



江西財經大學

实 验 报 告

课程名称: 软件工程综合实训

项目名称: Git 使用说明

组 长: 0174087 邬旻星辰

成员: 0174089 李伟波 0174055 刘家铭

0174115 张龙飞 0174025 旷琳蕴

时间: 2020 年 2 月

任务分工	小组成员	贡献率
1 软件准备	0174025 旷琳蕴	20%
2 Git 使用与常用命令	0174087 邬旻星辰	20%
3 标签管理	0174115 张龙飞	20%
4 分支管理	0174055 刘家铭	20%
5 MyEclipse 的使用	0174089 李伟波	20%

目录

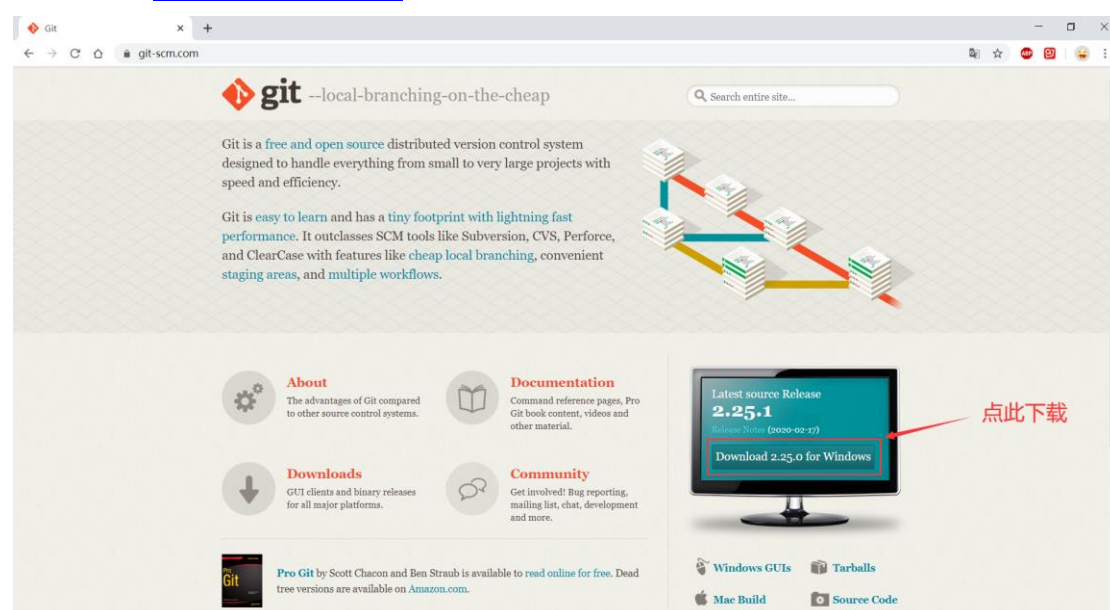
1 软件准备	- 3 -
1.1 下载并安装 Git	- 3 -
1.2 下载并安装 SourceTree	- 10 -
1.3 下载并安装 MyEclipse	- 15 -
2 Git 使用与常用命令	- 24 -
2.1 配置用户名及邮箱	- 24 -
2.2 创建版本库	- 24 -
2.3 文件的基本操作	- 25 -
2.4 Git 撤销修改和删除文件操作	- 29 -
3 标签管理	- 32 -
3.1 标签类型	- 32 -
3.2 创建标签	- 33 -
3.3 列出标签	- 33 -
3.4 查看标签	- 34 -
3.5 删除标签	- 35 -
3.6 后期打标签	- 35 -
3.7 共享标签	- 35 -
3.8 检出标签	- 37 -
4 分支管理	- 37 -
4.1 分支的基本命令	- 38 -
4.2 创建分支	- 38 -
4.3 合并分支	- 39 -
4.4 删除分支	- 39 -
4.5 解决冲突	- 40 -
4.6 分支管理	- 42 -
4.7 bug 分支	- 43 -
4.8 多人协作	- 44 -
5 MyEclipse 的使用	- 47 -
5.1 MyEclipse 上传项目至 GitHub	- 47 -
5.2 GitHub 导入项目至 MyEclipse	- 54 -

1 软件准备

1.1 下载并安装 Git

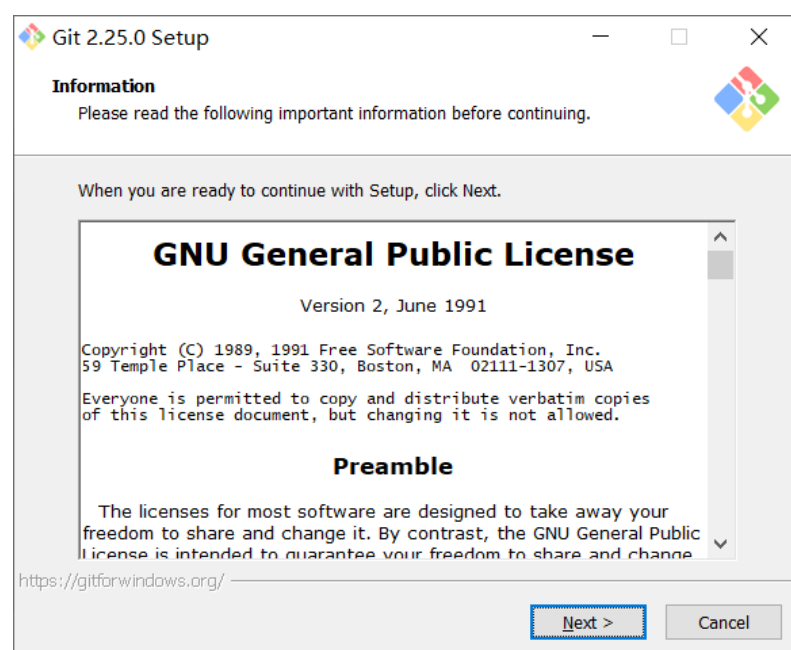
第一步：先从官网下载最新版本的 Git

官网地址：<https://git-scm.com/>



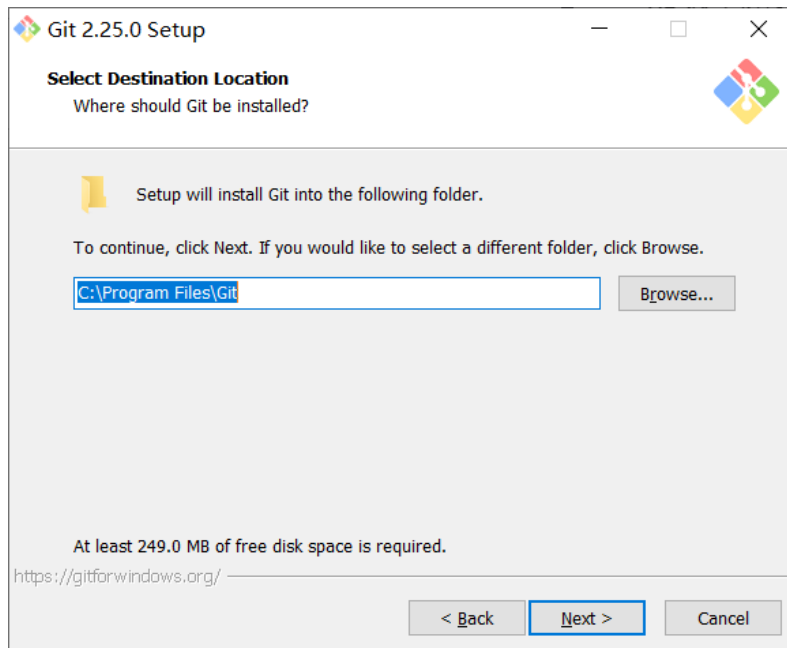
点击上图中红线框中处进行下载，得到 Git-2.25.0-64-bit.exe 文件，下载完成。

第二步：双击下载好的 Git 安装包，弹出提示框，如下图：



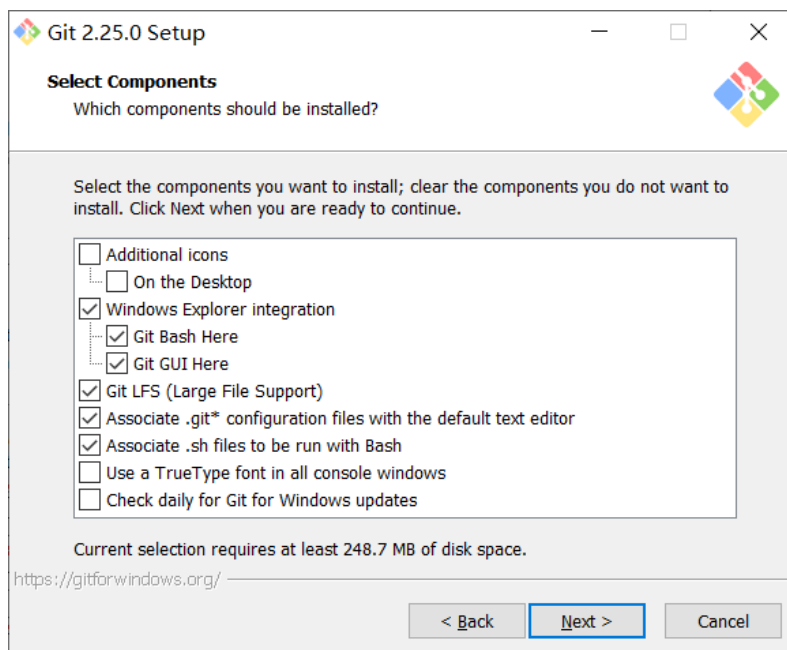
点击“Next”，进入下一步。

第三步：选择安装路径，如下图：



点击“Browse”可更改默认安装地址；点击“Next”，进入下一步。

第四步：弹出安装配置窗口，包括 git 命令行、git 图形窗口等，如下图所示：



这里选择默认配置，直接点击“Next”，进入下一步。

【选项解释：】

Additional icons 附加图标

On the Desktop 在桌面上

Windows Explorer integration Windows 资源管理器集成鼠标右键菜单

Git Bash Here

Git GUI Here

Git LFS (Large File Support) 大文件支持

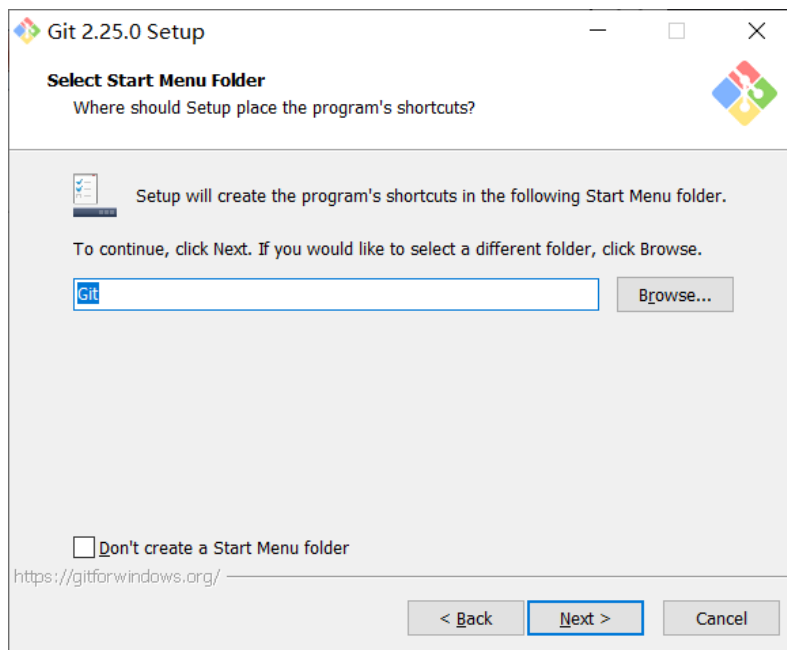
Associate .git configuration files with the default text editor 将 .git 配置文件与默认文本编辑器相关联*

Associate .sh files to be run with Bash 将 .sh 文件关联到 Bash 运行

Use a TrueType font in all console windows 在所有控制台窗口中使用 TrueType 字体

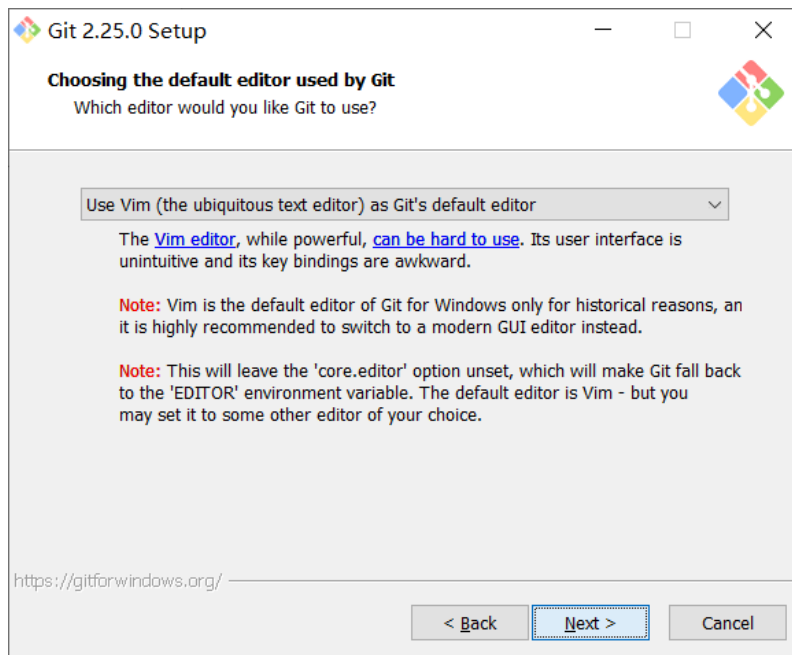
Check daily for Git for Windows updates 每天检查 Git 是否有 Windows 更新

第五步：弹出“选择开始菜单文件夹”的窗口，如下图所示：



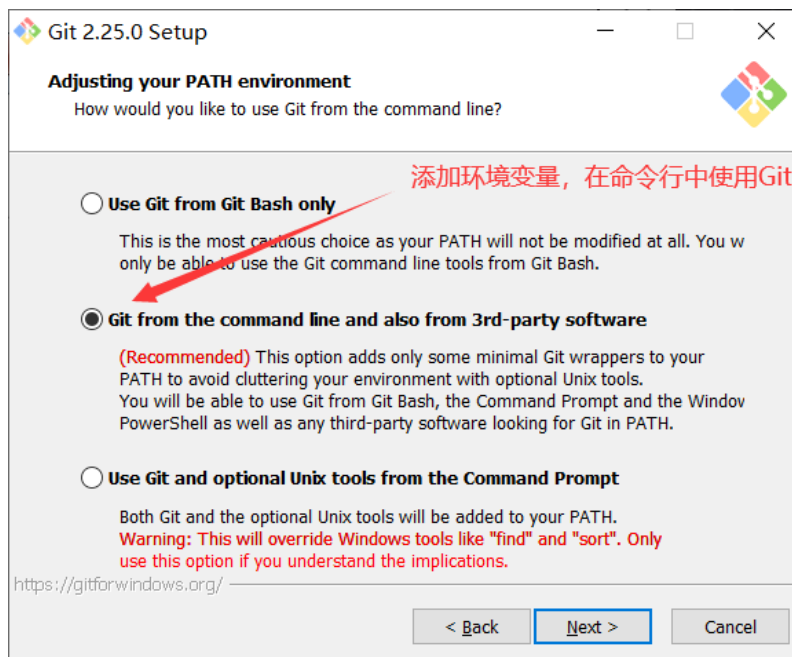
默认路径，直接点击"Next"，进入下一步。

第六步：进入“选择 Git 使用的默认编辑器”窗口，如下图所示：



使用 Vim 作为 Git 的默认编辑器，点击“Next”，进入下一步。

第七步：进入“调整 Path 环境变量”窗口，如下图所示：



选第二项，点击“Next”，进入下一步。

【选项解释：】

配置 PATH 环境

Use Git from Git Bash only

This is the safest choice as your PATH will not be modified at all. You will only be able to use the Git command line tools from Git Bash.

这是最安全的选择，因为您的 PATH 根本不会被修改。您只能使用 Git Bash 的 Git 命令行工具。

Use Git from the Windows Command Prompt

This option is considered safe as it only adds some minimal Git wrappers to your PATH to avoid cluttering your environment with optional Unix tools . You will be able to use Git from both Git Bash and the Windows Command Prompt.

这个选项被认为是安全的，因为它只向 PATH 添加一些最小的 Git 包，以避免使用可选的 Unix 工具混淆环境。您将能够从 Git Bash 和 Windows 命令提示符中使用 Git。

Use Git and optional Unix tools from the Windows Command Prompt

从 Windows 命令提示符使用 Git 和可选的 Unix 工具

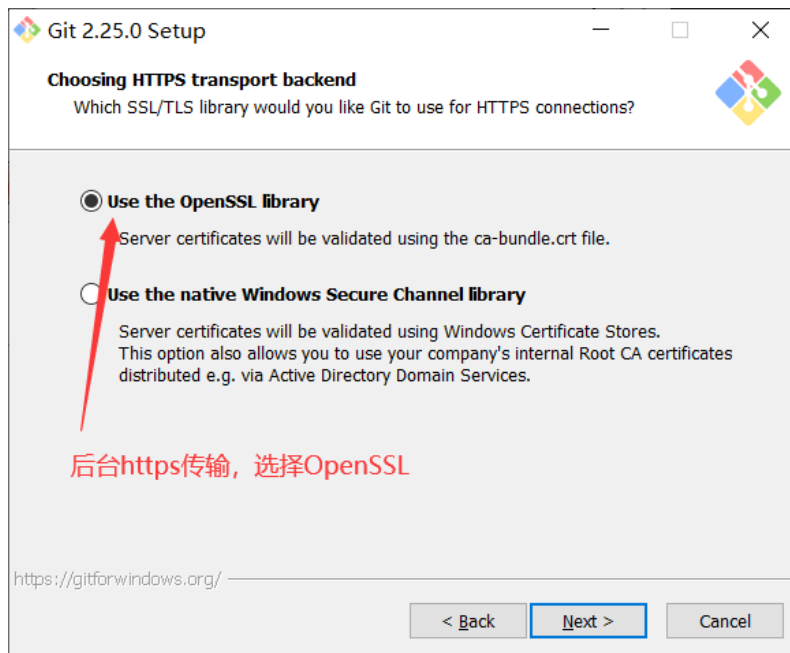
Both Git and the optional Unix tools will be added to you PATH

Git 和可选的 Unix 工具都将添加到您计算机的 PATH 中

Warning: This will override Windows tools like "find and sort". Only use this option if you understand the implications.

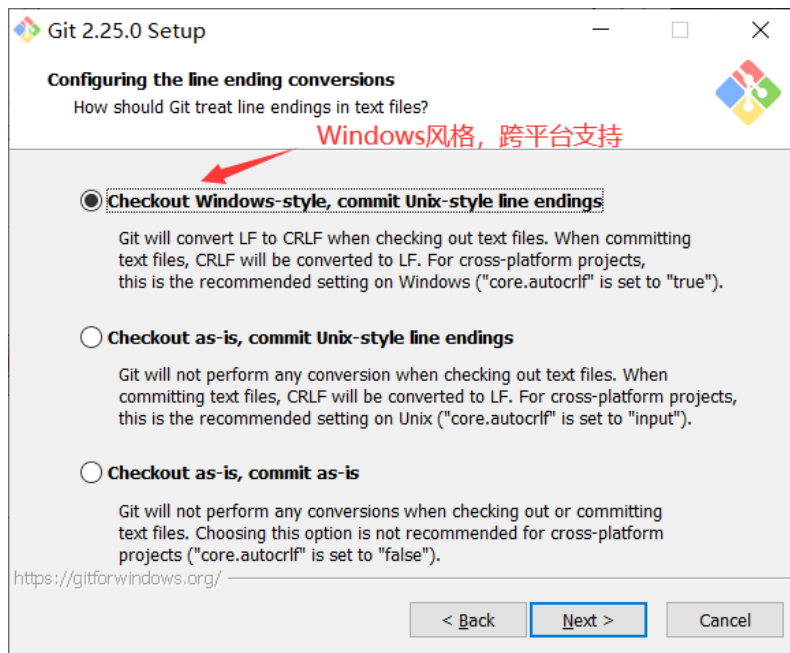
警告：这将覆盖 Windows 工具，如 “find 和 sort”。只有在了解其含义后才使用此选项。

第八步：选择 HTTPS 传输后端，如下图所示：



使用默认值，点击“Next”，进入下一步。

第九步：配置行结束符，如下图所示：



选择第一项，点击“Next”，进入下一步。

【选项解释：】

Checkout Windows-style,commit Unix-style line endings

Git will convert LF to CRLF when checking out text files.When committing text files,CRLF will be converted to LF .For cross-pltform projects,this is the recommended setting on Windows ("core.autocrlf" is set to "true")

在检出文本文件时，Git 会将 LF 转换为 CRLF。当提交文本文件时，CRLF 将转换为 LF。对于跨平台项目，这是 Windows 上推荐的设置（“core.autocrlf”设置为“true”）

Checkout as-is , commit Unix-style line endings

Git will not perform any conversion when checking out text files. When committing text files, CRLF will be converted to LF. For cross-platform projects,this is the recommended setting on Unix ("core.autocrlf" is set to "input")

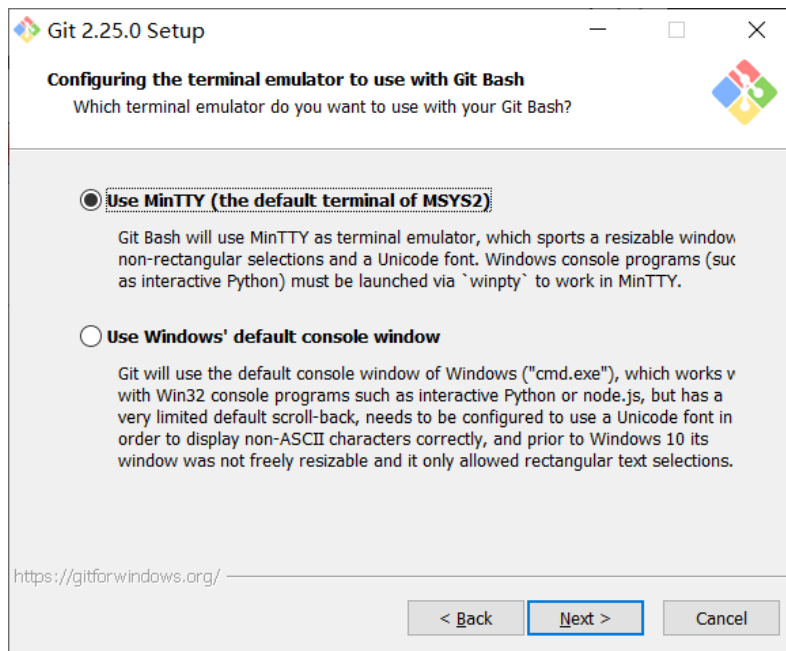
在检出文本文件时，Git 不会执行任何转换。提交文本文件时，CRLF 将转换为 LF。对于跨平台项目，这是 Unix 上的推荐设置（“core.autocrlf”设置为“input”）

