# Git

# Git简介

## 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gihub

#### 1.1.2.Git的特点

①Git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

⑥Git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

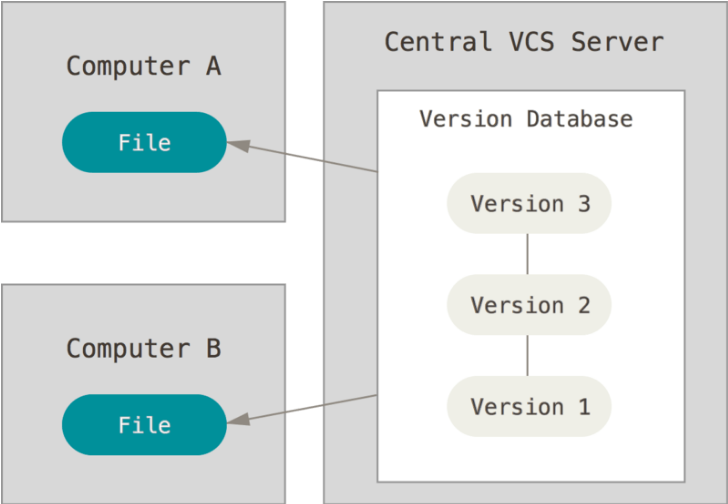
同时修改C代码,并分别进行提交

服务器(托管着我们的项目)

#### 1.2.2传统的集中式版本控制

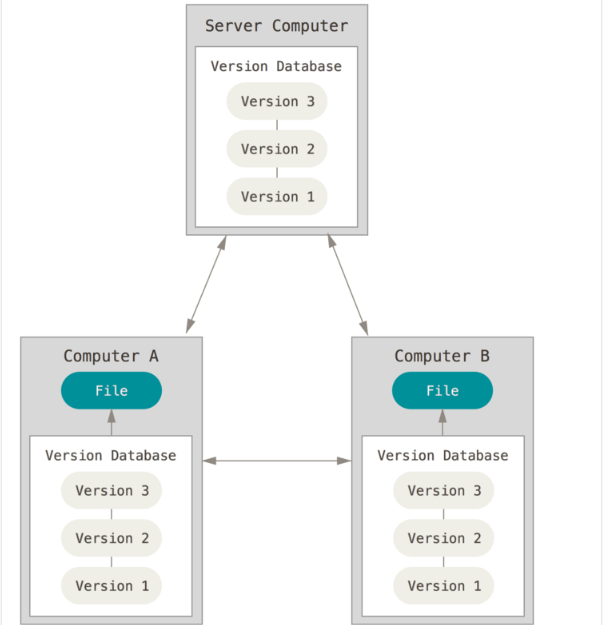
集中式版本控制系统（Centralized Version Control Systems，简称 CVCS），版本库是集中存放在中央服务器的，而干活的时候，用的都是自己的电脑，所以要先从中央服务器取得最新的版本，然后开始干活，干完活了，再把自己的活推送给中央服务器

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



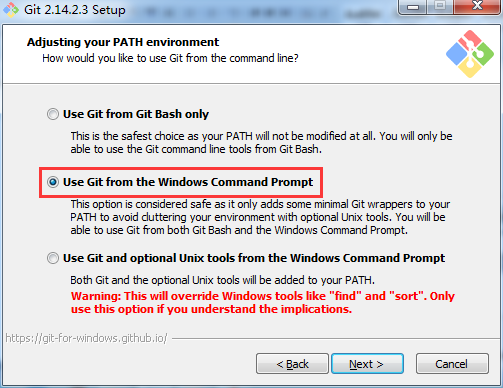
## 1.3Git的安装

①下载软件<https://git-scm.com/>官网地址

②进行安装

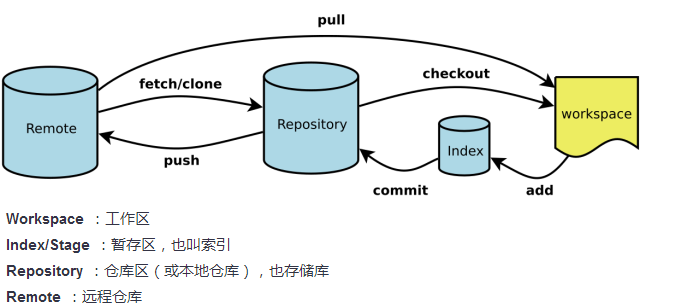
最重要的一步,其它可以直接走默认

选择这一步可以直接将我们的git命令,添加到系统变量中



# 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**

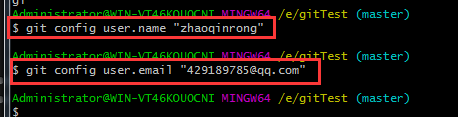
#### 2.1.1配置当前项目中的用户信息(局部)

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**

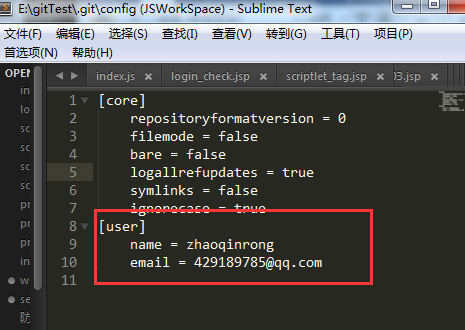
1.配置用户名:Git config user.name “用户名” 用来区分谁开发的代码

2.配置邮箱信息:git config uer.eamil “邮箱” 可能是多个国家的开发者,便于联系开发者



配置以后会可以在隐藏的.git文件夹的config文件中查看到,

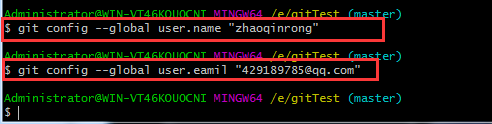
也可以使用git config –list进行查看



#### 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 uer.eamil “邮箱” 可能是多个国家的开发者,便于联系开发者



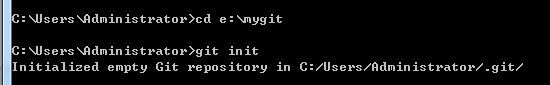
可以在C:\Users\Administrator\.gitconfig中查看

## 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



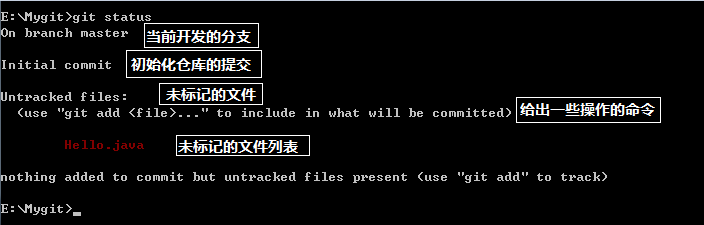
这时候会在当前文件夹下创建一个.git隐藏文件夹,.git文件夹是我们的仓库信息,一定不要修改

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

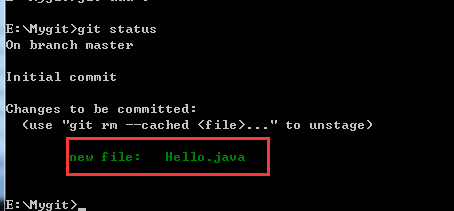
1. 查询仓库的状态:**git status**



1. 将文件加入到暂存区:**git add** 文件名



使用git status 再次查询出文件的状态



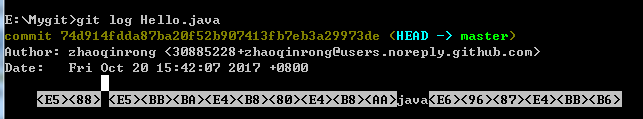
现在的文件并没有真正的提交到主分支上(主分支就是我们真正要运行的程序的所有代码).

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



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**

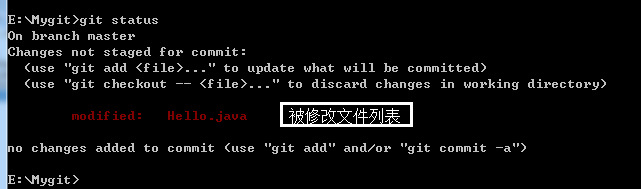
**或者使用简化命令 git commit –a –m”注释信息” 来将我们的文件添加到版本库**

## 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**

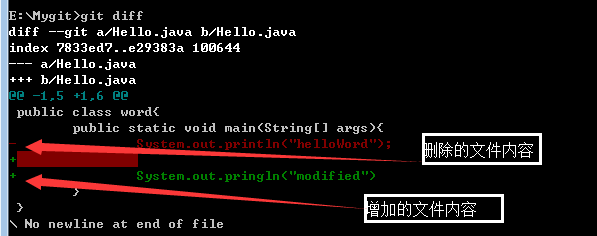


以上看到git的一个建议

|  |
| --- |
| (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



1. 使用git add 文件名,将文件加入到缓存
2. 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

HelloC.java

HelloD.java

HelloB.java

HEAD

工作区

版本库

暂存区

Master

**git add**

**git commit**

HelloA.java

**git checkout**

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 HARD~1 或使用git reset - - hard HARD^

#### 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 gid 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--

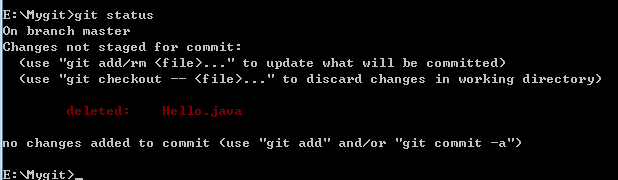
#### 2.7.2 已经执行git add 添加到了暂存区,但是没有执行git commit

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gid checkout - - Hello.java进行恢复

## 2.8删除文件

①.从磁盘上删除文件



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](mailto: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/zhaoqinrong/admin.git>

Git remote add origin+远程仓库地址

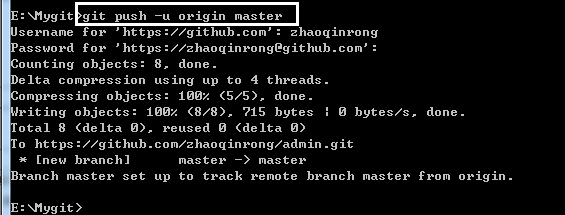


但是目前为止,远程仓库中没有我们本地仓库的内容

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



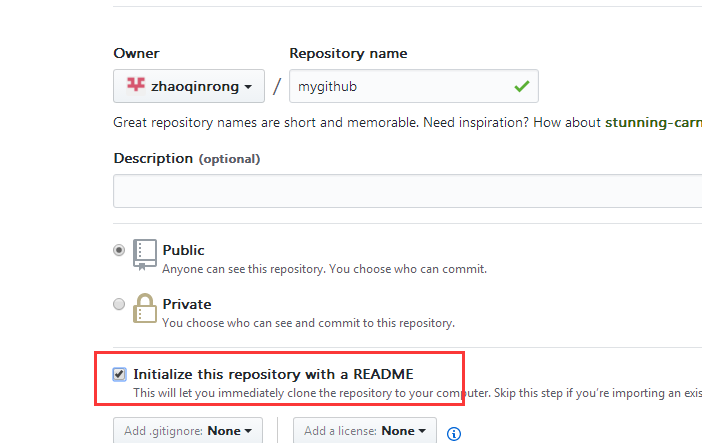
由于是第一次推送,而且推送的为master分支,就会使用”-u”的参数将远程master与本地master进行关联

