

myeclipse 中使用,包括解决冲突

Git

1. Git 简介

1.1 git 是什么

1.1.1 概念

Git:git 是一款**开源的分布式**的版本控制软件

Github:是一个基于 **git** 的面向开源及私有软件项目的托管平台

因仅支持 **git** 作为唯一的版本库格式进行托管 故名 **github**

1.1.2.Git 的特点

- ①**Git** 从服务器上克隆完整的项目到本机,相当于每一个开发者都拥有一个项目的完整版本
- ②开发者在自己的机器上创建分支,修改代码.
- ③将自己本地创建的分支提交到本地的版本库
- ④在单机上合并分支
- ⑤新建一个分支,把服务器上的最新版的代码 **fetch** 下来,然后跟自己的主分支合并
- ⑥**Git** 最大的亮点在于分支的管理.

1.2 什么是版本控制

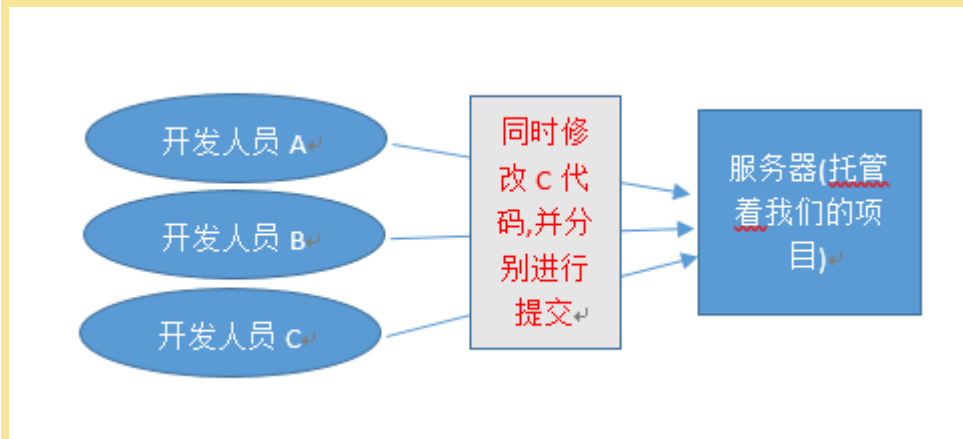
版本控制概念:

这种方法是工程图 (**engineering drawings**) 维护 (**maintenance**) 的标准做法,它伴随着工程图从图的诞生一直到图的定型。一种简单的版本控制形式,例如,赋给图的初版一个版本等级“**A**”。当做了第一次改变后,版本等级改为“**B**”,以此类推等等.

1.2.1 未引入版本控制的问题:

现实开发中最麻烦的是多人开发中的版本控制,如果未引入版本控制的概念,我们服务器上仅存在一个我们从最初开始开发的项目,我们每一次的增删改也是在这个项目之上,所以如果某一个开发者提交了带有 **bug** 的代码,或者对这个已经存在的项目进行更新操作,如果更新失败,则这个项目就废弃了

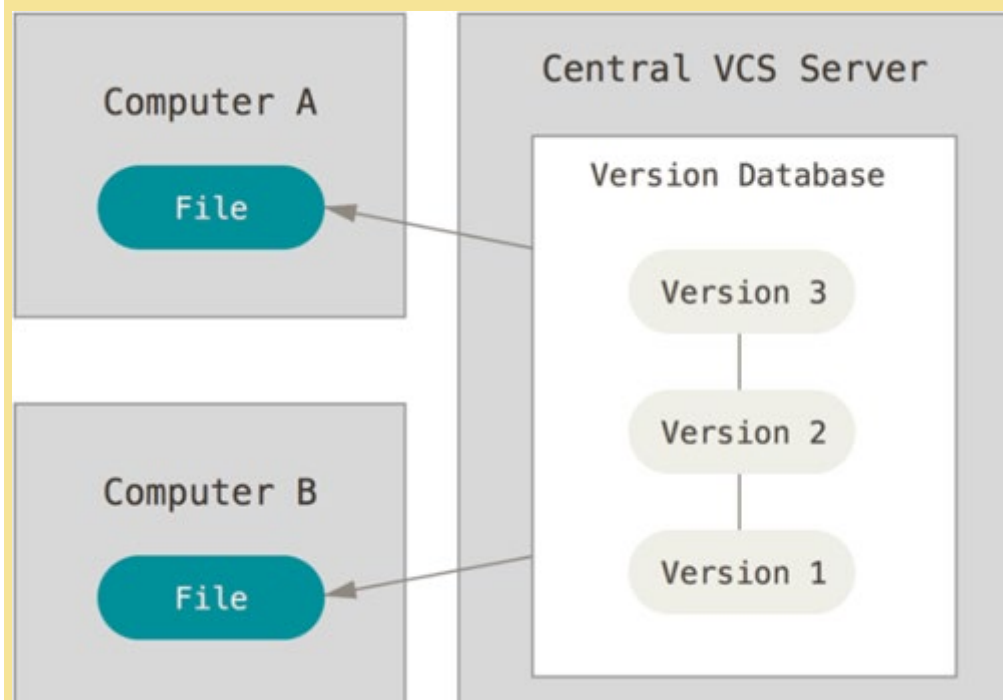
一个项目如果有多个人开发,开发人员 A,B,C,分别对项目中的同一代码进行了修改,那后一次提交的人的代码,就会覆盖前一个人的代码



1.2.2 传统的集中式版本控制

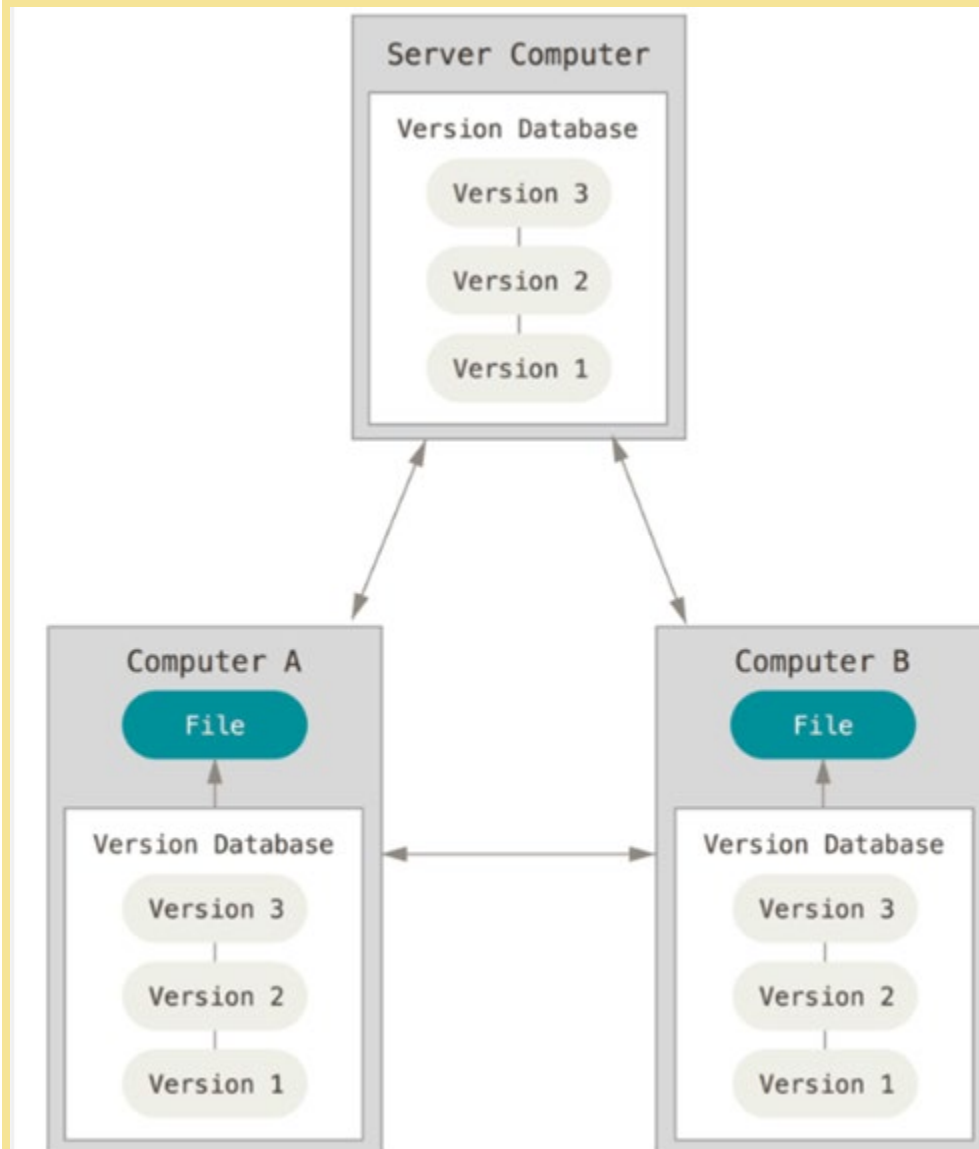
集中式版本控制系统 (Centralized Version Control Systems, 简称 CVCS), 版本库是集中存放在中央服务器的, 而干活的时候, 用的都是自己的电脑, 所以要先从中央服务器取得最新的版本, 然后开始干活, 干完活了, 再把自己的活推送给中央服务器

这么做最显而易见的缺点是中央服务器的单点故障。如果维修一小时, 那么在这一小时内, 谁都无法提交更新, 也就无法协同工作。如果中心数据库所在的磁盘发生损坏, 又没有做恰当备份, 毫无疑问你将丢失所有数据——包括项目的整个变更历史。



1.2.3 使用分布式版本控制系统

分布式,当我们连接共享版本库时,可以先将服务器上的项目,克隆到本地,相当于每一台电脑上都有整个项目的文件备份,在没有网时也可以开发,完成开发后,可以先提交到本地仓库,当有网的时候,再提交到共享版本库,这样一来,如果我们的服务器或者我们自己的电脑出故障,我们也没有任何担心



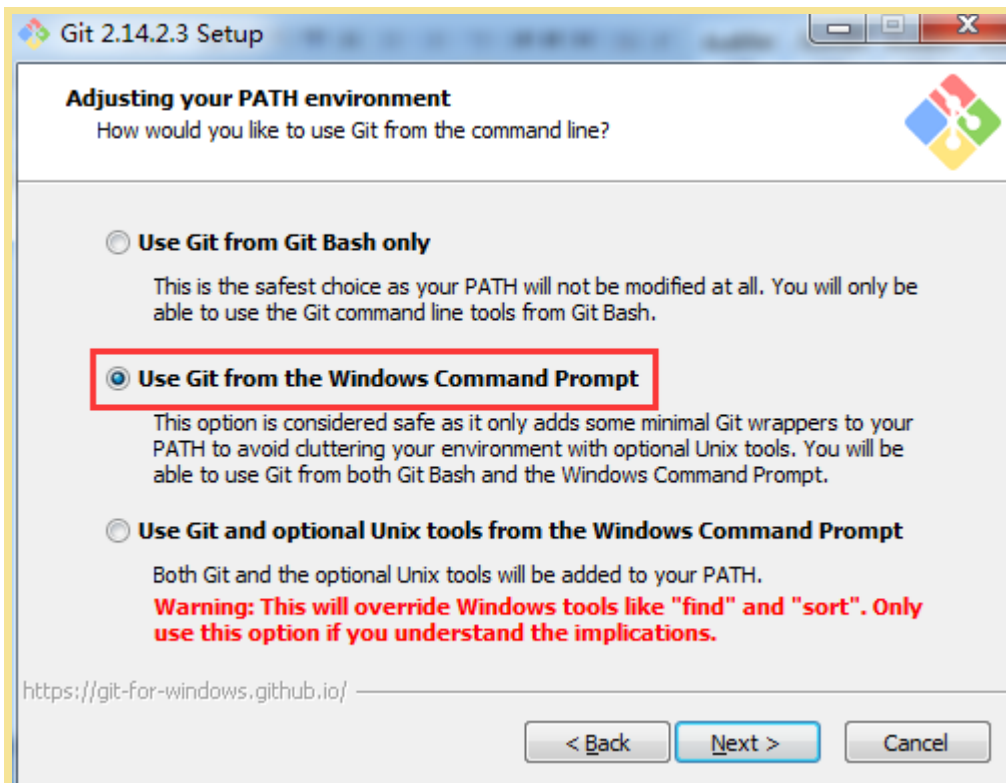
1.3Git 的安装

①下载软件 <https://git-scm.com/>官网地址

②进行安装

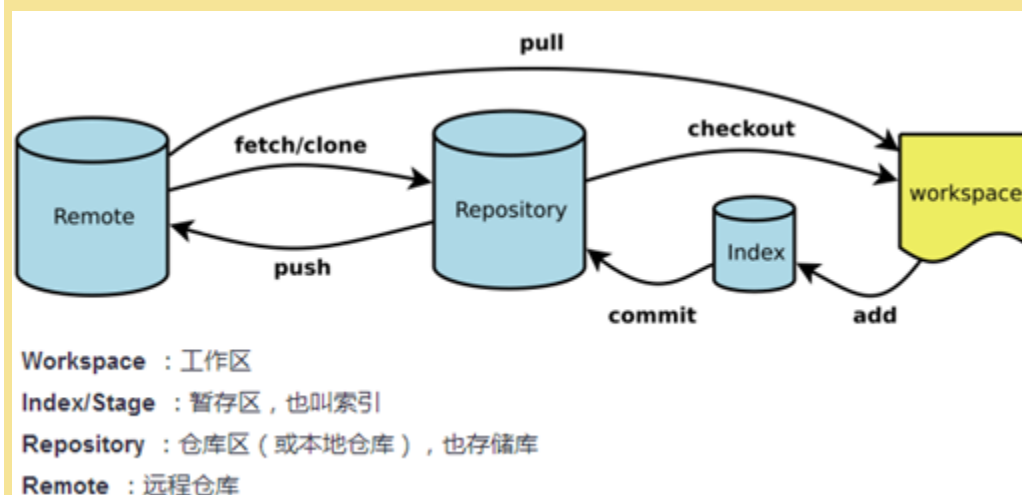
最重要的一步,其它可以直接走默认

选择这一步可以直接将我们的 `git` 命令,添加到系统变量中



2. git 入门

工作区,暂存区,主分支的概念



获取帮助

git help

如果向对某个具体的命令获取帮助,可以使用 `git help <verb>`

2.1 设置开发者的个人信息

多人开发的项目中,通过设置的用户名来区分开发者,设置 **email** 来联系开发者

2.1.1 配置当前项目中的用户信息(局部)

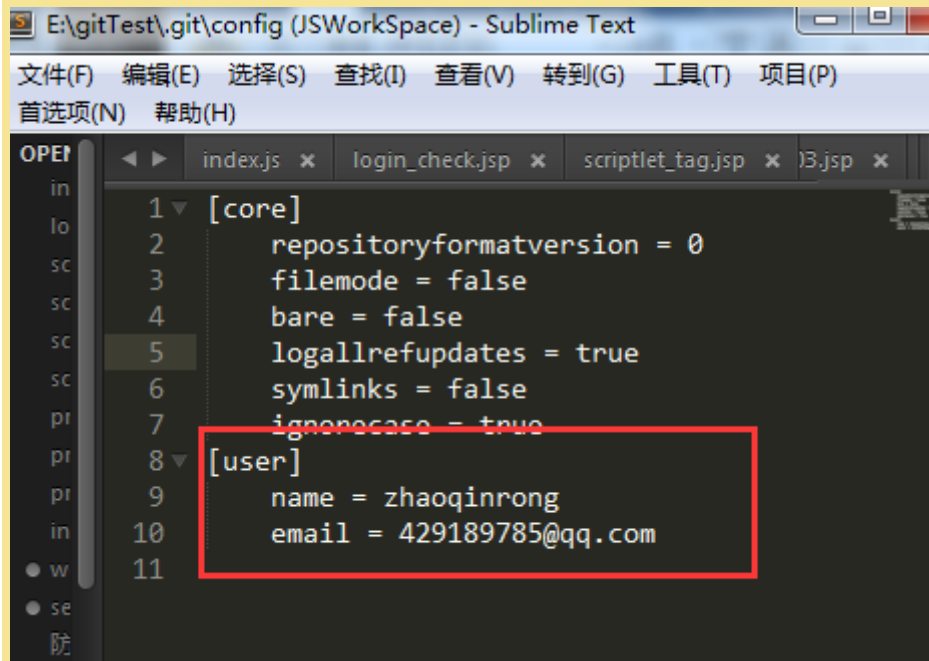
先创建一个本地版本库

在配置当前项目的用户信息时,需要进入 **git** 管理的项目中来进行设置

- 1.配置用户名:Git config user.name “用户名” 用来区分谁开发的代码
- 2.配置邮箱信息:git config user.email “邮箱” 可能是多个国家的开发者,便于联系开发者

```
Administrator@WIN-VT46KOU0CNI MINGW64 /e/gitTest (master)
$ git config user.name "zhaoqinrong"
Administrator@WIN-VT46KOU0CNI MINGW64 /e/gitTest (master)
$ git config user.email "429189785@qq.com"
Administrator@WIN-VT46KOU0CNI MINGW64 /e/gitTest (master)
$
```

配置以后会可以在隐藏的.git文件夹的 config 文件中查看到,
也可以使用 git config --list 进行查看



```
E:\gitTest\.git\config (JSWorkspace) - Sublime Text
文件(F) 编辑(E) 选择(S) 查找(I) 查看(V) 转到(G) 工具(T) 项目(P)
首选项(N) 帮助(H)

index.js x login_check.jsp x scriptlet_tag.jsp x 13.jsp x
1 [core]
2 repositoryformatversion = 0
3 filemode = false
4 bare = false
5 logallrefupdates = true
6 symlinks = false
7 ignorecase = true
8 [user]
9 name = zhaoqinrong
10 email = 429189785@qq.com
11
```

2.2.2 配置全局用户信息

- 1.配置用户名:git - -global config user.name “用户名” 用来区分谁开发的代码
- 2.配置邮箱信息:git - -global config user.email “邮箱” 可能是多个国家的开发者,便于联系开发者

```
Administrator@WIN-VT46KOU0CNI MINGW64 /e/gitTest (master)
$ git config --global user.name "zhaoqinrong"
Administrator@WIN-VT46KOU0CNI MINGW64 /e/gitTest (master)
$ git config --global user.email "429189785@qq.com"
Administrator@WIN-VT46KOU0CNI MINGW64 /e/gitTest (master)
$ |
```

可以在 C:\Users\Administrator\.gitconfig 中查看

2.2 创建仓库

①创建文件夹 E:\Mygit

②初始化仓库:

进入我们创建的文件夹 `cd e:\Mygit` ,使用命令 `git init` 进行初始化仓库

```
C:\Users\Administrator>cd e:\mygit  
  
C:\Users\Administrator>git init  
Initialized empty Git repository in C:/Users/Administrator/.git/
```

这时候会在当前文件夹下创建一个.git 隐藏文件夹,.git 文件夹是我们的仓库信息,一定不要修改

③对仓库信息进行配置,主要是设置 `user.name` 和 `user.email`,如果设置了全局的用户信息,可以忽略

2.3 添加文件

在仓库中添加一个 `hello.java` 的文件

① 查询仓库的状态:`git status`

```
E:\Mygit>git status  
On branch master 当前开发的分支  
Initial commit 初始化仓库的提交  
Untracked files: 未标记的文件  
  (use "git add <file>..." to include in what will be committed) 给出一些操作的命令  
    Hello.java 未标记的文件列表  
  
nothing added to commit but untracked files present (use "git add" to track)  
E:\Mygit>
```

② 将文件加入到暂存区:`git add` 文件名

```
E:\Mygit>git add hello.java
```

使用 `git status` 再次查询出文件的状态

```

E:\Mygit>git status
On branch master

Initial commit

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

    new file:   Hello.java

E:\Mygit>

```

现在的文件并没有真正的提交到主分支上(主分支就是我们真正要运行的程序的所有代码).

③ 提交到本地版本库:**git commit -m**"注释信息"

git commit -m"创建一个 java 文件"

此时文件就被提交到了主分支上

查询指定文件的日志记录:**git log** 文件名 如果出现 end 使用 q 退出

```

E:\Mygit>git log Hello.java
commit 74d914fdda87ba20f52b907413fb7eb3a29973de <HEAD -> master
Author: zhaoqinrong <30885228+zhaoqinrong@users.noreply.github.com>
Date:   Fri Oct 20 15:42:07 2017 +0800

    <E5><88> <E5><BB><BA><E4><B8><80><E4><B8><AA>java<E6><96><87><E4><BB><B6>

```

74d914fdda87ba20f52b907413fb7eb3a29973de 是我们的版本号

总结:

每一次修改或者创建新文件时,都需要先使用 **git add** 文件名 的命令来将文件添加到缓存区,再使用 **git commit -m** "注释信息" 来将我们的文件添加到版本库

或者使用简化命令 **git commit-a -m**"注释信息" 来将我们的文件添加到版本库

2.4 修改文件

Git 如何管理修改的文件

① 将我们之前创建的 Hello.java 文件进行修改

查询当前仓库状态 **git status**

```
E:\Mygit>git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   Hello.java

no changes added to commit (use "git add" and/or "git commit -a")
E:\Mygit>
```

以上看到 git 的一个建议

(use "git add <file>..." to update what will be committed)

(use "git checkout -- <file>..." to discard changes in working directory)

我们可以选择 **git add** 进行添加到缓存,或者我们可以使用 **git checkout** 进行恢复

使用 **git diff HEAD Hello.java** 可以查看我们对文件具体做了哪些修改

```
E:\Mygit>git diff
diff --git a/Hello.java b/Hello.java
index 7833ed7..e29383a 100644
--- a/Hello.java
+++ b/Hello.java
@@ -1,5 +1,6 @@
 public class word{
     public static void main(String[] args){
-        System.out.println("helloWord");
+        System.out.println("modified")
     }
 }
\ No newline at end of file
```

② 使用 **git add** 文件名,将文件加入到缓存

③ 使用 **git commit -m"注释信息"** 提交到本地版本库

如果进行了 vim 编辑器,首先 Esc 退出输入状态,然后 Shift+;,再输入 q!或 wq!
(不保存改动, wq!是保存文件的写入修改)退出

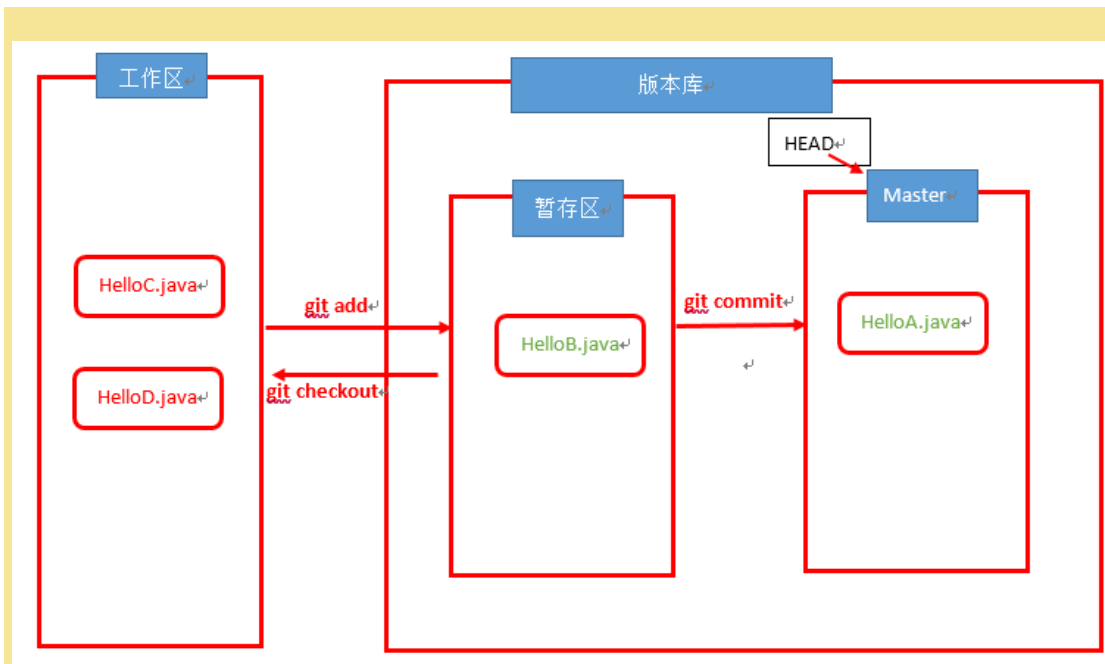
2.5.Git 工作区与缓存区

工作区:包含 .git 文件夹的文件夹(除了 .git 文件夹)

