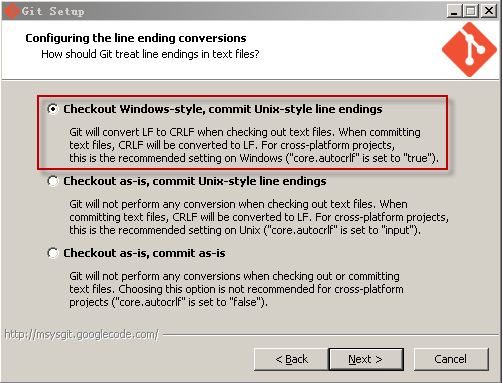
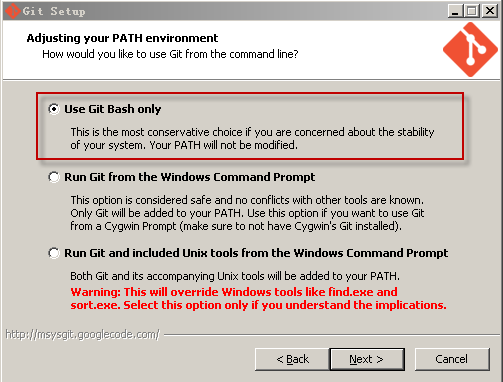
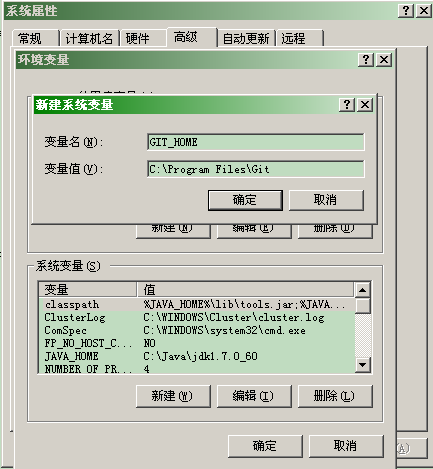
### 1、安装Git

运行Git-xxx.exe，选择Next，默认安装即可。



### 2、GIT\_HOME 配置

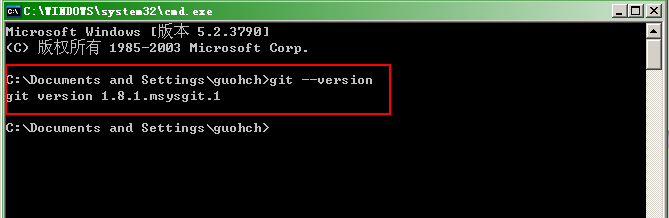
在系统环境变量中添加 GIT\_HOME 变量（如：C:\Program Files\Git）



在path中添加 git-cmd 、git-bin路径（.;%GIT\_HOME%\cmd;%GIT\_HOME%\bin;）



测试配置是否生效（打开cmd 输入“git --version”，显示如下即OK）



### 3、配置全局用户名和邮箱

安装完成后，打开Git Bash（安装Git后，会生成桌面快捷方式）。

执行下面两行命令（注意：参数名与参数值之间不要使用“=”，空格即可）

|  |
| --- |
| git config --global user.name 你的名字  git config --global user.email 你的邮箱 |

可通过下面两行命令查看配置

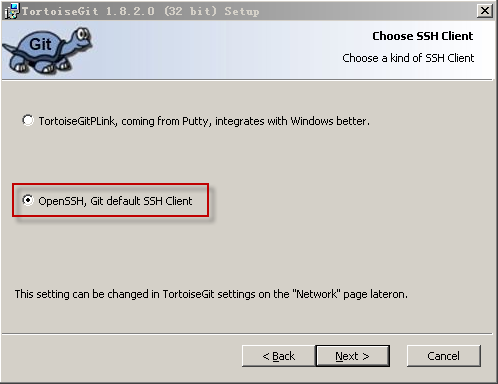
|  |
| --- |
| git config user.name  git config user.email |

### 4、安装TortoiseGit

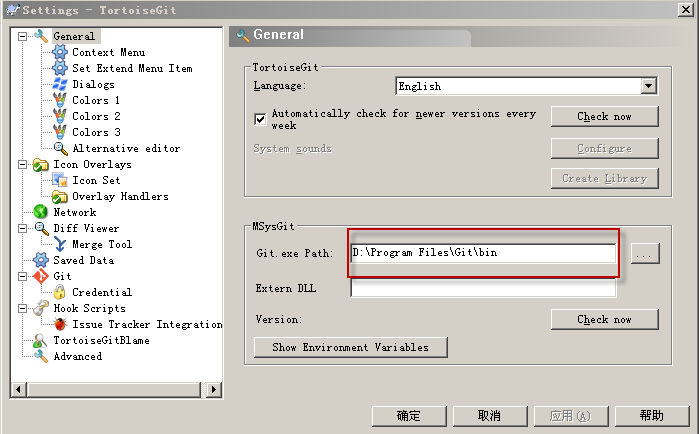
运行TortoiseGit-xxx.msi

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。

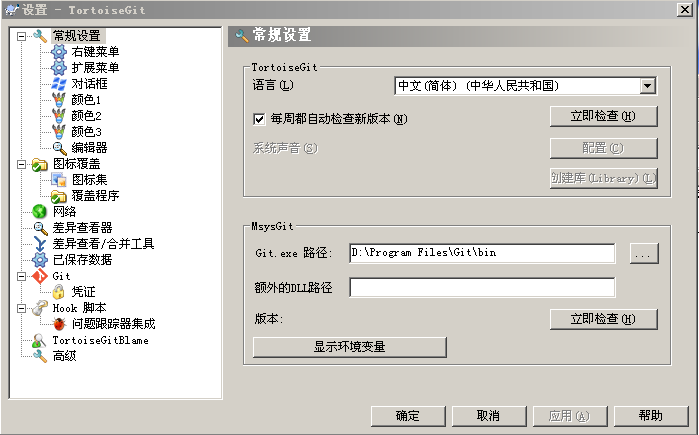
[选择SSH客户端错误解决](#_1、安装TortoiseGit时，Chosse_SSH_Client误)



