# Git基本原理介绍(1)—代码仓库的初始化

Linux命令watch

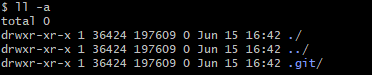


每隔1s的时间，刷新一下我们find后面的目录，find后面是一个.，表示当前目录

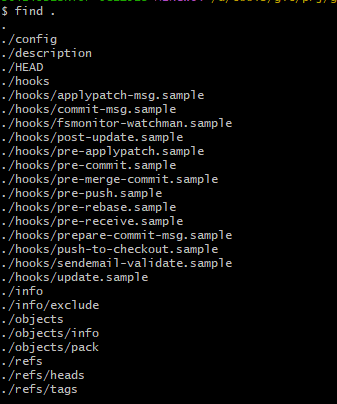
初始化代码仓库，使用git init



我们可以通过ls –a可以发现在本地出来了一个.git的目录：



这个.git的目录中有下面的主要内容：



下面我们来分别说一下，上面的几个文件或者文件夹的主要作用。

1. Config文件

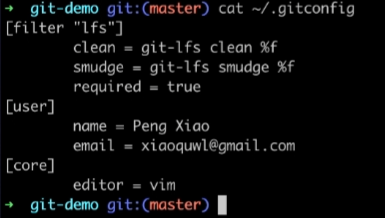
说明：当前代码仓库的配置文件，但是需要注意的是这是一个代码仓库的本地配置文件，同时它还有一个全局的配置文件，在home有个.gitconfig下面的，我们可以通过git config –gloabal 来配置里面的内容，如下图所示；



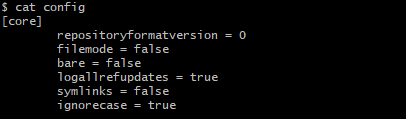
我们还可以通过git config –gloabal –l来查看gloabal的配置情况，如下：



全局的gloabal文件就在下面的位置处：

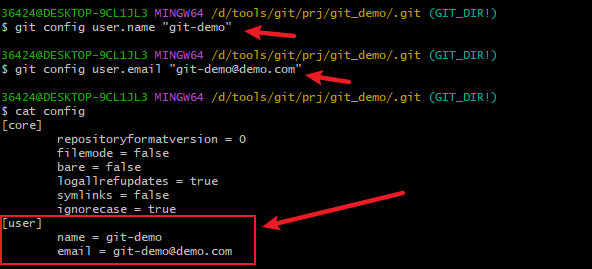


下面是我们本地的config文件，我们将其打开可以看见其内部的内容：



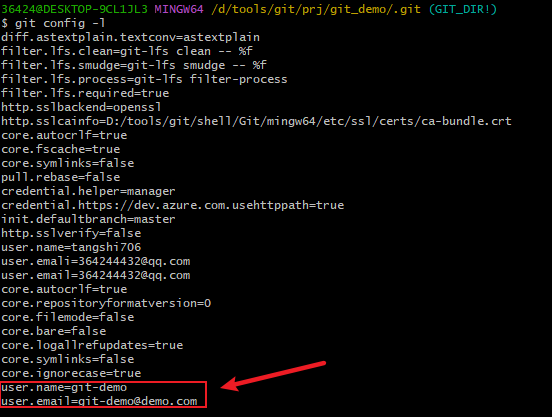
注意，我们在配置时，不加—gloabal的话，那配置的内容就是本地的config中的，例如：

此时配置的内容就会更新到config文件中去，如下图：



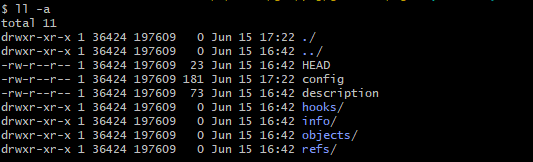
Note：此时代码仓库会优先使用本地的config文件配置的内容，而不是去全局的配置的文件去找，只有在本地找不到的情况下才去全局的配置文件去找；

同样我们也可以通过git config –l来查看本地的config配置，此时我们之前配置的username和email信息就有了，具体如下：



这样在每一个仓库下有一个config文件的好处在，不同仓库使用时，可以选择不同用户，避免了使用全局的只有一个配置造成冲突；

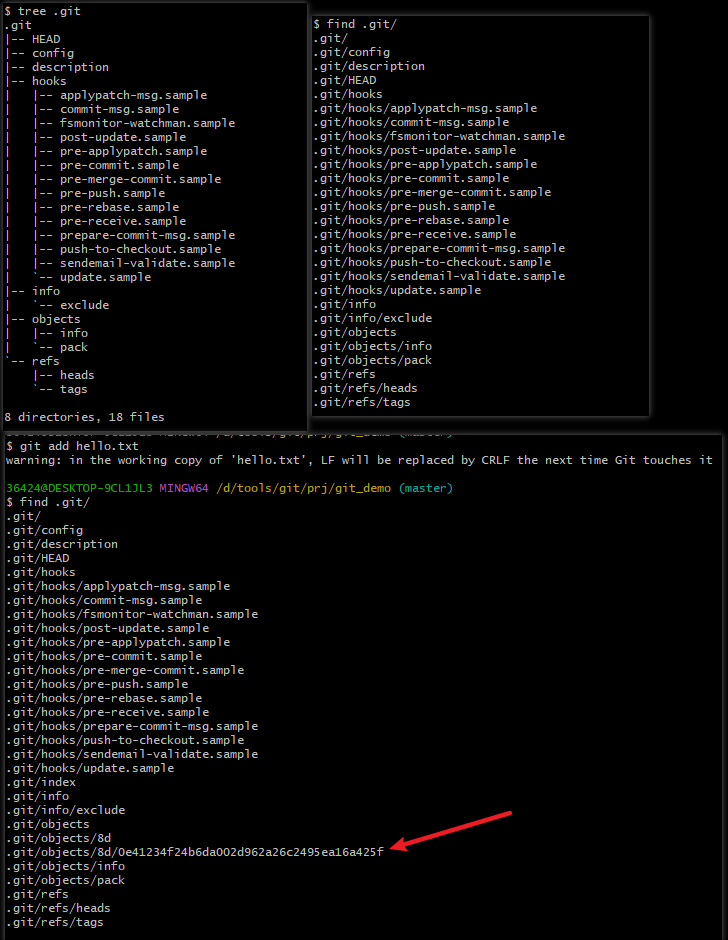
几点说明：



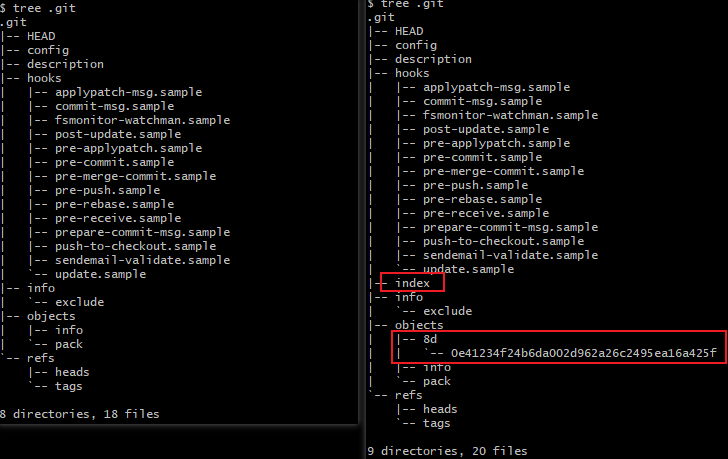
上面的objects是git最最中的东西，他是git存储的对象，后面会详细的说明。还有HEAD，refs是commit向前走的时候，指针指向的问题；