Checkout as-is,commit as-is

Git will not perform any conversions when checking out or committing text files.Choosing this option is not recommended for cross-platform projects ("core.autocrlf"is set to "false")

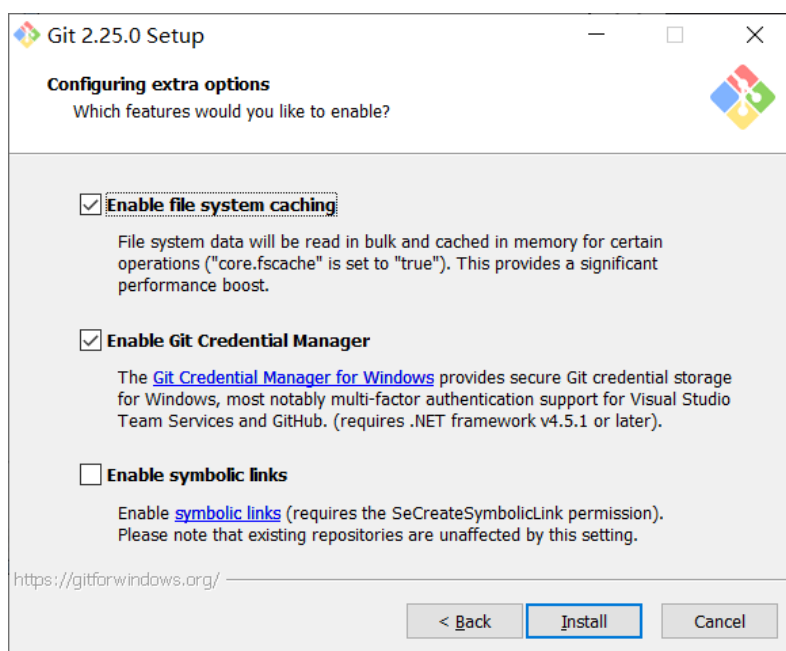
在检出或提交文本文件时，Git 不会执行任何转换。对于跨平台项目，不推荐使用此选项（“core.autocrlf”设置为“false”）

第十步：配置终端模拟器，如下图所示：

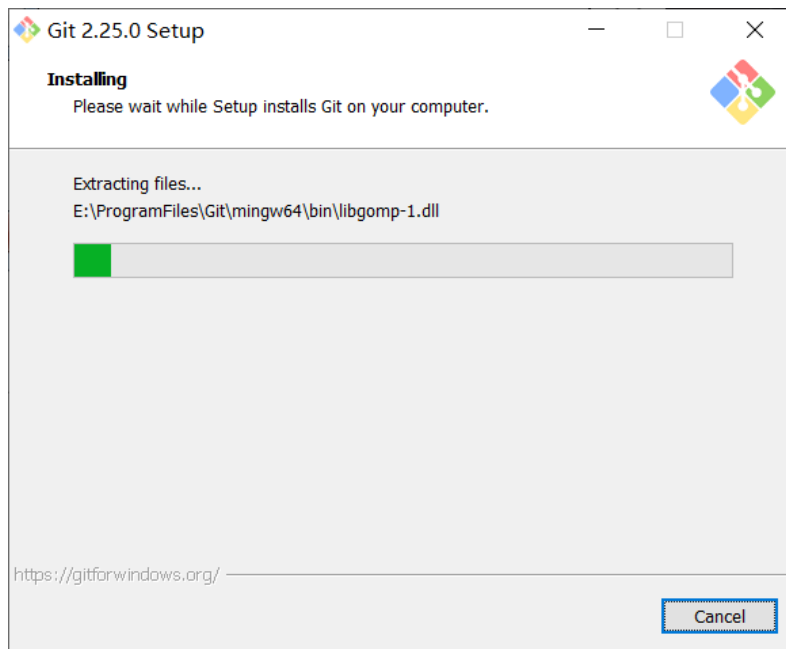


选第一项，点击“Next”，进入下一步。

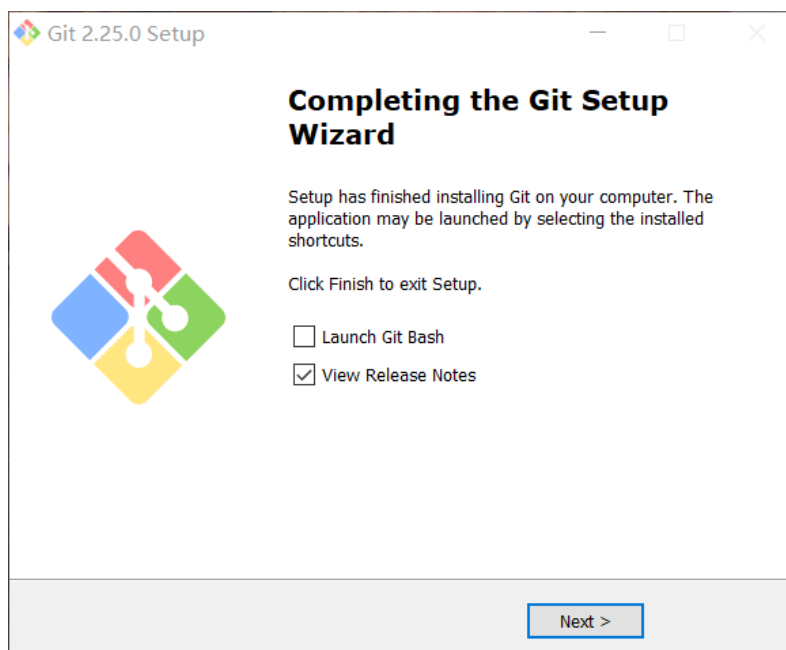
第十一步：配置额外选项，如下图所示：



使用默认配置，直接点击“Install”开始安装，如下图所示：



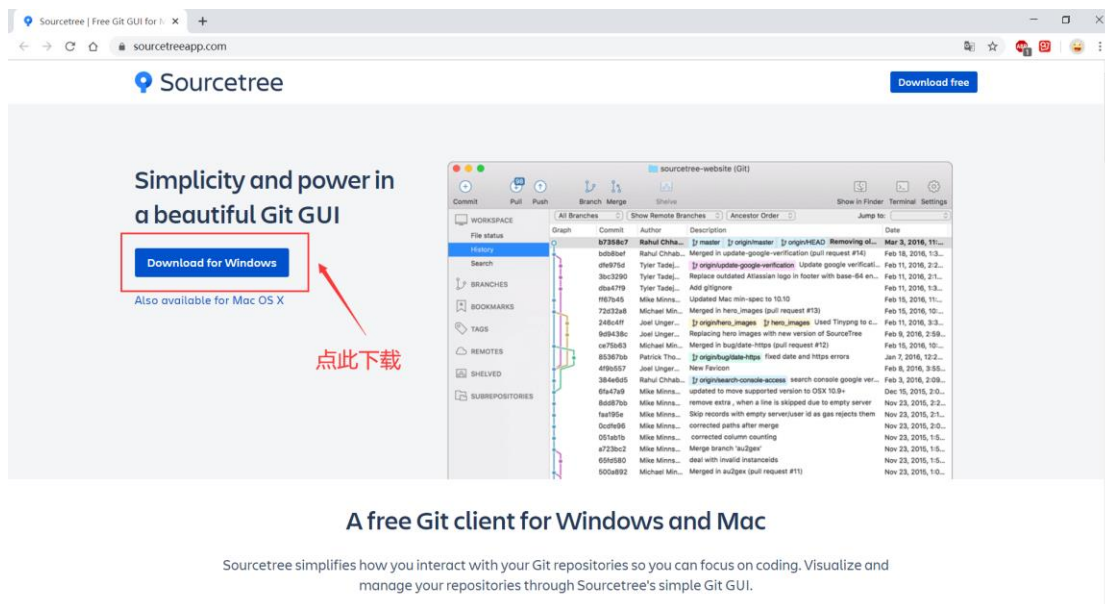
第十二步：安装完成。



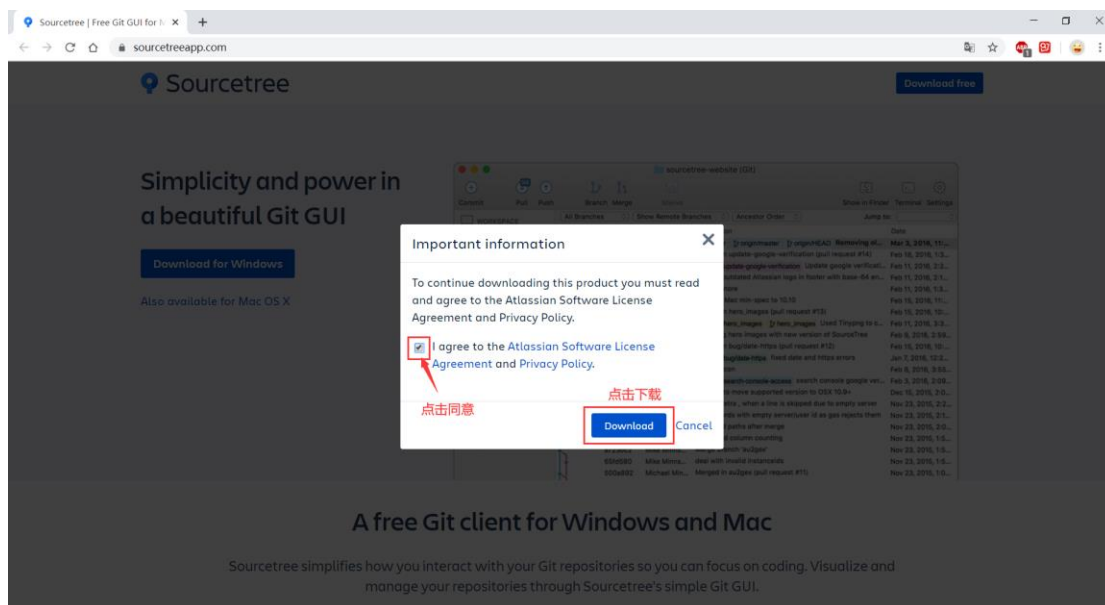
1.2 下载并安装 SourceTree

第一步：先从官网下载最新版本的 SourceTree

官网地址：<https://www.sourcetreeapp.com/>

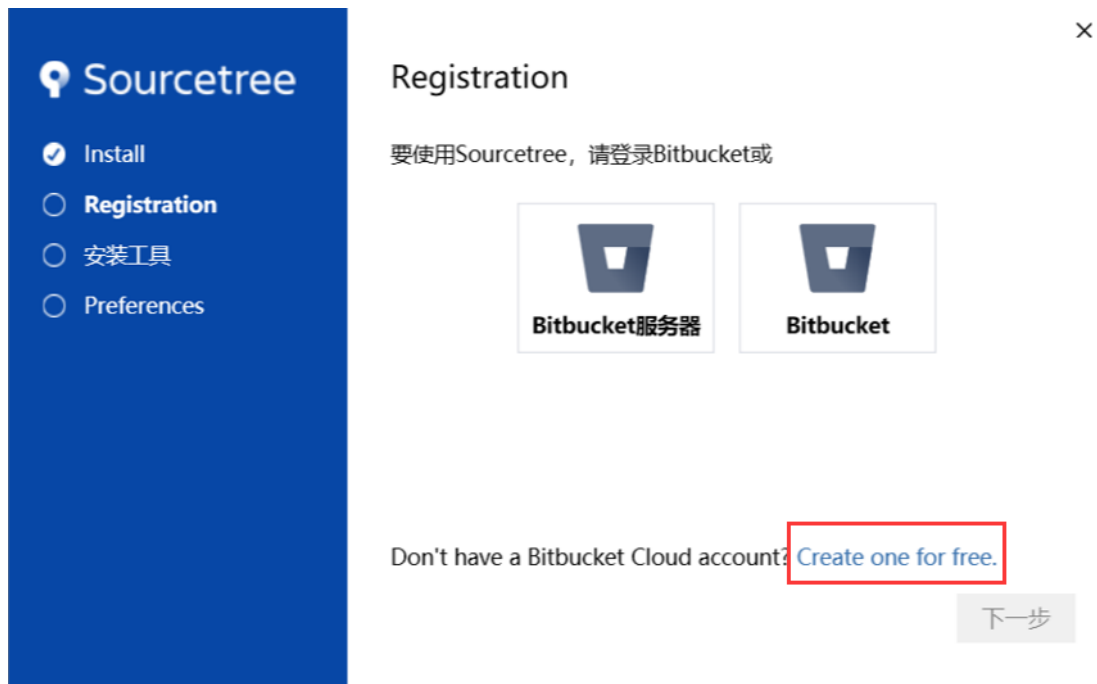


点击上图红线框中处下载。



点击红框处同意并开始下载，得到 SourceTreeSetup-3.3.8.exe 文件，下载完成。

第二步：双击下载好的 SourceTree 安装包，弹出提示框，如下图：



点击红线框中处，进入 <https://bitbucket.org/> 网站，如下图，免费注册一个 Bitbucket 云账号。



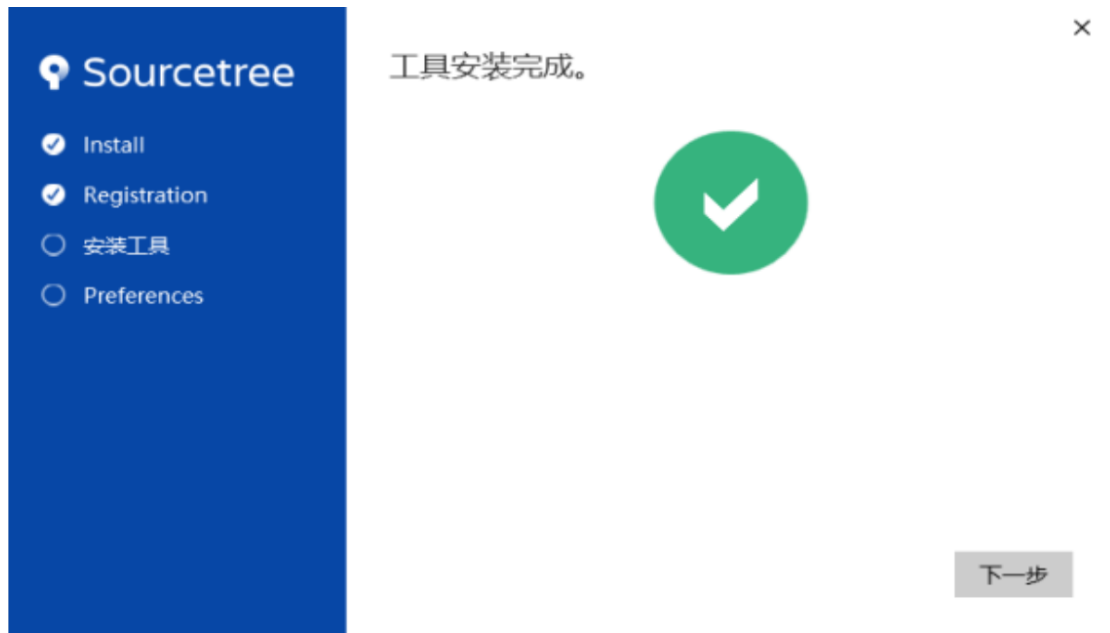
完成注册。

第三步：选择下载及安装所需工具。



勾选“Mercurial”以及“默认配置自动换行处理（推荐）”，点击“下一步”，开始下载版本控制系统。如下图：





工具安装完成。

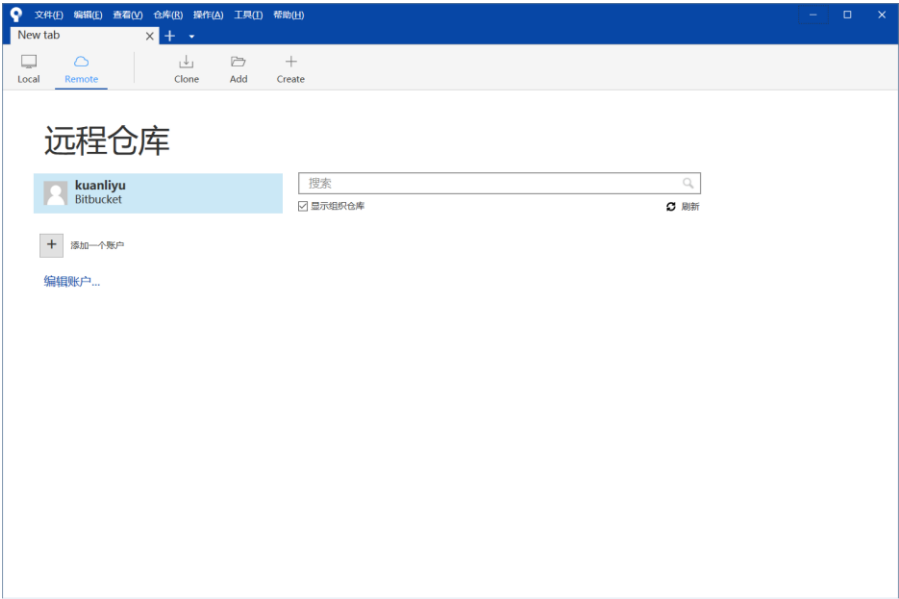
第四步：登录 Bitbucket 账户，点击“下一步”。



第五步：点击“否”。



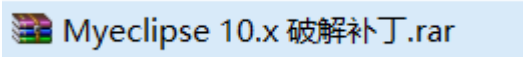
第六步：安装完成。



1.3 下载并安装 MyEclipse

第一步：先下载一个 Myeclipse10.x 破解补丁

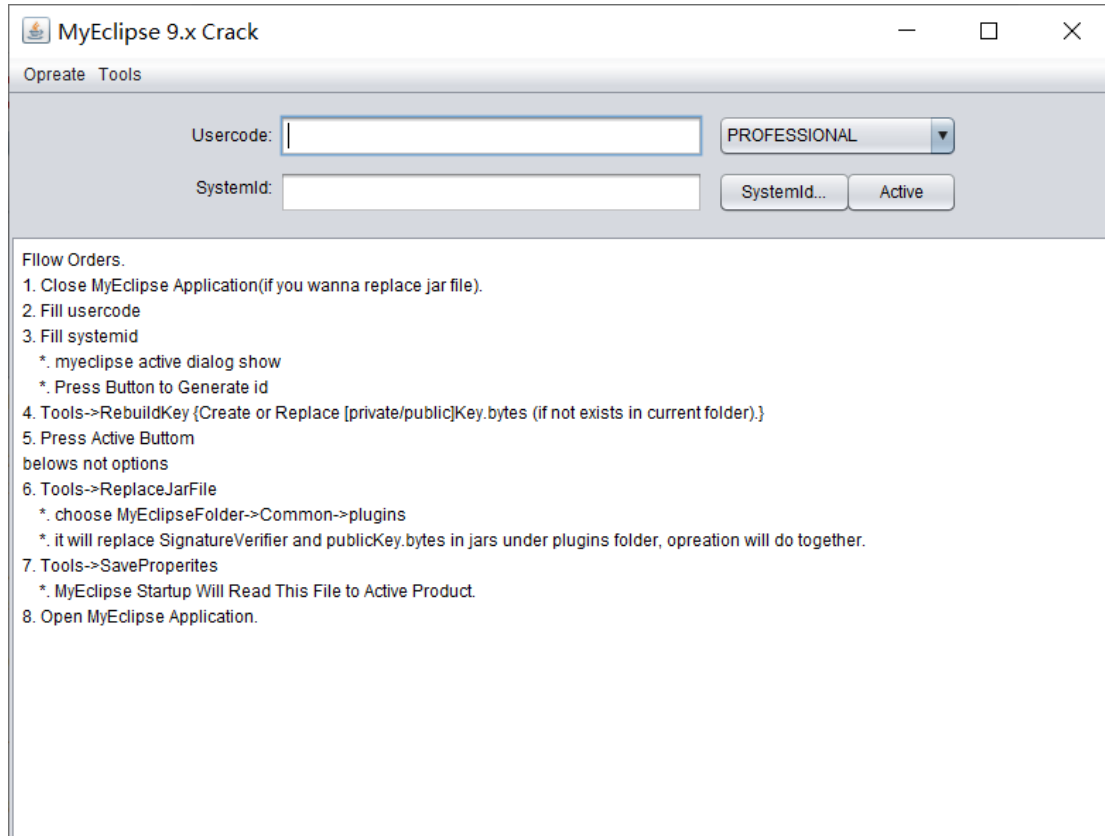
<https://pan.baidu.com/s/1ivE2yauZRDdDg8zBxpK06A>



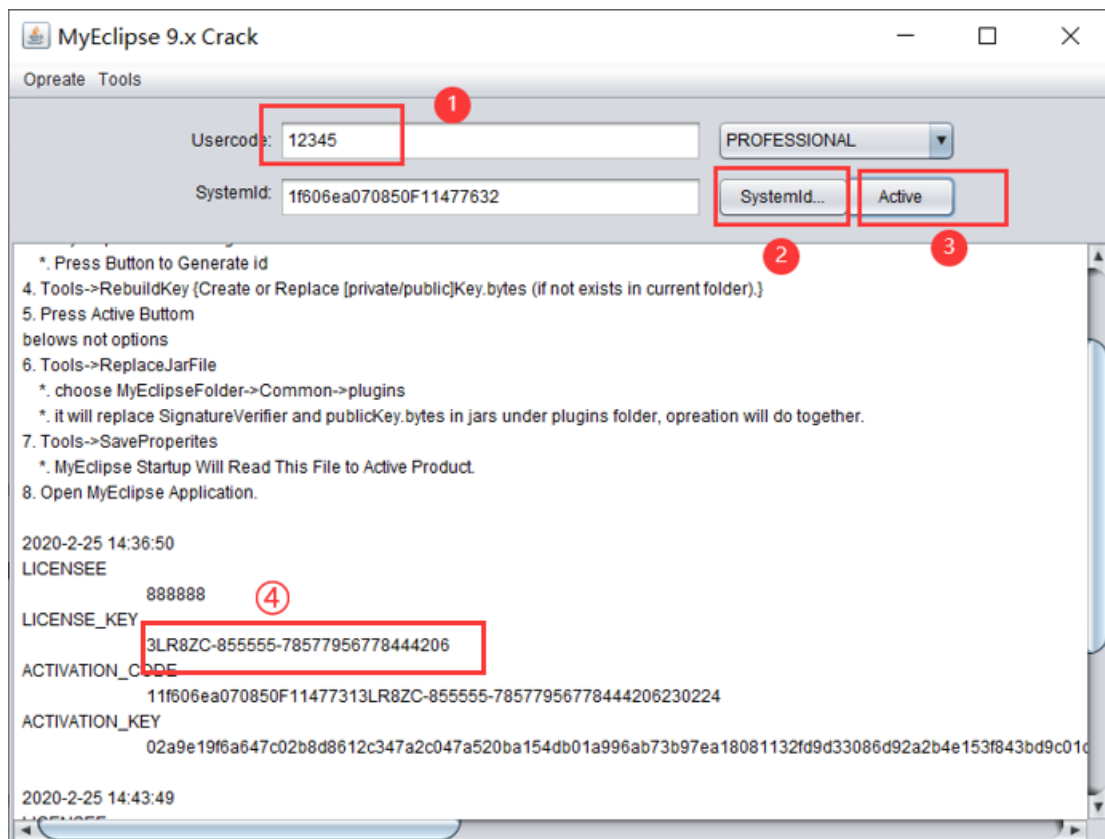
下载后解压，得以下文件：

.DS_Store	2020-02-25 12:23	DS_STORE 文件	7 KB
cracker.jar	2012-04-20 13:30	Executable Jar File	392 KB
jniwrap.dll	2011-05-05 17:43	应用程序扩展	49 KB
jniwrap.lic	2011-05-05 17:43	LIC 文件	1 KB
jniwrap64.dll	2011-05-05 17:43	应用程序扩展	34 KB
libjniwrap.jnilib	2011-05-05 17:43	JNILIB 文件	144 KB
libjniwrap.so	2011-05-05 17:43	SO 文件	27 KB
libjniwrap64.so	2011-05-05 17:43	SO 文件	48 KB
privateKey.bytes	2017-12-30 14:51	BYTES 文件	1 KB
publicKey.bytes	2017-12-30 14:51	BYTES 文件	1 KB
run.bat	2012-08-23 20:29	Windows 批处理...	1 KB
破解说明.txt	2020-02-25 12:24	文本文档	1 KB

