

# Git的使用

一>.git的介绍

二>.准备工作（下载git）

三>.安装、配置

- 1.安装git
- 2.产生.ssh文件
- 3.建立github连接

四>. Git Bash的使用

五>.Eclipse上git的使用

- 1.在eclipses上配置git
- 2.Eclipse使用git提交代码

- # Git介绍

git是一个分布式的版本控制系统，在Git中并不存在主库这样的概念，每一份复制出的库都可以独立使用，任何两个库之间的不一致之处都可以进行合并

- ## Git优点

- 1.分支更快、更容易 (在Git中并不存在主库概念，每一份复制出的库都可以独立使用，任何两个库之间的不一致之处都可以进行合并)

- 2.支持离线工作,本地提交可以稍后提交到服务器上

3. Git 提交都是原子的，且是整个项目范围的，而不像svn中一样是对每个文件的

4. Git 中的每个工作树都包含一个具有完整项目历史的仓库。

- 5.具有reset功能，在开发开始的时候首先checkout出代码，然后建立一个分支，开始开发。修改代码后，提交（仅在本地保存版本信息，未提交到服务器）。等一个任务完成后合并到主干，然后提交到代码服务器。也就是上面提到的离线开发。由于分支是在本地建立的，所以不管是提交还是建立分支，合并分支，速度都会很快。

# • 准备工作

1. Windows环境下载git地址: <http://code.google.com/p/msysgit/>;
2. Github:在<https://github.com/>上注册一个账户

## github介绍

Github可以托管各种git库，并提供一个web界面

- Git安装、配置

点击你下载的git,根据提示进行安装

- 注册github和设置ssh密钥

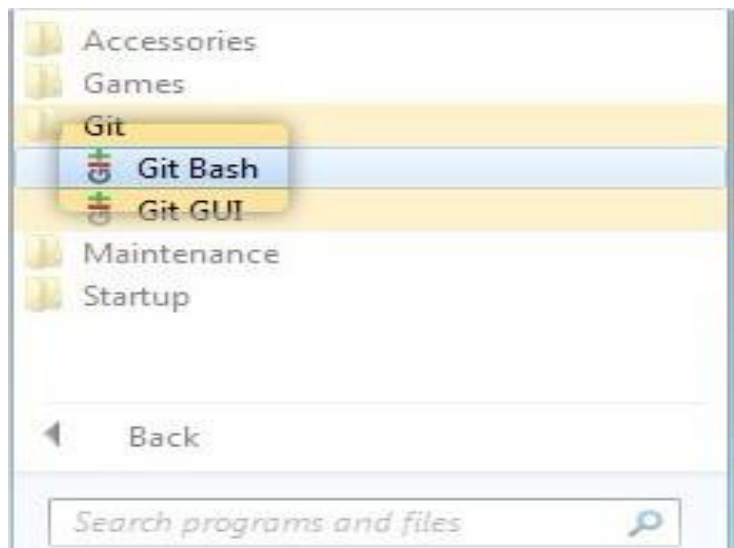
1.注册地址为:<https://github.com>

Join today and collaborate with the smartest developers in the world.



2.设置ssh密钥

打开git bash



1.输入email地址，该email为你注册在github上的email

```
$ ssh-keygen -t rsa -C "your_email@youremail.com"
Generating public/private rsa key pair.
Enter file in which to save the key
(/Users/your_user_directory/.ssh/id_rsa):<press enter>
```

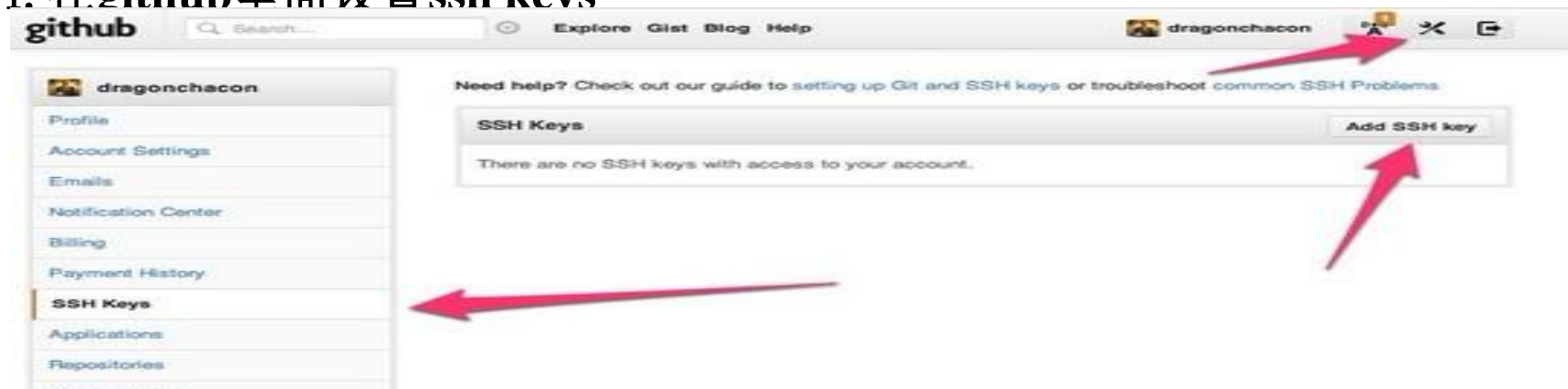
2.两次输入你要设置的密码，然后按enter

```
Enter passphrase (empty for no passphrase):<enter a passphrase>
Enter same passphrase again:<enter passphrase again>
```

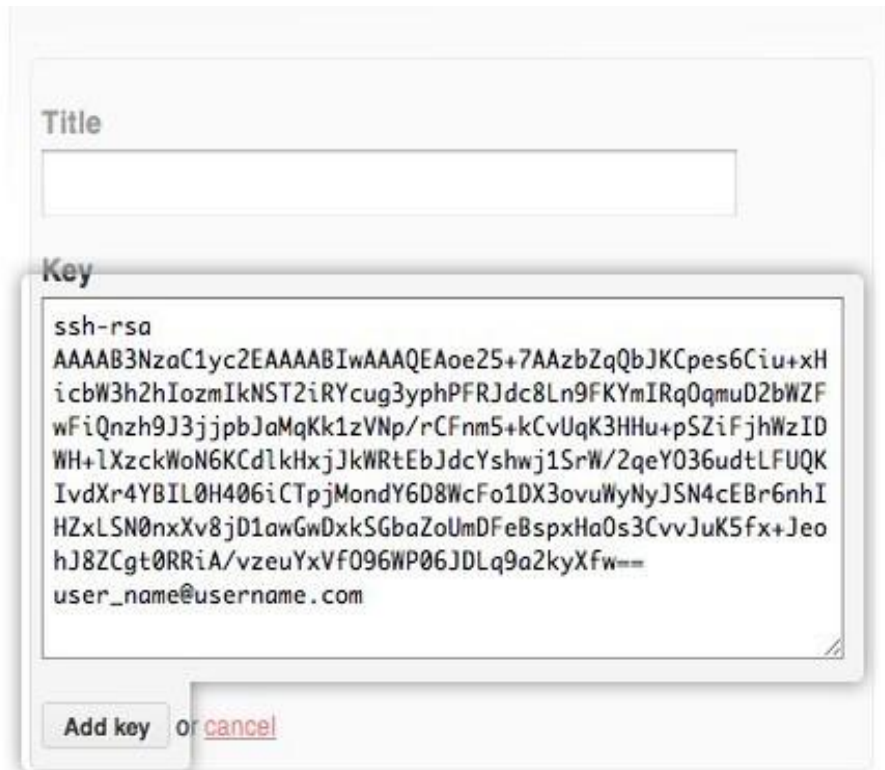
3.成功显示

```
Your identification has been saved in
/Users/your_user_directory/.ssh/id_rsa.
Your public key has been saved in
/Users/your_user_directory/.ssh/id_rsa.pub.
The key fingerprint is:
01:0f:f4:3b:ca:85:d6:17:a1:7d:f0:68:9d:f0:a2:db user_name@username.com
```

4. 在github里面设置ssh keys



5.在.ssh下的id\_rsa.Pub用文本格式打开  
复制里面内容，注意不要复制到空格，然后填进key里面



6. 检查是否建立连接

```
$ ssh -T git@github.com
```

7. 建立连接以后会显示信息如下

```
Hi username! You've successfully authenticated, but GitHub does not provide  
shell access.
```

# • git常用命令介绍

**git init**

创建一个数据库。

**git clone**

复制一个数据到指定文件夹

**git add** 和 **git commit**

把想提交的文件**add**上，然后**commit**这些文件到本地数据库。

**git pull**

从服务器下载数据库，并跟自己的数据库合并。

**git fetch**

从服务器下载数据库，并放到新分支，不跟自己的数据库合并。

**git whatchanged**

查看两个分支的变化。

**git branch**

创建分支，查看分支，删除分支

**git checkout**

切换分支

**git merge**

合并分支，把目标分支合并到当前分支

**git config**

配置相关信息，例如email和name

**git log**

查看版本历史

## git show

查看版本号对应版本的历史。如果参数是HEAD查看最新版本。

## git tag

标定版本号。

## git reset

恢复到之前的版本

----mixed 是 git-reset 的默认选项，它的作用是重置索引内容，将其定位到指定的项目版本，而不改变你的工作树中的所有内容，只是提示你有哪些文件还未更新。

--soft 选项既不触动索引的位置，也不改变工作树中的任何内容。该选项会保留你在工作树中的所有更新并使之处于待提交状态。相当于再--mixed基础上加上git add .

