# 获取file\_id

1. 获取repo base hash——计算方法待定
   1. 每一个repository是固定的，可以在初始要求用户输入，此后不再变化，代码里可以不用动态获取
2. 获取file path——实现中
   1. 需要根据git pull/git fetch下载的文件进行动态获取路径
   2. 创建/修改metadata的操作建议脱离git filter执行，在git add前提醒用户进行metadata的修改或创建
   3. 如果创建/修改metadata加进git filter的话，则要求filter的代码执行时自动检测用户是否有创建操作，并调出输入端口让用户输入新的metadata信息。

# 获取file path

1. 获取更新文件的路径信息（已完成）。

git pull = git fetch + git merge

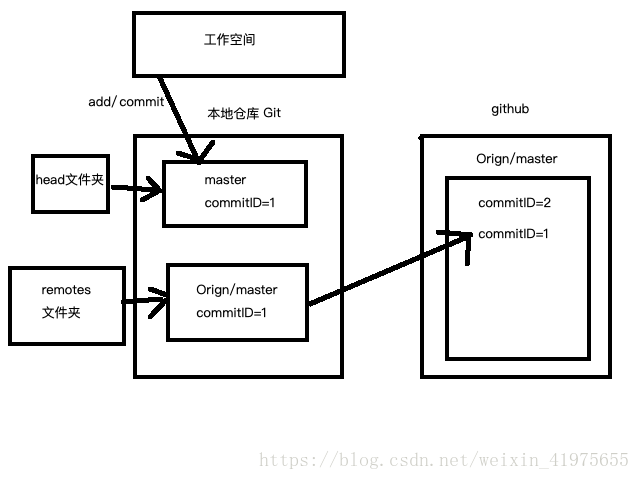
git分为本地仓库和远程仓库，用户写完代码，commit到本地仓库（生成本地仓库的commit ID，代表当前提交代码的版本号），然后push到远程仓库（记录这个版本号）。

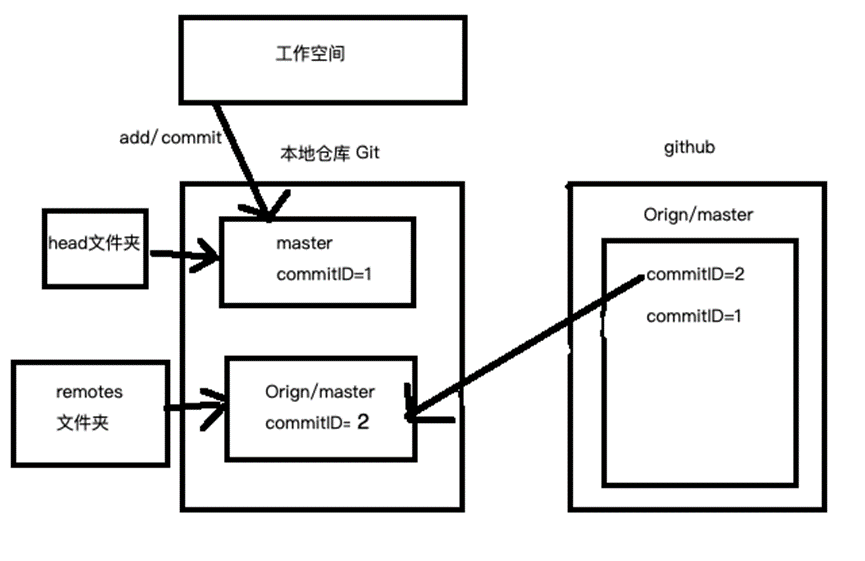
本地的.git文件夹里面对应也存储了git本地仓库master分支的commit ID 和 跟踪的远程分支orign/master的commit ID（可以有多个远程仓库）。打开git文件夹可以看到如下文件：

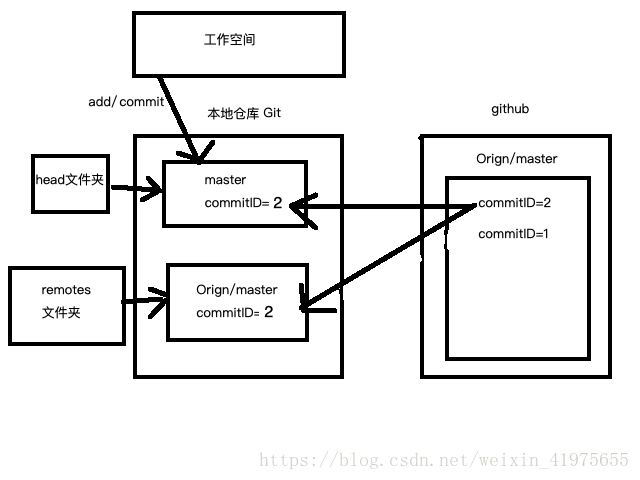
.git/refs/head/[本地分支]

.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）。





git fetch

git pull

git fetch从远程仓库获取本地仓库没有的更新内容，但是并没有更新本地的工作区。此时origin/master更改为最新的commit ID,FETCH\_HEAD文件也更改为获得的最新commit ID。

若用户认为更新内容可以与原内容合并，则直接执行git merge。这两个命令相当于执行了git pull。

git 文档里建议使用git fetch+git merge代替git pull, 因为git pull把过程的细节都隐藏了起来，一旦代码有问题，很难找到出错的地方。另外一个弊端是，本地工作目录在未经确认的情况下被远程分支更新，容易造成不可挽回的损失。

所以在获取file path的过程中，使用git fetch来操作会更加容易一些。当执行完git fetch之后，.git文件夹中的HEAD文件仍旧指向本地仓库的最新commit ID，而FETCH\_HEAD文件已经改为了远程仓库的最新commit ID。这个commit ID指向的文件内容，也就是用户当前要获取，代码需要验签解密的内容。

假设此commit ID为399db（40位hash值，查看时只选前几位即可）。

查看commit id为399db的文件目录

git ls-tree -r 399db

结果：

100644 blob 6a7dbe0b88c2ed9c3425c2557a2e95578ce7b614 abc.txt

100644 blob 21a93801b524e7245b3328c72aa4fd8b3dacccea abc\_temp.txt

100644 blob f7e3532175e604114694ab2b1c9cb38ff64aaf16 def.txt

100644 blob 084bd54f82ee14ae30bb9d2864dc8179466d2635 sss.txt

100644 blob 62b248406944abf4b4c9c58b869e41b40e2af5e2 test/aaa.txt

结果就是本次commit更新的文件内容，这些文件都会经过filter，需要根据他们的file\_id获取相应的metadata，进行验签以及解密。

2.根据文件内容hash确定文件路径。

以上已经获取了全部更新文件的hash值和文件路径，由于git调用filter时，会自动将文件内容作为程序输入值，自动将程序输出值作为经过filter的文件内容，并没有给出文件名等信息。

目前最直接的思路是，根据git传进来的文件内容，求相应的hash值，在最新commit的更新列表中检索，找到对应的文件信息，从而获取路径。

由于每个文件经过git filter，都相当于调用一次程序，所以每次都要获取commit文件列表，将其保存为合适的数据结构便于查询。

3. 最终将以上两步封装为动态链接库，由原先实现的代码调用。