运行 run.bat，会出现如下界面：

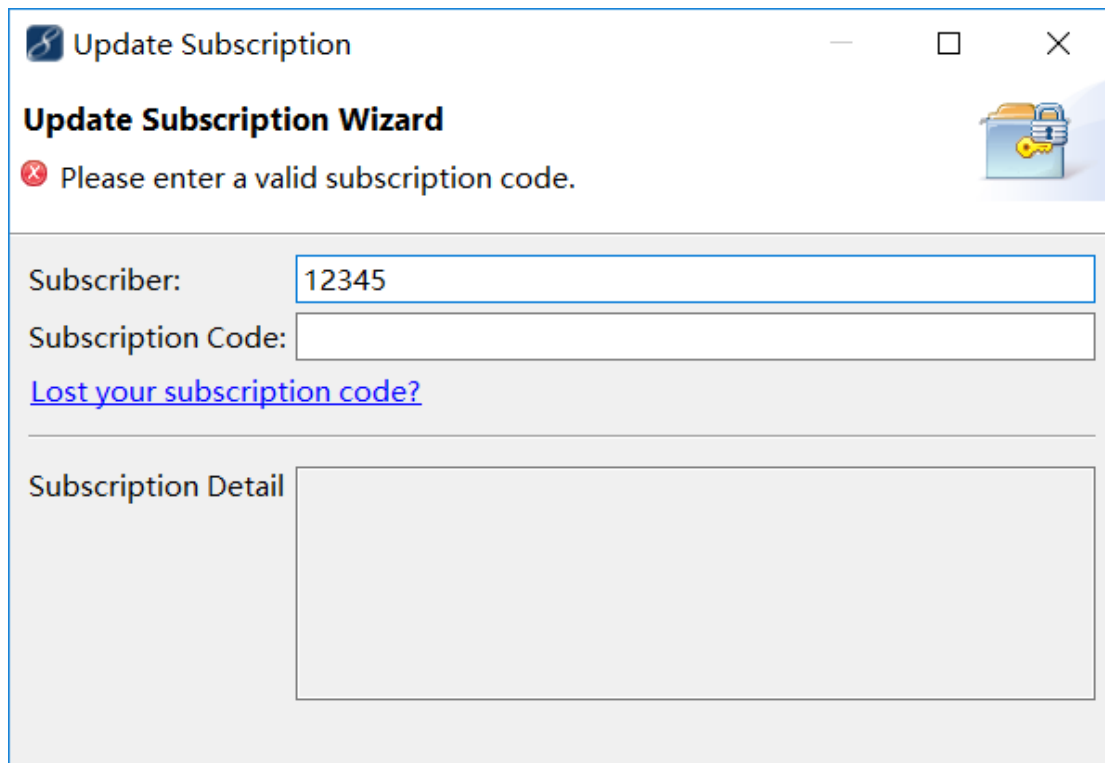


第二步：在 Usercode 处随便输入一串字符，然后点击 SystemId，再点击 Active，如果下面出现了 null，证明环境变量配置不对



复制 LICENSE_KEY 的内容，然后打开 MyEclipse。

第三步：点击菜单栏上的 MyEclipse，选择 Subscription Information，出现如下：



Update Subscription

Update Subscription Wizard

❌ Please enter a valid subscription code.

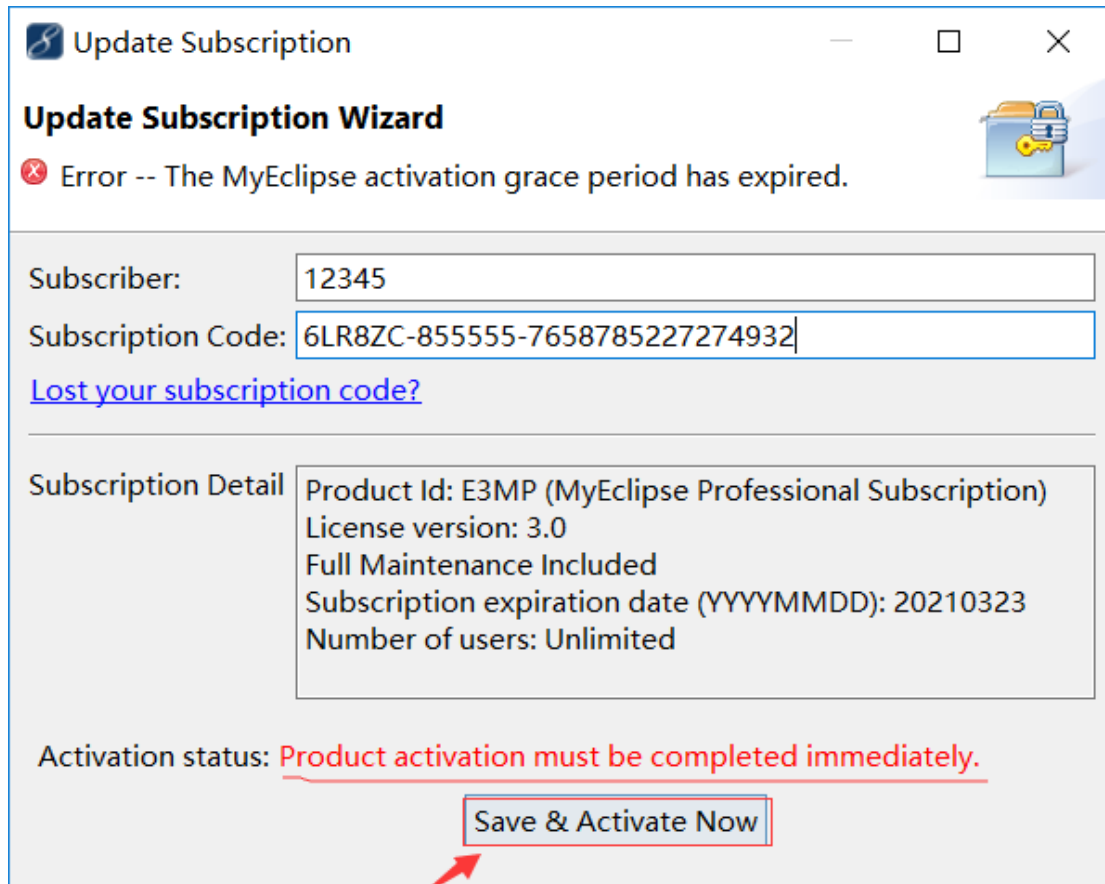
Subscriber: 12345

Subscription Code:

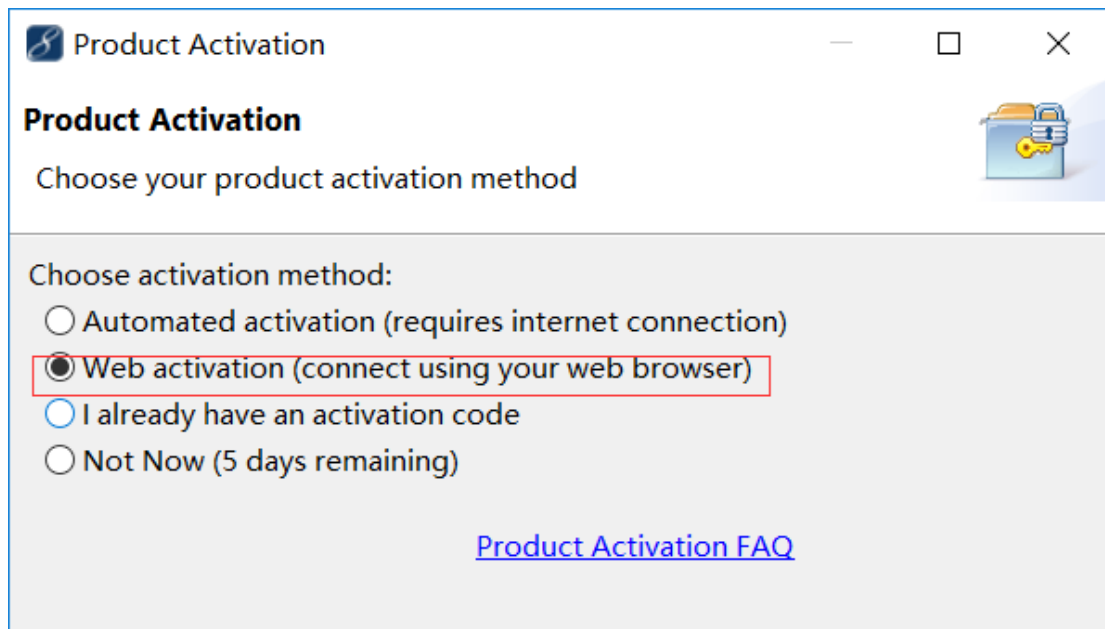
[Lost your subscription code?](#)

Subscription Detail

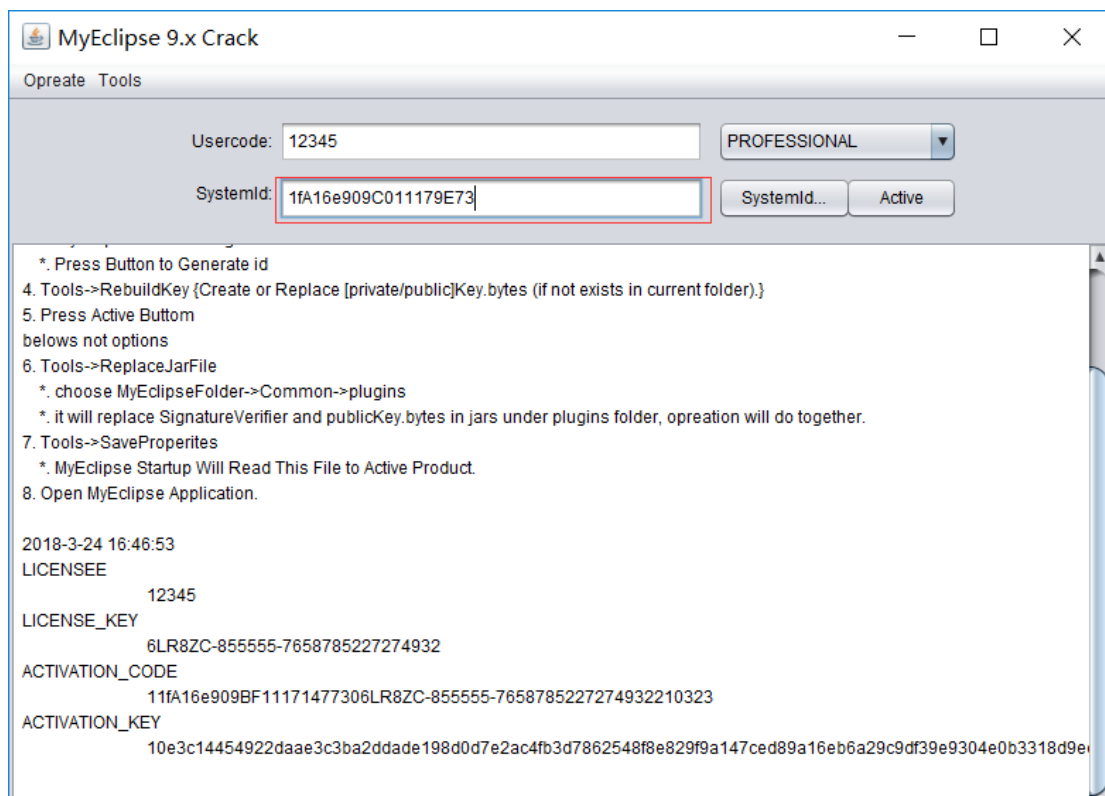
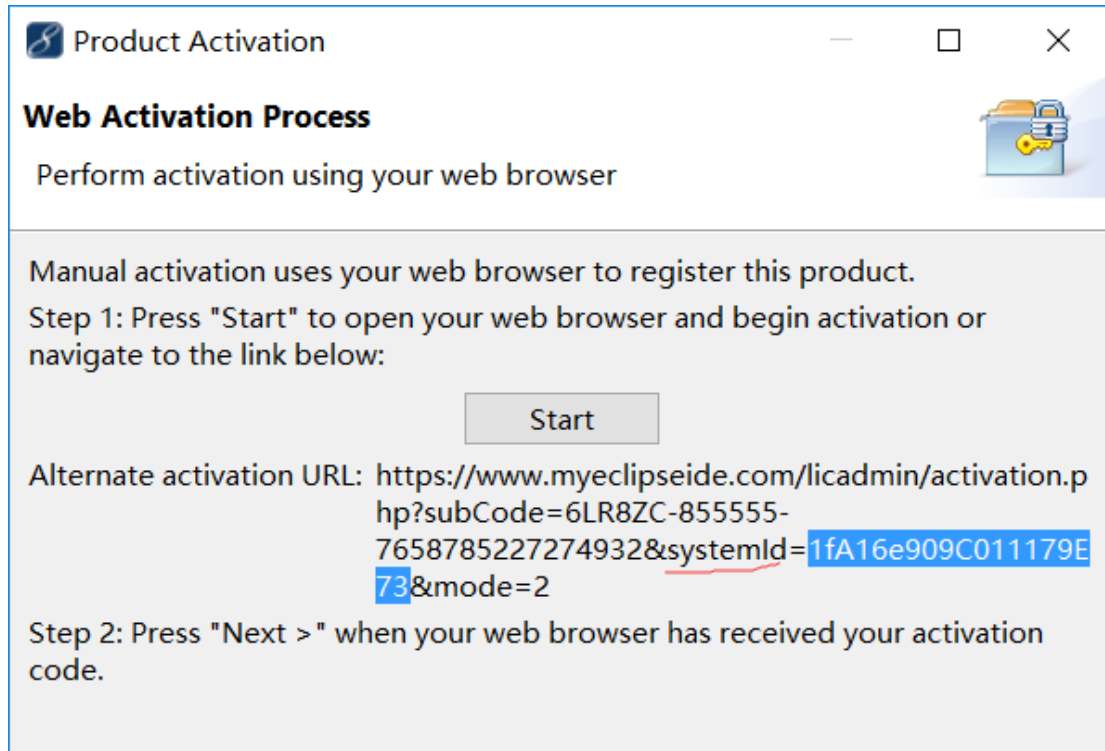
在 Code 那里粘贴刚刚复制的内容，然后点击 Save & Active Now



选择 Web activation, 点击“Next”