# Git基本原理介绍(2)—git add背后到底发生了啥？

我们可以通过git add将文件加到暂存区，我们看看在git add前后发生了什么？



多了2个文件夹，一个hash的文件夹和一个index的文件夹



我们可以通过git cat-file这个命令来查看对应object对应的内容，我们可以通过不同的选项来查看object所应用的内容；

例如git cat-file –t hash\_value查看这个object的类型，如下图：



注意：取值时，我们不需要到对应的文件夹下面，我们只需要文件夹有.git的目录即可；

上面的对象objects的类型是blob，他是在git存储系统中是用来存储文件内容的。

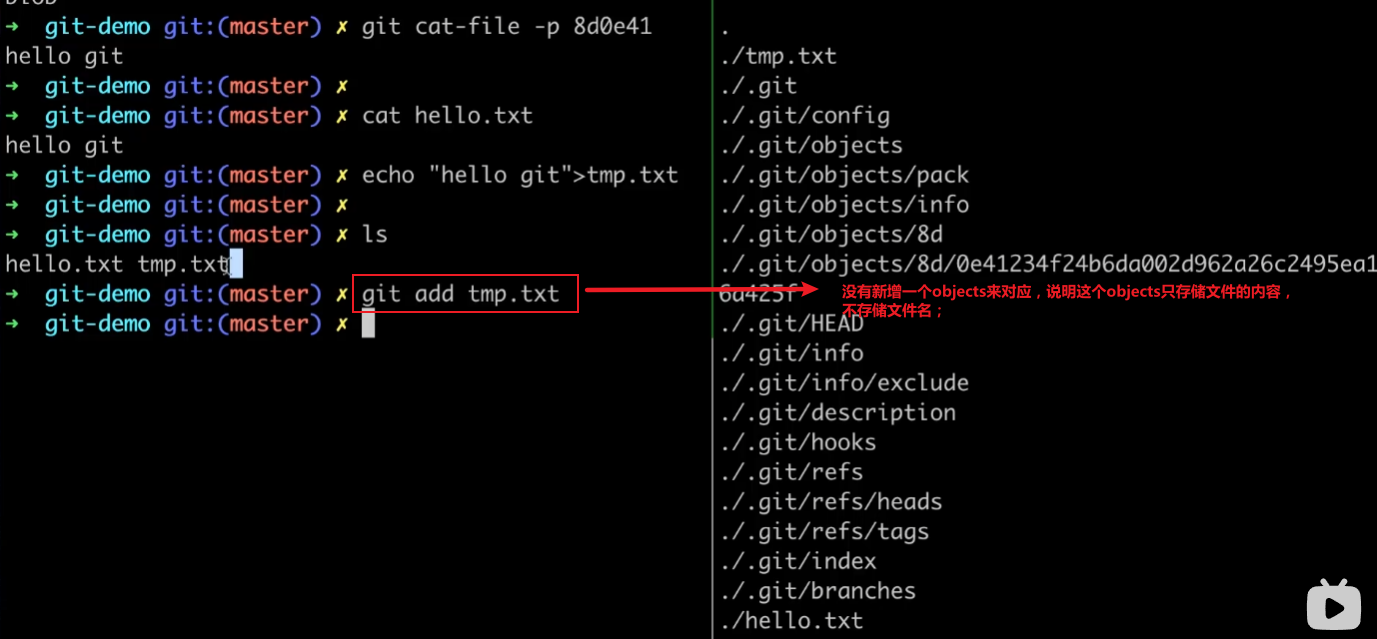
这样我们可以用-p来查看这个objects的内容，具体如下：



文件的内容也是和这个相同的，如下图：



注意的是，这个objects只存储文件的内容，不存储文件名的，如下图所示：

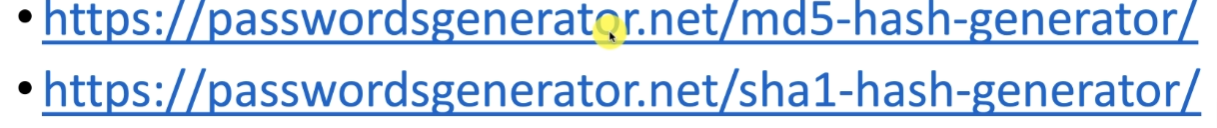


这样我们的objects内部的hash值，不光存储了文件内容，同时这个对象的类型；

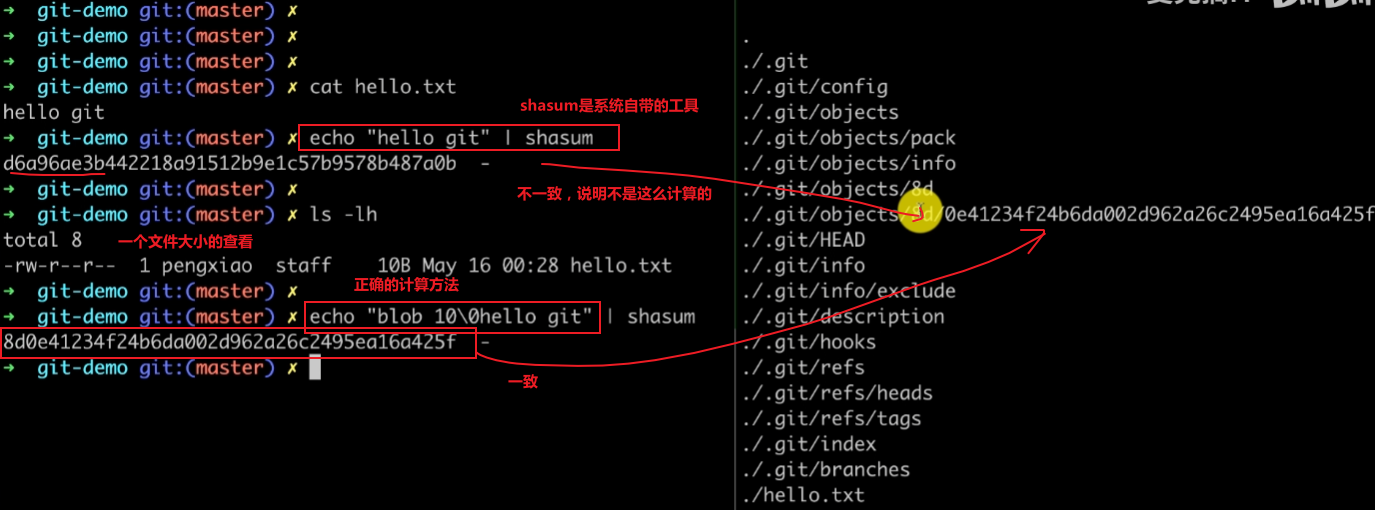
这个hash值是怎么算出来的呢？通过sha1的算法；

# Git基本原理介绍(3)—聊聊blob对象和SHA1哈希

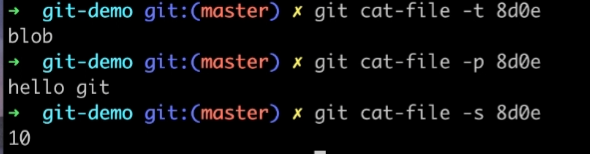




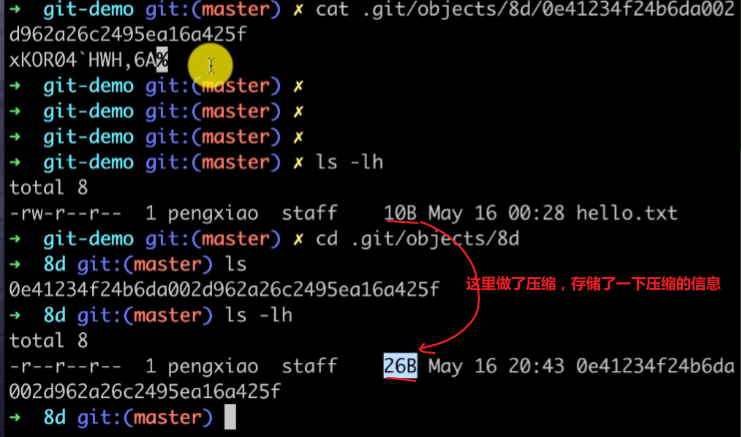
如何计算的？



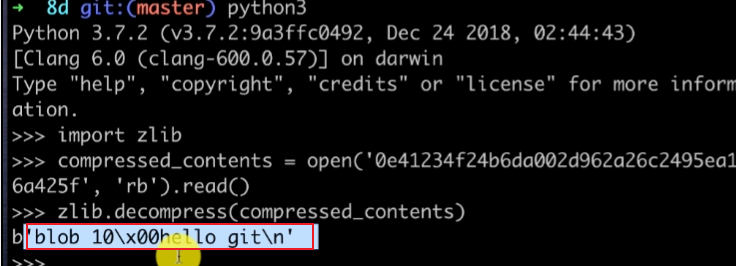
查看objects的几种选项：



我们可以用cat命令来获取这个objects的hash值看看这里面的内容是啥？

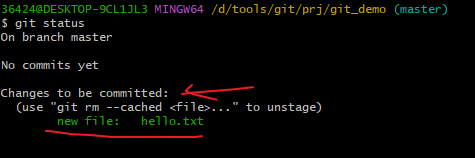


我们可以看到这个压缩的内容的，可以用python的一个库zlib：



注意：里面有一个x00是代表的文件权限；

这样我们通过git status就可以看到文件的状态发生变化了，告诉我们这个变化可以进行commit提交了，如同所示：



