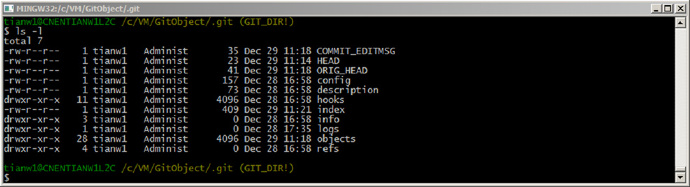
探索.git目录

# .git目录

下面就开始进入.git目录了，通过"ls"命令可以看到.git目录中的文件和子目录。对于这些文件和目录，下面给出了一些基本的描述。在后面后有logs、objects、refs、index和HEAD更详细的介绍



* hooks：这个目录存放一些shell脚本，可以设置特定的git命令后出发相应的脚本；在搭建gitweb系统或其他git托管系统会经常用到hook script
* info：包含仓库的一些信息
* logs：保存所有更新的引用记录（会在后面介绍引用）
* objects：所有的Git对象都会存放在这个目录中，对象的SHA1哈希值的前两位是文件夹名称，后38位作为对象文件名
* refs：这个目录一般包括三个子文件夹：heads、remotes和tags，heads中的文件标识了项目中的各个分支指向的当前commit
* COMMIT\_EDITMSG：保存最新的commit message，Git系统不会用到这个文件，只是给用户一个参考
* config：这个是Git仓库的配置文件
* description：仓库的描述信息，主要给gitweb等git托管系统使用
* index：这个文件就是我们前面文章提到的暂存区（stage），是一个二进制文件
* HEAD：这个文件包含了一个当前分支（branch）的引用，通过这个文件Git可以得到下一次commit的parent
* ORIG\_HEAD：HEAD指针的前一个状态

# Git引用

Git中的引用是个非常重要的概念，对于理解分支（branch）、HEAD指针以及reflog非常有帮助。Git系统中的分支名、远程分支名、tag等都是指向某个commit的引用。比如master分支，origin/master远程分支，命名为V1.0.0.0的tag等都是引用，它们通过保存某个commit的SHA1哈希值指向某个commit。

## 重新认识HEAD

HEAD也是一个引用，一般情况下间接指向你当前所在的分支的最新的commit上。HEAD跟Git中一般的引用不同，它并不包含某个commit的SHA1哈希值，而是包含当前所在的分支，所以HEAD直接指向当前所在的分支，然后间接指向当前所在分支的最新提交。

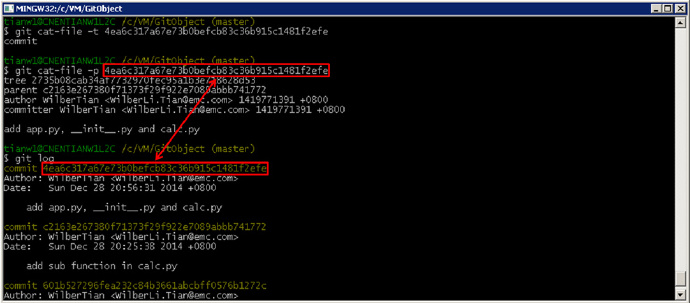
为了更形象的解释上面的描述，我们首先查看".git/HEAD"的内容：

ref: refs/heads/master

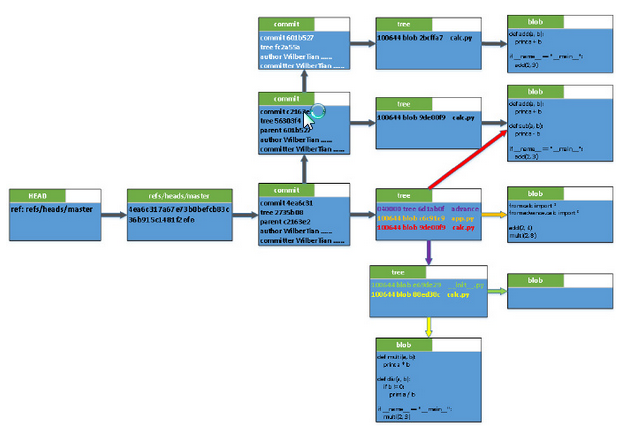
这就表示HEAD是一个指向master分支的引用，然后我们可以根据引用路径打开"refs/heads/master"文件，内容如下：

