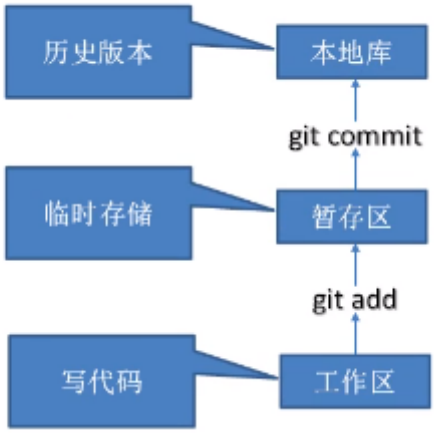
Git笔记

首次安装Git需要提交自己的用户签名和邮箱(无需验证真实性)，否则可能无法提交代码。

```
git config --global user.name//用户名
git config --global user.email//邮箱
```

Git工作机制



基本语法：

1.初始化本地库:git init(会生成一个.git隐藏文件)

查看本地库状态:git status(显示文件实在那个分支 是否有新增文件或者修改)

①有新增而且没上传暂存区(未使用git add):

```
22476@DESKTOP-ORPFREQ MINGW64 /d/桌面/实训笔记/GitDemo (master)
$ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        text.txt

nothing added to commit but untracked files present (use "git add" to track)
```

（红色text.xt表示文件中未上传更新到暂存区的文件，提示用git add 上传到暂存区）

②新文件已经上传暂存区：

```
22476@DESKTOP-ORPFREQ MINGW64 /d/桌面/实训笔记/GitDemo (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   text.txt
```

2.添加暂存区：git add(添加文件到暂存区)

初始化文件后需要使用git add 文件名或者 git add . 将文件添加至暂存区

```
22476@DESKTOP-ORPFREQ MINGW64 /d/桌面/实训笔记/GitDemo (master)
$ git add text.txt
```

注意：上传暂存区只是在中间的一个暂存阶段，可删除已经上传暂存区文件（远程库无法显示暂存区文件）

删除暂存区文件：git rm -cached 文件名

该命令可删除暂存区指定文件，用git status可以查看相关状态

```
22476@DESKTOP-ORPFREQ MINGW64 /d/桌面/实训笔记/GitDemo (master)
$ git rm --cached text.txt
rm 'text.txt'
```

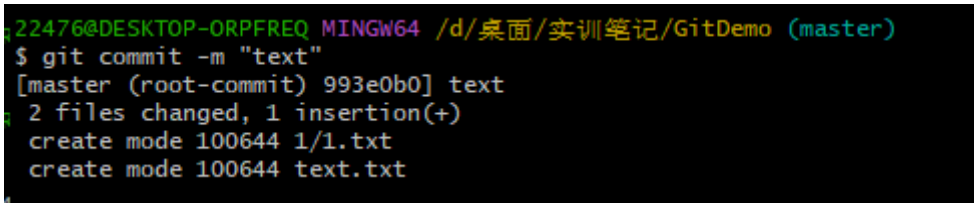
从暂存去恢复工作台已删除文件：git restore 文件或文件夹

暂存区已有文件但是自己本地已经删除

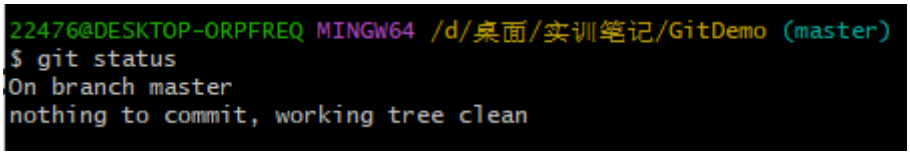
3.提交本地库：git commit -m "日志信息" 文件名

①-m ""是将你提交本库的文件提交日志信息，即使写程序也会要求你填写日志信息

②不填写文件名则会将文件中未提交本地库的文件或者已经修改后的文件全部上传

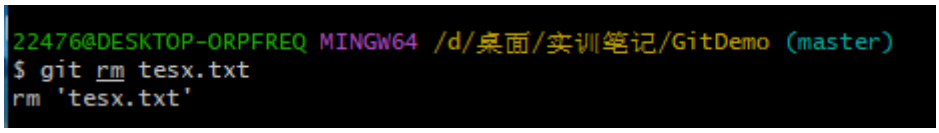


上传后在使用git status查看本地库状态：



显示该工作树已经干净(暂存区文件或修改后的文件已经全部上传本地库)

