# Git

## Git概述

Git(读音为/gɪt/。)是一个开源的分布式版本控制系统，可以有效、高速的处理从很小到非常大的项目版本管理。 [1]  Git 是 Linus Torvalds 为了帮助管理 Linux 内核开发而开发的一个开放源码的版本控制软件。

Torvalds 开始着手开发 Git 是为了作为一种过渡方案来替代 BitKeeper，后者之前一直是 Linux 内核开发人员在全球使用的主要源代码工具。开放源码社区中的有些人觉得BitKeeper 的许可证并不适合开放源码社区的工作，因此 Torvalds 决定着手研究许可证更为灵活的版本控制系统。尽管最初 Git 的开发是为了辅助 Linux 内核开发的过程，但是我们已经发现在很多其他自由软件项目中也使用了 Git。例如 很多 Freedesktop 的项目迁移到了 Git 上。

## Git入门

### 2.1Git的安装：

#### Wondods版：

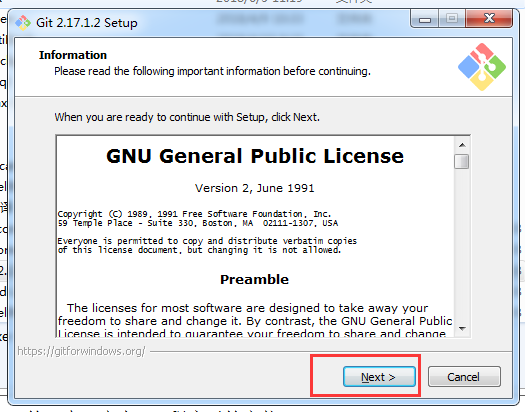
下载

网站：<https://git-scm.com/downloads>，选择Windows版

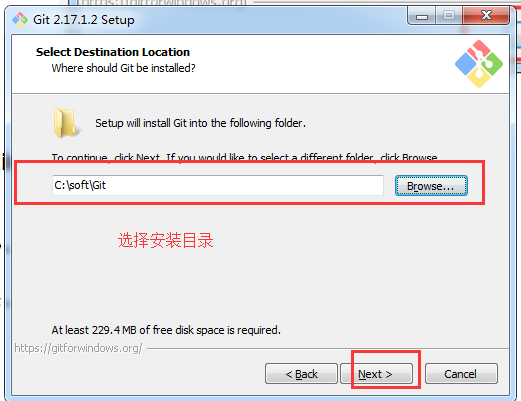


下载完成后:

