# 前言

GitHub是一个开源代码库及版本控制系统。GitHub是基于版本控制系统Git之上，而作为分布式版本控制系统，Git中不存在主库的概念，每一份复制出来的库都可单独使用，任何两个库之间的不一致处都可以进行合并。如果想要进行代码托管，进行团队协作，少不了Git工具。

GitHub的独特卖点在于从另外一个项目进行分支的简易性。为一个项目贡献代码非常简单：首先点击项目站点的Fork的按钮，然后将代码检出并将修改加入到刚才分出的代码库中，最后通过内建的pull request机制向项目负责人申请代码合并。

GitHub 的官网：<https://www.github.com>

# 术语解释

Repository：简称Repo，“仓库”。项目存放在仓库之中，建立项目前首先建立仓库；

Issues：“问题”，别人可以提Issue，我们解决Issue后可以将Issue关闭；

Star：“点赞”；

Fork：“拉分支”，复制一个完全相同的项目到自己的GitHub账户上，且独立于原项目；

Pull Request：“提交请求”，此功能建立在Fork之上的，如果Fork了一个项目，对其进行了修改，可以对原项目的拥有者提出一个Pull请求，等其对我们的请求进行审核，审核通过后，就可以把我们修改过的内容合并到原项目中，这时我们就成了该项目的贡献者；

Merge：“合并”，如果别人Fork了我们的项目，对其进行了修改并提出了Pull请求，这时我们可以对Pull请求进行审核。如果这个Pull请求的内容满足我们的要求，并与原项目没有冲突，就可以将其Merge到我们的项目中。

Watch：“观察”，如果我们Watch了一个项目，之后，如果这个项目有了任何更新，我们都会在第一时间收到通知。

# Git工具安装

安装Git工具；

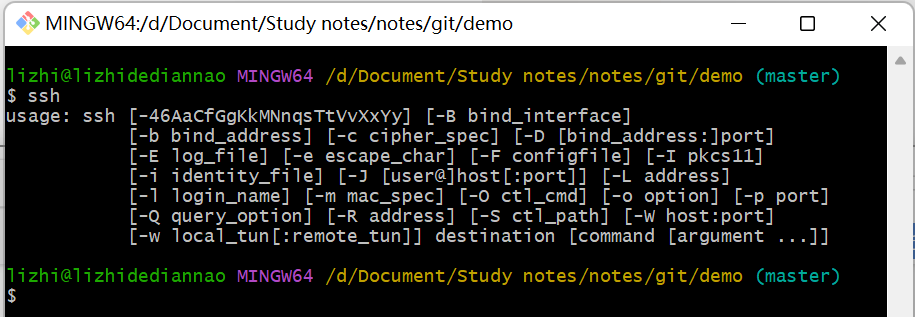
Git 的官网：git - -fast-version-control

# 利用SSH完成Git与GitHub的绑定

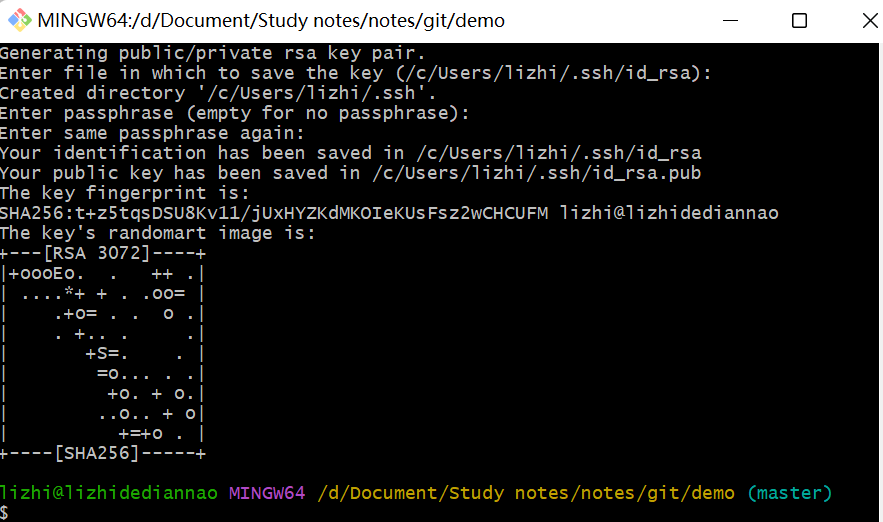
GitHub（代码库）是依赖Git进行版本控制，两者必须绑定才能通过Git向GitHub提交代码。GitHub一般是通过SSH(安全外壳协议)进行授权，而且大多数Git服务器都是使用SSH公钥来进行授权，所以想要向GitHub上提交代码，首先得在GitHub上添加SSH key配置。

## 生成SSH key

要想生成SSH key，首先要安装SSH，由于我们已经安装了Git Bash，所以自带了SSH，可通过Git Bash输入ssh命令，查看是否安装SSH：



如图表示已经安装了SSH。接下来，输入ssh-keygen -t rsa命令，表示指定RSA算法生成密钥，然后敲三（四）次回车键，期间不需要输入密码，之后会生成两个文件，分别为id\_rsa和id\_rsa.pub，即密钥id\_rsa和公钥id\_rsa.pub。这两个文件默认生成在C:\Users\lizhi\.ssh中。



## 添加SSH key

生成SSH key后，我们要把公钥id\_rsa.pub的内容添加到GitHub中，这样我们本地的密钥id\_rsa和GitHub中的公钥id\_rsa.pub才可以进行匹配，授权成功后，就可以向GitHub提交代码了。

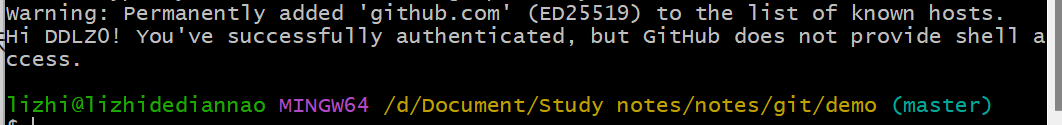
在GitHub网页中，点击个人头像进入setting设置界面，然后点击SSH and GPG Keys进入此子界面，点击New SSH key按钮，将我们生成的公钥id\_rsa.pub内容粘贴到key位置（titles可以不写），点击add SSH key即可。

## 验证绑定是否成功

通过Git Bash输入ssh -T [git@github.com](mailto:git@github.com)进行测试。第一次会出现：

The authenticity of host 'github.com (20.205.243.166)' can't be established.