1. 修改本地文件后,提交到本地版本库add 和commit命令
2. 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

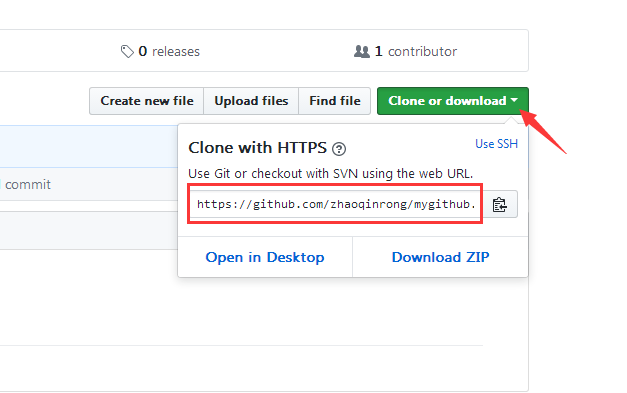
1. ,在github上建立仓库,并进行初始化



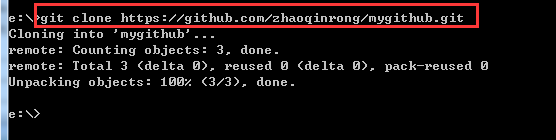
勾选**Initialize this repository with a README**初始化这个仓库

1. 克隆远程仓库

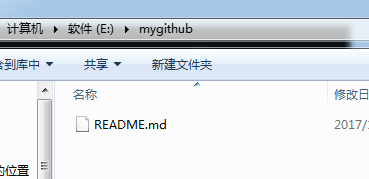
点击Clone or download,将仓库的地址复制下来



使用git clone +远程仓库地址,对远程仓库进行克隆



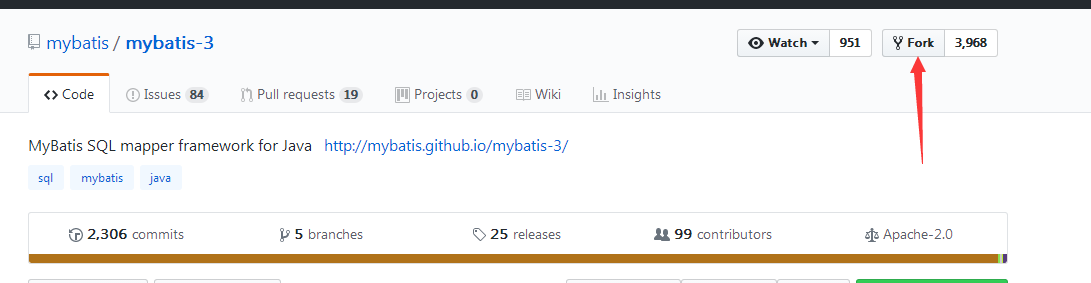
这样就克隆下来了



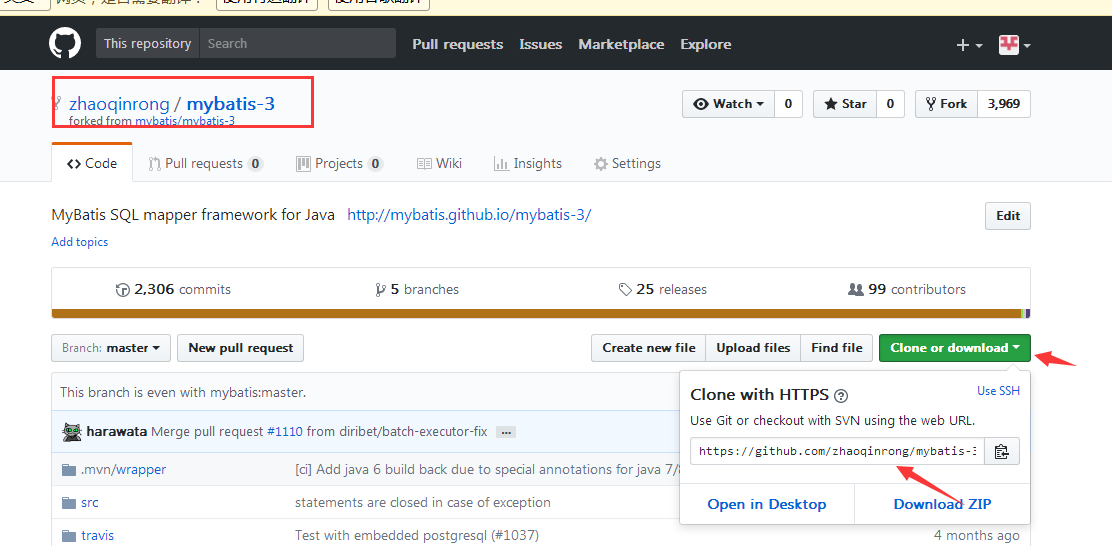
1. 在我们克隆的仓库中进行项目的开发,这里我们新建一个hello.java的文件 并使用add和commit命令进行本地版本库的提交
2. 使用**git push –u origin master** 进行推送 Master为分支名

#### 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

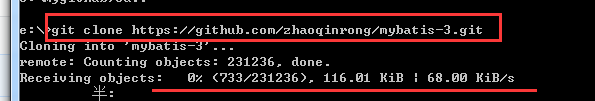


1. 复制自己仓库中这个项目的地址



1. 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/zhaoqinrong/Mygit**](https://github.com/zhaoqinrong/Mygit)

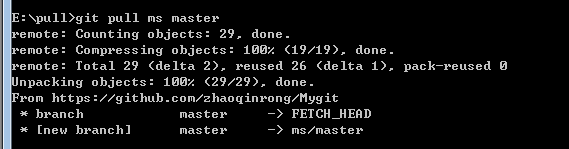
**可以使用命令git remote rm origin删除关联的远程仓库**

**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**

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



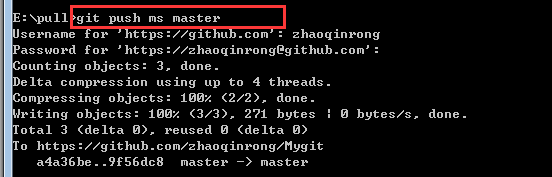
**如果直接使用git pull可以会出错,建议先git fetch到本地,然后使用git merge 合并**

#### 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

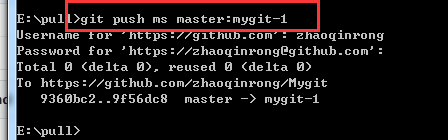


建议我们推送更新前都先使用pull获取远程仓库中的更新,然后在push推送

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

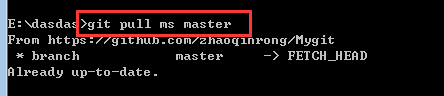


#### 3.7删除远程仓库中的文件

①先要将本地仓库和远程仓库进行关联



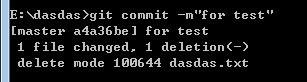
进入到我们的git命令行页面后，先将远程代码pull到本地，保持本地仓库跟远端仓库同步。



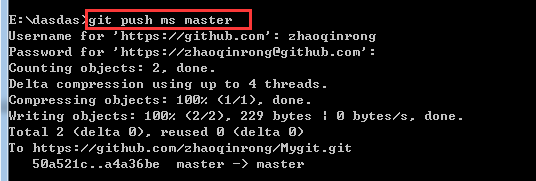
1. 使用**git rm 文件名** 删除 文件



1. ,进行提交(和数据库事务一样,提交后才会处理)



1. 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

Master

HEAD

1. 创建了新分支,HEAD指向了

Master

HEAD

新分支

1. 主分支和子分支进行合并

Master

HEAD

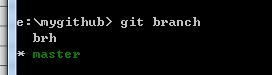
新分支

#### 4.2分支创建

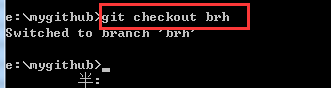
①创建分支:

使用命令 **git branch + 分支名**

**使用git branch 查看当前项目的分支**



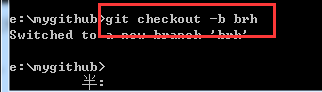
1. 切换分支 使用命令 **git checkout +分支名**



1. 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**



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/zhaoqinrong/mygithub.git>

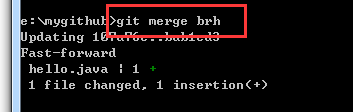
然后分别推送分支>git push origin master >git push origin brh

