

GitLab 基础使用教程

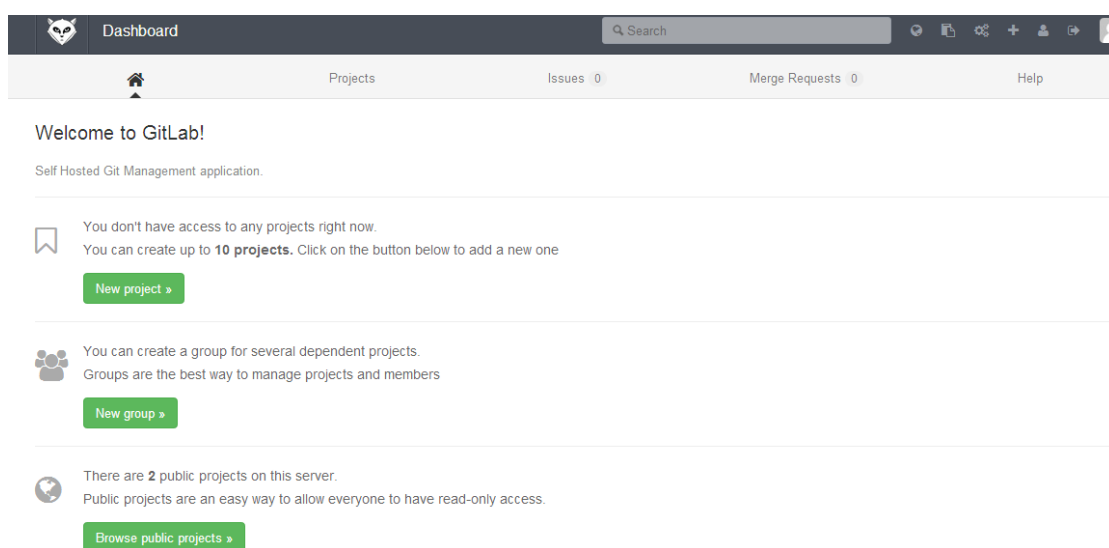
目录

1、GitLab 开发版本管理服务器	2
2、添加本地域名解析	2
3、修改密码并重新登录	3
4、Git Bash 下使用 Gitlab	3
5、MyEclipse 中使用 Gitlab	9
6、两个重要的文件.....	28

1、GitLab 开发版本管理服务器

GitLab，是一个利用 Ruby on Rails 开发的开源应用程序，实现一个自托管的 **Git** 项目仓库，可通过 **Web** 界面进行访问公开的或者私人项目。它拥有与 **Github** 类似的功能，能够浏览源代码，管理缺陷和注释。可以管理团队对仓库的访问，它非常易于浏览提交过的版本并提供一个文件历史库。团队成员可以利用内置的简单聊天程序（**Wall**）进行交流。它还提供一个代码片段收集功能可以轻松实现代码复用，便于日后有需要的时候进行查找。

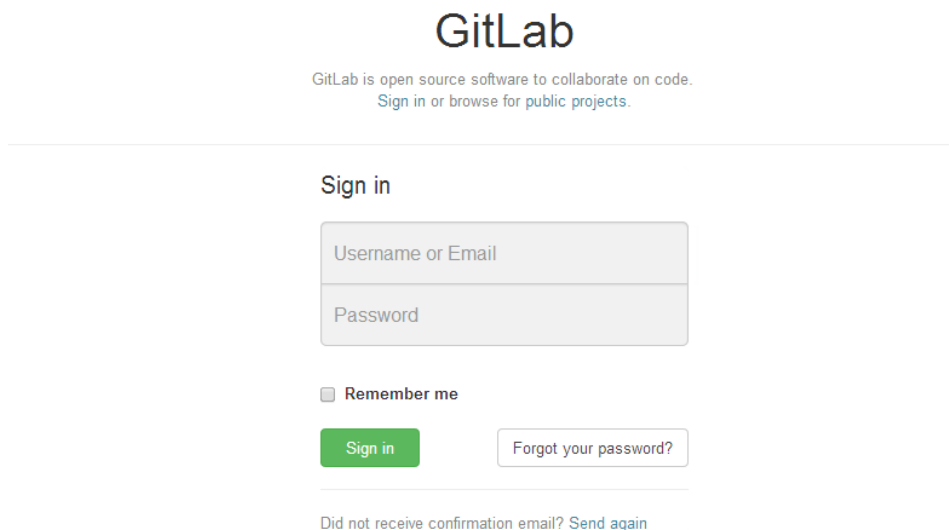
我们的 **GitLab** 开发版本管理服务器：<http://opc/>，通过 **Web** 浏览器访问我们的 **GitLab** 界面



2、添加本地域名解析

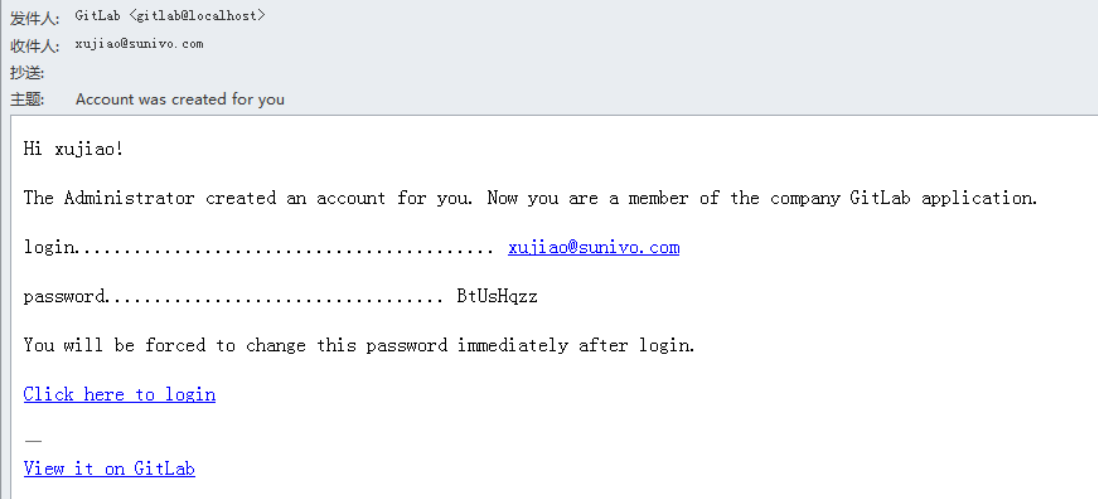
到系统目录 **C:\Windows\System32\drivers\etc** 下，拷贝出 **hosts** 文件，编辑 **hosts** 文件，添加“**192.168.222.89 opc**”行（，回车确保这一行 **IP** 映射生效，另存为时注意：1）无后缀名；2）保存编码为 **ASCII** 编码格式。

此时，在浏览器中直接输入 <http://opc/>”即可进入我们的 **GitLab**



3、修改密码并重新登录

管理员帮你注册后，你的注册邮箱会收到一封来自 GitLab 的邮件，邮件中包含登陆地址、首次登陆用的密码（系统随机生成的）

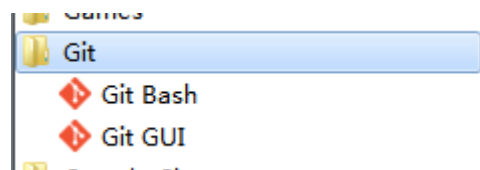


必须使用初始密码登陆后修改登陆密码，然后用新密码登陆即可。

4、Git Bash 下使用 Gitlab

1、在 Windows 下安装 Git 客户端软件

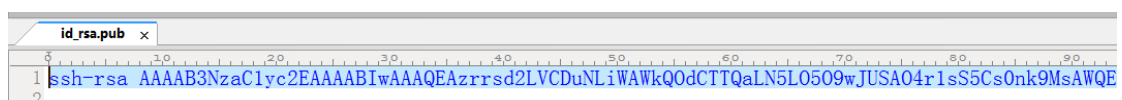
在你的 Windows 机器上，为了能够方便的检入检出项目，需要安装 Git 客户端软件，安装完之后会生成



2、生成 SSH 密钥验证身份

如果你想进行代码的上传与下载等操作，需要你把自己的 ssh key 导入到 gitlab 里，方法如下：（Linux 下生成方式相同，在自己的 home 目录下输入该命令即可）

打开上步安装生成的 Git Bash，输入 `ssh-keygen -t rsa -C "YOUR EMAIL ADDRESS"`，YOUR EMAIL ADDRESS 是你的邮箱地址，一直回车即可，此命令在 `C:\Users\<你的用户名>\.ssh` 目录下生成一对公私密钥，拷贝公钥（.pub 结尾的文件）的



内容，如我的 id_rsa.pub:

登陆你的 GitLab 账号之后，点击右上角的"Profile Setting" -> "SSH Keys"，输入 SSH Key 标题（可自定义），将拷贝的 id_rsa.pub 内容拷贝到 Key 中，"Add Key"即可。

Profile

Search

Account Emails 1 Password Notifications SSH Keys 0 Design Groups Profile settings

1 SSH Key

public key here. Read more about how to generate a key on the [SSH help page](#).

Title xjkey


Key

```
ssh-rsa
AAAAB3NzaC1yc2EAAAABIwAAAQEAzrrsd2LVCDuNLIWAWkQOdCTTQaLN5LO5O9wJUSAO4r1sS5Cs0nk9MsAWQEo2D6oH/UE02+HaCOTYuga7yTLE/Q5
hbot2TQjzjZwGzTG8FxiMowXK+a+NPO1/FjwCuP6AsLsgmJX8ggPlrj6wHHbEPscRTVtGID0gh8UHZN9jE2sQI0t6iKcA864DLIwbyUIXHbcFTLT8H2A2MHBf161c
NwuL0p7bbhzpPu57RS0M56VhtFjuso2QqCitA8IEYxUw0zGoOYPFSrycLd/PalgX82IFRpbArLoczc+k8F6xZ9987VKLLpi9k1A4J2Eu/XhQ5l/px2dfbBxZ6tHyJfw==
xujiao@suniv.com
```


Add key Cancel

3、在 Gitlab 界面创建自己的项目

点击右上角的 New Project，填写相关的 Project 相关信息后，选择 "Create Project"



New Project



Project name

hdconf

Namespace

xujiao

[Customize repository name?](#)

[Import existing repository?](#)

Description (optional)

it's hadoop configuration files.

Visibility Level (?)

☐ Private

Project access must be granted explicitly for each u

☒ Internal

The project can be cloned by any logged in user.