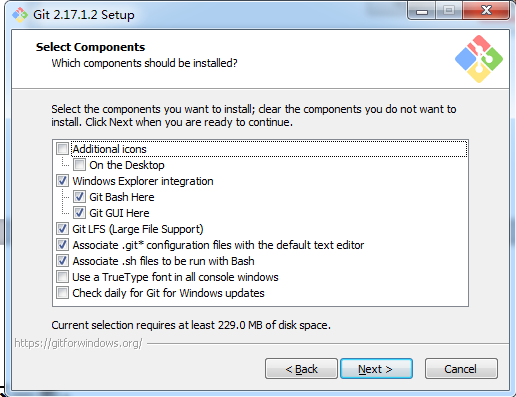
下一步：点击exe程序开始安装，点击next



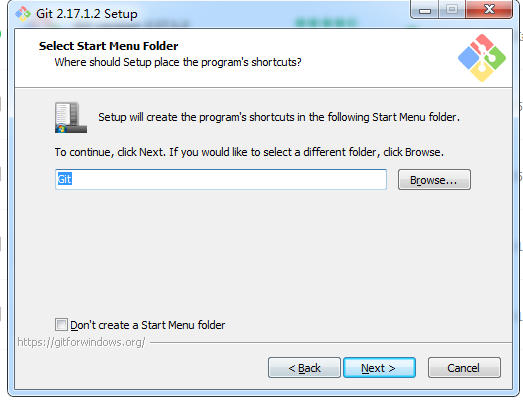
下一步：选择安装目录，点击next



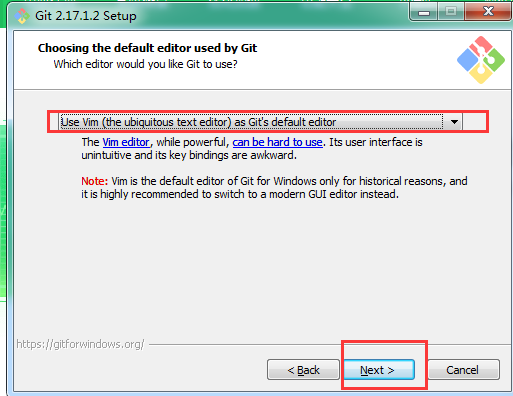
下一步：点击next



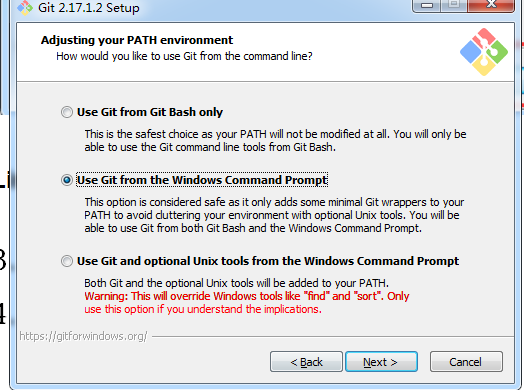
下一步：点击next



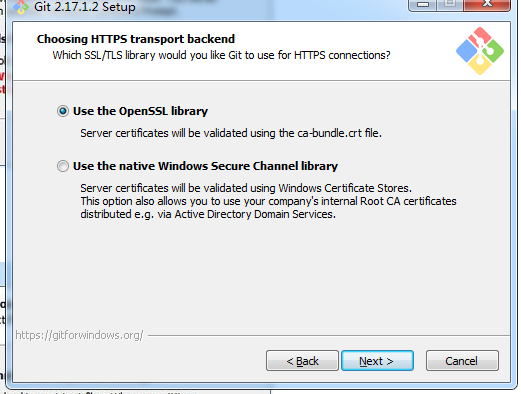
下一步：不用管，点击next



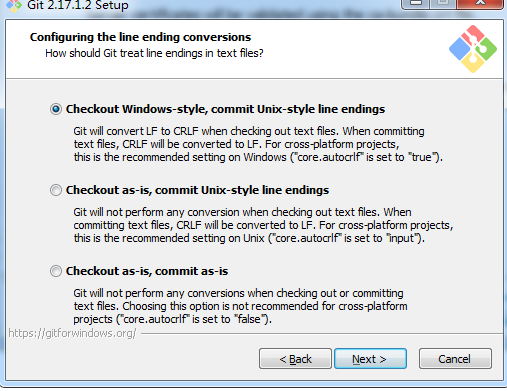
下一步：由于个人使用，所有选择第二项



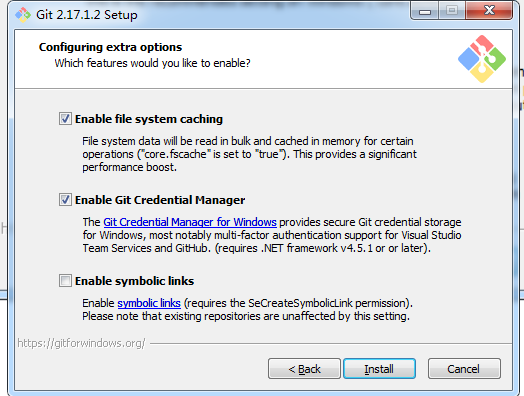
下一步：点击next



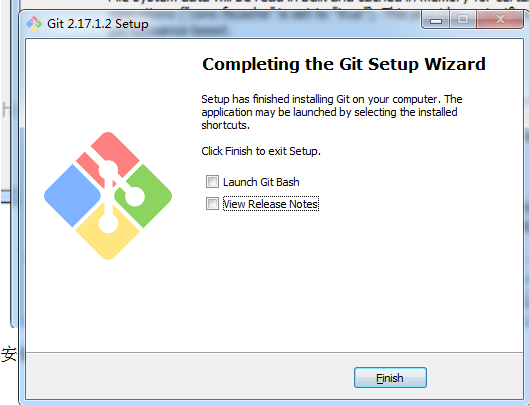
下一步：默认选第一项，点击next



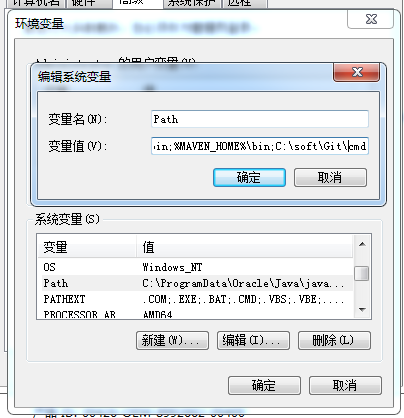
下一步：默认选项，点击intall安装



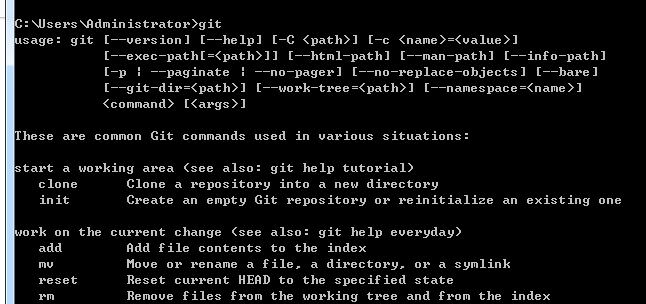
下一步：点击finish,安装完成



安装完成后配置环境变量，配置到git安装目录里面的cmd目录，此处我用的是2.1.7版本，环境变量是自动配置的



测试：cmd控制台输入git，出现如下内容即表示安装成功。



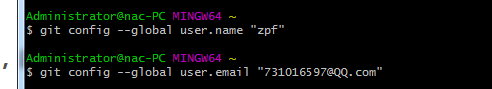
#### Linux版：

### 2.2基础配置:

安装完成进行配置：

因为Git是分布式版本控制系统，每个机器都需要一个标识，也就是：你的名字和Email地址;

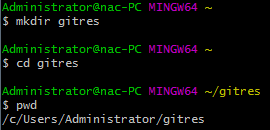
|  |
| --- |
| $ git config --global user.name "zpf"  $ git config --global user.email "731016597@QQ.com" |



### 2.3创建版本库

#### 2.3.1创建资源库所在目录

|  |
| --- |
| $ mkdir gitres  $ cd gitres  $ pwd |



#### 2.3.2通过git init命令把这个目录纳入GIt，使用git管理该目录

|  |
| --- |
| $ git init |



！注意：版本控制系统只能跟踪文本文件的改动，例如txt文件、网页和所有的程序代码。版本控制器可以告诉你你的每次改动，但是图片、视频等二进制文件没办法跟踪，只知道文件大小的改变。在Windows下word格式也是二进制文件，因此我们如果要真正使用版本控制系统，就要以纯文本方式来编写文件，并且强烈建议使用标准的UTF-8编码。并且编辑文本文件我们推荐Notepad++，并要记得设置默认编码为UTF-8 without BOM.

### 2.4添加文件到资源库

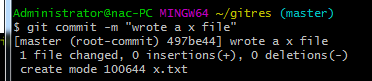
#### 2.4.1，创建一个txt文本文件到gitres目录下，用git add 把文件纳入到git管理

|  |
| --- |
| $ git add x.txt |



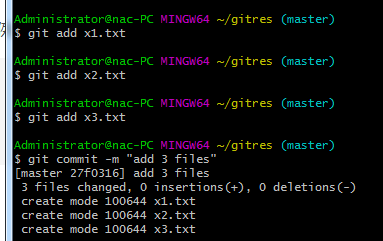
#### 2.4.2用git commit命令告诉git，把文件提交到仓库

|  |
| --- |
| $ git commit -m “wrote a x file” |



git commit命令：-m后面输入的是本次提交的说明，可以输入任意有意义的内容，这样方便从历史记录中找到改动记录。

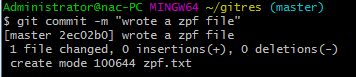
#### 2.4.3提交多个文件到git仓库



### 2.5版本退回

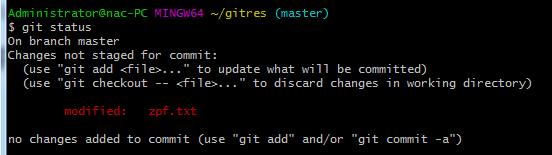
#### 2.5.1在gitres目录下创建一个zpf.txt文件，里面添加内容zpf,使用git add和git commit命令将文件添加到git仓库。





#### **2.5.2使用git status命令查看状态**

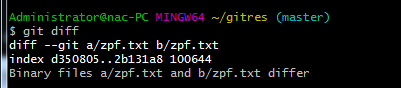
|  |
| --- |
| $ git status |



执行git status命令，会打印提示消息。上述提示消息中告诉我们文件被修改，并没有提交。

#### **2.5.3使用git diff 命令 查看修改详细内容**

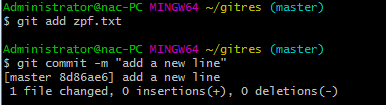
|  |
| --- |
| $ git diff |



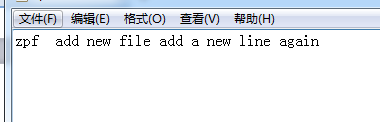
#### **2.5.4使用git add zpf.txt 命令和git commit -m “add a new line.”提交修改；然后在文本中新添加内容“add a new line again”，以上述命令重新执行一边。**

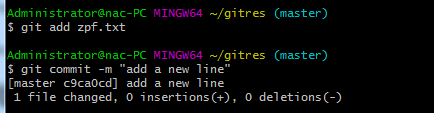
|  |
| --- |
| $ git add zpf.txt  $ git commit -m "add a new line" |

第一次执行



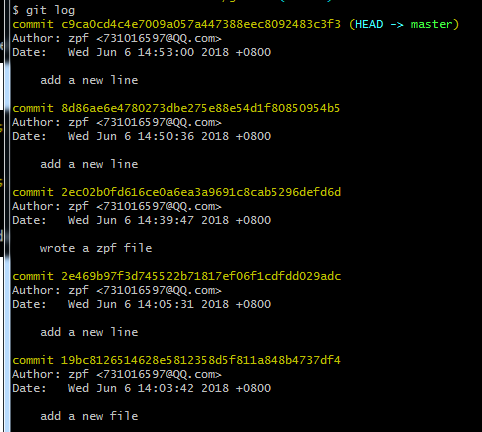
修改后执行





#### **2.5.5使用git log查看修改记录。**

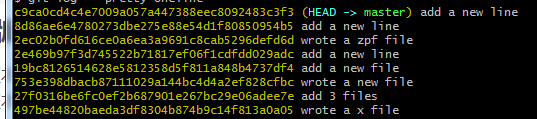
|  |
| --- |
| $ git log |



上述命令git log执行后会打印出具体日志信息。从上述信息中我们可以得到每次提交的记录，记录中包含提交的描述性信息例如“wrote a test file”，提交时间，提交人的具体信息等。而“commit 8d86ae6e4780273dbe275e88e54d1f80850954b5”则是我们每次提交的提交版本号，也称之提交的记录ID。

