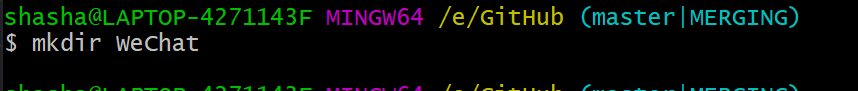
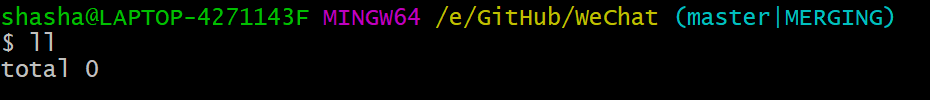
# 对本地库进行初始化

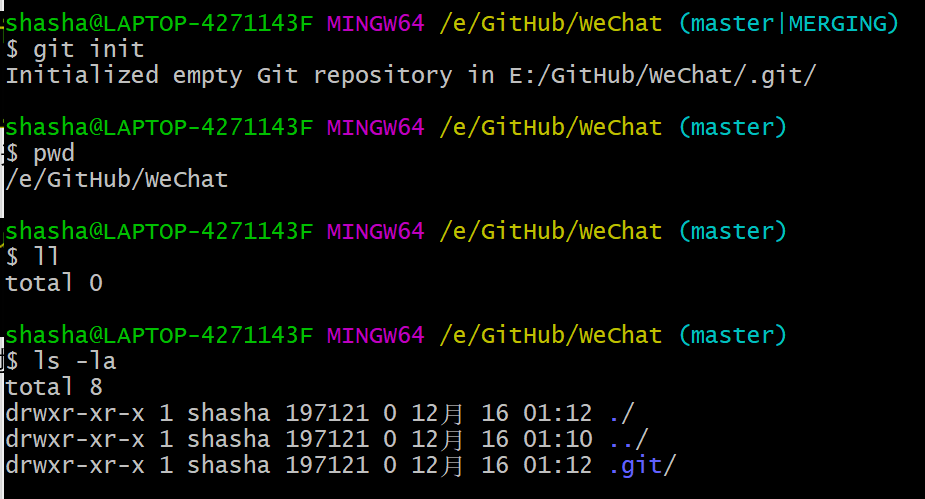
* Git base here打开命令行界面
* mkdir WeChat 创建一个名为“WeChat”的文件夹



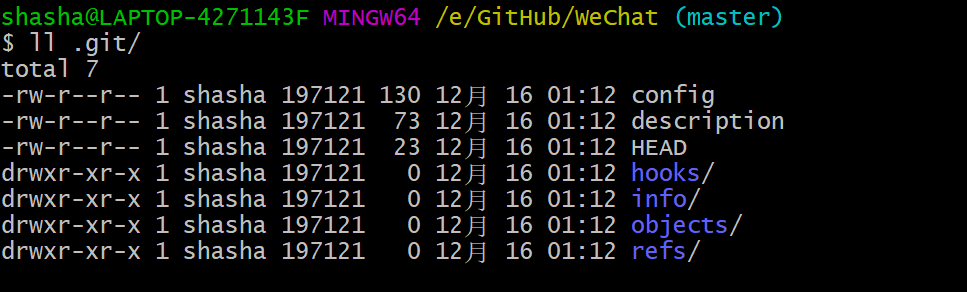
* cd WeChat/ 进入WeChat 文件夹，此时，该文件夹为空，可以通过ll来查看当前目录下文件：



git init初始化仓库，此时会生成.git隐藏文件，可通过ls -la查看当前目录下所有文件（包括隐藏文件）



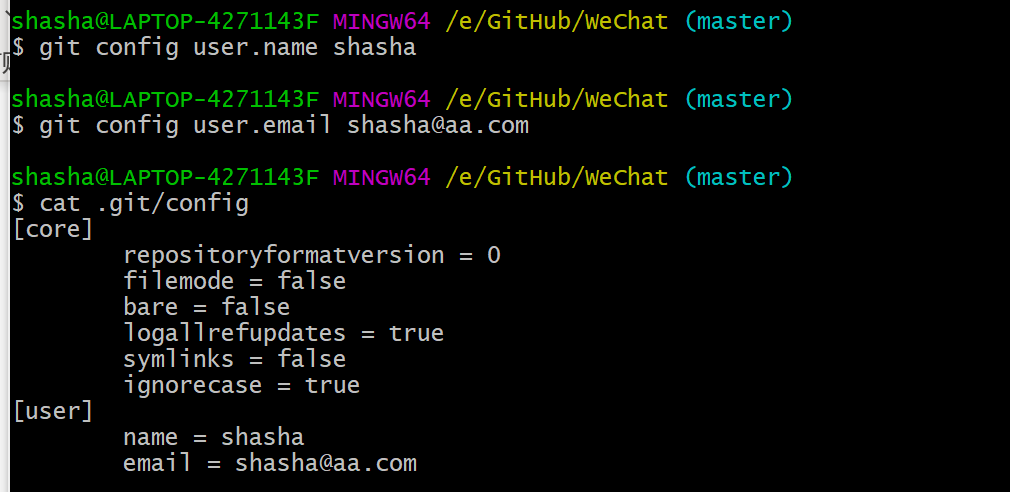
.git文件内部有如下文件



设置签名：git config 命令

项目级别:git config user.name xxx

Git config user.email [xxx@xxx.com](mailto:xxx@xxx.com)

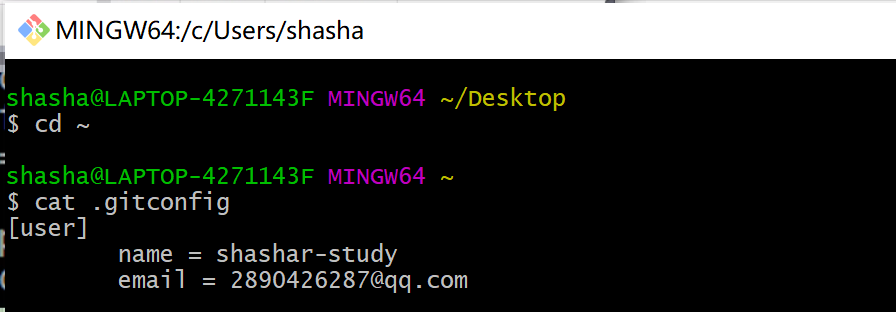


设置的信息存储在.git中的config文件中

系统用户级别：git config --global user.name xxx

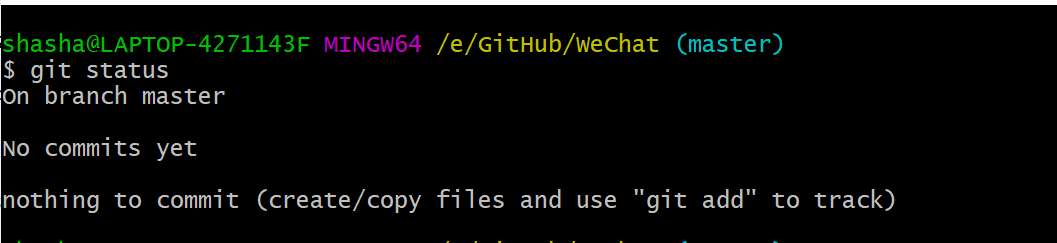
Git confid --global user.email [xxx@xxx.com](mailto:xxx@xxx.com)

同理的，系统用户变量存储在.gitconfig



# 基本操作：

git status查看工作区和缓存区状态



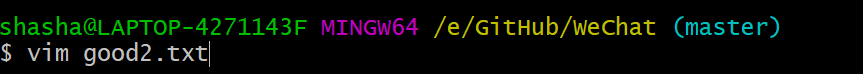
On branch master:在master分支上

No commits yet :没有任何提交

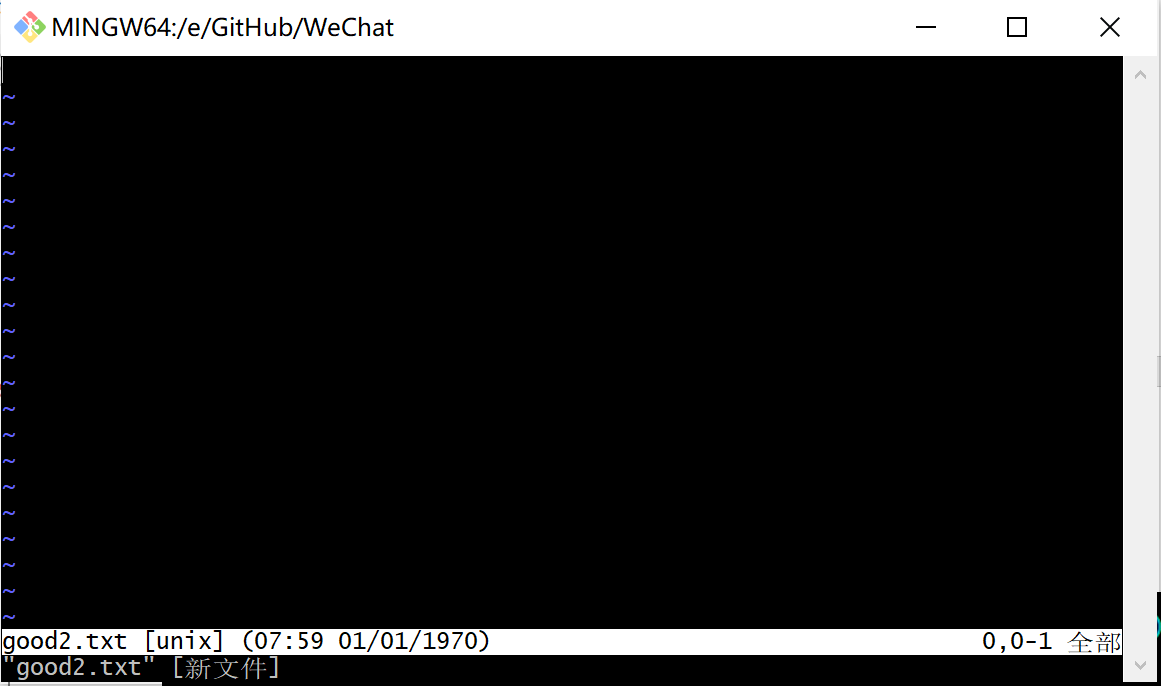
Nothing to commit :没有可提交的东西（暂存区里无东西）创建或者复制文件，使用git add命令追踪该文件

