# 1,git

## git历史

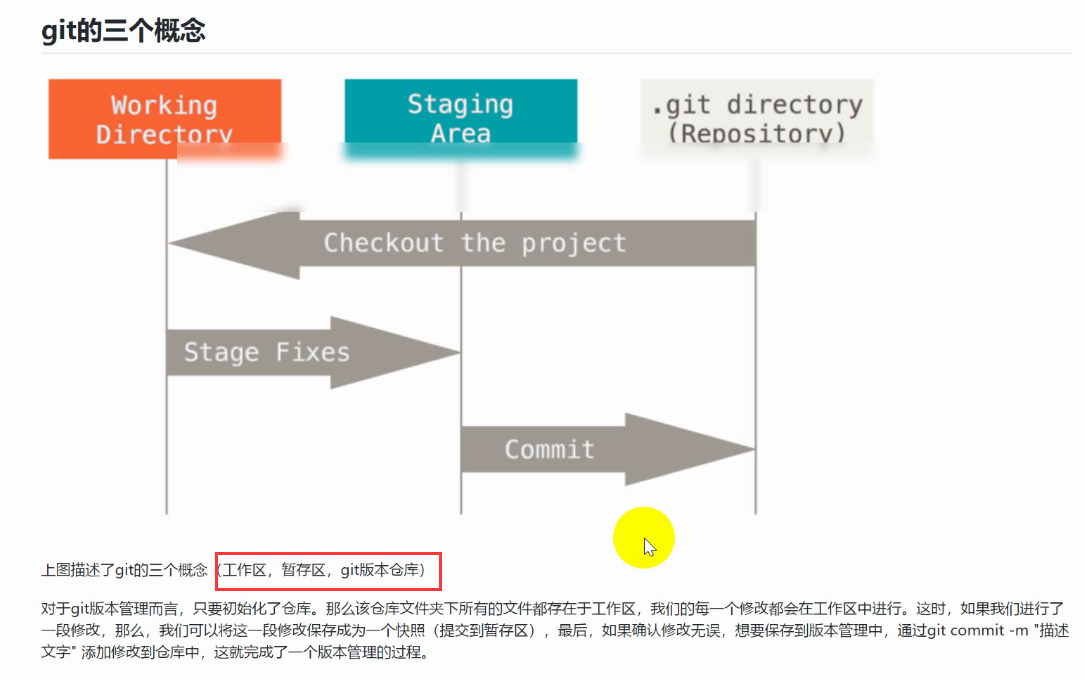
git是用来做版本管理的工具

版本管理的发展(了解):文件对比时代-----集中化版本管理-----分布式版本管理,git就是代表

不要把git和github搞混了,github是基于git的一个代码托管平台,可以理解为远程仓库或中央服务器。国内类似于github的代码托管平台有码云,coding等

## git使用

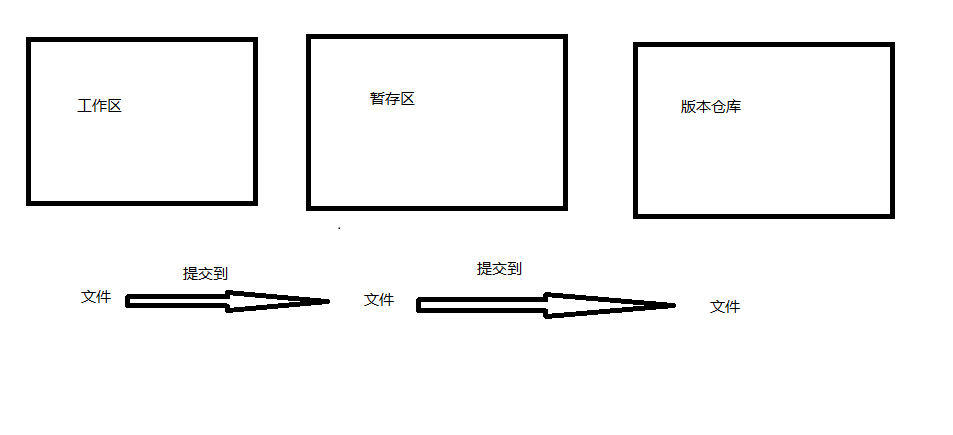
Git有三个区:*工作区*,*暂存区*,git*版本仓库*



文件的提交分为两步: 工作区=>暂存区 暂存区=>版本仓库

文件创建之后,保存在工作区 你需要把它提交到暂存区

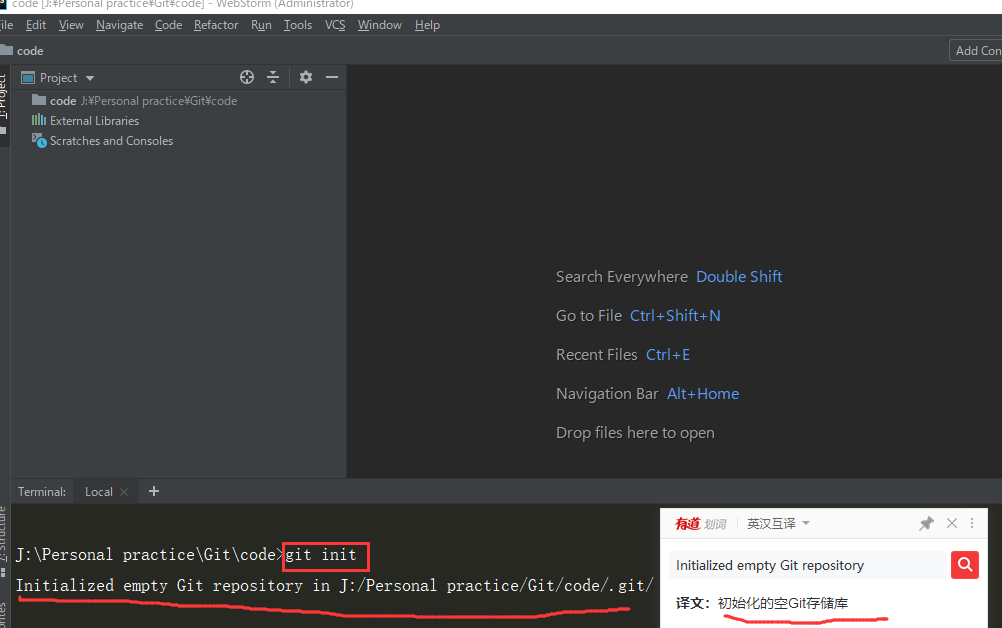
你需要把文件从暂存区提交到版本仓库 就像下面图中的一样



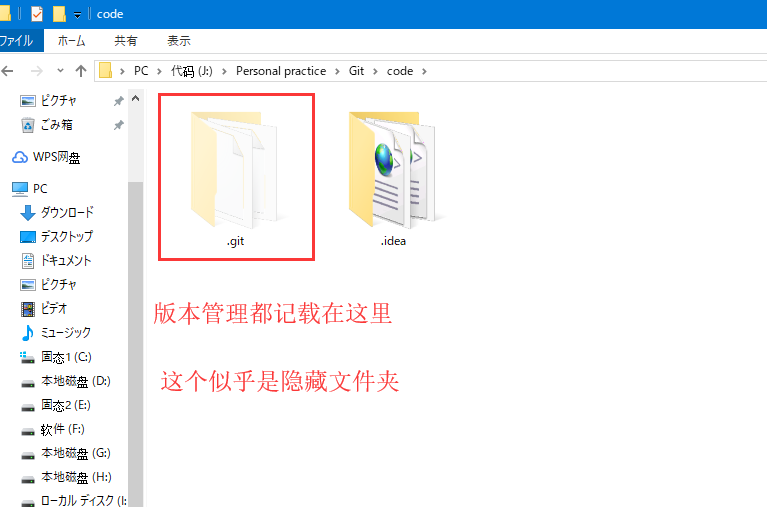
### (1)方法

#### 1,初始化一个仓库

git init初始化一个仓库 一旦初始化了一个仓库,就会对整个文件夹形成一个”工作区”

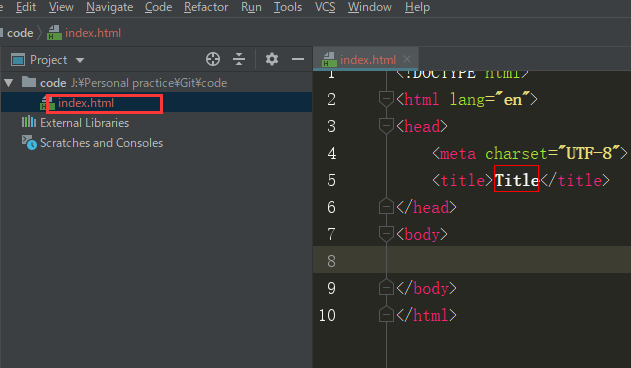


就出现了下面的两个文件夹



#### 2,创建一个文件

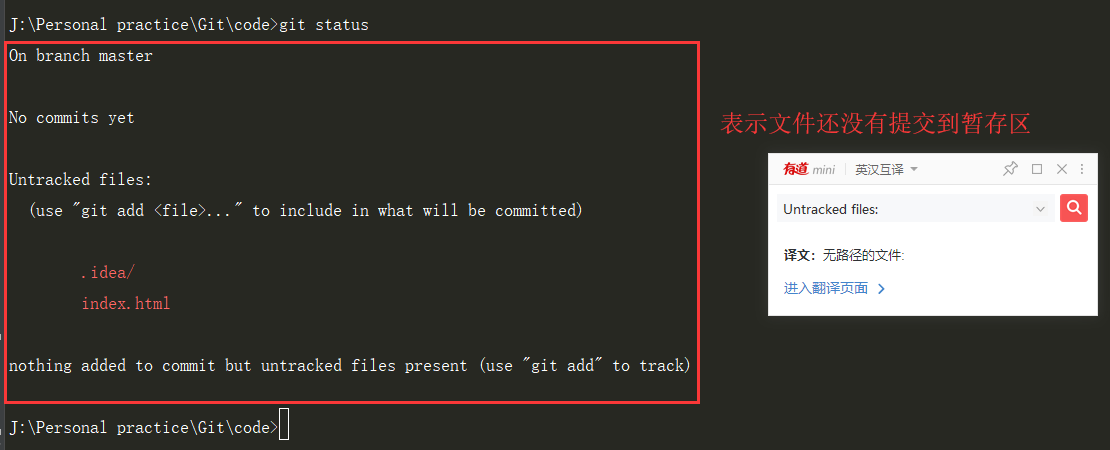
在code文件夹下新建一个html文件,叫index.html



打开终端



输入指令 “git status” 查看状态



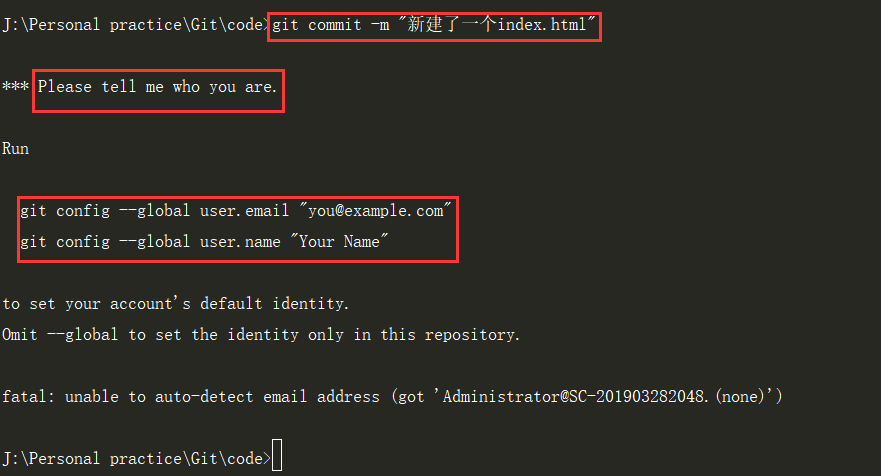
#### 3,提交到暂存区

git add 文件名 提交到暂存区



#### 4,提交到版本仓库

git commit -m “描述” 提交到版本仓库

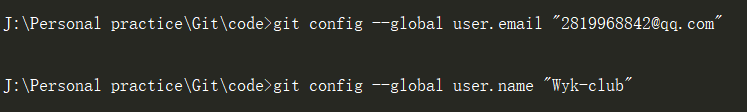


“请告诉我你是谁”

为什么会这样呢?因为git是一个多人的版本管理工具,每个人都要对其上传的代码负责

#### 5,提交邮箱 用户名 再次提交

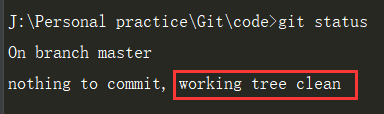
邮箱 用户名最好和你的github的信息一致



这时候就可以提交文件到版本仓库了

输入 git commit -m “描述” 提交

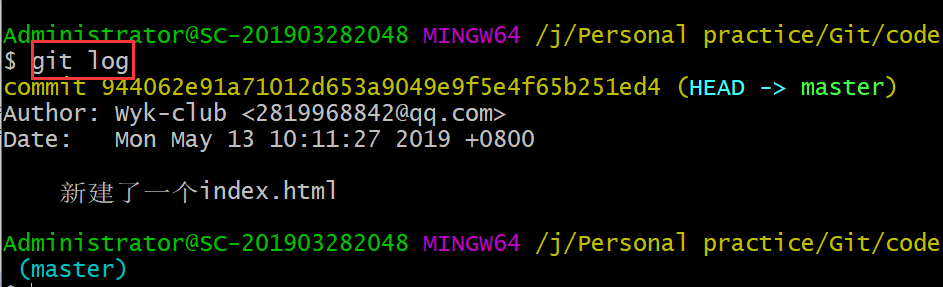
git status 查看状态



提交完成

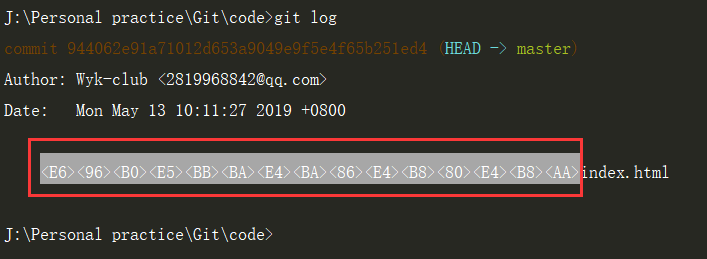
#### 6,查看版本记录

git log 查看版本记录



这里我是用git bash打开的 在目标文件夹右键 选择 “Git Bsah Here”