如果我们不需要提交人，提交时间等信息，我们也可以以一种更简洁的方式查看日志，只需要加上“--pretty=online”参数即可,可以看到7条日志;

|  |
| --- |
| $ git log --pretty=oneline |



#### **2.5.6使用git reset命令回退版本**

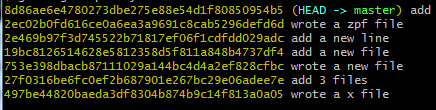
在工作当中，我们不可避免的使用回退版本，例如一个模块负责人提交了一部分代码，在项目负责人发编译发布版本前离开了工作岗位，项目负责人在编译发布版本的时候发现模块负责人工作失误造成项目无法编译通过，为了不耽误整个版本发布的工作，负责人不得不回退版本。

|  |
| --- |
| $ git reset --hard HEAD^ |

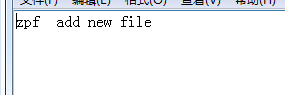


上述命令中Head在Git中的概念是一个指向你正在工作中的本地分支的指针（可以把HEAD想象为当前分支的别名），其所对应的分支本质是个指向commit对象的可变指针。截止到目前的学习中，在我们若干差提交后，我们已经有了一个指向最后一次提交的master分支，它在每次提交的时候都会自动向前移动

使用git log --pretty=oneline查看：此时只有6条日志记录；



这个时候我们发现记录中已经少了一条，并且打开文件也会发现，最后修改的内容已经不见了。



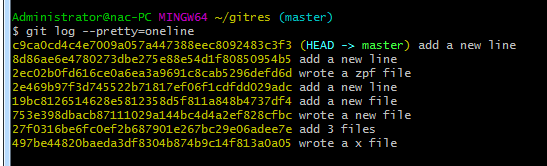
还拿上面的例子来说，如果项目管理员发现编译不能通过的原因不是模块负责人的误操作引起的，而是有其他原因引起的，并且当前需要发布的版本中需要模块负责人的改动，但是版本已经回退了，能否再回退到回退前的版本呢？

答案是肯定的，我们只需要知道我们需要回退到的那个版本号commit id即可（例如当前的命令窗口没有关闭，我们可以轻轻滑动滚轴就可以看到之前的版本号），或者知道前面一部分也可以。注意是版本id的前面部分类容

|  |
| --- |
| $ git reset --hard c9ca0c |

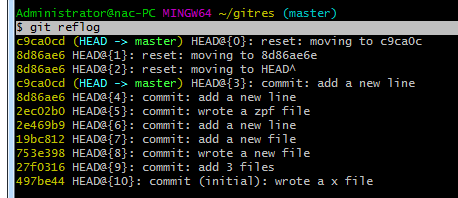


使用git log --pretty=oneline查看：此时之前回退的版本又有了，说明回退成功



但是如果当前的命令窗口已经关闭了，我们无法在命令窗口中查看我们之前打印的版本号了怎么办？git也为我们提供了一个命令来记录我们每次执行的命令“git reflog”

|  |
| --- |
| $ git reflog |



上述提示信息中最前面的字符串即是我们需要的版本号。

总结：

1. 执行git add 命令时，是将当前本底的文件添加到一个缓存区域内，只有当执行git commit时，数据才会提交到git仓库
2. 每次操作都会生成一个版本id ，head^代表当前版本

### 2.6撤销修改 git checkout

我们不能绝对的保证在日常的工作中不会出任何差错，如果我们在提交代码前发前发现有错误，但是我们没有执行了git add命令把修改的文件添加到了暂存区，那么能否撤销此次修改呢？

答案也是肯定的。

|  |
| --- |
| $ git checkout -- zpf.txt |

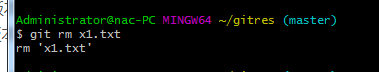
执行上述命令，没有任何提示消息。  
而且我们也可以使用命令git reset HEAD file把暂存区的修改撤掉（unstage），重新放回工作区。

|  |
| --- |
| $ git reset HEAD zpf.txt |



### 2.6删除文件

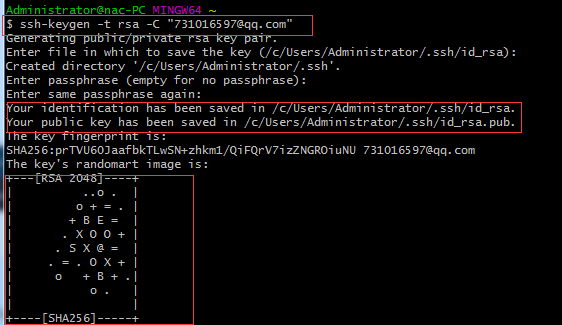
|  |
| --- |
| $ git rm x1.txt |



|  |
| --- |
| 1. 拉取远程的Repo到本地（如果已经在本地，可以略过） $ git clone xxxxxx 2. 在本地仓库删除文件 $ git rm 我的文件 3. 在本地仓库删除文件夹 $ git rm -r 我的文件夹/ 4. 此处-r表示递归所有子目录，如果你要删除的，是空的文件夹，此处可以不用带上-r。 5. 提交代码 $ git commit -m"我的修改" 6. 推送到远程仓库（比如GitHub） $ git push origin xxxxxx |

