# Git

## Git的概念

【1】Git技术：公司必备，一定要会

【2】Git概念：

Git是一个免费的、开源的分布式版本控制系统，可以快速高效地处理从小型到大型的项目。

【3】什么是版本控制？

版本控制是一种记录一个或若干文件内容变化，以便将来查阅特定版本修订情况的系统 。

【4】为什么要使用版本控制?

软件开发中采用版本控制系统是个明智的选择。

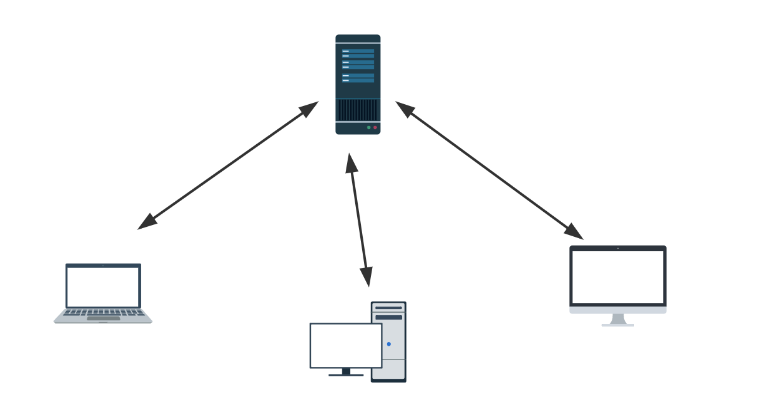
有了它你就可以将某个文件回溯到之前的状态,甚至将整个项目都回退到过去某个时间点的状态。

就算你乱来一气把整个项目中的文件改的改删的删，你也照样可以轻松恢复到原先的样子。

但额外增加的工作量却微乎其微。你可以比较文件的变化细节,查出最后是谁修改了哪个地方,从而找出导致怪异问题出现的原因，又是谁在何时报告了某个功能缺陷等等。

【5】版本控制系统的分类：

❀集中化的版本控制系统：



集中化的版本控制系统诸如CVS, SVN 以及Perforce 等，都有一个单一的集中管理的服务器,保存所有文件的修订版本，而协同工作的人们都通过客

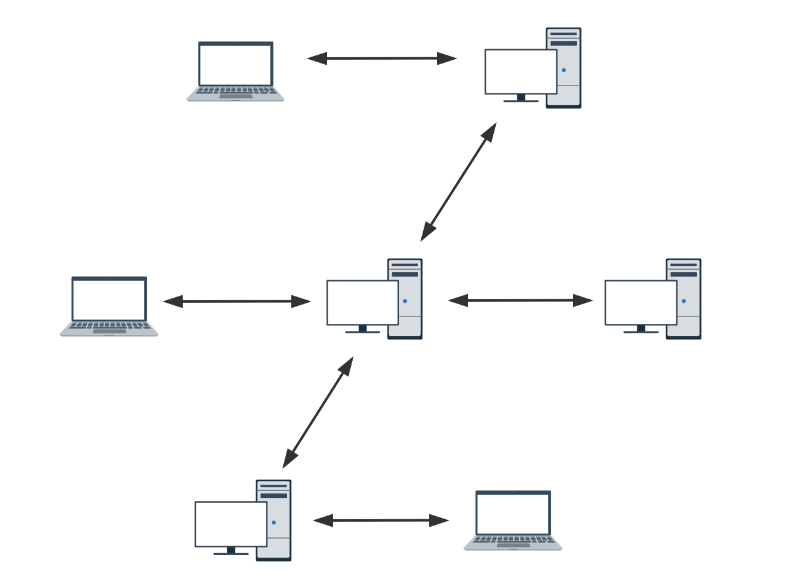
户端连到这台服务器,取出最新的文件或者提交更新。多年以来,这已成为版本控制系统的标准做法,这种做法带来了许多好处,现在,每个人都可以在一定程度上看到项目中的其他人正在做些什么。而管理员也可以轻松掌控每个开发者的权限，并且管理一个集中化的版本控制系统;要远比在各个客户端上维护本地数据库来得轻松容易。

事分两面，有好有坏。这么做最显而易见的缺点是中央服务器的单点故障。如果服务器宕机一小时，那么在这一小时内， 谁都无法提交更新，也就无法协同工作。

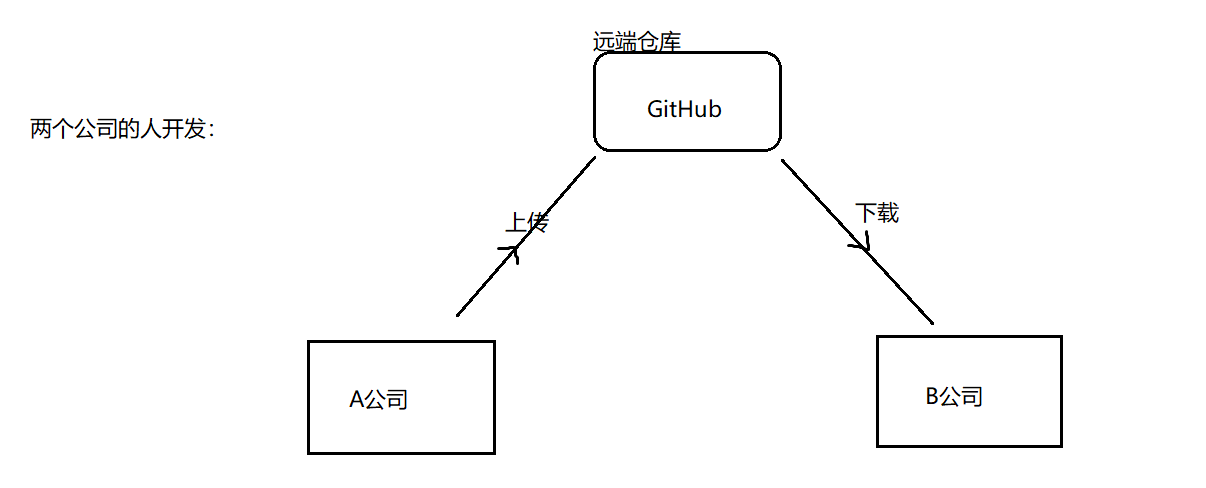
❀分布式的版本控制系统

由于上面集中化版本控制系统的那些缺点，于是分布式版本控制系统面世了。

在这类系统中，像Git, BitKeeper 等,客户端并不只提取最新版本的文件快照，而是把代码仓库完整地镜像下来。

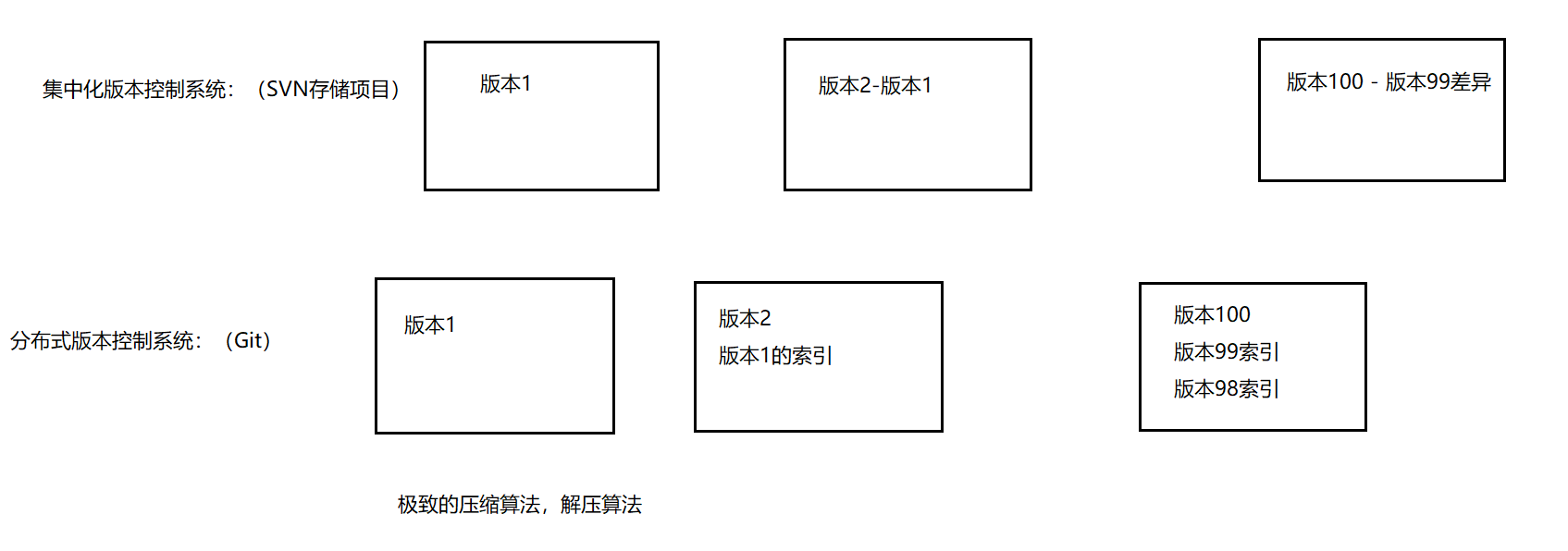


更进一步，许多这类系统都可以指定和若干不同的远端代码仓库进行交互。这样，你就可以在同一个项目中分别和不同工作小组的人相互协作。



分布式的版本控制系统在管理项目时存放的不是项目版本与版本之间的差异.它存的是索引(所需磁盘空间很少所以每个客户端都可以放下整个

项目的历史记录)