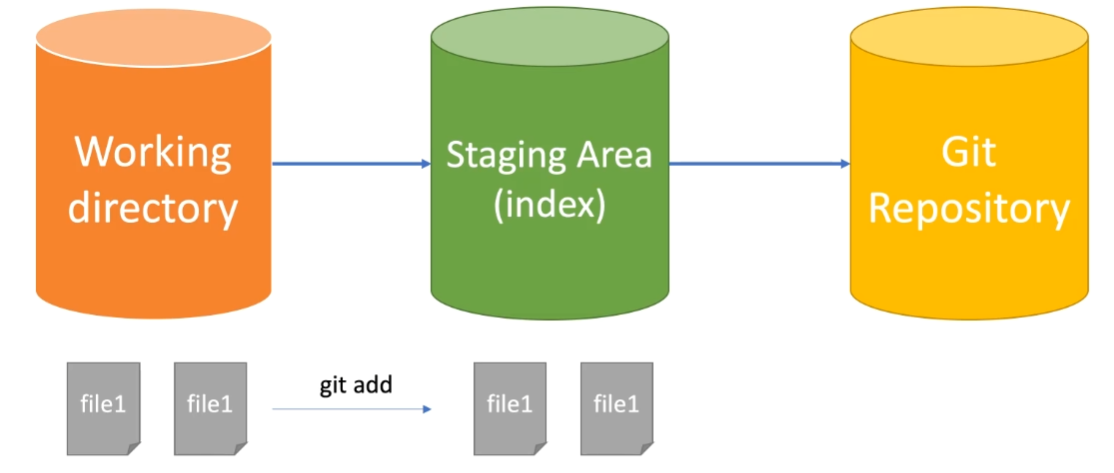
这个绿色的文件通过git add加载到了objects里面了，这样git就知道了。Git通过这些操作知道哪些文件添加到了objects里面了，哪些还在工作区域中（untracted）。

Git如何知道哪些是存储在objects里面，哪些在工作区的呢？git 如何知道的呢？具体看下面的视频。

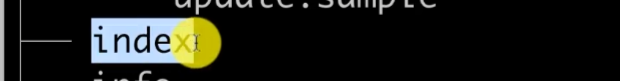
# Git基本原理介绍(4)——聊聊工作区和索引区

我们可以在本地建一些文件，进行编辑，然后我们将通过git add命令将我们编辑的文件加到暂存区，下面的一张图就可以很好的说明这点；

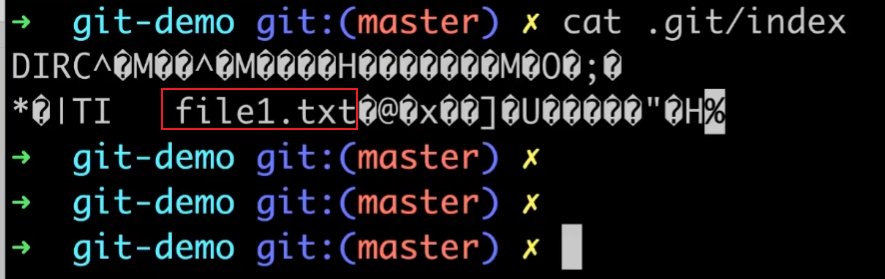
我们通过git add的命令，就可以对对应的文件生成对应的blob object；然后我们查看blob object中发现，git并没有把文件名存储在里面，只存储了对象的类型，大小和内容；那我们的文件名是存在index里面的。



在git add之后，除了objects里面更新了，其实还有一个文件也更新了，那么就是index，



查看index内的内容的话，虽然有一些乱码，但是我们发现里面是有文件名的，具体如下：

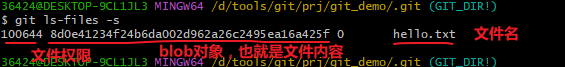


实际上index里面的内容，我们可以看到的，git提供了对应的命令来查看里面的内容，命令如下：

git ls-files命令是查看当前索引区里面的文件列出来，如图所示：



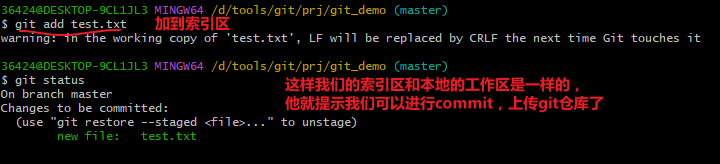
可以增加选项来查看更为详细的内容：-s



这样我们就知道索引区（index）是连接本地和远程代码仓库的，还没有将我们的代码提交到仓库，也就是还没有生成commit，没有版本的概念。

**这样staging area存在的意义是：为了上次代码仓库做准备，而内部的index它保存了待提交到仓库的状态（文件，文件夹，目录，每个文件的状态，也就是具体的内容是什么），这样有了index，我们就知道working directory和staging area的文件信息是有差别的，这样我们在使用git status时看到的结果就是对应的信息差别。**

