一、安装EGIT插件

二、使用EGIT前的配置

三、新建GIT仓库

四、配置.gitignore文件

五、查看历史记录

六、推送远程仓库

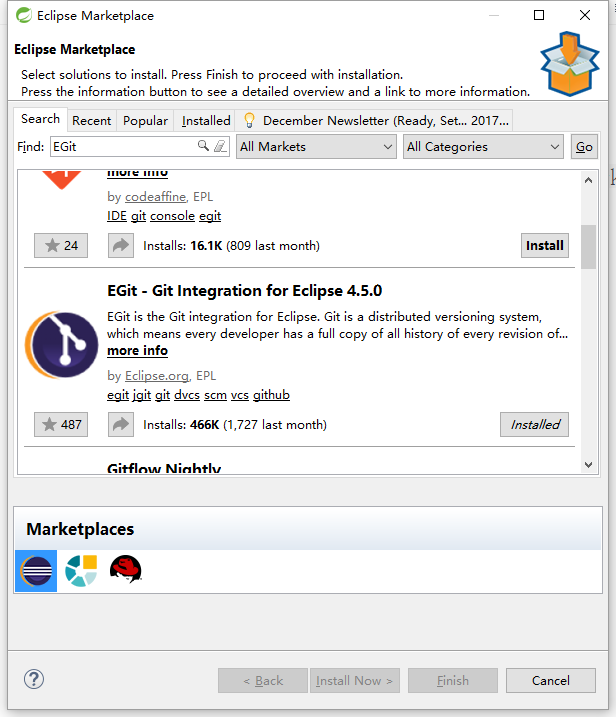
七、远程GIT仓库

八、解决推送冲突

九、重置功能

**一、 安装EGIT插件**

现在的eclipse一般都自带了，如果没有，使用Eclipse Marketplace，搜索EGit

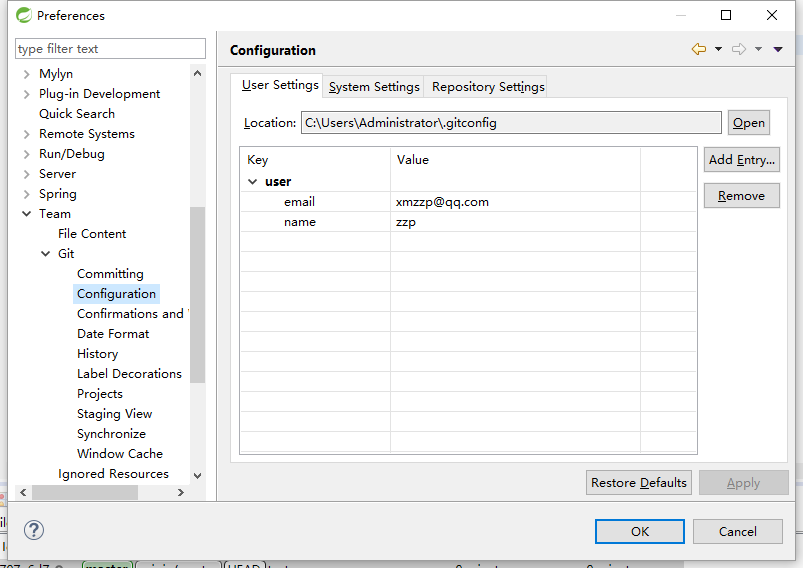


**二、 使用EGIT前的配置**

配置个人信息，最重要的是user.name和user.email

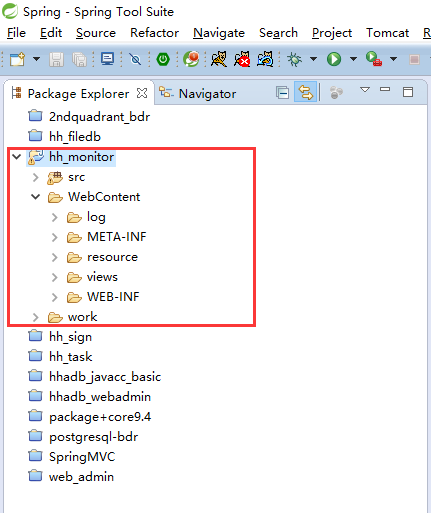
Preferences > Team > Git > Configuration