仓库:也称为版本库,这个不算工作区

暂存区:.git 版本库中有很多东西,包括重要要的称为 **stage** 的暂存区,还有 **git** 为用户

自动创建的主程序分支 **master**,以及指向 **master** 的 **HEAD** 指针,HEAD 指针永远指向我们当前项目的最高版本



这个就像我们之前学习的 `mysql` 数据库中的事务一样,使用 `insert` 增加了,但是没有使用 `commit` 是不会成功的

当使用 `commit` 后,会清空暂存区的内容

2.6 版本回退

我们每次提交代码后,都会生成一个 40 位的哈希值, 可以成为 `commit id`

使用 `git log` 来查看我们的提交记录

可以使用 `git show +版本号` 来查看修改 此版本号的修改内容

2.6.1 恢复到上一版本

使用 `git reset --hard HEAD~1` 或使用 `git reset --hard HEAD^`

2.6.2 恢复到指定版本

①查询当前的日志信息 `git reflog`

`Git log` 和 `git reflog` 的区别:

`git log` 只会查询历史有效版本的版本号

`git reflog` 会查询历史所有版本的版本号,包括已经废弃的

②找到想要恢复的版本号 使用 `git reset --hard 版本号`,进行恢复到指定版本

2.7 撤销修改

2.7.1 未执行 `git add` 添加到缓存区时

可以使用 `git checkout -- Hello.java` 与本地仓库中进行对比,同步

切记用于恢复文件时,文件名前面有--

2.7.2 已经执行 `git add` 添加到了暂存区,但是没有执行 `git commit`

可以使用 `git reset HEAD Hello.java` 将暂存区的内容恢复到了工作区

如果文件有错误,想恢复到上一次提交,还可以直接使用 `git checkout -- Hello.java` 进行恢复

2.8 删除文件

①.从磁盘上删除文件

```
E:\Mygit>git status
On branch master
Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        deleted:    Hello.java

no changes added to commit (use "git add" and/or "git commit -a")
E:\Mygit>
```

还是可以使用 `git` 恢复的命令进行恢复

如果使用

删除文件,`git rm <filename>`

删除文件夹,`git rm -r <文件夹名>`

同时时候 `git commit` 时本地仓库中的文才会删除

3 Git 和 github

3.1 注册账号

3.2 生成 SSH Key

使用 `git bash` 生成

`Ssh-keygen -t rsa -c 429189785@qq.com`

会生成两个文件

`id_rsa` --私钥

`id_rsa.pub` ---公钥

在 `github` 的 `setting` 中选择 `SSH Keys` 进行添加

将公钥的内容,粘贴进去

3.3 添加远程仓库

①将本地已有仓库与远程仓库相关联

使用命令 `git remote add origin https://github.com/zhaqinrong/admin.git`

Git remote add origin+远程仓库地址

```
E:\Mygit>git remote add origin https://github.com/zhaqinrong/admin.git
```

但是目前为止,远程仓库中没有我们本地仓库的内容

②将本地所有内容,提交到远程仓库

使用命令 `git push -u origin master`

会提示输入 github 的用户名和密码

```
E:\Mygit>git push -u origin master
Username for 'https://github.com': zhaqinrong
Password for 'https://zhaqinrong@github.com':
Counting objects: 8, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (5/5), done.
Writing objects: 100% (8/8), 715 bytes | 0 bytes/s, done.
Total 8 (delta 0), reused 0 (delta 0)
To https://github.com/zhaqinrong/admin.git
 * [new branch]      master -> master
Branch master set up to track remote branch master from origin.
E:\Mygit>
```

由于是第一次推送,而且推送的为 master 分支,就会使用“-u”的参数将远程 master 与本地 master 进行关联

④ 修改本地文件后,提交到本地版本库 add 和 commit 命令

⑤ 推送到远程仓库 git push

3.3 克隆仓库

将远程版本库克隆到本地

① ,在 github 上建立仓库,并进行初始化

Owner: zhaoqinrong / Repository name: mygithub ✓

Great repository names are short and memorable. Need inspiration? How about **stunning-carr**

Description (optional)

☒ **Public**
Anyone can see this repository. You choose who can commit.

☐ **Private**
You choose who can see and commit to this repository.

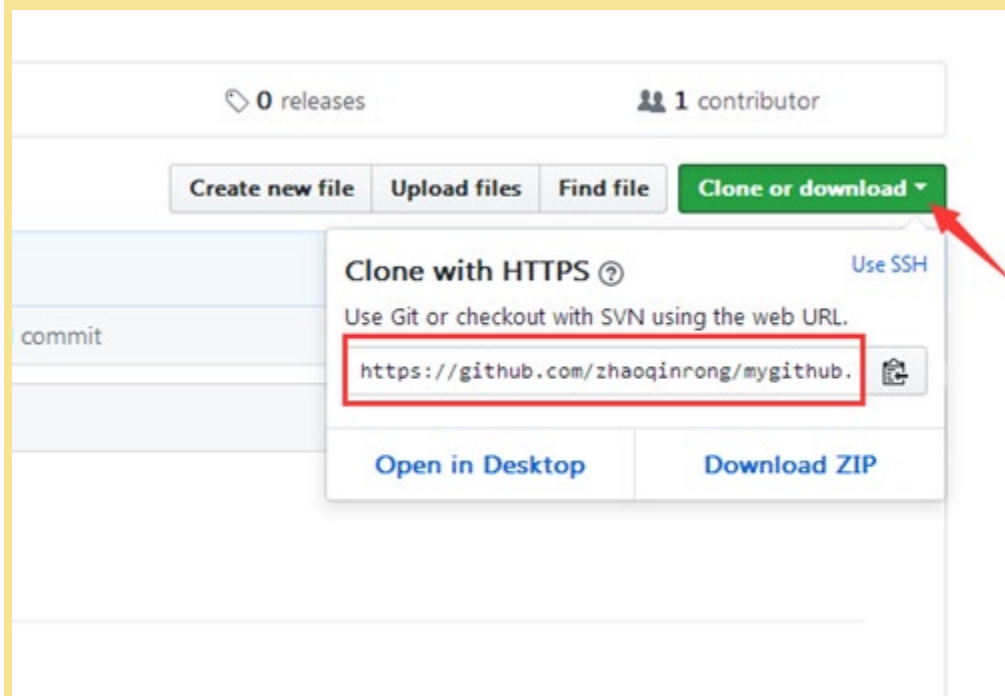
☒ **Initialize this repository with a README**
This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.

Add .gitignore: None Add a license: None ⓘ

勾选 **Initialize this repository with a README** 初始化这个仓库

② 克隆远程仓库

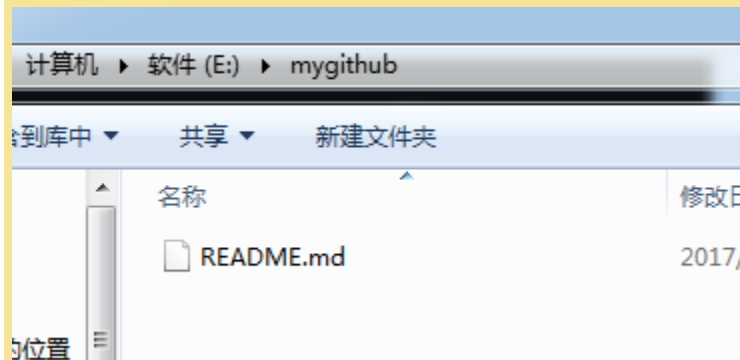
点击 **Clone or download**,将仓库的地址复制下来



使用 `git clone` + 远程仓库地址,对远程仓库进行克隆

```
e:\>git clone https://github.com/zhaqinrong/mygithub.git
Cloning into 'mygithub'...
remote: Counting objects: 3, done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), done.
e:\>
```

这样就克隆下来了



③ 在我们克隆的仓库中进行项目的开发,这里我们新建一个 `hello.java` 的文件并使用 `add` 和 `commit` 命令进行本地版本库的提交

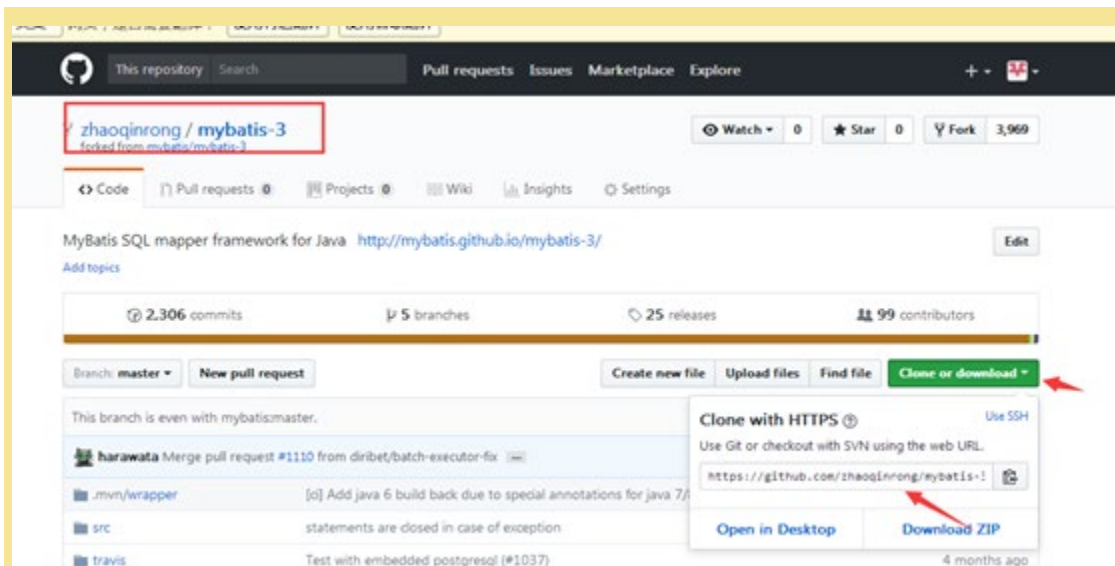
④ 使用 `git push-u origin master` 进行推送 Master 为分支名

3.4 克隆其他开源项目


① 找到需要克隆的开源项目的地址,然后 fork 到自己的远程仓库

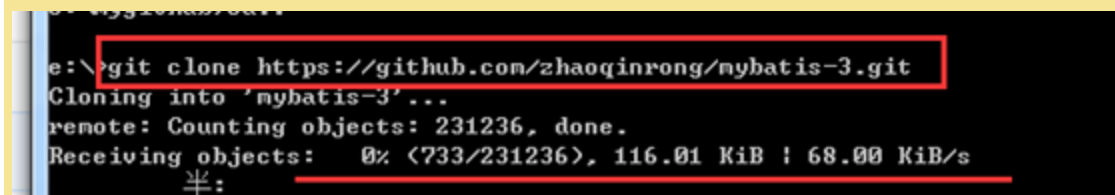


② 复制自己仓库中这个项目的地址



③ 使用 命令

Git clone + 远程仓库  地址 进行克隆



3.5 如何从远程仓库中获取更新

① 远程仓库与本地仓库进行关联

要向从远程仓库中获取更新,必须先将本地仓库与远程仓库进行关联,可以使用

git remote add origin <https://github.com/zhaqinrong/Mygit>

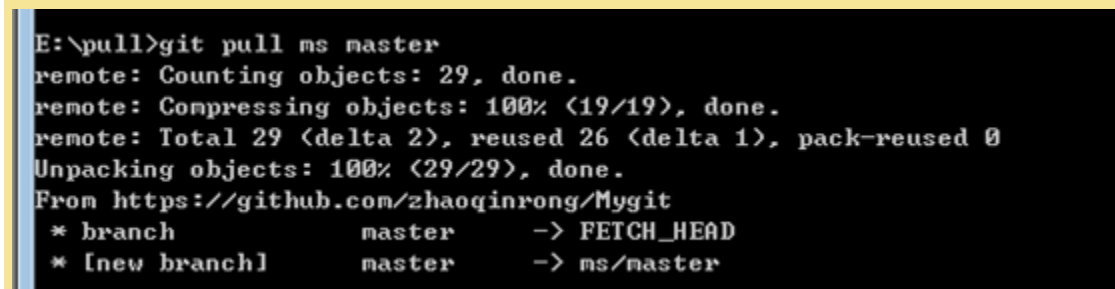
可以使用命令 **git remote rm origin** 删除关联的远程仓库

使用 **git remote show origin** 来查看 **ms** 远程仓库的具体分支

使用 **git remote -v** 来查看与当前本地仓库相关联的远程仓库

origin 为我们为远程仓库起的别名

③ 使用 **git pull origin master** 进行获取并合并到本地仓库



如果直接使用 **git pull** 可能会出错,建议先 **git fetch** 到本地,然后使用 **git merge** 合并

3.6 如何向远程仓库推送更新

我们编写项目后,需要将我们的项目推送到远程仓库

可以使用 **git push <url> <本地分支名>**

① 远程仓库与本地仓库进行关联

② 推送更新

```
E:\pull>git push ms master
Username for 'https://github.com': zhaoqinrong
Password for 'https://zhaoqinrong@github.com':
Counting objects: 3, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 271 bytes | 0 bytes/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To https://github.com/zhaoqinrong/Mygit
a4a36be..9f56dc8 master -> master
```

建议我们推送更新前都先使用 **pull** 获取远程仓库中的更新,然后在 **push** 推送

Ms 是我们远程仓库的一个别名,我们自定义

可以使用 **git push ms master:mygit-1**


将本地的 **master** 分支推送到 **ms** 远程仓库的 **mygit-1** 分支中

```
E:\pull>git push ms master:mygit-1
Username for 'https://github.com': zhaoqinrong
Password for 'https://zhaoqinrong@github.com':
Total 0 (delta 0), reused 0 (delta 0)
To https://github.com/zhaoqinrong/Mygit
9360bc2..9f56dc8 master -> mygit-1
E:\pull>
```

3.7 删除远程仓库中的文件

① 先要将本地仓库和远程仓库进行关联

```
E:\Mygit\Mygit>git remote add ms https://github.com/zhaoqinrong/Mygit.git
```

进入到我们的 **git** 命令  行页面后,先将远程代码 **pull** 到本地,保持本地仓库跟远端仓库同步。

```
E:\dasdas>git pull ms master
From https://github.com/Zhaoqinrong/Mygit
* branch          master    -> FETCH_HEAD
Already up-to-date.
```

④ 使用 **git rm** 文件名 删除 文件

```
E:\dasdas>git rm dasdas.txt
rm 'dasdas.txt'
```

⑤ ,进行提交(和数据库事务一样,提交后才会处理)

```
E:\dasdas>git commit -m"for test"
[master a4a36be] for test
1 file changed, 1 deletion(-)
delete mode 100644 dasdas.txt
```

⑥ 向远程仓库进行推送



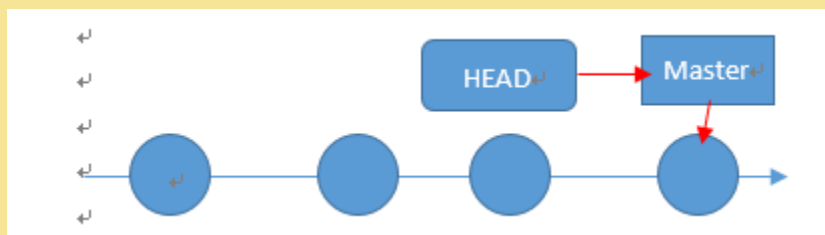
4 分支管理

Master 主分支,主要作为程序的发布

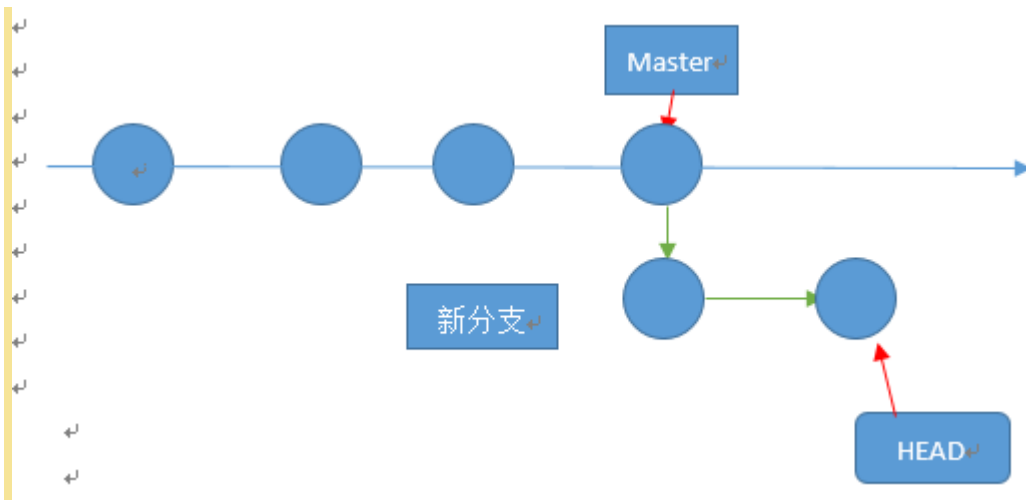
所以不能在 **master** 上进行开发,所以应该建立子分支进行开发

4.1 HEAD 指针和 Master 指针

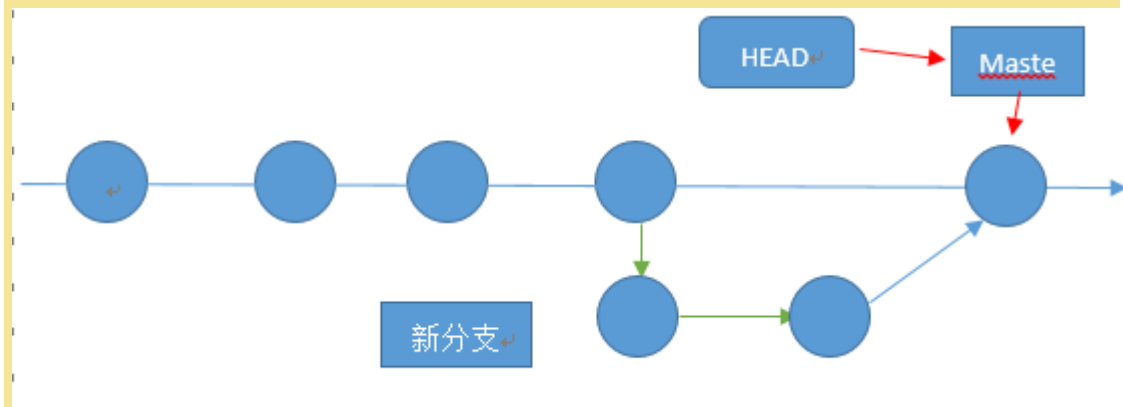
①在没有分支的情况下,**Master** 指针永远指向当前的最高版本
而 **head** 指针指向 **master**



② 创建了新分支,HEAD 指向了



③ 主分支和子分支进行合并



4.2 分支创建

① 创建分支:

使用命令 **git branch + 分支名**

使用 **git branch** 查看当前项目的分支

```
e:\mygithub> git branch
brh
* master
```

⑦ 切换分支 使用命令 **git checkout +分支名**

```
e:\mygithub> git checkout brh
Switched to branch 'brh'

e:\mygithub> _
半:
```

⑧ 使用命令 **git checkout +分支名** 切换到主分支,然后删除分支 使用命令 **git branch -d +分支名**

注意:要删除分支,必须要先切换到主分支

可以使用 **git checkout -b +分支名** 创建并切换分支

```
e:\mygithub>git checkout -b brh
Switched to a new branch 'brh'

e:\mygithub>
```

分支上的文件是相互独立的,修改文件不会影响

模拟:建立两个分支,并分别进行推送

先连接远程仓库 **git remote set-url origin <https://github.com/zhaqinrong/mygithub.git>**

然后分别推送分支 **git push origin master git push origin brh**

推送成功后可以在我们 github 上进行查看

4.3 合并分支

①切换回主分支 **git checkout master**

② 合并分支 **git merge brh**(子分支名)

```
e:\mygithub>git merge brh
Updating 107a76c..bab1cd3
Fast-forward
 hello.java | 1 +
 1 file changed, 1 insertion(+)
```

③,删除子分支 **git branch -d brh**

③ 推送到远程仓库 **git push origin master**

```
e:\mygithub>git push origin master
Username for 'https://github.com': zhaqinrong
Password for 'https://zhaqinrong@github.com':
Counting objects: 3, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 277 bytes | 0 bytes/s, done.
Total 3 (delta 0), reused 0 (delta 0)
```

现在在本地没有 brh 的子分支,但是远程仓库中的子分支还在

④删除远程子分支 **git push origin --delete brh**(子分支名称)

```
e:\mygithub>git push origin --delete brh
Username for 'https://github.com': zhaoqinrong
Password for 'https://zhaoqinrong@github.com':
To https://github.com/zhaoqinrong/mygithub.git
- [deleted]          brh

e:\mygithub>
```

4.4 删除分支

删除本地分支: `git branch -d <branch-name>`

删除远程分支: `git push origin -delete <branchName>`

4.4 解决代码冲突

`git fetch origin master`

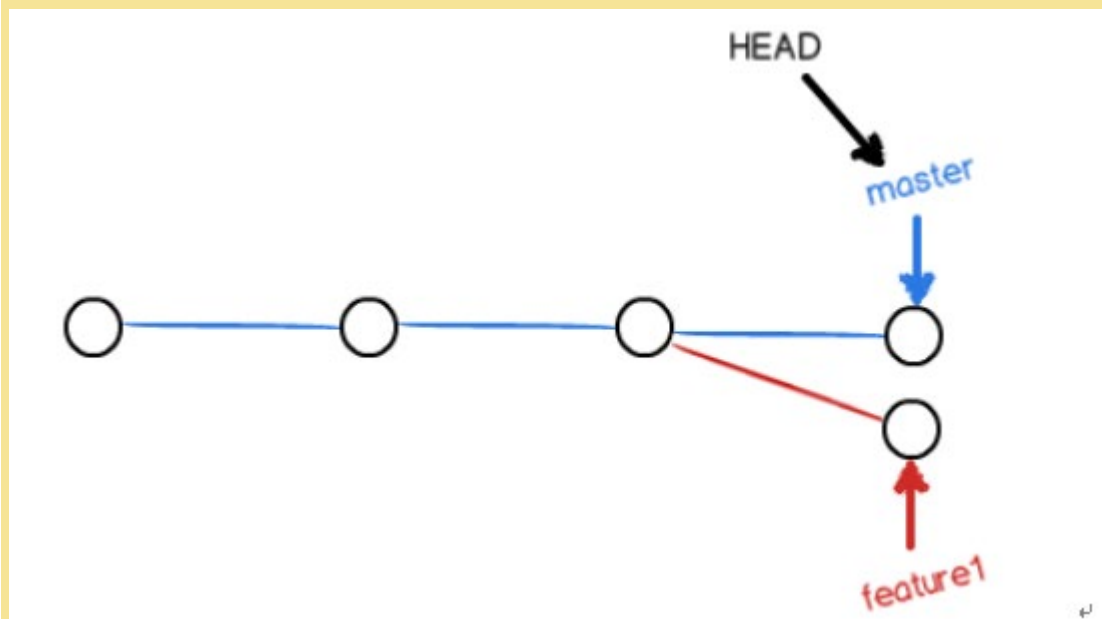
`git merge origin/master`

代码冲突如何产生的:

当我们在分支上进行开发的时候,难免遇到别的开发人员和我们自己向仓库中提交相同的代码

比如我们有一个商城的项目,开发人员 A 和开发人员 B 都对同一段代码做了修改,当 A 进行提交后并 `push` 到远程仓库中 `master` 合并,B 再进行提交并与远程仓库中的 `master` 进行合并

这时候就会出现代码冲突

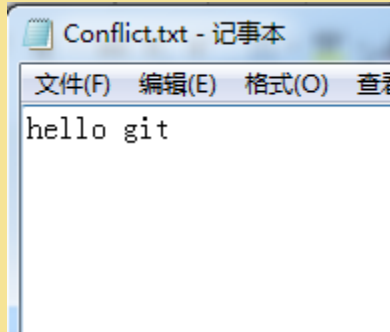


现在我们来模拟一下代码冲突,并试图解决

① 在 E 盘新建文件夹 `GitConflict`,并初始化仓库 `git init`

```
E:\GitConflict>git init
Initialized empty Git repository in E:/GitConflict/.git/
```

② 在工作区创建名为 Conflict.txt 的文件,并写上 hello git



③ 把文件提交到本地仓库

```
E:\GitConflict>git add .
E:\GitConflict>git status
On branch master

Initial commit

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

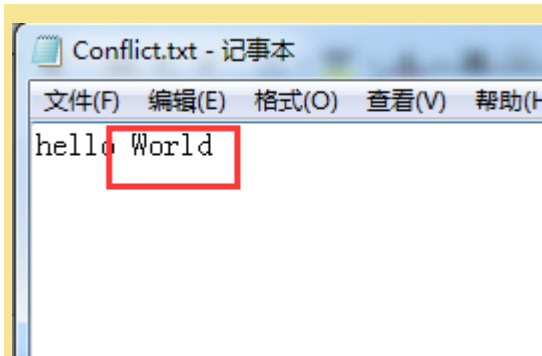
        new file:   Conflict.txt

E:\GitConflict>git commit -m"readme"
[master (root-commit) 631ebc7] readme
1 file changed, 1 insertion(+)
create mode 100644 Conflict.txt
```

④ 创建新分支,名为 Conflict 并切换

```
E:\GitConflict>git checkout -b "conflict"
Switched to a new branch 'conflict'
```

⑤ 在 conflict 分支下对 Conflict.txt 文件进行修改



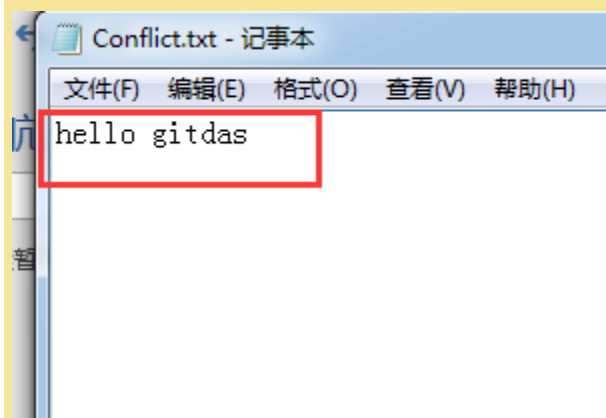
⑥ 然后在 conflict 分支下进行提交

```
E:\GitConflict>git add .  
E:\GitConflict>git commit -m"Myreadme"  
[conflict 3a5c178] myreadme  
1 file changed, 2 insertions(+), 1 deletion(-)  
E:\GitConflict>
```

⑦,切换到主分支

```
E:\GitConflict>git checkout master  
Switched to branch 'master'
```

⑧ 再次对文件进行修改



⑨ 提交修改后的文件

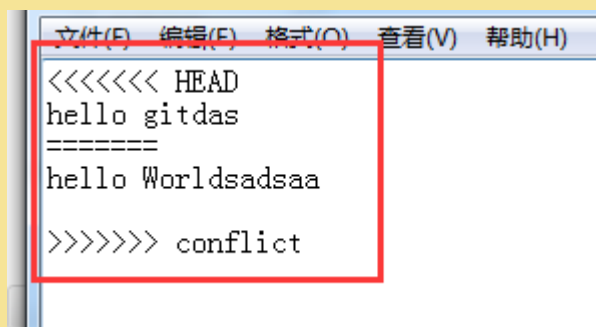
```
E:\GitConflict>git commit -a -m"hello git"  
[master 5948340] hello git  
1 file changed, 1 insertion(+), 1 deletion(-)  
E:\GitConflict>
```

⑩ 和分支 conflict 进行合并

```
1 file changed, 1 insertion(+), 1 deletion(-)
E:\GitConflict>git merge conflict
Auto-merging Conflict.txt
CONFLICT (content): Merge conflict in Conflict.txt
Automatic merge failed; fix conflicts and then commit the result.
E:\GitConflict>
```

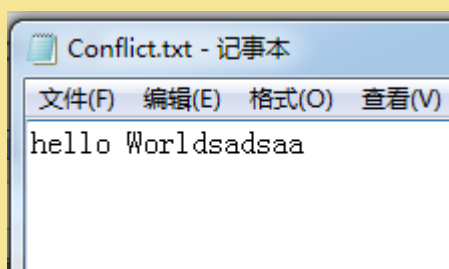
文件报错,有冲突

然后我们可以打开仓库中的 Conflict.txt 文件进行查看



```
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
<<<<<< HEAD
hello gitdas
=====
hello Worldsadsaa
>>>>>> conflict
```

中间内容表示有冲突的地方,我们可以选择保留一条信息,进行合并,假如我们保留一下数据



```
Conflict.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V)
hello Worldsadsaa
```

再次提交并合并

```
E:\GitConflict>git commit -a -m "已处理"
[master 0ff96a21] 已处理

Warning: Your console font probably doesn't support Unicode. If
r switching to a TrueType font such as Consolas!

E:\GitConflict>git merge conflict
Already up-to-date

E:\GitConflict>
```

OK 了

5.使用 tag

发布一个版本时，我们通常先在版本库中打一个标签（tag），这样，就唯一确定了打标签时刻的版本。将来无论什么时候，取某个标签的版本，就是把那个打标签的時刻的历史版本取出来。所以，标签也是版本库的一个快照。

打标签命令:`git tag <tagname>`

在以往历史中的提交 id 上打标签

使用:`git tag <tagname> <commitid>`

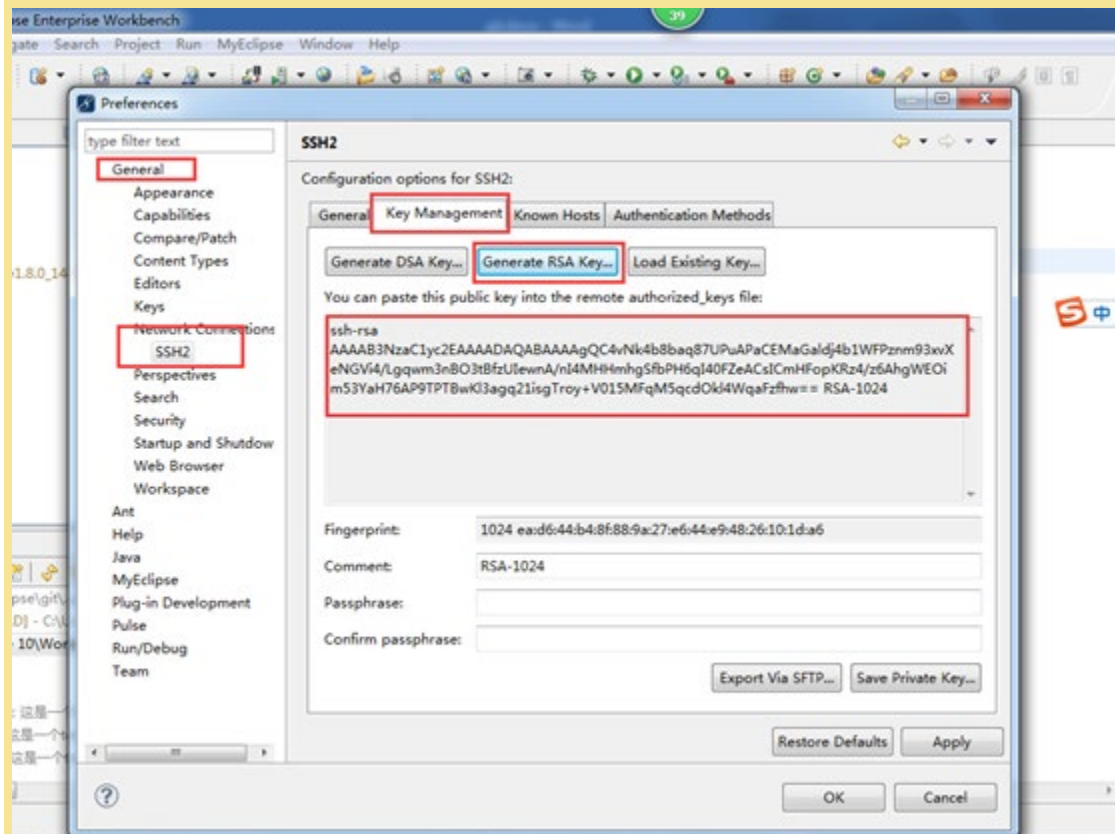
删除标签 `git tag -d<tagname>`

5 eclipse 中使用 git

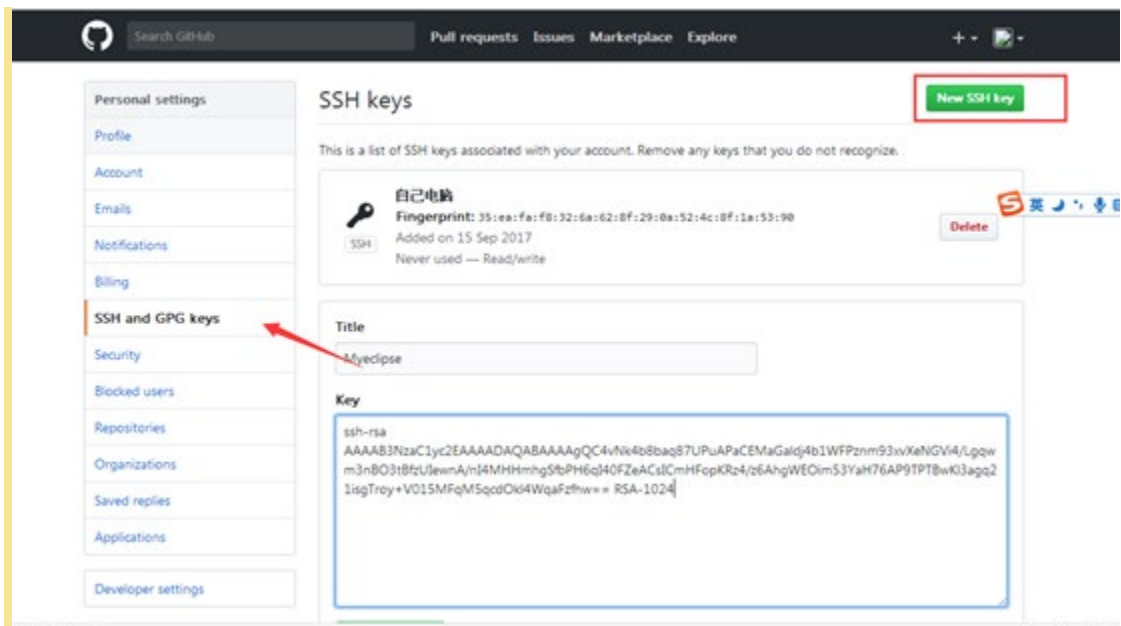
5.1 用 eclipse 将项目保存到 github 中

1 首先注册 github

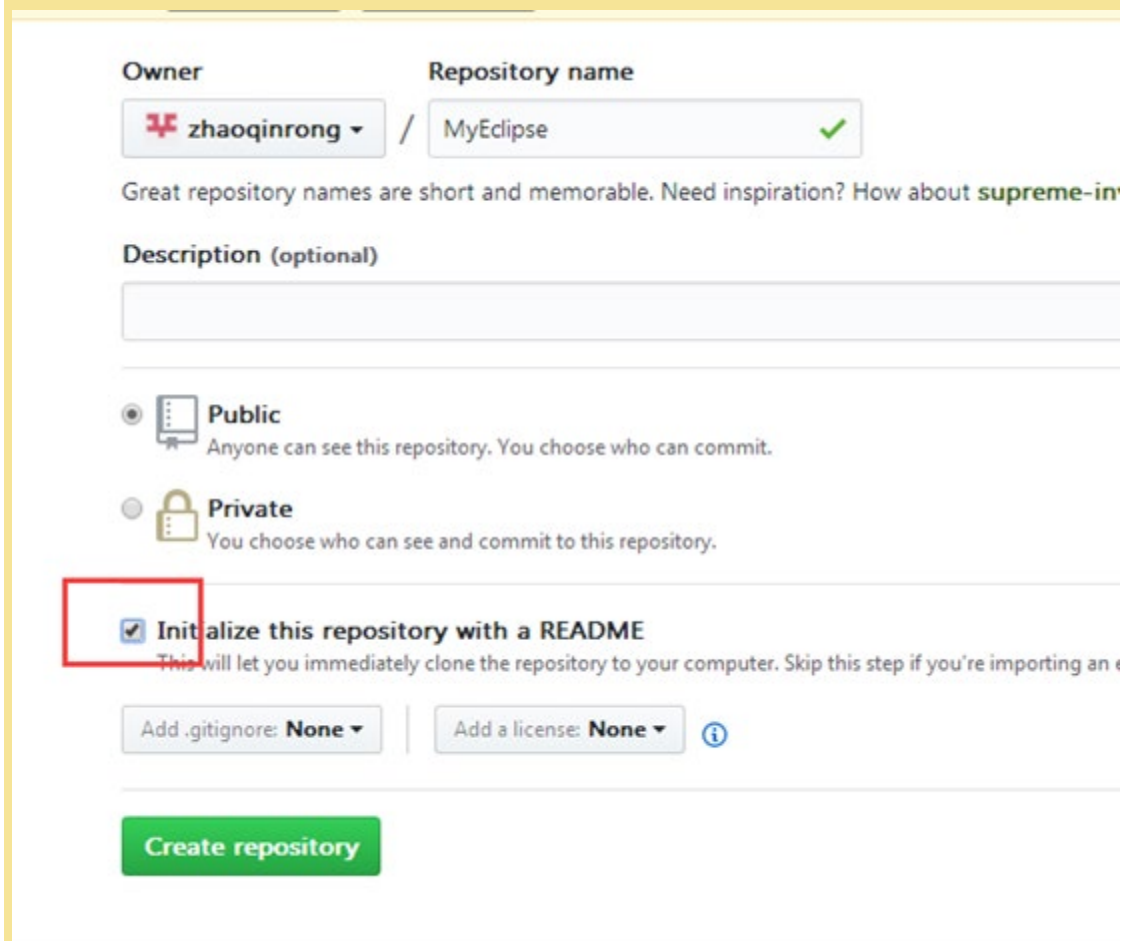
2.使用 eclipse 生成一个 SSHkey



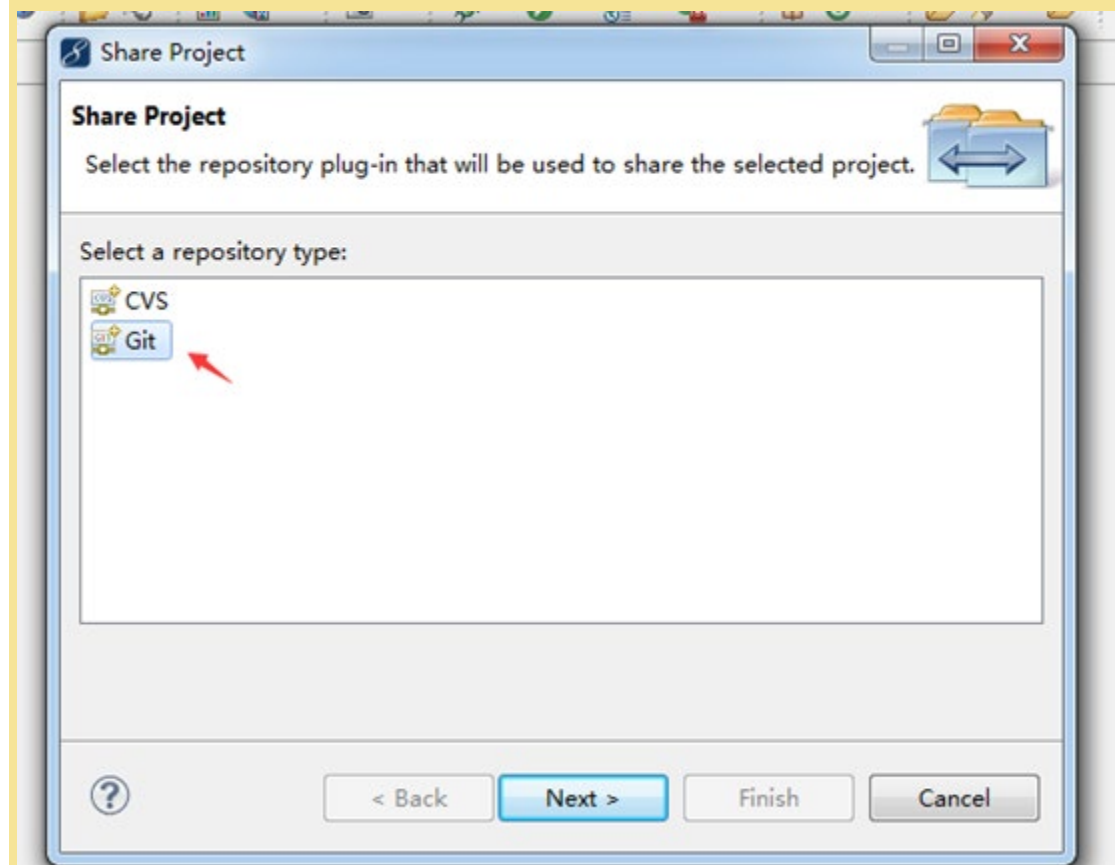
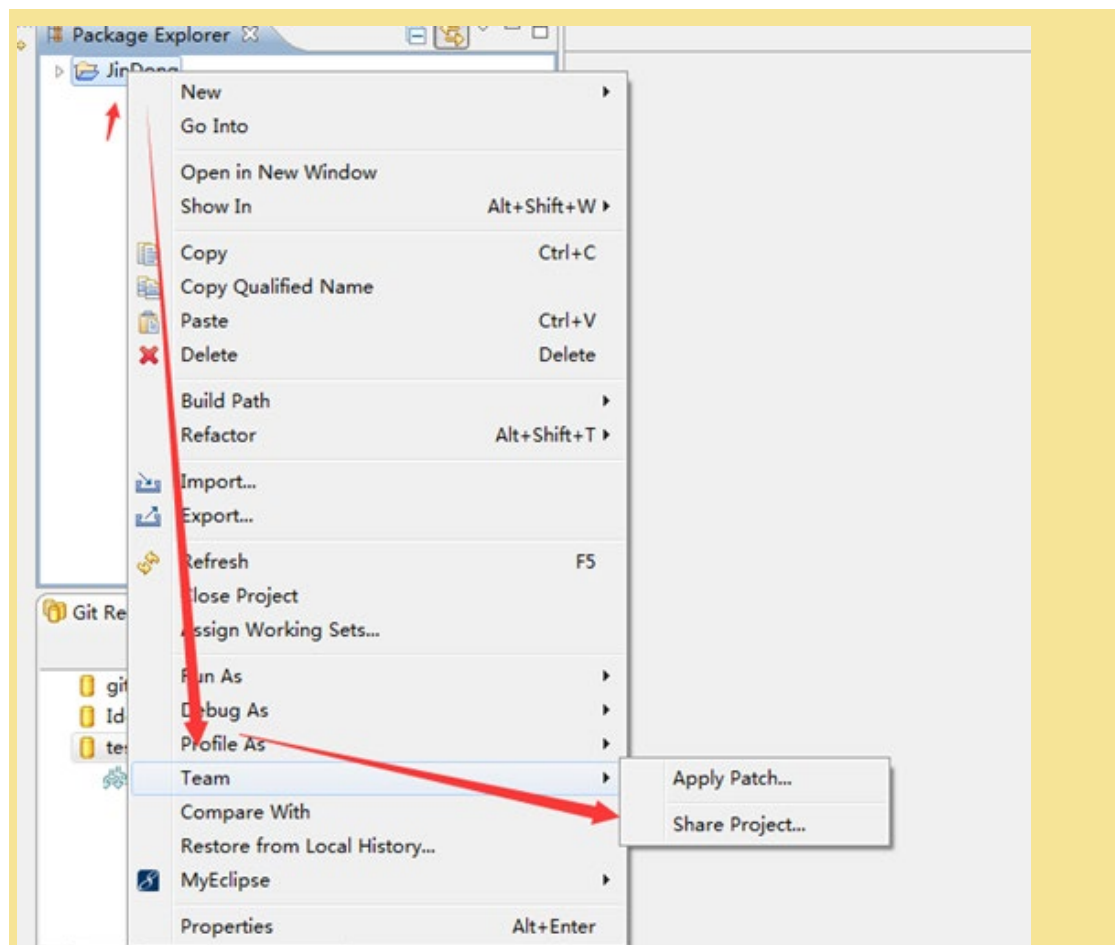
3.在 github 上对进行添加

