# 学Py，得永生

## 初步大体理解：

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基本命令：

### g**it 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)

### git add 命令

注意add命令提交是从工作区提交到暂存区，而且只是提交已经发生变化的文件（空文件夹也不算），

#### git add [filename]

提交单个文件

#### git add .

提交全部的修改文件

### git commit -m’ 描述’

从暂存区提交到本地仓库

### git status

**查看当前工作状态以及当前所在分支（有多少修改了没提交的，有多少提交到了暂存区没有提交到本地仓库的），**

### git push -u origin master

从本地仓库提交到远程仓库

### git pull origin master

从远程仓库拉取一下

### git clone origin master

从将远程仓库的分支克隆到本地，

个人开发的话，这个一般是克隆master分支上的，因为在github远程仓库中一般不建立其他分支，都是在本地仓库建立的其他分支，然后当其他分支做好了之后合并到master分支上，然后提交到远程仓库中，

当然在团队开发中，可能每一个人都有自己的分支，最后每个人的分支都完成之后，就会在GitHub的远程仓库中合并分支

最后，项目完成

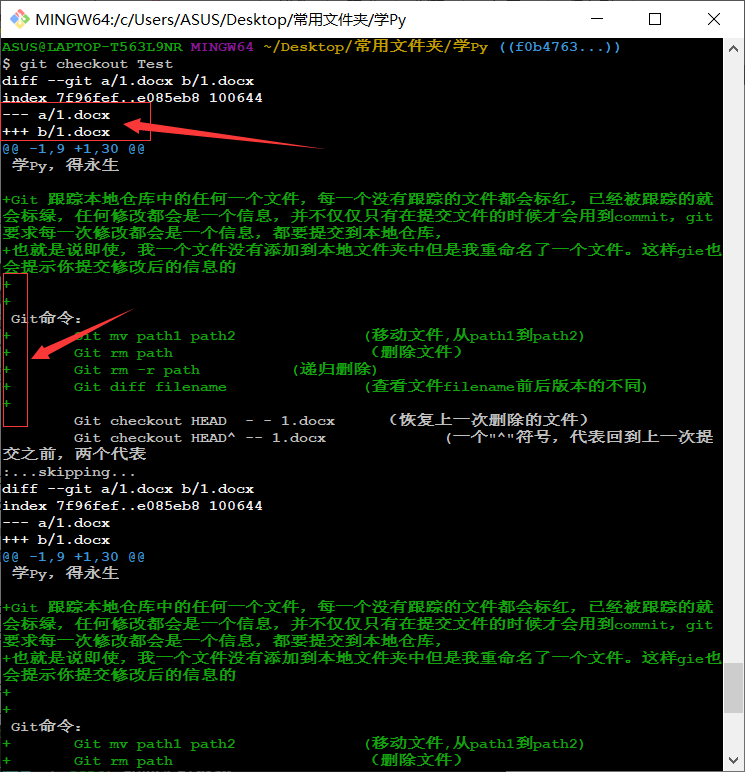
## 恢复版本命令

### Git checkout HEAD -- [filename]

**（恢复上一次提交删除的文件）**

### Git checkout HEAD^ -- [filename]

**(一个“^”符号，代表回到上一次提交之前，两个代表)**



--- 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，两个代表回到两次提交之前）

Git revert id

