Git 操作手册

密级：☐保密 ☒秘密 ☐机密 ☐绝密

文档编号：

拟制者	刘文	2013-4-26
审核	董建宇、谢东	
批准	谢东	

北京用友集团 UAP 中心

2013 年 4 月

目 录

1. 相关安装文件及资料获取

编制本文件为了统一 UAP 研发中心的代码管理规则及工具；统一管访问 http://20.10.129.77:8080/gitlab/test-team/download/tree/master/GitLabSoftware （最好使用谷歌浏览器访问）。

使用下面的用户/密码登录：

```
User: guest@yonyou.com Password: 123456
```

图 1-1

登录后，即可下载对应的安装文件：

msysGit： Git-1.8.1.2-preview20130201.exe

TortoiseGit：TortoiseGit-1.8.2.0-64bit.msi

TortoiseGit-1.8.2.0-32bit.msi

TortoiseGit 中文语言包：

TortoiseGit-LanguagePack-1.8.2.0-64bit-zh_CN.msi

TortoiseGit-LanguagePack-1.8.2.0-32bit-zh_CN.msi

eclipse 插件：org.eclipse.egit.repository-2.3.1.201302201838-r.zip

（适用于 3.7.* 以上版本的 eclipse，最新版 eclipse（Juno）中已包含此插件，无需安装）。

参考文档：《Git_Community_Book_中文版》

图 1-2

2. 安装 MSysGit

运行 Git-1.8.1.2-preview20130201.exe，选择 Next，默认安装即可。

图 2-1

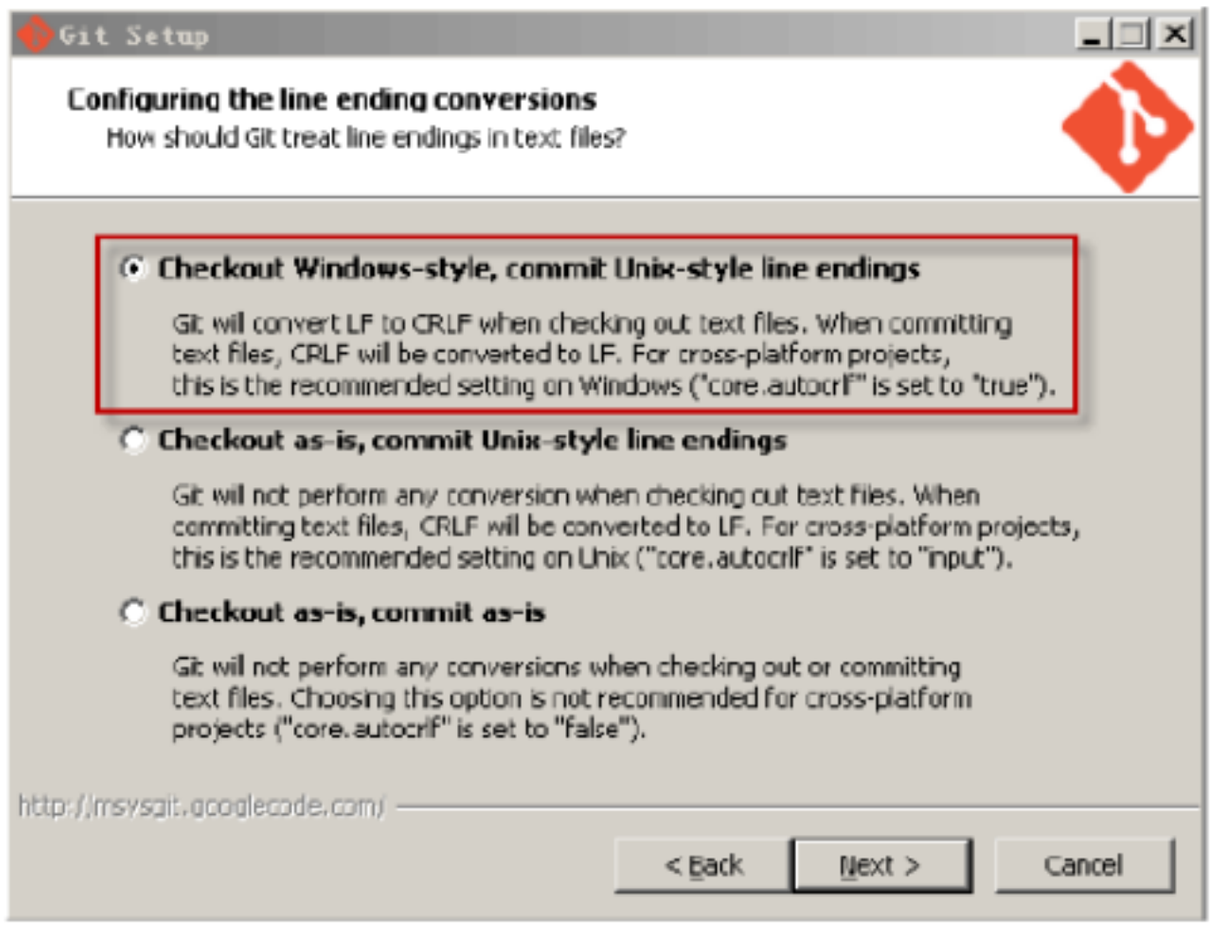

图 2-2

安装完成后，打开 Git Bash（安装 MSysGit 后，会生成桌面快捷方式）。配置全局用户名和邮箱，执行下面两行命令。

git config --global user.name yourname

git config --global user.email youremail

（注意：参数名与参数值之间不要使用“=”，空格即可。）

可通过下面两行命令查看配置

git config user.name

git config user.email

3. 安装 TortoiseGit

（从 TortoiseGit 1.7.15.2 版本起，需要系统 Windows Installer 版本为 4.5。）

已提供 Windows Server 2003 的补丁：

32 位：WindowsServer2003-KB942288-v4-x86.rar

64 位：WindowsServer2003-KB942288-v4-x64.rar

其他版本系统可从下面网址下载安装 Windows Installer 4.5。

http://www.microsoft.com/en-us/download/details.aspx?id=8483

运行 TortoiseGit-1.8.2.0-32bit.msi 或 TortoiseGit-1.8.2.0-64bit.msi

注意：选择 SSH 客户端时，要选第二项，OpenSSH，Git default SSH Client，其他默认安装。

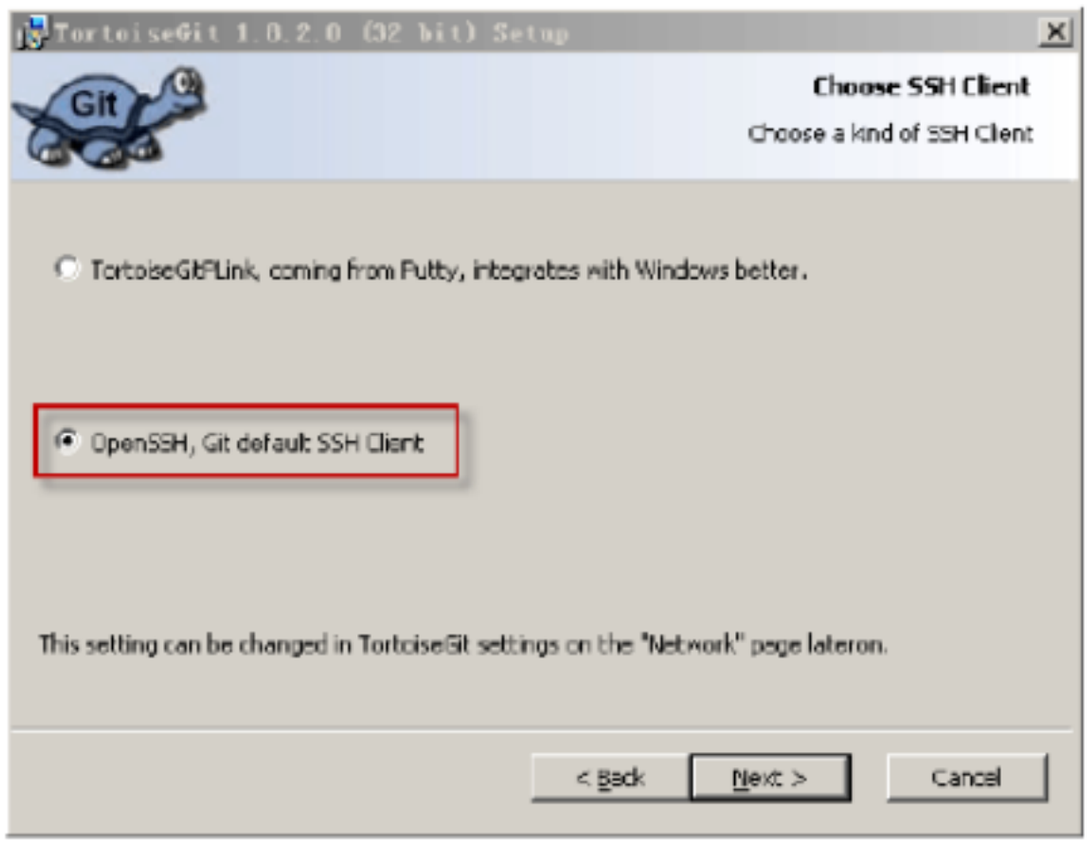

图 3-1

安装完成后，右键-TortoiseGit-Settings，查看设置。

TortoiseGit 会自动检测到 MSysGit 的安装路径。

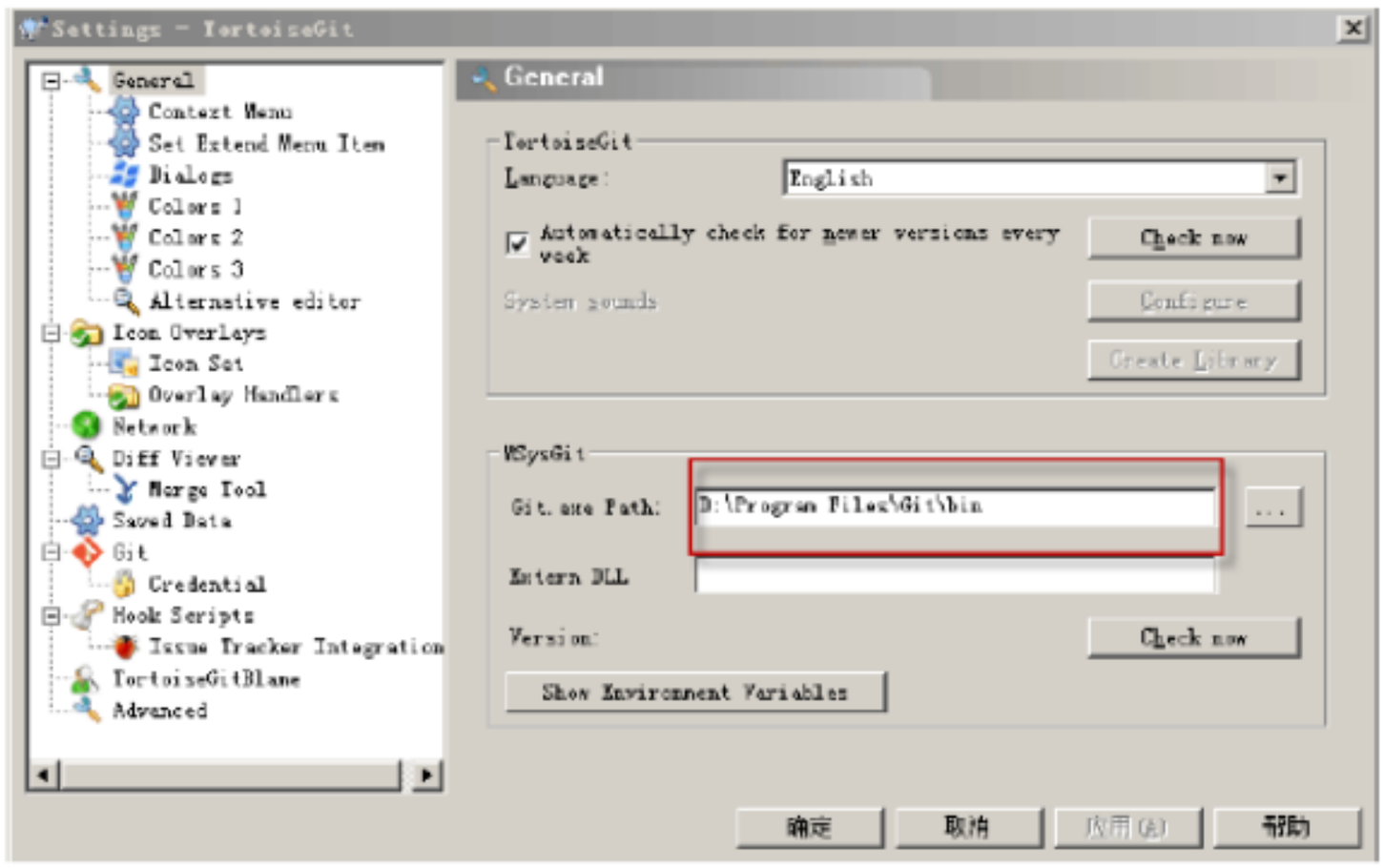

图 3-2

另外可选择对应的语言包，安装简体中文。

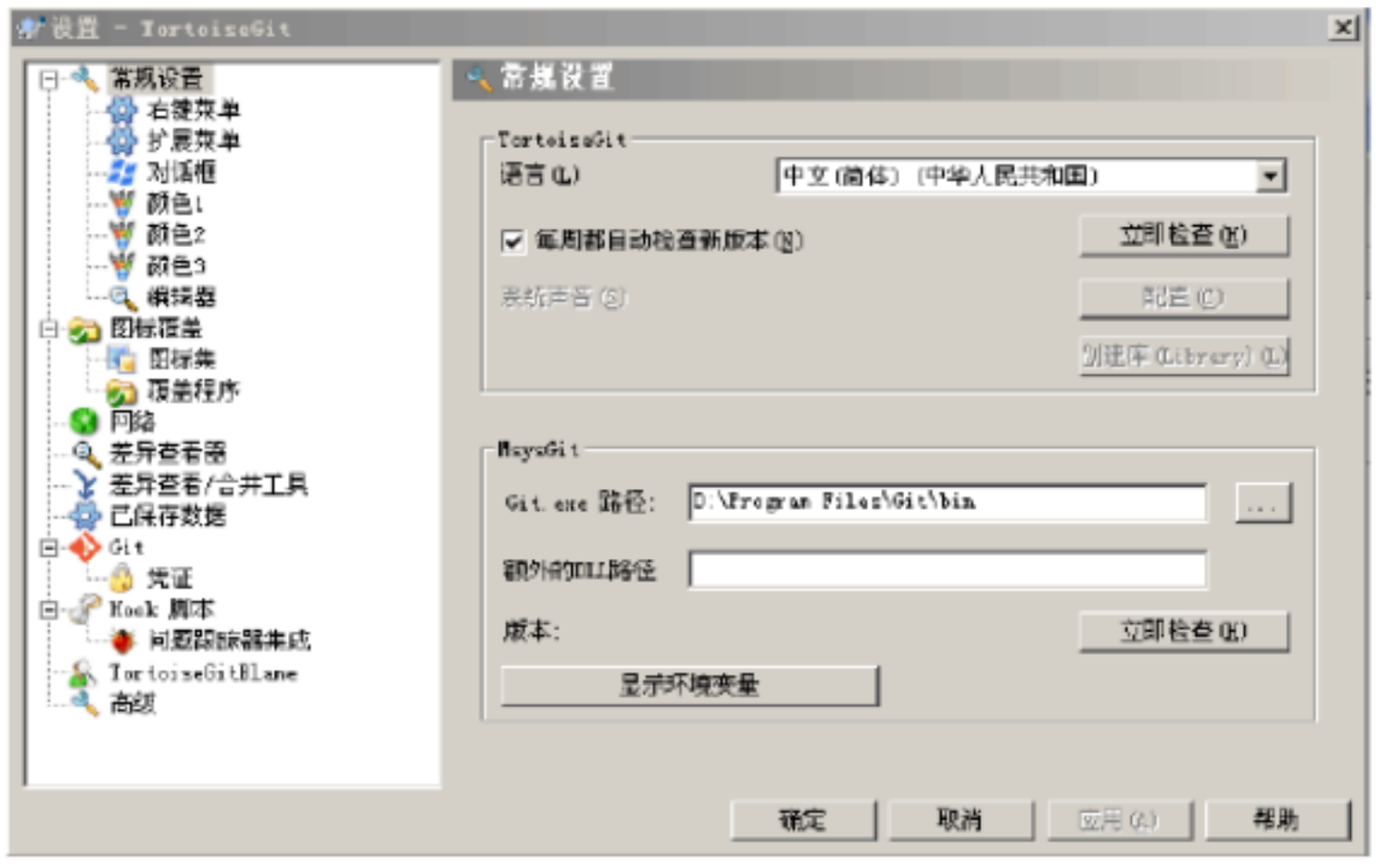

图 3-3

选择 SSH 客户端错误解决。

4. SSH 公钥认证

用户通过 SSH 公钥认证，建立与服务器端的通信。

4.1 创建自己的公钥/私钥对。

打开 Git Bash（安装 MSysGit 后，会生成桌面快捷方式）。

输入： ssh-keygen -t rsa 其它输入回车略过。

会在用户主目录（C:\Documents and Settings\user_name）下的.ssh 目录下生成私钥 id_rsa 及公钥 id_rsa.pub 两个文件。

（注意：不要修改文件名和存放路径）

```
liuwend@LIUWEND ~
$ ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/c/Documents and Settings/liuwend/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /c/Documents and Settings/liuwend/.ssh/id_rsa.
Your public key has been saved in /c/Documents and Settings/liuwend/.ssh/id_rsa.pub.
The key fingerprint is:
63:a3:37:22:1c:d9:65:37:d7:da:54:a1:0c:19:a1:57 liuwend@LIUWEND
liuwend@LIUWEND ~
```

图 4-1

4.2 公钥 id_rsa.pub 配置到服务器。

通过谷歌浏览器访问服务器网址（以http://20.10.129.77:8080/gitlab为例，具体要使用对应服务器的网址）。

输入自己的用户名/密码登录（邮箱登录，默认密码 111111）。

进入右上角的 My Profile 界面。

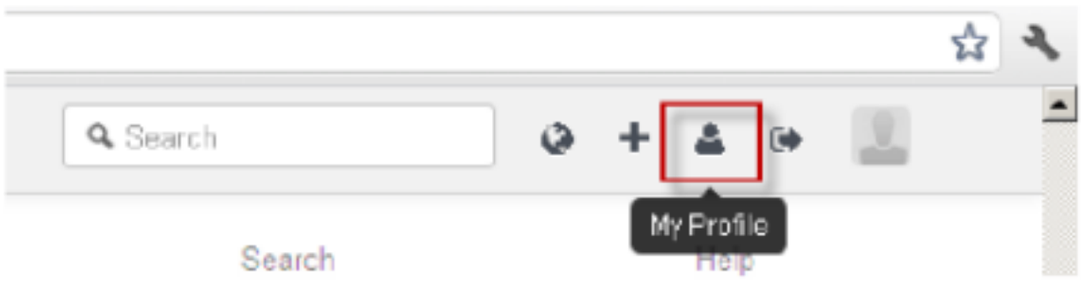

图 4-2

进入 SSH Keys 界面，点击 Add new，并点击 save 保存。

图 4-3

公钥会出现在 SSH Keys 的列表中。

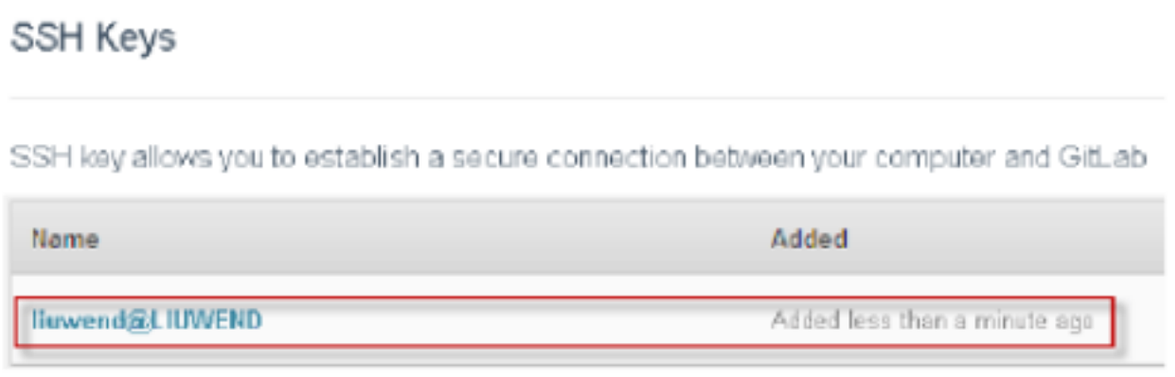

图 4-4

到此，通过 SSH 公钥认证，建立与服务器端的通信完成。

5. 开发过程中使用的 Git 分支模型

Git 鼓励通过分支进行管理，以方便日常开发。最主要的两个分支为 master 与 develop 分支，分别代表 发布/开发中 两个状态。develop 是每日

构建的来源，当到了稳定的状态就可以合并到 master 上，也就是发布了新版本 。

此外，辅助性分支有特性分支（feature branch）、待发布分支（release branch）及补丁分支（hotfix branch）。特性分支由各开发创建，主要用来实现新需求，从 develop 分支上分出，待开发完成后合并到 develop 分支上。待发布分支在开发到一定的阶段进入固化状态时创建，从 develop 分支分出，只做小的 bug 修复，达到理想状态后，把 release 分支合并到 master 分支，也就是发版了， 且可以随时合并到 develop 分支上。补丁分支一般从 master 分支分出，主要用来修改已发布版本的 bug。相当于累积的通版补丁，且必须合并回 develop 分支上。