删除工作区文件(删除后需要将修改后的添加暂存区,提交本地库,然后上传到远程残酷)： git rm 文件名



4.历史版本

查看引用日志信息：git reflog

```
22476@DESKTOP-ORPFREQ MINGW64 /d/桌面/实训笔记/GitDemo (master)
$ git reflog
993e0b0 (HEAD -> master) HEAD@{0}: commit (initial): text
```

前面的993e0b0是版本号(版本号前7位) text是git commit -m后的版本

查看详细日志：git log

```
22476@DESKTOP-ORPFREQ MINGW64 /d/桌面/实训笔记/GitDemo (master)
$ git log
commit 993e0b0dd94a8fc168ab6bdade45b0005b5e4d1d (HEAD -> master)
Author: yuboqi <2247678912@qq.com>
Date:   Sat Jun 25 10:24:35 2022 +0800

    text
```

commit后的字符串是完整的版本号

Anthor：提交该版本的用户信息

！！ 版本穿梭：git reset --hard 版本号！！

使用git reflog查看历史版本查看其版本号和现在所指向的版本

使用git reset --hard 版本号回到历史版本并修改文件

可以在使用git reflog查看具体信息

```
22476@DESKTOP-ORPFREQ MINGW64 /d/桌面/实训笔记/GitDemo (master)
$ git reset --hard a8bf7d3
HEAD is now at a8bf7d3 second

22476@DESKTOP-ORPFREQ MINGW64 /d/桌面/实训笔记/GitDemo (master)
$ git reflog
a8bf7d3 (HEAD -> master) HEAD@{0}: reset: moving to a8bf7d3
993e0b0 HEAD@{1}: reset: moving to 993e0b0
a8bf7d3 (HEAD -> master) HEAD@{2}: commit: second
993e0b0 HEAD@{3}: commit (initial): text
```

修改文件信息：vim 文件名：(进入修改页面修改文件信息)

```
22476@DESKTOP-ORPFREQ MINGW64 /d/桌面/实训笔记/GitDemo (master)
$ vim text.txt
```

esc后使用 :wq 保存并退出编辑

5.Git分支操作

查看分支：git branch -v

```
22476@DESKTOP-ORPFREQ MINGW64 /d/桌面/实训笔记/GitDemo (master)
$ git branch -v
* master a8bf7d3 second
```

创建分支：git branch 分支名

```
22476@DESKTOP-ORPFREQ MINGW64 /d/桌面/实训笔记/GitDemo (master)
$ git branch  newmaster

22476@DESKTOP-ORPFREQ MINGW64 /d/桌面/实训笔记/GitDemo (master)
$ git branch -v
* master     a8bf7d3 second
  newmaster a8bf7d3 second
```

切换分支：git checkout 分支名

```
22476@DESKTOP-ORPFREQ MINGW64 /d/桌面/实训笔记/GitDemo (master)
$ git checkout newmaster
Switched to branch 'newmaster'

22476@DESKTOP-ORPFREQ MINGW64 /d/桌面/实训笔记/GitDemo (newmaster)
$ git branch -v
  master    a8bf7d3 second
* newmaster a8bf7d3 second
```

合并分支：git merge 分支名（将分支名合并到当前分支内 ）

正常合并：

```
22476@DESKTOP-ORPFREQ MINGW64 /d/桌面/实训笔记/GitDemo (master)
$ git merge newmaster
Already up to date.
```