add Entry

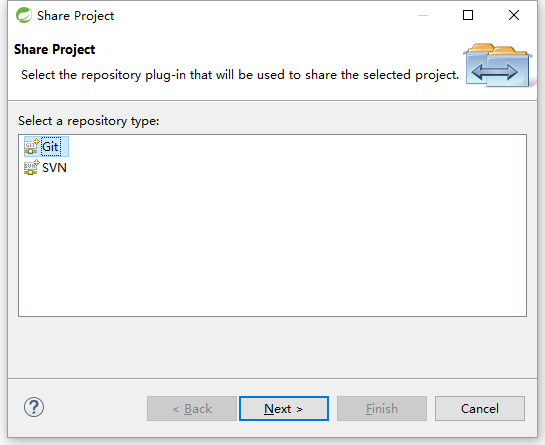


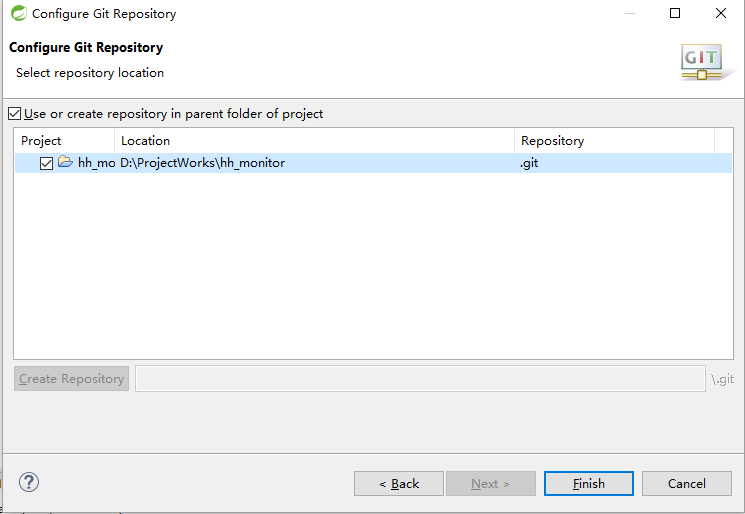
**三、 新建GIT仓库**

新建hh\_monitor项目



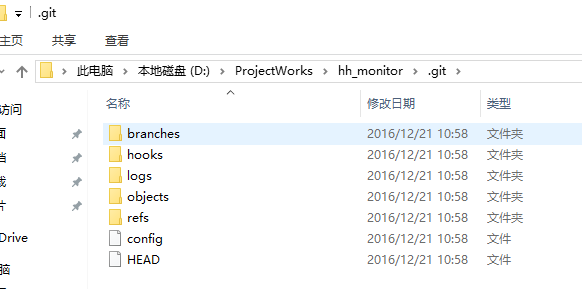
File > Team > Share Project 选择GIT



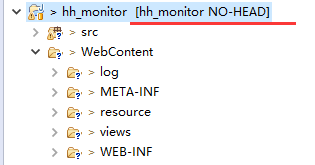


创建仓库后，在$ProjectWorks\hh\_monitor目录下的.git文件夹，就是git的仓库地址。

和CVS、SVN不同，GIT不会在每一个目录下建立版本控制文件夹，仅在根目录下建立仓库。



同时，eclipse中的project也建立git版本控制，此时未创建分支，处于NO-HEAD状态

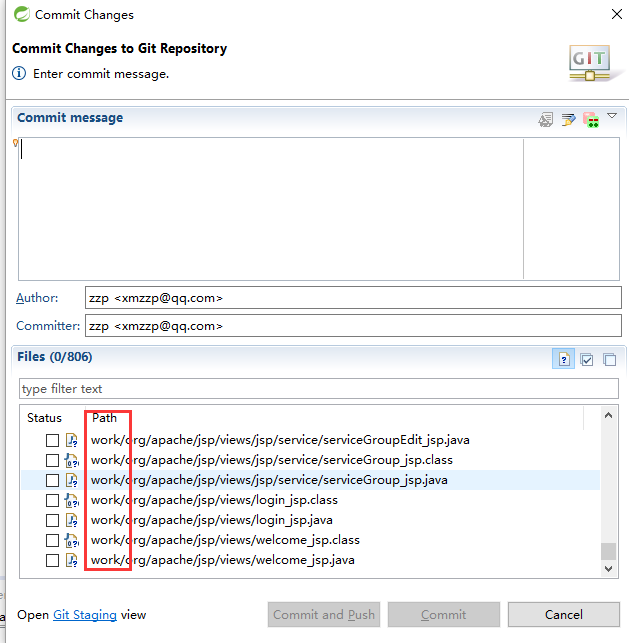


文件夹中的符号”?”表示此文件夹处于untracked状态，这样就成功创建GIT仓库。

**四、 配置.gitignore文件**

配置.gitignore，排除不需要提交的文件。

此时我们尝试做一次提交,Team -> Commit

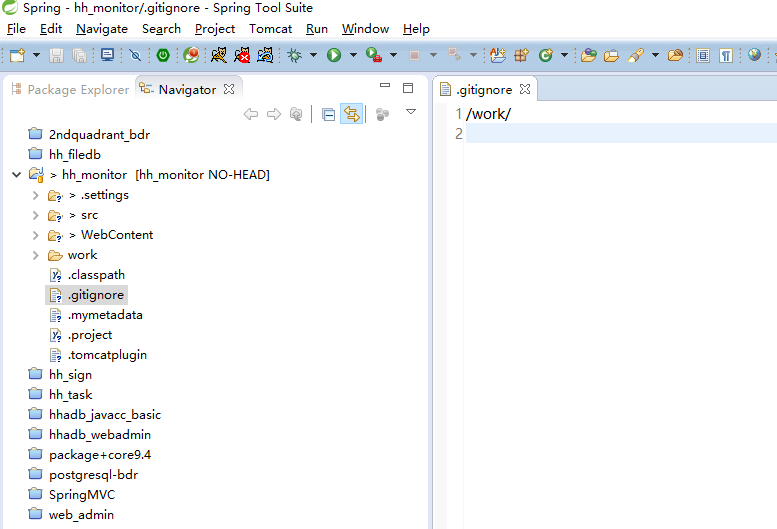