输入yes回车，此时C:\Users\lizhi\.ssh中会生成known\_hosts和known\_hosts.old文件。



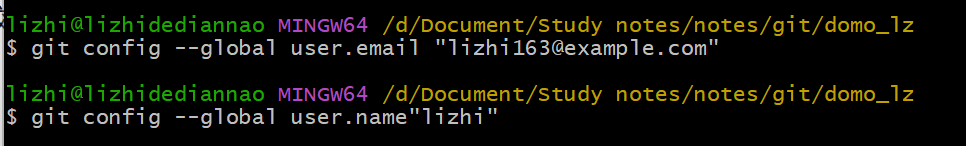
上图表示Git和GitHub绑定成功。

# Git初体验及其常用命令介绍

在Git中，所有的命令都是以git开头，例如：git init 其作用就是初始一个Git仓库。

注：在进行git操作前，必须切换到Git的仓库目录，即我们得先进到Git仓库（自己定义的）的最顶层文件目录下，然后从此目录中进入Git Bash，之后的操作才能顺利。

举例：建一个空文件夹demo，并在其中新建一个hit.txt，进入demo目录中，右键打开Git Bash。输入自己的Email和name，随便起的，与GitHub的账号无关。（提交到远程仓库GitHub有两种方式，第一种https方法是需要每次提交代码的时候输入GitHub的username和email，配置的地方与git初始化配置的地方没有直接关系且不存在覆盖关系，当然我们也可以自动登陆GitHub，第二种是通过SSH秘钥提交代码，与git初始化配置的地方没有直接联系。）



注：如果你的Git还没有初始化，可以通过这个指令：

$ git config --global user.name “输入你的用户名”

$ git config --global user.email “输入你的邮箱”

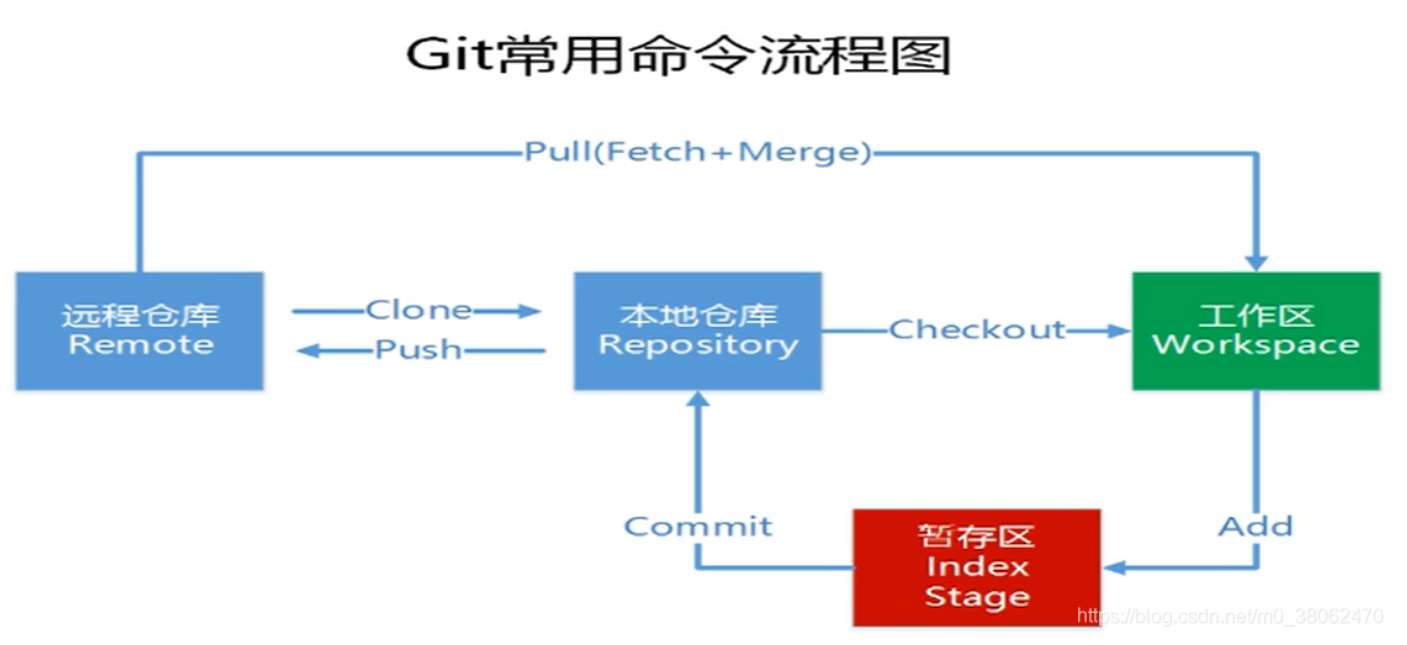
Eg:

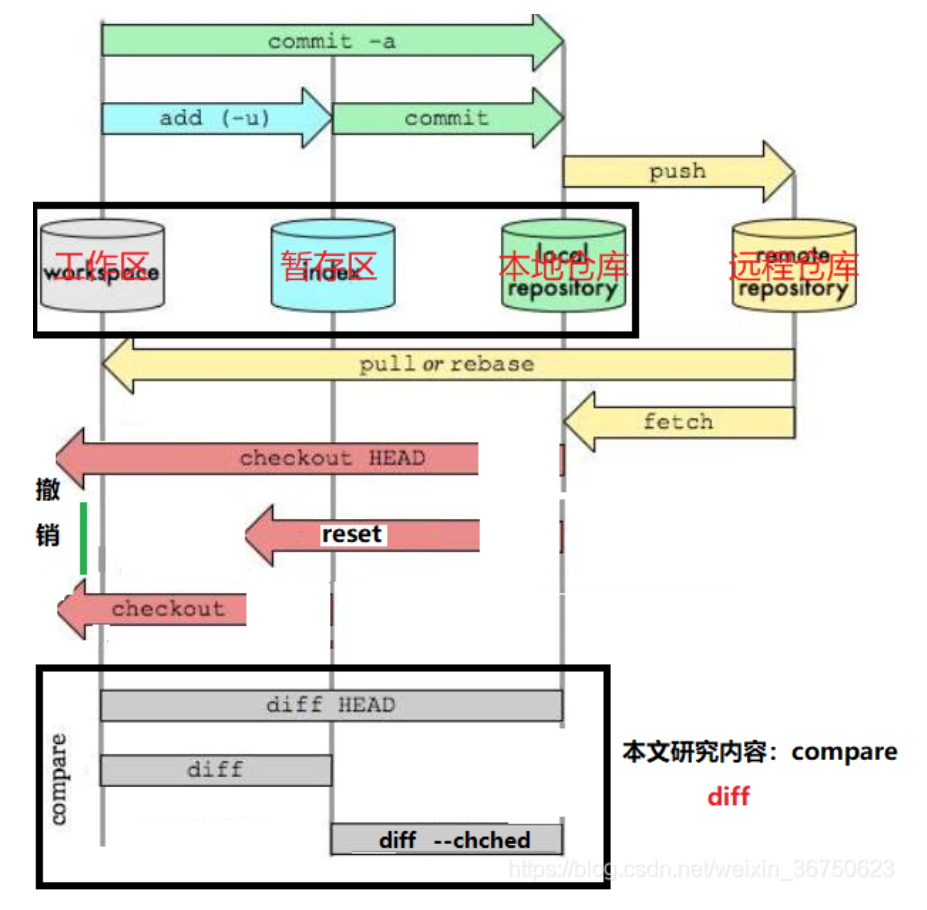
$ git config --global user.name “lizhi”

$ git config --global user.email “lizhi163@example.com”