--hard 把整个目录还原到一个版本，包括所有文件。

## git push

向其他数据库推送自己的数据库。

## git status

显示当前的状态。

## git mv

重命名文件或者文件夹。

## git rm

删除文件或者文件夹。

## git help

查看帮助，还有几个无关紧要的命令，请自己查看帮助



# • 使用github（用命令行来提交代码）

## 1.先在github上建立资源库(Hello-Wrold)


**Repositories (14)**

New repository

All RepositoriesPublicPrivateSourcesForksMirrors


Owner


Repository name

 zhonglihua

Great repository names are short and memorable. Need inspiration? How about **glow**

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 allow you to `git clone` the repository immediately.  
Add additional files: **None**

1.在git bash里创建并进入Hello-Wrold目录，用git init初始版本库

```
Feng@FENG-PC ~  
$ mkdir Hello-world  
  
Feng@FENG-PC ~  
$ cd Hello-world  
  
Feng@FENG-PC ~/Hello-World  
$ git init  
Initialized empty Git repository in c:/Users/Feng/Hello-World/.git/  
  
Feng@FENG-PC ~/Hello-World (master)  
$
```

2.用git add跟踪我们要提交的文件，然后commit，记得用-m设置提交说明






3. 然后push到远程

```
Feng@FENG-PC ~/Hello-World (master)  
$ git remote add origin git@github.com:yongfenghua/Hello-World.git
```

```
Feng@FENG-PC ~/Hello-World (master)  
$ git add helloworld.java  
  
Feng@FENG-PC ~/Hello-World (master)  
$ git commit -m "first commit"  
[master (root-commit) b6104a2] first commit  
1 files changed, 5 insertions(+), 0 deletions(-)  
create mode 100644 helloworld.java
```

```
Feng@FENG-PC ~/Hello-World (master)  
$ git push -u origin master  
Enter passphrase for key '/c/Users/Feng/.ssh/id_rsa':  
Counting objects: 3, done.  
Delta compression using up to 4 threads.  
Compressing objects: 100% (2/2), done.  
Writing objects: 100% (3/3), 312 bytes, done.  
Total 3 (delta 0), reused 0 (delta 0)  
To git@github.com:yongfenghua/Hello-World.git  
* [new branch]      master -> master  
Branch master set up to track remote branch master from origin.
```


No description or homepage.

 ZIP  SSH  HTTP  Git Read-Only   Read+Write access


branch: master Files Commits Branches 1

Latest commit to the master branch

first commit

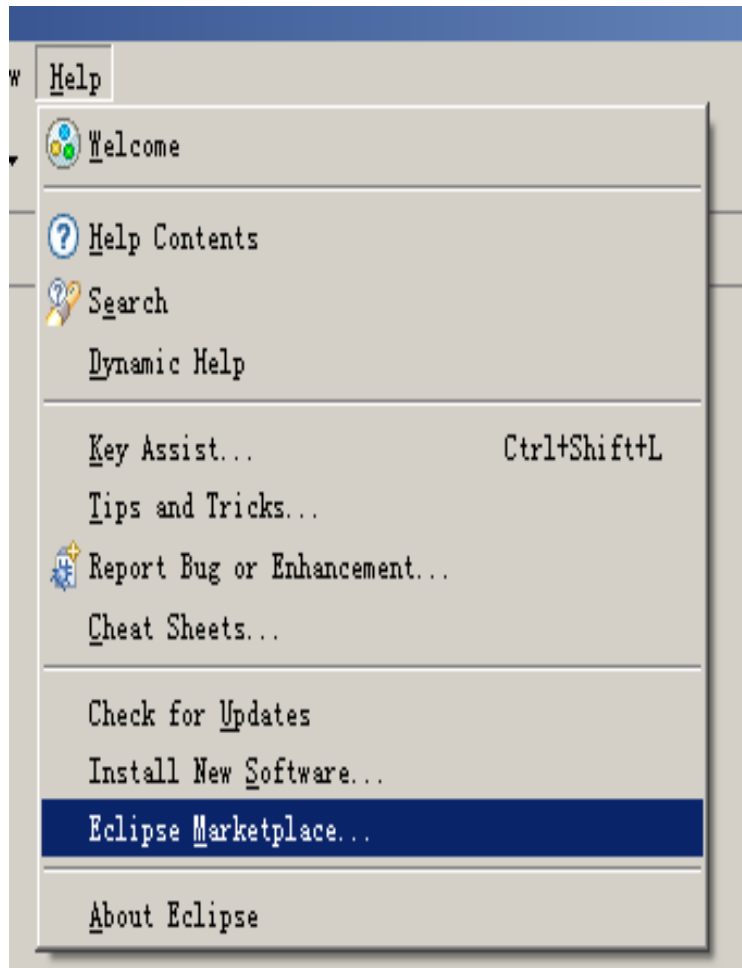
 yongfenghua authored 17 minutes ago

Hello-World /

name	age	message
 helloworld.java	17 minutes ago	first commit [yongfenghua]

# • Eclipse上Git使用

## 1.在Eclipse上安装git插件

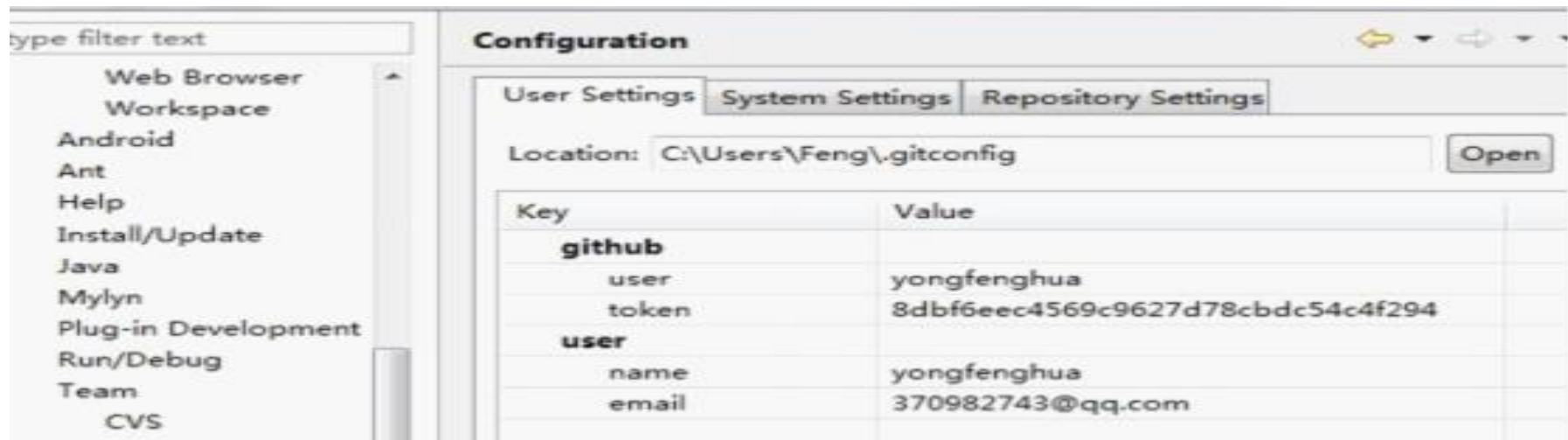


ssh: windows>preferences>general>network connection>ssh



**Ssh2 home:**为你.ssh的地址, **private keys:**为公钥和私钥, 不能错  
配置个人信息, 最重要的是**user.name**和**user.email**

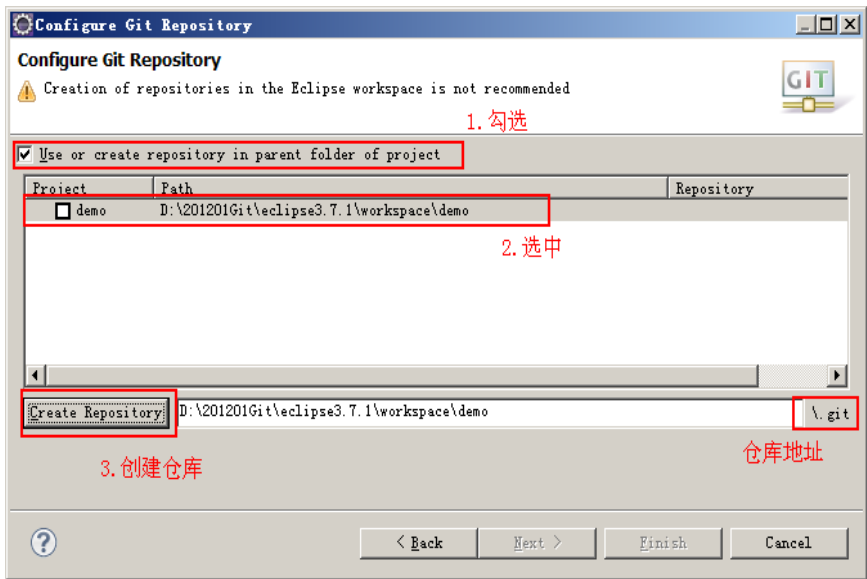
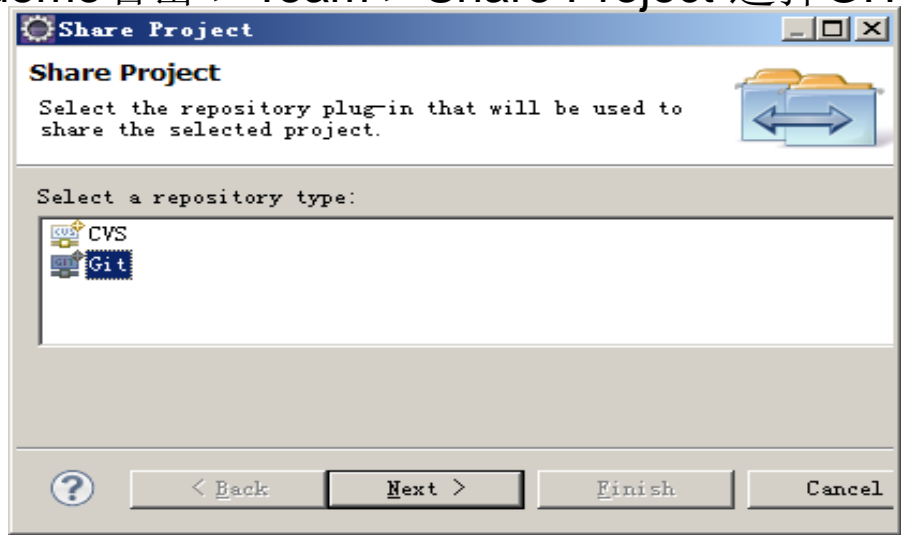
**I Preferences > Team > Git > Configuration**