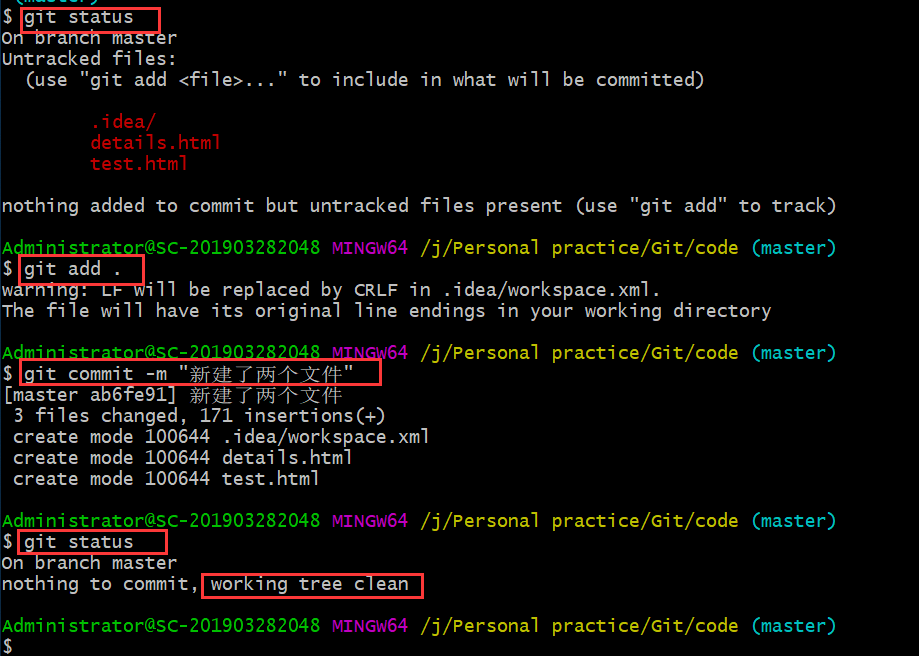
因为webstorm无法显示中文描述



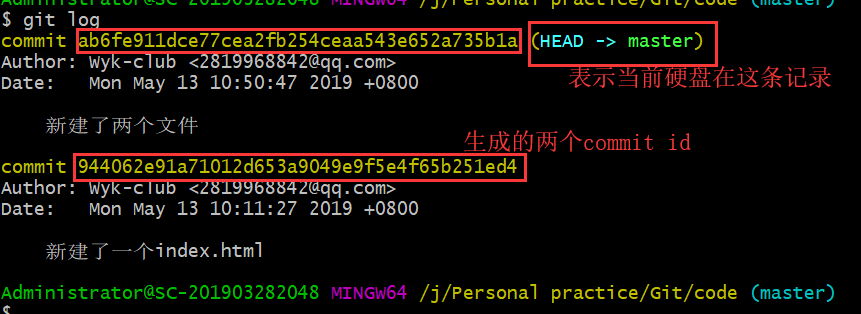
#### 7,批量提交到暂存区

git add . 提交所有文件到暂存区

我再新建两个文件,把他们完成提交



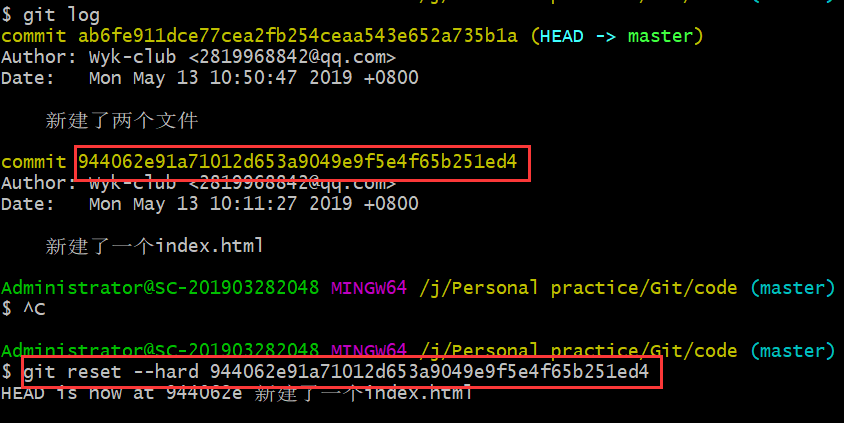
git log查看版本记录 HEAD表示当前硬盘在这条记录



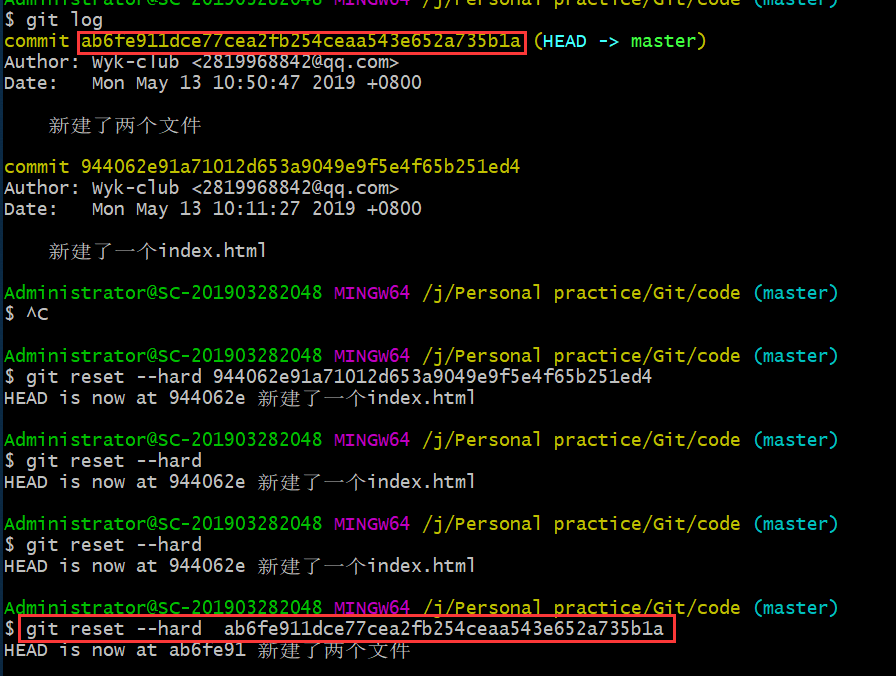
#### 7,恢复版本记录

如果你想恢复到上一条版本

git reset --hard 那条版本commit id



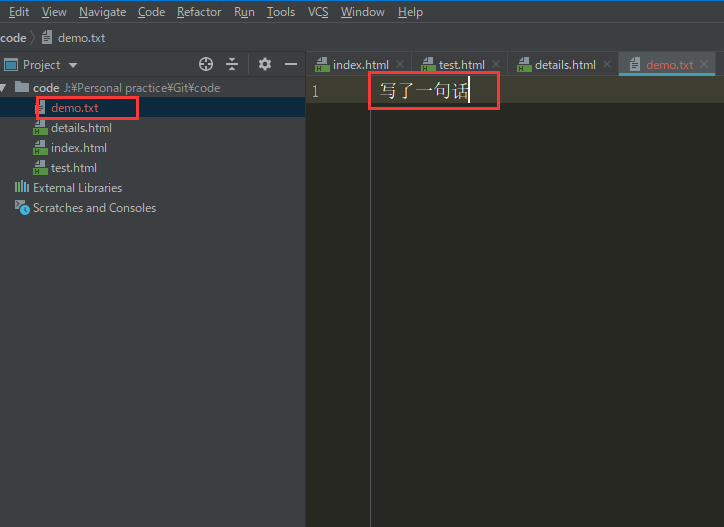
如果你还想恢复过来 输入对应的commit id就行了



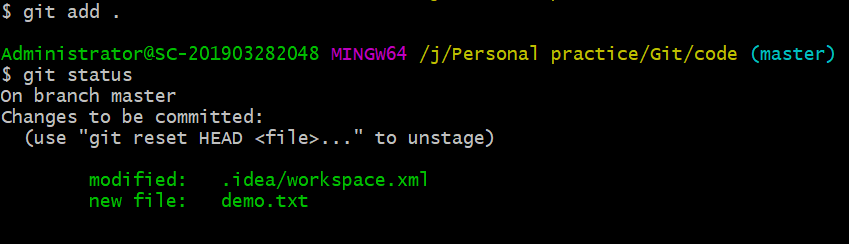
#### 8,回滚操作/放弃提交到暂存区之后的的修改

git checkout

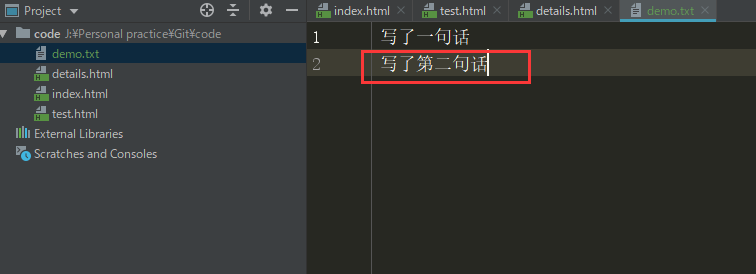
我新建了一个文件 demo.txt 写了一句话



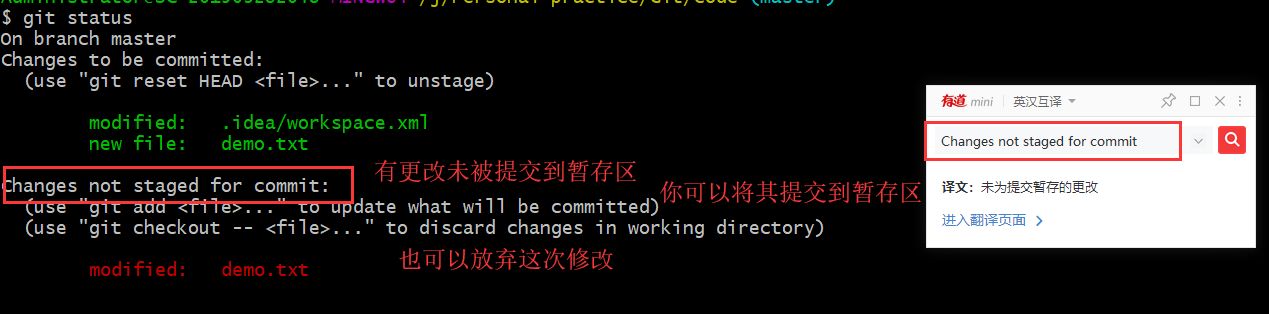
我将其提交到暂存区



然后对demo.txt进行修改



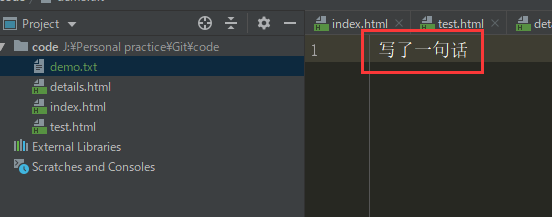
这时候 git status 查看状态



放弃提交到暂存区之后的修改



第二句话就消失了



#### 9,回滚操作总结:

回滚操作必须建立在 文件已经提交到暂存区的操作基础上

git add 有两个功能:

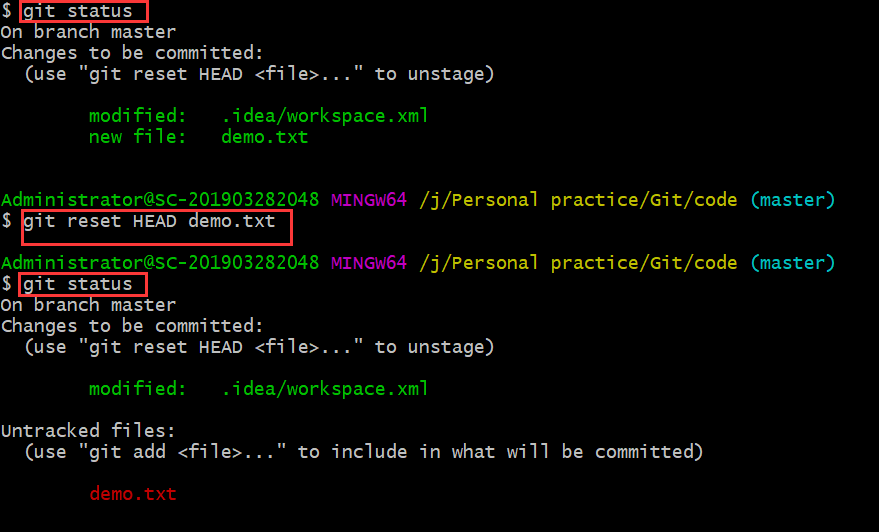
1,将文件添加到暂存区;

2,将文件进行追踪,一旦其中有修改,就会被git版本工具追踪到其的工作区修改

#### 10,将文件从暂存区移除

git reset HEAD 文件名

比如移除demo.txt



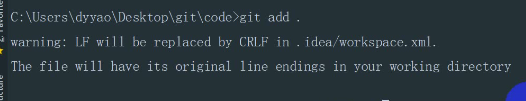
demo.txt就移除了

#### 11,.idea文件夹及其移除

.idea文件夹是编辑器创建的,里面存储的是编辑器的配置文件。

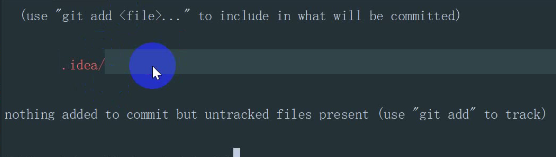
如果你要将文件共享给别人,提交过程中使用的 git add . ,把.idea文件夹也提交上去了,这其实是不太合理的。因为别人用的编辑器不一定和你的一样,.idea文件夹就显得有些多余。

##### 1,移除.idea文件夹





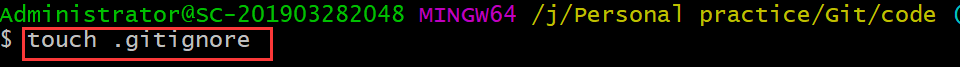
.idea文件夹没有被git add追踪



##### 2,.gitignore 配置git不追踪的文件

实际使用时,都是用这种方法事先配置好,使.idea不被git add追踪

我们永远都不想让.idea被追踪 创建.gitignore文件



或者你直接手动创建.gitignore文件也可以

具体的配置方法见下面的方法总结 .gitignore

### 方法总结

#### 1, git init

初始化一个git仓库

#### 7, git status

查看git状态

#### 2, git add <file>

添加文件到暂存区中

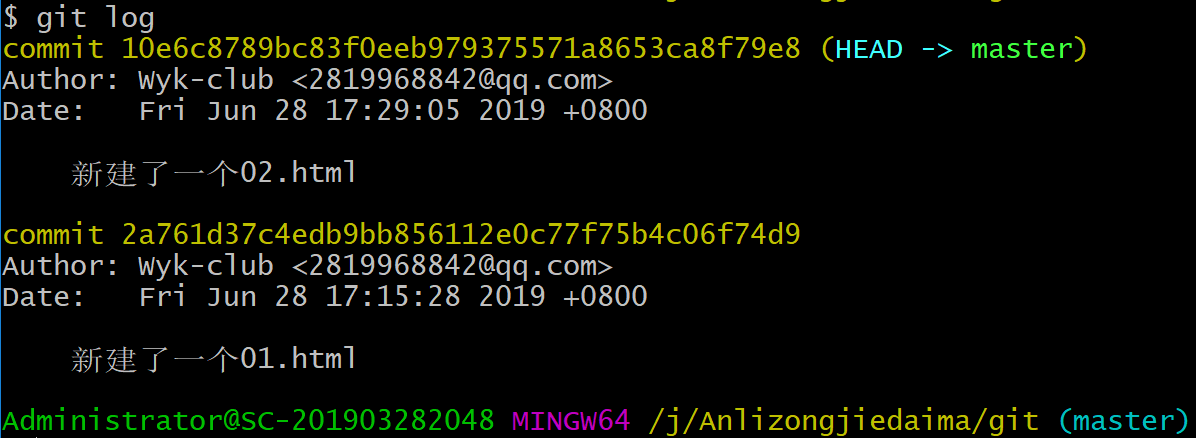
git add 文件名 提交单个文件

#### 3, git commit -m “描述”

提交暂存区的文件到版本管理库

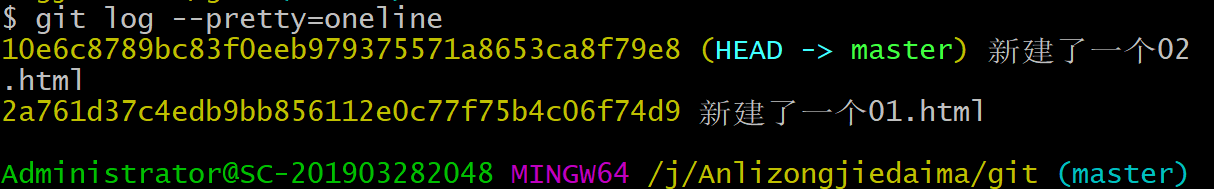
#### 4, git log

查看版本记录



#### 4, git log --pretty=oneline

查看简洁的版本记录(只显示哈希值和提交说明)



#### 4, git reset HEAD <file>

将文件从暂存区移除

#### 5, git checkout -- <file>

操作回滚

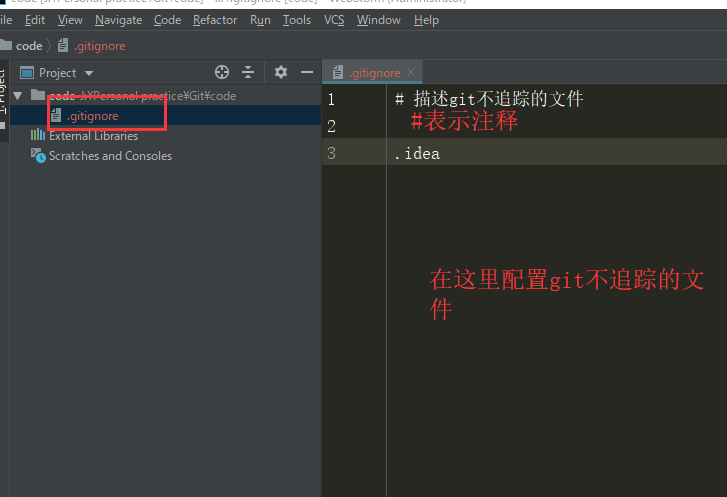
将工作区的文件恢复到上一个版本(放弃工作区的修改)