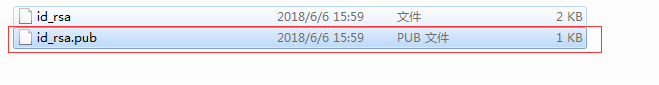
### **2.7创建远程仓库**

#### **2.7.1输入ssh-keygen -t rsa -C “731016597@qq.com”,下面的信息不会一次全部显示出来，需要一直回车，一直到下面的图出现位置。**



这里生成了两个文件在.ssh 分别是id\_rsa和id\_rsa.pup

我们需要的密匙在id\_rsa.pup中

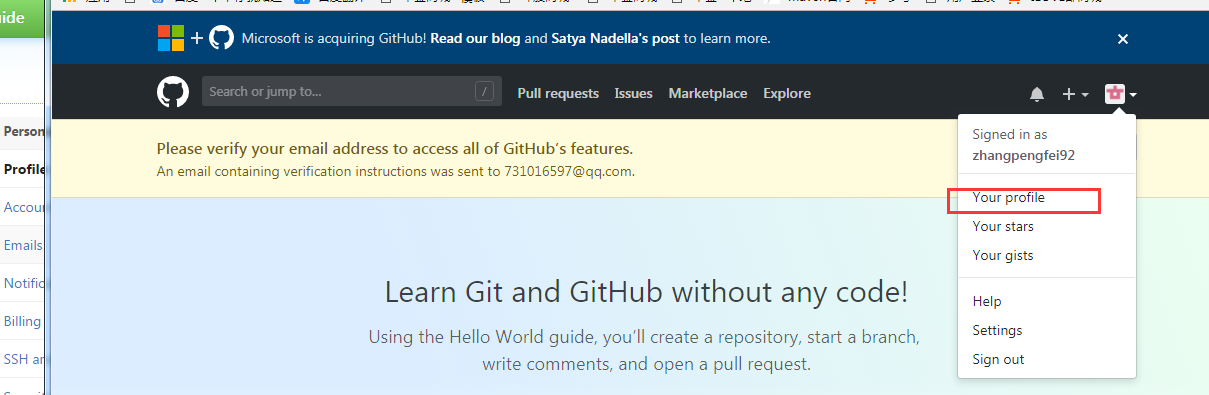




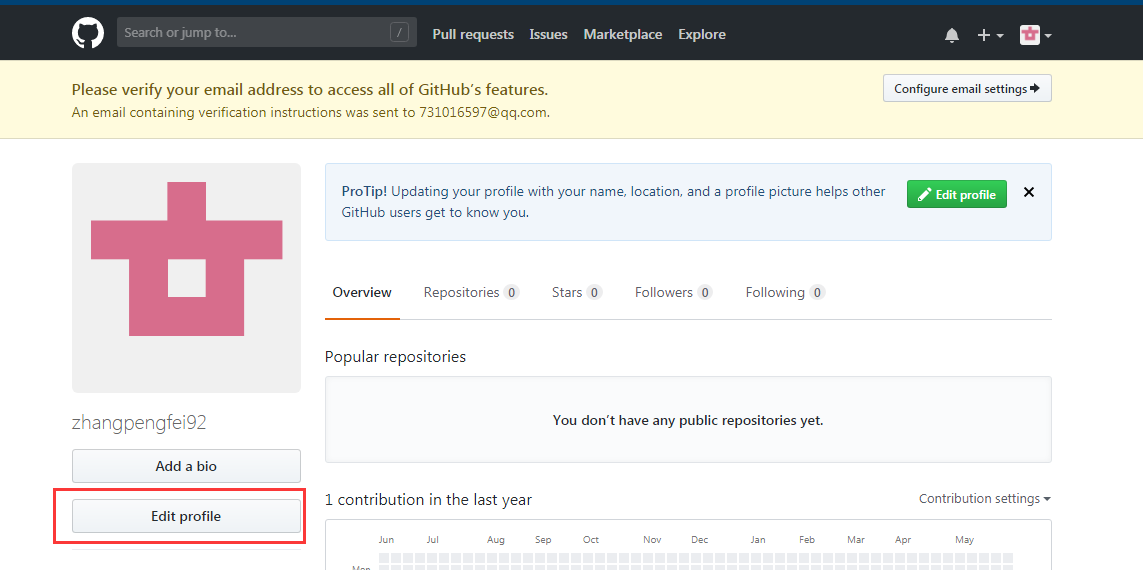
#### 2.7.2申请GitHub账户，并登陆。(此处略过)