推送成功后可以在我们github上进行查看

#### 4.3合并分支

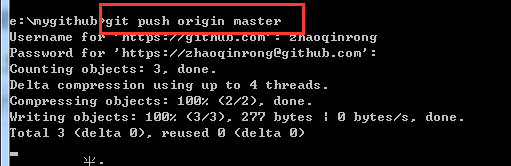
①切换回主分支**git checkout master**

1. 合并分支**git merge brh(子分支名)**



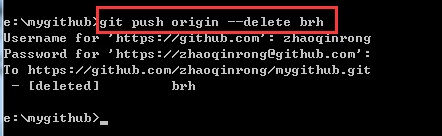
③,删除子分支 **>git branch -d brh**

1. 推送到远程仓库 **git push origin master**



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)**



#### 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>

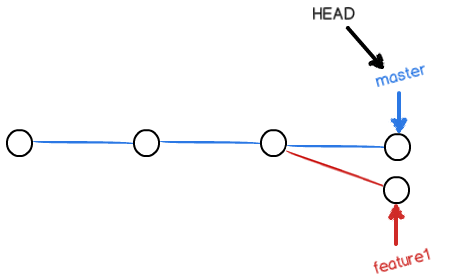
#### 4.4 解决代码冲突

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

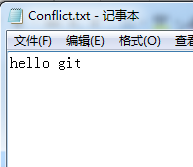


现在我们来模拟一下代码冲突,并试图解决

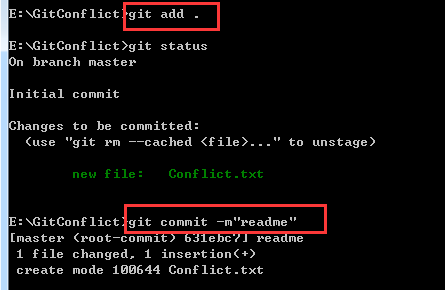
1. 在E盘新建文件夹 GitConflict,并初始化仓库git init



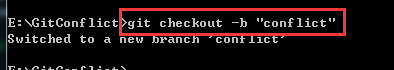
1. 在工作区创建名为Conflict.txt的文件,并写上hello git



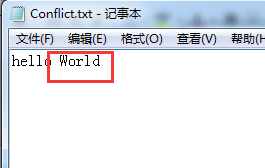
1. 把文件提交到本地仓库



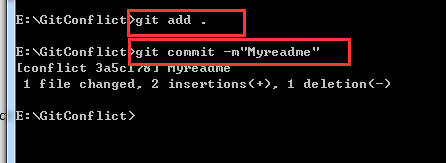
1. 创建新分支,名为Conflict并切换



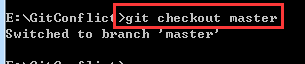
1. 在conflict分支下对Conflict.txt文件进行修改



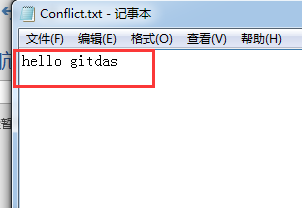
1. 然后在conflict分支下进行提交



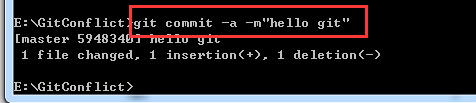
⑦,切换到主分支



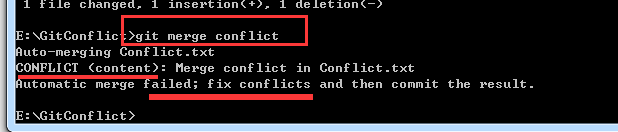
1. 再次对文件进行修改



1. 提交修改后的文件

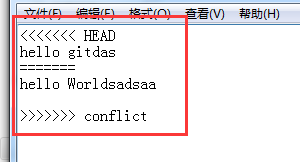


1. 和分支conflict进行合并

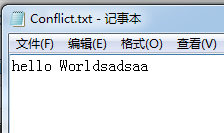


文件报错,有冲突

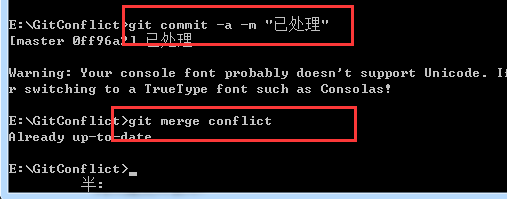
然后我们可以打开仓库中的Conflict.txt文件进行查看



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



再次提交并合并



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时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>