安装完成后，右键-TortoiseGit-Settings，查看设置（TortoiseGit会自动检测到Git的安装路径）。



另外可选择对应的语言包，安装简体中文。



5、SSH公钥认证

用户通过SSH公钥认证，建立与服务器端的通信。

① 创建自己的公钥/私钥对。

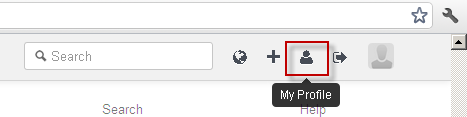
打开Git Bash，输入以下命令，其它输入回车略过。

|  |
| --- |
| ssh-keygen -t rsa |
|  |

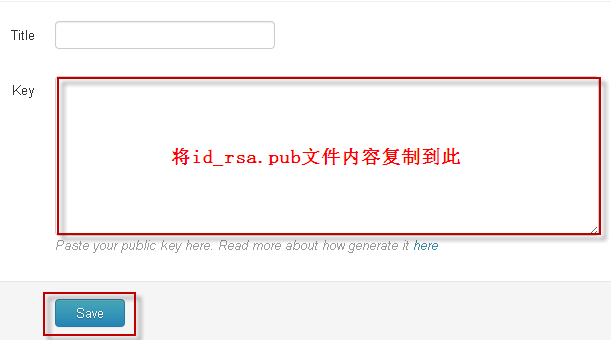
完成后会在用户主目录（C:\Documents and Settings\user\_name）下的.ssh目录下生成私钥id\_rsa及公钥id\_rsa.pub两个文件。

② 公钥id\_rsa.pub配置到GIT服务器（也可以添加到自己的GITHUB）。

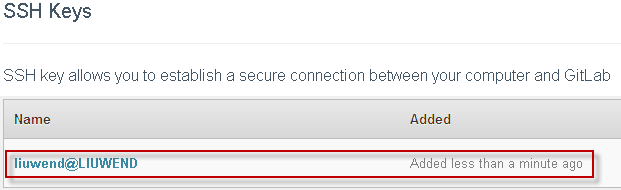
通过谷歌浏览器访问服务器网址，输入自己的用户名/密码登录，进入右上角的My Profile界面。