冲突合并：

```
22476@DESKTOP-ORPFREQ MINGW64 /d/桌面/实训笔记/GitDemo (master)
$ git merge newmaster
Auto-merging text.txt
CONFLICT (content): Merge conflict in text.txt
Automatic merge failed; fix conflicts and then commit the result.

22476@DESKTOP-ORPFREQ MINGW64 /d/桌面/实训笔记/GitDemo (master|MERGING)
$
```

master|MERGING：正在合并中即合并冲突系统无法合并

产生冲突原因：合并分支时，两个分支在同一个文件的同一个位置有两套完全不同的修改，Gti无法决定使用哪个。必须人为决定。

合并冲突解决方法：（修改后的内容只是合并到的分支的内容，合并进来的分支内容不做改变）

①git status查看哪些文件冲突

```
22476@DESKTOP-ORPFREQ MINGW64 /d/桌面/实训笔记/GitDemo (master|MERGING)
$ git status
On branch master
You have unmerged paths.
  (fix conflicts and run "git commit")
  (use "git merge --abort" to abort the merge)

Unmerged paths:
  (use "git add <file>..." to mark resolution)
        both modified:   text.txt

no changes added to commit (use "git add" and/or "git commit -a")
```

②vim命令进入冲突文件

```
<<<<<<< HEAD
112
达娃大
213213
=======
iiiiii112
达娃大
123123123
123123123
>>>>>>> newmaster
```

<<<<HEAD 与====之间:当前分支代码

>>>>newmaster 与===之间：合并的分支代码

③把不要的代码和 <<<< ==== >>> 行删除并保存

④用git add 文件名 将其上传暂存区

```
22476@DESKTOP-ORPFREQ MINGW64 /d/桌面/实训笔记/GitDemo (master|MERGING)
$ git add text.txt
```

⑤提交本地库（使用git commit 时不要再带文件名）

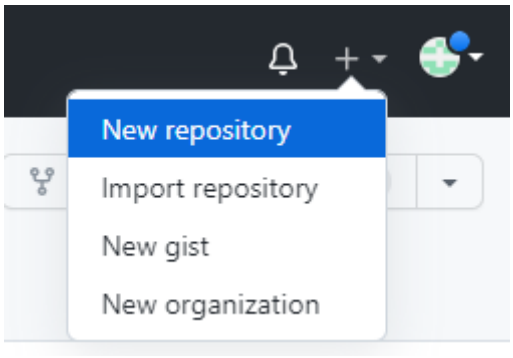
```
22476@DESKTOP-ORPFREQ MINGW64 /d/桌面/实训笔记/GitDemo (master|MERGING)
$ git commit -m "merge text"

[master 5f946d5] merge text

22476@DESKTOP-ORPFREQ MINGW64 /d/桌面/实训笔记/GitDemo (master)
$
```

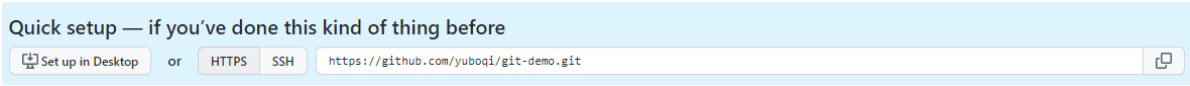
GitHub操作

1.进入GitHub，在右上角创建仓库



2.给仓库创建别名

创建远程地址别名：git remote add 别名 远程仓库地址




```
22476@DESKTOP-ORPFREQ MINGW64 /d/桌面/实训笔记/GitDemo (master)
$ git remote add git-demo https://github.com/yuboqi/git-demo.git
```

查看当前所有远程地址别名：git remote -v

```
22476@DESKTOP-ORPFREQ MINGW64 /d/桌面/实训笔记/GitDemo (master)
$ git remote -v
git-demo        https://github.com/yuboqi/git-demo.git (fetch)
git-demo        https://github.com/yuboqi/git-demo.git (push)
```

3.将本地库代码上传到GitHub上

本地库代码上传(必须本地库有代码，详情在Git操作里面)：git push 别名 分支

```
22476@DESKTOP-ORPFREQ MINGW64 /d/桌面/实训笔记/GitDemo (master)
$ git push git-demo master
Enumerating objects: 27, done.
Counting objects: 100% (27/27), done.
Delta compression using up to 12 threads
Compressing objects: 100% (17/17), done.
Writing objects: 100% (27/27), 1.97 KiB | 504.00 KiB/s, done.
Total 27 (delta 2), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (2/2), done.
To https://github.com/yuboqi/git-demo.git
 * [new branch]      master -> master
```