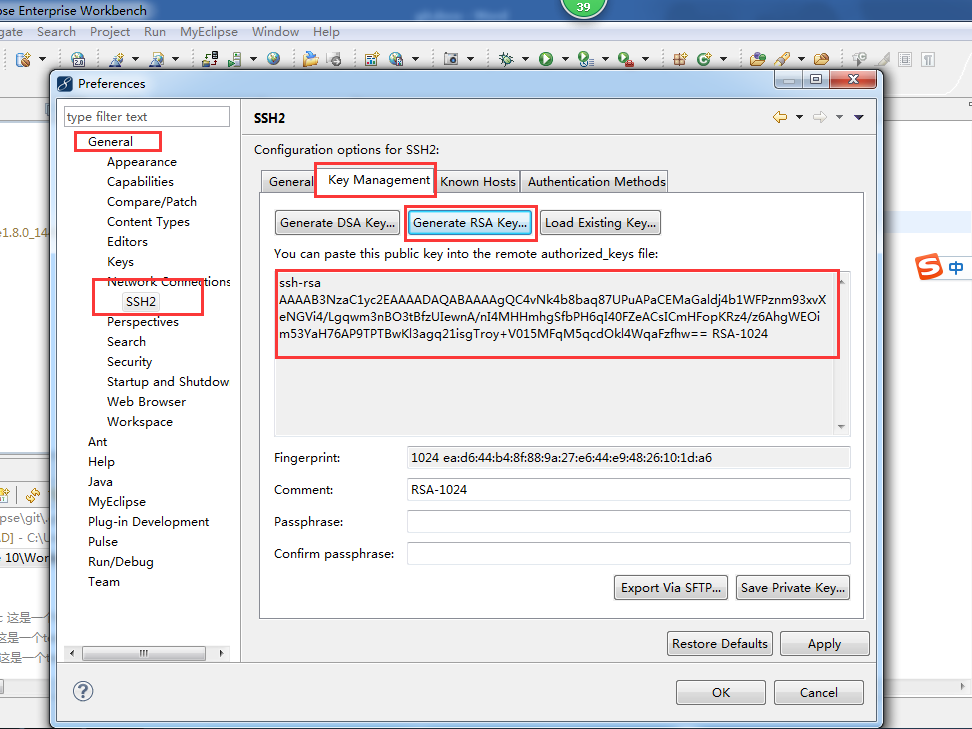
删除标签 git tag -d<tagname>

# 5 eclipse中使用git

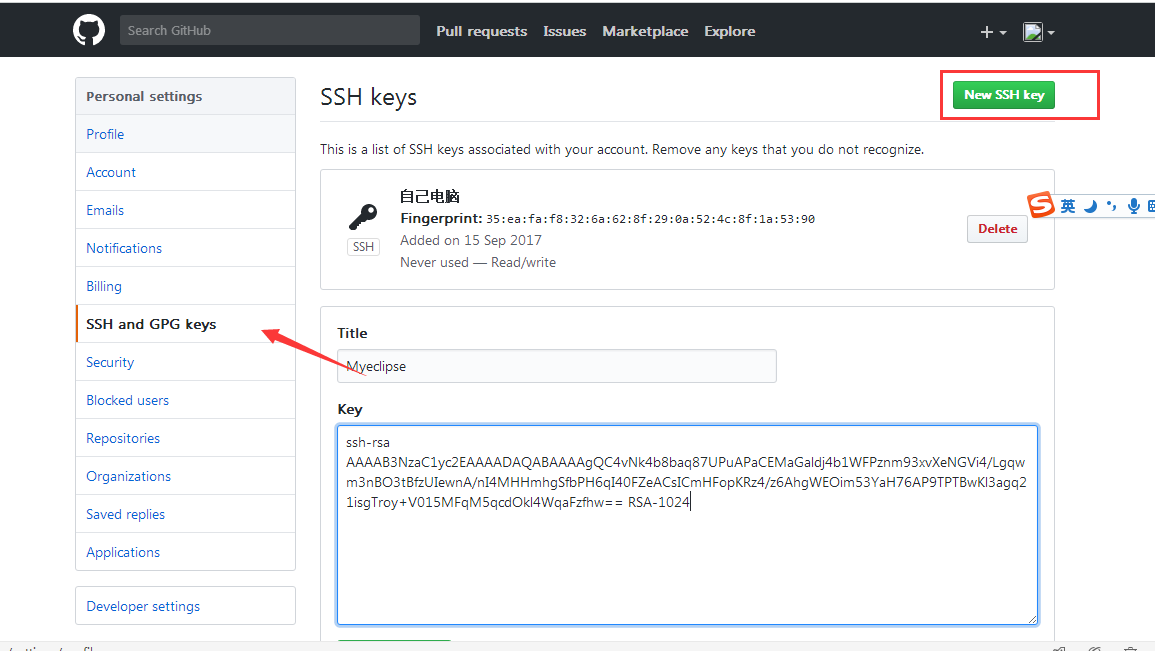
## 5.1用eclipse将项目保存到github中

1首先注册github

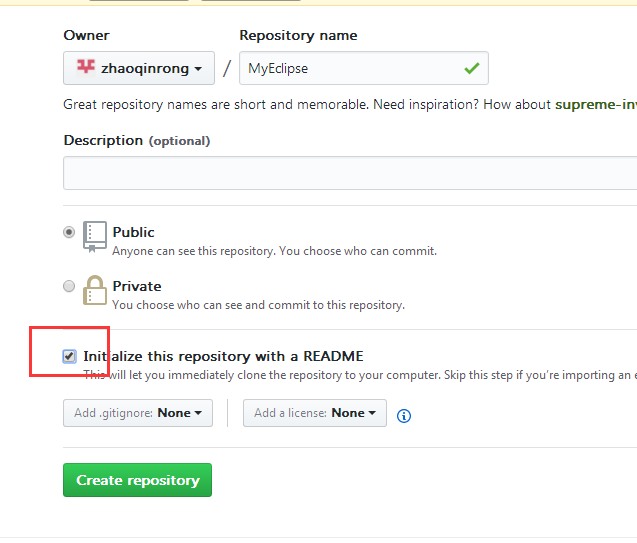
2.使用eclipse生成一个SSHkey



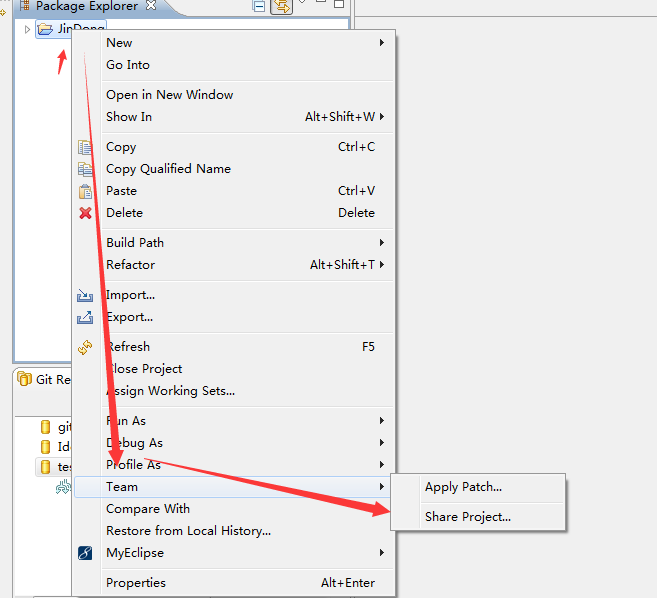
3.在github上对进行添加

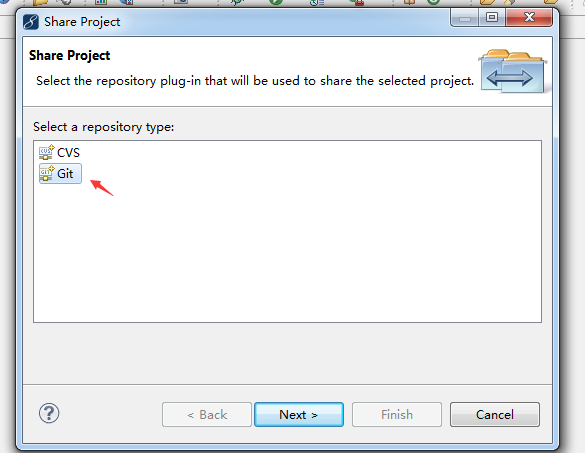


4.在github上创建一个新的仓库,并对仓库进行初始化

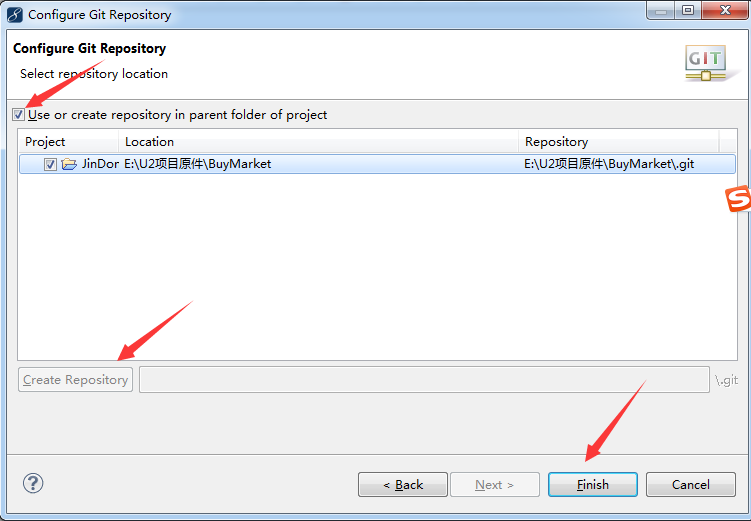


5.将我们的项目发布到刚刚创建的仓库中

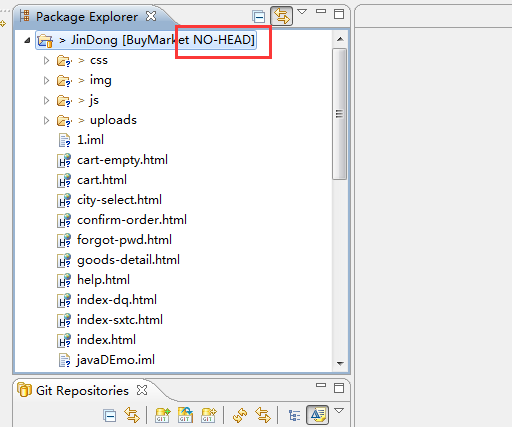




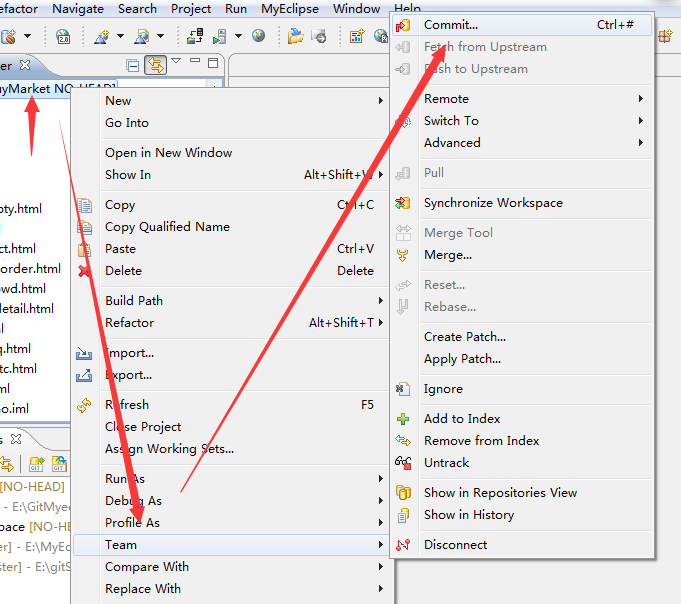
因为我们这里是新项目,所以选择创建一个repository



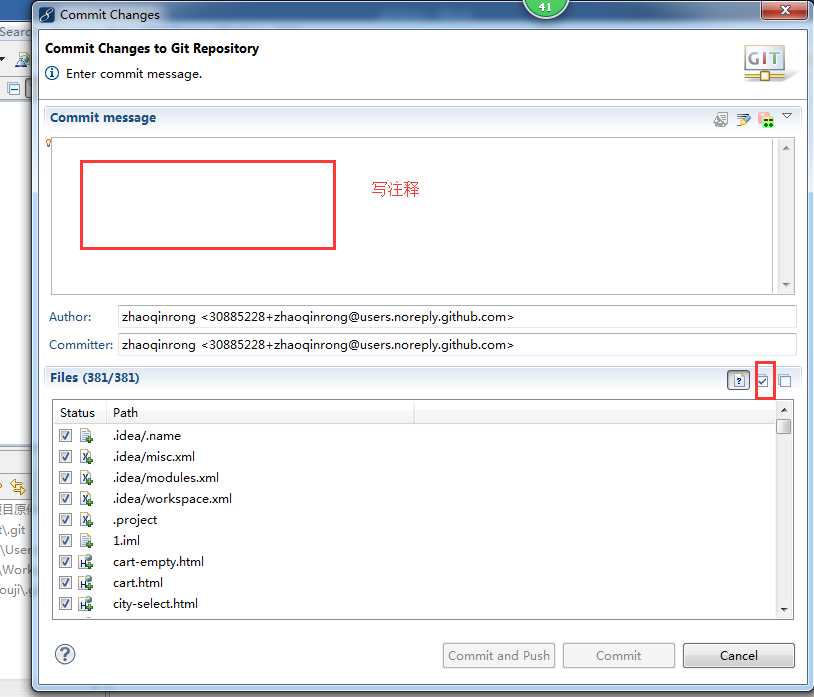
表示我们的项目还没有指针

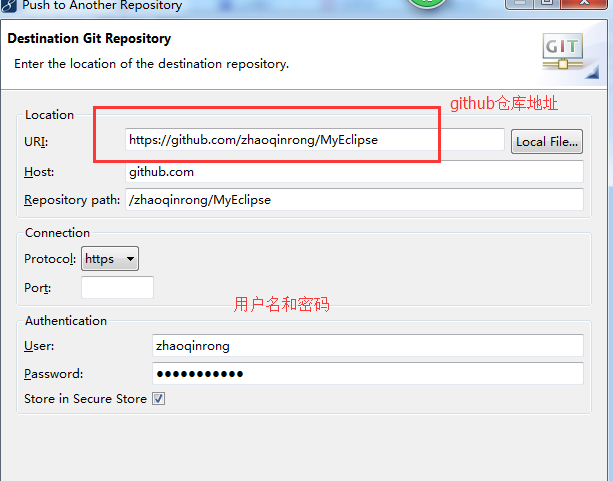


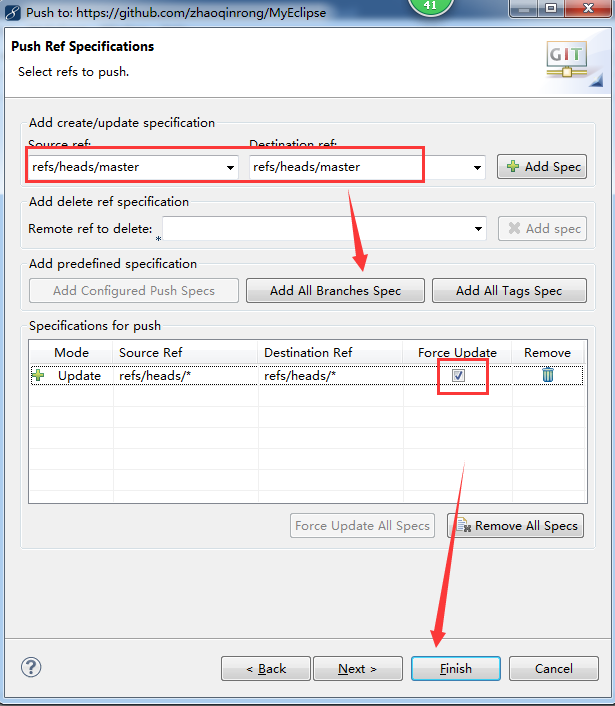
将项目提交到本地仓库



写上注释,选上所有文件进行commit and push

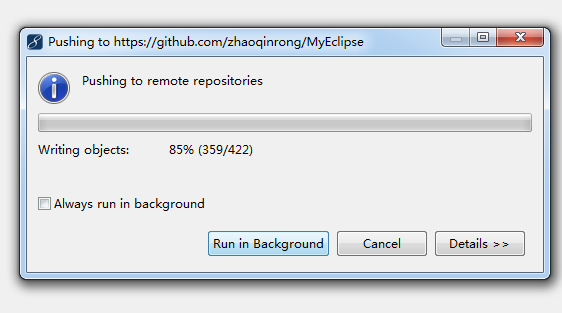


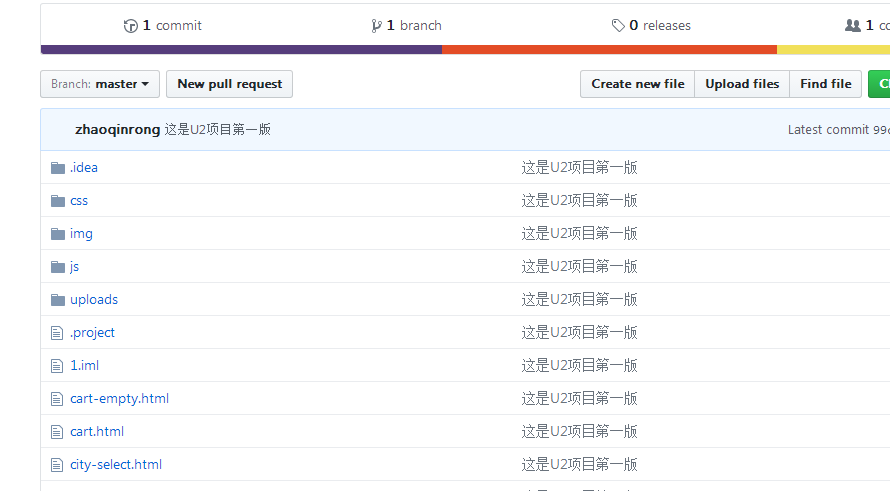




选择master分支