进入SSH Keys 界面，点击 Add new，并点击save保存。



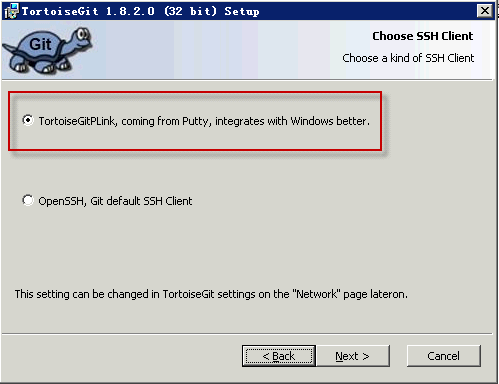
这样公钥就会出现在SSH Keys的列表中。



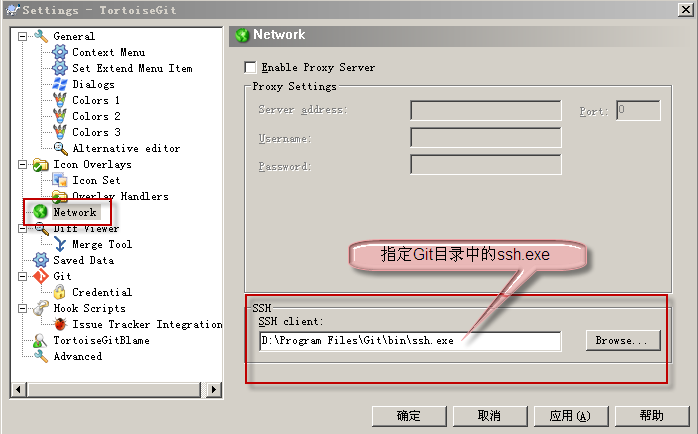
到此，通过SSH公钥认证，建立与服务器端的通信完成。

### final、常见问题解决

#### 1、安装TortoiseGit时，Chosse SSH Client误选了第一项。



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。

#### 2、其他问题

（具体的Git命令及使用可参考《Git\_Community\_Book\_中文版》）

# 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。

# 开发过程中常用操作简介（使用TortoiseGit）

## 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](mailto: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。

### 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：

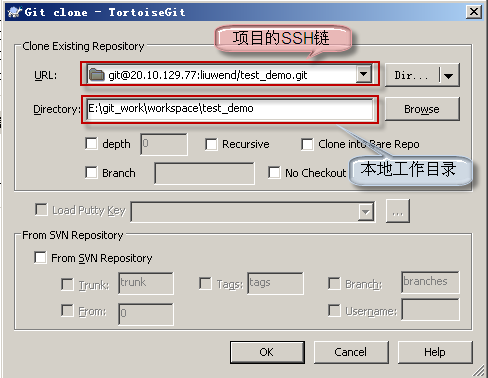


图6-1-1

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）。

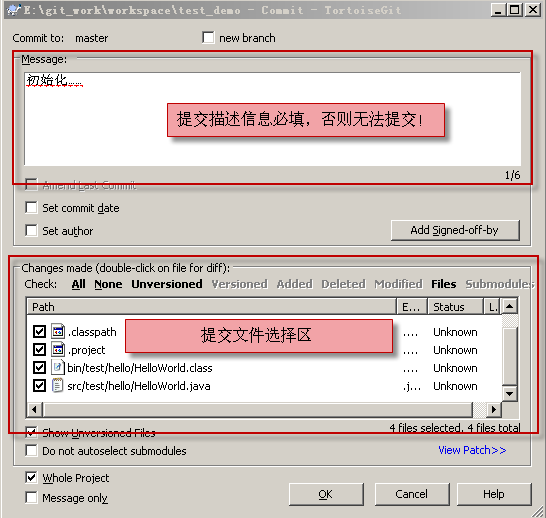


图6-1-2

选择OK，完成Commit操作。

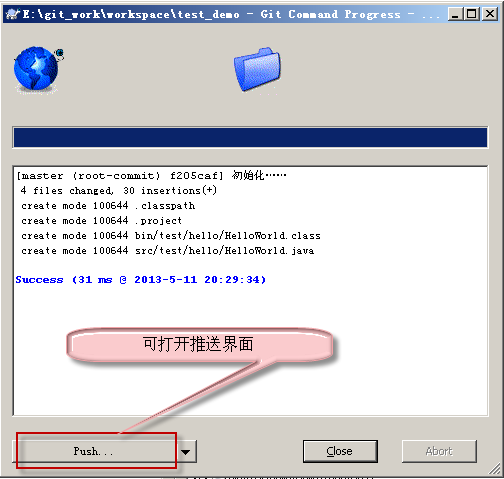


图6-1-3

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）

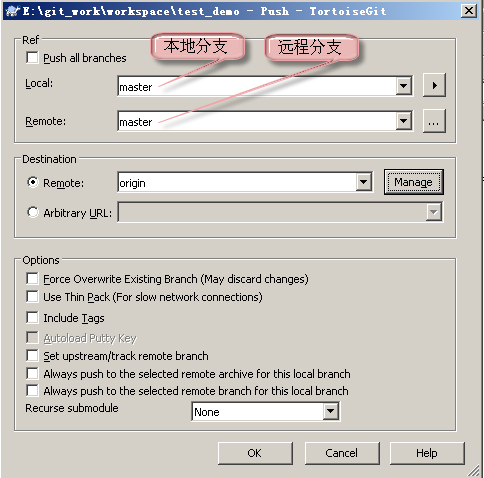