各分支详细介绍以及在开发过程中使用到的分支模型可查看《配置管理规范—基于 Git 代码管理.doc》，了解具体的信息。

6. 开发过程中常用操作简介（使用 TortoiseGit）

6.1 初化 git 项目

开发经理明确产品命名，产品内项目的分布，代码库的访问人员及各项目的人员访问权限后，提供相关信息及各用户公钥给配置管理员，提出建库申请。

配置管理员根据开发经理的申请，在代码服务器上创建 git 项目，配置访问权限，并将对应的 SSH 链接反馈给开发经理，如：
git@20.10.129.77:liuwend/test_demo.git（之后均以此为例）。

git 项目的初始化，可以从某个前期发布的金盘版本建立或从零开始建立初始代码库。

6.1.1 基于空库初始化 git 项目

适用于对全新产品的开发，由零开始建立初始代码库。

首先通过 Clone 操作将新创建的 git 项目，由服务器克隆到本地。

Clone 操作（右键-Git Clone）：

填入相关信息（注意本地工作目录必须为空目录），如下图所示：

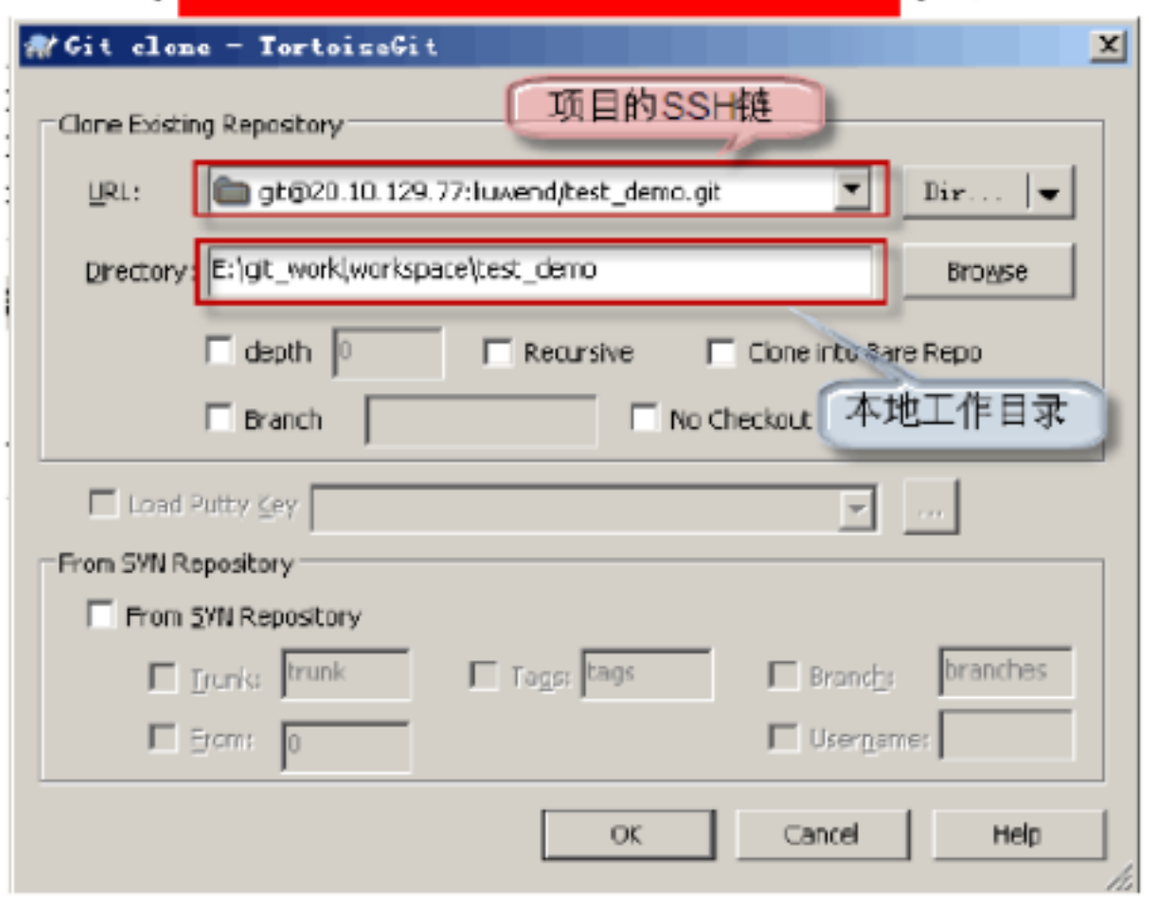

图 6-1-1

选择 OK，完成 clone 操作。

将初始项目代码导入到本机工作目录内，通过 Commit 操作将代码提交到本地的 git 仓库，然后通过 Push 操作将代码推送到服务器，作为初始的代码基线。（注意只有提交到本地 git 仓库的代码才会被推送到服务器）

Commit 操作（右键-Git Commit→”***”…）：(***为当前分支名)

在文件选择区选择要提交的文件，并填写提交描述信息（信息不能为空）。

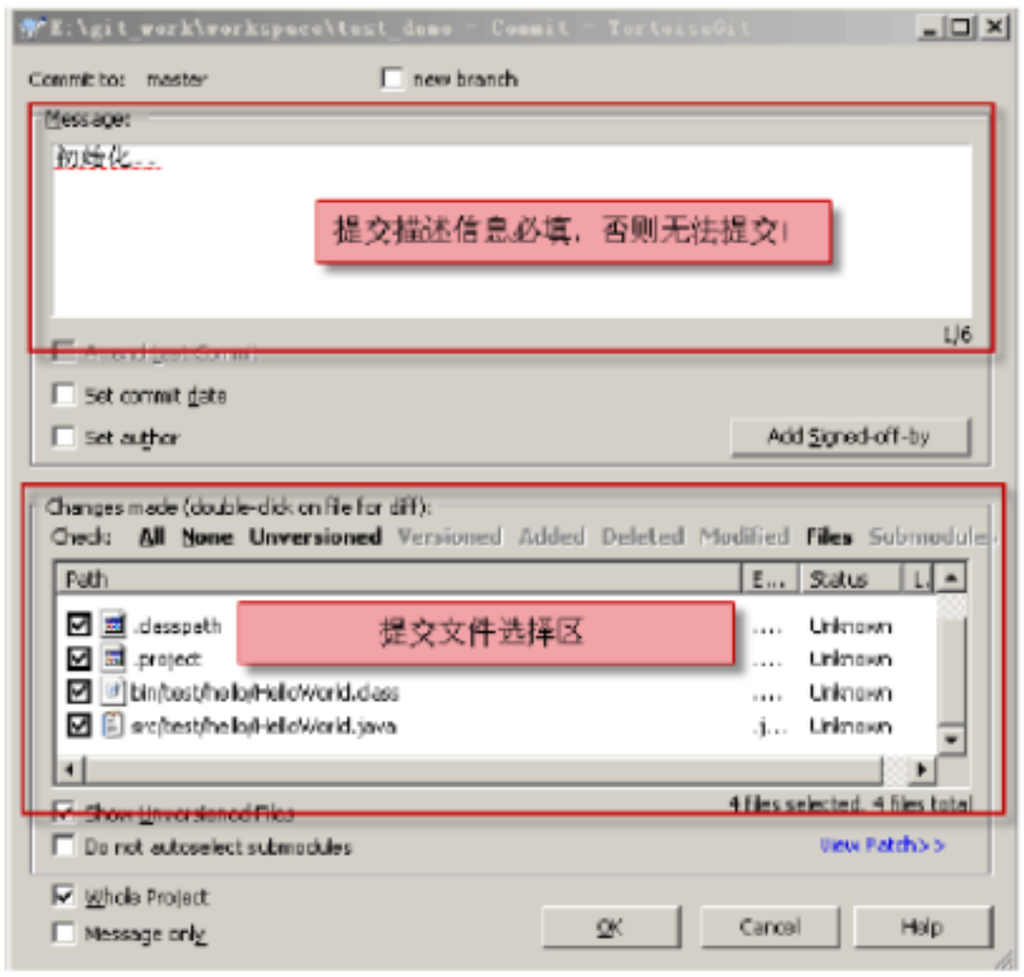

图 6-1-2

选择 OK，完成 Commit 操作。

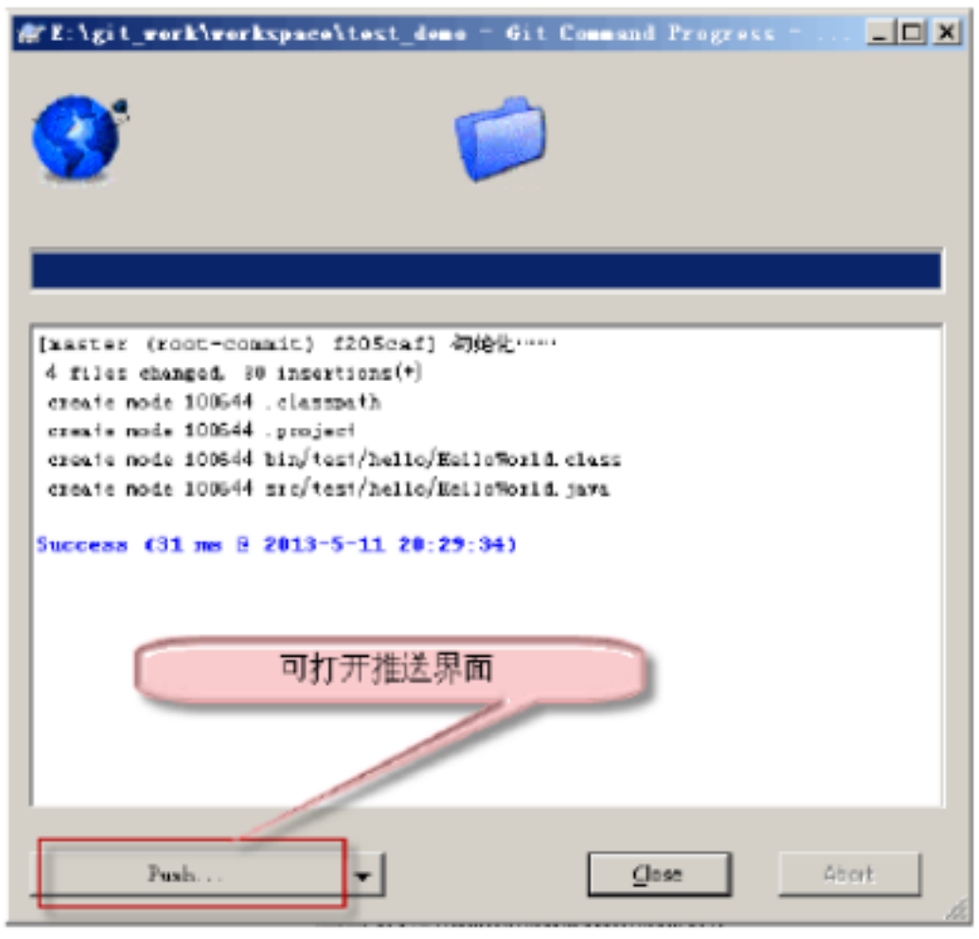

图 6-1-3

Push 操作（右键-TortoiseGit-push）：

可在 Commit 操作完成时，通过 Push 按键打开推送界面，或右键-TortoiseGit-push 亦可。

选择要 push 的本地分支和对应的远程分支（初始只有 master 分支）

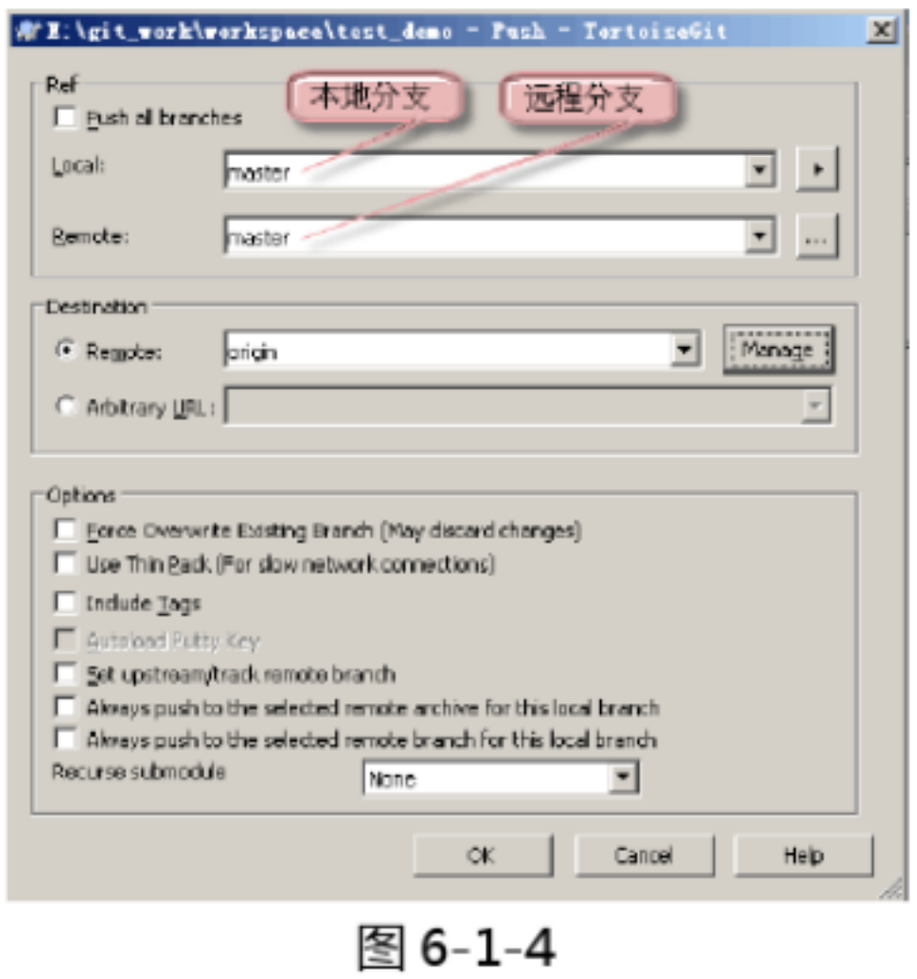

图 6-1-4

选择 OK，完成 Push 操作。

所有代码提交推送完成后，git 项目基于空库的初始化完成。

6.1.2 基于已有版本库初始化 git 项目

产品基于某个前期发布的金盘版本进行开发，可选择这种方式初始化 git 项目。

首先进入本地已发布金盘版本的 git 仓库，通过 Checkout 操作切换到要作为新 git 项目初始代码的分支。切换到目标分支后，通过 Pull 操作拉取远程最新代码到本地，更新本地代码。并通过 Push 操作，将分支代码推送到新 git 项目。（推送时要修改 Remote 信息）

