学Py，得永生

Git 跟踪本地仓库中的任何一个文件，每一个没有跟踪的文件都会标红，已经被跟踪的就会标绿，任何修改都会是一个信息，并不仅仅只有在提交文件的时候才会用到commit，git要求每一次修改都会是一个信息，都要提交到本地仓库，

也就是说即使，我一个文件没有添加到本地文件夹中但是我重命名了一个文件。这样gie也会提示你提交修改后的信息的

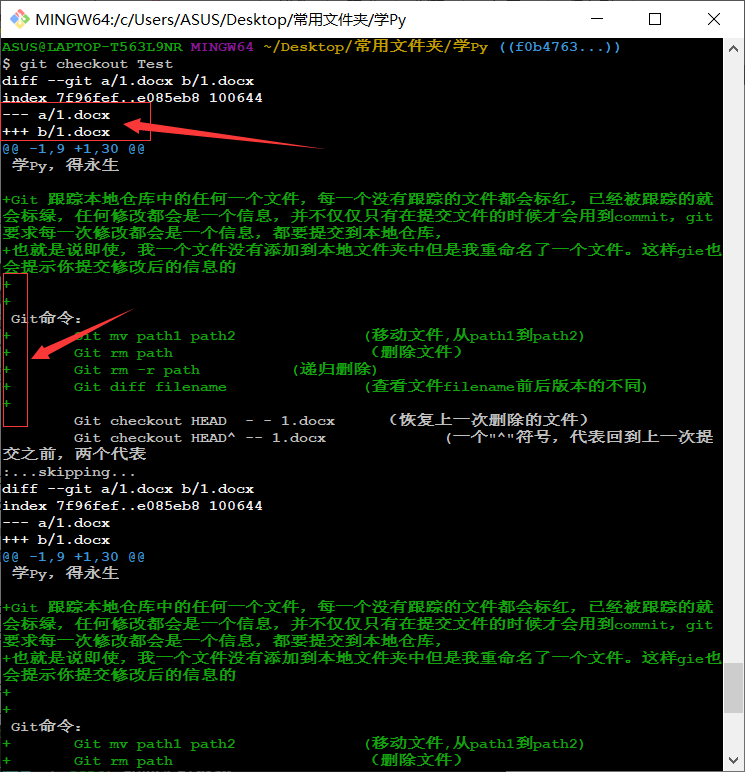
Git命令：

**Git mv [path1] [path2]**  (移动文件,从path1到path2)

**Git rm [path]**  （删除文件）

**Git rm -r [pat]h**  (递归删除)

**Git diff [filename]**  (查看文件filename前后版本的不同)



--- a的意思就是-代表的是之前的版本

+++ b代表的是后来修改的版本

下面的绿色字体都是后来添加的

可以发现每一行绿色的字体前面都是带着+号的

**Git checkout HEAD –- [filename]**  （恢复上一次删除的文件）

**Git checkout HEAD^ -- [filename]**  (一个“^”符号，代表回到上一次提交之前，两个代表

回到上两次提交之前）

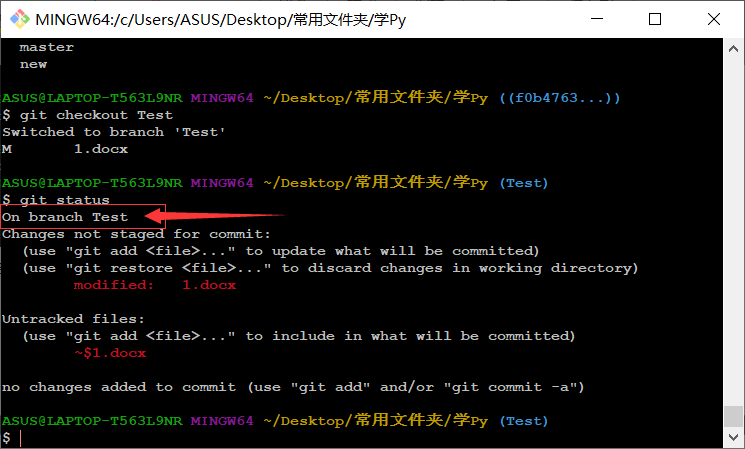
**Git revert id**(那一次提交的id就在提交日志的前面都有，比如第一行的id就是f788180)----------返回id所指向的那次提交，恢复提交