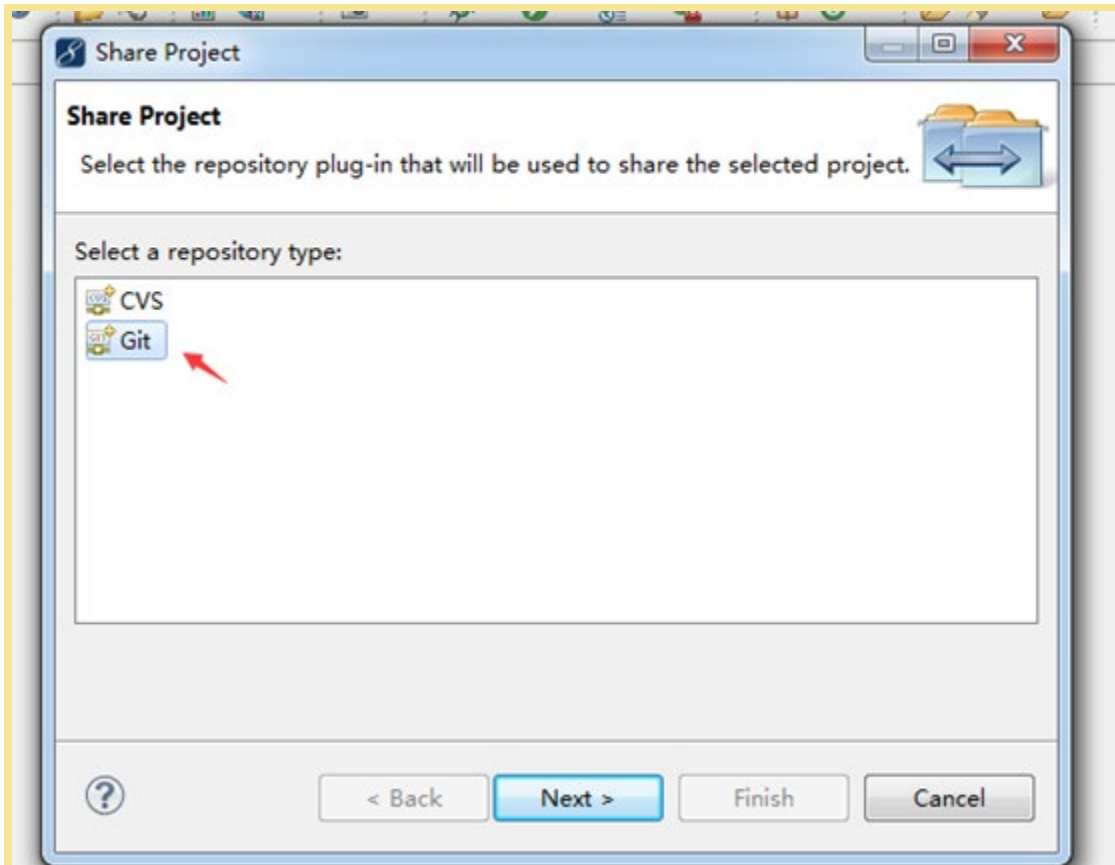


4.在 github 上创建一个新的仓库,并对仓库进行初始化

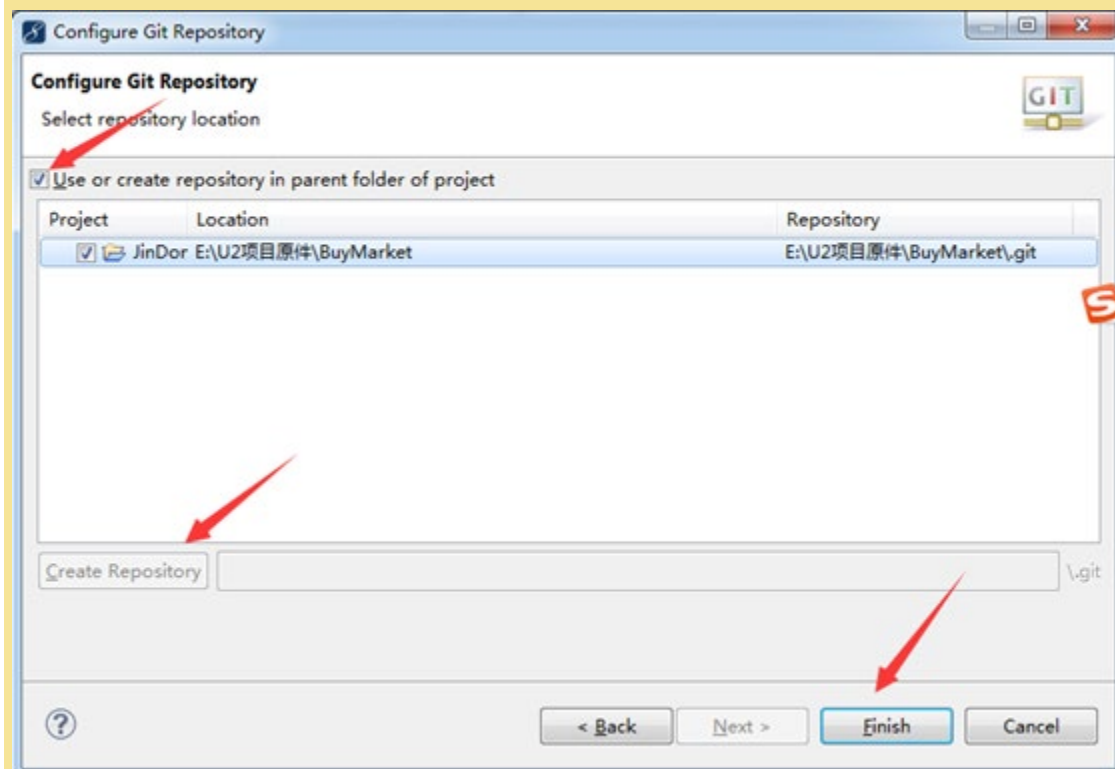


5.将我们的项目发布到刚刚创建的仓库中

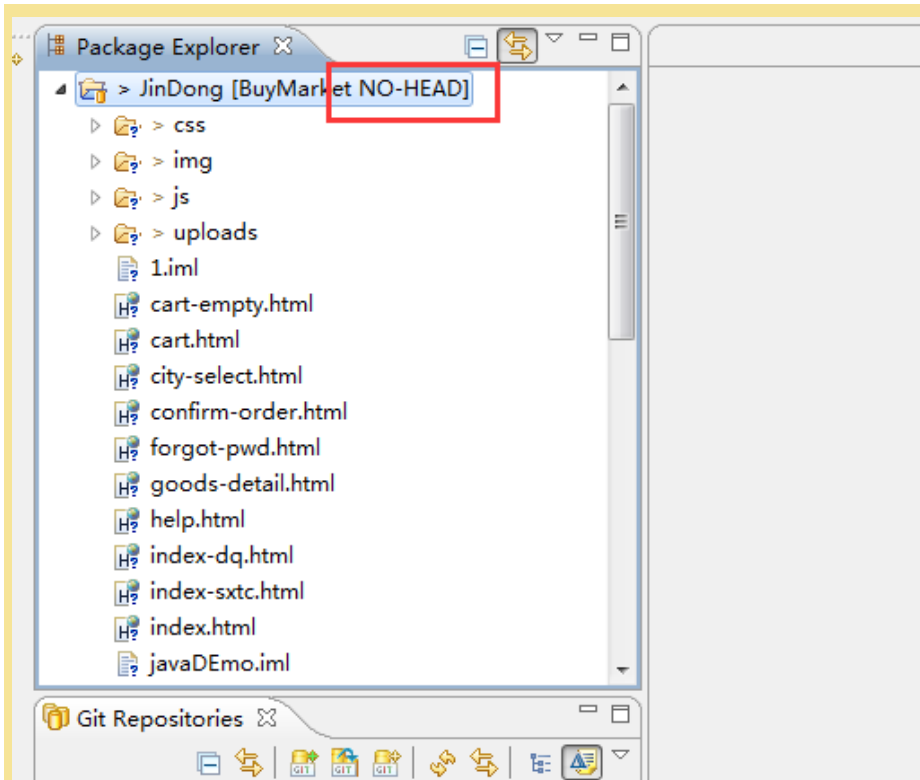




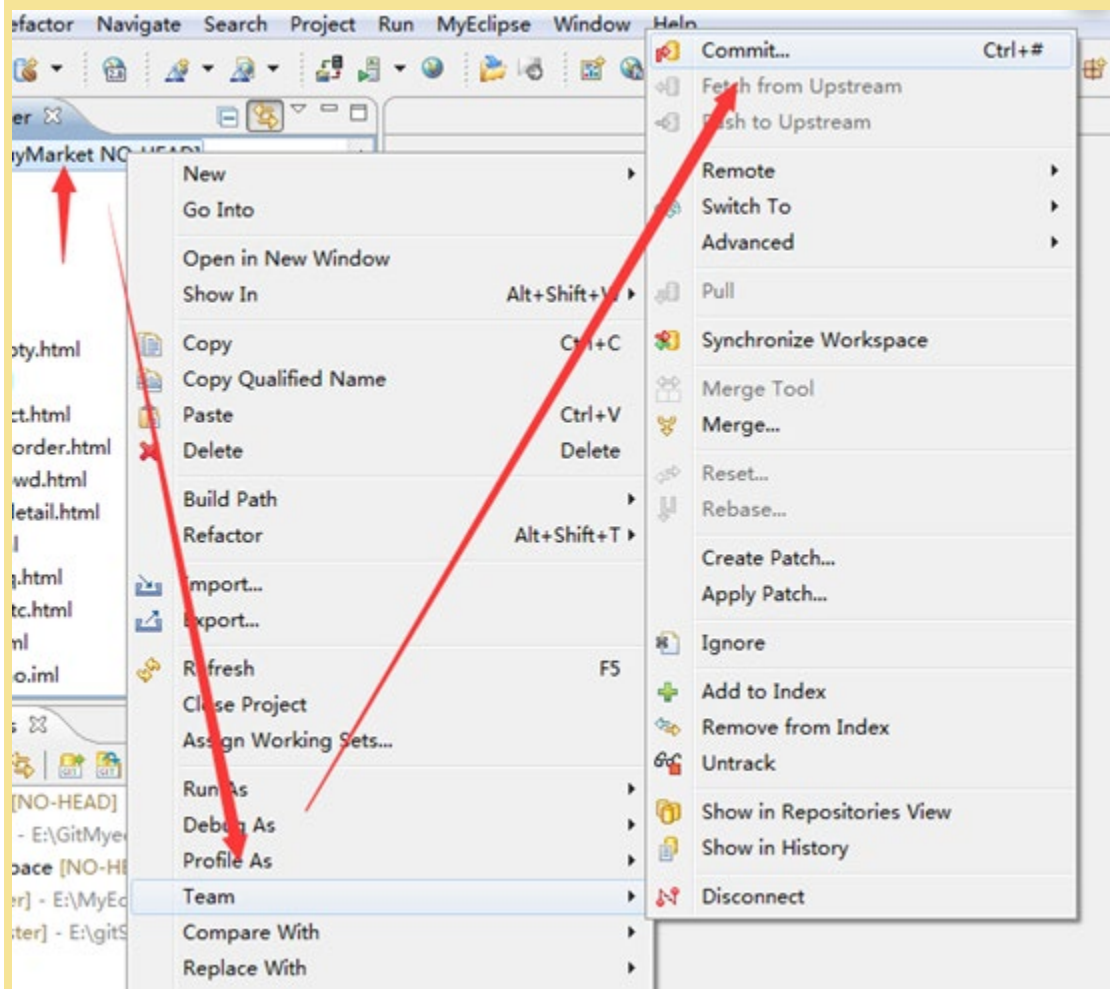
因为我们这里是新项目,所以选择创建一个 repository



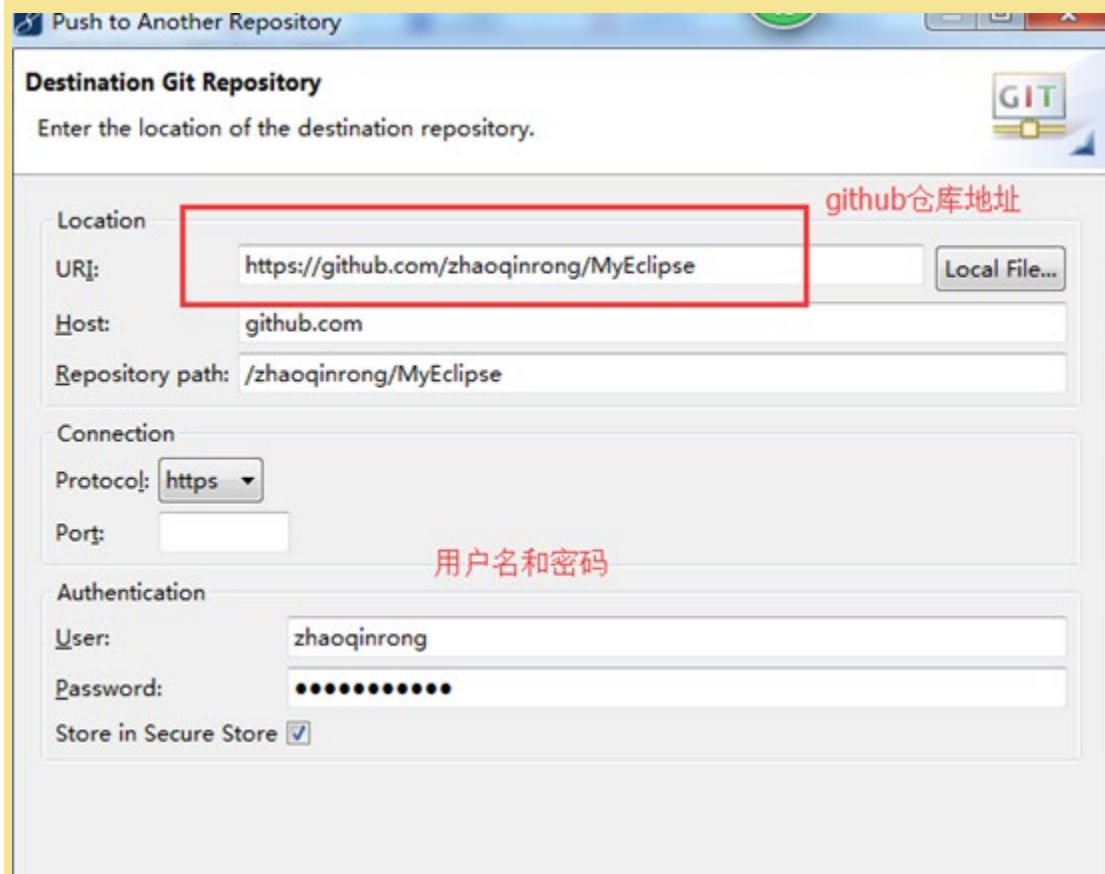
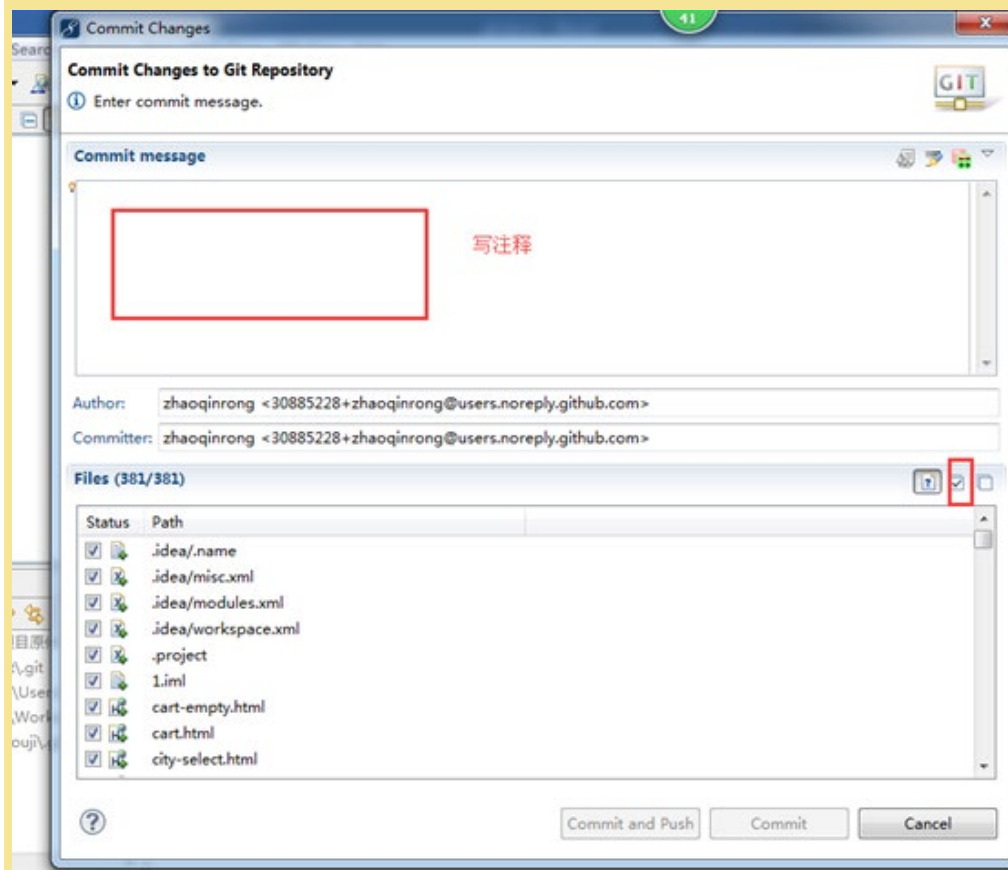
表示我们的项目还没有指针