正在推送



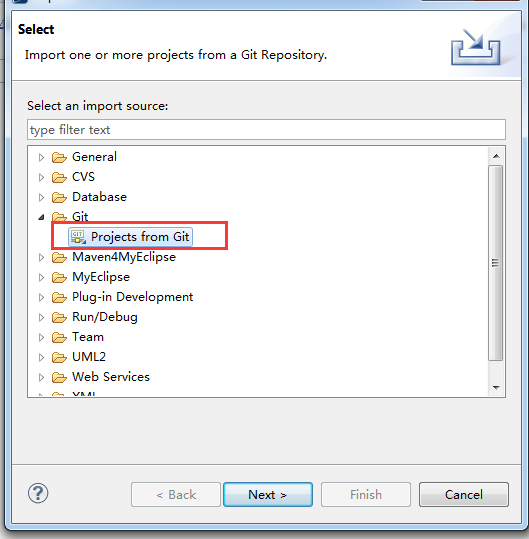


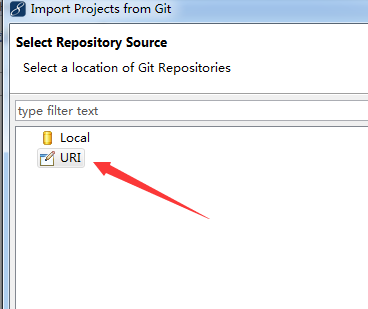
成功

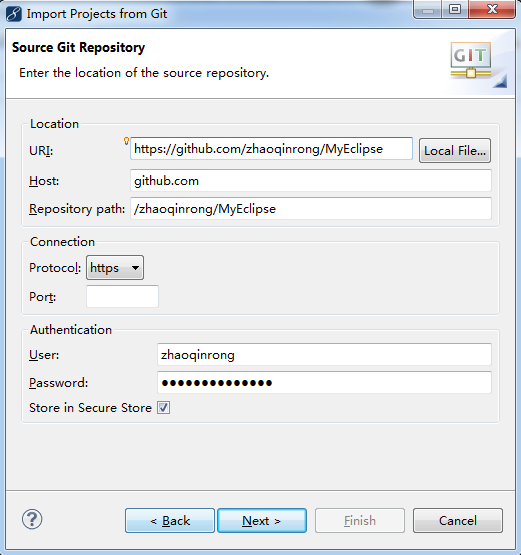
这是我们成功发布的项目,提交给github托管,当我们需要对项目进行修改的时候

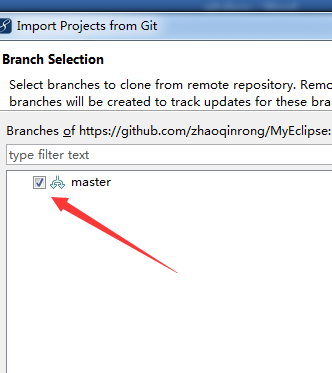
## 5.2我们可以将项目从github上导入到本地

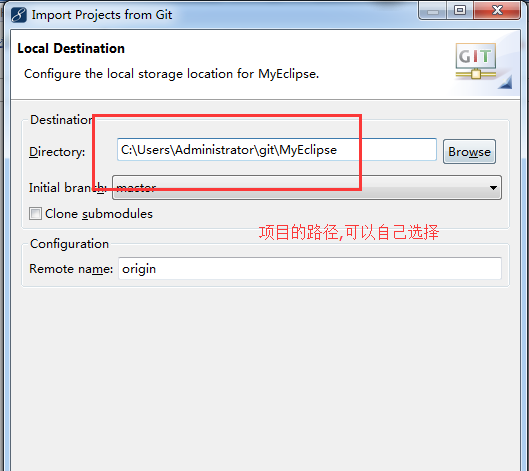
打开MyEclipse🡪file🡪import

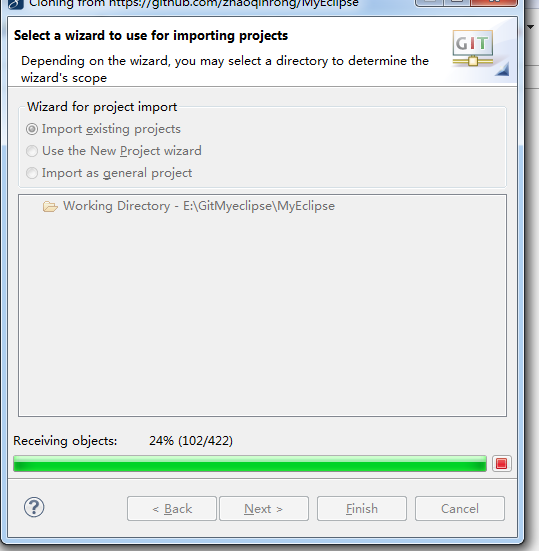




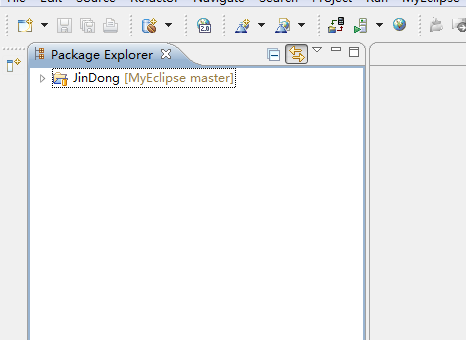






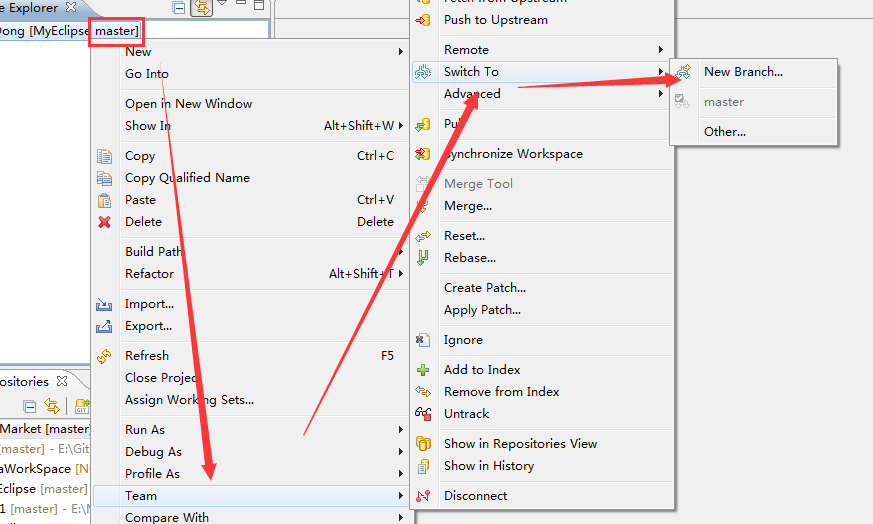


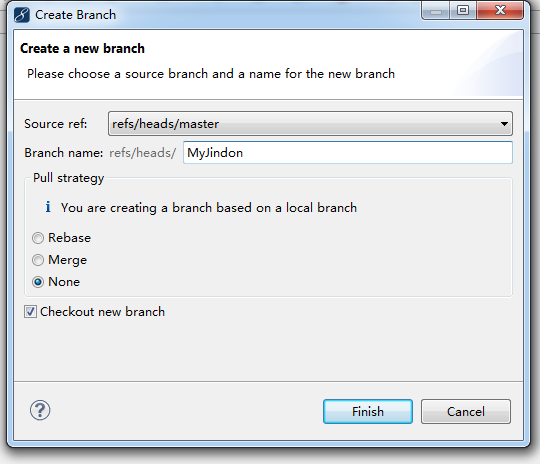
正在导入

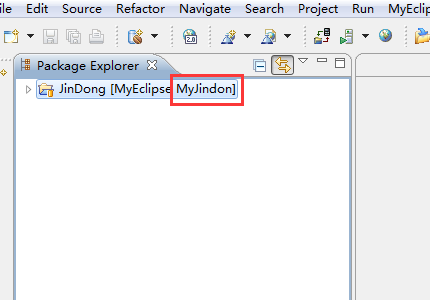


导入完成,然后我们在这个项目上进行开发,不会影响到我们仓库中的项目的完整性

目前是master分支,我们需要新建分支进行我们的开发



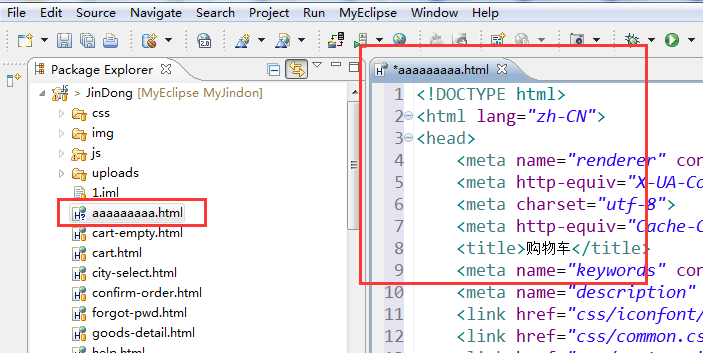




新建分支后,自动切换分支

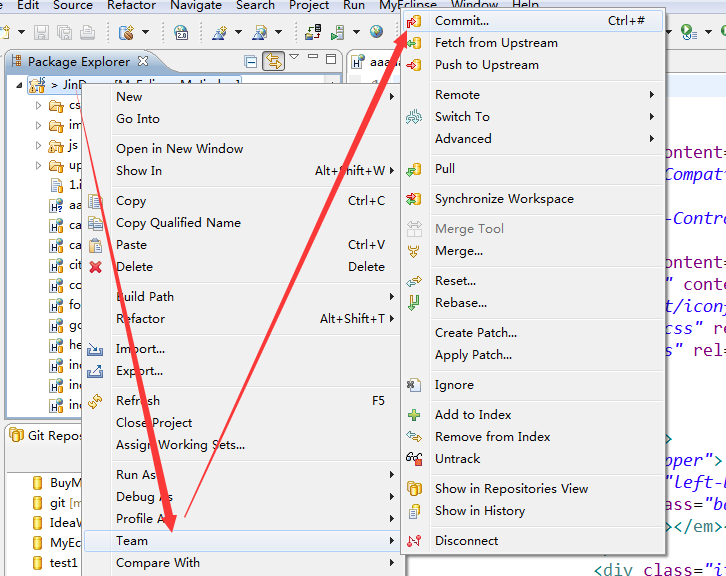
## 5.3在分支上进行开发,并git到github分支上

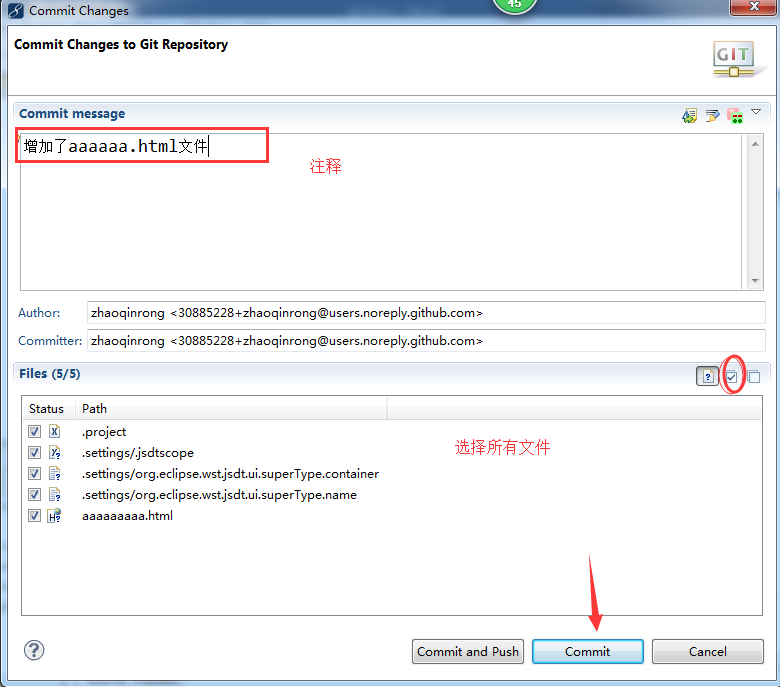
1.现在我们已经切换到了上一节创建的分支上,现在我们在我们之前的项目上做一点修改