(如果你已经用命令行配置, 这里可以直接看你的配置, 不用再配)

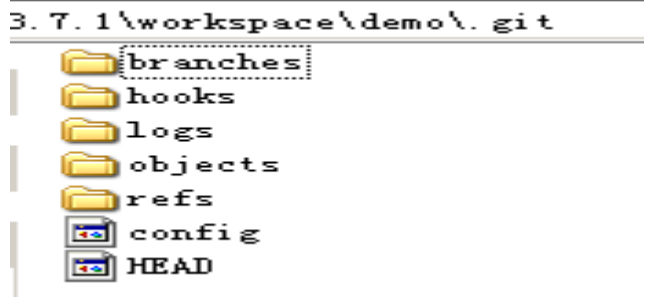
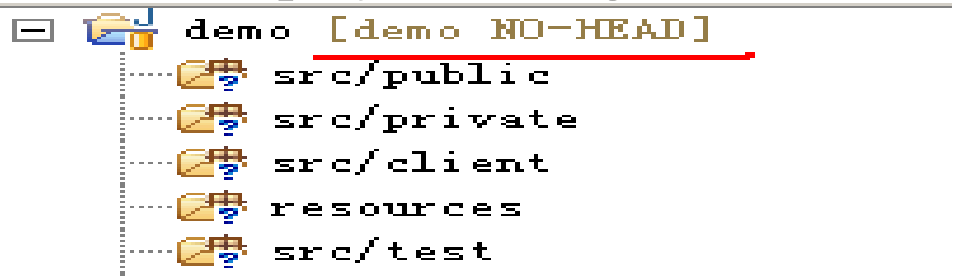
如果我要把eclipse中的demo传到自己的github上去，步骤如下  
先在github里面创建名为demo资源库（前面已说创建方法）

demo右击 > Team > Share Project 选择GIT



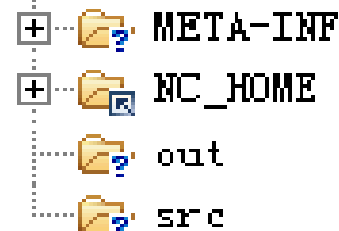
创建仓库后，在\$workspace\demo目录下的.git文件夹，就是git的仓库地址。

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



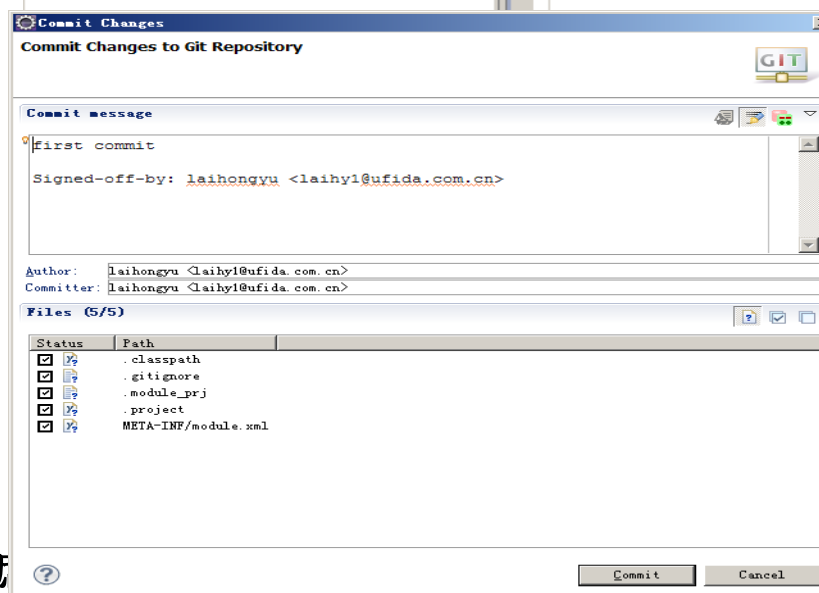
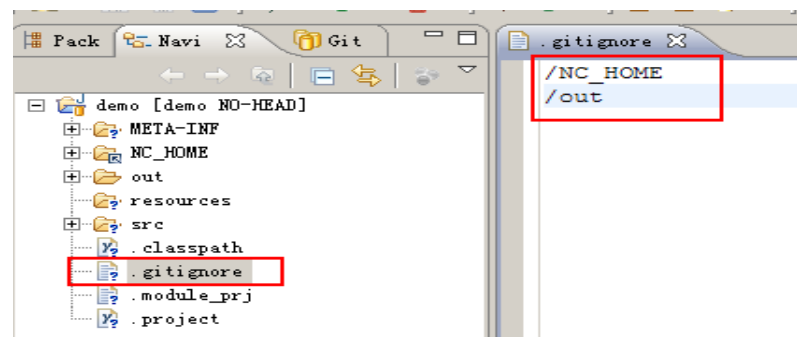
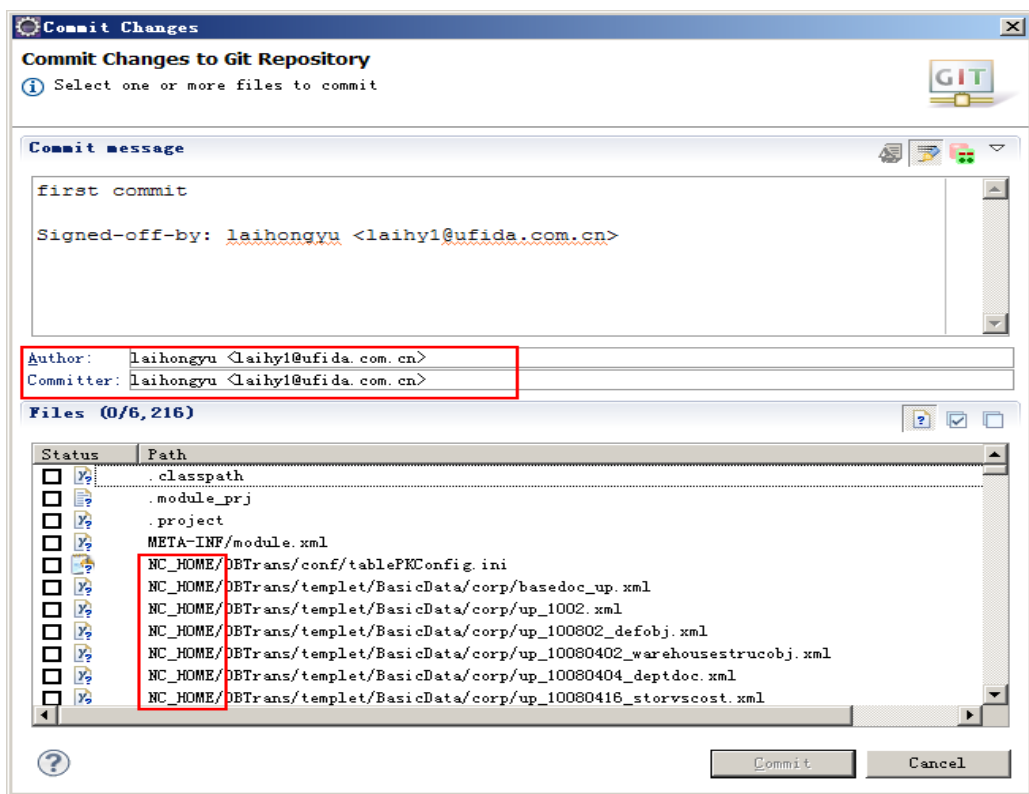
文件夹中的符号“？”表示此文件夹出于untracked状态，这样就成功创建GIT创库此时我们尝试做一次提交。Team->Commit

如下图所示，**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**NC\_HOME**的文件，此时可以猜测出，在我们的**project**中链接的**NC\_HOME**也被**GIT**默认到版本控制中了，如下



显然**NC\_HOME**和**out**是不需要进行版本控制的，我们可以通过配置**.gitignore**来排除这两个文件夹

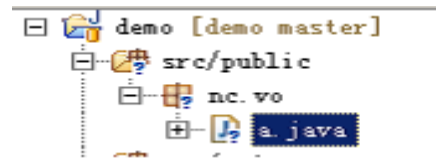
打开**Navigator**窗口，在**project**根目录中添加**.gitignore**文件，将需要排除控制的目录写入**.gitignore**文件中