可以直接配置你的用户名和邮箱。

如果已经初始化了而需要修改，不太好改。





## git init

在Git Brash 中输入git status命令，将该目录初始化为一个Git本地仓库，并**默认进入Git仓库的master分支**，即主分支。

## git status

查看仓库状态，实时了解仓库信息。

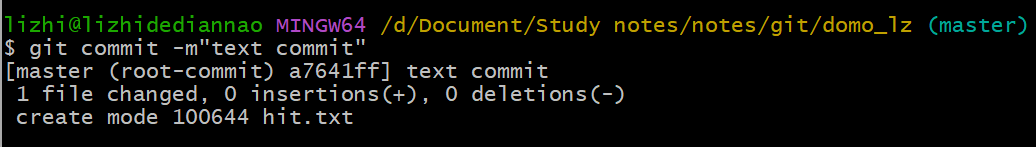
## git add

git add hit.txt，即将hit.txt文件添加到Git仓库中，注意git add命令并没有将文件提交到Git仓库，而是把文件添加到**【临时缓冲区】**，这个命令有效防止我们错误提交的可能性。

git add . //将该目录下所有文件加入进来（相当于把目录下的文件加入一个仓库方便你管理）

## git commit

git commit -m “text commit”命令，将hit.txt文件提交到Git**本地仓库**，-m表示提交信息，提交信息写在双引号“”内。提交信息必须得有，对此次提交作以解释说明。



git commit -m用于提交暂存区的文件；

git commit -am用于提交跟踪过的文件（第一次已经add过，被跟踪了，如果后面修改了源文件，可以不用add，直接git commit -am）

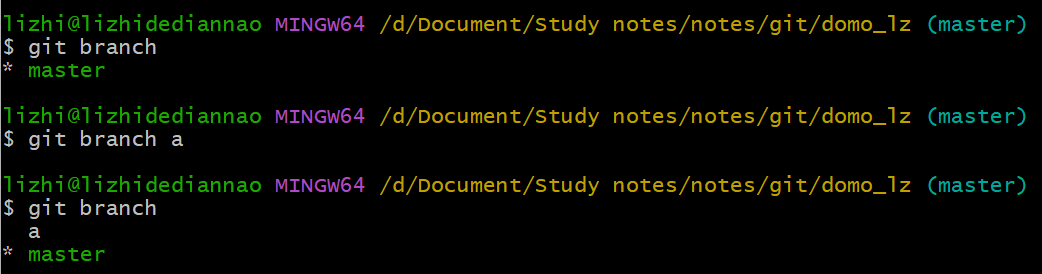
## git log

git log //打印Git仓库提交的日志 。

git log 文件名 //打印这个文件的日志

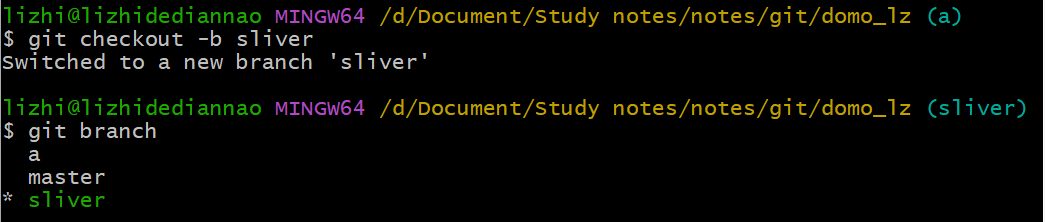
## git branch

查看Git仓库的分支情况，\*表示“当前所在的分支” ；git branch a命令，即建立一个新分支a，用git branch命令查看时就可以到a分支。



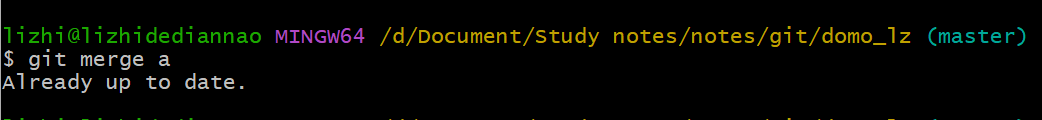
## git checkout

git checkout a命令，即切换到a分支。我们也可以在创建分支的同时直接切换到新分支，命令为：git checkout -b，例如git checkout -b sliver，下图，我们是在a的分支下创建的sliver分支（sliver是a的分支），并直接切换到sliver分支上。



## git merge

切换到master分支，然后输入git merge a命令，将a分支合并到master分支：



如图，已经将a分支合并到主分支了。此外，在合并分支的时候，如果两个分支有冲突，则不能直接合并，需要先解决冲突；反之，则可以直接合并；

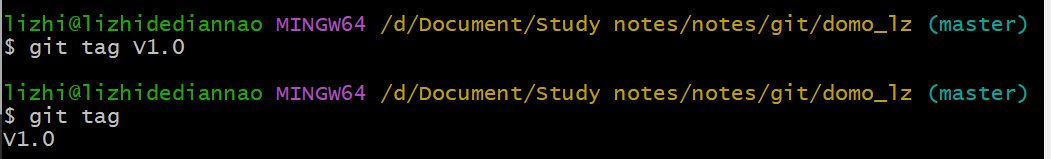
## git branch -d &git branch -D

在命令窗输入git branch -d a命令，删除a分支。

需要注意的是，有时候通过git branch -d命令删除不了分支，原因可能是分支a的代码没有合并到主分支，这时如果我们一定要删除该分支，那么可以用git branch -D命令强制删除。

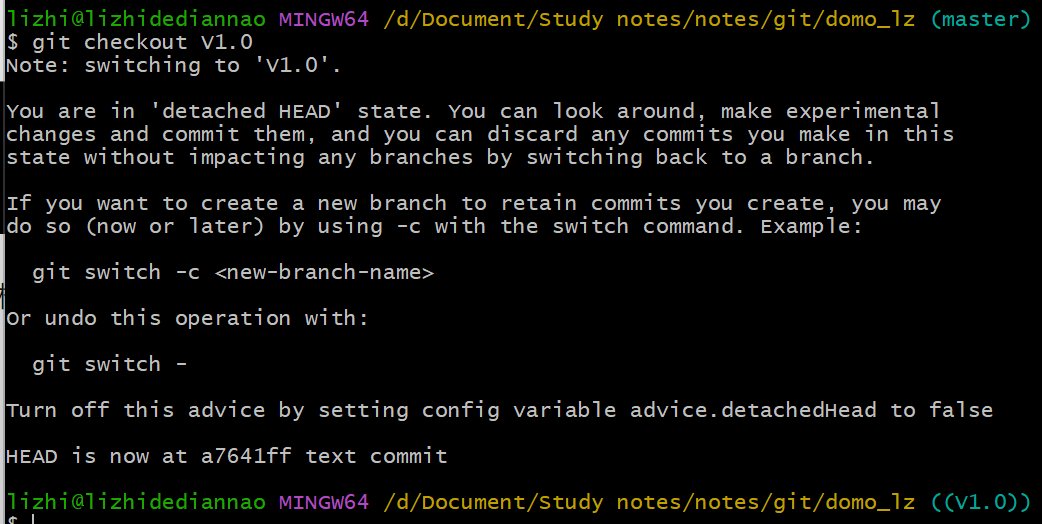
## git tag

在命令窗口输入git tag V1.0命令，为当前分支添加标签：



打完标签后，可以通过git tag 命令查看标签记录。

通过git checkout V1.0即可切换到该标签下的代码状态：



## git restore

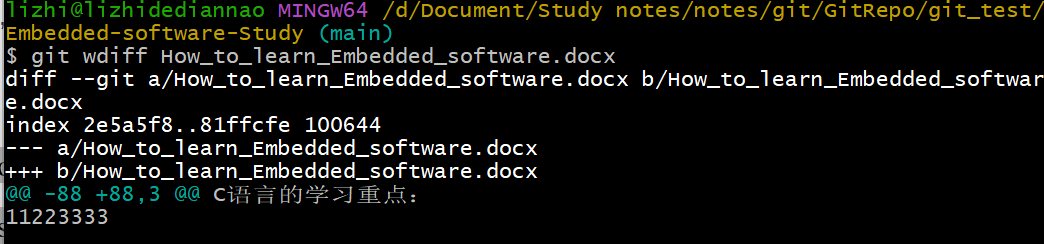
git restore --staged <file\_name> 将暂存区的修改重新放回工作区（包括对文件自身的操作，如添加文件、删除文件）

git restore <file\_name> 丢弃工作区的修改（不包括对文件自身的操作，如添加文件、删除文件）

## git wdiff

git wdiff How\_to\_learn\_Embedded\_software.docx

比较工作区和本地仓库(commit后)How\_to\_learn\_Embedded\_software.docx文件的不同;



git wdiff //比较工作区和本地仓库的不同

（白色为未修改部分，绿色为修改部分）

## pandoc

pandoc -s How\_to\_learn\_Embedded\_software.docx -t markdown -o How\_to\_learn\_Embedded\_software.md