☐ Public

The project can be cloned without any authentication

Create project

[Homepage](#) [Blog](#) [@gitlabhq](#)

然后会看到如下信息，根据“Create Repository”创建项目库，右上角为 SSH 地址：




xujiao / hdconf

[Issues](#) 0

[Merge Requests](#) 0

[Wiki](#)



xujiao / hdconf

[SSH](#) [HTTP](#) [git@mycloud:xujiao/hdconf.git](#)

it's hadoop configuration files. — [Edit](#)

Git global setup:

```
git config --global user.name "xujiao"
git config --global user.email "xujiao@sunivo.com"
```

Create Repository

```
mkdir hdconf
cd hdconf
git init
touch README
git add README
git commit -m 'first commit'
git remote add origin git@mycloud:xujiao/hdconf.git
git push -u origin master
```

Existing Git Repo?

```
cd existing_git_repo
git remote add origin git@mycloud:xujiao/hdconf.git
git push -u origin master
```

4、上传项目

在 GitBash 中上传：

新建目录：mkdir testwcmc & cd testwcmc

新建测试文件：hello.java，写入一些测试内容

上传过程如下：

步骤说明：git status 命令检查当前的 git 状态，是常用命令

- 1) Git init：初始化项目库（新建项目时使用）；
- 2) Git add 文件或目录：添加文件或目录；
- 3) Git commit -am “此次提交的说明”：提交到本地；
- 4) Git remote add origin 项目的 SSH 地址：远程添加到新项目中；
- 5) Git push origin master：将本地新增文件上传到 GitLab 中；

```
xujiao@XUJIAO-PC ~/hadoopconf
$ git init
Initialized empty Git repository in c:/Users/xujiao.SUNIVO/hadoopconf/.git/

xujiao@XUJIAO-PC ~/hadoopconf (master)
$ git add core-site.xml
core-site.xml

xujiao@XUJIAO-PC ~/hadoopconf (master)
$ git add core-site.xml
core-site.xml

xujiao@XUJIAO-PC ~/hadoopconf (master)
$ git add core-site.xml
core-site.xml

xujiao@XUJIAO-PC ~/hadoopconf (master)
$ git add core-site.xml
core-site.xml

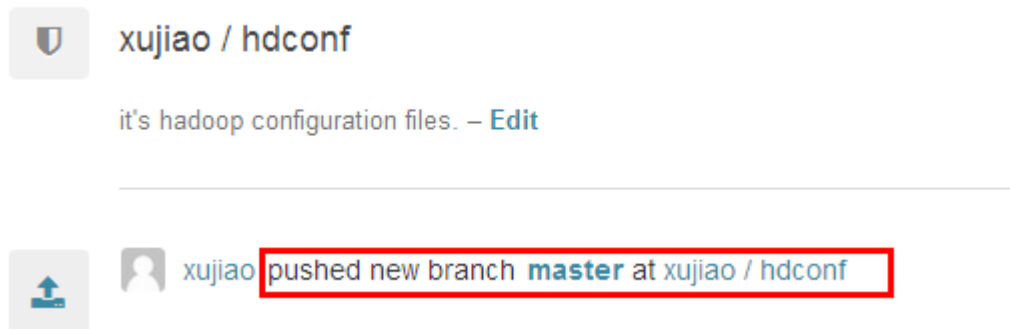
xujiao@XUJIAO-PC ~/hadoopconf (master)
$ git commit -m"CONF"
[master (root-commit) e5eed8e] CONF
1 file changed, 13 insertions(+)
create mode 100644 core-site.xml

xujiao@XUJIAO-PC ~/hadoopconf (master)
$ git remote add origin git@mycloud:xujiao/hdconf.git

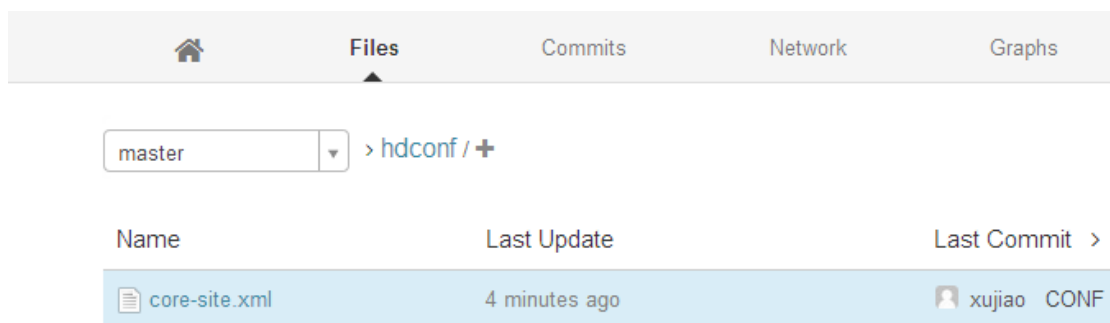
xujiao@XUJIAO-PC ~/hadoopconf (master)
$ git push -u origin master
Counting objects: 3, done.
Delta compression using up to 2 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 402 bytes | 0 bytes/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To git@mycloud:xujiao/hdconf.git
 * [new branch]      master -> master
Branch master set up to track remote branch master from origin.

xujiao@XUJIAO-PC ~/hadoopconf (master)
$
```

然后在 GitLab 上刷新刚刚新建的项目可以看到：



进入该项目的“Files”，可以看到刚刚上传的 core-site.xml



点击该文件可以查看文件内容



5、分支操作

分支可以看做是某个项目的版本。

分支操作主要包括：查看、新建、切换、删除，详见下图：

```
xujiao@XUJIAO-PC ~/hadoopconf (master)
$ git branch 查看分支
develop
* master

xujiao@XUJIAO-PC ~/hadoopconf (master)
$ git branch test 新建分支

xujiao@XUJIAO-PC ~/hadoopconf (master)
$ git branch
develop
* master
test

xujiao@XUJIAO-PC ~/hadoopconf (master)
$ git checkout test 切换当前所在分支
Switched to branch 'test'

xujiao@XUJIAO-PC ~/hadoopconf (test)
$ git branch
develop
master
* test
```

- 1) git branch 为查看当前分支，第一条 git branch 命令可以看到有 master 和 develop 两个分支，且 master 前为带星号绿色，表示当前所处的分支；
- 2) git branch test 新建名为 test 的分支；
- 3) git checkout test 将当前所在分支切换到 test 下；

准备在 test 分支中添加新的文件或修改，提交时用 git push origin **test** 即可：

```
xujiao@XUJIAO-PC ~/hadoopconf (test)
$ git push origin test
Counting objects: 4, done.
Delta compression using up to 2 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 271 bytes | 0 bytes/s, done.
Total 3 (delta 1), reused 0 (delta 0)
To git@mycloud:xujiao/hadoopconf.git
 * [new branch]      test -> test
```

查看 GitLab 中的分支情况：

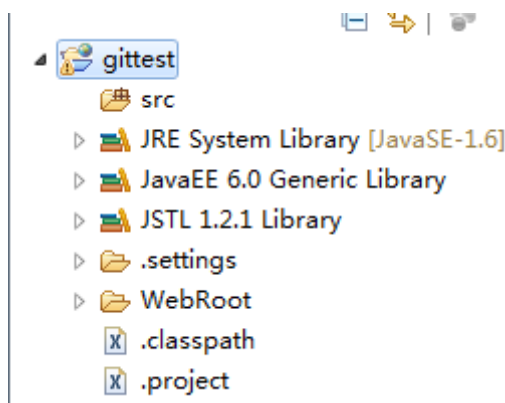
test	> hdconf / +		
Branches	Last Update	Last Commit	> 9ac73a8cc8f
develop	about 3 hours ago	xujiao	another
master	about 3 hours ago	xujiao	CONF
test	about 3 hours ago	xujiao	two files
forth.txt			
test.txt	3 minutes ago	xujiao	test version
third.txt	about 3 hours ago	xujiao	two files

5、MyEclipse 中使用 Gitlab

最新的 MyEclipse2014 已自带 Git，安装文件以及破解包在 ftp 位置（打开“计算机”或“我的电脑”）：ftp://192.168.222.2/released_by_others/xujiao/ 下的“程序软件”中，请按照破解包中的安装步骤自行安装及破解。