**Id 就是版本号**

(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

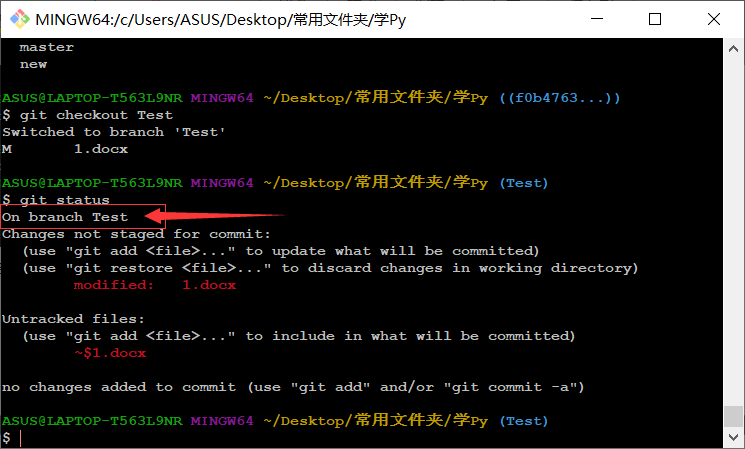


### Git reset id

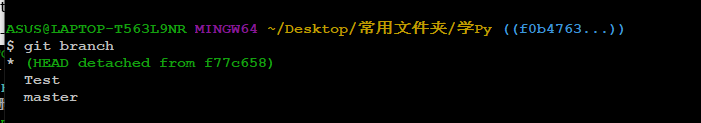
**（重新设置头指针HEAD指向，一般来说头指针指向最新的版本，也就是最后一次提交的版本）**

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

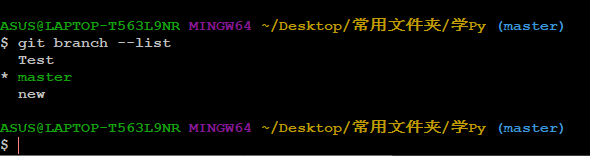
查看当前分支：**git status**



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)**]**



## Log命令

### Git log

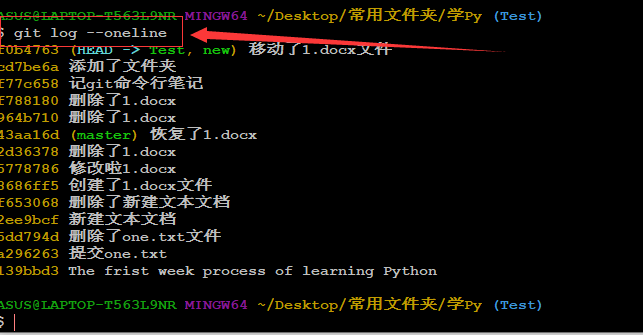


查看提交日志命令：git log （详细查看所有提交信息）

**显示详细日志信息**

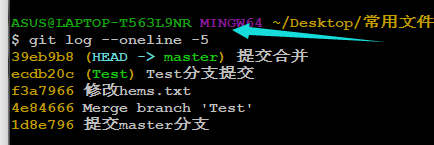
### git log –oneline

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)



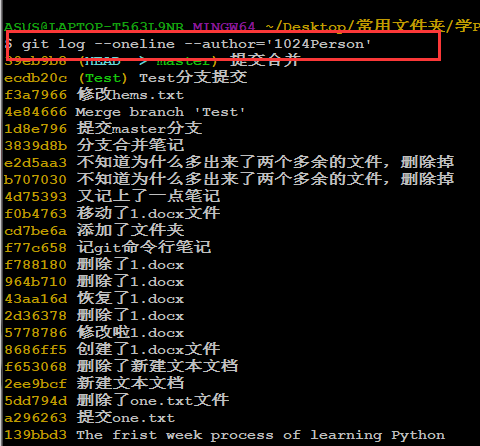
### git log –oneline -n

控制输出的函数**：git log –oneline -5 显示最近五条提交日志**



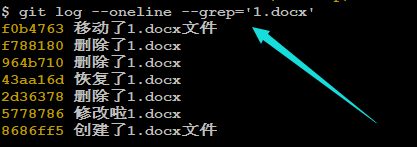
### Git log --oneline --author=’username’

查看某个提交者的提交：git log –oneline –author=’1024Person’



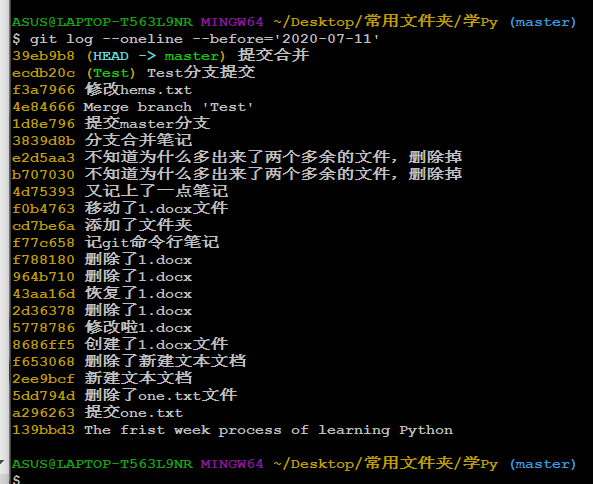
### Git log --oneline --grep=’keyword’

