# Git 基础

## 记录每次更新到仓库

检查文件状态

git status

检查当前git项目中，和.git记录对比的差异。

跟踪新文件

git add

文件处于未跟踪状态时，使用该命令开始跟踪文件。

暂存已修改文件

git add

文件处于已修改状态，同样使用该命令将文件放入暂存区域。

忽略某些文件。

.gitignore

在git目录中添加一个该文件，并加入过滤规则文本，这样，指定的文件不会被跟踪，或显示到未跟踪列表。

查看已暂存的更新

git diff –cached

git diff –staged

两个是一样的效果，高版本蔡支持staged。表明当前暂存的文件与上次提交的文件之间的差异。

查看未暂存的更新

git diff

不带参数的就是表明未暂存的和暂存的差异了，其实就是已修改的内容和未跟踪的内容。

提交更新

git commit

git commit -m

git commit -a

第一个是提交，但是通过默认编辑器来输入提交信息。第二个是直接在命令后输入提交信息；第三个是跳过暂存区域，直接提交

## 查看提交历史

git log

git log -p

git log -3

git log –stat

git log –oneline

git log –pretty

git log –format

上述命令分别对应：查询全部提交记录、查询带差异的提交日志、查询最近几条记录、带统计行的提交记录、指定一条记录一行的提交记录、指定格式的提交记录、格式化的提交记录。

在使用过程中发现需要根据关键字来搜索记录，比如我需要搜索F4相关的命令，

git log –grep F4

比起可视化的tortoisegit或者gitk快多了。

而查看某个commit-id对应的信息时，可以用：

git show commit-id

可以查看这条commit-id对应的改动。

## 撤销操作

修改最后一次提交

git commit --amend

该命令将暂存区域合并到上一次提交。如果暂存区域没有变化，那么修改提交说明。如果暂存区域变化了，就合并到上次提交，可以修订提交说明。但是提交时间并没有变化。

撤销已暂存

git reset HEAD <file>

将暂存的文件撤销到已修改状态或者是未跟踪状态。

撤销已修改

git checkout -- <file>

将已经修改的文件撤销到未修改状态。

## 远程仓库使用

github创建仓库

密钥生成和配置

添加远端仓库

推送代码

克隆远端仓库

git clone url [remote-name]

查看当前的远程库

git remote  
git remote -v

git remote show [remote-name]

添加远程仓库

git remote add [shortname] [url]

抓取远端仓库数据

git fetch remote-name

推送数据到远端仓库

git push [remote-name] [branch-name]

抓取数据并合并

git pull [remote-name]

重命名

git remote rename old-name new-name

删除

git remote rm remote-name

## 标签操作

查看标签

git tag

添加带附注标签

git tag -a tag-name -m ‘XXX’