下面的例子就是使用git add将本地工作的文件提交到索引区，这样进行git status时，就可以发现本地和暂存区是一样的，提示我们可以将文件提交到远程的仓库中去；

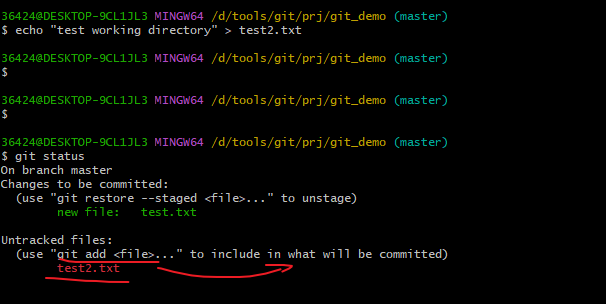


如果此时我在建一个文件，没有git add这是工作区和暂存区是不同的，这样我们在使用git status时，git就会告诉我们存在不一样的文件，例如：

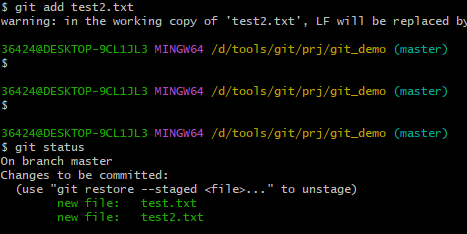
**此时通过使用git ls-files –s查看index索引区内的信息发现没有新建文件的信息：**



使用**git status**时会报出不同的信息，**此时会报出untracked files没有被追踪的文件**，这个就是在工作区中新建的文件：

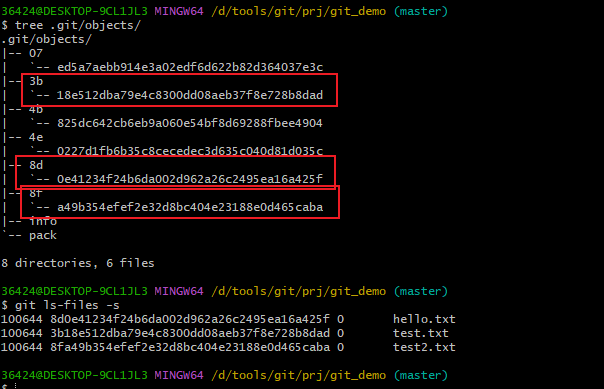


这时我们通过git add命令将新的文件加到索引区，这时用git status时就是没有untraced files，具体如下：

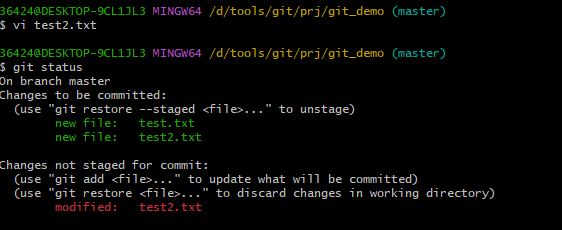


这时，我们的工作区和索引区又同步了；

我们可以查看本地文件有哪些：

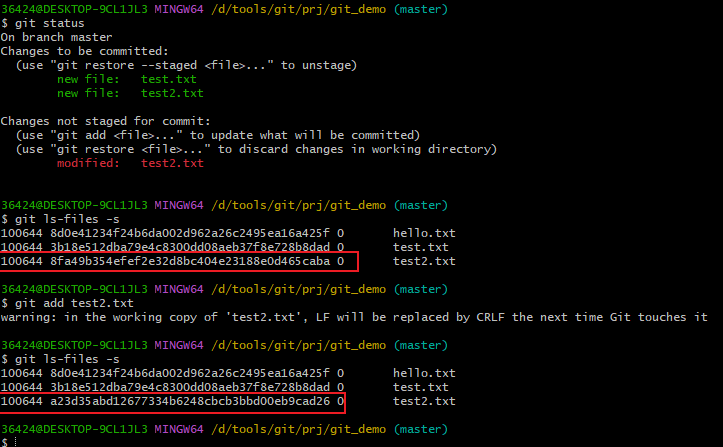


下图是，我们修改了某一个文件，这是我们的工作区和索引区的信息又不同步了，这时git status又会报出新的状态信息，如下所示：



Git通过对比index中的内容（文件名）发现和本地文件名一样，这个git就知道存在相同文件名，但是通过**blob对象的sha1的值不同**，这样git就知道被修改过了，通过git status就可以报出来modified提示，告诉这样在工作区的内容和索引区的内容不一样。然后通过git add将其加到索引区，这样两者又相同了。

Git add前后索引区的blob对象的值不同：

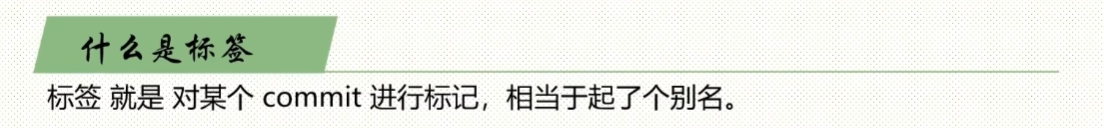


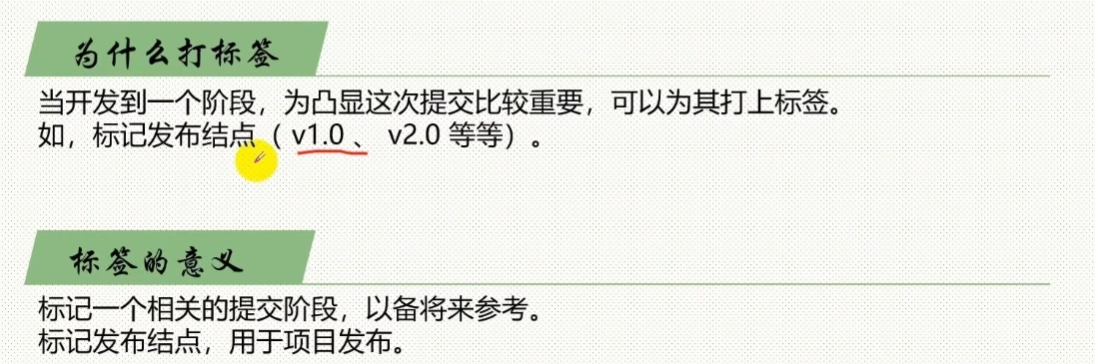
这是我们发现test2.txt存在2个blob对象，也就是说，这时，此文件就出现了2个版本，这是这两个版本是如何存在的呢？下面的git commit会进行说明。

