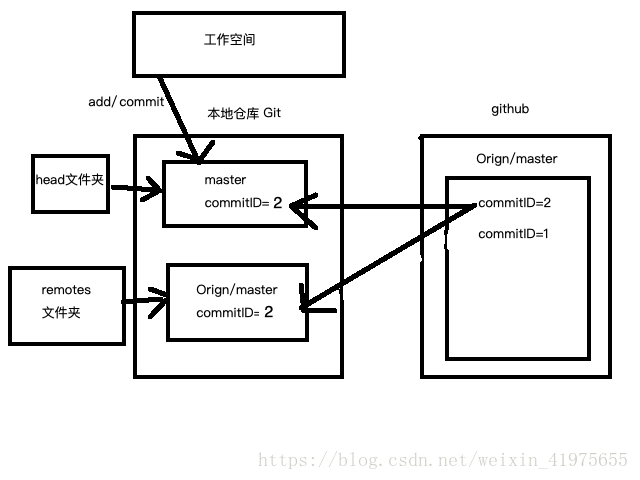
这时候本地的代码版本可能就变成了commit ID=3，即生成了一个新的代码版本。



相当于fetch的时候本地的master没有变化，但是与远程仓关联的那个版本号被更新了，我们接下来就是在本地合并这两个版本号的代码。

**2. git pull**

是用git pull更新代码的话就比较简单暴力了，看下图。



使用git pull的会将本地的代码更新至远程仓库里面最新的代码版本

**3. 总结**

由此可见，git pull看起来像git fetch+git merge，但是根据commit ID来看的话，他们实际的实现原理是不一样的。

这里借用之前文献看到的一句话:

不要用git pull，用git fetch和git merge代替它。

git pull的问题是它把过程的细节都隐藏了起来，以至于你不用去了解git中各种类型分支的区别和使用方法。当然，多数时候这是没问题的，但一旦代码有问题，你很难找到出错的地方。看起来git pull的用法会使你吃惊，简单看一下git的使用文档应该就能说服你。

将下载（fetch）和合并（merge）放到一个命令里的另外一个弊端是，你的本地工作目录在未经确认的情况下就会被远程分支更新。当然，除非你关闭所有的安全选项，否则git pull在你本地工作目录还不至于造成不可挽回的损失，但很多时候我们宁愿做的慢一些，也不愿意返工重来。