git tag -a tag-name commit-id

添加简单标签

git tag

显示某条标签信息

git show tag-name

上传标签到远端仓库

git push remote-name tag-name

git push –-tags remote-name

其他技巧

## 别名

git config –global alias.new-name name

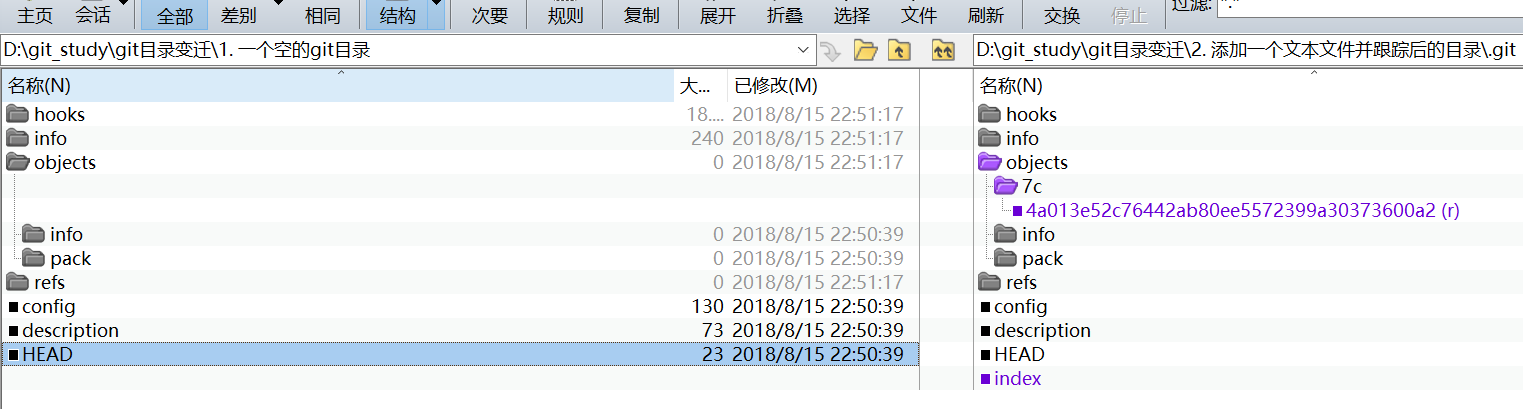
# 分支

## 何为分支

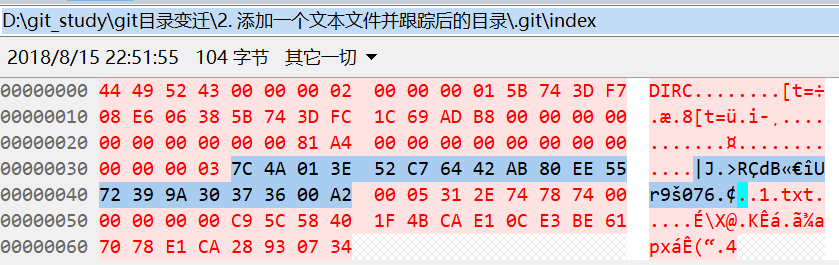
### 一次分支和提交操作的git目录变化

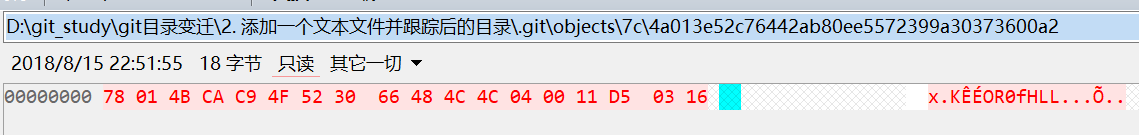
#### git init 对比 第一次add

左侧为使用git init后的目录，右侧为添加一个名为1.txt文本文件并git add暂存的目录。差异可见：右侧多了一个SHA1码的文件，以及一个index文件。



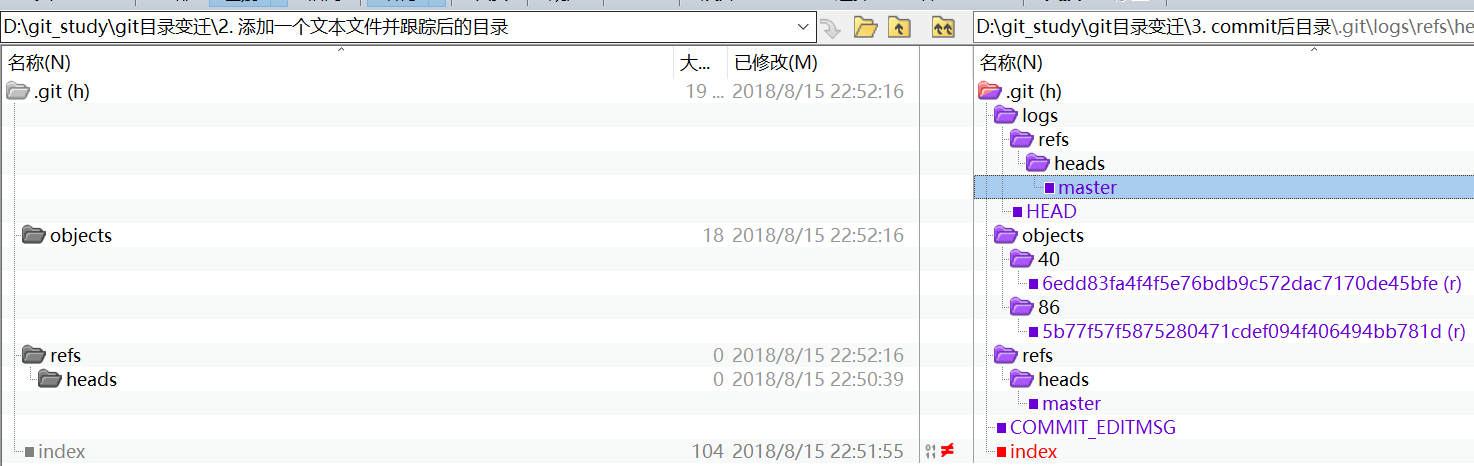
我们看下index文件的内容：其中选中的内容对应了这个SHA1码，后面包括文件名1.txt.



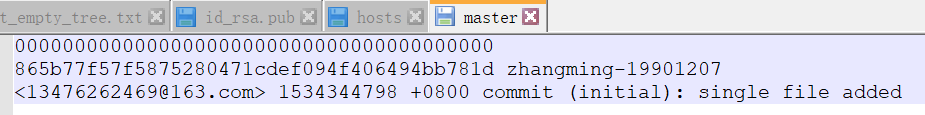


#### 第一次提交

提交之后的目录结果又发生了变化。注意commit-id为865b77f57f5875280471cdef094f406494bb781d。



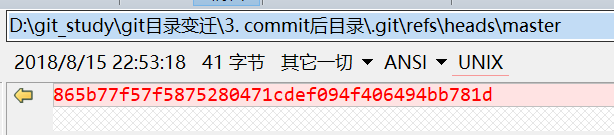
多了一个logs目录，其中HEAD，和一个master文件，内容是一模一样的。如下图：包括了commit-id的变化。以及本次commit的提交人名字和邮箱，以及提交注释。