指定关键字进行查找：git log –oneline –grep=’1.docx’



### Git log –oneline --before=’time’

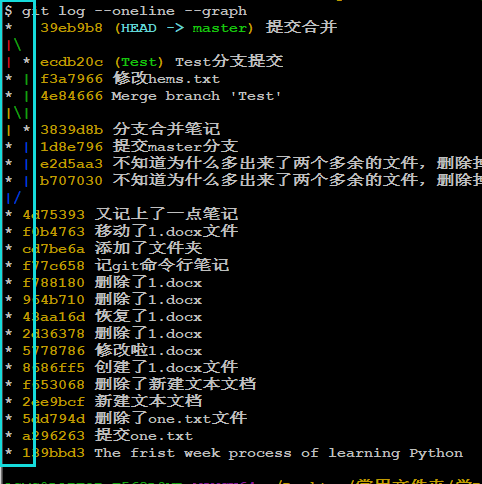
**按时间找出提交日志git log –oneline –before =’2020-07-11’(当前日期：2020-07-10)**



### Git log --oneline --before=’1 week’

显示一周前的，还可以1day、2day，或者1 month，2month，

### Git log –oneline –graph

加上一点图形

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**

### 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

## alias命令(取别名)

**给常用命令设置别名，便于记忆，**

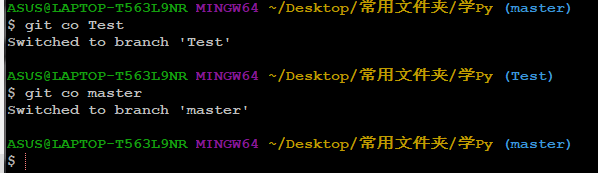