图6-1-4

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。

### 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。

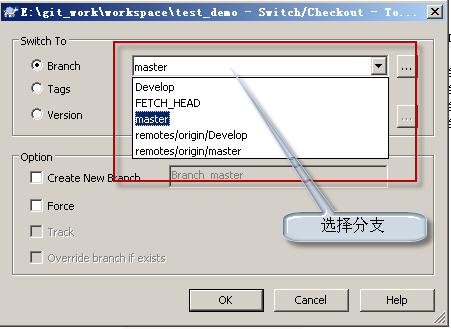


图6-1-5

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。

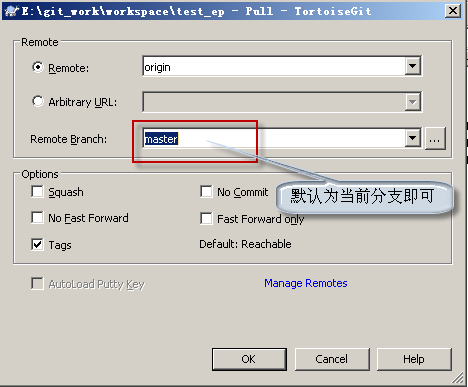


图6-1-6

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。

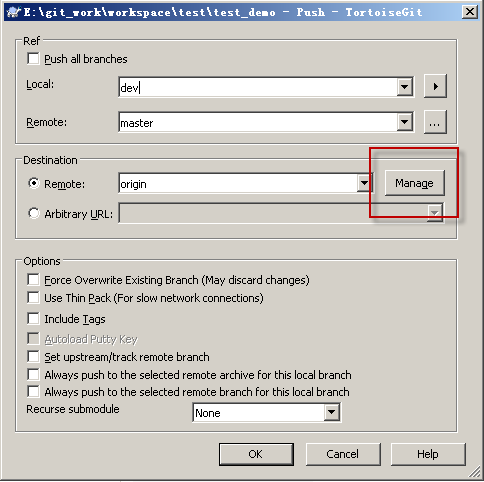


图6-1-7

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。

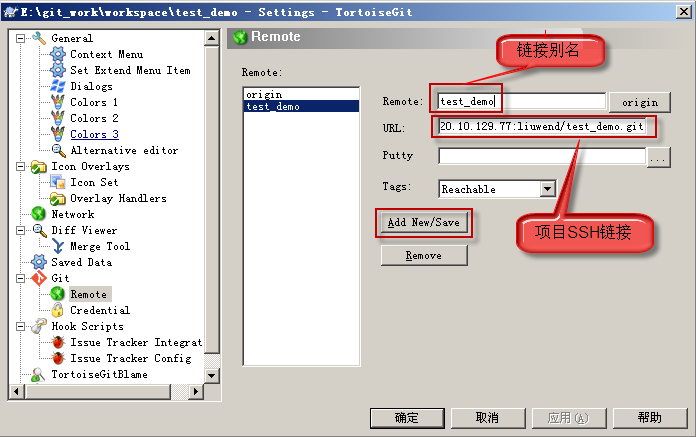


图6-1-8

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。

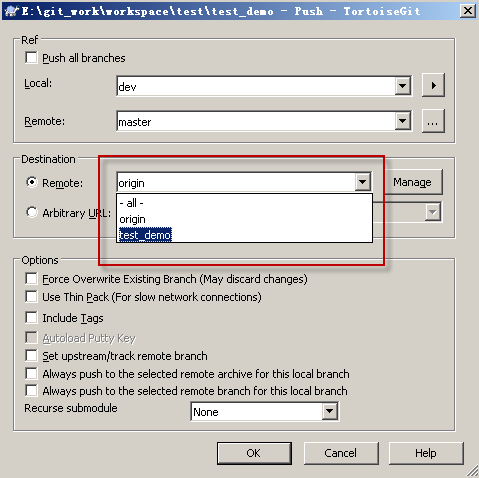


图6-1-9

推送完成后，git项目基于已有版本库的初始化完成。

### 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。

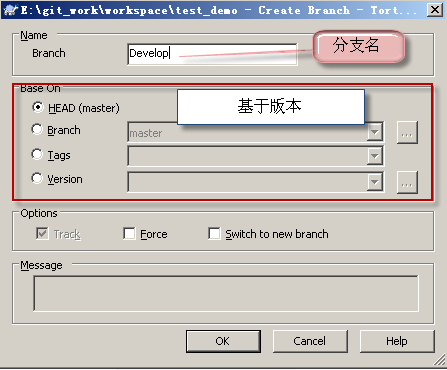


图6-1-10

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）。

## 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：

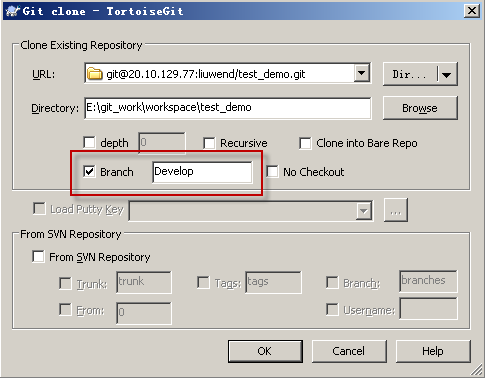


图6-2-1

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：