通过pandoc将How\_to\_learn\_Embedded\_software.docx文件转化为md格式，以后上传git可以只上传md文件

经过配置及，可以在用git wdiff时，先将word转成md格式对比，其实不需要pandoc转化。这个转化的意义在于网页Github能否预览文档。

注意：pandoc命令下的文件名不能有空格

# 通过Git将代码提交到GitHub

本地Git与远程GitHub绑定后，我们就可以通过Git向GitHub提交代码了。

这里经常用到的命令是：push和pull。

push意思“推上去”，如果我们本地的代码有了更新，为了保持本地与远程的代码同步，需要把本地的代码推到远程的仓库：

git push origin master

pull意思是“拉下来”，如果远程仓库的代码有了更新，为了保持本地与远程代码的同步，需要将远程代码拉到本地：

git pull origin master

此外，之前提到的pull requst在这里能更好的理解：如果我们fork了别人的项目，并对其进行了修改，想要把我们修改后的代码合并到原项目中，我们就需要提交一个pull request，让原作者将我们的代码拉到Ta的项目中，对于Ta来说，我们属于远程端。

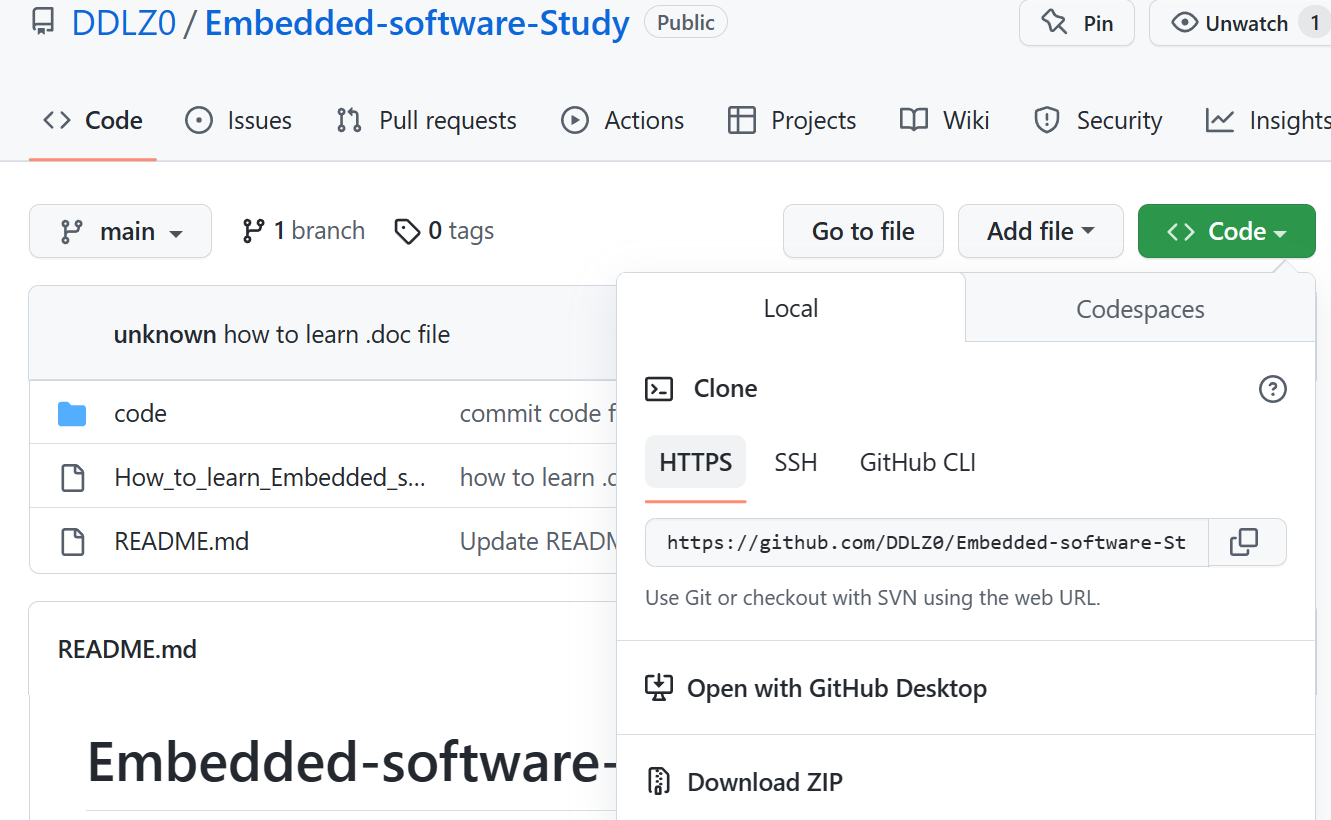
一般情况下，在push操作之前都会进行pull操作，这样不容易造成冲突。

## 提交代码

向远程仓库（GItHub）提交代码大体为：本地没有Git仓库，这时可以直接将远程仓库clone到本地。通过clone命令创建的本地仓库，其本身就是一个Git仓库了，不需要再init初始化操作了，而且自动关联远程仓库。后续只需要在这个仓库进行修改或者添加等操作，然后commit即可；

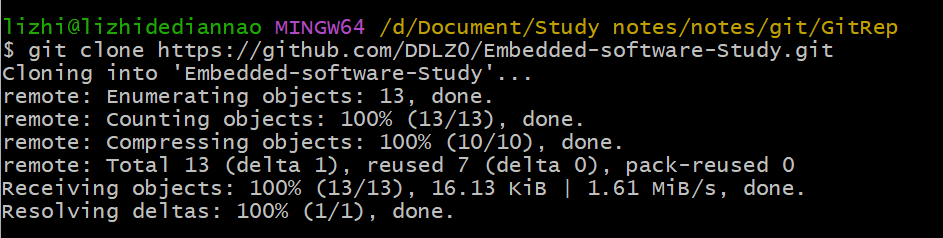
## 实例操作

1. 进入GitHub个人主页，打开一个项目，复制HTTPS的链接：

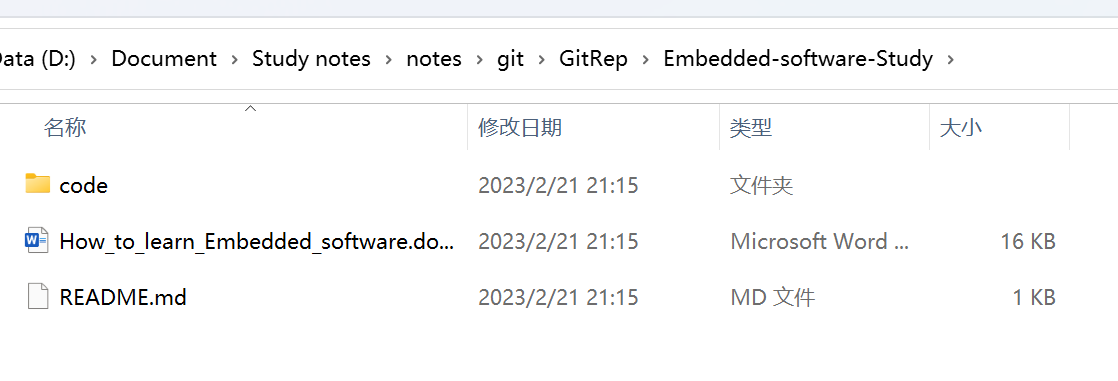


1. 在本地新建一个GitRep目录，在该目录下进入Git Bash，输入:

git clone <https://github.com/DDLZ0/Embedded-software-Study.git>命令，这个链接就是刚才在GitHub上复制的远程仓库。