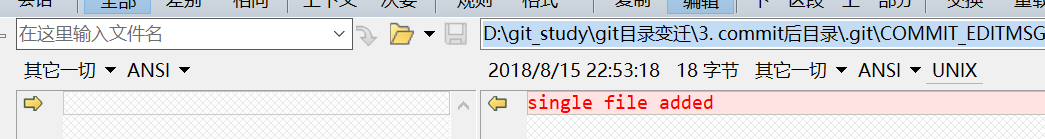
接着在objects中多了两个SHA1文件。

一个是本次commit-id命名的，另外一个暂时不清楚。

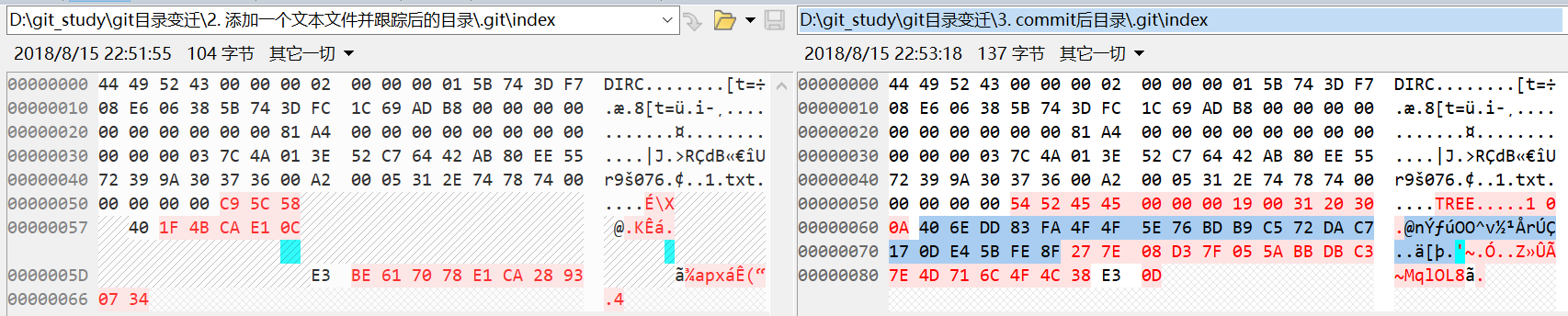
在refs中也多了一个master。他就是一个当前commit-id。

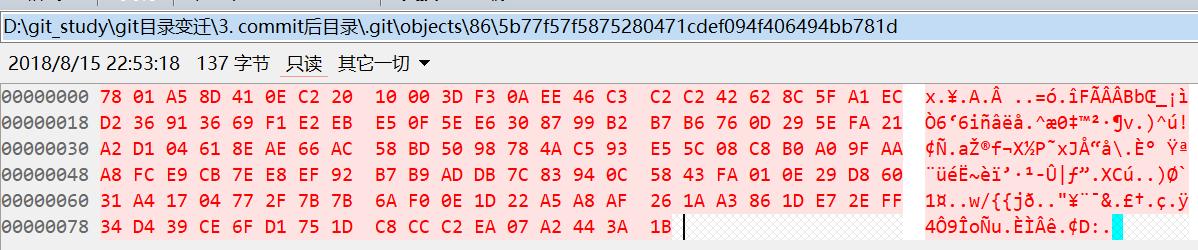


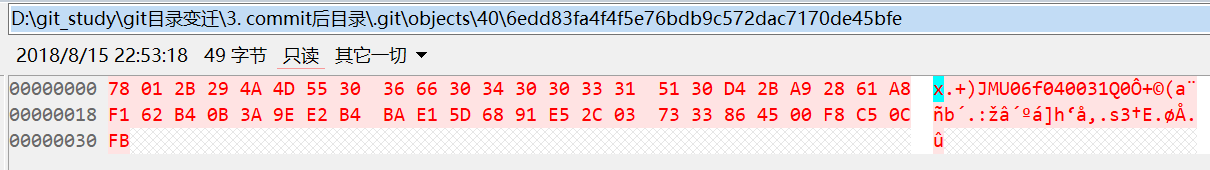
多了一个COMMIT\_EDITMSG。该文件保存着commit信息。



index文件也发生了变化。可以发现多了一个TREE字符串，然后选中的区域展示了一个文件的SHA1（该SHA1不是commit-id）。然后之前的SHA1文件也保存在里面。