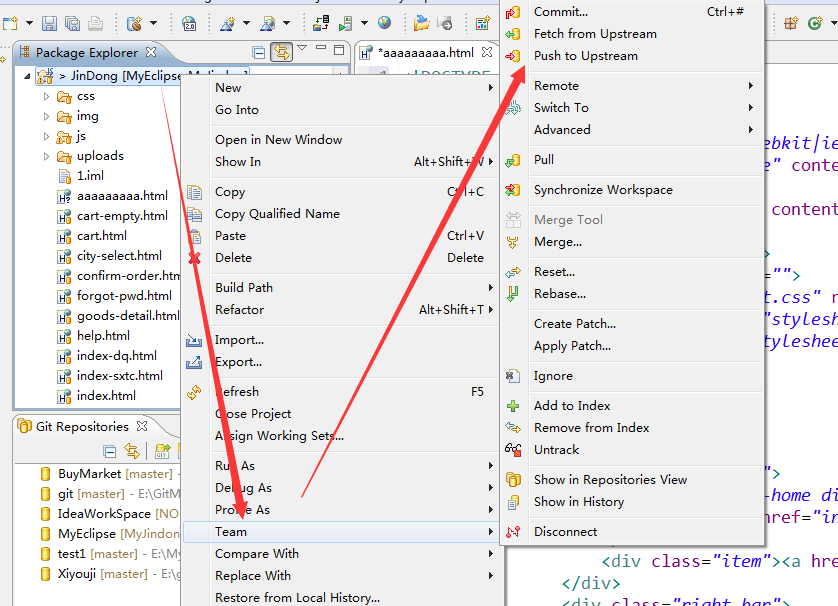
这是我在MyJindong分支上对项目进行的一个修改,新增了一个aaaaa.html的文件

2.先将我们修改的项目git到本地的仓库进行一次commit



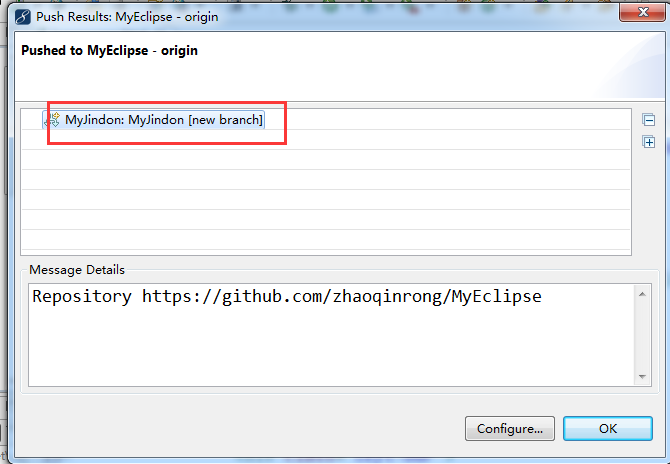


2. 将更新推送到远程仓库,并自动创建分支

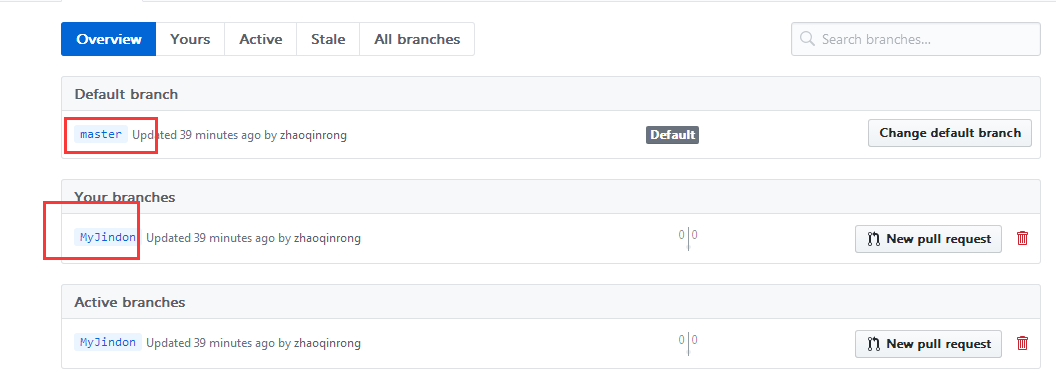


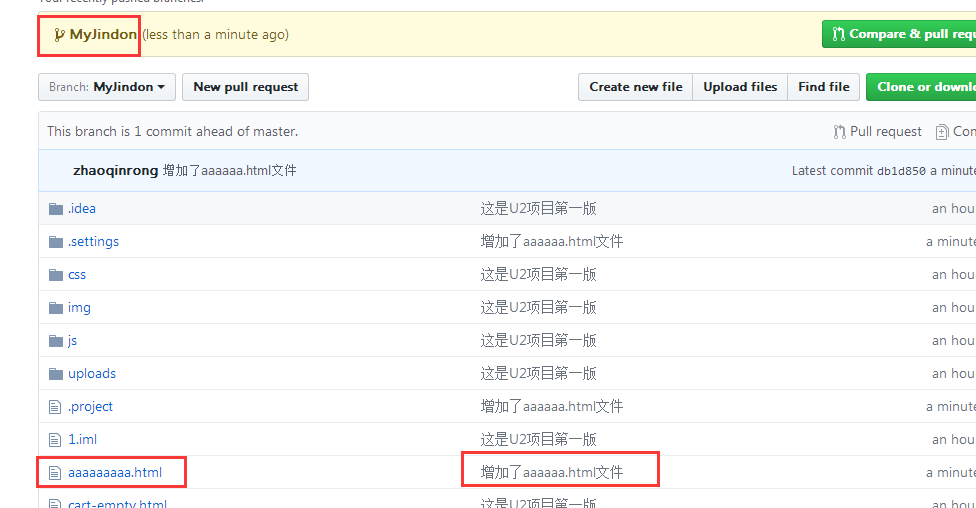
这是检查本地和远程仓库的差异:

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



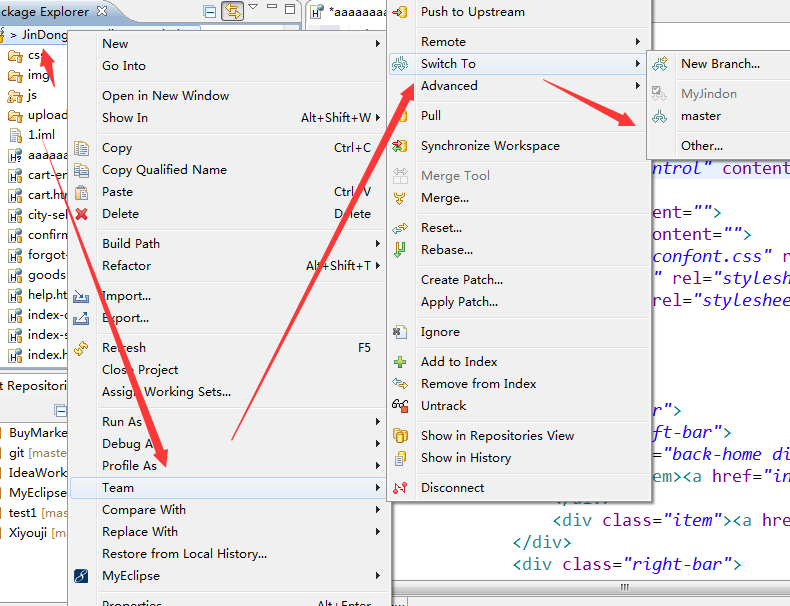


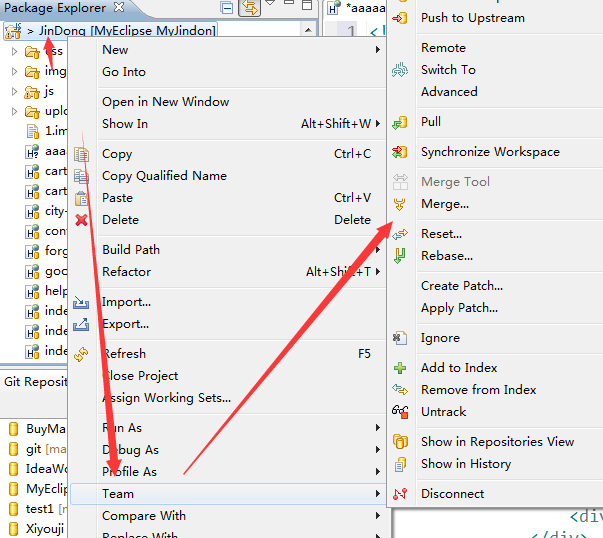
## 5.4将我们分支开发的项目和master分支的进行合并

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

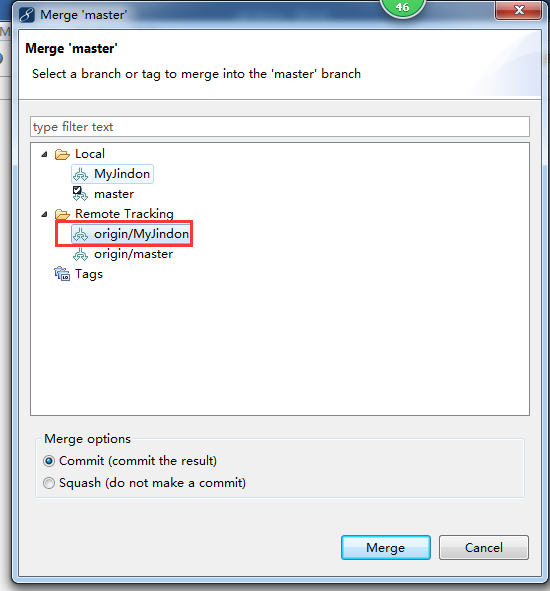
1. 首先我们要确定我们要与哪个分支合并,然后将我们的项目切换到哪个分支下,然后使用Merge进行合并

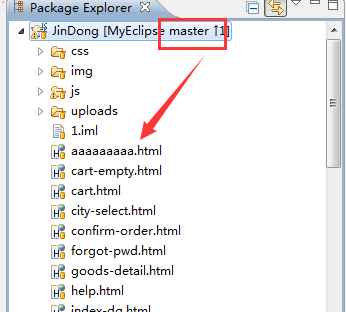
我们这里要与主分支合并,首先切换到主分支,看主分支是否有更新,然后我们合并到本地项目



