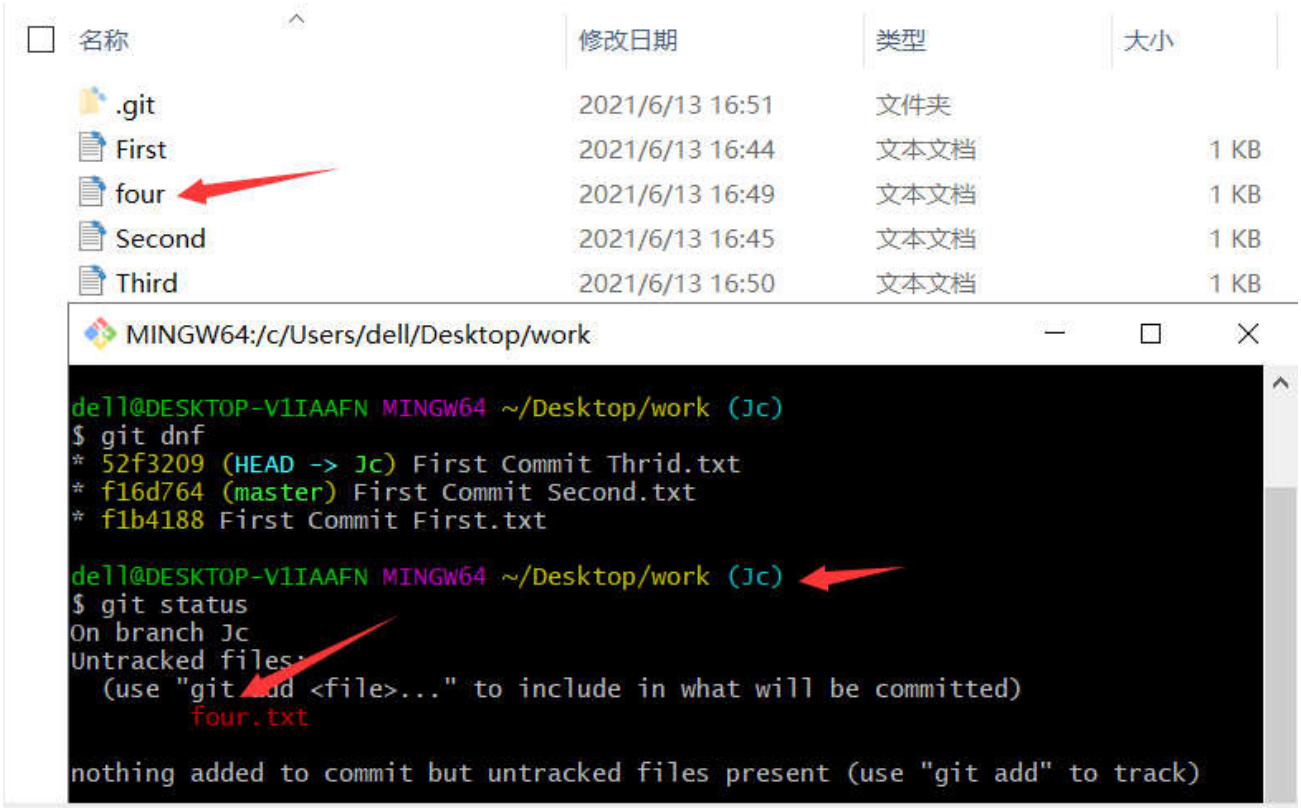
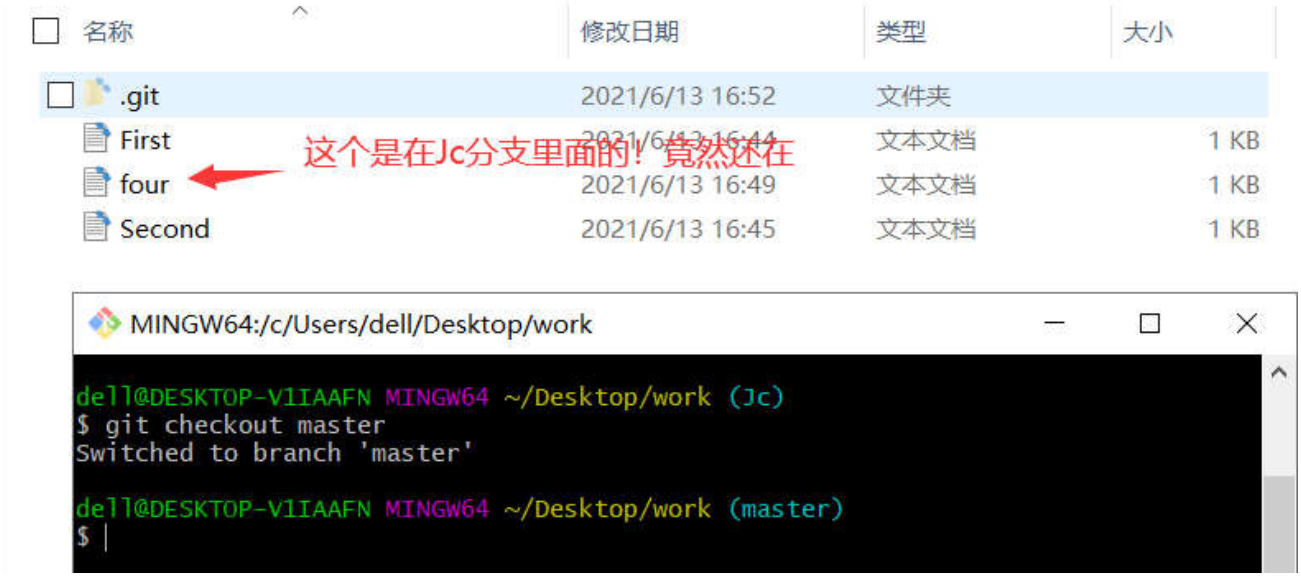
前言

