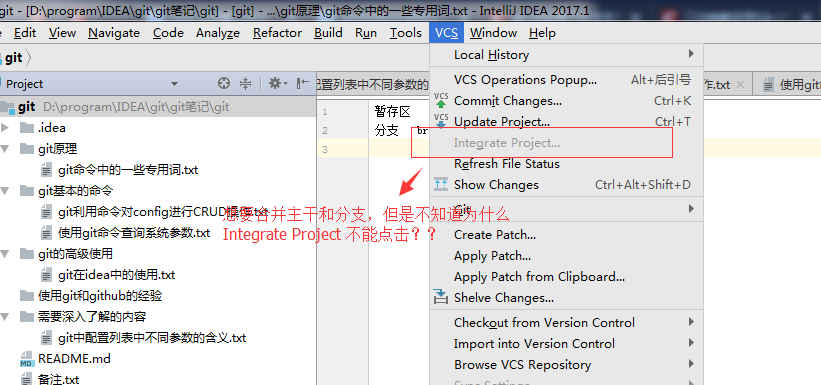
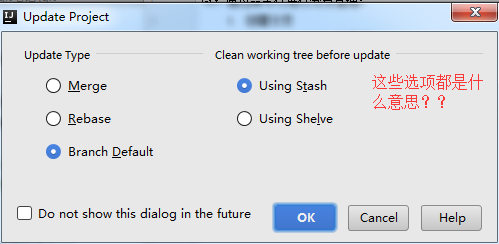
疑惑：



合并主干和分支：



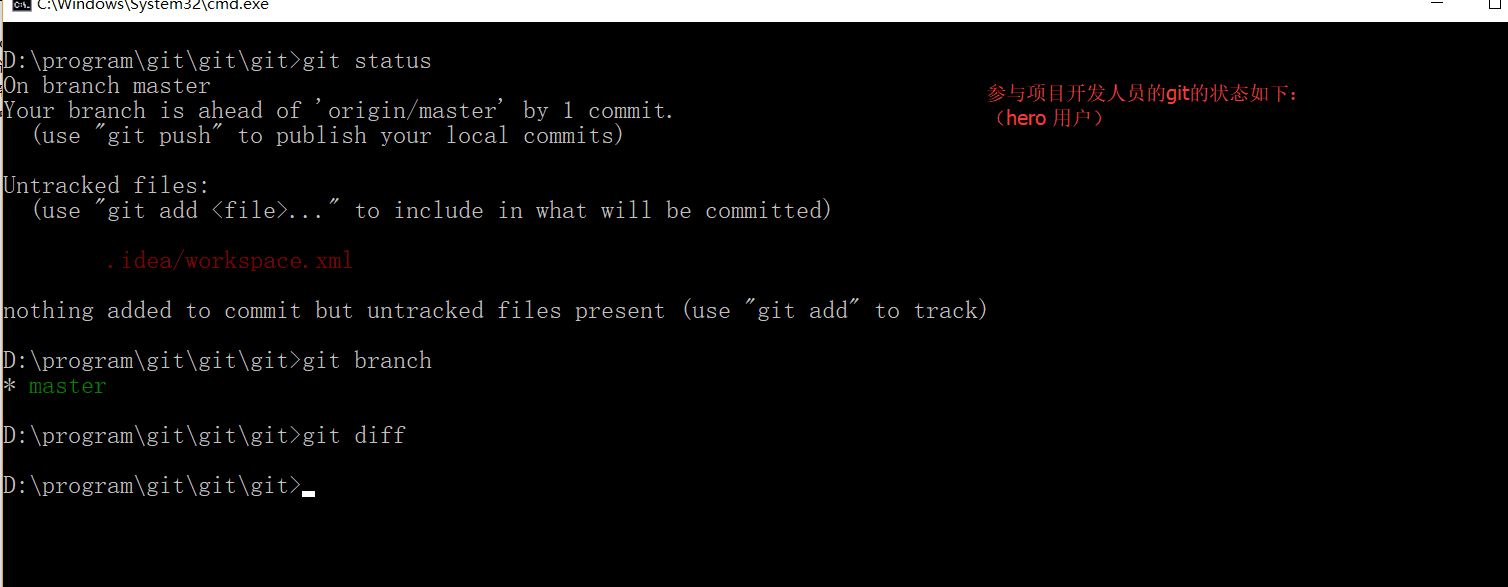


共同开发一个项目的时候，参与者修改了master分支的代码之后，提交到github上的时候，出现了下面的错误。。

The request URL returned error: 403



而参与者的git的状态是：



http 403 该状态表示[服务器](https://baike.baidu.com/item/%E6%9C%8D%E5%8A%A1%E5%99%A8)理解了本次请求但是拒绝执行该任务， 可能是参与者 没有权限修改github上面的文件。尝试重新登录。