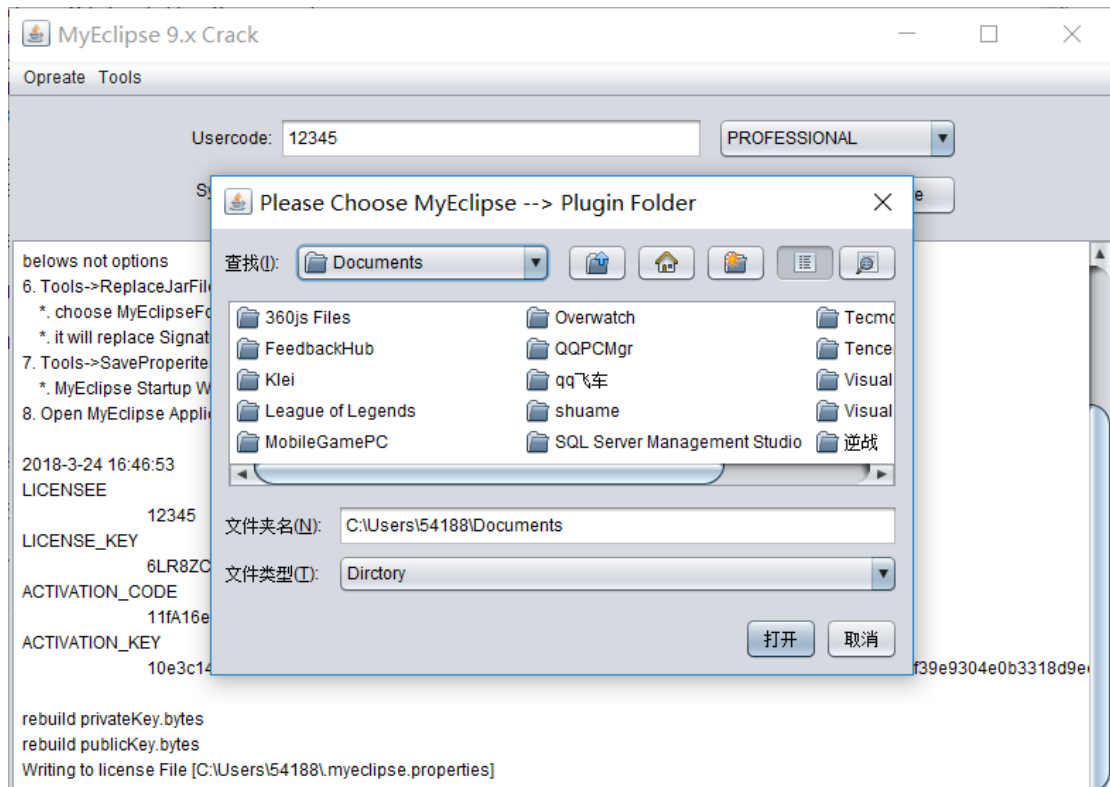


第四步：复制 systemId=后边的内容，即蓝色部分，然后粘贴到破解补丁界面的 systemId

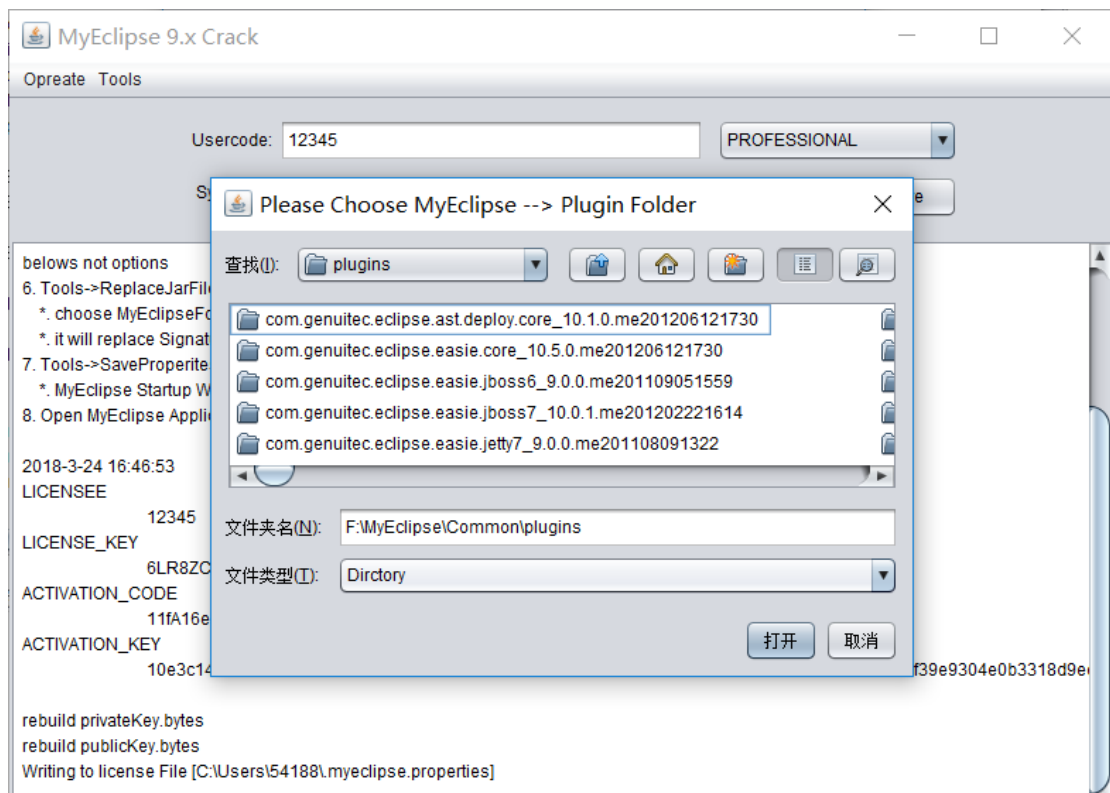


第五步：关闭 MyEclipse，来到破解界面

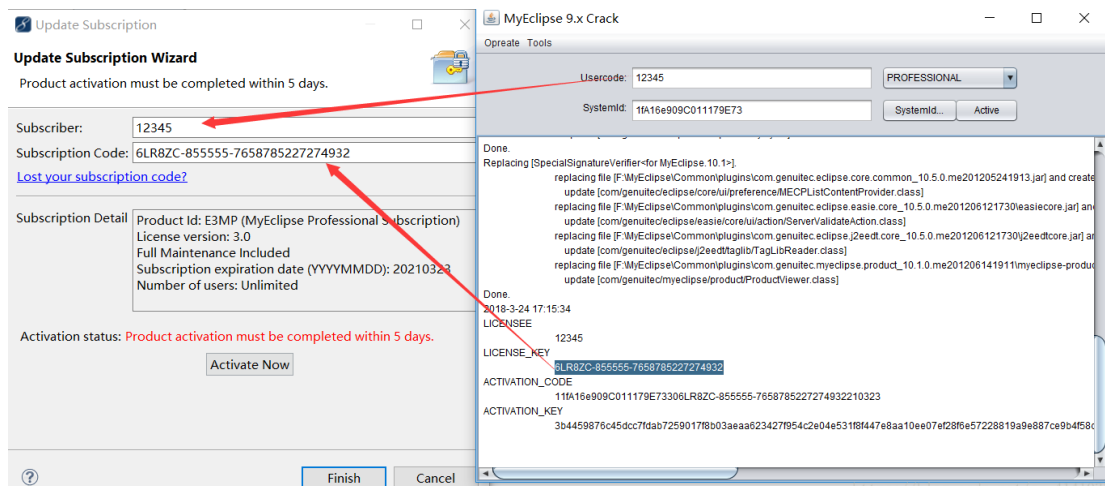
点击 Tools，依次点击 0 2 1，
点击 1 后会出现弹窗



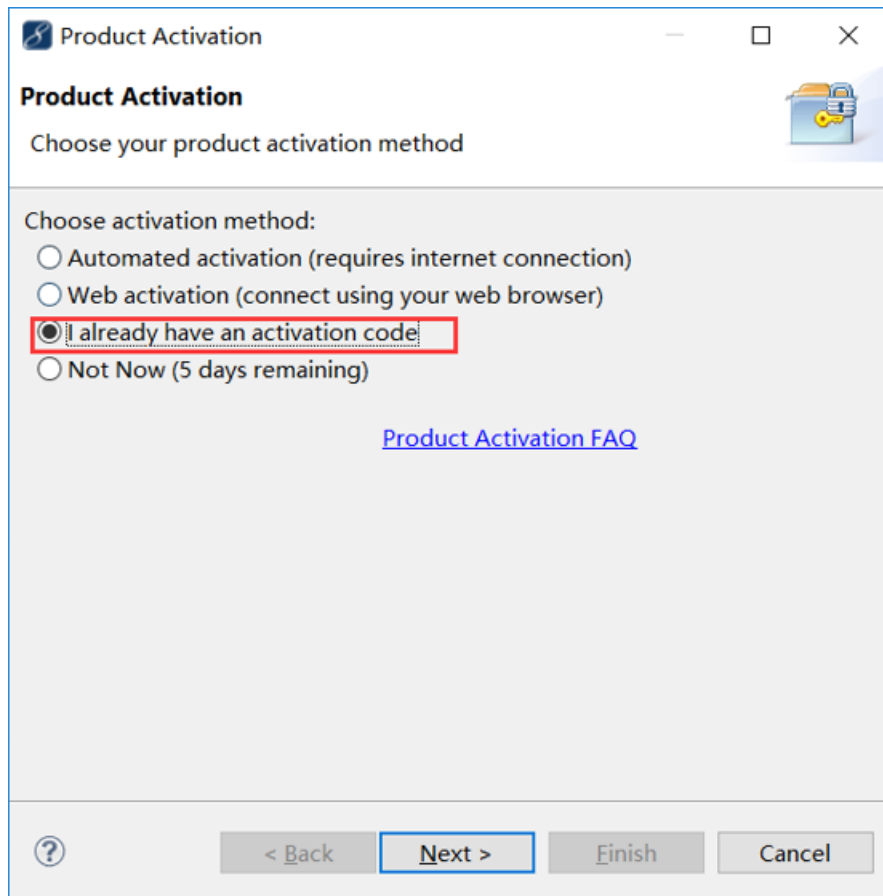
选择安装目录下的 Common/plugin



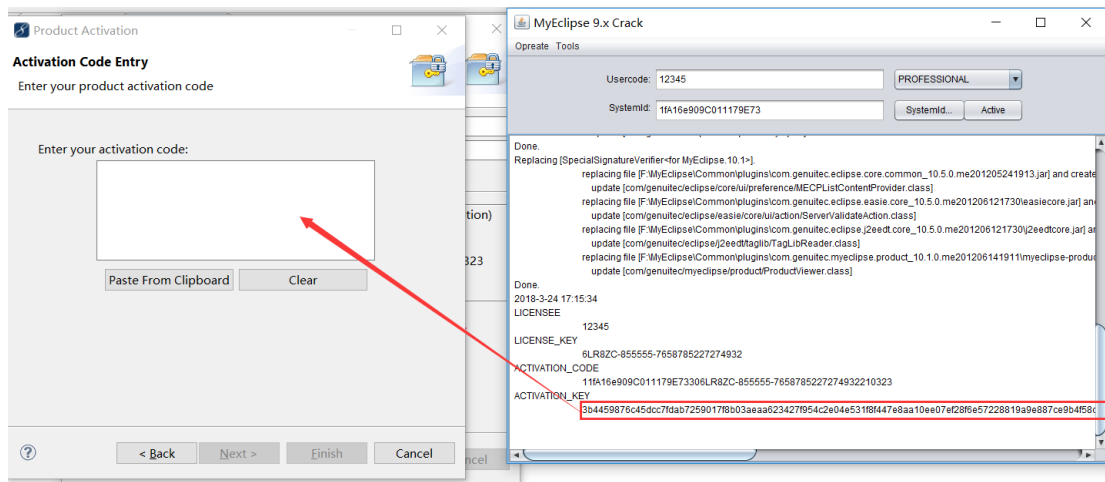
点击“打开”，就会关闭这个界面，这里需要等一会，当出现 Done 时，证明文件替换成功。



然后点击 Activate Now
弹出一个窗口，选择第三个

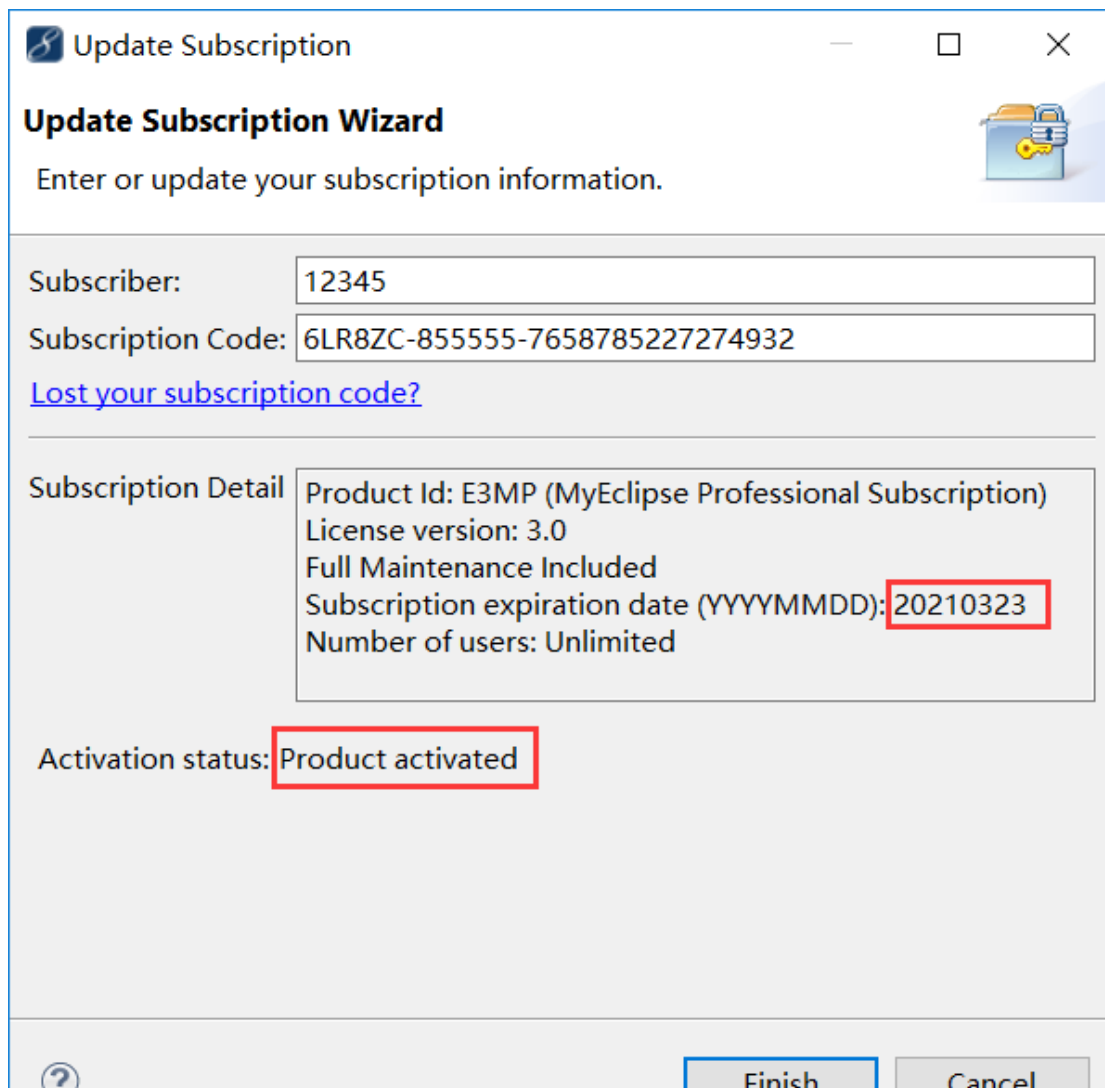


点击“Next”，把破解界面最后一行 ACTIVATION_KEY 后的内容复制到空白区域，如下图：



然后一直点击“Next”，最后点击“finish”。

第六步：点击菜单栏上的 MyEclipse，选择 Subscription Information



最后点击“finish”，完成安装。

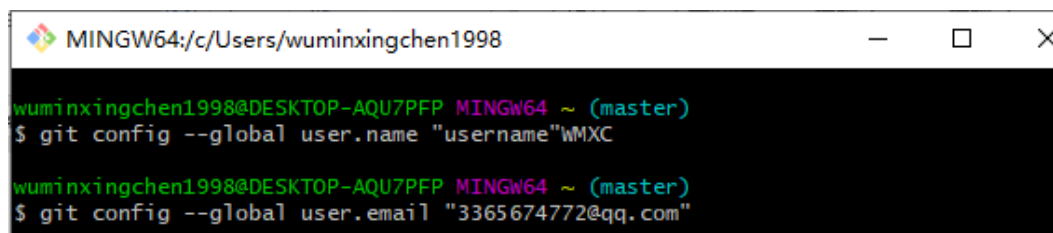
2 Git 使用与常用命令

2.1 配置用户名及邮箱

设置机器信息，配置用户名及邮箱。这台机器上的所有 Git 仓库都会使用这个配置
打开 Git Bash，弹出窗口，在其中输入以下命令：

```
git config --global user.name "username"//配置用户名
```

```
git config --global user.email "email@example.com"//配置用户邮箱
```

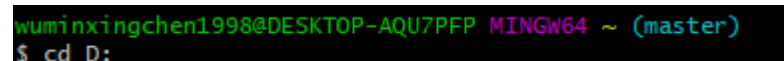


```
MINGW64:/c/Users/wuminxingchen1998
wuminxingchen1998@DESKTOP-AQU7PFP MINGW64 ~ (master)
$ git config --global user.name "username"WMXC
wuminxingchen1998@DESKTOP-AQU7PFP MINGW64 ~ (master)
$ git config --global user.email "3365674772@qq.com"
```

2.2 创建版本库

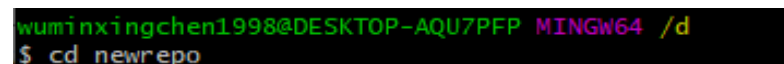
2.2.1 创建一个空目录

(1) 打开 D:盘



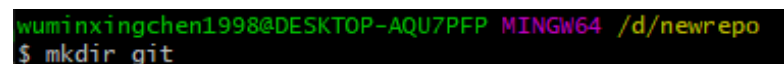
```
wuminxingchen1998@DESKTOP-AQU7PFP MINGW64 ~ (master)
$ cd D:
```

(2) 打开 D:盘目录下自己创建的新repo 文件夹



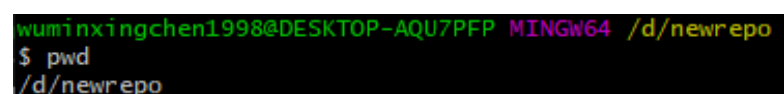
```
wuminxingchen1998@DESKTOP-AQU7PFP MINGW64 /d
$ cd newrepo
```

(3) 在 newrepo 文件夹创建 git 文件夹



```
wuminxingchen1998@DESKTOP-AQU7PFP MINGW64 /d/newrepo
$ mkdir git
```

(3) pwd 命令显示当前目录



```
wuminxingchen1998@DESKTOP-AQU7PFP MINGW64 /d/newrepo
$ pwd
/d/newrepo
```


2.2.2 初始化一个 Git 仓库

Git 使用 `git init` 命令来初始化一个 Git 仓库，Git 的很多命令都需要在 Git 的仓库中运行，在执行完成 `git init` 命令后，Git 仓库会生成一个 `.git` 目录，该目录包含了资源的所有元数据，其他的项目目录保持不变（不像 SVN 会在每个子目录生成 `.svn` 目录，Git 只在仓库的根目录生成 `.git` 目录）

(1) 使用当前目录作为 Git 仓库，需要使它初始化。

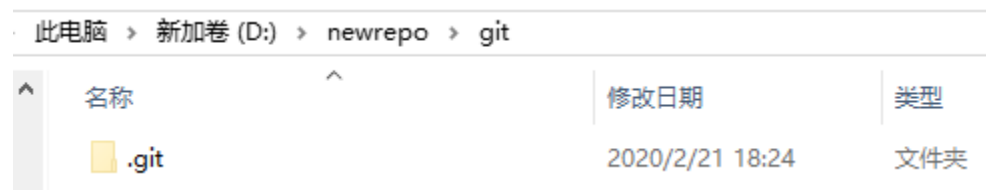
```
wuminxingchen1998@DESKTOP-AQU7PFP MINGW64 ~  
$ git init  
Initialized empty Git repository in C:/Users/wuminxingchen1998/.git/
```

(2) `git init` 命令使用我们指定目录 `git` 作为可以管理的仓库。

初始化了这空的仓库，目录下多了 `.git` 目录，这个目录是 Git 来跟踪管理版本的，没事千万不要手动乱改这个目录里面的文件，否则，会把 git 仓库给破坏了。

```
wuminxingchen1998@DESKTOP-AQU7PFP MINGW64 /d/newrepo/git  
$ git init  
Initialized empty Git repository in D:/newrepo/git/.git/
```

(3) 成功创建了 Git 仓库（Git 仓库为隐藏文件，需要打开显示隐藏文件）

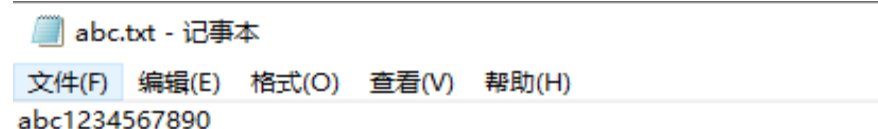


(4) 系统自动创建了唯一一个 `master` 分支（版本控制系统只能跟踪文本文件的改动，且编码方式是 utf-8）

```
wuminxingchen1998@DESKTOP-AQU7PFP MINGW64 /d/newrepo/git (master)  
$ |
```

2.3 文件的基本操作

创建一个 `abc.txt` 文件，内容为：`abc1234567890`



2.3.1 添加文件到仓库里面去

(1) 使用命令 `git add abc.txt` 添加到暂存区里面去。(如果没有任何提示, 说明已经添加成功了)

```
wuminxingchen1998@DESKTOP-AQU7PFP MINGW64 /d/newrepo/git (master)
$ git add abc.txt
```

(2) 用命令 `git commit` 告诉 Git, 把文件提交到仓库。(-m 后面输入的是本次提交的说明)

```
wuminxingchen1998@DESKTOP-AQU7PFP MINGW64 /d/newrepo/git (master)
$ git commit -m "a new file"
[master (root-commit) e2eb77b] a new file
1 file changed, 1 insertion(+)
create mode 100644 abc.txt
```

提交成功后会显示:

1 file changed: 1 个文件被改动 (我们新添加的 abc.txt 文件);
2 insertions: 插入了一行内容 (abc.txt 有一行内容)。

备注:

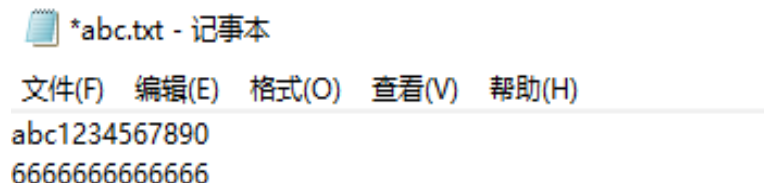
关于 *commit* 的意义: 一开始我并不明白为什么 Git 添加文件需要 *add*, *commit* 一共两步到后期才反应过来 *commit* 可以一次提交很多文件, 所以你可以多次 *add* 不同的文件, 最后一起提交。

2.3.2 查看工作文件状态及历史

(1) 用命令 `git status` 来查看是否还有文件未提交 (没有任何文件未提交)

```
wuminxingchen1998@DESKTOP-AQU7PFP MINGW64 /d/newrepo/git (master)
$ git status
On branch master
nothing to commit, working tree clean
```

(2) 修改 abc.txt 文件如下:



(3) 修改后用 `git status` 来查看下结果。

(这告诉我们 abc.txt 文件已被修改, 但是未被提交的修改)

```
wuminxingchen1998@DESKTOP-AQU7PFP MINGW64 /d/newrepo/git (master)
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   abc.txt

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

(4) 用命令 git diff 查看 abc.txt 文件到底改了什么内容

```
wuminxingchen1998@DESKTOP-AQU7PFP MINGW64 /d/newrepo/git (master)
$ git diff abc.txt
diff --git a/abc.txt b/abc.txt
index 11bc0d7..3068c44 100644
--- a/abc.txt
+++ b/abc.txt
@@ -1,2 +1,3 @@
   abc1234567890
-66666666666666
\ No newline at end of file
+66666666666666
+00000000000000
```

备注：

笔者因提前将文件提交到了库中，导致了无法查看文件修改内容。所以重新修改过了一遍文件，添加了一行 000000000000，特此说明。

(5) 提交修改后件（每次 commit 都会生成一个“快照”）

知道了对 abc.txt 文件做了什么修改后，我们可以放心的提交到仓库了，提交修改和提交文件是一样的 2 步

第一步是 git add；第二步是：git commit

```
wuminxingchen1998@DESKTOP-AQU7PFP MINGW64 /d/newrepo/git (master)
$ git add abc.txt

wuminxingchen1998@DESKTOP-AQU7PFP MINGW64 /d/newrepo/git (master)
$ git commit -m "append 666"
[master 9ba4321] append 666
1 file changed, 2 insertions(+), 1 deletion(-)
```

(6) 查看历史记录

git log 显示最近到最远的提交日志，我们可以看到两次提交，最后一次是 append 666

```
wuminxingchen1998@DESKTOP-AQU7PFP MINGW64 /d/newrepo/git (master)
$ git log
commit 9ba4321fd0ae7e638c1a6ce427ef1103b877ae2b (HEAD -> master)
Author: usernameWXC <3365674772@qq.com>
Date: Fri Feb 21 23:20:39 2020 +0800

    append 666


commit e2eb77b701d80eebf0ee0820b16dd98cc39857fa
Author: usernameWXC <3365674772@qq.com>
Date: Fri Feb 21 22:48:49 2020 +0800

    a new file
```

2.3.3 回退历史版本

当我完成了对文件的修改，我想使用版本回退操作，把当前的版本回退到上一个版本，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。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 abc.txt 文本内容：

 abc.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)


```
abc1234567890
66666666666666
00000000000000
```

(1) 用命令 git reset --hard HEAD^回退上一版本

```
wuminxingchen1998@DESKTOP-AQU7PFP MINGW64 /d/newrepo/git (master)
$ git reset --hard HEAD^
HEAD is now at e2eb77b a new file
```

(2) 通过命令 cat abc.txt 查看 abc.txt 文本内容

```
wuminxingchen1998@DESKTOP-AQU7PFP MINGW64 /d/newrepo/git (master)
$ cat abc.txt
abc1234567890
```

 abc.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

```
abc1234567890
```

备注：

笔者因未将后添加的一行 000000000000 文件提交到了仓库中，导致了会退后变成了最开始的版本，特此说明。

可以看到，内容已经回退到上一个版本了。

(3) 我们可以继续使用 `git log` 来查看下历史记录信息

```
wuminxingchen1998@DESKTOP-AQU7PFP MINGW64 /d/newrepo/git (master)
$ git log
commit e2eb77b701d80eebf0ee0820b16dd98cc39857fa (HEAD -> master)
Author: usernameWXC <3365674772@qq.com>
Date:   Fri Feb 21 22:48:49 2020 +0800

    a new file
```

(4) 是不记得刚才的版本号了，可以使用以下命令：

`$ git reflog`

```
wuminxingchen1998@DESKTOP-AQU7PFP MINGW64 /d/newrepo/git (master)
$ git reflog
e2eb77b (HEAD -> master) HEAD@{0}: reset: moving to HEAD^
9ba4321 HEAD@{1}: commit: append 666
e2eb77b (HEAD -> master) HEAD@{2}: commit (initial): a new file
```

2.4 Git 撤销修改和删除文件操作

2.4.1 关于 Git 中工作区与暂存区的思考

(1) 工作区：就是你在电脑上看到的目录，比如目录下的文件(.git 隐藏目录版本库除外)。或者以后需要再新建的目录文件等等都属于工作区范畴。

(2) 版本库(Repository)：工作区有一个隐藏目录.git,这个不属于工作区，这是版本库。其中版本库里面存了很多东西，其中最重要的就是 stage(暂存区)，还有 Git 为我们自动创建了第一个分支 master,以及指向 master 的一个指针 HEAD。

(3) Git 提交文件到版本库有两步：

第一步：是使用 `git add` 把文件添加进去，实际上就是把文件添加到暂存区。

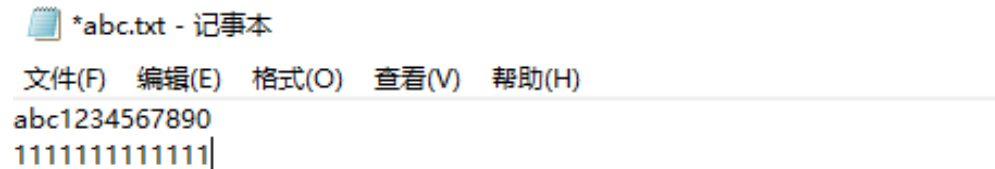
第二步：使用 `git commit` 提交更改，实际上就是把暂存区的所有内容提交到当前分支上。

(4) 笔者之前的操作中，修改 txt 文本后，用 `git add` 命令把文件都添加到暂存区。之后便再一次修改了文件，之后一起提交到仓库里。结果只有第一次修改的内容，第二次的修改未提交。在查阅资料分析后，发现笔者的行为为：第一次修改-->`git add`-->第二次修改-->`git commit`。`add` 将工作区的修改存入暂存区，但是第二次修改并未存入暂存区，`git commit` 只负责把暂存区的修改提交，所以正确的顺序应该是：第一次修改 --> `git add` --> 第二次修改 --> `git add` --> `git commit`。通过 `git commit` 可以把之前的多次 `add` 文件一起进行提交。

2.4.2 具体的修改操作

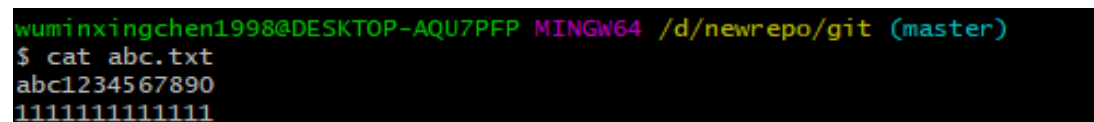
2.4.2.1 撤销工作区修改

现在在 abc.txt 文件里面增加一行,内容为



