**Git**

**使**

**用**

**说**

**明**

**书**

目录

[一、什么是Git什么是GitHub 2](#_Toc2854390)

[二、GitHub的操作说明 2](#_Toc2854391)

[2.1 申请GitHub账号 2](#_Toc2854392)

[2.2 创建组织Wlonglong123 3](#_Toc2854393)

[2.3 创建组织仓库 Wllgit 4](#_Toc2854394)

[2.4 邀请组织成员 5](#_Toc2854395)

[三、Git的安装与配置（刘磊） 11](#_Toc2854396)

[3.1 Git的安装 11](#_Toc2854397)

[3.2 配置用户名和邮箱 12](#_Toc2854398)

[3.3 创建版本库 13](#_Toc2854399)

[四、Git的操作说明 15](#_Toc2854400)

[4.1 版本回退（付廷松） 15](#_Toc2854401)

[4.2 Git撤销修改（张凤） 18](#_Toc2854402)

[4.3 删除文件（张凤） 20](#_Toc2854403)

[五、远程仓库（黄浩） 22](#_Toc2854404)

[5.1 生成密钥对 22](#_Toc2854405)

[5.2 关联本地仓库和远程仓库 23](#_Toc2854406)

[5.3 克隆一个本地仓库 25](#_Toc2854407)

[六、创建与合并分支（万隆隆） 26](#_Toc2854408)

[6.1 创建分支 26](#_Toc2854409)

[6.2 合并分支 27](#_Toc2854410)

[6.3 删除分支 28](#_Toc2854411)

[6.4 解决冲突 28](#_Toc2854412)

[6.5 分支策略管理 32](#_Toc2854413)

# 一、什么是Git什么是GitHub

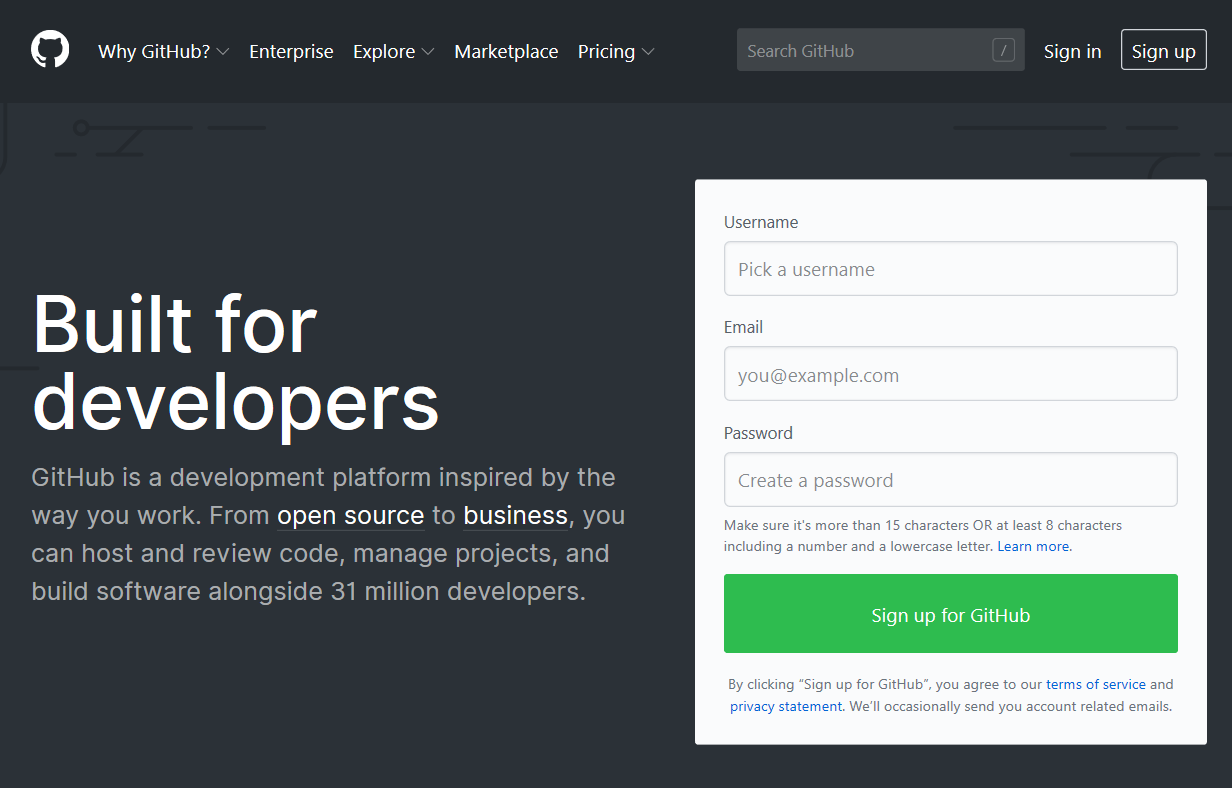
Git是一个版本管理工具，是可以在你电脑不联网的情况下，只在本地使用的一个版本管理工具，其作用就是可以让你更好的管理你的程序，比如你原来提交过的内容，以后虽然修改了，但是通过Git这个工具，可以把你原来提交的内容重现出来，这样对于你后来才意识到的一些错误的更改，可以进行还原。   
 GitHub这是一个网站，就是每个程序员自己写的程序，可以在GitHub上建立一个网上的仓库，你每次提交的时候可以把代码提交到网上，这样你的每次提交，别人也都可以看到你的代码，同时别人也可以帮你修改你的代码，这种开源的方式非常方便程序员之间的交流和学习。