4ea6c317a67e73b0befcb83c36b915c1481f2efe

根据前面一片文章的介绍，我们通过这个哈希值查看对象的类型和内容，可以看到这个哈希值对应一个commit，并且通过"git log"可以发现这个commit就是master分支上最新的提交。

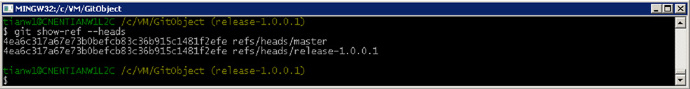


所以可以看到，所有的内容都是环环相扣的，我们通过HEAD找到一个当前分支，然后通过当前分支的引用找到最新的commit，然后通过commit可以找到整个对象关系模型，看下图：



## 引用和分支

假设我们现在除了master分支，又创建了一个release-1.0.0.1的分支，再次查看".git/refs/heads/"目录，可以看到除了master文件之外，又多了一个release-1.0.0.1文件，查看给文件的内容也是一个哈希值。通过"git show-ref –heads"命令就可以产看所有的头，这些都是HEAD的候选值：



根据前面的讲解，这个commit就是就是release-1.0.0.1分支上最新的提交。同样，当我们把当前分支切换到release-1.0.0.1的时候，HEAD文件的内容也会相应的变成：ref: refs/heads/release-1.0.0.1

# reflog详解

接下了，我们进入".git/logs"文件夹，可以看到这个文件夹也有一个HEAD文件和refs目录，些就是记录reflog的地方。查看HEAD文件的内容，发现这个文件将会包含所有分支的reflog记录：

|  |
| --- |
| 0000000000000000000000000000000000000000 601b527296fea232c84b3661abcbff0576b1272c WilberTian <Wilber\*\*\*.com> 1419759347 +0800    commit (initial): add calc.py into repo  601b527296fea232c84b3661abcbff0576b1272c c2163e267380f71373f29f922e7089abbb741772 WilberTian <Wilber\*\*\*.com> 1419769538 +0800    commit: add sub function in calc.py  c2163e267380f71373f29f922e7089abbb741772 4ea6c317a67e73b0befcb83c36b915c1481f2efe WilberTian <Wilber\*\*\*.com> 1419771391 +0800    commit: add app.py, \_\_init\_\_.py and calc.py  4ea6c317a67e73b0befcb83c36b915c1481f2efe 4ea6c317a67e73b0befcb83c36b915c1481f2efe WilberTian <Wilber\*\*\*.com> 1419822744 +0800    checkout: moving from master to release-1.0.0.1 |

进入".git/logs/refs"目录，同样会有master和release-1.0.0.1两个文件，两个文件将会保存各自分支的reflog记录

master的内容：

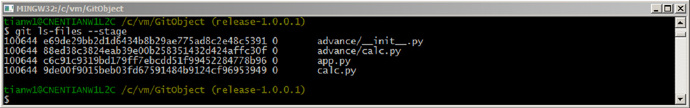
|  |
| --- |
| 0000000000000000000000000000000000000000 601b527296fea232c84b3661abcbff0576b1272c WilberTian <Wilber\*\*\*.com> 1419759347 +0800    commit (initial): add calc.py into repo  601b527296fea232c84b3661abcbff0576b1272c c2163e267380f71373f29f922e7089abbb741772 WilberTian <Wilber\*\*\*.com> 1419769538 +0800    commit: add sub function in calc.py  c2163e267380f71373f29f922e7089abbb741772 4ea6c317a67e73b0befcb83c36b915c1481f2efe WilberTian <Wilber\*\*\*.com> 1419771391 +0800    commit: add app.py, \_\_init\_\_.py and calc.py |