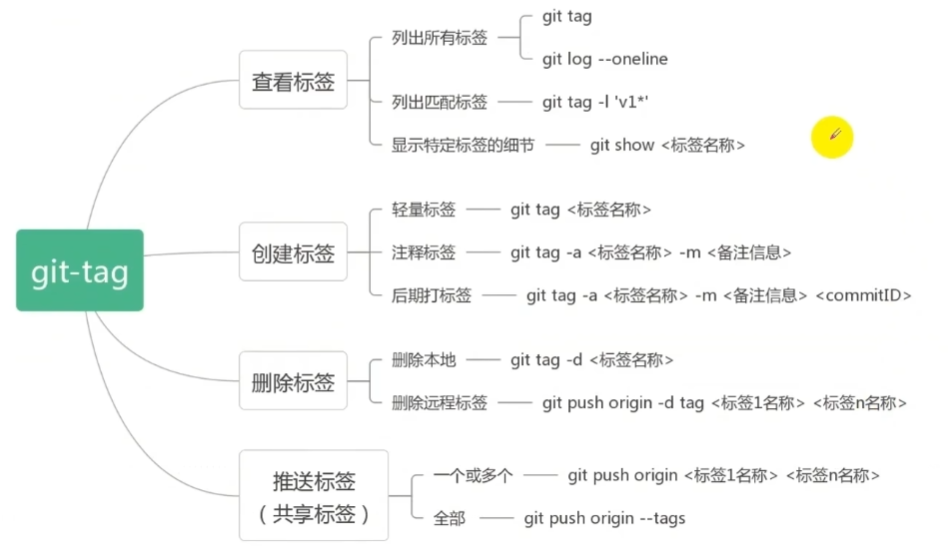
# Git基本原理介绍(5)——git commit背后到底发生了什么

# git-tag标签管理



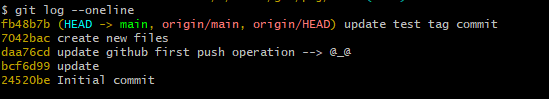




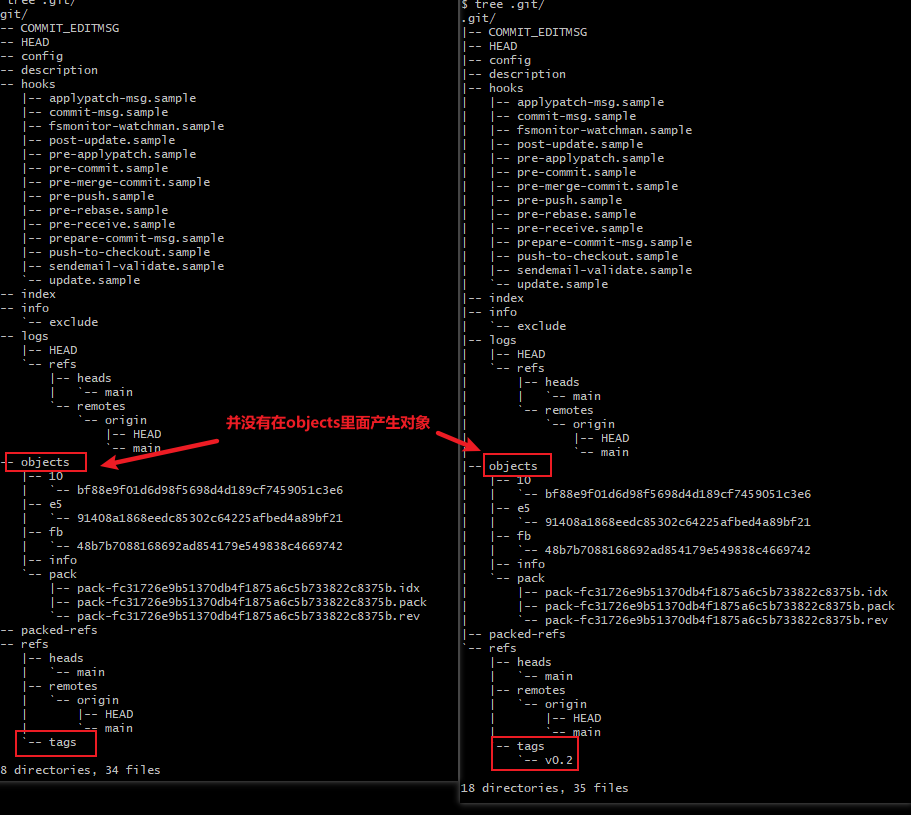


查看项目的日志：

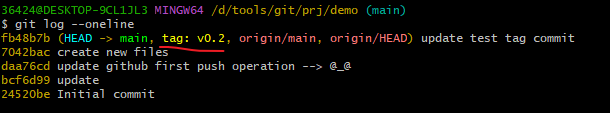
Git log –oneline，这里我们可以看到有4次提交；

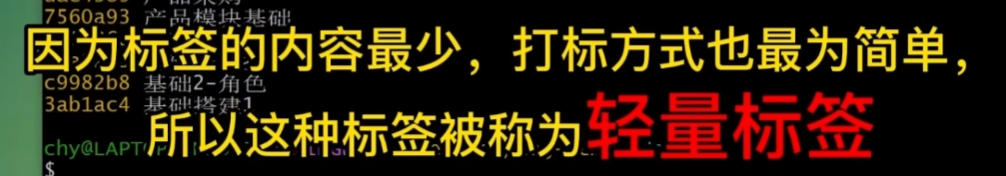


打标签有两种方式，第一种是轻量级的查找，就是git tag 标签名，这个不会在objects里面产生一个对象，操作如下：



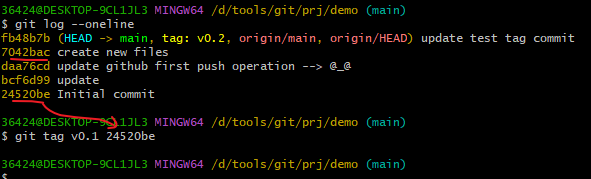
这时这个标签是给我们这个仓库的最新的commit id打的标签；这时我们用git log –oneline可以查看到tag的信息；