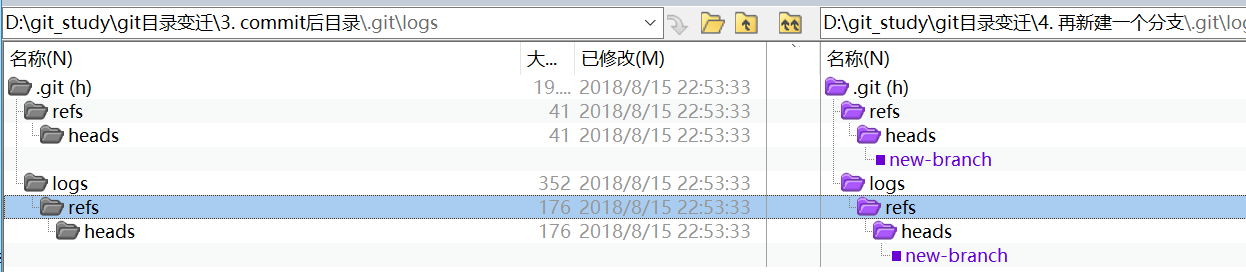




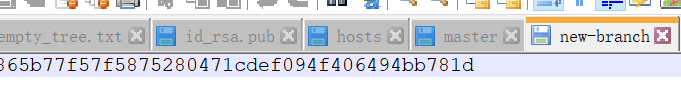


#### 新增分支

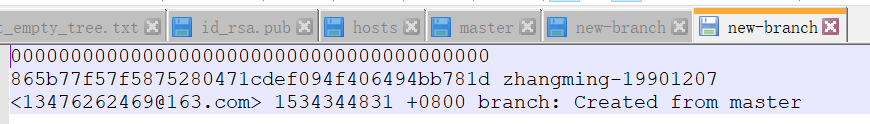
新增一个叫做new-branch的分支。可见在两个refs目录下多了一个分支名字的文件。