**注意：**

### 在用户区域下

**例：git config –global alias.co checkout-----------------------🡪给checkout命令起别名叫做co**

**global是表示在用户区域下，而不是在系统区域下**

**alias.co 表示co是checkout命令的别名**



**可发现使用 git co Test也能切换分支**

### 在系统区域下设置别名：

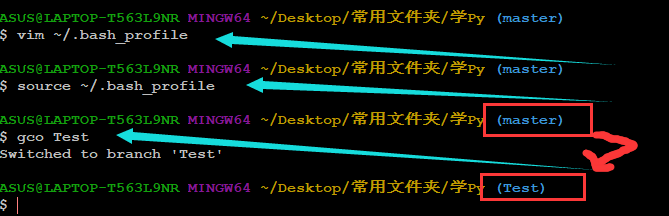
**vim ~/.bash\_profile**

**alias goc= ‘git checkout’ （在bash\_profile文件中编辑的）**

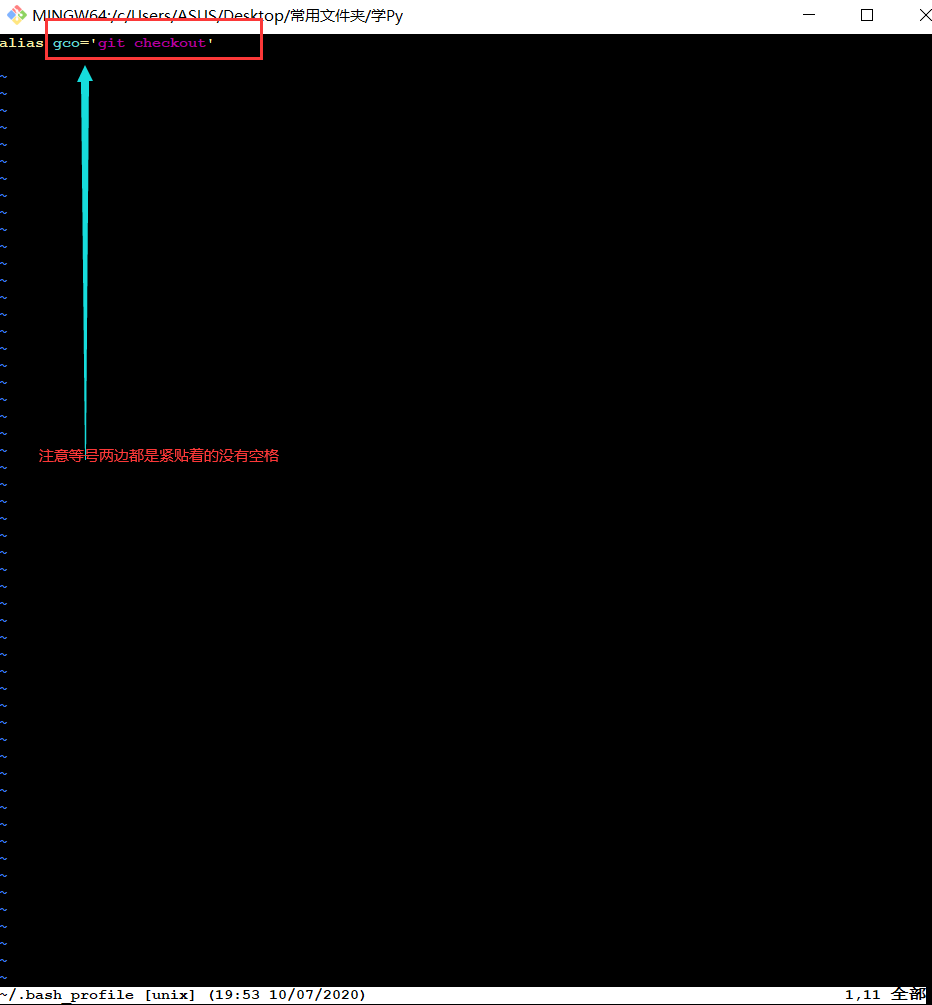
**source ~/.bash\_profile 好像是编译一下设置好的文件**