Checkout 操作（右键-TortoiseGit-Switch/Cheakout）

右键-TortoiseGit-Switch/Cheakout，打开 Checkout 界面，填入要切换到的分支，OK 即可。

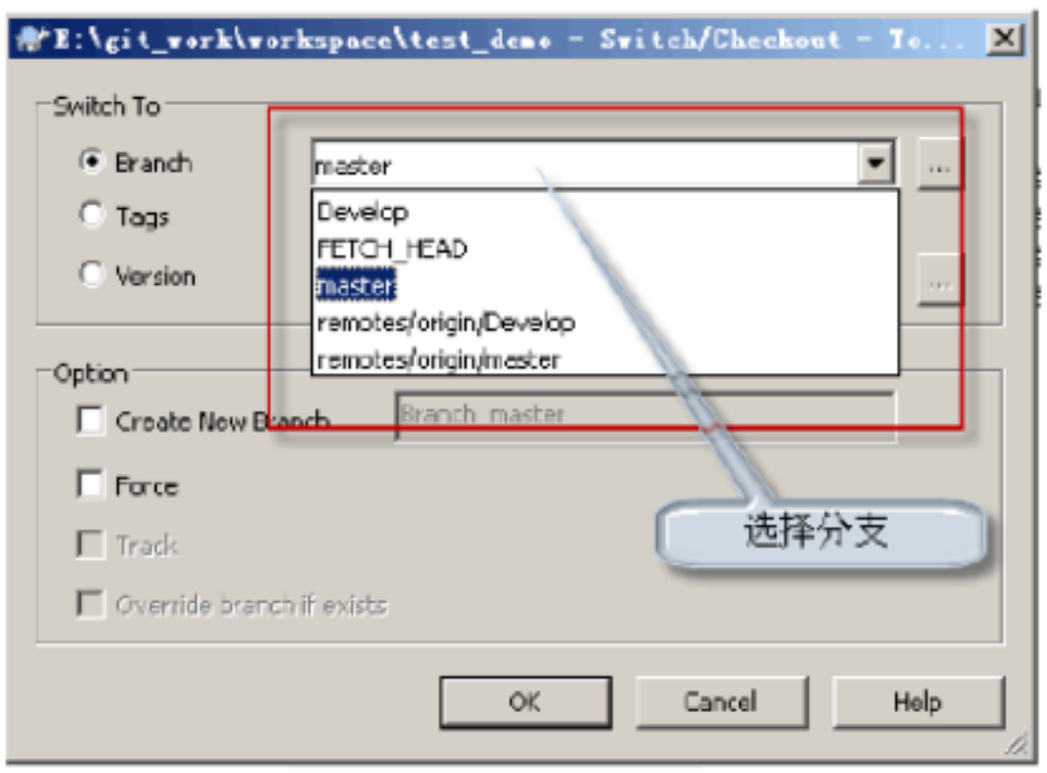

图 6-1-5

Pull 操作（右键-TortoiseGit-pull）

右键-TortoiseGit-pull，打开拉取界面，远程分支默认为当前分支（切勿使用其他分支，造成混乱），OK 即可。

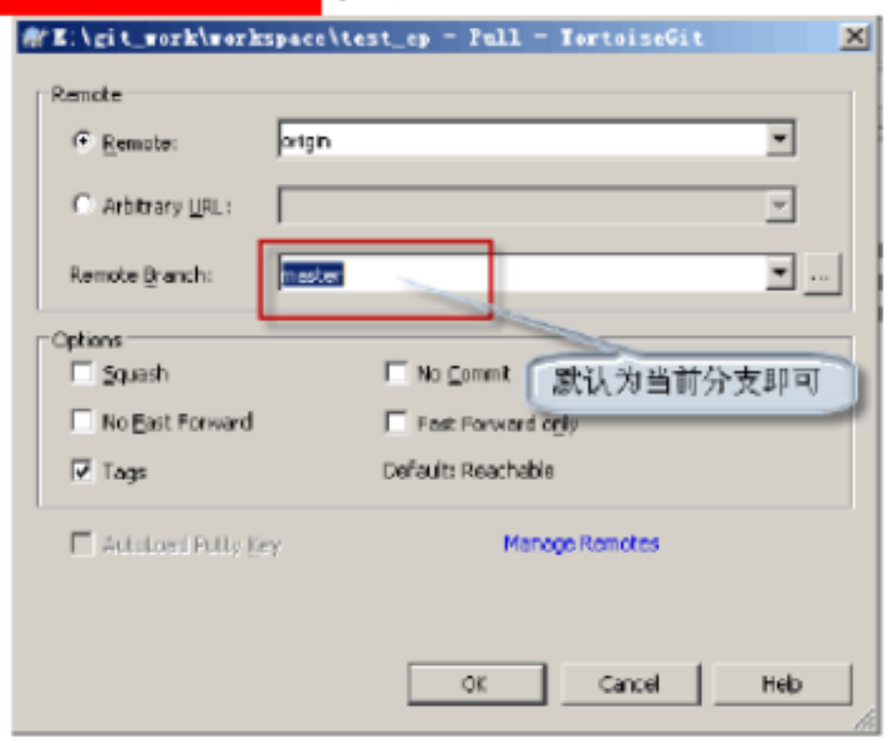

图 6-1-6

代码更新后，推送代码到新 git 项目，首先要修改 Remote 信息。

右键-TortoiseGit-push 打开推送界面，点开 Manage 按键，打开 Remote 管理界面。

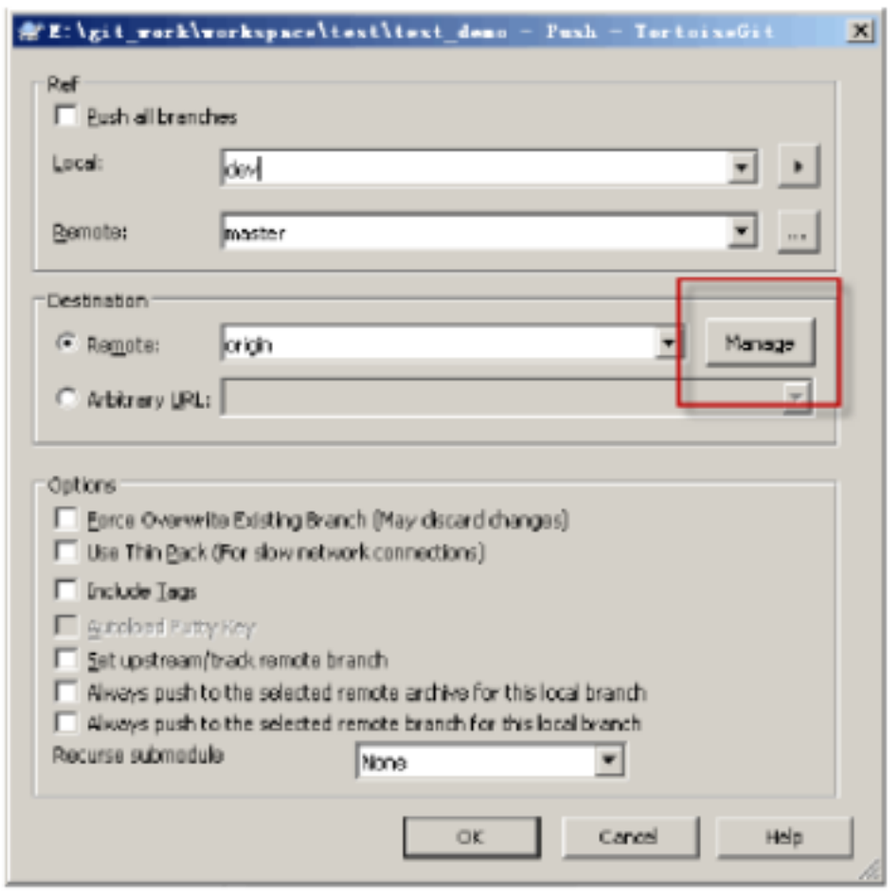

图 6-1-7

修改 origin 的 URL 信息为新建 git 项目的 SSH 链接，或者添加新的 Remote 信息，下面以添加新的 Remote 信息为例。

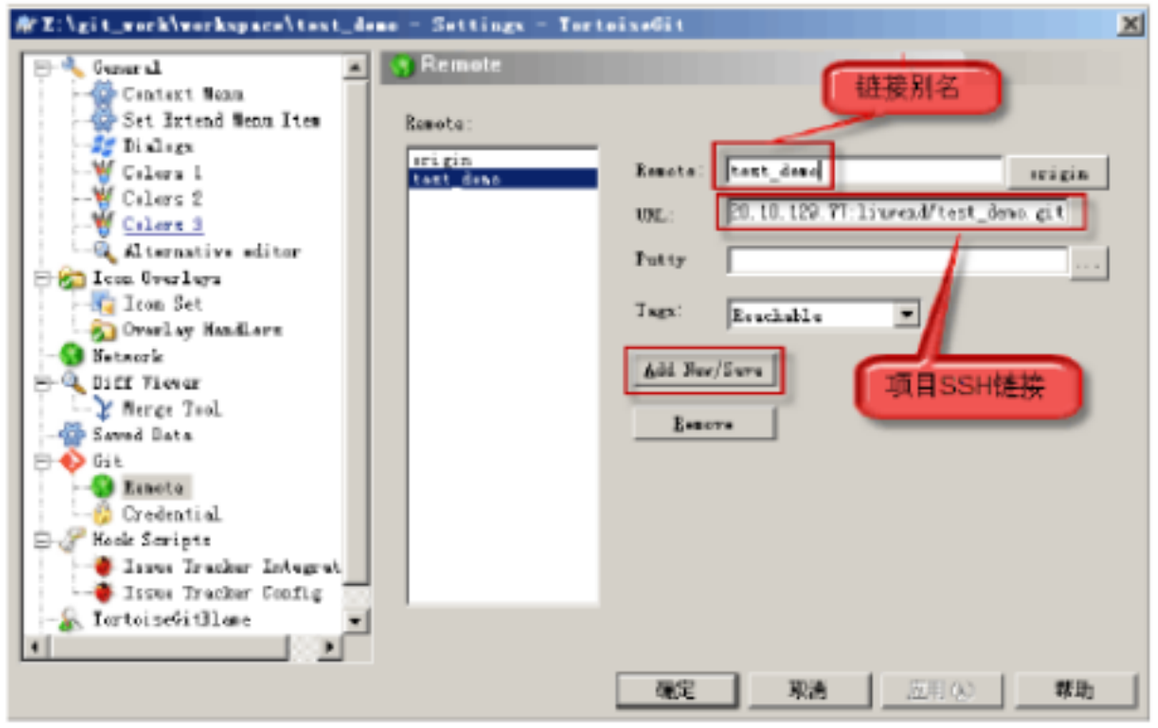

图 6-1-8

选择 Manager，输入新建 git 项目的 SSH 链接及链接别名（链接别名自己随意设置，最好有意义），点击 Add New/Save 按键后，确定。

返回推送界面，填好作为初始代码的本地分支及远程分支（初始只有 master 分支），Remote 选择新建的链接别名，OK，完成推送操作。

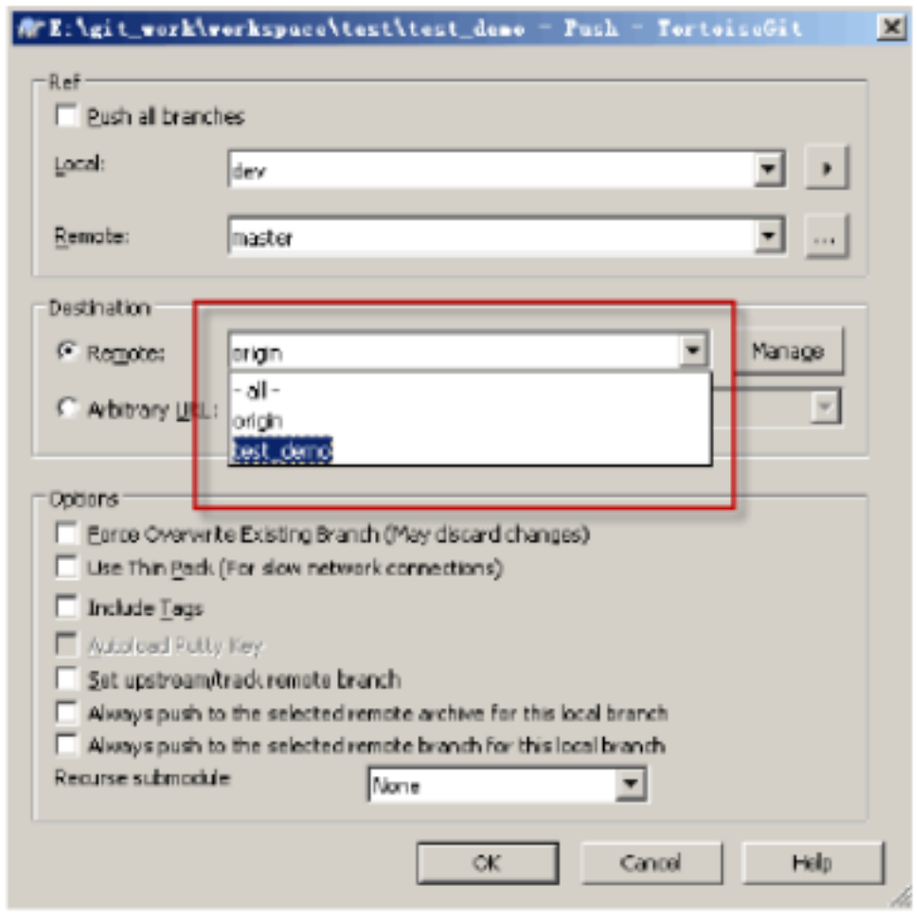

图 6-1-9

推送完成后，git 项目基于已有版本库的初始化完成。

6.1.3 创建开发分支 develop branch

在 git 项目代码初始化完成后，开发经理需要基于初始化的 master 分支创建开发分支，单一命名为 Develop。通过 Create Branch 操作本地创建 Develop 分支，然后通过 Push 操作将 Develop 分支推送到服务器。

开发分支是开发中，某一产品集成分支，为开发人员日常开发的分支，是每日构建的代码来源，产品提交产品线集成后， 开发分支同时可以继续做其他特性的开发。

Create Branch 操作（右键-TortoiseGit-Create Branch）

首先，通过 Clone 操作把 git 项目克隆到本地（已克隆则通过 Pull 操作更新本地代码），进入项目工作目录后，右键-TortoiseGit-Create Branch，打开分支创建界面。填入分支名 Develop，基于 master 分支（此处同 HEAD），OK，完成。

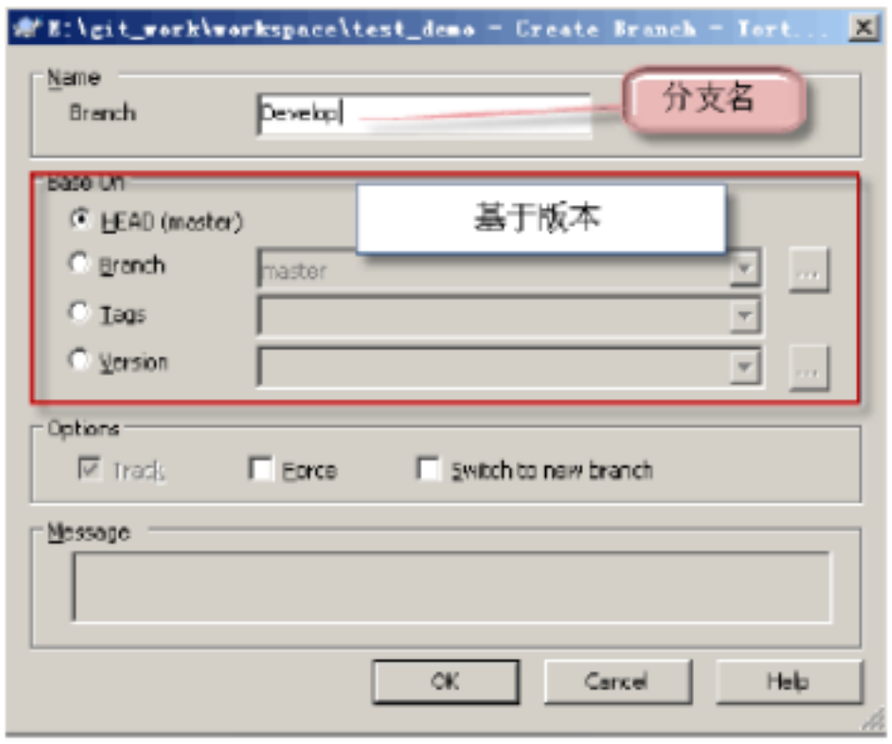

图 6-1-10

Develop 分支创建完成后，通过 Push 操作将 Develop 分支推送到服务器（本地分支与远程分支均填写 Develop 即可）。

6.2 基于 Develop 分支进行开发

开发人员基于 Develop 分支建立本地库，根据开发任务建立特性分支或缺陷分支，作为独立开发的环境，以特性或缺陷为单位进行独立完整的开发及提交。

首先，从服务器上获取 git 项目的 Develop 分支，有下面两种方法：

第一种方法，从服务器将 git 项目的 Develop 分支克隆到本地，Clone 操作同前文所讲，对特定分支克隆，注意勾选 Branch，填入分支名（如 Develop），如下图所示：

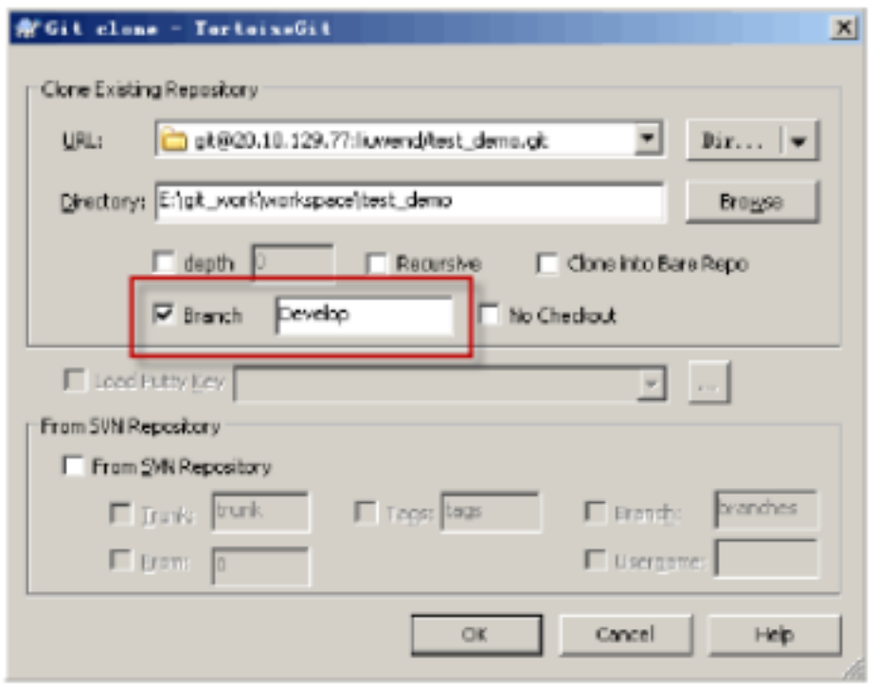

图 6-2-1

第二种方法，另一种是在本地目录下，右键-Git Init Here，创建新库。然后通过 Pull 操作将服务器 git 项目的 Develop 分支拉取到本地 Git 仓库。

Pull 操作同前文所讲，由于本地新库没有 Remote 信息，先在拉取界面通过 Manage Remote 添加 Remote 信息，如下图：

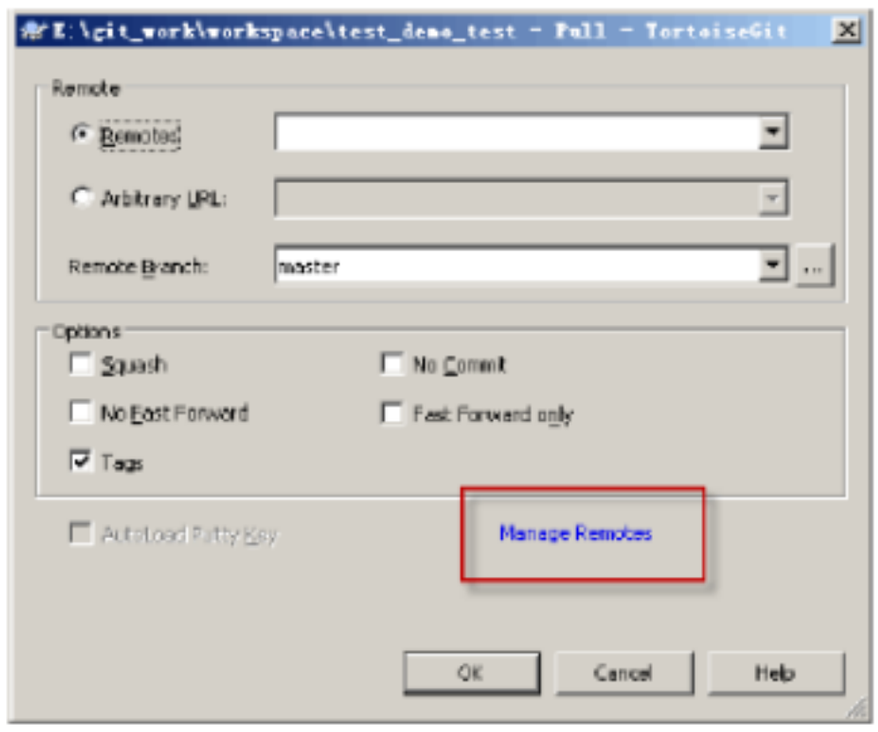

图 6-2-2

添加 Remote 信息过程同前文所讲，添加完成后，在拉取界面 Remote 选择新添加的 Remote，远程分支填入 Develop，OK，完成拉取。

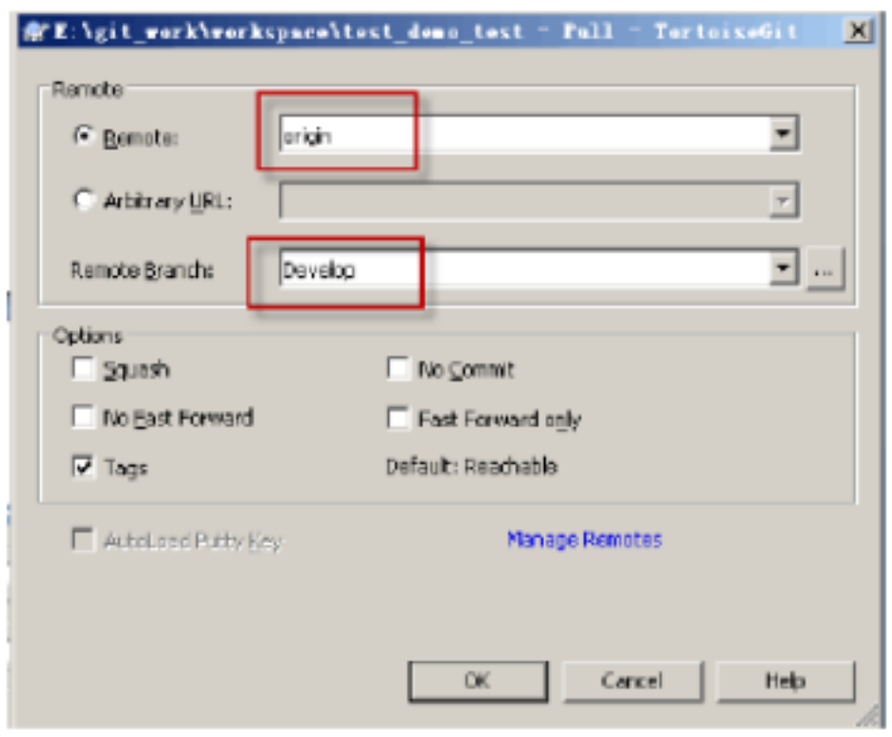

图 6-2-3

（注意，第一种方法本地主分支名为 Develop，第二种方法本地主分支名为 master）

这样，本地 Develop 分支创建完成。

开发人员在本地开发过程中，以特性（或缺陷）为单位展开代码开发工作，即根据特性（缺陷）为独立单位建立特性代码分支，在特性分支上完成独立代码开发的功能；特性分支从开发分支初始建立（缺陷分支从发布分支初始建立）。在特性分支上进行的开发工作后面会介绍。下面先介绍在本地 Develop 分支上需要进行的操作。

同步操作（右键-Git Sync...）：

Develop 分支在本地建立完成后，可通过同步操作进行本地与服务器之间的代码同步，在工作目录下，右键-Git Sync... ，打开同步界面（集成了 Pull, Push, Commit 及 Show log 等功能）。使用时，注意填写好本地分支与远程分支，并选择正确的 SSH 链接别名（可新建），界面如下图所示：

图 6-2-4

Pull（拉取）：将远程分支代码同步到本地分支。开发人员在提交本地代码前，首先需要将服务器集成分支的最新代码拉取到本地，进行本地代码合并及冲突的解决(冲突解决在下面介绍)。冲突解决后代码方可推送至服务器分支，即进行下面的 Push 操作。如在 Pull 后长时间没有 Push，导致服务器代码有新的提交，可能仍旧推送失败，需要重新 Pull 解决冲突。

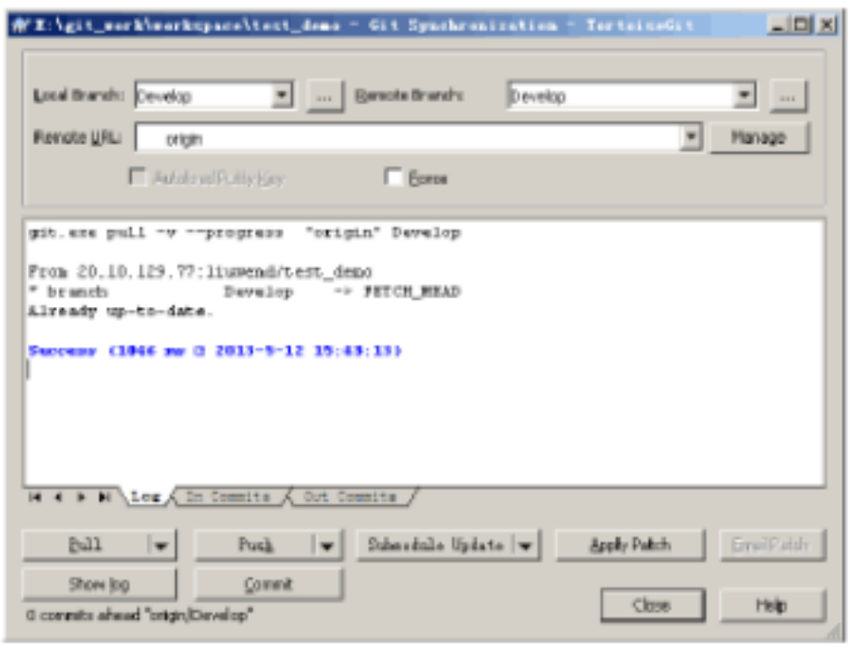

图 6-2-5

Push（推送）：将本地分支同步到远程分支。开发人员在本地验证测试通过后，将代码推送至主服务器的集成分支，要求提交代码为具有质量保证自测无问题代码，开发人员原则上要求每天做到本地代码库提交，每三天至少进行一次集成分支的合并提交，提交尽