在.git下的refs目录就保存了一个commit-id

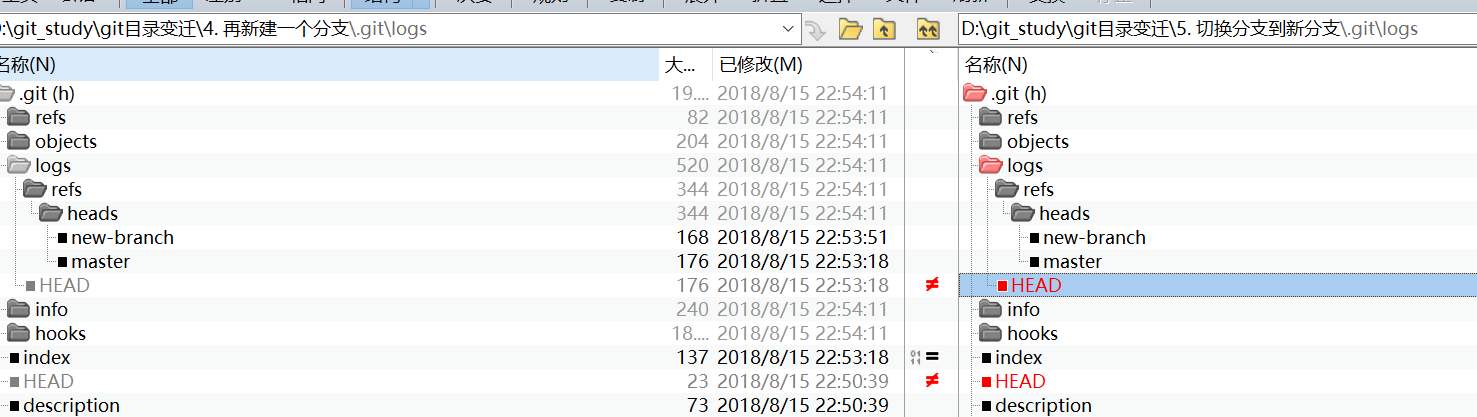


在logs下的refs目录下保存了commit-id变化，还有提交信息，以及分支来源。

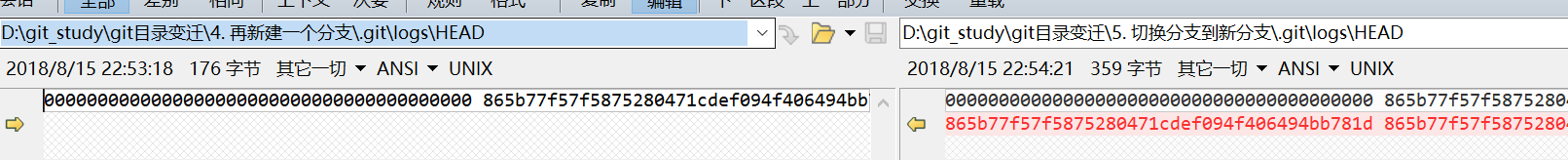


#### 切换分支

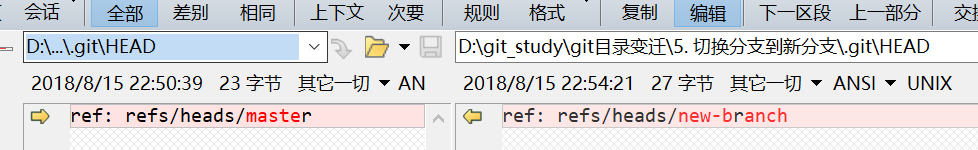
见两个HEAD文件发生了改变。



上面的HEAD多了一条信息，就是当前的commit信息，以及分支切换信息。

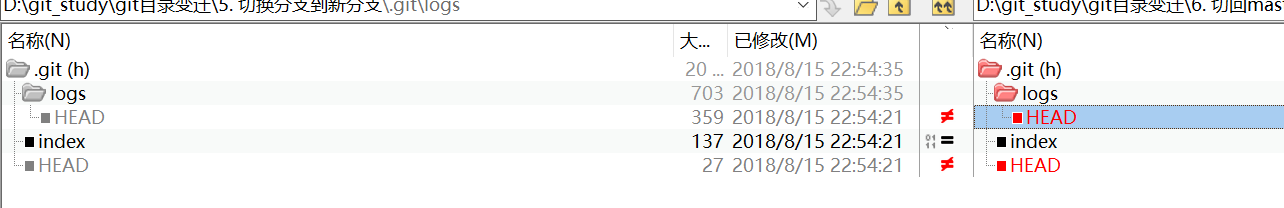


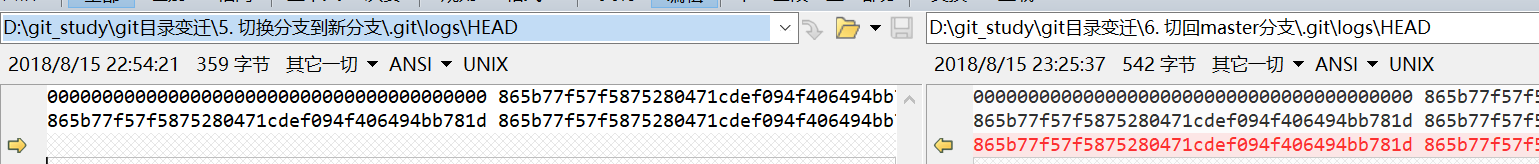
下一个HEAD改变了，就是说当前的分支指向改变了。