注意:一旦执行这条指令,将无法复原

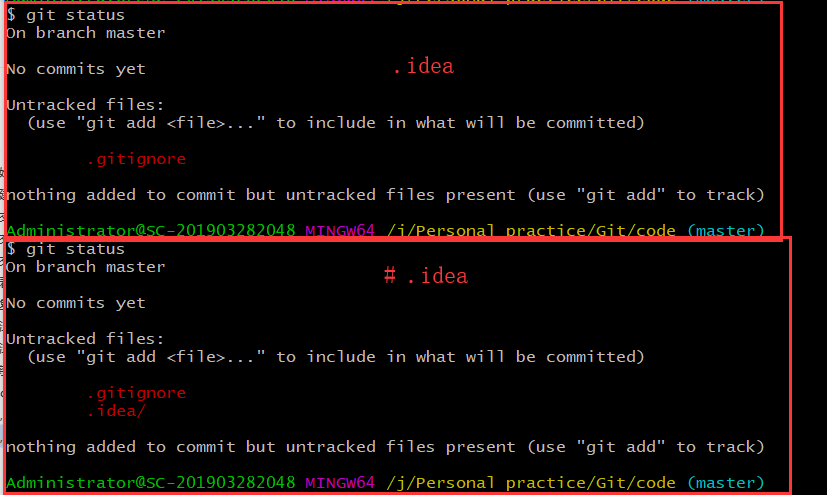
#### 6, .gitignore

声明例外文件

文件将声明一个git不追踪的文件。在这里的声明的文件将不被追踪,这个文件最好在项目新建的时候,就提前写好。(书写规则:#代表注释 直接写忽略的文件夹或文件名即可)



可以看出 确实有用



当然,提交文件的时候也要吧.gitignore提交上去

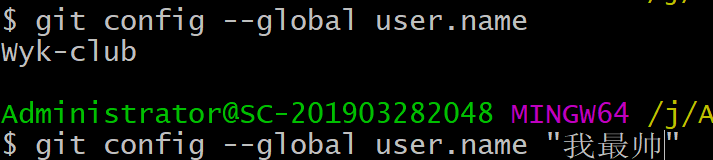
#### 8, git reset --hard commit的id

回到某个版本

#### 9, git config --global user.name “”

此条命令如果不加””,表示读取全局设置的用户名

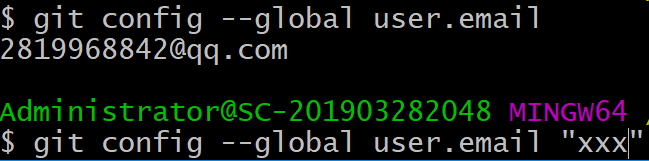
如果加””,表示设置全局设置的用户名



#### 10, git config --global user.email “”

此条命令如果不加””,表示读取全局设置的邮箱

如果加””,表示设置全局设置的邮箱



## git远程仓库

### (1)方法

之前使用的都是是本地的仓库。现实中,多人开发的项目,需要使用远程仓库,即将我们的本地仓库提交到因特网上的仓库。

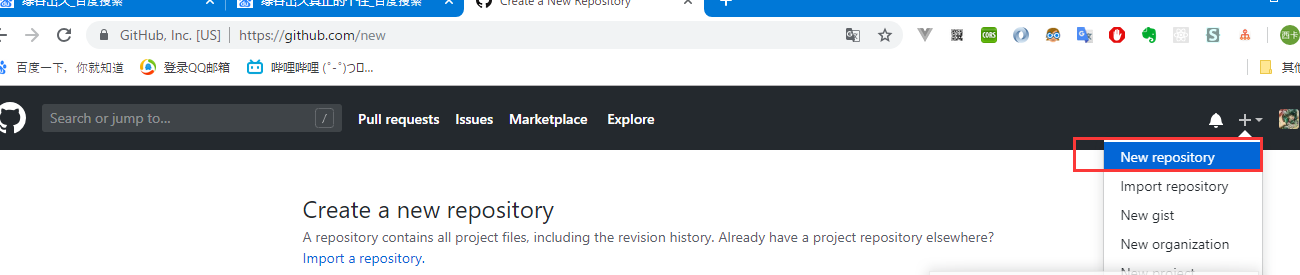
常用的远程仓库主要有开源的github、coding、码云、以及公司内部搭建的gitlab

这里我们使用github

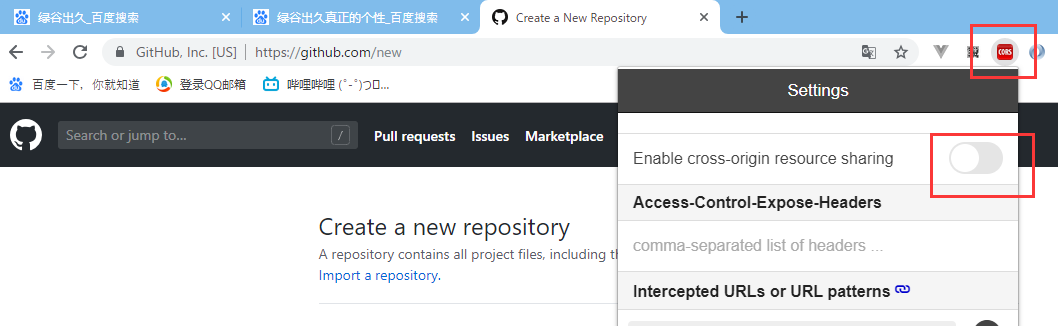
<https://github.com/>

#### 1,新建远程仓库

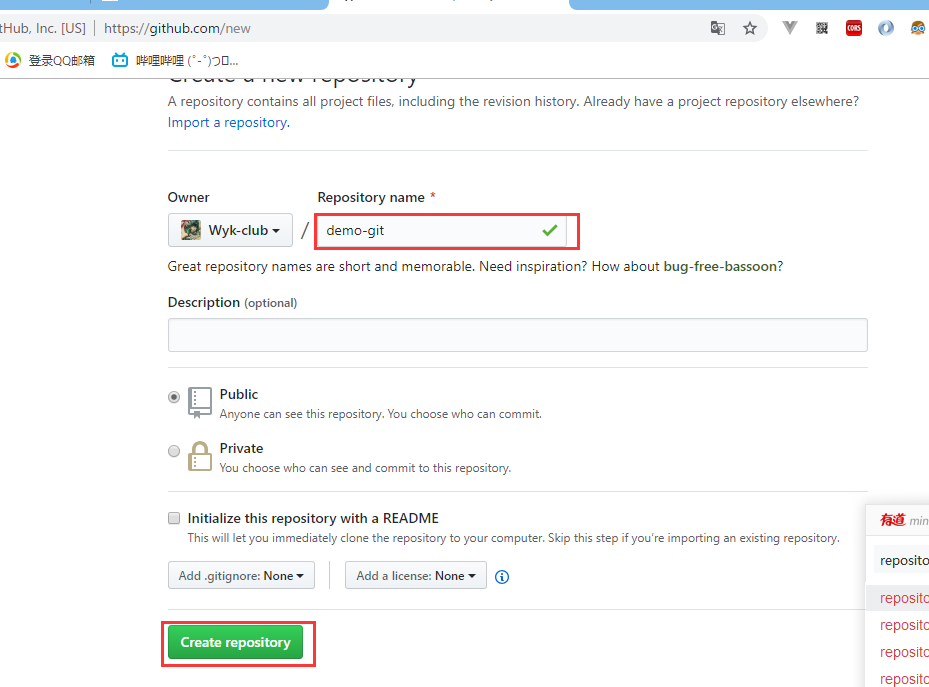
##### 1)新建一个仓库



关闭跨域插件(跨域插件会导致用户名不能通过)



设置你的仓库 点击创建按钮



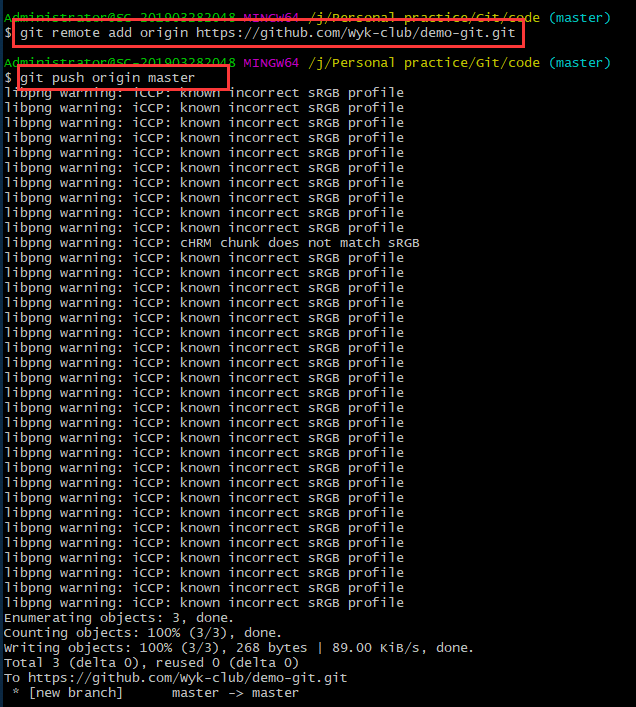
出现以下的内容

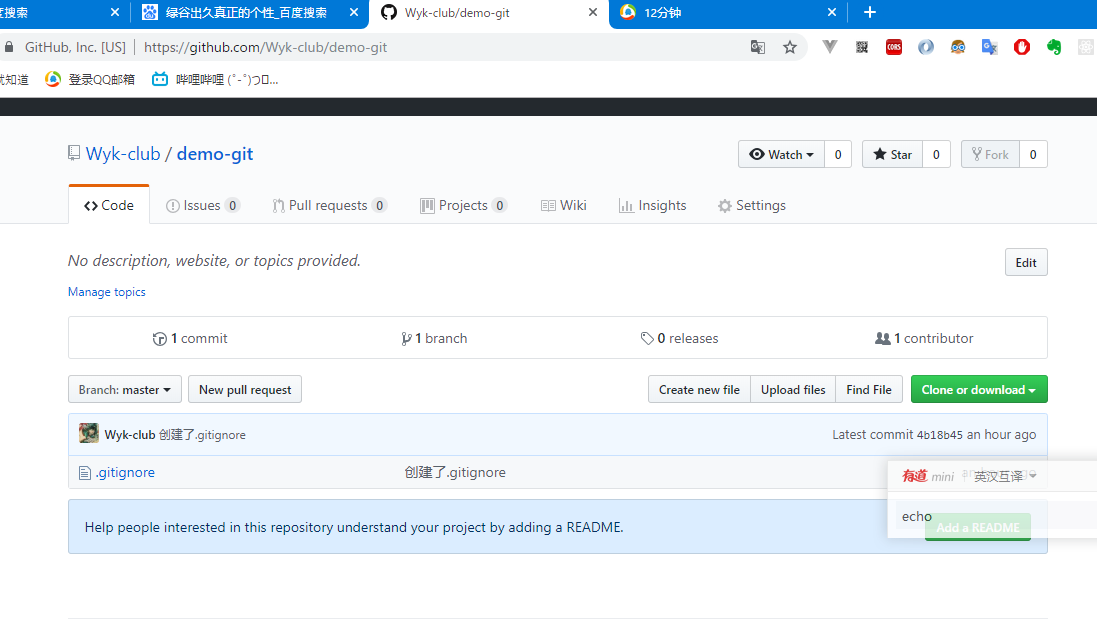
##### 2)指令输入



##### 3)推送成功

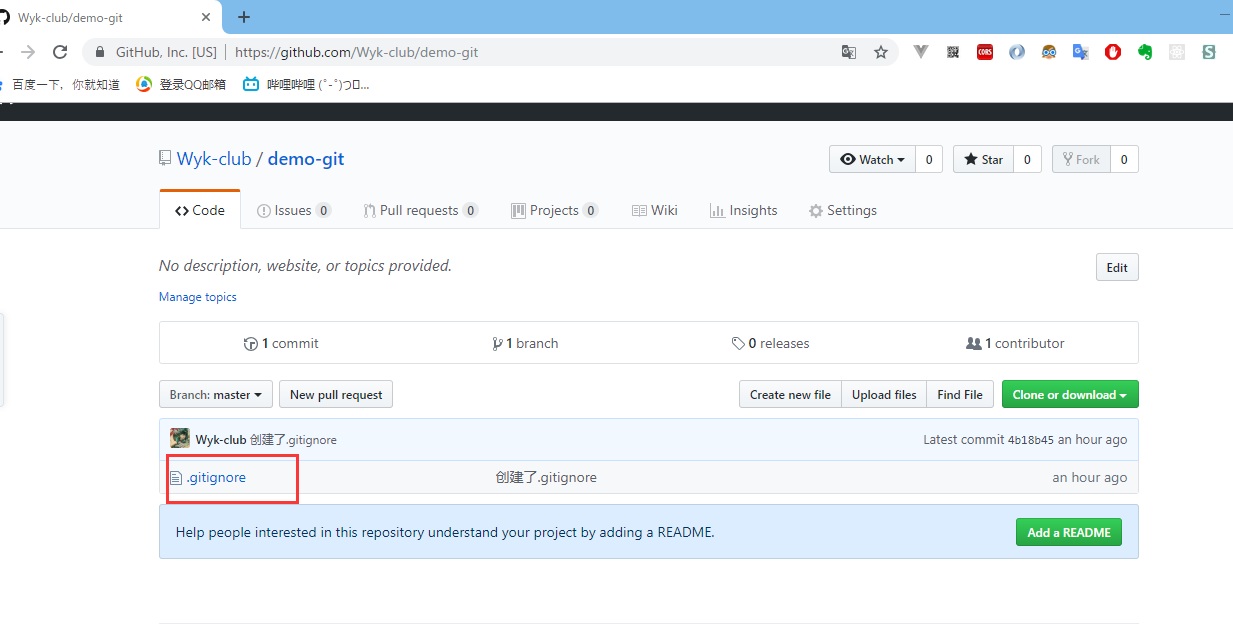
如此操作 代码就推送到远程仓库了



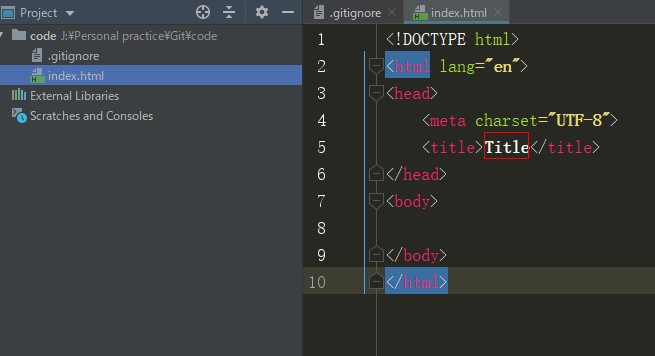


#### 2,更新你远程仓库里的代码

如果你在本地修改了代码,想让远程仓库里的代码同步更新,不需要全部再次上传



我又新建了一个index.html文件

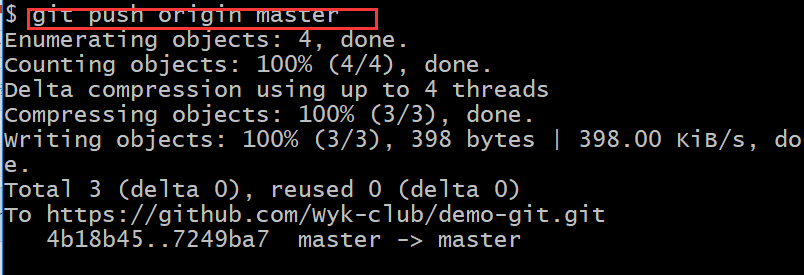


依次操作

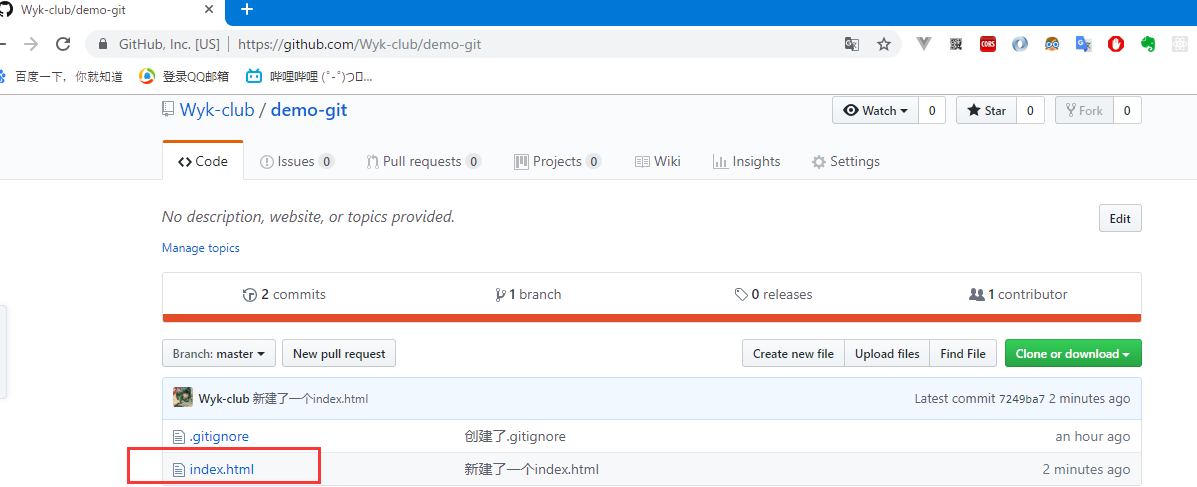
git add .