## Git简史



Linux  --->  人越来越多 ---->代码优化的越来越好 ---》项目管理工具

---》Bitkeeper ---》终止合作 ---》一周Git --》一个月内将Linux管理在Git上---》开源，免费  ---》用户群大

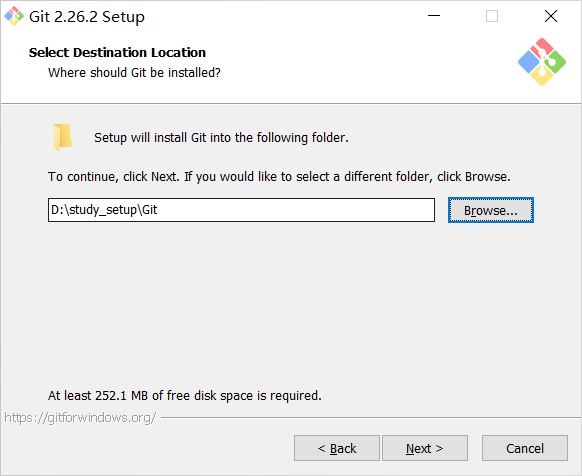
## Git的安装过程

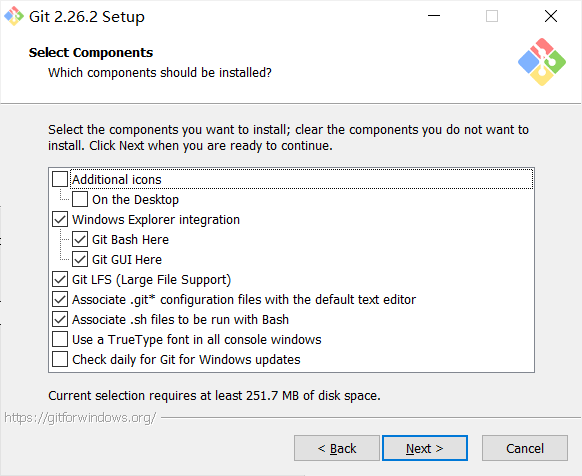
【1】Git官网：

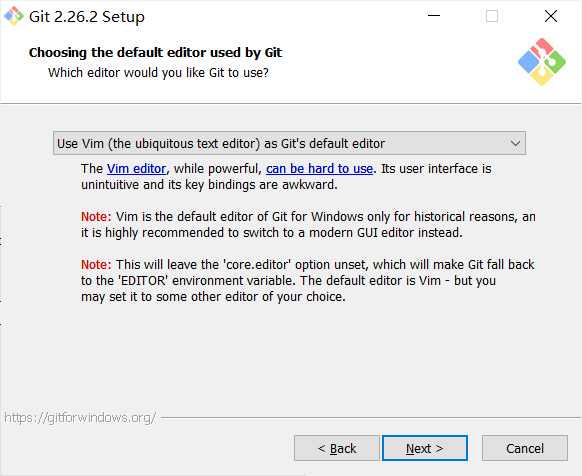
<https://git-scm.com/>

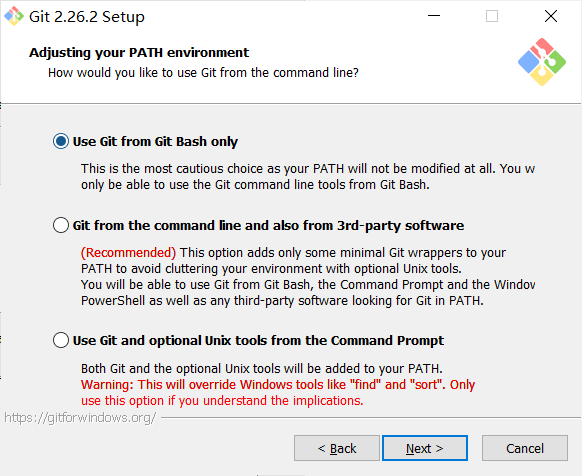
【2】安装过程：

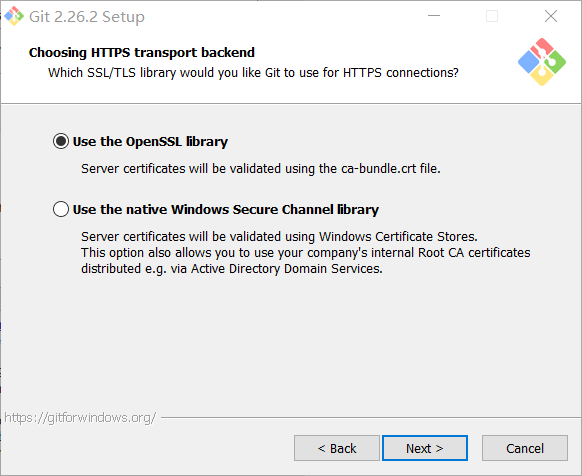
一直下一步

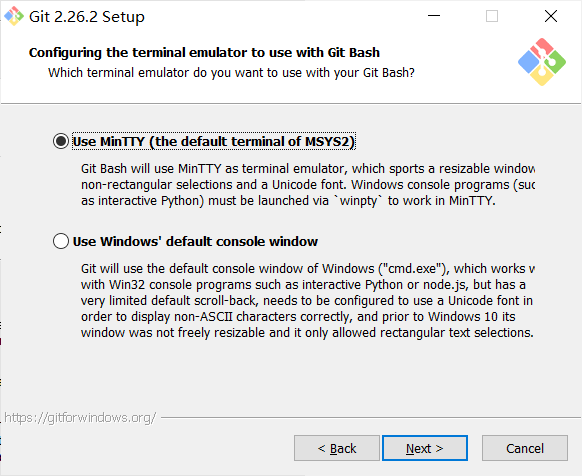




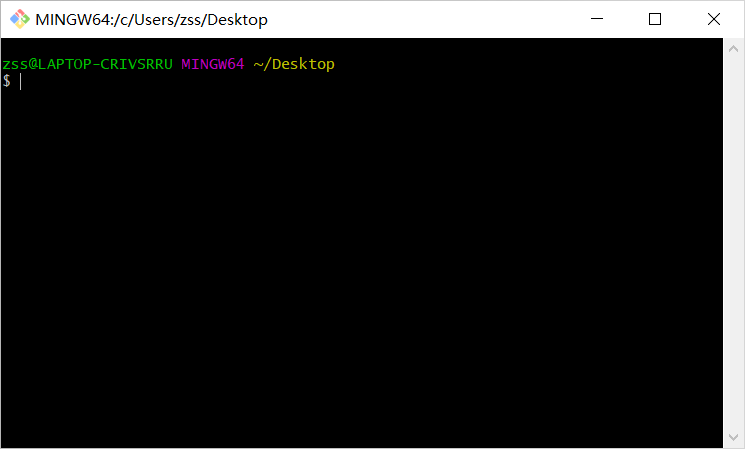




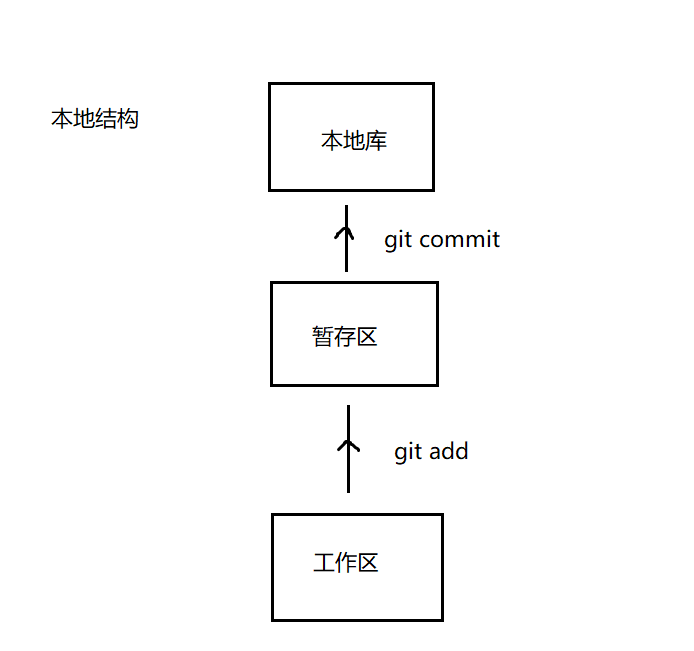




点击Git Bash Here打开Git终端：



## Git结构



## 代码托管中心\_本地库和远程库的交互方式

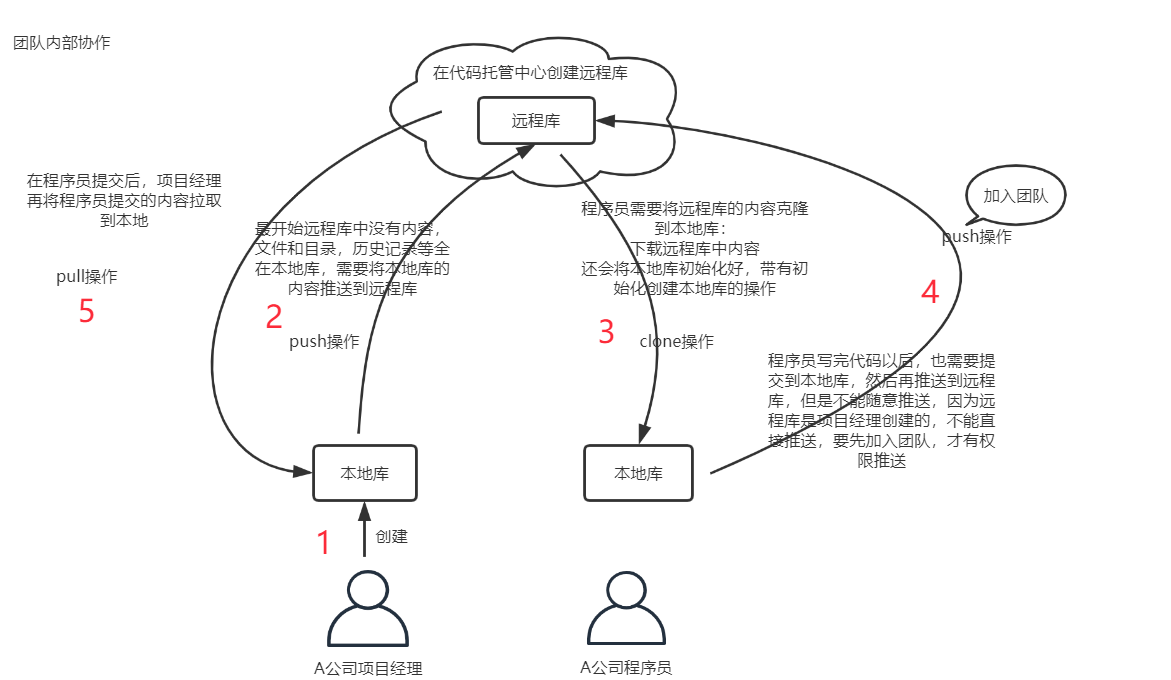
【1】代码托管中心是干嘛的呢？

我们已经有了本地库，本地库可以帮我们进行版本控制，为什么还需要代码托管中心呢？

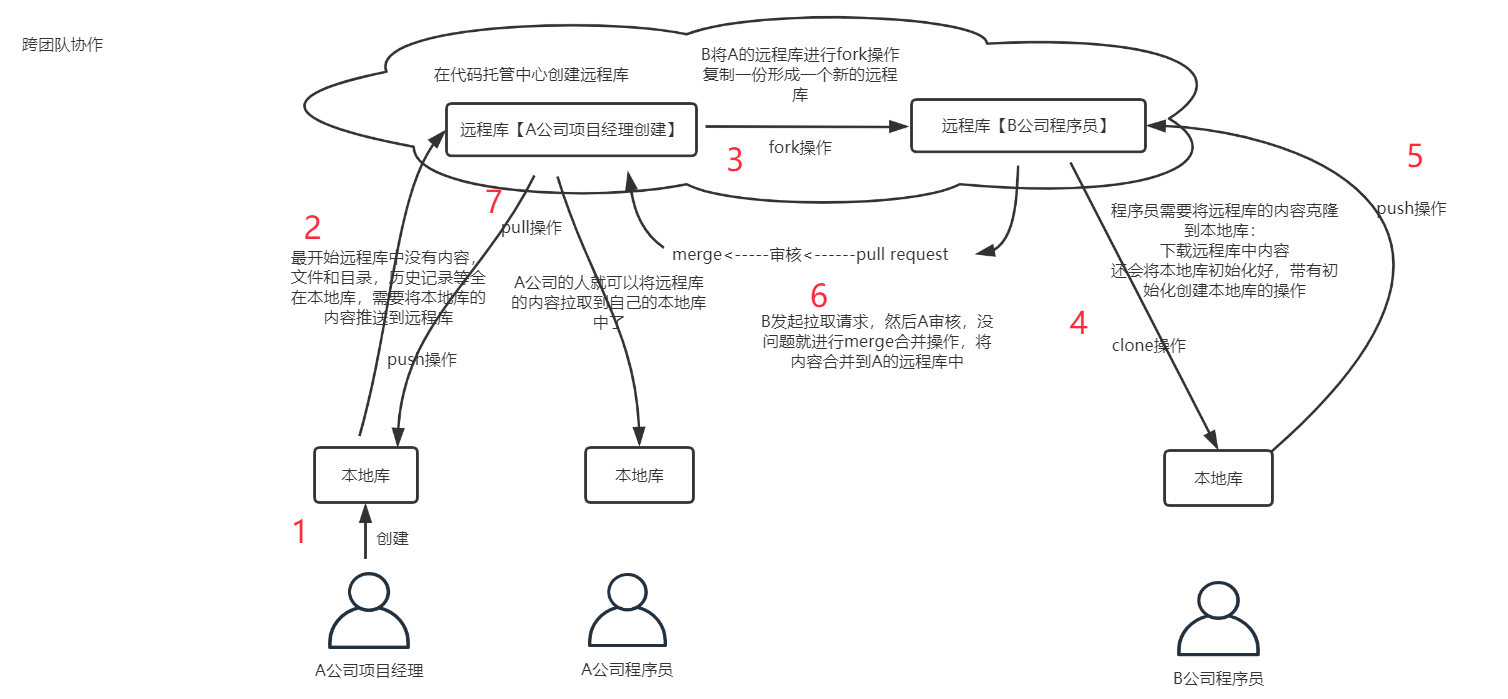
它的任务是帮我们维护远程库，

下面说一下本地库和远程库的交互方式，也分为两种：

（1）团队内部协作



（2）跨团队协作



【2】托管中心种类：

局域网环境下：  可以搭建 GitLab服务器作为代码托管中心，GitLab可以自己去搭建

外网环境下：可以由GitHub或者Gitee作为代码托管中心，GitHub或者Gitee是现成的托管中心，不用自己去搭建

## 初始化本地仓库

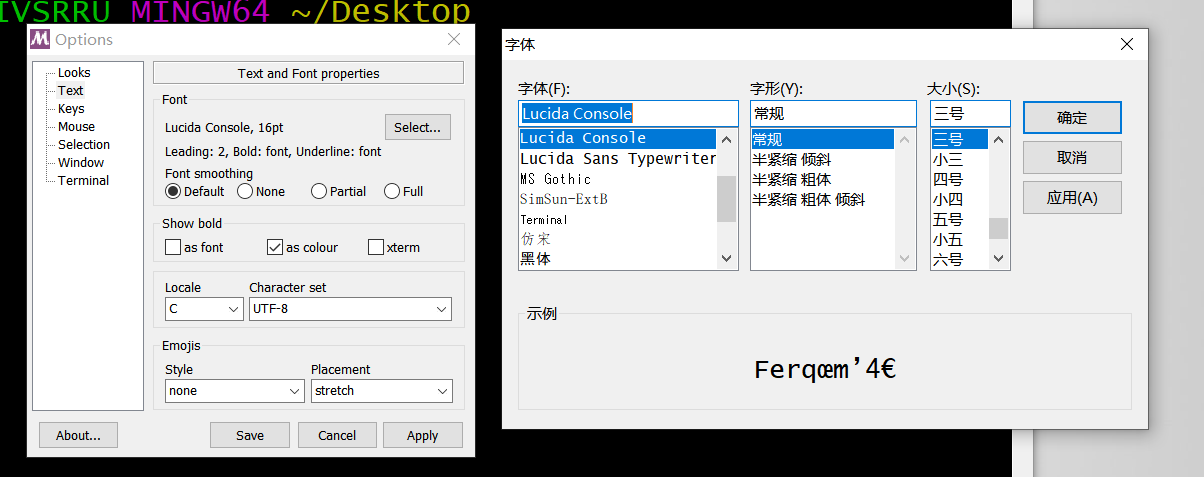
【1】创建一个文件夹：

IMG_256

【2】打开Git终端：

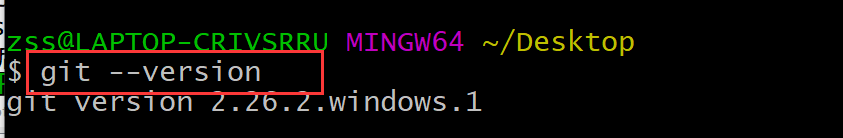
Git Bash Here:

进入以后先对字体和编码进行设置：

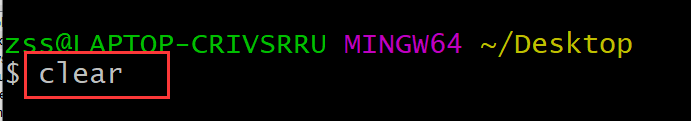


在Git中命令跟Linux是一样的：

（1）查看git安装版本：

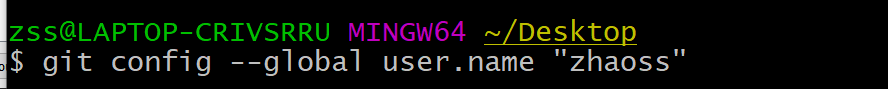


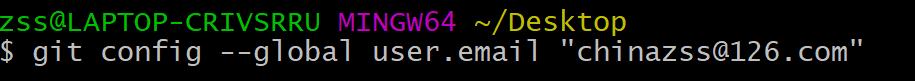
（2）清屏：



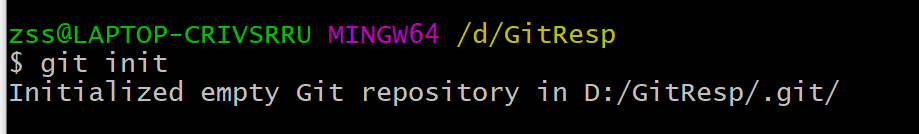
（3）设置签名：

设置用户名和邮箱：





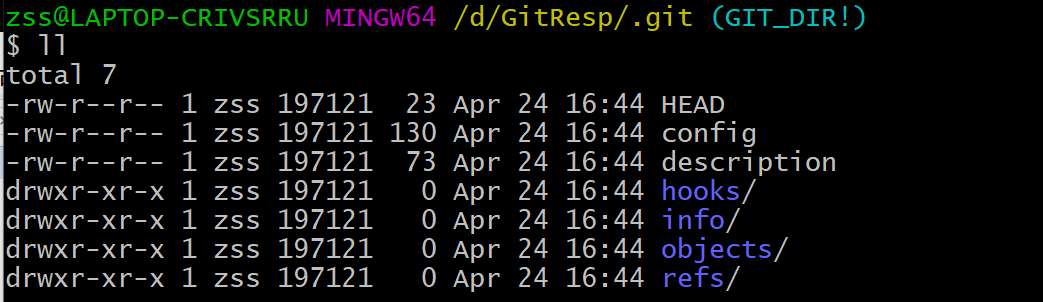
（4）本地仓库的初始化操作：



.git目录是隐藏的：可以调出来查看：



查看.git下内容：



注意事项： .git目录下的本地库相关的子目录和子文件不要删除，不要胡乱修改。

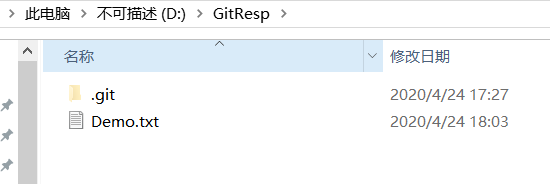
## Git常用命令

### add和commit命令

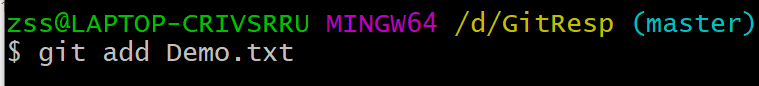
添加文件： add   提交文件：commit

展示：

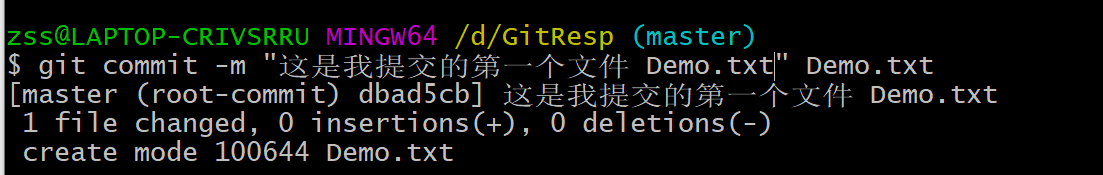
【1】先创建一个文件：



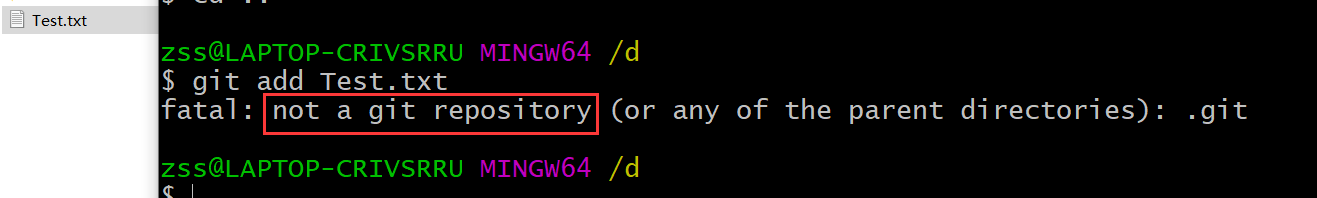
【2】将文件提交到暂存区：



【3】将暂存区的内容提交到本地库：



注意事项：



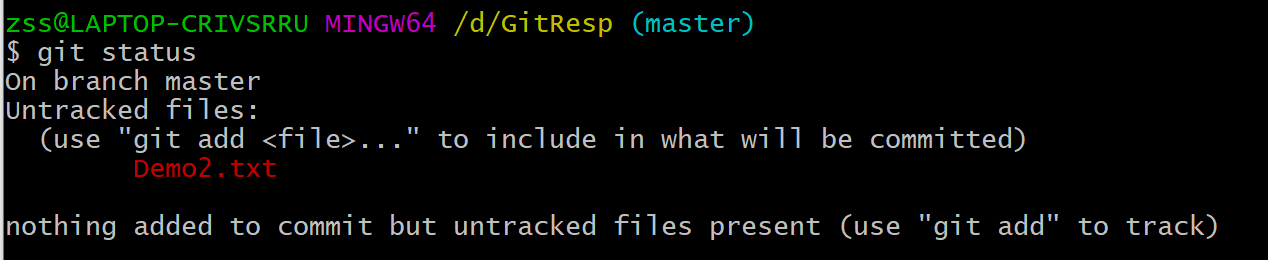
（1）不放在本地仓库中的文件，git是不进行管理

（2）即使放在本地仓库的文件，git也不管理，必须通过add,commit命令操作才可以将内容提交到本地库。

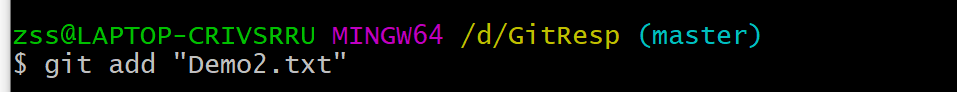
### status命令

git status看的是工作区和暂存区的状态

创建一个文件，然后查看状态：

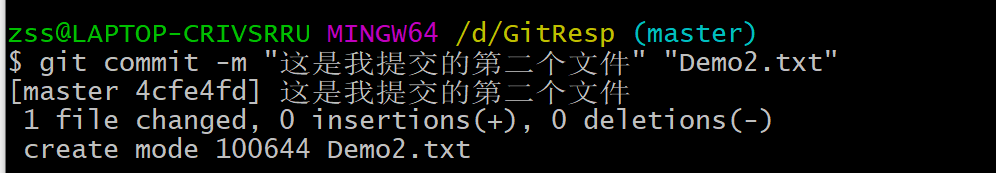


然后将Demo2.txt通过git add命令提交至：暂存区：

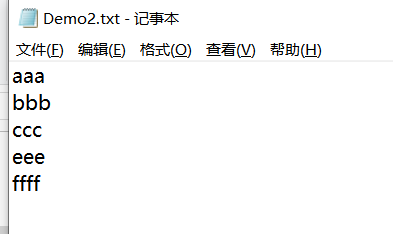


查看状态：

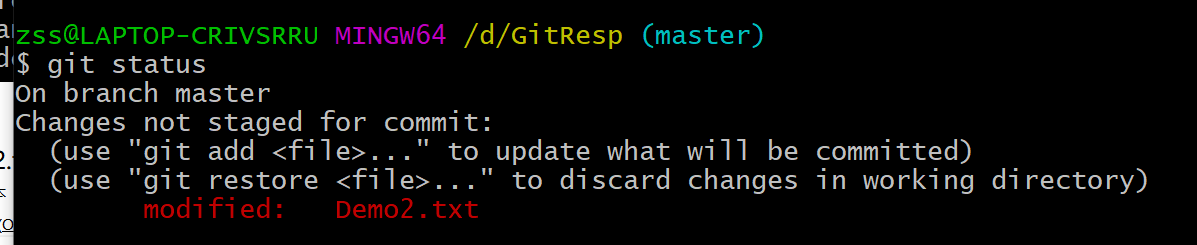


利用git commit 命令将文件提交至：本地库  


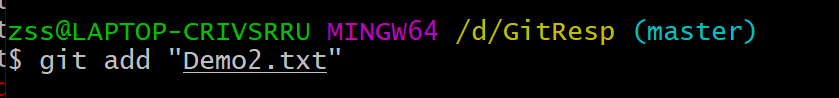
现在修改Demo2.txt文件中内容：

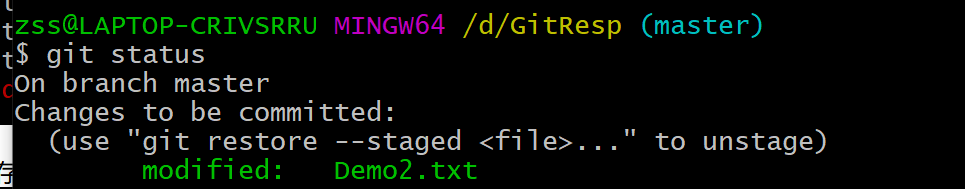


然后再查看状态：

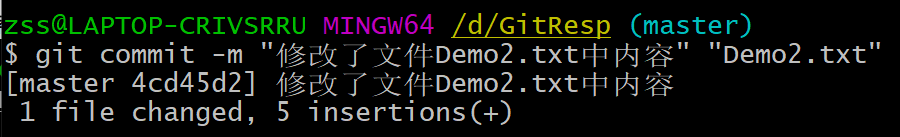


重新添加至：暂存区：

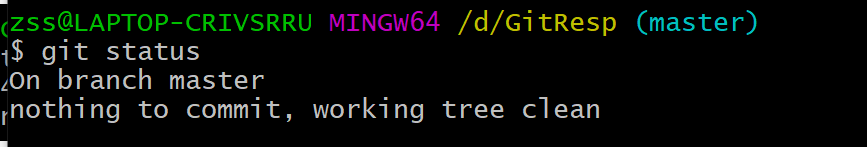




然后将暂存区的文件提交到本地库：



提交完再查看状态：



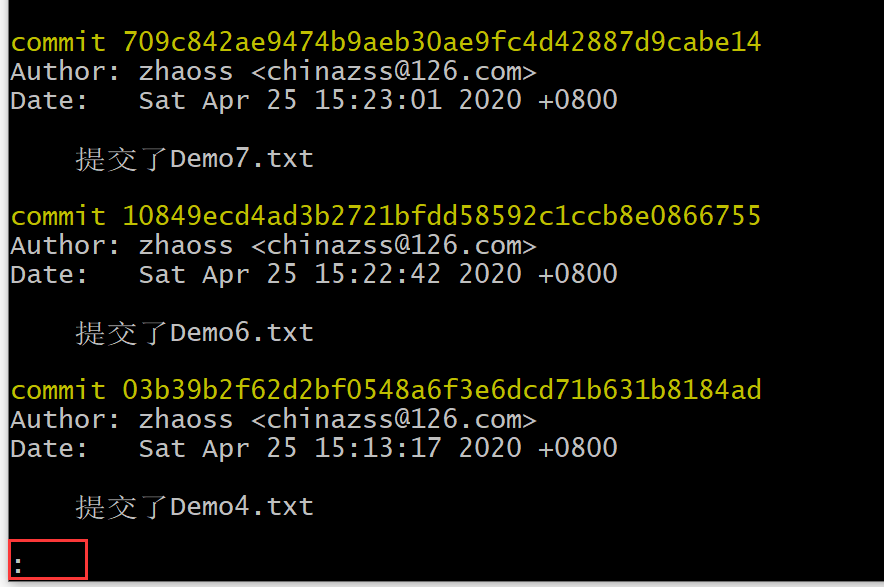
### log命令

git log 可以让我们查看提交的，显示从最近到最远的日志



### log命令2

当历史记录过多的时候，查看日志的时候，有分页效果，分屏效果，一页展示不下：



下一页：空格

上一页： b

到尾页了 ，显示END

IMG_257

退出：q

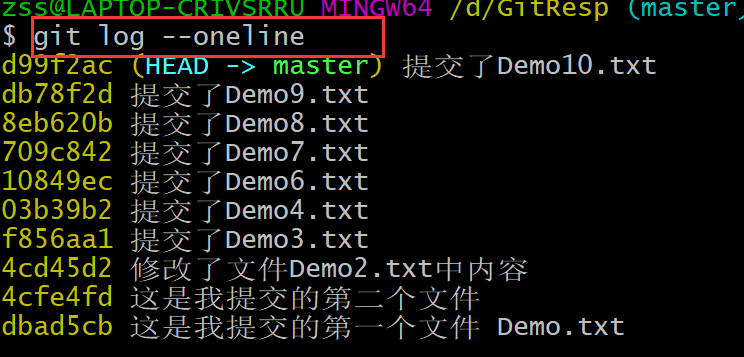
日志展示方式：

【1】方式1：git log  ---》分页

【2】方式2：git log --pretty=oneline



【3】方式3：git --oneline



【4】方式4：git reflog

多了信息：HEAD@{数字}

这个数字的含义：指针回到当前这个历史版本需要走多少步

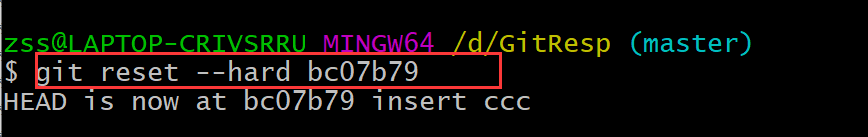


### reset命令

reset命令：前进或者后退历史版本

复制：在终端中选中就是复制了

粘贴： 右键：paste

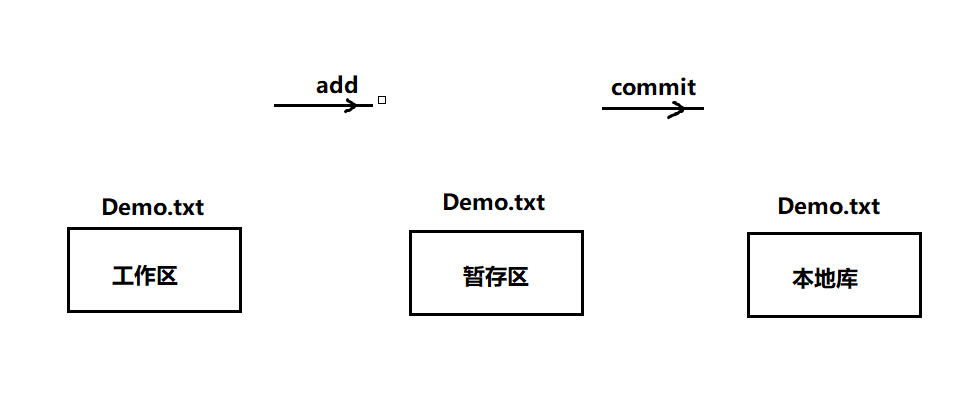
Z

### hard参数/mixed参数/soft参数

【1】hard参数：

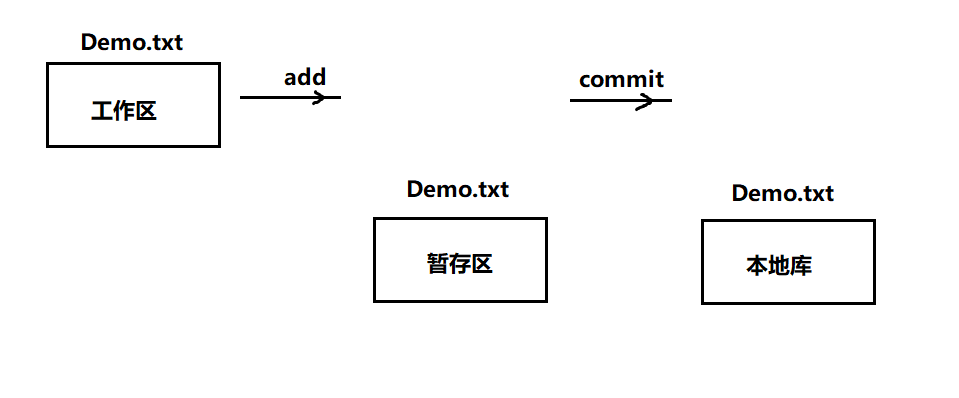
git reset --hard [索引]