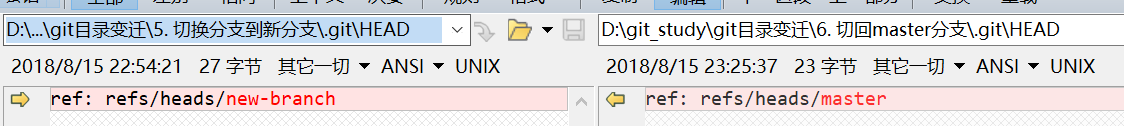


#### 切换回master分支

可以看到又是两个HEAD文件改变了，git下的记录了新的分支名，而log下又增加了一次切换。

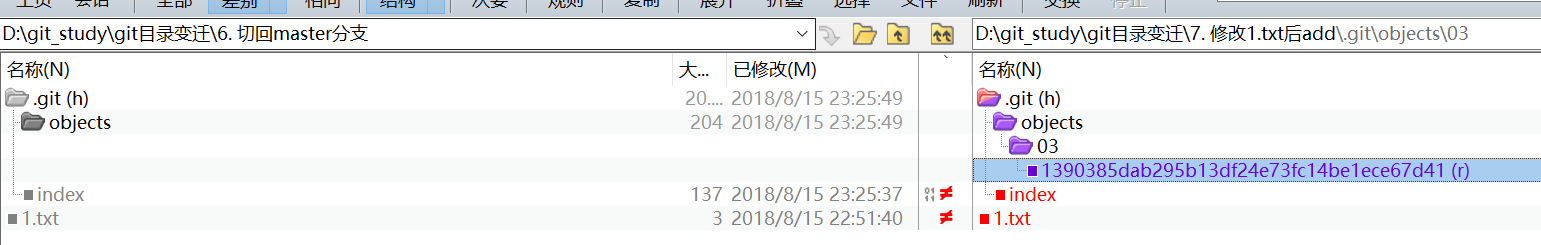




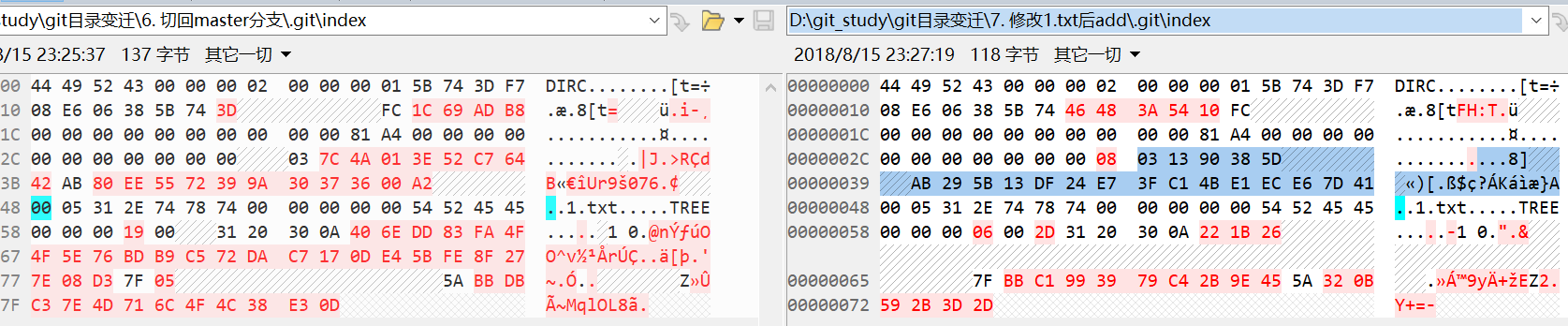


#### 再次add一个修改

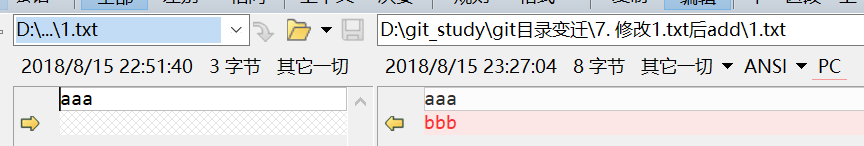
又多了一个SHA1文件。



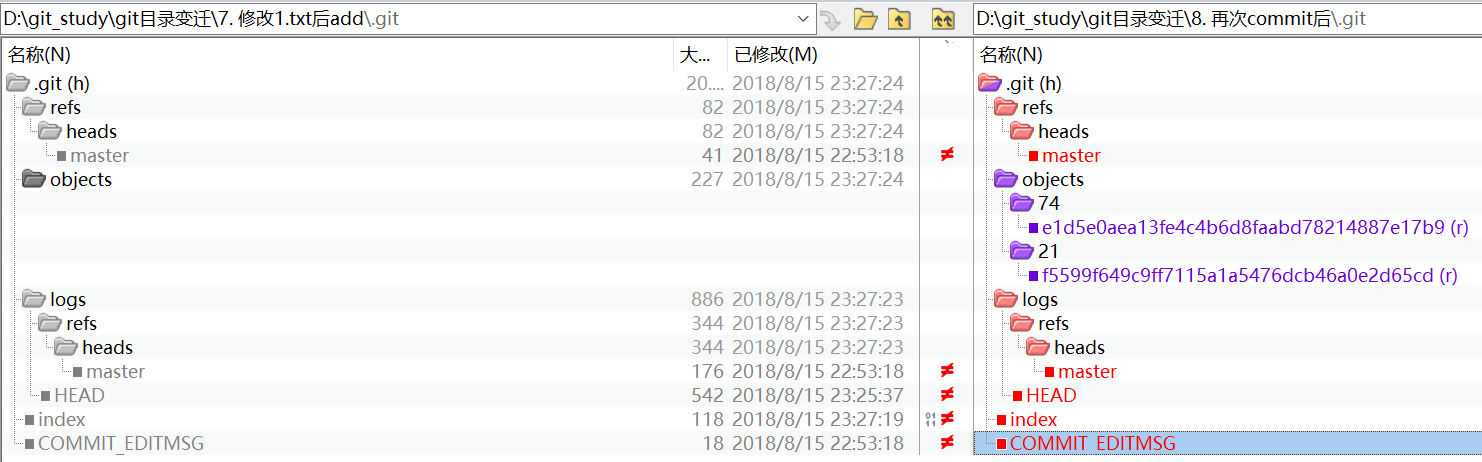
index变化如下，显然多了当前的SHA1码。而第一次add的SHA1码这时候没有了。