如上图所示，Author和Committer会默认为Git配置的用户信息。

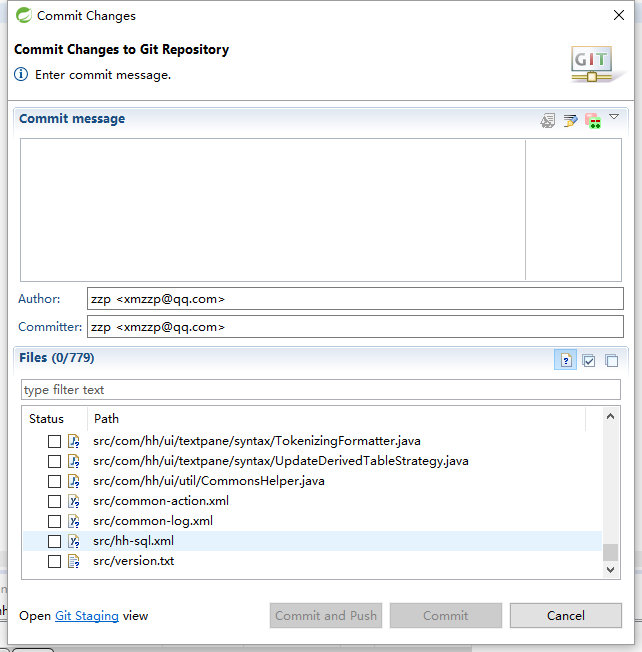
下面的Files窗口中可以看到此次提交的文件，其中有非常多带有work的文件，此时，在我们的project中链接的work也被GIT默认到版本控制中了。

work下面的文件不需要进行版本控制的，我们可以通过配置.gitignore来排除这这个文件夹。

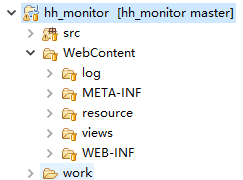
打开Navigator窗口，在project根目录中添加.gitignore文件，将需要排除控制的目录写入.gitignore文件中。



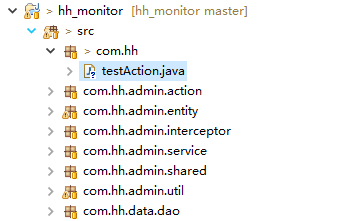
再次尝试commit，需要提交的文件已经被过滤。



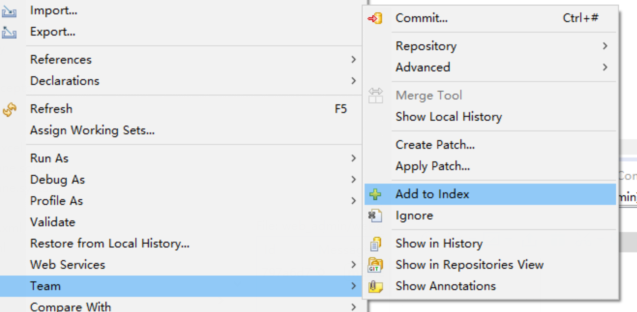
首次提交后，会自动生成master分支



然后在com.hh中新建一个文件，可以看到图标依然是问号，处于untracked状态，即git没有对此文件进行监控。



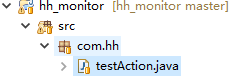
通过Team -> Add to index可以将文件加入git索引，进行版本监控