总结来说，git是一个软件，可在本地建立仓库，能够帮你更好的写程序，github则是一个网站，一个网上仓库，这个网站可以帮助程序员之间互相交流和学习。

# 二、GitHub的操作说明

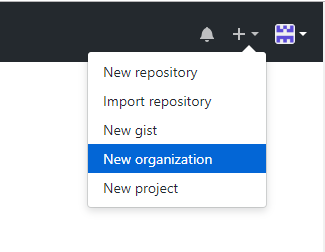
## 2.1 申请GitHub账号（付廷松）

地址：<https://github.com/>



可以仅仅注册一个账号即可，不用勾选其他服务（最后还有一个验证邮箱地址）。

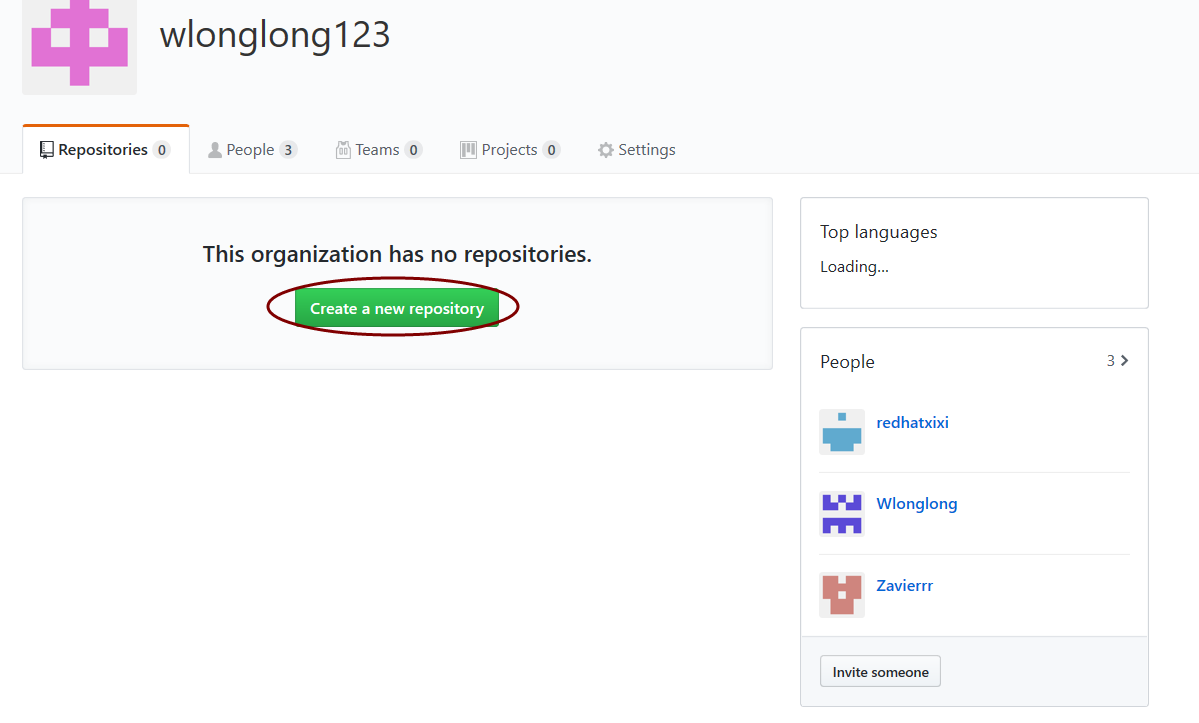
## 2.2 创建组织Wlonglong123（付廷松）

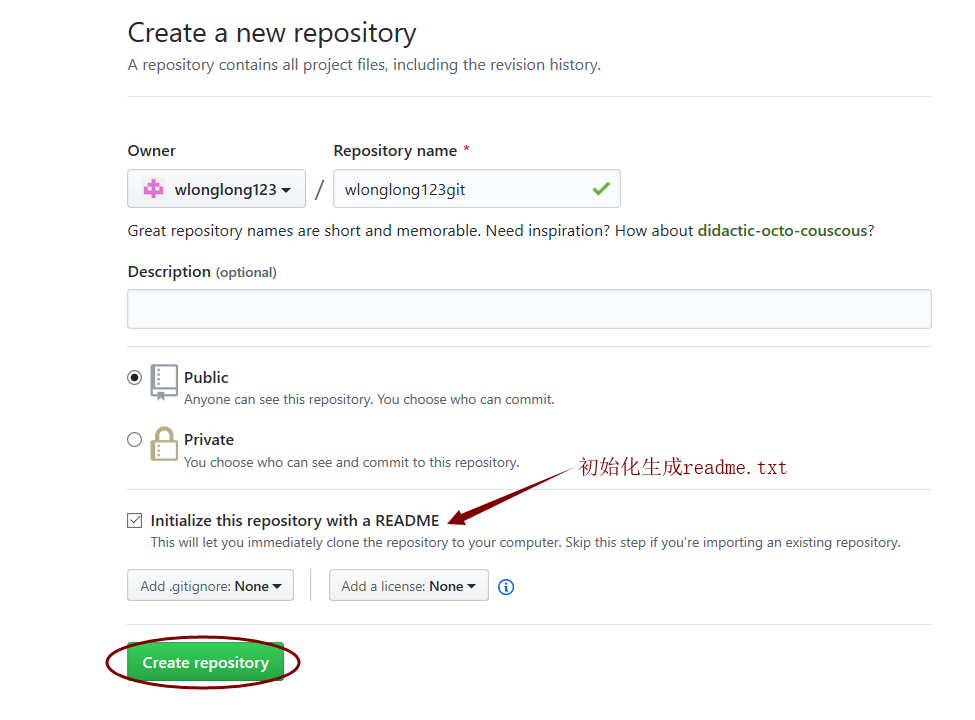


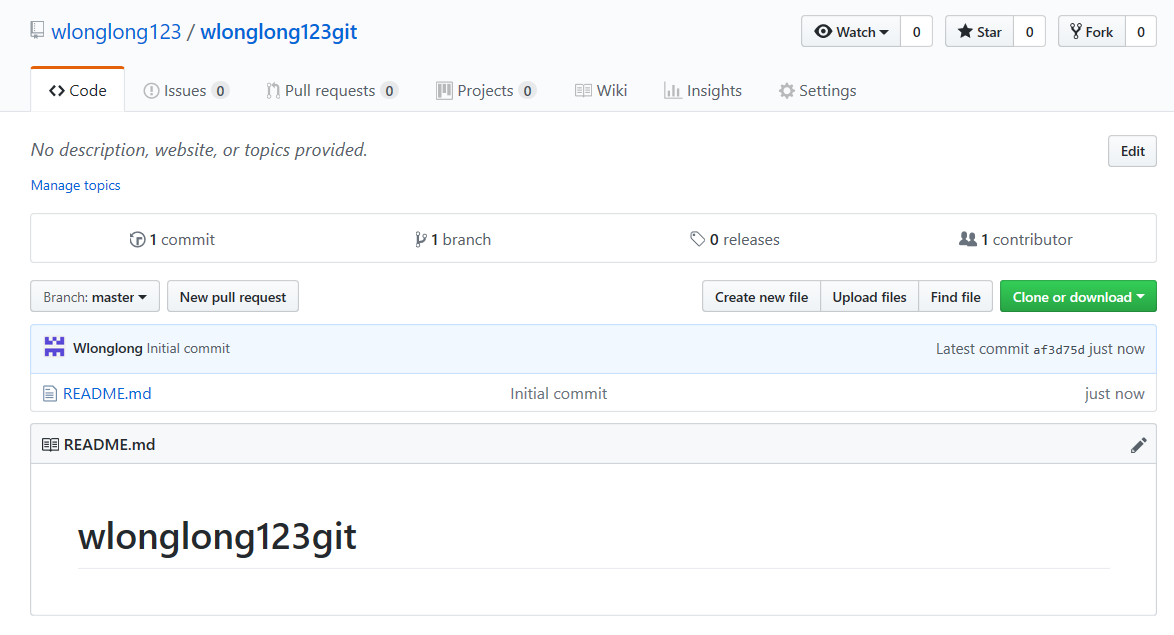


邀请组织成员后面再做，组织细节可跳过

## 2.3 创建组织仓库 Wllgit（张凤）

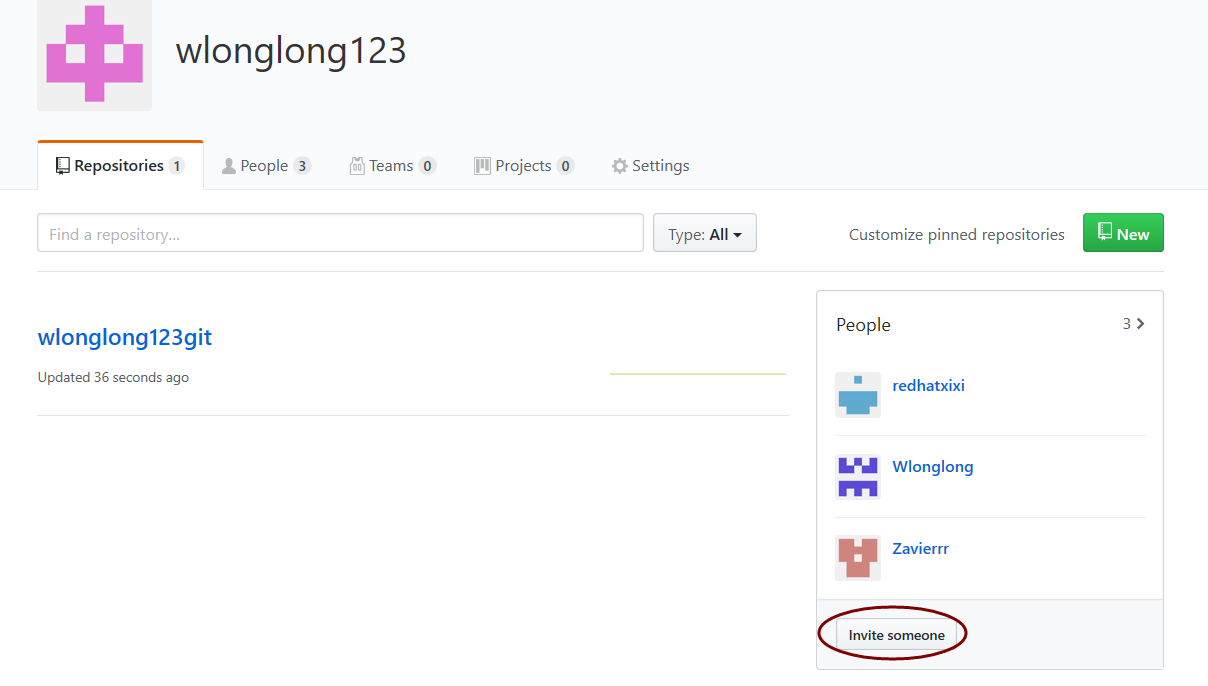


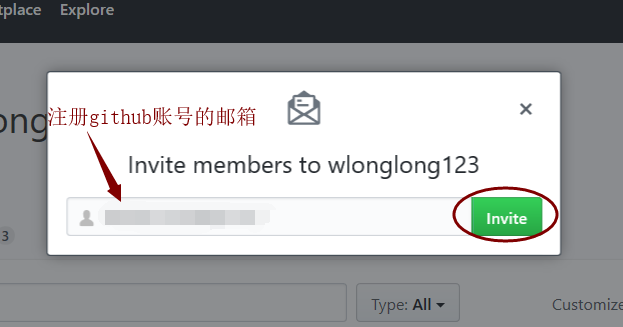


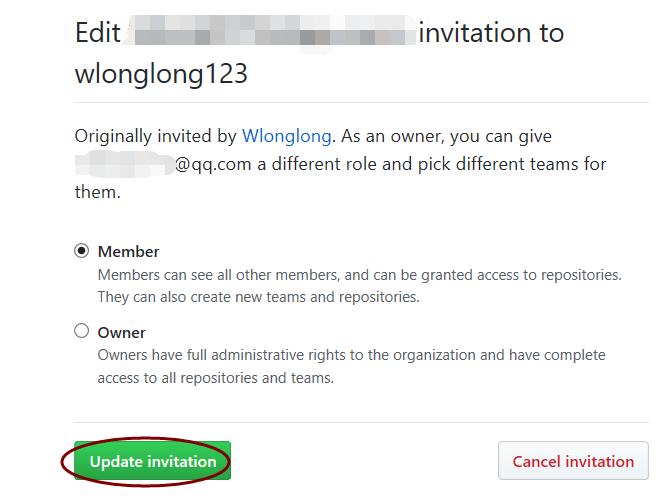


## 2.4 邀请组织成员（张凤）

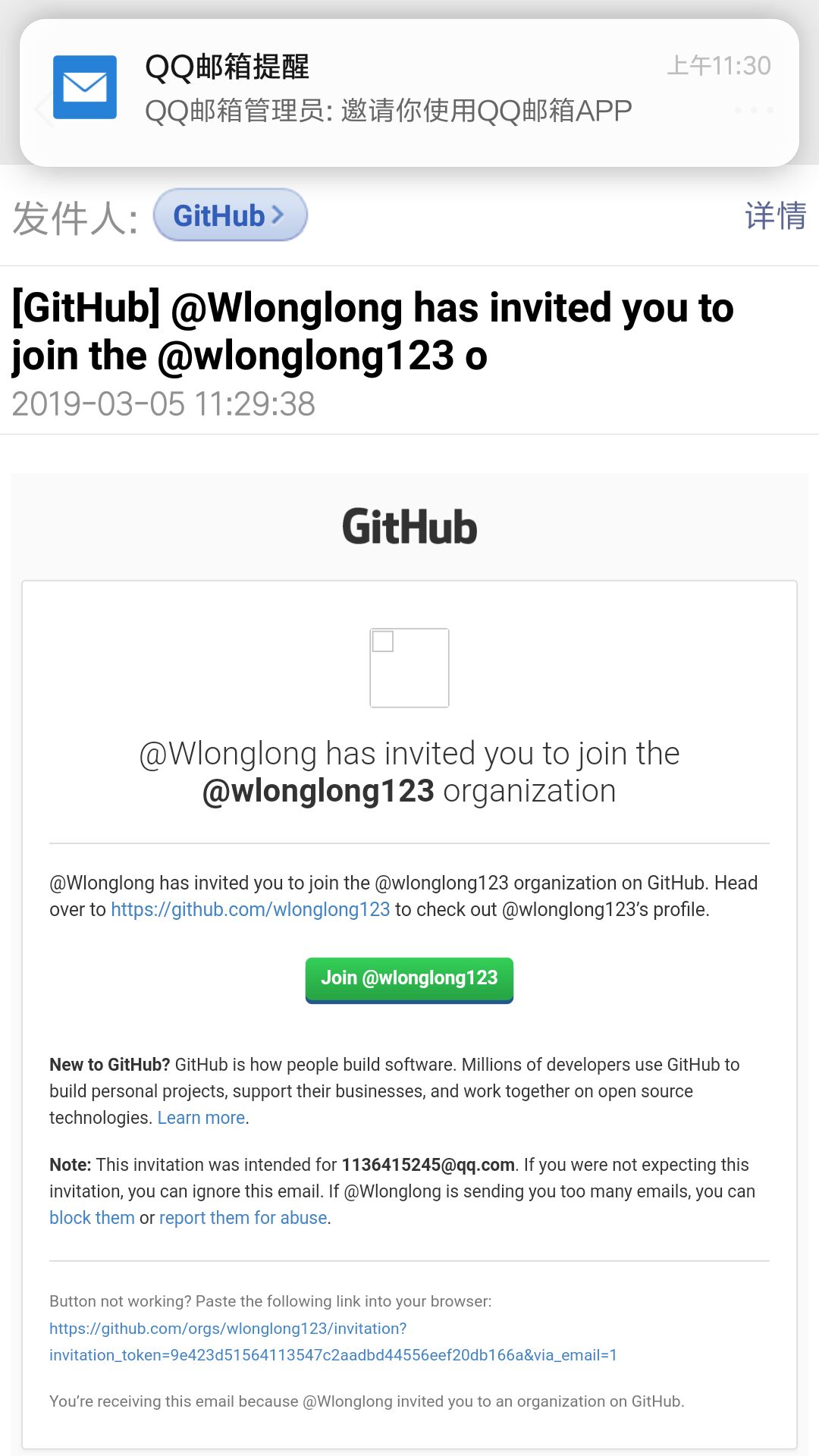
进入组织