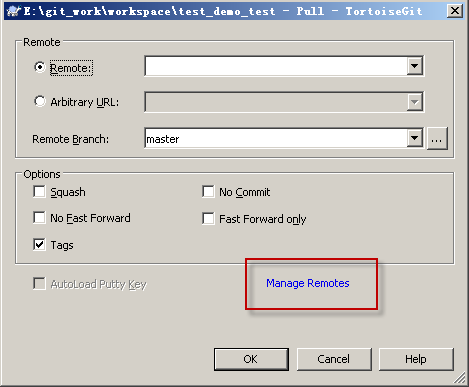


图6-2-2

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。

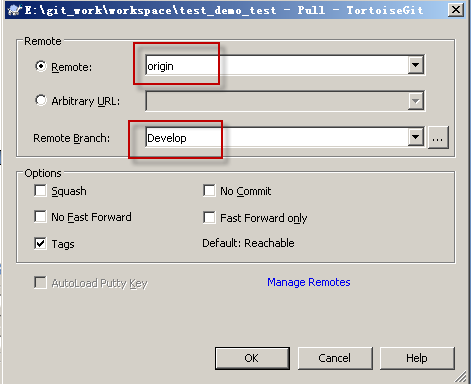


图6-2-3

（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：

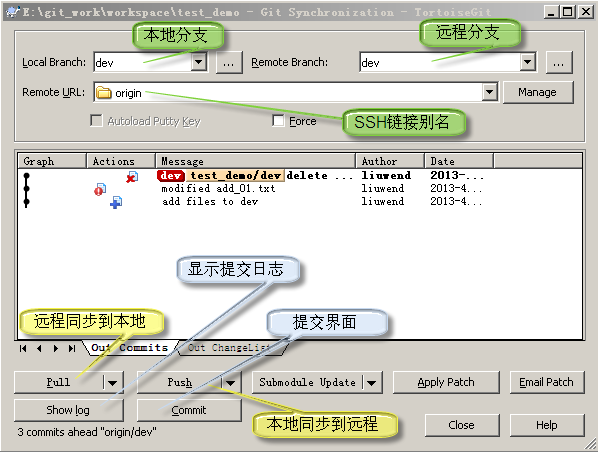


图6-2-4

**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。

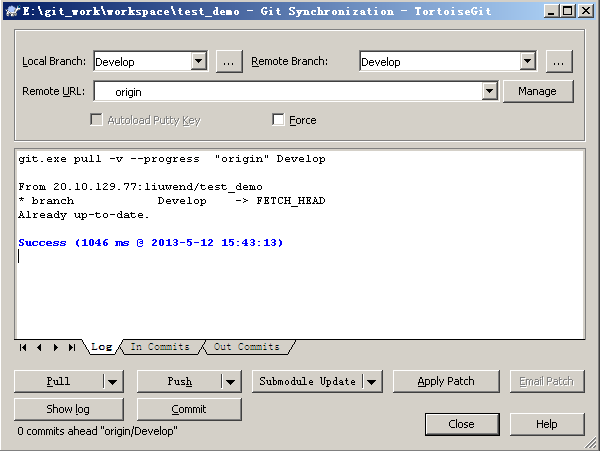
****

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。

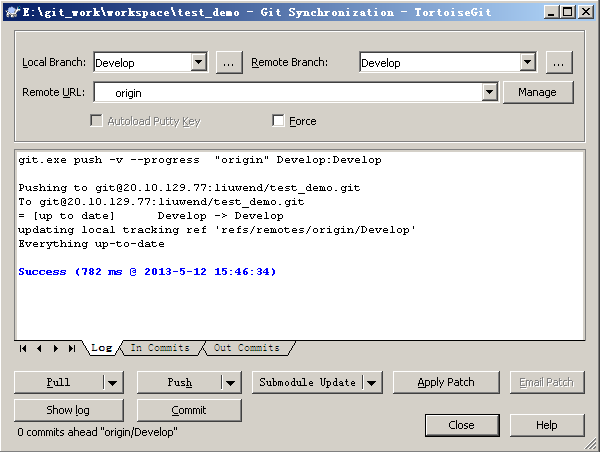


图6-2-6

Show log：显示以往的提交日志。

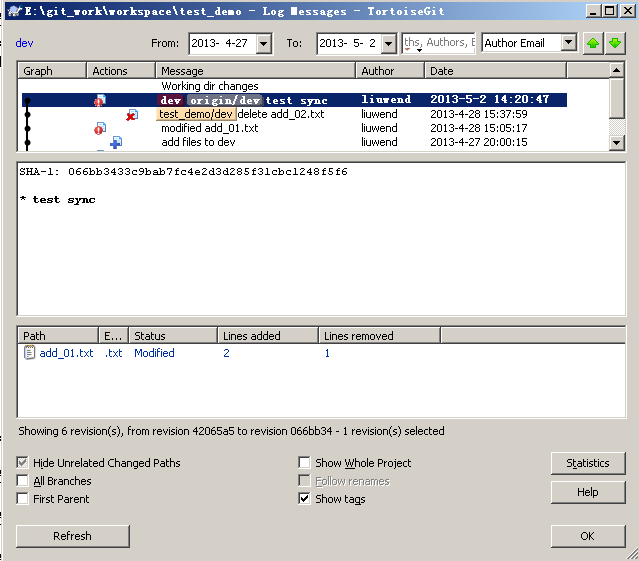


图6-2-7

Commit：打开本地分支提交界面。

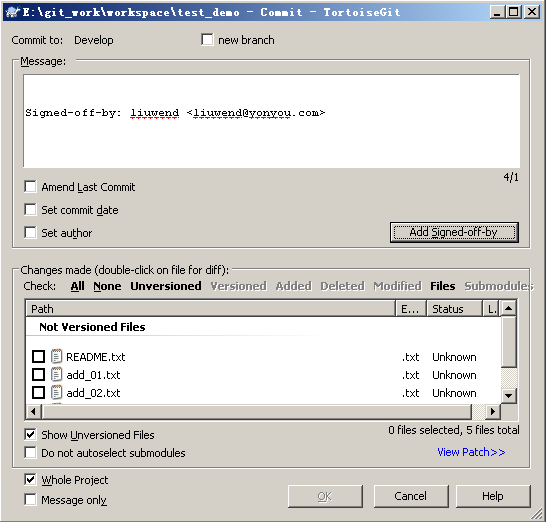


图6-2-8

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。

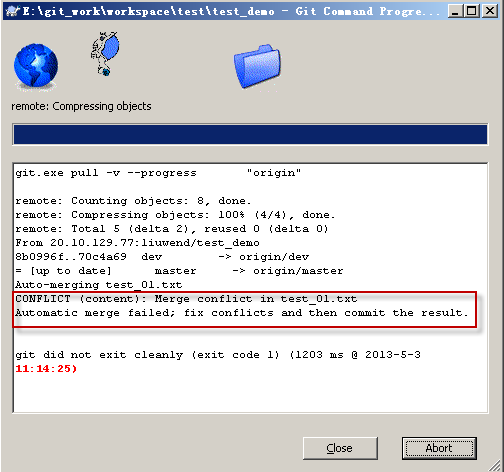


图6-2-9

在同步界面执行pull，可更清晰地显示冲突。

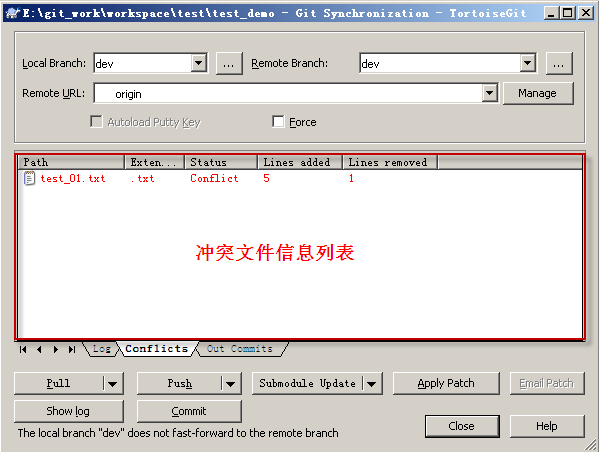


图6-2-10

双击文件，可查看冲突文件内容。

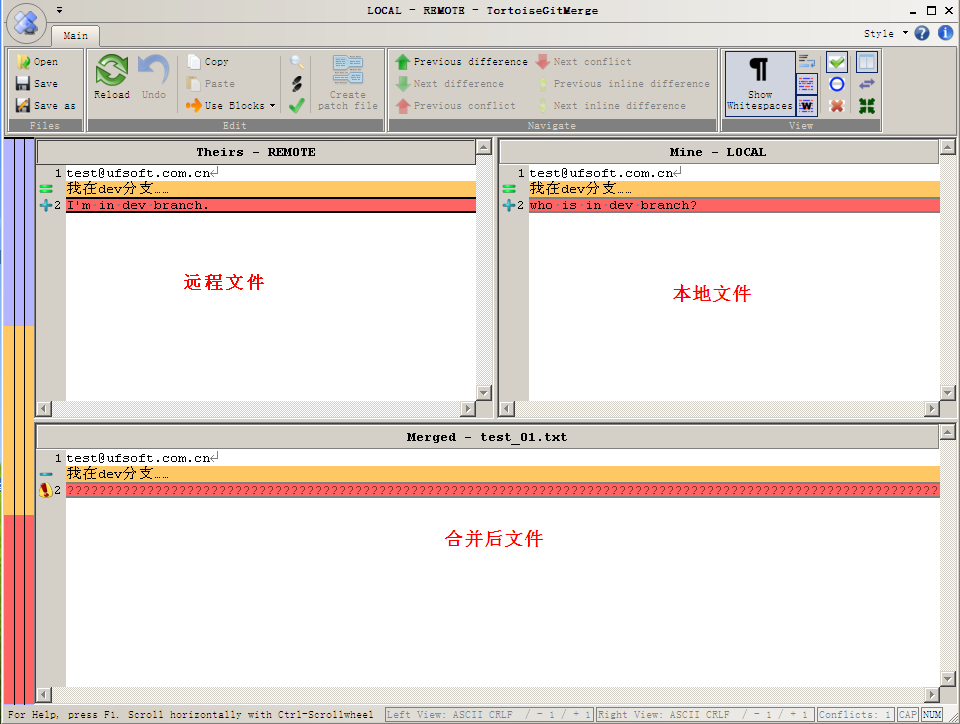


图6-2-11

在文本块右键，选择解决冲突的方式，然后保存文件。

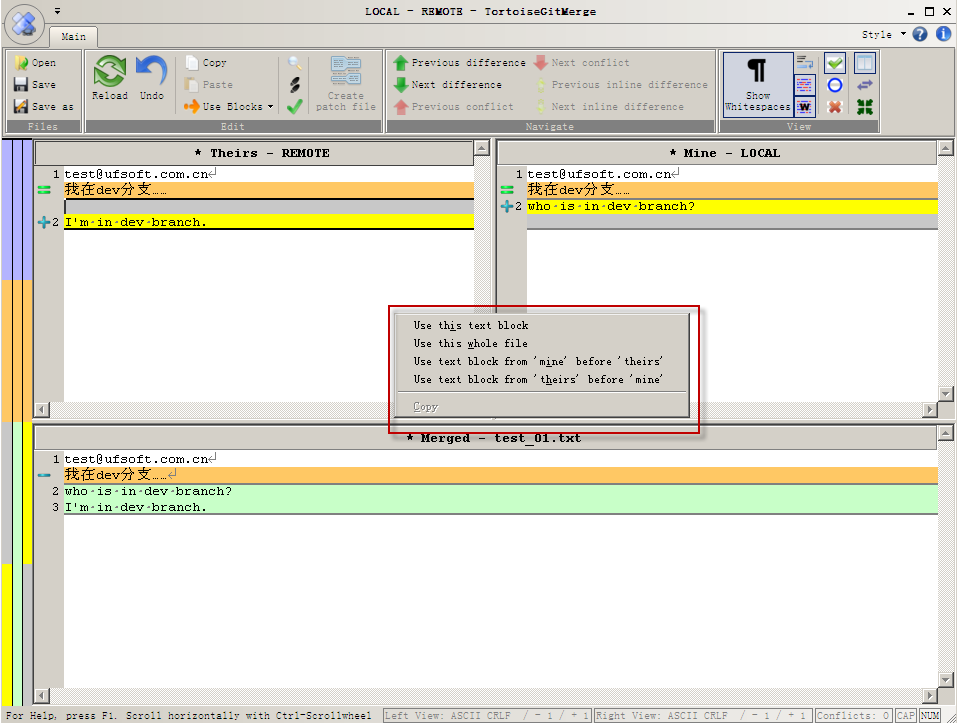