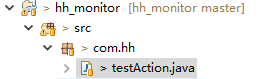
PS:可以看到图标显示也有了变化（EGIT中只要Commit就可以默认将untracked的文件添加到索引再提交更新，不需要分开操作）

也可以通过Team -> Untrack将文件从索引控制中排除。

将此次新增的文件commit到仓库中，文件将处于unmodified状态，或者说，这就是一种staged状态

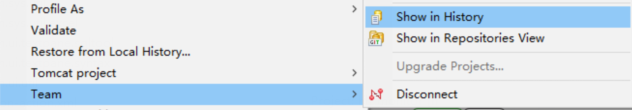


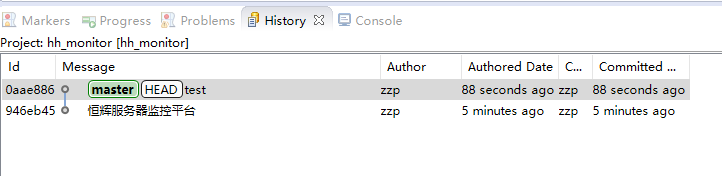
然后修改文件的内容，文件将处于modified状态



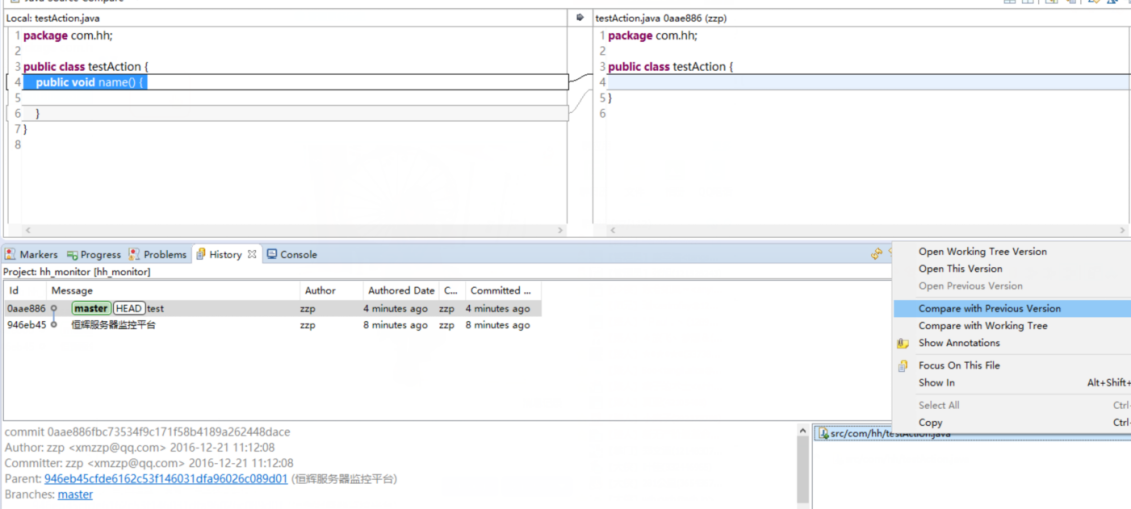
**五、 查看历史记录**

Team -> Show in history可以查看版本历史提交记录





可以选择对比模式