常见问题和解决方案:

问题1：

```
22476@DESKTOP-ORPFREQ MINGW64 /d/桌面/实训笔记/GitDemo (master)
$ git push git-demo master
fatal: unable to access 'https://github.com/yuboqi/git-demo.git/': SSL certific
te problem: unable to get local issuer certificate
```

原因：一般是因为服务器的SSL证书没有经过第三方机构的签署，所以才报错。

解决：使用 git config --global http.sslVerify “false”再git push即可

问题2：

```
22476@DESKTOP-ORPFREQ MINGW64 /d/桌面/实训笔记/GitDemo (master)
$ git push git-demo master
fatal: bad boolean config value '“false”' for 'http.sslverify'
```

原因：再解决问题1的时候 .gitconfig文件中的false赋值的时候加上了“”

解决：在C盘的用户文件夹下找到.gitconfig文件 将sslverify的值false

4.将远程库的代码拉取到本地库(拉取远程库更新内容)

拉取远程库代码： git pull 别名 分支

```
22476@DESKTOP-ORPFREQ MINGW64 /d/桌面/实训笔记/GitDemo (master)
$ git pull git-demo master
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), 637 bytes | 3.00 KiB/s, done.
From https://github.com/yuboqi/git-demo
 * branch            master     -> FETCH_HEAD
   7ee158d..dda0656  master     -> git-demo/master
Updating 7ee158d..dda0656
Fast-forward
 text.txt | 1 +
 1 file changed, 1 insertion(+)
```

常见问题和解决方案:

问题：

```
22476@DESKTOP-ORPFREQ MINGW64 /d/桌面/实训笔记/GitDemo (newmaster)
$ git pull git-demo newmaster
fatal: couldn't find remote ref newmaster
```

原因(个人认为)：pull代码将代码从远程仓库拉取到本地库时只能拉取到master主分支，其他分支无法拉取成功。

解决：将 git pull 别名 分支 中的 分支 改为 主分支 即newmaster改为master

5.克隆远程仓库到本地

①在需要的克隆文件的文件夹下面右键git bash here

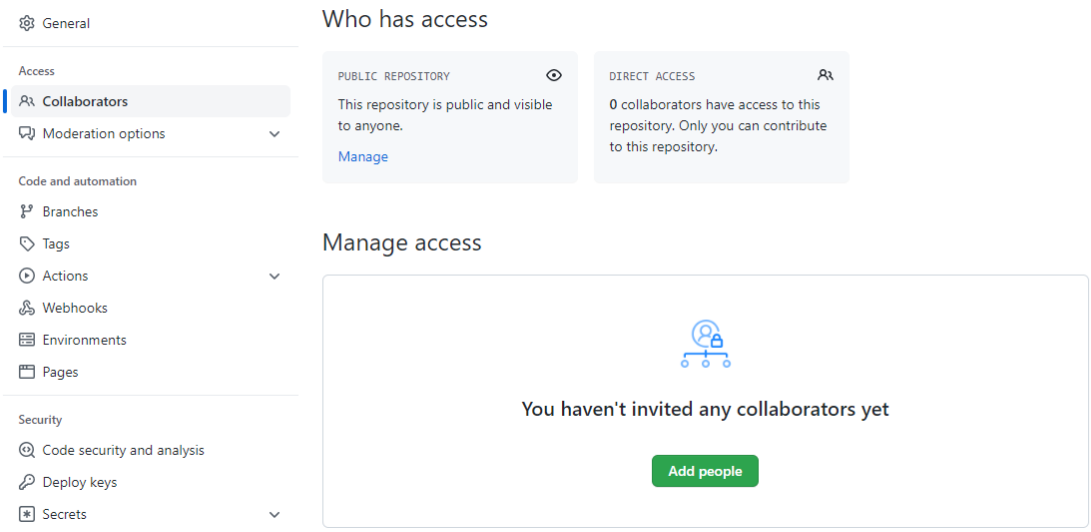
② 克隆远程仓库代码：git clone 远程仓库地址

```
22476@DESKTOP-ORPFREQ MINGW64 /d/桌面/实训笔记/GitClone
$ git clone https://github.com/yuboqi/git-demo.git
Cloning into 'git-demo'...
remote: Enumerating objects: 35, done.
remote: Counting objects: 100% (35/35), done.
remote: Compressing objects: 100% (21/21), done.
Receiving objects: 100% (35/35), done.
Resolving deltas: 100% (3/3), done.
remote: Total 35 (delta 3), reused 28 (delta 2), pack-reused 0
```

