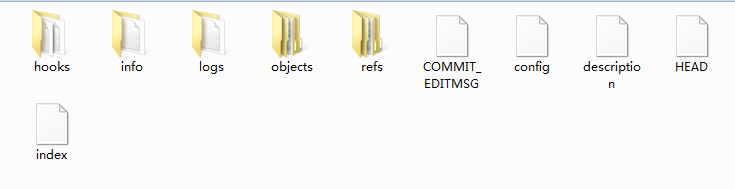
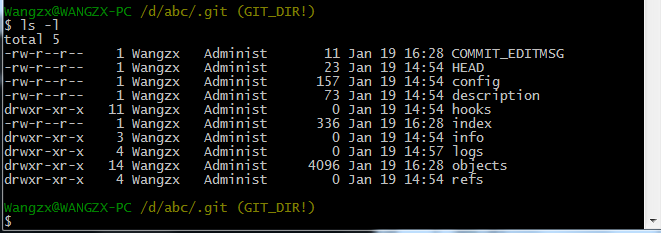
# **.git目录**

下面就开始进入.git目录，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。

* hooks:这个目录存放一些shell脚本，可以设置特定的git命令后触发相应的脚本；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
* 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档期分支（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**

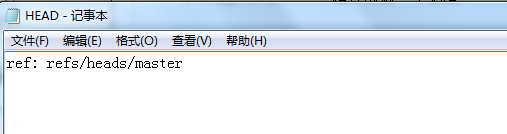
Git中的引用是一个非常重要的概念，对于理解分支（branch）、HEAD指针以及reflog非常有帮助。

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，它们通过该保存某个commit的SHA1哈希值指向某个commit

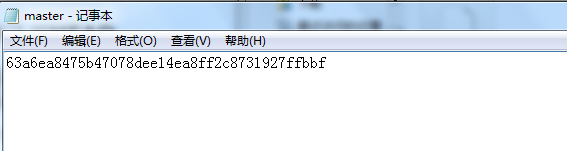
## **重新认识HEAD**

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，所有HEAD直接执行当前所在的分支，然后间接指向当前所在分支的最新提交。