```
*abc.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
abc1234567890
1111111111111|
```

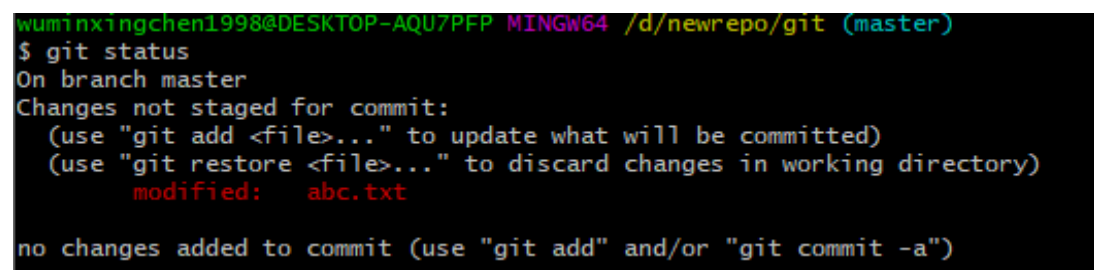
(1) 通过命令 `cat abc.txt` 查看 abc.txt 文本内容如下:



```
wuminxingchen1998@DESKTOP-AQU7PFP MINGW64 /d/newrepo/git (master)
$ cat abc.txt
abc1234567890
1111111111111
```

如果我们要撤销修改,可以通过直接修改文件,或回退版本。但文件内容过多的话会很麻烦。
如果想简单一点操作,可以通过 `git checkout -- file` 命令丢弃工作区的修改。

(2) 首先用 `git status` 查看下当前的状态



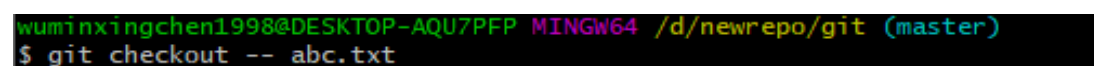
```
wuminxingchen1998@DESKTOP-AQU7PFP MINGW64 /d/newrepo/git (master)
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   abc.txt

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

(3) 可以发现, Git 会告诉你, `git checkout -- file` 可以丢弃工作区的修改,
(--很重要, 没有--, 就变成了“切换到另一个分支”的命令)

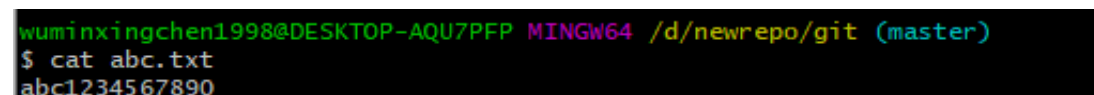
如下命令:

`git checkout -- abc.txt`



```
wuminxingchen1998@DESKTOP-AQU7PFP MINGW64 /d/newrepo/git (master)
$ git checkout -- abc.txt
```

(4) 通过命令 `cat abc.txt` 查看 abc.txt 文本内容



```
wuminxingchen1998@DESKTOP-AQU7PFP MINGW64 /d/newrepo/git (master)
$ cat abc.txt
abc1234567890
```

2.4.2.2 撤销暂存区修改（重新放回工作区）

```
wuminxingchen1998@DESKTOP-AQU7PFP MINGW64 /d/newrepo/git (master)
$ git reset HEAD abc.txt
Unstaged changes after reset:
M   abc.txt
```

2.4.2.3 删除文件

(1) 我们在 git 目录添加一个文件 a.txt

此电脑 > 新加卷 (D:) > newrepo > git

名称	修改日期	类型	大小
.git	2020/2/22 21:36	文件夹	
a.txt	2020/2/22 21:45	文本文档	0 KB
abc.txt	2020/2/22 21:36	文本文档	1 KB

(2) 然后把 a.txt 提交到仓库里

```
wuminxingchen1998@DESKTOP-AQU7PFP MINGW64 /d/newrepo/git (master)
$ git add a.txt
```

```
wuminxingchen1998@DESKTOP-AQU7PFP MINGW64 /d/newrepo/git (master)
$ git commit -m "a new a.txt"
[master 4d1d9f4] a new a.txt
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 a.txt
```

(3) 通过 rm a.txt 命令在工作区中删除 a.txt 文件

```
wuminxingchen1998@DESKTOP-AQU7PFP MINGW64 /d/newrepo/git (master)
$ rm a.txt
```

(4) 用 git status 查看下当前的状态

```
wuminxingchen1998@DESKTOP-AQU7PFP MINGW64 /d/newrepo/git (master)
$ git status
On branch master
Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        deleted:    a.txt
        modified:   abc.txt

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

(5) 从版本库中删除该文件，那就用命令 git rm 命令删掉，并且 git commit 命令提交：

```
wuminxingchen1998@DESKTOP-AQU7PFP MINGW64 /d/newrepo/git (master)
$ git rm a.txt
rm 'a.txt'

wuminxingchen1998@DESKTOP-AQU7PFP MINGW64 /d/newrepo/git (master)
$ git commit -m "remove abc.txt"
[master d59fdcc] remove abc.txt
1 file changed, 0 insertions(+), 0 deletions(-)
delete mode 100644 a.txt
```

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 a.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。但我想在版本库中恢复此文件，就要在没有 `commit` 之前，

(6) 使用如下命令 `git checkout -- a.txt`

```
wuminxingchen1998@DESKTOP-AQU7PFP MINGW64 /d/newrepo/git (master)
$ rm a.txt

wuminxingchen1998@DESKTOP-AQU7PFP MINGW64 /d/newrepo/git (master)
$ git checkout -- a.txt
```

(7) 目录恢复了 `a.txt` 文件（用 `git rm` 库中删除了文件就无法恢复了）

此电脑 > 新加卷 (D:) > newrepo > git

名称	修改日期	类型	大小
.git	2020/2/22 22:21	文件夹	
a.txt	2020/2/22 22:21	文本文档	0 KB
abc.txt	2020/2/22 21:36	文本文档	1 KB

3 标签管理

像其他版本控制系统（VCS）一样，Git 可以给历史中的某一个提交打上标签，以示重要。比较有代表性的是人们会使用这个功能来标记发布结点（v1.0 等等）。

3.1 标签类型

Git 使用两种主要类型的标签：轻量标签（lightweight）与附注标签（annotated）。一个轻量标签很像一个不会改变的分支——它只是一个特定提交的引用。然而，附注标签是存储在 Git 数据库中的一个完整对象。它们是可以被校验的；其中包含打标签者的名字、电子邮件地址、日期时间；还有一个标签信息；并且可以使用 GNU Privacy Guard (GPG) 签名与验证。通常建议创建附注标签，这样你可以拥有以上所有信息；但是如果你只是想用一个临时的标签，或者因为某些原因不想要保存那些信息，轻量标签也是可用的。

3.2 创建标签

3.2.1 创建轻量标签

命令: `git tag tagName`

```
ZLF@DESKTOP-ZLFGYJL MINGW64 /k/库/ZLF/Desktop/GitTest (master)
$ git tag v1.0_lightTag
```

注意，标签名中不能含有“?”、“*”，因为这两个是通配符。

```
ZLF@DESKTOP-ZLFGYJL MINGW64 /k/库/ZLF/Desktop/GitTest (master)
$ git tag v1.0_?
fatal: 'v1.0_?' is not a valid tag name.

ZLF@DESKTOP-ZLFGYJL MINGW64 /k/库/ZLF/Desktop/GitTest (master)
$ git tag v1.0_*
fatal: 'v1.0_*' is not a valid tag name.
```

3.2.2 创建附注标签

命令: `git tag -a tagName -m "This is an annotation message"`

与创建轻量标签相比，这条命令多了两个选项：

- ①-a，表示这是一条附注标签；
- ②-m (--message)，指定了一条将会存储在标签中的信息，即附注。如果创建附注标签时（带有-a选项）没有此选项，Git会运行编辑器要求你输入附注。

```
ZLF@DESKTOP-ZLFGYJL MINGW64 /k/库/ZLF/Desktop/GitTest (master)
$ git tag -a v2.0_weightTag -m "相比v1.0，新增了一个文件"
```

3.3 列出标签

命令: `git tag [-l | --list]`

这个命令以字母顺序列出标签，但是它们出现的顺序并不重要。可以增加选项--sort指定列出顺序。

```

ZLF@DESKTOP-ZLFGYJL MINGW64 /k/库/ZLF/Desktop/GitTest (master)
$ git tag
v1.0_lightTag
v2.0_weightTag

ZLF@DESKTOP-ZLFGYJL MINGW64 /k/库/ZLF/Desktop/GitTest (master)
$ git tag -l
v1.0_lightTag
v2.0_weightTag

ZLF@DESKTOP-ZLFGYJL MINGW64 /k/库/ZLF/Desktop/GitTest (master)
$ git tag --list
v1.0_lightTag
v2.0_weightTag

```

3.4 查看标签

命令: `git tag tagName`

这个命令通过指定标签名, 查看标签信息与对应的提交信息, 查看最开始创建的轻量标签“v1.0_lightTag”:

```

ZLF@DESKTOP-ZLFGYJL MINGW64 /k/库/ZLF/Desktop/GitTest (master)
$ git show v1.0_lightTag
commit ee444d7f76c16b7248d6d084bc5e70ea3c72ab21 (tag: v1.0_lightTag)
Author: Zhang Longfei <153679929@qq.com>
Date: Mon Feb 24 21:29:38 2020 +0800

    提交了一个姓名文件

diff --git a/name.txt b/name.txt
new file mode 100644
index 0000000..05a6057
--- /dev/null
+++ b/name.txt
@@ -0,0 +1 @@
+张龙飞

```

查看后面创建的附注标签“v2.0_weightTag”:

```

ZLF@DESKTOP-ZLFGYJL MINGW64 /k/库/ZLF/Desktop/GitTest (master)
$ git show v2.0_weightTag
tag v2.0_weightTag
Tagger: Zhang Longfei <153679929@qq.com>
Date: Mon Feb 24 21:48:06 2020 +0800

    相比v1.0, 新增了一个文件

commit bfaf0f5b146f8fed4be0ae67a87b0f911369e768 (HEAD -> master, tag: v2.0_weightTag)
Author: Zhang Longfei <153679929@qq.com>
Date: Mon Feb 24 21:47:01 2020 +0800

    新增了学号信息文件

diff --git a/id.txt b/id.txt
new file mode 100644
index 0000000..0a6a450
--- /dev/null
+++ b/id.txt
@@ -0,0 +1 @@
+0174115

```

通过对比我们可以看见, 轻量标签仅有提交信息, 而附注标签多了相关的标签信息(打标者、打标时间等)。

3.5 删除标签

命令: `git tag -d|--delete tagName`

```
ZLF@DESKTOP-ZLFGYJL MINGW64 /k/库/ZLF/Desktop/GitTest (master)
$ git tag --list
v1.0_lightTag
v2.0_weightTag

ZLF@DESKTOP-ZLFGYJL MINGW64 /k/库/ZLF/Desktop/GitTest (master)
$ git tag --delete v1.0_lightTag
Deleted tag 'v1.0_lightTag' (was ee444d7)

ZLF@DESKTOP-ZLFGYJL MINGW64 /k/库/ZLF/Desktop/GitTest (master)
$ git tag --list
v2.0_weightTag
```

3.6 后期打标签

命令: `git tag [-a] tagName 校验和/部分校验和`

这个命令用于对过去的某个没有打标的提交打标, 需要在命令的末尾指定该提交的校验和或部分校验和, 还可以添加-a 选项打标为附注标签

通过查看日志我们可以发现第一次的提交(校验和为 ee444d)没有标签(之前打了一个轻量标签, 刚才演示删除标签时删了, 可以回过去查看), 使用此命令

```
ZLF@DESKTOP-ZLFGYJL MINGW64 /k/库/ZLF/Desktop/GitTest (master)
$ git log --pretty=oneline
bfaf0f5b146f8fed4be0ae67a87b0f911369e768 (HEAD -> master, tag: v2.0_weightTag) 新增了学号信息文件
ee444d7f76c16b7248d6d084bc5e70ea3c72ab21 提交了一个姓名文件

ZLF@DESKTOP-ZLFGYJL MINGW64 /k/库/ZLF/Desktop/GitTest (master)
$ git tag -a v1.0_weightTag --message "这个标签是重新打过的附注标签" ee444d 刚刚删除了这个提交的标签

ZLF@DESKTOP-ZLFGYJL MINGW64 /k/库/ZLF/Desktop/GitTest (master)
$ git log --pretty=oneline
bfaf0f5b146f8fed4be0ae67a87b0f911369e768 (HEAD -> master, tag: v2.0_weightTag) 新增了学号信息文件
ee444d7f76c16b7248d6d084bc5e70ea3c72ab21 (tag: v1.0_weightTag) 提交了一个姓名文件
```

3.7 共享标签

命令: `git push url/shortName tagName` (推送单个标签)

`git push url/shortName --tags` (推送多个标签)

默认情况下, `git push` 命令并不会传送标签到远程仓库服务器上, 所以在创建完标签后你必须显式地推送标签到共享服务器上。

```

ZLF@DESKTOP-ZLFGYJL MINGW64 /k/库/ZLF/Desktop/GitTest (master)
$ git remote add test https://gitee.com/zhang_153/test

ZLF@DESKTOP-ZLFGYJL MINGW64 /k/库/ZLF/Desktop/GitTest (master)
$ git push https://gitee.com/zhang_153/test v1.0_weightTag
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 4 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (4/4), 438 bytes | 438.00 KiB/s, done.
Total 4 (delta 0), reused 0 (delta 0)
remote: Powered by GITEE.COM [GNK-3.8]
To https://gitee.com/zhang_153/test
 * [new tag]          v1.0_weightTag -> v1.0_weightTag

ZLF@DESKTOP-ZLFGYJL MINGW64 /k/库/ZLF/Desktop/GitTest (master)
$ git push test --tags
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 4 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (4/4), 497 bytes | 497.00 KiB/s, done.
Total 4 (delta 0), reused 0 (delta 0)
remote: Powered by GITEE.COM [GNK-3.8]
To https://gitee.com/zhang_153/test
 * [new tag]          v2.0_weightTag -> v2.0_weightTag

```

可以在服务器上看到推送的标签：

The screenshot shows the Gitee web interface for the repository '张153 / test'. The URL in the browser is 'https://gitee.com/zhang_153/test/tree/v1.0_weightTag/'. The repository page shows the repository name, license (GPL-3.0), and navigation links for code, issues, pull requests, and attachments. Below the repository information, there is a section for tags. The 'v1.0_weightTag' tag is selected, and a dropdown menu shows the list of tags: 'v2.0_weightTag' and 'v1.0_weightTag'.

3.8 检出标签

命令：`git checkout tagName`

如果你想查看某个标签所指向的文件版本，就可以使用 `git checkout` 命令。

```
ZLF@DESKTOP-ZLFGYJL MINGW64 /k/库/ZLF/Desktop/GitTest (master)
$ git checkout v1.0_weightTag
Note: switching to 'v1.0_weightTag'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by switching back to a branch.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -c with the switch command. Example:

    git switch -c <new-branch-name>

Or undo this operation with:

    git switch -

Turn off this advice by setting config variable advice.detachedHead to false
HEAD is now at ee444d7 提交了一个姓名文件
```

注意到最下面的提示信息，这条命令会使你的仓库处于“分离头指针（detached HEAD）”状态——这个状态有些不好的副作用：

在“分离头指针”状态下，如果你做了某些更改然后提交它们，标签不会发生变化，但你的新提交将不属于任何分支，并且将无法访问，除非确切的提交哈希。因此，如果你需要进行更改——比如说你正在修复旧版本的错误——这通常需要创建一个新分支。

4 分支管理

分支管理可以将版本进行灵活有效的控制，举个例子，假设你准备开发一个新功能，但需要两周才能完成，第一周写了 60%，如果提交，由于代码还没写完，不完整的代码库会导致别人不能干活，如果等代码全部写完在一次提交，又会存在丢失每天进度的风险。有了分支，可以避免上述问题，创建一个属于自己的分支，别人看不到，还继续在原来的分支上正常工作，而我们在自己的分支上干活，想提交就提交，直到开发完毕后，在一次性合并到原来的分支上，这样，即安全又不影响别人工作。

其特点是：Git 分支是与众不同的，无论创建、切换、和删除分支，Git 在非常短的时间内就能完成，无论版本库是 1 个文件还是 1 万个文件。

对于各类的分支管理，可以进行如下理解：

master 分支：在版本回退中，每次提交，Git 都把它们串成一条时间线，在 git 里，这个分支叫主分支，即 master 分支，HEAD 严格来说不是指向提交，而是指向 master，master 才是指向提交的，所以，HEAD 指向的就是当前分支。每次提交，master 分支都会向前移动一步，这样，随着不断提交，master 分支的线也越来越长；

develop 分支：与 master 分支并行的另一个分支，即称之为 develop 分支。当 develop 分支的源码到达了一个稳定状态待发布，所有的代码变更需要以某种方式合并到 master 分

支，然后标记一个版本号；

release 分支：Release 分支可能从 develop 分支分离而来，但是一定要合并到 develop 和 master 分支上，它是为新产品的发布做准备的，允许开发者在最后时刻做一些细小的修改。他们允许小 bugs 的修改和准备发布元数据（版本号，开发时间等等）。当在 Release 分支完成这些所有工作以后，对于下一次打的发布，develop 分支接收 features 会更加明确。

具体分支管理的实例如下。

4.1 分支的基本命令

查看分支：\$ git branch

创建分支：\$ git branch [name]

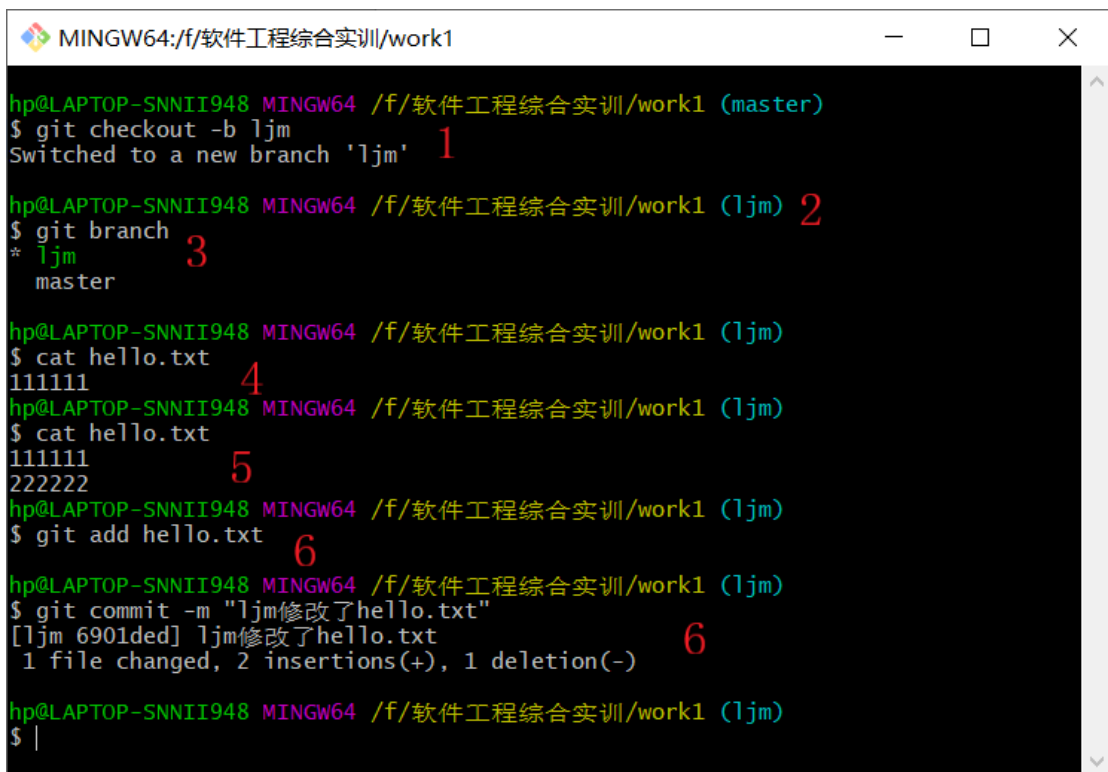
切换分支：\$ git checkout [name]

创建并切换分支：\$ git checkout -b [name]

合并某分支到当前分支：\$ git merge [name]

删除分支：\$ git branch -d [name]

4.2 创建分支



```
hp@LAPTOP-SNNII948 MINGW64 /f/软件工程综合实训/work1 (master)
$ git checkout -b ljm
Switched to a new branch 'ljm' 1

hp@LAPTOP-SNNII948 MINGW64 /f/软件工程综合实训/work1 (ljm) 2
$ git branch
* ljm 3
  master