vim xxx.txt 使用vim命令创建一个xxx.txt 文件，随便写一点内容

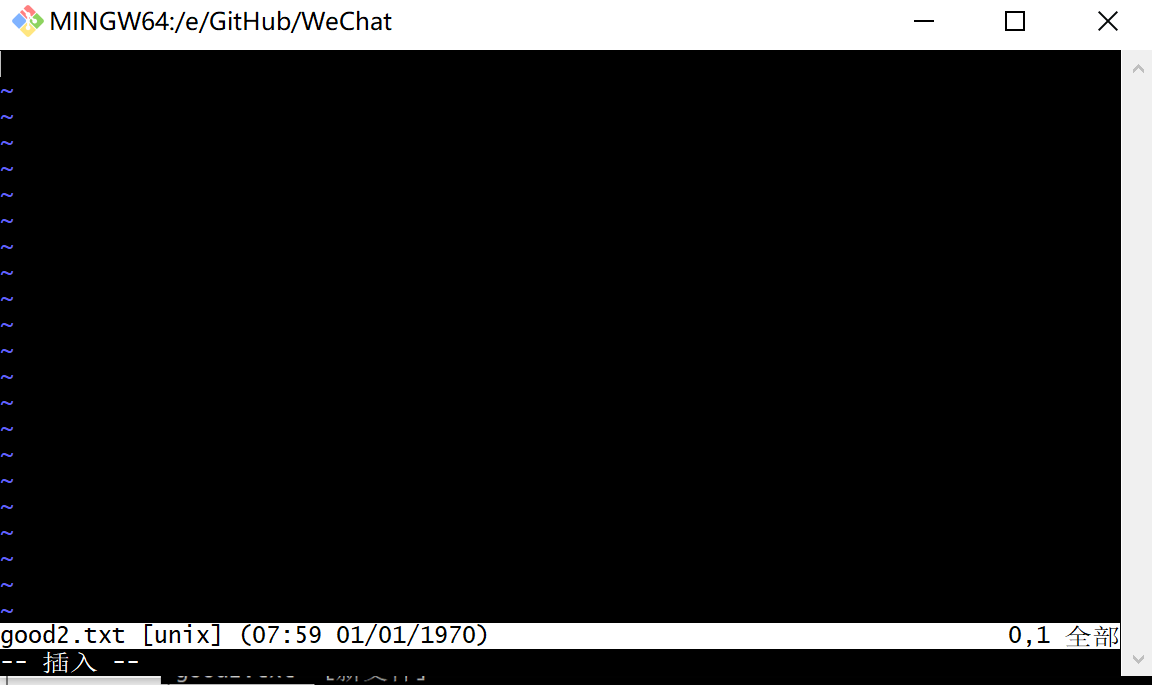
注意 :步骤1：vim good2.txt



回车，进入vi 的一般模式：

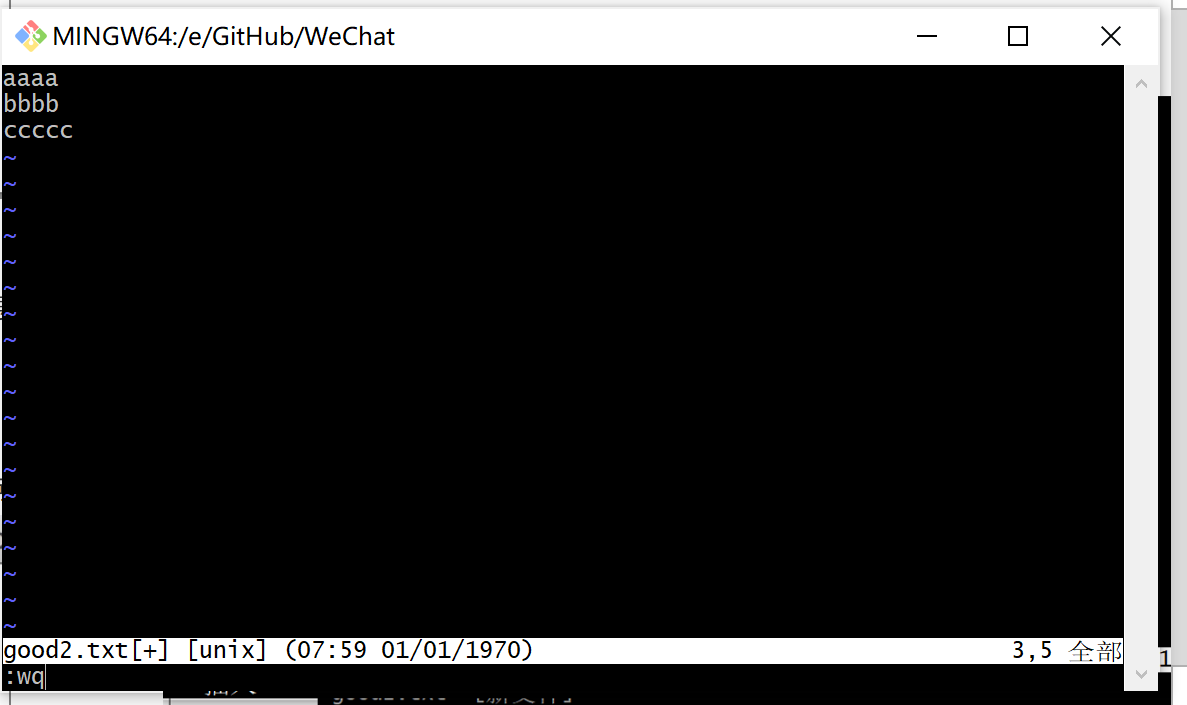


在这里~代表无内容，在一般模式下，按下I,o,a等字符可以进入编辑模式

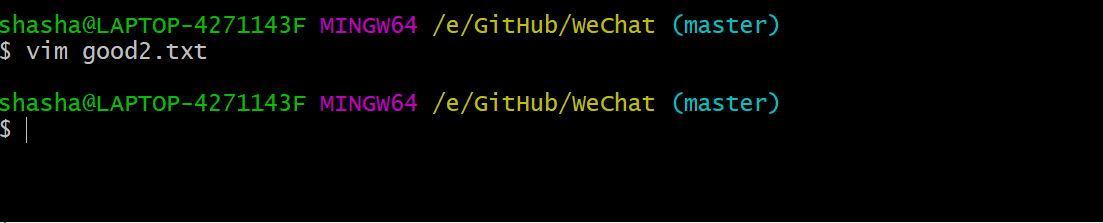


此时，可以进行写入部分内容，写入完成后，esc回到一般模式

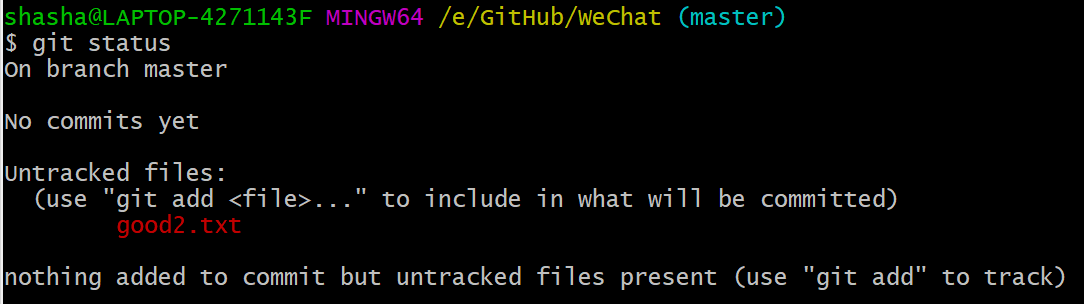
在一般模式下输入:wq储存后离开



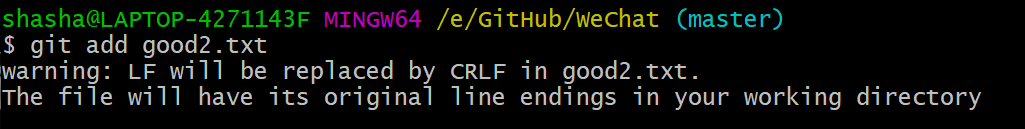
再次回车后回到上一界面：



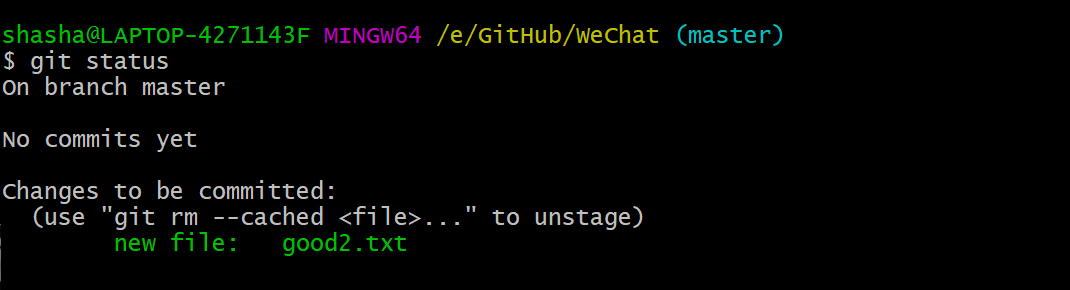
此时使用 git status 命令



发现未追踪文件，提示可以使用git add把该文件放入暂存区，根据提示使用命令



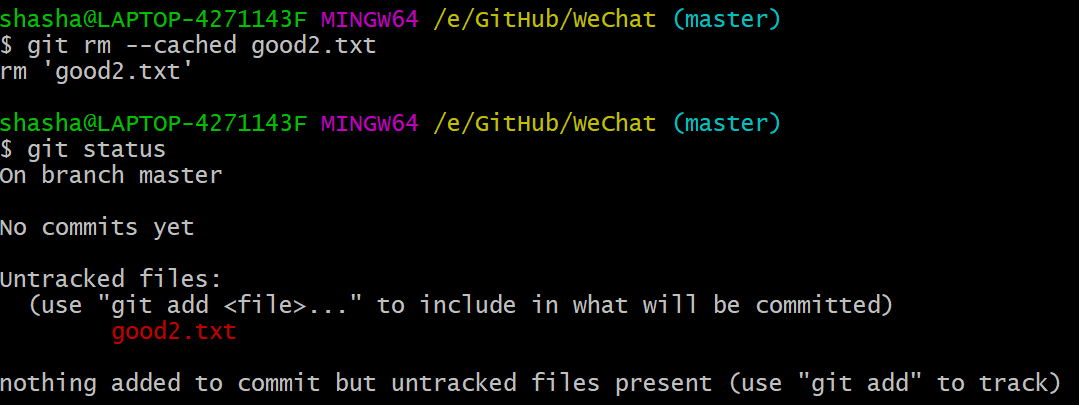
警告：LF会被替换为CRLF



此时，good2.txt状态发生变化，存入暂存区，颜色变为绿色

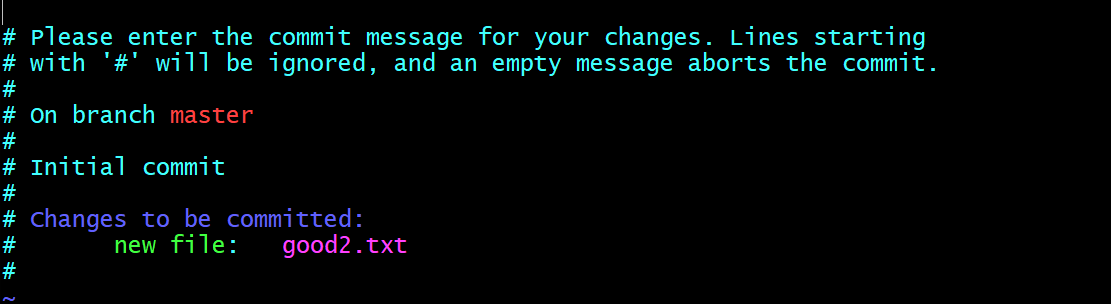
有改变可以被提交，出现了新的文件

根据提示，使用命令git rm --cached命令可以将文件重新转移出暂存区



可以看到 ，good2.txt回复之前的状态，在工作区内而不再暂存区中

接下来使用 git commit good2.txt命令进行提交，出现以下界面：