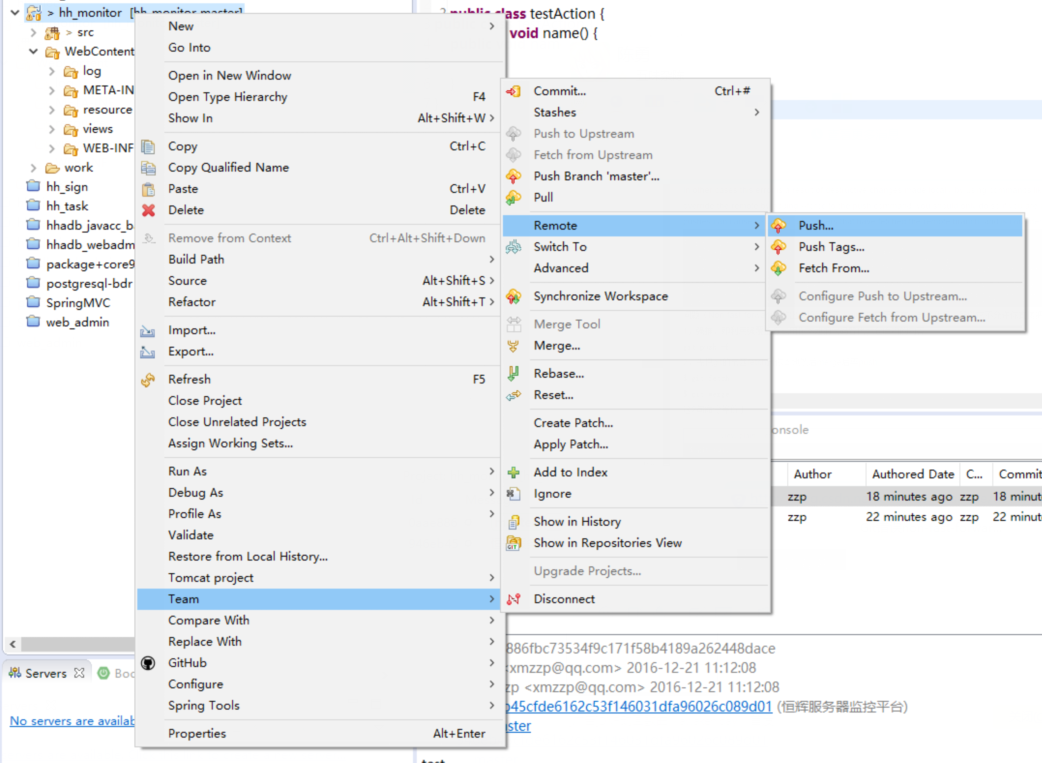


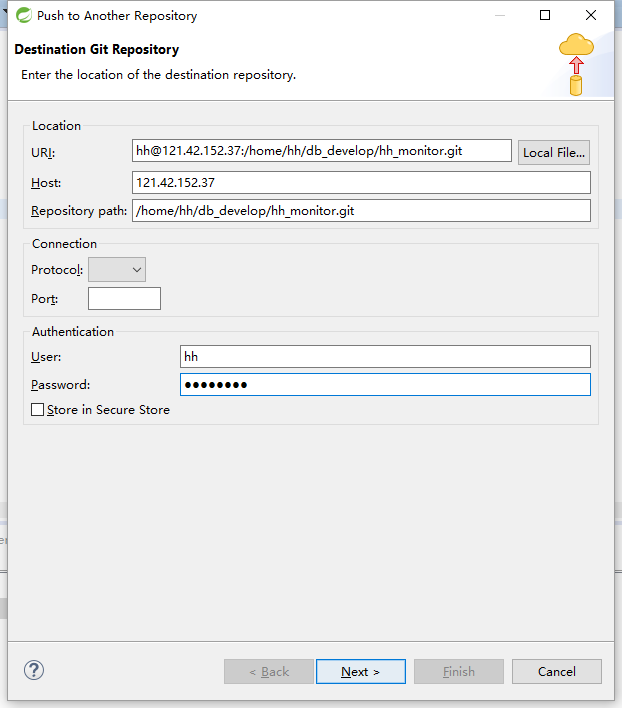
**六、 推送远程仓库**

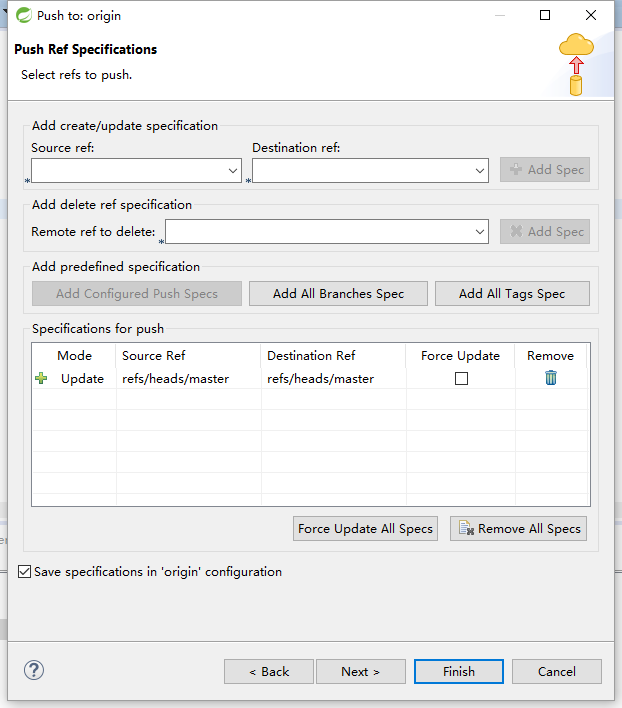
我们的git仓库已经在服务器建好。

我在服务器已创建hh\_monitor.git。

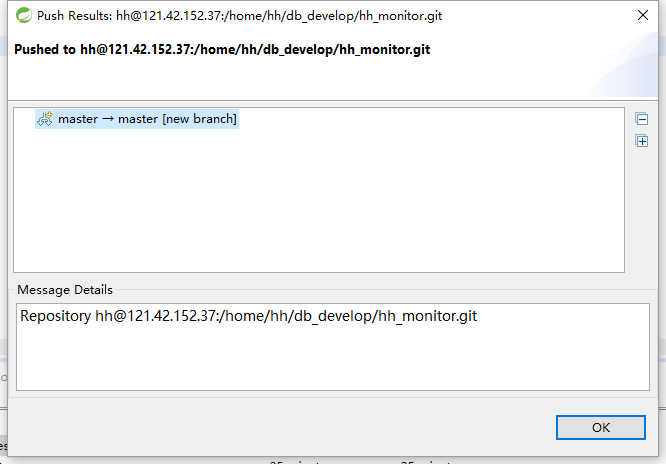
Team -> Remoto->Push





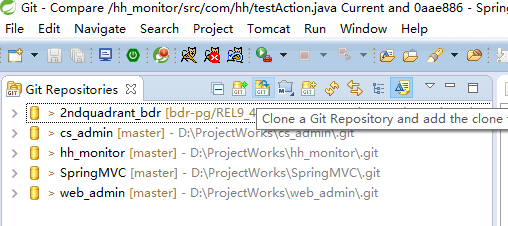


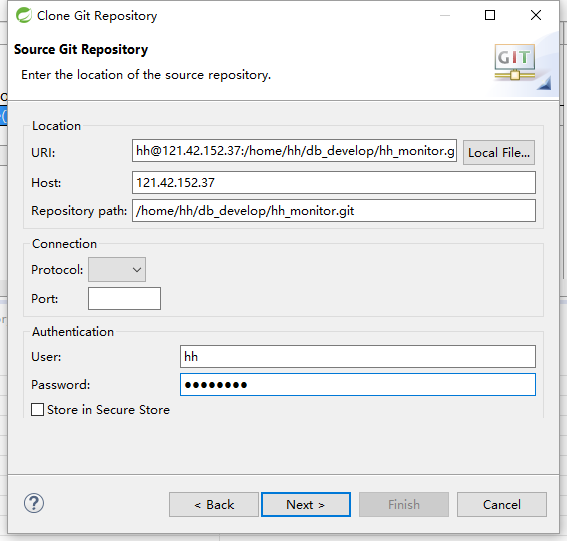
完成推送后，弹出如下界面。

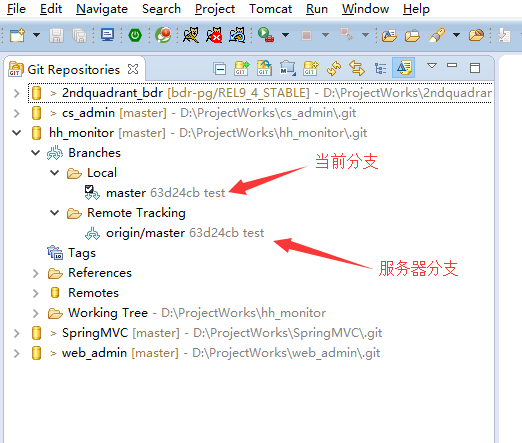


**七、远程GIT仓库**

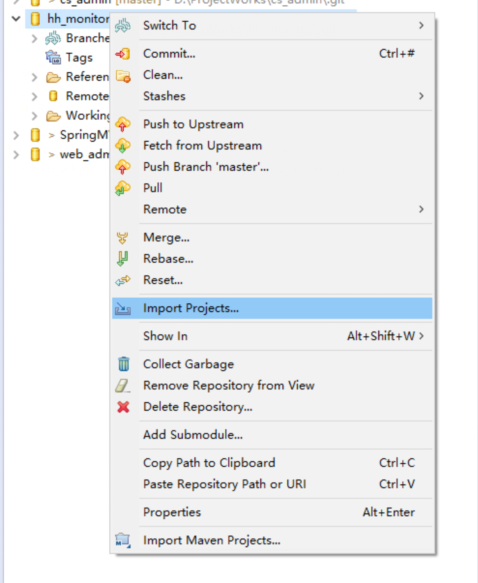