本地库的指针移动的同时，重置暂存区，重置工作区



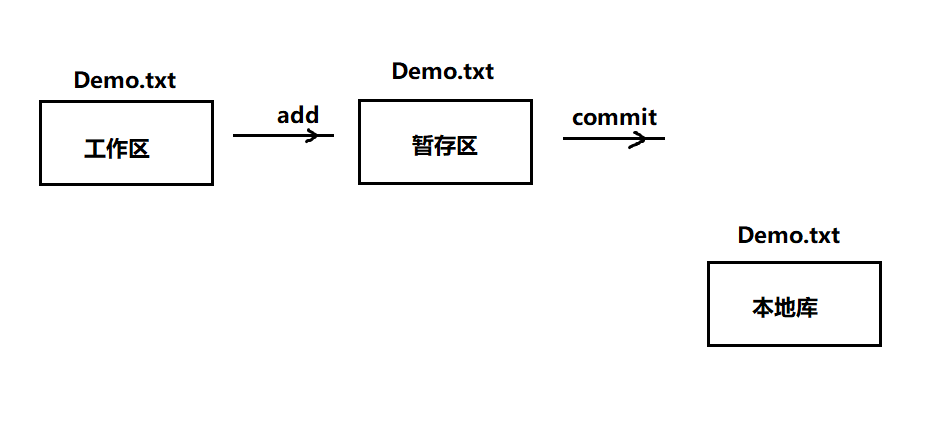
【2】mixed参数：

本地库的指针移动的同时，重置暂存区，但是工作区不动



【3】soft参数：

本地库的指针移动的时候，暂存区，工作区都不动



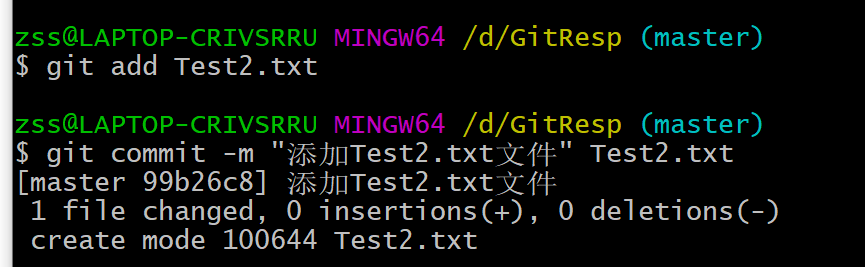
总结：以后用的多的就是  第一种hard参数

### 删除文件\_找回本地库删除的文件

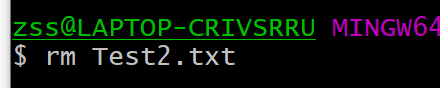
【1】新建 一个Test2.txt文件

【2】将它add到暂存区中

【3】再通过commit提交到本地库

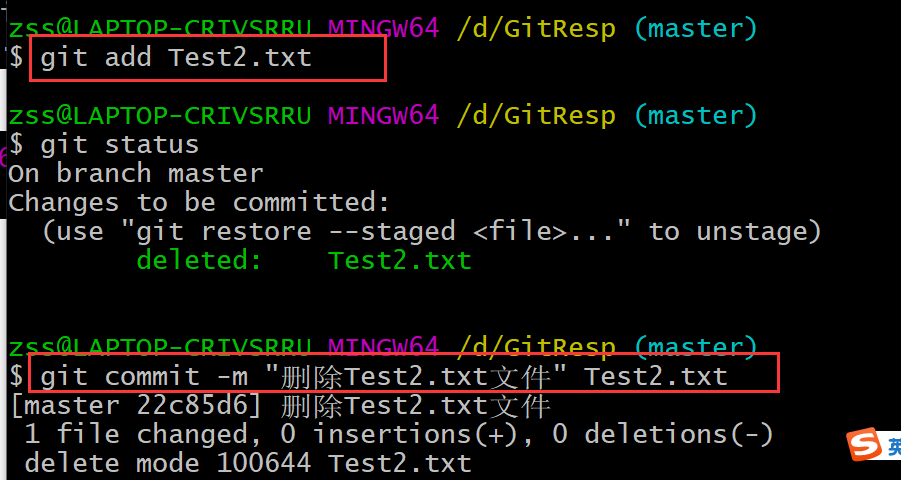


【4】删除工作区中的Test2.txt

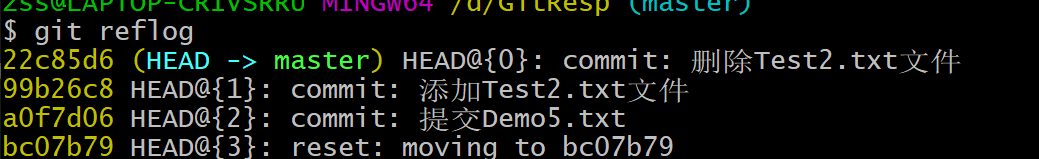


【5】将删除操作同步到暂存区：

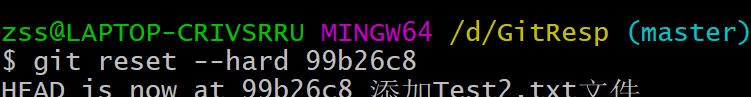
【6】将删除操作同步到本地库：



【7】查看日志：

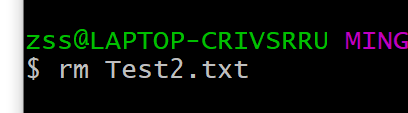


【8】找回本地库中删除的文件，实际上就是将历史版本切换到刚才添加文件的那个版本即可：

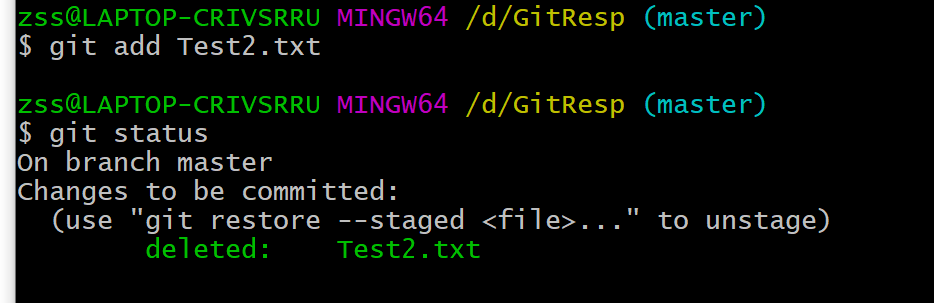


### 找回暂存区删除的文件

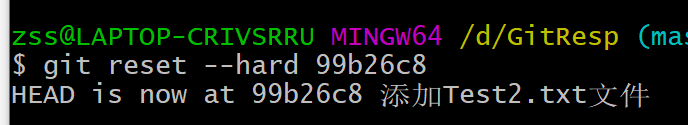
【1】删除工作区数据：

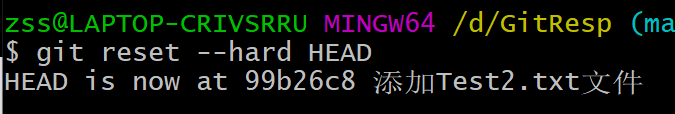


【2】同步到缓存区：



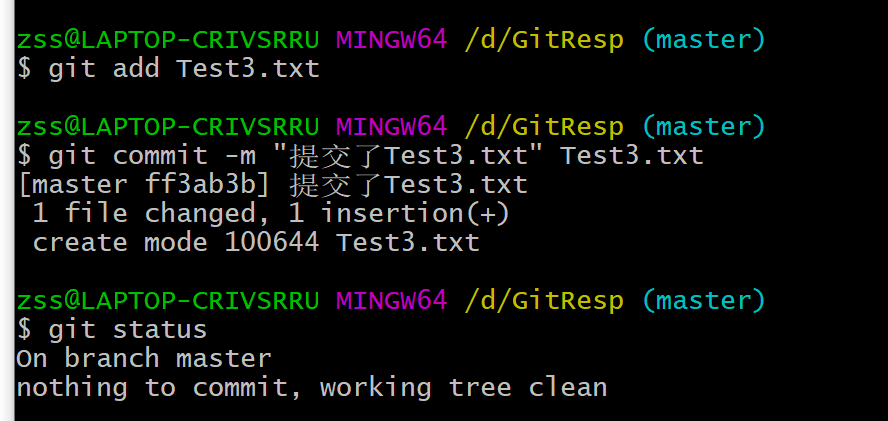
【3】后悔了，恢复暂存区中数据：



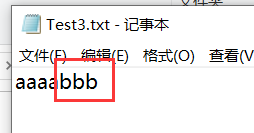


### diff命令

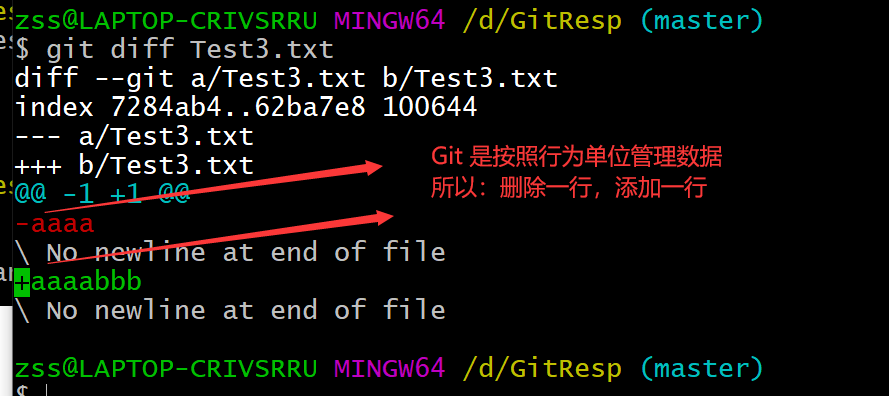
【1】先创建一个文件，添加到暂存区，再提交到本地库：



【2】更改工作区中Test3.txt中内容，增加内容：

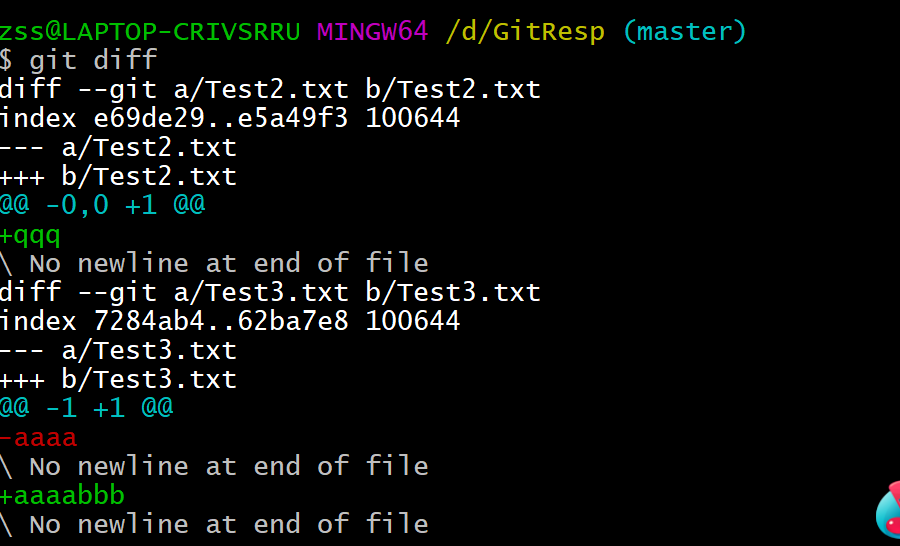


导致：工作区  和 暂存区  不一致，比对：



总结：   git diff [文件名]   ---》   将工作区中的文件和暂存区中文件进行比较

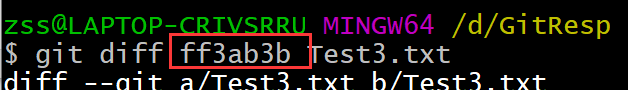
多个文件的比对：

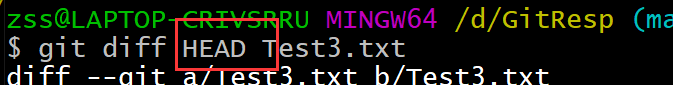


总结：git diff --->比较工作区中和暂存区中 所有文件的差异

比较暂存区和本地库中差别：

git diff [历史版本][文件名]   ---》比较暂存区和本地库中内容





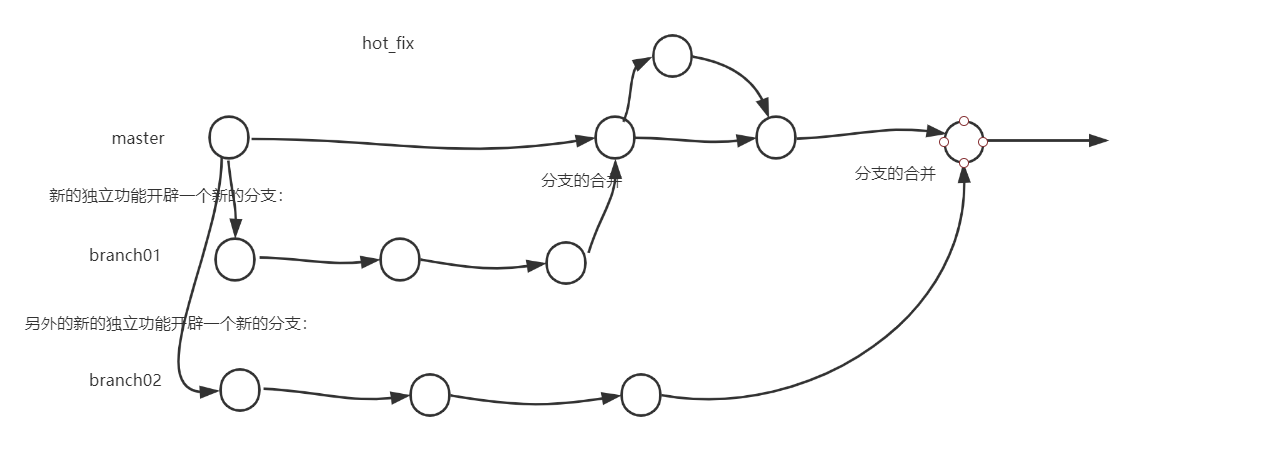
## 分支

### 什么是分支

【1】什么是分支：

在版本控制过程中，使用多条线同时推进多个任务。这里面说的多条线，就是多个分支。

【2】通过一张图展示分支：



【3】分支的好处：

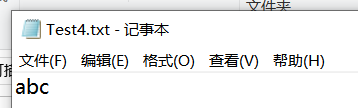
同时多个分支可以并行开发，互相不耽误，互相不影响，提高开发效率

如果有一个分支功能开发失败，直接删除这个分支就可以了，不会对其他分支产生任何影响。

### 操作分支

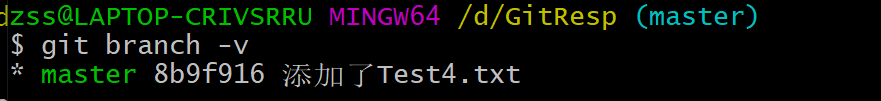
#### 查看，创建，切换分支

【1】在工作区创建一个Test4.txt文件，然后提交到暂存区，提交到本地库：

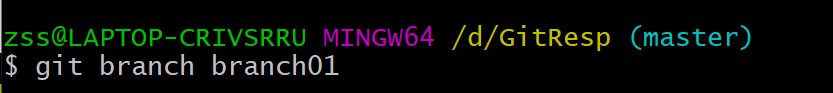




【2】查看分支：



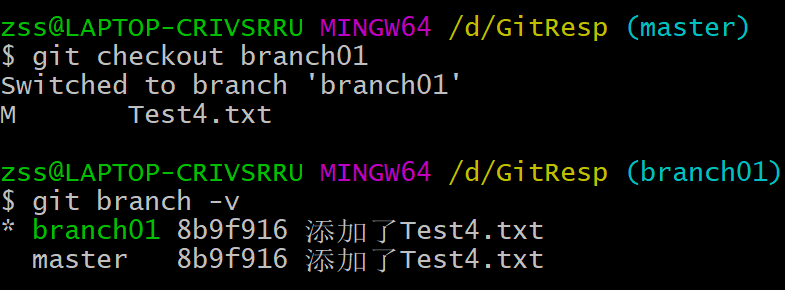
【3】创建分支：



再查看：

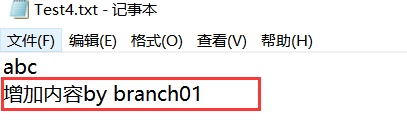


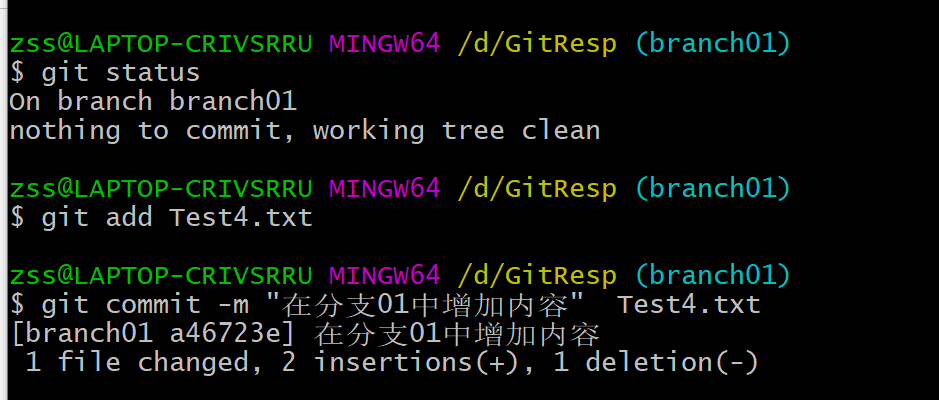
【4】切换分支：



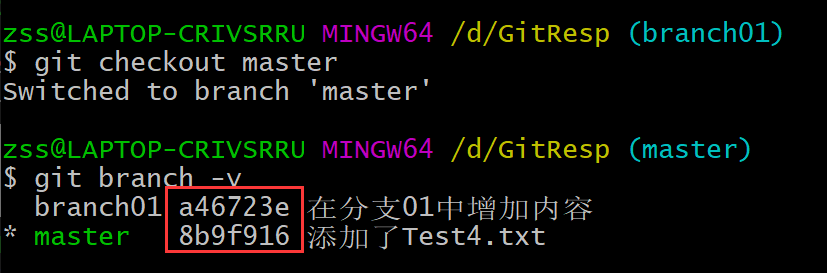
#### 冲突问题，如何解决冲突题

【1】进入branch01分支，增加内容：

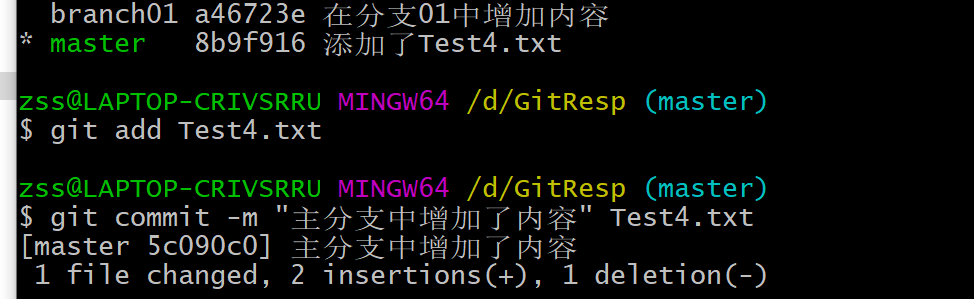


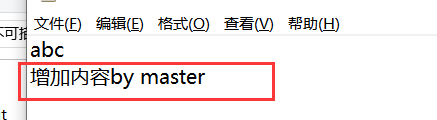


【2】将分支切换到master:

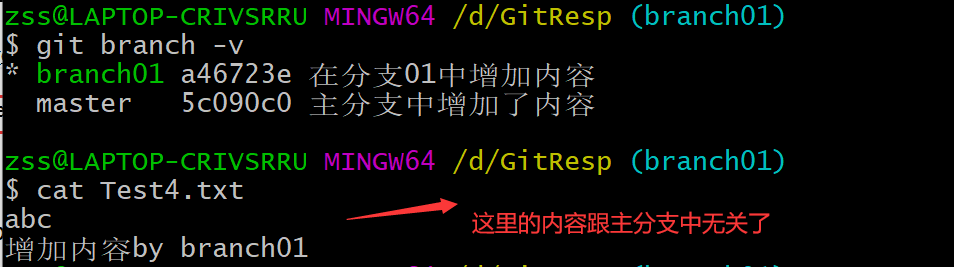


然后在主分支下 加入内容：



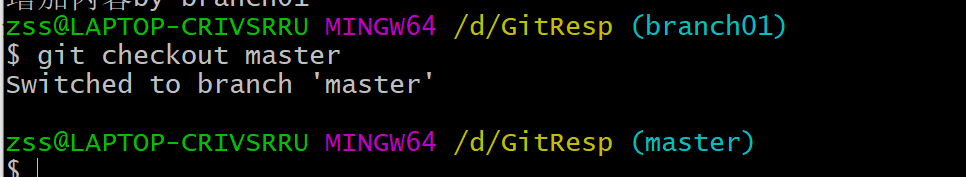


【3】再次切换到branch01分支查看：

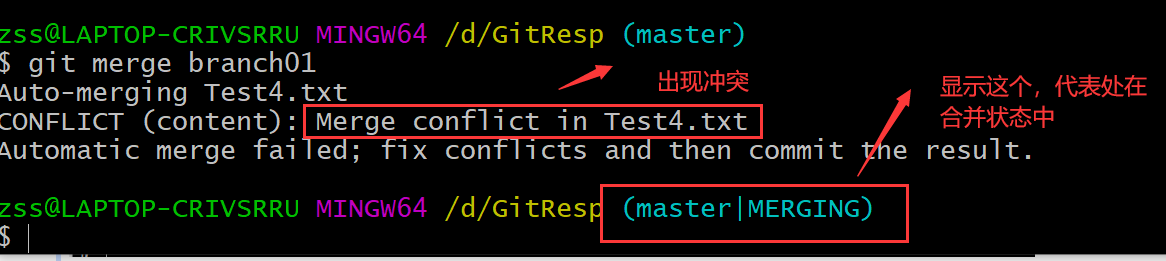


【4】将branch01分支  合并到  主分支 ：

（1）进入主分支：



（2）将branch01中的内容和主分支内容进行合并：

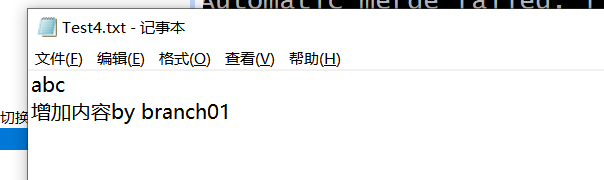


查看文件：出现冲突：



解决：

公司内部商议解决，或者自己决定  人为决定，留下想要的即可：



将工作区中内容添加到暂存区：

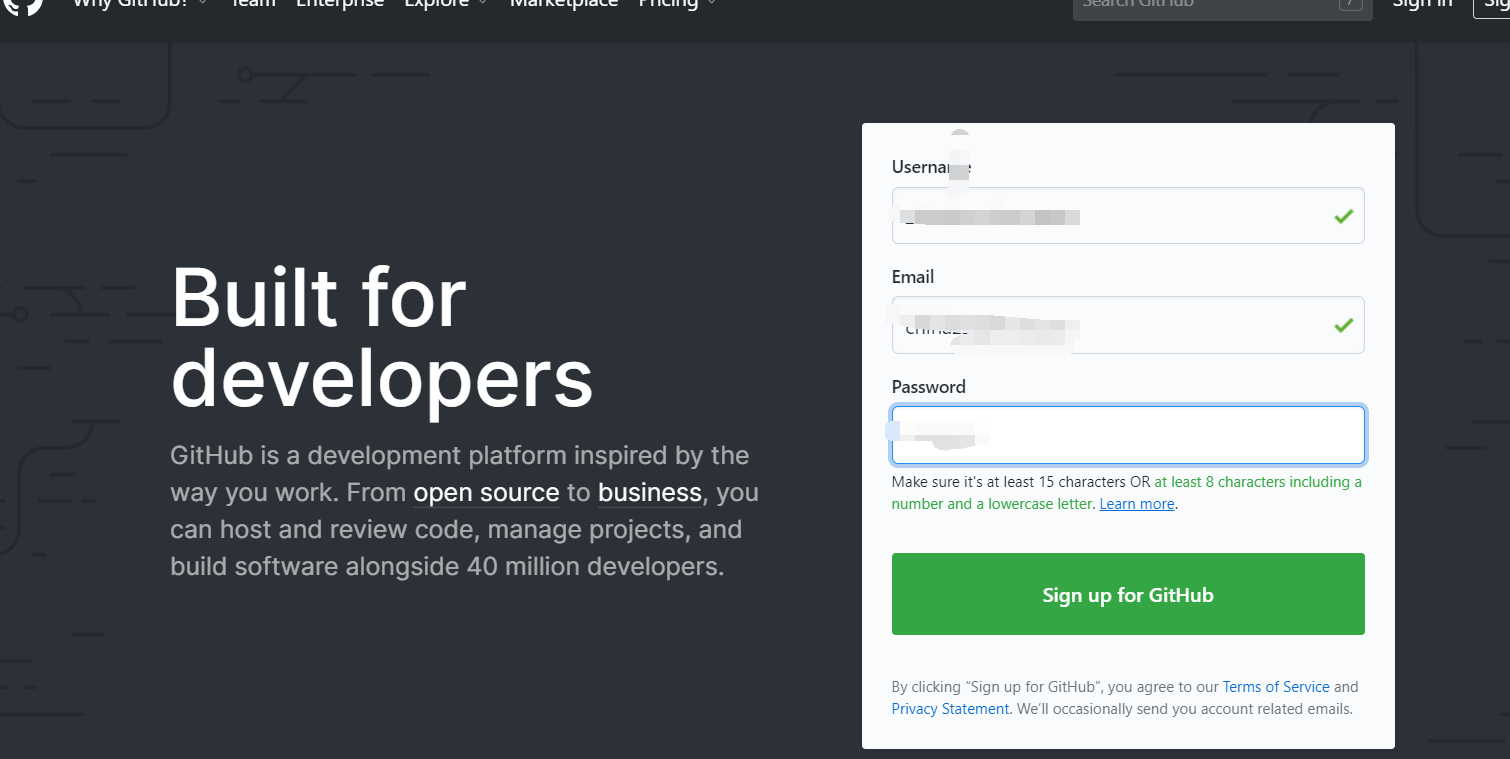


然后进行commit操作：

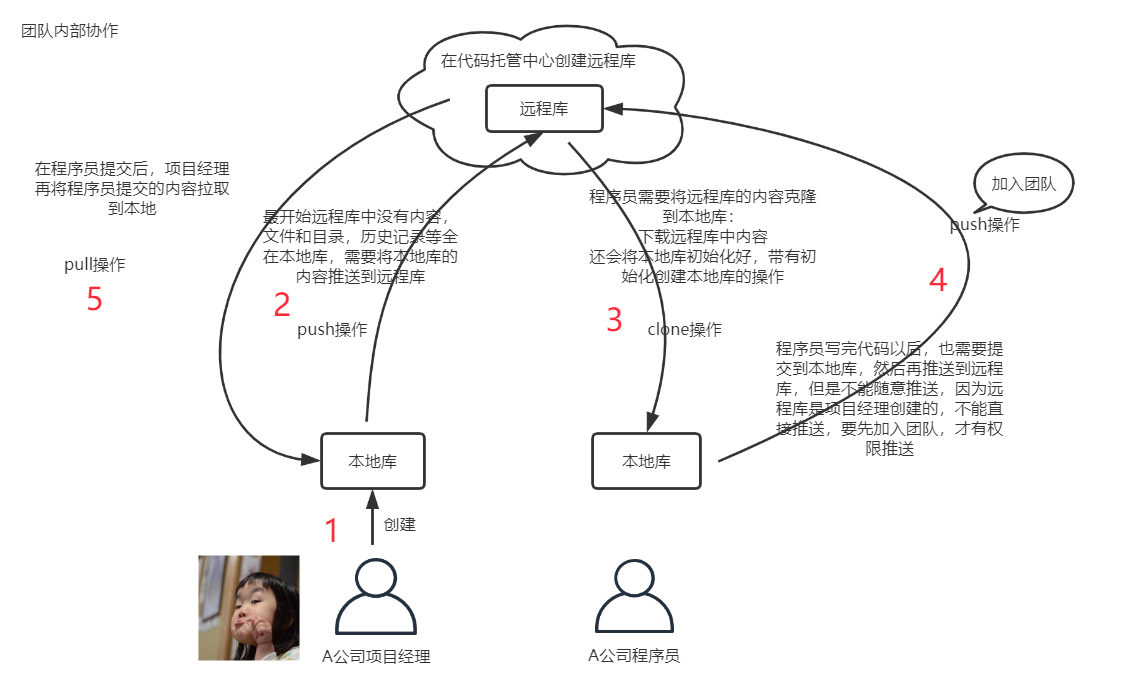


## GitHub账号注册

官网：<https://github.com/>



## 回顾本地库和远程库交互方式

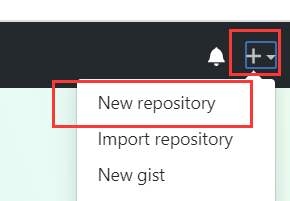


## 初始化本地库

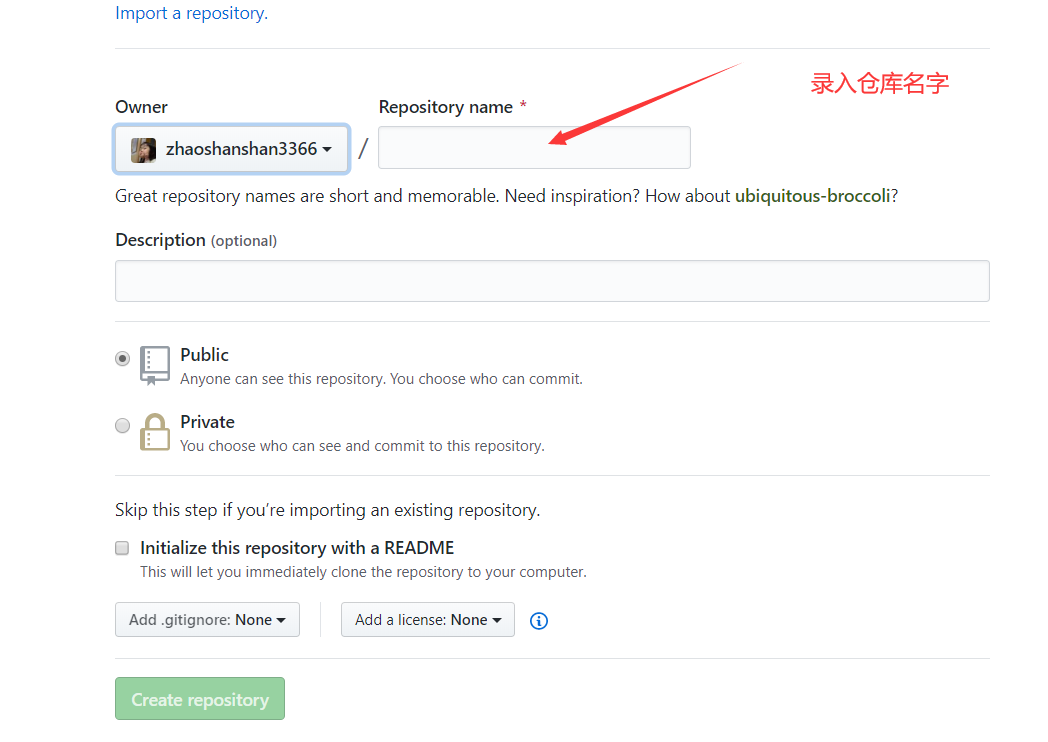


## 创建远程库

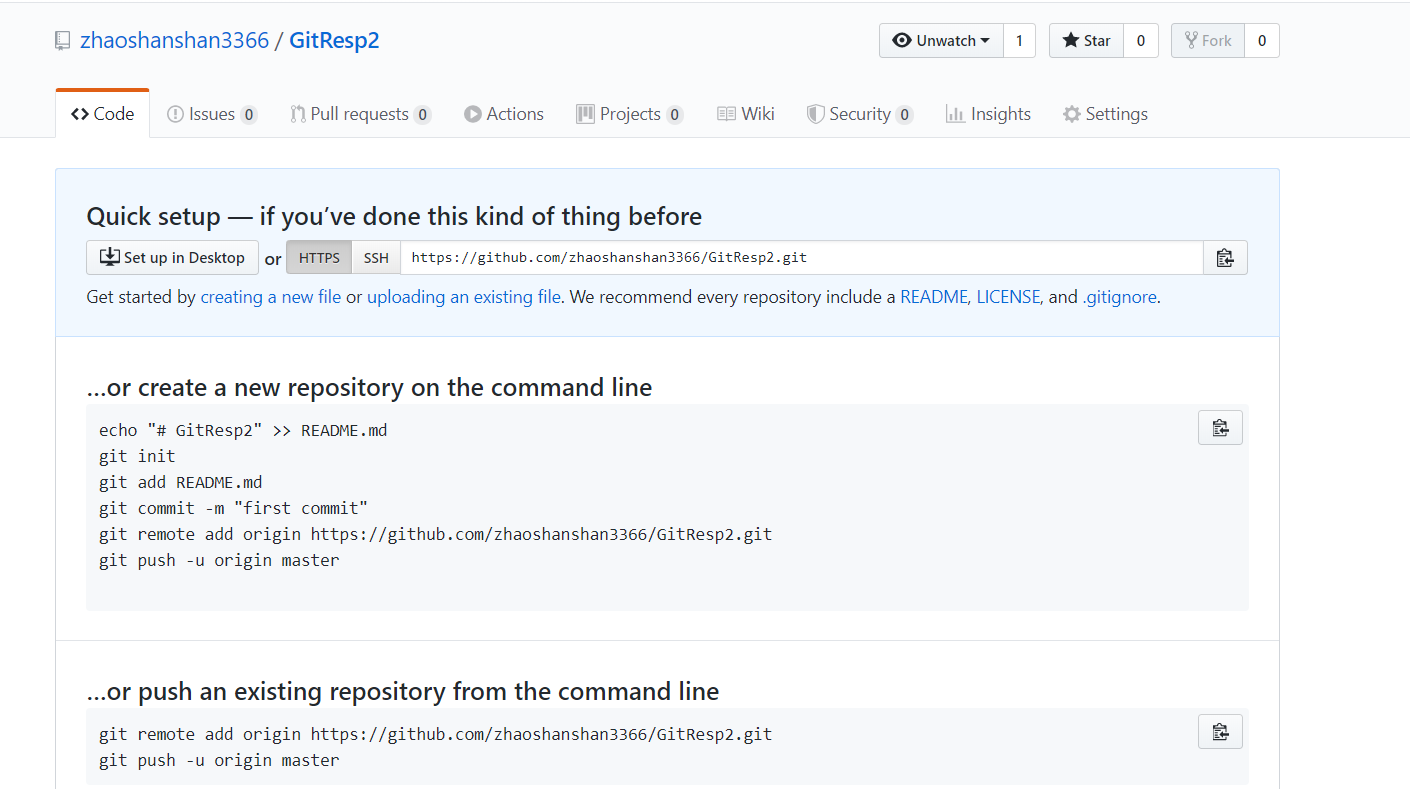
【1】创建远程库



【2】录入信息：

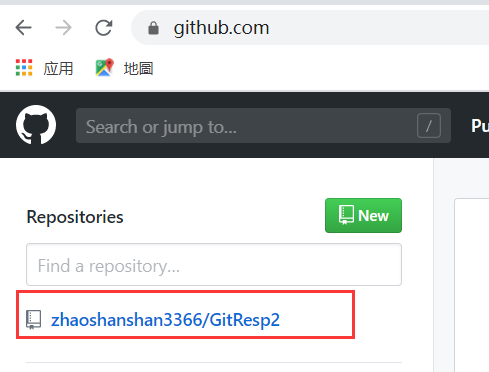


【3】完成状态：

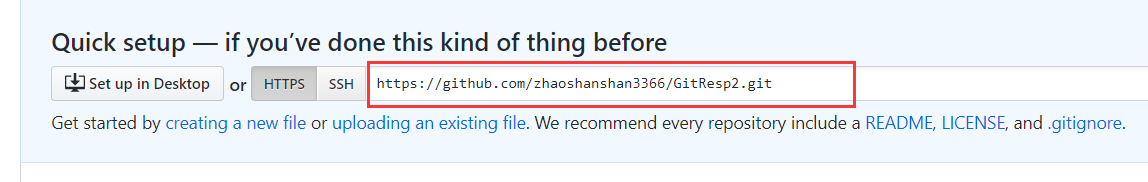


## 在本地创建远程库地址的别名

远程库的地址：



点击进入：



远程库地址比较长，每次复制比较麻烦

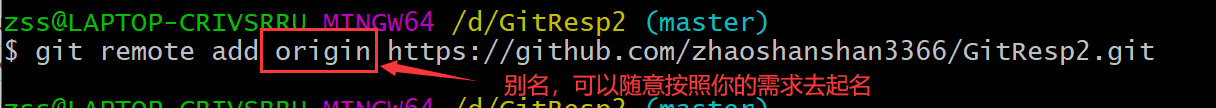
<https://github.com/zhaoshanshan3366/GitResp2.git>

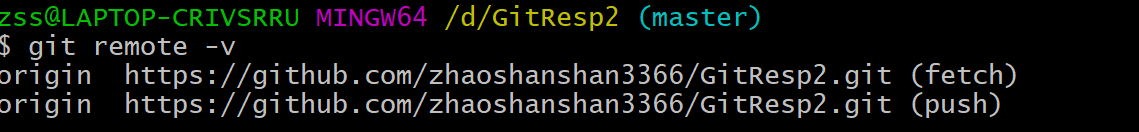
在Git本地将地址保存，通过别名

查看别名：



起别名：

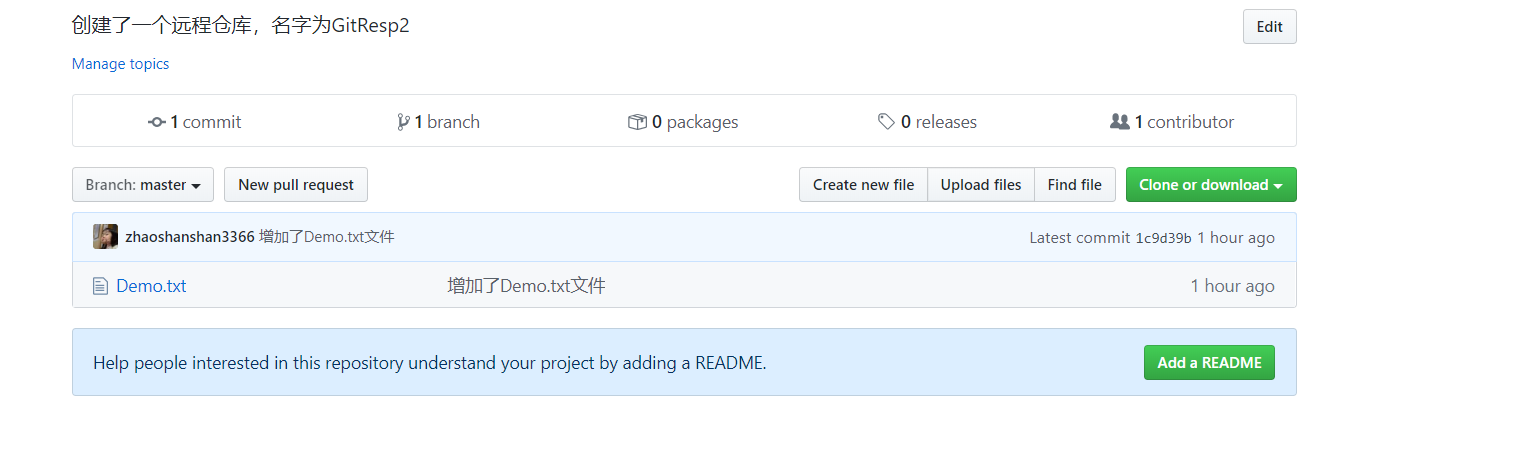




## 推送操作

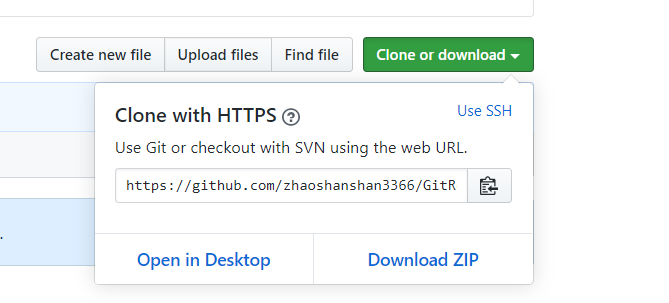


推送成功以后，查看自己的远程库：

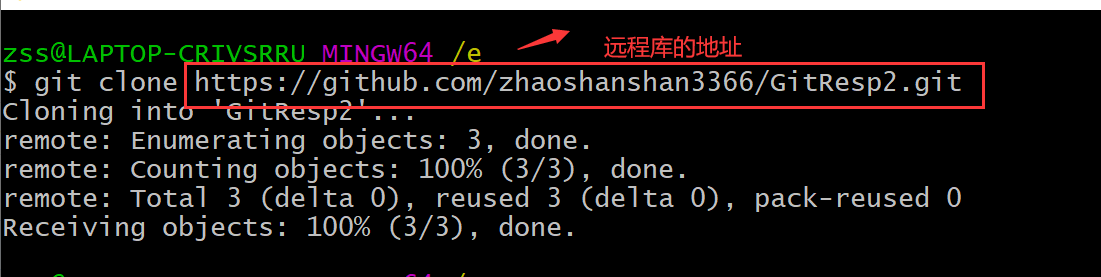


## 克隆操作

远程库地址复制：



克隆操作：

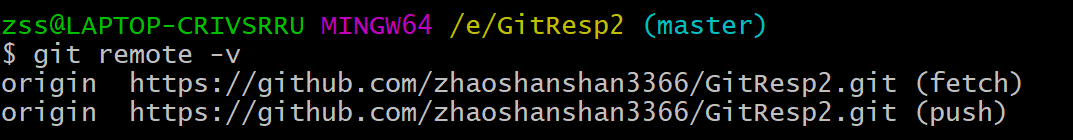


克隆操作可以帮我们完成：

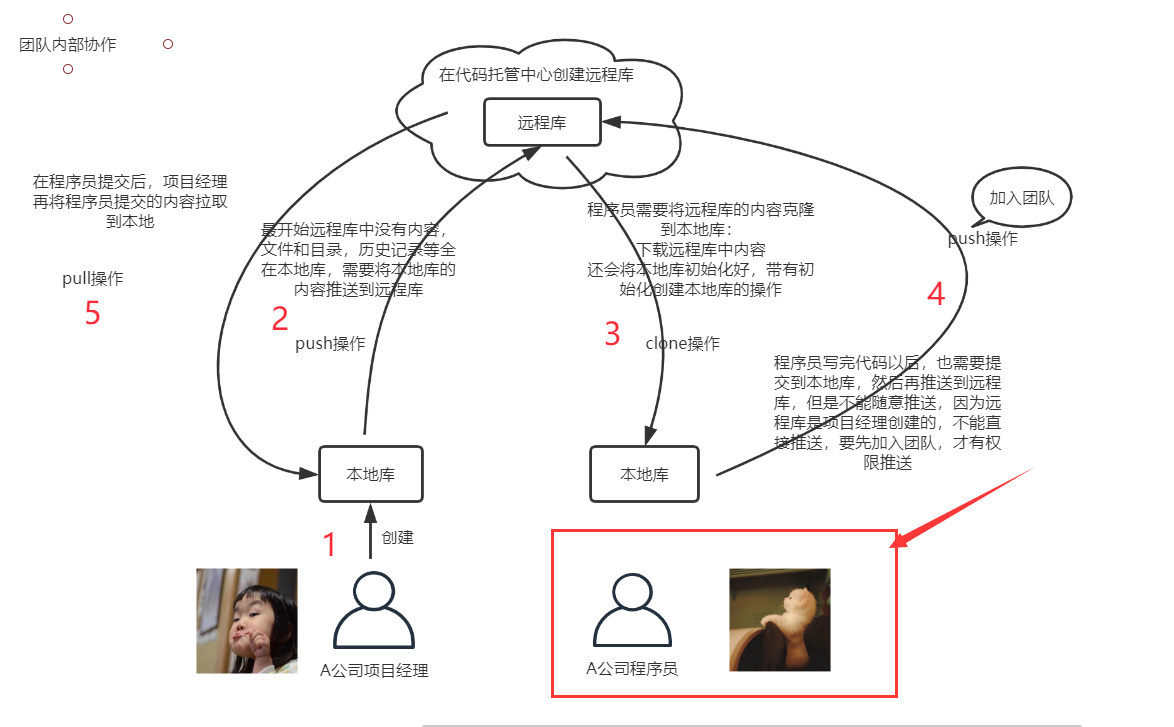
（1）初始化本地库

（2）将远程库内容完整的克隆到本地

（3）替我们创建远程库的别名：



## 邀请加入团队，push操作



【1】更新本地库信息：



【2】push内容到远程库中去：

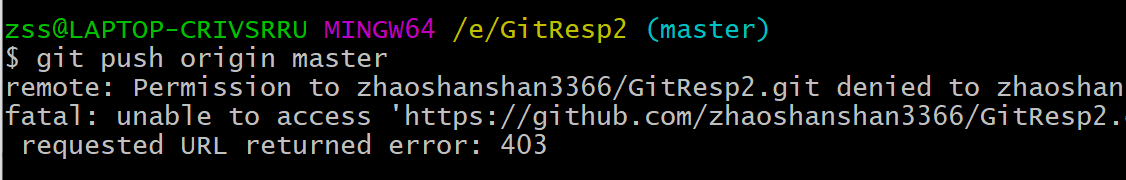
发现可以直接push进去，并没有让我录入账号密码，或者也没有提示错误 - --》结果 很诡异