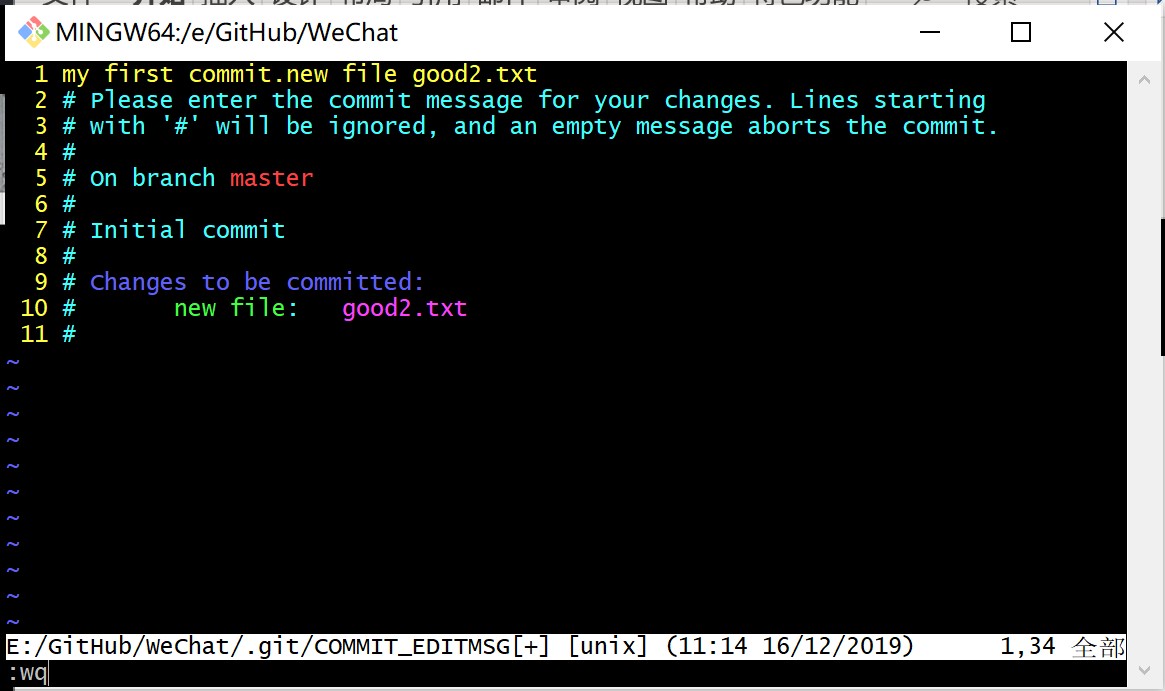


第一行提示输入提交信息，相当于注释，进入了vim编辑器中

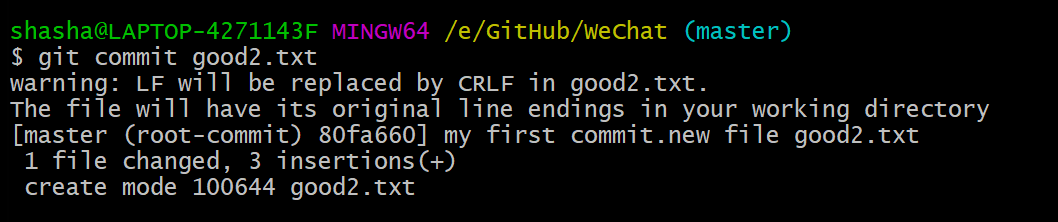
进入一般模式后，使用:set nu 可以显示行号



接下来，进入编辑模式，编辑信息后退回一般模式，使用:wq保存并退出



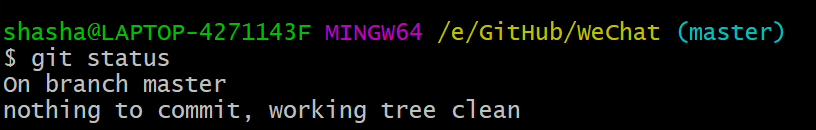
此时git commit 命令已经完成



第一次提交作为root-commit,后面的80fa660目前可以认为是版本号

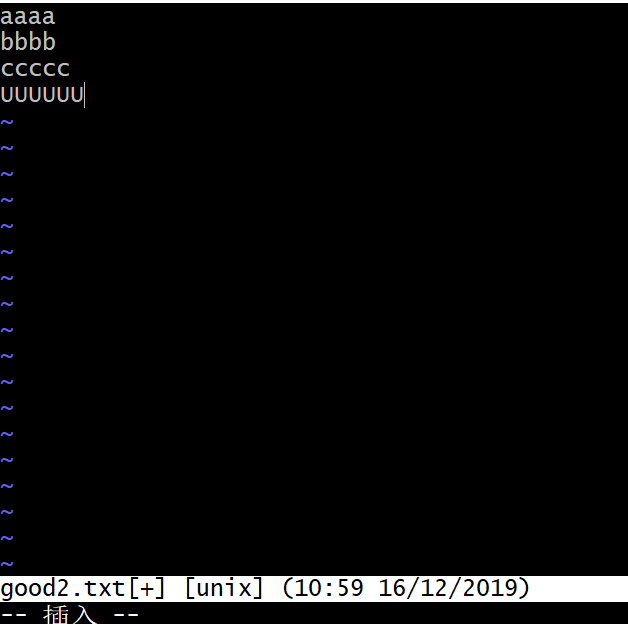
一个文件改变了，新增了3行，因为之前在good2.txt中写入了三行文字们可以对应的上

此时，再看一下状态

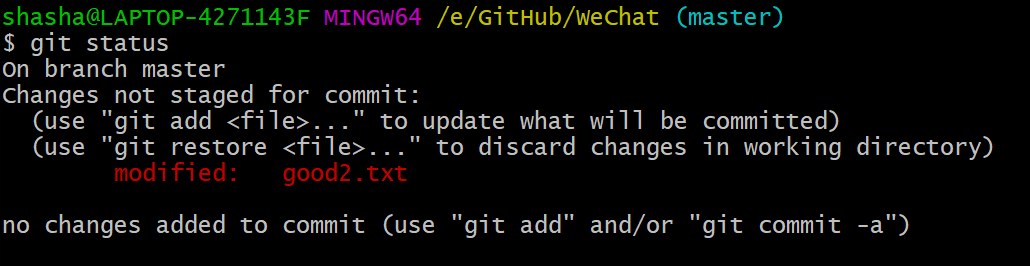


目前暂存区里没有要提交文件，工作区无修改

接下来，我们对于good2.txt做一下修改，使用vim 命令，写入保存退出，如下：



再使用git status查看



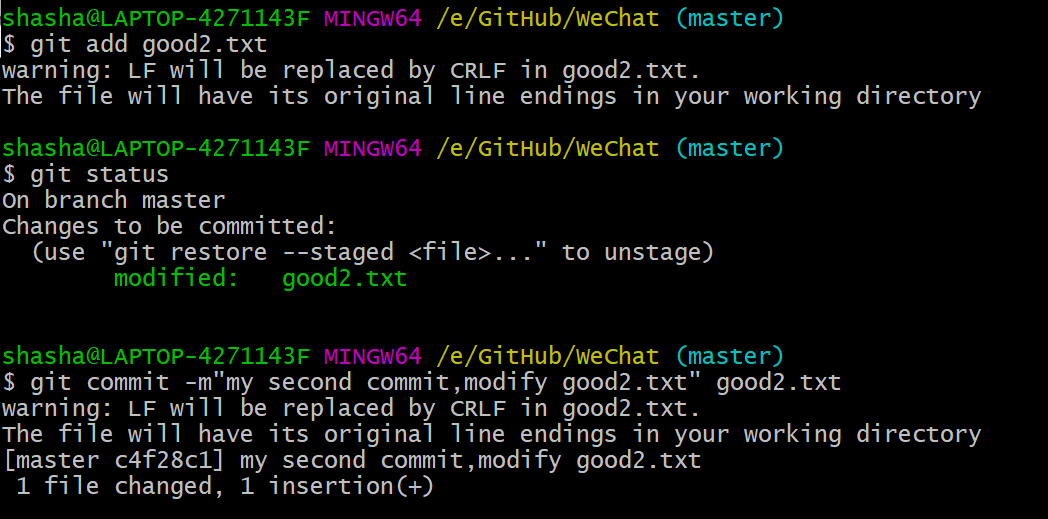
红色字体提示modified,文件修改，但是没有放进暂存区

提示可以使用git add命令进行update;或者使用 git checkout 取消修改

可以使用git add或者直接使用 git commit -a提交

接下来使用git add，修改放入暂存区

提交时，推荐使用git commit -m”XXX” good2.txt命令，可以不进入VIM编辑器，直接在XXX部分写入注释，结果如下，可以看到，版本号已经更新