#### 2.7.3找到 ssh -key输入秘钥（安装下图遭到输入密匙位置·）

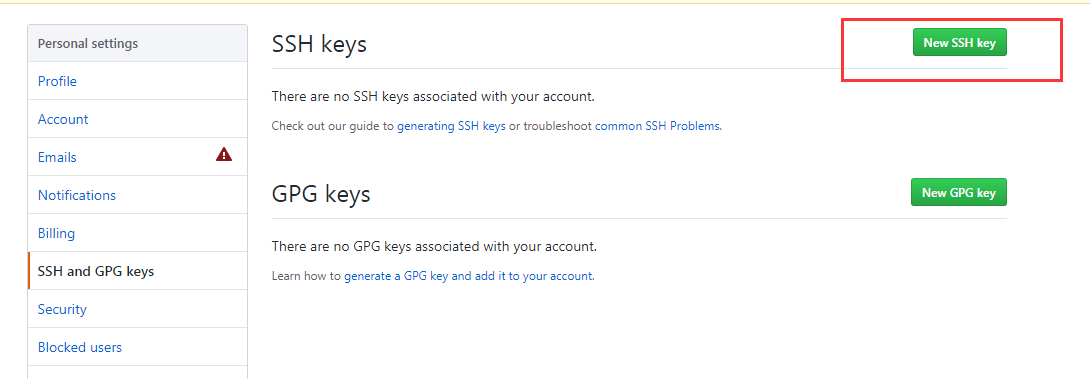
第一步：



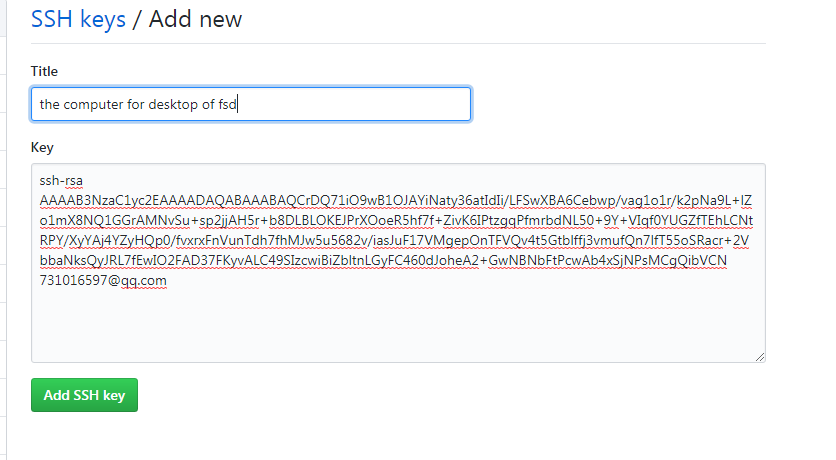
第二步：



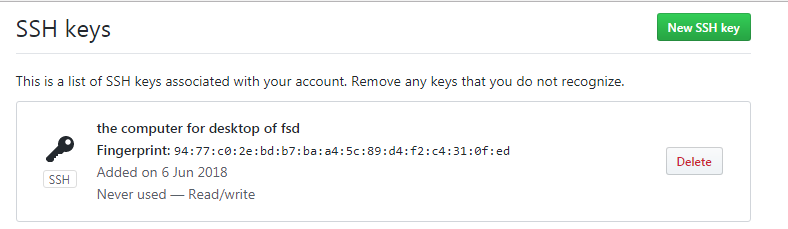
第三步：



第四步：



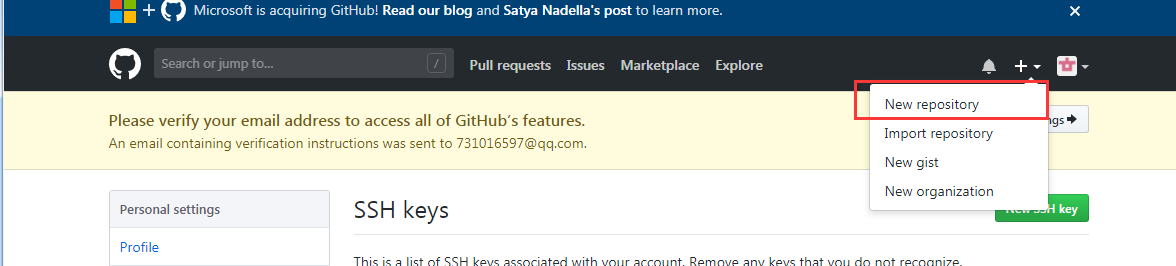
第五步：



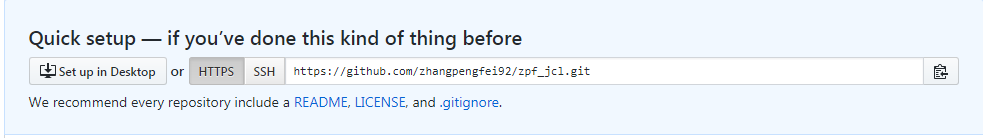
至此，GitHub仓库注册成功

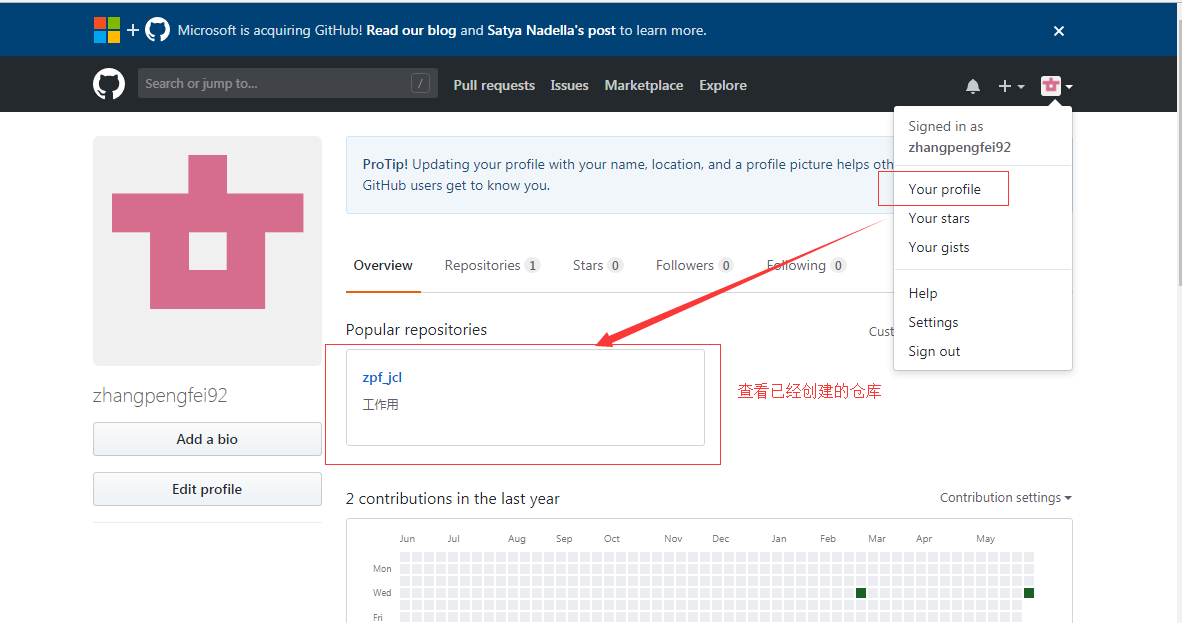
#### 2.7.4创建个人远程仓库

第一步：注意在创建仓库过程中可能需要邮箱验证，按指示操作即可。



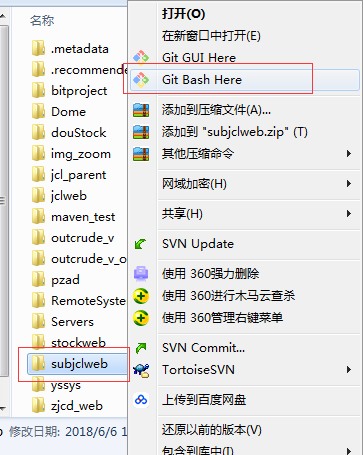






### **2.8将本地的一个项目上传到git远程仓库上**

#### 2.8.1找到要提交的项目，执行Git base here



会弹出一个Git操作窗口



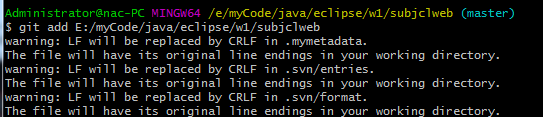
#### 2.8.2执行git init 命令

|  |
| --- |
| $ git init |



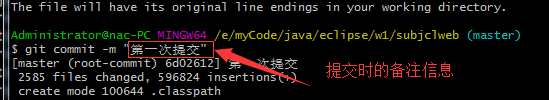
#### 2.8.3执行 git add 命令

|  |
| --- |
| $ git add E:/myCode/java/eclipse/w1/subjclweb |



执行该命令后，会将整个项目纳入到缓存中，等待提交；

#### 2.8.4执行git commit提交到本地仓库



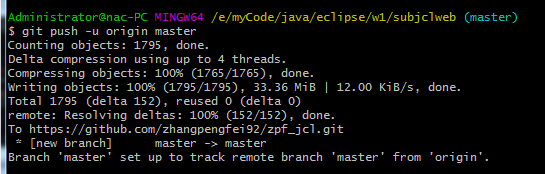
#### 2.8.5执行 git remote add origin--连接远程仓库

|  |
| --- |
| $ git remote add origin https://github.com/zhangpengfei92/zpf\_jcl.git |