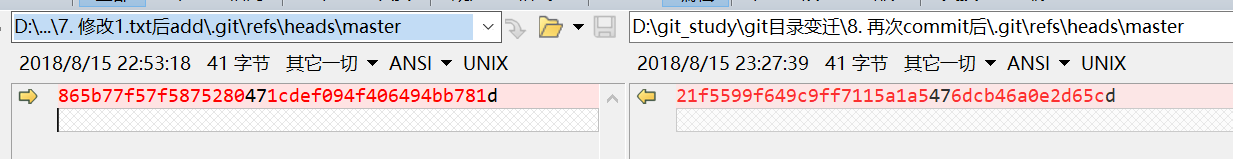
文本改动如下：



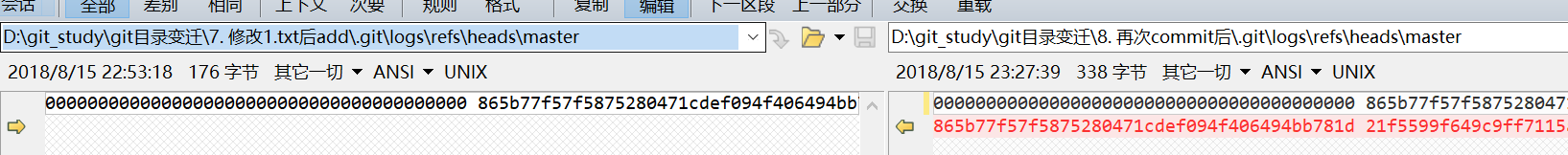
#### 第二次commit



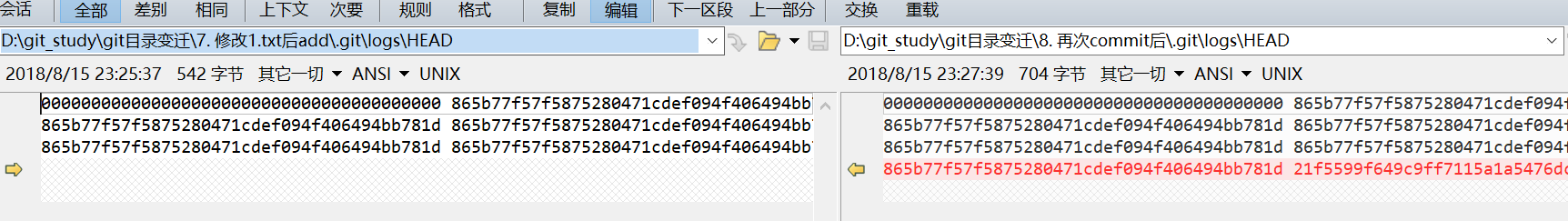
master文件记录着当前的commit-id，那么其发生了变化。在git bash中确认commit-id就是21f5599f649c9ff7115a1a5476dcb46a0e2d65cd。



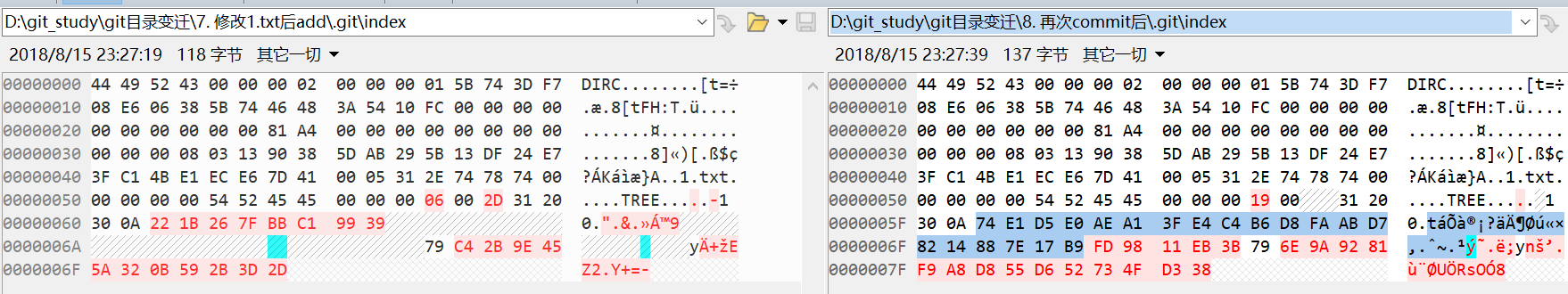
另一个master记录了commit-id的变迁：



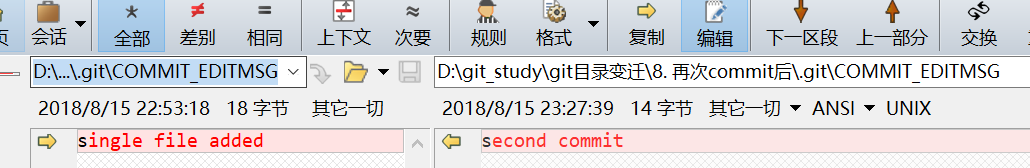
而HEAD文件记录了当前工作分支的commit-id变迁：



index发生了变化，其中包含了SHA1，不是commit-id的那个。

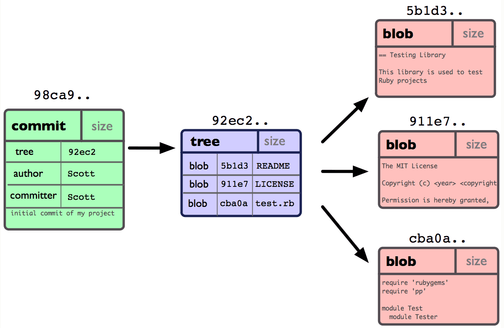


COMMIT\_EDTMSG则记录是最新一次commit的提交信息：

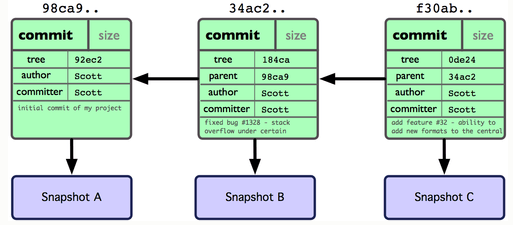


### Git如何保存数据

一个具有三个文件的目录提交后，对应的结构如下。一个commit对应一个对象，包含快照印象数的指针、提交信息。一个tree对象，就是表示目录的快照，里面包含各个子文件blob对象的文件快照索引；还有三个记录文件快照的blob对象。