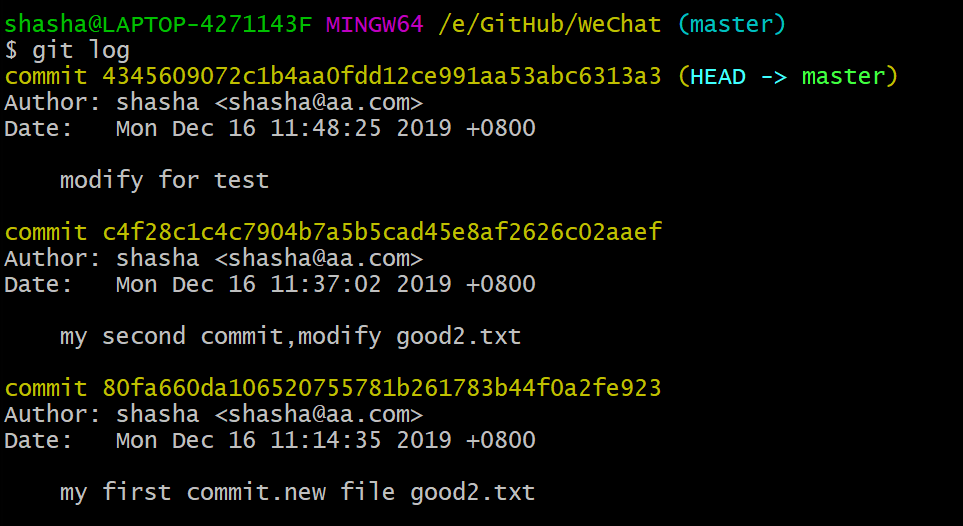


使用 git log,查看历史版本记录，可以看到，有两个版本的历史记录



HEAD是一个指针，指向当前版本

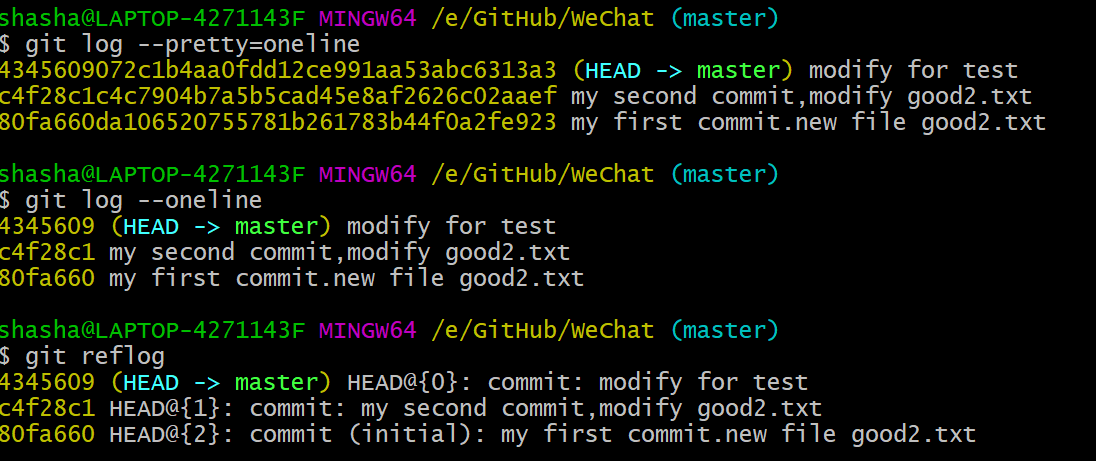
为了更直观的感受，我们再进行一次版本更替



若日志过多时，多屏显示控制方式：  
空格向下翻页  
b 向上翻页  
q 退出

可以使用 git log --pretty=oneline

或者 git log --oneline或者 git reflog



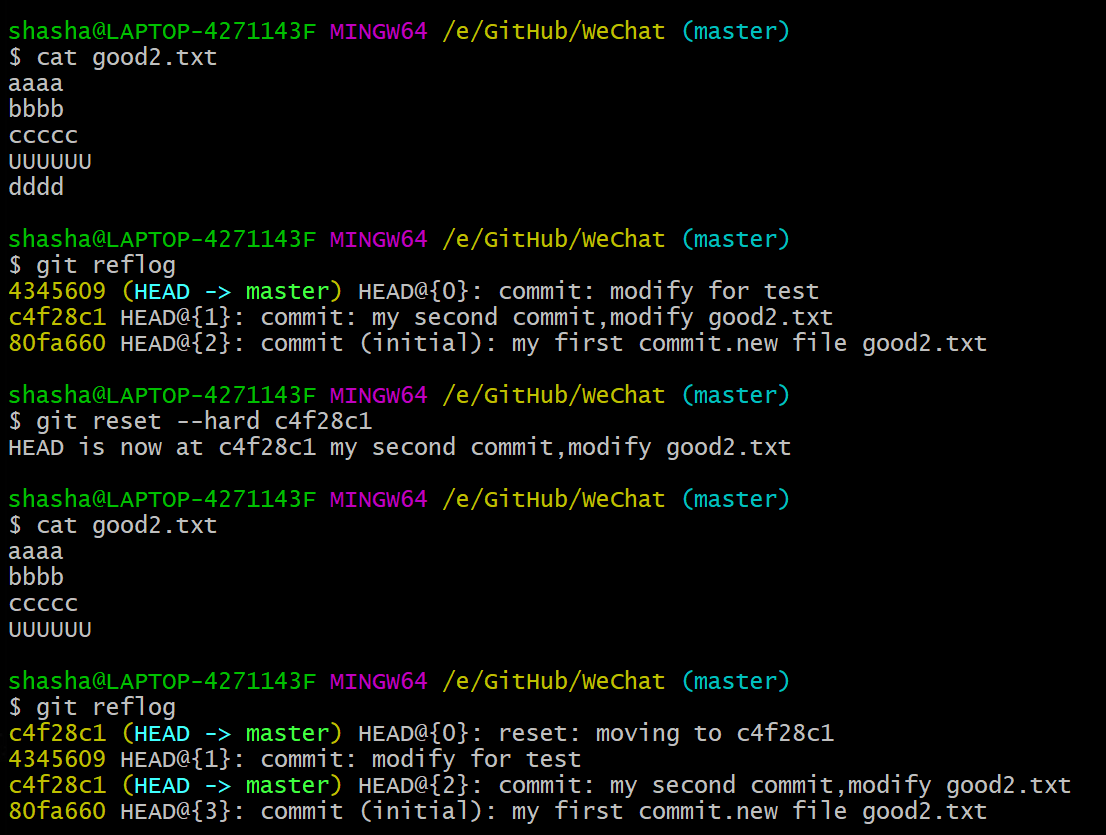
HEAD@{移动到当前版本需要多少步}

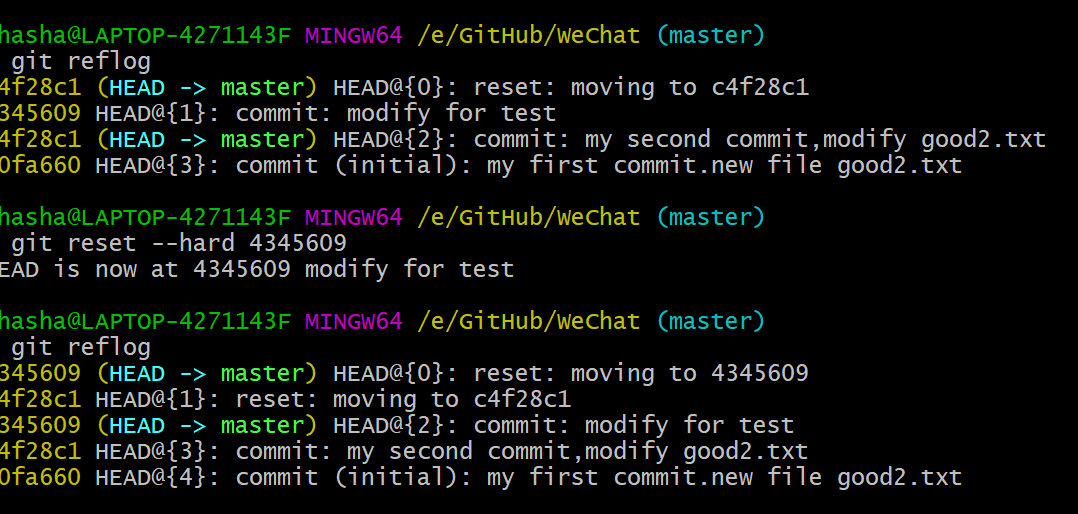
# 版本控制

进行版本控制时，需要先将git reflog 命令调出局部索引值

进行版本控制的三种方法：

1. 基于索引值操作[推荐] 可以进行前进或者后退  
 git reset --hard [局部索引值]

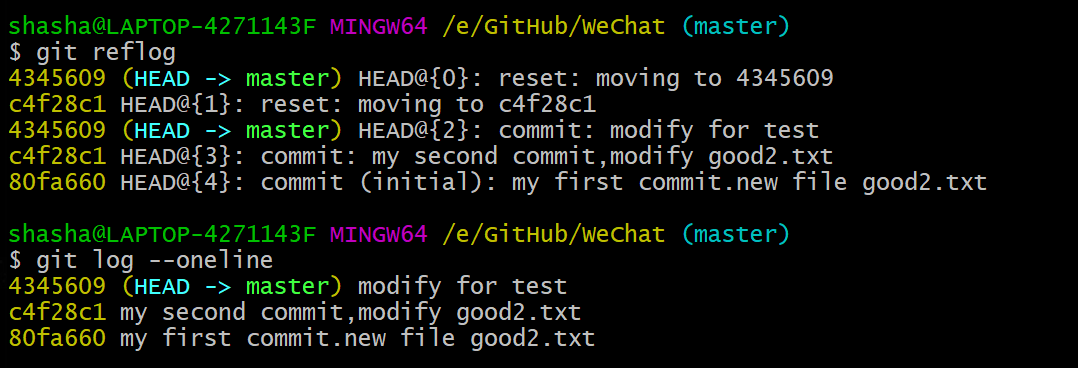


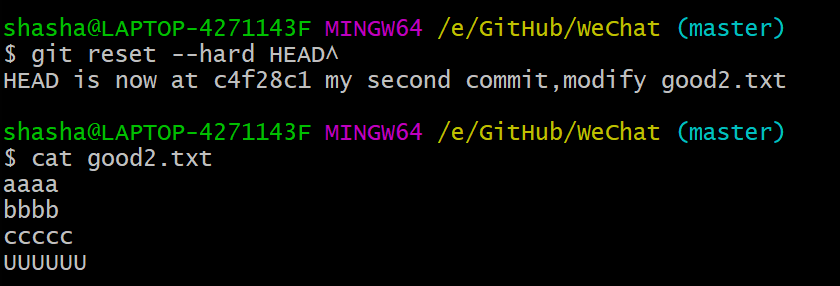


1. 使用^符号, 只能后退  
    git reset --hard HEAD^  
   注： 一个^表示后退一步， n 个表示后退 n 步

这个更适合用git log --oneline来显示版本

我们可以和git reflog 进行比较一下





1. 使用~符号： 只能后退  
    git reset --hard HEAD~n  
   注： 表示后退 n 步

git reset [<模式>] [<提交>]

该表格会将当前分支头重置为，<commit>并可能根据来更新索引（将其重置为的树<commit>）和工作树<mode>。如果<mode>省略，则默认为--mixed。的<mode>必须是下列之一：

-- soft

仅仅在本地库移动HEAD指针，而不会更改暂存区和工作区。完全不触摸索引文件或工作树（而是将头重置为<commit>，就像所有模式一样）。就像git status 这样，这会将所有更改的文件保留为“要提交的更改” 。

使用git reset --soft 命令之后，git reflog命令变化与--hard一致，但是good2.txt文件内容没有变化，使用git status命令可以看到待提交的修改（绿色modify）

--mixed

在本地库移动HEAD指针且会重置暂存区，但不会更改工作区。重置索引，但不重置工作树（即，已更改的文件将保留，但未标记为提交），并报告尚未更新的内容。这是默认操作。

使用git reset --soft 命令之后，git reflog命令变化与--hard一致，但是good2.txt文件内容没有变化，使用git status命令可以看到待上传至暂存区的文件（红色字体，提示git add）