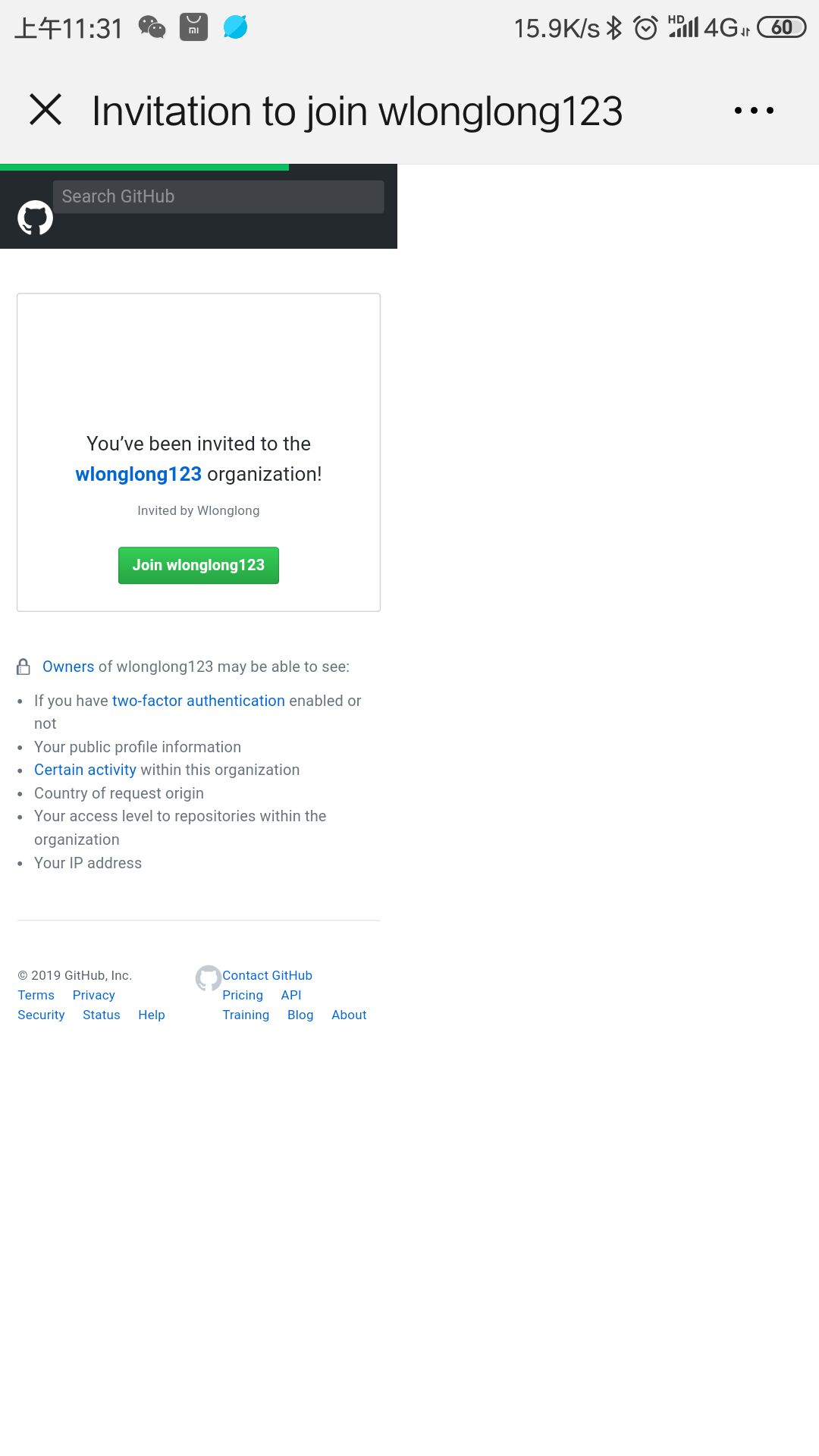


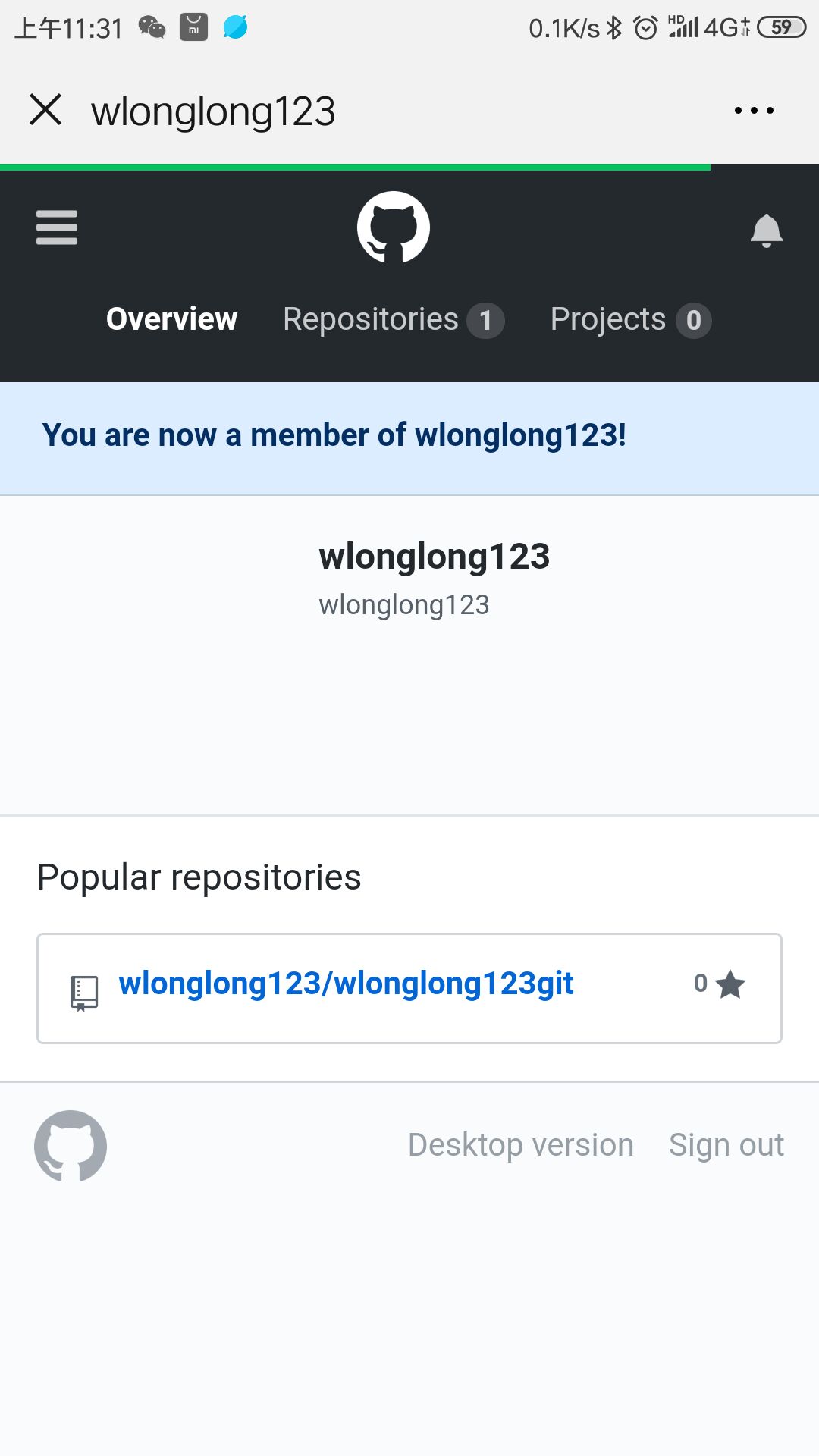


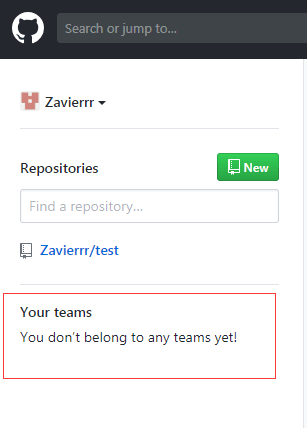


验证并同意加入（点两次joiin即可）





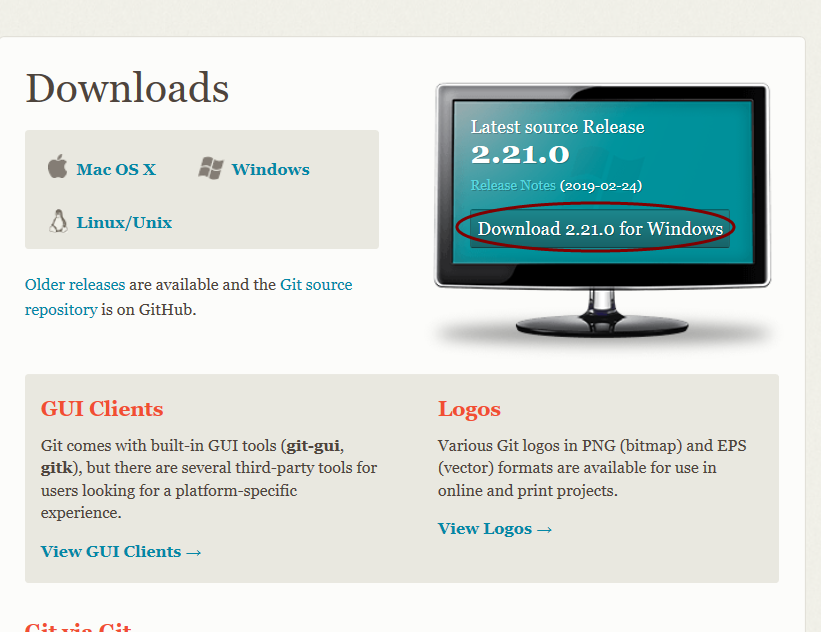




# 三、Git的安装与配置（刘磊）

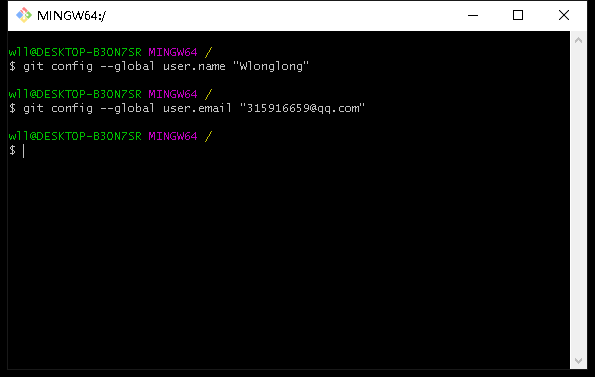
## 3.1 Git的安装

下载地址：<https://git-scm.com/downloads>



安装是傻瓜式安装，一直下一步即可（注意：可以选择安装目录）。

## 3.2 配置用户名和邮箱

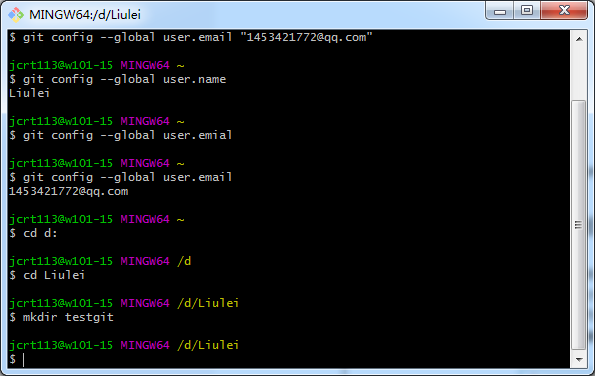


## 3.3 创建版本库

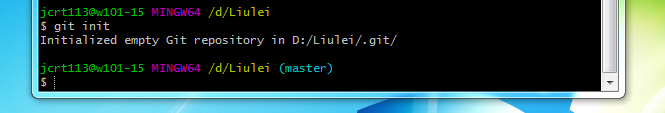
版本库简单理解其实就是一个目录，在这个目录里面的所有文件都可以被Git管理起来，每个文件的修改、删除，Git都能跟踪，以便任何时刻都可以追踪历史，在将来某个时刻可以“还原”成历史时刻中的随意一个版本。

首先，打开Git Bash，在配置好name和email标识后，选择一个合适的路径（避免中文和空格的出现），创建一个空目录，进入该目录

第一步，新建一个文件夹Liulei，再使用命令$ mkdir testgit来创建一个版本库