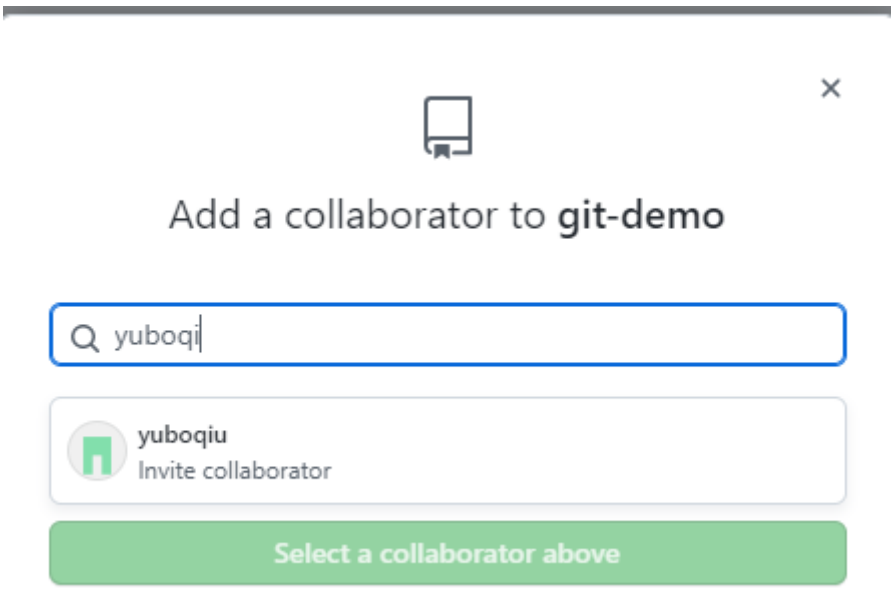
克隆执行的步骤：克隆代码 初始化本地仓库(代码的.git文件) 创建别名(但是别名为origin)

6.GitHub团队合作

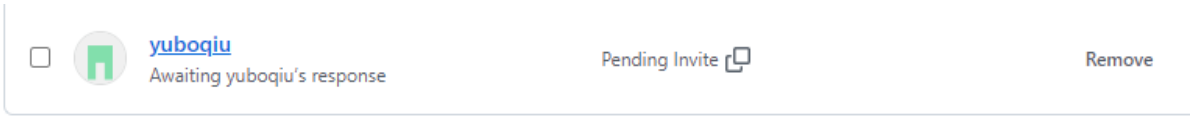
①创建仓库者进入仓库Settings 在collaborators中点击 Add people



②在弹出框输入需加入成员的用户名或者邮箱



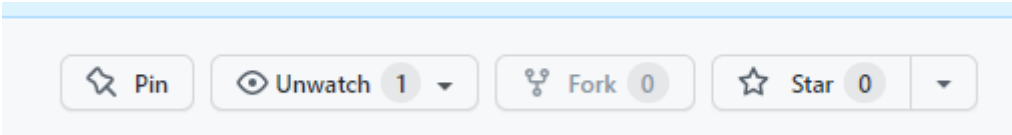
③复制所得的邀请凭证(Pending invite)发送给成员，成员打开并接受



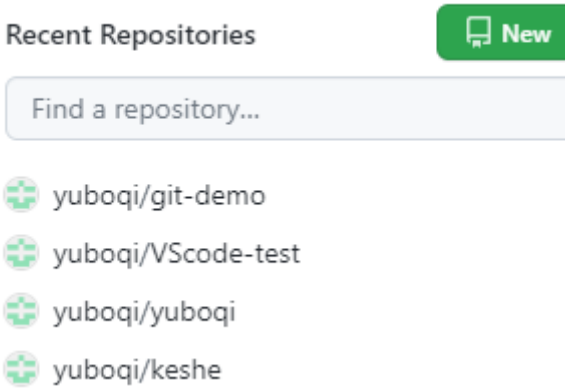
④成员可以使用链接进行push等相关操作

7.跨团队合作(未加入团队且修改代码)