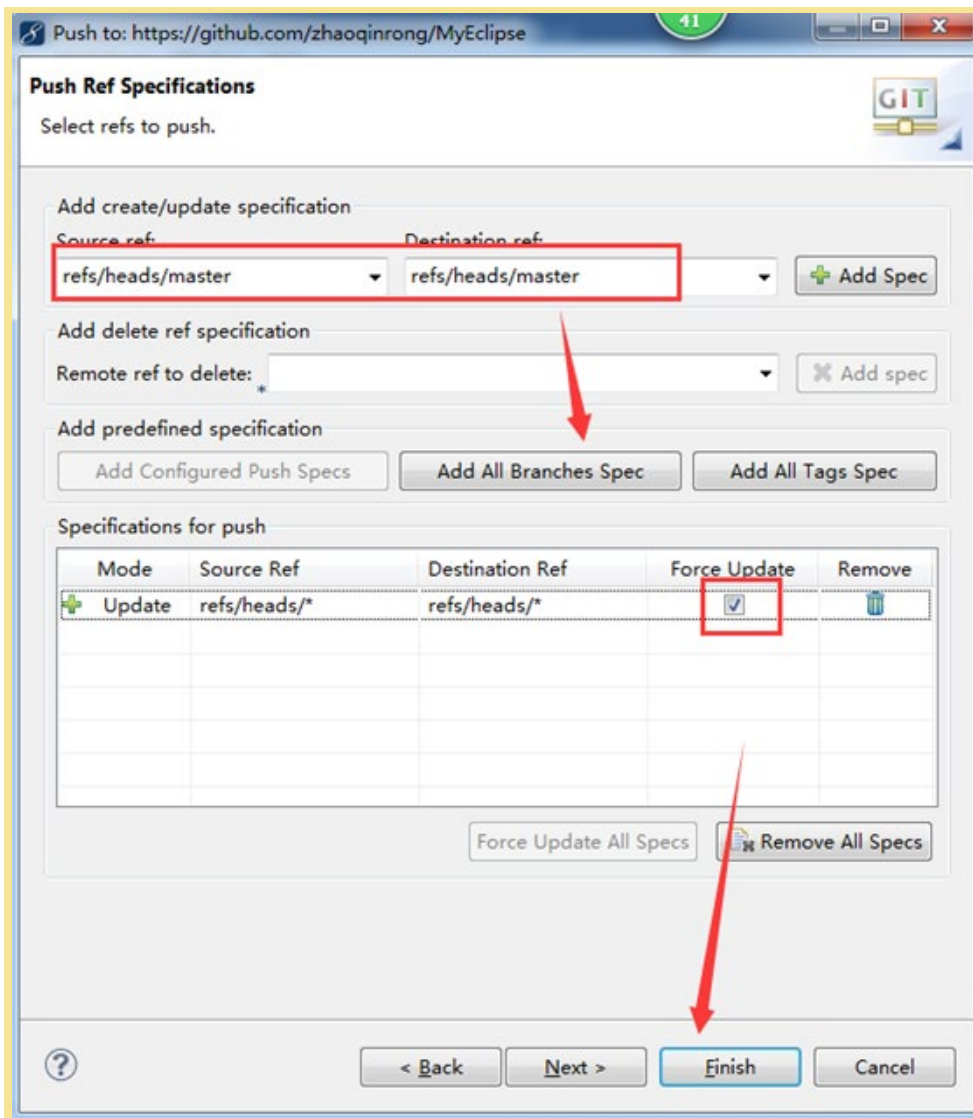


将项目提交到本地仓库

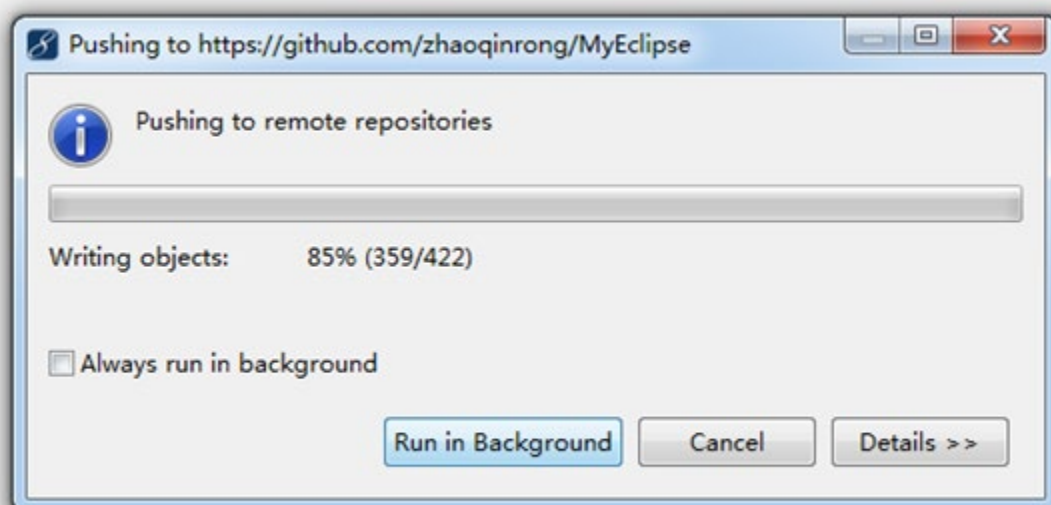


写上注释,选上所有文件进行 commit and push

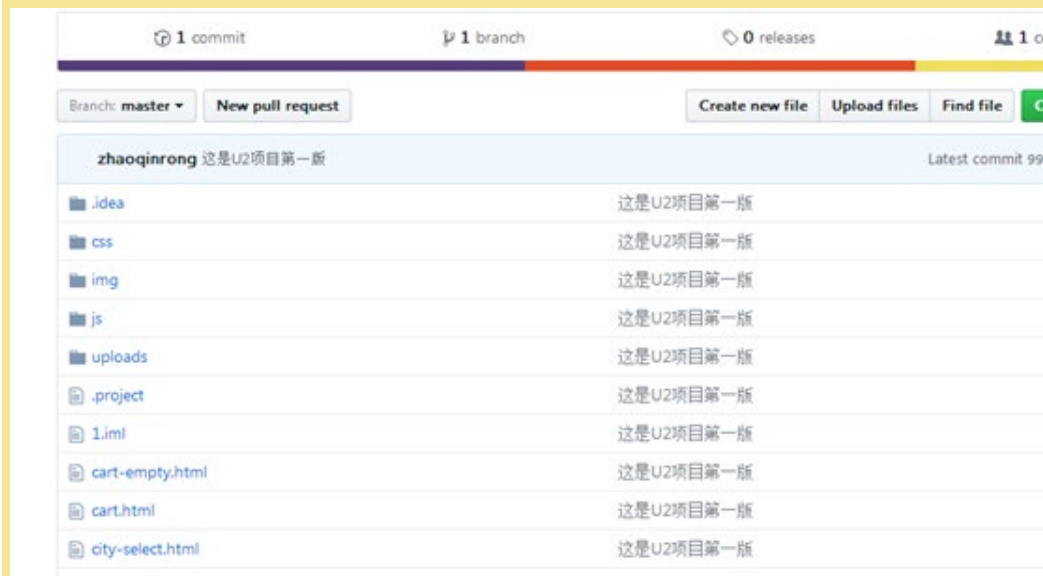




选择 master 分支



正在推送

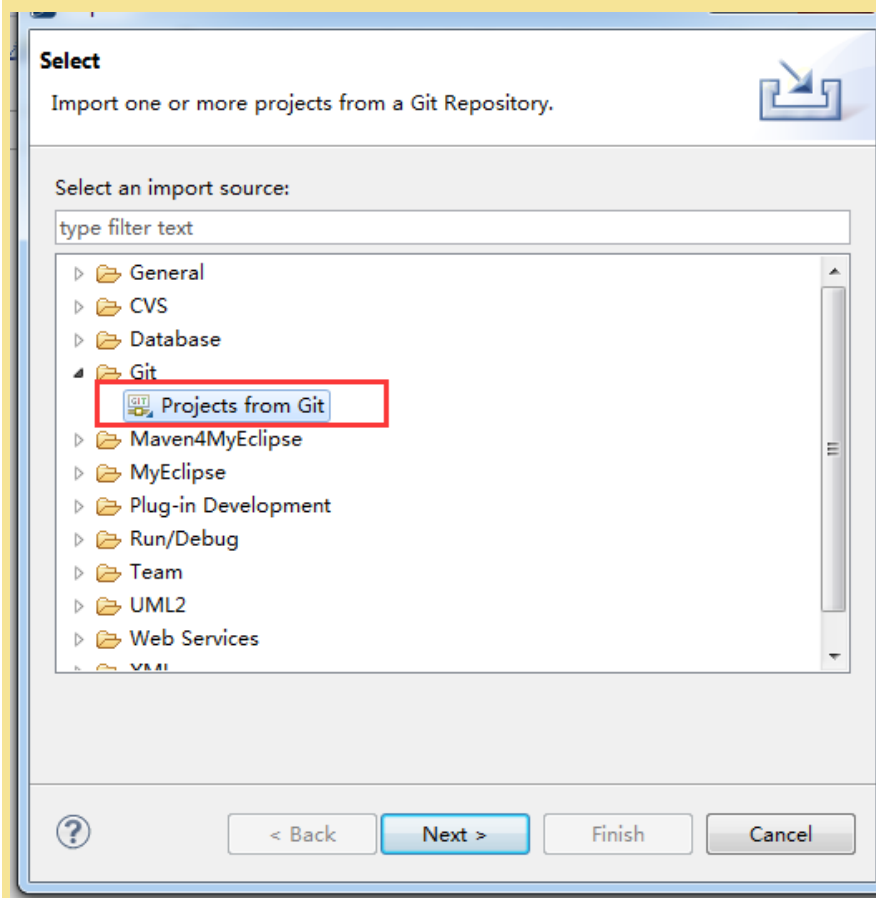


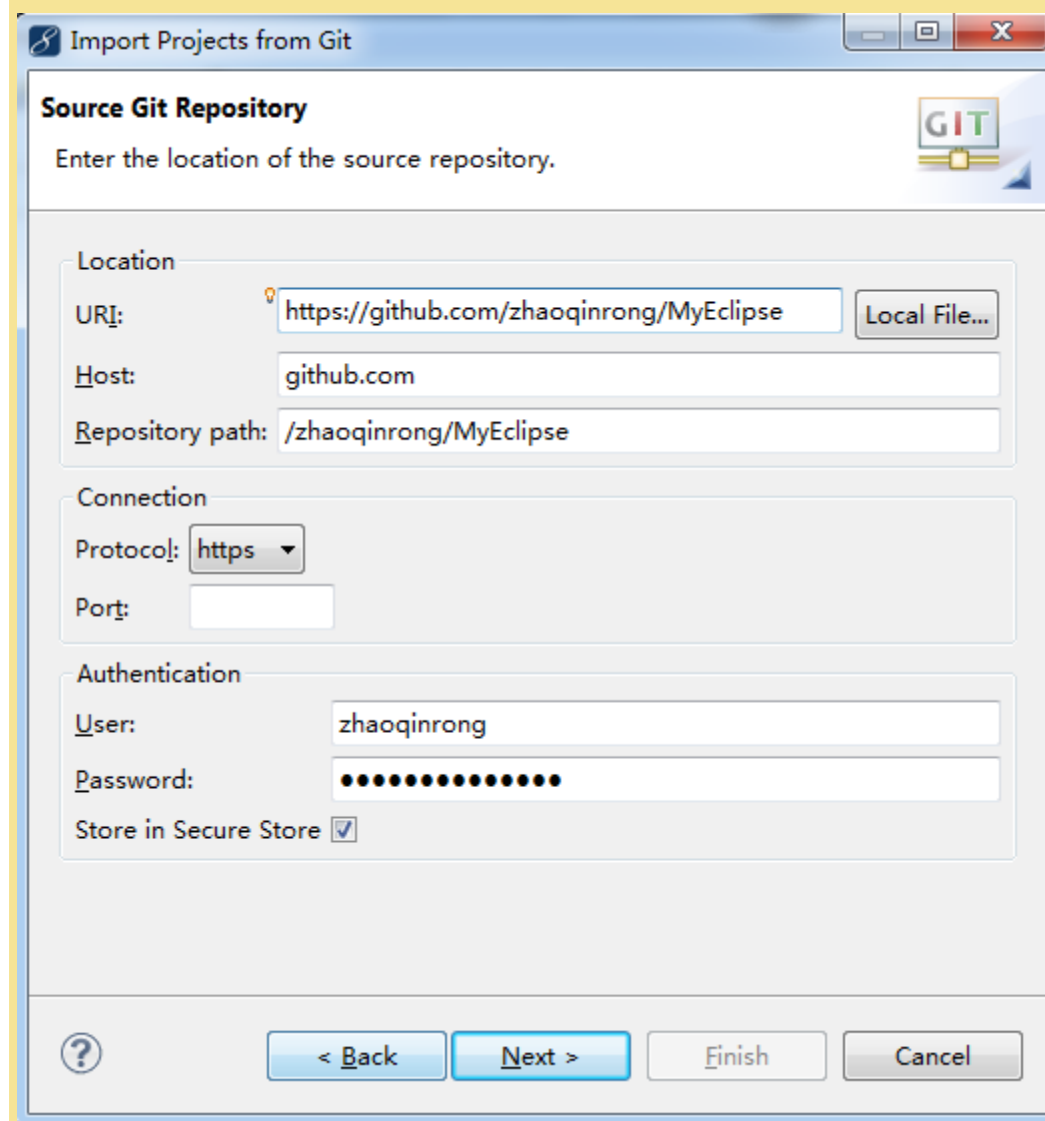
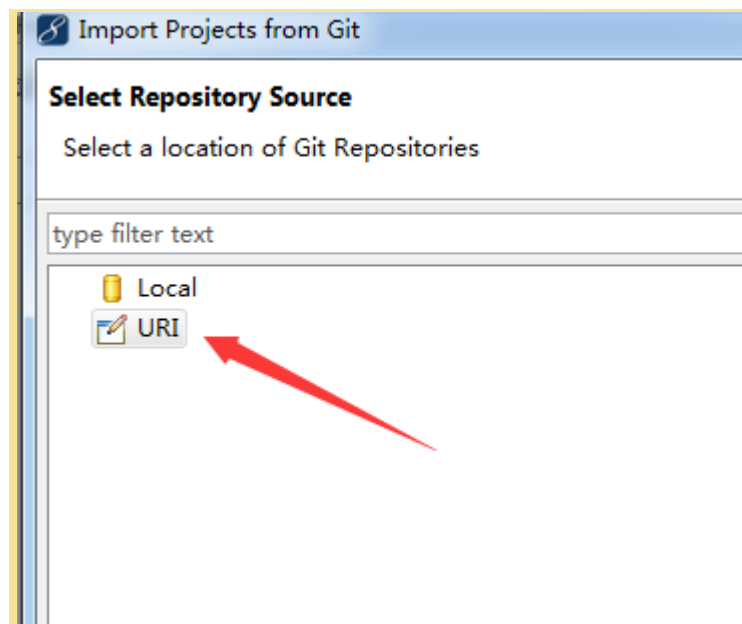
成功

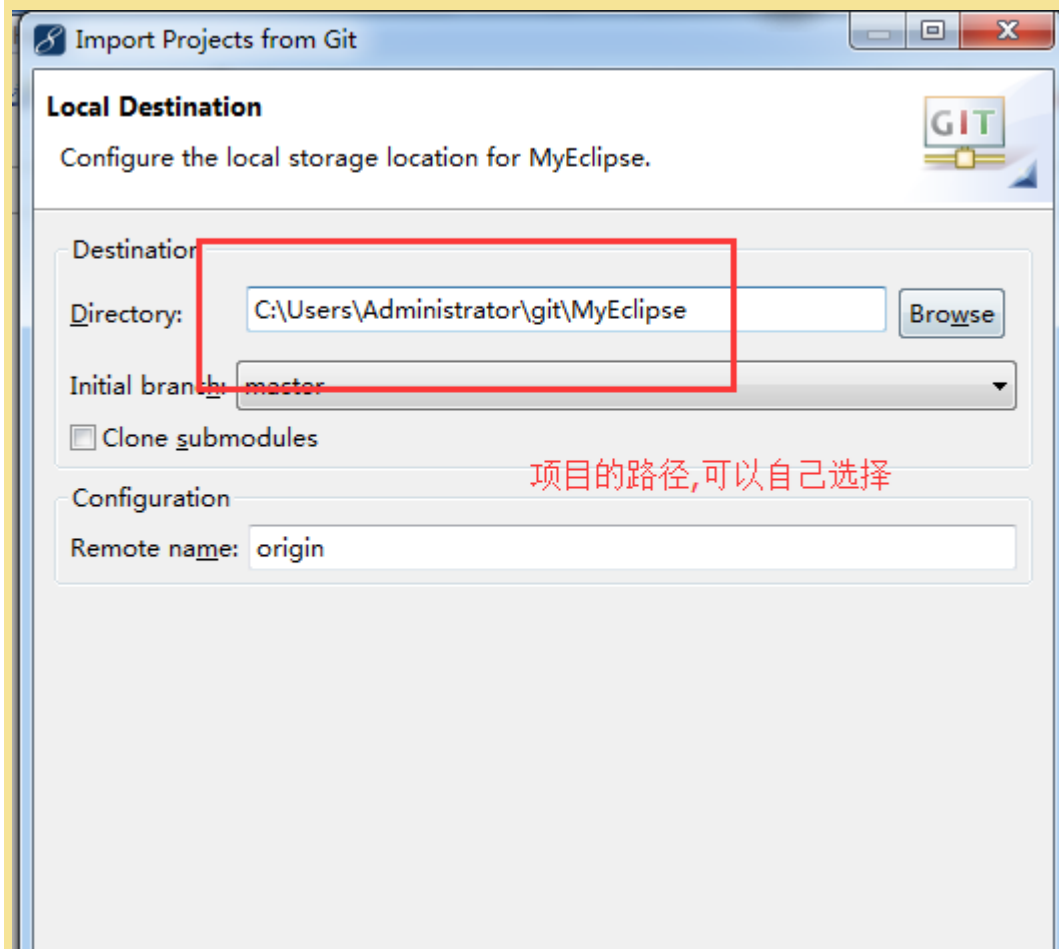
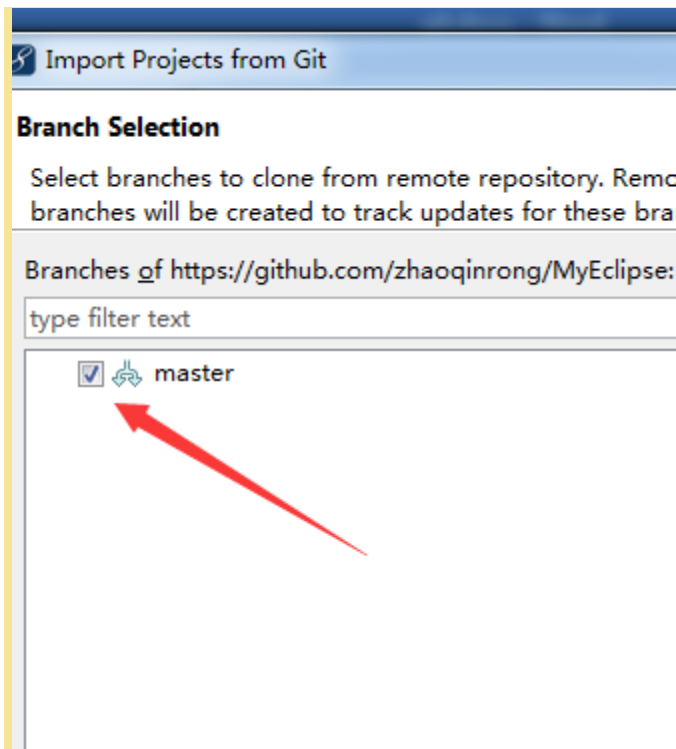
这是我们成功发布的项目,提交给 github 托管,当我们需要对项目进行修改的时候

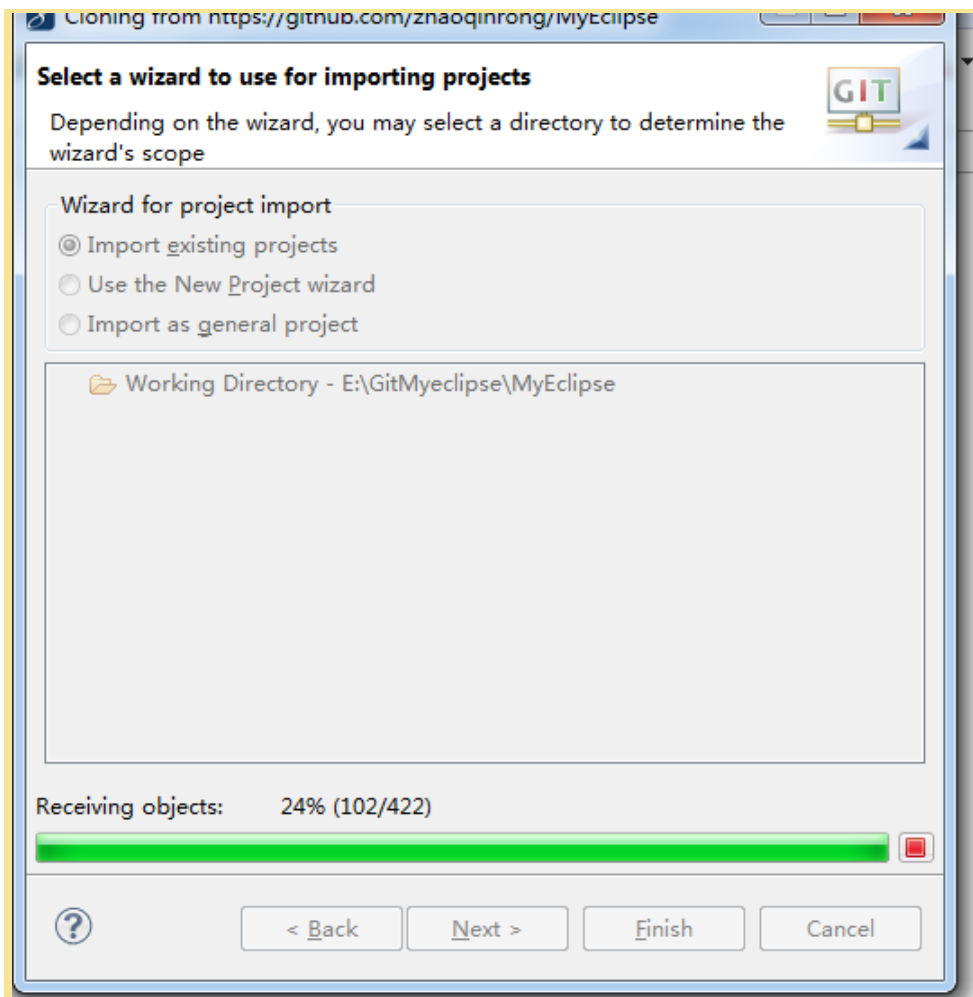
5.2 我们可以将项目从 github 上导入到本地

打开 MyEclipseàfileàimport

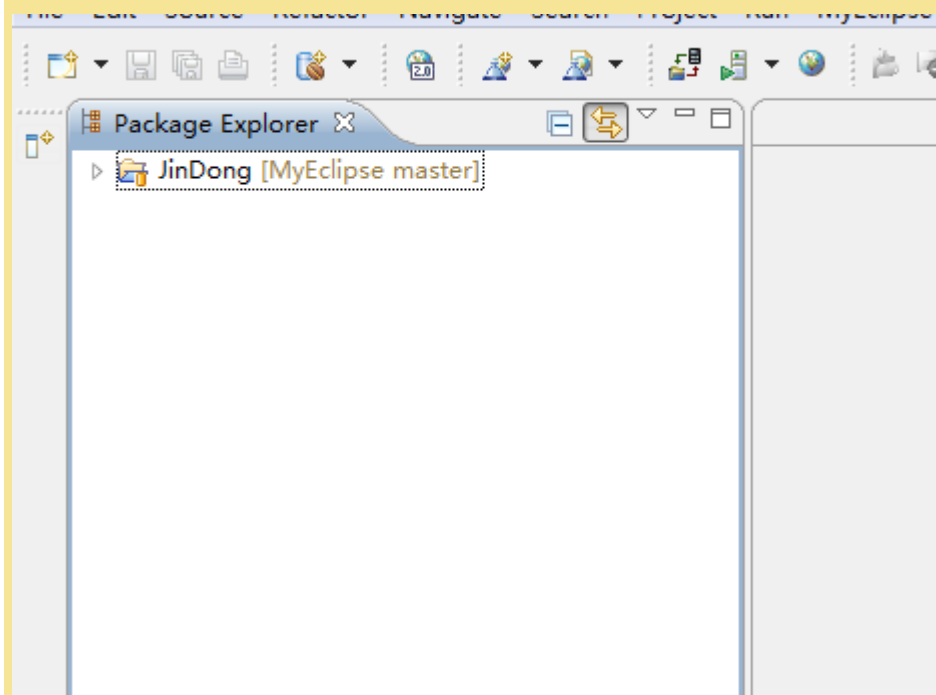






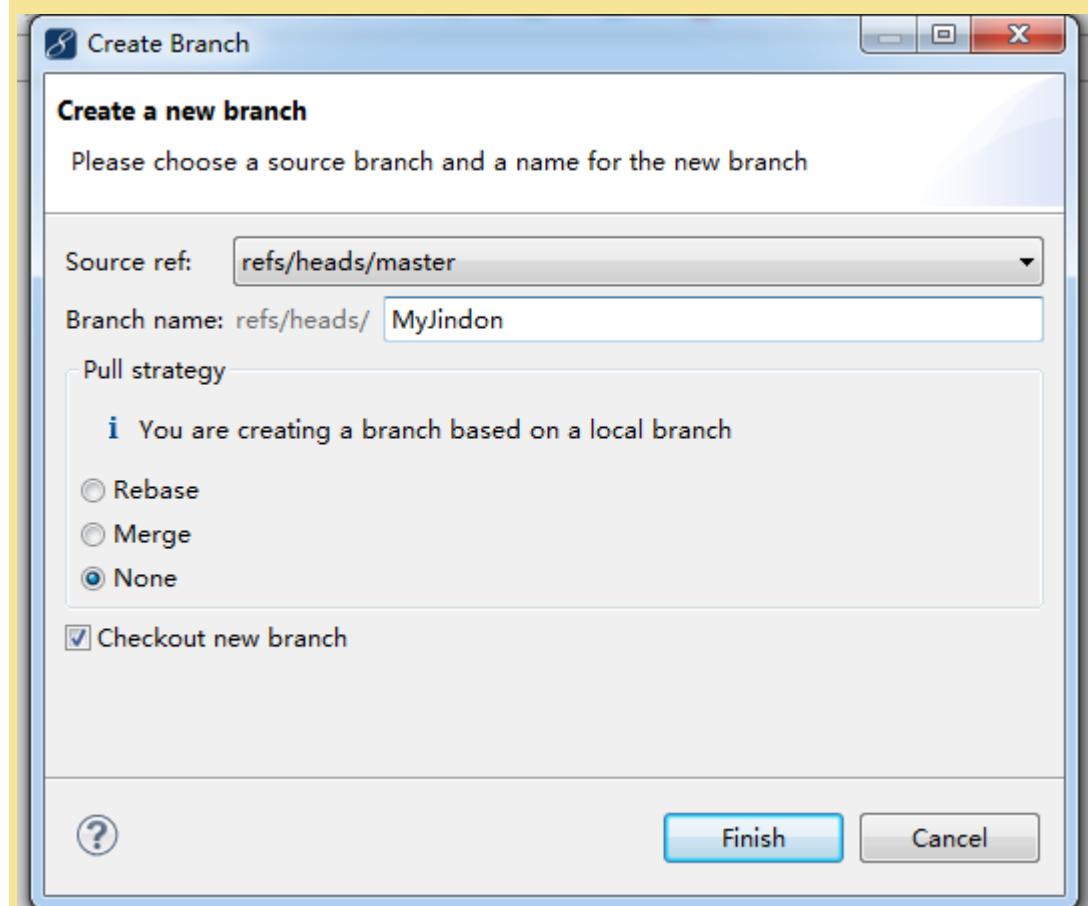
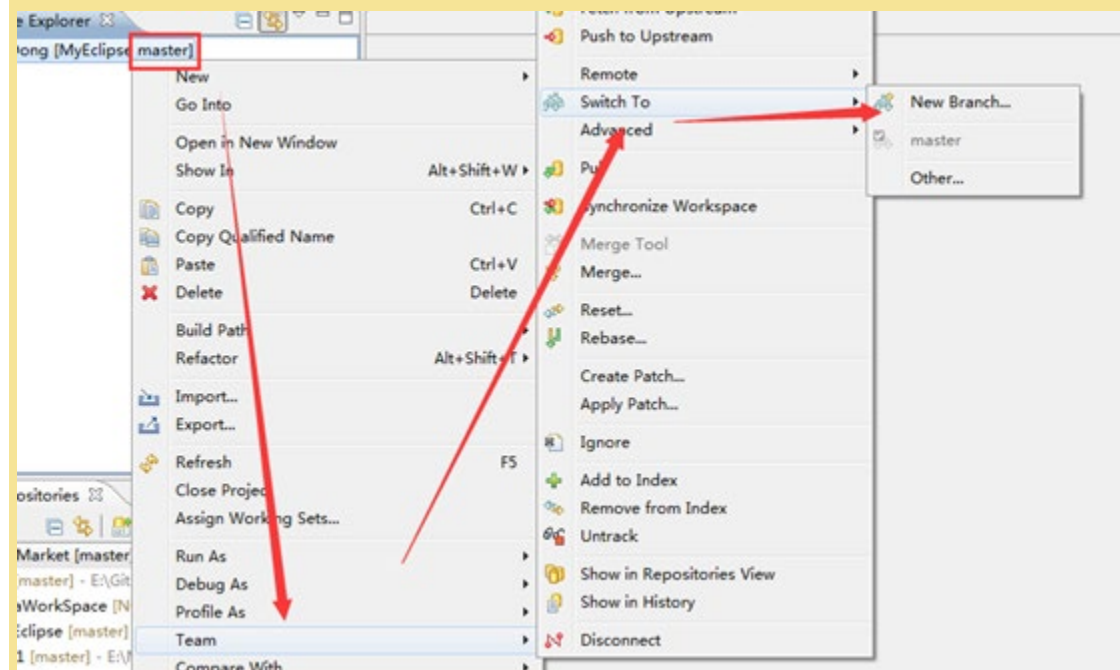