 第二步，初始化版本库，通过git init命令把这个目录初始化成一个Git仓库

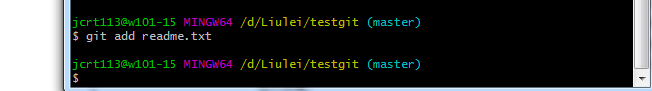


初始化后，在该目录下会多一个.git文件夹，这个文件夹是用来管理当前这个git仓库的，一定要避免修改这个文件夹，它是一个隐藏文件夹，也可以用ls -ah让它显示。

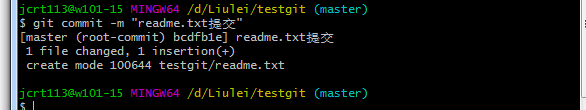
文件添加版本库

版本控制的版本跟踪，只能跟踪文本文件的改动，比如TXT文件，网页，所有的程序代码等等，Git也不例外。版本控制系统可以告诉你每次的改动，比如在第2行加了一行数字“22222222”.而图片、视频这些二进制文件，版本控制系统无法跟踪文件的变化，只能把二进制文件每次改动串起来，也就是只知道图片从100KB改成了120KB，但到底改了啥，版本控制系统不知道，也没法知道。

第一步，在仓库里创建一个read.txt，然后通过下面的命令将这个添加到本地仓库中，也是就让git记录下添加的这个修改动作。

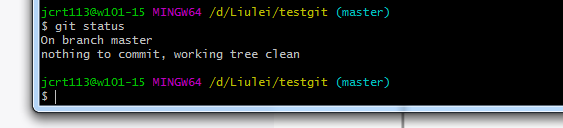


  第二步，用命令git commit告诉Git，把文件提交到仓库



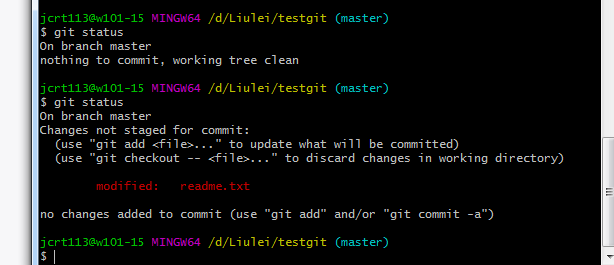
git commit <文件名> -m “<提交是添加的注释>”