如果-N指定，则将删除的路径标记为要添加的意图（请参见 git-add（1））。

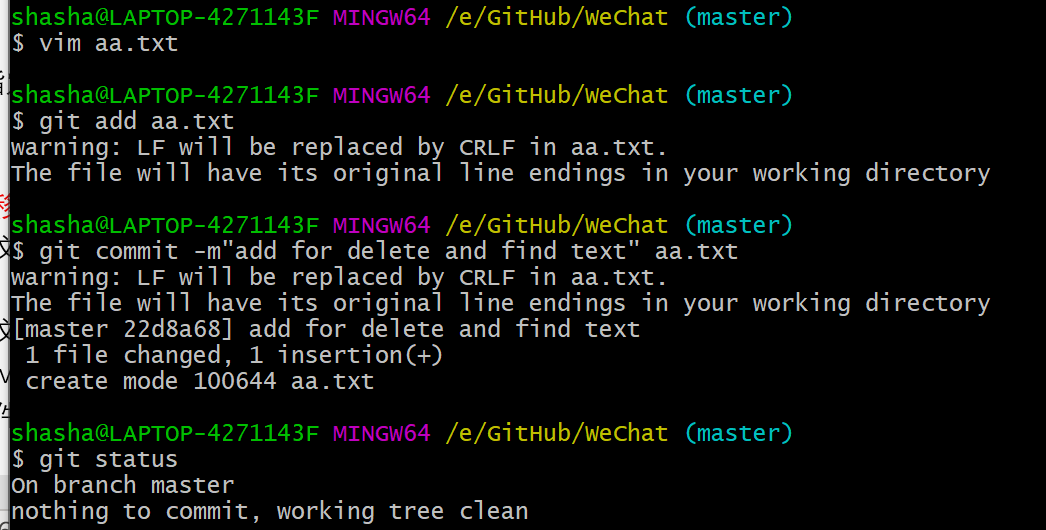
- -hard

在本地库移动HEAD指针，且会重置暂存区和工作区。重置索引和工作树。由于对工作树中跟踪文件的任何更改，<commit>都将被丢弃。

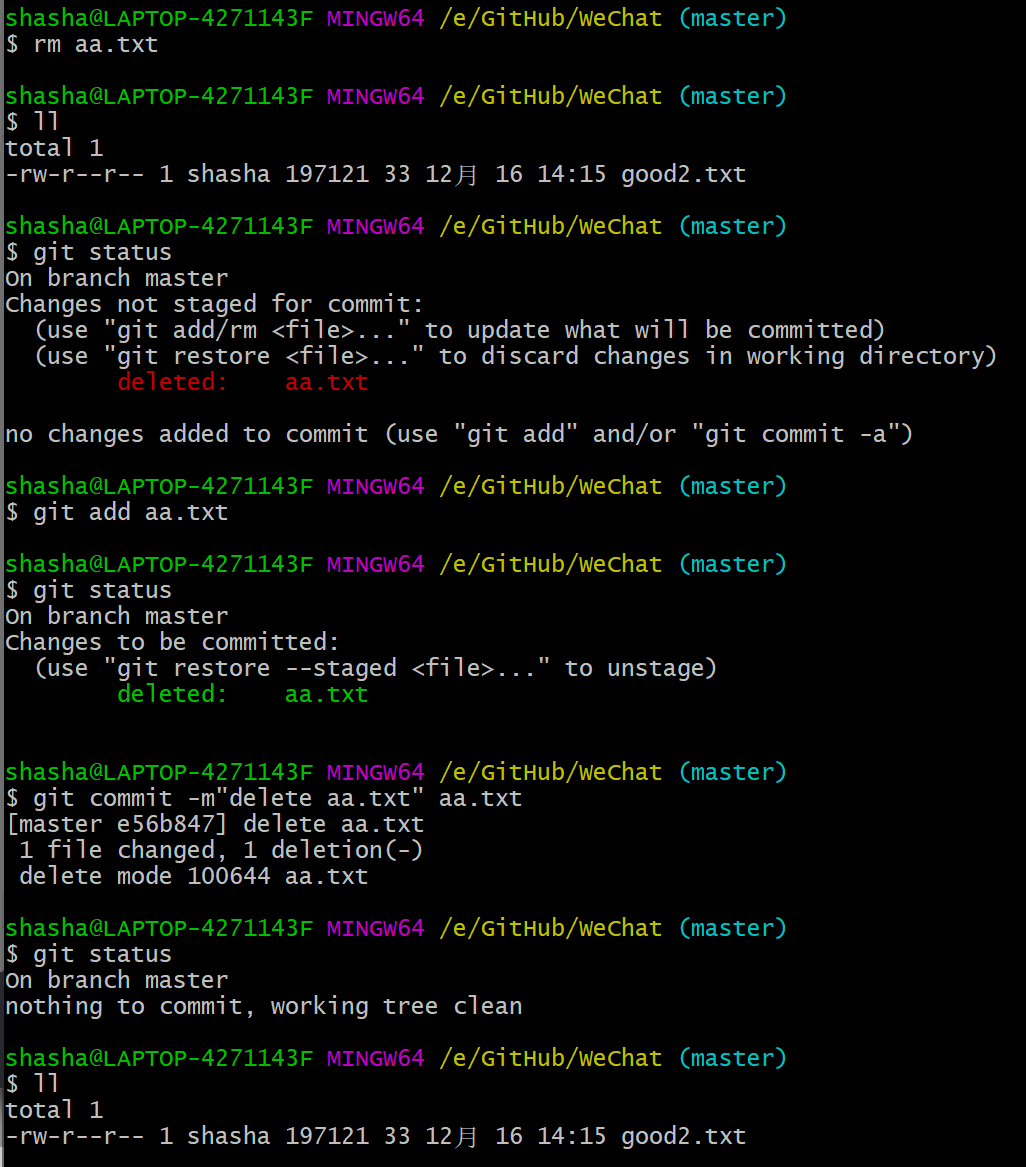
永久删除文件后找回

1. 删除文件——提交删除至本地库（产生了版本更替，可通过版本控制方式找回）

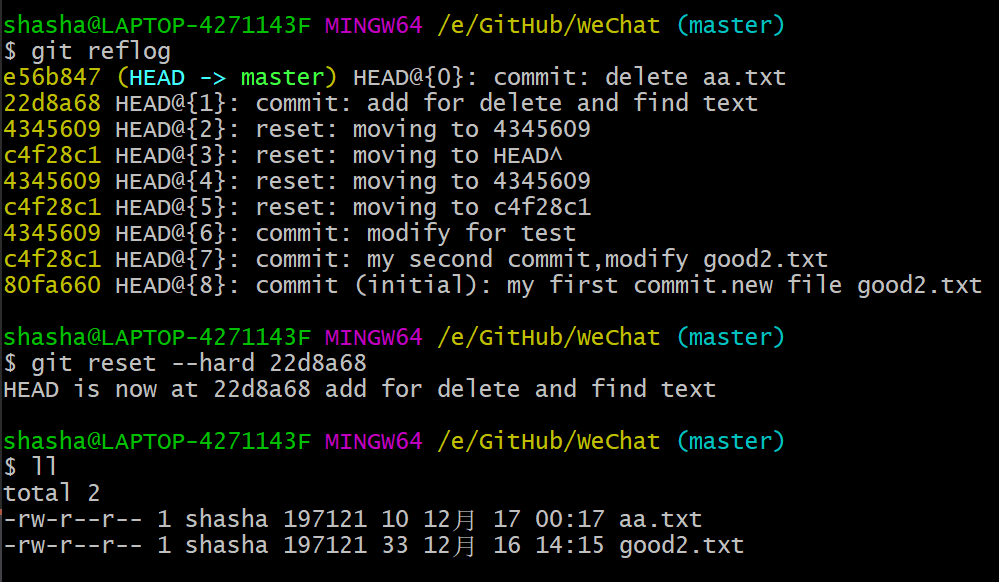
首先使用vim 命令新建一个文件，例如aa.txt,删除该文件通常指删除已经提交至本地库之后的文件



之后，使用rm 文件名 进行删除操作，删除之后，通过ll命令，可以发现本地库已经没有了这个文件，将改动提交



然后，可以通过版本控制方式，可以找会已经删除的文件

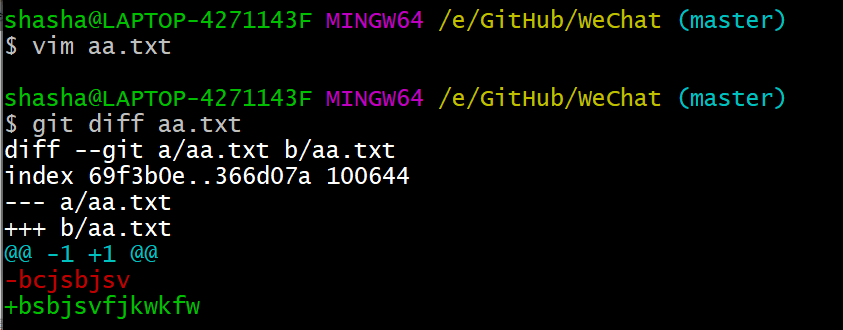


1. 找回文件——删除操作没有提交至本地库,仍然可以通过版本控制方式找回

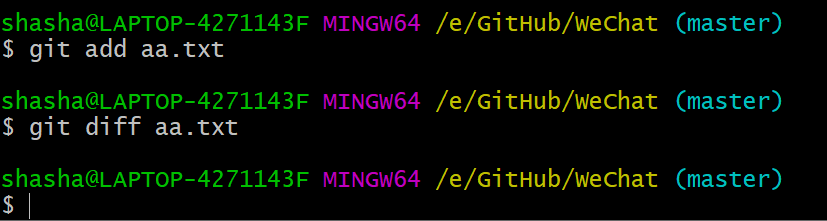
# 比较文件差异

修改aa.txt

git diff查找不同，红色是减少，绿色是增加；，因为git是以行来进行版本控制，所以相当于删掉原有行，又新增了一行

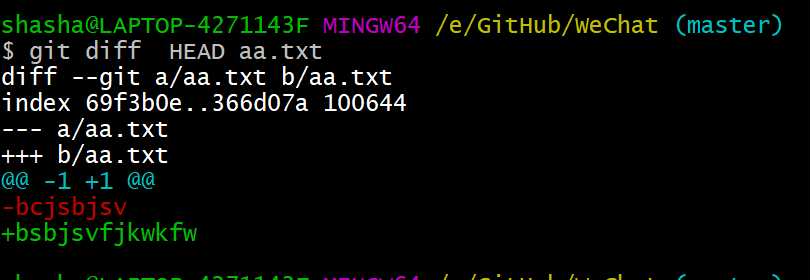


如果将改动提交至暂存区，再使用git diff命令，会发现，没有区别；说明，git diff不附加参数的话，是将工作区文件与暂存区中文件进行比较，不带文件名 是比较多个文件