git commit -m “描述”

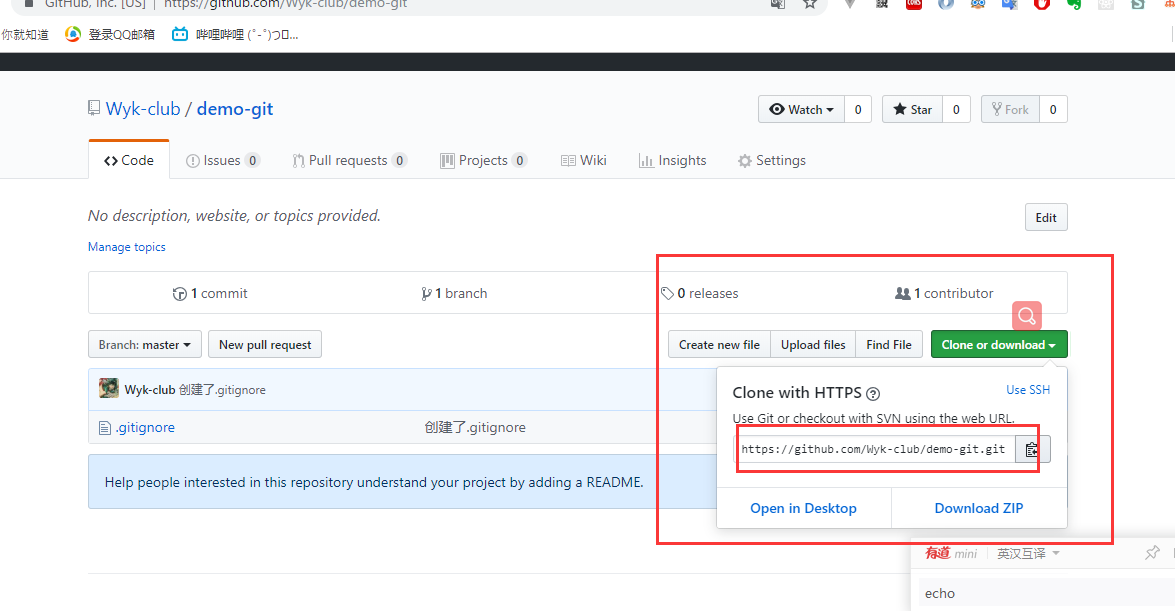
git push origin master



再去云端刷新,已经更新了

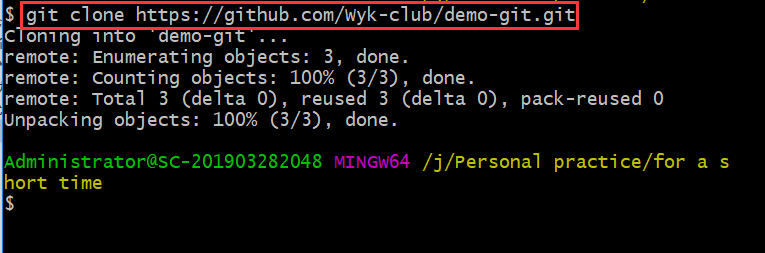


#### 3,将远程库代码克隆到本地

复制如下链接

在目标文件夹右击打开 git bash

输入 git clone 链接

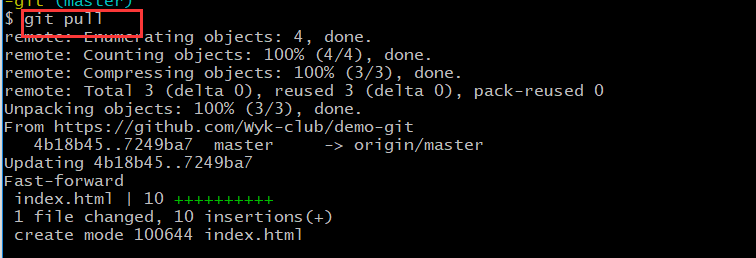


代码就拷贝到本地了

#### 4,克隆者的本地代码更新

在目标文件夹

git pull 表示拉取最新的代码



本地代码就更新了

#### 5,git push-u中-u的含义

github网站推荐的指令是

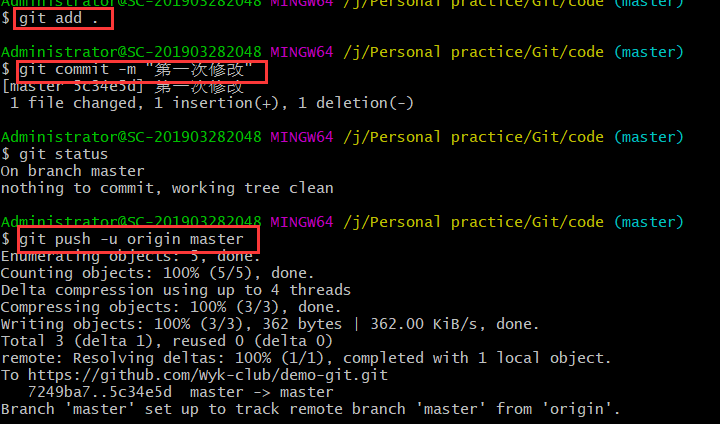
git push -u origin master 而我们上面用的是

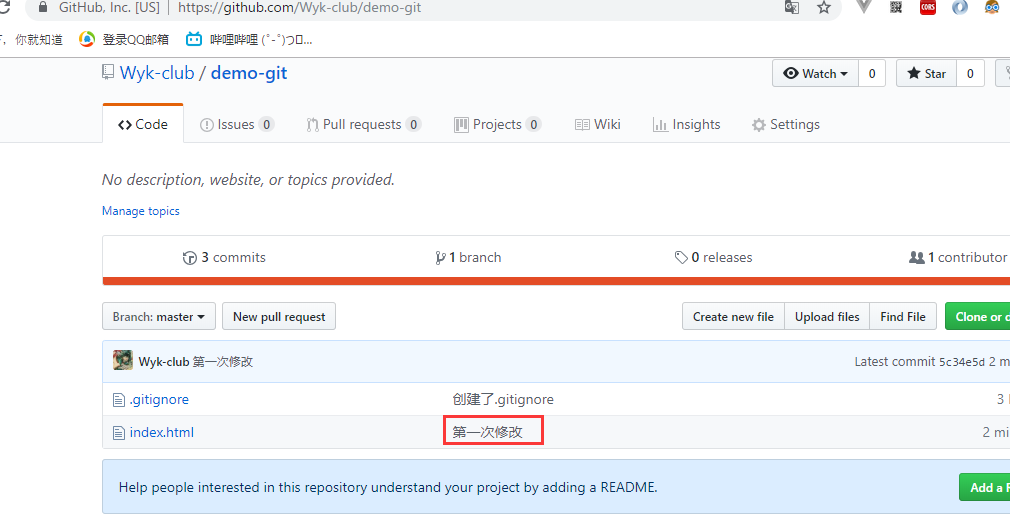
git push origin master

-u表示的是默认路径,比如这次推送你加了-u,那么下次推送你只需要输 git push就可以了,就会推送到你的默认路径。

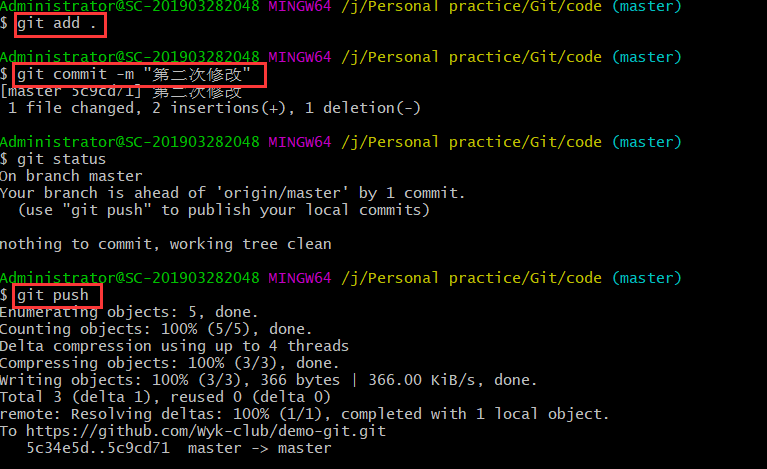
下面演示一下

第一次修改,然后推送 使用 git push -u origin master





第二次修改,然后推送 使用git push





### (2)方法总结

#### 1,推送到远程库

① git init

② git add .

③ git commit -m “描述”

④ git remote add origin master 远程库的地址

⑤ git push -u origin master 推送到哪个远程库

注意: 在多人合作项目中,最好在push之前先pull拉取一下代码,合并到当前的分支,可能别人刚刚提交了代码

#### 2,推送新的更改

① git add . 添加所有的修改到暂存区

② git commit -m “描述你的修改信息” 将你的更改提交到版本管理库

③ git push origin master 提交到远程库

#### 3,克隆远程仓库

git clone 远程库的地址

#### 4,拉取最新的更新

git pull

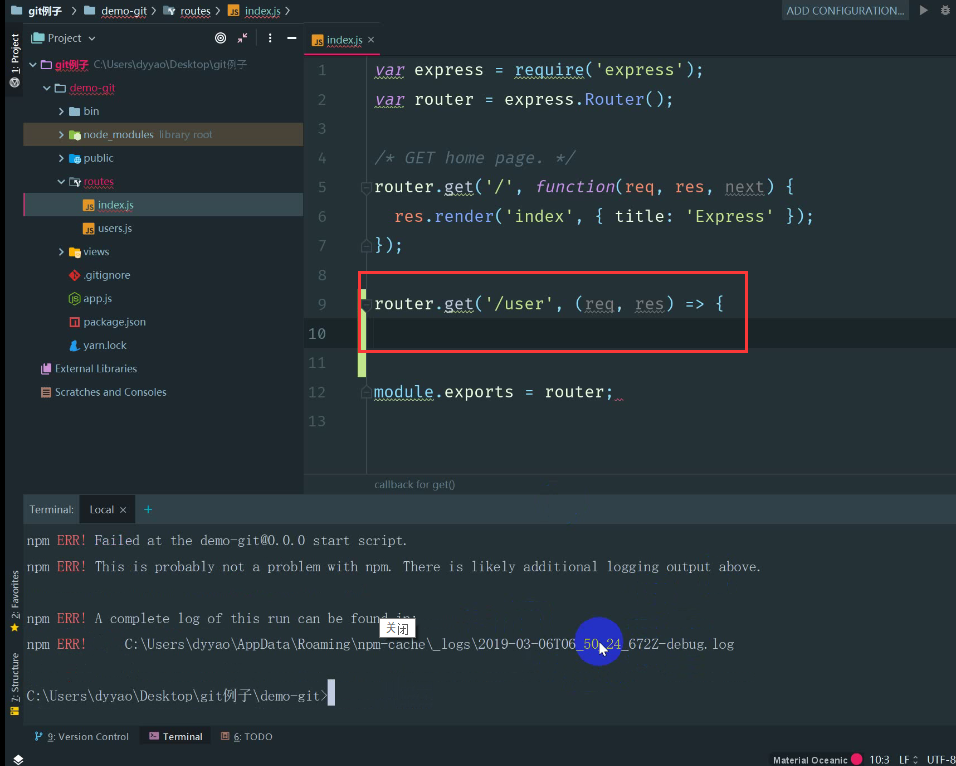
## 4)git分支系统

### (1)分支应用场景

通常用在后端 或是 新加功能 多一点

下面就以后端为例子

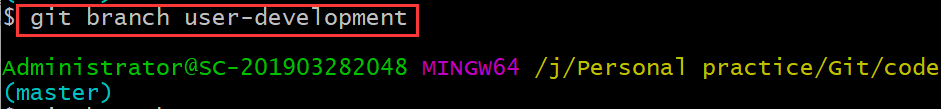
比如下面这种情况,代码还没写完,有事要出去,这时候代码是无法npm run dev的,会报错



### (2)使用方法

#### 1,新建一个分支

git branch 分支名



#### 2,查看所有分支

git branch

\*表示当前所在分支



#### 3,切换分支

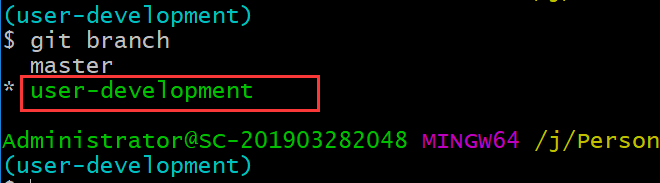
git checkout 分支名 转换到这个分支



例子:

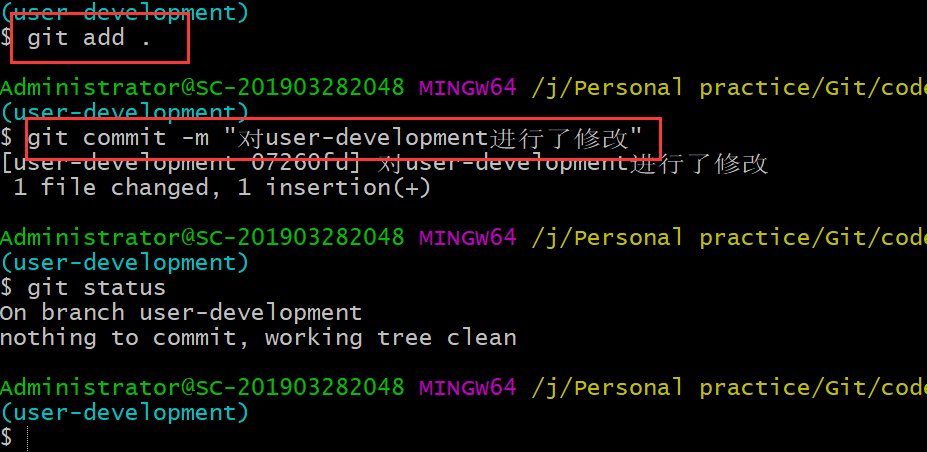
除了默认分支master, 我创建了另外的一个分支,叫user-development

我在user-development分支下进行代码修改



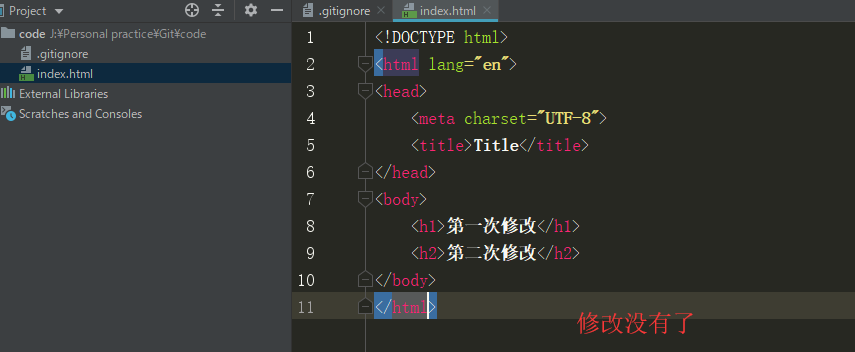


将修改提交到版本仓库



然后我切换会master分支

会发现修改没有了



结论:两个分支是互不干扰的

#### 4,新建分支并切换到该分支

git checkout -b 分支名

相当于git branch 分支名 并且加上 git checkout 分支名 的简洁写法

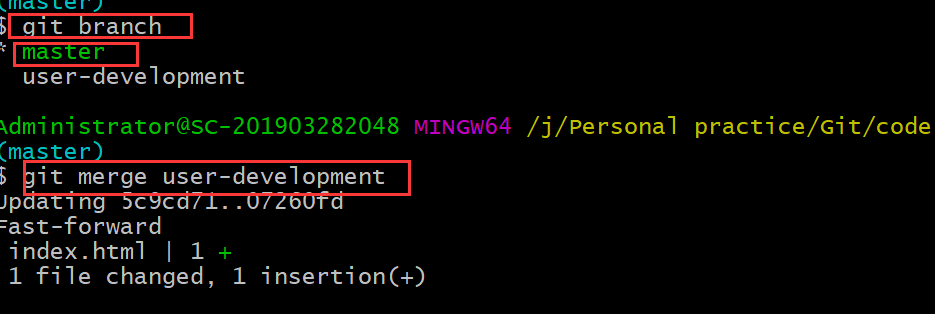
#### 4,合并分支

情景:就像上面转换分支里面的例子展示的那样,假设master是线上版本,user-development是开发版本,开发版本的功能已经测试通过了,但是还没有上线到线上版本。这个时候,我们就可以进行分支的合并。