再次尝试**commit**，需要提交的文件已经被过滤

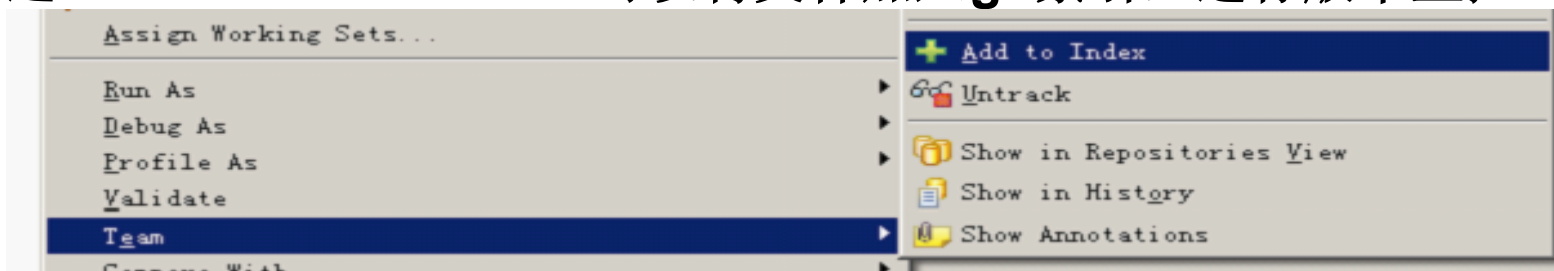


首次提交后，会自动生成**master**分支



然后在**public**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



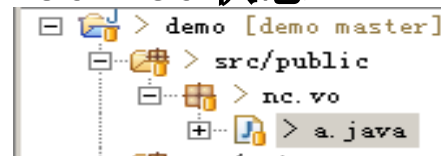
可以看到图标显示也有了变化（**EGIT**中只要**Commit**就可以默认将**untracked**的文件添加到索引再提交更新，不需要分开操作）



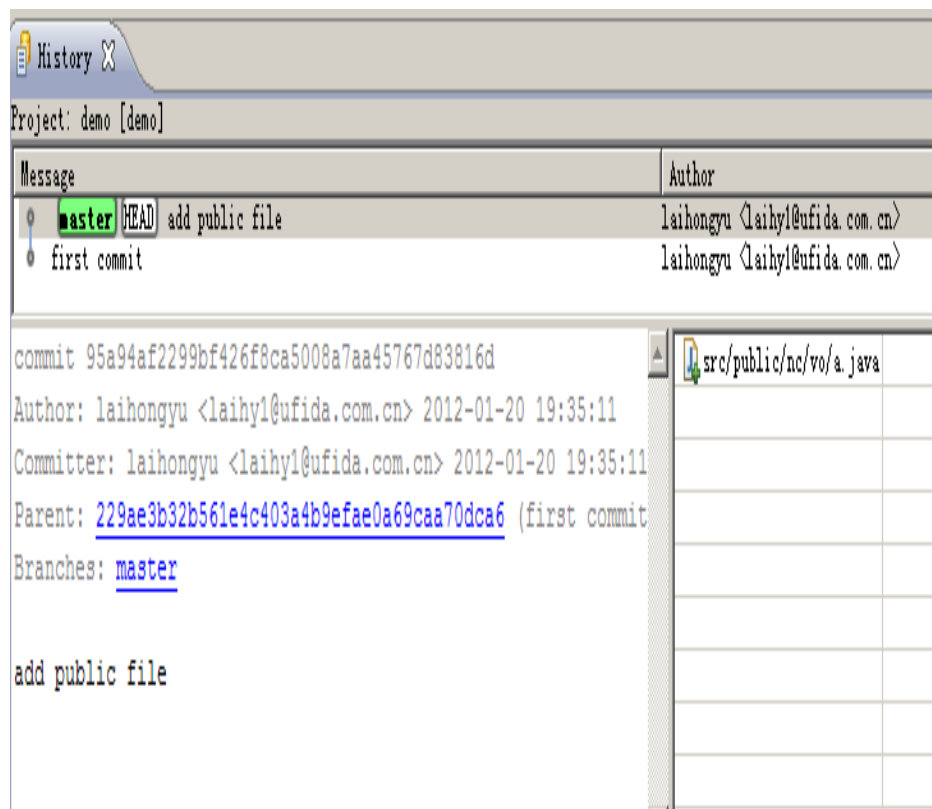
也可以通过**Team -> Untrack**将文件从索引控制中排除。

将此次新增的文件**commit**到仓库中，文件将处于**unmodified**状态

然后修改文件的内容，文件将处于**modified**状态

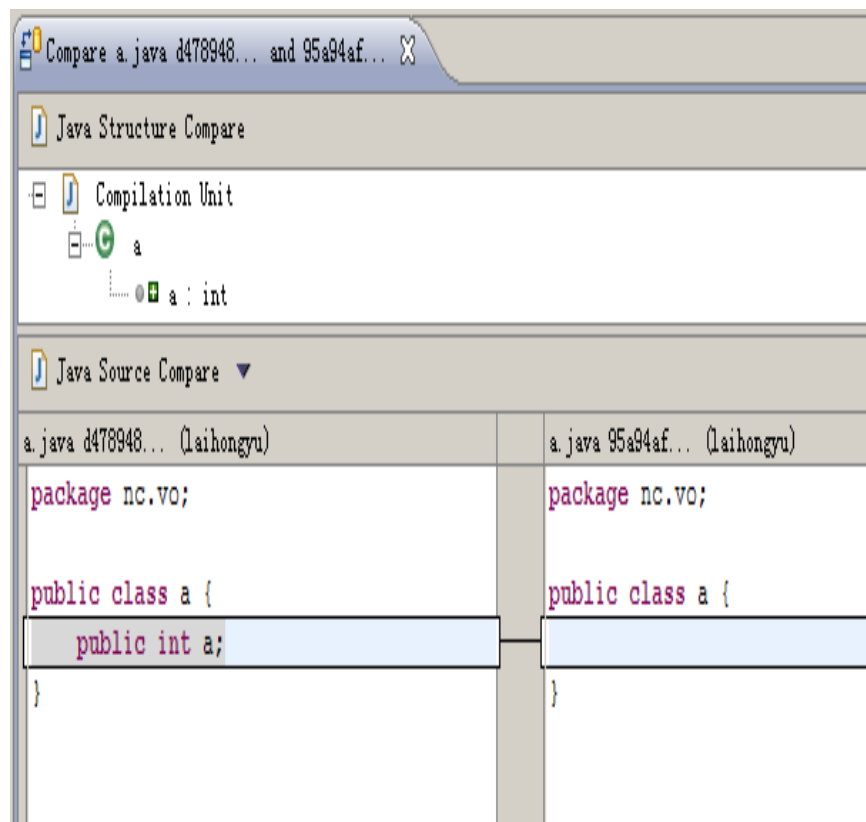
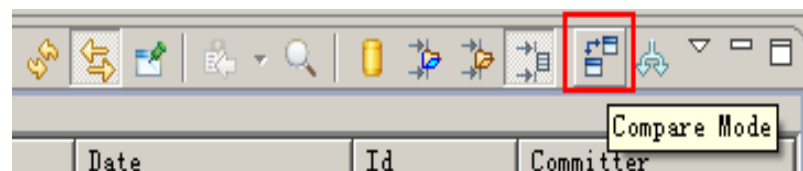