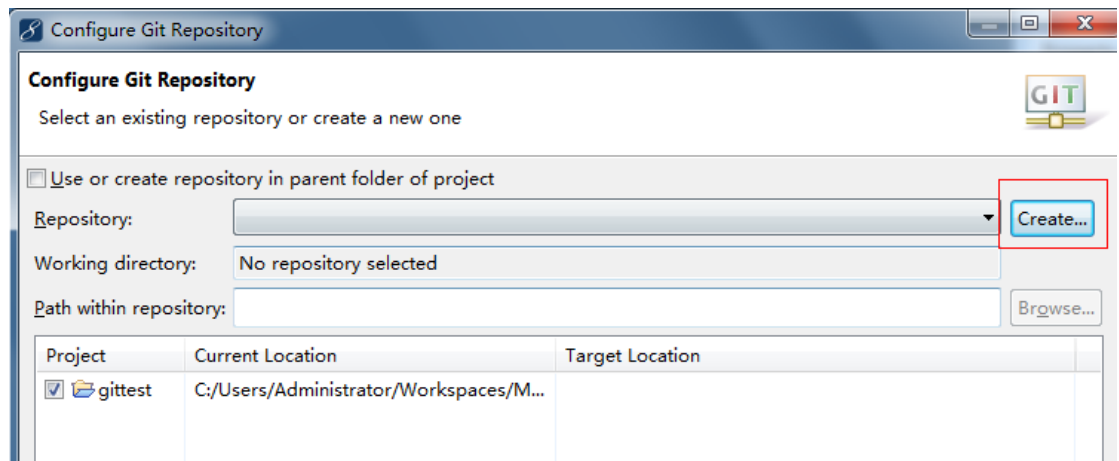
1 新建项目上传至 GitLab

1) 在 MyEclipse 中新建一个 Project，此处新建测试用的 gittest

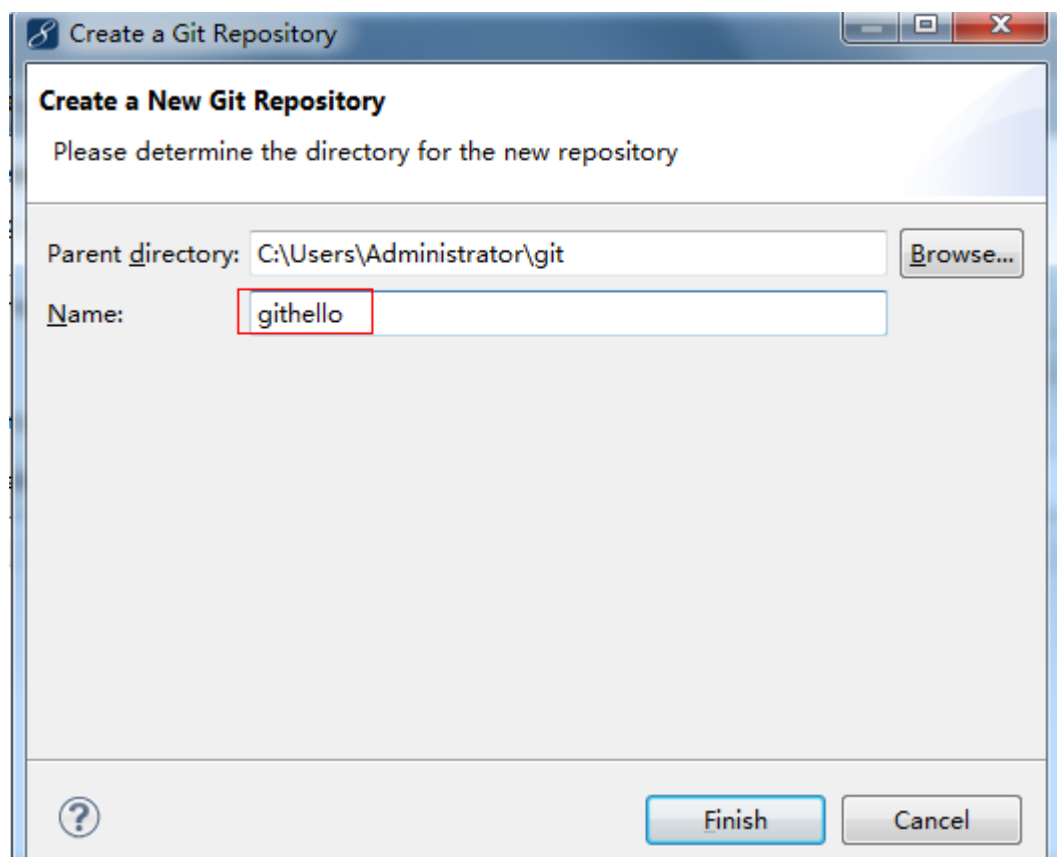


2) 新建 gittest 项目的 git 仓库

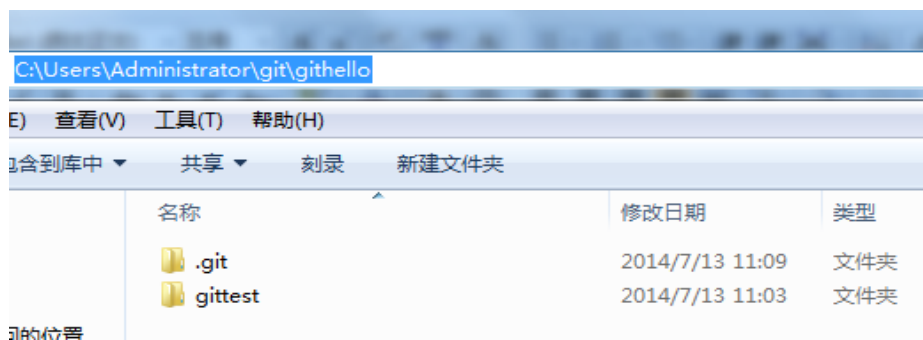
在项目上右键 -> Team -> Share Project -> Git -> Next



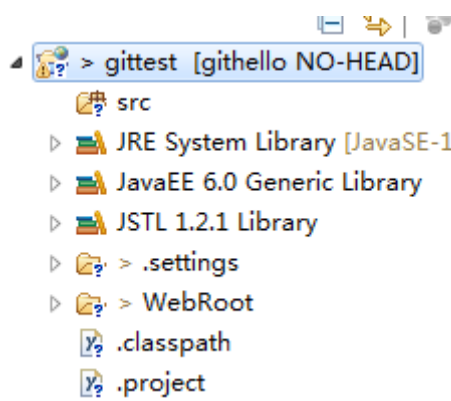
Create -> 自定义仓库名称 -> Finish



在 C:\Users\Administrator\git 目录中可以看到 githello 仓库，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



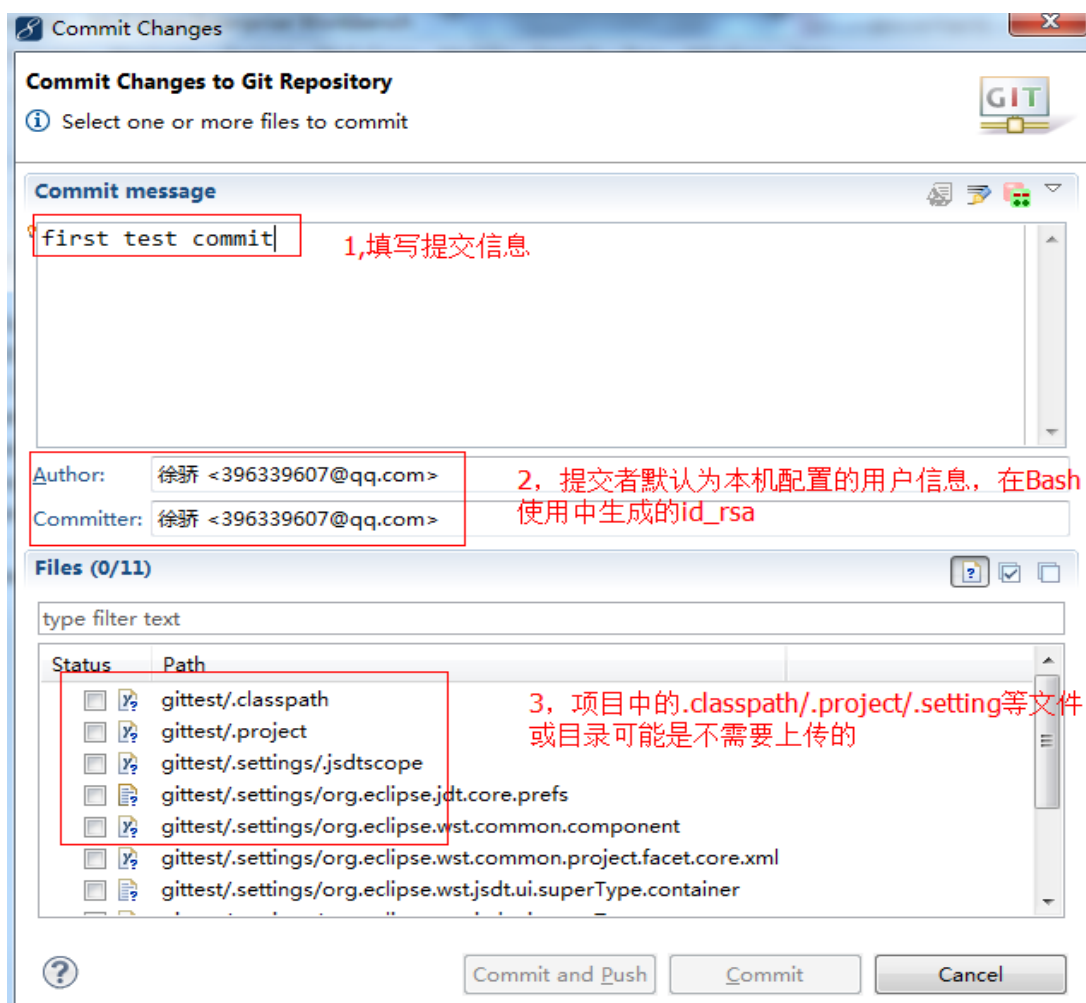
同时，eclipse 中的 project 也建立 git 版本控制，此时未创建分支，处于 NO-HEAD 状态



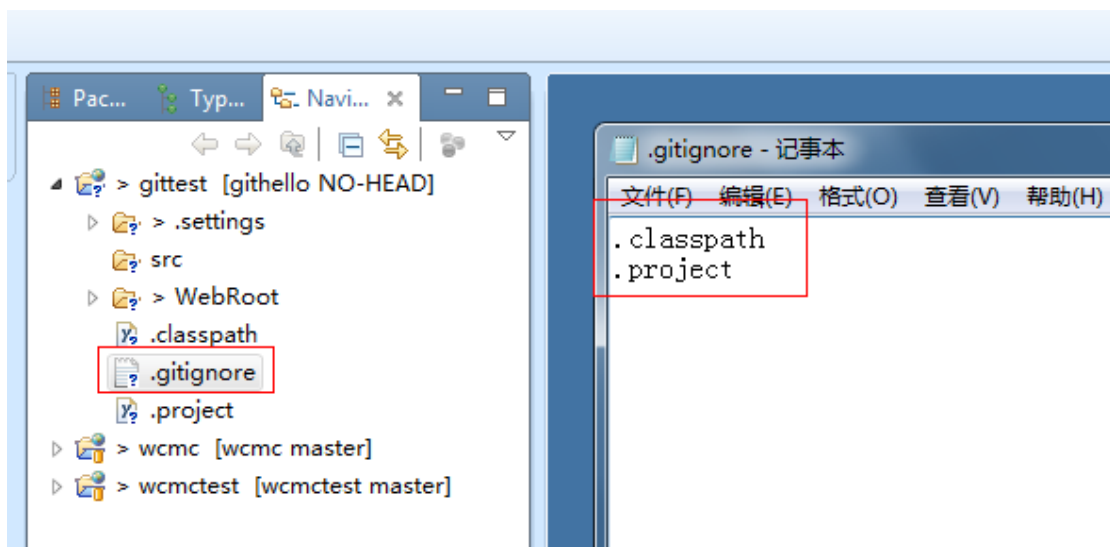
文件夹中的符号“?”表示此文件夹处于 untracked 状态，这样就成功创建 GIT 仓库。