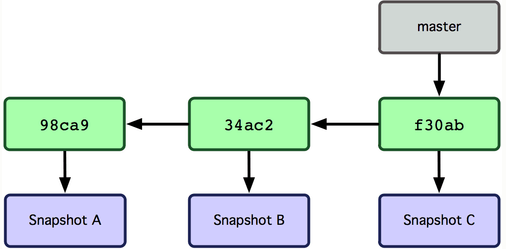
再做两次提交后，仓库中保存的数据如下：



每个commit对象都有一个parent对象，就是上次提交对象。首次提交没有，有的有多个。

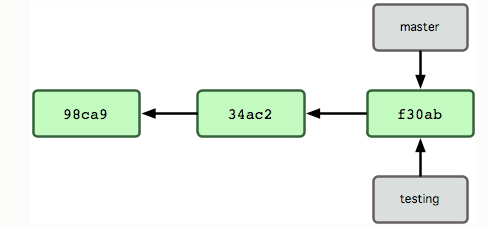
### 分支实质

分支实际上就是一个commit对象的可变指针，提交后会自动跟着前进。分支就是commit对象往回看的历史。



### 创建分支

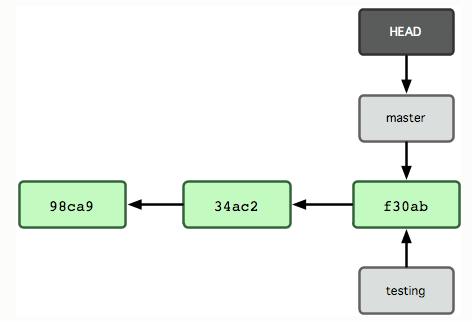
git branch 分支名



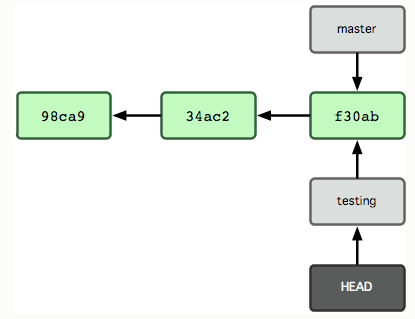
创建一个分支，就是创建一个指向当前commit对象的指针。从2.1.1来看创建分支改动非常小。

### HEAD指针

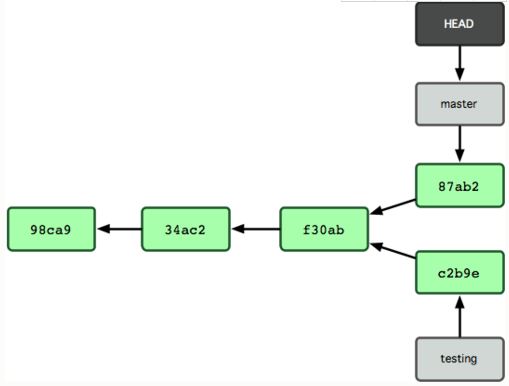
HEAD指针是一个指向正在工作的分支指针的指针。git branch创建分支时，并没有切换分支，因此HEAD分支没有变化，依然指向master。



当我们切换分支到tesing的时候，干两件事，第一个，HEAD指向testing分支对象，第二个，将testing的快照内容提取出来。



那么我们在testing上提交一次，并切换回master，进行一次提交，结果如下：



## 分支创建和切换

创建,会基于当前的commit-id创建一个分支

git branch branch-name

切换，切换到新的分支，切换前需要确保工作目录是干净的，就是都提交了的。否则不能切换。

git checkout branch-name

还有个一步到位的方式，创建+切换

git checkout –b branch-name.

删除分支,如果要删除的分支，没有完全合并到当前分支，那么使用大写D来强制删除。

git branch –d branch-name

git branch –D branch-name

## 分支管理

查看分支信息

git branch

查看分支信息，包含最后一次commit信息

git branch –v

查看已经合并到当前分支的分支，这个列表里面的非当前分支，都可以直接删除，如果不关注的话。

git branch –merged

查看没有合并到当前分支的分支。这里的分支如果不关注，只能使用-D强制删除。因为，他还有没有合并过来的信息，因此，不能随意删除。

git branch –no-merged

## 利用分支进行开发的工作流程