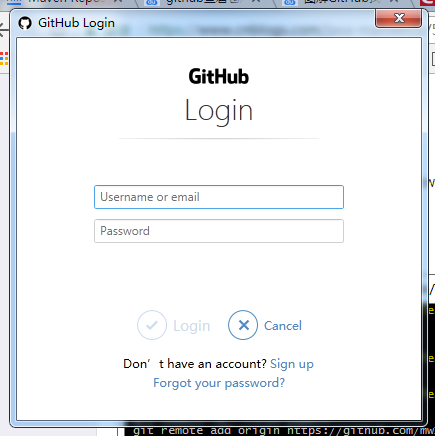


#### 2.8.6执行 git push -u origin master，将本地库同步到远程仓库

|  |
| --- |
| $ git push -u origin master |

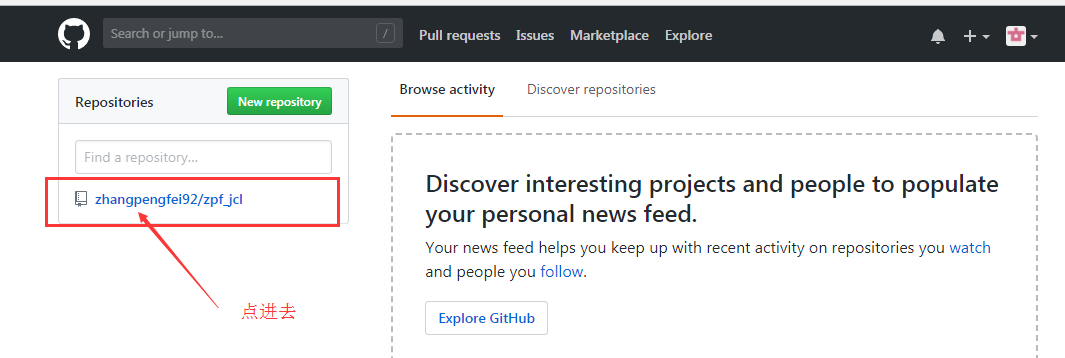


输出此命令后弹簧输入密码的弹框，输入账户密码，完成同步。

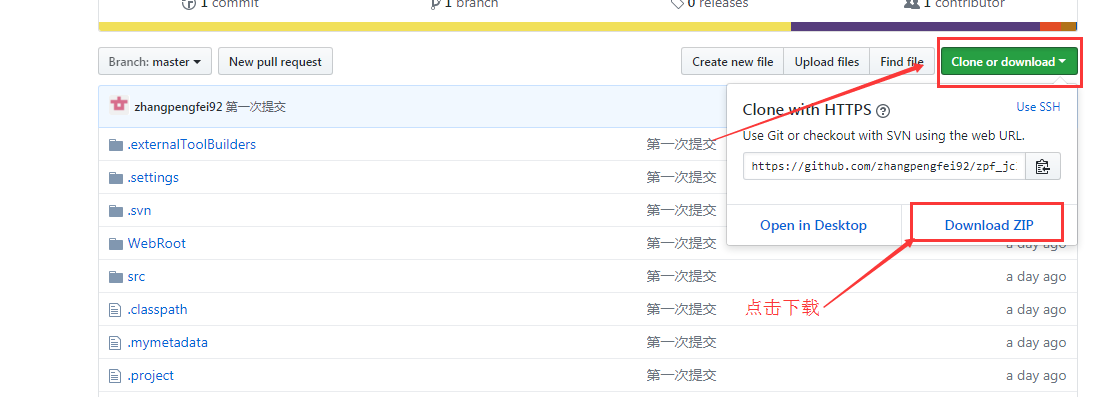


### **2.9将远程git仓库的项目下载到本地**

#### 2.9.1登录自己的GitHub账户，点击自己的远程仓库。



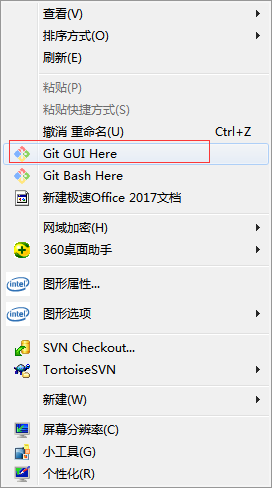
#### 2.9.2点击下载



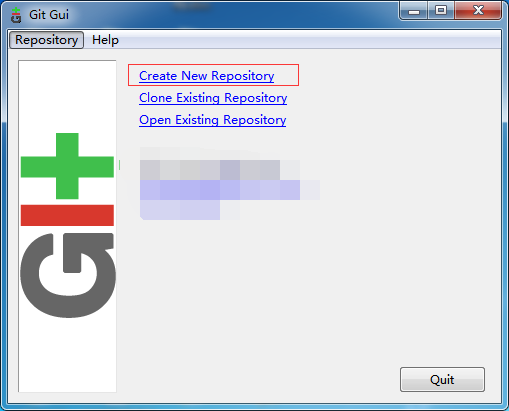