## Team -> Show in history 可以查看 版本历史提交记录



2015/3/24

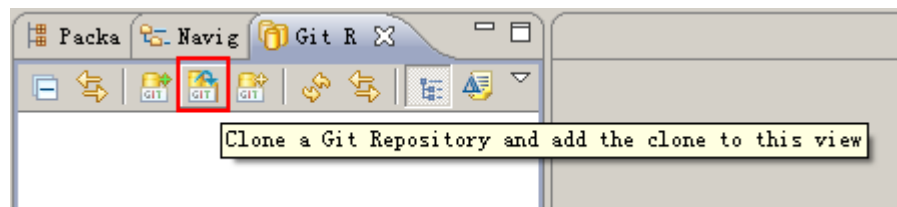
## 可以选择对比模式



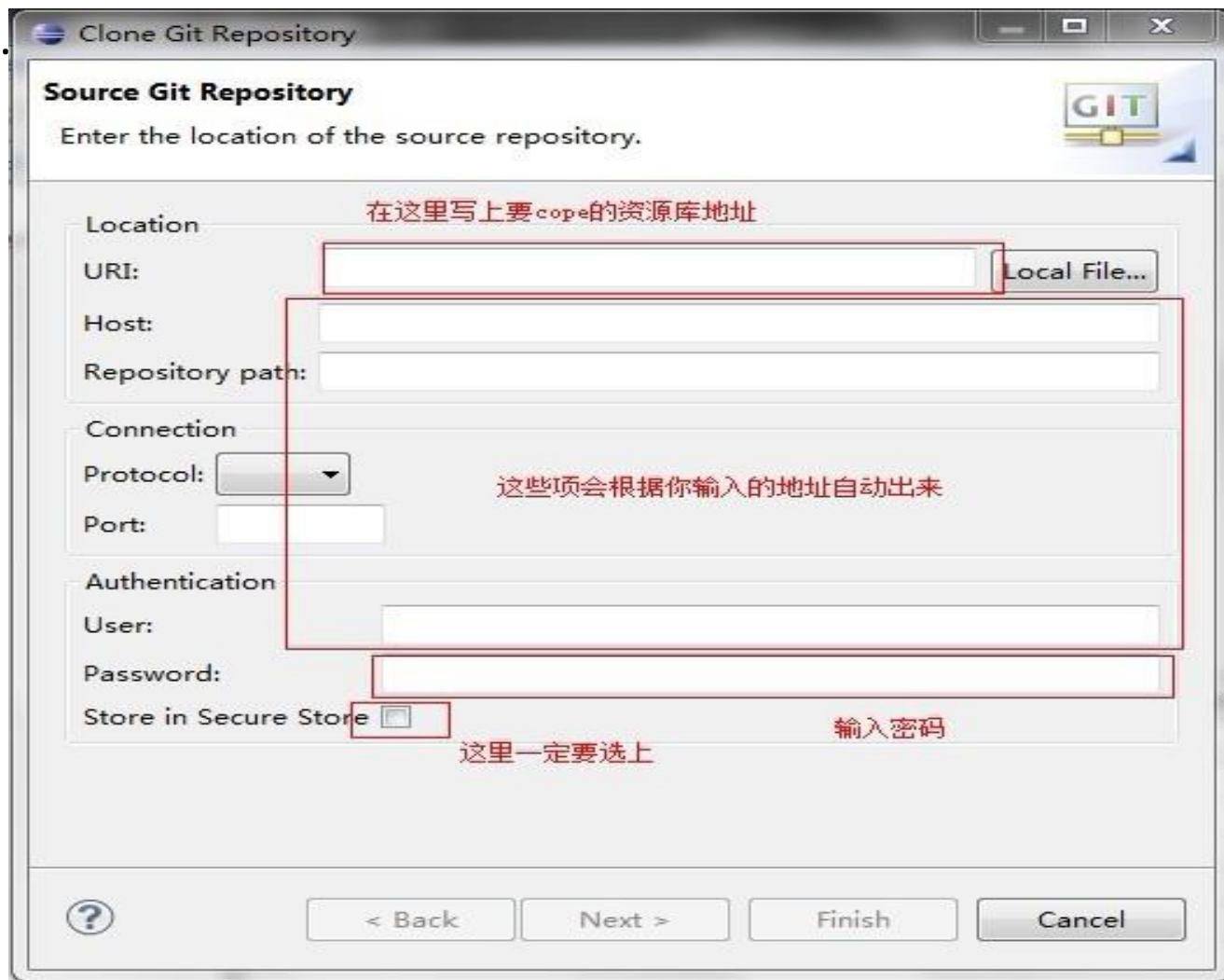


打开GIT资源库窗口

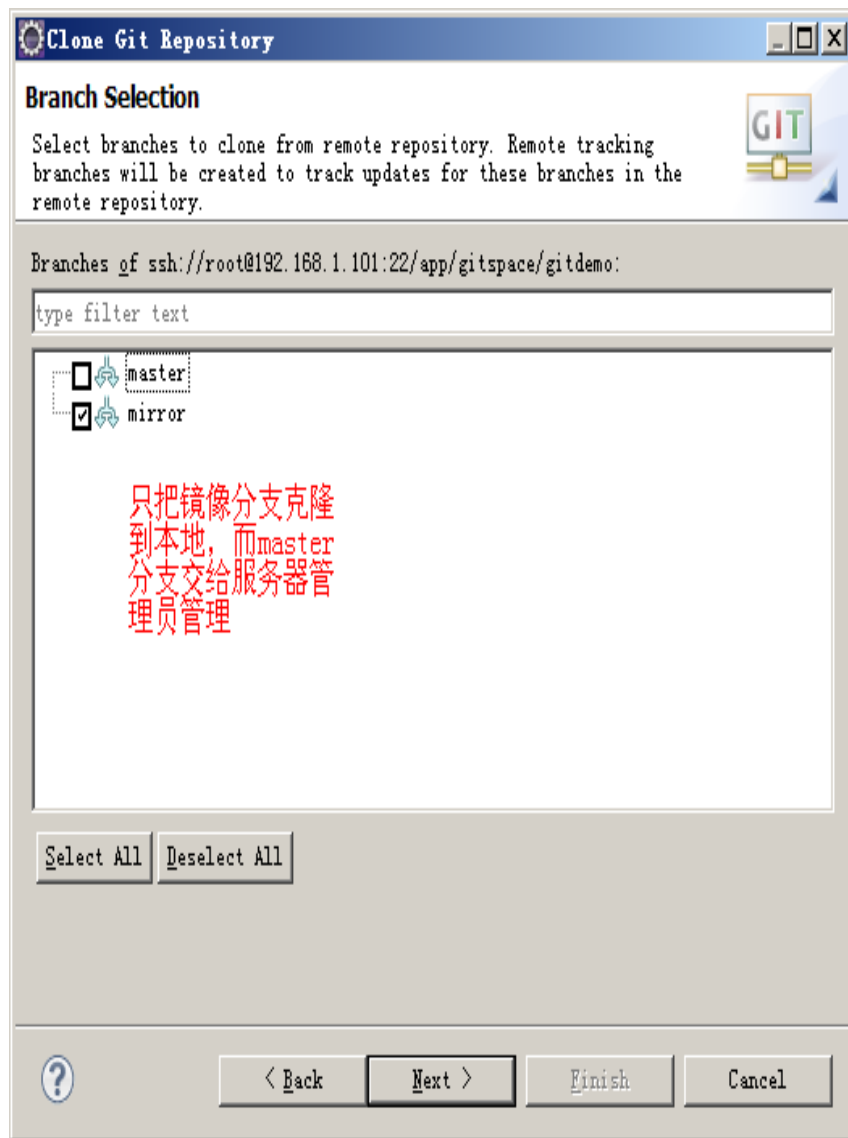
1.



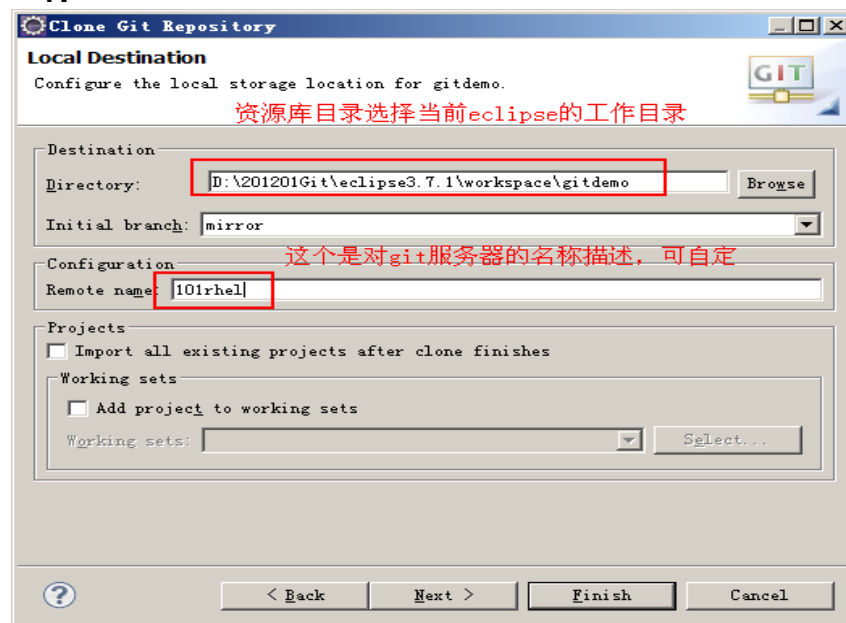
选择克隆资源库 2.



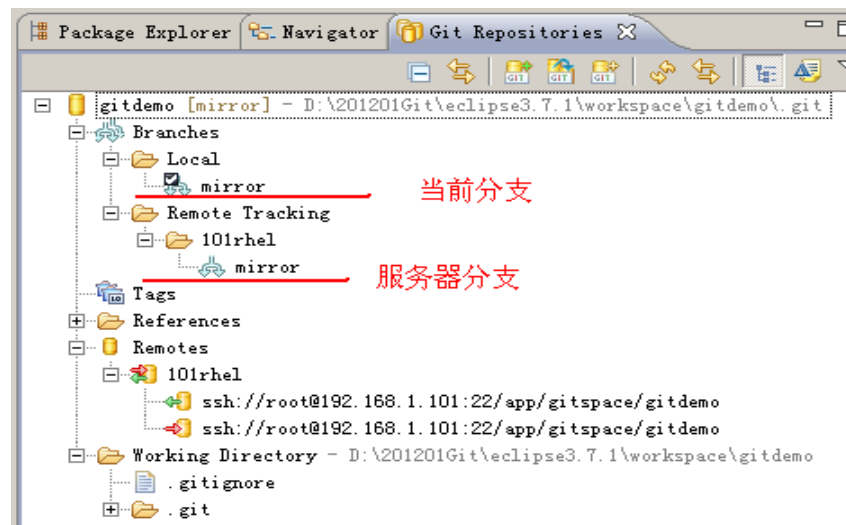
3



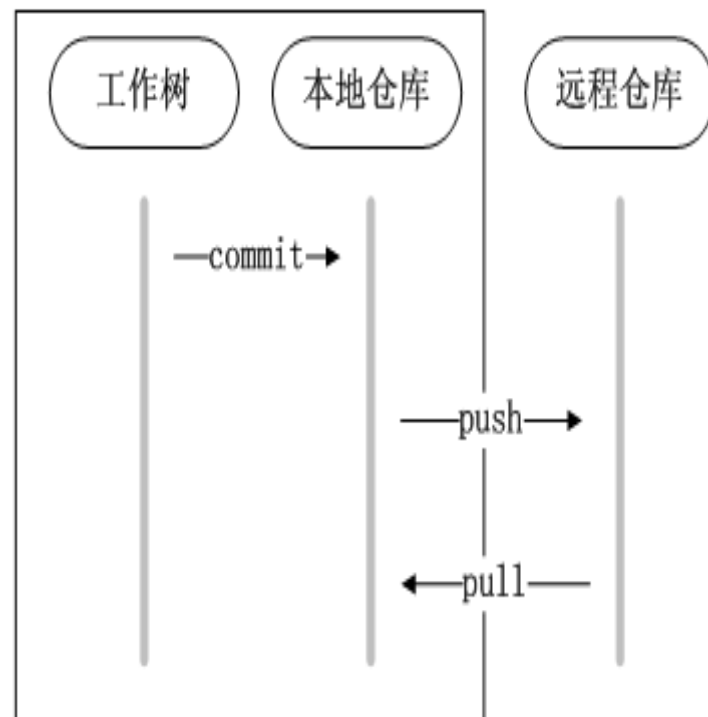
4.