若是增加HEAD参数，表明比较文件是本地库中的文件，此时，可以看到，又出现了差异

还可以和本地库中某一历史版本进行比较，如git diff HEAD^ aa.txt

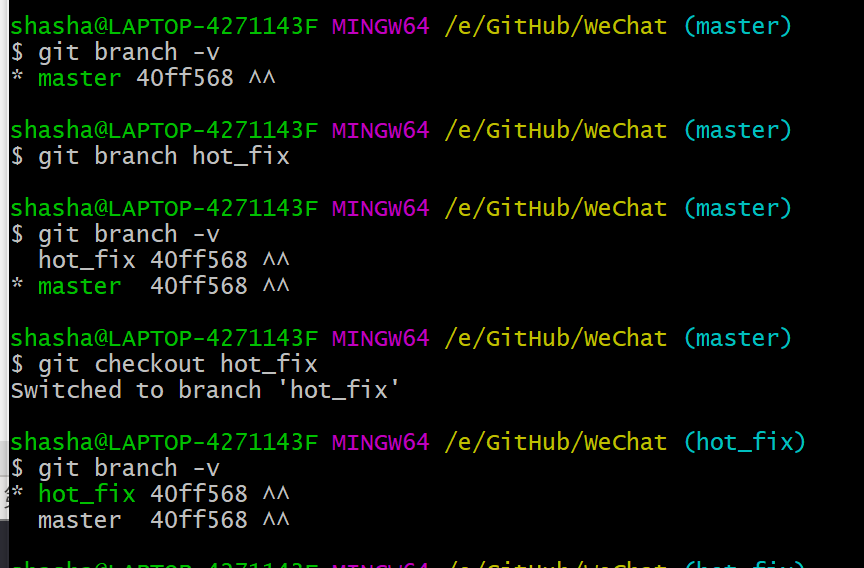


# 分支管理

查看分支 git branch -v

创建分支 git branch 分支名

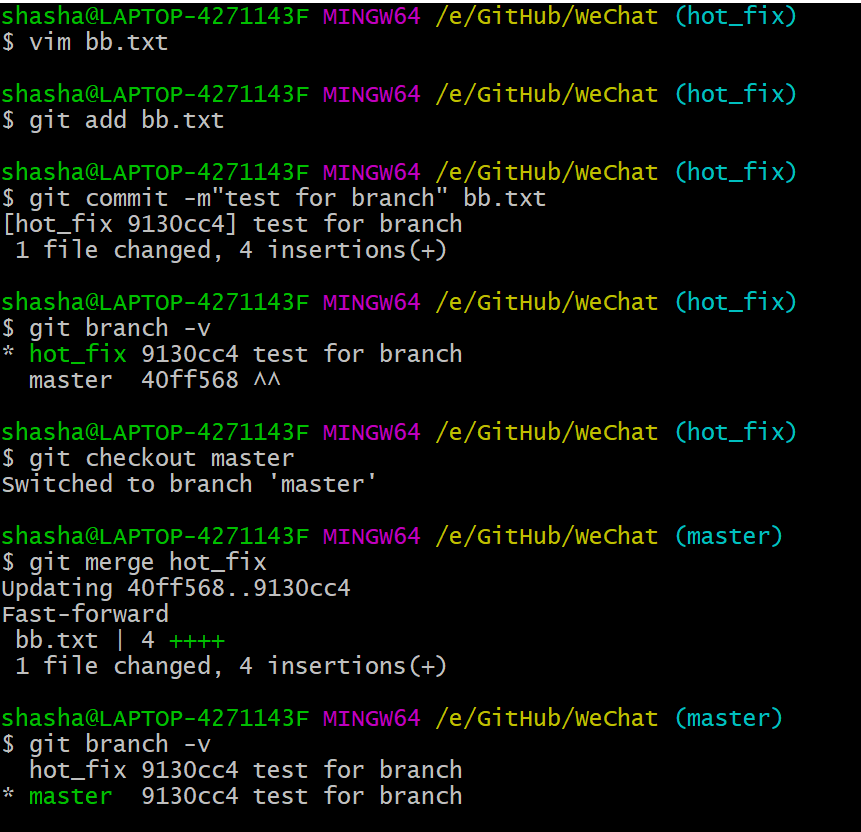
切换分支 git checkout 分支名



分支合并：1.切换至接受合并的分支 2.git merge [分支名]

例如，在进行创建分支之后，hot\_fix分支一开始同master版本一致，首先，更新hot\_fix

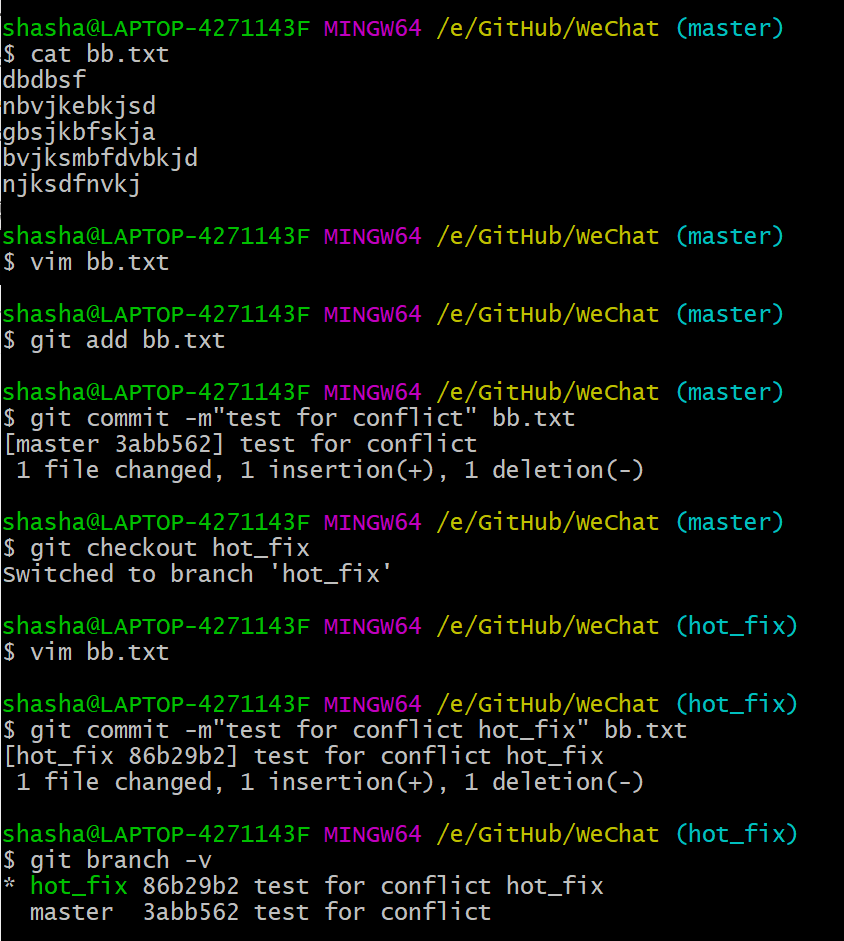
接下来，切换至master，执行git merge 命令合并两个分支，可以看到master 和hot\_fix版本一致



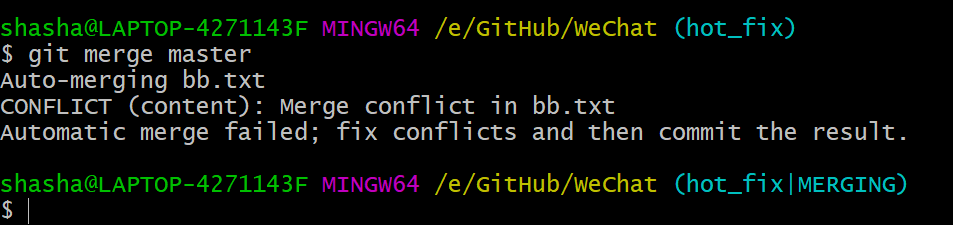
# 分支合并时解决冲突

冲突的原因：两个分支同时对某一文件进行修改再合并

首先在master分支对bb.txt（例如第三行）进行修改，提交；切换至hot\_fix 分支对同一 文件同一行进行不同的修改



接下来，将master 合并到hot\_fix分支，可以看见冲突产生，自动合并失败，需要进行手动合并，且分支名不再是hot\_fix,而是hot\_fix|MERGING 表明处于冲突合并的过程中

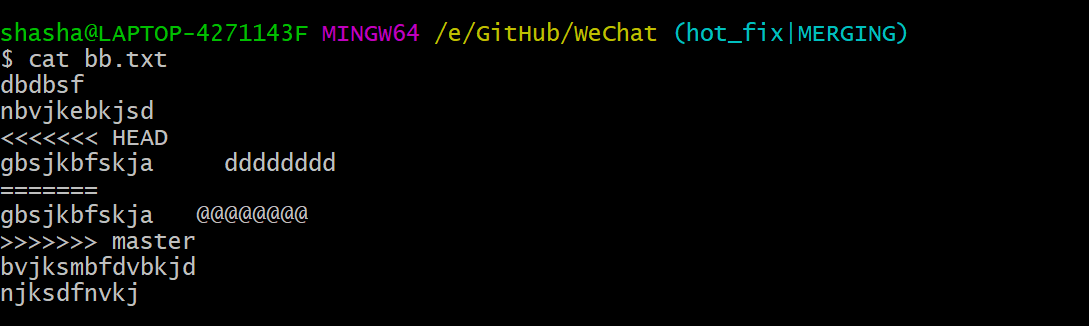


冲突表现如下：

<<<<<<<HEAD 指当前分支内容

======= 分隔

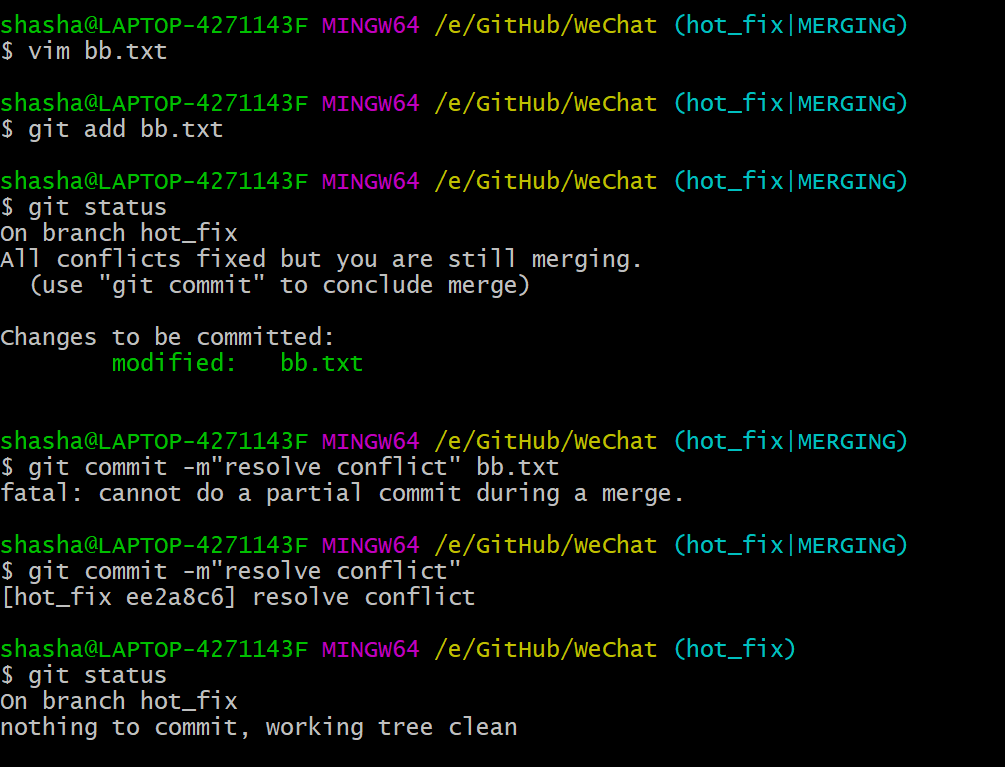
>>>>>>> master master分支里面的内容



冲突解决方式：1.编辑文件，删除特殊符号并修改文件至满意;2.git add [文件名]；3. git commit -m “XX”