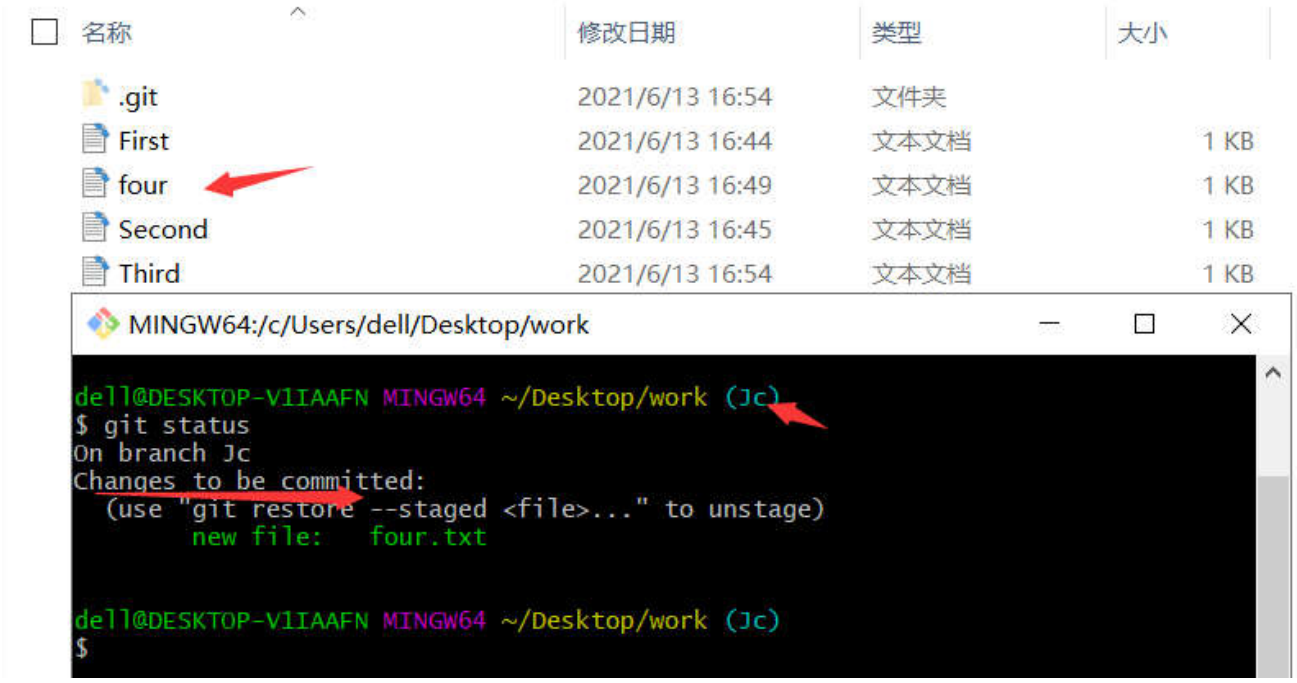
众所周知，前几天要考英语六级的没办法还是用了几天复习了一下英语(该死的六级，真难~)，然后就拖更了~这一次做一次分支实战演练也算是巩固所学的知识，废话不多说开始咯~在演练之前先来思考几个小小的问题。