图6-2-12



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”）。

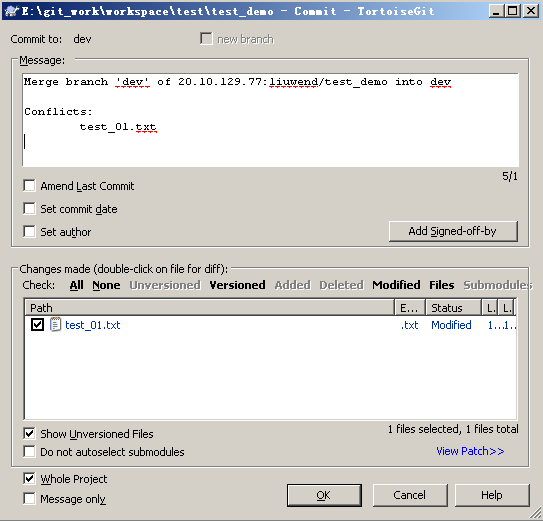


图6-2-14

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。

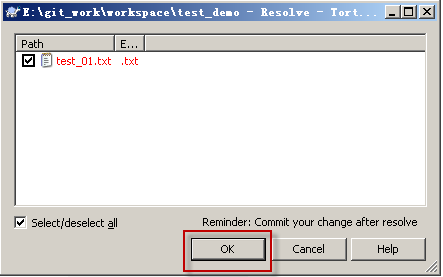


图6-2-15

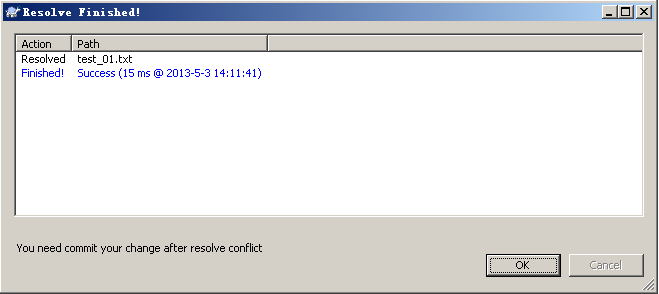


图6-2-16

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。

## 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。

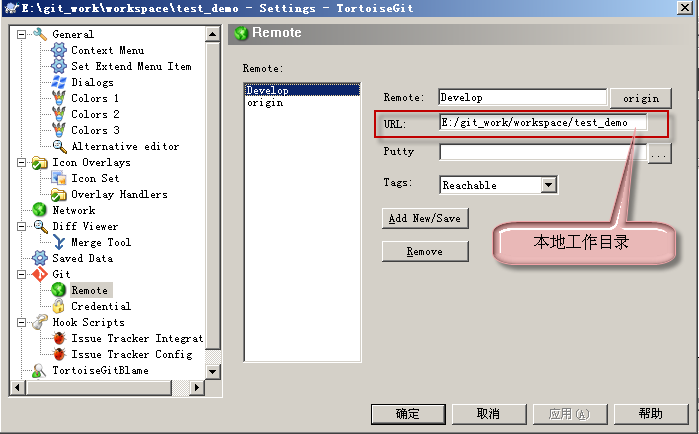


图6-3-1

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。

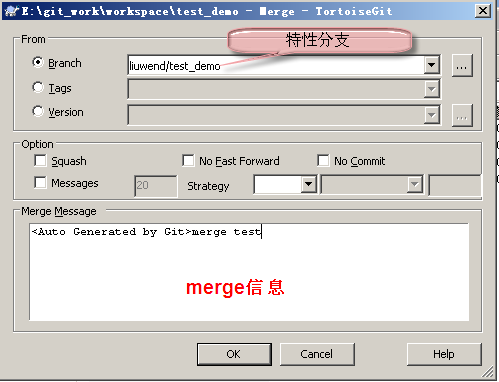


图6-3-2

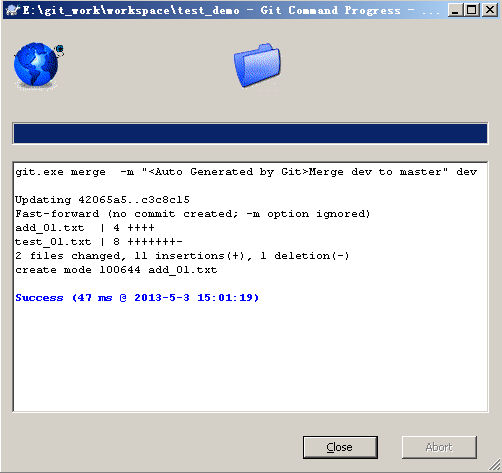


图6-3-3

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。

## 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。

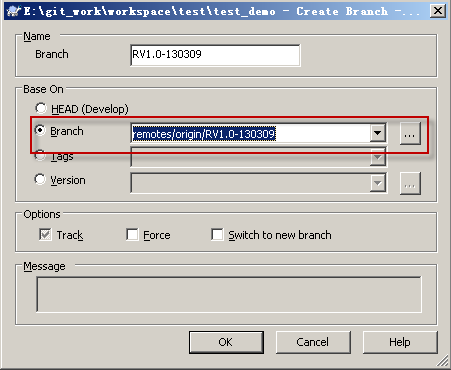


图6-4-1

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。

## 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。

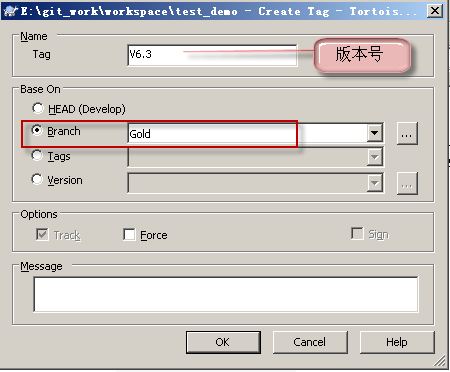