正在导入

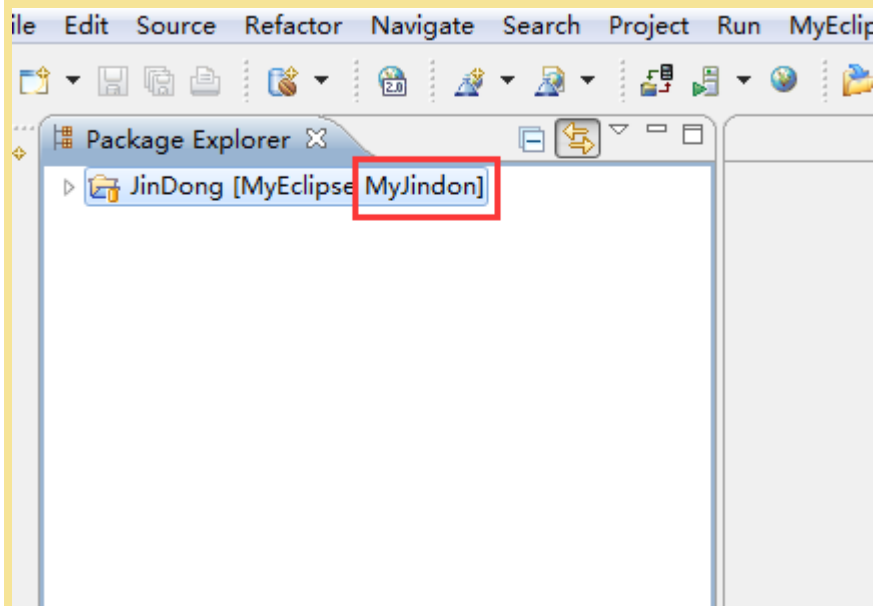


导入完成,然后我们在这个项目上进行开发,不会影响到我们仓库中的项目的完整性

目前是 **master** 分支,我们需要新建分支进行我们的开发

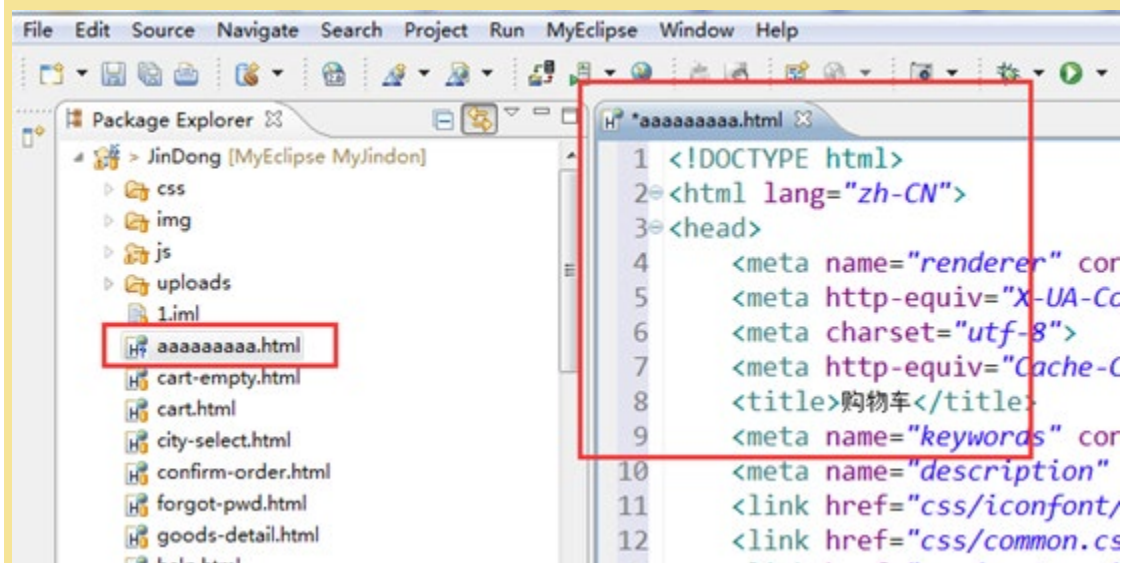


新建分支后,自动切换分支



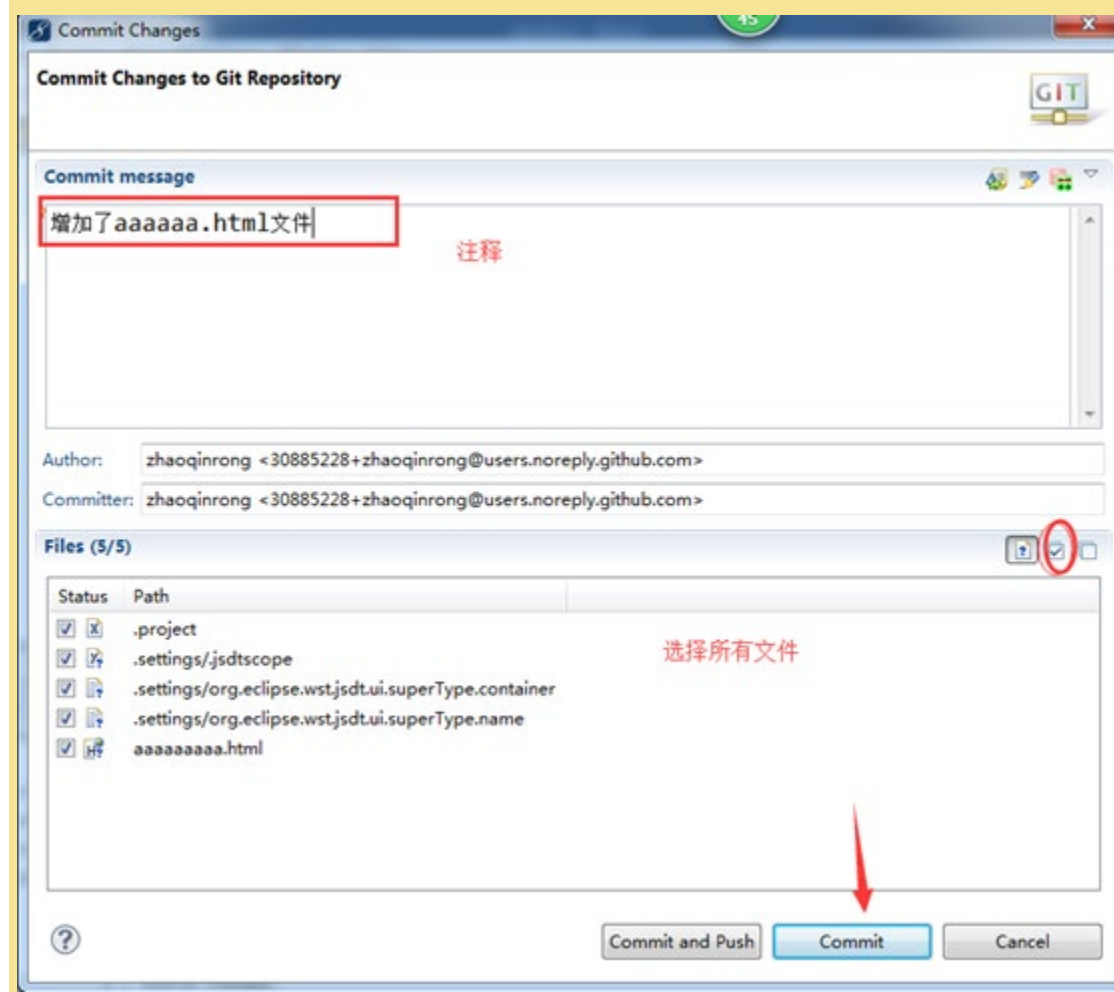
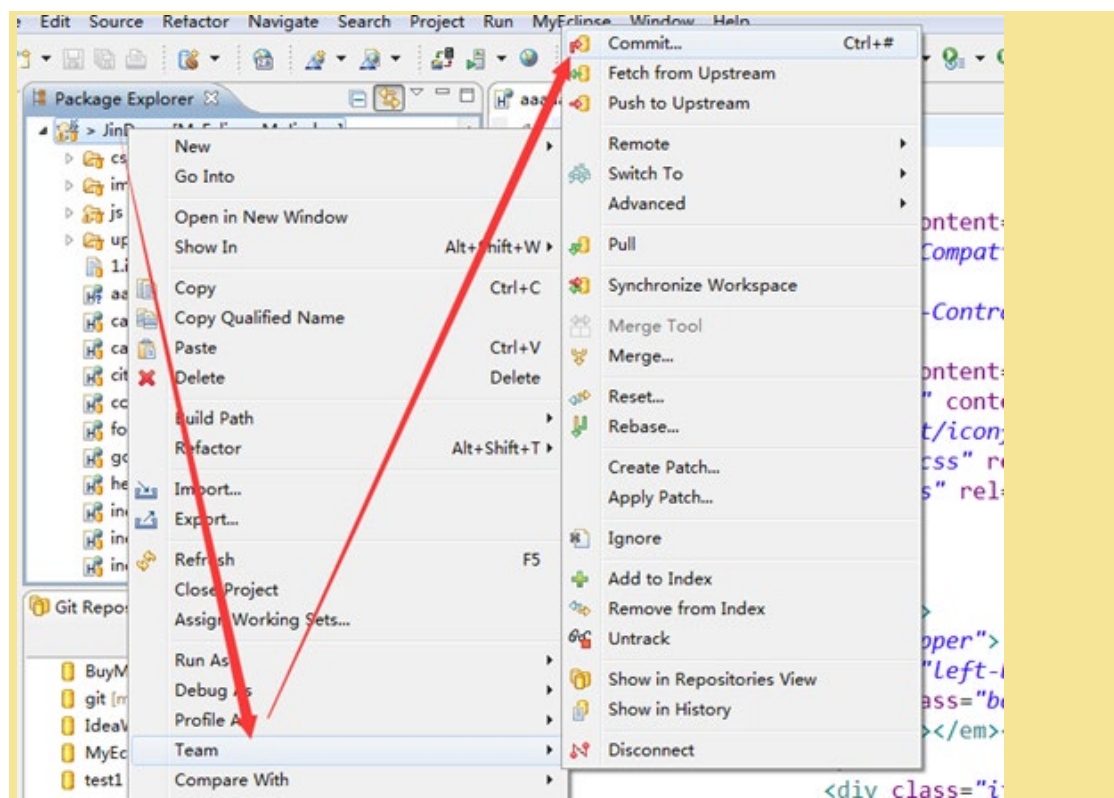
5.3 在分支上进行开发,并 git 到 github 分支上

1.现在我们已经切换到了上一节创建的分支上,现在我们在我们之前的项目上做一点修改

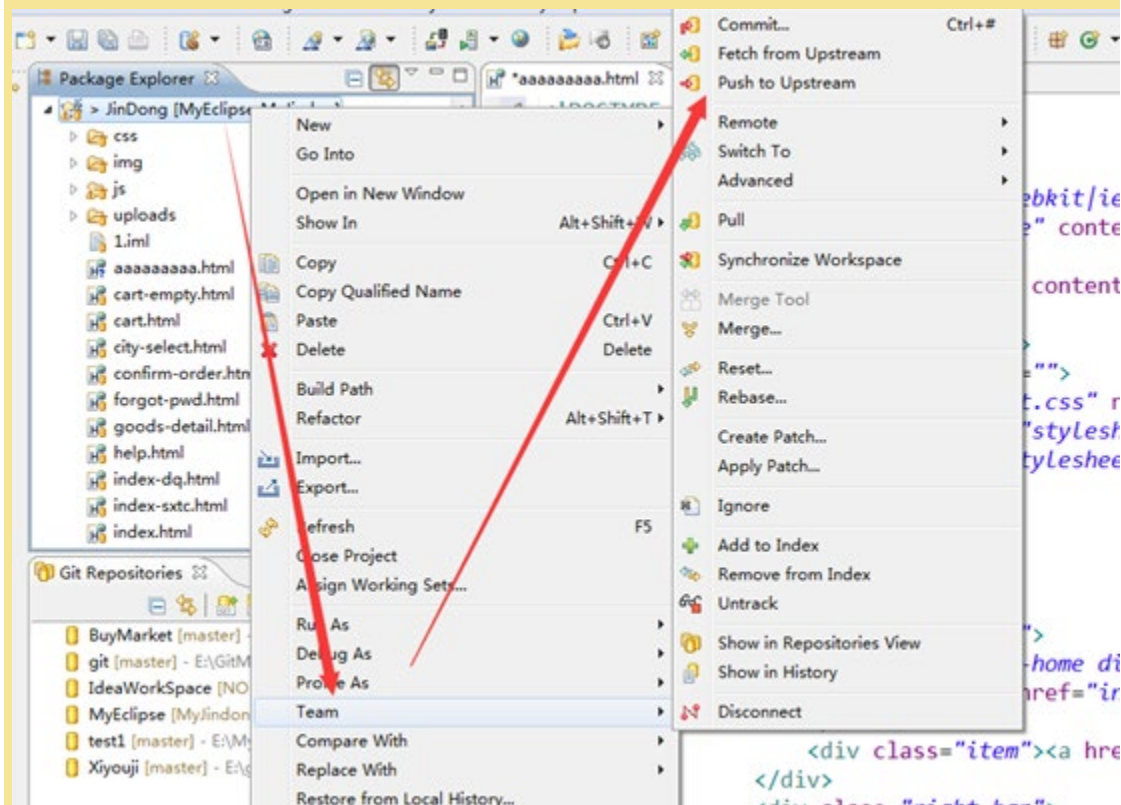


这是我在 MyJindong 分支上对项目进行的一个修改,新增了一个 aaaaa.html 的文件

2.先将我们修改的项目 git 到本地的仓库进行一次 commit

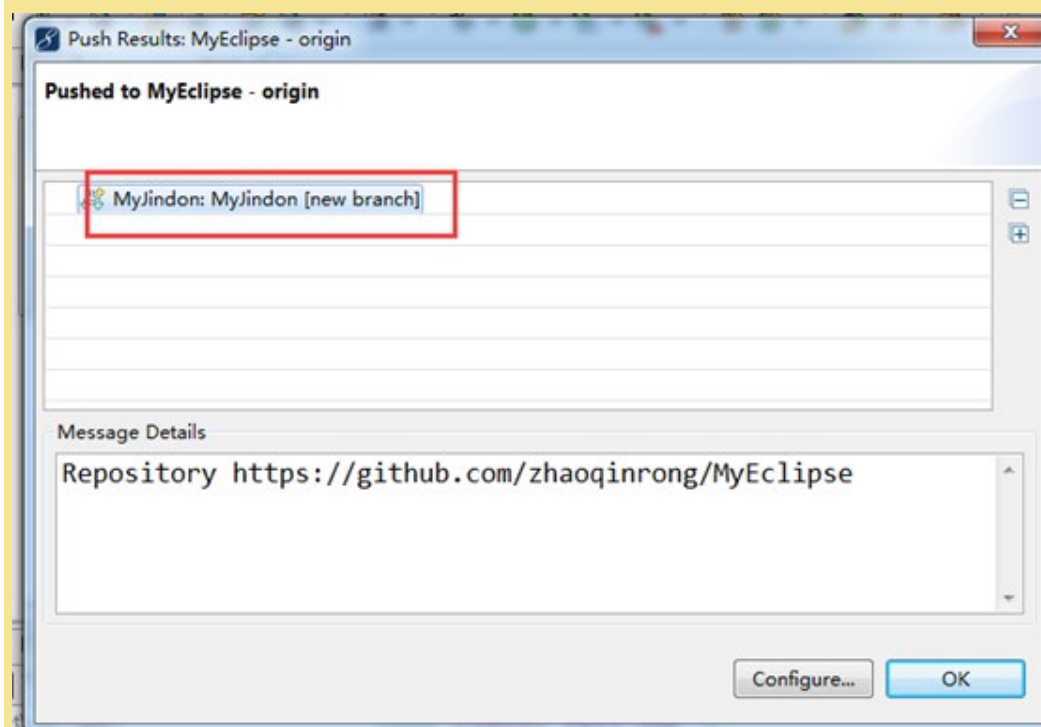


2. 将更新推送到远程仓库,并自动创建分支

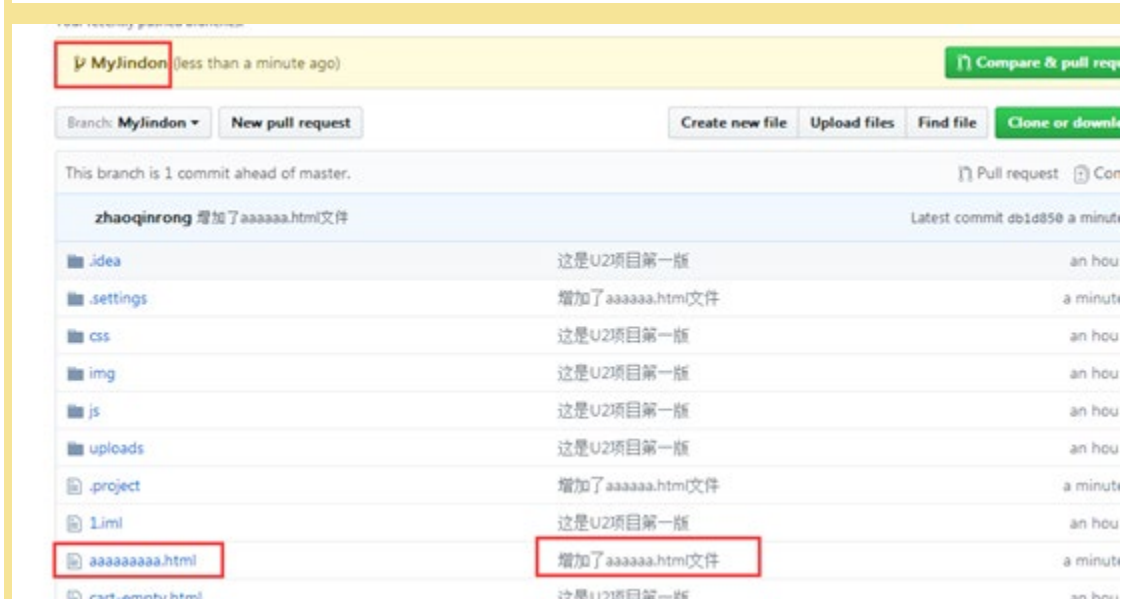
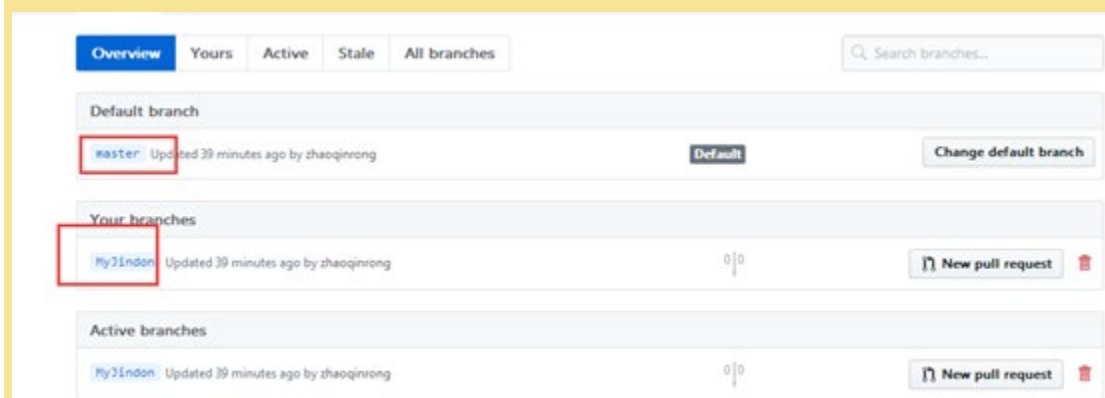


这是检查本地和远程仓库的差异:

检查出本地和远程仓库有差异,可以进行,然后在仓库中创建新分支,将我们的修改后的项目 git 到远程仓库



3.这是我们的新分支和 master 分支,这就成功的,将新分支 git 到了远程仓库

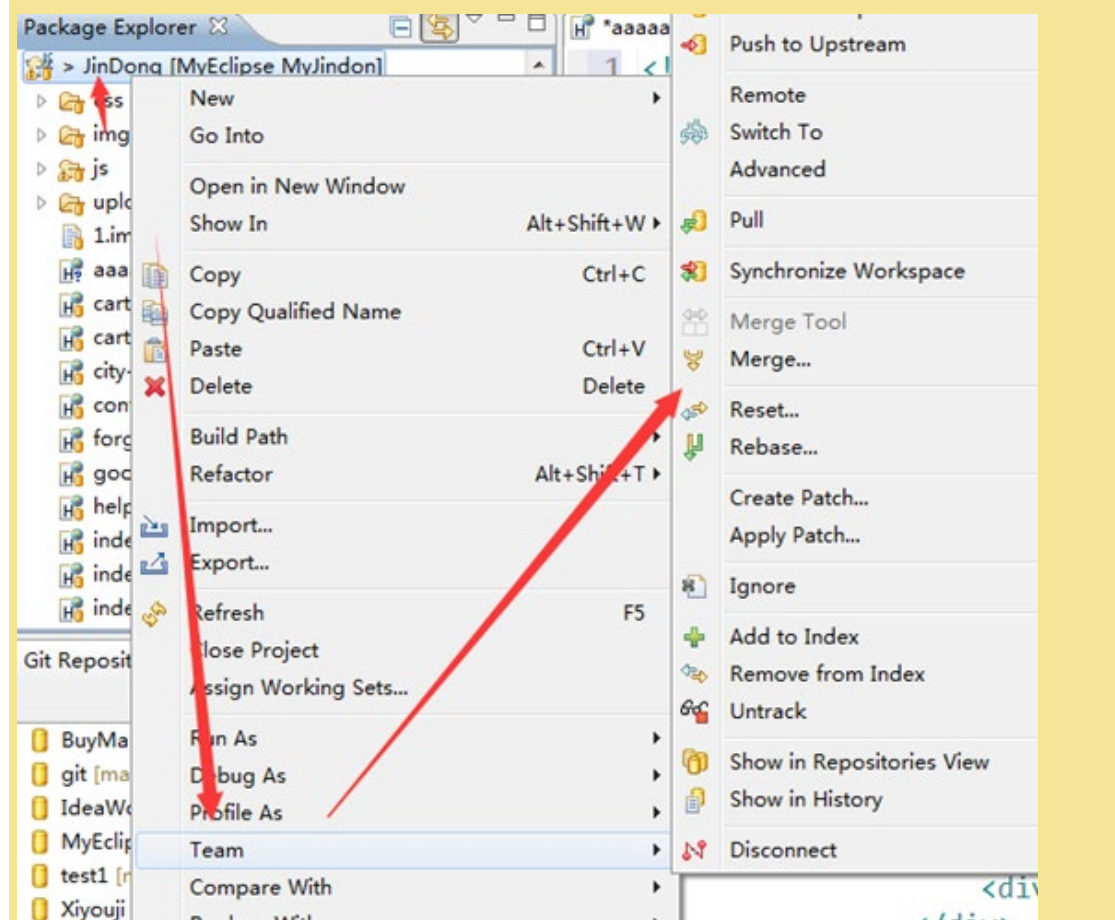
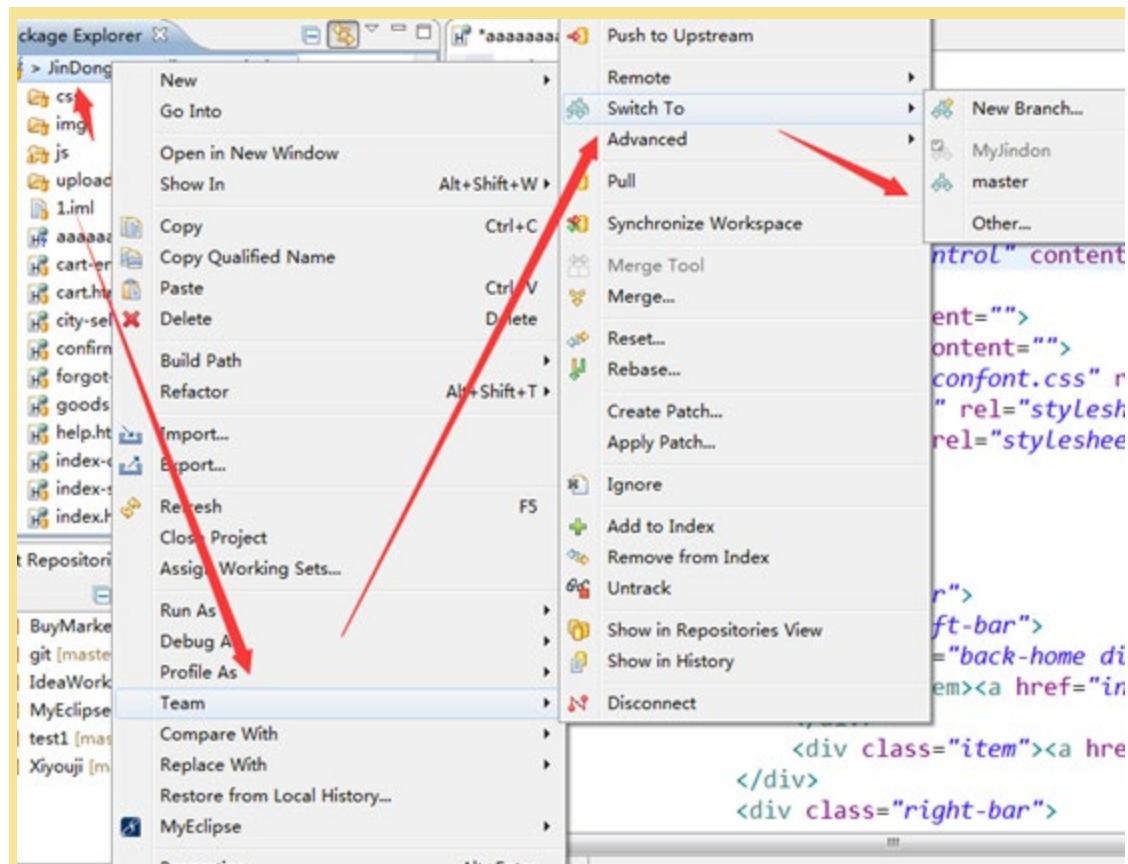