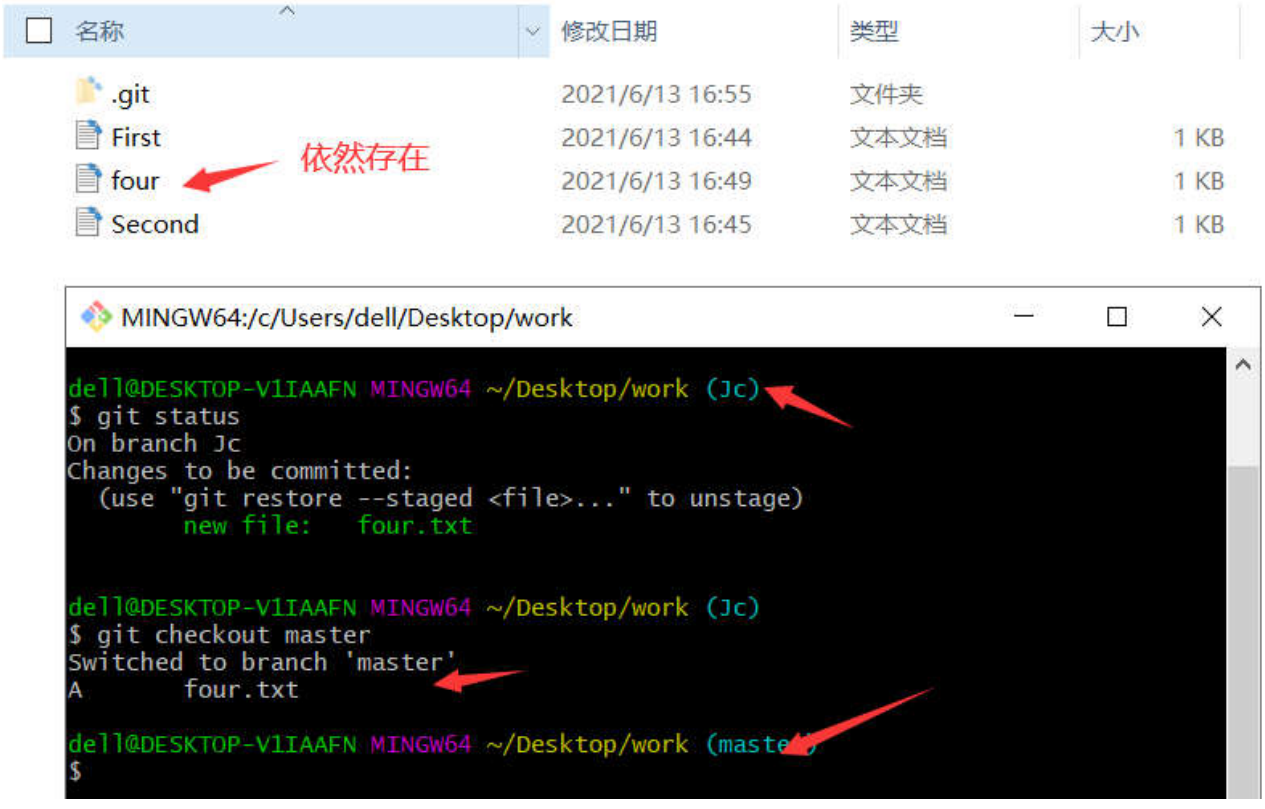
- 切换分支什么区域会发生变化?
  1.HEAD文件内容会发生变化，因为HEAD文件里面是当前正在使用的分支。
  2.暂存区内容发生变化，因为切换分支意味这你要在项目的某一个版本开发一个新的功能所以暂存区也会发生变化。
  3.工作区内容发生变化，切换分支就是进行版本穿梭那工作区肯定会发生变化的。
- 如果工作区刚创建完一个新文件,或者说创建的文件已经加入暂存但是没有提交，然后切换分支会咋样？(直接上图演示吧~)