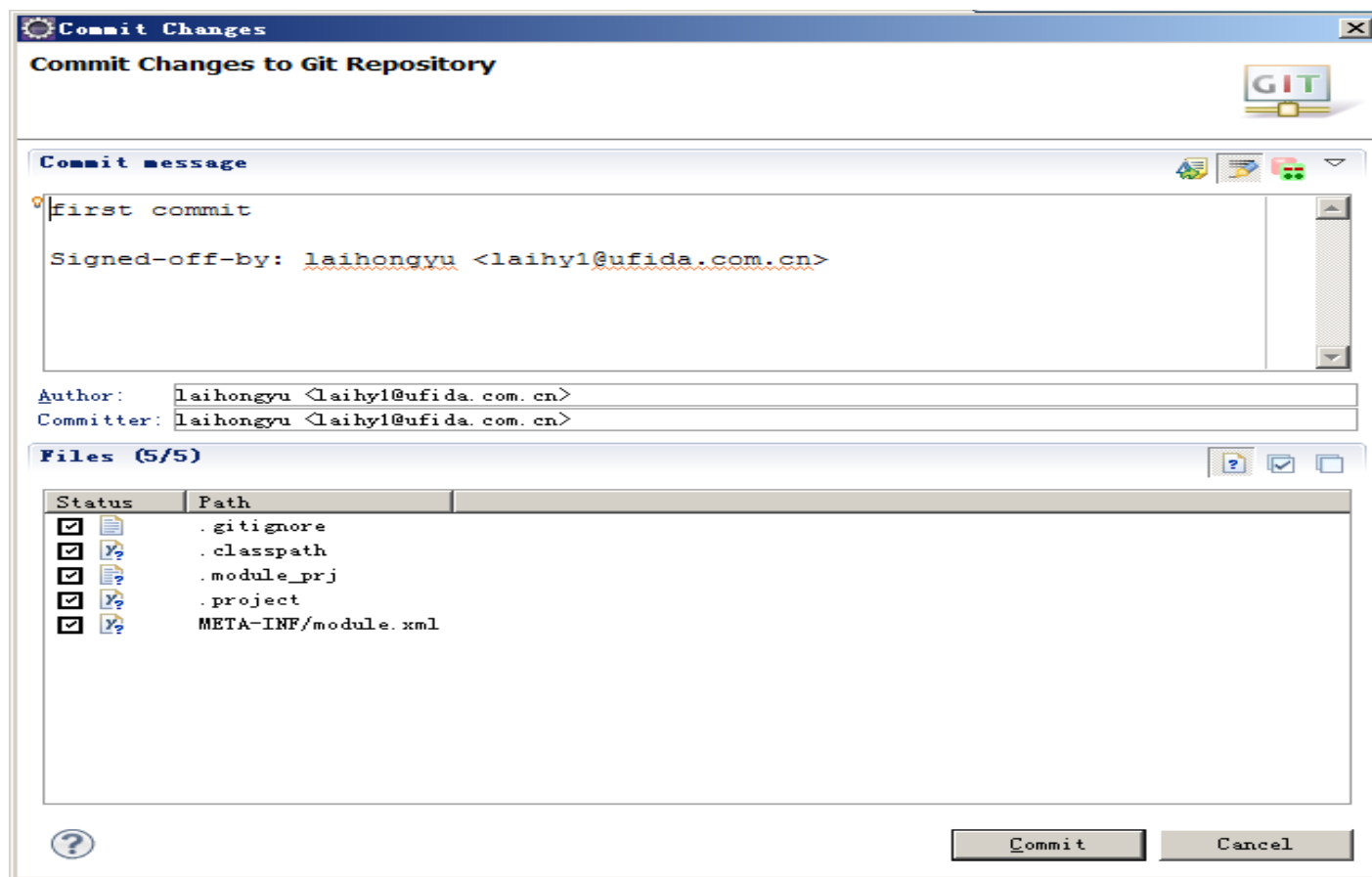
5.



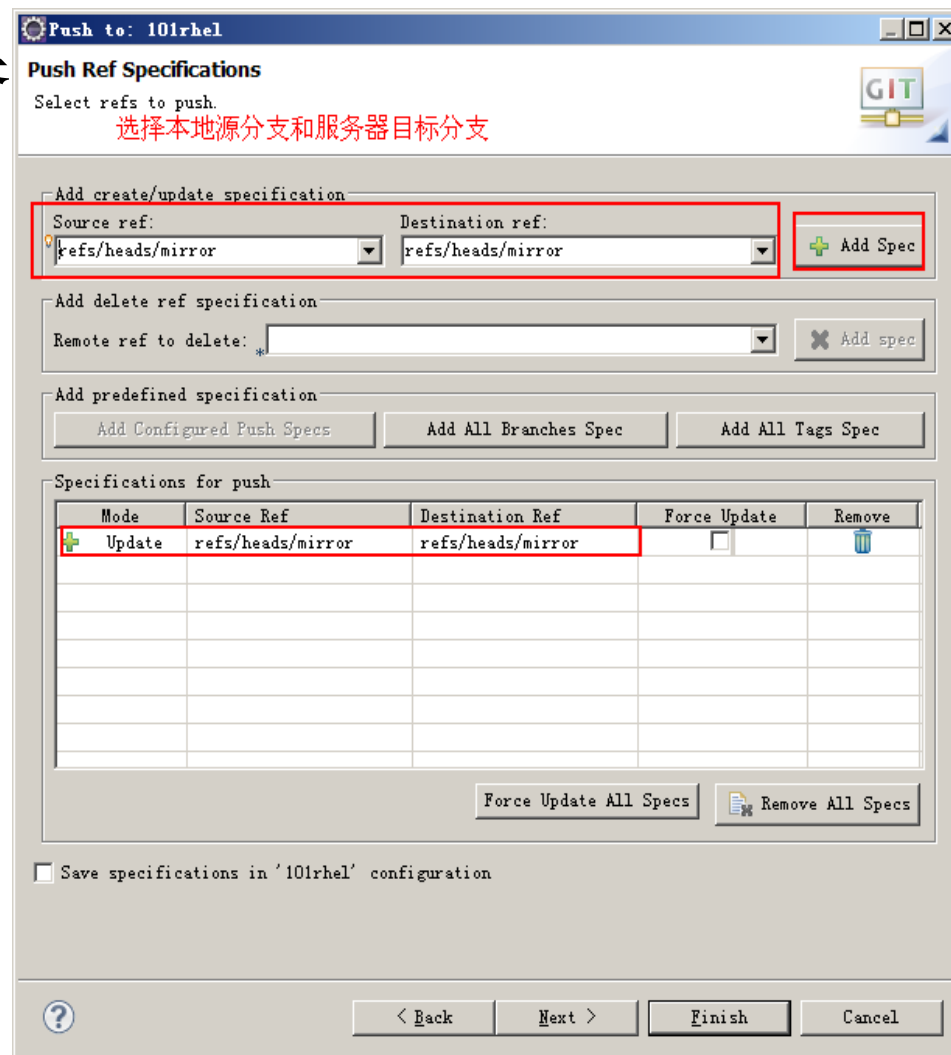
- 克隆服务器端仓库后，会在本地建立一个一样的仓库，称本地仓库。在本地进行commit操作将把更新提交到本地仓库，然后将服务器端的更新pull到本地仓库进行合并，最后将合并好的本地仓库push到服务器端，这样就进行了一次远程提交



- 先提交一次到本地仓库



- 然后push到服务器端的mirror分支，  
Team -> remote -> Push



- 完成推送后，可以在服务器端mirror镜像的log中查看到此次记录

```
[root@dans gitdemo]# git log mirror
commit 2ae7d294585bb66a0d98c7f7c5215392fa120f74
Author: laihongyu <laihy1@ufida.com.cn>
Date: Sat Jan 21 19:52:57 2012 +0800

    first commit

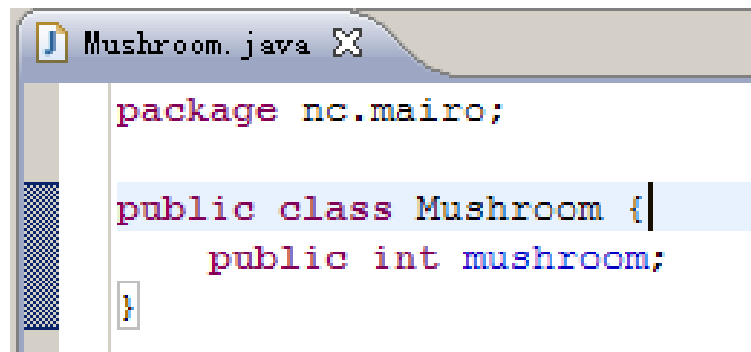
Signed-off-by: laihongyu <laihy1@ufida.com.cn>

commit 9e8f761f6b26b4db553f3bedad15ede7972711d3
Author: laihongyu <lai@lai.net>
Date: Sat Jan 21 12:16:39 2012 +0800

    init project gitdemo
```

- 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

假设现在Mairo兄弟在用GIT协作开发NewSuperMairoBro游戏，目前服务器端的mushroom.java文件的内容如下：

A screenshot of a code editor window titled 'Mushroom.java'. The code is written in Java and consists of three lines: a package declaration, a class declaration, and a class attribute. The text is as follows:

```
package nc.mairo;  
  
public class Mushroom {  
    public int mushroom;  
}
```