原因：git使用的时候在本地有缓存：

将缓存删除：

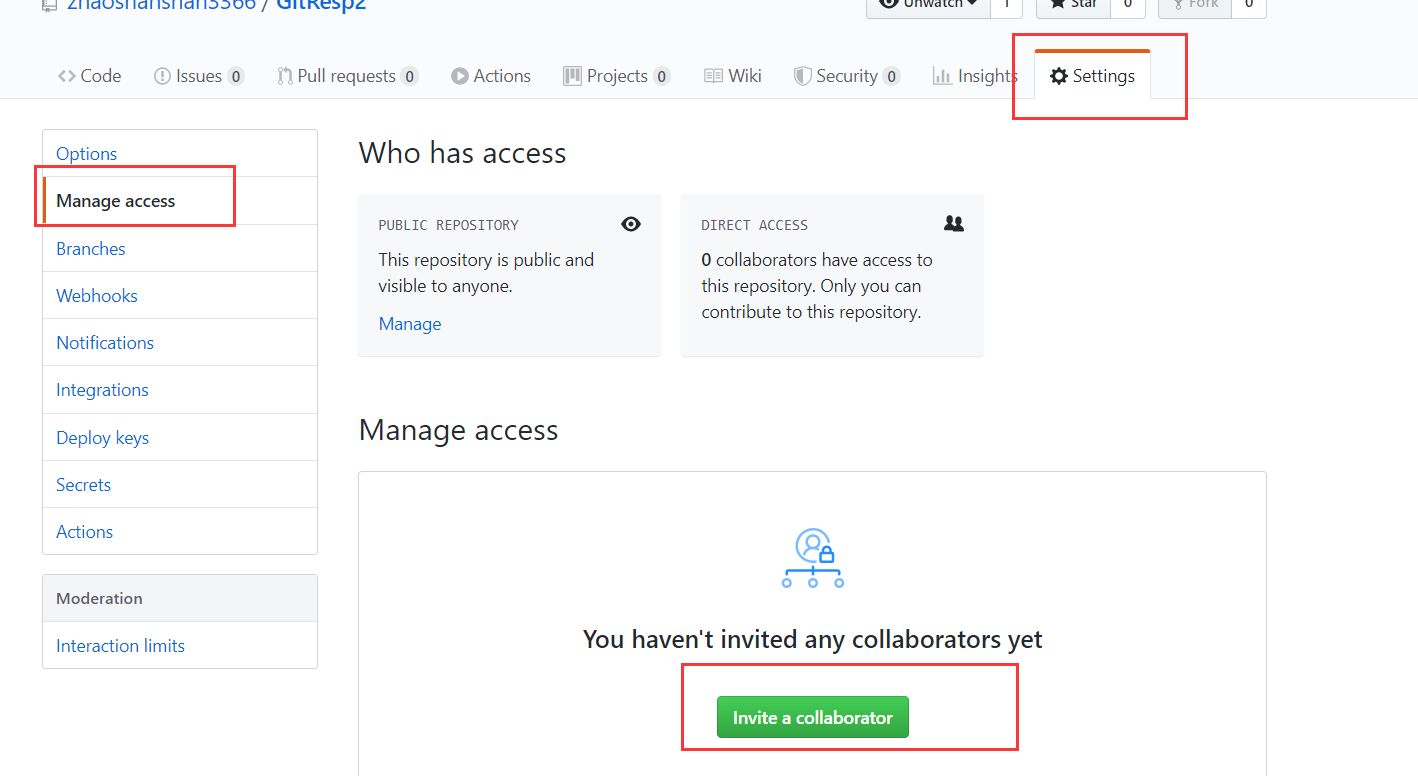


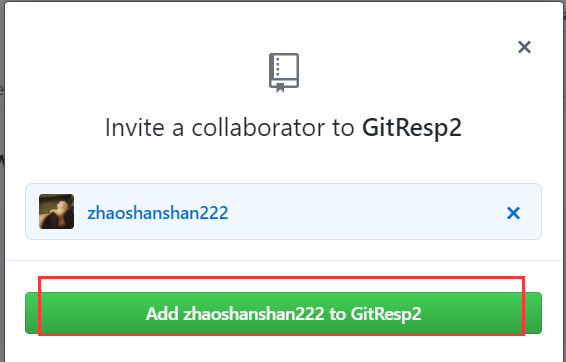
现在再次重新push，发现出错了：

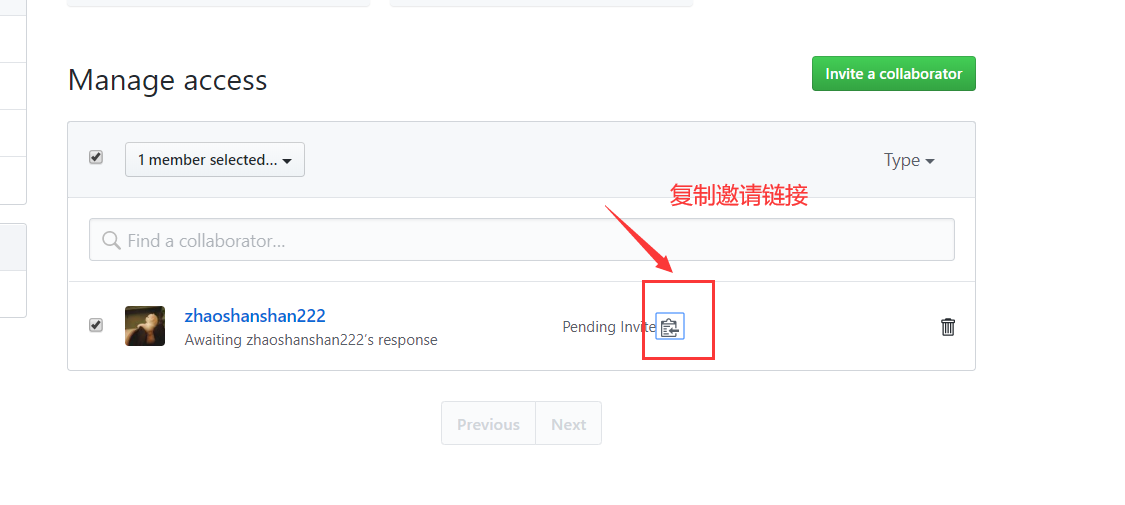


必须要加入团队：

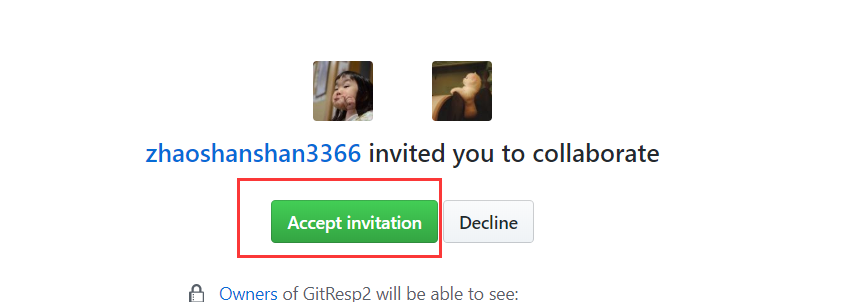
登录项目经理的账号，邀请普通成员：





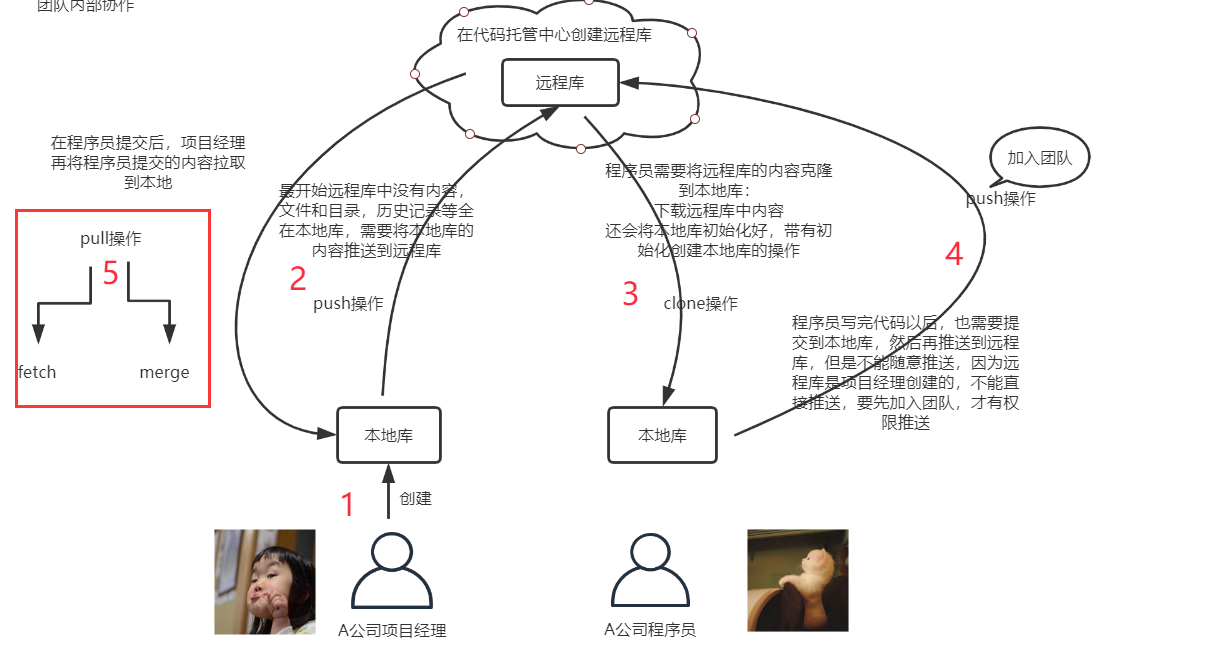


登录被邀请者的账号，接收邀请：（在地址栏录入邀请链接即可：<https://github.com/zhaoshanshan3366/GitResp2/invitations>）



## 远程库修改的拉取操作

【1】拉取操作 pull操作，相当于  fetch+merge



【2】项目经理先先确认远程库内容是否更新了：

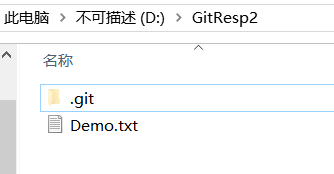


【3】项目经理进行拉取：

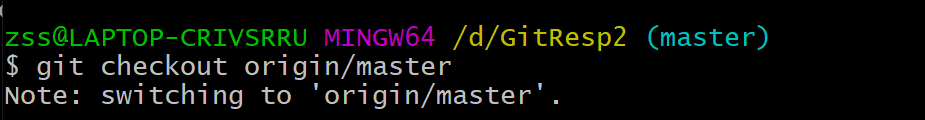
（1）先是抓取操作：fetch:

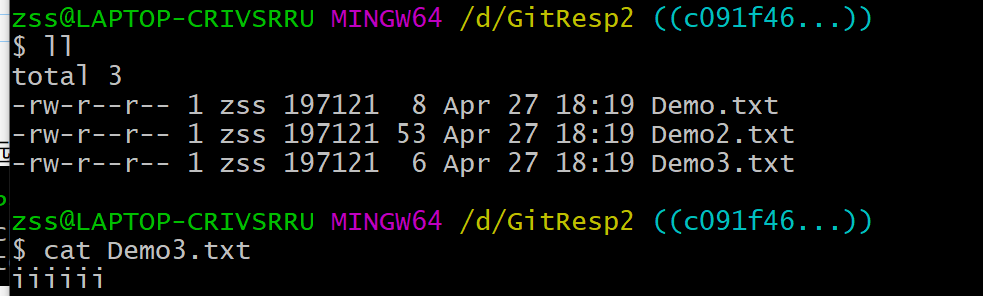


在抓取操作执行后，只是将远程库的内容下载到本地，但是工作区中的文件并没有更新。工作区中还是原先的内容：



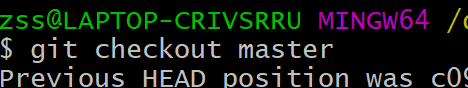
抓取后可以去远程库看看内容是否正确：



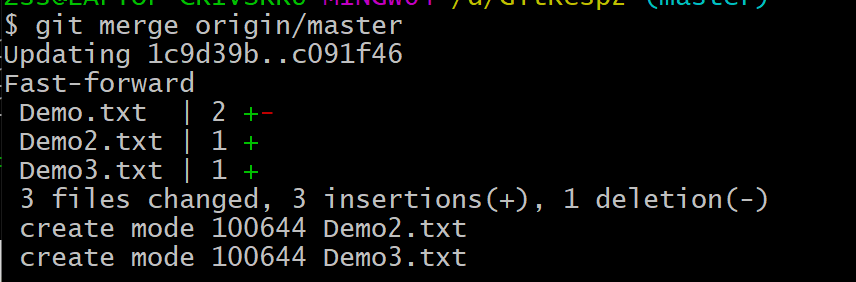


然后发现内容都正确，就可以进行合并操作了：

合并之前应该将分支切换回来：

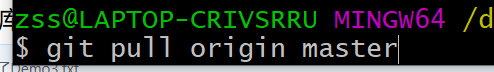


（2）进行合并：merge:



## 远程库修改的拉取操作2

远程库的拉取可以直接利用pull命令来完成：



fetch+merge操作：---》为了保险，慎重

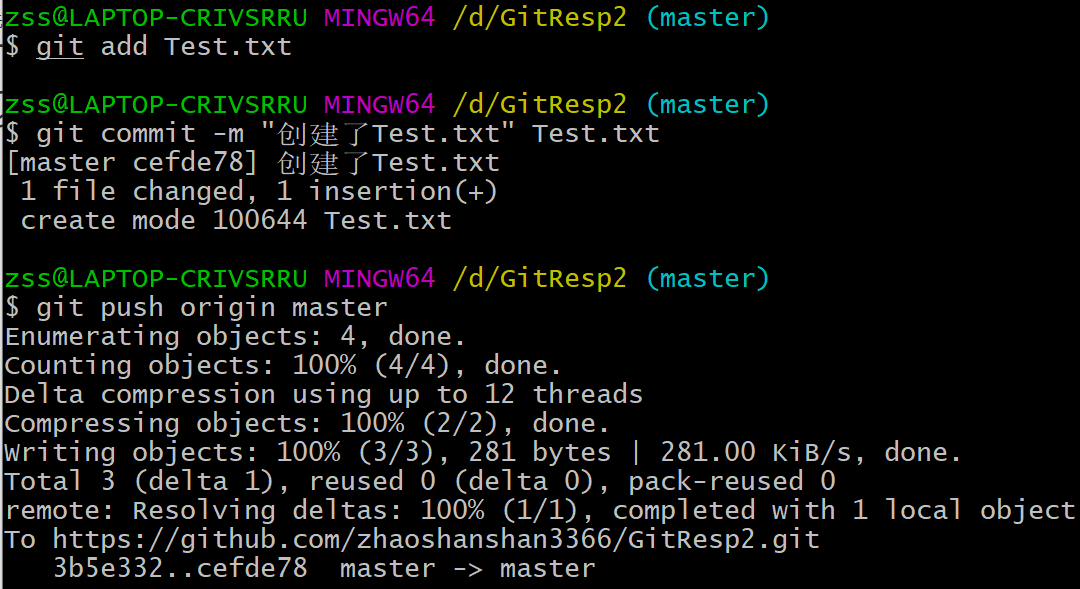
pull --->代码简单，省事

## 协同开发合作时冲突的解决办法

【1】



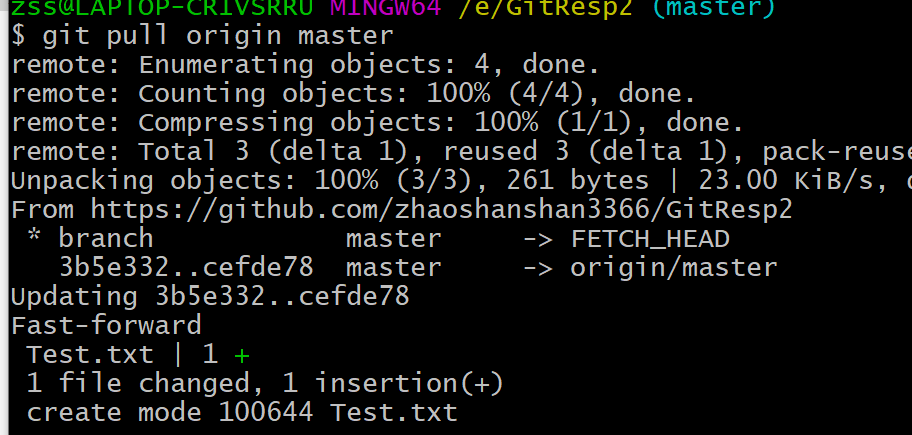
向远程库推送数据：



【2】



做了一个拉取操作：



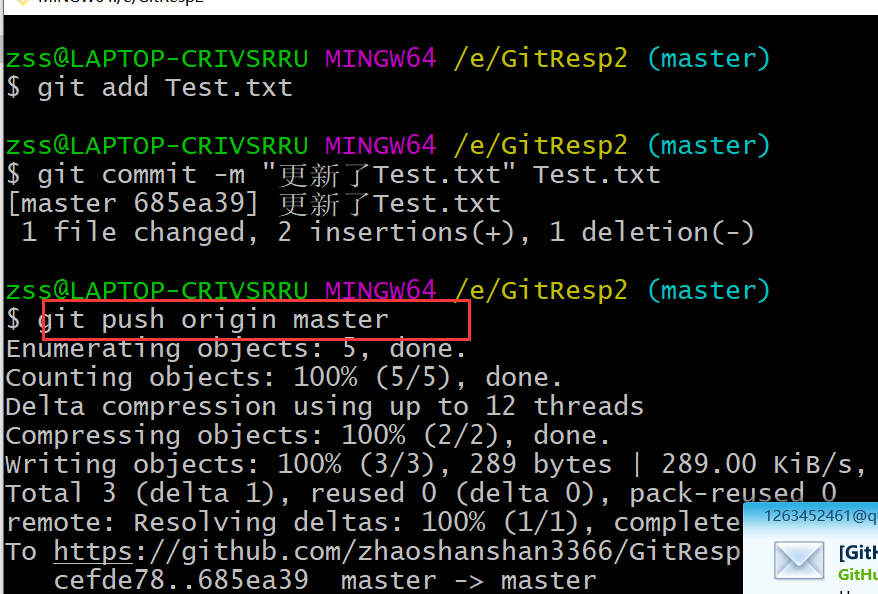
到这里为止，现在远程合作没有任何问题。

现在操作同一个文件的同一个位置的时候，就会引起冲突：

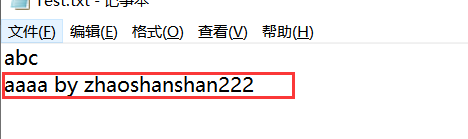
【3】



再次做了推送操作：



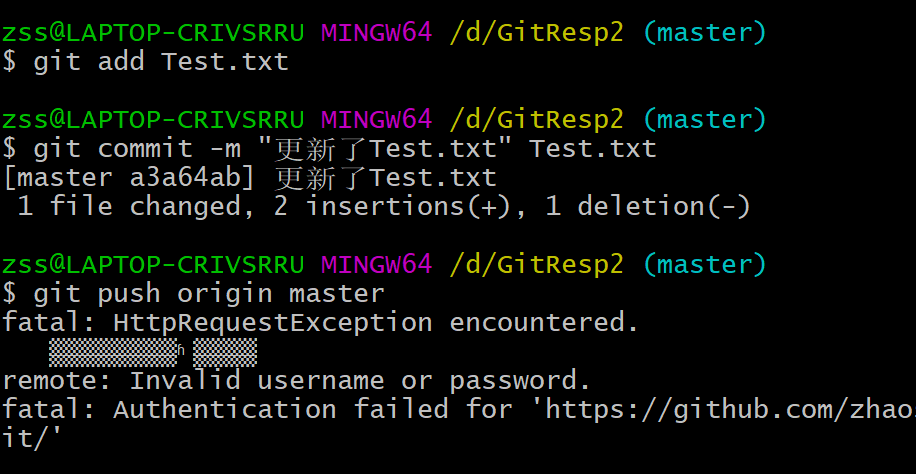
改动位置：



【4】



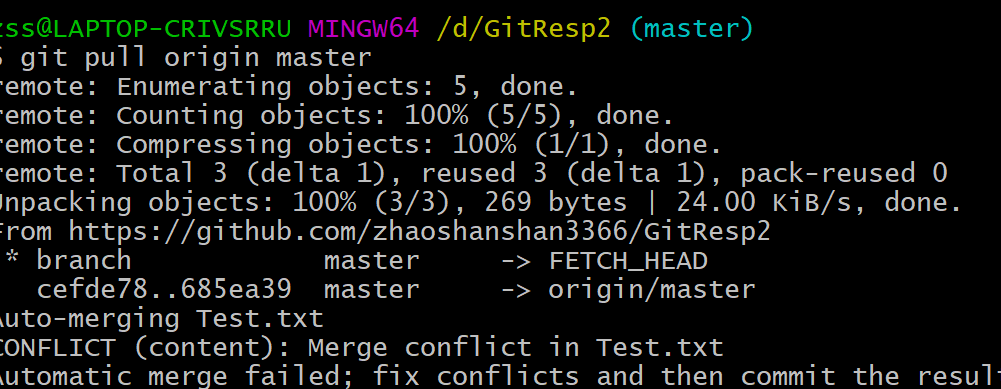
改动Test.txt中内容，然后进行推送：



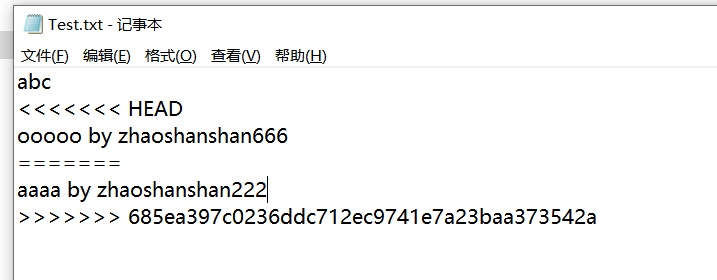
发现 推送失败！

在冲突的情况下，先应该拉取下来，然后修改冲突，然后再推送到远程服务器：

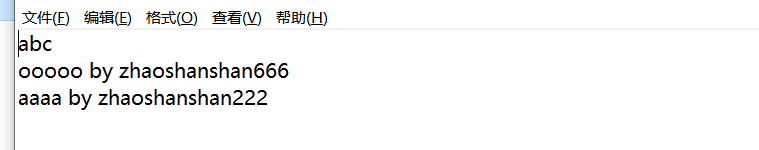
先拉取：



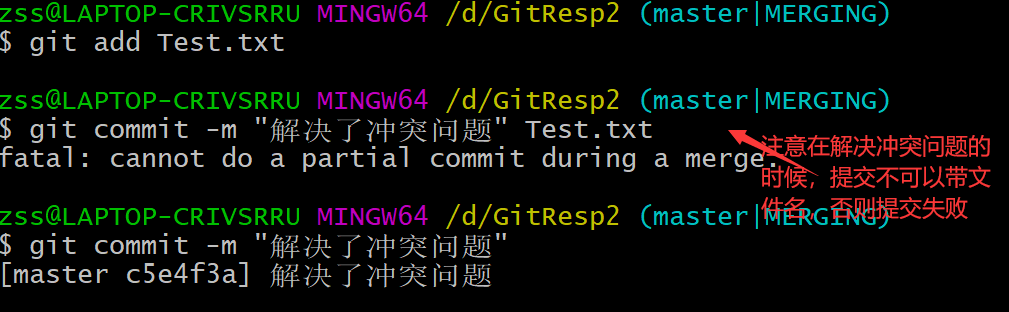
查看冲突：



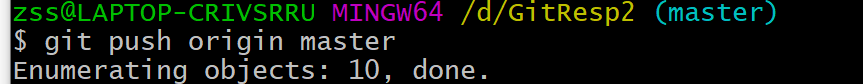
人为解决这个冲突：（该删的删，该留的留）



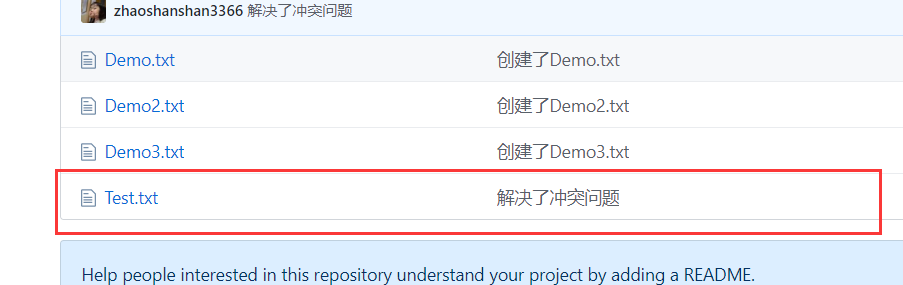
解决完冲突以后，向服务器推送：



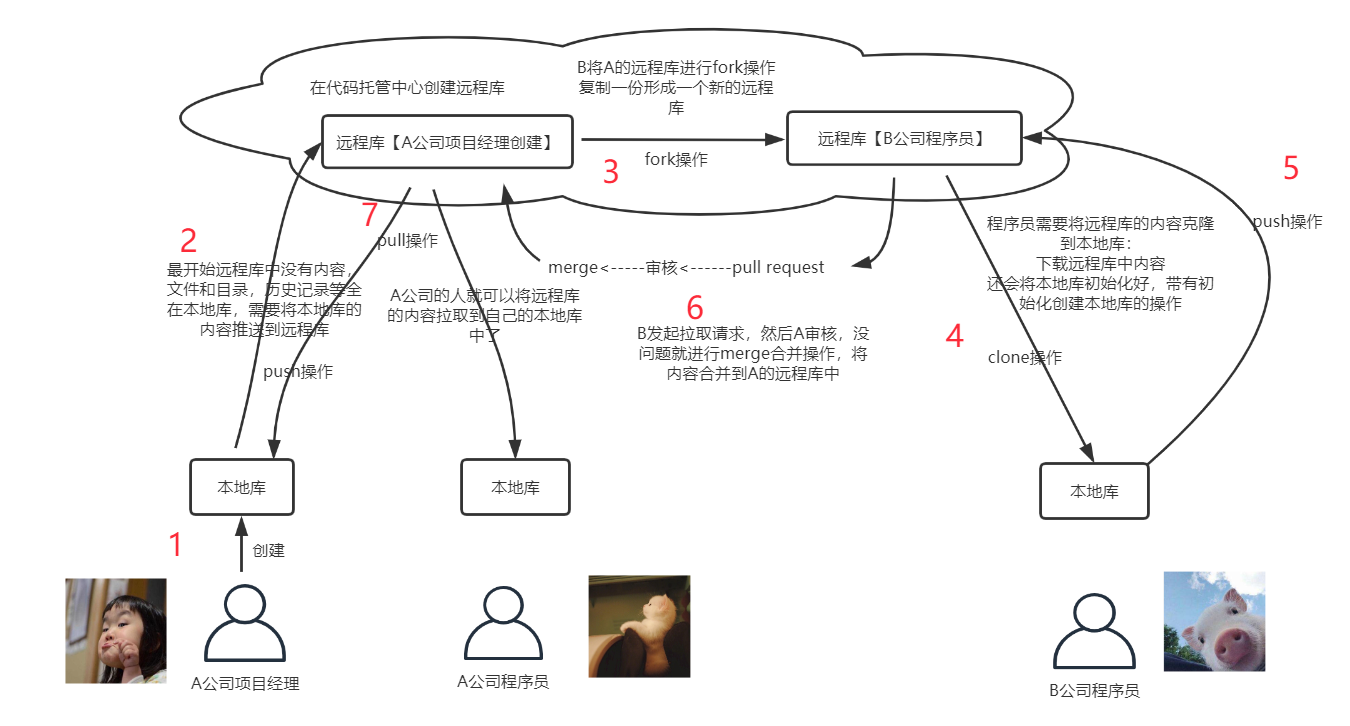
推送：



解决了冲突问题：



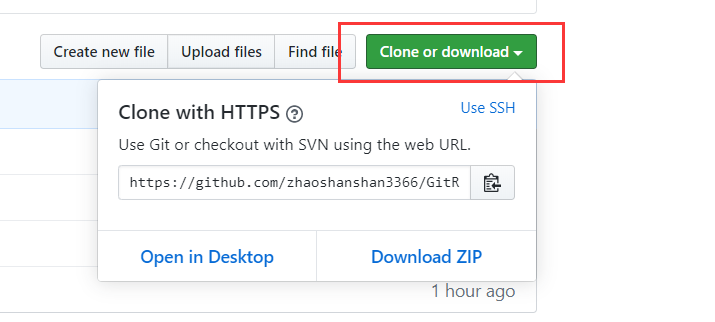
## 回顾跨团队合作交互方式



## 跨团队合作

【1】得到远程库的地址：





地址：

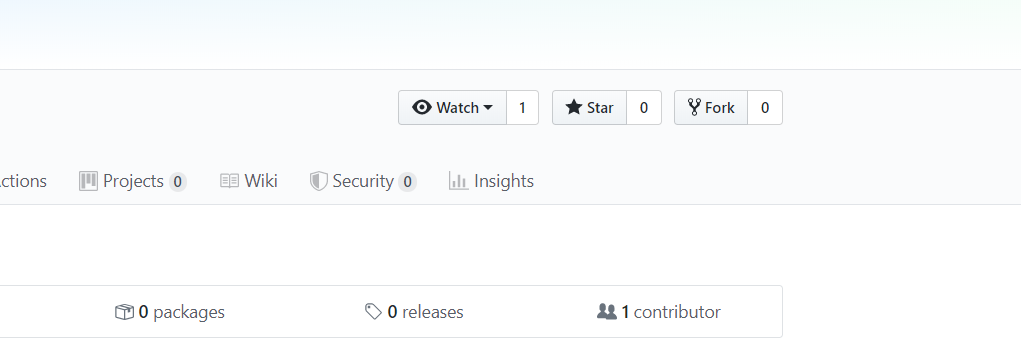
<https://github.com/zhaoshanshan3366/GitResp2.git>