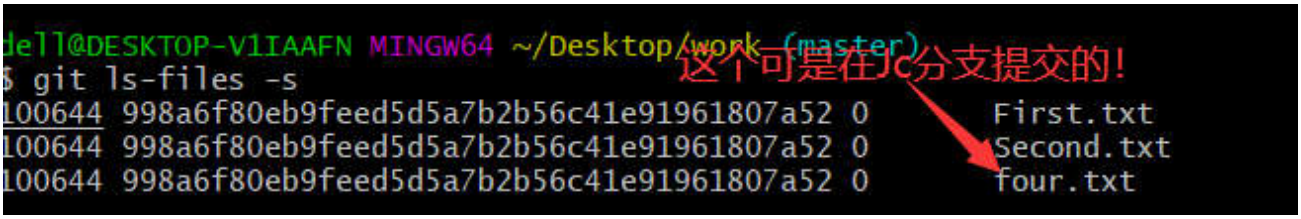




让我们开看看第二种情形。

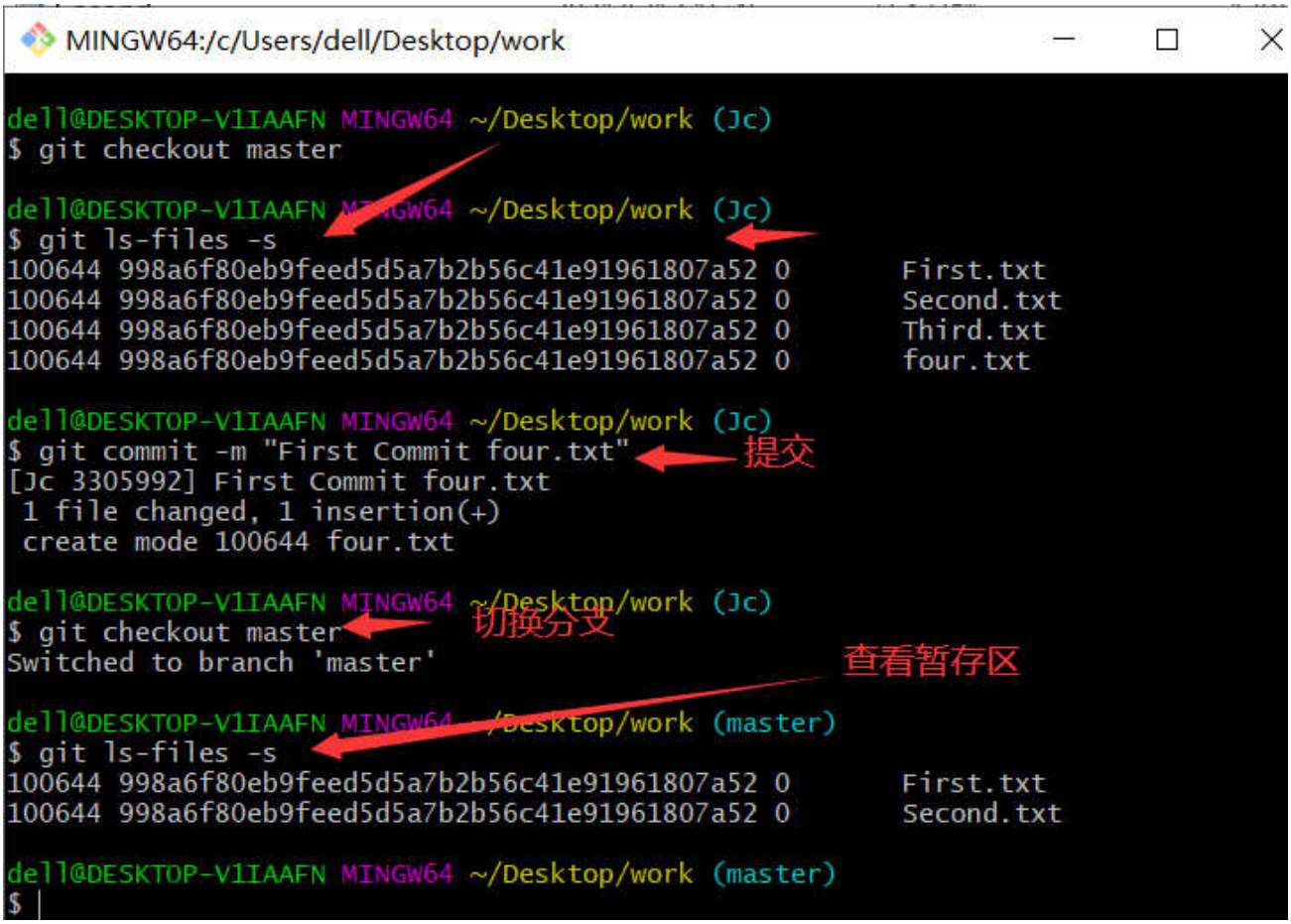

名称
修改日期
类型
大小
.git
2021/6/13 16:54
文件夹
First
2021/6/13 16:44
文本文档
1 KB
four
2021/6/13 16:49
文本文档
1 KB
Second
2021/6/13 16:45
文本文档
1 KB
Third
2021/6/13 16:54
文本文档
1 KB
MINGW64:/c/Users/dell/Desktop/work
dell@DESKTOP-V1IAAFN MINGW64 ~/Desktop/work (Jc)
$ git status
On branch Jc
Changes to be committed:
(use "git restore --staged <file>..." to unstage)
new file: four.txt
dell@DESKTOP-V1IAAFN MINGW64 ~/Desktop/work (Jc)
$


名称
修改日期
类型
大小
.git
2021/6/13 16:55
文件夹
First
2021/6/13 16:44
文本文档
1 KB
four
2021/6/13 16:49
文本文档
1 KB
Second
2021/6/13 16:45
文本文档
1 KB
依然存在
MINGW64:/c/Users/dell/Desktop/work
dell@DESKTOP-V1IAAFN MINGW64 ~/Desktop/work (Jc)
$ git status
On branch Jc
Changes to be committed:
(use "git restore --staged <file>..." to unstage)
new file: four.txt
dell@DESKTOP-V1IAAFN MINGW64 ~/Desktop/work (Jc)
$ git checkout master
Switched to branch 'master'
A four.txt
dell@DESKTOP-V1IAAFN MINGW64 ~/Desktop/work (master)
$