3) 配置.gitignore 文件

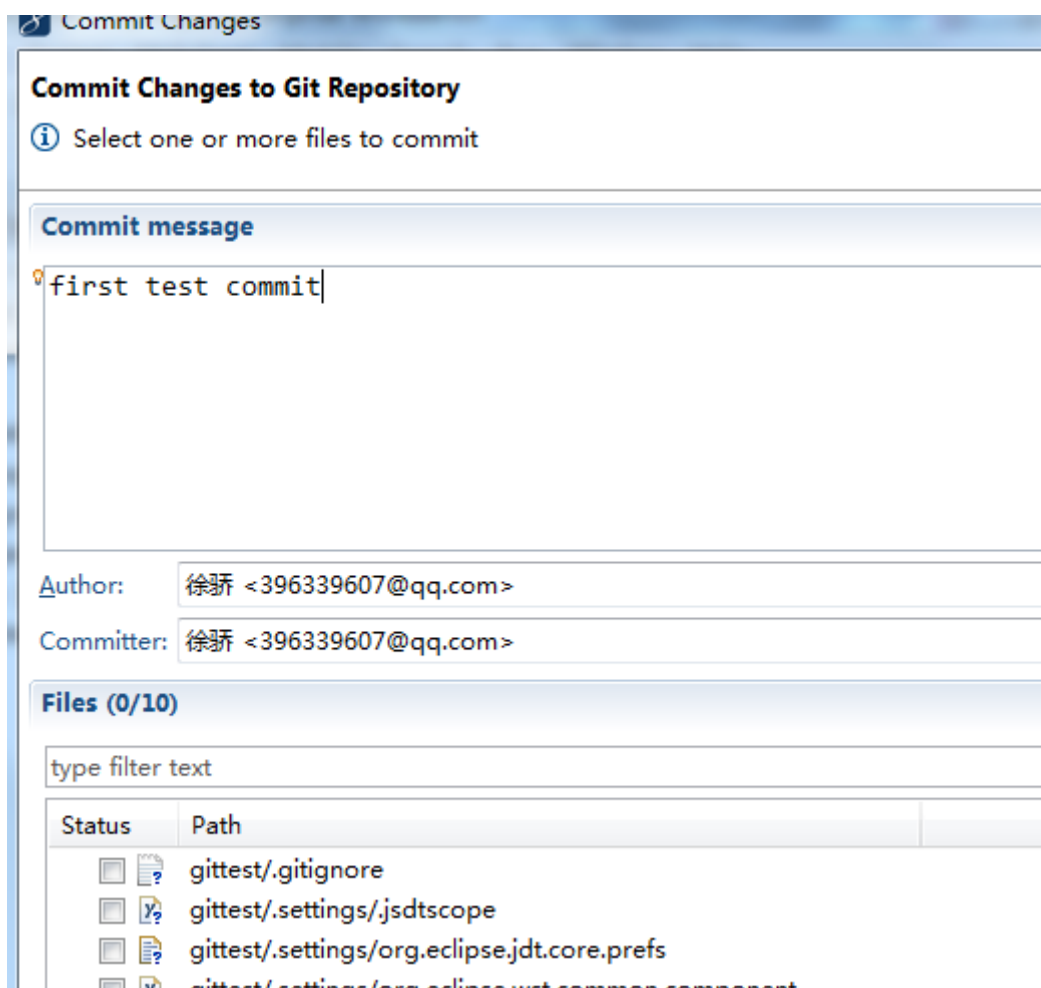
尝试提交 gittest 项目，右键 -> Team -> Commit



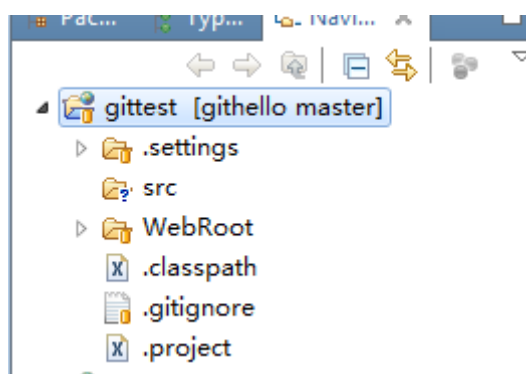
我们可以通过配置.gitignore来排除不需要上传的文件,打开 Navigator 窗口,在 project 根目录中添加.gitignore 文件,将需要排除控制的目录写入.gitignore 文件中



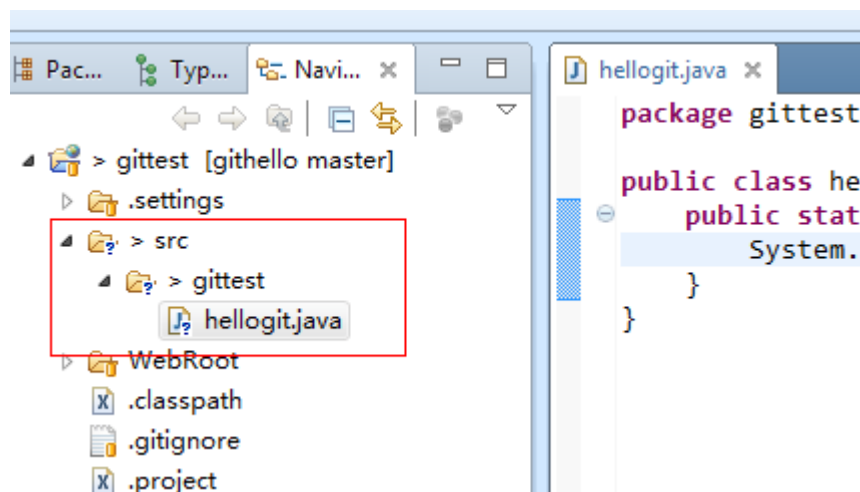
再次 Commit 可以看到.gitignore 的文件被过滤掉



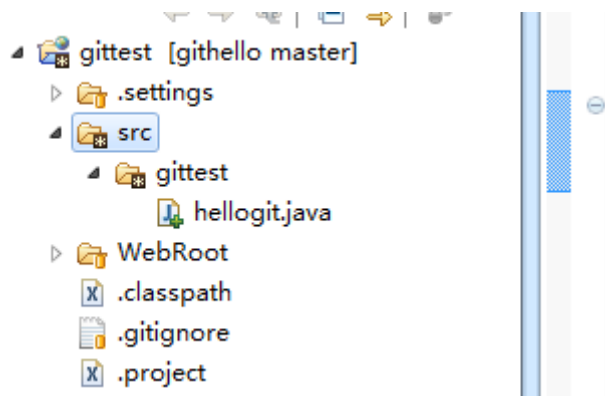
勾选文件，然后 Commit，可以看到首次提交后，会自动生成 master 分支



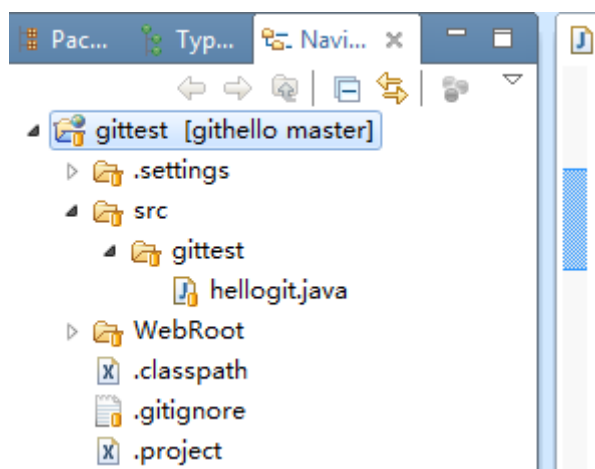
新建一个 hellogitjava 文件，可以看到 src 目录带有一个问号图标，这表示处于 untracked 状态，即 git 没有对此文件进行监控



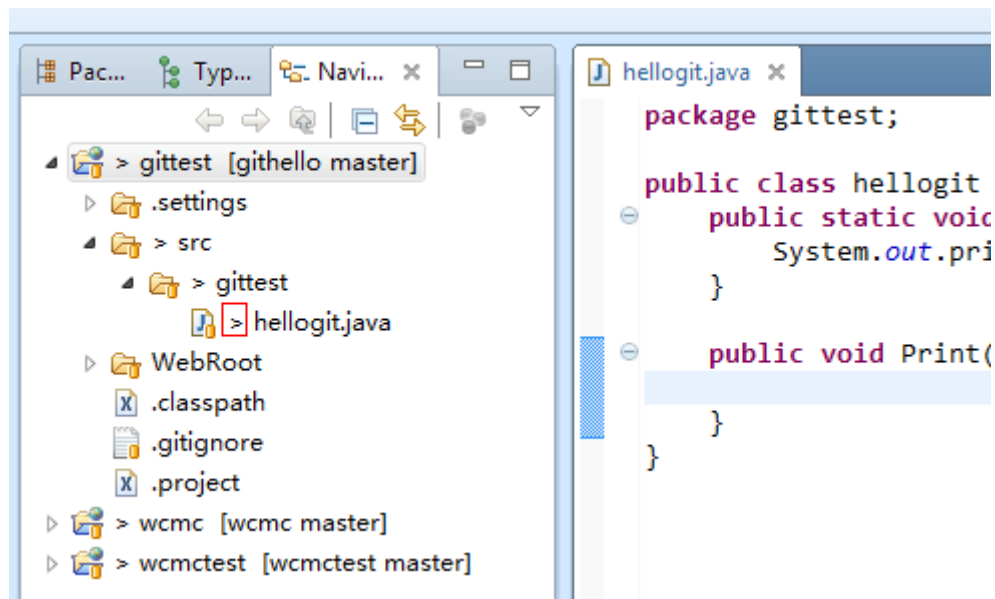
通过 Team -> Add to index 可以将文件加入 git 索引，进行版本监控，可以看到图标显示也有了变化（EGIT 中只要 Commit 就可以默认将 untracked 的文件添加到索引再提交更新，不需要分开操作），也可以通过 Team -> Untrack 将文件从索引控制中排除。



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

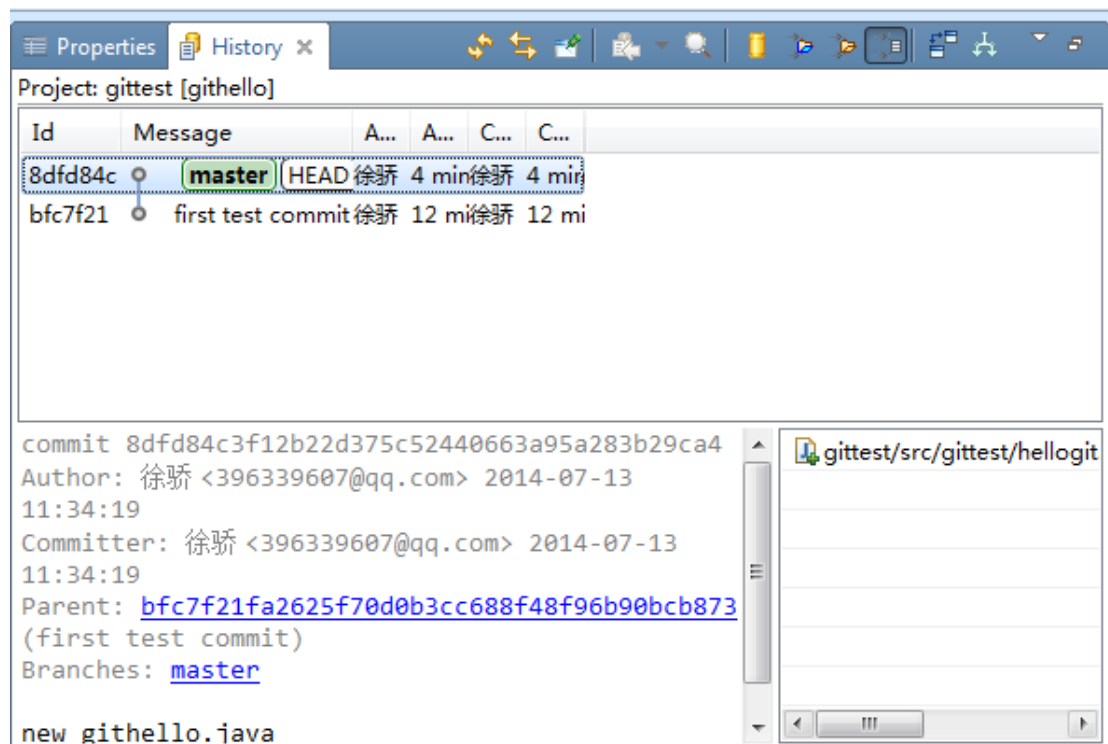


一旦修改文件，文件将处于 modified 状态，可以从 ">" 变化看出来



查看历史提交记录

Team -> Show in history 可以查看版本历史提交记录



4) 上传至 GitLab

在 GitLab 上新建一个 githello 的仓库

Team -> Remote -> Push 填写相关信息后 -> next -> Add All Branches Specs -> Finish

Push to Another Repository

Destination Git Repository

Enter the location of the destination repository.