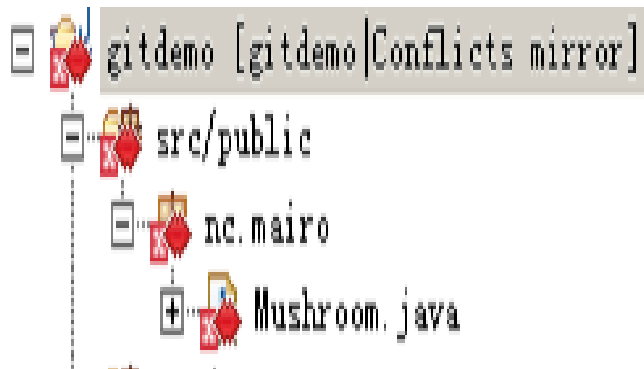
**MairoBro**克隆出代码后，**Mairo**哥哥做了如下修改

```
package nc.mairo;  
  
public class Mushroom {  
    public int mushroomA;  
}
```

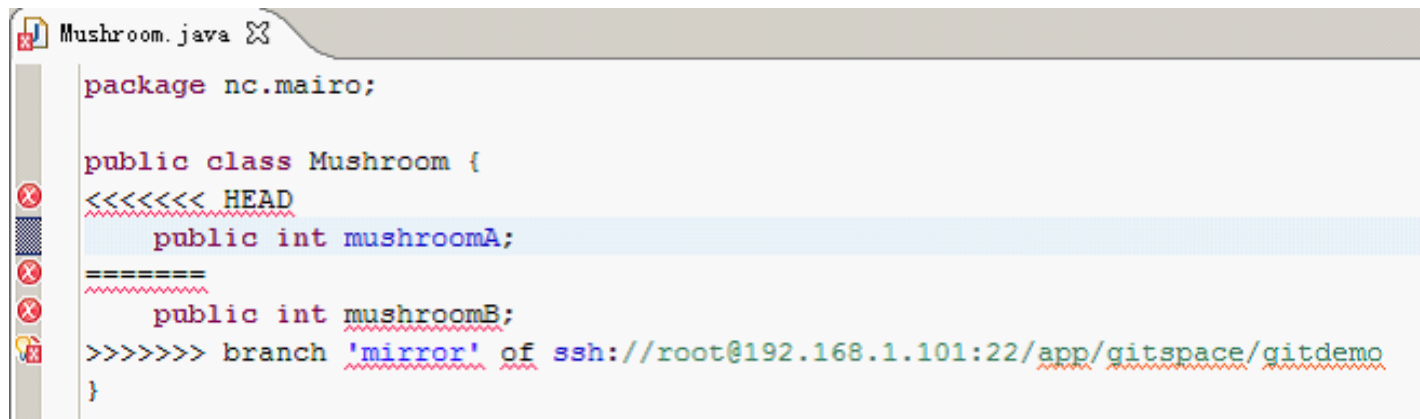
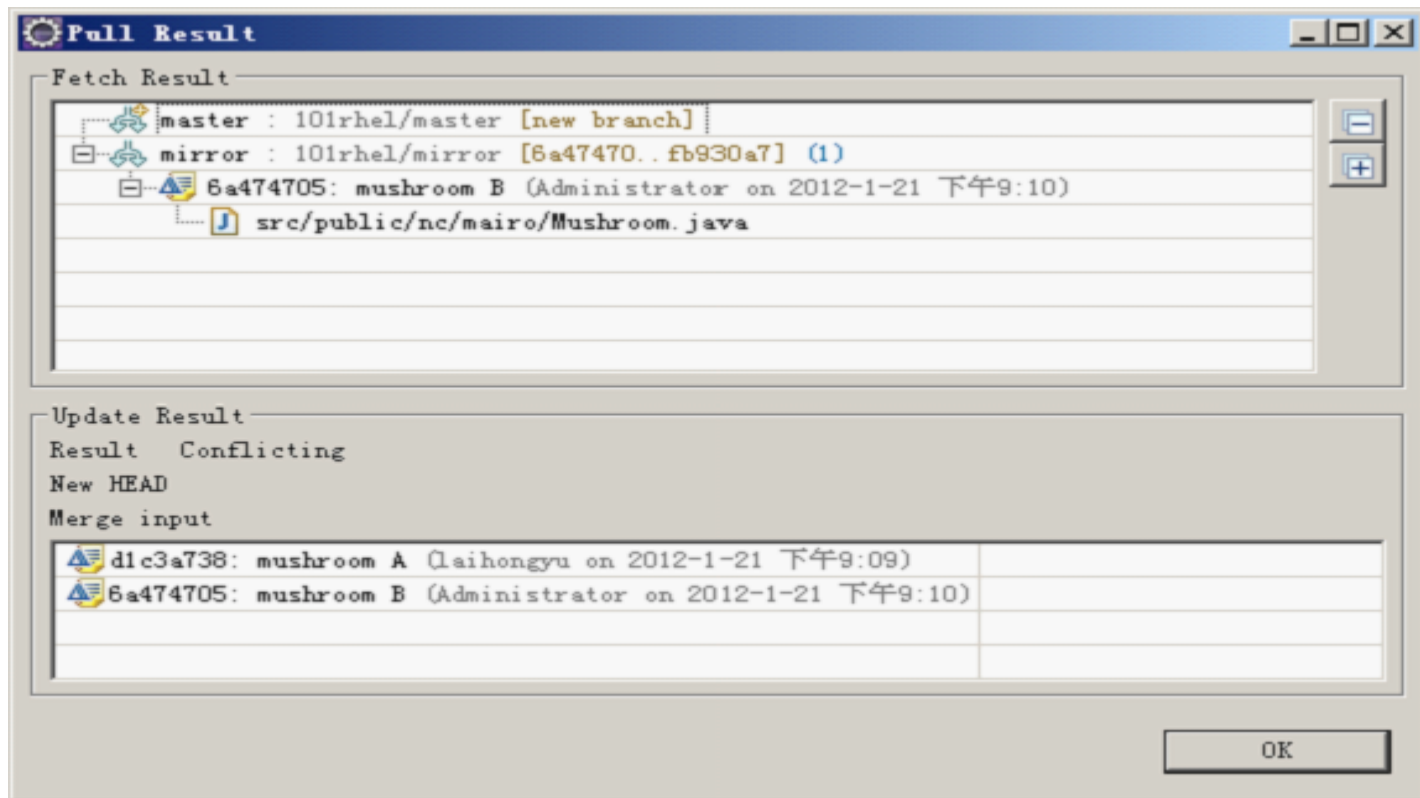
**Mairo**弟弟做了如下修改

```
package nc.mairo;  
  
public class Mushroom {  
    public int mushroomB;  
}
```

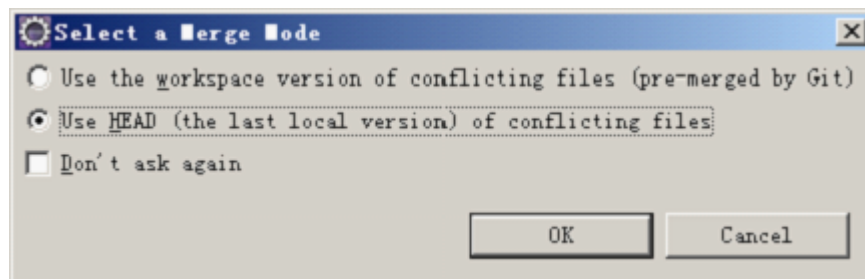
然后**Mairo**弟弟先**push**代码，**Mairo**哥哥使用**pull**来合并本地仓库和远程仓库，将发行文件出现冲突，此时**GIT**会自动合并冲突的文件，如下图所示：







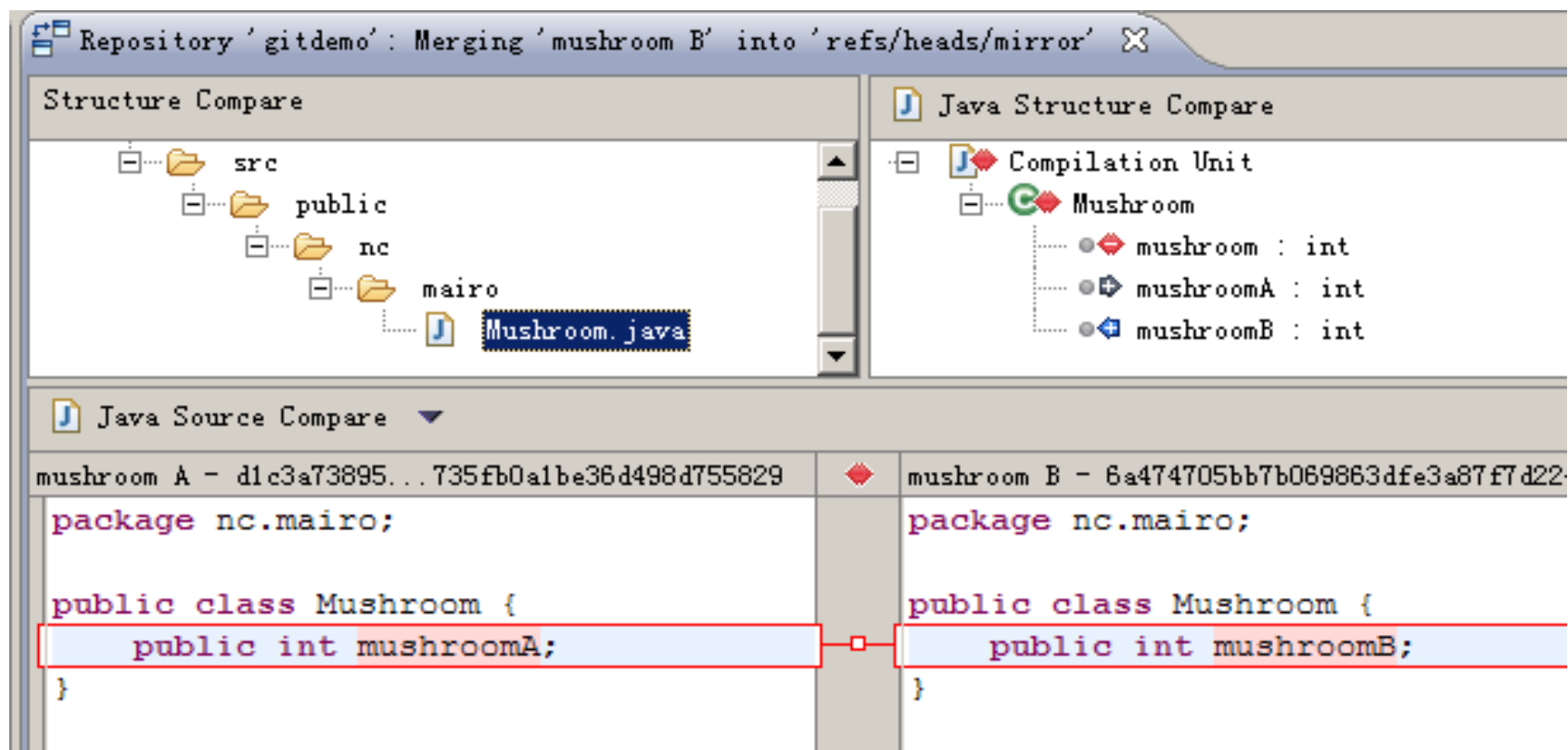
很明显自动合并的冲突文件不能直接使用，我们可以手动调整，右键发生冲突的文件，选择**Team -> Merge Tool**