release-1.0.0.1的内容：

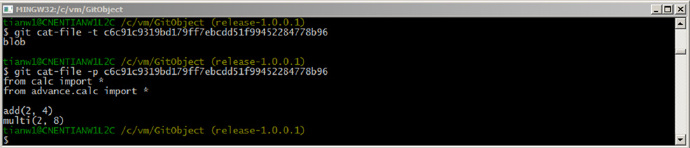
|  |
| --- |
| 0000000000000000000000000000000000000000 4ea6c317a67e73b0befcb83c36b915c1481f2efe WilberTian <Wilber\*\*\*.com> 1419822744 +0800    branch: Created from master |

# Git索引（index）

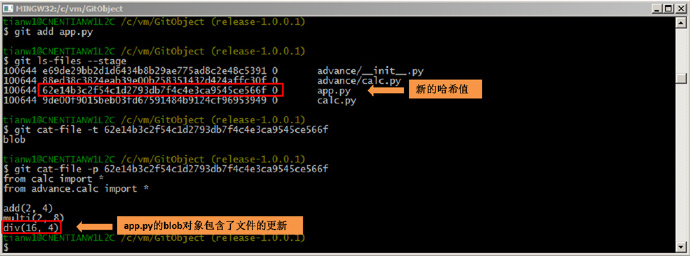
index（索引）示一个存放了已排序的路径的二进制文件，并且每个路径都对应一个SHA1哈希值。在Git系统中，可以通过"git ls-files --stage"来显示index文件的内容：



从命令的输出可以看到，所有的记录都对应仓库中的文件（包含全路径）。通过"git cat-file"命令查看app.py对应的哈希值，可以看到这个哈希值就是代表app.py的blob对象。



现在我们更新app.py文件，加上一个"div(16, 4)"的调用并通过"git add"添加到暂存区，这时发现index中app.py对象的哈希值已经变化了。



通过这个例子，我们也可以理解diff操作应该会有怎样的输出了：

* git diff：比较WorkSpace和stage，add之前有diff输出；add之后没有diff输出
* git diff HEAD：比较WorkSpace和repo，add之前之后都有diff输出
* git diff --cached：比较stage和repo，add之前没有diff输出；add之后有diff输出

# 对象的存储

前面提到所有的Git对象都会存放在".git/objects"目录中，对象SHA1哈希值的前两位是文件夹名称，后38位作为对象文件名。所以，我们前面提到的master上最新的commit对象的哈希值是"4ea6c317a67e73b0befcb83c36b915c1481f2efe"，那么这个对象会被存储在".git/objects/4e/a6c317a67e73b0befcb83c36b915c1481f2efe"。进入objects目录后，我们确实找到了这个文件。在Git系统中有两种对象存储的方式，松散对象存储和打包对象存储。

## 松散对象（loose object）

松散对象存储就是前面提到的，每一个对象都被写入一个单独文件中，对象SHA1哈希值的前两位是文件夹名称，后38位作为对象文件名。

## 打包对象（packed object）

对于松散存储，把每个文件的每个版本都作为一个单独的对象，它的效率比较低，而且浪费空间。所以就有了通过打包文件（packfile）的存储方式。

Git使用打包文件(packfile)去节省空间.。在这个格式中,，Git只会保存第二个文件中改变了的部分，然后用一个指针指向相似的那个文件。

一般Git系统会自动完成打包的工作，在已经发生过打包的Git仓库中，".git/objects/pack"目录下会成对出现很多"pack-\*\*\*.idx"和"pack-\*\*\*.pack"文件。关于打包就介绍这么多了，暂时还没有去研究两个文件的内容和原理。