Location

URI:

Host:

Repository path:

Connection

Protocol:

Port:

Authentication

User:

Password:

Store in Secure Store ☐

Push to: ssh://git@opc/xujiao/wcmc.git

Push Ref Specifications

Select refs to push.

Add create/update specification

Source ref: Destination ref: + Add Spec

Add delete ref specification

Remote ref to delete: x Add spec

Add predefined specification

Add Configured Push Specs
Add All Branches Spec
Add All Tags Spec

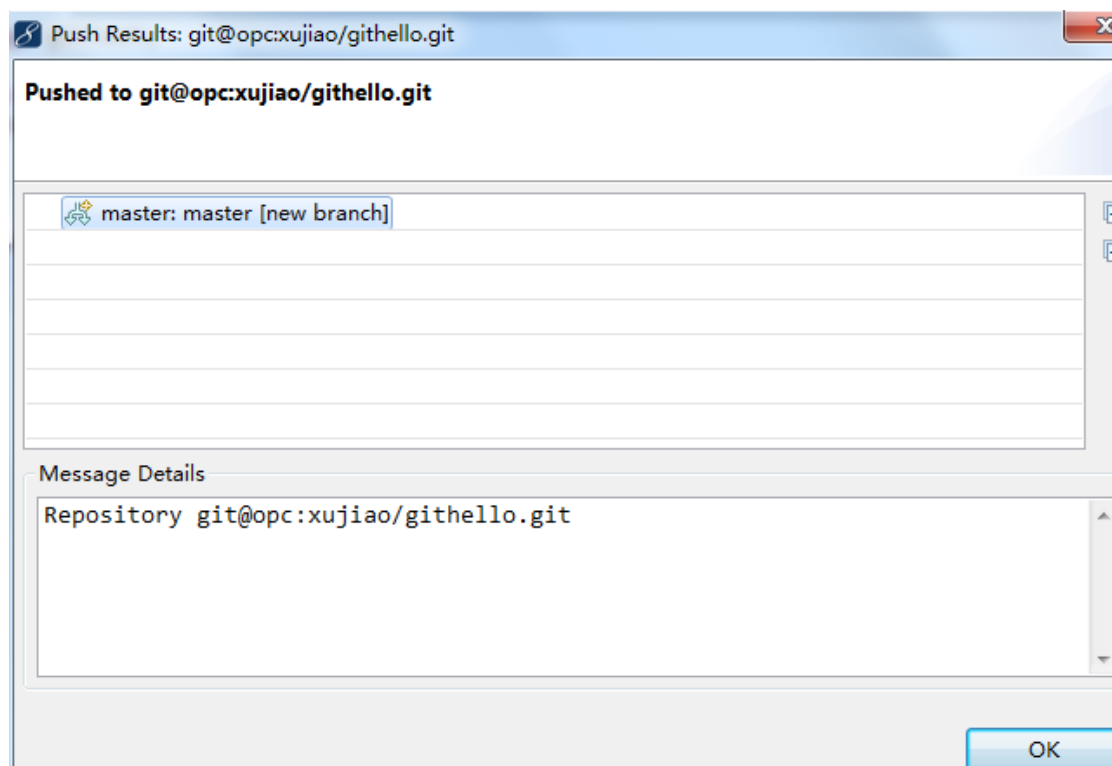
Specifications for push

Mode	Source Ref	Destination Ref	Force Update	Remove
+ Update	refs/heads/*	refs/heads/*	<input type="checkbox"/>	

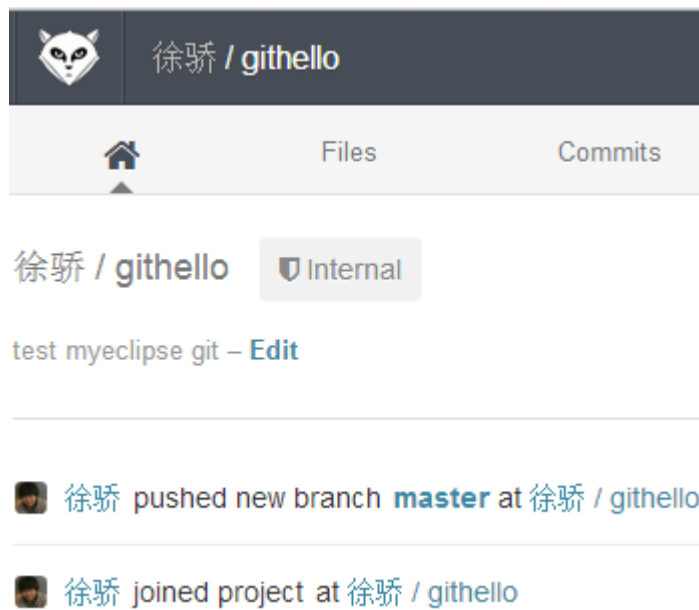
Force Update All Specs
 Remove All Specs