如上图，成功将远程仓库clone到本地。本地目录如下：



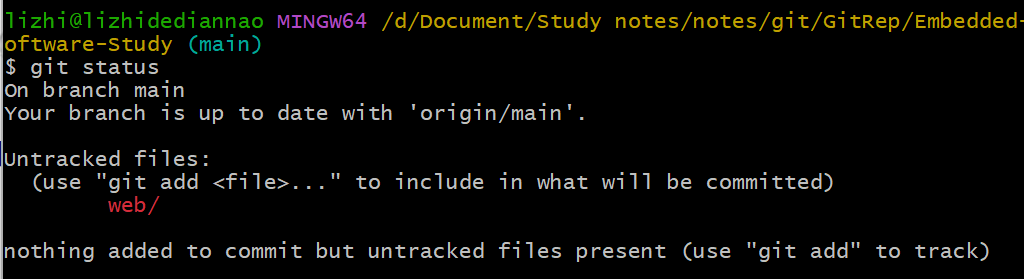
注意：新建目录后，第一次登陆，输入自己的name和Email：

$ git config --global user.name “lizhi”

$ git config --global user.email “lizhi163@example.com”

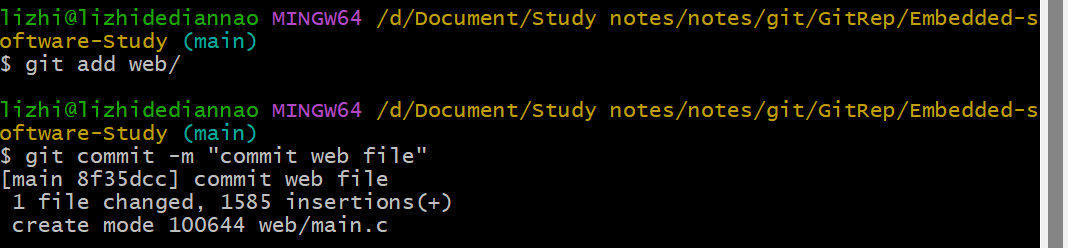
文件名建议用英文，不能带空格；

1. 将新增的一个文件web放入本地Embedded-software-Study项目的目录下，然后从Embedded-software-Study目录下进入Git Bash，输入git status查看仓库状态：

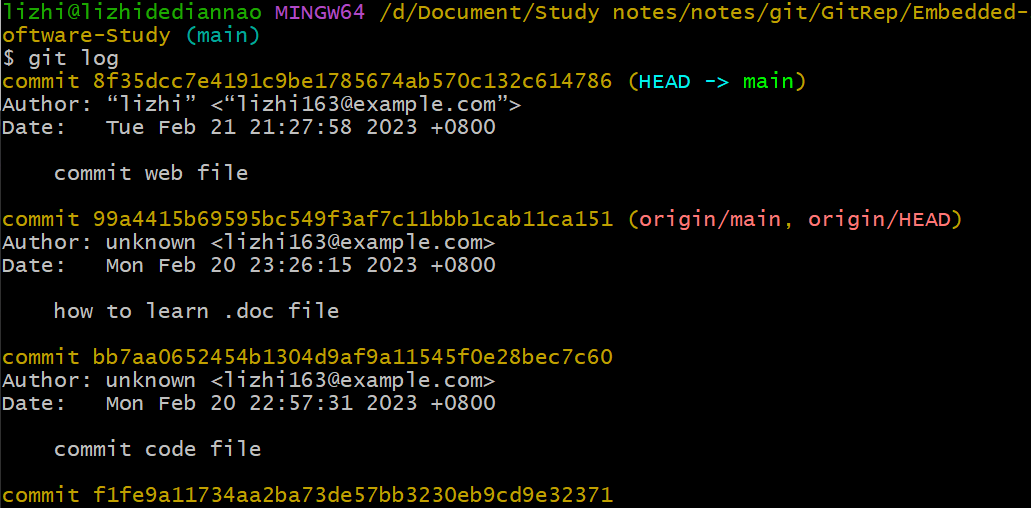


显示web文件未被追踪，就是刚刚复制过来的文件没有提交；

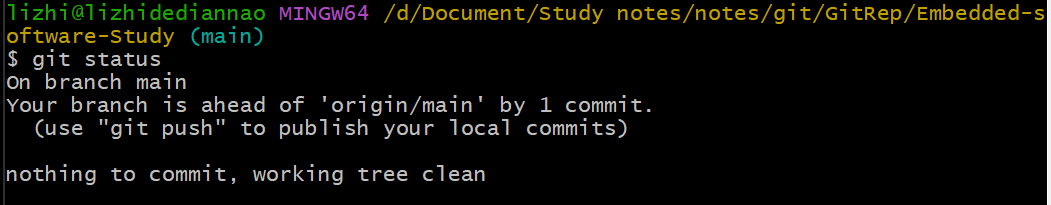
1. 输入git **add** web/命令，将web文件夹放入**临时缓存区**
2. 输入git commit -m “commit web file”命令，将**临时缓存区的文件提交到本地仓库：**