### 区别

**用户和系统的区别：在系统区域下设置别名：在任意本地仓库中都能使用**



**.bash\_profile隐藏文件**



## Ignore命令(还未完成)

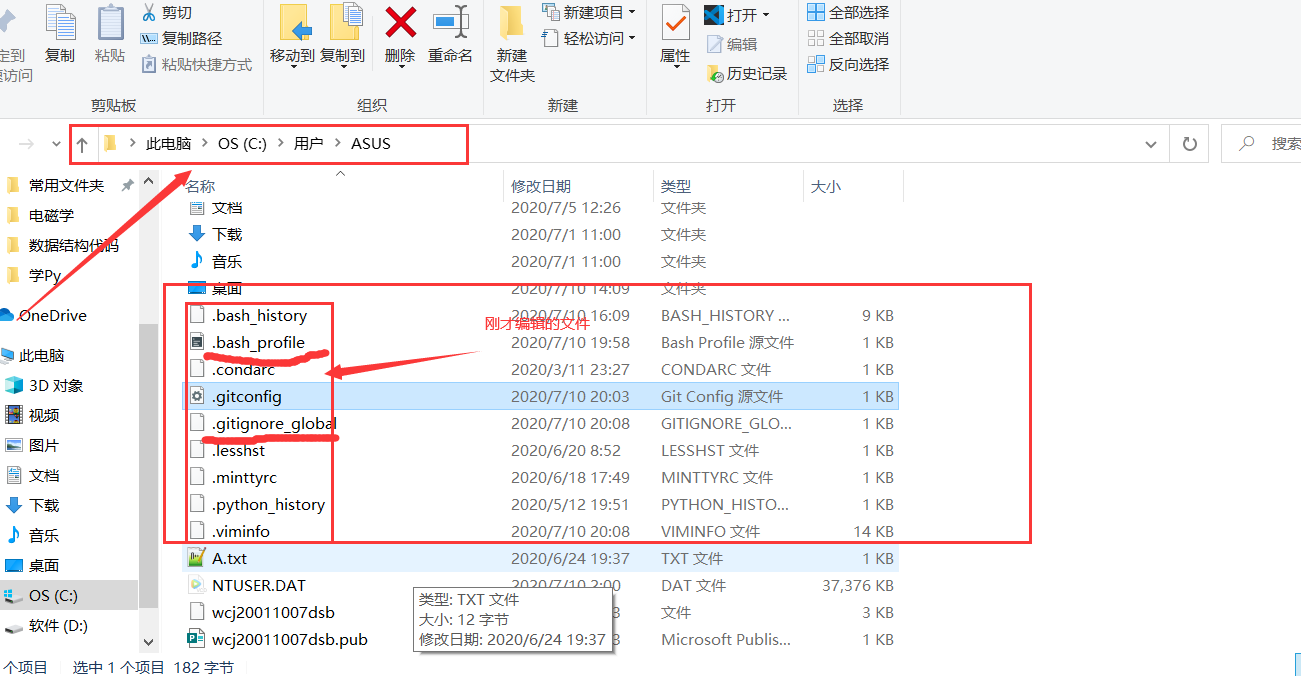
**告诉git忽视的文件，不要让git跟踪的文件**

### git config –global core.excludesfile ~/.gitignore\_global

**这句话的意思就是，在文件.gitignore\_global文件中储存着不让git跟踪的文件列表，这是直接在系统下设置的，无论哪个用户那个仓库都有效**

**这里的.gitignore\_global是一个文件，在根目录下**

**Vim ~/.gitignore\_global ：编辑忽视**

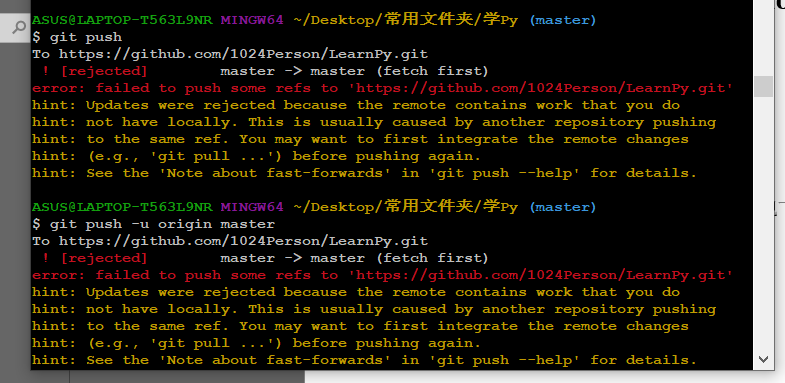


## push命令

**注意：和远程库交互的一定是本地库**

### 错误总结1：

在向远程仓库中推送push的时候总是出现下面的错误：



通过提示，使用了git push -u origin master结果还是错误！！

最后从网上找到了一个解决方法

### 解决方法1：

#### git push -u origin +master

### 错误总结2：

有的时候会提示，本地分支与远程分支版本不一致的问题，推送失败

### 解决办法2：

从远程仓库pull一下就好了，

git pull origin master

在次提交应该就可以了

### origin

**在本地仓库连接远程仓库的时候，要提供网址，而origin就是这个网址的别名，下面就是创建这个别名的操作(注意这个origin只是惯例，可以叫不同的名字，一个网址的名字可以叫不同的别名)：**

**Git remote add origin** **https://github.com/1024Person/LearnPy.git**

如果必为这个网址取一个别名的话每一次push的时候都要指定网址，很麻烦

git push -u [**https://github.com/1024Person/LearnPy.git**](https://github.com/1024Person/LearnPy.git) **master**