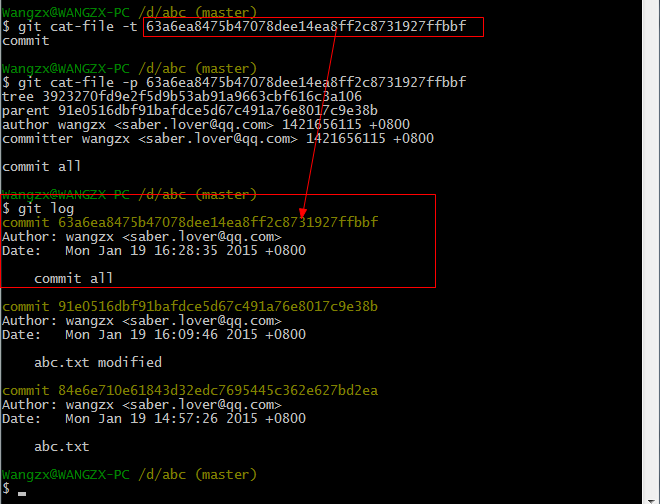
为了更形象的解释上面的描述，我们首先查看“.git/HEAD”的内容：

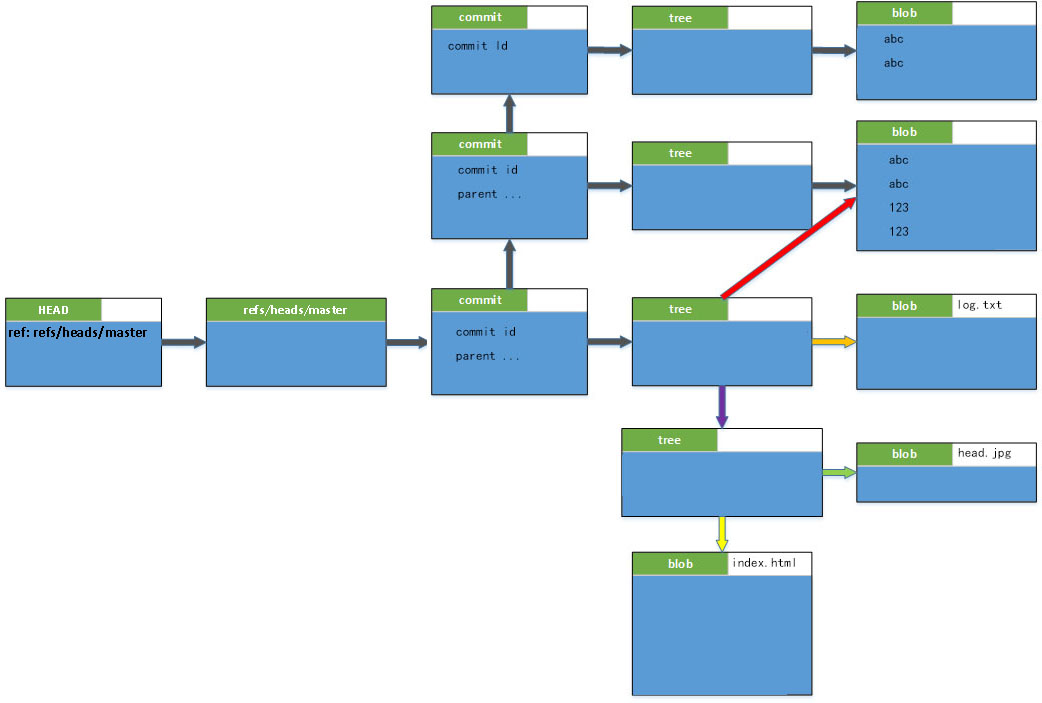


这就表示HEAD是一个指向master分支的引用，然后我们可以根据引用路径打开“refs/heads/master”文件，内容如下：



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雷人，可以看到这个哈希值对应着一个commit，并通过“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**

直到现在我们都没有开始介绍分支（branch），先大概展示一下引用和分支的关系。

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，查看该文件的内容也是一个哈希值。

通过“git show-ref --heads”命令就可以查看所有的头，这些都是HEAD的候选值：

IMG_262

HEAD文件的内容是commit的分支，当我们把分支切换到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

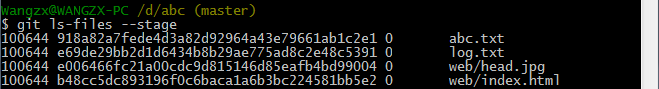
# **日志**

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，这些就是记录commit历史记录的地方。我们可以通过commit的哈希值，把repo退到一个指定的状态。

# **Git索引index**

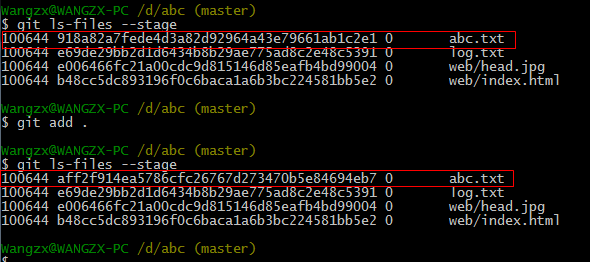
前面文章我们也提到过index/stage，就是更新的暂存区，下面来看看index文件。

index（索引）是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）。上面显示的哈希值就是abc.txt的blob对象的哈希值。

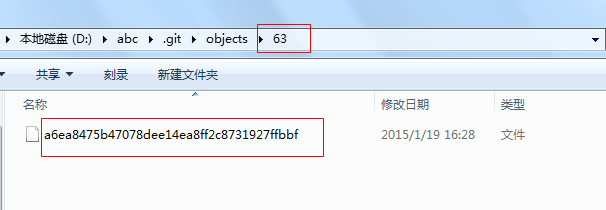
现在我们更新abc.txt文件，并通过“git add”添加到暂存区，这时发现index中的abc对象的哈希值已经变化了。



# **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。下面是之前提到的master最新的commit对象的哈希值：

63a6ea8475b47078dee14ea8ff2c8731927ffbbf



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）**

对应松散存储，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，object/pack目录下回成对出现很多“pack-\*\*\*.idx”和“pack-\*\*\*.pak”文件。

# **Git对象**

在Git系统中有四中类型的对象，所有的Git操作都是基于这四种类型的对象。

* "blob":这种对象用来保存文件的内容。
* "tree":可以理解成一个对象关系树，它管理一些"tree"和“blob”对象。
* "commit":指向一个"tree"，它用来标记项目某一个特定时间点的状态。它包括以下关于时间点的元数据，如时间戳、最近一次提交的作者、指向上次提交、
* "tag":给某个提交增添一个标记。