1. 可以输入git log查看仓库日志：

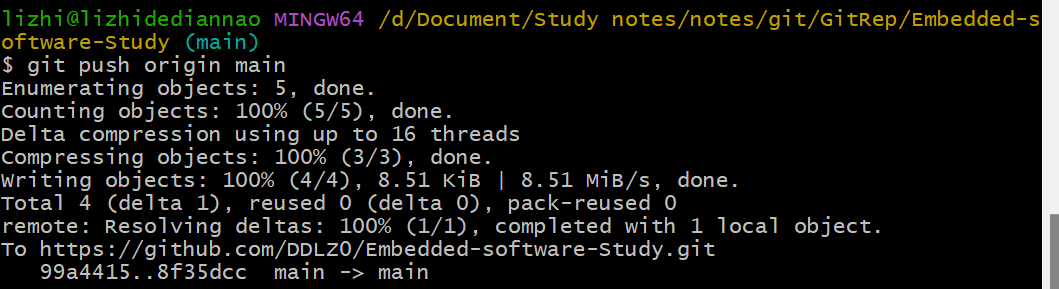


1. 再输入git status查看仓库状态：

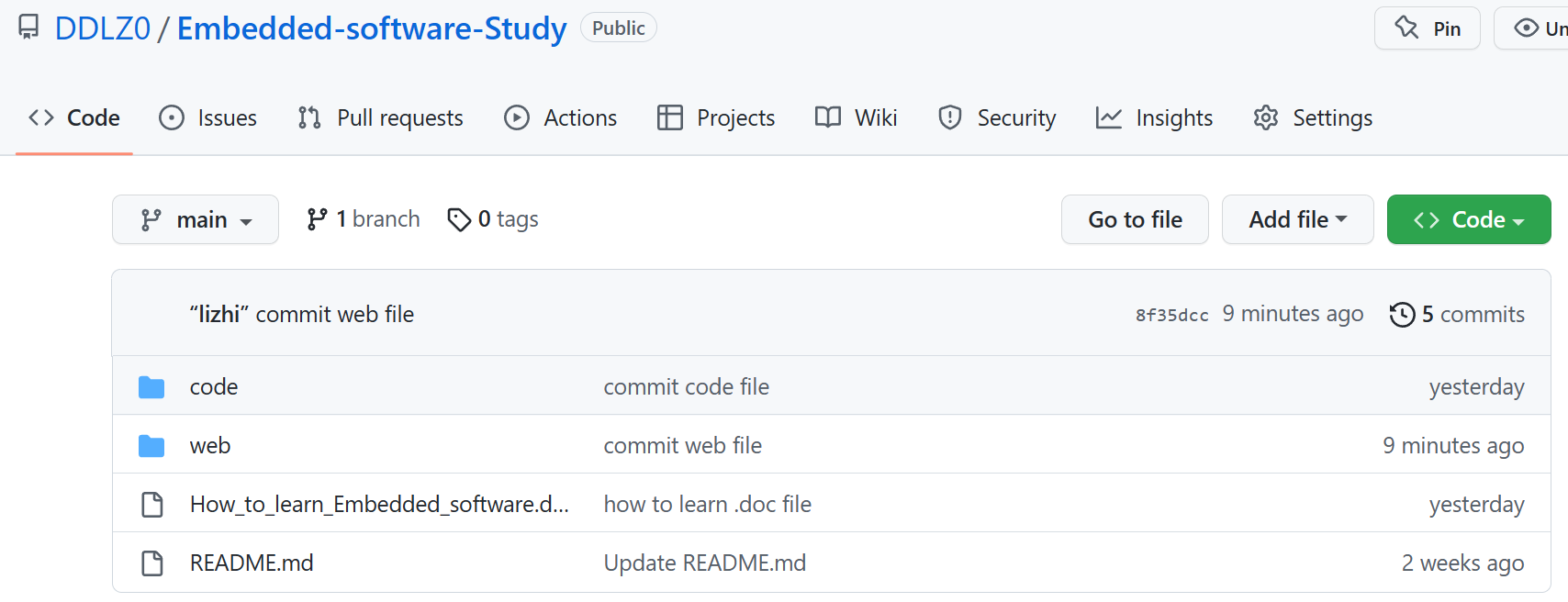


这里显示你的本地分支早于远程主分支有一个commit，需要git push到远程仓库

1. 输入git push origin main，远程是main，这里就是main；远程是master，这里就是master：

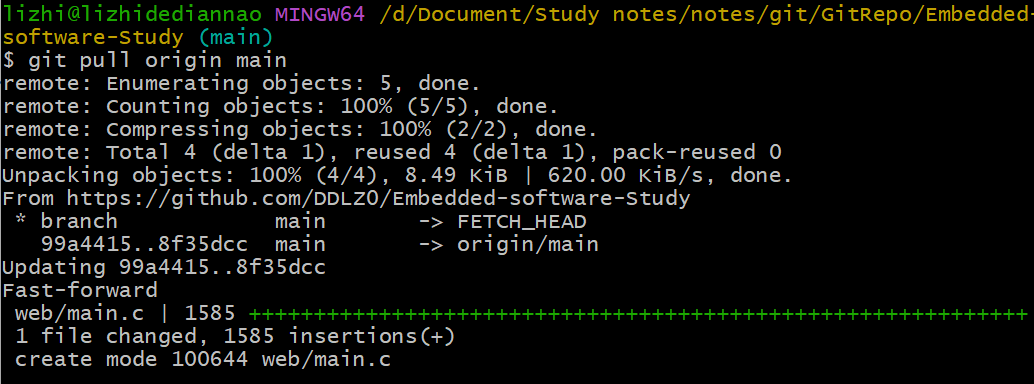


这样就push成功了，可以在GitHub上查看远程仓库：



注：第一次需要输入个人访问令牌（token），后面有介绍；

**强调一遍**：在我们向远程仓库提交代码的时候，一定要先进行pull操作，再进行push操作，防止本地仓库与远程仓库不同步导致冲突的问题。如下，先输入git pull origin main命令，将远程数据拉下来更新本地仓库，如果远程有新数据，则会显示更新信息：



## 提交文本

由于doc文件在Github上无法预览，所以可以用pandoc将doc转为md文件。

1. git clone xxx克隆一个仓库；
2. 写个这个后缀的文档放在根目录下：

\*.docx diff=pandoc

在用git wdiff（比较工作区与本地仓库的区别）时，先将word转成md格式对比;

1. 放入其他文件.doc，用pandoc -s XXX.docx -t markdown -o XXX.md命令将XXX.docx文件转化为md格式。注意：文件名不能有空格，可以是中文（可有可无，方便Github预览，图片表格格式不推荐）；
2. git add. 命令将本目录下所有文件放入暂存区；
3. git commit -m”描述”
4. 输入git push origin main，远程是main，这里就是main；远程是master，这里就是master：

# 版本回退

这是使用git管理word文档的核心价值所在。

## 未使用git add缓存代码

git checkout -- 文件名

注：中间--，如果忘了，就成了切换分支了！！

git checkout .

此命令为放弃所有还没加入缓存区（git add命令）的修改，即撤销工作区修改，但并不会删除新建的文件，因为新建的文件还没有加入git的管理系统中，自己手动删除即可。

## 已使用git add缓存代码，未使用git commit提交到本地仓库

git reset HEAD 文件名

放弃某文件的缓存

git reset HEAD

放弃所有文件的缓存

相当于撤销git add命令，使用本命令后，本地的修改并不会消失，而是回到了第一步未使用git add缓存代码，继续使用git checkout – 文件名，就可以放弃工作区修改。

## 已使用git commit提交到本地仓库

### git reset

git reset -- hard HEAD^

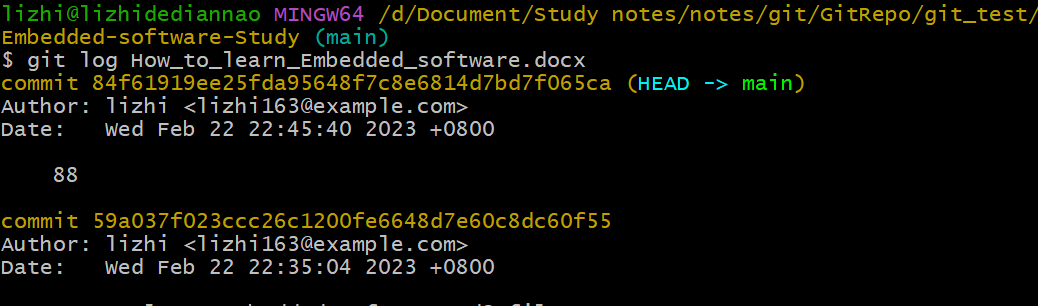
退回到上一次commit的状态

git reset -- hard commit\_id

退回到任意版本，先使用git log命令查看git提交历史和commit\_id

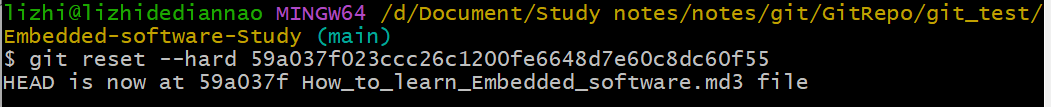
例如：

首先打印该文档的日志，选择想要回退的版本号：



用以下命令：

git reset --hard commit\_id // commit\_id就是上图中commit之后的版本号，选择对应版本号即可



通过reset的方式，把head指针指向之前的某次提交，reset之后，后面的版本就找不到了，不太推荐。还有一种：revert（推荐）

### git revert

这种方式不会把版本往前回退，而是生成一个新的版本。所以，你只需要让别人更新一下代码就可以了，你之前操作的提交记录也会被保留下来

# 注意区别

## git commit和git push

git commit把你的修改放到你的本地仓库中，而git push会把你的修改发送到远程位置；

## fork和clone

在GitHub中，Fork与Clone的功能是比较相像的，因为它们的基本用途都是拷贝（copy）仓库（Repository）。但Fork与Clone这两种拷贝又很不一样。为了便于初学者理解，我们可从下列几个方面作一个简略的比较：