如果进入vim按I进行输入，按Esc退出输入模式，然后输入：wq（或者其他）命令退出vim编辑器。

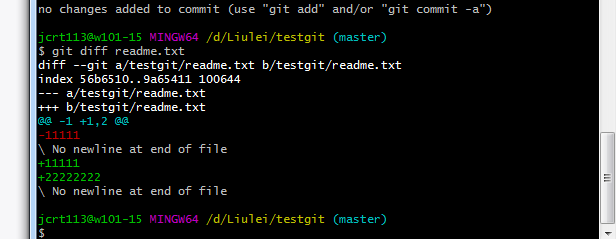


为什么Git添加文件需要add，commit一共两步呢？因为commit可以一次提交很多文件，所以你可以多次add不同的文件

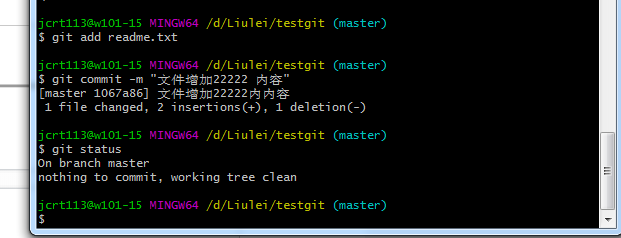
在对仓库内的文件进行了修改，但是还未执行添加和提交命令时，git status命令可以让我们时刻掌握仓库当前的状态。



从上方的红字 modified： read.txt  告诉我们readme.txt文件进行了修改，如果我们记不住修改了什么，可以通过git diff 查看修改了哪个部分。



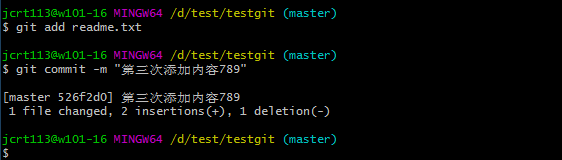
提交修改之后的文件



# 四、Git的操作说明

## 4.1 版本回退（付廷松）

现在我继续对readme.txt文件进行修改，再增加一行内容为789.继续执行命令如下：



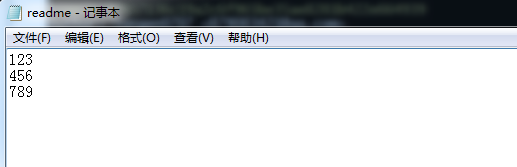
现在我已经对readme.txt文件做了三次修改了，那么我现在想查看下历史记录，如何查呢？我们现在可以使用命令 git log 演示如下所示：



git log命令显示从最近到最远的显示日志，我们可以看到最近三次提交，最近的一次是,增加内容为789.上一次是添加内容456，第一次默认是123.如果嫌上面显示的信息太多的话，我们可以使用命令 git log –pretty=oneline 演示如下：



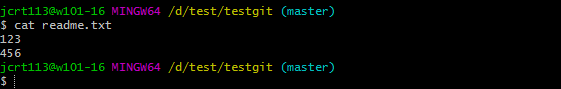
现在我想使用版本回退操作，我想把当前的版本回退到上一个版本，要使用什么命令呢？可以使用如下2种命令，第一种是：git reset  –hard HEAD^ 那么如果要回退到上上个版本只需把HEAD^ 改成 HEAD^^ [以此类推](https://www.baidu.com/s?wd=%E4%BB%A5%E6%AD%A4%E7%B1%BB%E6%8E%A8&tn=24004469_oem_dg&rsv_dl=gh_pl_sl_csd)。那如果要回退到前100个版本的话，使用上面的方法肯定不方便，我们可以使用下面的简便命令操作：git reset  –hard HEAD~100 即可。未回退之前的readme.txt内容如下：



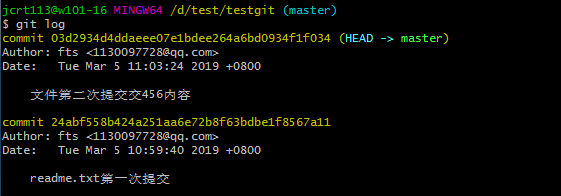
如果想回退到上一个版本的命令如下操作：



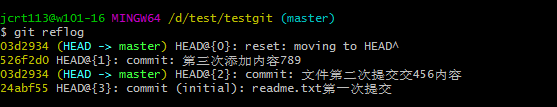
再来查看下 readme.txt内容如下：通过命令cat readme.txt查看：



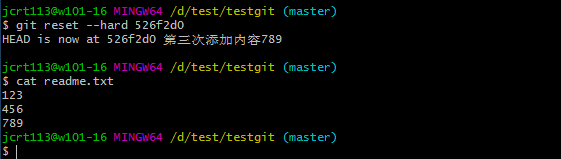
可以看到，内容已经回退到上一个版本了。我们可以继续使用git log 来查看下历史记录信息，如下：



我们看到 增加789 内容我们没有看到了，但是现在我想回退到最新的版本，如：有789的内容要如何恢复呢？我们可以通过版本号回退，使用命令方法如下：git reset  –-hard 版本号 ，但是现在的问题假如我已经关掉过一次命令行或者789内容的版本号我并不知道呢？要如何知道增加789内容的版本号呢？可以通过如下命令即可获取到版本号：git reflog 。演示如下：

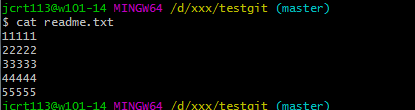


通过上面的显示我们可以知道，增加内容789的版本号是526f2d0我们现在可以命令git reset  –-hard 526f2d0来恢复了。恢复后用命令cat readme.txt查看。演示如下：



## 4.2 Git撤销修改（张凤）

比如我现在在readme.txt文件里面增加一行 内容为55555，我们先通过命令查看如下：

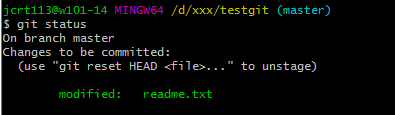


在我未提交之前，我发现添加55555内容有误，所以我得马上恢复以前的版本，现在我可以有如下几种方法可以做修改：

第一：如果我知道要删掉那些内容的话，直接手动更改去掉那些需要的文件，然后add添加到暂存区，最后commit掉。

第二：我可以按以前的方法直接恢复到上一个版本。使用 git reset --hard HEAD^

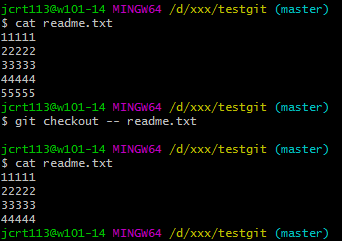
但是现在我不想使用上面的2种方法，我想直接想使用撤销命令该如何操作呢？首先在做撤销之前，我们可以先用 git status 查看下当前的状态。如下所示：



可以发现，Git会告诉你，git checkout -- file 可以丢弃工作区的修改。

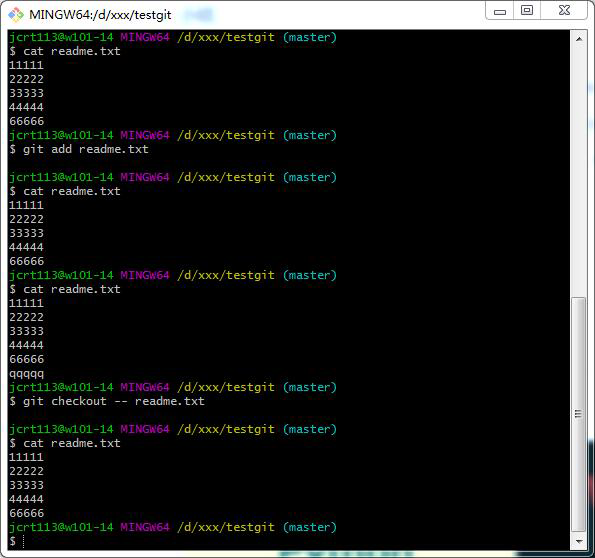
命令git checkout -- readme.txt意思就是，把readme.txt文件在工作区的修改全部撤销，这里有两种情况：