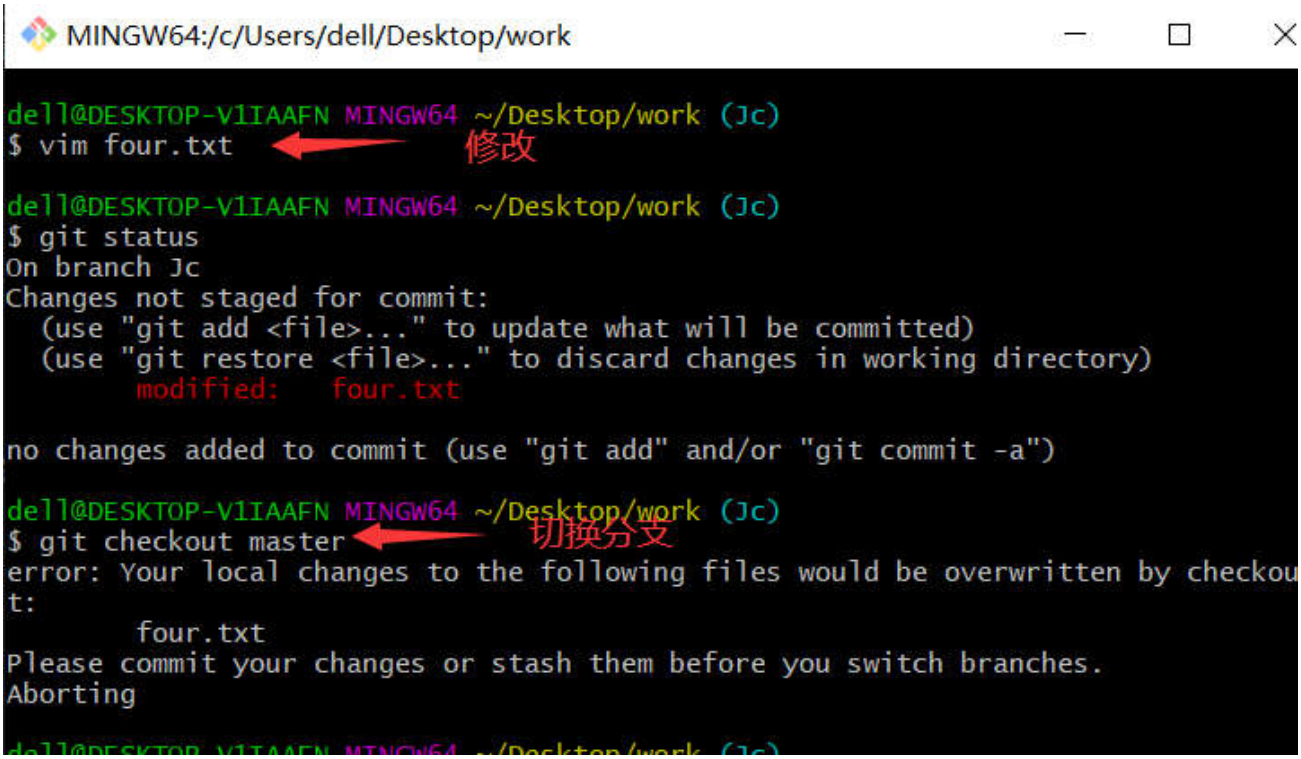


通过上面我们可以看到这两种情形都是可以切换的，那么这样以后再进行开发的时候可能会遇到麻烦，比如我在Jc分支想实现一个功能还没来得及提交我就返回的master分支这样master分支就是多了文件，这样就会导致很多的错误。

如果我在Jc分支上four文件的暂存区我已经给提交了，那么切换到master分支后暂存区还会有four.txt吗?答案很显然是没有的，既然提交了那么就属于Jc分支了。



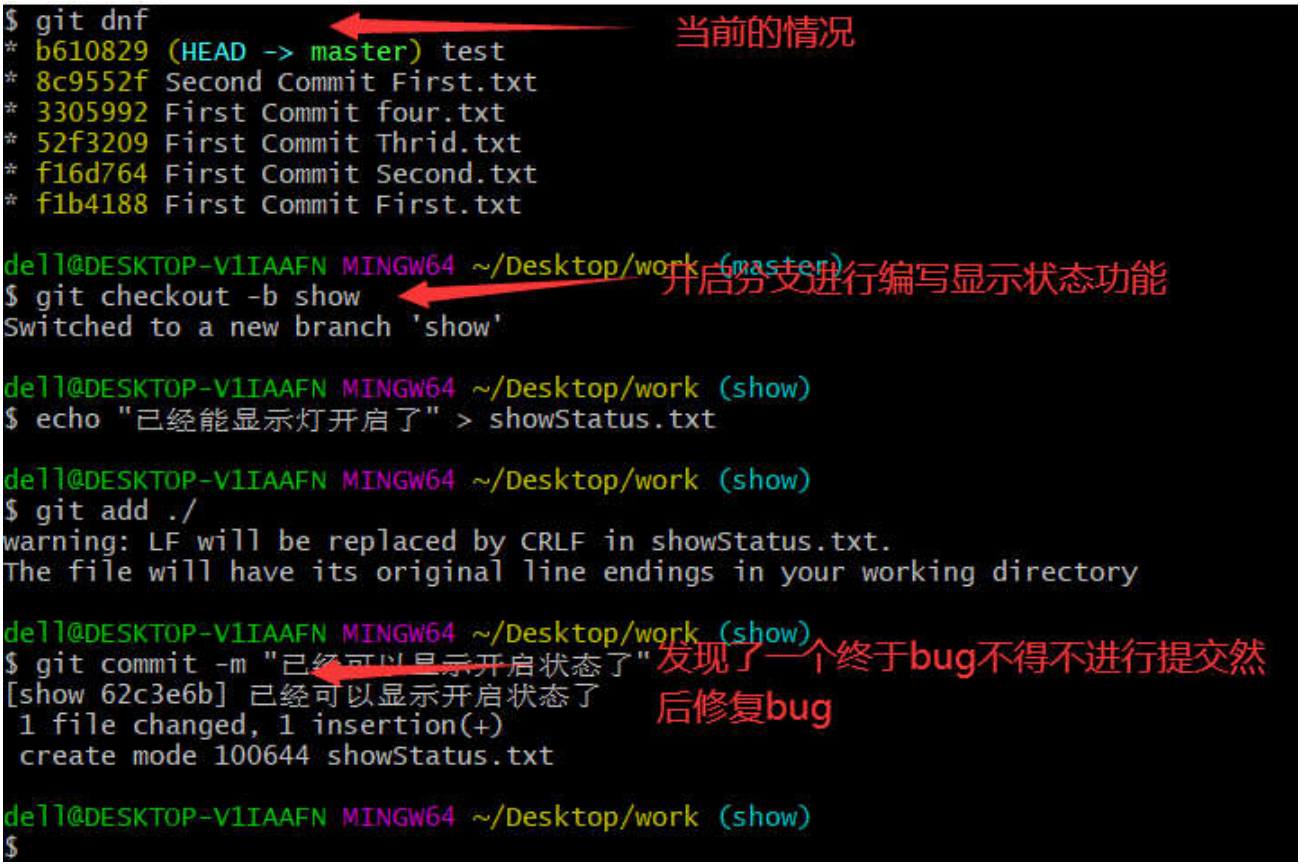
- 如果修改了Jc分支上面已经提交的four文件那还能切换分支吗？答案你猜~那肯定不让切换哈哈



通过上面的解析，我们以后再切换分支的时候一定定定要注意，要不然一失手成千古恨~不多说了直接进入今日的主题了！

一、分支实战

内容：假设你正在花式点灯(本人物联网工程学生只能用点灯打法举例了),突然你想把灯的开关显示再显示屏上面所以开了一个新分支来完成给功能，在修改的代码时候你突然发现以前的(master分支)代码有些问题需要修复，然后在创建一个分支修复完代码之后合并修改分支，将修改推送到线上分支，最后回到最初的分支继续工作。直接上图咯~