可能以完整功能为单位，如规定时间内的成果无法做到完整提交，则需要尽快提交，不能大于一周，从而避免产生大量的代码冲突。

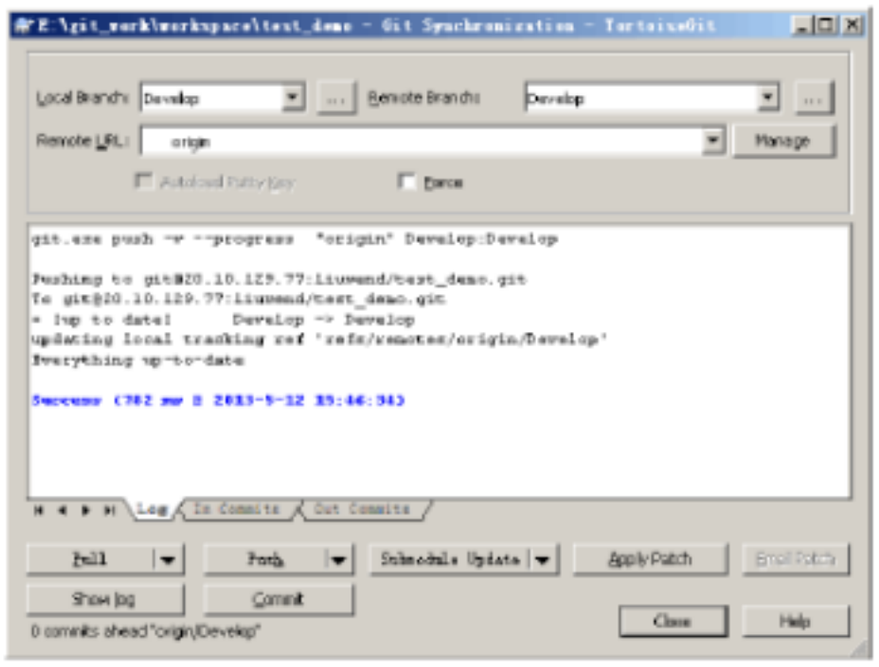

图 6-2-6

Show log：显示以往的提交日志。

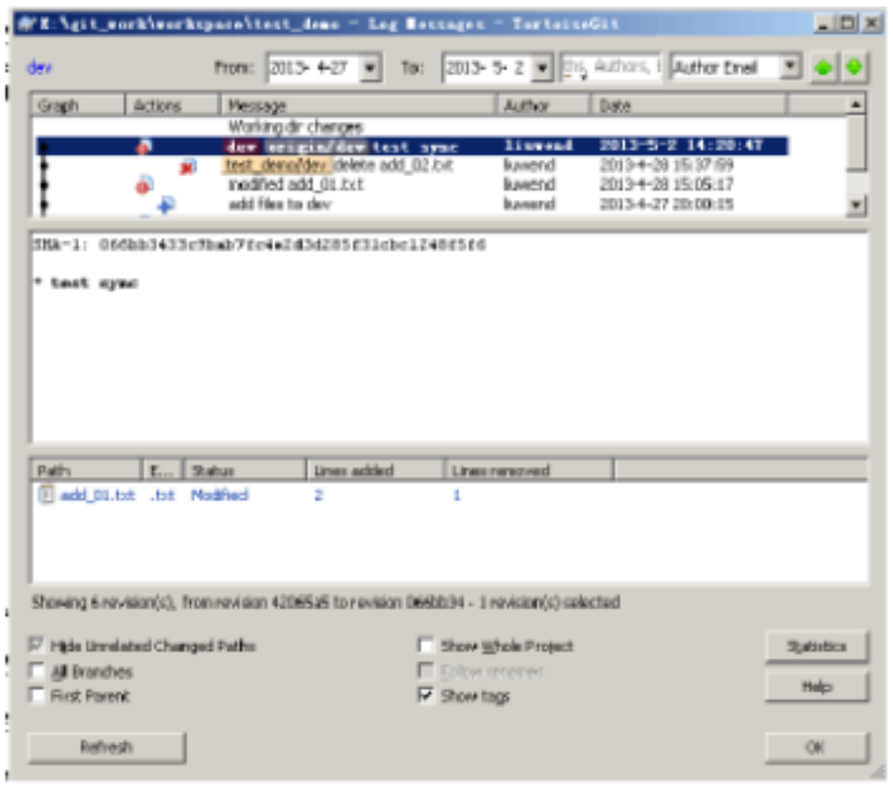

图 6-2-7

Commit：打开本地分支提交界面。

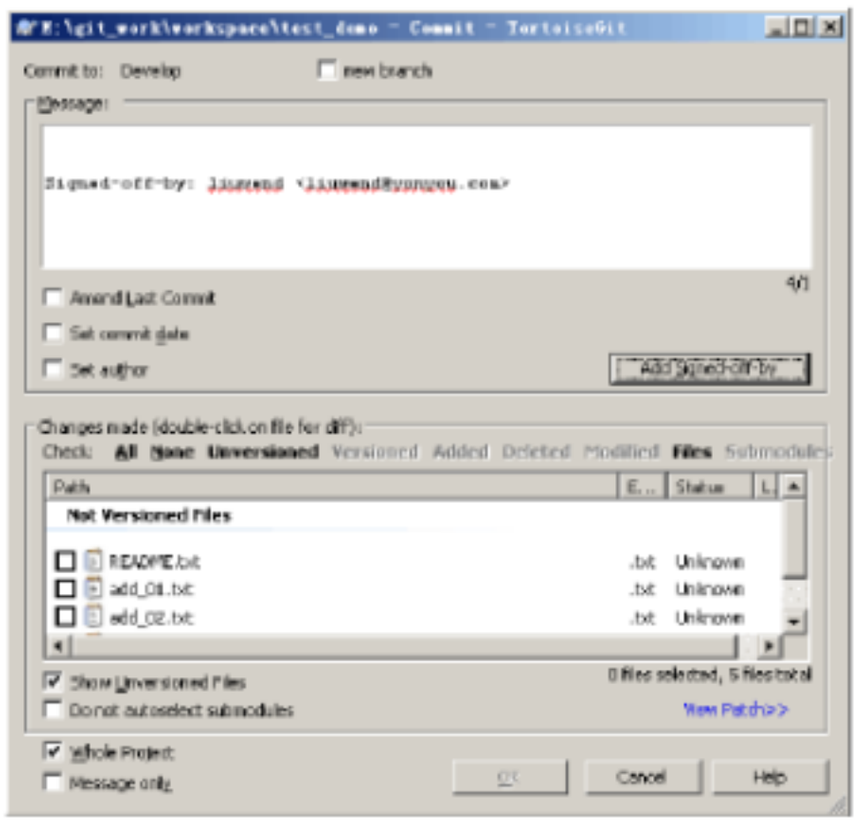

图 6-2-8

冲突解决：

本地分支文件的修改，可能会与远程分支文件的修改存在冲突，此时无法将本地提交推送到远程分支。执行 pull，会发现存在冲突文件。

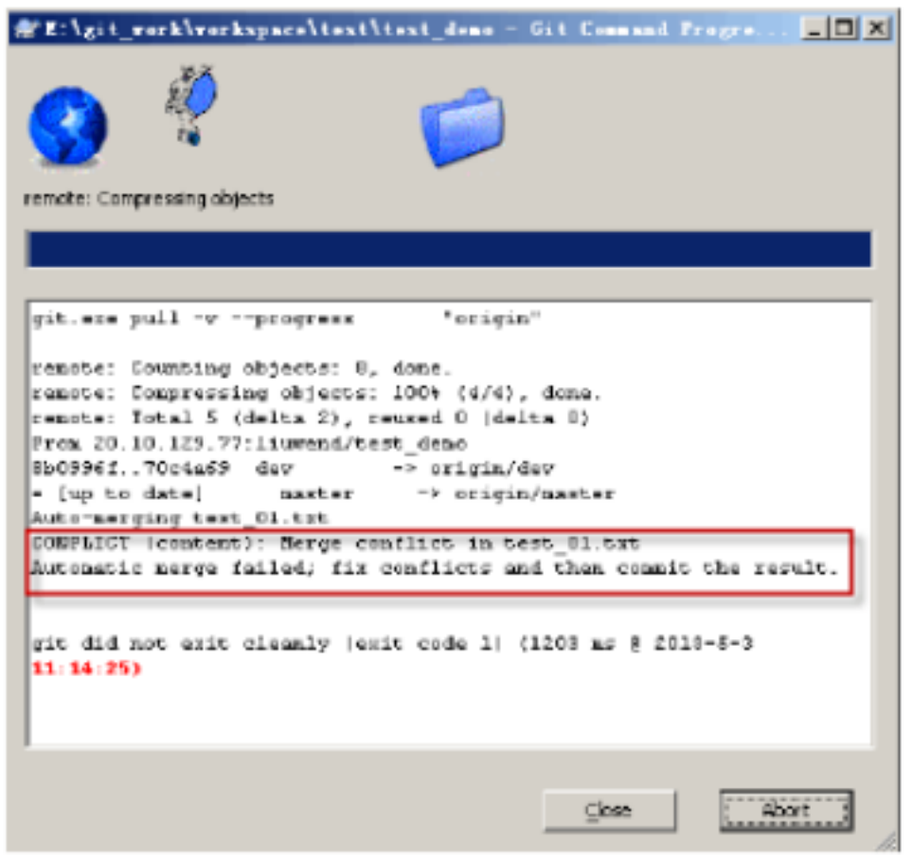

图 6-2-9

在同步界面执行 pull，可更清晰地显示冲突。

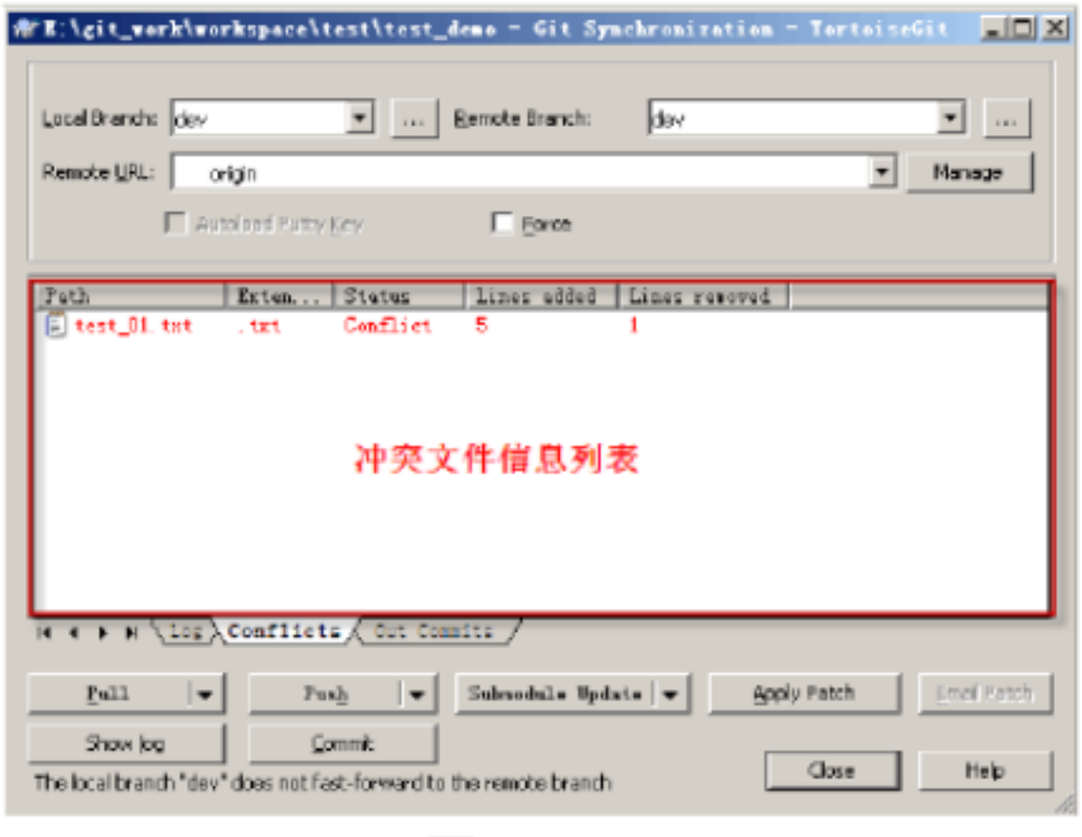

图 6-2-10

双击文件，可查看冲突文件内容。

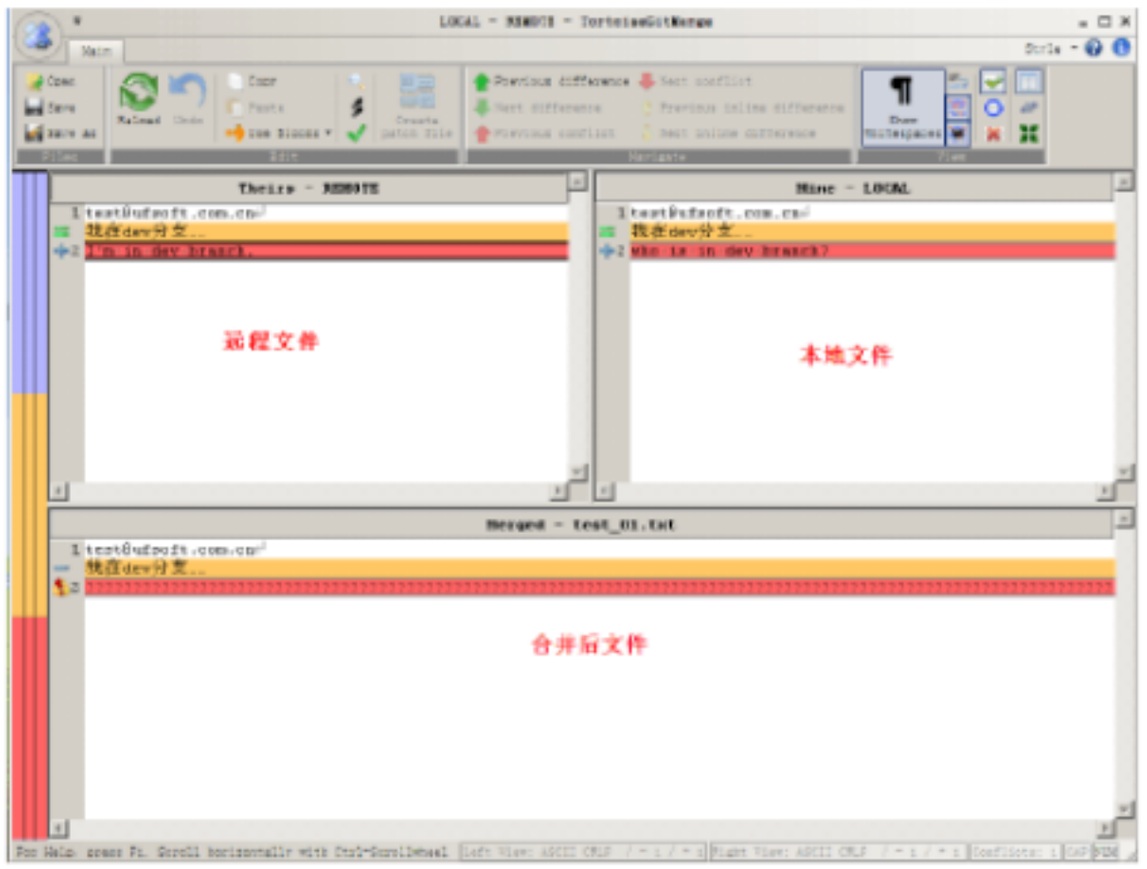

图 6-2-11

在文本块右键，选择解决冲突的方式，然后保存文件。

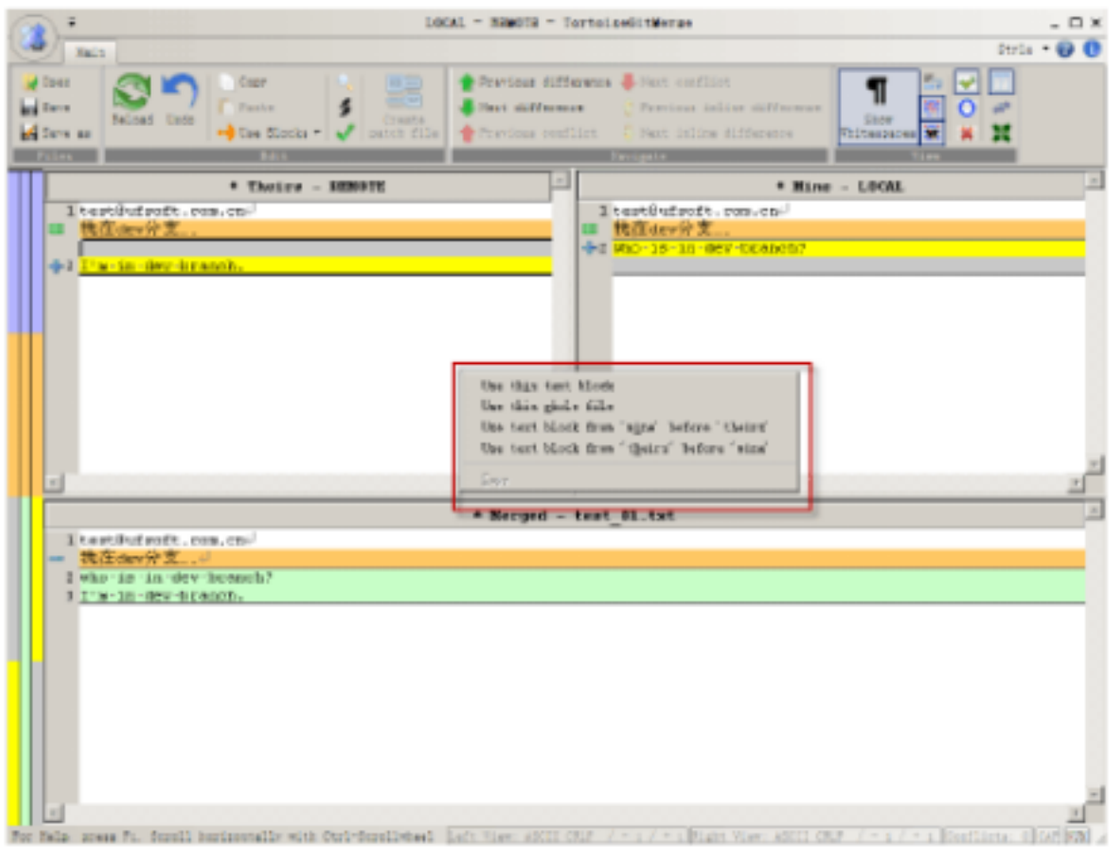

图 6-2-12

图 6-2-13

解决完冲突后，提交解决结果（同步界面 Commit 或右键-Git Commit→”dev”）。

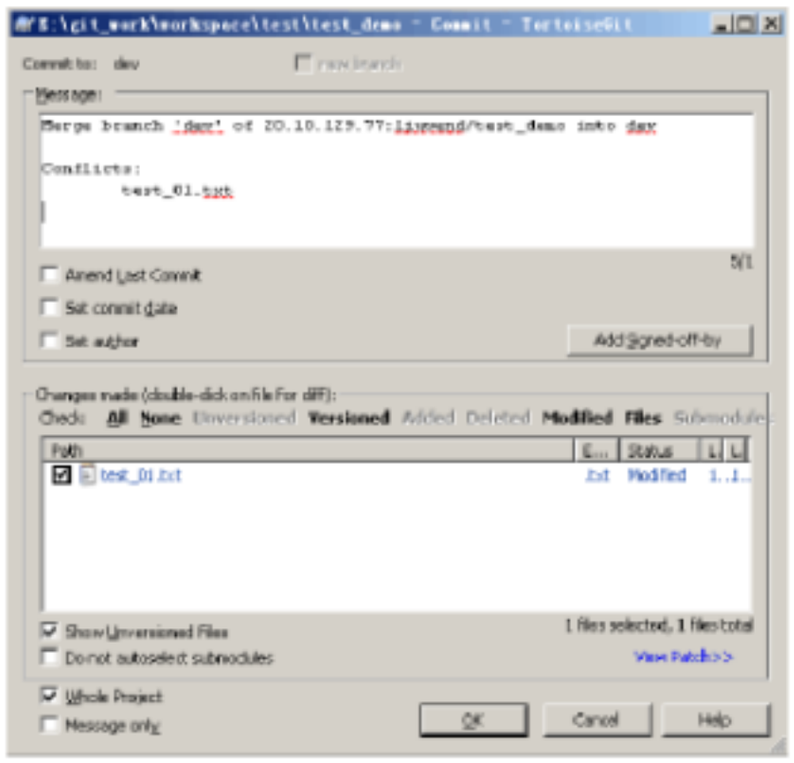

图 6-2-14

之后，将修改 push 到远程分支，完成冲突解决及代码的推送（无需推送时可忽略此步）。

发现冲突后，还可以通过右键- TortoiseGit-Resolve...，进行冲突解决。

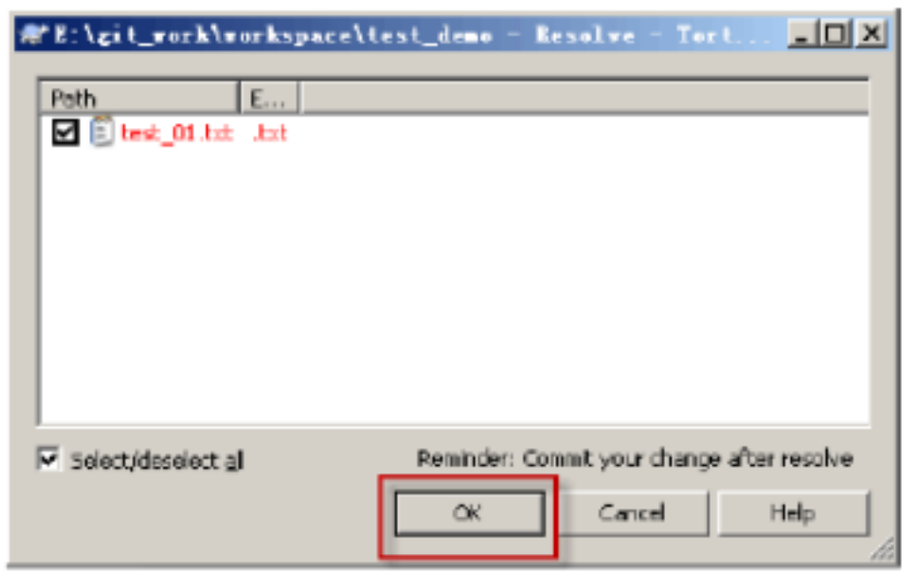

图 6-2-15

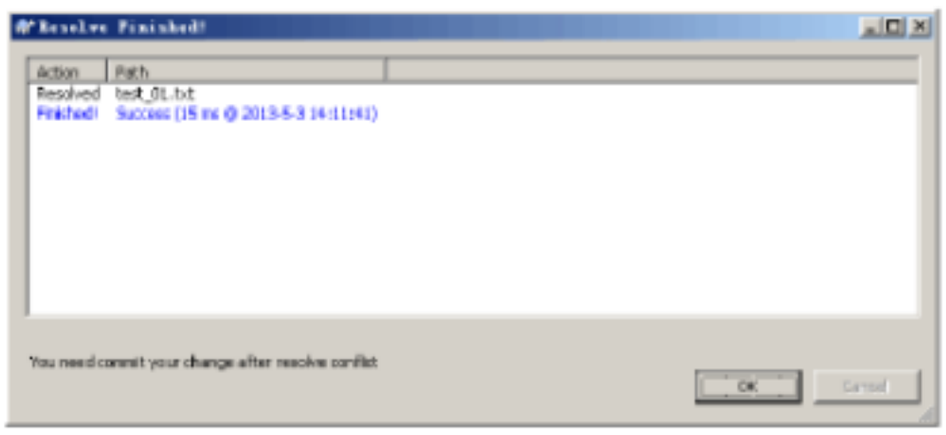

图 6-2-16

之后，进行 Commit 及 push 操作（同前一种方法）。

这种默认的解决冲突，最后的合并文件中包含本地及远程修改的信息。

图 6-2-17

在本地 Develop 分支上进行的操作还有创建特性分支及合并特性分支，将会在下面介绍。

6.3 基于特性分支（feature branch）进行开发

开发人员在本地开发过程中，根据特性（缺陷）为独立单位建立特性代码分支，在特性分支上完成独立代码开发的功能。

特性分支在开发中，用于新需求的实现；由开发人员基于 Develop 分支创建，作为开发人员日常开发的基础分支。一般只存在于各开发的本地库，不 push 到主数据库中，命名规则为：账号/分支特性名（或特性编号），如：iujian/add_new_dialog（WFP-599）。