可以使用vim 命令对冲突部分进行修改，并提交修改，结束冲突；注意分支名称的变化

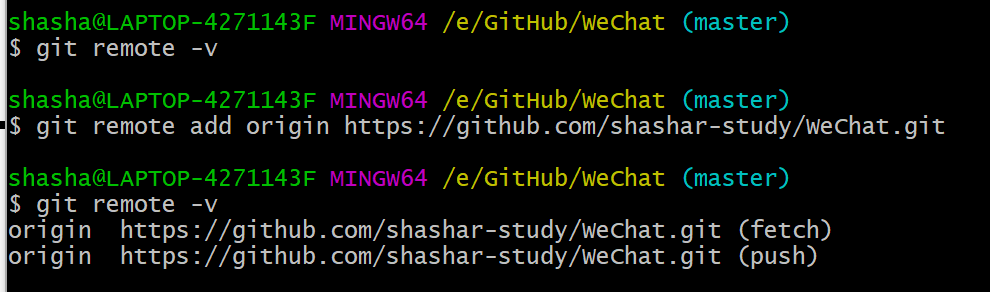
可以看到，git add 使用方式不变，但使用后git status会提示冲突仍未解决，需要进行commit 操作，但是，此时git commit命令不能加冲突文件名，否则会报错。应该直接使用git commit -m”XXXX”进行提交，之后冲突解决，分支名重新变成hot\_fix



# 将本地库pull到远程库

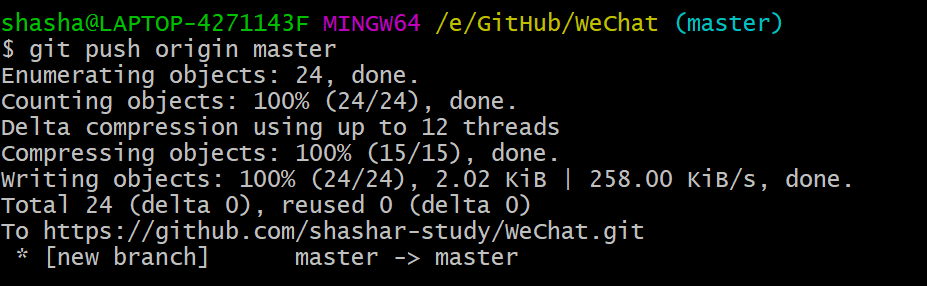
<https://github.com/shashar-study/WeChat.git>

1. git remote add [别名] [仓库的HTTPS地址],将远程仓库地址起别名为origin，这里，fetch指取回，push指推送



1. git push [刚起的别名] [分支名]

从本地master到远程库master；对于远程库来说，新的空仓库里有了新的master分支

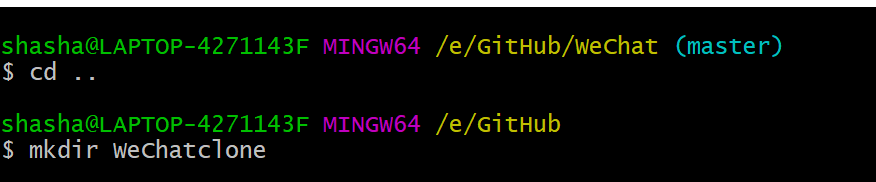


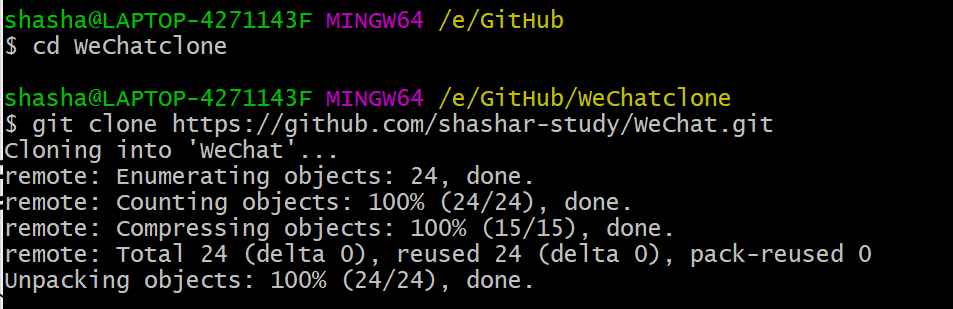
此时，刷新github中刚建好的仓库，可以看见已经完成push



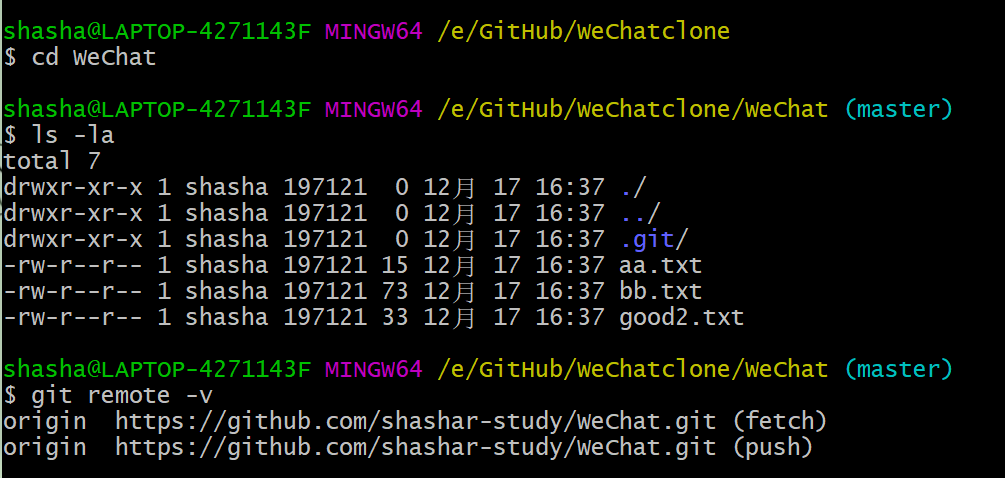
# 从远程库clone至本地库

新建一个文件夹WeChatclone，cd到该目录之下，git clone [远程仓库HTTPS地址]





此时远程库已经克隆完毕，可以看到，目录之下存在.git文件，这意味着我们不需要自己进行初始化操作，即git init;并且，之前进行的操作，即origin别名也一起复制下来了



克隆命令效果:完整下载本地库；创建origin远程地址别名；初始化本地库

# 邀请加入团队成员

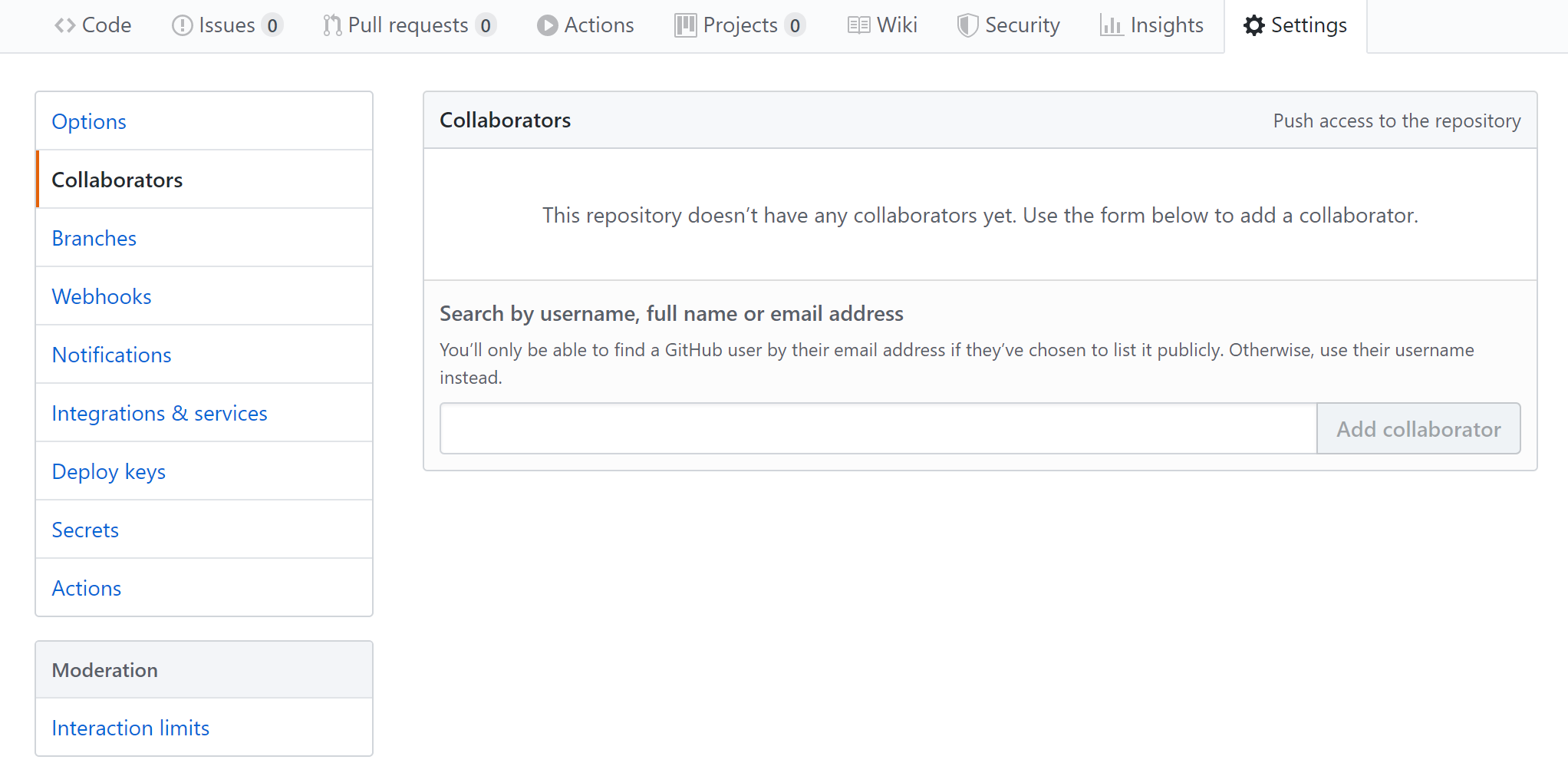
在WeChatclone中，可以将 修改提交至本地库，但无法讲修改提交至远程库，因为没有权限。