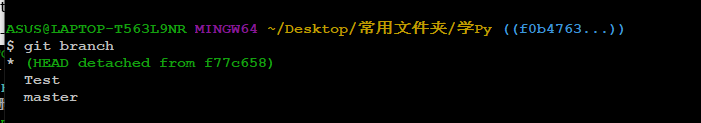
**Git reset id （重新指向头指针）**

分支：每一个分支都是独立的，在Test分支上的操作不会影响其他分支上的内容

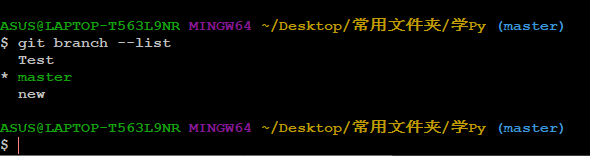
查看当前分支：**git status**



查看所有分支：**git branch**-----分支前面的’\*’号代表的是当前在哪个分支上



Git branch --list 项目分支的列表



创建分支：**git branch [name]**(分支的名字)



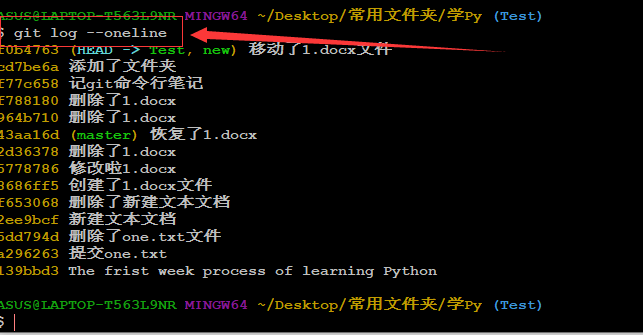
切换分支：**git checkout [name**(分支名称)**]**





查看提交日志命令：git log （详细查看所有提交信息）

将每一次的提交日志简略为一行输出**：git log –oneline**  (粗略查看，注意oneline紧跟在—后面)

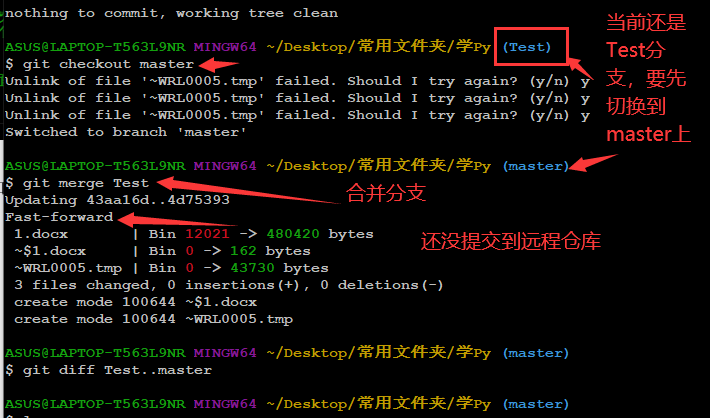


查看两个分支之间的区别：git diff **branch1..branck2 [要对比的文件名，不加则显示两个分支的所有区别]**



合并分支merge：**git merge [branch]**(分支的名字)

注意这里合并之前要先切换到**要合并到的分支上，比如说，要将Test分支合并到master分支上，那么就是在合并之前先切换到master分支上**

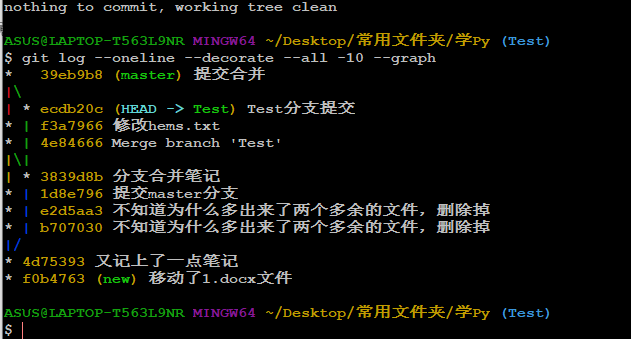


查看两个分支的不同，

发现没有不同，

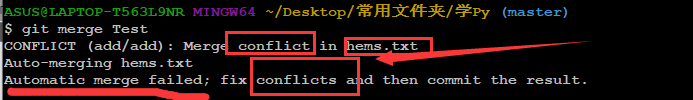
因为已经合并了

提交查看日志：**git log –oneline –decorate –all -10 -graph**(注意这里的10前面是一个‘-’，表示查看十条提交日志)