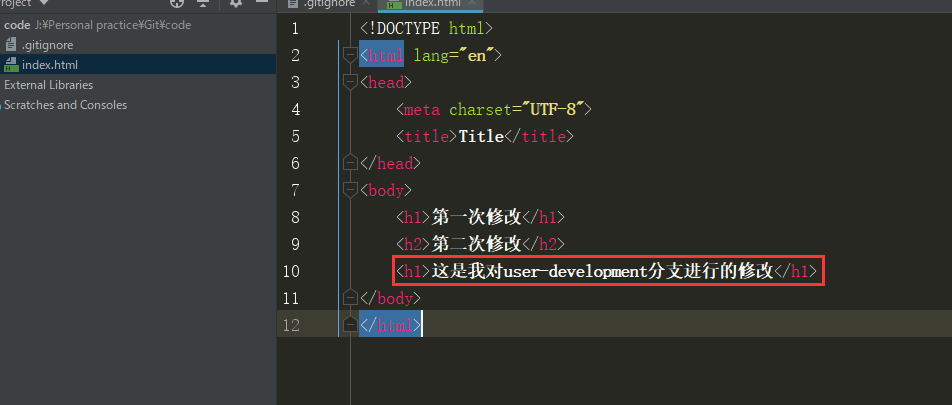
注意:分支合并是从当前所在分支开始合并的

现在的情况是:user-development版本是先于master版本的

git merge “版本靠前分支(目标版本分支)” 将user-development分支合并到master分支

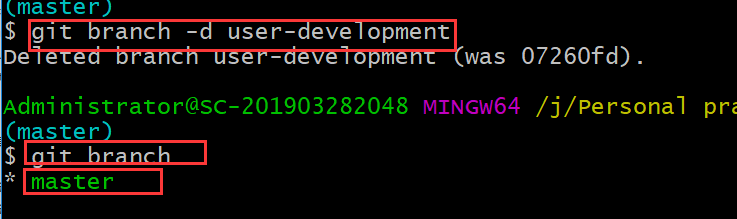


合并成功了



#### 5,删除分支

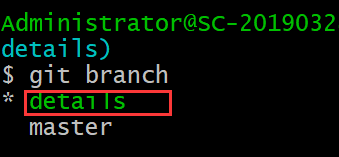
git branch -d 要删除分支

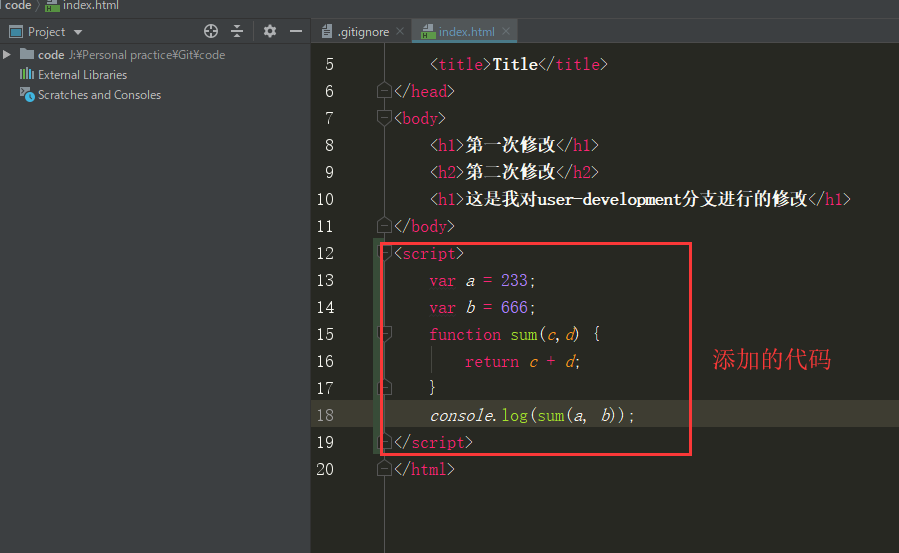


删除成功

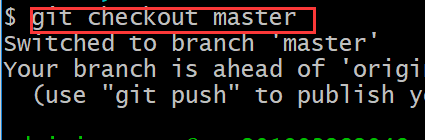
#### 6,解决分支冲突问题

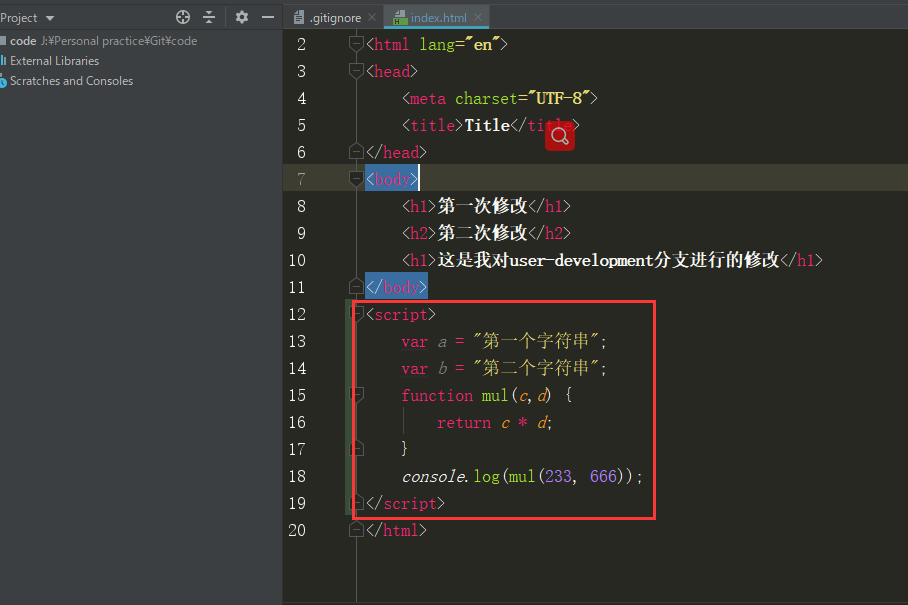
我新建了一个分支details,并对文件进行了修改



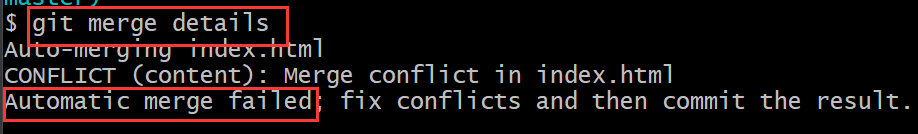


然后我切换回主分支,也进行代码的修改

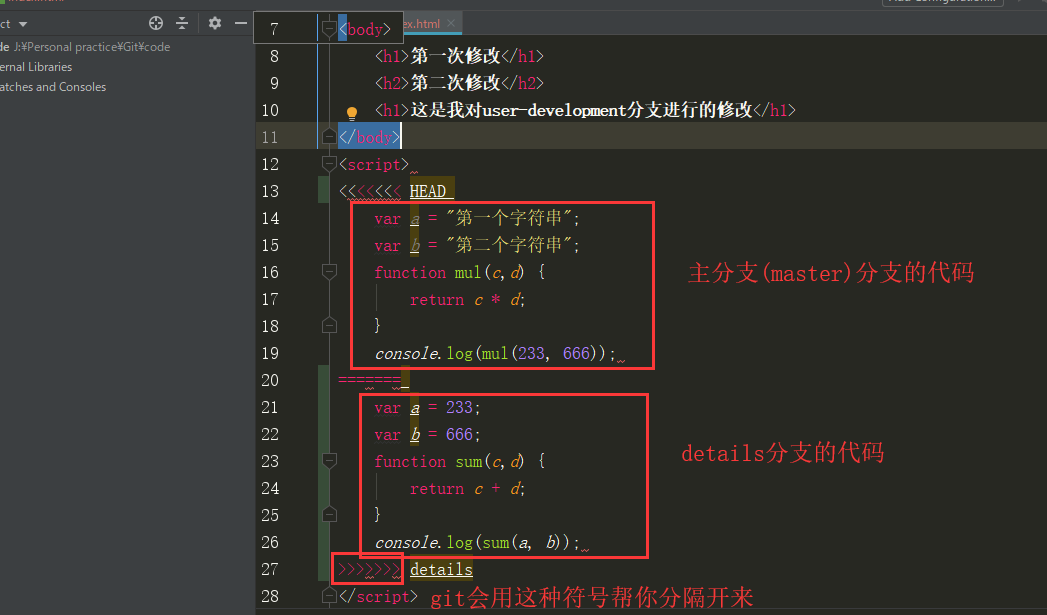




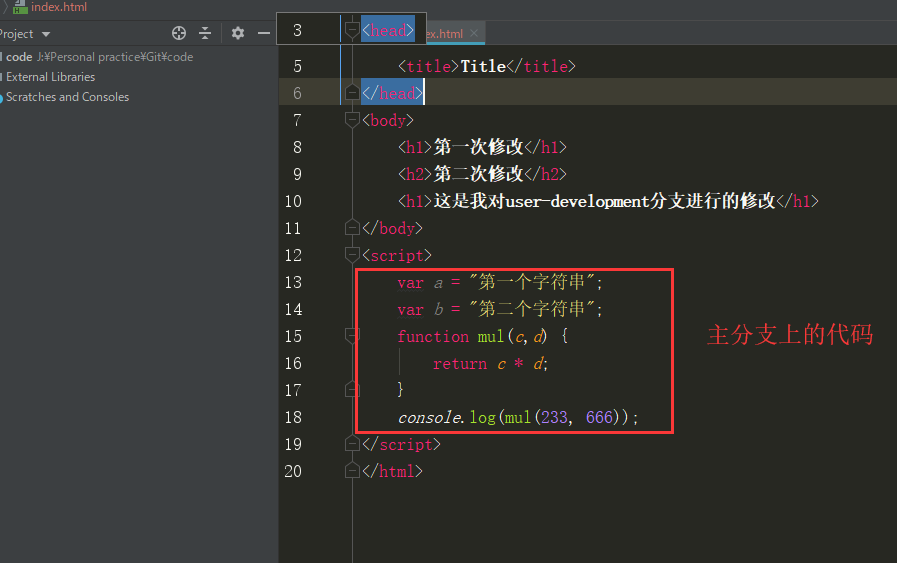
然后我在master分支,对details分支进行合并



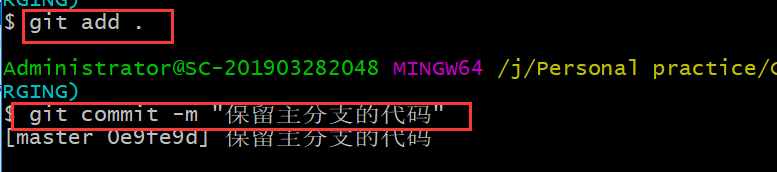
提示合并失败,代码会变成下面这样



你需要确认你需要保留哪一部分,然后把另外一部分删掉就行了。比如这里我需要保留主分支上的内容



然后你就可以进行提交了



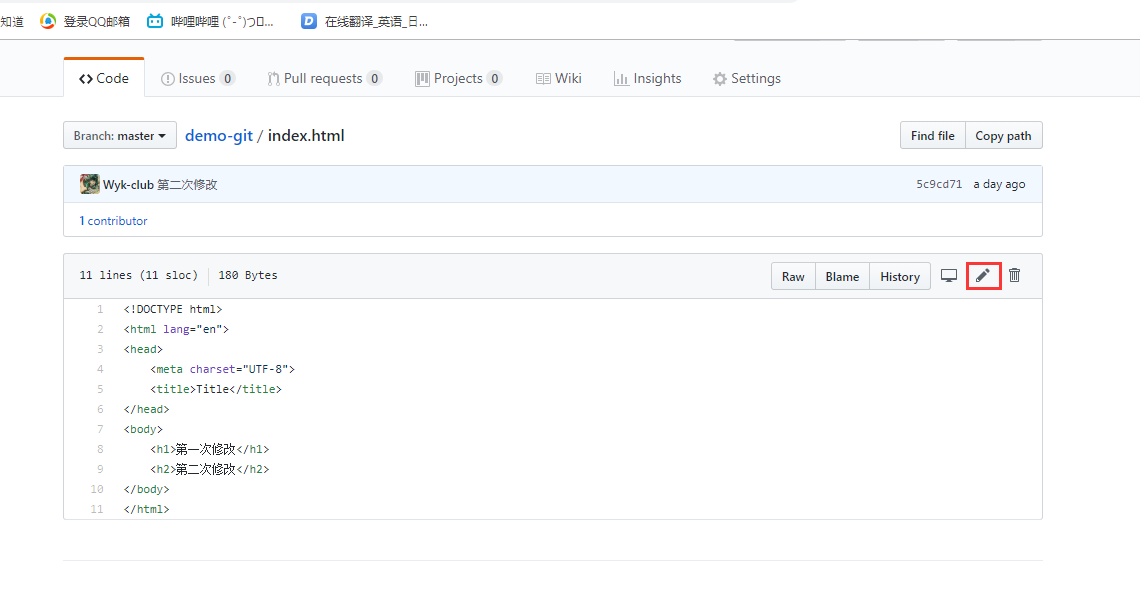
之后你还可以把details分支删除了,因为它里面的代码已经没用了

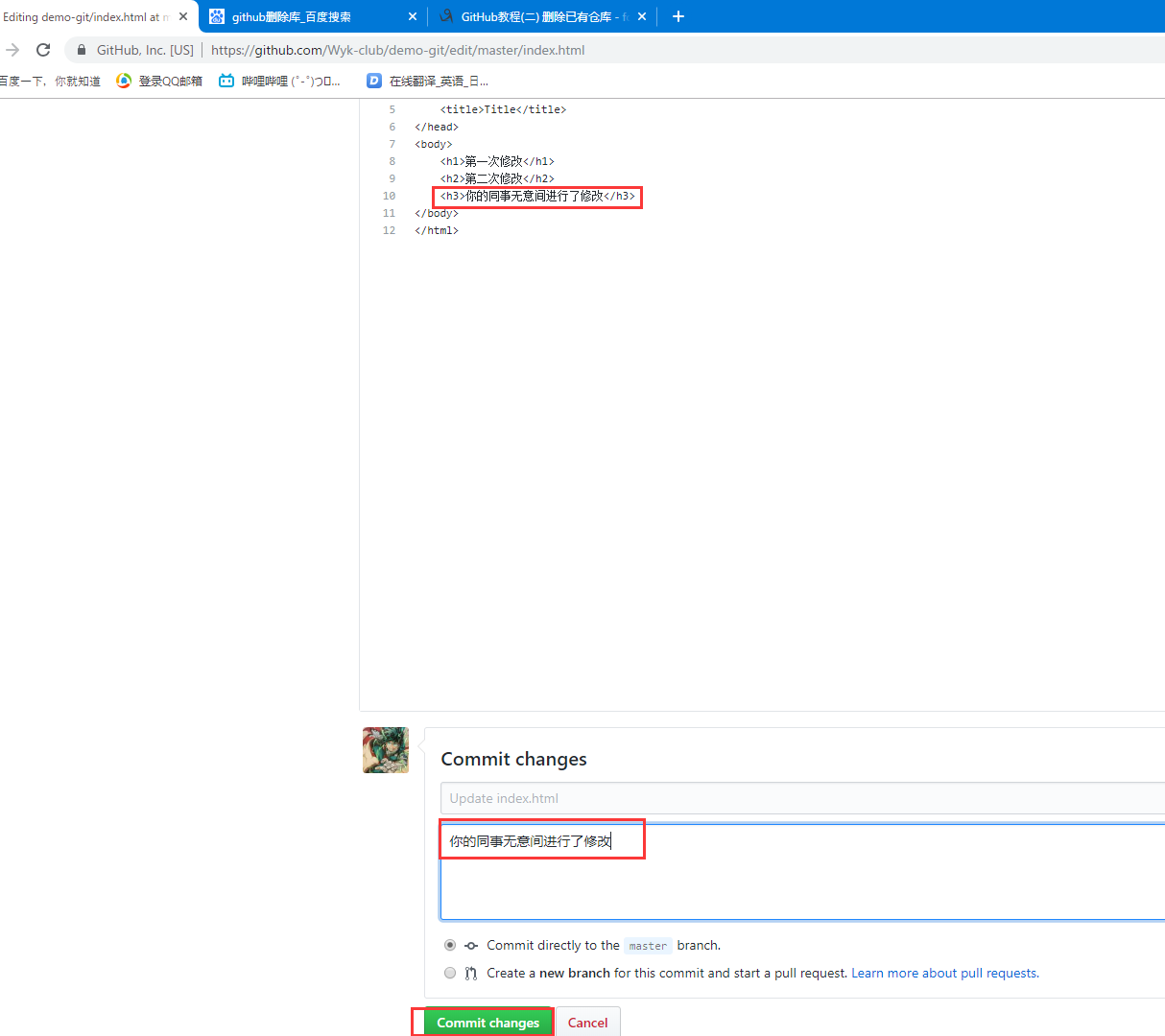


#### 7,解决push远程库失败的问题

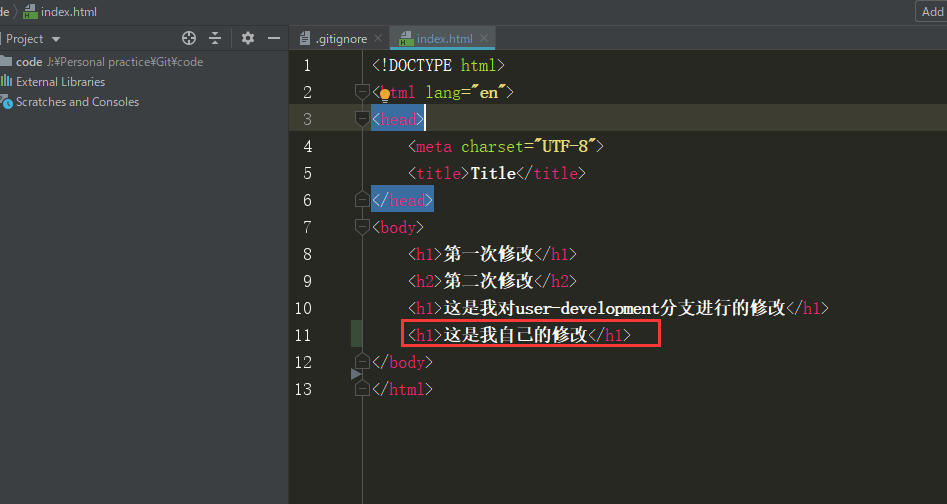
情景是:你的同事在线修改了你在github上的代码,你在修改相应文件时,忘了进行拉取,直接进行了修改,当你push的时候,就会出现问题,上传不上去