非团队成员可以进行的操作：git add git commit

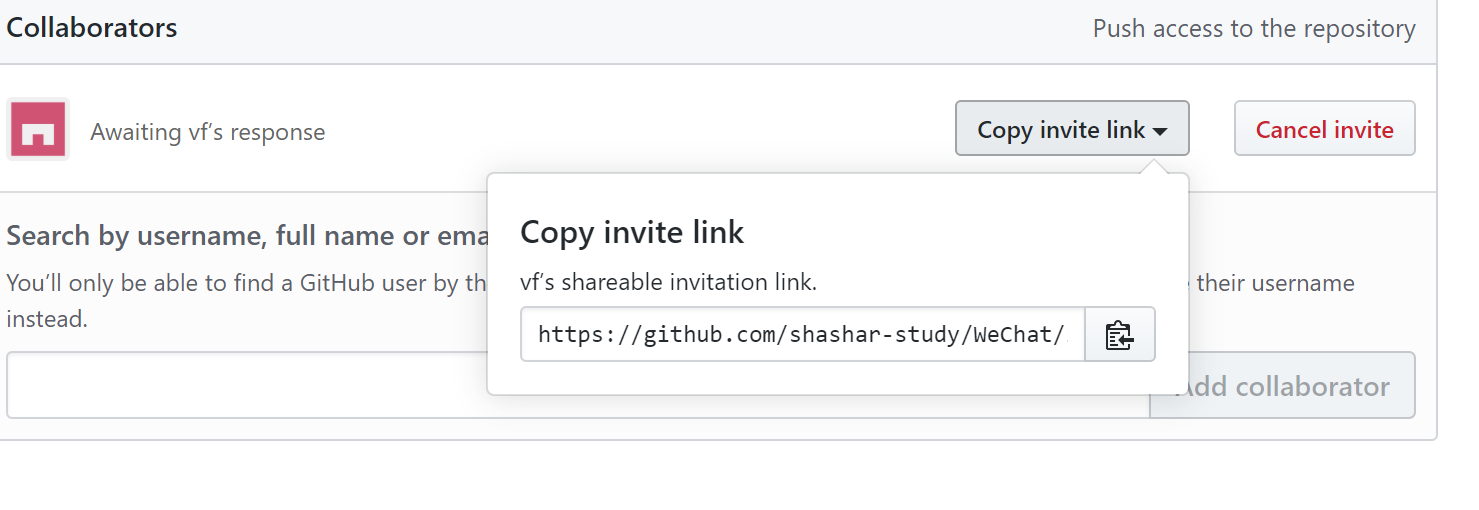
不可以进行的操作：git push [远程地址别名] [分支名]

团队成员可以将修改push至远程库 ，可以进行 git push [远程地址别名] [分支名]（例如 git push origin master）

邀请新成员方式：github网络端登陆，找到仓库：settings-Collaborators-输入被邀请人的github账号



填写之后进入，可以将图链接发送给被邀请人，对方打开链接接受即可



在本地进行这项功能测试时，由于网页密码自动登录 ，无法呈现出多个账户费团队成员的情况，如果想要切换账号，可以在win10控制面板-所有控制面板-凭据管理器-Windows凭据，把普通凭据中的github凭据删除即可

pull=fetch+merge

git fetch origin master 该命令将远程仓库master分支的内容抓取，是一个只读操作，不会更改本地库里面的内容，如果想查看抓取的内容，需要切换分支，命令为git checkout origin/master,然后可以查看；如果要合并，需要使用git merge origin/master 命令

当操作比较复杂的时候，可以使用fetch和merge.

git fetch [远程地址别名][远程分支名]

git merge [远程仓库别名]/[远程分支名]

# 解决同时开发遇到的冲突

两人同时 对同一位置文件进行修改，然后进行提交，两者都可以正确提交至本地库，但是，只有先提交至远程仓库的人可以正确提交，下一个人必须先git pull远程仓之后再提交

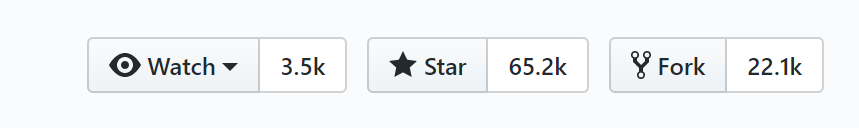
解决冲突的方法与前文解决各分支之间冲突解决方法一样

只允许基于github远程仓库最新版本下修改

# 跨团队合作

步骤：团队外开发者fork 远程仓--合作工作完成之后pull request--原远程仓创建者审通过之后merge合并

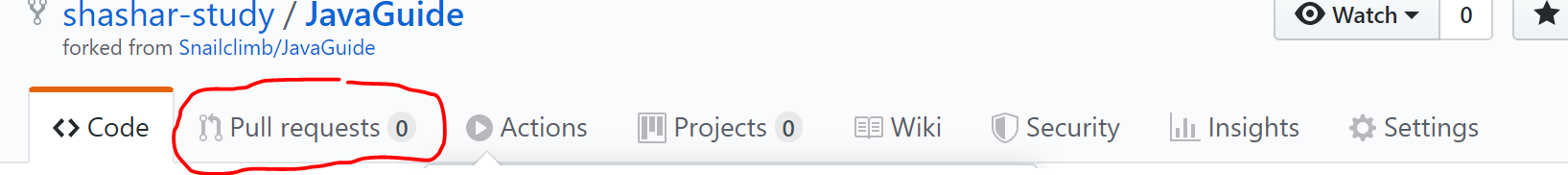
将团队远程仓库（A远程仓 A团队）的HTTP地址发送给团队外成员（B远程仓 B团队）；B登陆github并复制地址找到A仓库，然后点击fork,将远程库复制一份变成自己的远程库

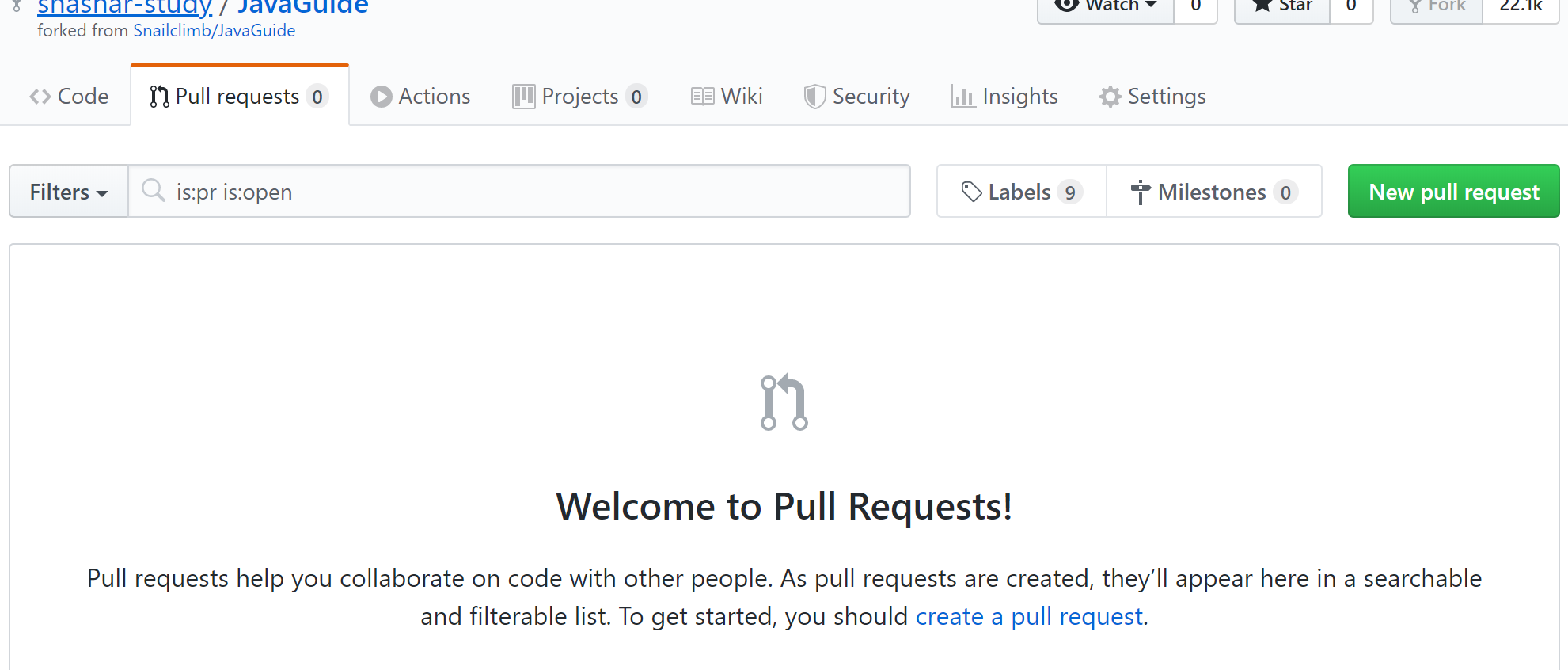


说明了fork来源，但是现在已经是自己的远程仓库了



B对fork来的远程仓库B commit 修改，提交给A，使用pull request ,pull request边角上的数字表示接到拉取请求的个数





A团队收到pull request请求，点击，查看，及审核，过程中A和B可以进行交流。A确认过没问题之后，通过merge合并 ，远程库发生改动，然后将变动pull到本地库即可。

# SSH免密登陆

仓库有两种地址，HTTPS和 SSH（前者需要登陆账号及密码，后者不需要）

* 进入当前用户的家目录  
  $ cd ~
* 删除.ssh 目录

$ rm -rvf .ssh

* 运行命令生成.ssh 密钥目录  
  $ ssh-keygen -t rsa -C atguigu2018ybuq@aliyun.com  
  [注意： 这里**-C** 这个参数是大写的 **C**]

生成两个文件，如下，前者为私钥，后者公钥



* 进入.ssh 目录查看文件列表  
  $ cd .ssh  
  $ ls -lF
* 查看 id\_rsa.pub 文件内容

$ cat id\_rsa.pub

* 复制 id\_rsa.pub 文件内容， 登录 GitHub， 点击用户头像→Settings→SSH and GPG keys  
  New SSH Key
* 输入复制的密钥信息，添加该密钥
* 回到 Git bash 创建远程地址（这时用的是仓库的ssh地址）别名为origin\_ssh  
  git remote add origin\_ssh [git@github.com:atguigu2018ybuq/huashan.git](mailto:git@github.com:atguigu2018ybuq/huashan.git)
* 推送文件进行测试 git push origin\_ssh master

部分操作过程截图，推送文件时，同之前用HTTPS远程地址时一样的操作，只是无需登录