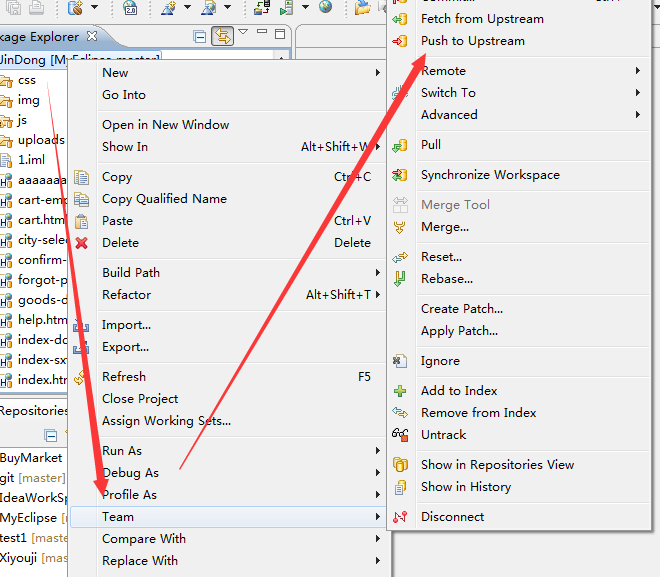


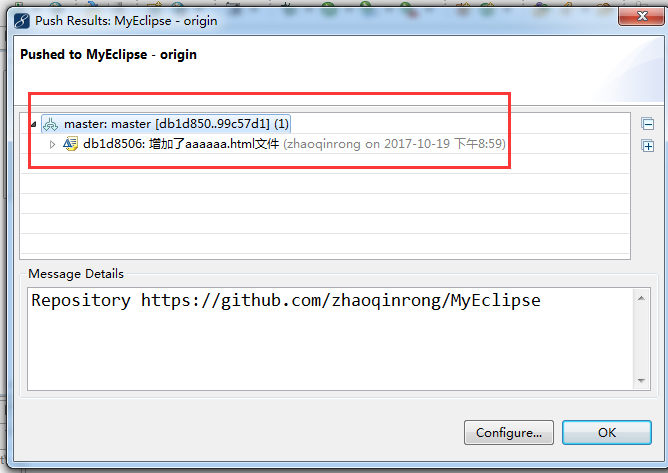
选择我们修改的项目进行本地合并

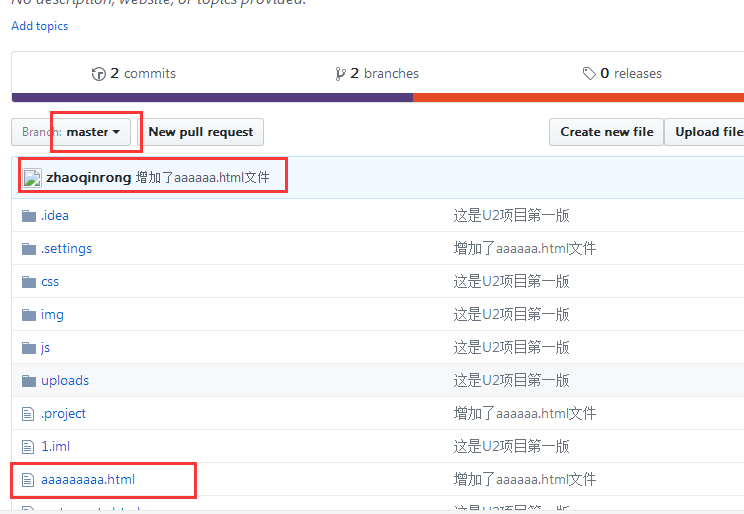


合并后的结果

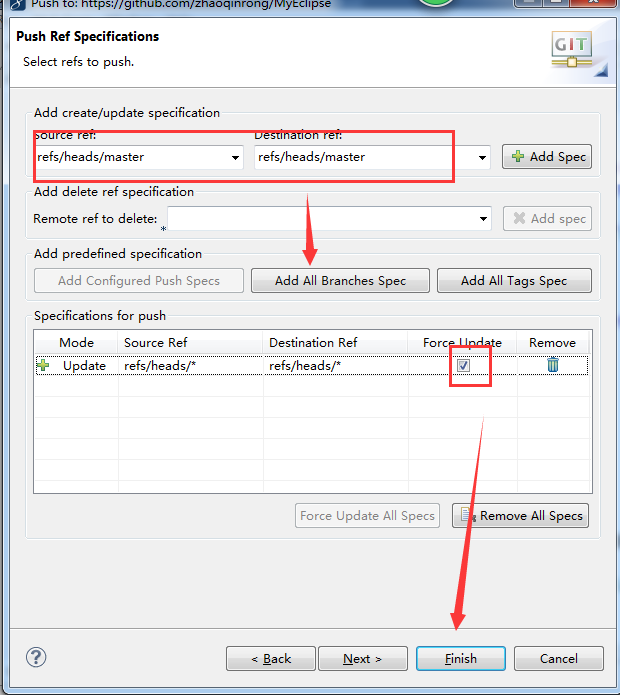
1. 然后选择push to upstream进行检查,是否有更新需要上传到我们的远程服务器