**就是 网址 分支，但是为每一个网址取一个别名就好了**

**注意这里的，origin就是一个名字，想取什么就取什么，origin只是一个惯例**

## branch相关命令

### git diff [branchname\_1]..[branchname\_2]

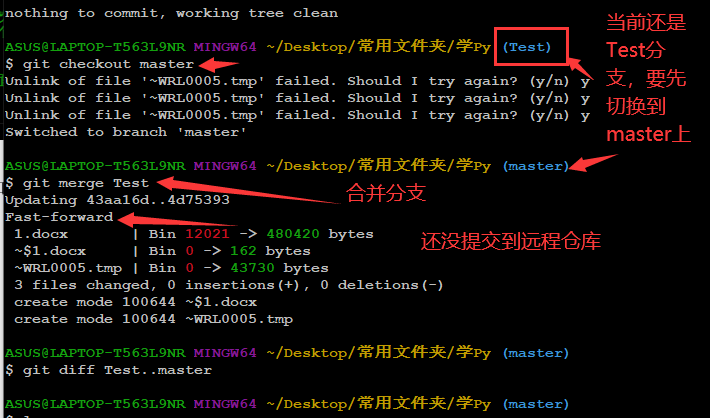
**查看两个分支的不同**



### git merge [branch](分支的名字)

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**

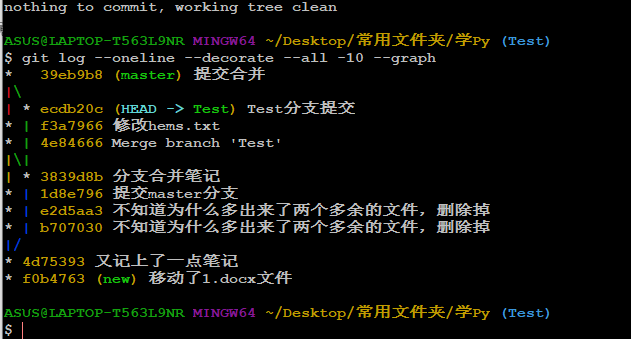


查看两个分支的不同，

发现没有不同，

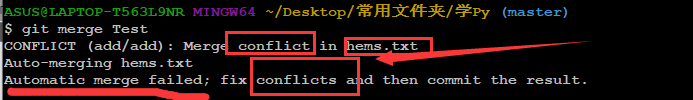
因为已经合并了

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：

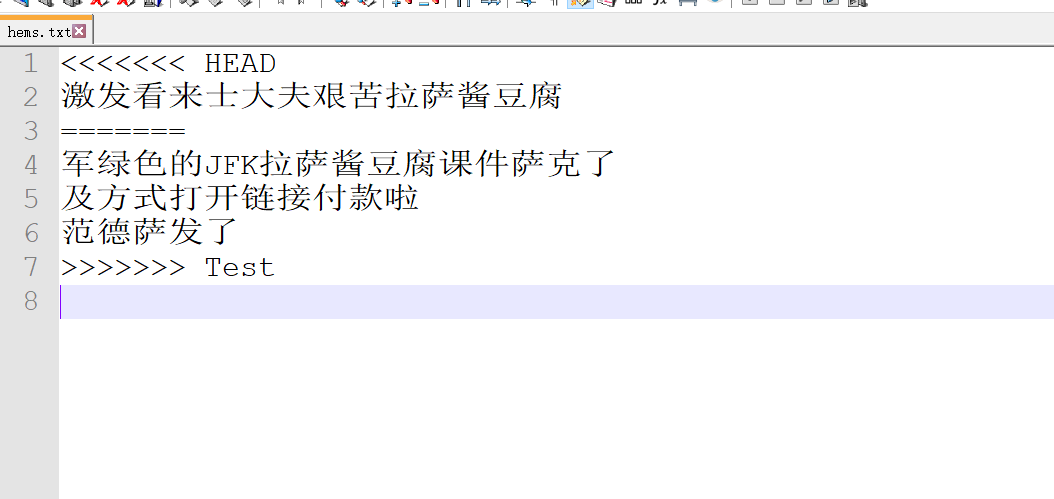
就是提醒你把你，让你手动把当前分支上的hems.txt文件提交一次，当然这个操作不影响其他分支