开发人员在本地特性分支进行代码修改，根据开发内容的不同，在不同的特性分支之间进行切换，从而保证更改是独立的，且在一个干净的环境中开始和结束。

特性分支的创建与前文介绍的 Develop 分支的创建相同，通过 Checkout 操作切换到 Develop 分支，Pull 操作更新本地 Develop 分支，Create Branch 操作完成创建。注意命名规则及选择基于 Develop 分支。（特性分支本地保留，无需 push 到服务器）

通过 Checkout 操作切换到要修改的特性分支，进行新需求的开发完善或是缺陷修复工作。对文件的添加、修改及删除操作，在本地项目目录中正常进行即可。由开发人员自行控制，在适当的时候将修改通过 Commit 操作提交到本地的 git 仓库中。

独立的特性(或缺陷)完成并独立验证后，通过 Merge 操作合并至本地的 Develop 分支上。分支合并时，要求开启--no-ff 选项，以保留特性分支的痕迹，方便日后区分很哪些提交是在一起是为了实现一个特性。(TortoiseGit 默认已开启--no-ff 选项)

Merge 操作（右键-TortoiseGit-Merge...）：

首先，在工作目录下，通过 Checkout 操作切换到要被合并的特性分支，通过 Pull 操作拉取本地 Develop 分支（需新加对应本地

目录的 Remote 信息，如下图所示），若存在冲突文件，通过前文介绍的方法解决冲突。

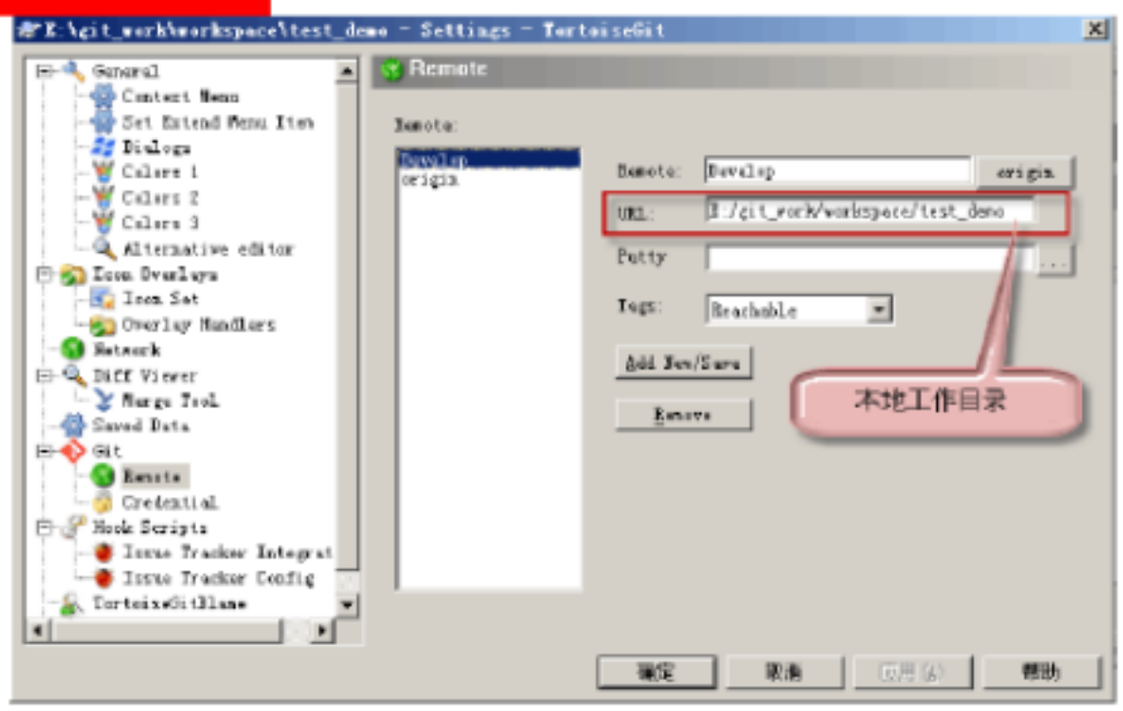

图 6-3-1

解决完冲突后，通过 Checkout 操作切换到 Develop 分支，右键-TortoiseGit-Merge，填入要合并的特性分支，填写 Merge 信息，OK 确定。

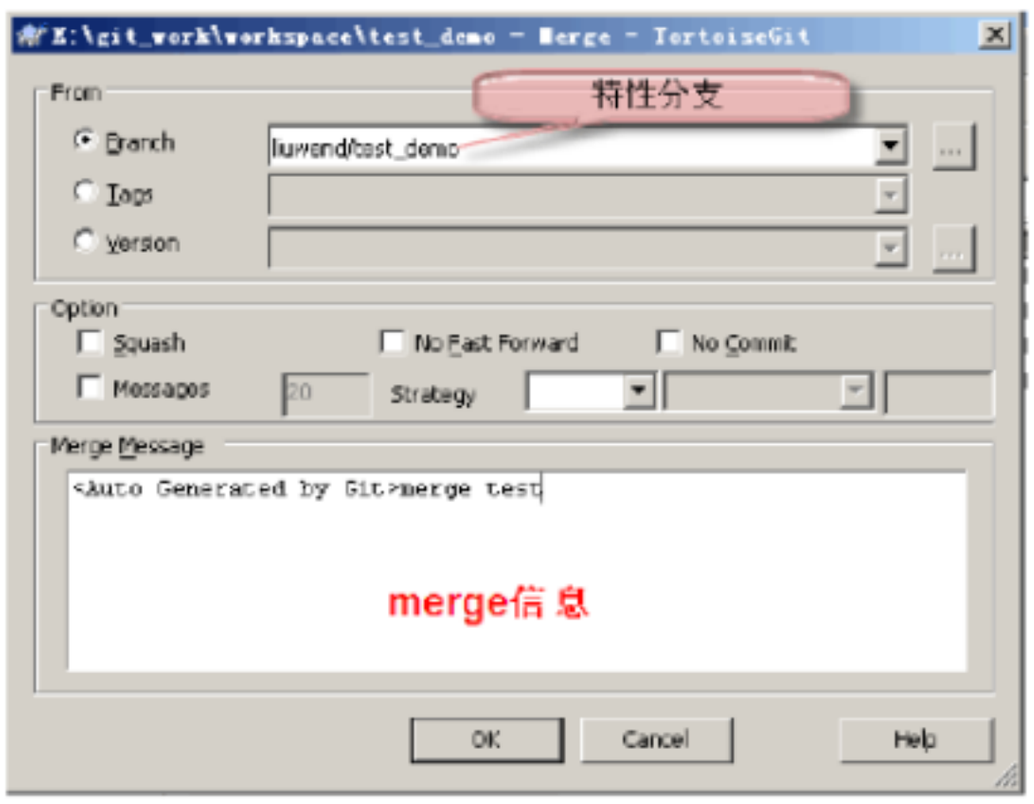

图 6-3-2

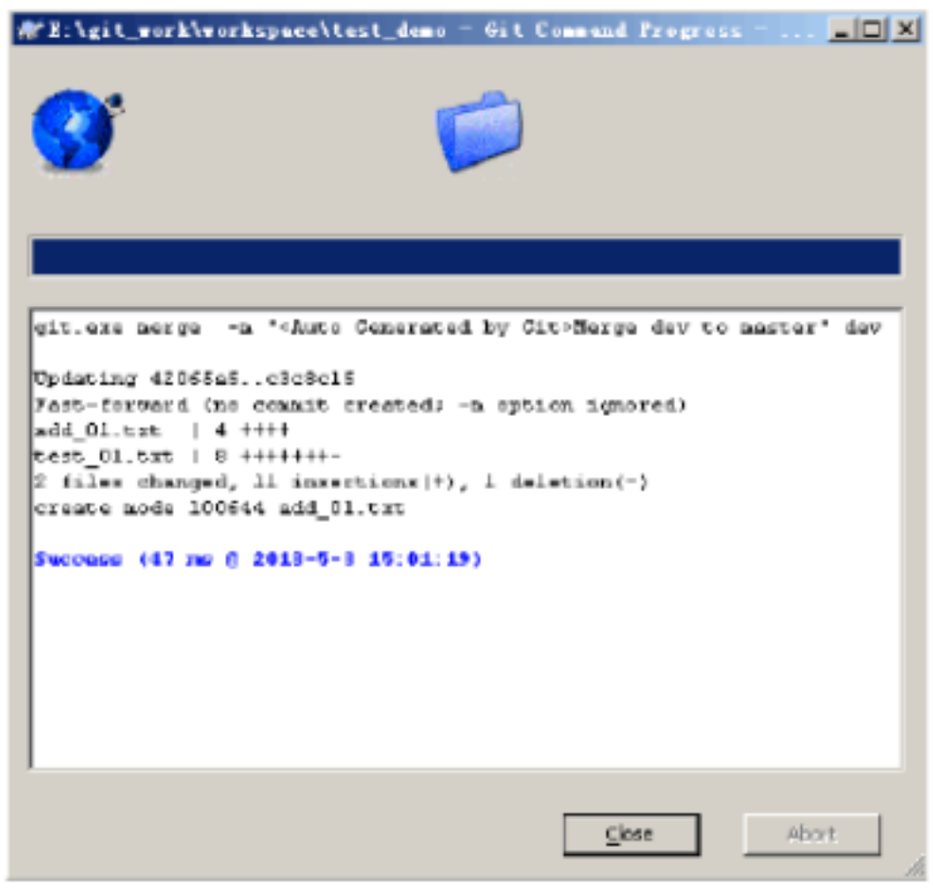

图 6-3-3

这样 merge 操作完成，特性分支仍存在，可以继续开发完善并合并到 Develop 分支。

6.4 基于 release 分支进行补丁修复

当部门内完成内部集成，达到一个相对稳定版本后，则提交产品线集成；此时由开发经理建立待发布分支（release 分支），产品线集成版本来源于待发布分支，集成过程中发现缺陷，在此分支上进行修改。

创建待发布分支同创建 Develop 分支，命名规则：R+版本号+集成日期，如：RV1.0-130309，代表 1.0 版本 13 年 3 月 9 日提交产品线集成盘对应代码分支。并推送待发布分支到服务器。

开发人员在本地基于远程待分布分支创建本地待发布分支。

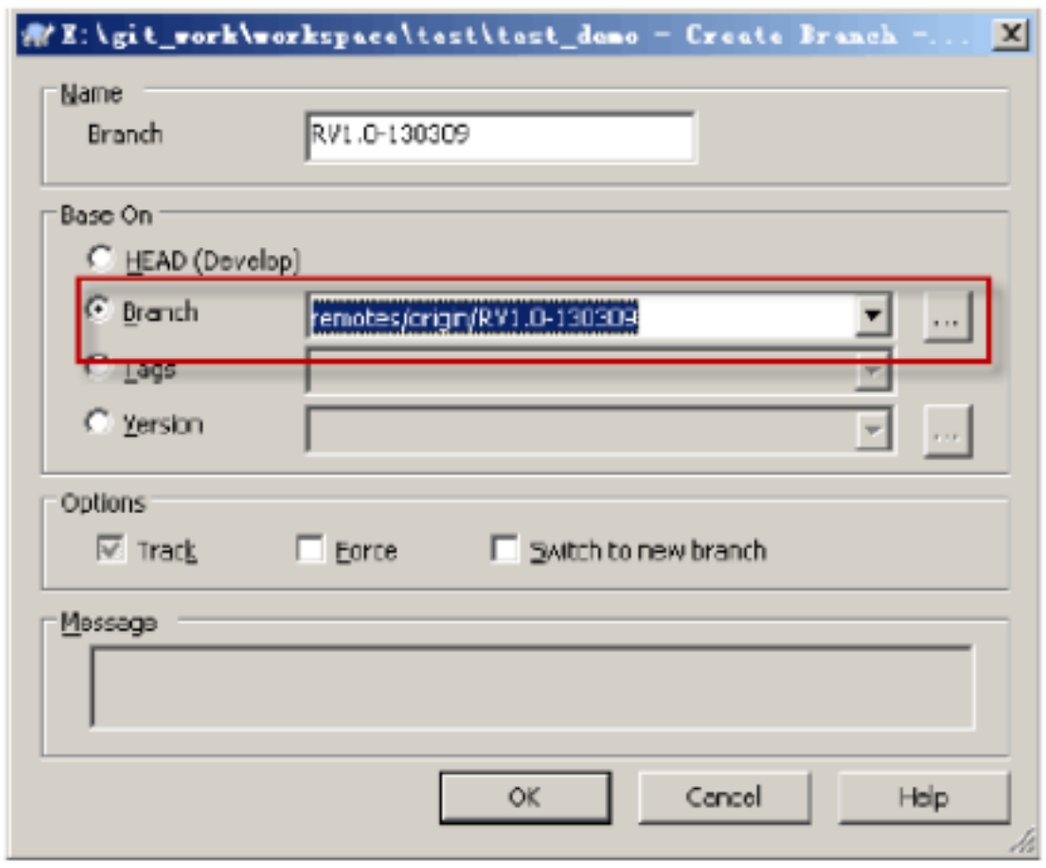

图 6-4-1

基于本地待发布分支，建立缺陷分支，完成缺陷的代码修改及合并到待发布分支（同对特性分支的操作，基于待发布分支而不是 Develop 分支）；开发人员可以继续在 develop 分支上进行新特性的开发。

在待发布分支上进行的缺陷修复，在缺陷关闭后，需要实时将更改同步到 Develop 分支上（通过 Merge 操作），从而避免后期产生大量的冲突，开发经理负责定期检查本产品的分支合并情况。避免后期出现代码合并冲突风险。

6.5 建立发布分支

产品经过产品线集成测试后，进入发版环节，此时在待发版分支建立发布分支，表明金盘对应的版本。

发布分支基于待发布分支创建，单一命名为 Gold，通过 tag 标识表明金盘对应的版本。并推送分支及标识到服务器。

创建 tag 操作（右键-TortoiseGit-Create Tag）：

在工作目录下，右键-TortoiseGit-Create Tag，填写 tag 名（以版本命名）并选择基于何处创建 tag。

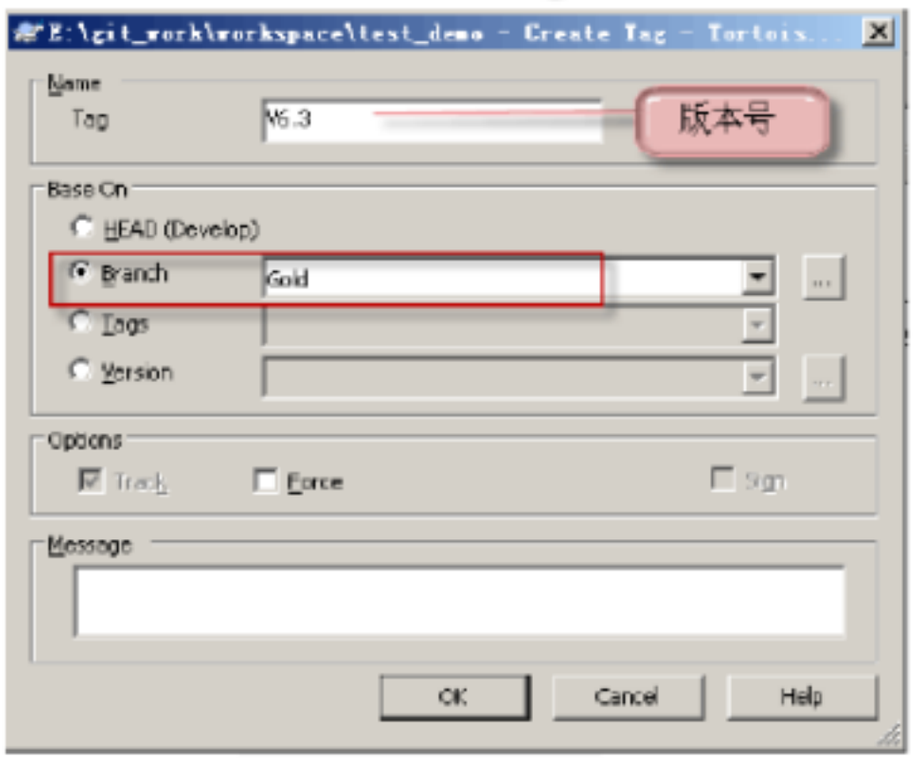

图 6-5-1

创建之后，执行 push 操作，推送到远程服务器，注意需要勾选 Include Tags 选项。

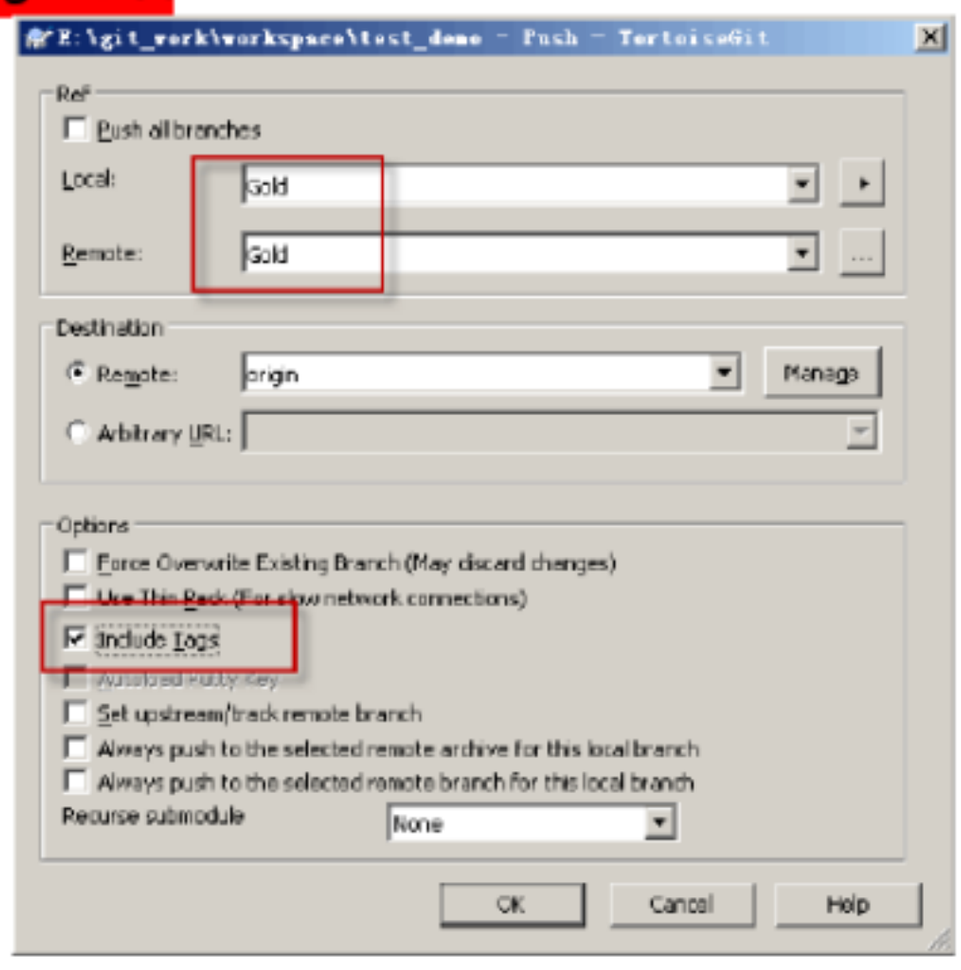

图 6-5-2

6.6 维护阶段(补丁分支)

发版后，进入维护阶段，产品在使用过程中发现的问题，通过建立补丁分支实现对问题的修订。

补丁分支基于 Gold 分支创建，单一命名为 Hotfix，主要用来修改已发布版本的 bug，基于服务器的 Hotfix 分支创建本地补丁分支，本地 Bug 修订完毕，通过开发自我验证后，推送到服务器 Hotfix 分支。

持续将补丁分支的修订代码 Merge 至主干开发分支，以保证主开发版本的完整性和质量；开发经理同样需要关注维护分支与开发集成分支的代码同步，定期进行合并。

补丁版本发布，提交至金盘分支；同时进行 tag 标识。

7. Eclipse 中使用 Git

在最新版是的 Eclipse 中已包含 egit 插件。

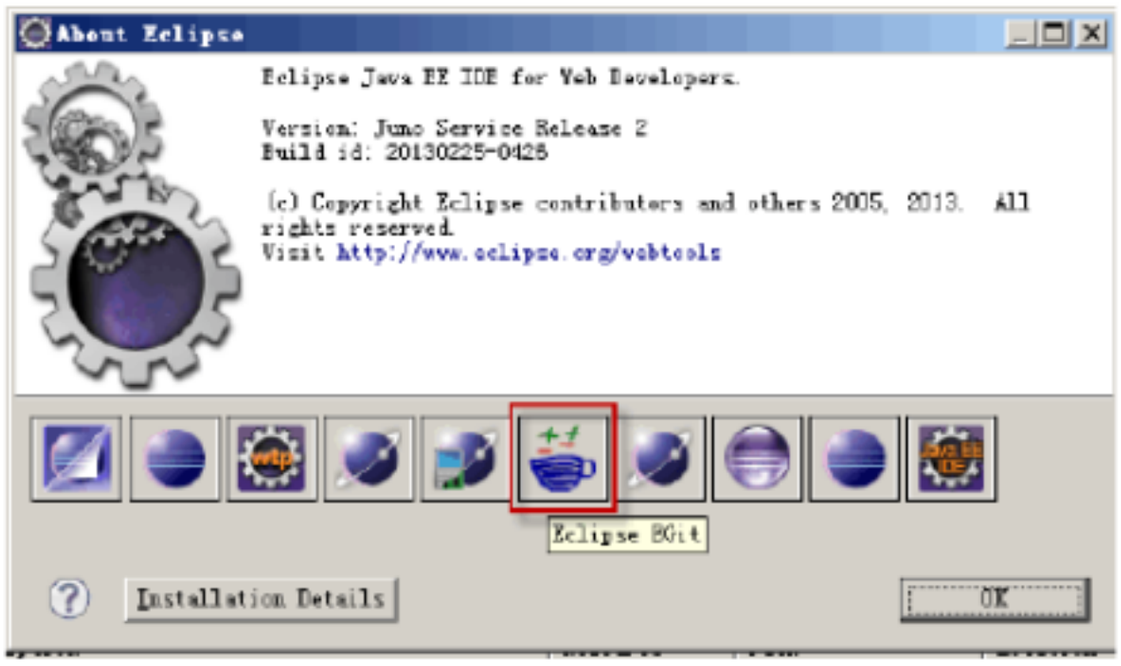

图 7-1

插件：org.eclipse.egit.repository-2.3.1.201302201838-r.zip 适用于 3.7.*以上版本的 Eclipse。

将插件中的 features 及 plugins 解压到 Eclipse 安装目录下，重启 Eclipse，Window->Preference，在 Team 下会有 Git 项。