第一项是将**GIT**自动合并过的文件和服务器端文件进行对比

第二项是用本地最新版本的文件和服务器端文件进行对比，建议用此项

- 接下来就是对比界面



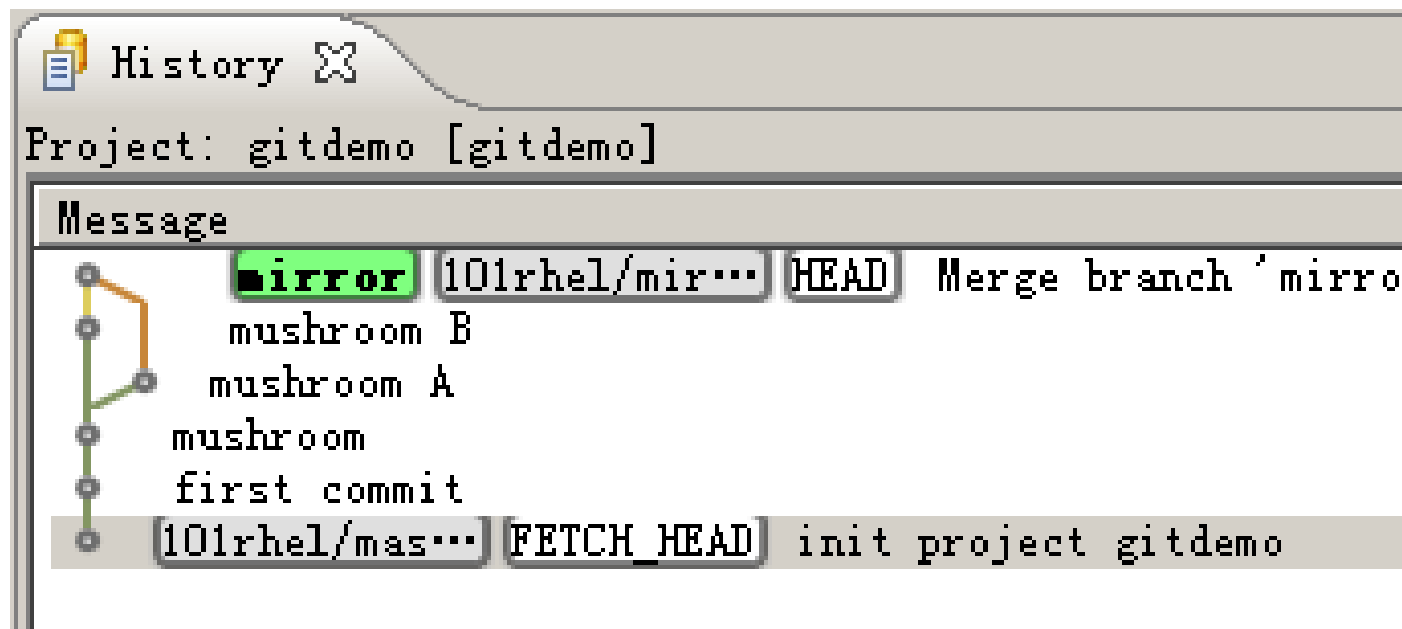
**Mairo哥哥**将冲突文件修改如下

```
package nc.mairo;  
  
public class Mushroom {  
    public int mushroomAB;  
}
```

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，Mairo弟弟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mushroomA的记录，然后是mushroomB的记录，最后历史分支合并。



Rebase和Merge操作最终的结果是一样的，但是实现原理不一样  
从上面的MairoBro例子可以知道pull大概对历史记录进行了怎样的合并操作，其实默认pull的操作就是一个分支的merge操作，如下图重现一下

**Mairo**弟弟的提交记录如下



**Mairo**哥哥的提交记录如下

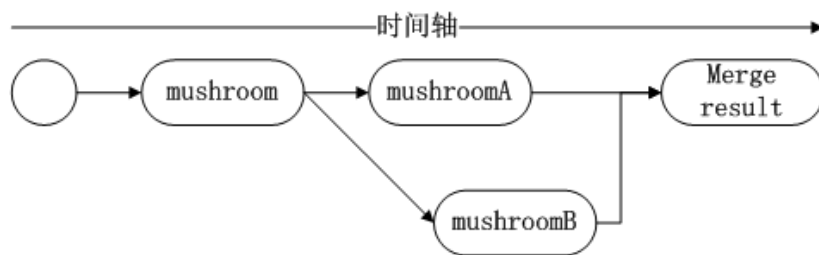


首先是Mairo弟弟把更新push到服务器，这样服务器端的记录就和Mairo弟弟本地的记录是一样的，接着Mairo哥哥执行pull操作，现在分析下pull是如何操作的

- I pull默认就是先把服务器端的最新记录更新到本地的Remote Tracking中对应的mirror分支
- I 接着对Local的mirror分支和Remote Tracking的mirror分支进行merge操作



- Merge操作后的结果就是会新增加一个merge记录节点，如下所示：



从上图可以看出，**mushroomA**是在**mushroomB**之前的，这个时间关系不取决于谁先执行**push**，而取决于本地仓库中谁先执行**commit**。所以**merge**会按照时间顺序严格的记录每一次**commit**



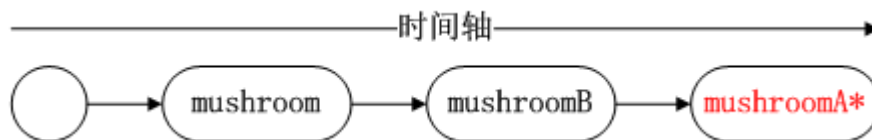
- 接下来看看rebase，其实rebase也是把两个分支进行合并的操作，当Mairo弟弟push更新后，服务器端的mirror分支的历史如下



**Mairo**哥哥本地的历史如下：

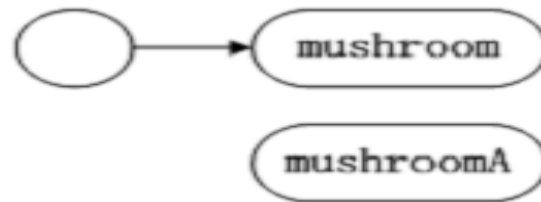


现在**Mairo**哥哥不是执行merge操作，而是执行rebase操作，最后结果如下

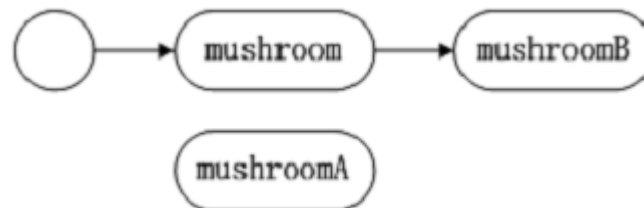


很明显的区别是没有出现分支的记录，而且注意到**mushroomA\***，请注意这个记录和**mushroomA**不是同一个记录，我们先分析下**rebase**操作下，**Mairo**哥哥的历史记录都做了哪些变化

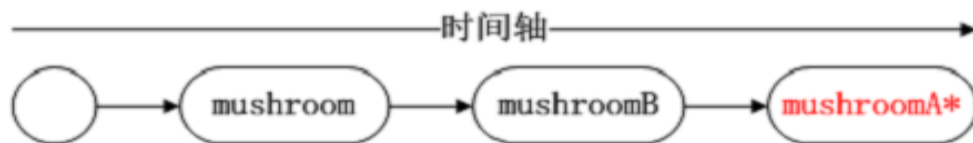
I 先将当前分支的更新部分保存到临时区域，而当前分支重置到上一次**pull**的记录



I 然后将服务器端的更新添加到当前分支，此时当前分支和服务端分支是一样的



I 最后将原分支的更新部分mushroomA提交到当前分支的后面，就是要在mushroomB的后面添加mushroomA的更新，当然此时更新记录已经不是之前的mushroomA了，如果出现冲突则使用对比工具解决冲突，最后记录变成mushroomA\*。



如果Mairo哥哥提交过mushroomA1、mushroomA2、mushroomA3，那么执行rebase后会对mushroomA1、mushroomA2、mushroomA3分别顺序执行上图所示的合并，最后记录为mushroomA1\*、mushroomA2\*、mushroomA3\*。很显然rebase操作更复杂，冲突的概率也更高，并且不是按照时间顺序记

# reference

- <http://wenku.baidu.com/view/929d7b4e2e3f5727a5e962a8.html>