?
< Back
Next >
Finish
Cancel

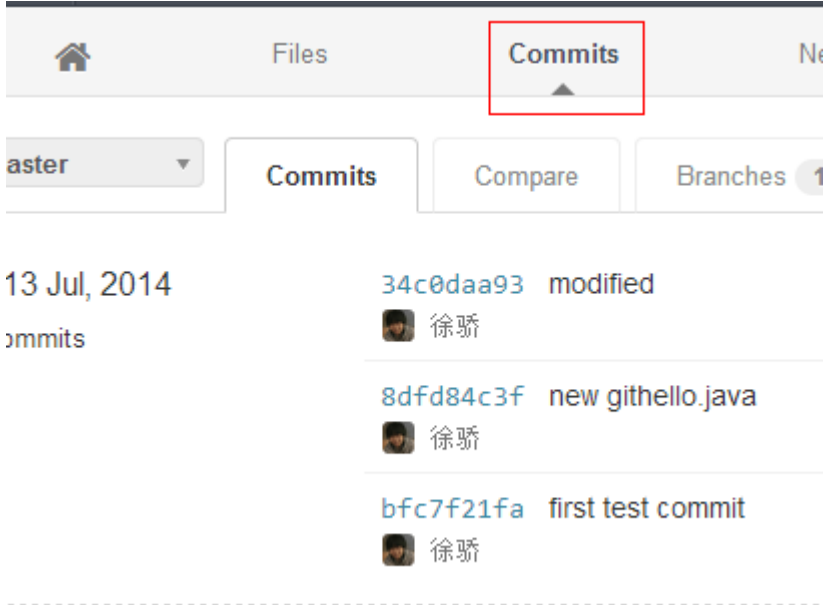
完成之后



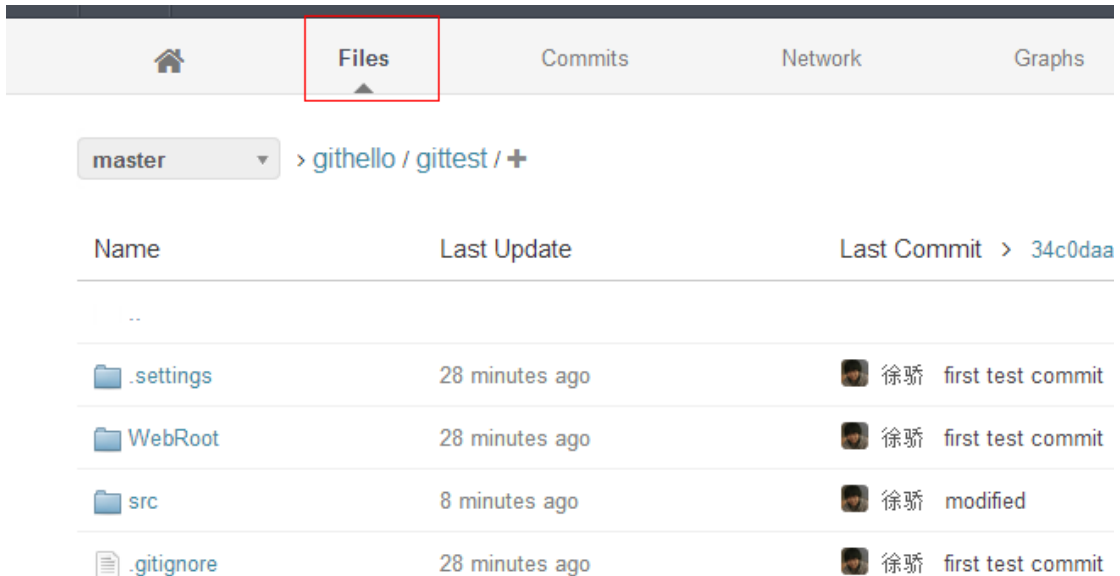
还可以通过在 GitLab 中查看相关信息











Commits



Files



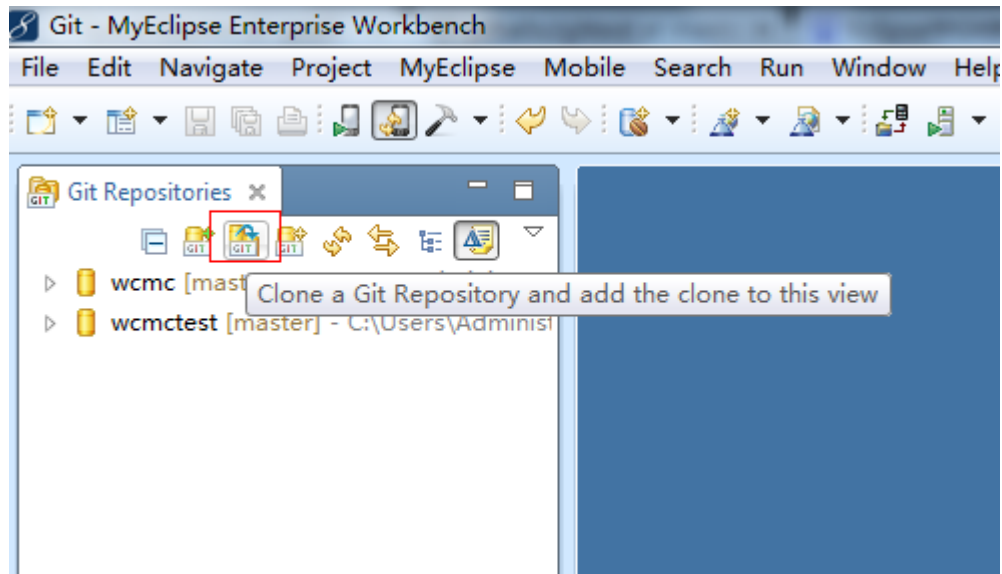
Name	Last Update	Last Commit > 34c0daa
..		
 .settings	28 minutes ago	 徐骞 first test commit
 WebRoot	28 minutes ago	 徐骞 first test commit
 src	8 minutes ago	 徐骞 modified
 .gitignore	28 minutes ago	 徐骞 first test commit

2 获得 GitLab 中已有项目

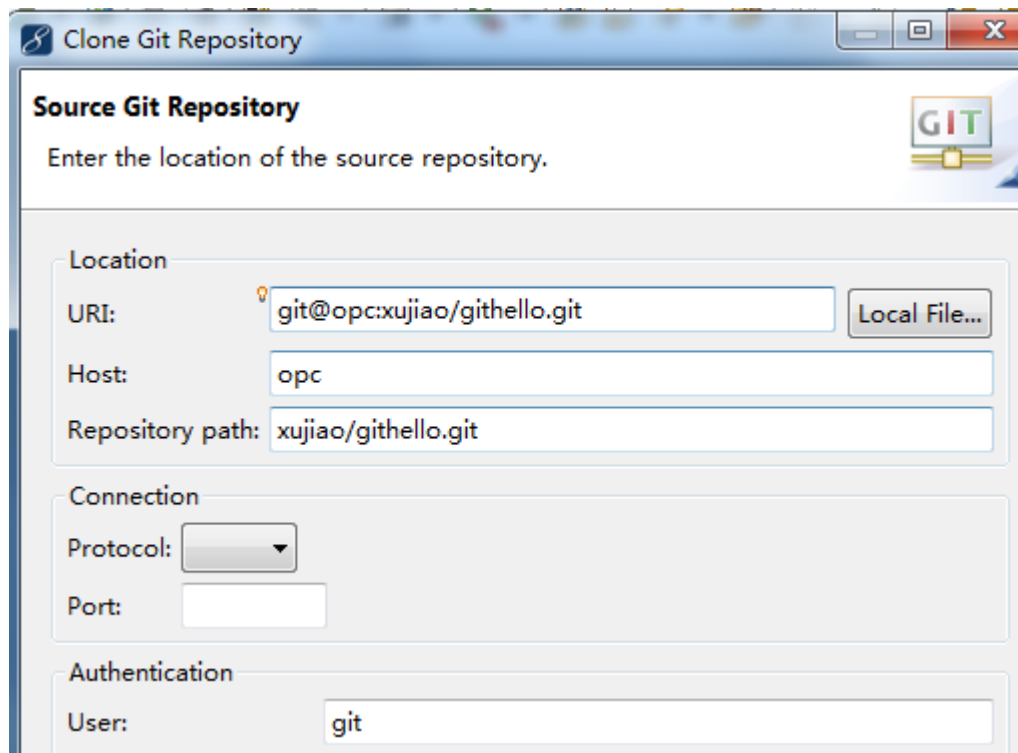
为演示，先删除刚刚在 Eclipse 创建的 gittest 项目

1) 打开 GIT 资源库窗口，选择克隆资源库

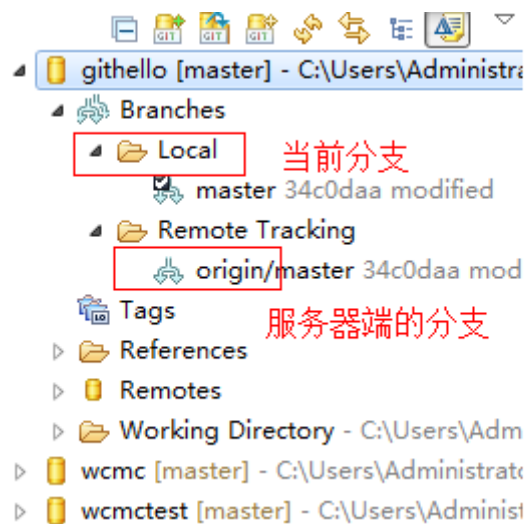
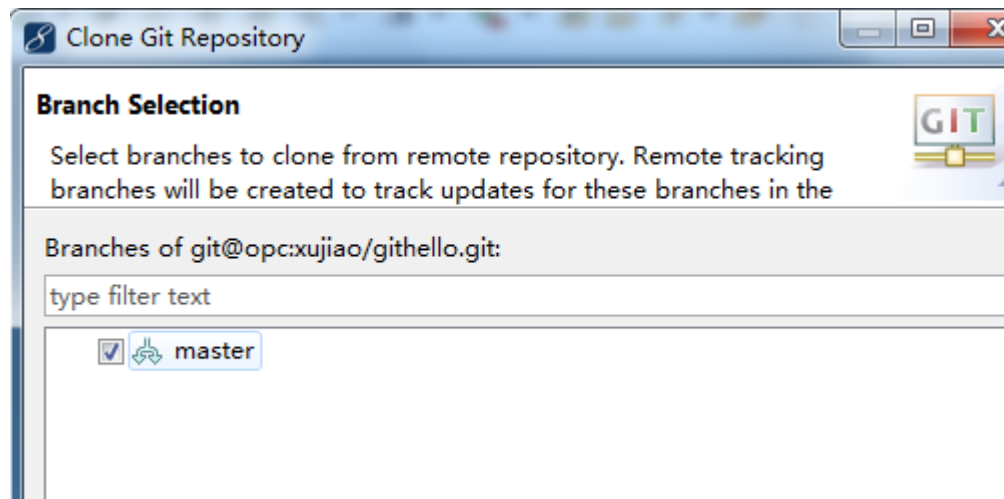
Window -> Open Perspective -> Other 中选择 Git