合并失败了就是这样

在合并的时候，hems.txt文件发生冲突

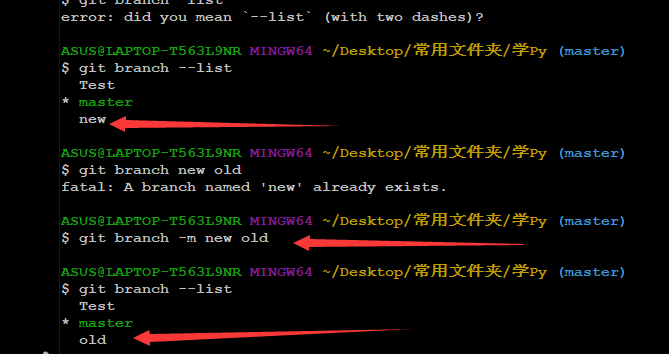
自动合并hems.txt文件

自动合并hems.txt文件失败；合并冲突，然后需要手动提交结果



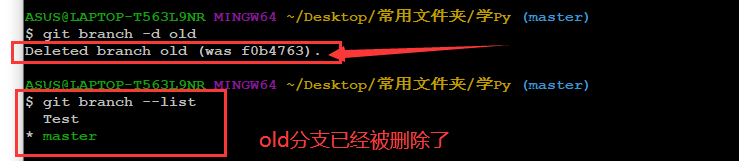
### Git branch -m [old\_name\_branch] [new\_name\_branch]

**重命名分支：**



### Gir branch -d [branchname]

删除分支：



## Stash命令：

**保存当前工作进度，使用场景：有一个文件1修改了，而且都存放在了暂存区（add），提交的时候，不想把这个文件一起和以后要来的文件一起提交，那么就使用stash命令，**

### Git stash save ’描述’

将工作进度保存下来（就是返回到上一次刚提交的状态）

### Git stash list

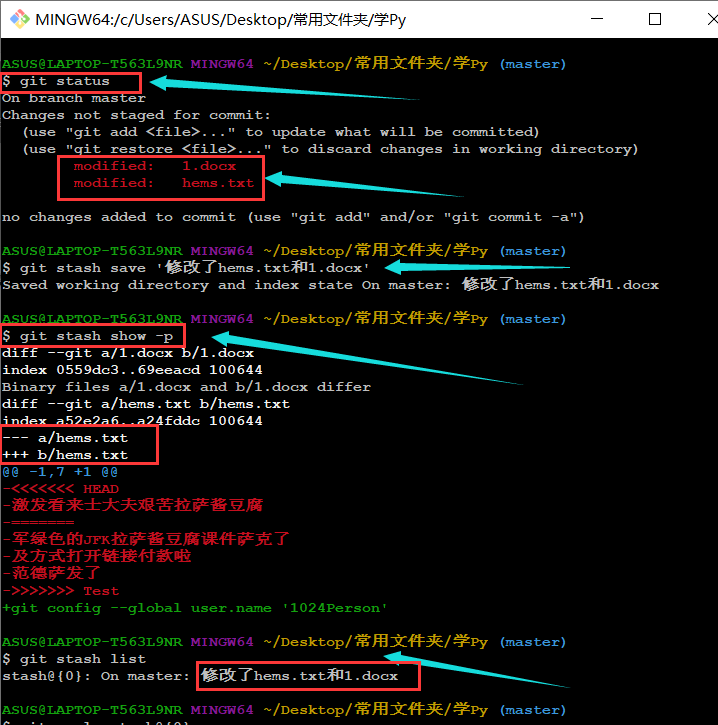
**显示保存的工作进度，注意：最前面的|“stash@{0}”|这个是每个工作进度的代号，如果想要复原某一个工作进度的时候，还要用到这个工作代号**

### Git stash show -p

**显示修改的内容，+表示增加了，-表示删除了比如像下面**

**红色字体的 – 表示修改的时候，将这些内容删除了，**

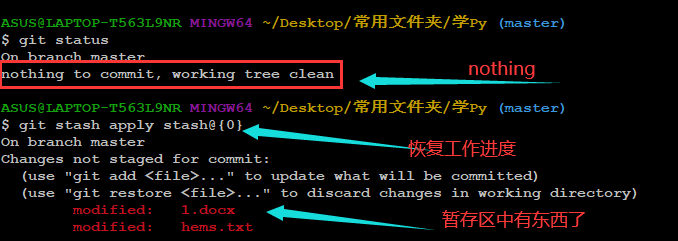
**绿色字体的 + 表示修改的时候，将这些内容添加了。**



### Git stash apply stash@{n}

**恢复代号为{n}的工作进度**

**注意{}里面的n代表的是保存的工作进度的代号**



### Git stash drop stash{n}

**删除第n个工作进度**

### Git stash pop stash{n}

**删除第n个工作进度**