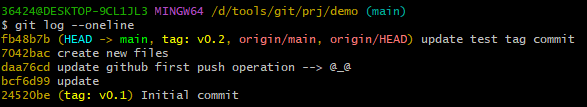


可以给特定的commit打标签，可以使用下面的指令，如下图：

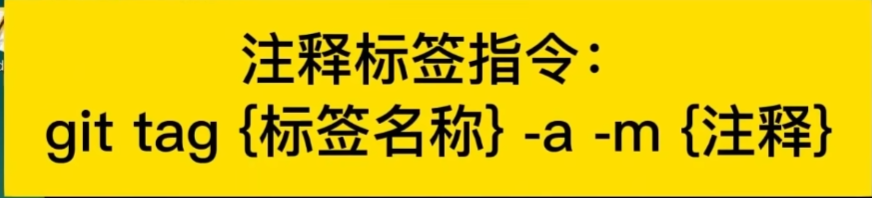




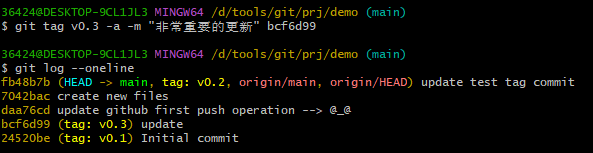
我们通过查看log，可以看到标签是否打上，如下图：



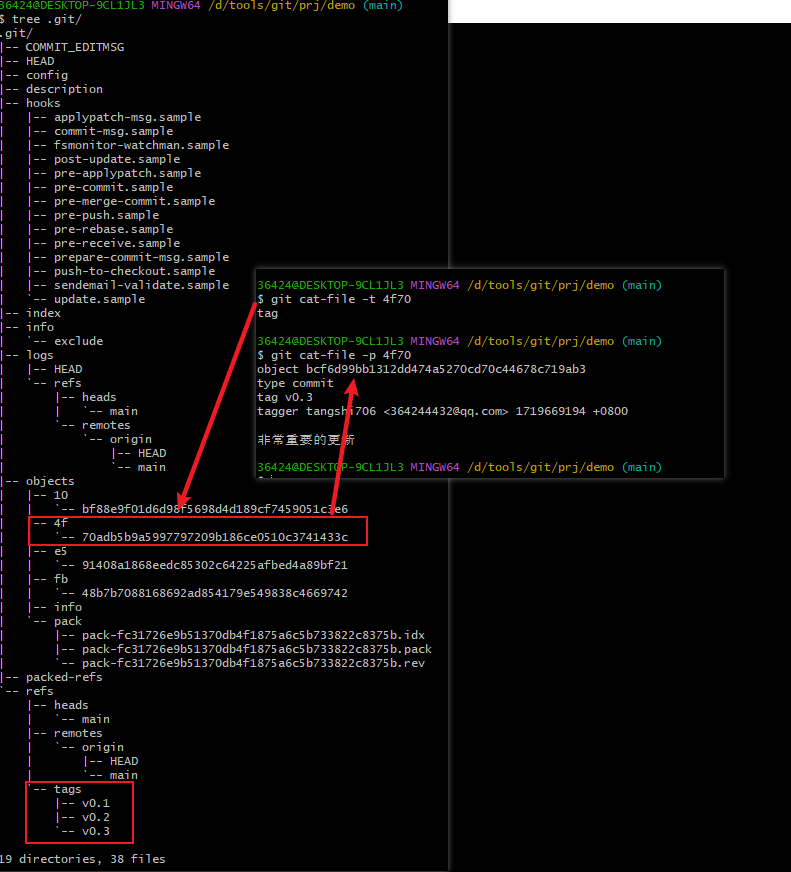
第二种标签，我们称为注释标签，就是在创建标签的同时，增加一个注释，和git commit有点相识，如下图所示：



注意：-a表示这个标签为注释标签；



这里我们使用了注释标签，它不仅在tag中产生了一个标签，而且它在objects中同时也产生了一个对象，他的指向就是一个标签的数据类型，如下所示：

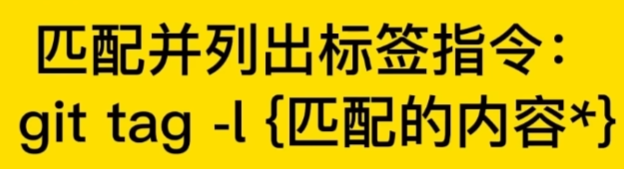


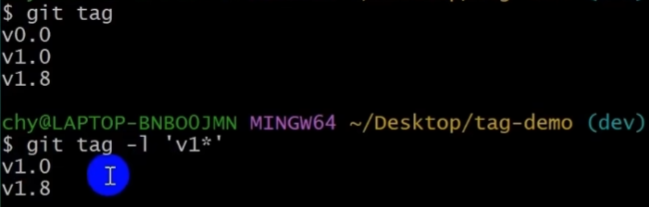
查看我们设置的标签，前面使用了，git log –oneline来查看，这里就不多说了；

第二种方式就是使用git tag来查看：



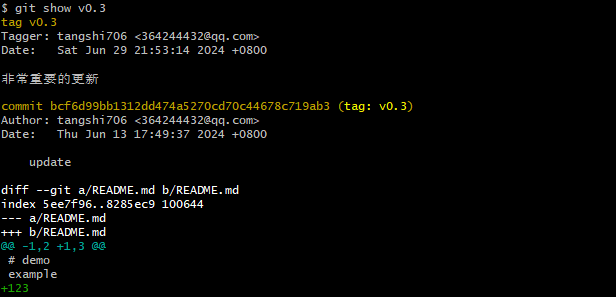
我们可以在这个命令的后面加上一些参数来查看更为详细的内容，命令如下：





使用git show，查看更为详细的信息：



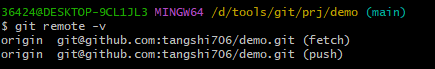




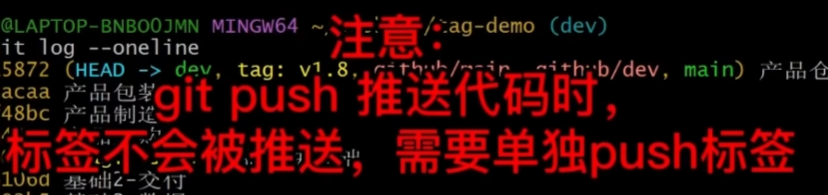
标签如何推送到远端库呢？也就是本地库和远端库之间的共享；

## 标签的推送

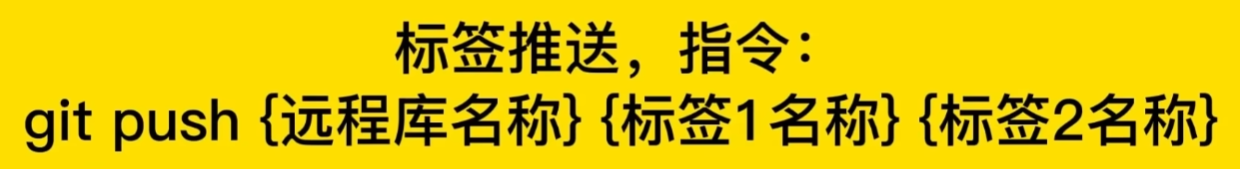
我们看一下本地库对应的远端库的信息，使用git remote –v



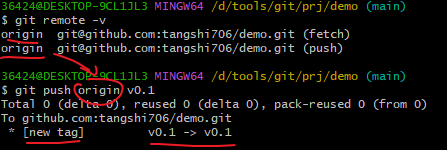
注意一点：在使用git push在进行代码提交的使用，标签是不会被提交的，需要用一些命令来提交；需要我们单独来推送；