如果其他同事参加到此项目开发。打开GIT资源库窗口，选择克隆资源库。

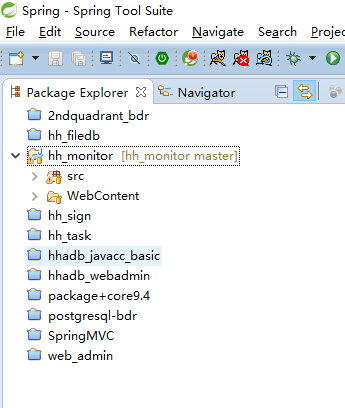






现在已经把远程的GIT仓库克隆到本地，接下来需要将仓库检出为项目。



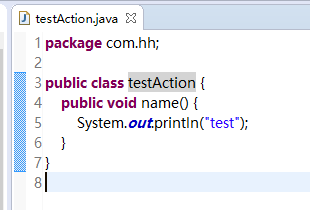


**八、 解决推送冲突**

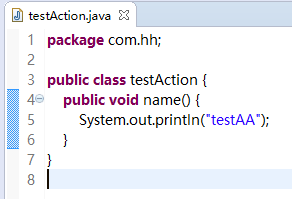
多人协作开发的情况下，往服务器推送更新时难免出现冲突，所以推送之前需要解决服务器端的最新版本和本地仓库的冲突。

Pull操作就是把服务器端的更新拉拢到本地仓库进行合并，解决好合并冲突后，就可以顺利push到服务器分支了。

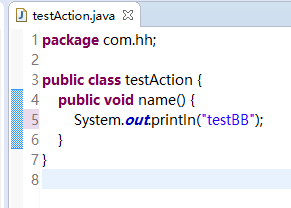
假设，我和乐子在用GIT协作开发hh\_monitor项目，目前服务器端的testAction.java文件的内容如下：