$ git dnf
* b610829 (HEAD -> master) test
* 8c9552f Second Commit First.txt
* 3305992 First Commit four.txt
* 52f3209 First Commit Thrid.txt
* f16d764 First Commit Second.txt
* f1b4188 First Commit First.txt
当前的情况
dell@DESKTOP-V1IAAFN MINGW64 ~/Desktop/work (master)
$ git checkout -b show
Switched to a new branch 'show'
开启分支进行编写显示状态功能
dell@DESKTOP-V1IAAFN MINGW64 ~/Desktop/work (show)
$ echo "已经能显示灯开启了" > showStatus.txt
dell@DESKTOP-V1IAAFN MINGW64 ~/Desktop/work (show)
$ git add ./
warning: LF will be replaced by CRLF in showStatus.txt.
The file will have its original line endings in your working directory
dell@DESKTOP-V1IAAFN MINGW64 ~/Desktop/work (show)
$ git commit -m "已经可以显示开启状态了"
[show 62c3e6b] 已经可以显示开启状态了
1 file changed, 1 insertion(+)
create mode 100644 showStatus.txt
发现了一个终于bug不得不进行提交然后修复bug
dell@DESKTOP-V1IAAFN MINGW64 ~/Desktop/work (show)
$

```
$ git checkout master
Switched to branch 'master'

dell@DESKTOP-V1IAAFN MINGW64 ~/Desktop/work (master)
$ git checkout -b improve
Switched to a new branch 'improve'

dell@DESKTOP-V1IAAFN MINGW64 ~/Desktop/work (improve)
$ ls
First.txt  Second.txt  Third.txt  four.txt

dell@DESKTOP-V1IAAFN MINGW64 ~/Desktop/work (improve)
$ vim First.txt

dell@DESKTOP-V1IAAFN MINGW64 ~/Desktop/work (improve)
$ git status
On branch improve
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   First.txt

no changes added to commit (use "git add" and/or "git commit -a")

dell@DESKTOP-V1IAAFN MINGW64 ~/Desktop/work (improve)
$ git commit -a -m "bug 修复完毕"
[improve 726e203] bug 修复完毕
 1 file changed, 1 insertion(+)

dell@DESKTOP-V1IAAFN MINGW64 ~/Desktop/work (improve)
$ git status
On branch improve
nothing to commit, working tree clean

dell@DESKTOP-V1IAAFN MINGW64 ~/Desktop/work (improve)
$ git dnf
* 726e203 (HEAD -> improve) bug 修复完毕
| * 62c3e6b (show) 已经可以显示开启状态了
|/
* b610829 (master) test
* 8c9552f Second Commit First.txt
* 3305992 First Commit four.txt
* 52f3209 First Commit Thrid.txt
* f16d764 First Commit Second.txt
* f1b4188 First Commit First.txt
```