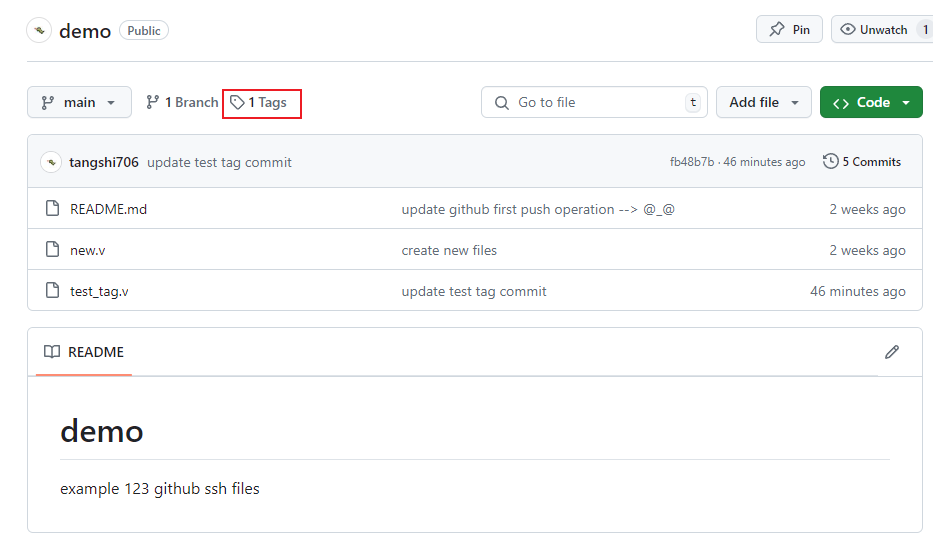
在推送标签时，我们可以一个的推送，也可以一把推送，如下所示：

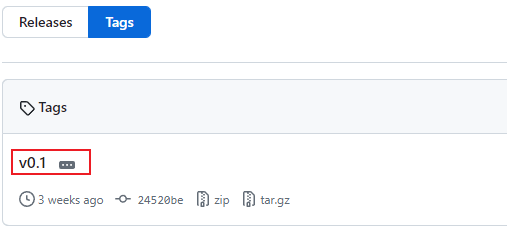


推送一个标签：



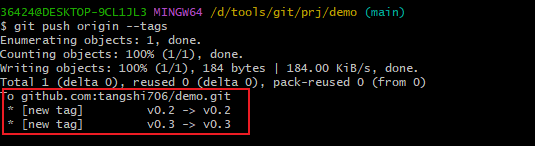
远端的仓库更新了一个仓库：



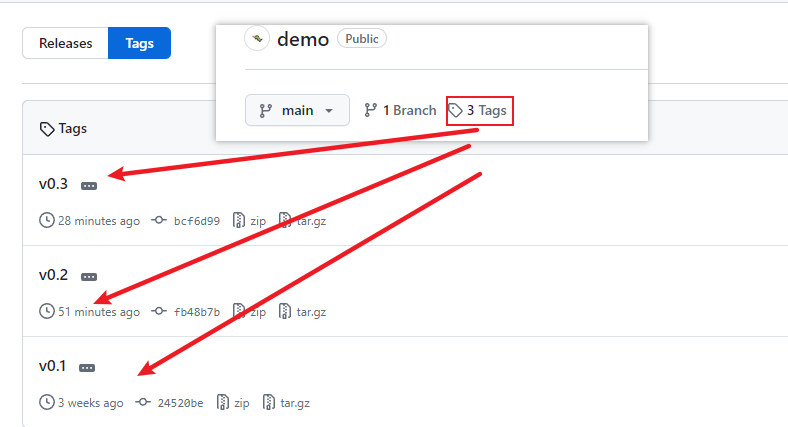


我们可以一次性把本地的所有的标签都推送给远端库，命令如下：

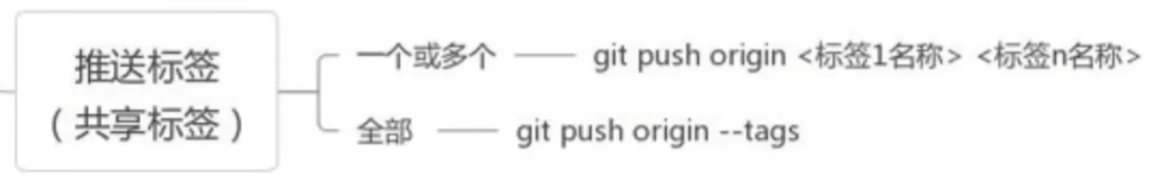




远端仓库的查看：

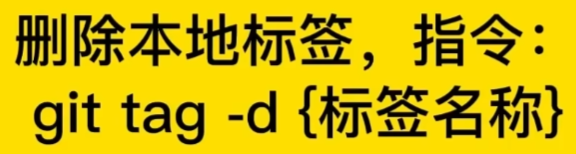


总结一下：



## 标签的删除

先是删除本地的标签，注意这里我们有两种标签，一种是轻量级的标签，因为不涉及对象objects，所以删除就完了，另外一个就是加注释的，在打标签时生成标签的同时，也生成了一个objects，这时你把标签删除，当时这个对象还在，就变成了一个垃圾的对象；