### 2.10 git-gui的使用

#### 2.10.1创建本地仓库

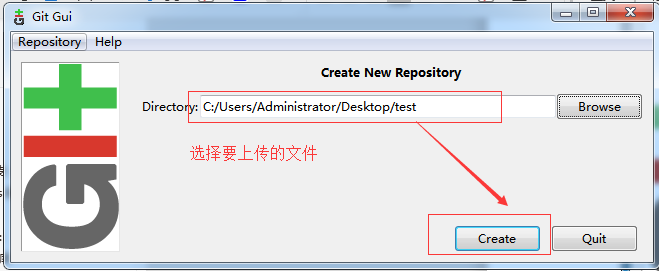
2.10.1.1

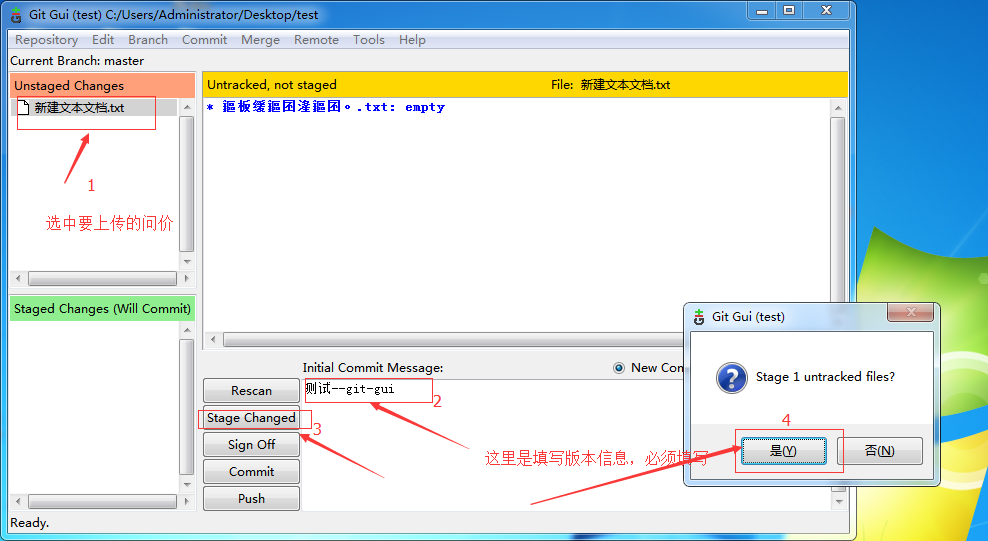


2.10.1.2



2.10.1.3





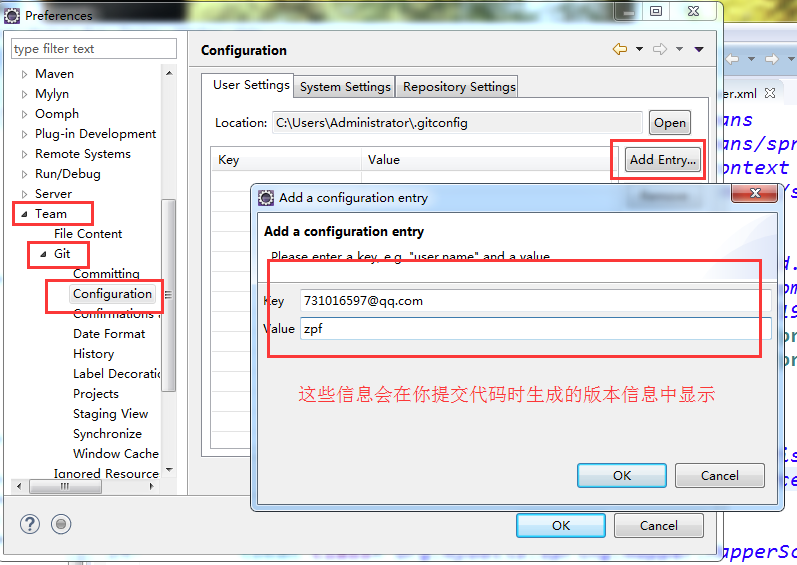
#### 2.10.2

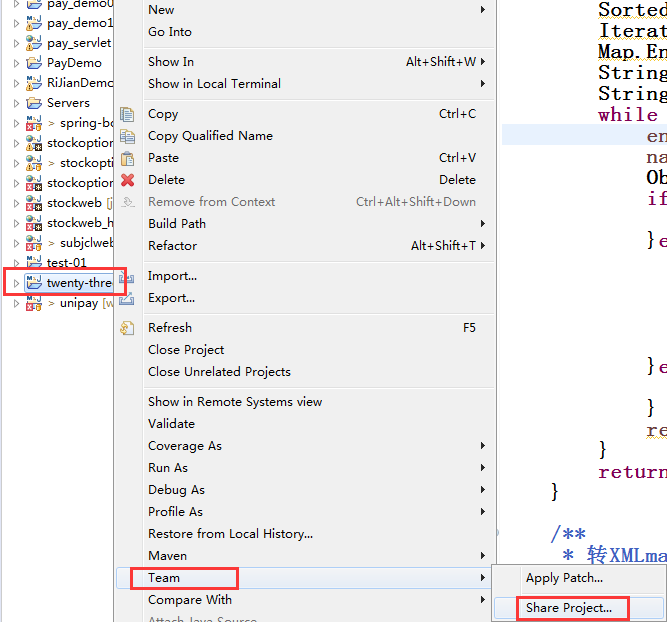
#### 2.10.3

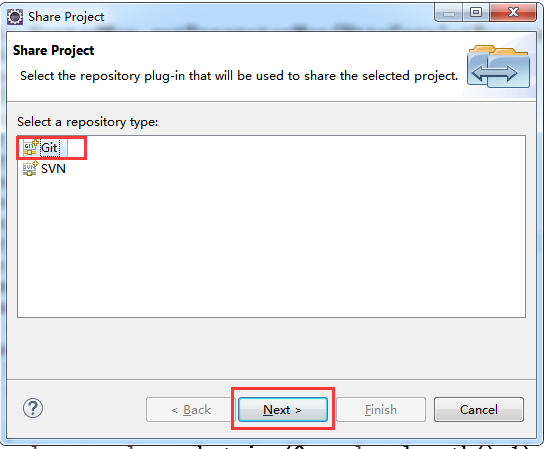
#### 2.10.4

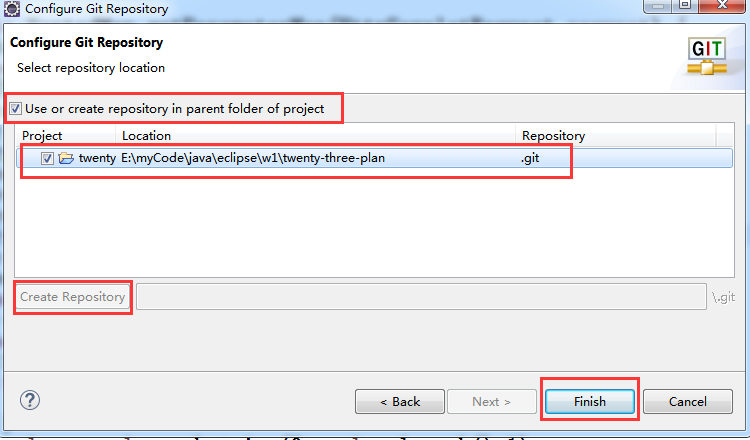
## 在eclipse中使用Git

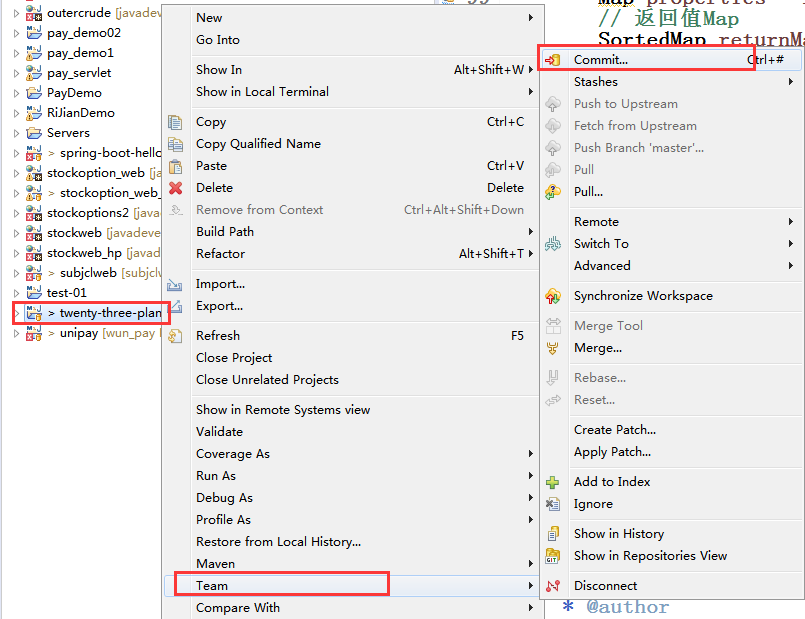
3.1

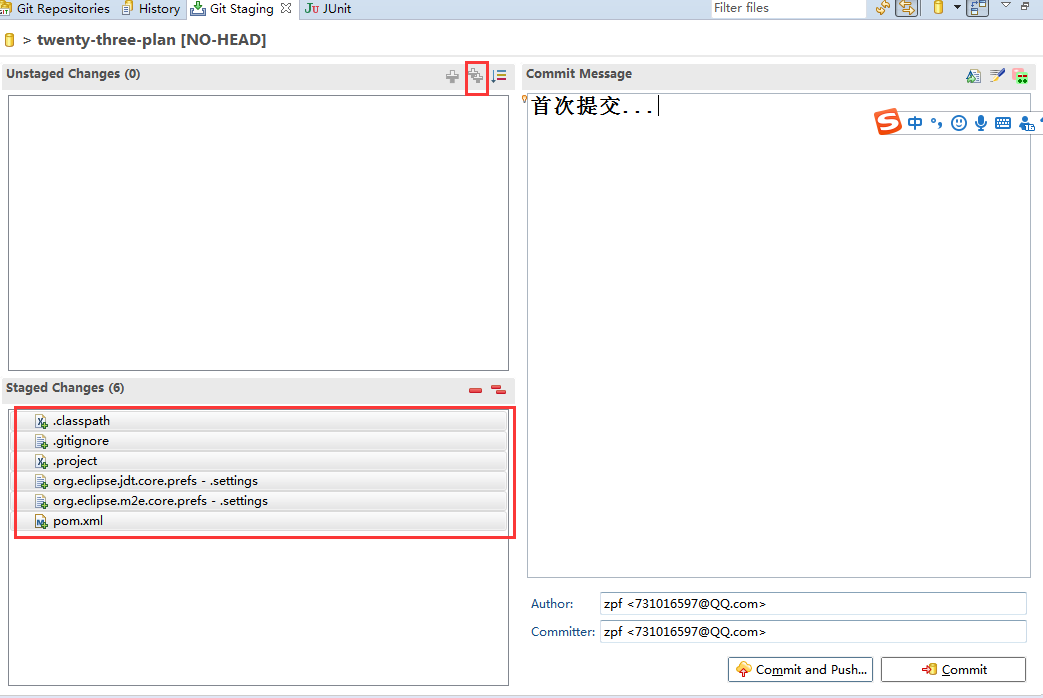