【2】进行fork操作：



进入到账号后：复制地址：https://github.com/zhaoshanshan3366/GitResp2.git

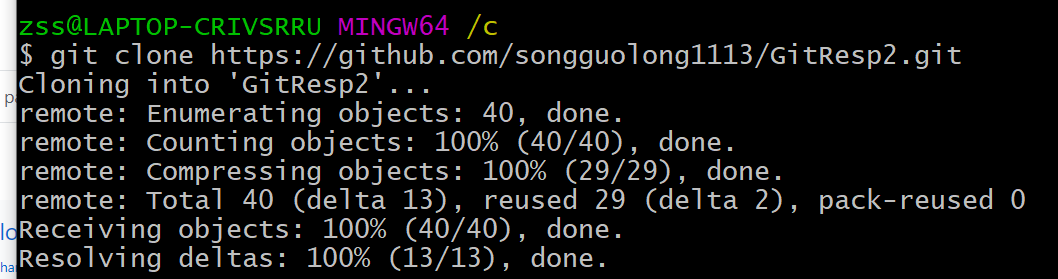
然后点击下面的fork操作：



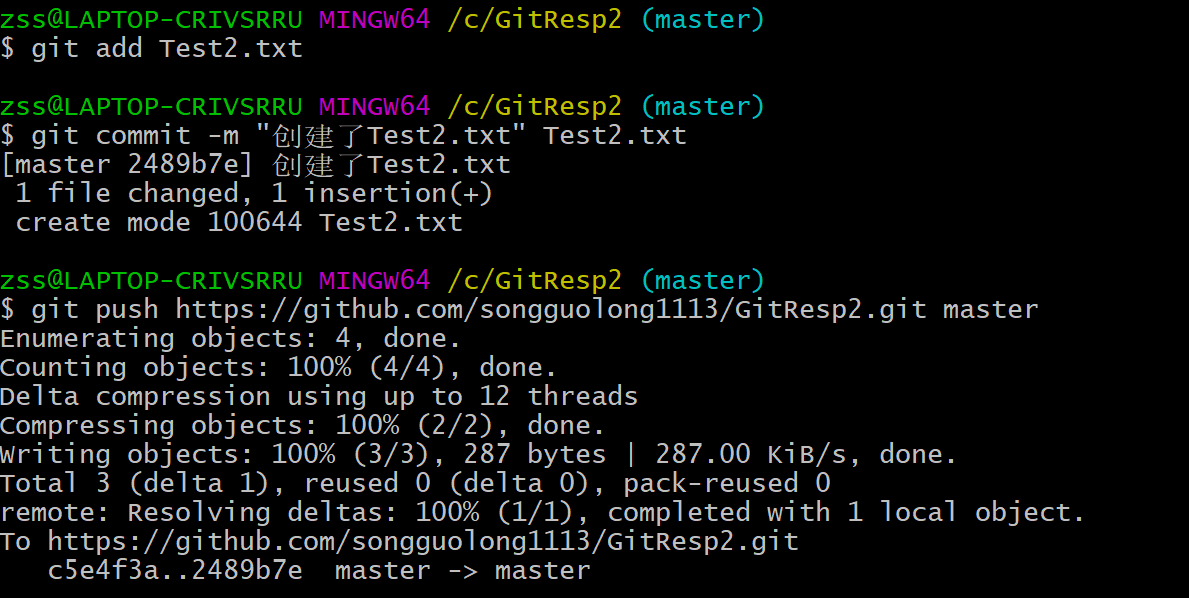


【3】然后就可以克隆到本地，并且进行修改：



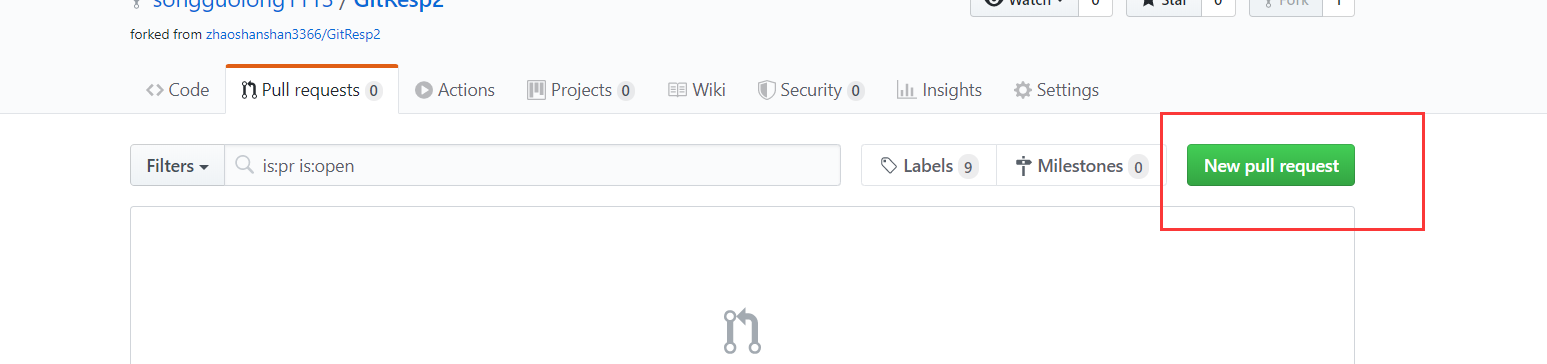


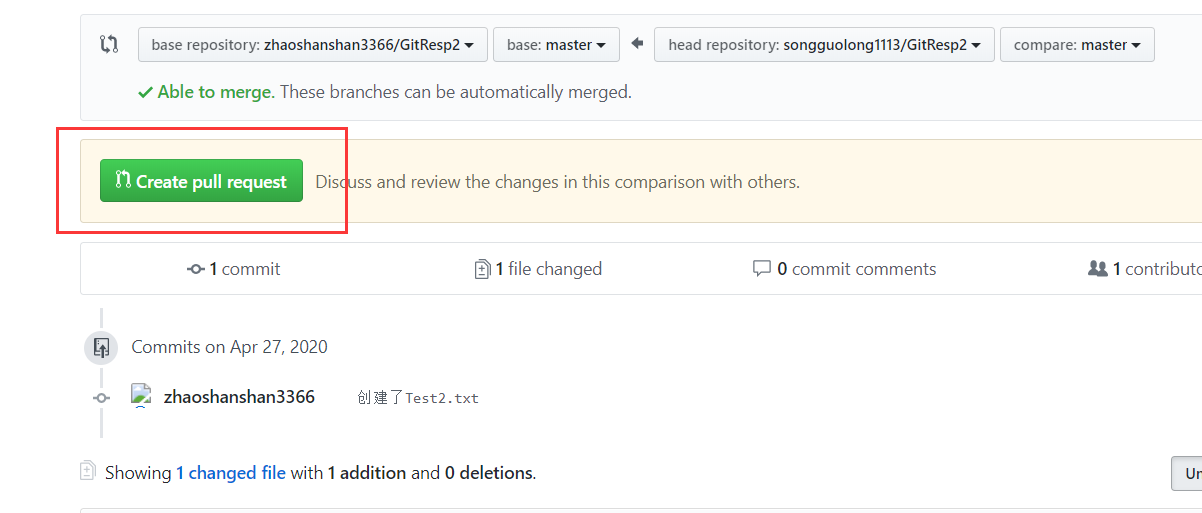
然后更改数据：添加到暂存区，然后提交到本地库，然后push到远程库：

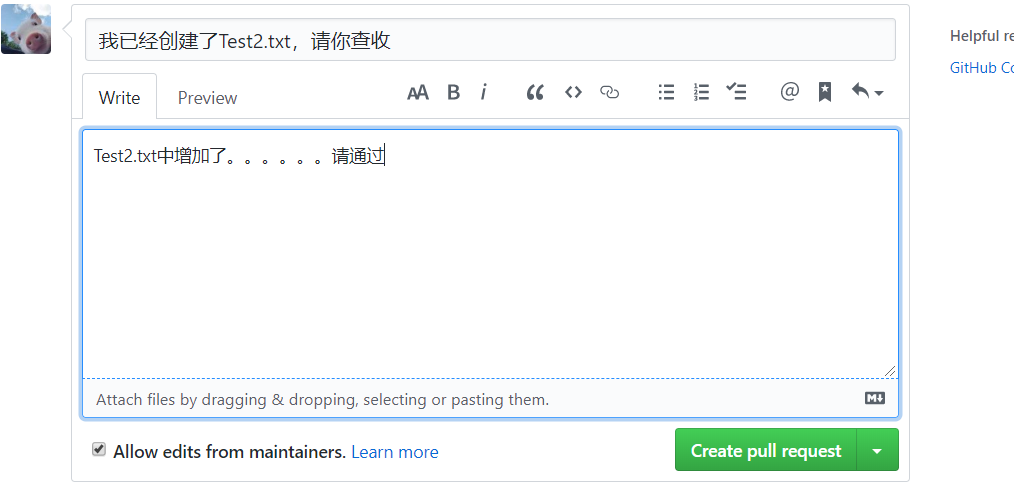


【4】进行pull request操作：











【5】进行审核操作：

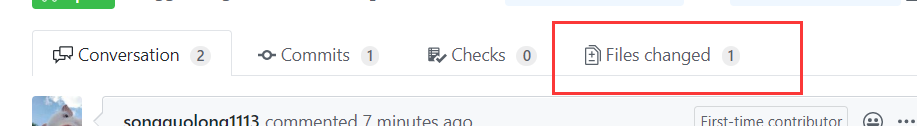




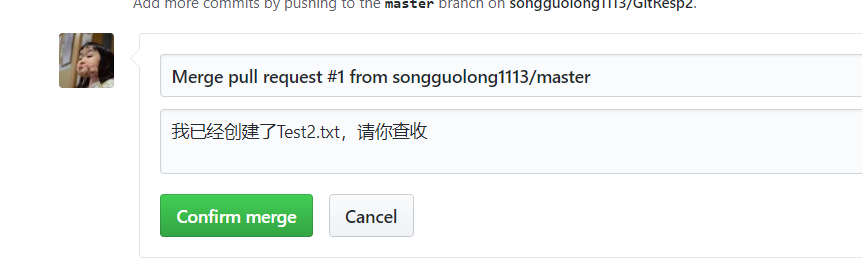
可以互相留言：



查看具体提交的内容：



确定通过以后，可以进行合并：



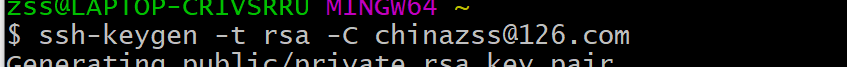
## SSH免密登录

免密操作：

【1】进入用户的主目录中：

IMG_256

【2】执行命令，生成一个.ssh的目录：

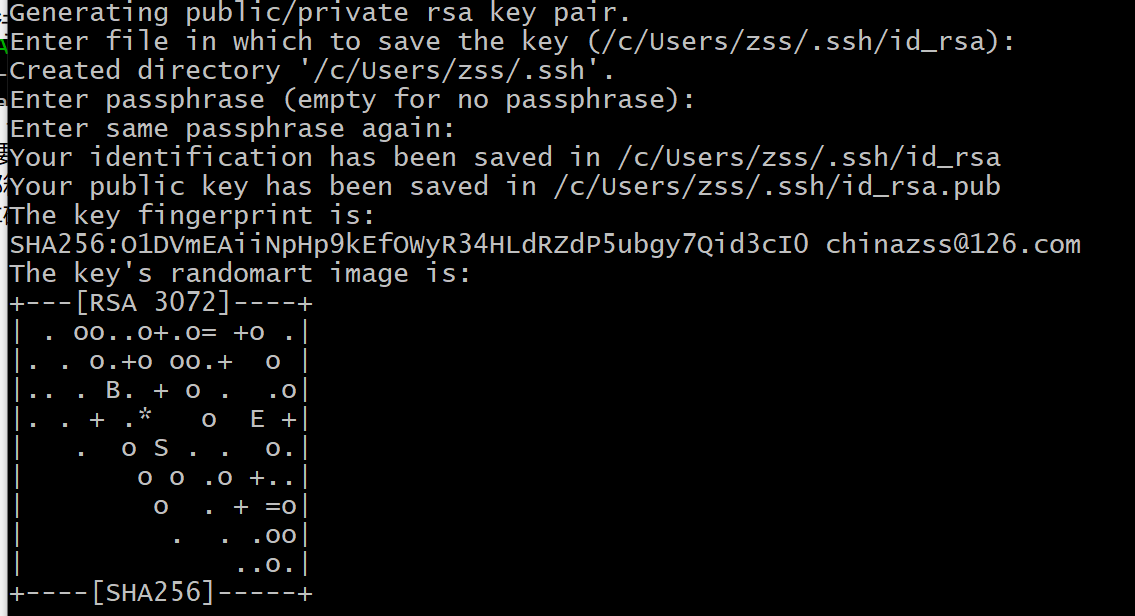


keygen  --- >  key generation

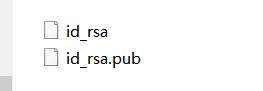
注意：C要大写

后面的邮箱，是你的github注册的账号的时候对应的邮箱

三次回车确认默认值即可



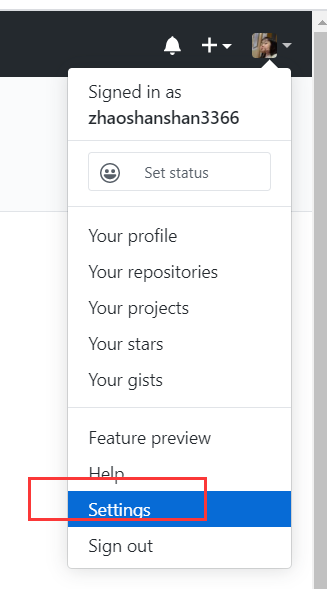
发现在.ssh目录下有两个文件：

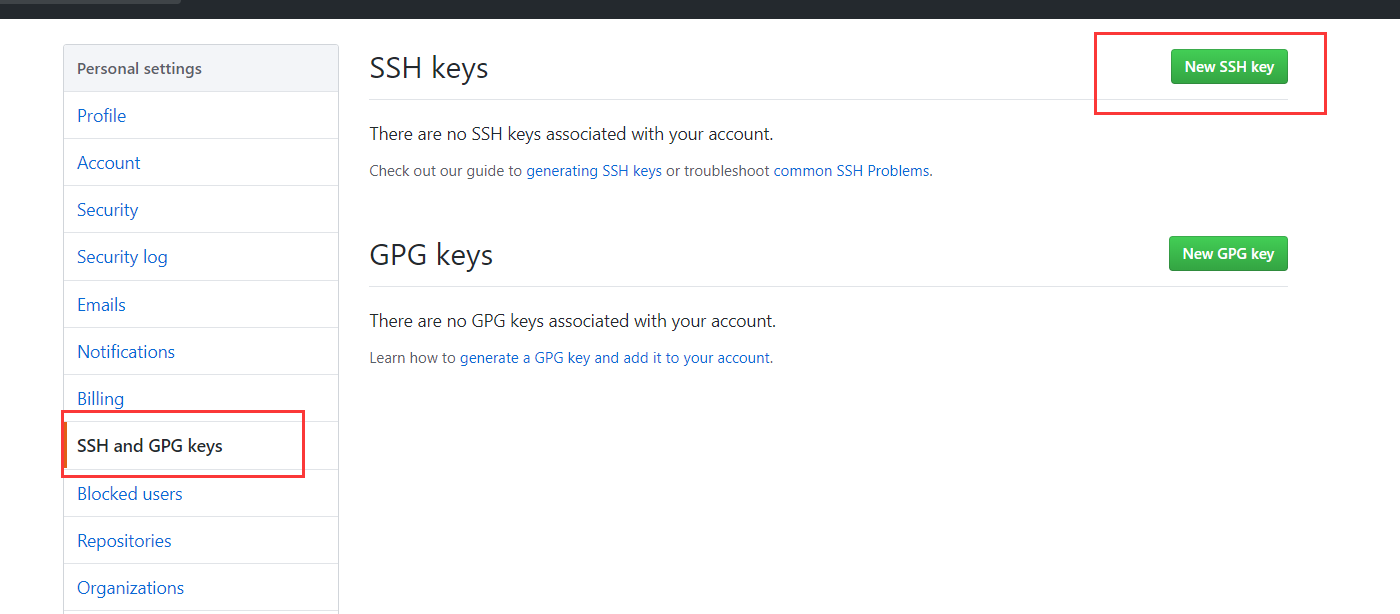


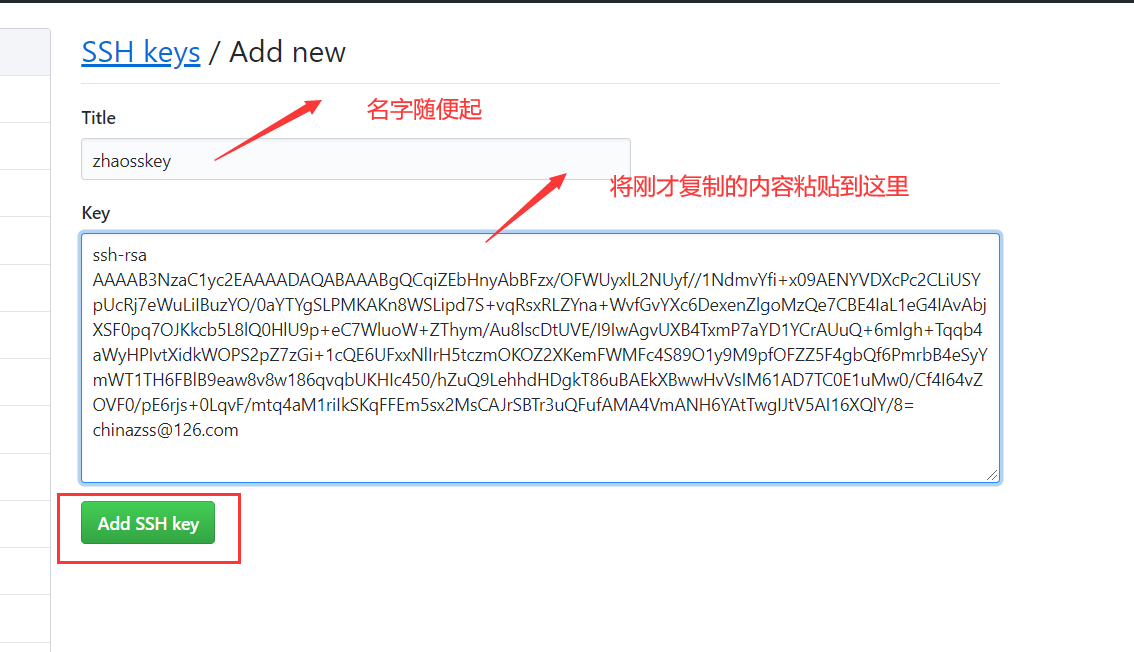
【3】打开id\_rad.pub文件，将里面的内容进行复制操作：

ssh-rsa  chinazss@126.com

【4】打开github账号：

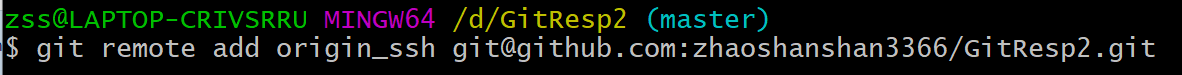




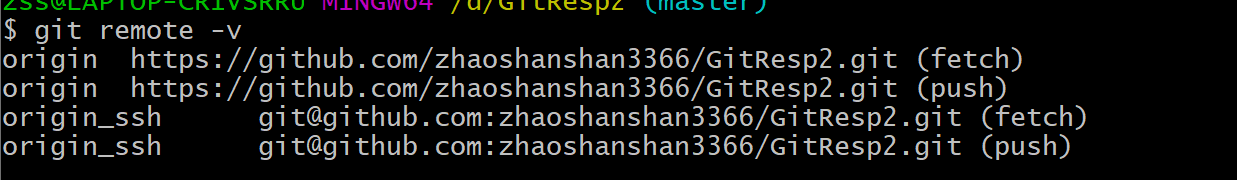


【5】生成密钥以后，就可以正常进行push操作了：

对ssh远程地址起别名：



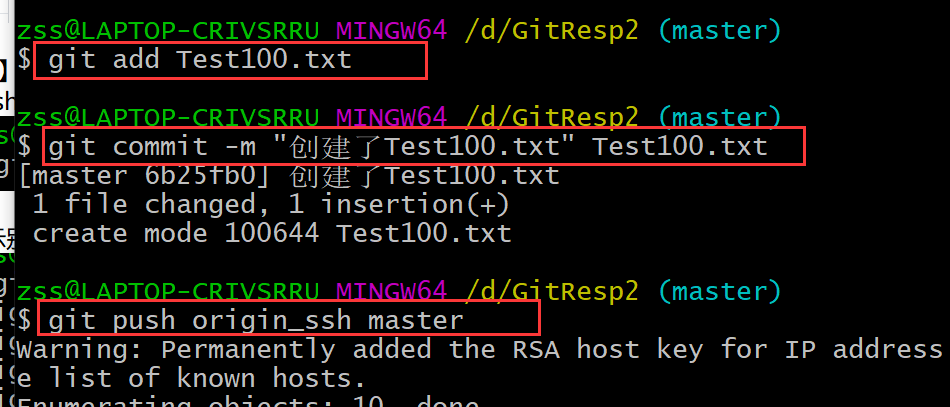
展示别名：



创建一个文件：

IMG_265

添加到暂存区，提交到本地库，然后push到远程库（地址用的是ssh方式的地址）

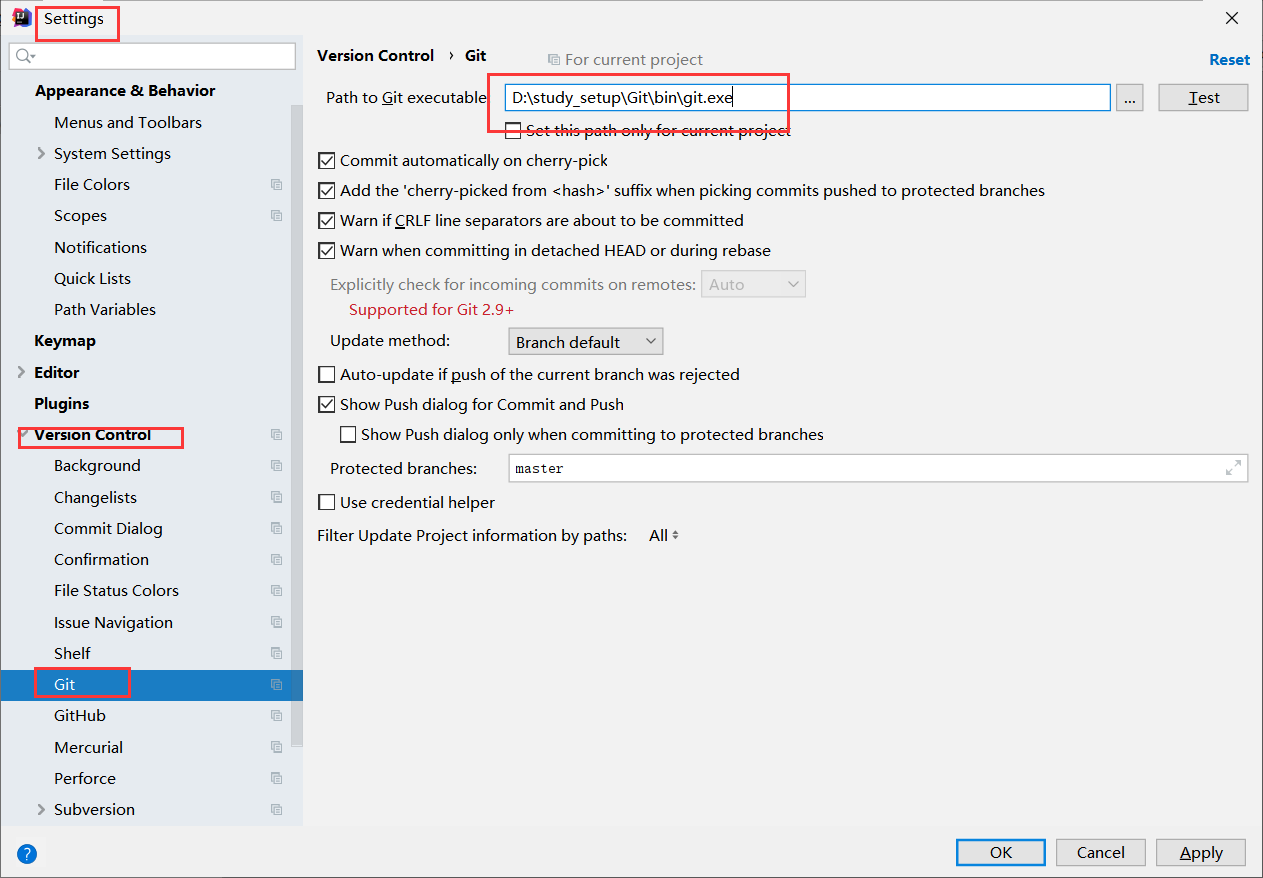


ssh方式好处：   不用每次都进行身份验证

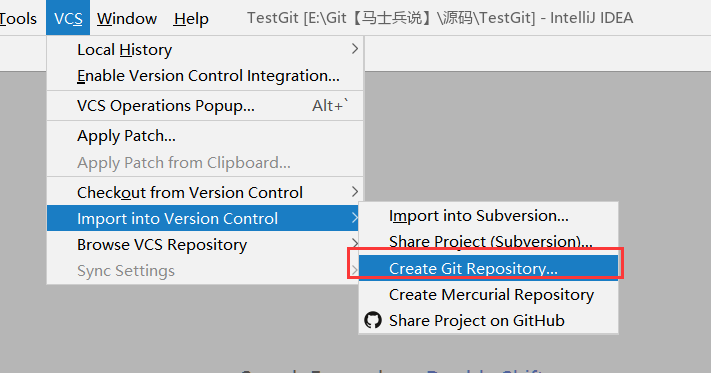
缺陷：只能针对一个账号

## IDEA集成Git

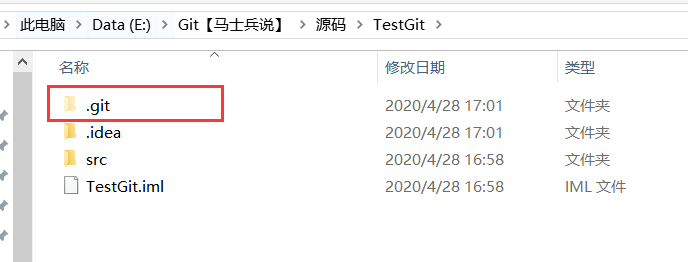