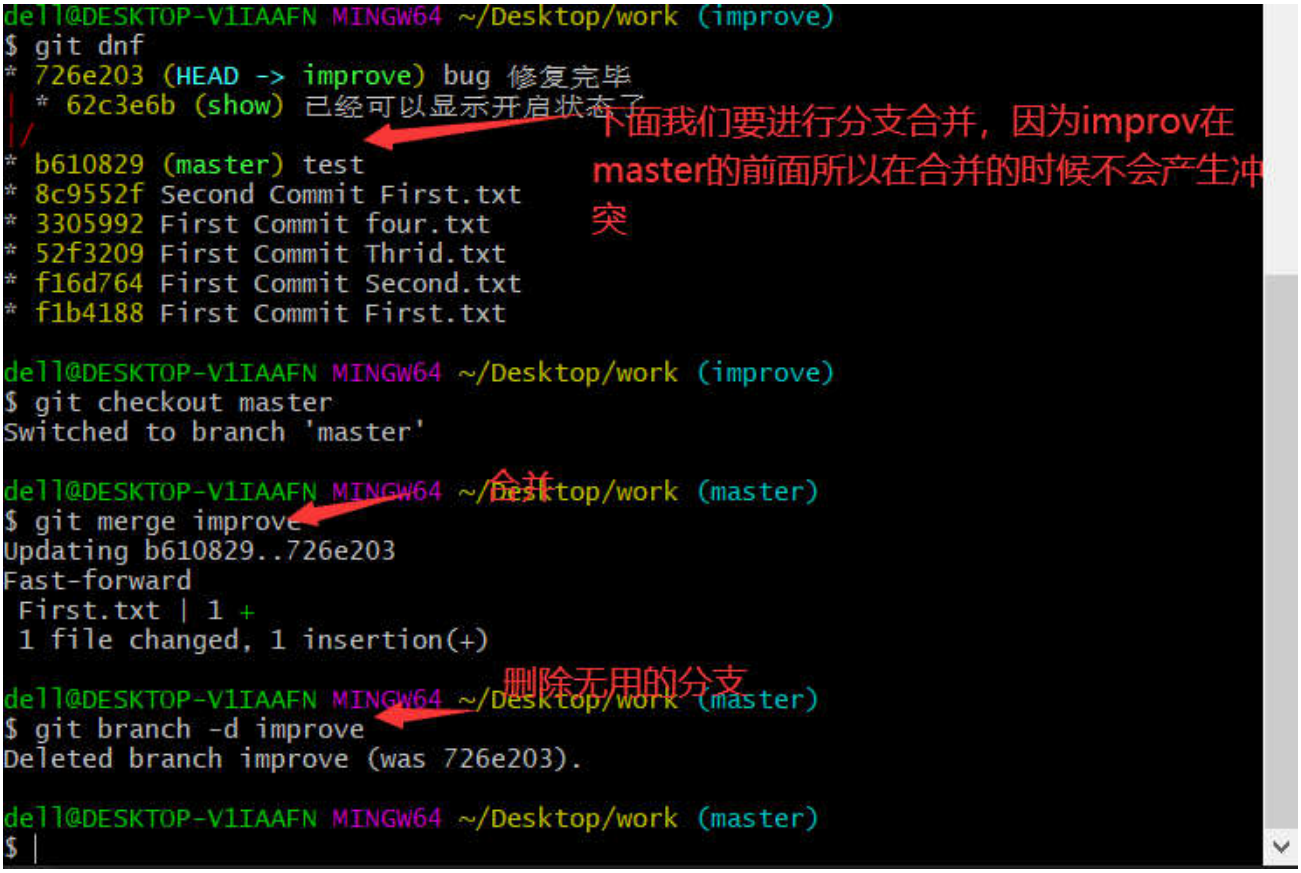
切换主分支

创建新的分支来修复bug

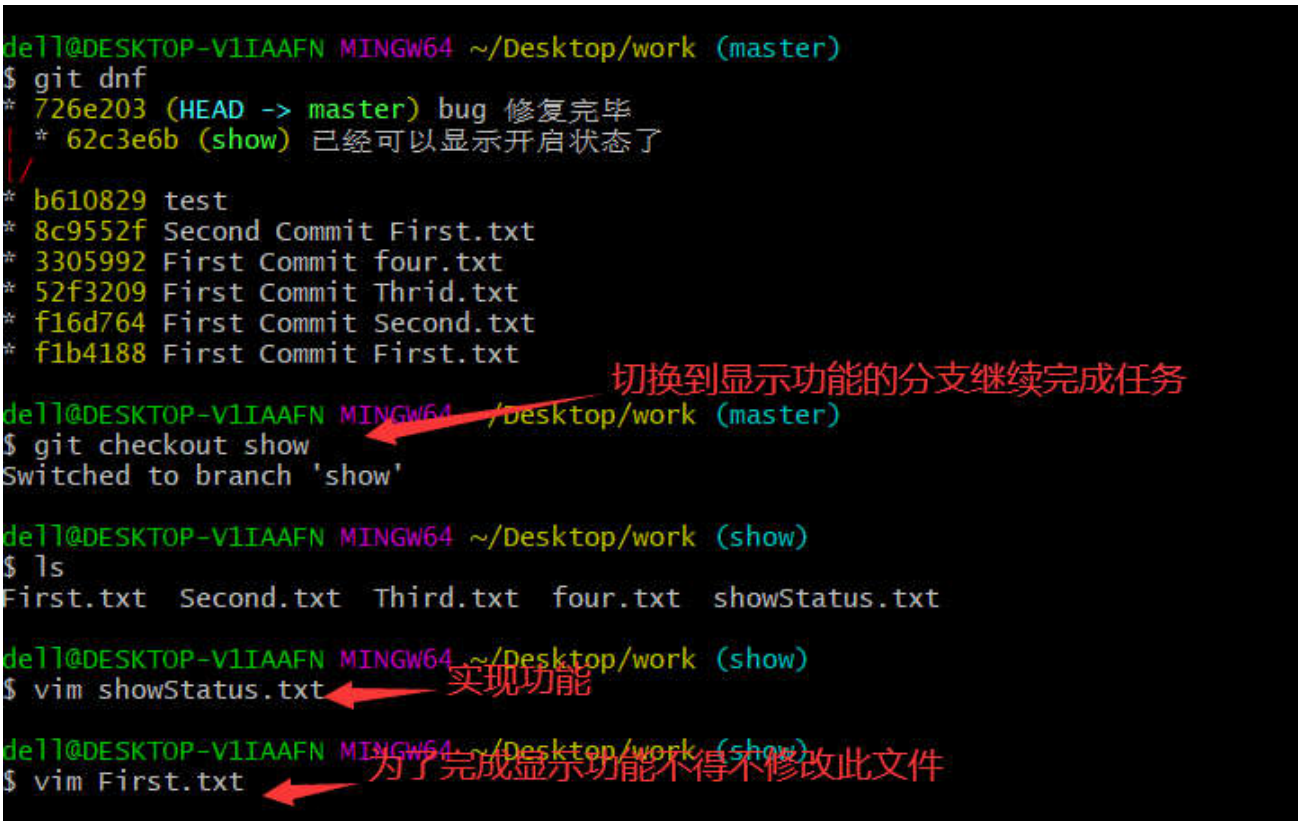
修复bug

修复完毕

目前分支图



下面要返回到我们进行显示灯的状态的那个分支上面去了！(加油！)



```
dell@DESKTOP-V1IAAFN MINGW64 ~/Desktop/work (show)
$ git commit -a -m "已经完成显示功能"
[show b7c2dce] 已经完成显示功能
 2 files changed, 2 insertions(+)

dell@DESKTOP-V1IAAFN MINGW64 ~/Desktop/work (show)
$ git dnf
* b7c2dce (HEAD -> show) 已经完成显示功能
* 62c3e6b 已经可以显示开启状态了
| * 726e203 (master) bug 修复完毕
|/
* b610829 test
* 8c9552f Second Commit First.txt
* 3305992 First Commit four.txt
* 52f3209 First Commit Thrid.txt
* f16d764 First Commit Second.txt
* f1b4188 First Commit First.txt

dell@DESKTOP-V1IAAFN MINGW64 ~/Desktop/work (show)
$ git checkout master
Switched to branch 'master'

dell@DESKTOP-V1IAAFN MINGW64 ~/Desktop/work (master)
$ git merge show
Auto-merging First.txt
CONFLICT (content): Merge conflict in First.txt
Automatic merge failed; fix conflicts and then commit the result.

dell@DESKTOP-V1IAAFN MINGW64 ~/Desktop/work (master|MERGING)
$
```