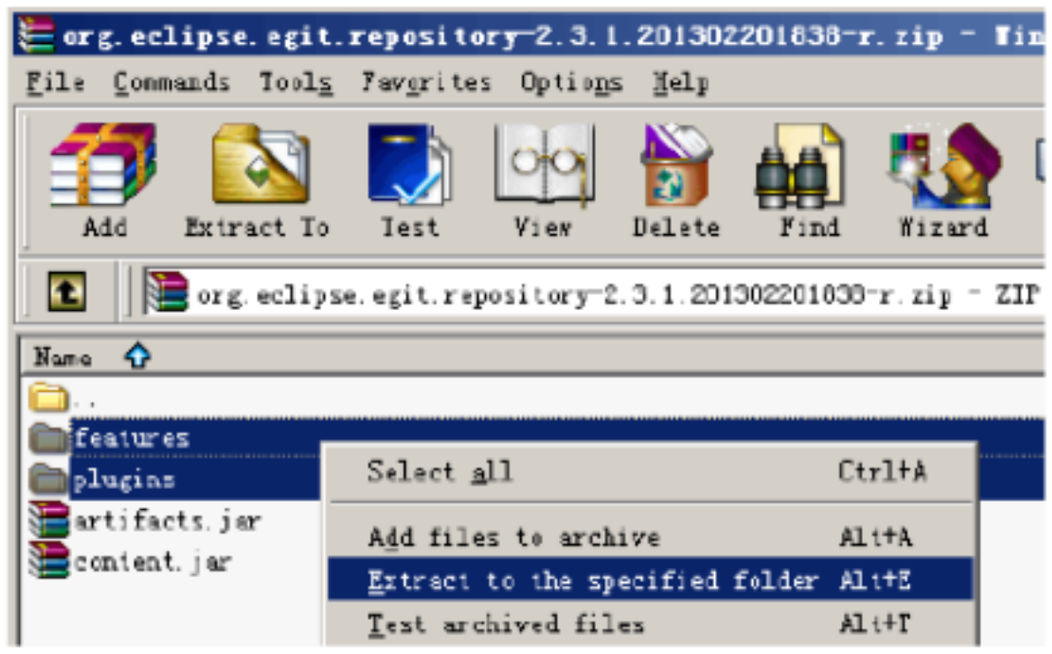

图 7-2

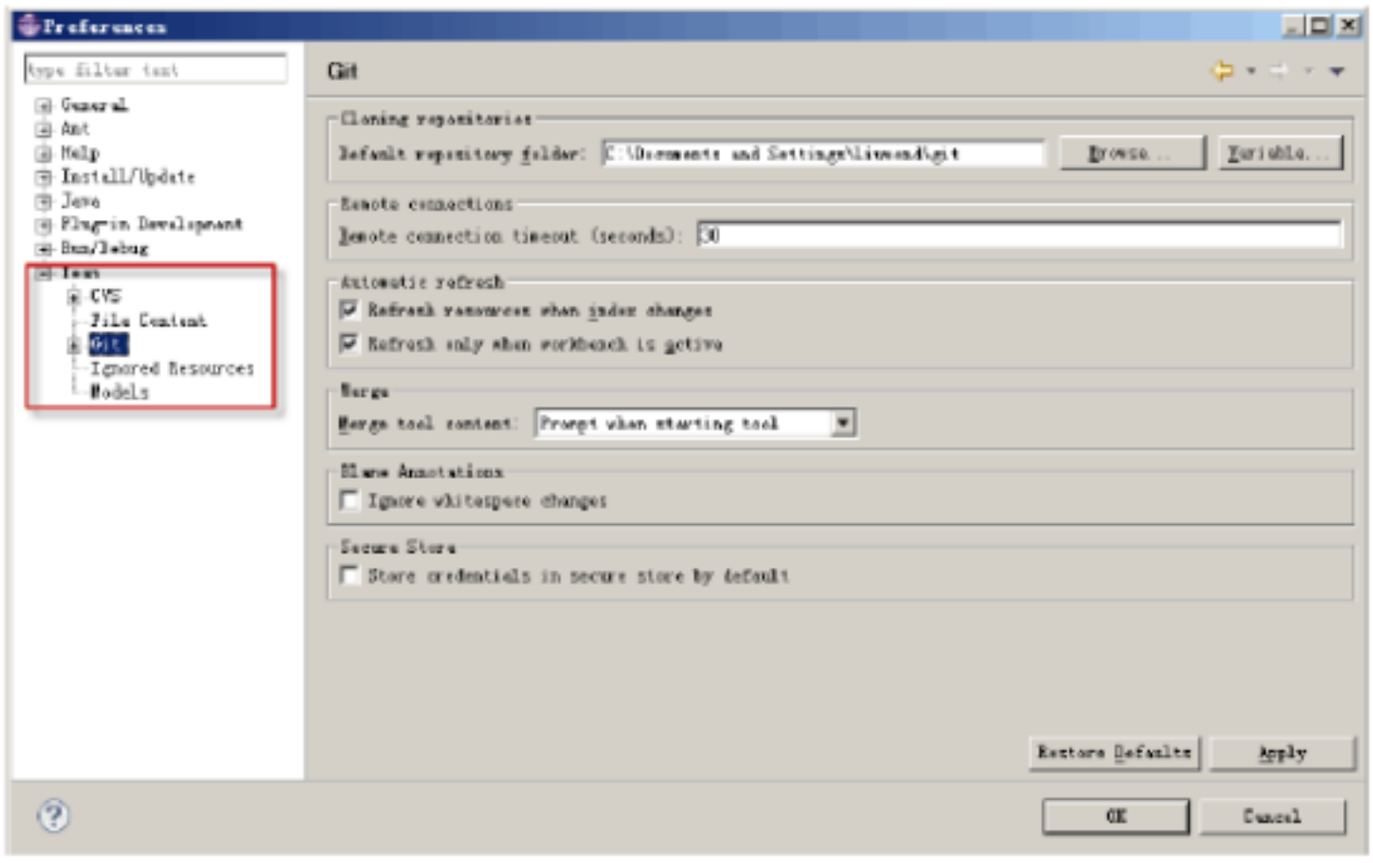

图 7-3

Window->Show View->Other...，选择 Git Repositories，显示 Git 窗口。

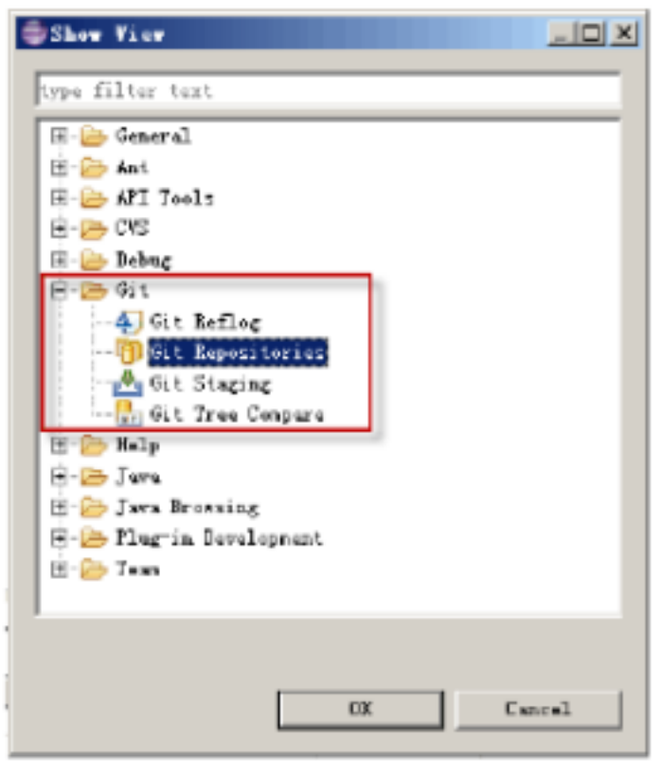

图 7-4

图 7-5

将本地 Git 仓库引入 Eclipse 中，File->Import,选择 Git 下的 Projects from Git，Next；

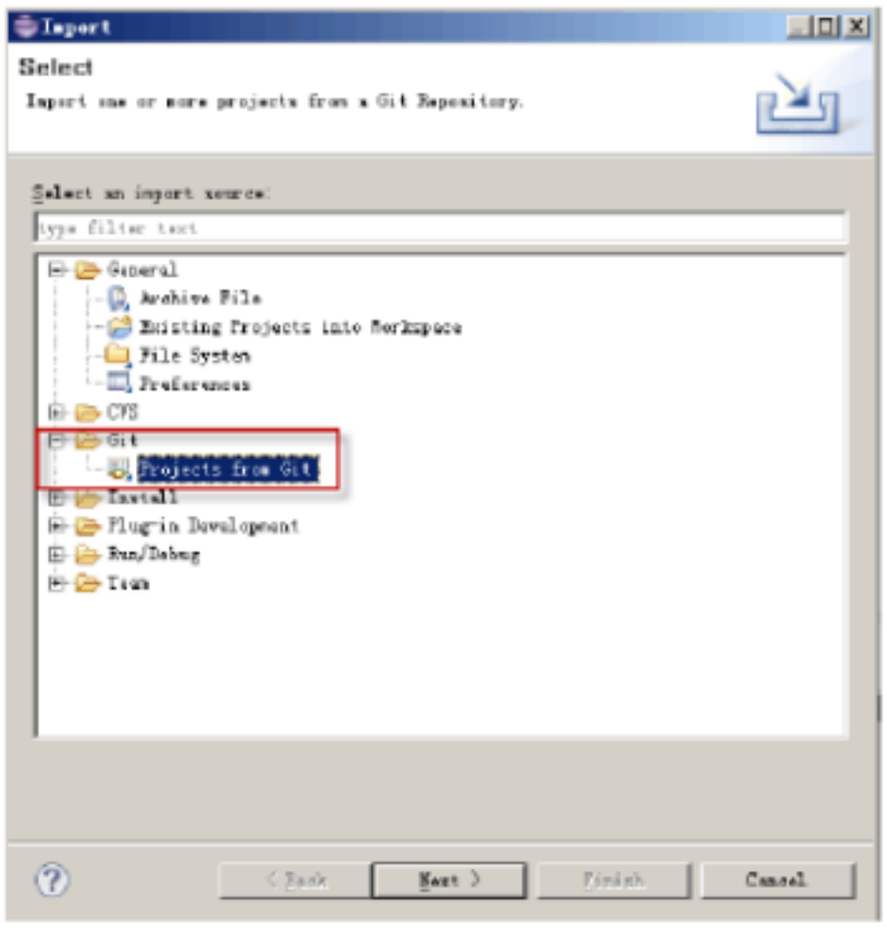

图 7-6

选择 local，Next；

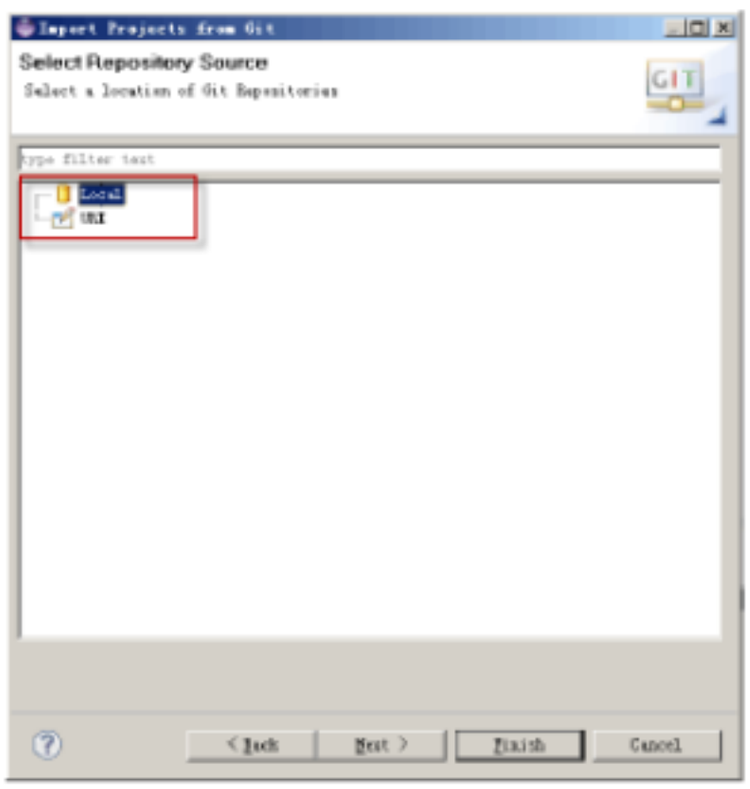

图 7-7

Add... 找到要引入的 Git 仓库，Finish。

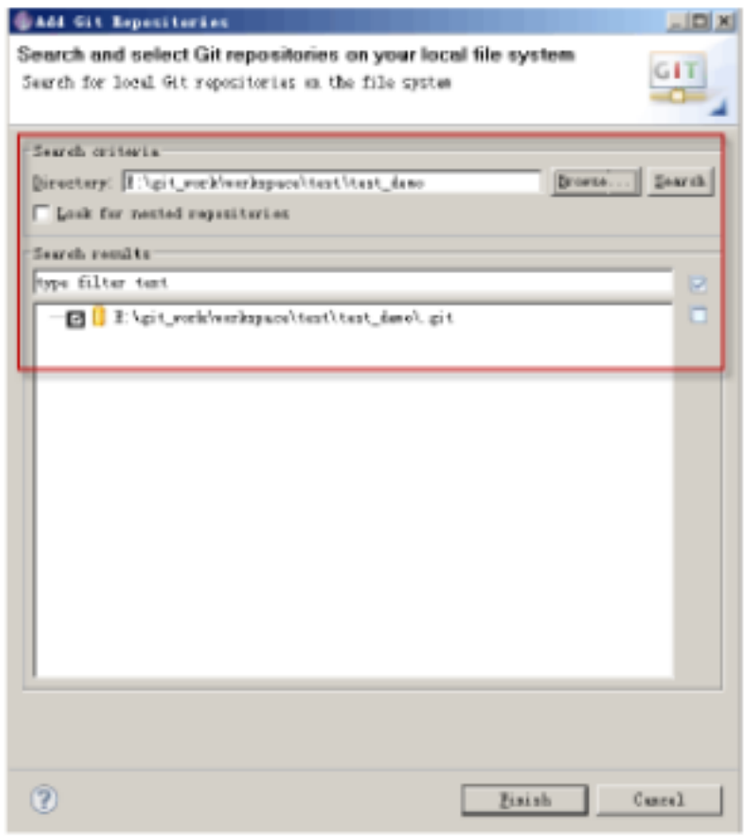

图 7-8

可继续 Add... 添加其他仓库，添加完成后，选择要引入的 Git 仓库，Next。然后选择 Import existing projects，Next，Finish。