图6-5-1

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。

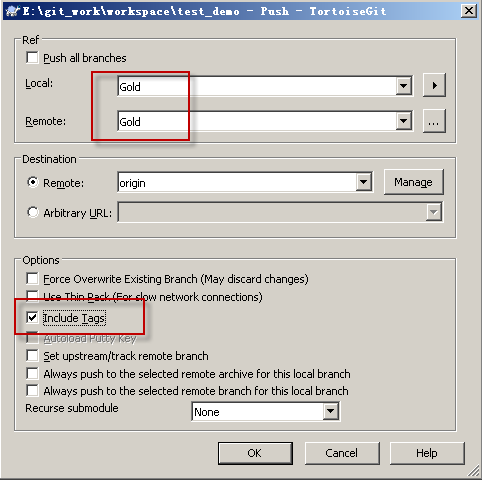


图6-5-2

## 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。

# 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。

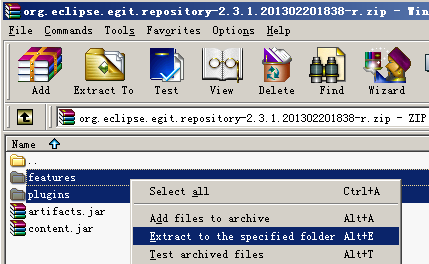


图7-2

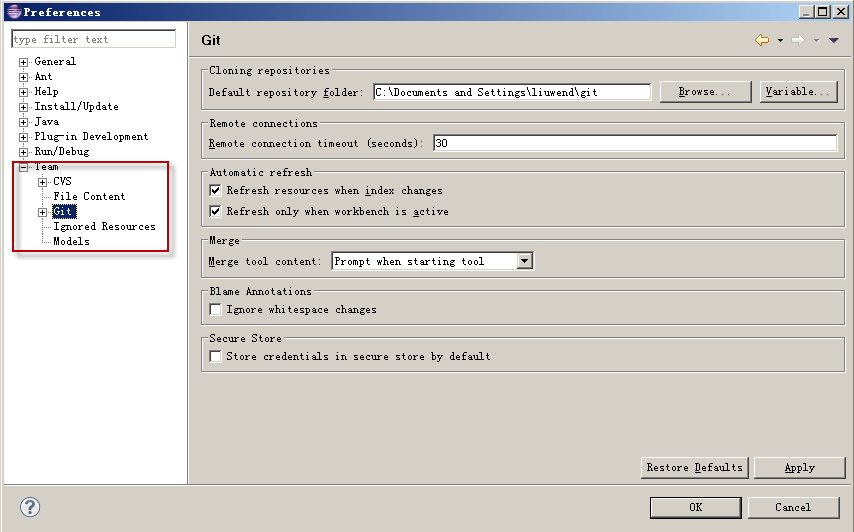


图7-3

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

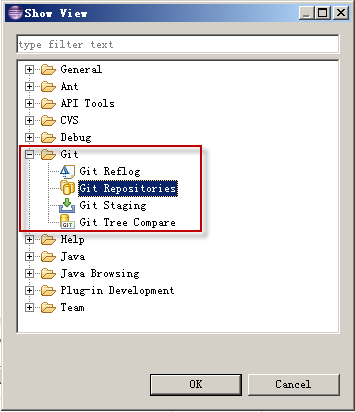


图7-4

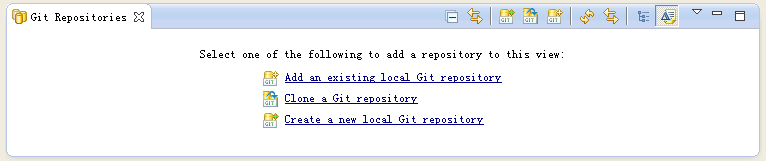


图7-5

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

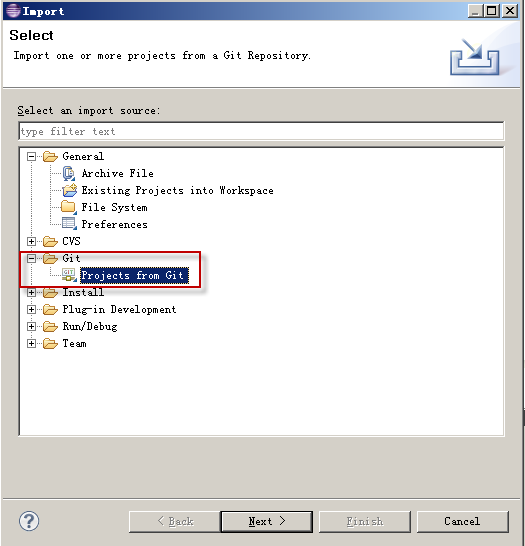


图7-6

选择local，Next；

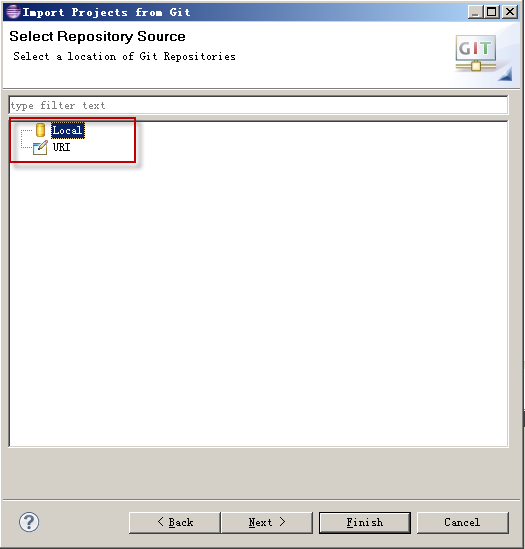


图7-7

Add…找到要引入的Git 仓库，Finish。

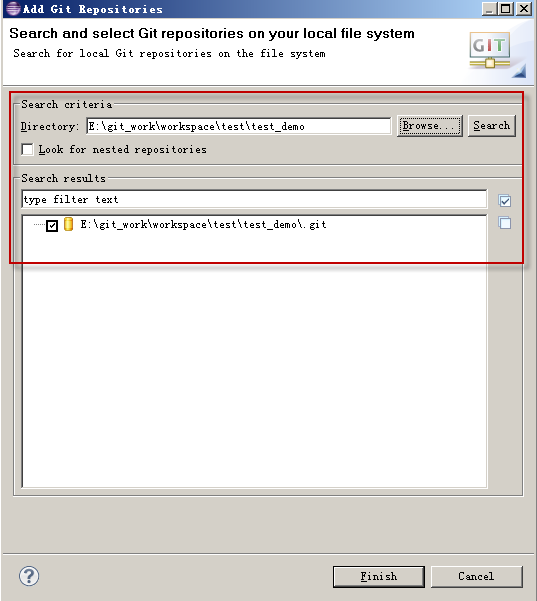


图7-8

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。

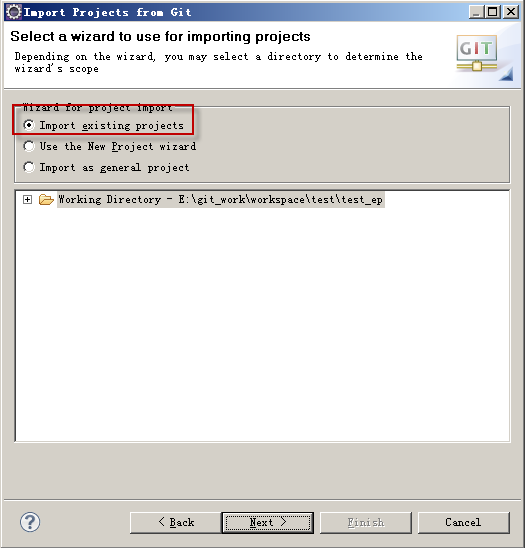


图7-9

在Package及Git窗口，可看到引入的Git仓库的具体信息。

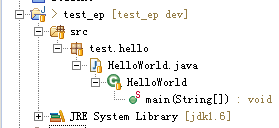
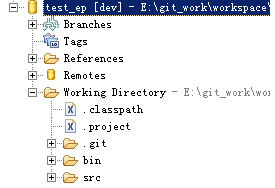
 