# **SHA1哈希值**

在Git系统中，每个Git对象都有一个特殊的ID来代表这个对象，这个特殊的ID就是我们所说的SHA1哈希值。

SHA1哈希值是通过SHA1算法计算出来的哈希值，对于内容不同的对象，会有不同的SHA1哈希值。使用commit id撤销更新，这里的commit id 就是一个SHA1哈希值。

# **Git对象模型实例**

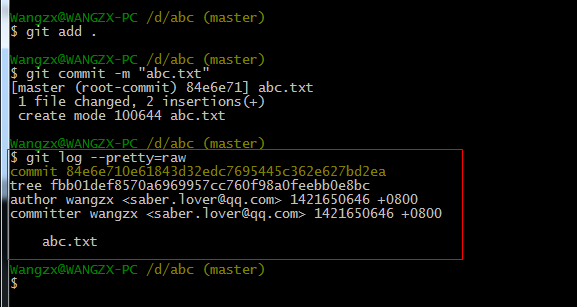
下面我们通过一个例子来认识一下上面的四种对象，为了更加清楚，这里将一步步展示经过一系列操作后对象的关系变化。

## ****第一步：新建一个仓库，添加一个“abc.txt”的文件，并提交到版本库。****

通过“git log --pretty=raw”可以得到每个commit的SHA1哈希值，也可以得到这个commit对应的tree的哈希值。

所有，一个commit对象一般包含以下信息：

* 代表commit的哈希值
* 指向tree对象的哈希值
* 作者
* 提交者
* 注释

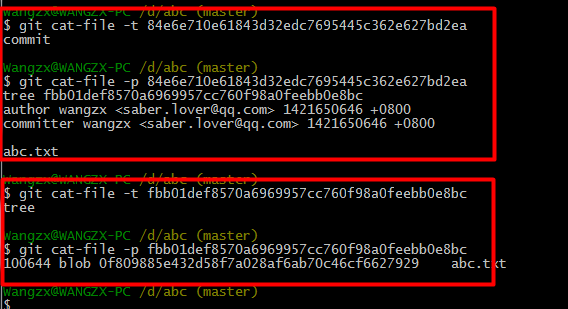


在Git对象模型的研究中，有一个很有用的命令“**git cat-file**”，可以通过该这个命令查询特定对象的信息：

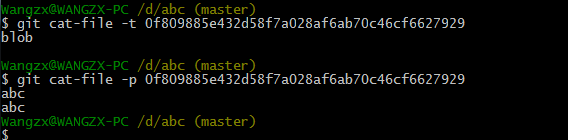
git cat-file -t key:通过一个对象的哈希值可以通过这条命令查看对象的类型（blob、tree、commit或tag）

git cat-file -p key:通过对象的哈希值可以查看这个对象的内容

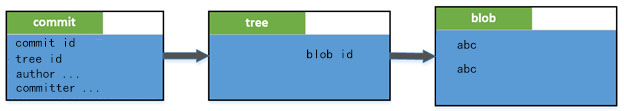
下面我们看一下在这次提交中commit、tree对象的类型和内容。这里可以看到，在这一次的commit中tree对象的内容指向一个blob对象。