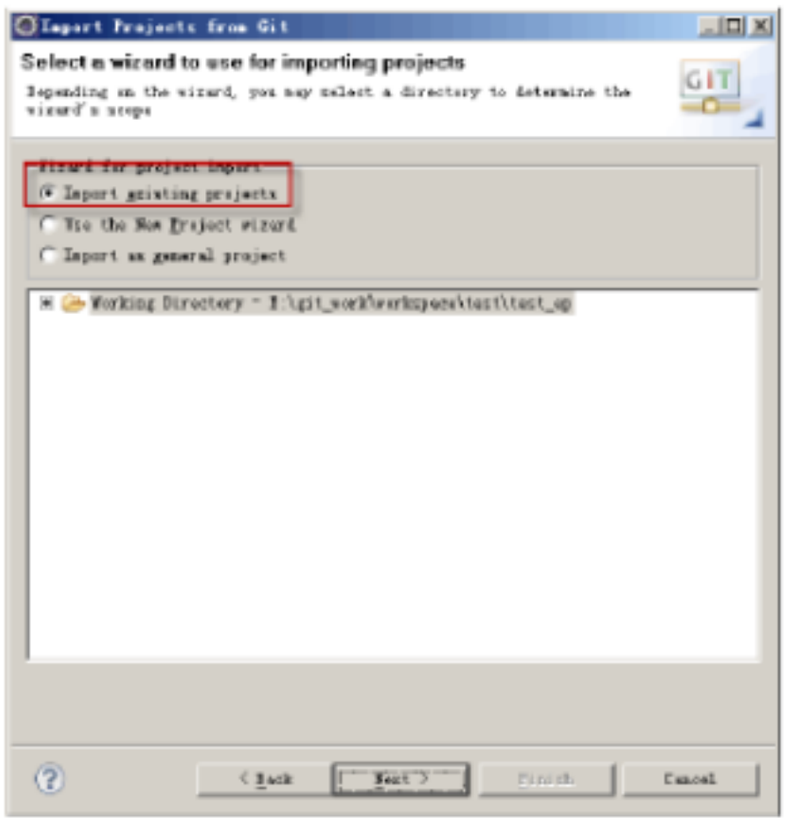

图 7-9

在 Package 及 Git 窗口，可看到引入的 Git 仓库的具体信息。

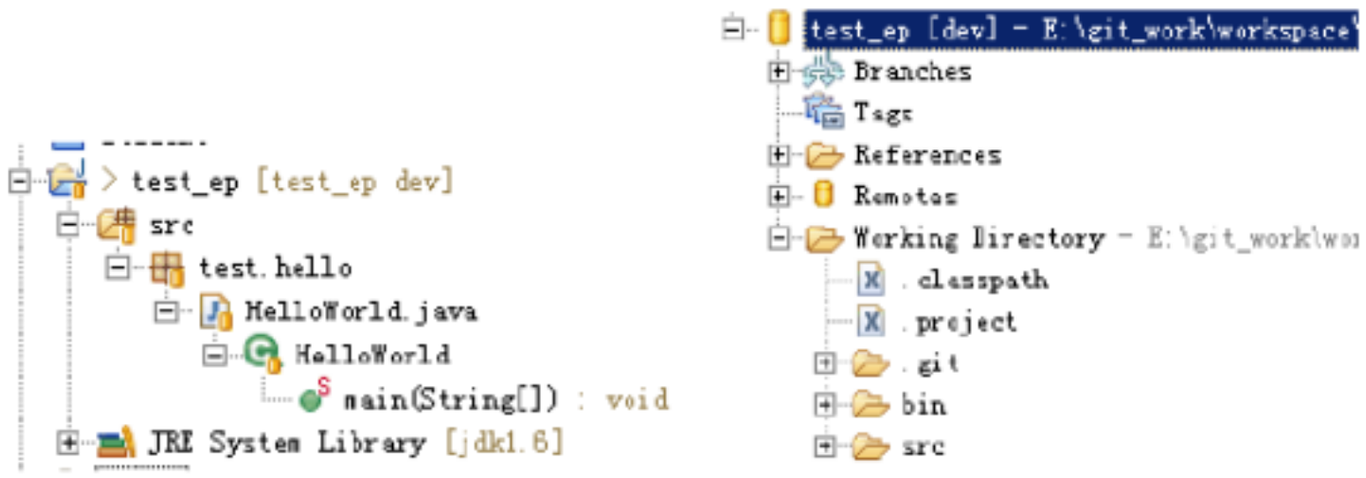

图 7-10

在 Working Directory 中对文件进行操作；在 Git 仓库上右键，会出现相关 Git 操作的选项。

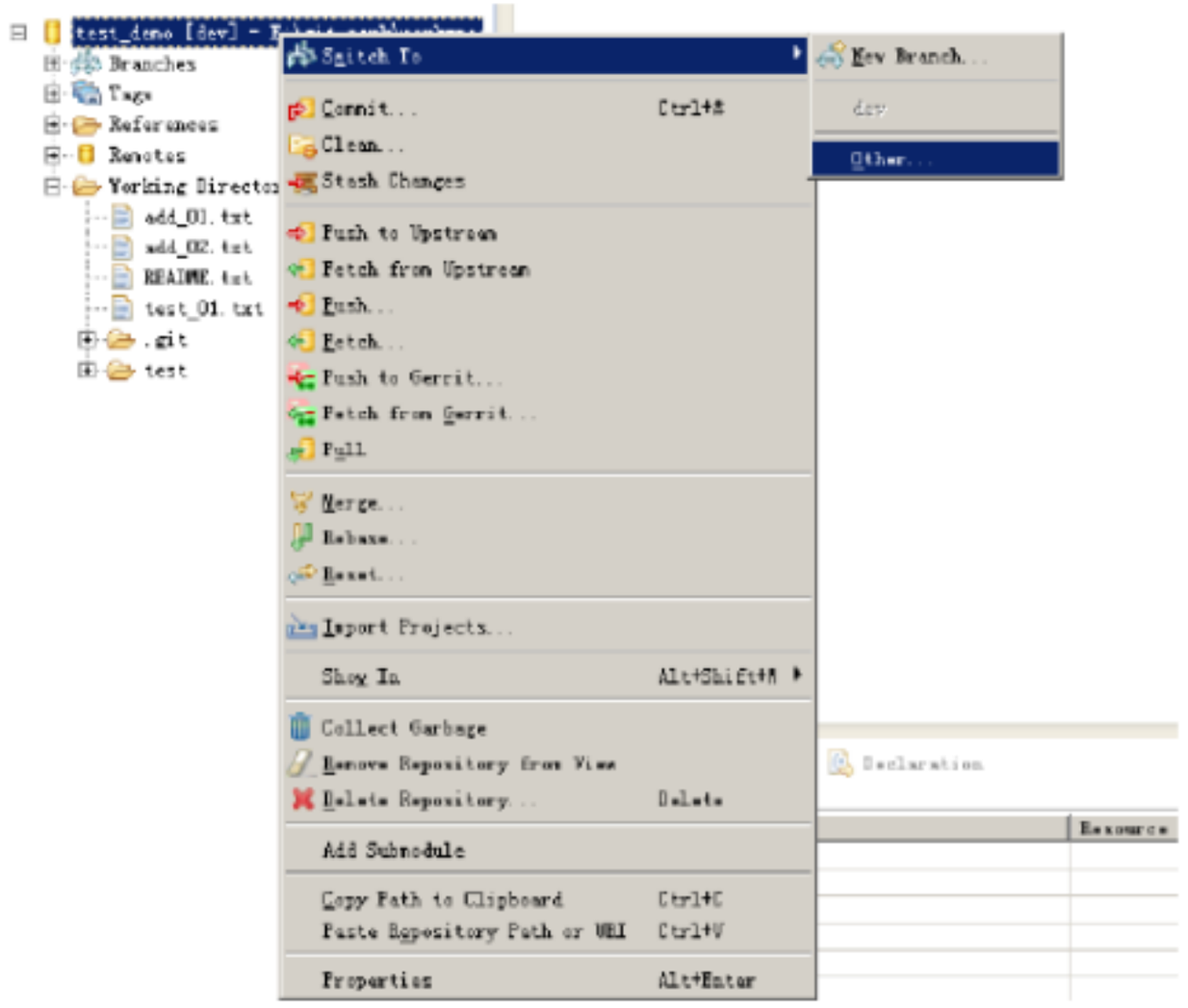

图 7-11

相关操作与在工作目录下使用 TortoiseGit 操作基本相同。

Clone 操作：

在 Git 窗口左上角点下拉三角符号，选择 Clone a Repository，打开 Clone 引导。

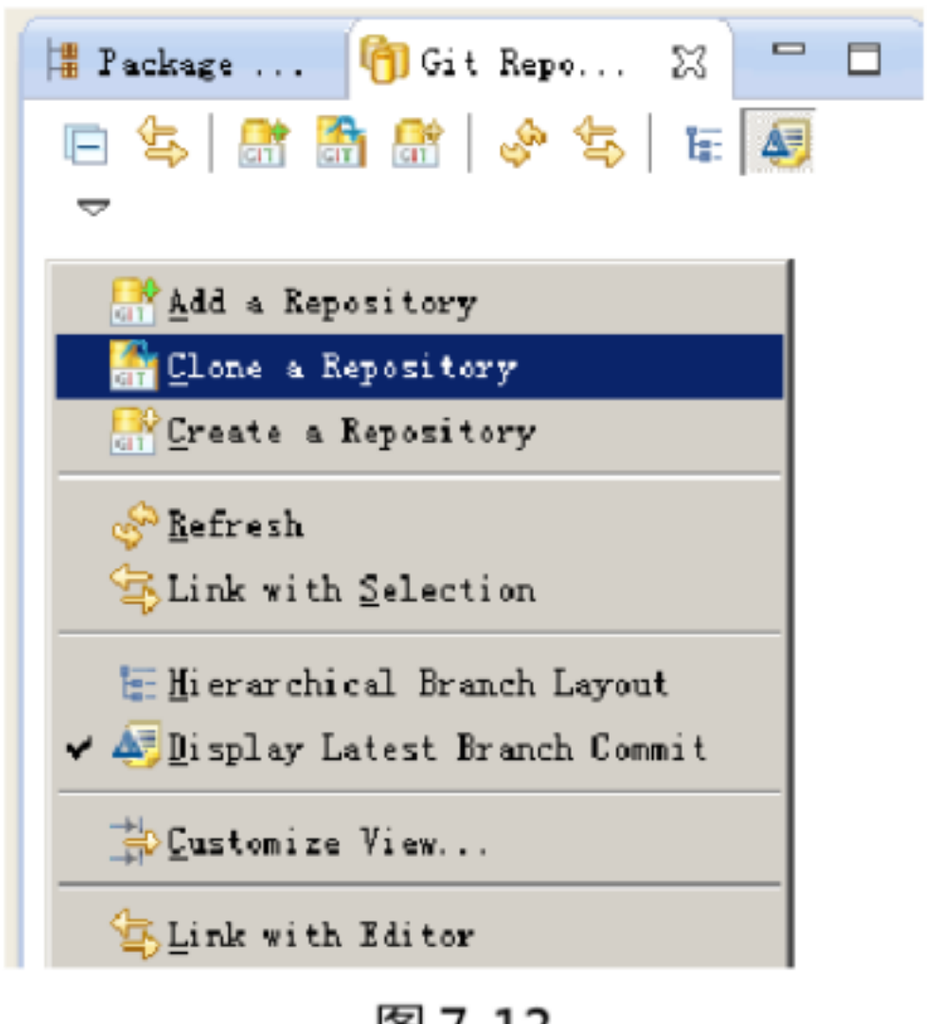

图 7-12

输入 SSH 链接，Next。

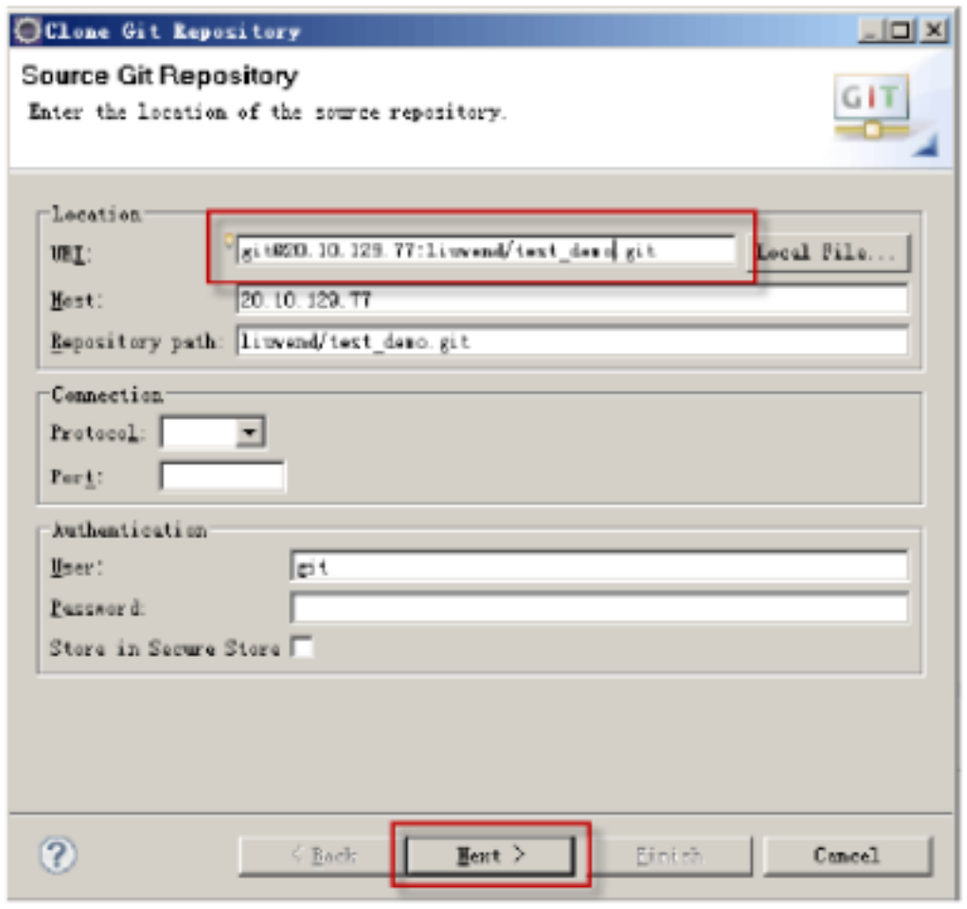

图 7-13

选择要 clone 的分支，Next。

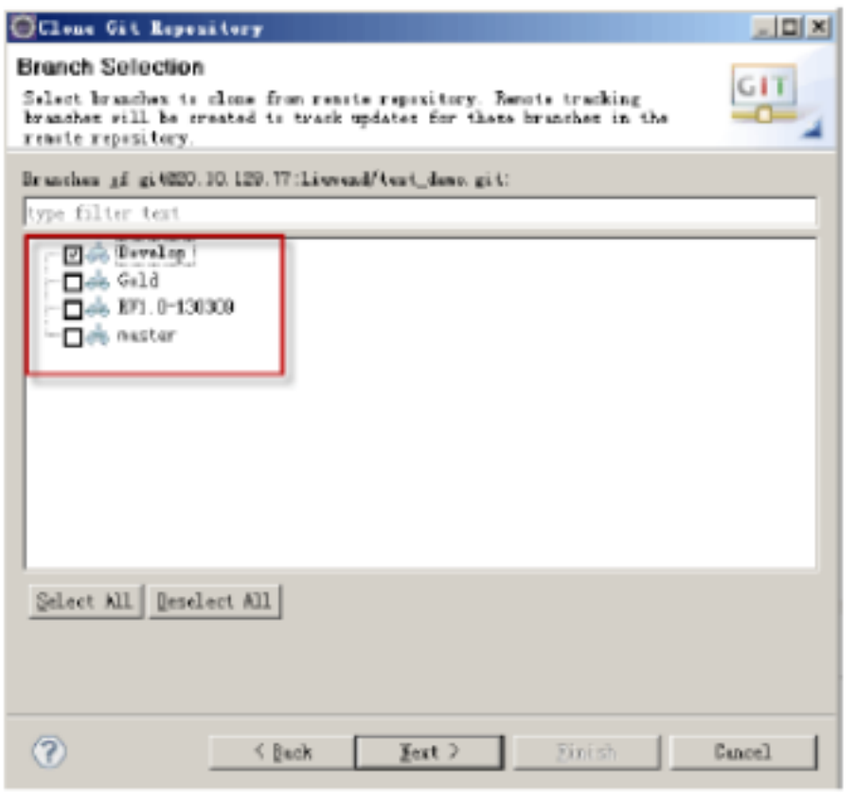

图 7-14

选择本地目录，Finish。

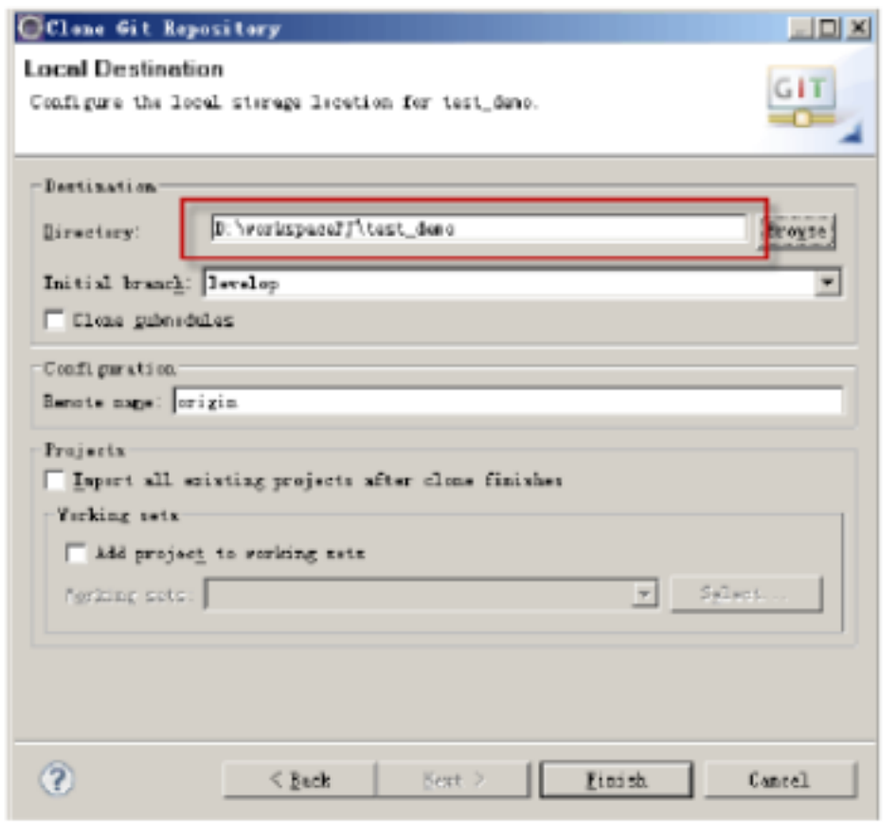

图 7-15

Create Branch 操作：

在 Git 仓库处右键-Switch To-New Branch，打开引导，选择源分支，并填入新分支名，Finish。

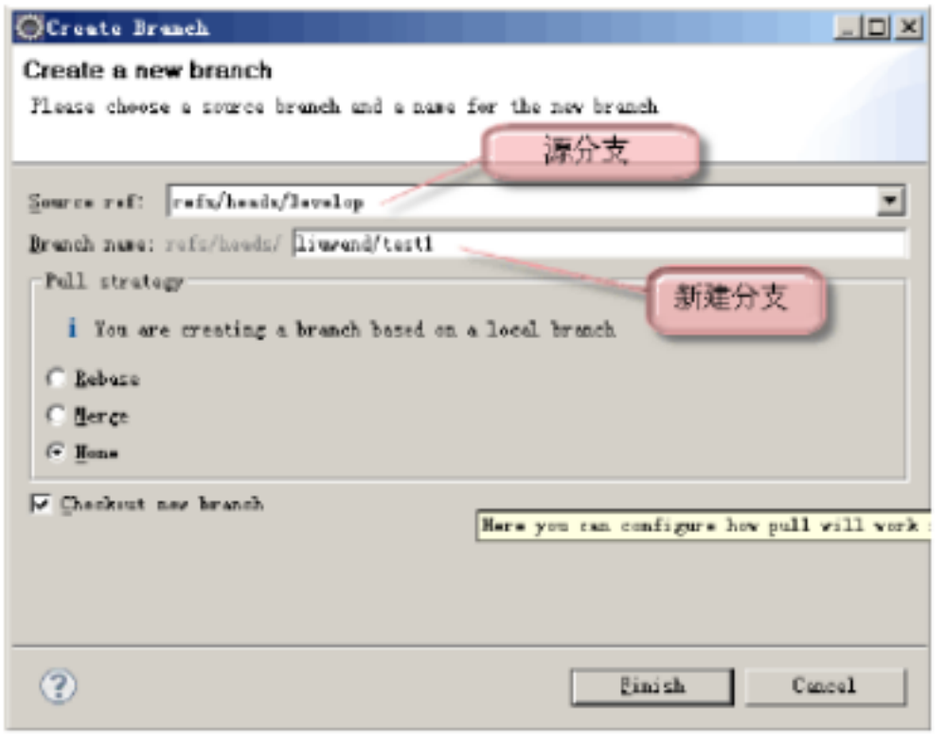

图 7-16

Checkout 操作：

在 Branches 里选择要切换的分支，双击即可（或右键-Checkout）。

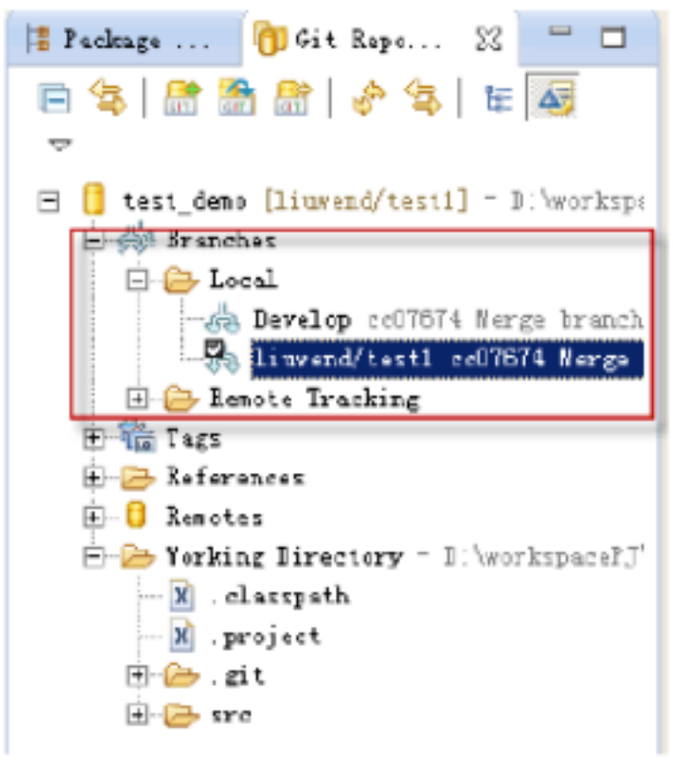

图 7-17

Commit 操作：

在 Git 仓库处右键-Commit，打开提交界面。选择提交的文件，提交信息必填，Commit 即可。

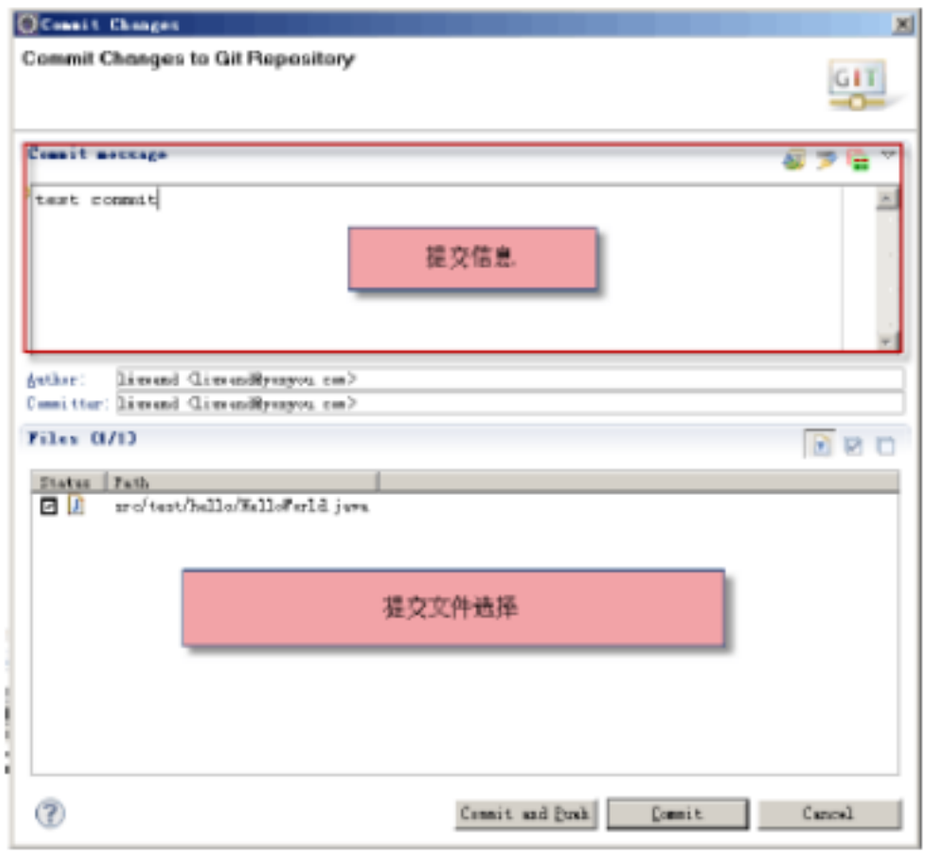

图 7-18

Merge 操作：

首先切换到目标分支，然后在 Git 仓库处右键-Merge，打开 Merge 界面，选择源分支，Merge。（如存在冲突，通过前文介绍的使用 TortoiseGit 的方法解决冲突）