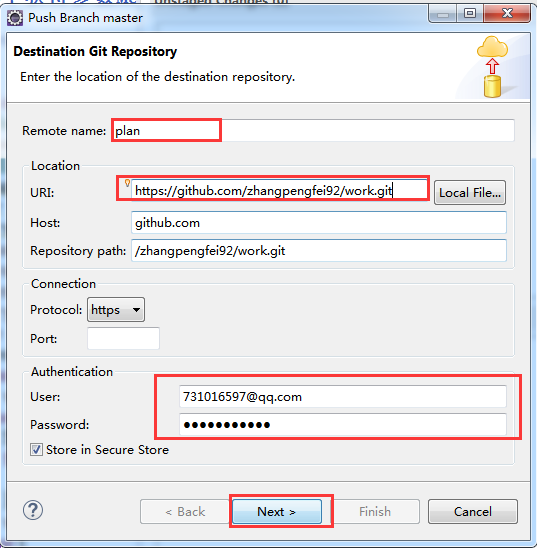


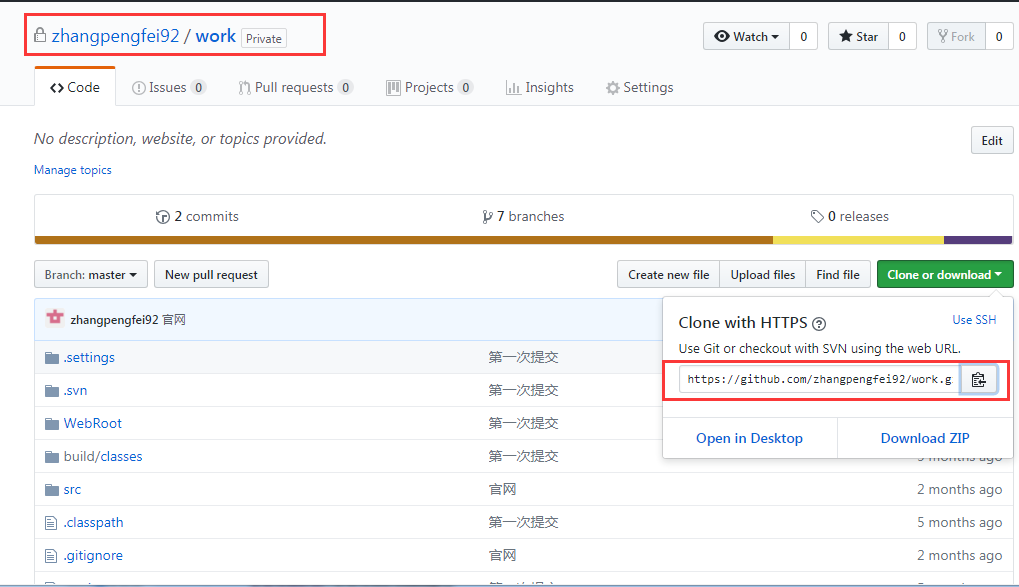




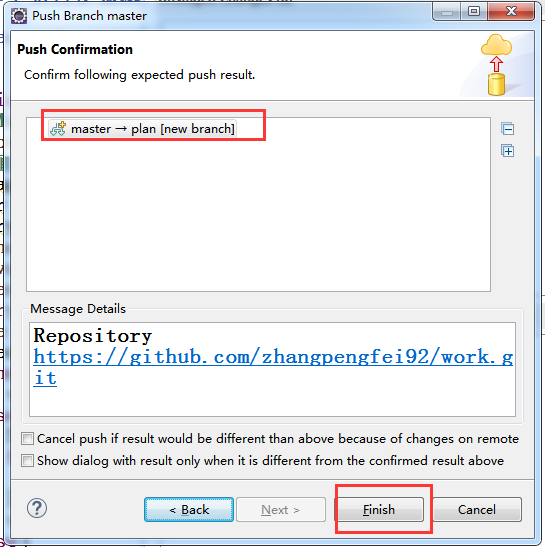






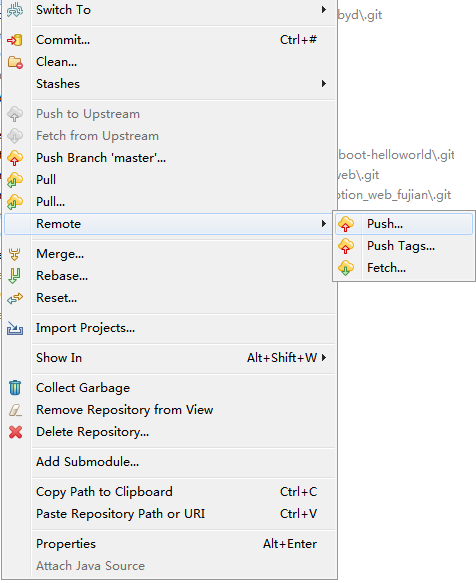


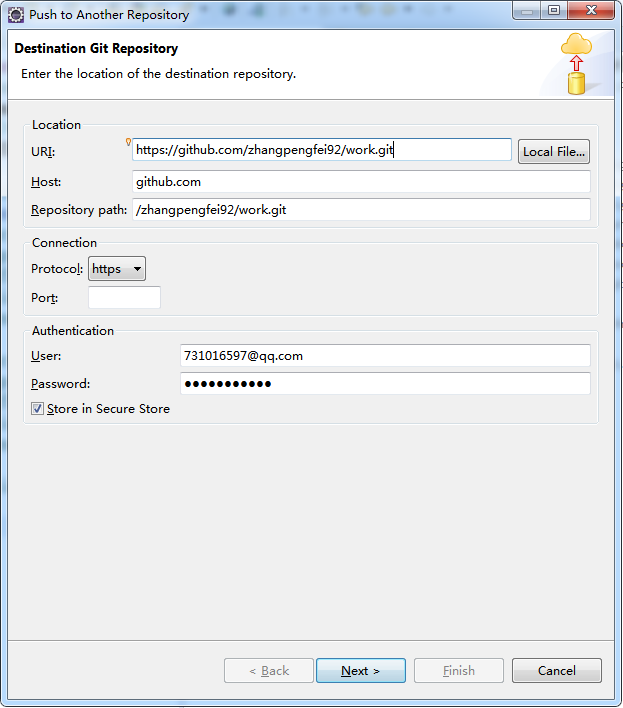


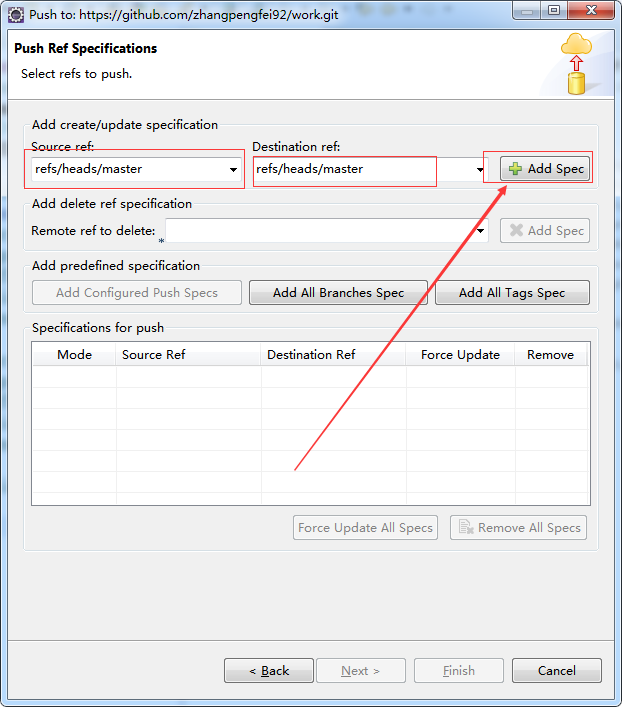


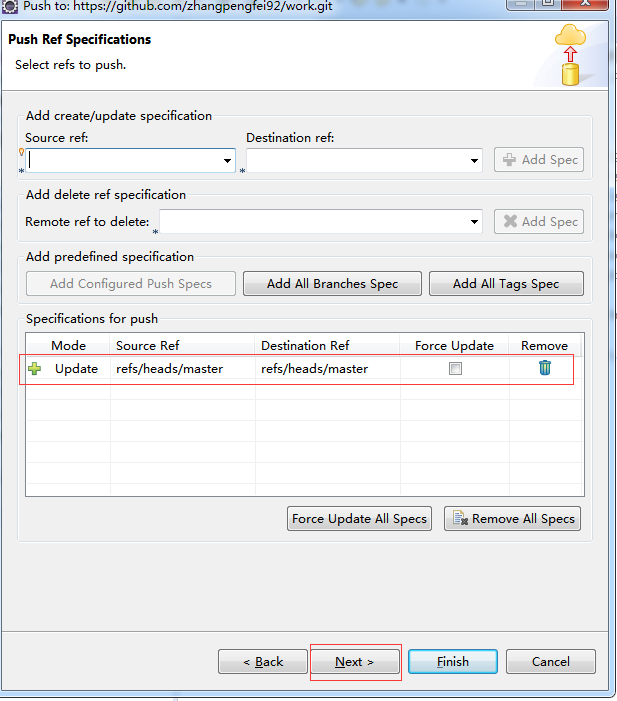


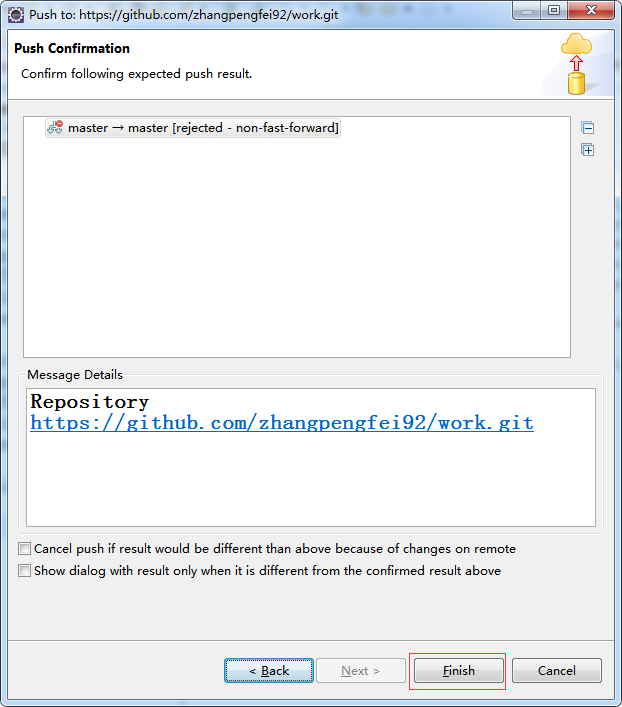
把已上传到本地eclipse-git的项目

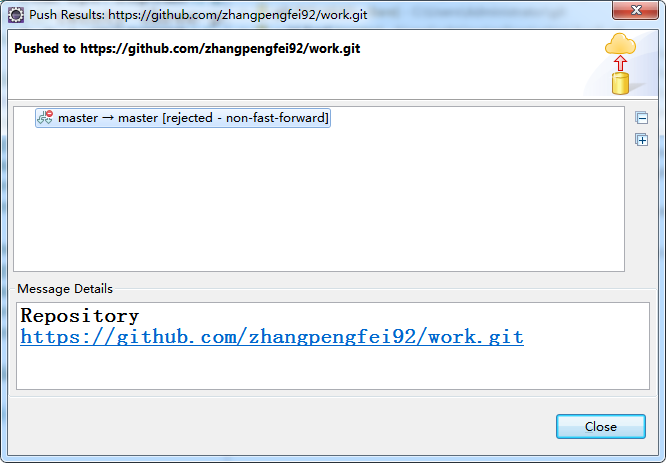












## 分布式中的Git

## 深入Git