IDEA集成Git：



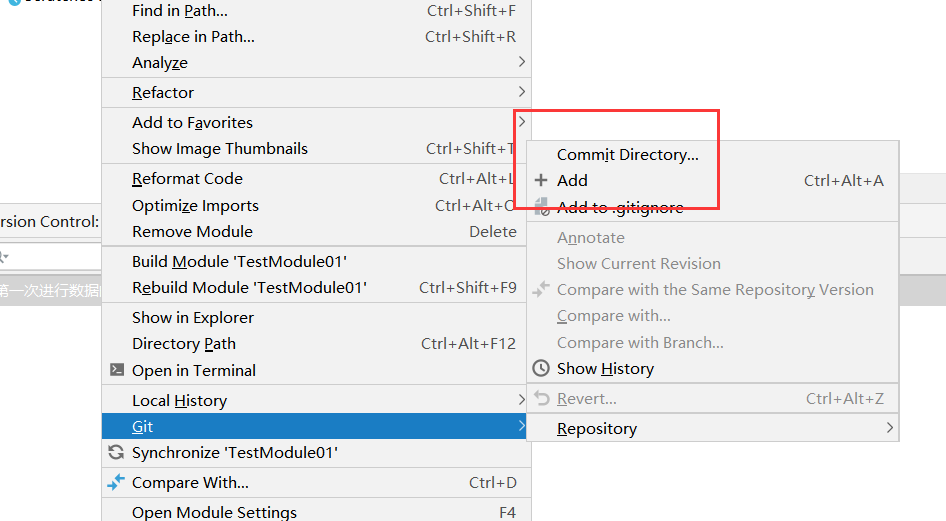
本地库的初始化操作：

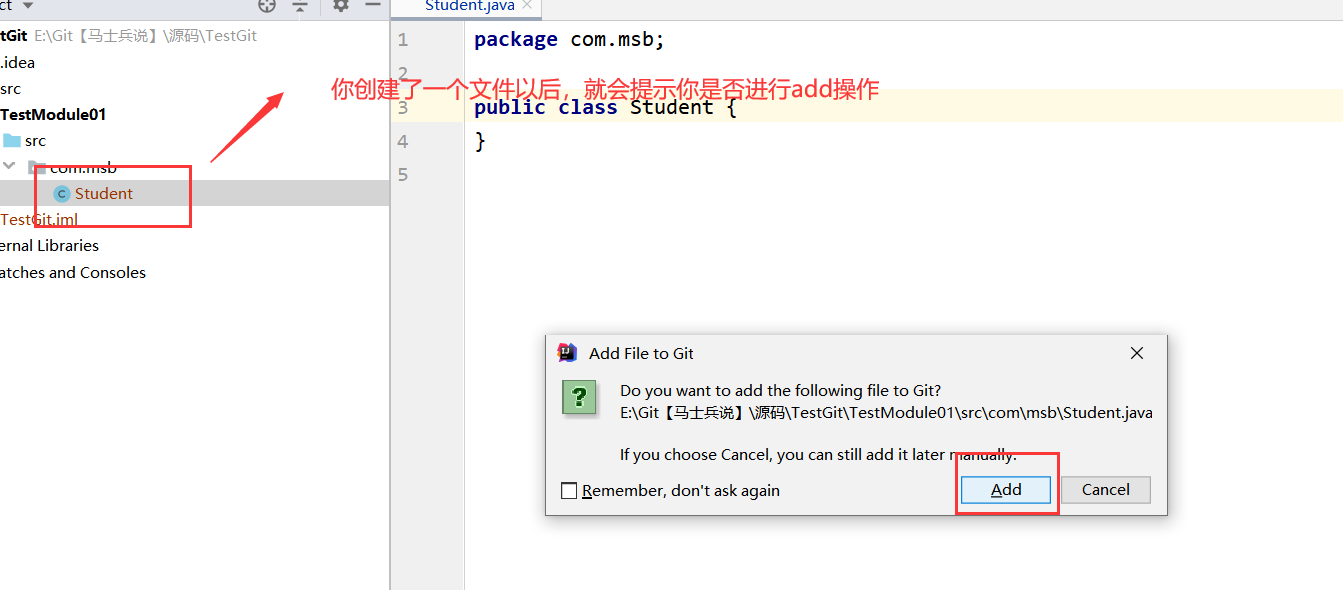


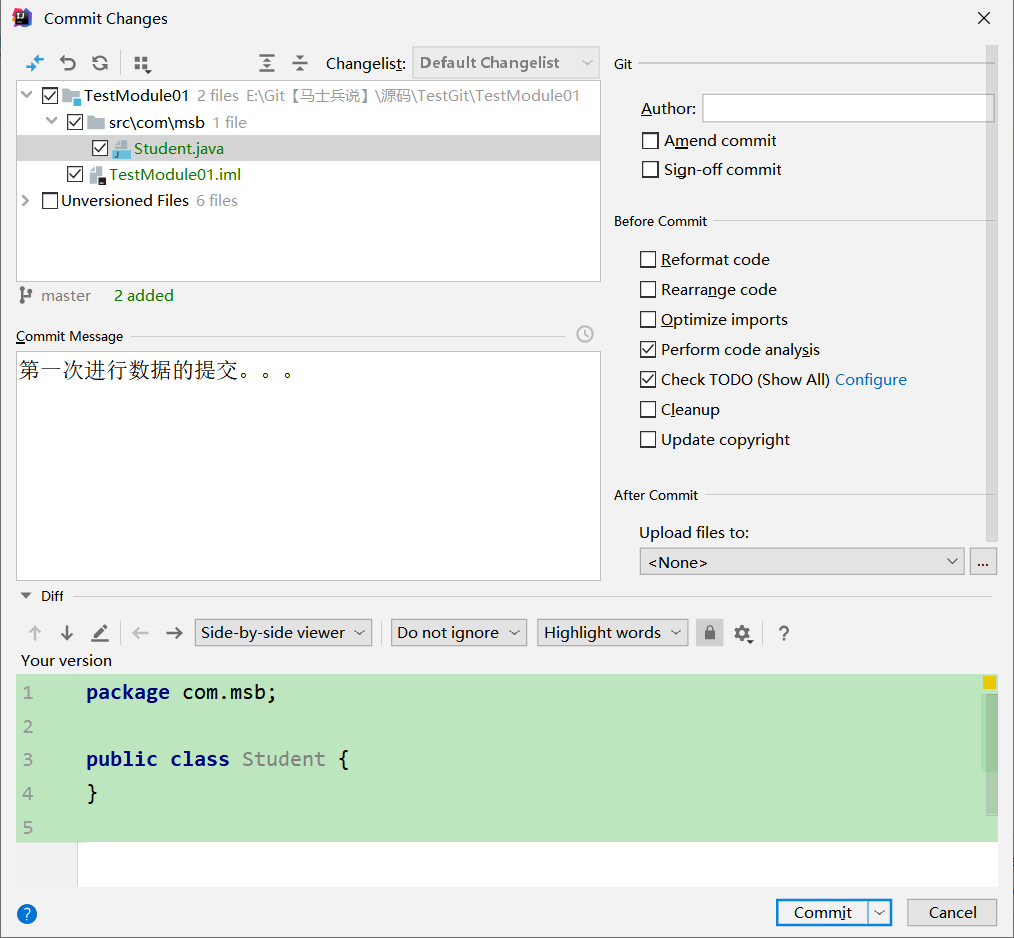
本地库初始化完成了，生成了.git目录：

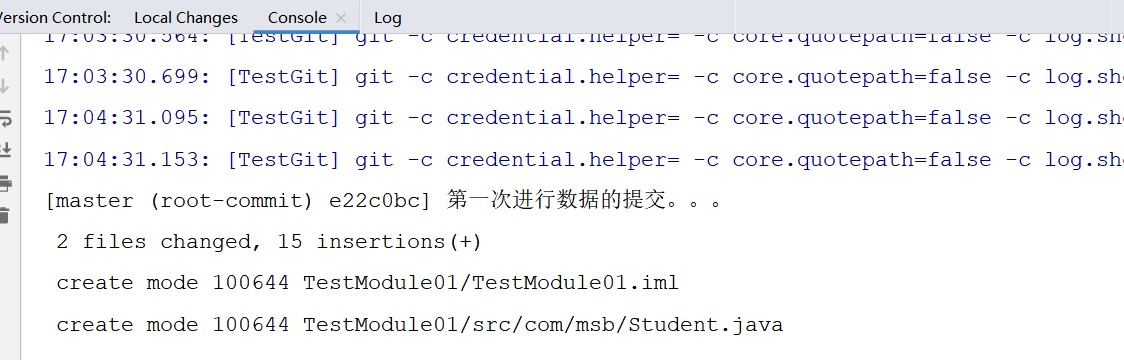


添加到暂存区，再提交到本地库操作；  add +commit:

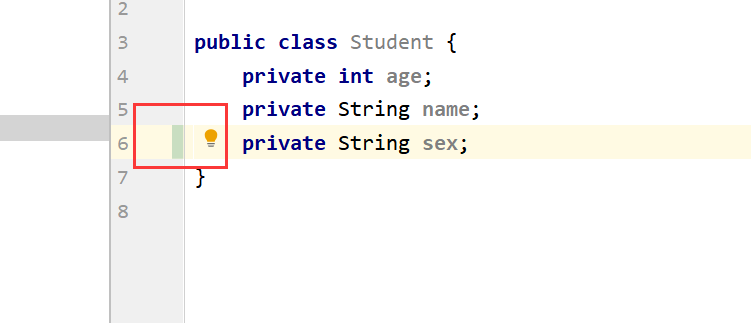








当你更改内容以后，前面跟本地库内容不一致的地方会显示绿色：



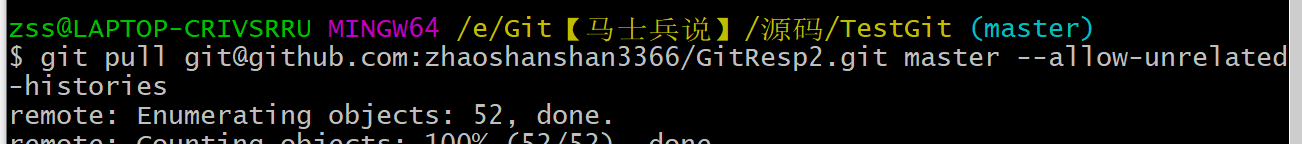
## 本地库和远程库的交互

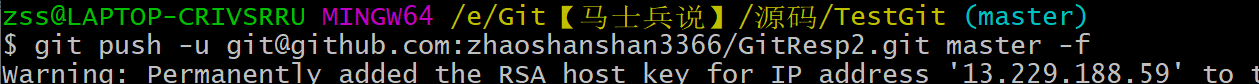
因为他们是两个不同的项目，要把两个不同的项目合并，git需要添加一句代码，在  git pull  之后，这句代码是在git 2.9.2版本发生的，最新的版本需要添加  --allow-unrelated-histories  告诉 git 允许不相关历史合并。

假如我们的源是origin，分支是master，那么我们需要这样写 git pull origin master --allow-unrelated-histories

这个方法只解决因为两个仓库有不同的开始点，也就是两个仓库没有共同的 commit 出现的无法提交。如果使用本文的方法还无法提交，需要看一下是不是发生了冲突，解决冲突再提交

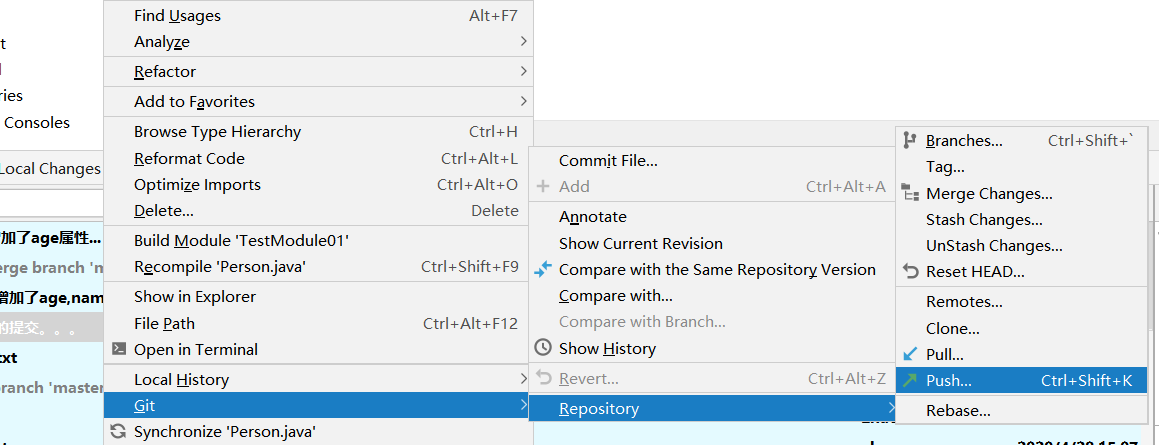
push推送： git push -u origin master -f

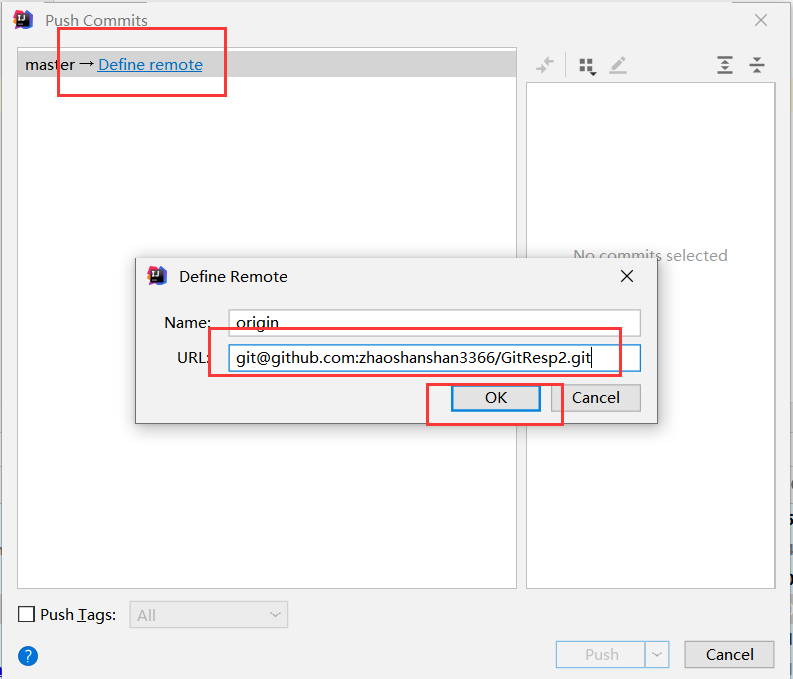


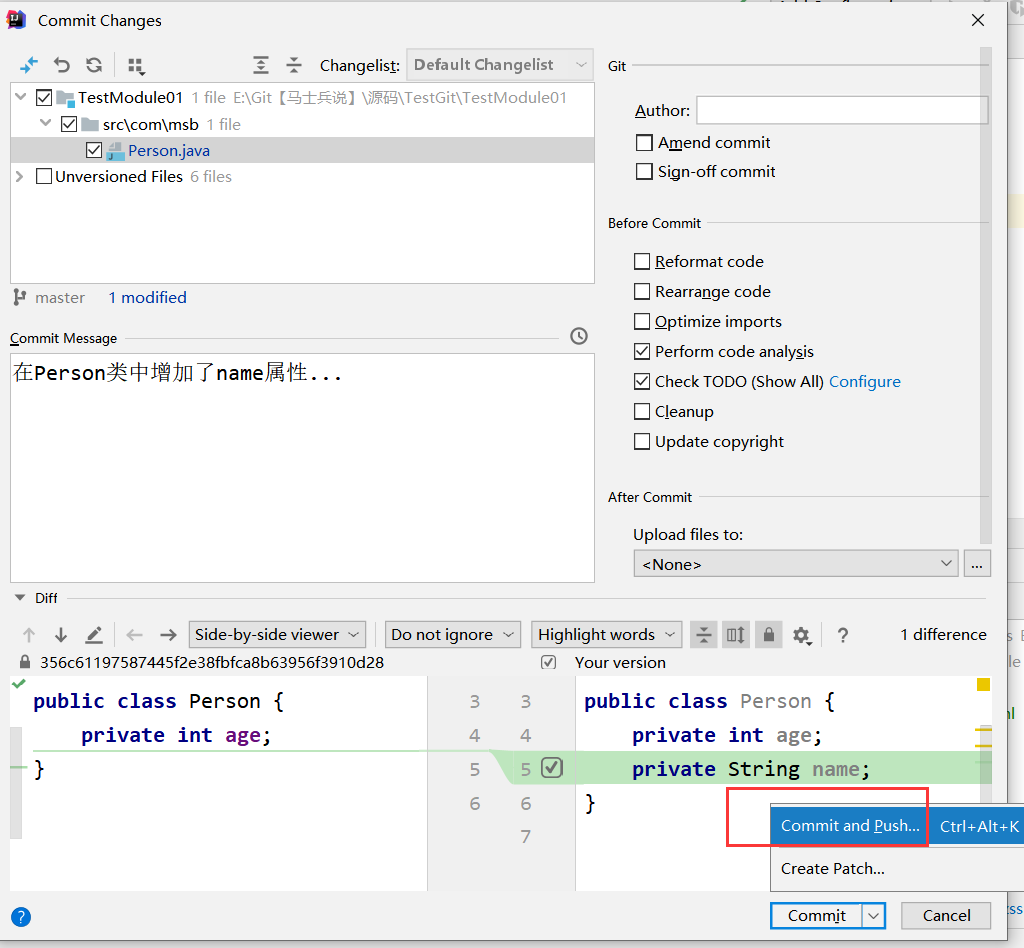


到这里 远程库和本地库就可以进行交互了。

在IDEA中进行推送:



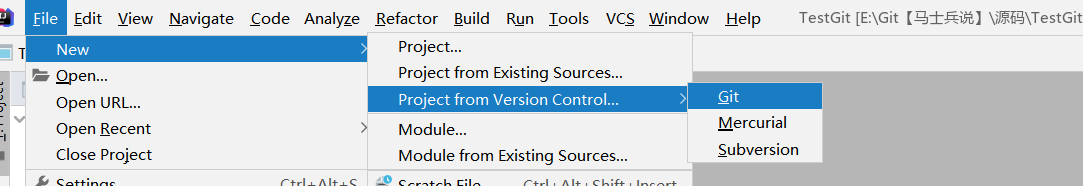


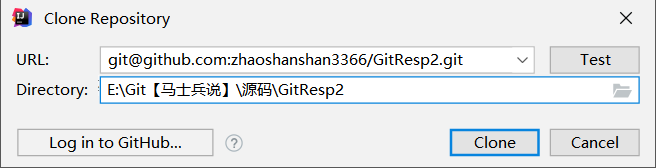


一般在开发中先pull操作，再push操作，不会直接进行push操作！！

## 使用IDEA克隆远程库到本地

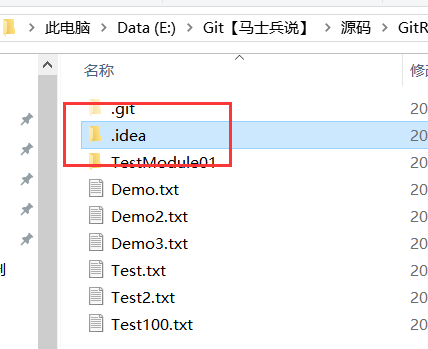
利用IDEA进行克隆项目：





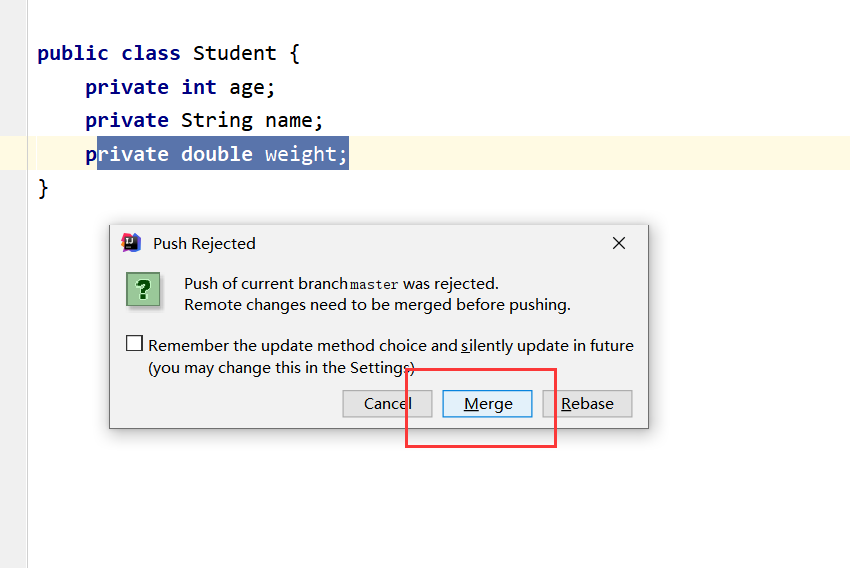
克隆到本地后：

这个目录既变成了一个本地仓库，又变成了工作空间。

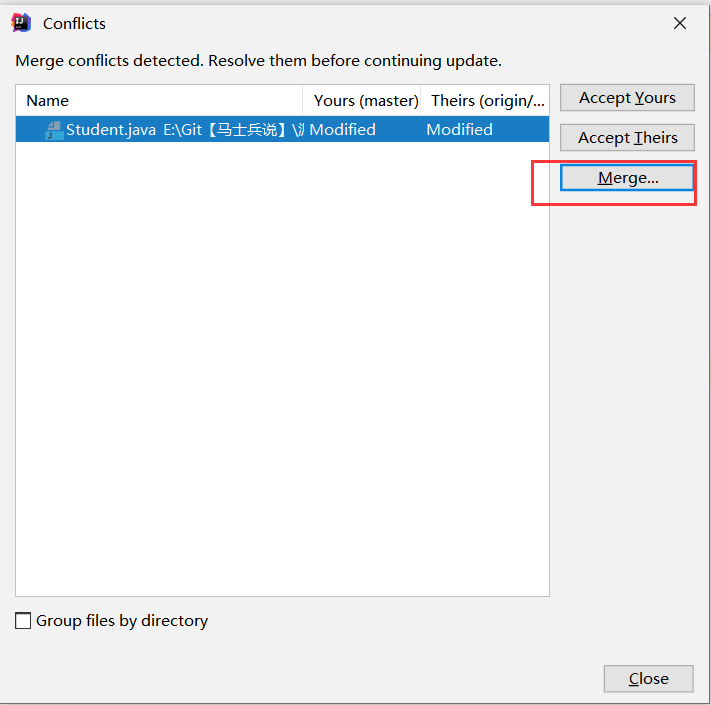


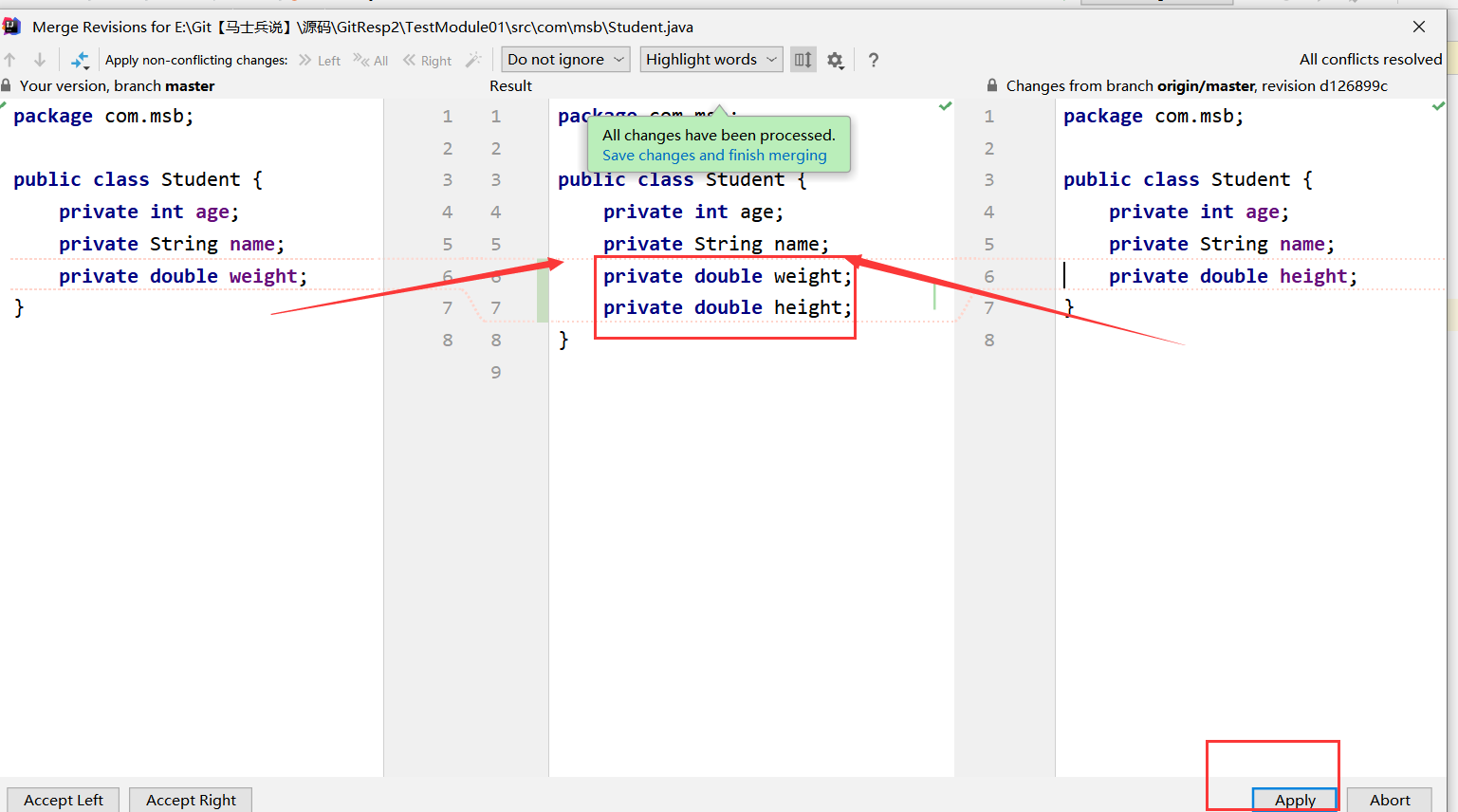
## 解决冲突

【1】在你push以后，有冲突的时候提示合并操作：



合并：





## 如何避免冲突

【1】团队开发的时候避免在一个文件中改代码

【2】在修改一个文件前，在push之前，先pull操作