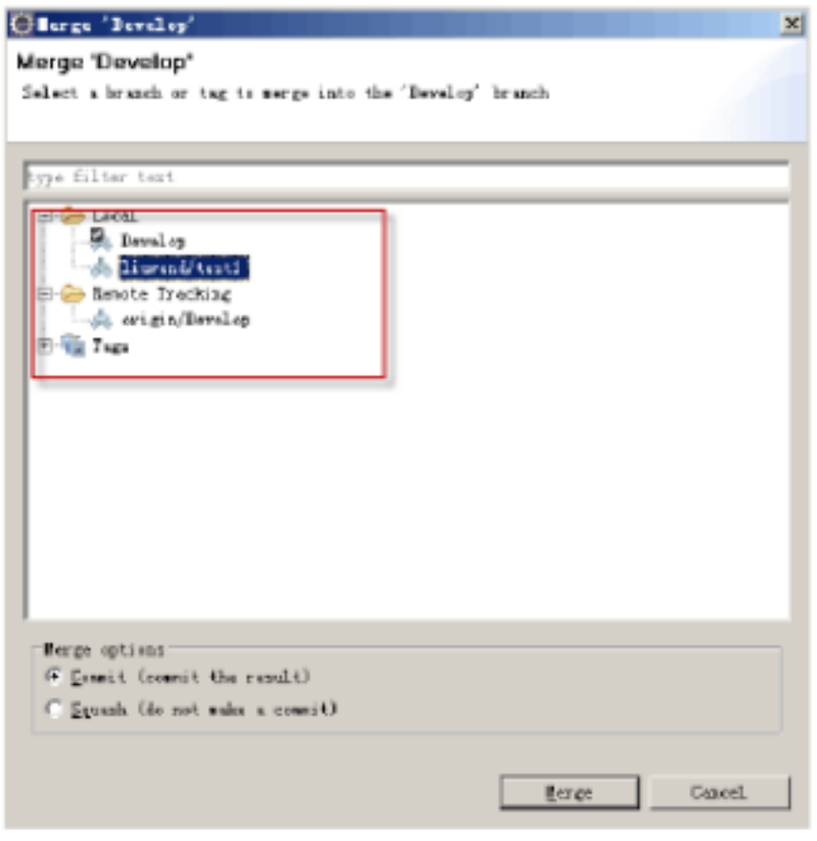

图 7-19

Create Tag 操作：

首先 Checkout 到要建立 tag 的分支，然后在 Tags 上右键-Create Tag，填入 tag 名及信息，OK 即可。

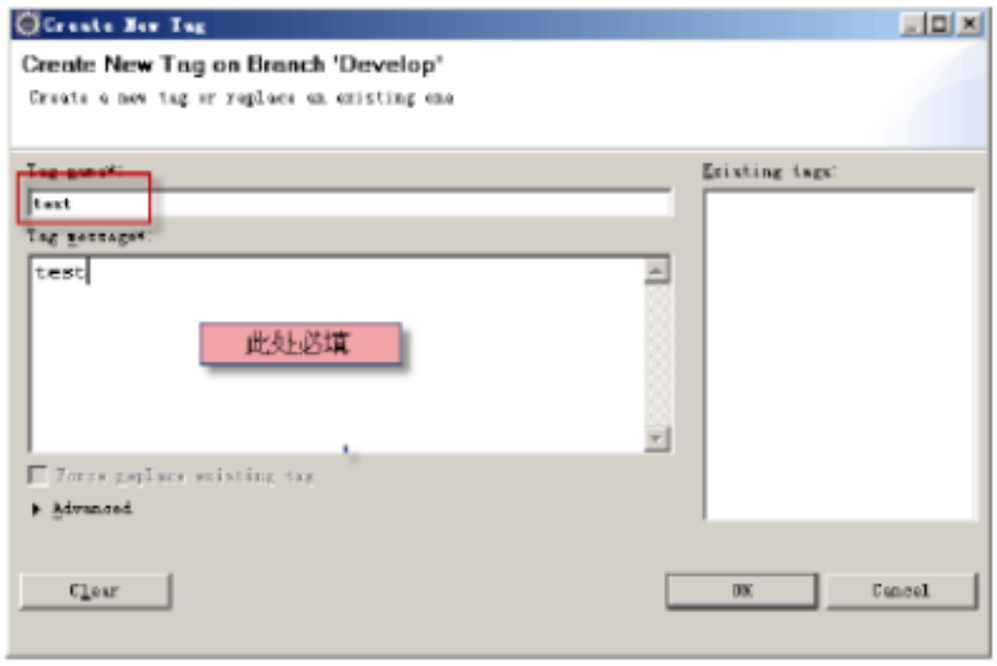

图 7-20

Pull 操作：

在 Git 仓库处右键-Pull，执行完结果如下：

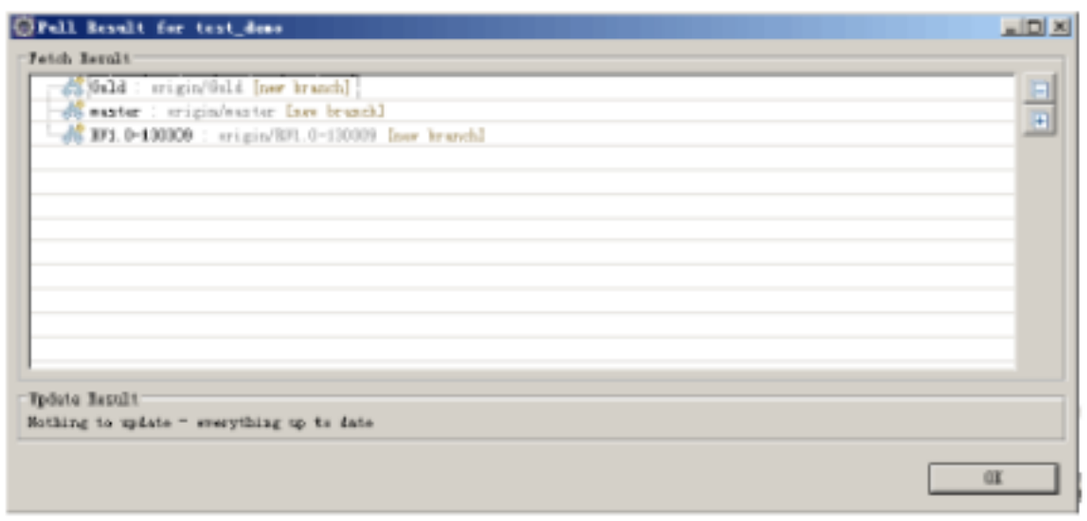

图 7-21

Push 操作：

在 Git 仓库处右键-Push，打开引导界面，选择远程仓库，Next。

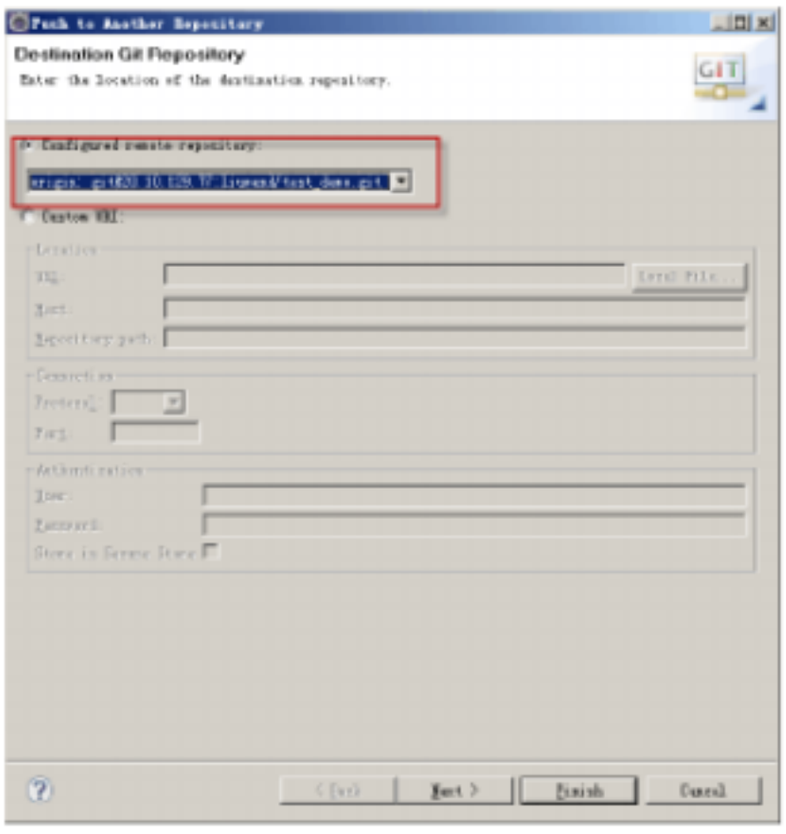

图 7-22

选择源分支与目标分支，Add Spec 添加入列表（如果要推送 tag，点击 Add All Tags Spec），Finish 即可。

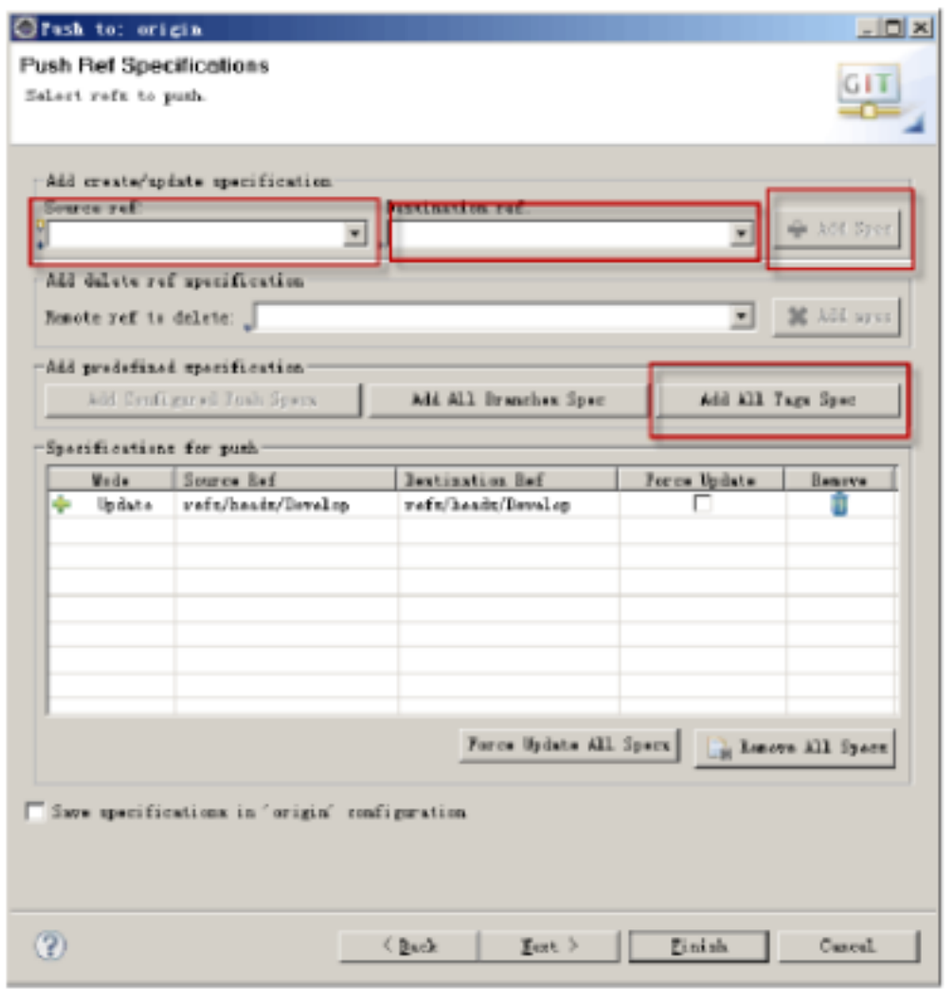

图 7-23

结果如下：

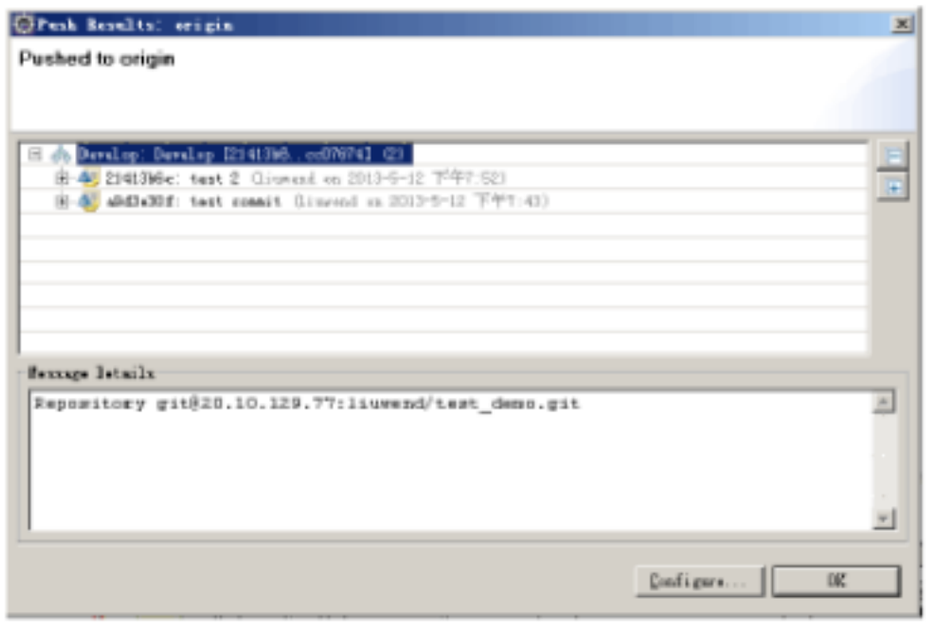

图 7-24

查看提交日志：

在 Git 仓库处右键-Show In-History。

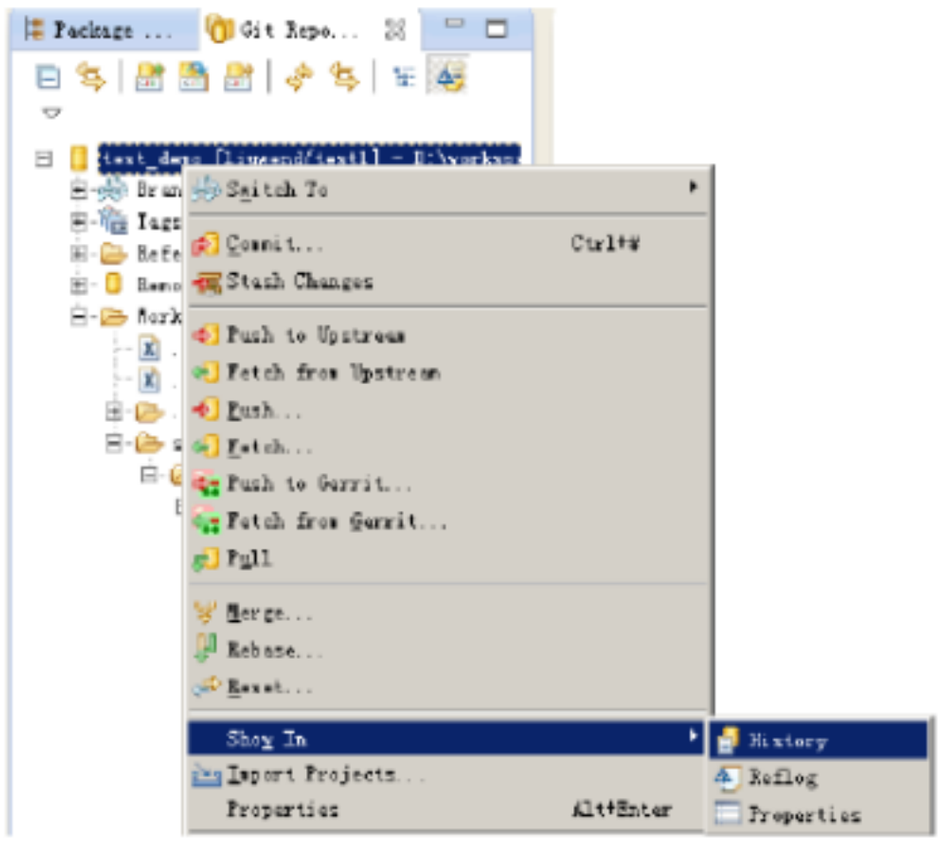

图 7-25

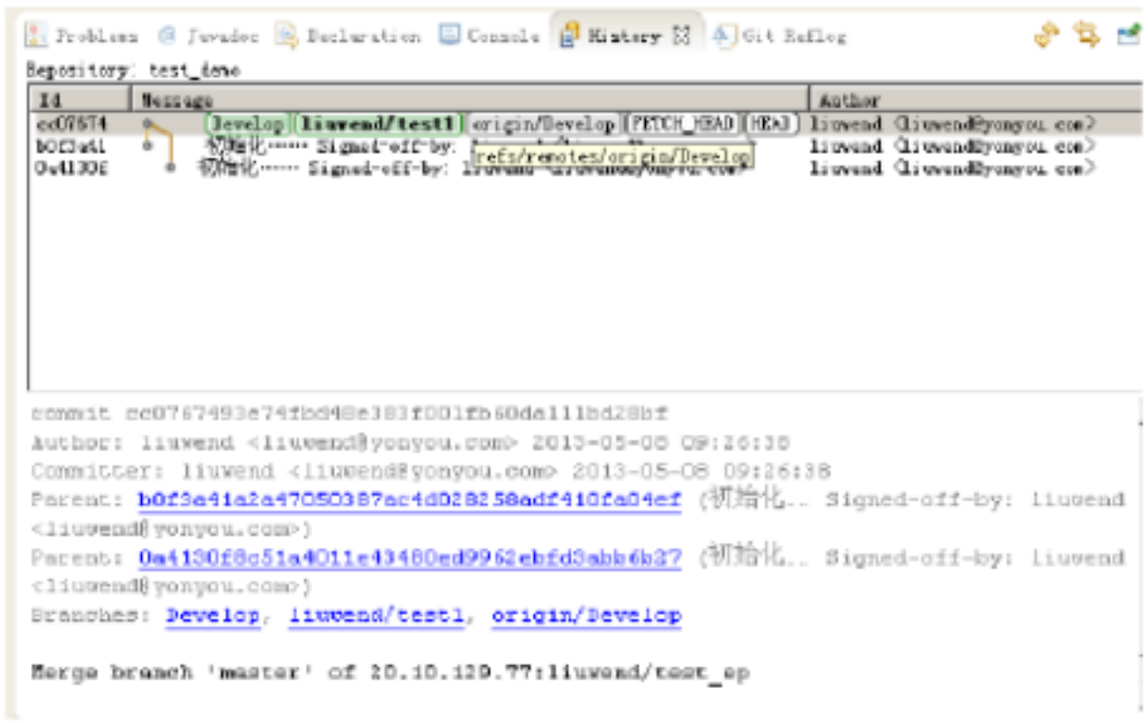

图 7-26

（同步操作在 Eclipse 中没有集中的界面显示；冲突解决，需使用之前在工作目录中解决冲突的方法）

8. 常见问题解决

8.1 安装 TortoiseGit 时，Chosse SSH Client 选择了第一项。

如下图所示：

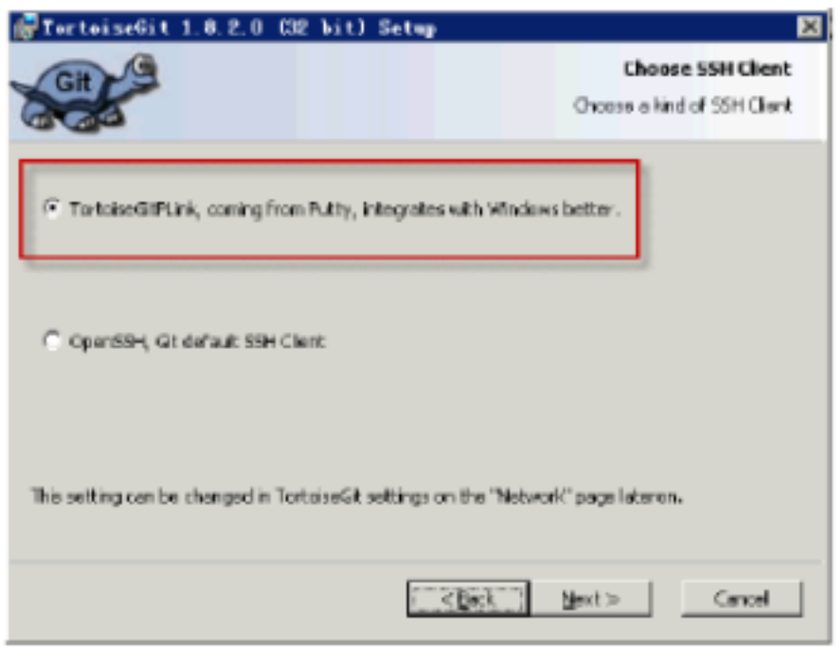

图 8-1-1

解决方法：

右键->TortoiseGit->Settings，选择 Network 标签，指定 Git 目录中的 ssh.exe，确定即可。

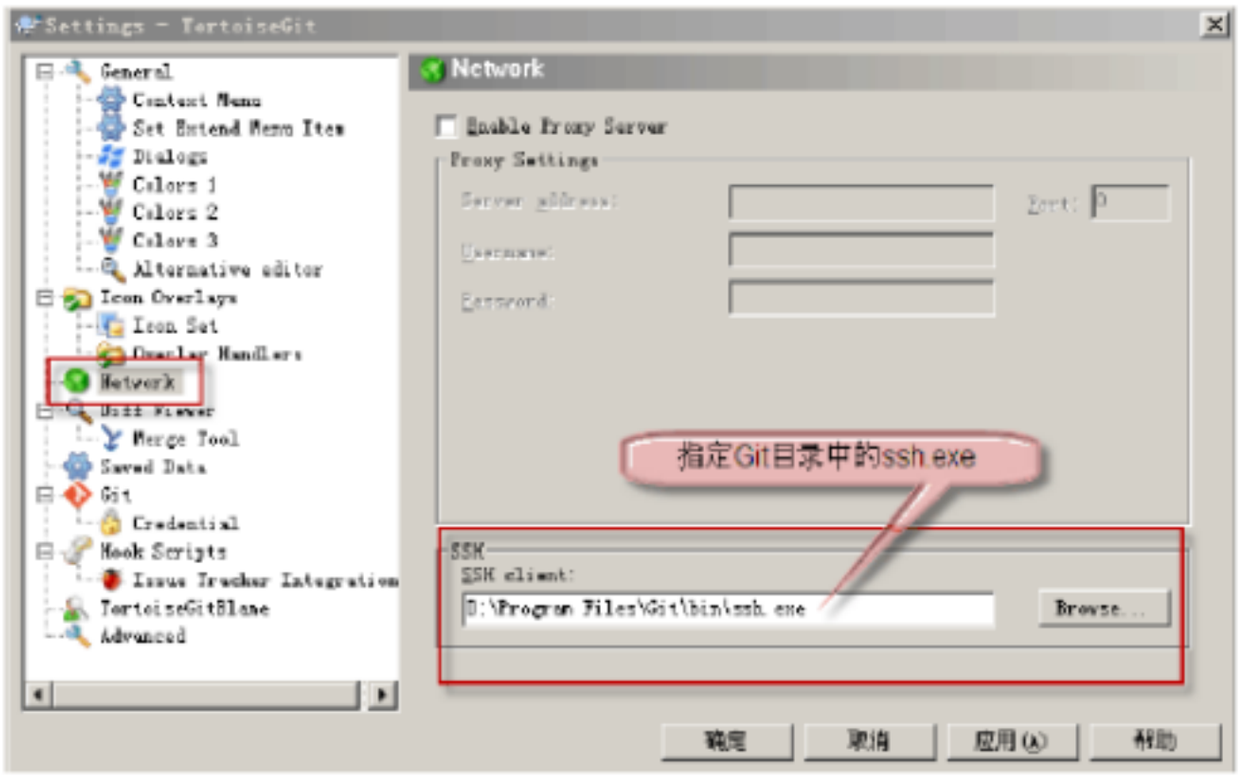

图 8-1-2

8.2 其他问题

（具体的 Git 命令及使用可参考《Git_Community_Book_中文版》）