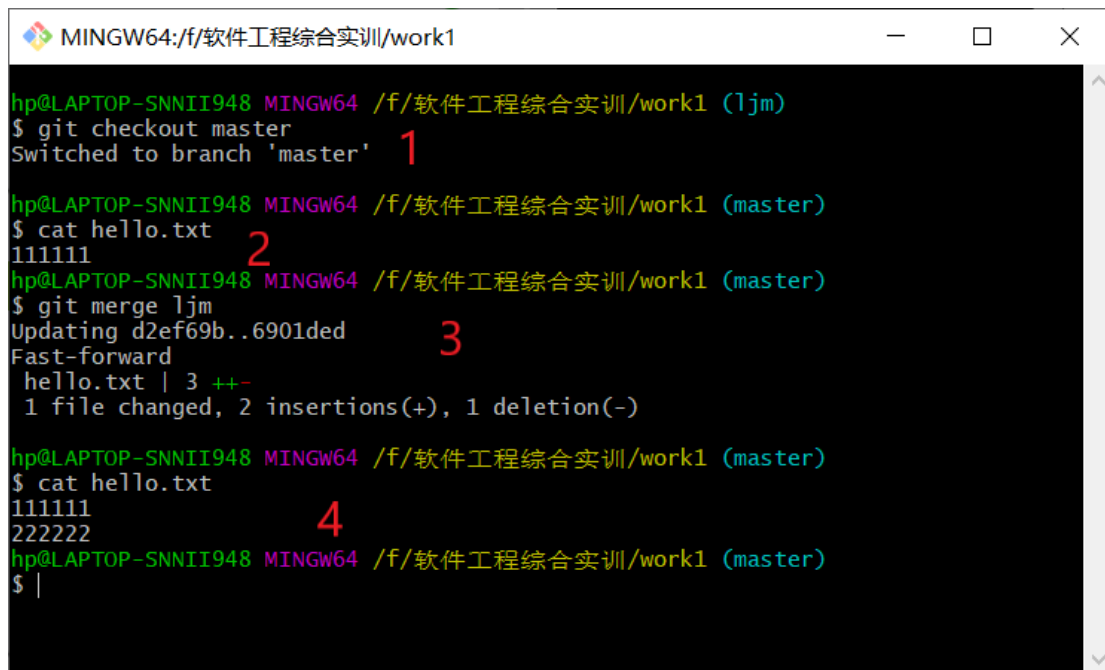
hp@LAPTOP-SNNII948 MINGW64 /f/软件工程综合实训/work1 (ljm)
$ cat hello.txt
111111 4
hp@LAPTOP-SNNII948 MINGW64 /f/软件工程综合实训/work1 (ljm)
$ cat hello.txt
111111
222222 5
hp@LAPTOP-SNNII948 MINGW64 /f/软件工程综合实训/work1 (ljm)
$ git add hello.txt 6

hp@LAPTOP-SNNII948 MINGW64 /f/软件工程综合实训/work1 (ljm)
$ git commit -m "ljm修改了hello.txt"
[ljm 6901ded] ljm修改了hello.txt
1 file changed, 2 insertions(+), 1 deletion(-) 6

hp@LAPTOP-SNNII948 MINGW64 /f/软件工程综合实训/work1 (ljm)
$ |
```

- ①新建分支 ljm，并进入新分支；
- ②原主分支 master 转换成了 ljm；
- ③查看目前的所有分支；
- ④查看项目中原文件 hello.txt 的内容；
- ⑤增加一行内容 222222；
- ⑥将修改进行提交。

4.3 合并分支



```
MINGW64:/f/软件工程综合实训/work1

hp@LAPTOP-SNNII948 MINGW64 /f/软件工程综合实训/work1 (ljm)
$ git checkout master
Switched to branch 'master' 1

hp@LAPTOP-SNNII948 MINGW64 /f/软件工程综合实训/work1 (master)
$ cat hello.txt
111111 2

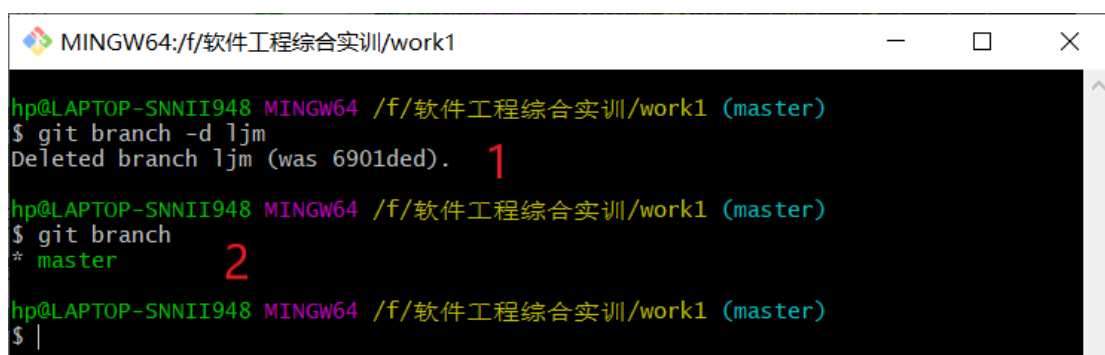
hp@LAPTOP-SNNII948 MINGW64 /f/软件工程综合实训/work1 (master)
$ git merge ljm
Updating d2ef69b..6901ded 3
Fast-forward
 hello.txt | 3 ++-
 1 file changed, 2 insertions(+), 1 deletion(-)

hp@LAPTOP-SNNII948 MINGW64 /f/软件工程综合实训/work1 (master)
$ cat hello.txt
111111
222222 4

hp@LAPTOP-SNNII948 MINGW64 /f/软件工程综合实训/work1 (master)
$ |
```

- ①切换到主分支 master;
- ②查看 hello.txt, 发现内容并没有变换, 即 master 并没有收到 ljm 的影响;
- ③将 ljm 的修改合并到总分支 master;
- ④查看 hello.txt 的内容, 发现主分支 master 和分支 ljm 最新的修改变成一致。

4.4 删除分支



```
MINGW64:/f/软件工程综合实训/work1

hp@LAPTOP-SNNII948 MINGW64 /f/软件工程综合实训/work1 (master)
$ git branch -d ljm
Deleted branch ljm (was 6901ded). 1

hp@LAPTOP-SNNII948 MINGW64 /f/软件工程综合实训/work1 (master)
$ git branch
* master 2

hp@LAPTOP-SNNII948 MINGW64 /f/软件工程综合实训/work1 (master)
$ |
```

- ①合并分支 ljm 后, 删除分支 ljm;
- ②查看所有分支, 发现只剩下主分支 master。

4.5 解决冲突

```
MINGW64:/f/软件工程综合实训/work1

hp@LAPTOP-SNNII948 MINGW64 /f/软件工程综合实训/work1 (master)
$ git checkout -b ljm1
Switched to a new branch 'ljm1' 1

hp@LAPTOP-SNNII948 MINGW64 /f/软件工程综合实训/work1 (ljm1)
$ cat hello.txt
111111
222222 1
hp@LAPTOP-SNNII948 MINGW64 /f/软件工程综合实训/work1 (ljm1)
$ cat hello.txt
111111
222222 1
333333
hp@LAPTOP-SNNII948 MINGW64 /f/软件工程综合实训/work1 (ljm1)
$ git add hello.txt 2
hp@LAPTOP-SNNII948 MINGW64 /f/软件工程综合实训/work1 (ljm1)
$ git commit -m "ljm1修改增加了一行333333"
[ljm1 f5d347d] ljm1修改增加了一行333333
1 file changed, 2 insertions(+), 1 deletion(-) 2
hp@LAPTOP-SNNII948 MINGW64 /f/软件工程综合实训/work1 (ljm1)
$ git checkout master
Switched to branch 'master' 3
hp@LAPTOP-SNNII948 MINGW64 /f/软件工程综合实训/work1 (master)
$ cat hello.txt
111111
222222 3
333333
hp@LAPTOP-SNNII948 MINGW64 /f/软件工程综合实训/work1 (master)
$ cat hello.txt
111111
222222 3
333333
444444
hp@LAPTOP-SNNII948 MINGW64 /f/软件工程综合实训/work1 (master)
$ git add hello.txt 4
hp@LAPTOP-SNNII948 MINGW64 /f/软件工程综合实训/work1 (master)
$ git commit -m "master修改增加了一行444444"
[master 756a047] master修改增加了一行444444
1 file changed, 3 insertions(+), 1 deletion(-) 4
hp@LAPTOP-SNNII948 MINGW64 /f/软件工程综合实训/work1 (master)
$ git merge ljm1
Auto-merging hello.txt
CONFLICT (content): Merge conflict in hello.txt 5
Automatic merge failed; fix conflicts and then commit the result.
hp@LAPTOP-SNNII948 MINGW64 /f/软件工程综合实训/work1 (master|MERGING)
$ git status
```

- ①新建一个新分支 ljm1，在 hello.txt 添加一行内容 333333；
- ②提交修改内容的相关信息；
- ③切换到 master 分支，并且在原有基础上添加一行内容 444444；
- ④提交修改内容的相关信息；
- ⑤在 hello.txt 上发现冲突；


```
MINGW64:/f/软件工程综合实训/work1
hp@LAPTOP-SNNII948 MINGW64 /f/软件工程综合实训/work1 (master)
$ git merge ljm1
Auto-merging hello.txt
CONFLICT (content): Merge conflict in hello.txt
Automatic merge failed; fix conflicts and then commit the result.

hp@LAPTOP-SNNII948 MINGW64 /f/软件工程综合实训/work1 (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:   hello.txt

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

hp@LAPTOP-SNNII948 MINGW64 /f/软件工程综合实训/work1 (master|MERGING)
$ cat hello.txt
111111
222222
<<<<<< HEAD
333333
444444
=====
333333
>>>>>> ljm1

hp@LAPTOP-SNNII948 MINGW64 /f/软件工程综合实训/work1 (master|MERGING)
$ cat hello.txt
111111
222222
333333
444444

hp@LAPTOP-SNNII948 MINGW64 /f/软件工程综合实训/work1 (master|MERGING)
$ git log
commit 756a0478102b75b685d987c6a459922c95b1d87e (HEAD -> master)
Author: 2311762665 <2311762665@qq.com>
Date:   Sun Feb 23 13:41:04 2020 +0800

    master修改增加了一行444444

commit 1fee7da22b41d162fa2002445b4842f730e34a05
Author: 2311762665 <2311762665@qq.com>
Date:   Sun Feb 23 13:37:20 2020 +0800

    add hello.txt

hp@LAPTOP-SNNII948 MINGW64 /f/软件工程综合实训/work1 (master|MERGING)
$
```

⑥查看冲突状态;

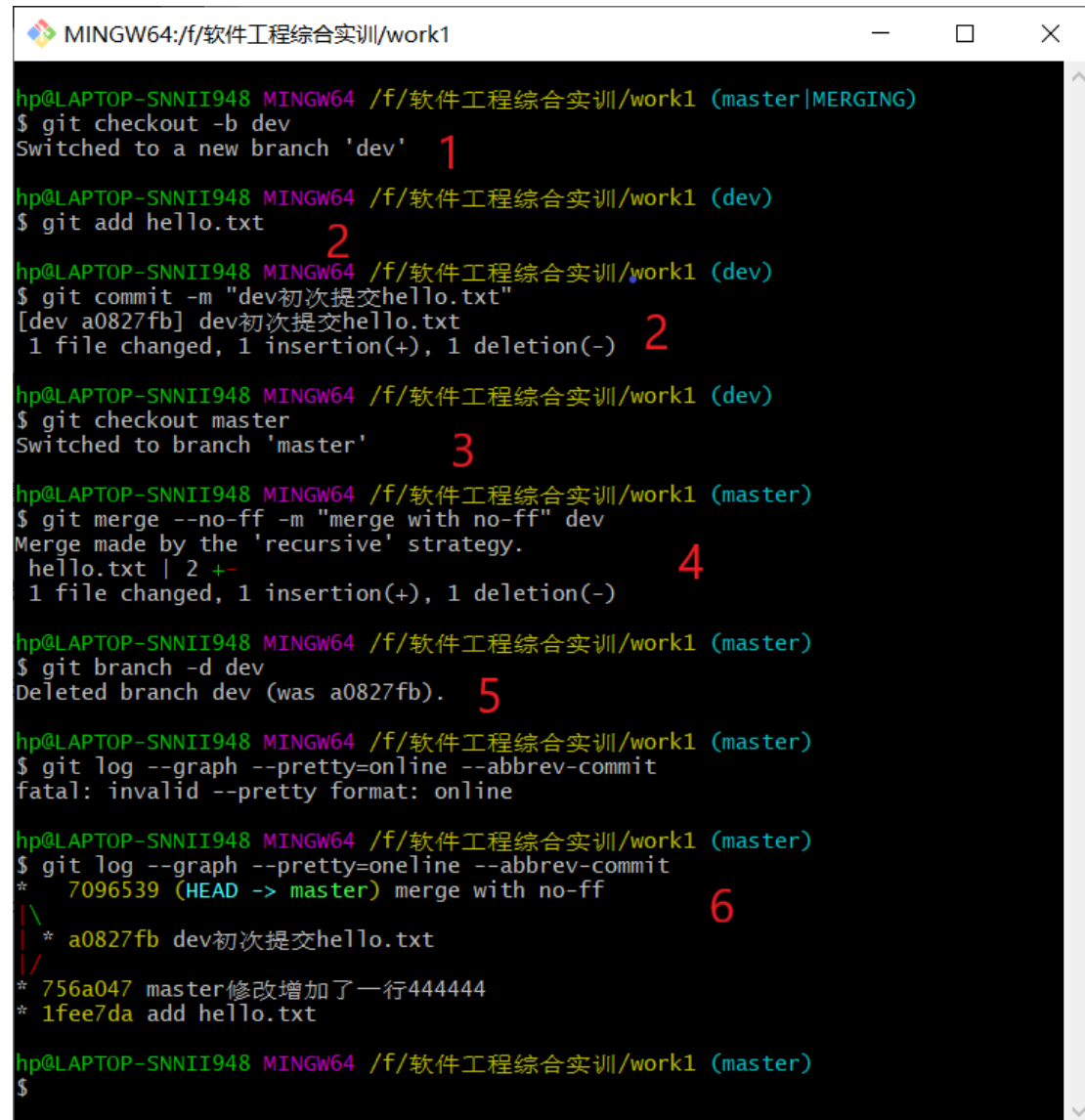
⑦查看冲突的信息, <<<<<<,=====>>>>>> 标记出不同分支的内容, 其中, <<<<<<HEAD 是指主分支修改的内容, >>>>>>ljm1 是指分支 ljm1 修改的内容;

⑧修改保存后的内容;

⑨使用 git log 命令可查看分支合并的情况;

4.6 分支管理

通常合并分支时，git 一般使用“Fast forward”模式，在这种模式下，删除分支后，会丢失分支信息，可通过使用带参数--no-ff 来禁用“Fast forward”模式。操作情况如下：