乐子做了如下修改

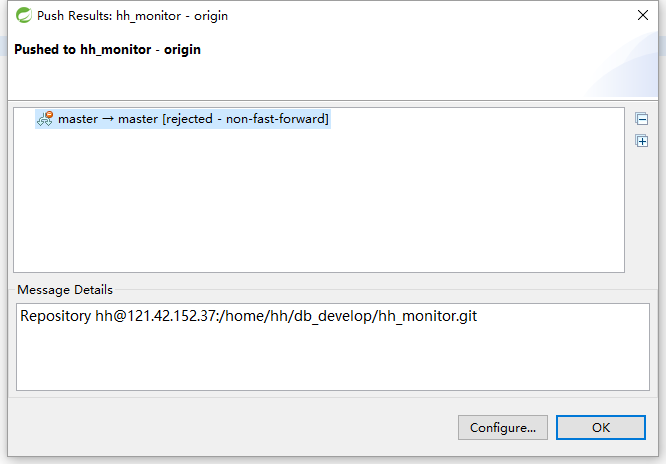


我做了如下修改

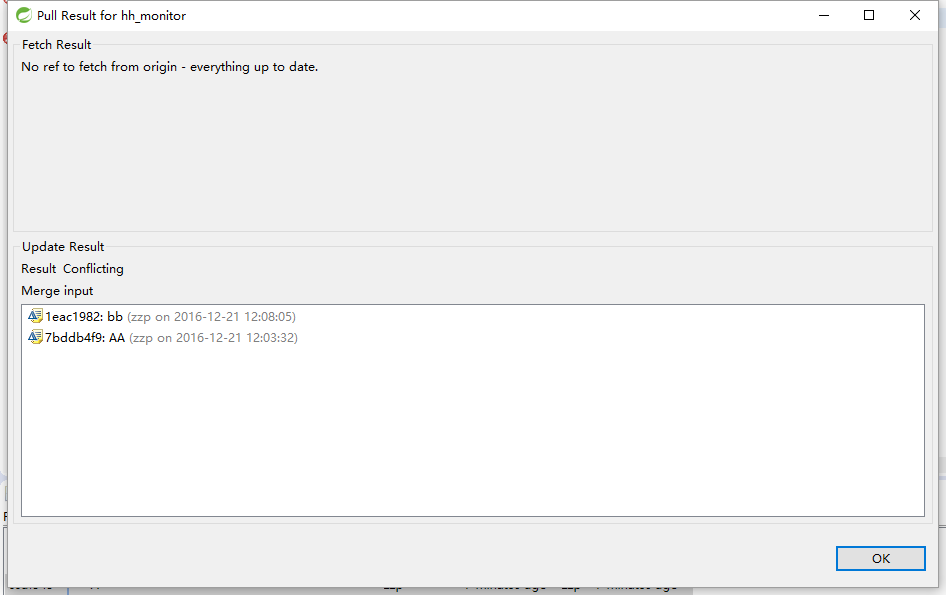


然后乐子先push代码，我commit到本地仓库。

Push会出现如下提示：



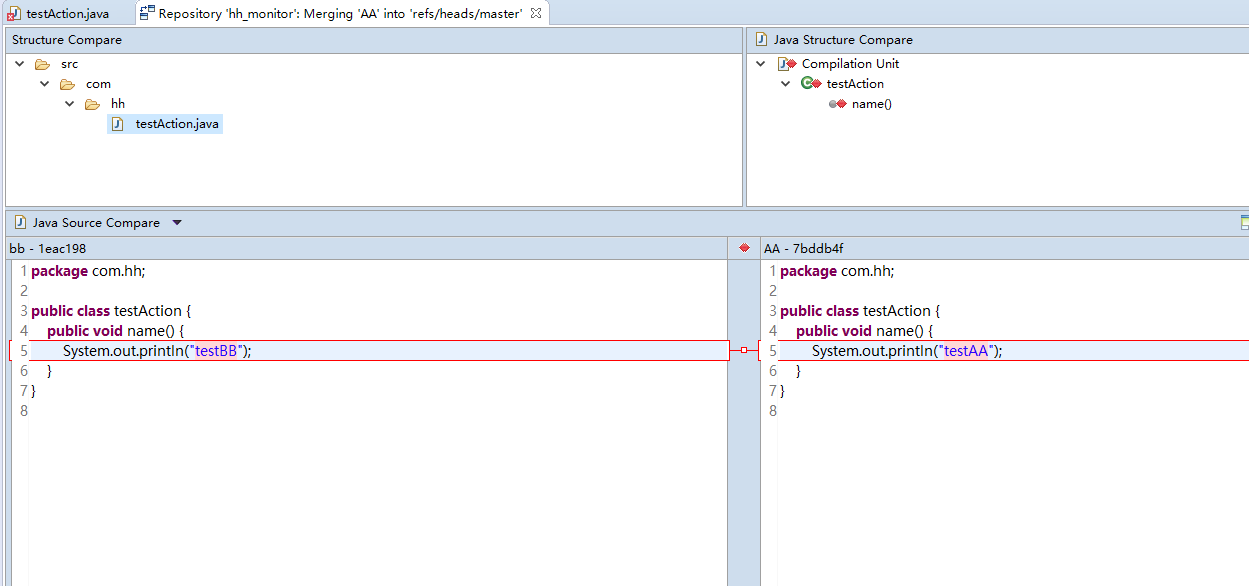
使用pull来合并本地仓库和远程仓库，发现文件出现冲突，此时GIT会自动合并冲突的文件，如下图所示：





很明显自动合并的冲突文件不能直接使用，我们可以手动调整，右键发生冲突的文件，选择Team -> Merge Tool

接下来就是熟悉的对比界面



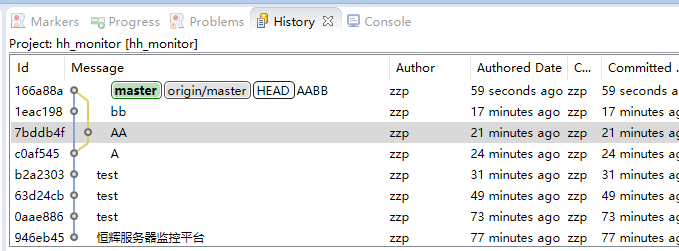
Team -> Untrack将文件从索引控制中排除

然后右键点击此冲突文件，选择Team -> Add to index再次将文件加入索引控制。

此时文件已经不是冲突状态，并且可以进行提交并push到服务器端。

解决合并冲突后，乐子只需要将服务器中合并后的版本pull到本地，就完成了一次协作开发的代码合并。

从历史记录中可以看到，从mushroom开始历史进入分支，先是AA的记录，然后是BB的记录，最后历史分支合并。



**九、重置功能**

GIT中有三种重置功能，分别是soft、mixed、hard，区别如下：

Soft - 当前分支重置到指定commit记录位置，索引和工作树不变；

Mixed - 当前分支重置到指定commit记录位置，索引被更新，工作树不变；

Hard - 当前分支重置到指定commit记录位置，索引和工作树都更新。

查看项目历史纪录，选择Reset,工作树是指左边的项目列表。

例如，我的项目被同事修改了很多功能，我想重置回到几天前的项目。就可以直接Reset--Hard到指定的commit记录位置。整个项目就重置了。