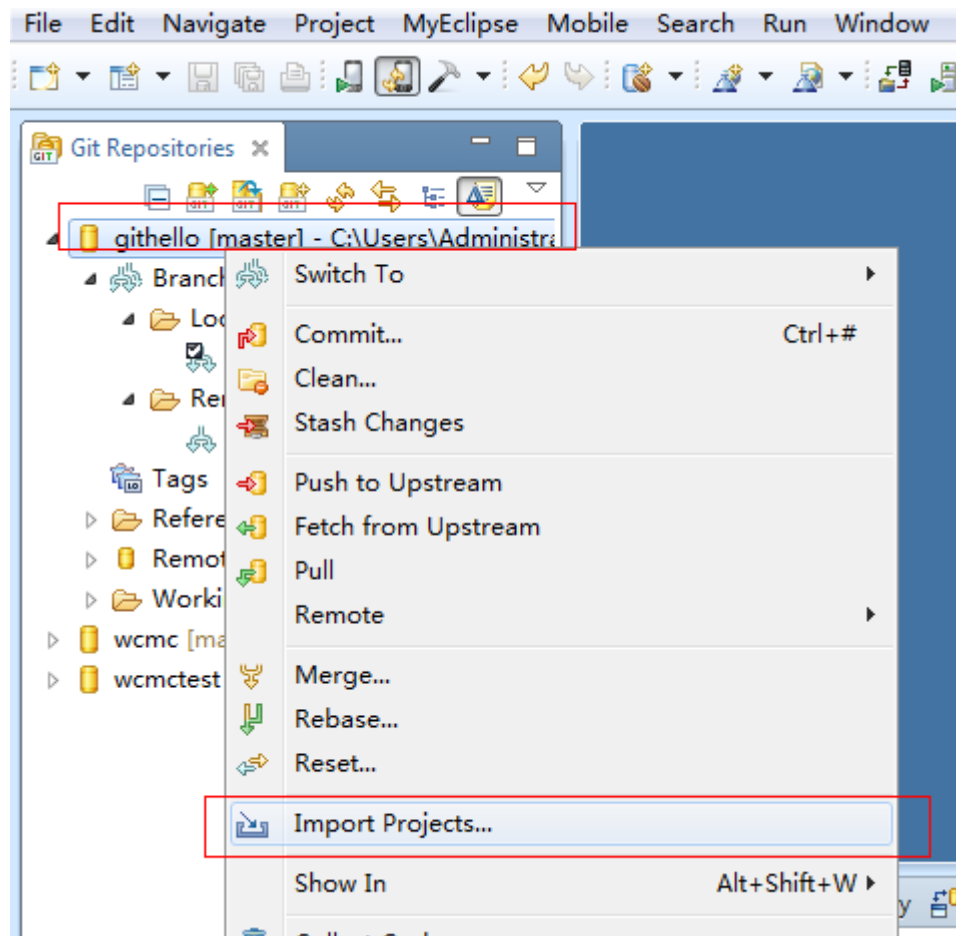
输入 GitLab 中的 SSH 地址



选择分支，由于此处只有 master 分支，所以 Clone master 分支，然后 finish



2) 完成 Git 的本地克隆，接下来需要将仓库检出为 Web Project

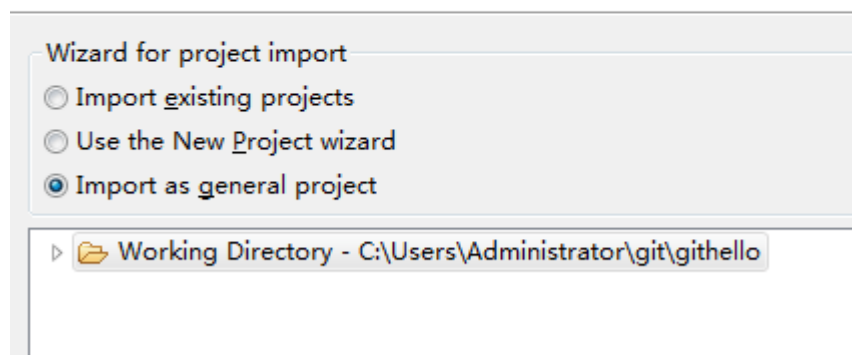


选择 Import as general project

Import Projects from Git Repository C:\Users\Administrator\git\githello

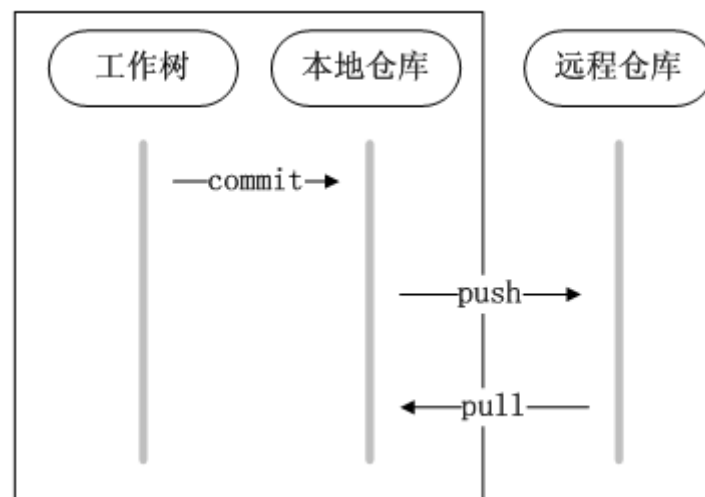
Select a wizard to use for importing projects

Depending on the wizard, you may select a directory to determine the

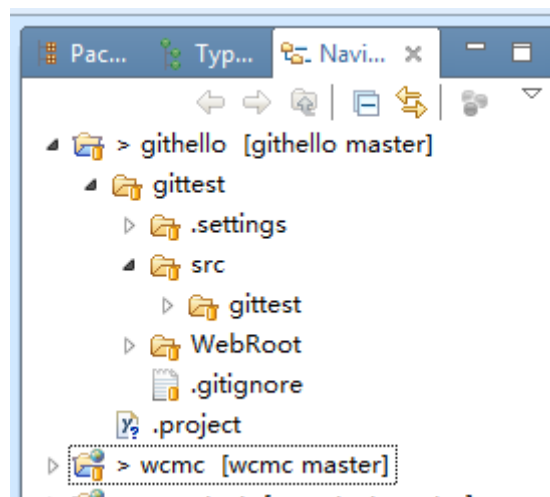


3 推送远程仓库

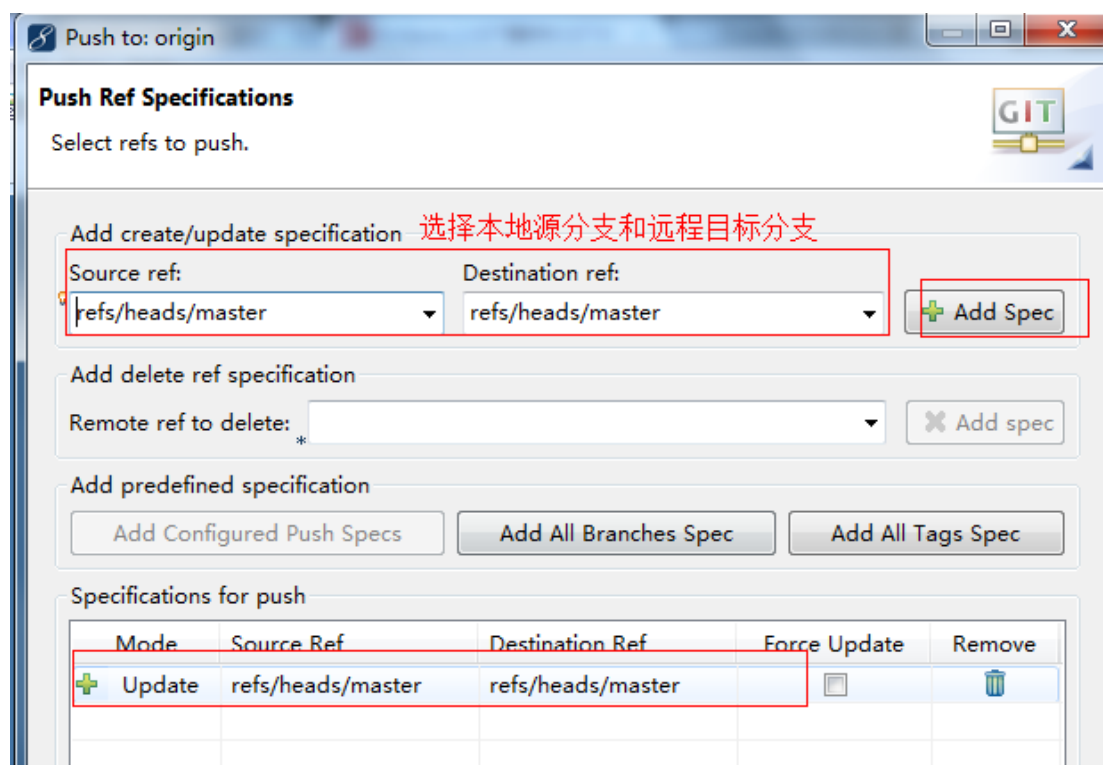
克隆服务器端仓库后，会在本地建立一个一样的仓库，称本地仓库。在本地进行 commit 操作将把更新提交到本地仓库，然后可以将服务器端的更新 pull 到本地仓库进行合并，最后将合并好的本地仓库 push 到服务器端，这样就进行了一次远程提交。



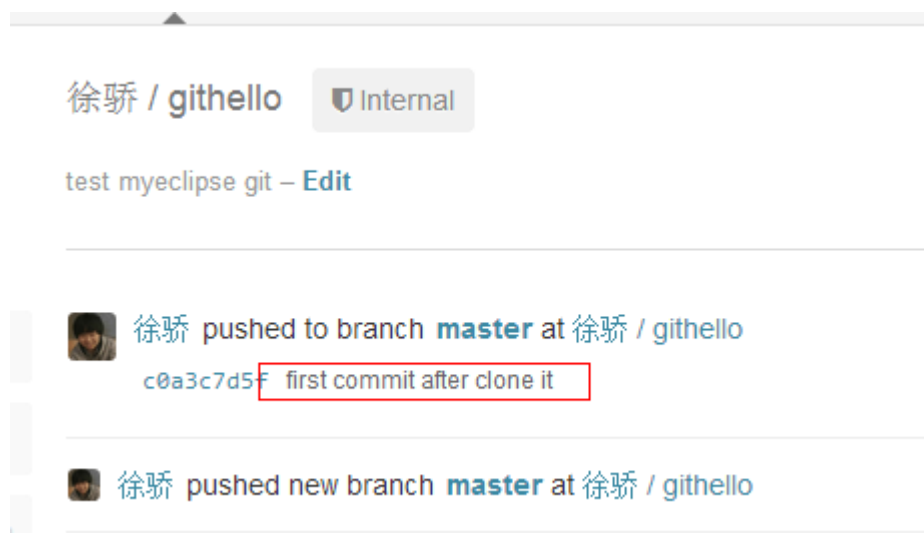
经过克隆并检出后，会得到克隆后的 master 分支项目



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



推送完成之后在 GitLab 上可以看到



4 推送冲突解决

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。

假设现在服务器上的 `hellogit.java` 已经被更新，添加了第 9 行代码

```
hellogit.java 194 Bytes
1  package gittest;
2
3  public class hellogit {
4      public static void main(String[] args) {
5          System.out.println("Hello Git!");
6      }
7
8      public void Print(){
9          System.out.println("I'm New");
10     }
11 }
```

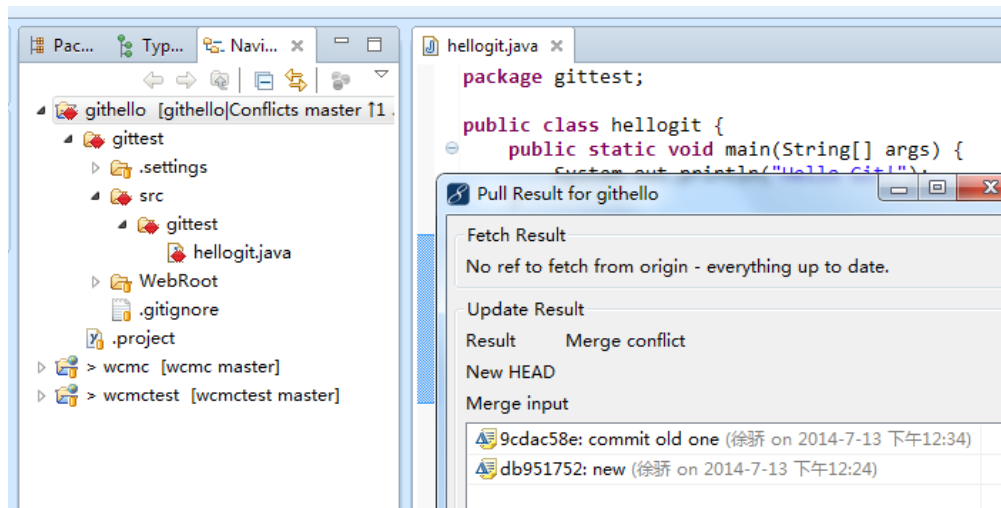
而我在第二步克隆下来的版本后，添加的语句是这样的：

```
hellogit.java x
package gittest;

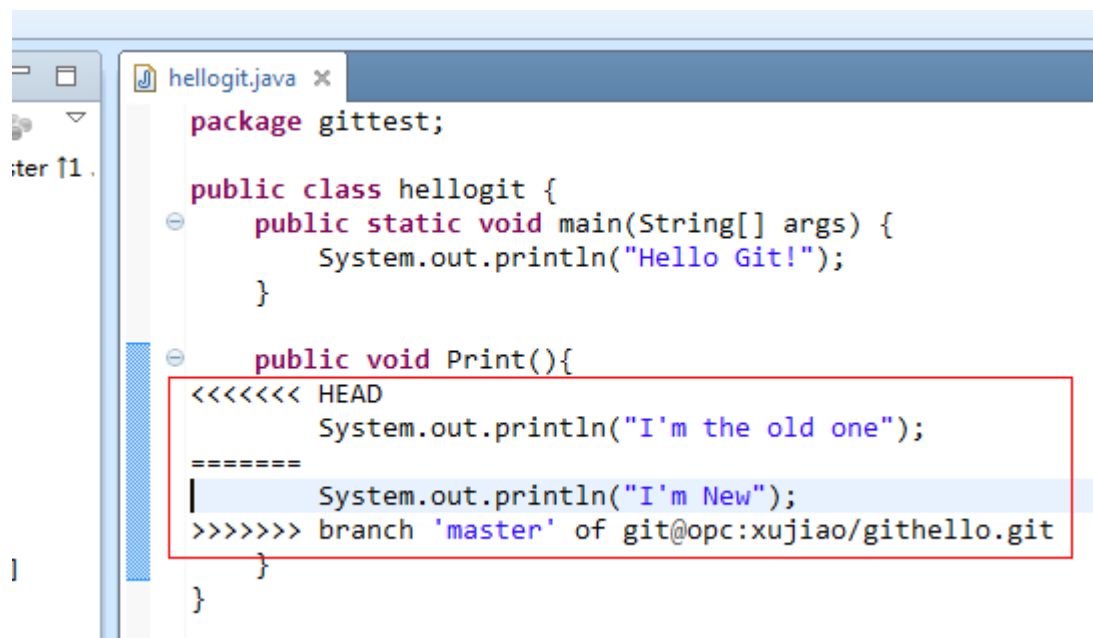
public class hellogit {
    public static void main(String[] args) {
        System.out.println("Hello Git!");
    }

    public void Print(){
        System.out.println("I'm the old one");
    }
}
```