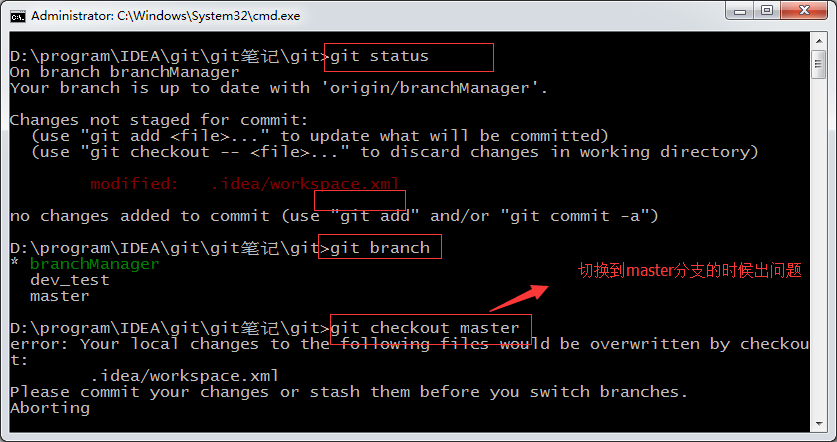
|  |
| --- |
| 修改： $ git config --local –e 修改下面的代码     1. [remote "origin"] 2. url = https://github.com/wangz/example.git   为：   1. [remote "origin"] 2. url = https://wangz@github.com/wangz/example.git   再次git push，弹出框输入密码，即可提交 |

经过上面的测试，发现还是不可以。

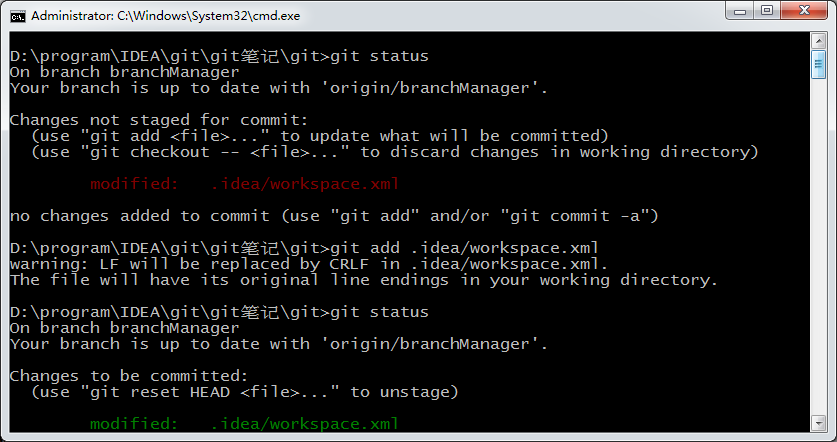
后来终于发现了原因，是因为，自己的代码是直接通过idea来通过 \*.git 来下载下来的，但是，但是对于其他人的邀请，自己并没有，在github上面，接受邀请。 通过在浏览器中输入 \*.git 来接受邀请之后，就可以正确的提交代码了。