你的同事进行了修改





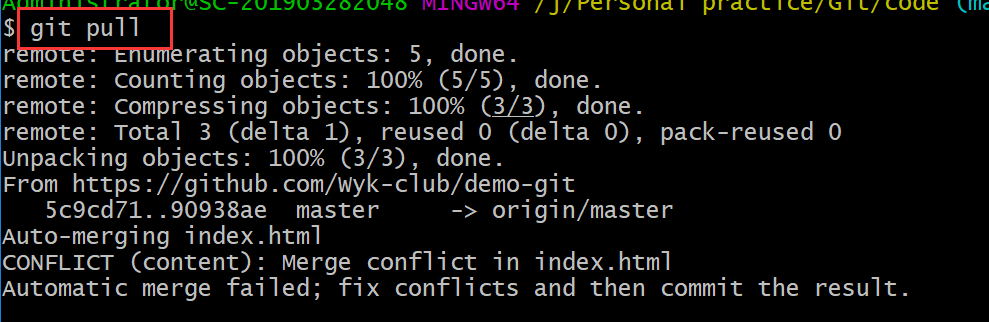
你不知道你同事的修改,没有git pull 直接开始写你的代码

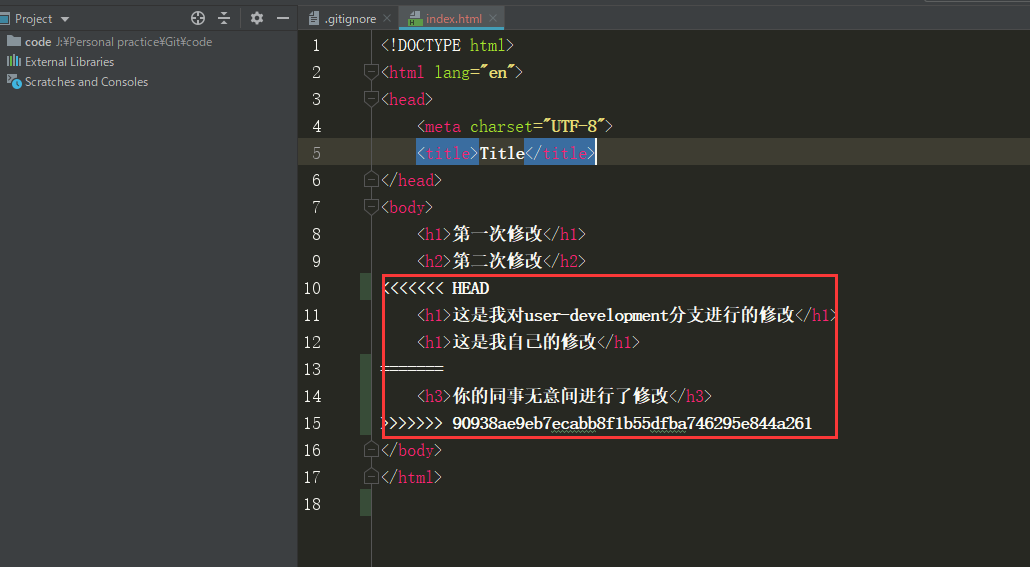


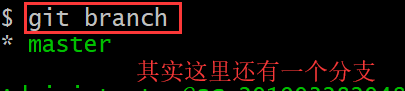
然后你进行提交



你需要先git pull,把服务器代码拉下来

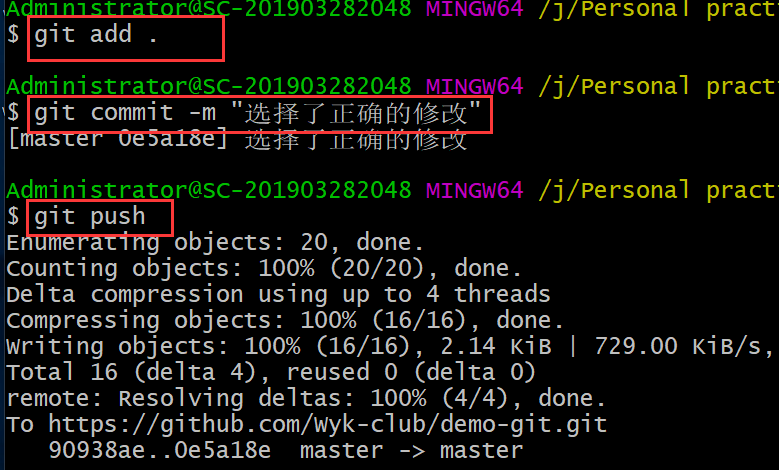




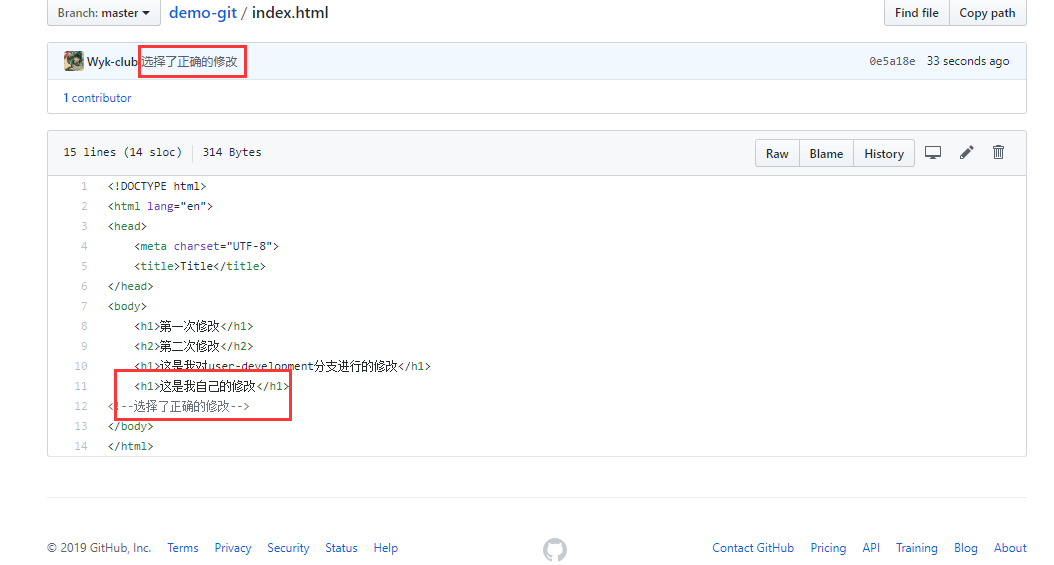
查看分支,看似只有一个,其实还有一个分支没有显示出来  


你需要把两个分支合并。选择你需要保留的代码,然后提交,然后推送到服务器





这样就push了你需要的代码

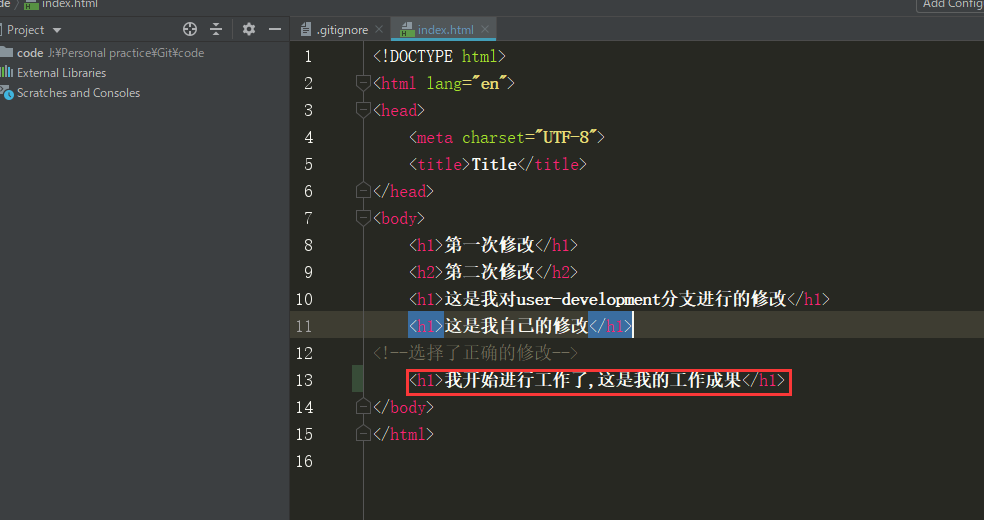


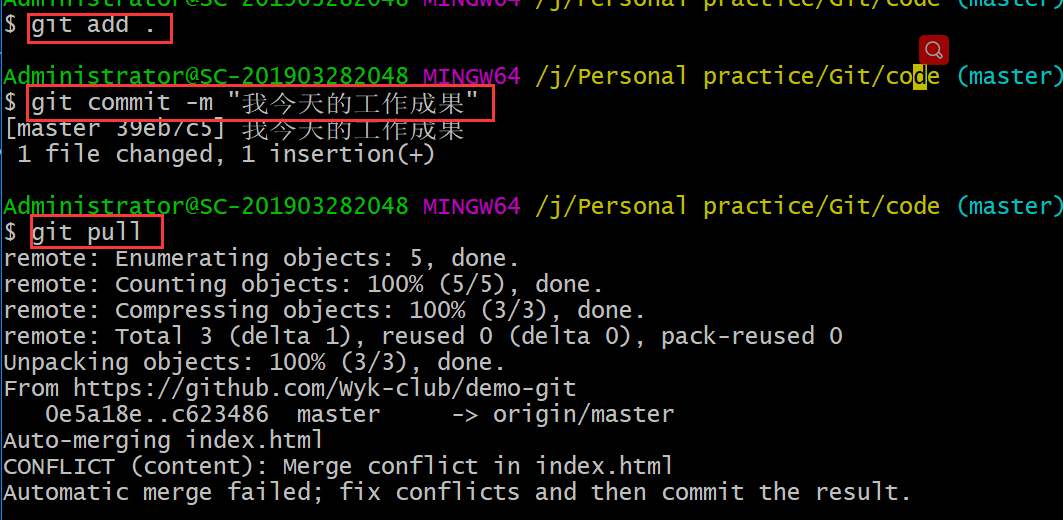
#### 8,解决pull失败的问题

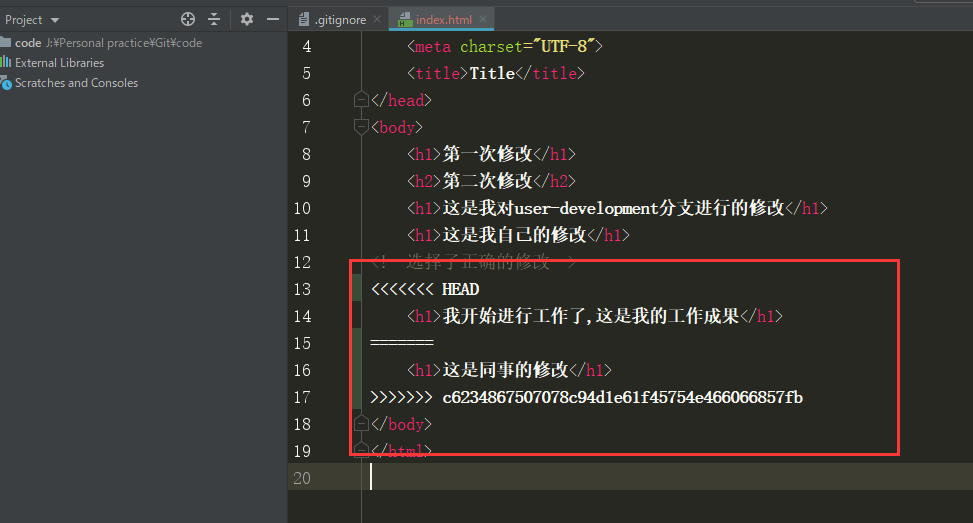
和上面(7)的情景类似,不过一个是push,一个是pull

情景是:你的同事在线修改了你在github上的代码,你在修改相应文件时,忘了进行拉取,直接进行了修改,当你pill拉取代码的时候,就会出现问题

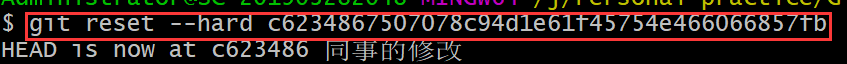




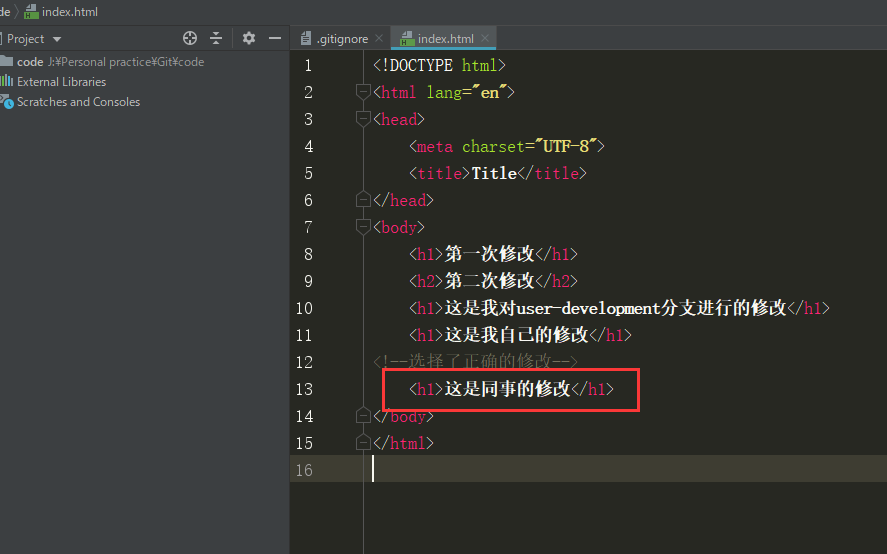
代码变成了下面这样