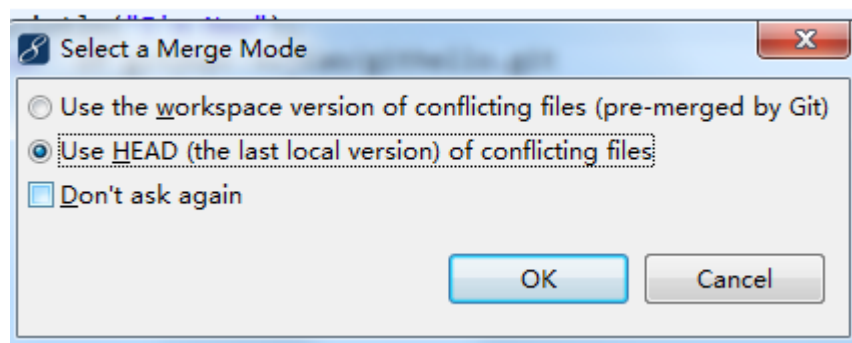
先将修改进行 Commit，然后使用 pull 来合并本地仓库和远程仓库，将发行文件出现冲突，此时 GIT 会自动合并冲突的文件，如下图所示：



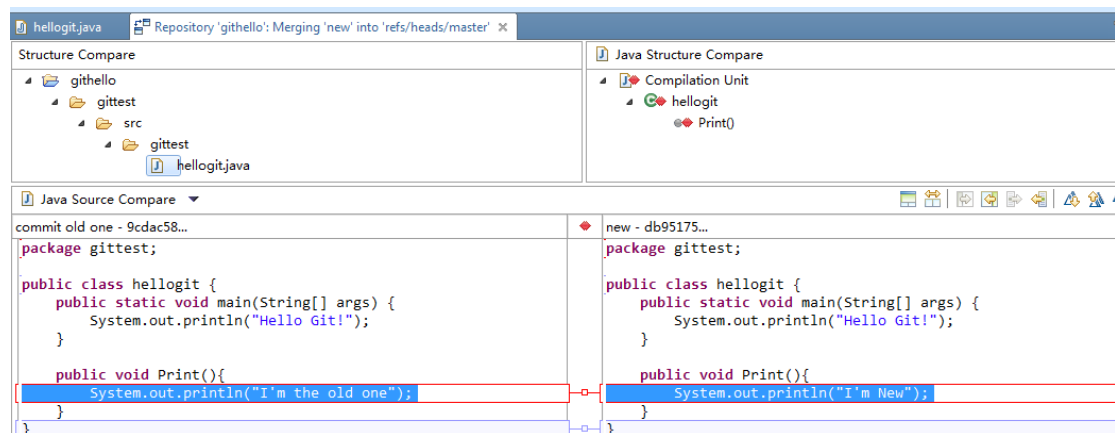
合并结果如下



显然这样的合并结果并不能直接使用，我们可以手动调整，右键发生冲突的文件，选择 Team -> Merge Tool

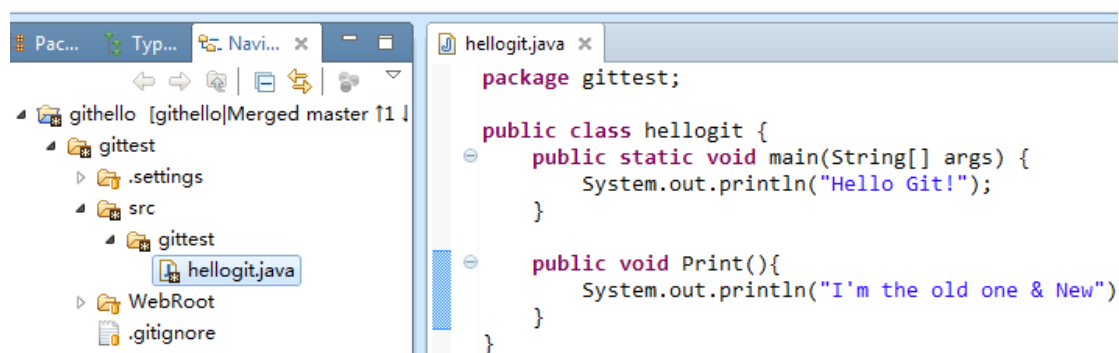


第一项是将 GIT 自动合并过的文件和服务器端文件进行对比，第二项是用本地最新版本的文件和服务器端文件进行对比，建议用此项，接下来就是熟悉的对比界面

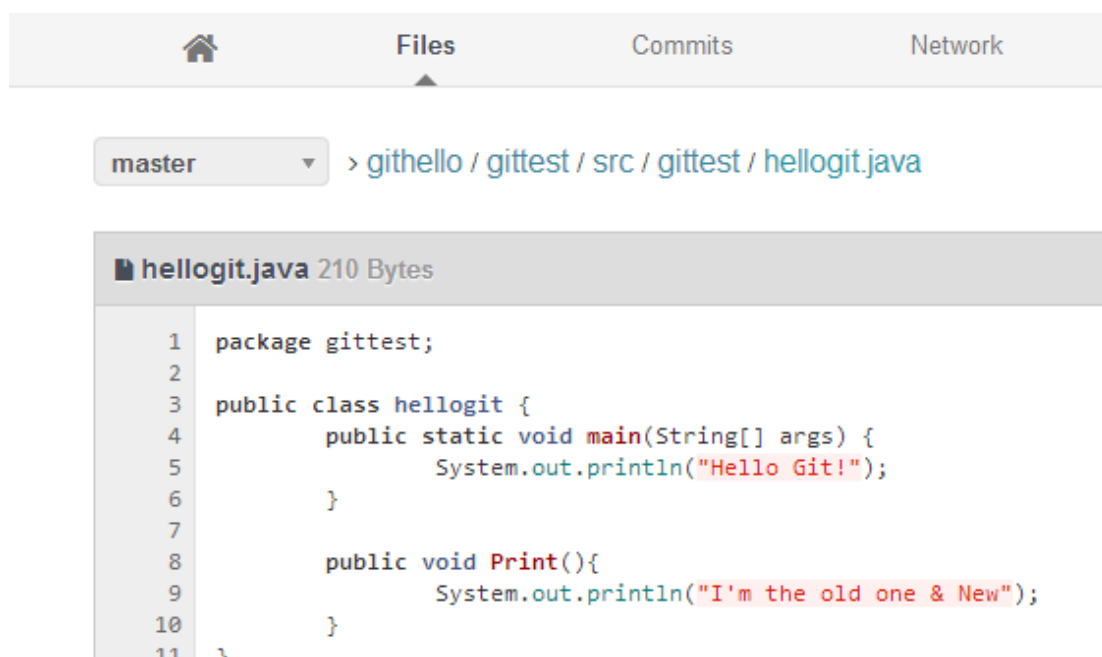


手动对冲突进行合并，此处替换成服务器上最新的 “old one & New”

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。



然后 Commit -> Push 即可，使得 GitLab 服务器上的为最新版本



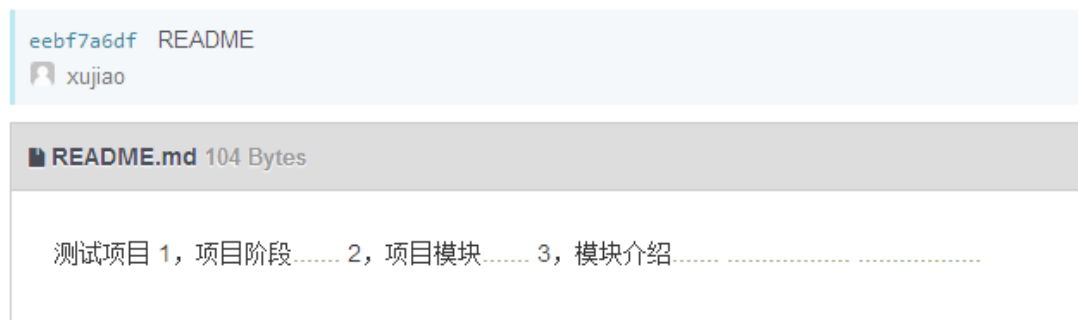
从历史记录中可以看到，从 first commit after clone it 开始历史进入分支，先是 new 的记录，然后是 commit old one 的记录，最后历史分支合并。

Project: githello [githello]						
Id	Message	Author	Authore...	Committ...	Committed Date	
cd4d8ad	merge origin/master to HEAD	徐骄	3 minutes ago	徐骄	3 minutes ago	
9cdac58	commit old one	徐骄	17 minutes ago	徐骄	17 minutes ago	
db95175	new	徐骄	27 minutes ago	徐骄	27 minutes ago	
c0a3c7d	first commit after clone it	徐骄	33 minutes ago	徐骄	33 minutes ago	
34c0daa	modified	徐骄	64 minutes ago	徐骄	64 minutes ago	
8dfd84c	new githello.java	徐骄	77 minutes ago	徐骄	77 minutes ago	

6、两个重要的文件


README.md/.gitignore

1) README.md 用于对项目进行概要说明，供项目成员快速熟悉和了解项目。示例：



2) .gitignore 文件，用户指定在上传项目到 GitLab 时，忽略的文件。



 **.gitignore** 38 Bytes

```
1 .classpath
2 .project
3 .settings/
4 target/
```