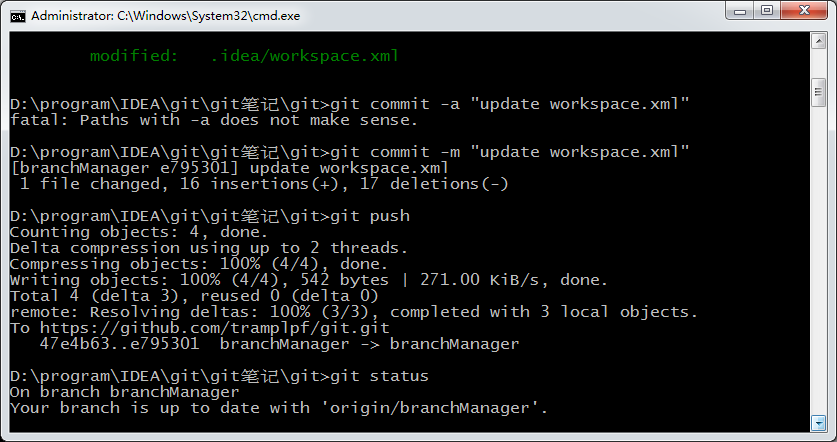
使用git命令合并分支到master操作的时候，出现了问题：

问题出现在了由分支切换到master的时候。



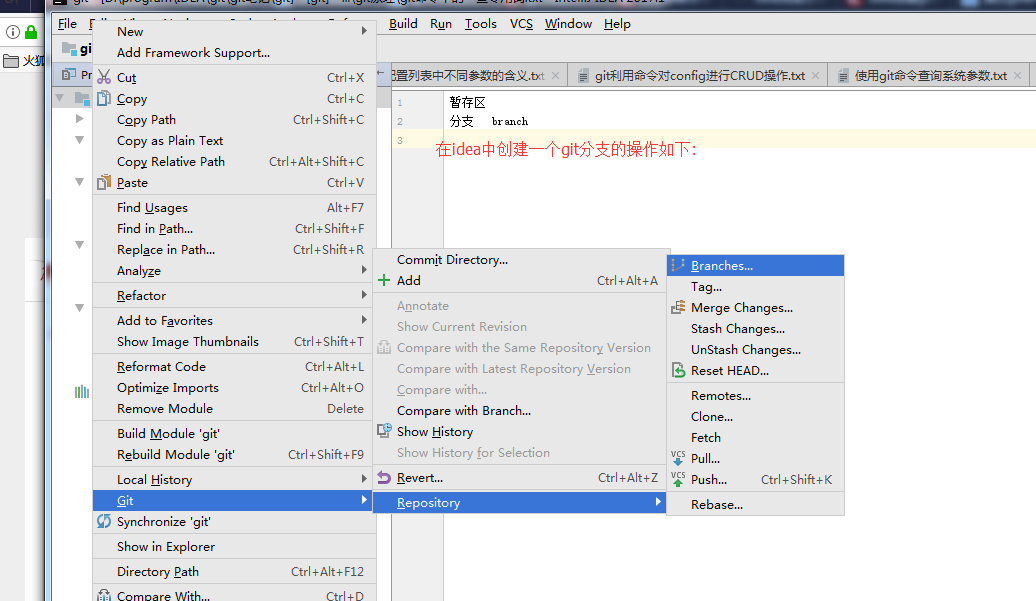
问题：

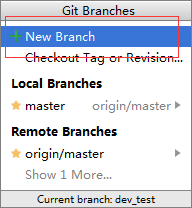


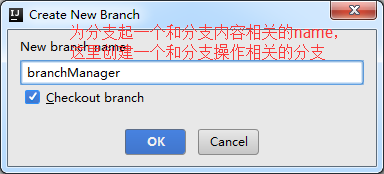


## Idea中分支管理

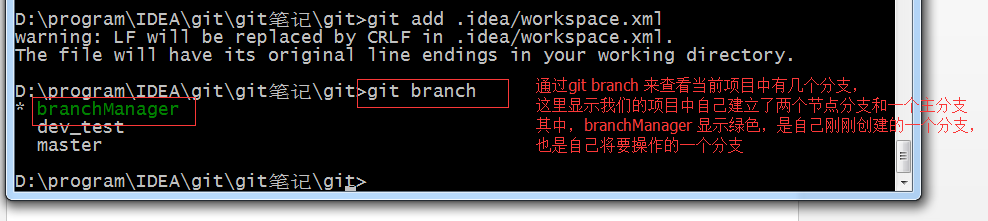
在idea中创建分支



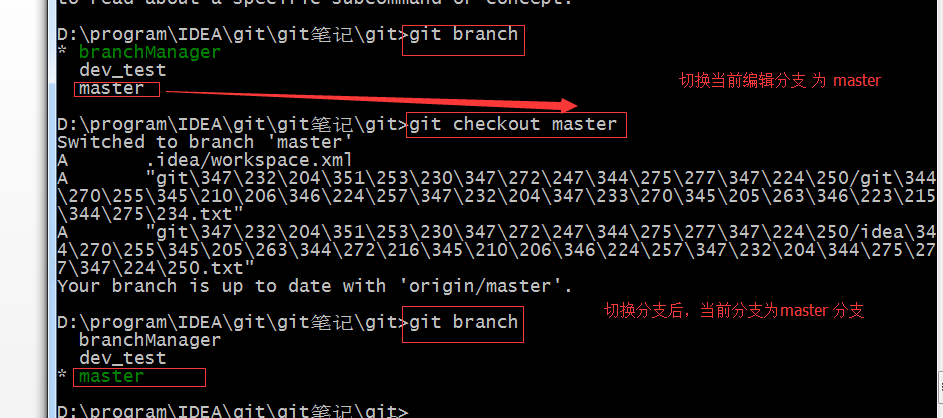


**%**

**此时通过 $ git branch 查看自己的项目中有几个分支**

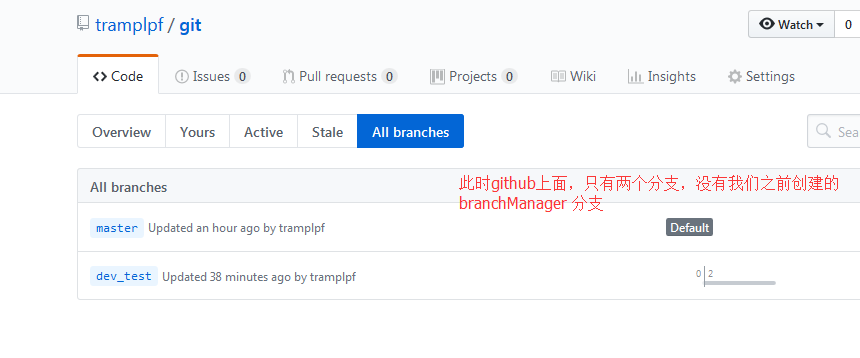


**切换分支： （通过git 命令）**

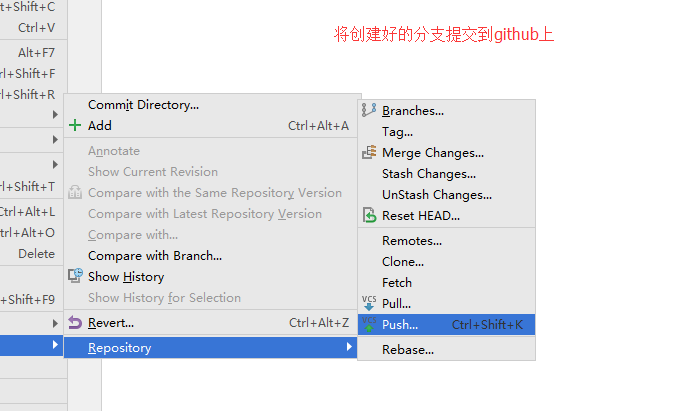