看你自己选择,你可以修改选择你需要的代码,或是你可以直接恢复到你同事修改之后的版本。我这里恢复到同事的版本

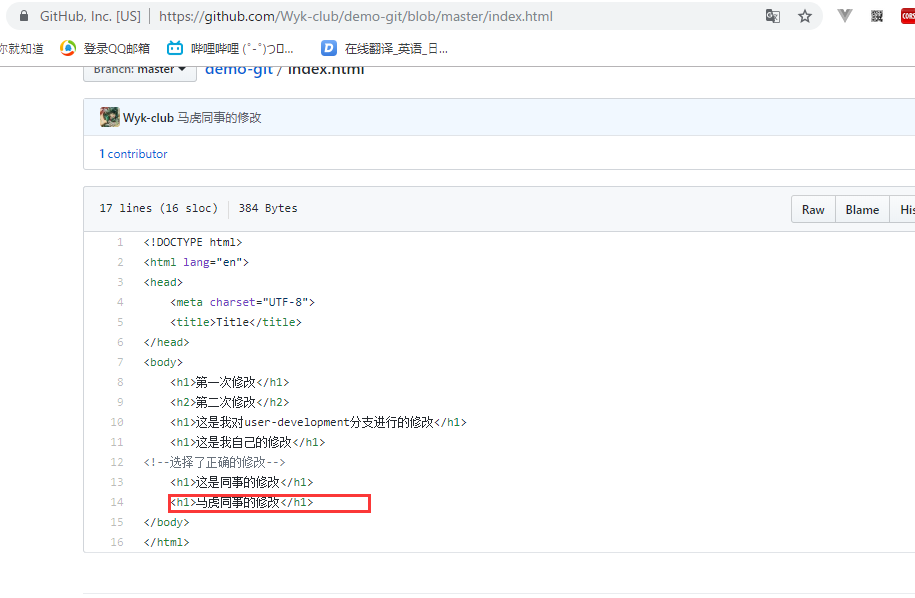


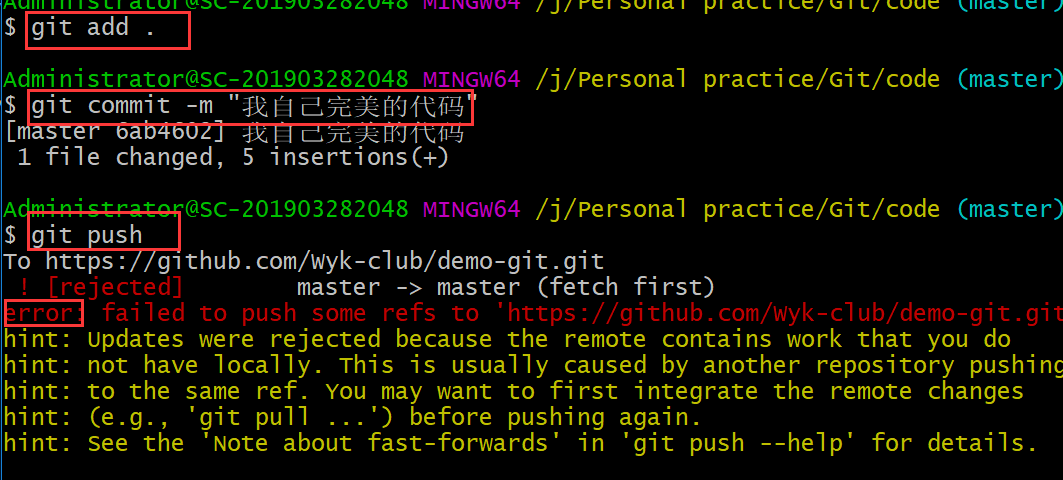
代码就恢复到了对应版本



#### 9,强制提交—push失败的另一种解决方法

情景是上面7,出现push远程库失败





除了上面的先pull,修改代码之后再上传,还有一个简单粗暴的方法(不推荐使用),就是

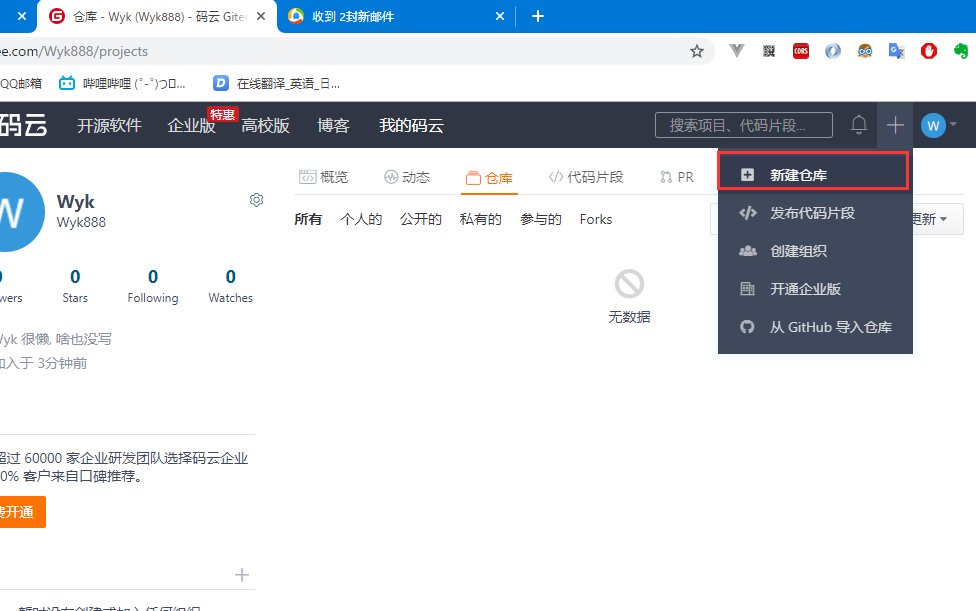
git push -f

这种提交方式有一个很大的缺点,它不会保留服务器的所有版本资料,但是能保证本地库和服务库完全一样

## 5)ssh公钥

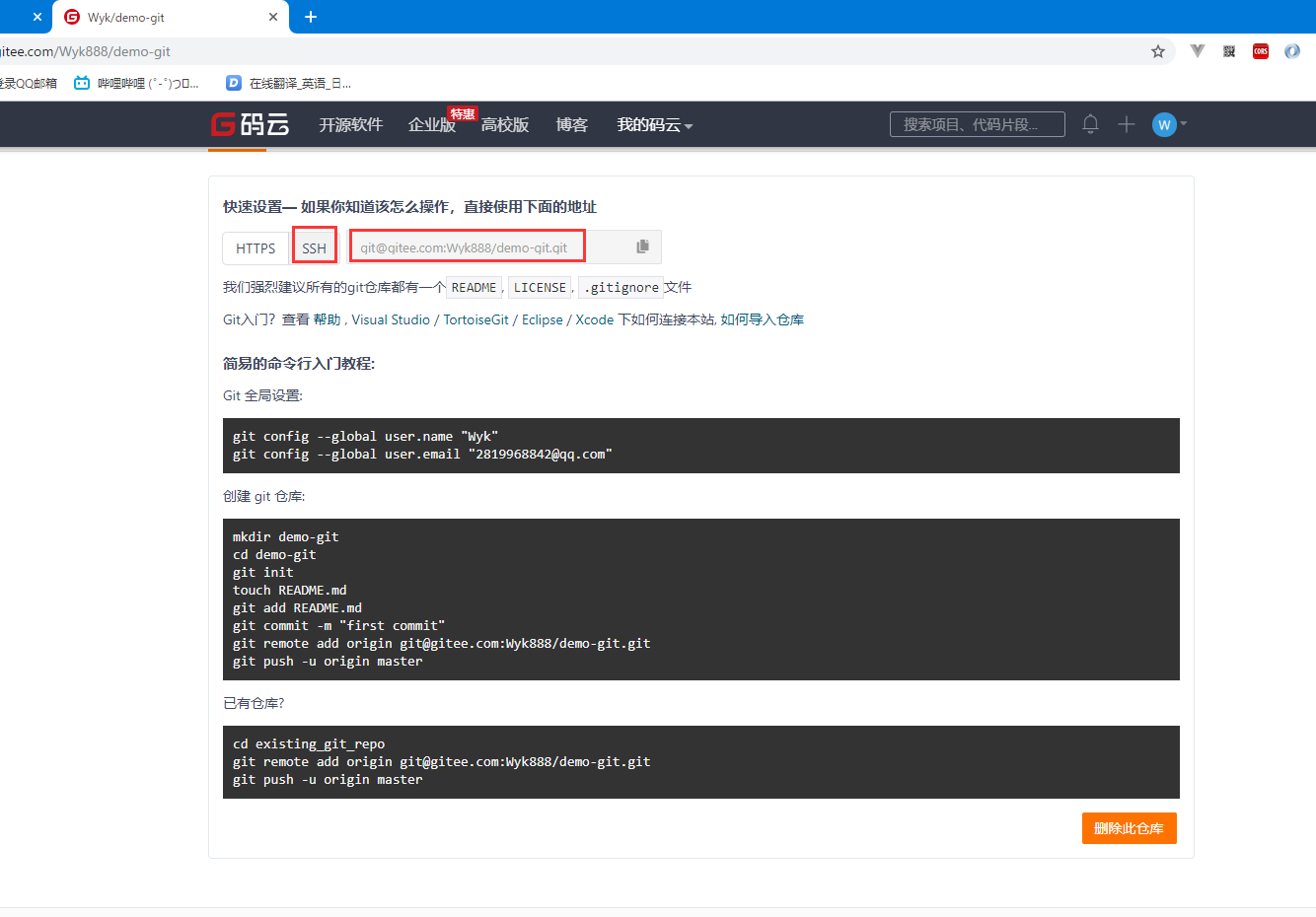
使用码云

新建一个仓库





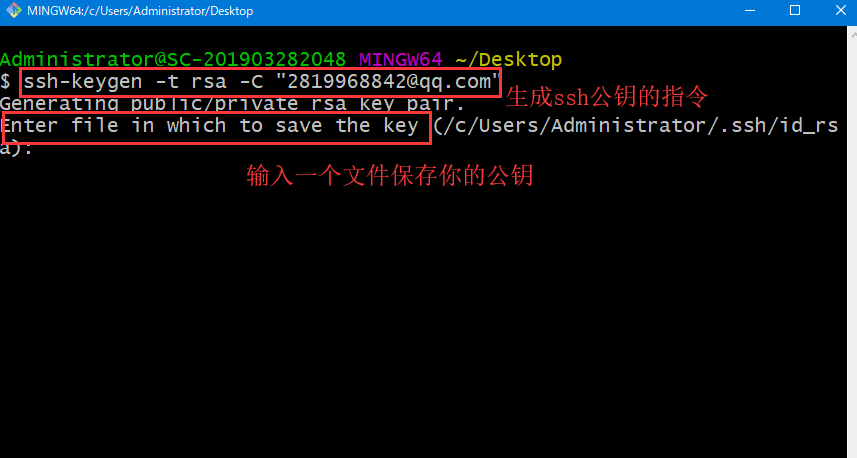
选择SSH



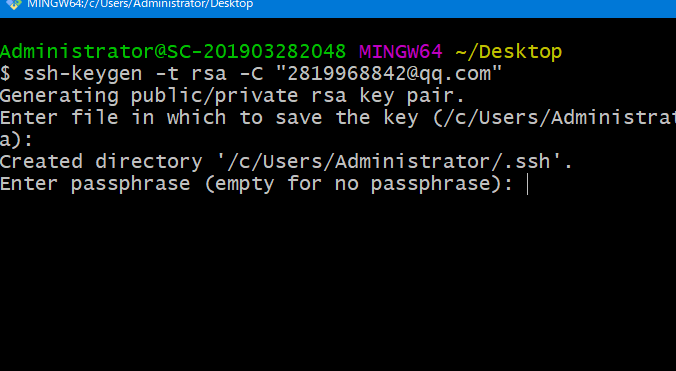
在桌面右键,选择Gt Bash Here

-C “xxx” -C后面的相当于注释

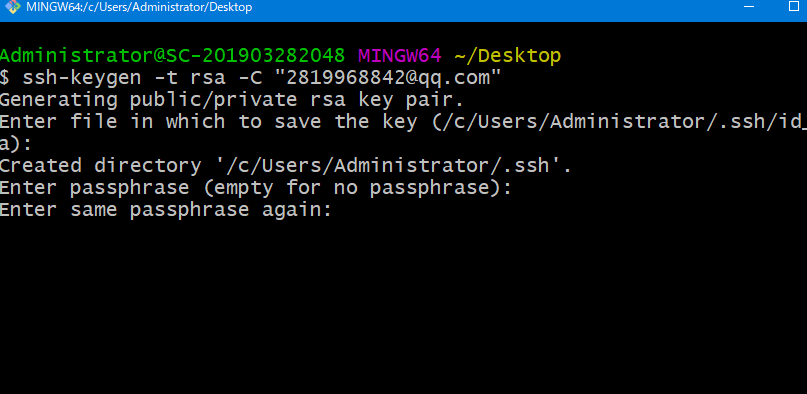
输完此命令后敲3次回车,生成公钥



我不输,直接回车



接着回车

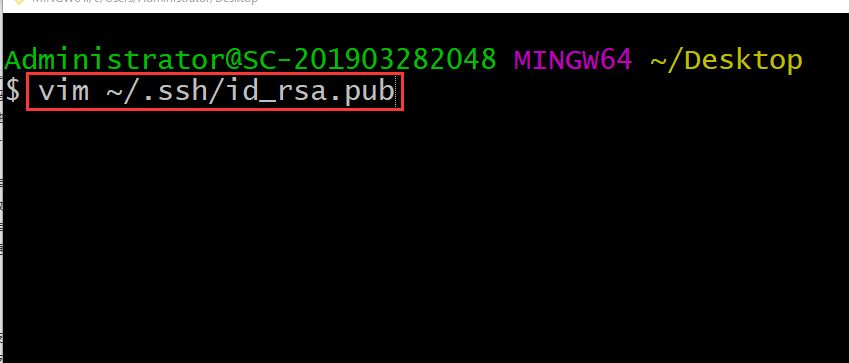


再回车

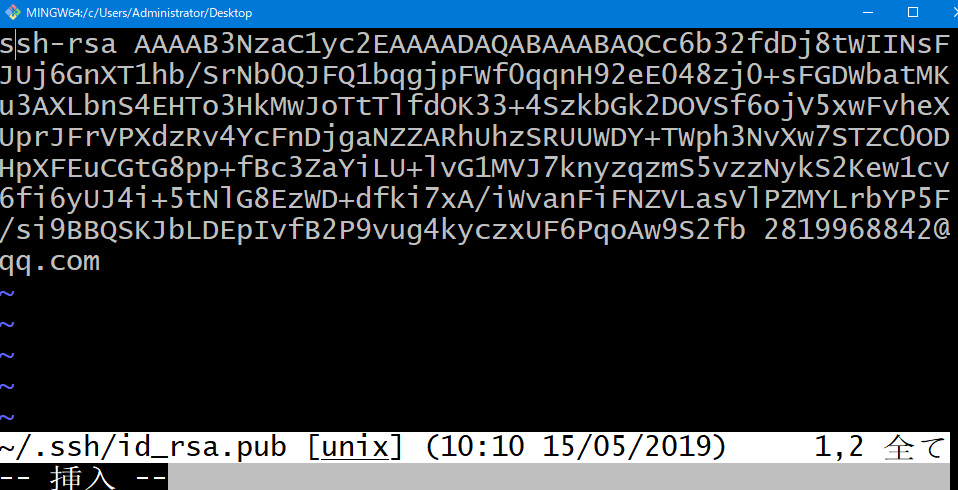
这样就生成了一个公钥

那么生成的公钥在哪呢?