```
MINGW64:/f/软件工程综合实训/work1
hp@LAPTOP-SNNII948 MINGW64 /f/软件工程综合实训/work1 (master|MERGING)
$ git checkout -b dev
Switched to a new branch 'dev' 1
hp@LAPTOP-SNNII948 MINGW64 /f/软件工程综合实训/work1 (dev)
$ git add hello.txt 2
hp@LAPTOP-SNNII948 MINGW64 /f/软件工程综合实训/work1 (dev)
$ git commit -m "dev初次提交hello.txt"
[dev a0827fb] dev初次提交hello.txt
1 file changed, 1 insertion(+), 1 deletion(-) 2
hp@LAPTOP-SNNII948 MINGW64 /f/软件工程综合实训/work1 (dev)
$ git checkout master
Switched to branch 'master' 3
hp@LAPTOP-SNNII948 MINGW64 /f/软件工程综合实训/work1 (master)
$ git merge --no-ff -m "merge with no-ff" dev
Merge made by the 'recursive' strategy.
hello.txt | 2 +-
1 file changed, 1 insertion(+), 1 deletion(-) 4
hp@LAPTOP-SNNII948 MINGW64 /f/软件工程综合实训/work1 (master)
$ git branch -d dev
Deleted branch dev (was a0827fb). 5
hp@LAPTOP-SNNII948 MINGW64 /f/软件工程综合实训/work1 (master)
$ git log --graph --pretty=online --abbrev-commit
fatal: invalid --pretty format: online
hp@LAPTOP-SNNII948 MINGW64 /f/软件工程综合实训/work1 (master)
$ git log --graph --pretty=oneline --abbrev-commit
* 7096539 (HEAD -> master) merge with no-ff 6
| \
| * a0827fb dev初次提交hello.txt
| /
* 756a047 master修改增加了一行444444
* 1fee7da add hello.txt
hp@LAPTOP-SNNII948 MINGW64 /f/软件工程综合实训/work1 (master)
$
```

- ①创建一个 dev 分支；
- ②提交内容的相关信息；
- ③切换到 master 分支；
- ④合并 dev 分支，使用命令 `git merge --no-ff -m “注释” dev`；
- ⑤删除 dev 分支，可观察到其版本号为 a0827fb；
- ⑥查看历史记录，能找到 dev 分支的版本号，分支信息并未丢失。

4.7bug 分支

在开发中，如果碰到 bug 问题，就需要进行修复。在 Git 中，每个 bug 都可以通过一个临时分支来修复，修复完成后，合并分支，然后将临时分支删除。操作情况如下：

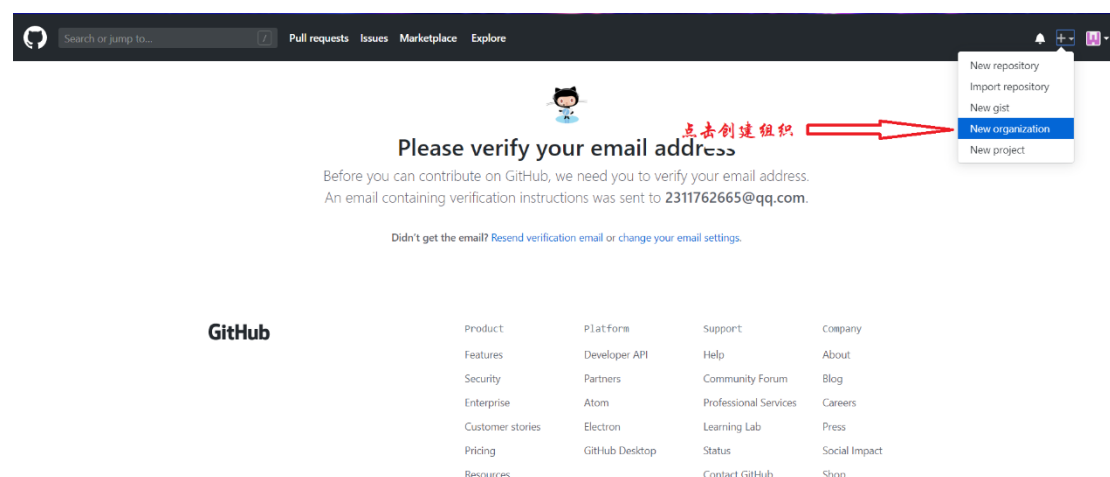
```
MINGW64:/f/软件工程综合实训/work1
hp@LAPTOP-SNNII948 MINGW64 /f/软件工程综合实训/work1 (master)
$ git checkout -b 404
Switched to a new branch '404' 1
hp@LAPTOP-SNNII948 MINGW64 /f/软件工程综合实训/work1 (404)
$ cat hello.txt
111111
222222
333333
444444
hp@LAPTOP-SNNII948 MINGW64 /f/软件工程综合实训/work1 (404)
$ cat hello.txt
111111
222222
333333
444444
555555
hp@LAPTOP-SNNII948 MINGW64 /f/软件工程综合实训/work1 (404)
$ git add hello.txt
hp@LAPTOP-SNNII948 MINGW64 /f/软件工程综合实训/work1 (404)
$ git commit -m "404修改增加了一行555555"
[404 5840886] 404修改增加了一行555555
1 file changed, 1 insertion(+)
hp@LAPTOP-SNNII948 MINGW64 /f/软件工程综合实训/work1 (404)
$ git checkout master
Switched to branch 'master' 2
hp@LAPTOP-SNNII948 MINGW64 /f/软件工程综合实训/work1 (master)
$ git merge 404
Updating 7096539..5840886
Fast-forward
 hello.txt | 1 +
1 file changed, 1 insertion(+) 3
hp@LAPTOP-SNNII948 MINGW64 /f/软件工程综合实训/work1 (master)
$ cat hello.txt
111111
222222
333333
444444
555555
hp@LAPTOP-SNNII948 MINGW64 /f/软件工程综合实训/work1 (master)
$ git branch -d 404
Deleted branch 404 (was 5840886). 4
hp@LAPTOP-SNNII948 MINGW64 /f/软件工程综合实训/work1 (master)
$ git checkout dev
error: pathspec 'dev' did not match any file(s) known to git
hp@LAPTOP-SNNII948 MINGW64 /f/软件工程综合实训/work1 (master)
$ git checkout -b dev
Switched to a new branch 'dev' 5
hp@LAPTOP-SNNII948 MINGW64 /f/软件工程综合实训/work1 (dev)
$ git status
```

①假设开发接收到 bug 消息，通过创建一个 404 分支来修复；

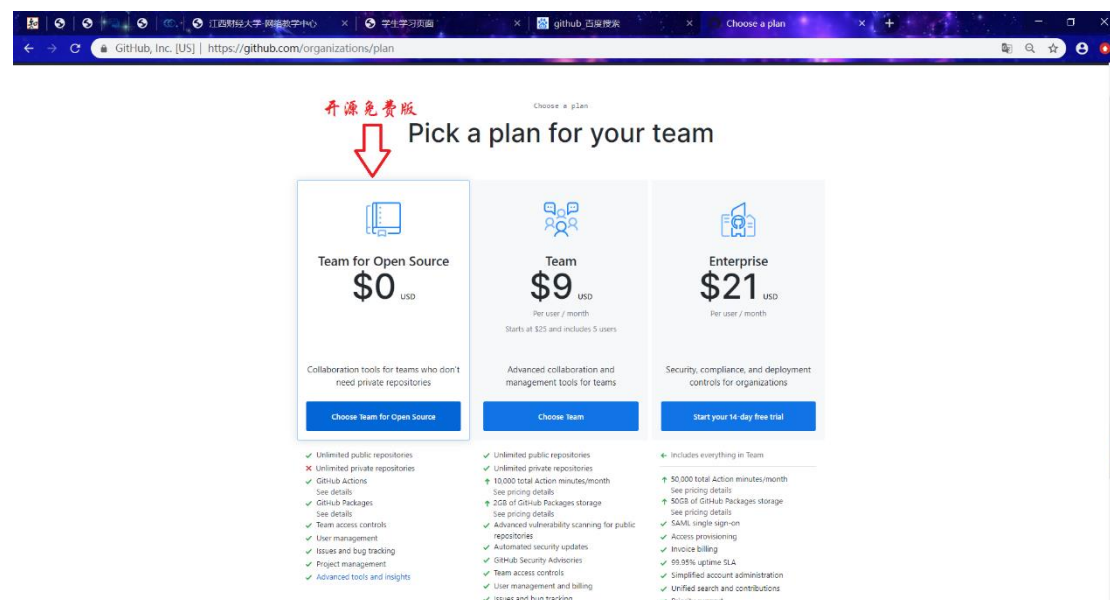
- ②切换到 master 分支上;
- ③完成合并 404 分支并查看内容;
- ④删除 404 分支;
- ⑤切换到其他分支继续工作。

4.8 多人协作

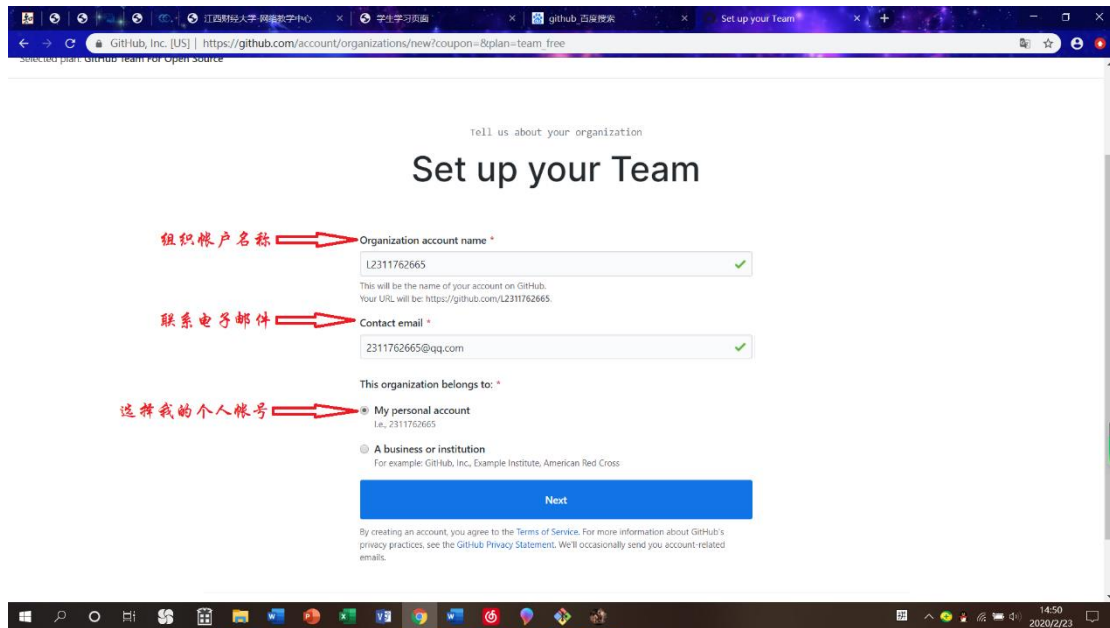
登录 GitHub 官网，以下介绍是基于英文解析，用户可以选择方便通过使用 Chrome 浏览器进行中文解析，更加适合理解。



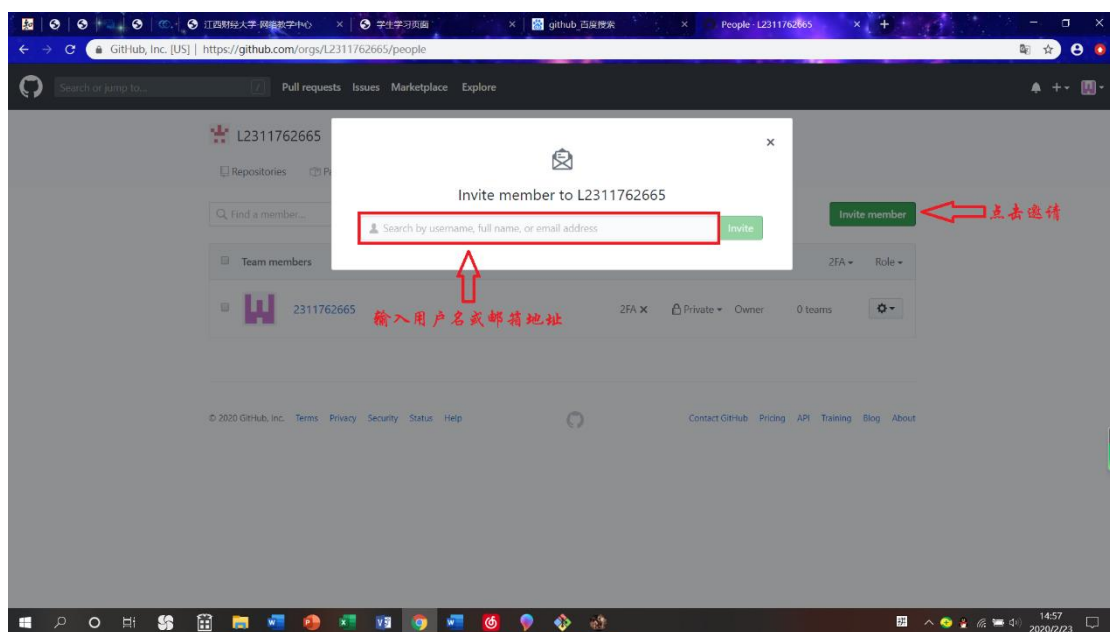
点击页面所指区域，新建组织，准备多人合作，可将每个成员理解为分支。



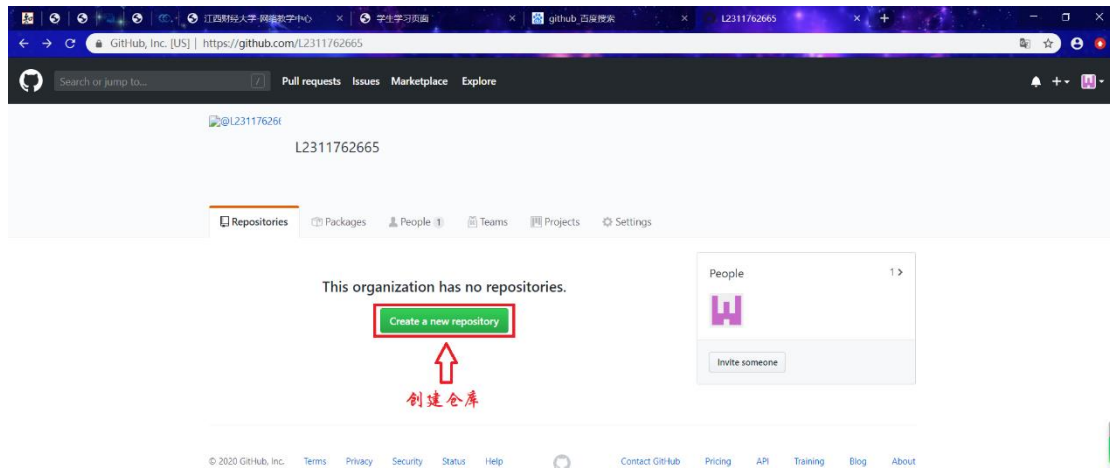
点击页面所指区域，选择\$0 版本，属于开源免费版，虽然功能不相比于其他二者，不过也供正常使用。



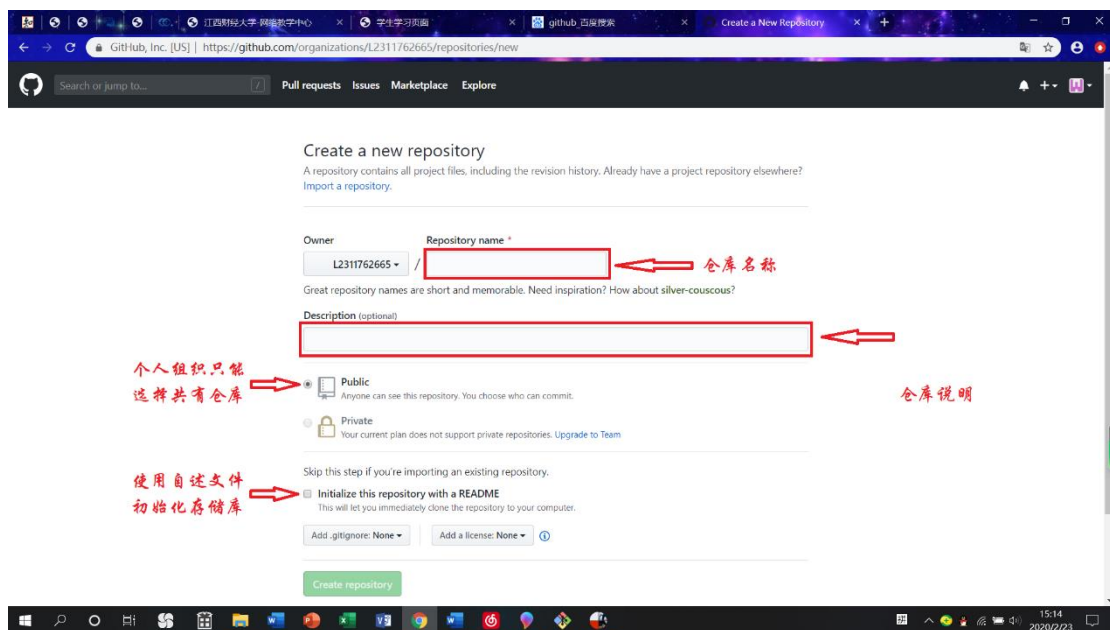
点击页面所指区域，填写组织帐户名称、联系电子邮件及选择所属为“我的个人帐号”。



点击页面所指区域，先点击邀请按钮，然后通过输入用户名或邮箱地址邀请同伴进入。



点击页面所指区域，开始建立仓库。



点击页面所指区域，填写仓库名称、仓库说明，个人组织只能默认选择共有仓库，选择是否使用自述文件初始化仓库。

5 MyEclipse 的使用

5.1 MyEclipse 上传项目至 GitHub

5.1.1 在 GitHub 上创建自己的代码仓库

Github 注册成功后，点击主页右上角的“+”号，选择 New repository 开创建代码仓库，创建页面如下图所示：

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Owner

Repository name *

WhitbyLi ▾

/ myRepository

Great repository names are short and memorable. Need inspiration? How about [upgraded-tribble?](#)