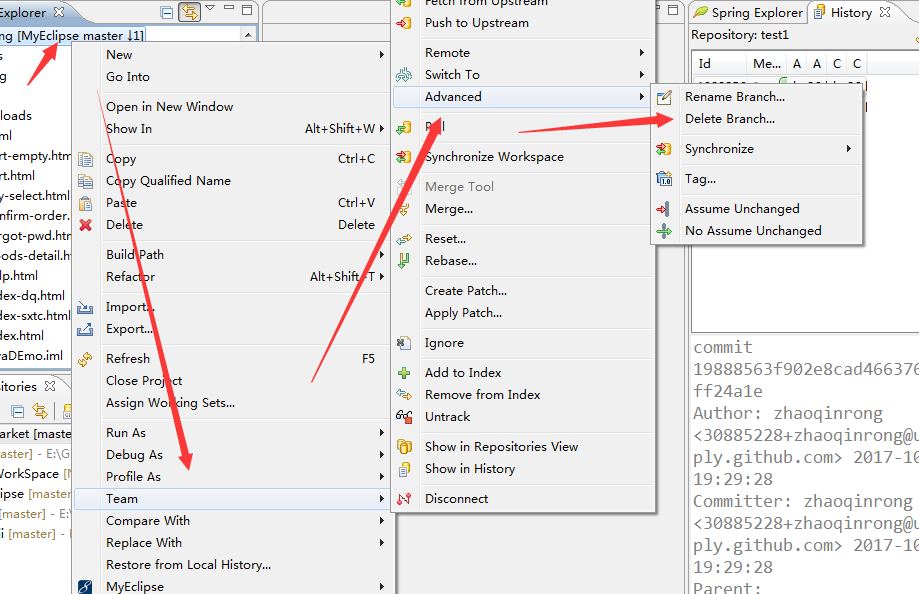


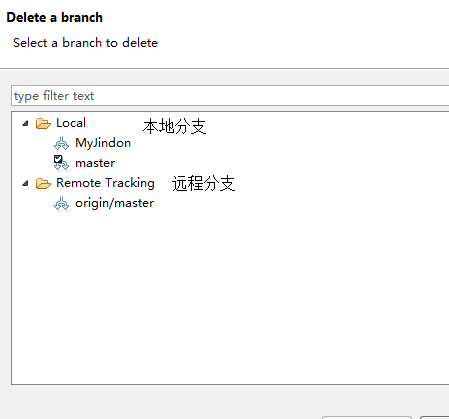


这是合并到远程仓库的结果.成功



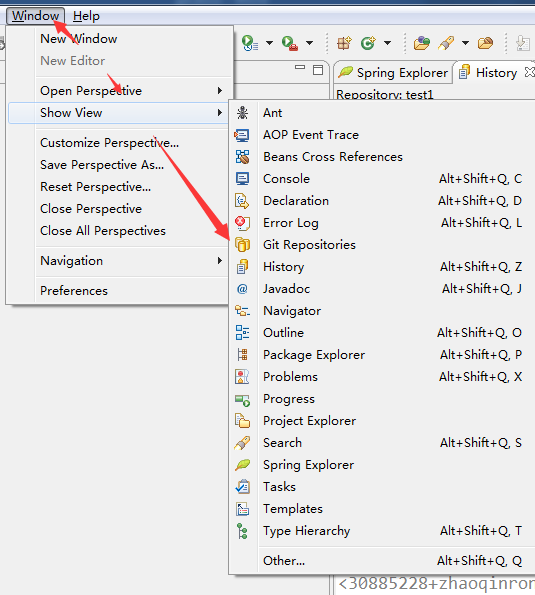
## 5.5删除本地分支和远程分支

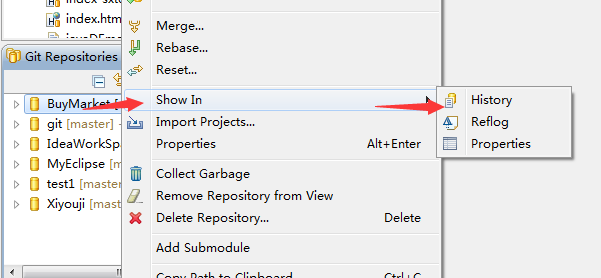


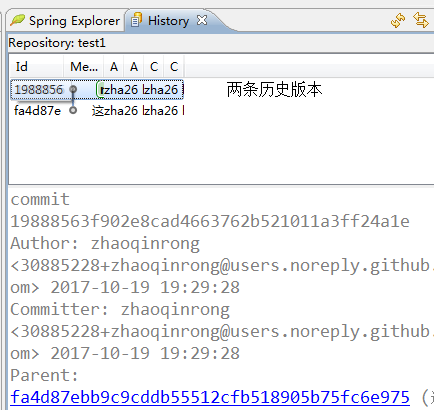


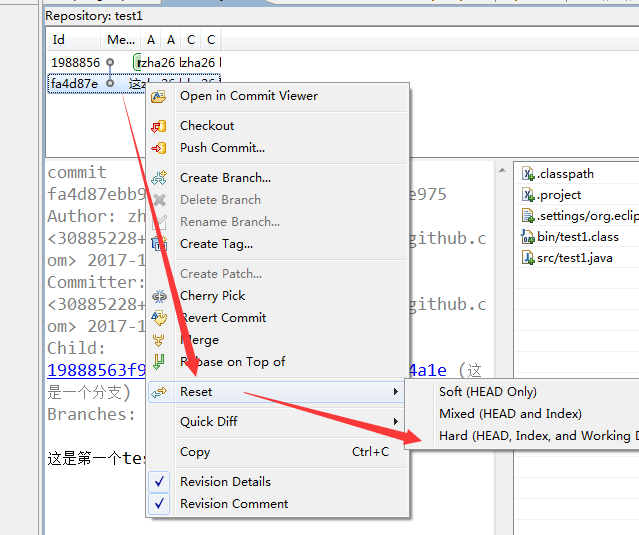
选择以后点击OK即可

## 5.6版本回退









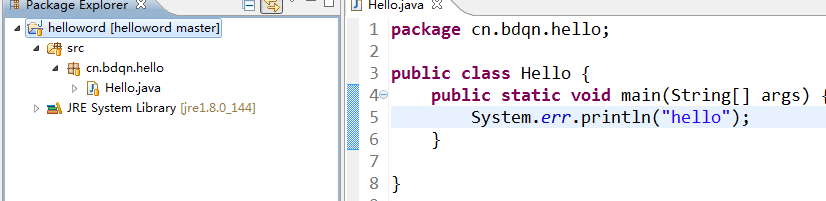
便可以进行版本的回退

## 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eclipe中

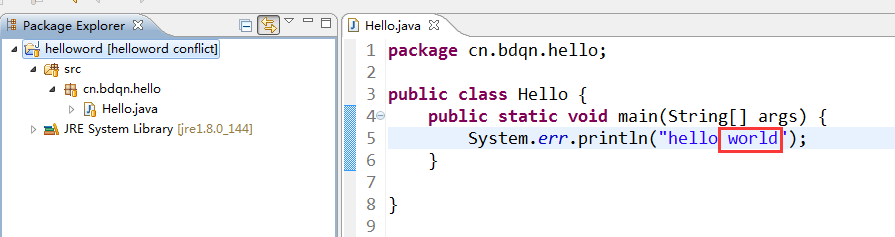
导入过程不过讲解



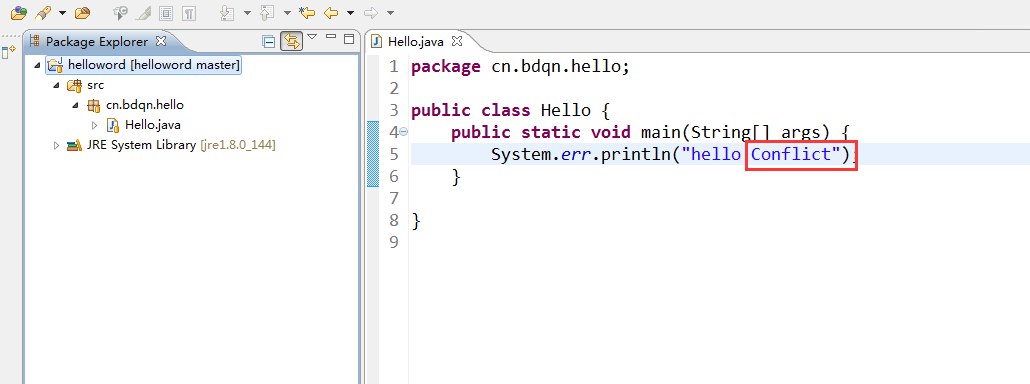
这是我们导入的项目,并进行本地版本库的提交

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

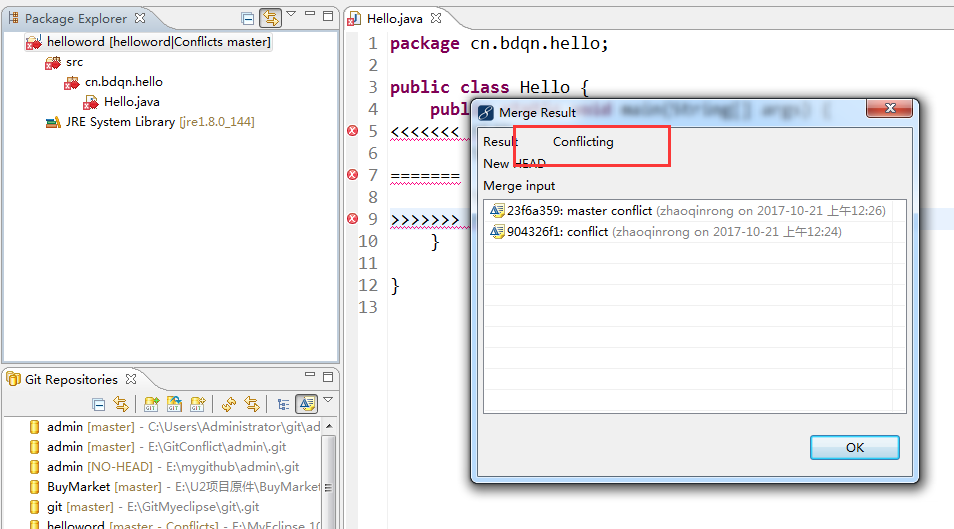


1. 切换到主分支master,对项目进行修改,并提交本地版本库

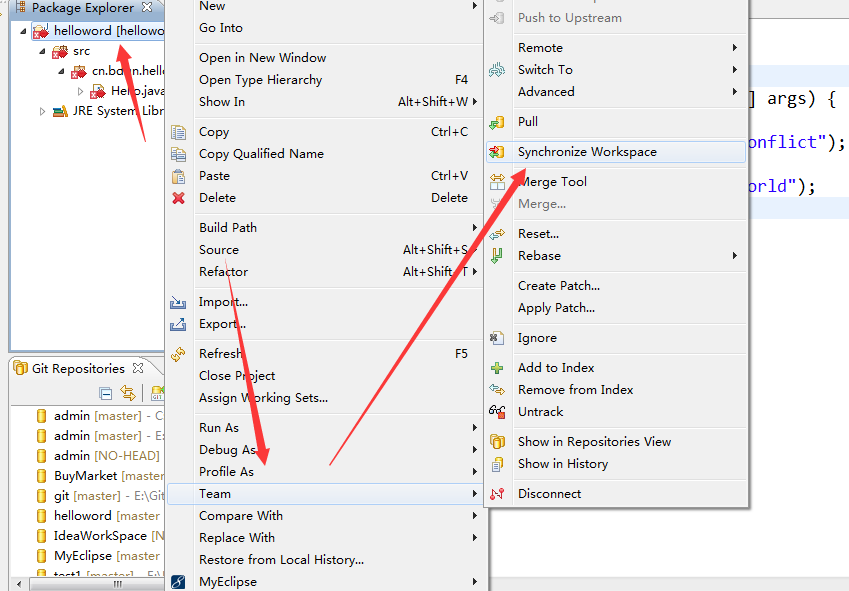


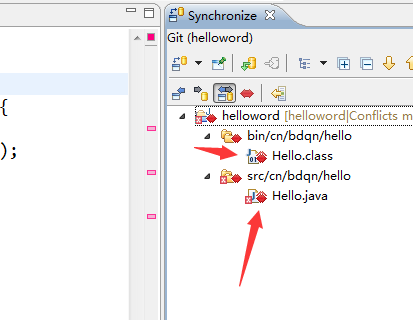
1. ,与conflict分支进行合并

便出现了我们期待的结果



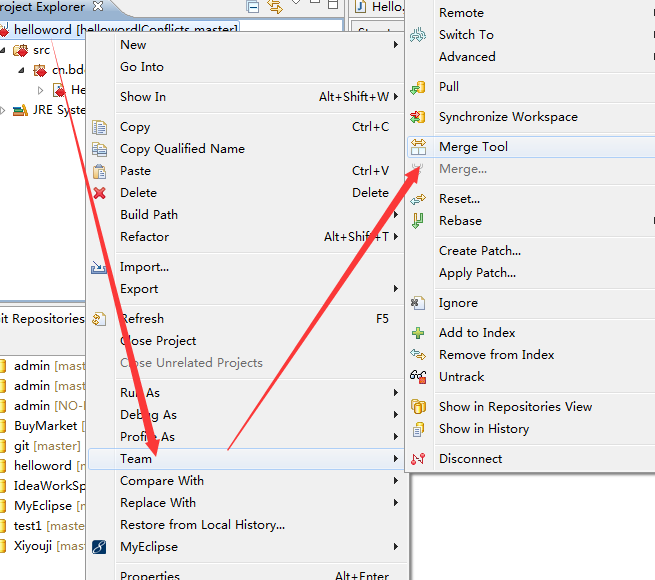
1. 查看冲突的代码

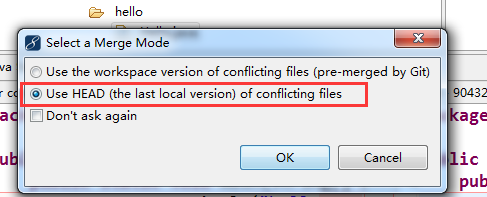


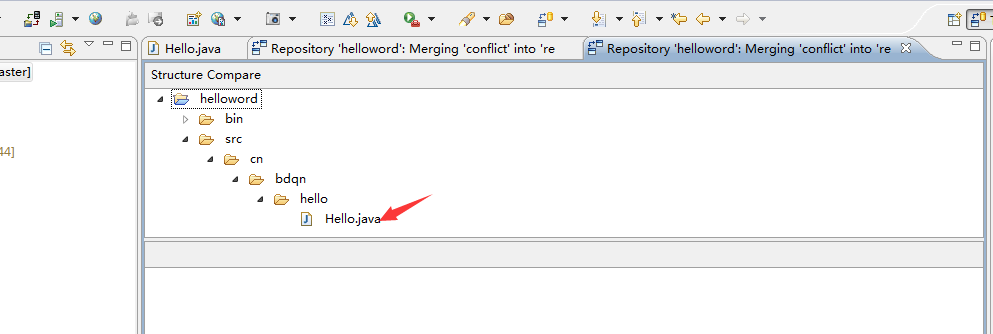


标红的文件代表有冲突

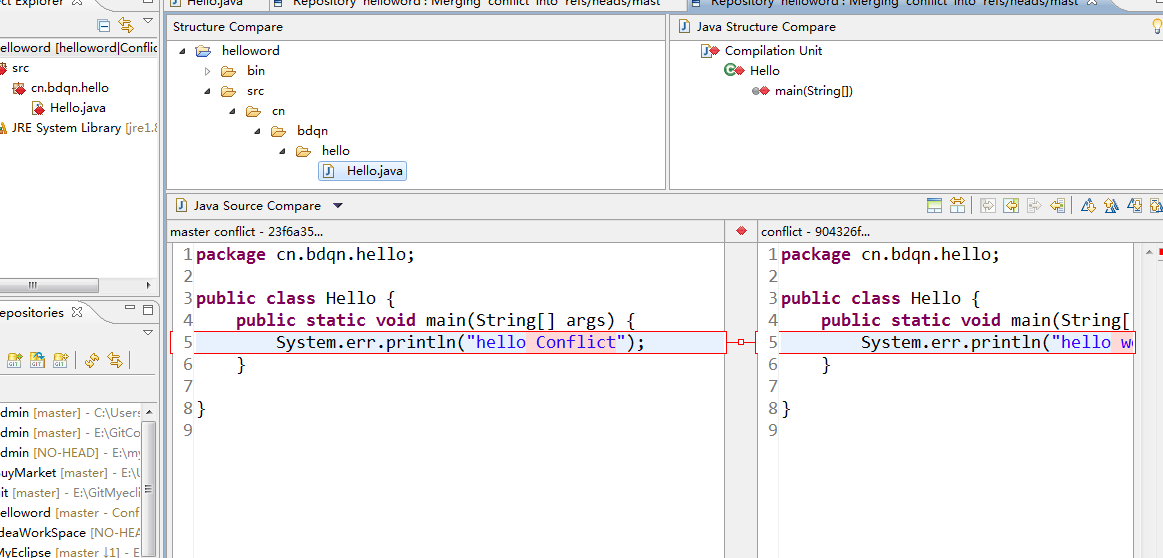
1. 查看冲突的代码





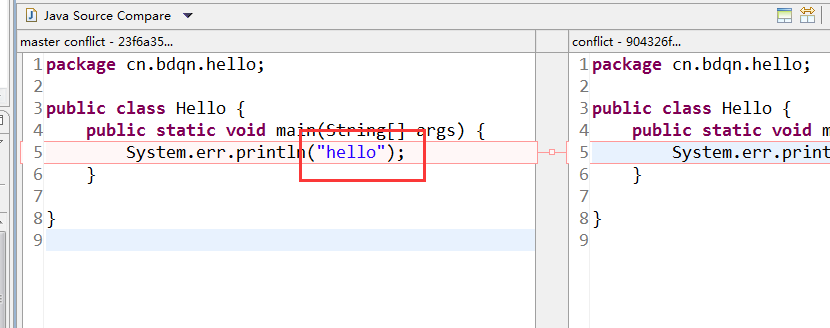


双击



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