一种是readme.txt自修改后还没有被放到暂存区，现在，撤销修改就回到和版本库一模一样的状态；



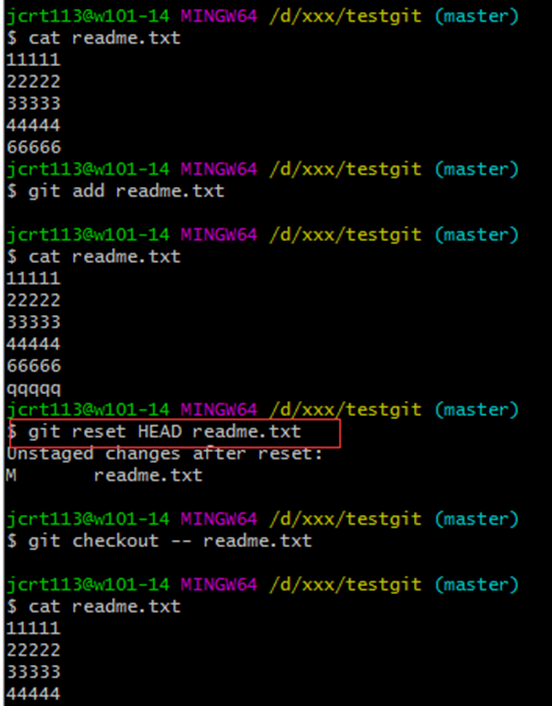
一种是readme.txt已经添加到暂存区后，又作了修改，现在，撤销修改就回到添加到暂存区后的状态。

对于第二种情况，假如现在我对readme.txt添加一行 内容为66666，我git add 增加到暂存区后，接着添加内容qqqqq，我想通过撤销命令让其回到暂存区后的状态。如下所示：



发现只是让这个文件回到最近一次git commit或git add时的状态。也就是只恢复已存入缓存区之后的修改。

所以当你不但改乱了工作区某个文件的内容，还添加到了暂存区时，想全部丢弃修改，分两步，第一步用命令git reset HEAD <file>，第二步再用命令git checkout – file。如下所示：



**小结**

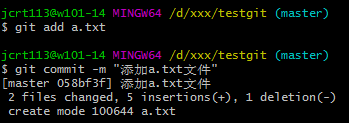
场景1：当你改乱了工作区某个文件的内容，想直接丢弃工作区的修改时，用命令git checkout -- file。

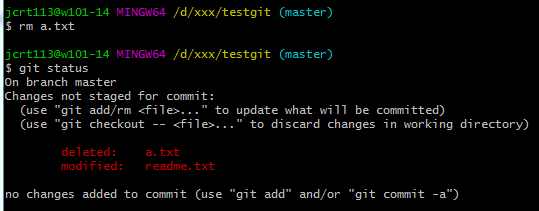
场景2：当你不但改乱了工作区某个文件的内容，还添加到了暂存区时，想丢弃修改，分两步，第一步用命令git reset HEAD <file>，就回到了场景1，第二步按场景1操作。

场景3：已经提交了不合适的修改到版本库时，想要撤销本次提交，参考版本回退一节，不过前提是没有推送到远程库。

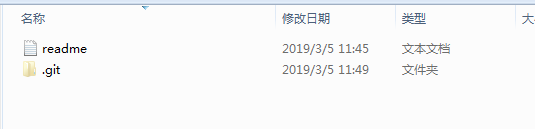
## 4.3 删除文件（张凤）

假如我现在版本库testgit目录添加一个文件a.txt,然后提交。如下：



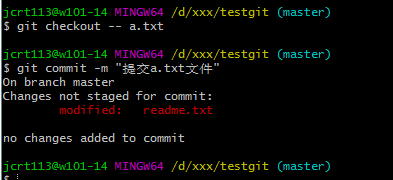


如上：一般情况下，可以直接在文件目录中把文件删了，或者使用如上rm命令：rm b.txt ，如果我想彻底从版本库中删掉了此文件的话，可以再执行commit命令 提交掉，现在目录是这样的，



只要没有commit之前，如果我想在版本库中恢复此文件如何操作呢？

可以使用如下命令 git checkout --a.txt，如下所示：



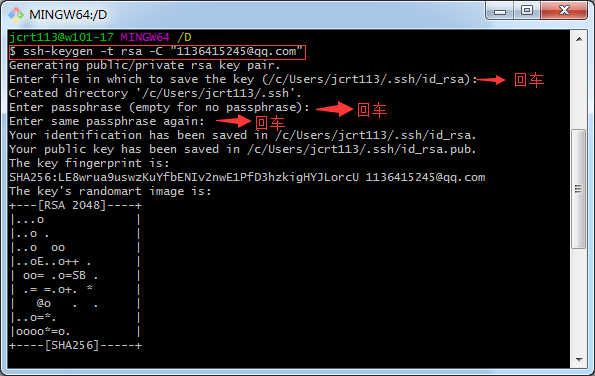
再来看看我们testgit目录，已经添加文件了。如下所示：

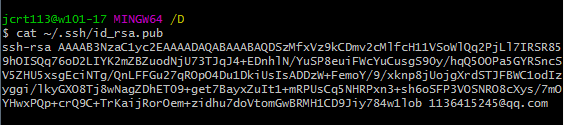


# 五、远程仓库（黄浩）

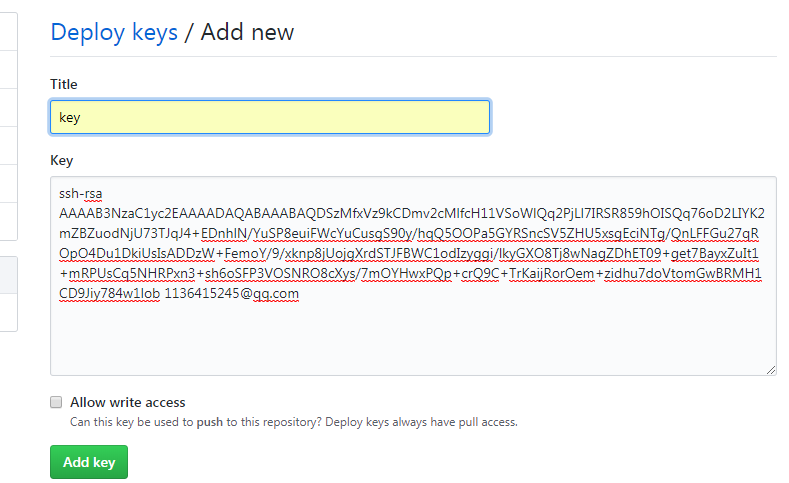
## 5.1 生成密钥对

输入$ ssh-keygen -t rsa -C "your\_email@youremail.com"， 其中邮箱地址填之前配置的邮箱

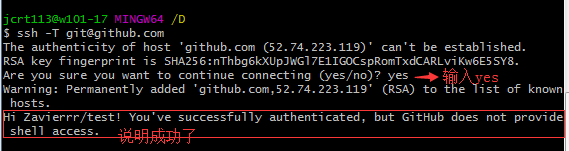
查看生成的密钥对，输入$ cat ~/.ssh/id\_rsa.pub



然后你复制上面的公钥内容，粘贴到Github🡪Settings🡪Deploy keys🡪add deploy keys内的“Key”文本域内；而title域，则是给密钥起个名字。然后点击add key。