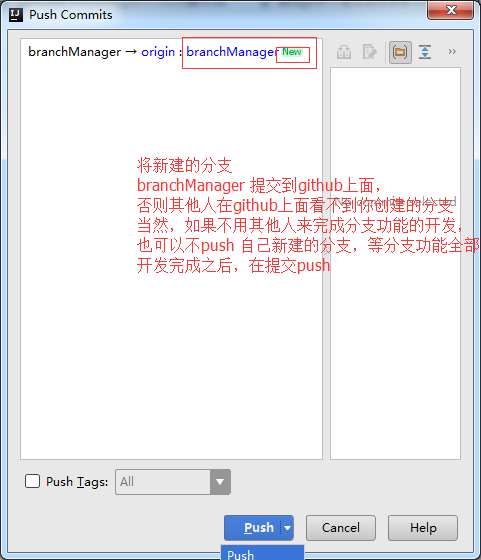
**切换分支: 通过 idea 来切换分支**

**此时在github上面，只有之前创建的两个分支，而没有新建的分支 branchManager。**



**将本地创建的分支提交到github上面(idea中)**





**在被git管理的目录中，删除文件(通过 git 命令)**

**方式一： rm + git commit –am “注释” + git push 命令**

**这里先通过 rm 命令来将目录中的文件删除。 然后将本地所有的修改加入到 stage当中，最后，通过git push 将文件修改的操作更新到github上面。**

**方式二： git rm + git commit – m “注释” + git push 命令**

**Git commit –am “< message>” , 将所有修改，但未进 stage的改动加入 stage， 并记录 commit信息 (某种程度上，相当于 git add 和 git commit -m) 的组合技，前提是被改动文件已经是 tracked。**