我们用下面这条指令取到公钥

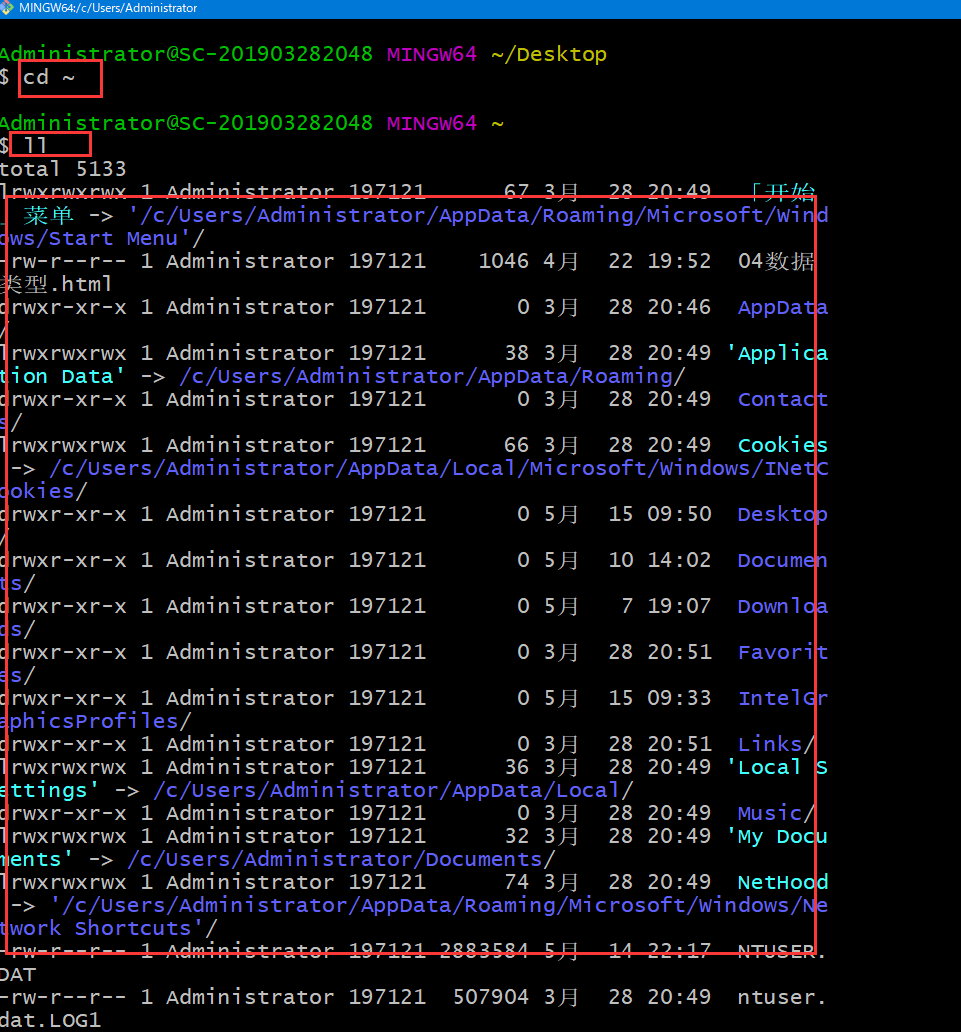


回车 显示的就是你的公钥信息



注:~表示用户目录

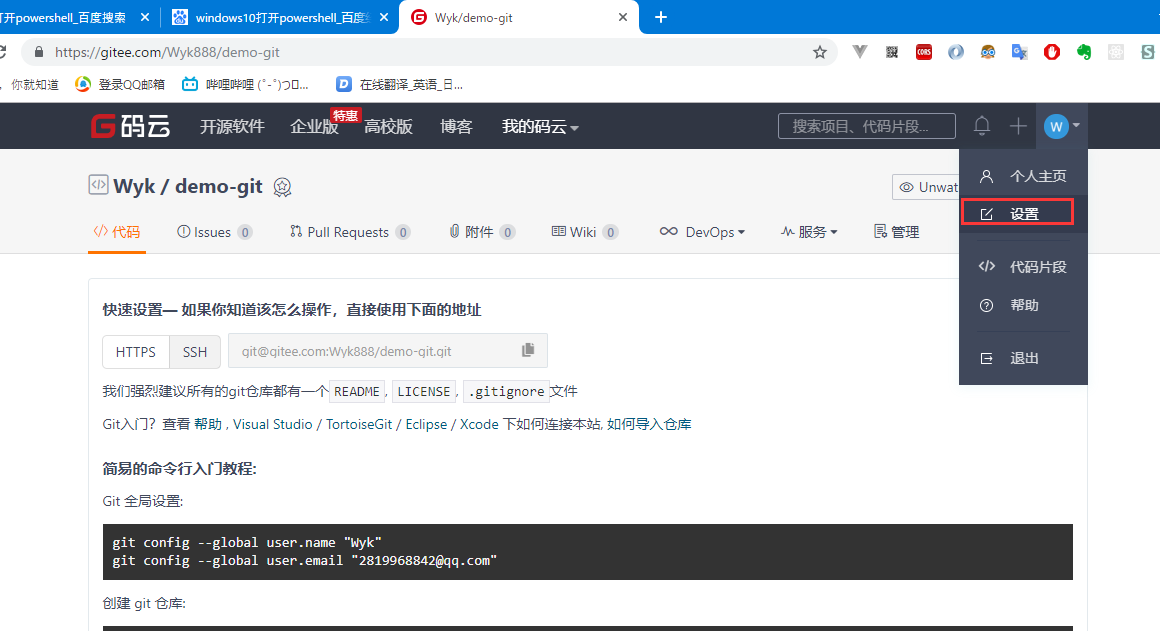
我们可以看下~代表什么 我们cd 到~目录,然后ll



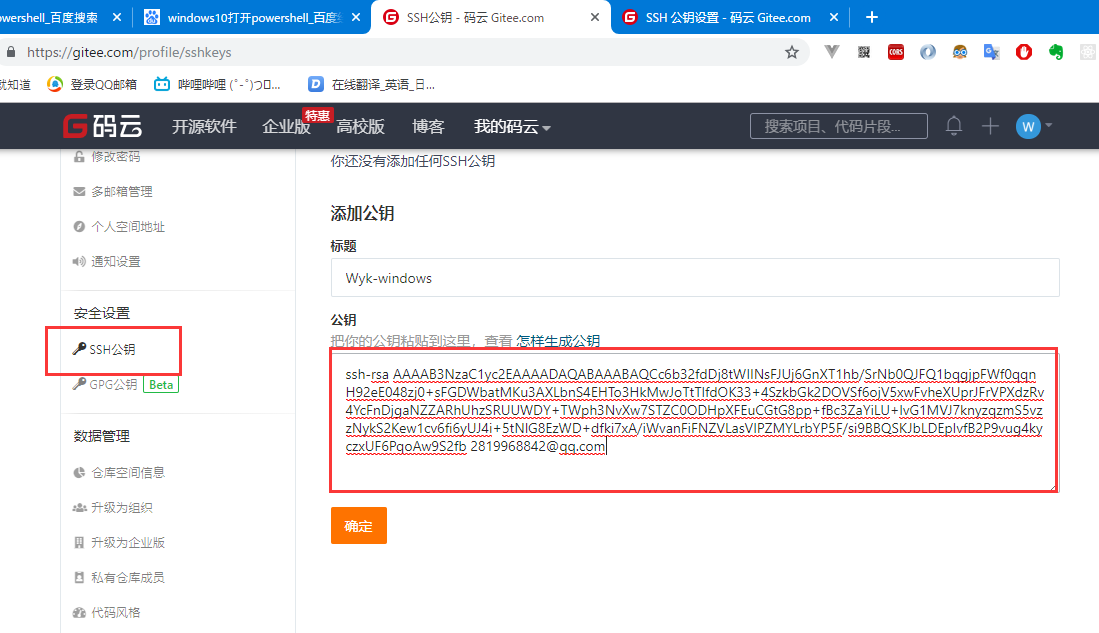
很明显,~表示的是你的用户目录

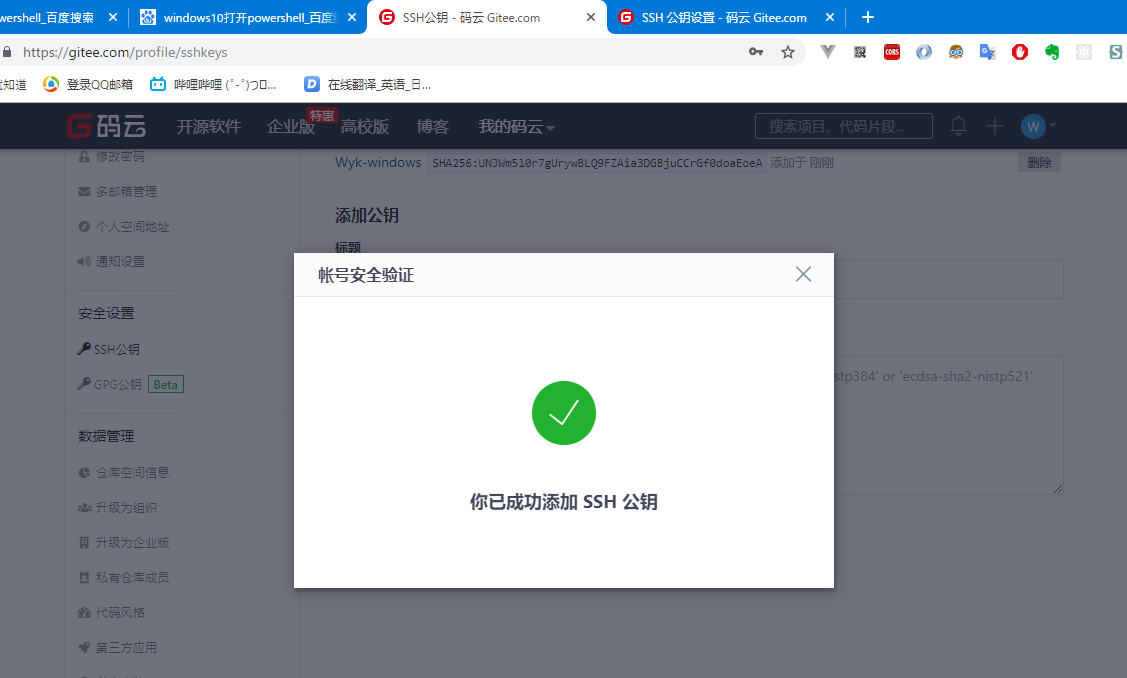
那么,公钥要怎么使用呢?

选择设置

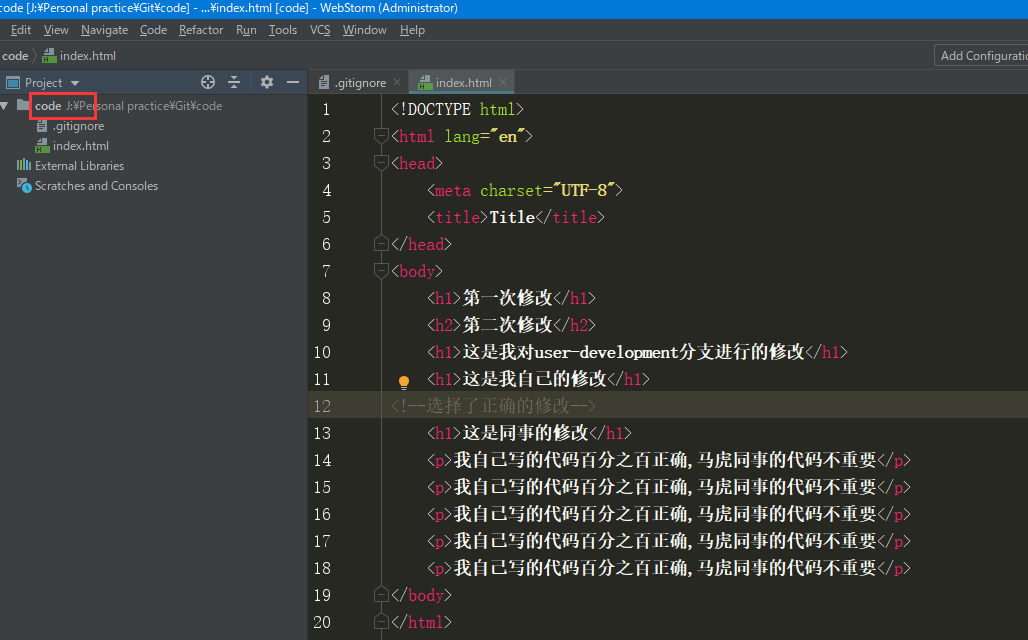


选择ssh公钥,把你的公钥粘上去





比如我要往远程库推送code文件夹

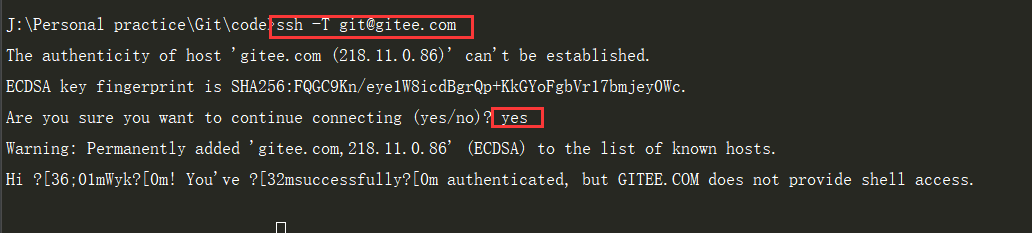


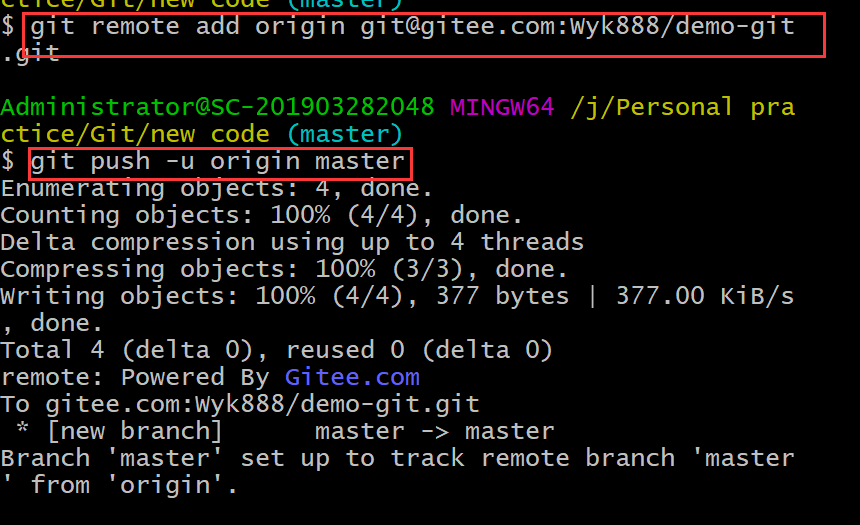


然后粘贴使用下面的两行代码



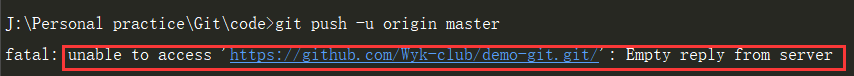




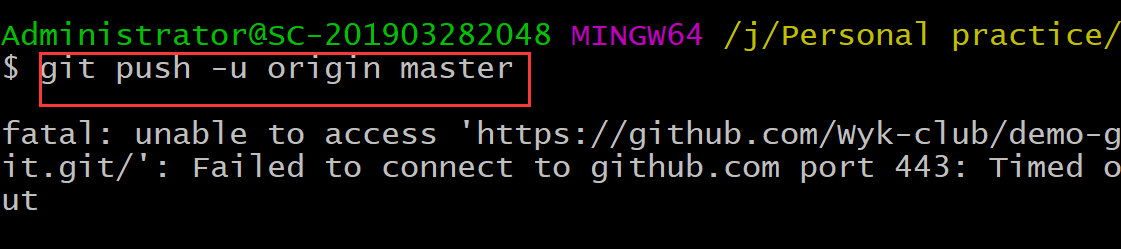


上传成功了

这里出现了一个小插曲,我将已经上传到github上的文件没有修改,原封不动的用码云的ssh进行了上传,会报错



然后我新建了一个文件夹,初始化了仓库,就上传成功了。我又试了一下,就算更改了代码,再上上传,也还是会报错。



好像是这个文件夹已经连接到github上的库,不能上传到码云的库了。想要上传到码云的库,必须重新初始化一个新的仓库。