Description (optional)

MyEclipse开发项目共享

☒ **Public**

Anyone can see this repository. You choose who can commit.

☐ **Private**

You choose who can see and commit to this repository.

Skip this step if you're importing an existing repository.

☐ **Initialize this repository with a README**

This will let you immediately clone the repository to your computer.

Add .gitignore: **None** ▾

Add a license: **None** ▾

Create repository

填写说明：

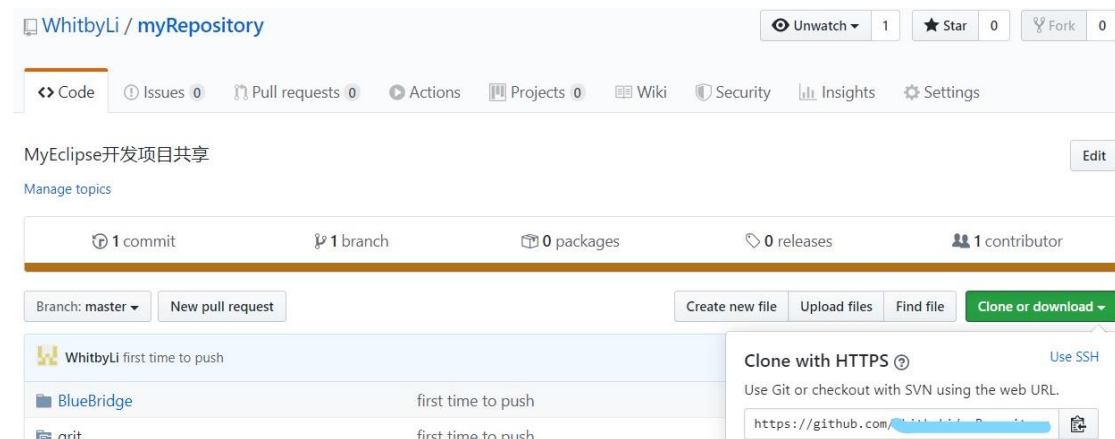
- (1) Repository name: 填写自己定义的仓库名（必填项）
- (2) Description: 填写描述（选填项）
- (3) Public: 保持选中（如果选 Private 可以选择谁能看到谁能提交，public 只能选谁能提交，private 是要钱的）
- (4) Initialize this repository with a README: 决定是否生成一个 README 文件来初始化仓库（可选可不选）
- (5) Create repository: 点击生成代码仓库

5.1.2 获取 GitHub 代码仓库地址

点击 Create repository 后生成 http 地址就是该代码仓库的地址，如下图所示：

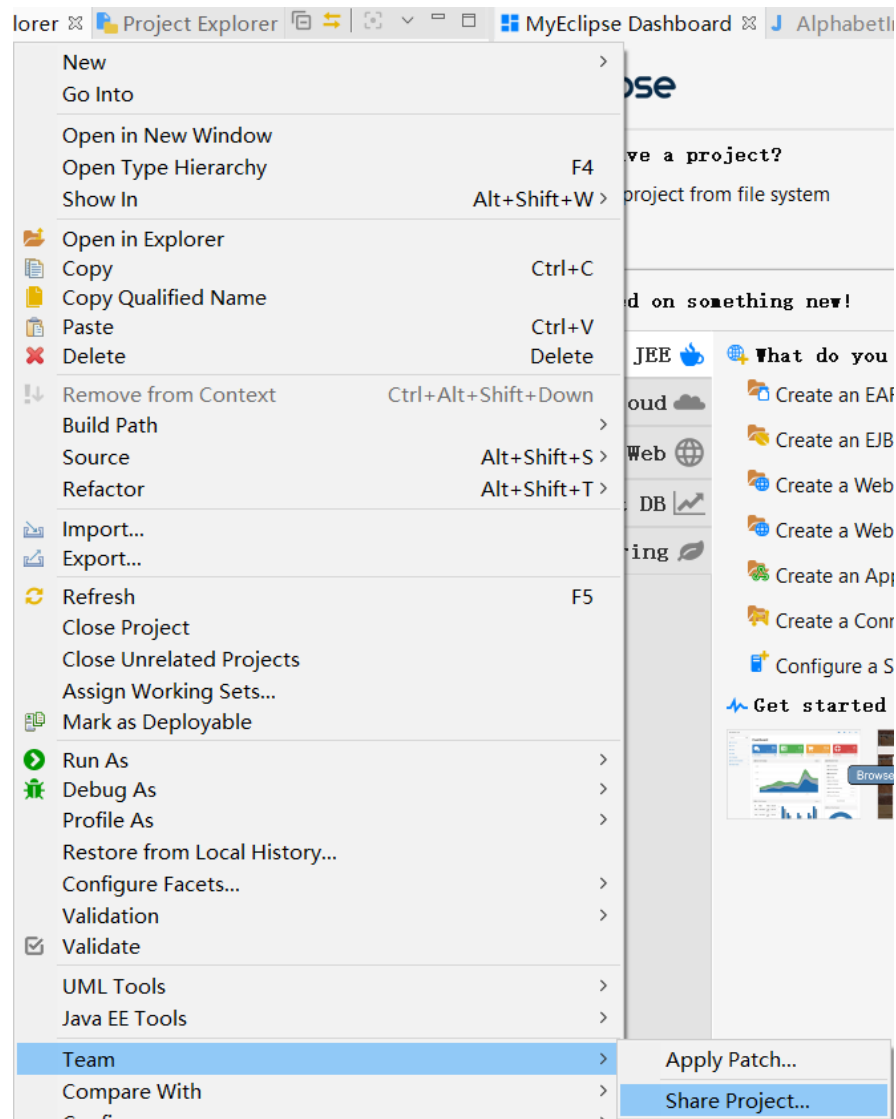


如果代码仓库已上传项目文件，可通过单机 Clone or download 获取 http 地址，如下图所示：

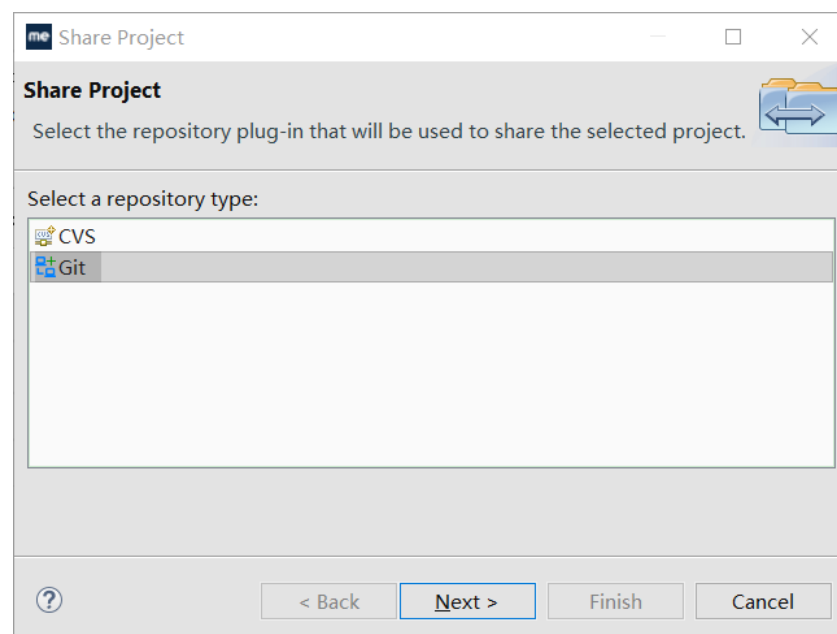


5.1.3 MyEclipse 上传项目至本地 git 仓库

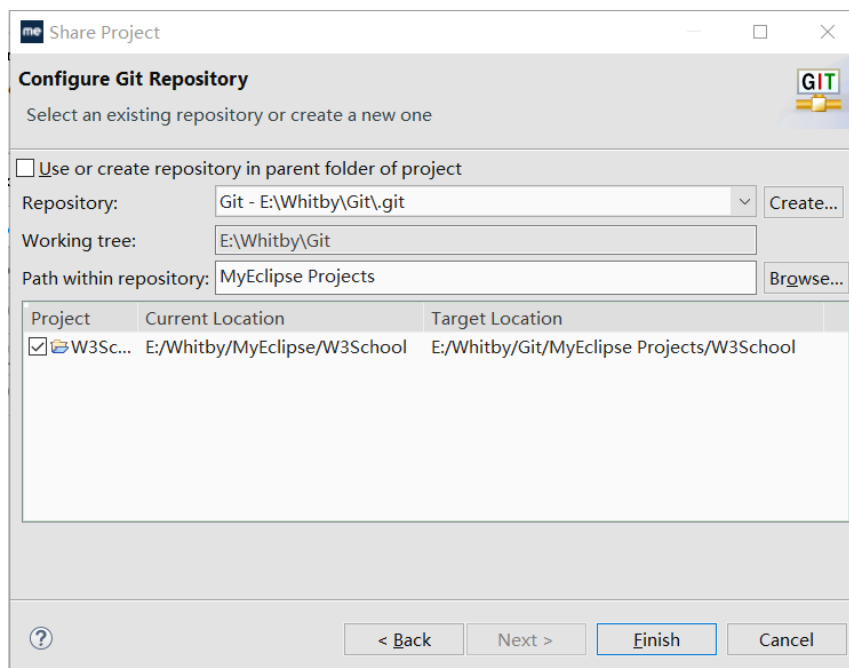
1、MyEclipse 高级版本一般自带 git 插件，可无需另外下载 git。右键点击项目 ->Team->Share Project，如下图所示：



2、选中 git，单击 next，如下图所示：



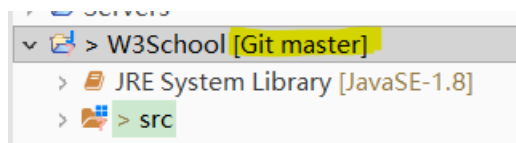
3、单击 Create 可选择指定目录，若该目录下不存在 git 仓库，则创建一个，否则选中该仓库，不创建，最终完成界面如下图所示：



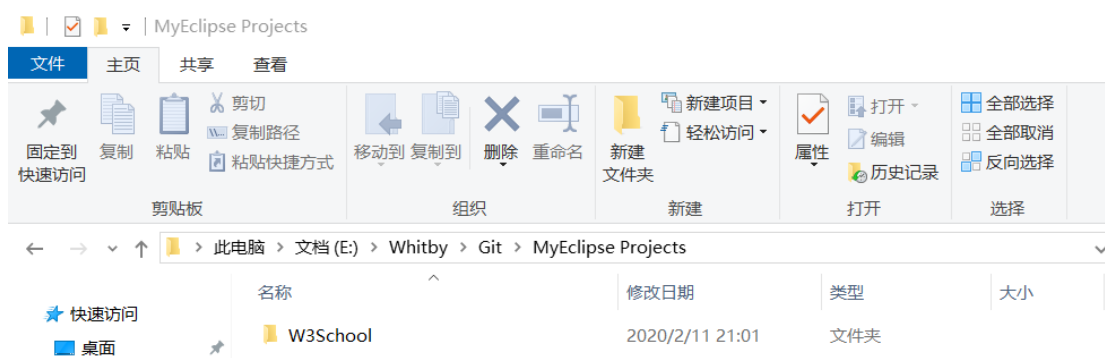
填写说明：

- (1) Use or create repository in parent folder of project: 使用项目所在父文件夹目录下的 git 仓库或创建，此处不勾选。
- (2) Repository: 单击 Create 按钮即可浏览选择或创建 git 仓库的位置。
- (3) Path within repository: 单击 Browser 选择上传到仓库的文件夹，选填项。

4、单击 finish 后，项目上传至本地 git 仓库，MyEclipse 界面如下：

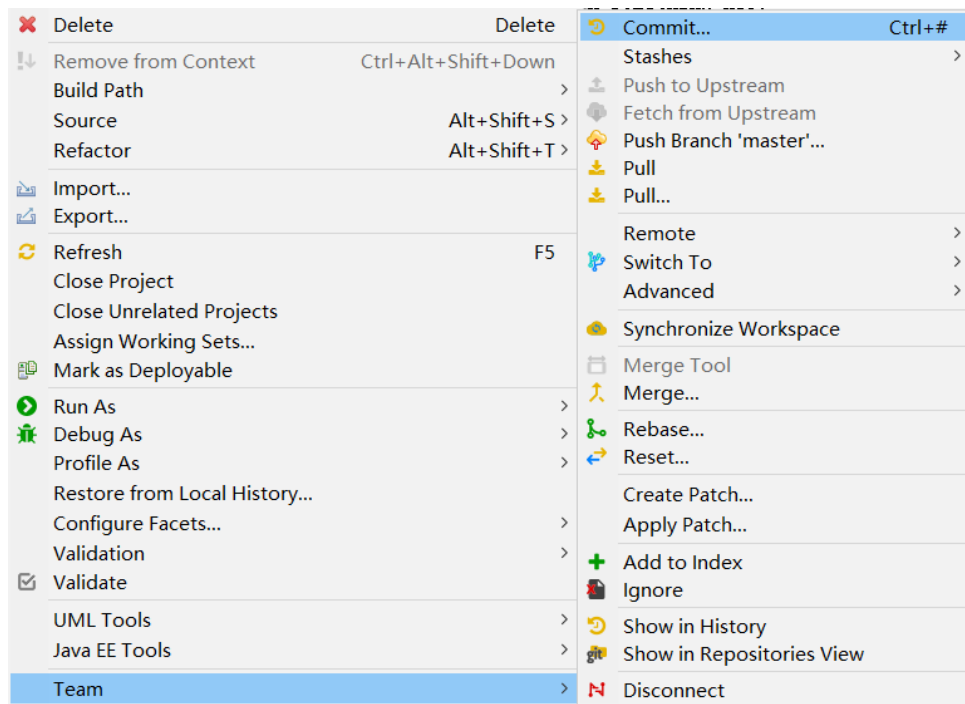


项目在文件管理器中的位置如下：

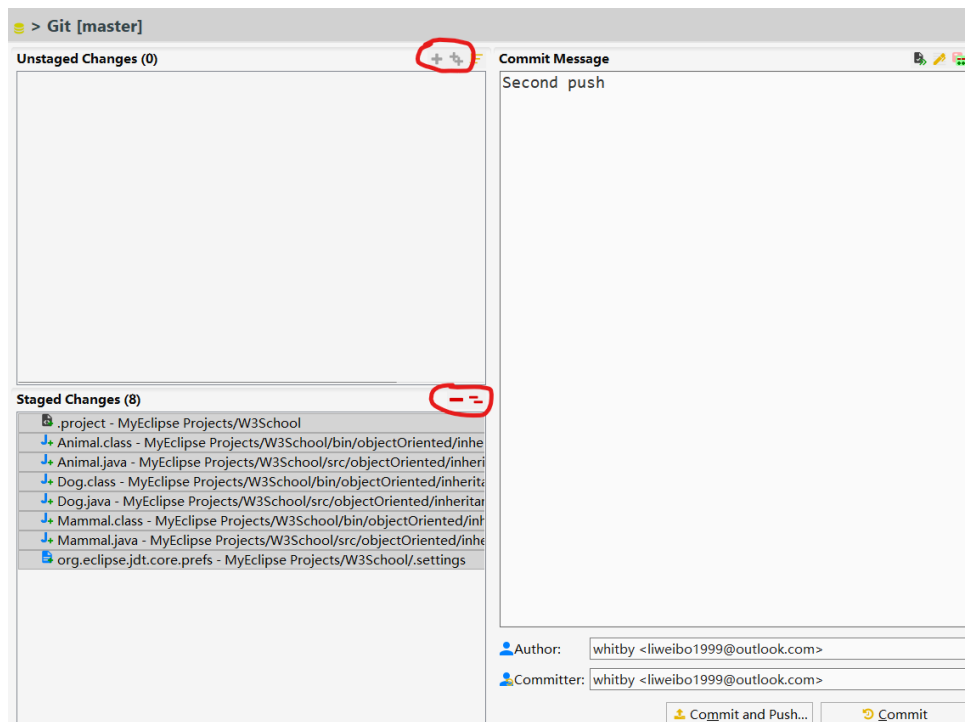


5.1.4 提交项目至远程

1、右键单击项目->Team->Commit，如下图所示：



2、选择项目文件，单击窗口右上角添加/移除符号，选择要上传的文件，并完成 Commit Message、Author、Committer 信息填写，单击右下角 Commit and Push 按钮进入下一步，如下图所示：



3、填写 git 仓库 url，输入 git 登录凭证，单击 next：如下图所示：

Push Branch master

Destination Git Repository
Enter the location of the destination repository.

Remote name:

Location

URI:

Host:

Repository path:

Connection

Protocol:

Port:

Authentication

User:

Password:

☒ Store in Secure Store

操作说明：

- (1) Remote name：可任意填写。
- (2) URI：GitHub 网站上创建的代码仓库地址，完成填写后，Host、Repository path、Protocol 自动填充。
- (3) Authentication：为登录 GitHub 凭证，输入登录的用户名及密码即可。
- (4) Store in Secure Store：选择，记住用户名、密码。

4、单击 next，进入下一步操作，默认填写如下：

Push Branch master

Push to branch in remote
Select a remote and the name the branch should have in the remote.

Source:

6c3d0e7 Second push

Destination:

Remote:

Branch:

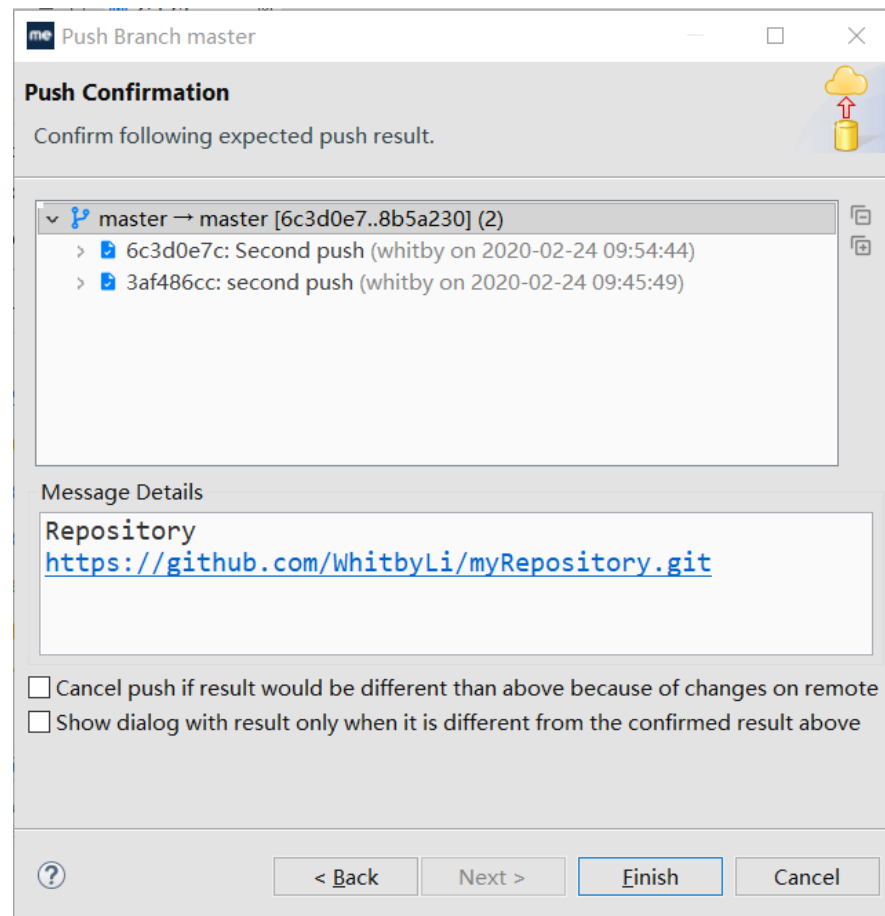
☒ Configure upstream for push and pull

When pulling:

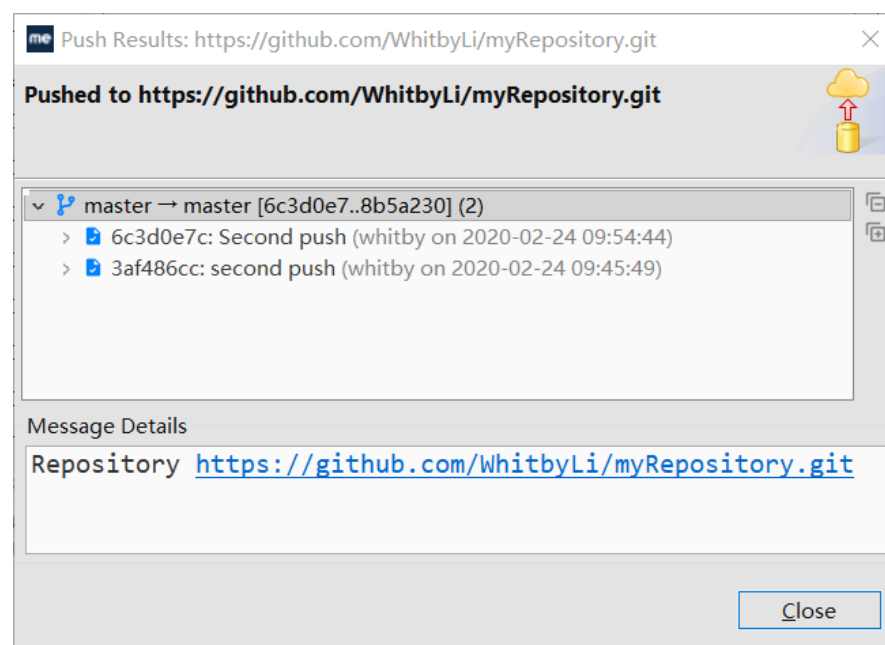
☐ Force overwrite branch in remote if it exists and has diverged

[Show advanced push dialog](#)

5、单击 next 进入下一步，显示上传的相关信息，如图所示：



6、单击 finish 即可完成上传，MyEclipse 完成界面如下：



GitHub 官网代码仓库上传项目后界面如下：

MyEclipse开发项目共享 Edit

[Manage topics](#)

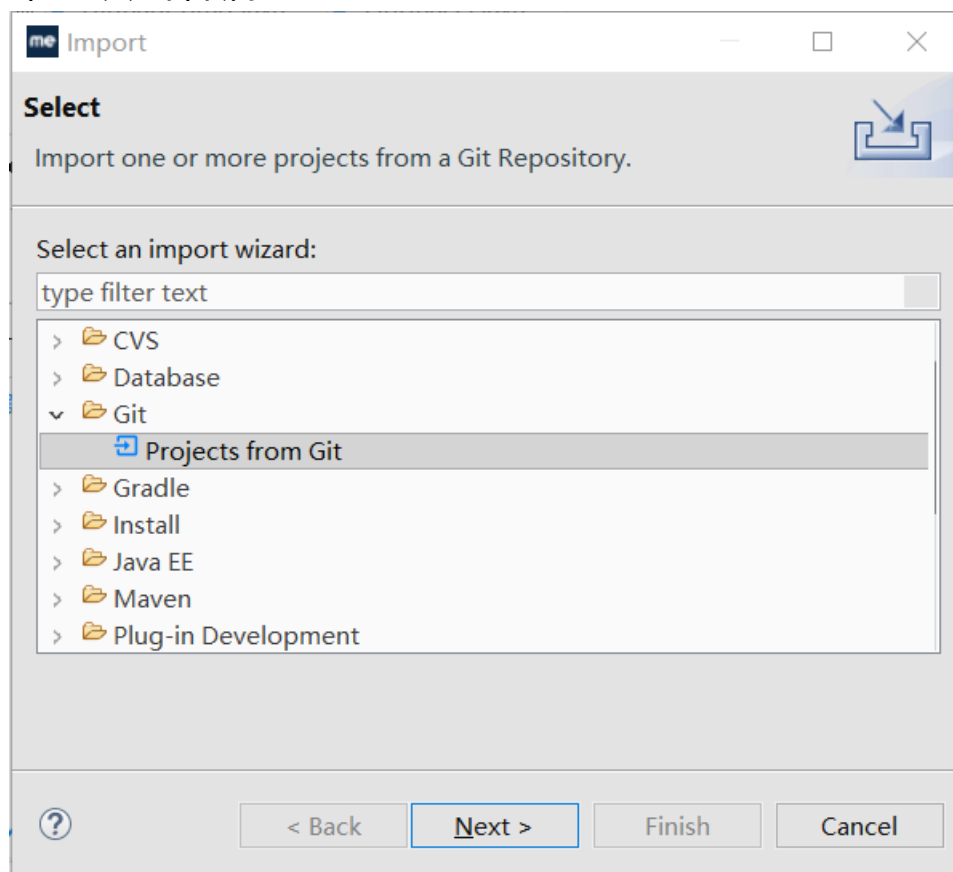
3 commits 1 branch 0 packages 0 releases 1 contributor

Branch: master New pull request Create new file Upload files Find file Clone or download

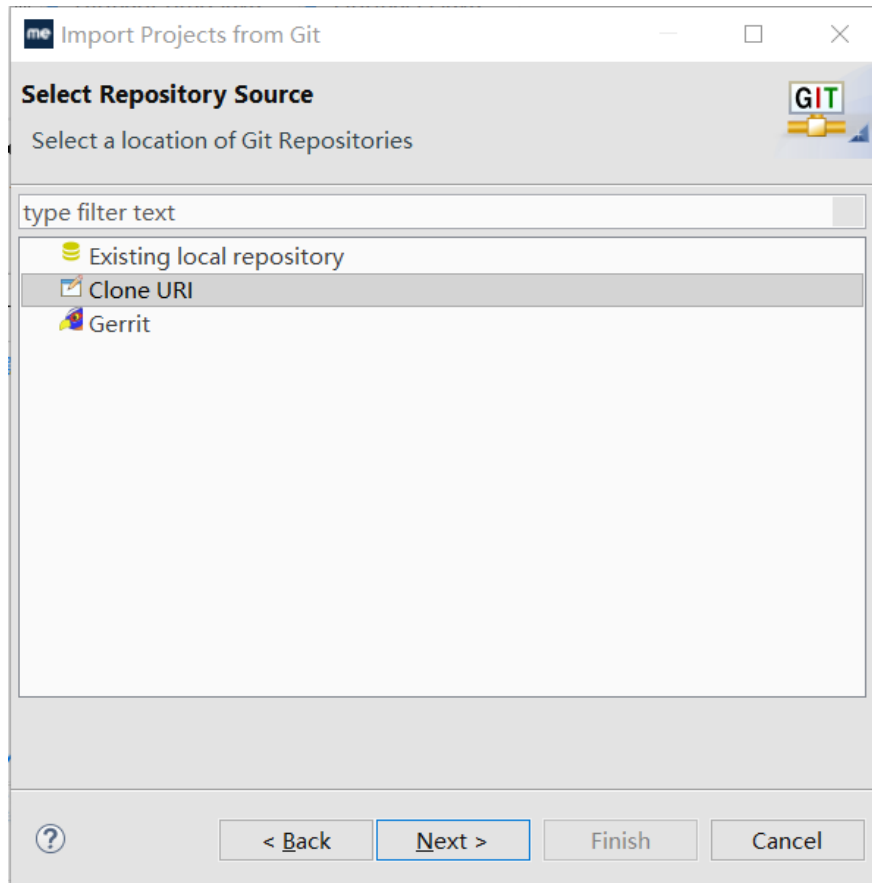
WhitbyLi Second push		Latest commit 6c3d0e7 16 minutes ago
BlueBridge	first time to push	14 hours ago
MyEclipse Projects/W3School	Second push	16 minutes ago
grit	first time to push	14 hours ago
code.txt	first time to push	14 hours ago

5.2 GitHub 导入项目至 MyEclipse

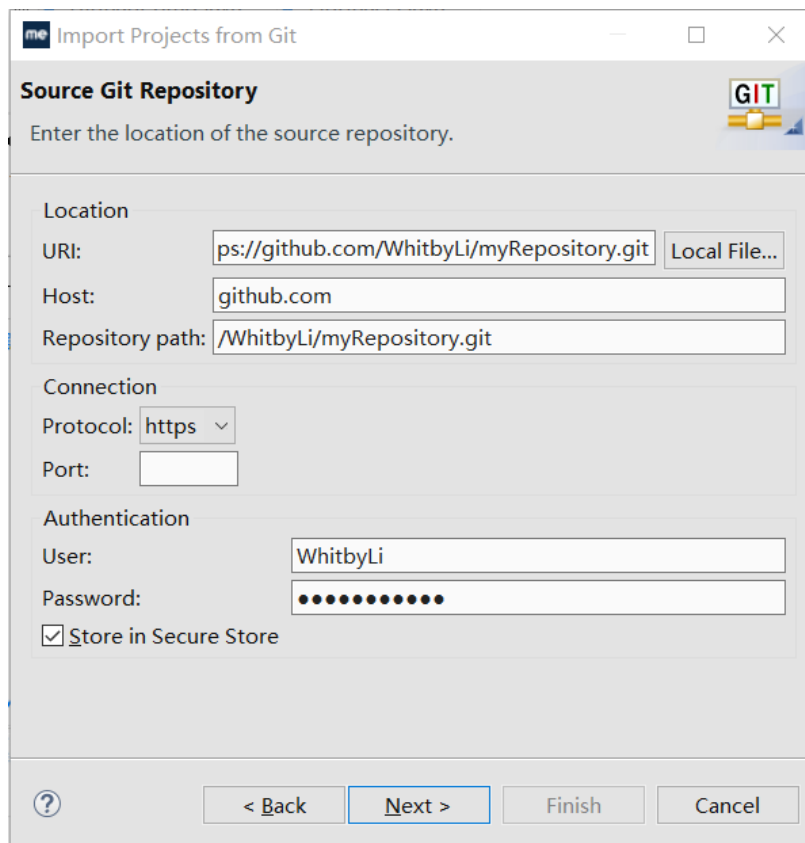
1、打开 MyEclipse 工具的文件（file）菜单列表，单击 import，选择 Projects from Git，单击 next，如下图所示：



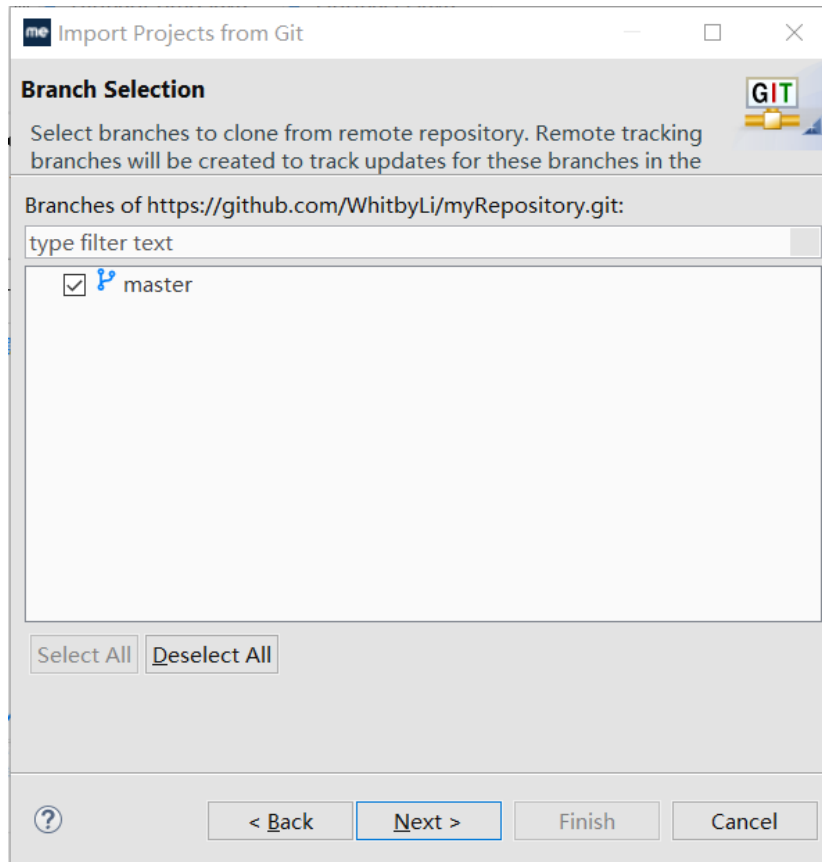
2、选择 Clone URI，单击 next，如下图所示：



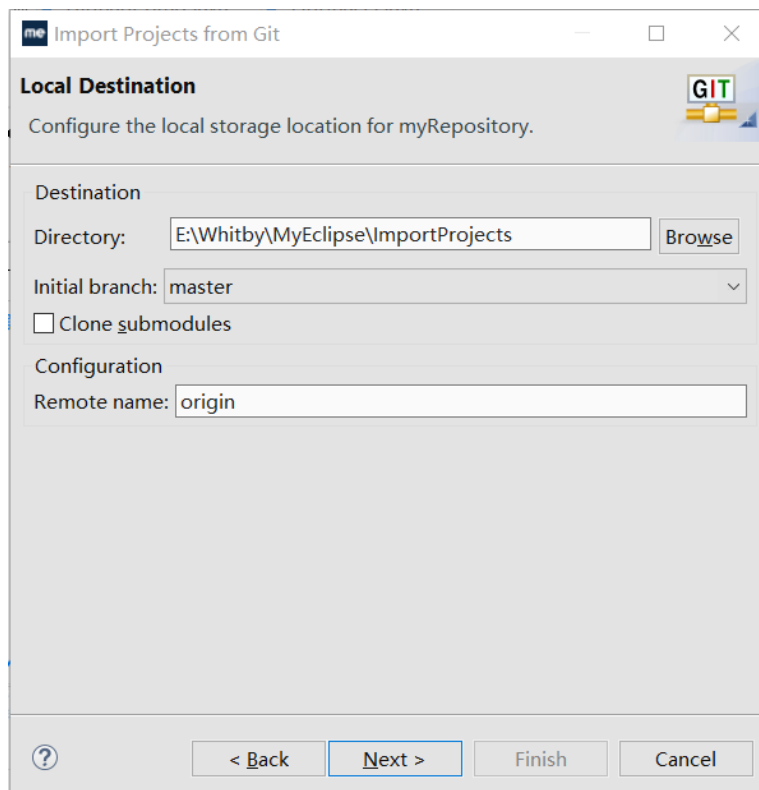
3、填写 GitHub 代码仓库 URI 及登录凭证，单击 next，如下图所示：



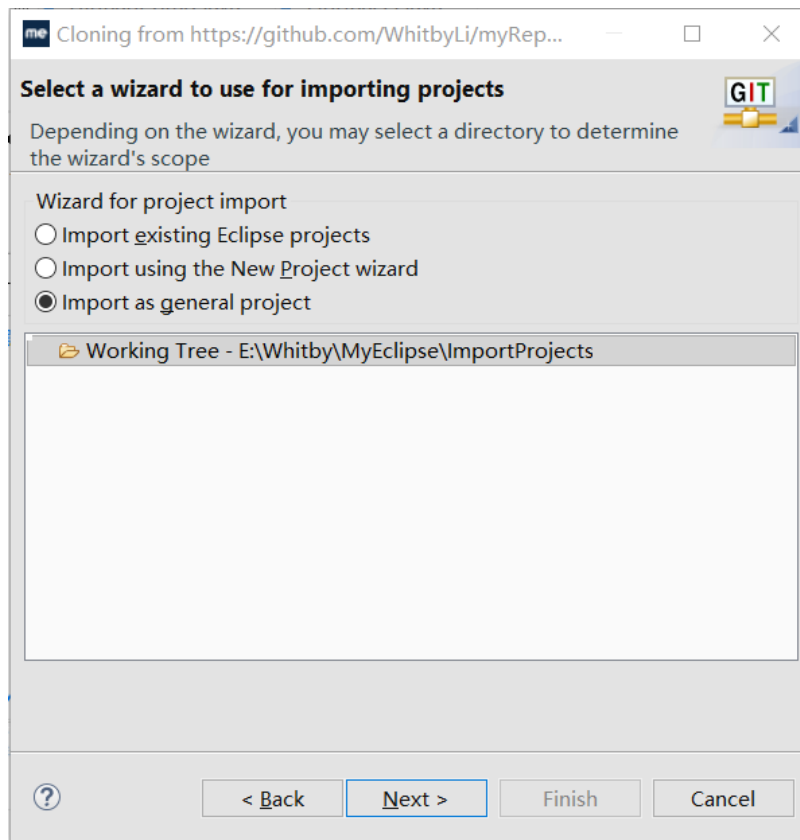
4、选择，分支，此处仅有一个分支（master），单击 next，如下图所示：



5、选择 git 项目在本地的存放信息，如下图所示：



6、点击下一步，选择项目导入方式为：Import as general project，如下图所示：



7、单击 next 进入最终页面，单击 finish 即可完成。导入成功界面如下：