1. 往哪儿拷贝( where )：fork把一个GitHub账户的仓库拷贝成另一个GitHub账户的仓库，是“云对云”拷贝；而clone把一个GitHub账户的仓库拷贝到本地安装了Git的电脑中，是“云对地”拷贝。
2. 为什么拷贝（why）：作为“云对云”拷贝，fork常用于“拿来”（即把云中某个开源仓库拷贝回来）的情形；而“云对地”拷贝，clone则常用于“保护”（即对云中仓库的备份、维护、开发新版本等）的情形；可能更多的时候，是把二者结合起来使用。
3. 变成了什么（what）：用fork拷贝所得到的仓库，可成为源仓库的一个Git分支（branch），因而其文件的修改提交，可以直接被源仓库在云中合并（merge）；而通过“云对地”拷贝的仓库就变成本地的Git仓库。要把修改的内容同步到云中，则必须在本地用Git 的推送（push）去完成。

## 仓库目录下文件名

文件名建议使用英文，且无空格键

## 增删远程

git remote # 查看有哪些remote，一般就一个，叫做origin

git remote -v# 查看远程及地址

git remote remove origin # 删除，一般不用到

git remote add origin xxx # 新增，一般不用到。xxx 请用最开始的 git clone xxx 中的 xxx 替代

其实，一开始的 git clone xxx 命令，可以看做是一个复合命令，它做了两件事：第一，下载文件，第二，执行 git remote add origin xxx

## 添加一个远程仓库，独立管理

示例：

指定一个默认的远程库 orgin

新增一个远程库 git remote add tina https://github.com/tingtingtina/xxx.git

有新的提交推到到 repo git push (默认 推到 origin）

如果要同步到远程库 tina 中 需要手动指定仓库别名 push git push tina：

$ git remote -v

origin http://git.xxx (fetch)

origin http://git.xxx (push)

tina https://github.com/tingtingtina/xxx.git (fetch)

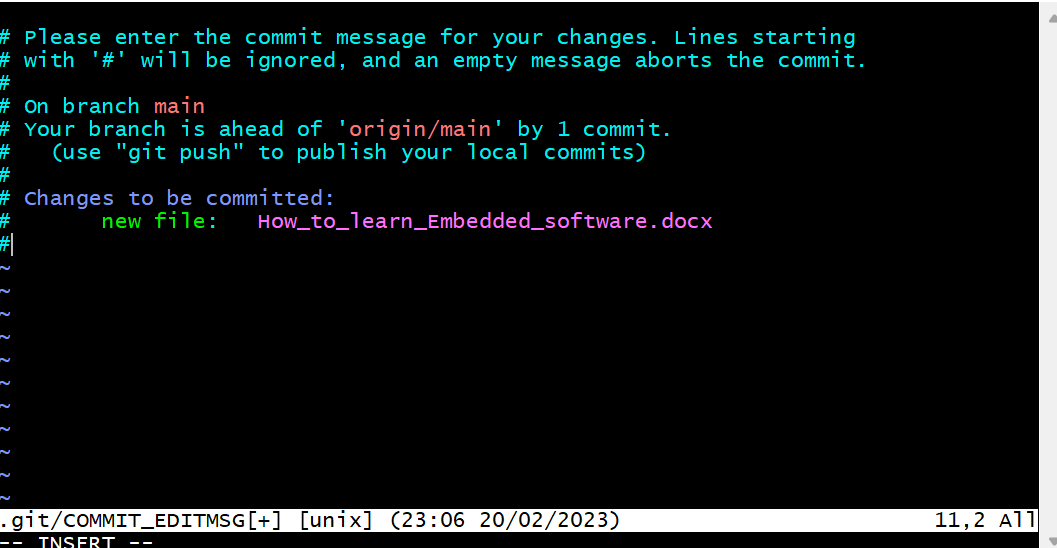
tina https://github.com/tingtingtina/xxx.git (push)

从信息上也可以看出来，这两个仓库需要独立管理。

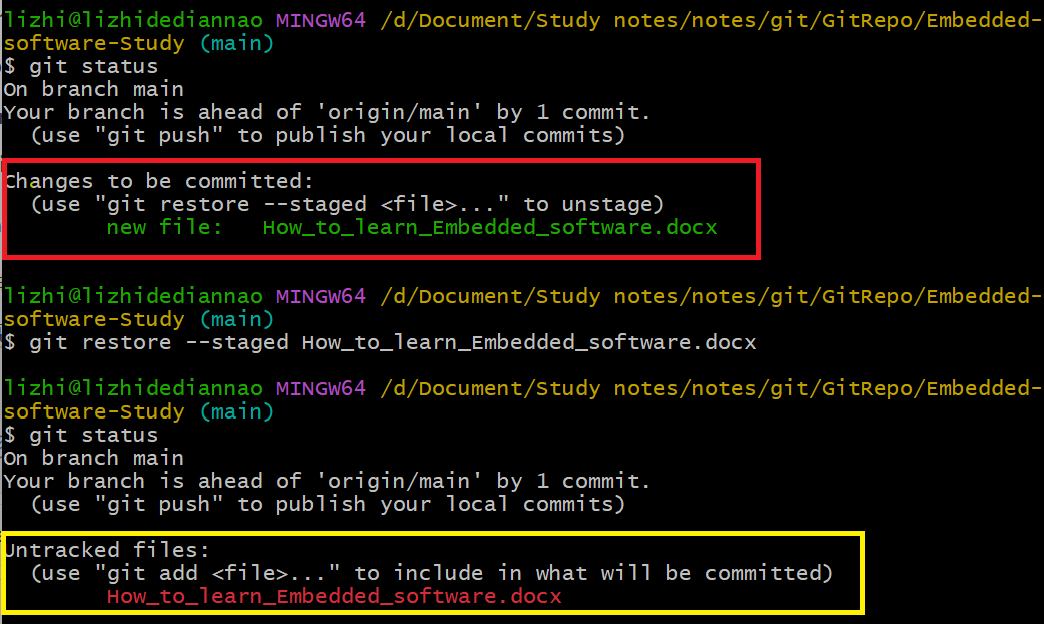
# 问题

## git commit报错

git commit 如果不加-m “提交信息”，则会被终止提交:



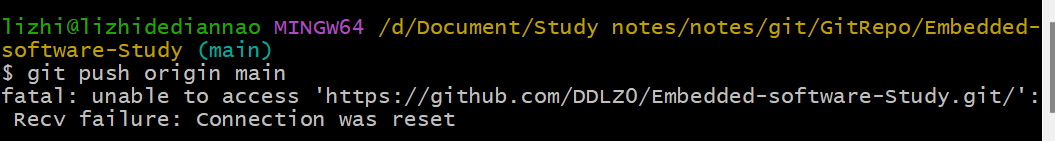
重新进入Git Bash后，查看status可以看到change commit解仓库的命令：git restore –staged <文件名>，解除之后就可以重新add了