同样，我们可以查看blob对象的类型和内容，这里可以看到blob的内容就是我们提交的文件内容。

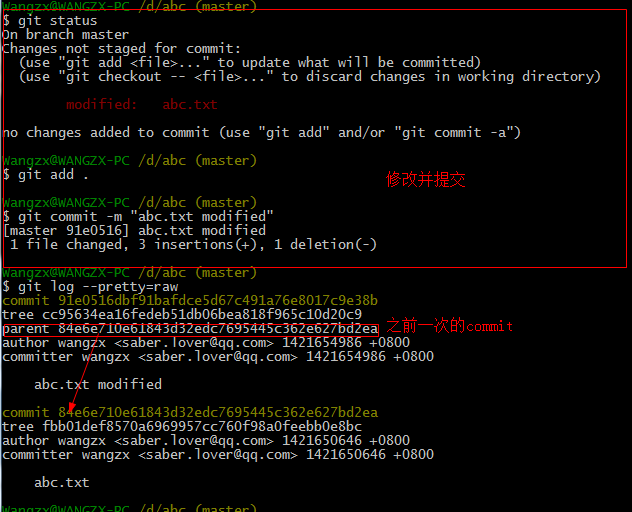


所以，根据上面的一些数据，我们可以看到在这一次的提交中产生了三个对象，同时看到了commit、tree、blob三个对象的关系如下：

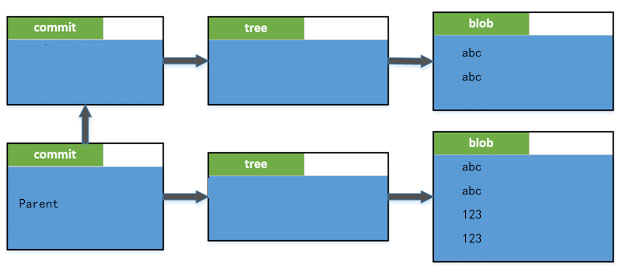


## ****第二步：更新“abc.txt”文件，并提交****

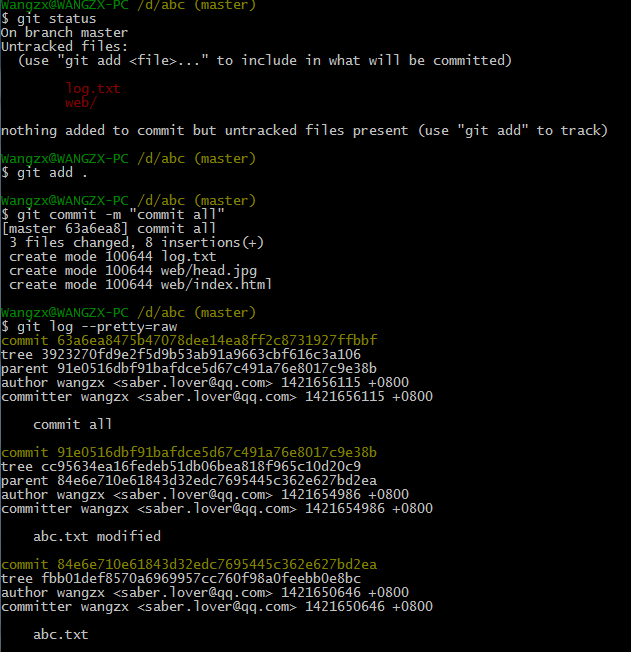
继续使用“git log --pretty=raw”查看commit log。



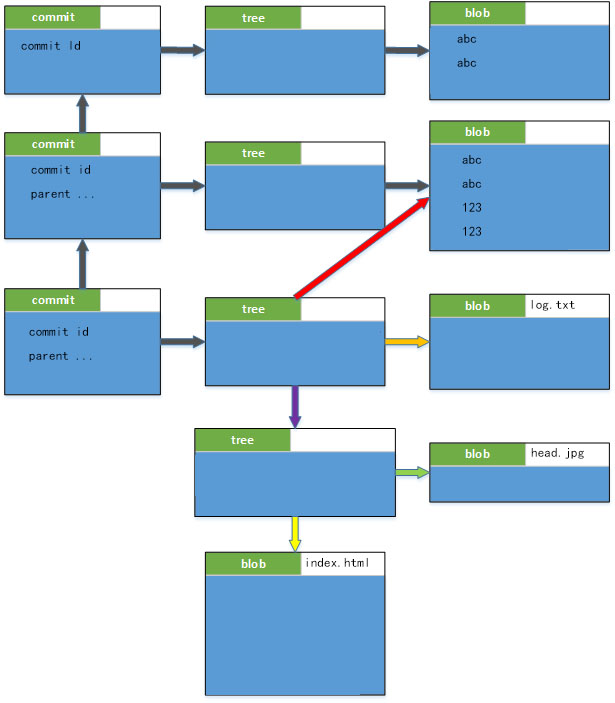
同样通过“git cat-file”我们可以看到每一个对象的类型和内容。这里需要注意的一点，Perforce、SVN和CVS属于“增量文件系统”，它们每次只存储提交之间的差异。而对于Git，它会把你每次提交的文件的全部内容都会记录下来。



## ****第三步：增加文件:"log.txt"。增加文件夹“web”，包括“index.html”和“head.jpg”。然后提交。****



同样的方式，我们可以得到一张更复杂的对象关系图：



## **总结**

Git对象模型就像是Git系统特有的文件系统，以特定的方式存储更新内容、元数据以及版本历史信息。