5.4 将我们分支开发的项目和 master 分支的进行合并

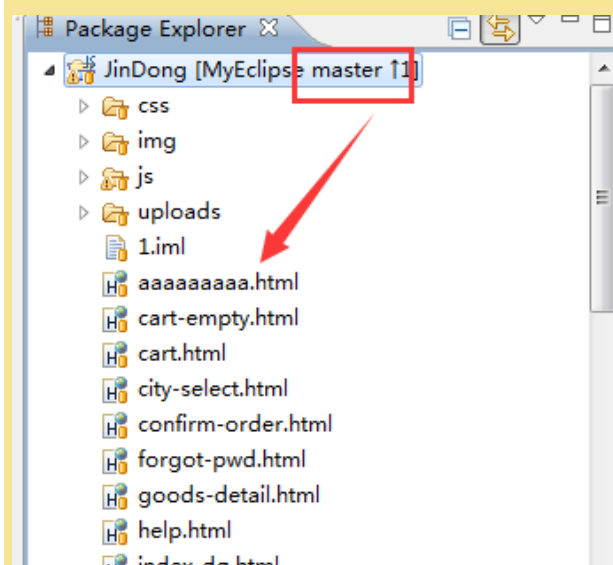
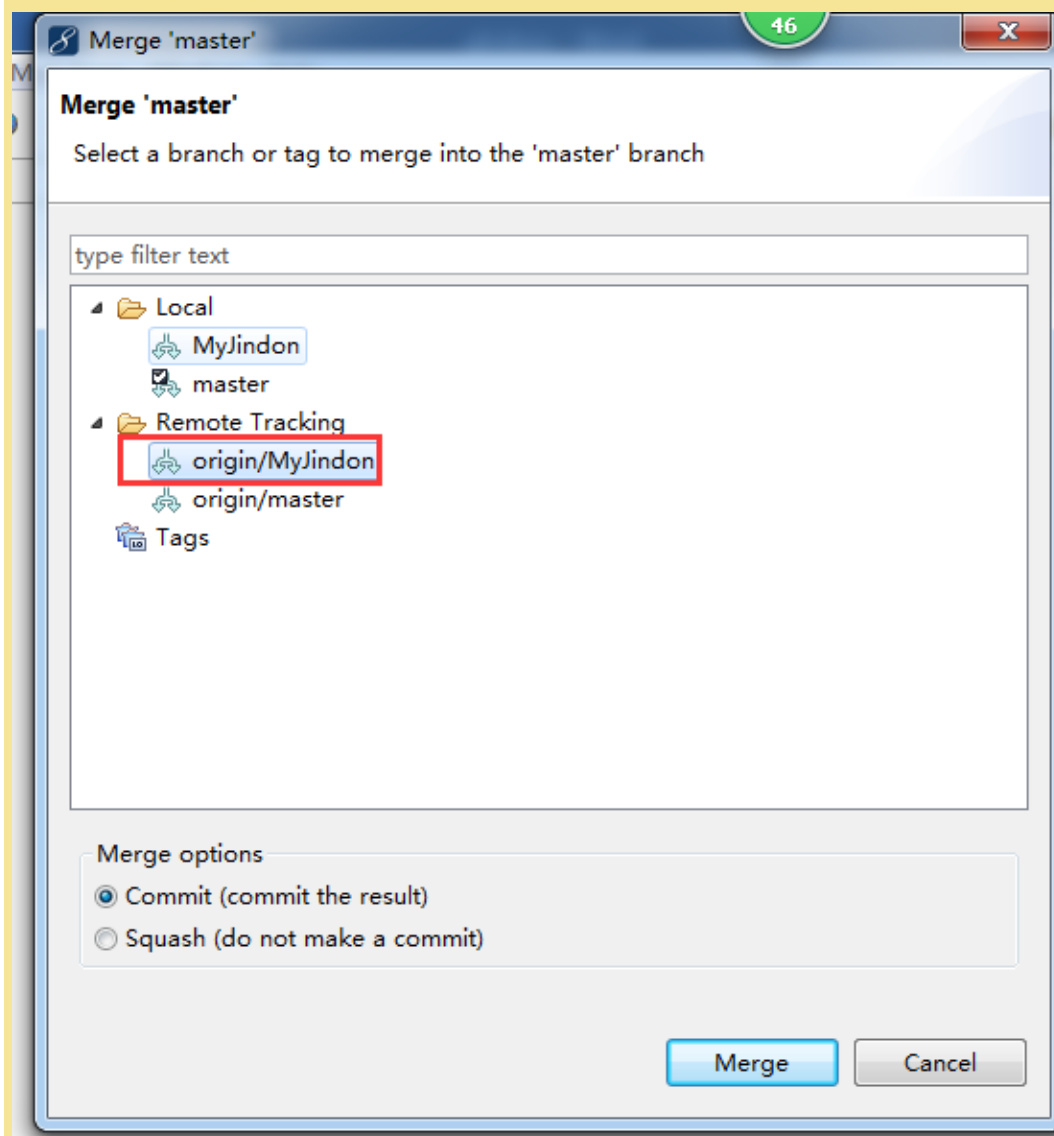
我们第一版的项目有 bug,我们在新分支上进行修改,将修改后的项目与我们的之前的项目进行合并

1. 首先我们要确定我们要与哪个分支合并,然后将我们的项目切换到哪个分支下,然后使用 Merge 进行合并

我们这里要与主分支合并,首先切换到主分支,看主分支是否有更新,然后我们合并到本地项目

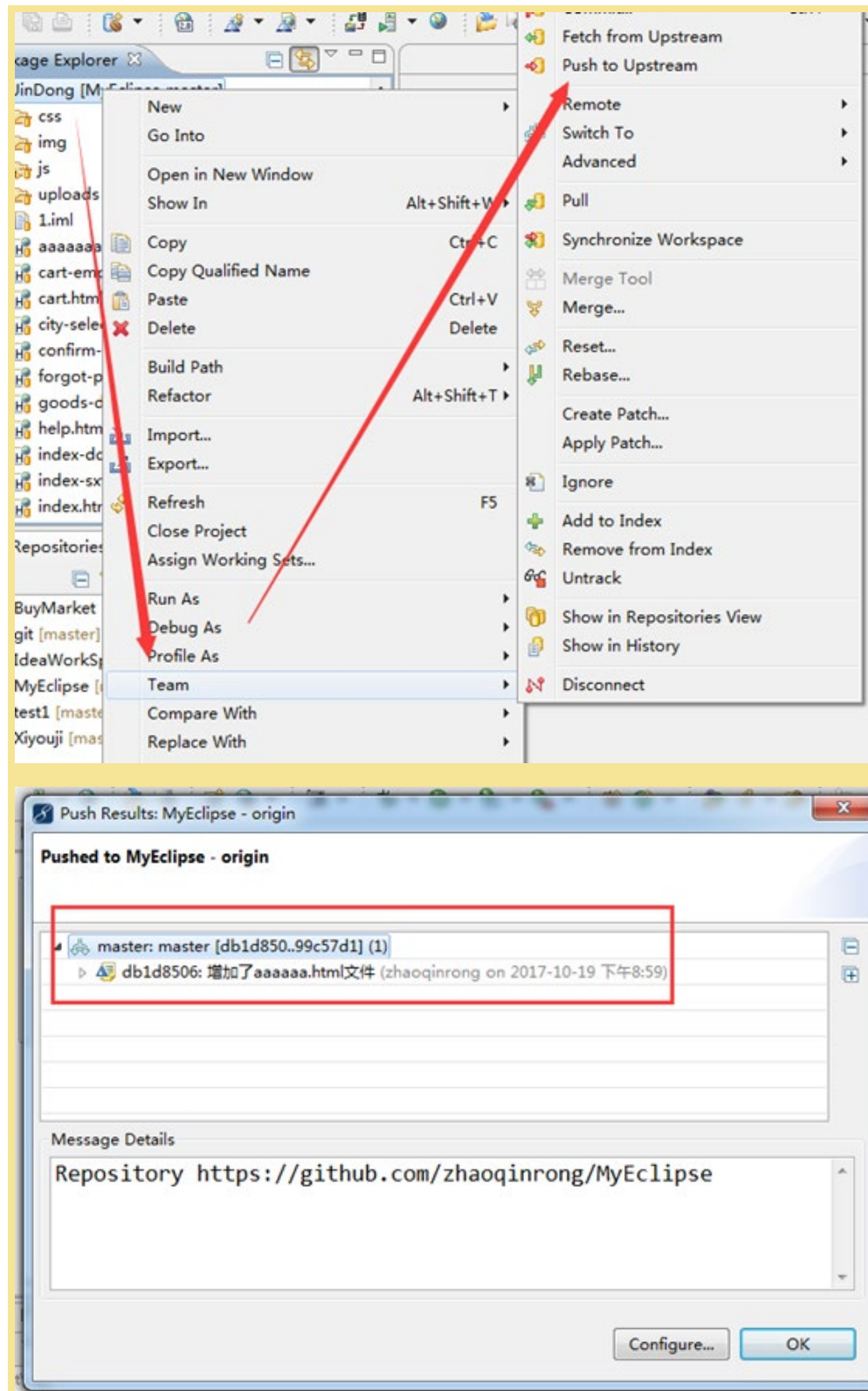


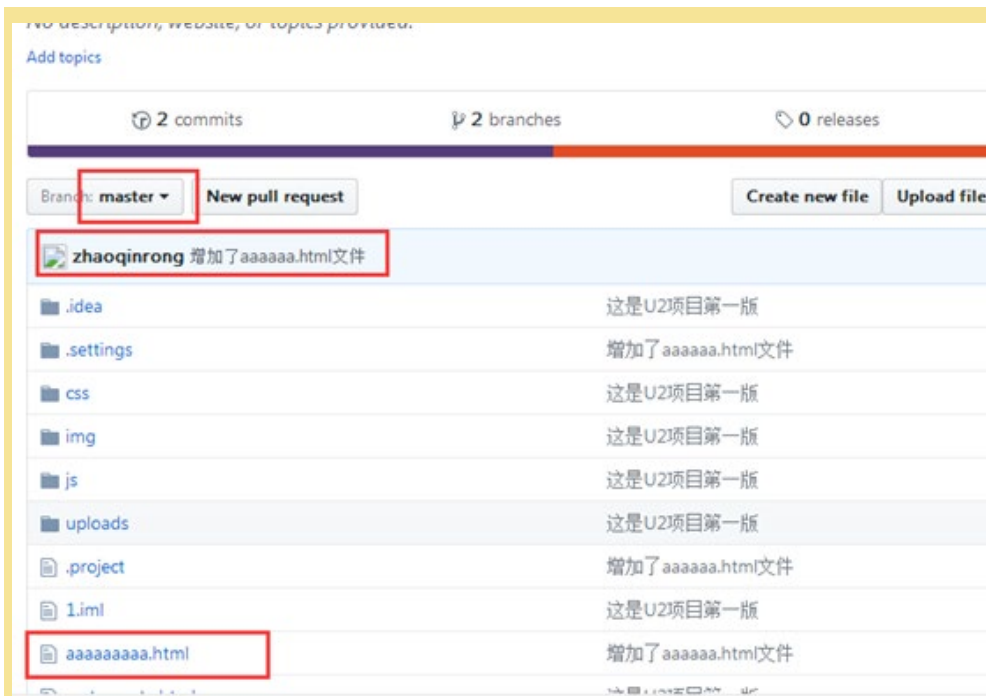
选择我们修改的项目进行本地合并



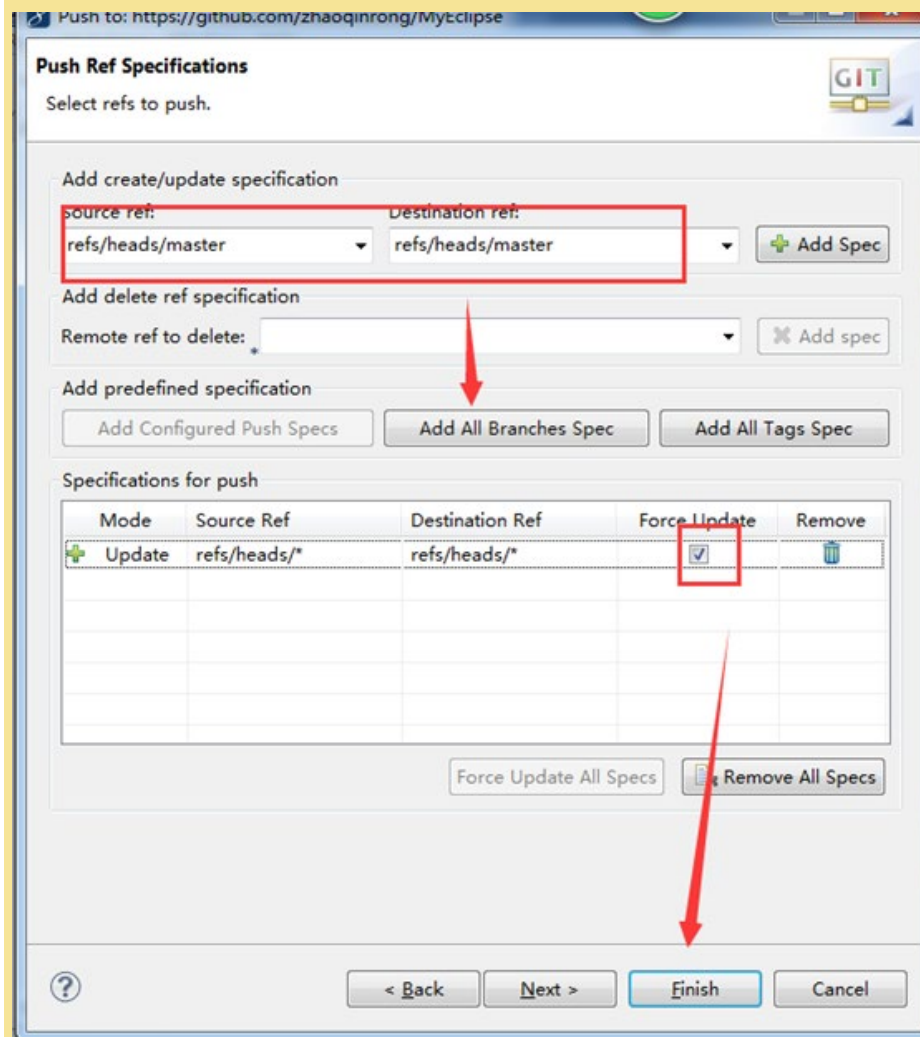
合并后的结果

1. 然后选择 push to upstream 进行检查,是否有更新需要上传到我们的远程服务器

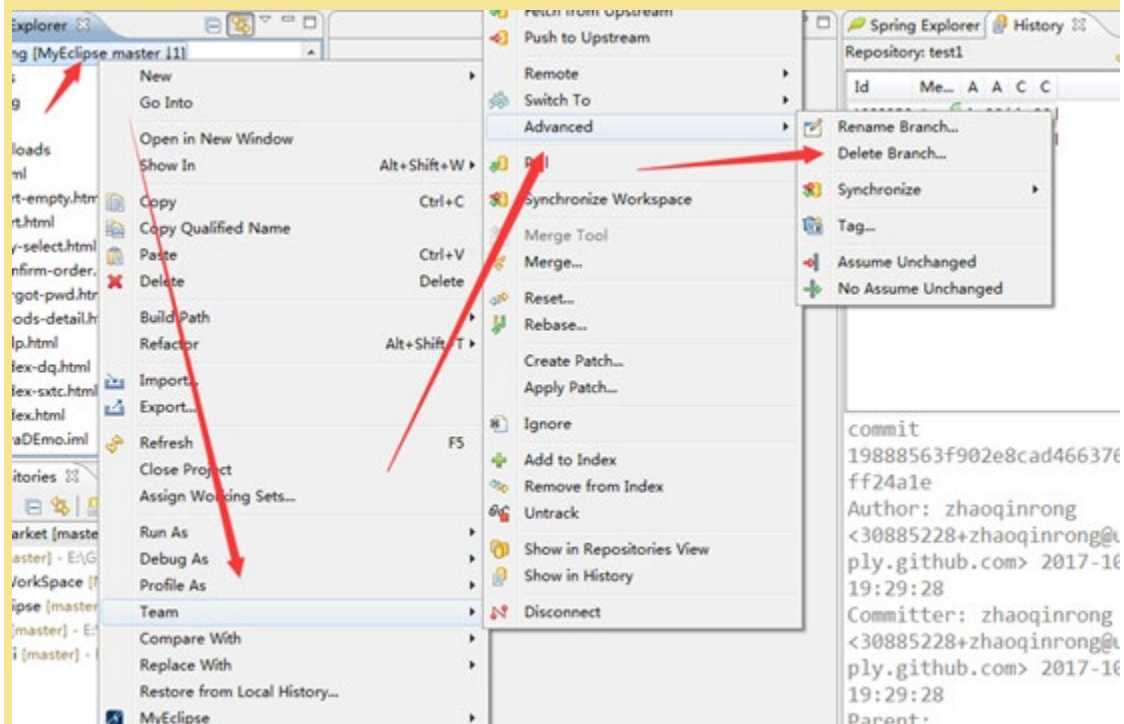




这是合并到远程仓库的结果.成功

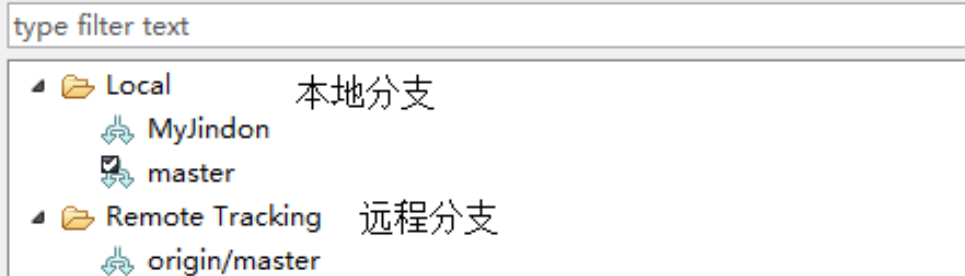


5.5 删除本地分支和远程分支



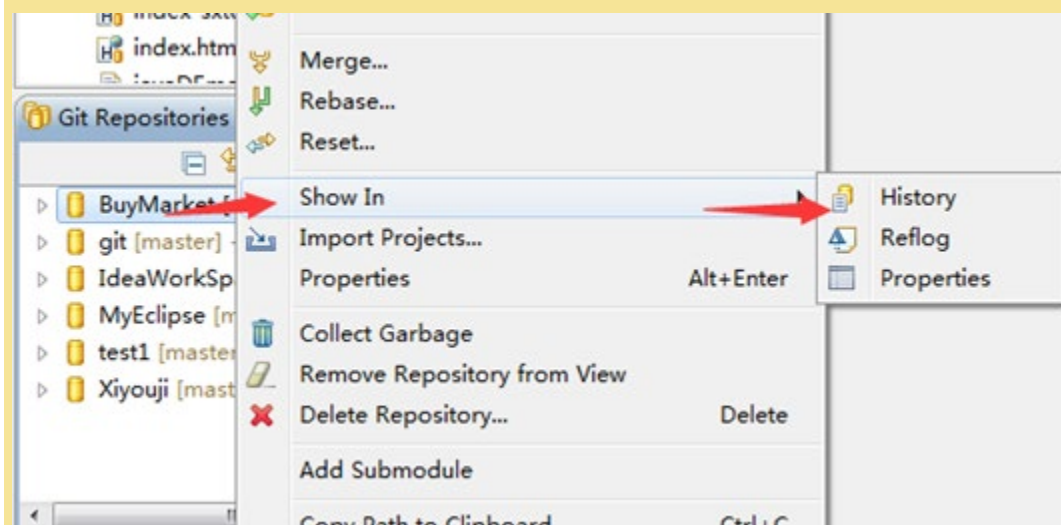
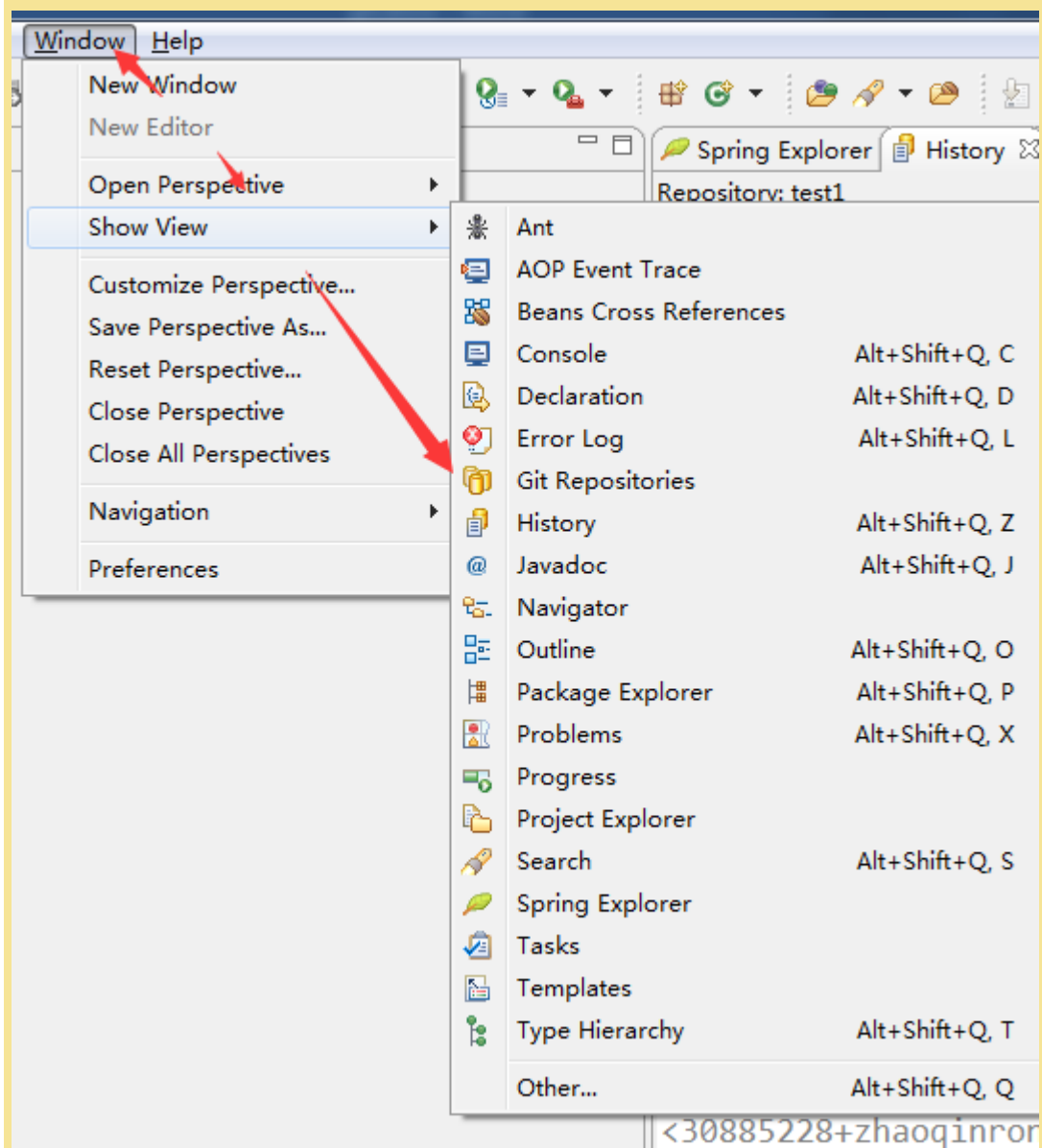
Delete a branch