关于前面的那些符号，表示的是啥意思，没听懂

**Conflict合并时的冲突：符号<<<<和=====以及>>>>>>>直接的内容在两个分支上发生了冲突，也就是说在两个分支上的第一行都有内容，但是在合并的时候，git不知道要保留哪一个，所以就会造成冲突，解决方法就是在修改自己想要修改的内容，然后再当前分支上直接提交，然后，解决冲突问题了**



这句提示的意思大致就是：

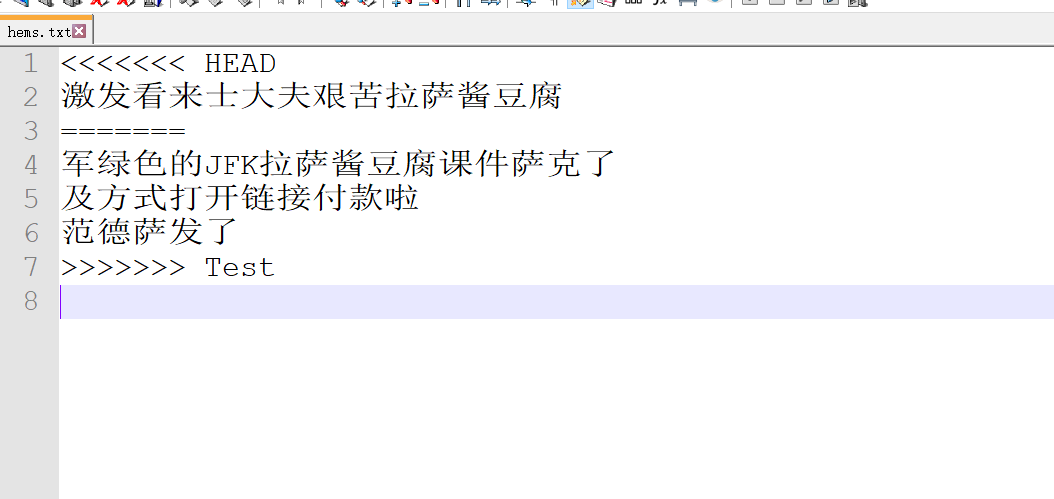
就是提醒你把你，让你手动把当前分支上的hems.txt文件提交一次，当然这个操作不影响其他分支

合并失败了就是这样

在合并的时候，hems.txt文件发生冲突

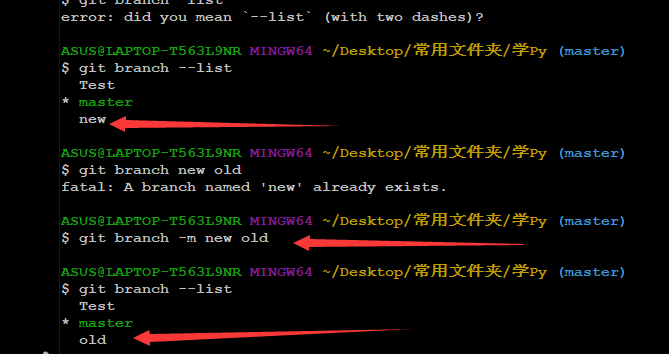
自动合并hems.txt文件

自动合并hems.txt文件失败；合并冲突，然后需要手动提交结果



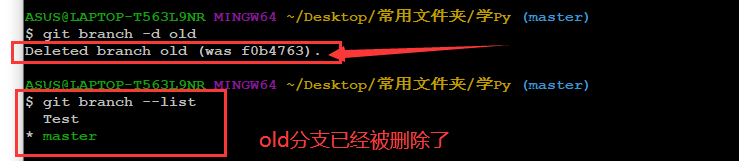
重命名分支：

**Git branch -m [old\_name\_branch] [new\_name\_branch]**



删除分支：

**Gir branch -d [branchname]**



Stash命令：

**保存当前工作进度，使用场景：有两个文件1.txt和2.txt都被修改了，而且都存放在了暂存区（add），提交的时候，不想把这两个文件一起提交，那么就使用stash命令，**

**Git stash**