现在主分支在show分支后面，因为在修改bug和实现显示功能都修改First文件所以会有冲突

提示有冲突

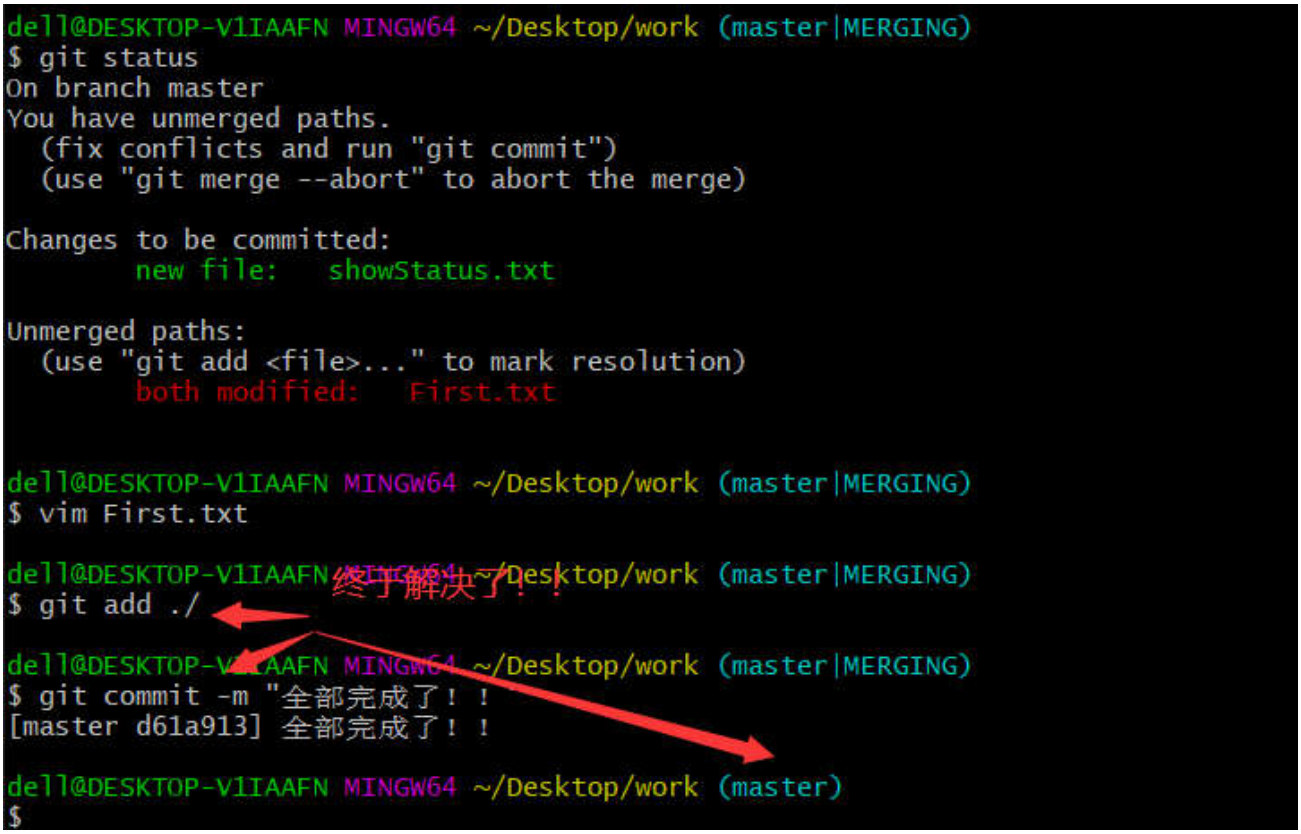
合并没有完成成功

```
dell@DESKTOP-V1IAAFN MINGW64 ~/Desktop/work (master|MERGING)
$ vim First.txt
```

```
V1
V2
<<<<<<< HEAD
V3
=======
为了显示功能所以才修改
>>>>>>> show
~
~
~
```

MINGW64:/c/Users/dell/Desktop/work

```
V1
V2
V3
为了显示功能所以才修改
~
```



到这一个很简单的实战就演练完毕，不知道大家搞没搞懂关系没搞懂得，可以自己画一画图这样就很容易啦~~

总结：通过上面我们可以总结出分支的本质就是指向提交对象的可变指针，说的再仔细一点就是分支就是提交对象HEAD就是可变指针。分支自己不会动是HEAD指针带着分支向前移动。

二、 结尾

至此呢Git分支功能简易实战演练完毕，如果感觉有用可以点个赞的哦！我会持续更新，如果有错误还请指出来,感谢观众老爷的赏脸。

若想获得上述内容的PDF版本移步到GitHub下载。

地址：Git 学习笔记专区。

-----缱绻