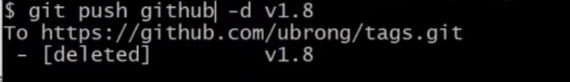


在查看的时候，这个标签就没有了：

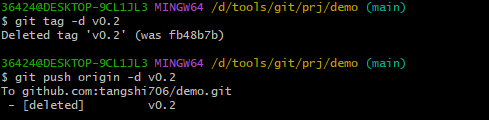


删除远端库的方式：

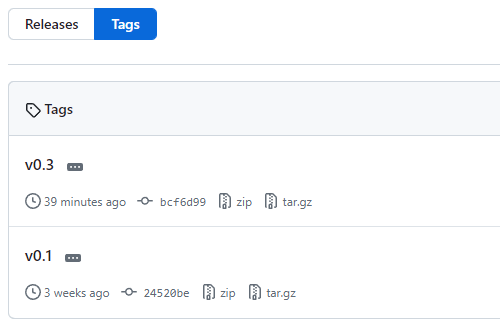


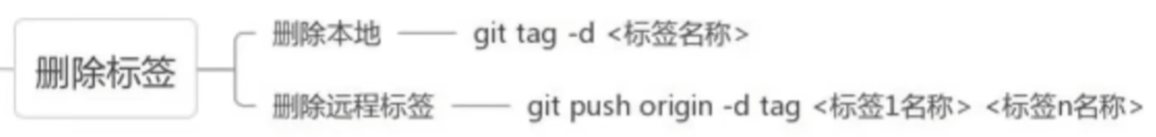


先删除本地的标签，然后在删除远端库的标签：



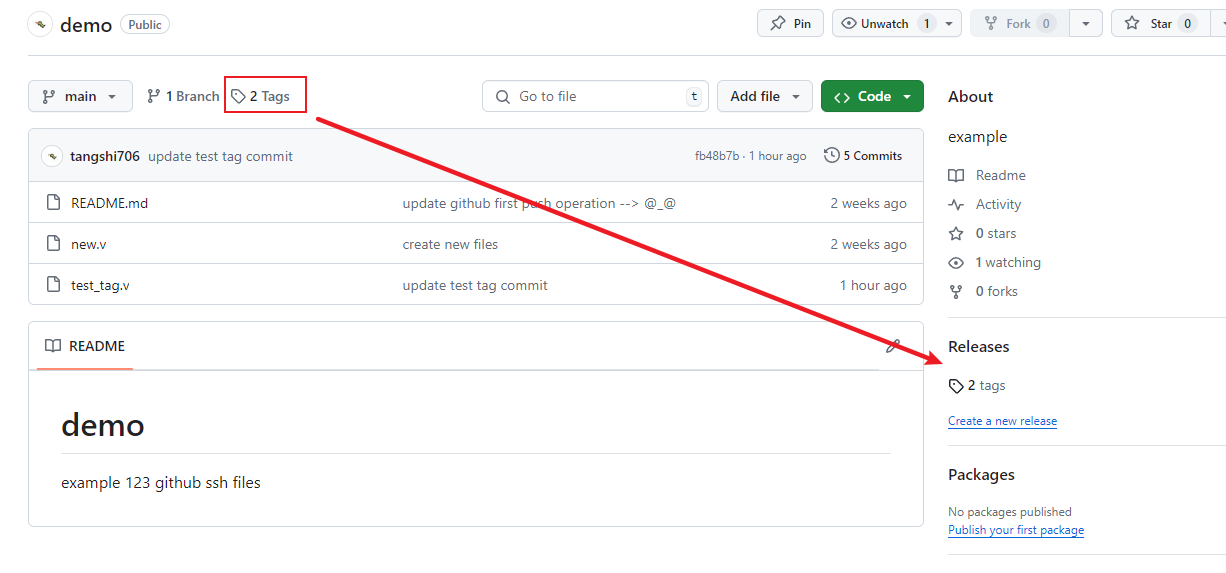
0.2的标签就被删除了；



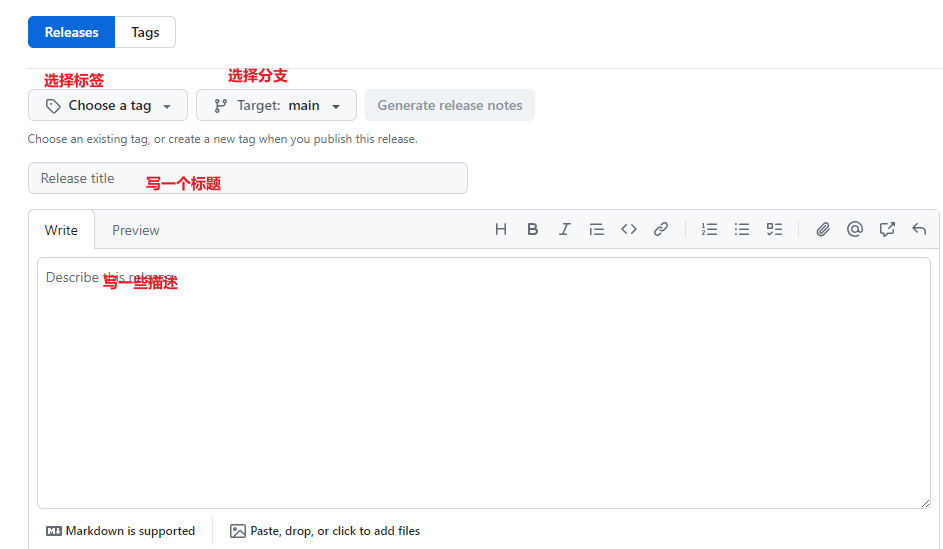


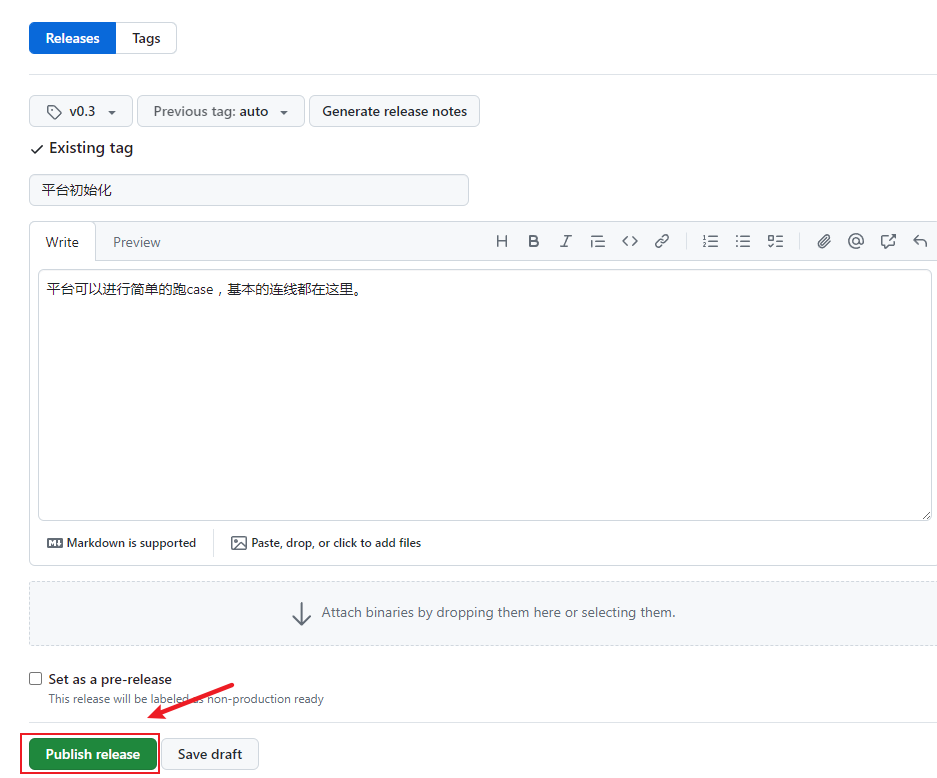
我们打标签的目的就是标记项目发布的结点，也就是说打的标签将来可以是要发布的，这样在git中如何进行版本的发布？

需要到网页上去操作：

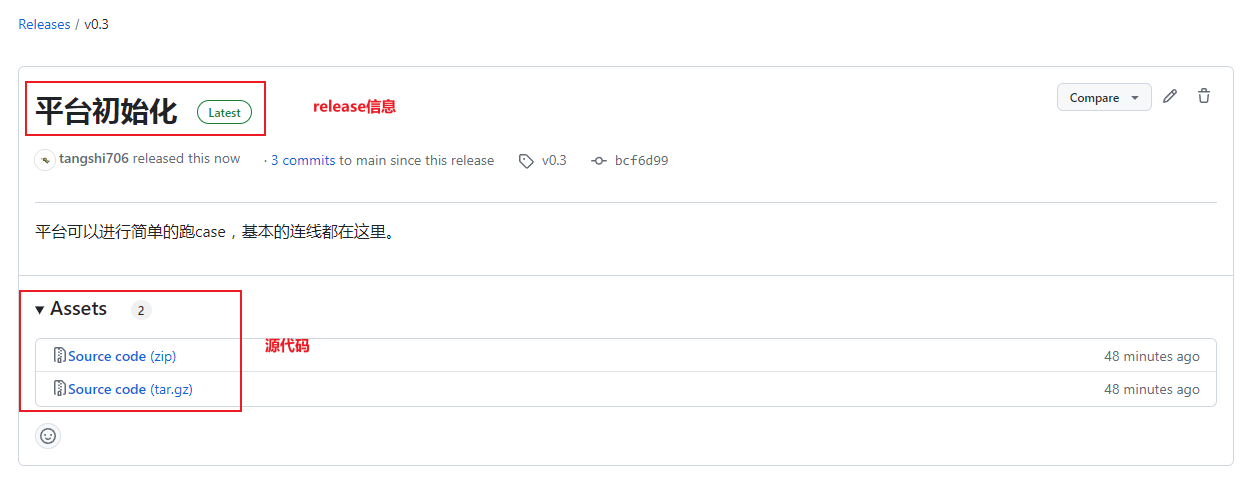


点击：





Release的信息：



可以主页查看到我们的设置的信息汇总：