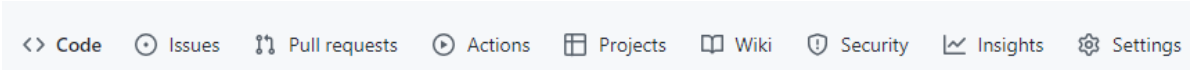
①未加入用户A通过搜索或者链接进入远程仓库，点击右上角Fork



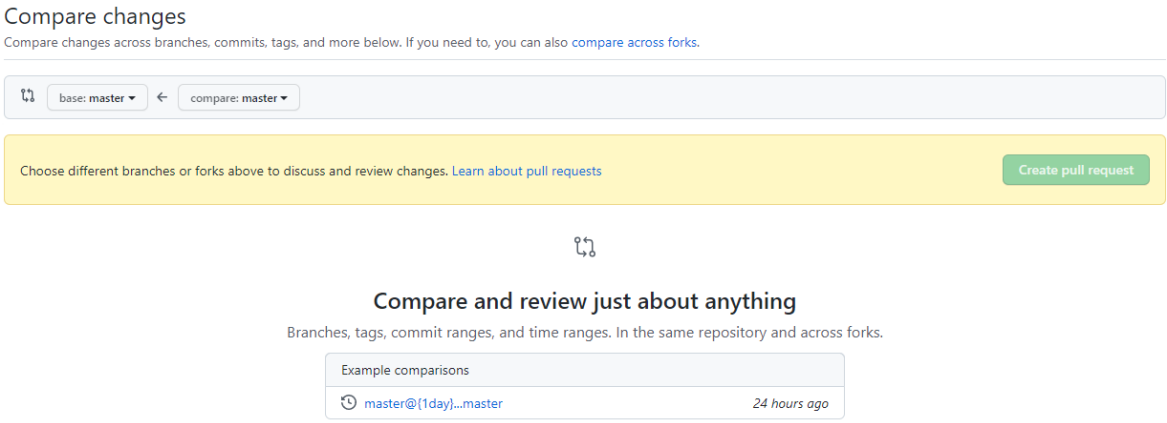
②用户A在自己的代码仓库列表找到Fork的仓库并进入



③用户A修改代码后点击左上角Pull requests请求



④点击New pull requsets按钮并点击Create pull request按钮



⑤在该远程仓库创建者的账号中点击Pull requsets，查看详情信息并同意合并

8.SSH免密登录

①进入C盘用户下的用户名文件夹右键Git Bash Here

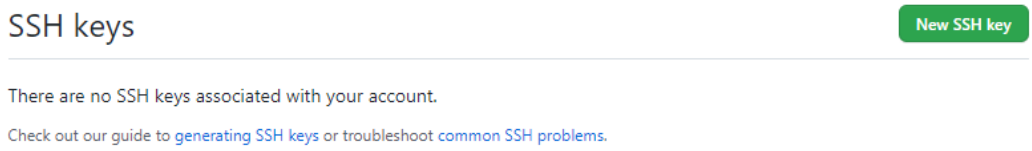
②输入 ssh-keygen -t -rsa -C 邮箱地址(GitHub的邮箱地址)然后三次回车(如果已经有.ssh文件的删除在操作即可)

```
22476@DESKTOP-ORPFREQ MINGW64 ~
$ ssh-keygen -t rsa -C 2247678912@qq.com
Generating public/private rsa key pair.
Enter file in which to save the key (/c/Users/22476/.ssh/id_rsa):
Created directory '/c/Users/22476/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /c/Users/22476/.ssh/id_rsa
Your public key has been saved in /c/Users/22476/.ssh/id_rsa.pub
The key fingerprint is:
SHA256:F1EDFSJiiGAGttJhOD2369BM8iKjyILp8ra2S8hAkzQ 2247678912@qq.com
The key's randomart image is:
+---[RSA 3072]----+
|+Eo. .o . ++=.   |
|BoB.o. . . o .   |
|.*.o .    .      |
|o o o      .     |
|.  * .  S .      |
|=.o =    .       |
|*=.+             |
|Boo .            |
|+==+             |
+----[SHA256]-----+
```