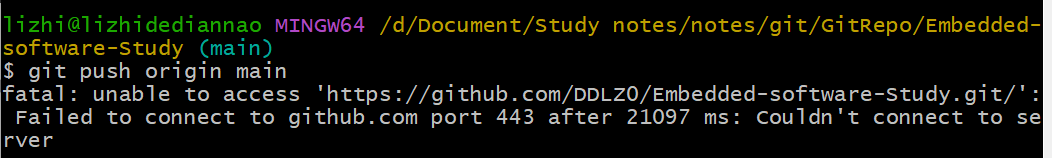


## 退出正在运行的git命令

q或ctrl+c

## 无法连接服务器：





解决办法一：重新输入自己的name和Email：

$ git config --global user.name “lizhi”

$ git config --global user.email “lizhi163@example.com”

（第一次输入name和Email后，在C:\Users\lizhi\.gitconfig中已经生成了邮箱和名字）

另外：

设置git的全局代理。查看代理命令如下：

$ git config user.name

$ git config user.email

设置命令如下：

$ git config --global user.name "username"

$ git config --global user.email "email"

取消命令如下：

$ git config --global --unset http.proxy

$ git config --global --unset https.proxy

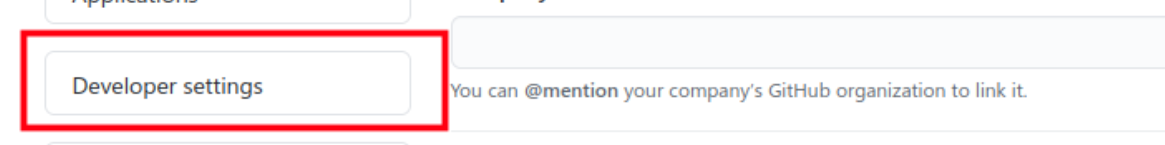
解决办法二：将Git Bash关了重开；

## Push云端时的bug

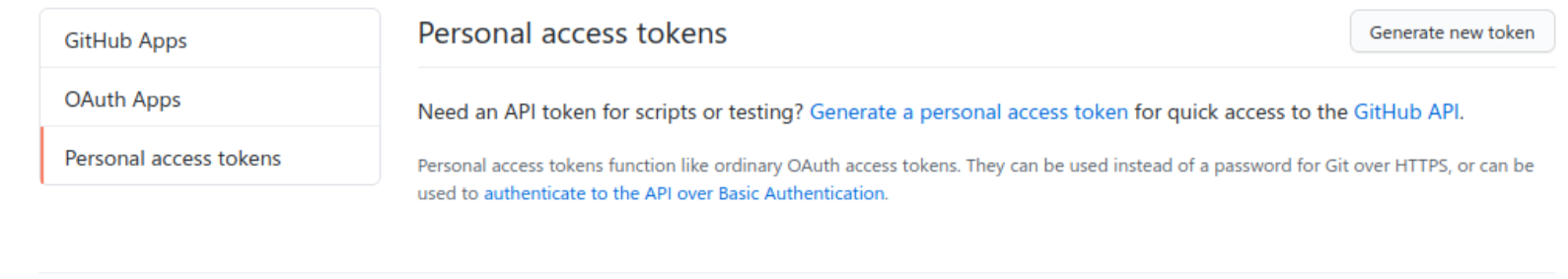


git push origin main后，用自己的账户登录后提示错误，remote报错：2021年8月13日，移除了对密码验证方式的支持。**请改用个人访问令牌**。

如何生成个人访问令牌，在github.com的个人设置中：



接着选择personal access tokens，然后点右上角的Generate new token，



在Note中填入这个token的具体用途（随便填方便记忆即可），在Expiration中选择token的有效时间默认30天，在Select scopes中设置该token的权限，这里我只是需要push下代码所以只勾选了repo，你可以根据自己的需求进行设置。然后点击最下方的Generate token按钮即可生成。最后保存好这个token，以后就不会在显示了

个人访问令牌：ghp\_jqGmVHAWKrbvPdNYRsdLpGM3n8d5Bp1IwS4z

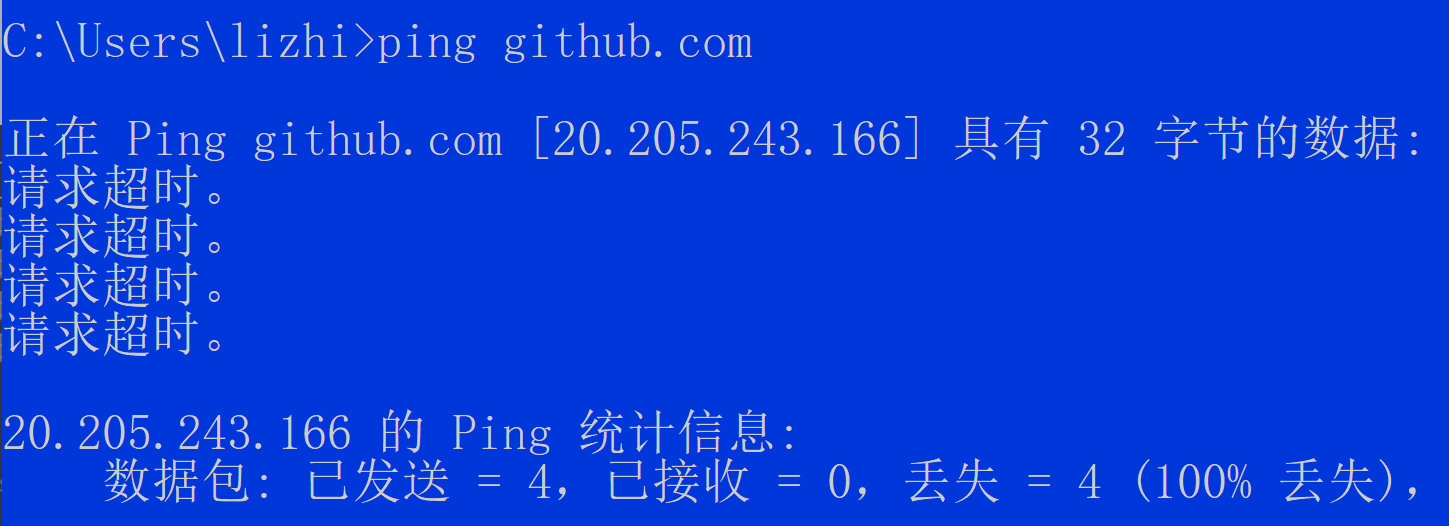
之后push代码时密码使用这个token就可以啦

第二版令牌：无期限

ghp\_pvykJwuXjqYY66pSD4PwmskUY1ELkG1MHKgk

# 备注

在终端输入ping github.com，如果通了则OK，不通就有问题：



(ping (Packet Internet Groper)，因特网包探索器，用于测试网络连接量的程序。Ping发送一个ICMP；回声请求消息给目的地并报告是否收到所希望的ICMP echo （ICMP回声应答）。它是用来检查网络是否通畅或者网络连接速度的命令

ping命令通常用来作为网络可用性的检查。ping命令可以对一个网络地址发送测试数据包，看该网络地址是否有响应并统计响应时间，以此测试网络。)