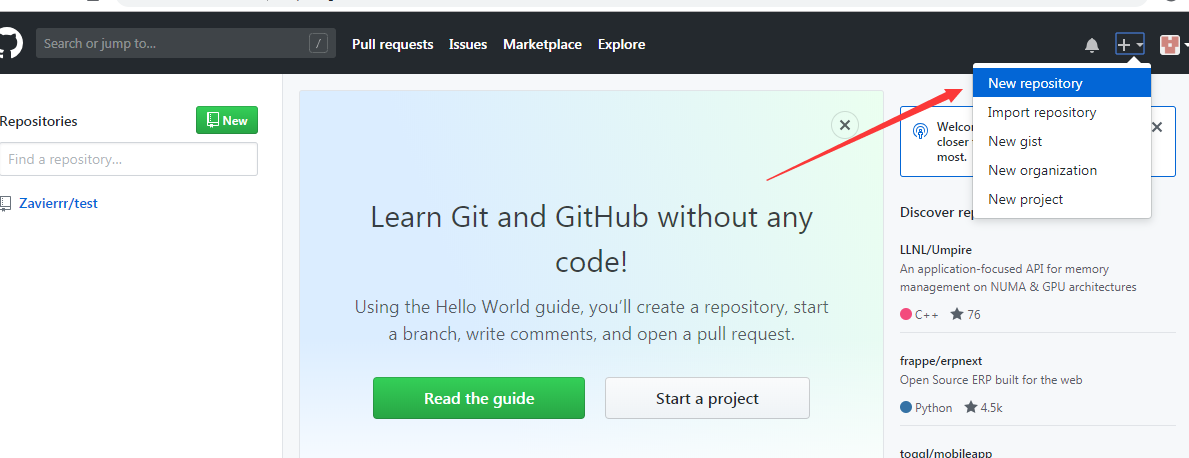


完成以后，验证下这个key是不是正常工作，输入$ ssh -T git@github.com

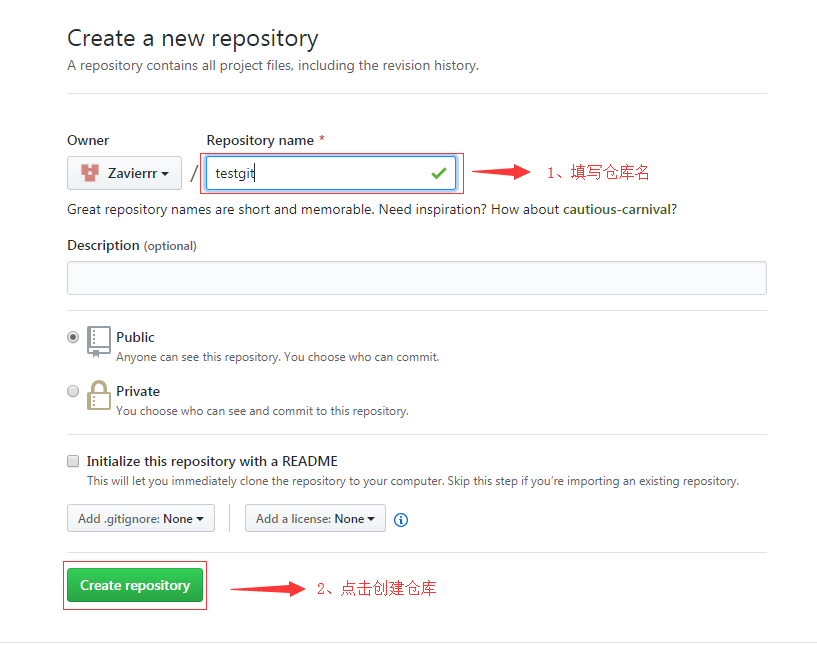


## 5.2 关联本地仓库和远程仓库

首先，找到New repository按钮，点击创建远程仓库，如图



然后给仓库起个名字，根据自己意愿可加些对仓库的描述，最后点击创建仓库按钮，如图

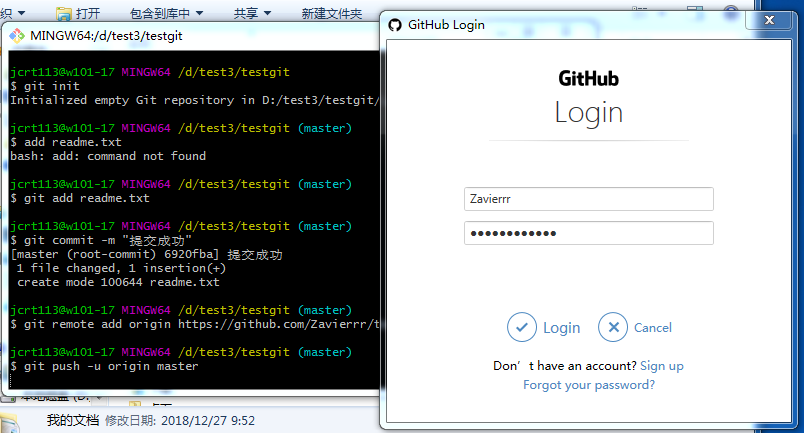


现在的情景是：我们已经在本地创建了一个Git仓库test3后，又想在github创建一个Git仓库test3，并且希望这两个仓库进行远程同步，这样github的仓库可以作为备份，又可以其他人通过该仓库来协作

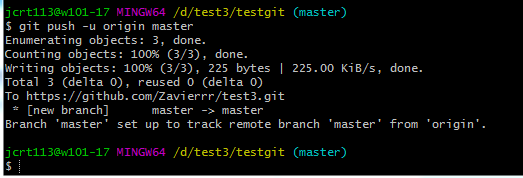
使用$ git remote add origin https://github.com/Zavierrr/test3.git

以及$ git push -u origin master

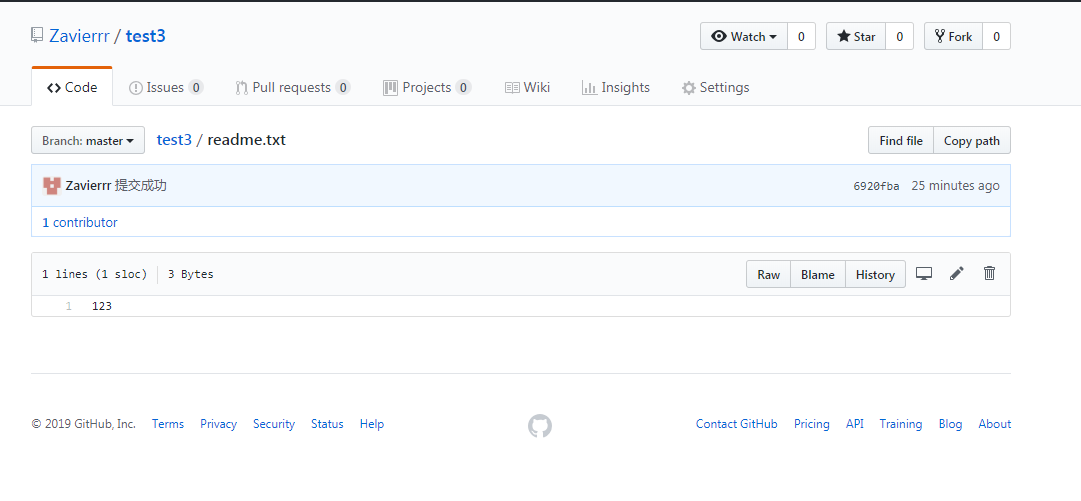
结果如图



登陆成功后显示如下，则说明两个仓库已经关联了

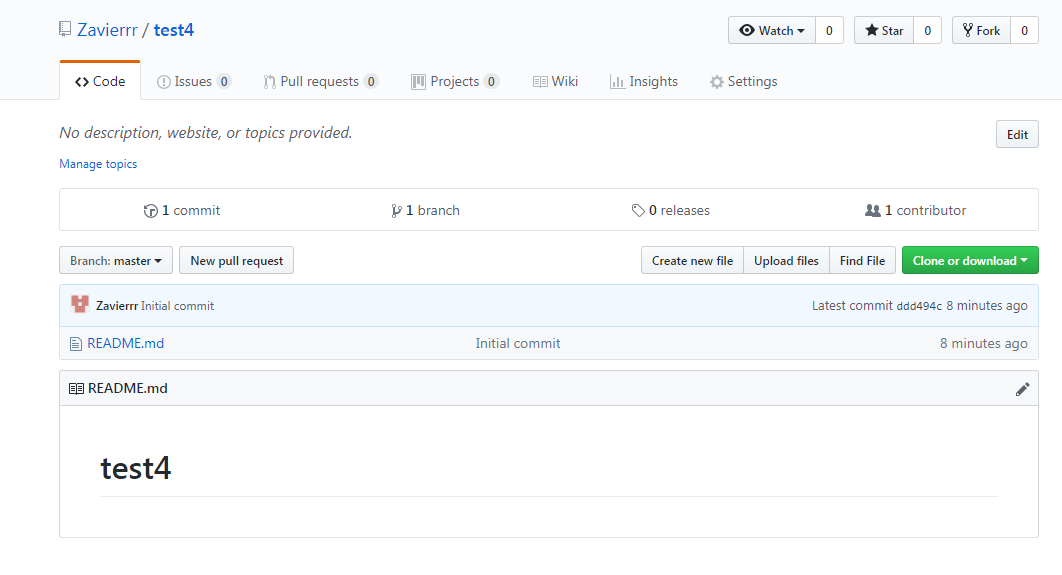


Github内的test3仓库存在本地test3仓库里创建的readme.txt文件

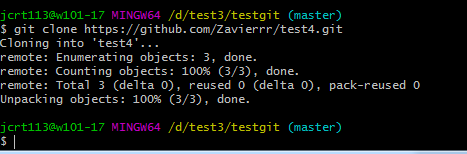


## 5.3 克隆一个本地仓库

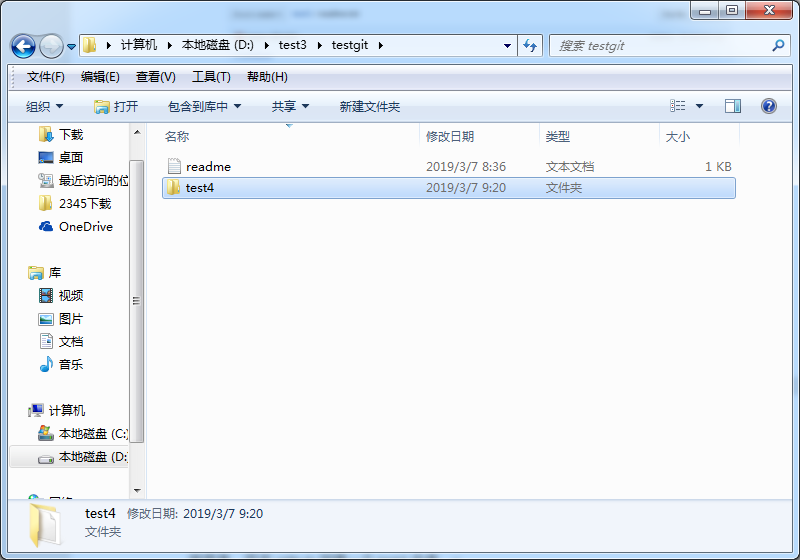
很简单，先在github创建一个test4仓库，



然后使用$ git clone <https://github.com/Zavierrr/test4.git> 即可，如图



由于忘记了在D盘克隆，导致文件被克隆在之前创建的test3仓库内



# 六、创建与合并分支（万隆隆）

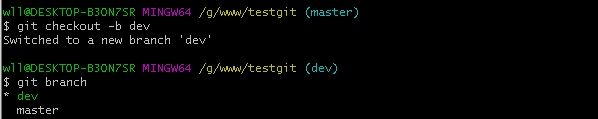
## 6.1 创建分支

创建并切换分支：

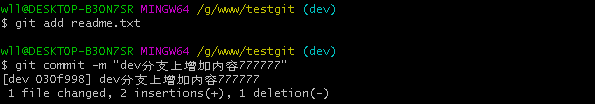
$ git checkout -b dev

查看当前分支：

$ git branch











## 6.2 合并分支

合并分支：

$ git merge dev



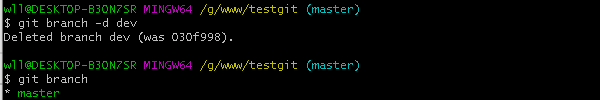
## 6.3 删除分支

删除分支：

$ git branch -d dev

再查看当前分支:

$ git branch



## 6.4 解决冲突

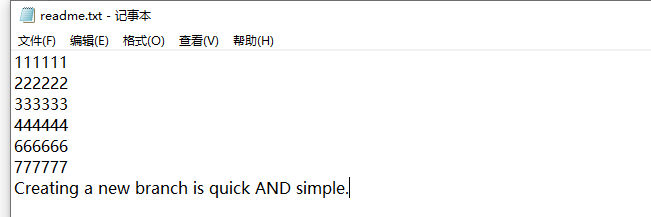
人生不如意之事十之八九，合并分支往往也不是一帆风顺的。

准备新的feature1分支，继续我们的新分支开发：

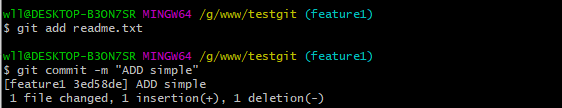


修改readme.txt最后一行，改为：

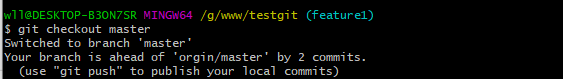
Creating a new branch is quick AND simple.



在feature1分支上提交：



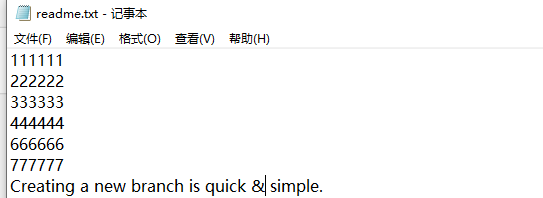
切换到master分支：



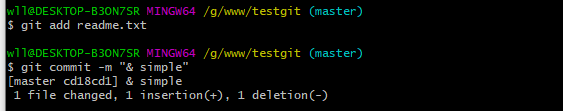
Git还会自动提示我们当前master分支比远程的master分支要超前1个提交。

在master分支上把readme.txt文件的最后一行改为：

Creating a new branch is quick & simple.



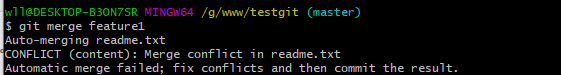
提交：



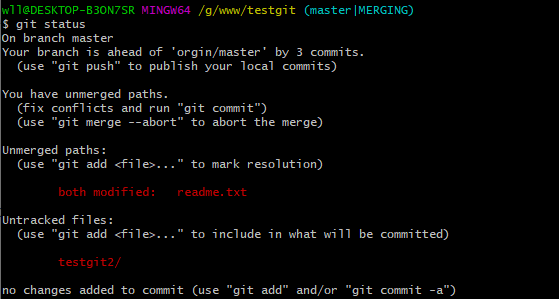
现在，master分支和feature1分支各自都分别有新的提交，变成了这样：



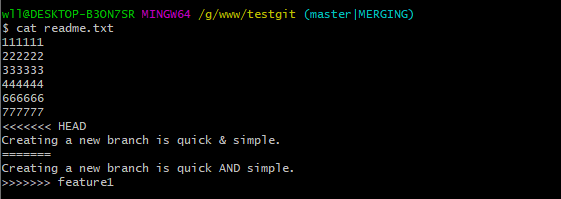
这种情况下，Git无法执行“快速合并”，只能试图把各自的修改合并起来，但这种合并就可能会有冲突，我们试试看：



果然冲突了！Git告诉我们，readme.txt文件存在冲突，必须手动解决冲突后再提交。git status也可以告诉我们冲突的文件：

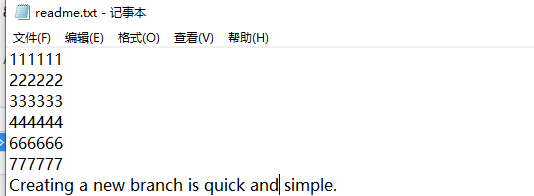


我们可以直接查看readme.txt的内容：

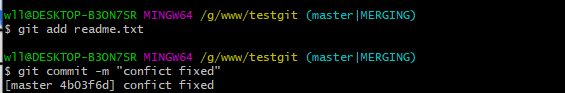


Git用<<<<<<<，=======，>>>>>>>标记出不同分支的内容，我们修改如下后保存：

Creating a new branch is quick and simple.



再提交：

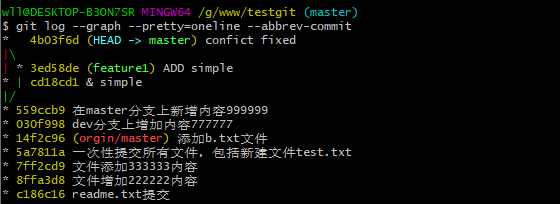


现在，master分支和feature1分支变成了下图所示：



用带参数的git log也可以看到分支的合并情况：

$ git log --graph --pretty=oneline --abbrev-commit



最后，删除feature1分支：



工作完成。

**小结**

当Git无法自动合并分支时，就必须首先解决冲突。解决冲突后，再提交，合并完成。

解决冲突就是把Git合并失败的文件手动编辑为我们希望的内容，再提交。

用git log --graph命令可以看到分支合并图。

## 6.5 分支策略管理

通常，合并分支时，如果可能，Git会用Fast forward模式，但这种模式下，删除分支后，会丢掉分支信息。

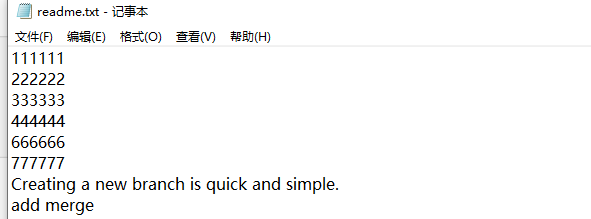
如果要强制禁用Fast forward模式，Git就会在merge时生成一个新的commit，这样，从分支历史上就可以看出分支信息。

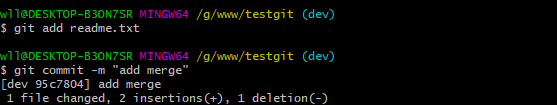
下面我们实战一下--no-ff方式的git merge：

首先，仍然创建并切换dev分支：

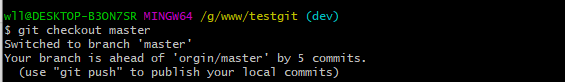


修改readme.txt文件，并提交一个新的commit：

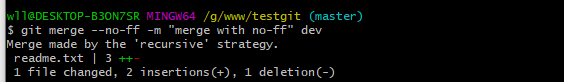




现在，我们切换回master：

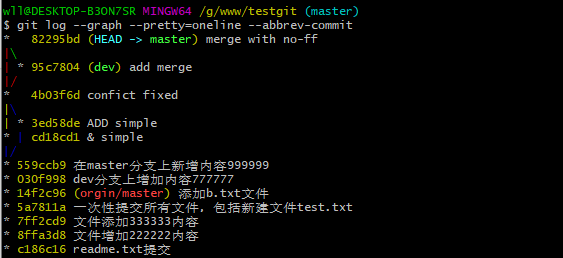


准备合并dev分支，请注意--no-ff参数，表示禁用Fast forward：



因为本次合并要创建一个新的commit，所以加上-m参数，把commit描述写进去。

合并后，我们用git log看看分支历史：



可以看到，不使用Fast forward模式，merge后就像这样：



分支策略

在实际开发中，我们应该按照几个基本原则进行分支管理：

首先，master分支应该是非常稳定的，也就是仅用来发布新版本，平时不能在上面干活；

那在哪干活呢？干活都在dev分支上，也就是说，dev分支是不稳定的，到某个时候，比如1.0版本发布时，再把dev分支合并到master上，在master分支发布1.0版本；

你和你的小伙伴们每个人都在dev分支上干活，每个人都有自己的分支，时不时地往dev分支上合并就可以了。

所以，团队合作的分支看起来就像这样：



**小结**

Git分支十分强大，在团队开发中应该充分应用。

合并分支时，加上--no-ff参数就可以用普通模式合并，合并后的历史有分支，能看出来曾经做过合并，而fast forward合并就看不出来曾经做过合并。