## 分支管理之git命令方式

1. 创建分支
   1. 创建分支的git命令很简单： git branch <分支名>
2. 切换分支
   1. $ git checkout <分支名>
   2. 可以用一条语句完成分支的创建和切换
      1. $ git checkout –b <分支名>
3. 分支合并
   1. 首先切换到要合并的目标分支，一般是master主分支 $ git checkout master
   2. 执行合并操作 $ git merge 要合并到master分支的分支名
4. 分支衍合
   1. 分支衍合合分支合并的差别在于， 分支衍合不会保留合并的日志，不留痕迹，而分支合并则会保留合并的日志。
   2. 切换到要合并到的目标分支： $ git checkout master （以master分支为例）
   3. 执行衍合操作： $ git rebase develop

在分支合并和分支衍合的过程中，如果出现了冲突，需要调用 git add， git rm 等命令将冲突解决后提交后，再进行合并或衍合操作。

1. 删除分支
   1. 在将子分支合并到master分支之后，一般会将分支删除， $ git branch –d <要删除的分支名>

注意： 如果分支没有合并到master的时候，执行 git branch –d 分支名 的时候，会报错，这种情况下，根据实际的情况，来决定是强制删除分支还是先合并分支，再删除。如果强制删除的话， 执行 $ git branch –D <分支名>

## 控制文件的上传与否之git 命令操作( 通过 .gitignore 文件来控制)

这里有一点必须注意： 如果想要使某些文件不熟git的版本控制，就要保证在github上面，这些要屏蔽的文件不存在副本。因此，最好，刚开始的时候，就将那些需要屏蔽的文件写到.gitignore 文件当中。

如果该刚开始的时候，没有控制好，那么就需要在之后的操作中，将github上面这些文件删除掉。

（.gitignore只能忽略那些原来没有被track的文件，如果某些文件已经被纳入了版本管理中，则修改.gitignore是无效的。）

一定要养成在项目开始就创建.gitignore文件的习惯，否则一旦push，处理起来会非常麻烦。

在window下 没有后缀的文件创建，比较麻烦，这里给出一个好的方法，创建一个 .gitignore. 文件（前后各一个小数点），就可以直接创建一个.gitignore 文件

在 .gitignore 文件中，可以用于控制那些文件不被加入到版本控制，也可以控制，那些文件夹下的所有文件都不被添加到版本控制当中， 需要注意的是，在 .gitignore 文件中，可以控制哪些文件需要被加入到版本控制当中。如下：

|  |
| --- |
| \*.a  # 忽略所有 .a 结尾的文件  !lib.a    # 但 lib.a 除外 /TODO     # 仅仅忽略项目根目录下的 TODO 文件，不包括 subdir/TODO build/    # 忽略 build/ 目录下的所有文件  doc/\*.txt # 会忽略 doc/notes.txt 但不包括 doc/server/arch.txt |

|  |
| --- |
| 其他的一些过滤条件      \* ？：代表任意的一个字符     \* ＊：代表任意数目的字符     \* {!ab}：必须不是此类型     \* {ab,bb,cx}：代表ab,bb,cx中任一类型即可     \* [abc]：代表a,b,c中任一字符即可      \* [ ^abc]：代表必须不是a,b,c中任一字符      由于git不会加入空目录，所以下面做法会导致tmp不会存在 tmp/\*             //忽略tmp文件夹所有文件      改下方法，在tmp下也加一个.gitignore,内容为 |

**需要注意的地方:**

**当一个git仓库中存在多个分支的时候，对于同一个目录，不同分支里面的内容可能不同， 在git的世界中， git add， git commit， git push 都是有一个前提的，都是针对某一个分支而言的。 如果针对 branchManager 分支进行上面的git add， git commit， git push 操作一个文件夹下的文件，那么操作完成之后，在 master分支下，该文件下的内容是不变的。除非将branchManger分支合并到 master分支上。**