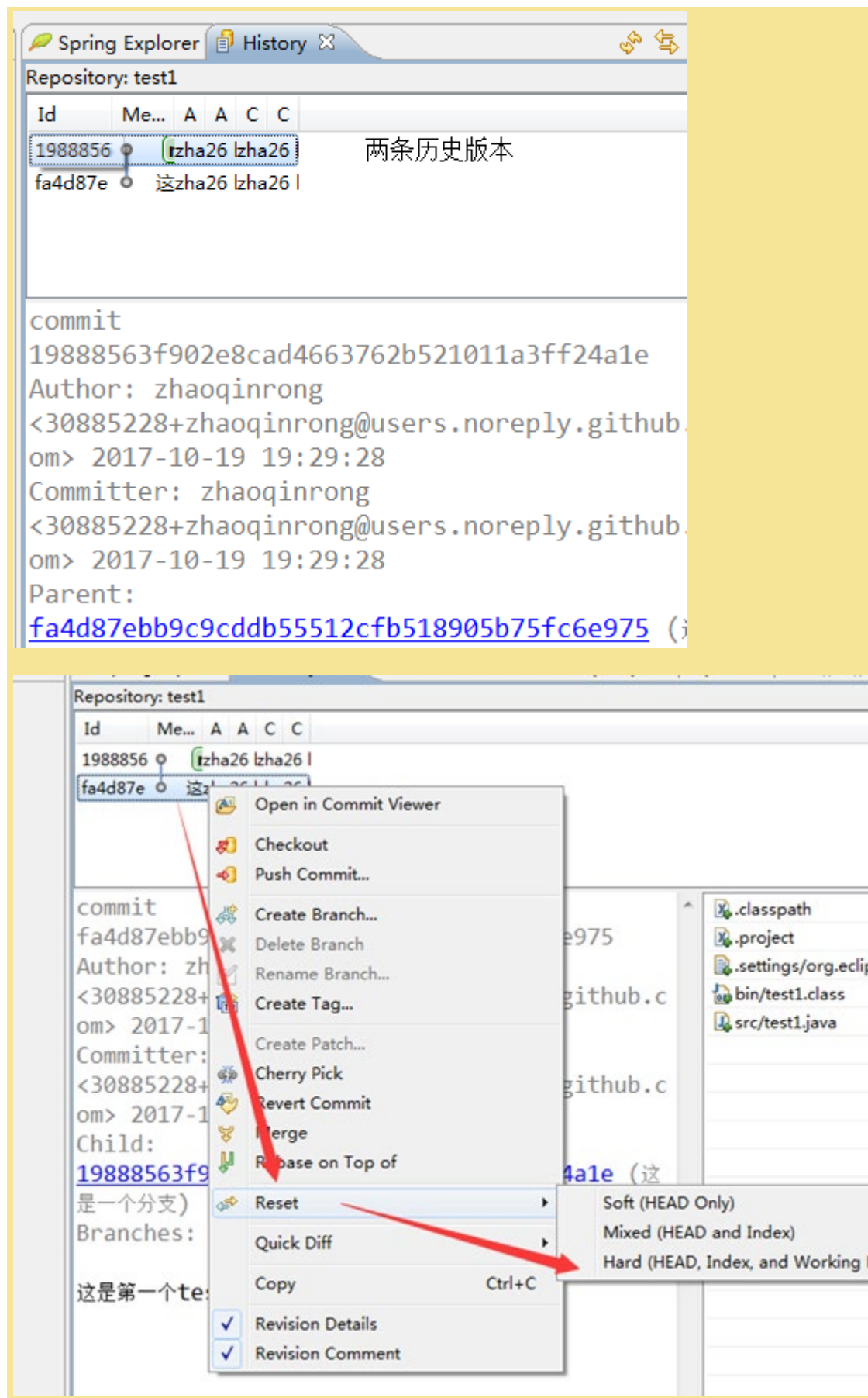
Select a branch to delete



选择以后点击 OK 即可

5.6 版本回退





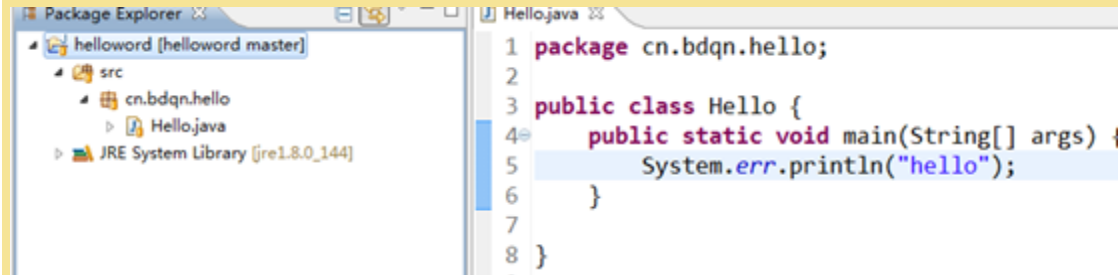
便可以进行版本的回退

5.7 使用 eclipse 解决代码的冲突

模拟

①从远程仓库中导入一个项目到我们的 eclipse 中

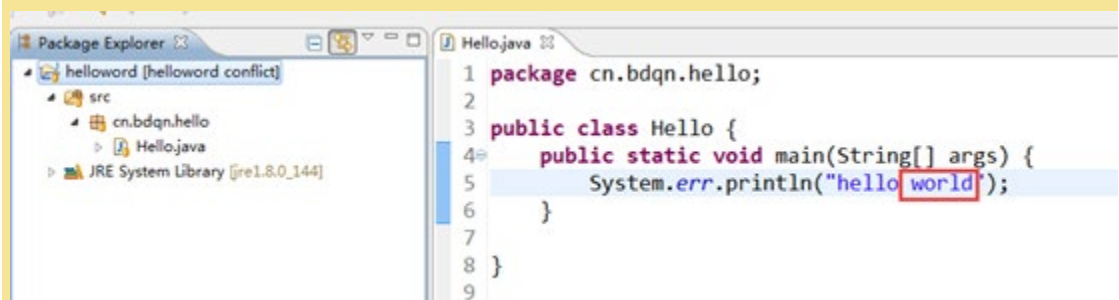
导入过程不过讲解



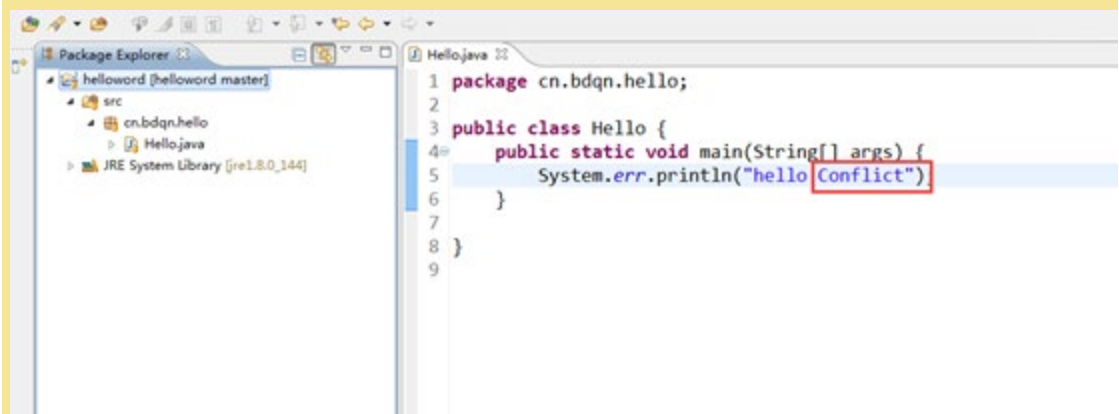
这是我们导入的项目,并进行本地版本库的提交

②创建新分支

并对项目内容进行修改.然后提交到本地版本库

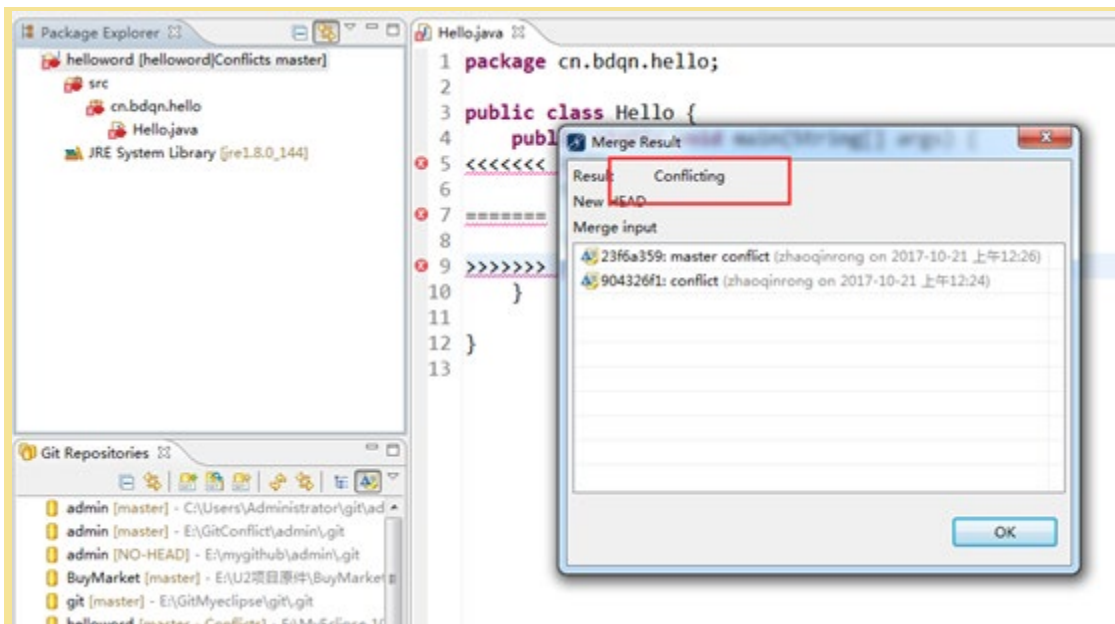


④ 切换到主分支 master,对项目进行修改,并提交本地版本库

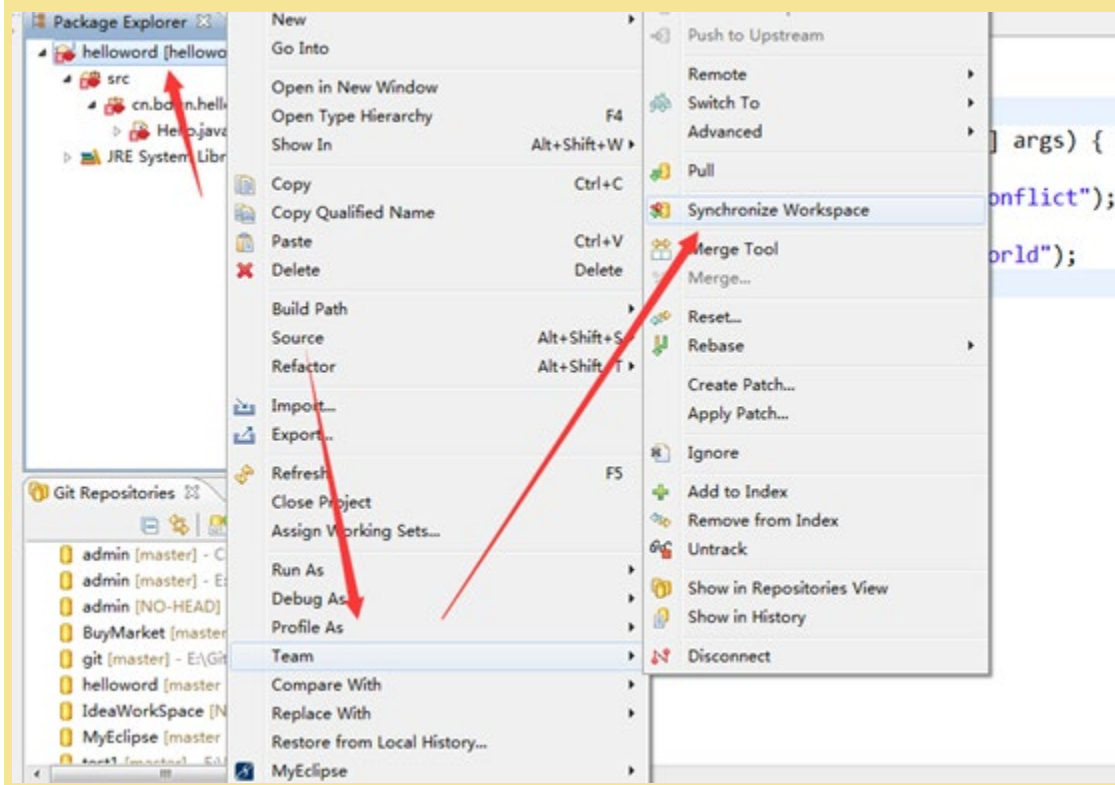


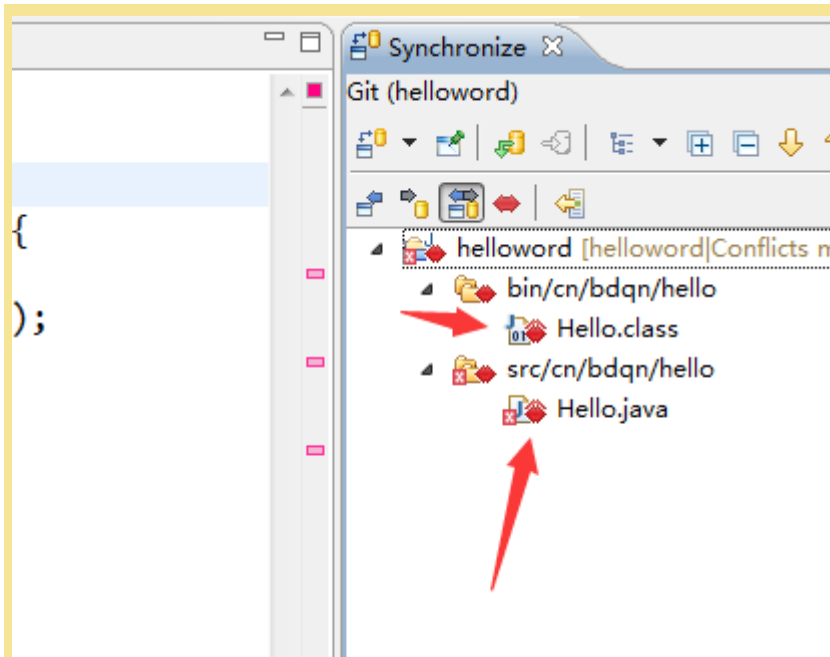
⑤ ,与 conflict 分支进行合并

便出现了我们期待的结果



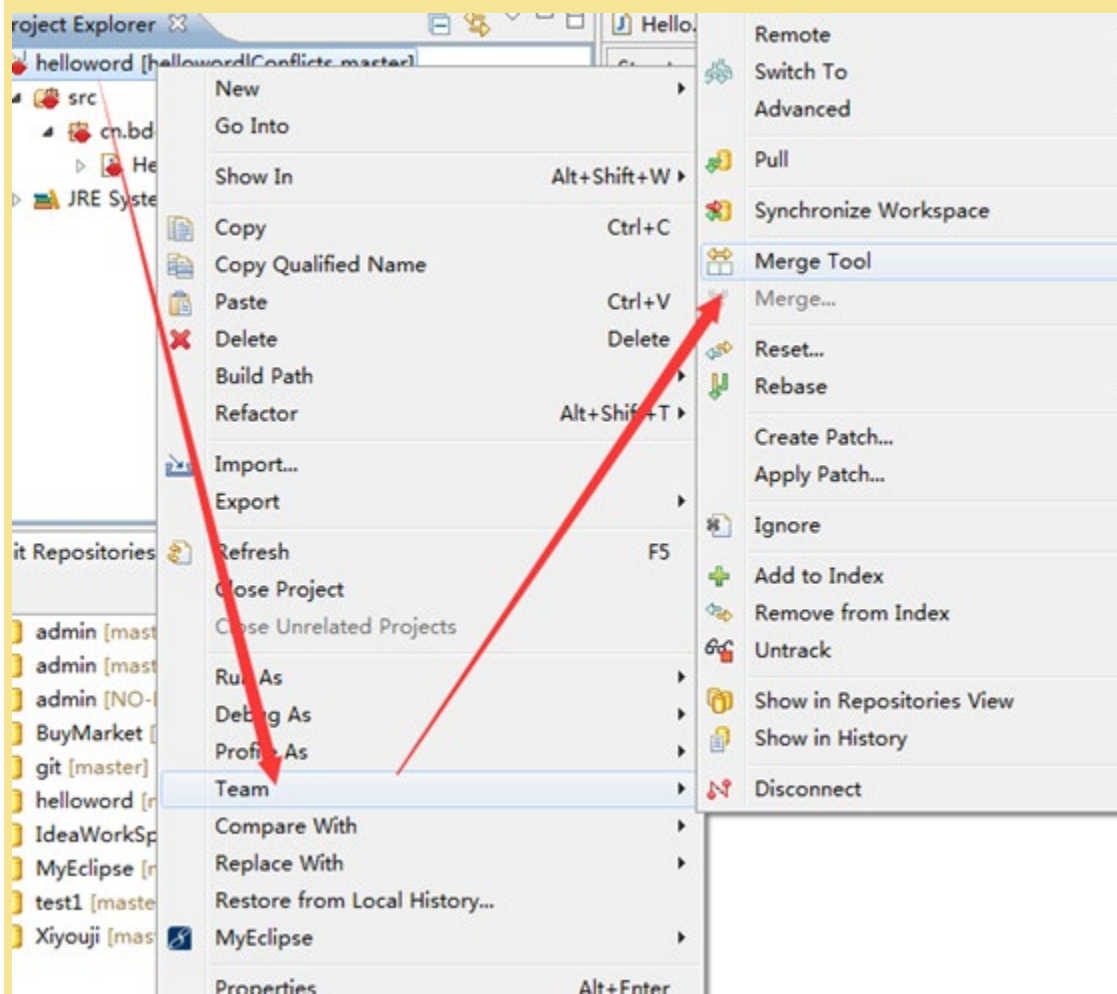
⑥ 查看冲突的代码

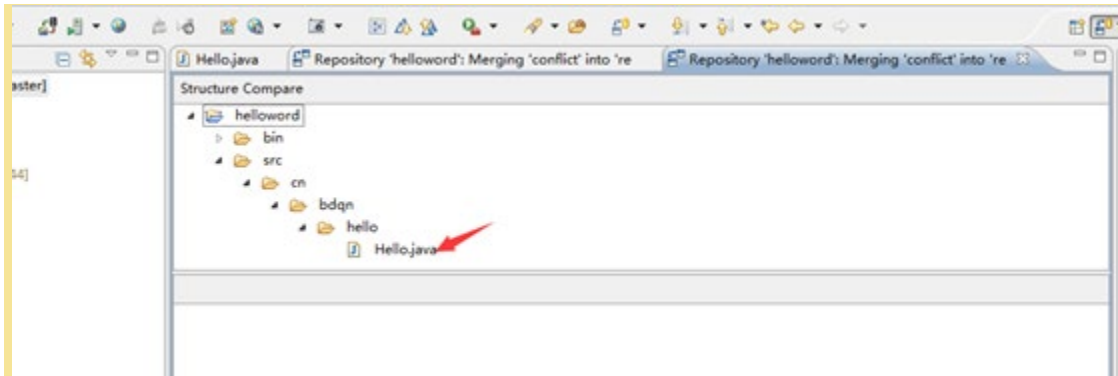




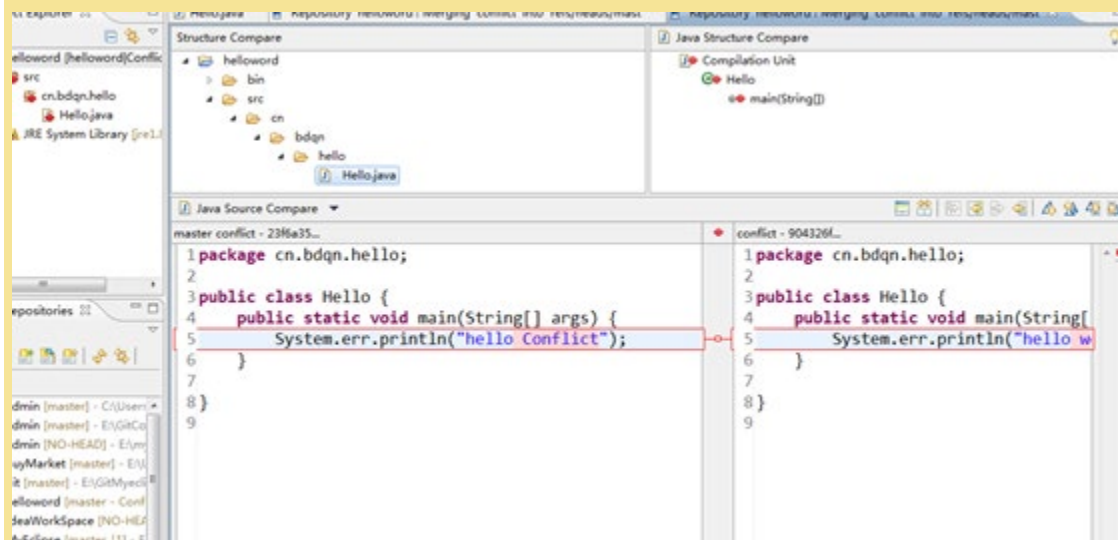
标红的文件代表有冲突

⑦ 查看冲突的代码



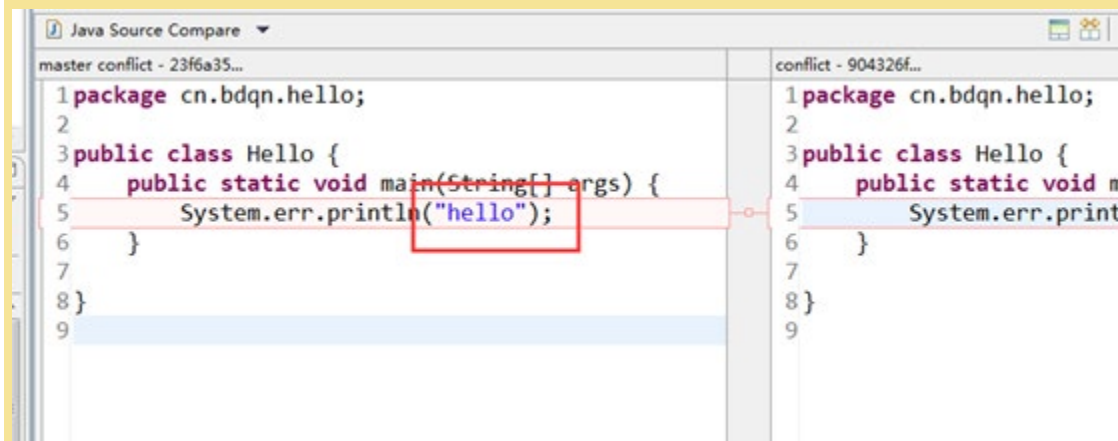


双击



红色框起来的代码就是有冲突的代码,再手动进行修改

假设我们现在修改左侧的代码为



再次提交,然后合并查看效果

注意:如果 git pull 失败,则需要添加参数

[remote "origin"]

url = <https://github.com/zhaoqinrong/test.git>

```
fetch = +refs/heads/*:refs/remotes/origin/*
```

```
[branch "master"]
```

```
remote = origin
```

```
merge = refs/heads/master
```

```
E:\pull>git pull ms master
remote: Counting objects: 29, done.
remote: Compressing objects: 100% (19/19), done.
remote: Total 29 (delta 2), reused 26 (delta 1), pack-reused 0
Unpacking objects: 100% (29/29), done.
From https://github.com/zhaqinrong/Mygit
 * branch            master      -> FETCH_HEAD
 * [new branch]      master      -> ms/master
```

posted @ 2017-11-05 17:00 懒人写博客 阅读(...) 评论(...) 编辑 收藏
努力加载评论中...

[刷新评论](#)[刷新页面](#)[返回顶部](#)