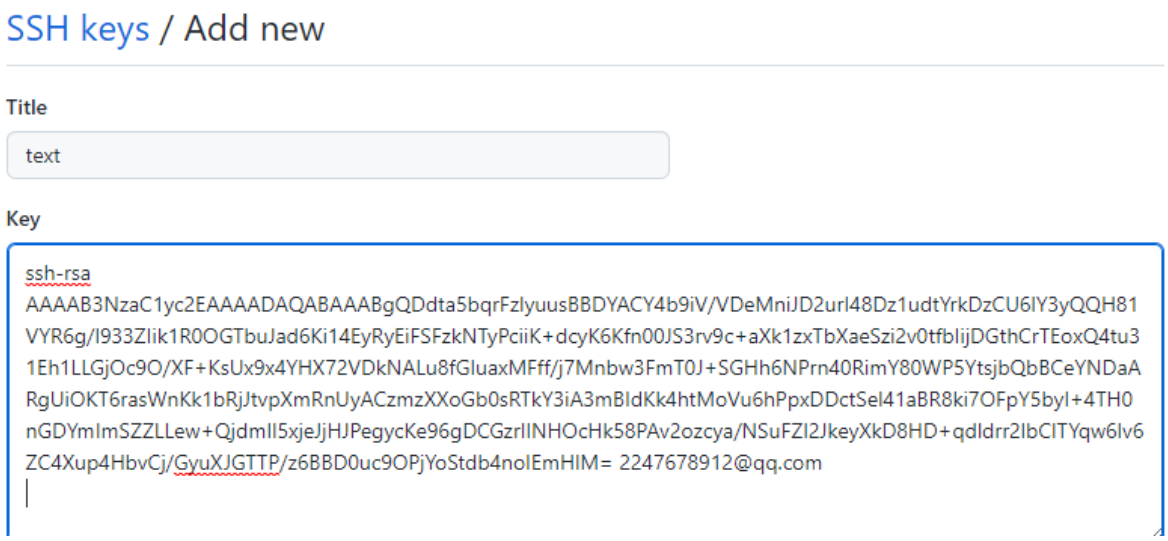
③进入.ssh文件打开id_rsa.pub文件并复制公钥代码

```
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABgQDdta5bqrFzlyuusBBDYACY4b9iV/VDeMniJD2urI48
Dz1udtYrkDzCU6lY3yQQH81VYR6g/l933ZIik1ROOGTbuJad6Ki14EyRyEiFSFzkNTyPciiK+dcyK6Kf
nOOJS3rv9c+aXk1zxTbXaeSzi2vOtfbIijDGthCrTEoxQ4tu31Eh1LLGjOc9O/XF+KsUx9x4YHX72VDk
NALu8fGIuaxMFff/j7Mnbw3FmTOJ+SGHh6NPrn4ORimY8OWP5YtsjbQbBCeYNDaARgUiOKT6rasWnKk1
bRjJtvpXmRnUyACzmzXXoGbOsRTkY3iA3mBIdKk4htMoVu6hPpxDDctSel41aBR8ki7OFpY5byI+4THO
nGDYmImSZZLLew+QjdmIl5xjeJjHJPegycKe96gDCGzrlINHOcHk58PAv2ozcya/NSuFZl2JkeyXkD8H
D+qdldrr2lbCITYqw6lv6ZC4Xup4HbvCj/GyuXJGTTP/z6BBDOuc9OPjYoStdb4noIEmHlM= 2247678
912@qq.com
```

④进入仓库创建者的个人信息，右上角头像点击Settings，点击SSH and GPG keys添加SSH keys。



⑤将公钥复制进去然后 Add SSH key即可



⑤之后就可以用SSH链接进行pull push操作免密登录