图7-10

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。

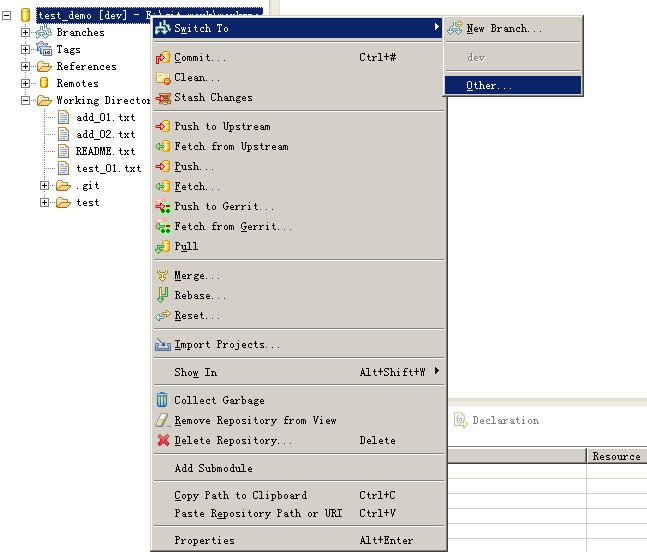


图7-11

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。

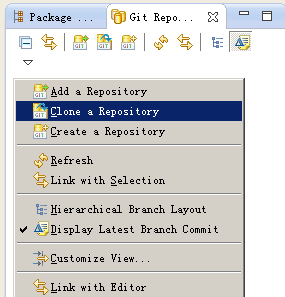


图7-12

输入SSH链接，Next。

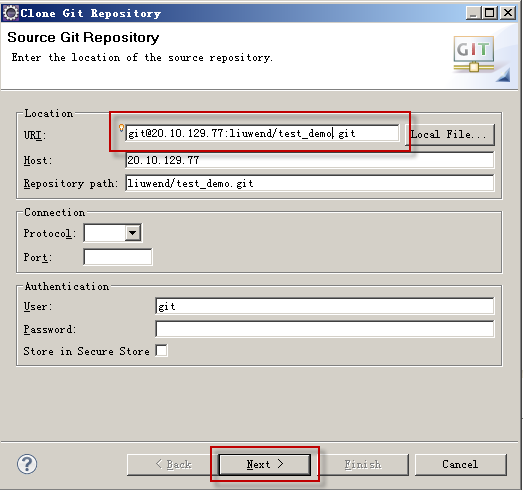


图7-13

选择要clone的分支，Next。

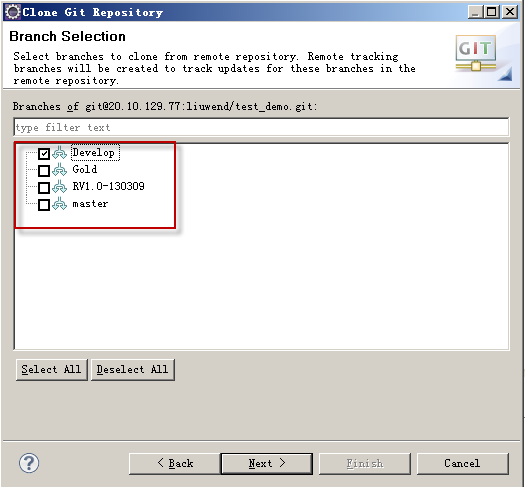


图7-14

选择本地目录，Finish。

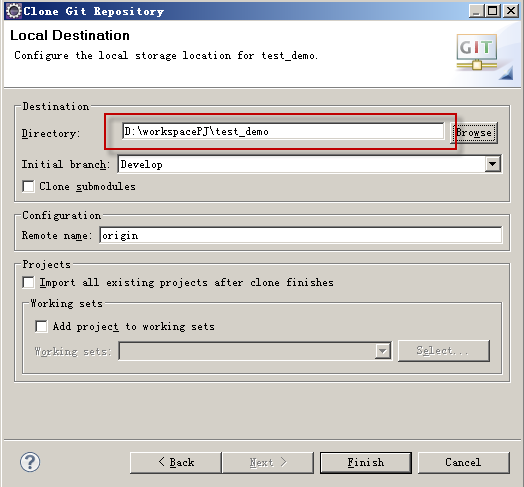


图7-15

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。

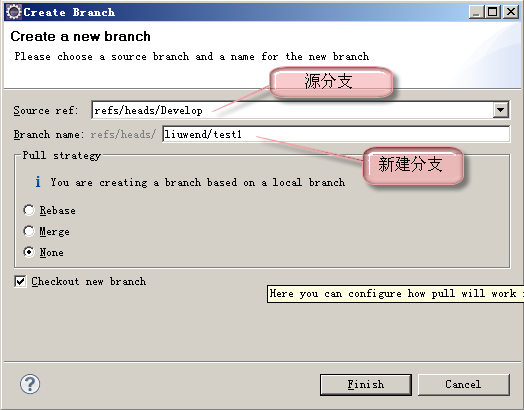


图7-16

Checkout操作：

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

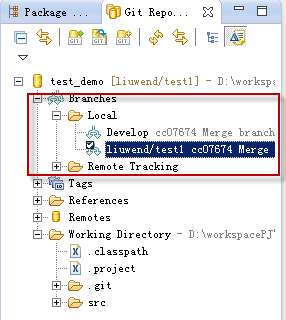


图7-17

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。

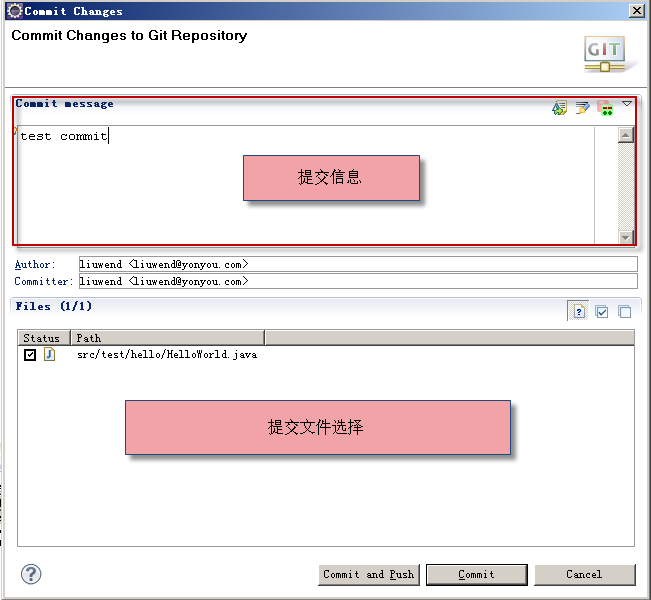


图7-18

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）

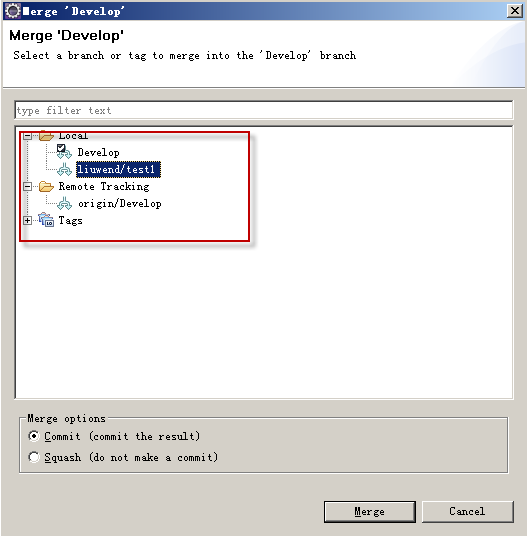


图7-19

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。

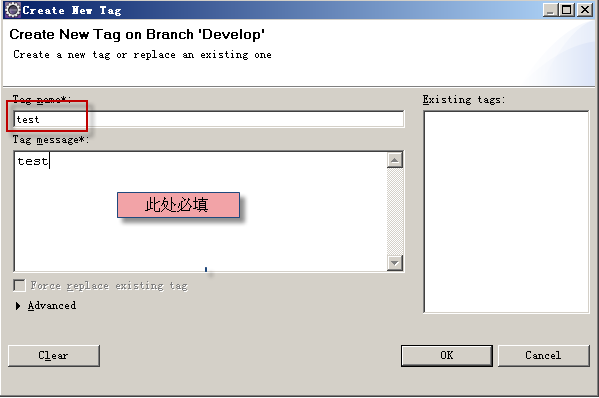


图7-20

Pull操作：

在Git仓库处右键-Pull，执行完结果如下：

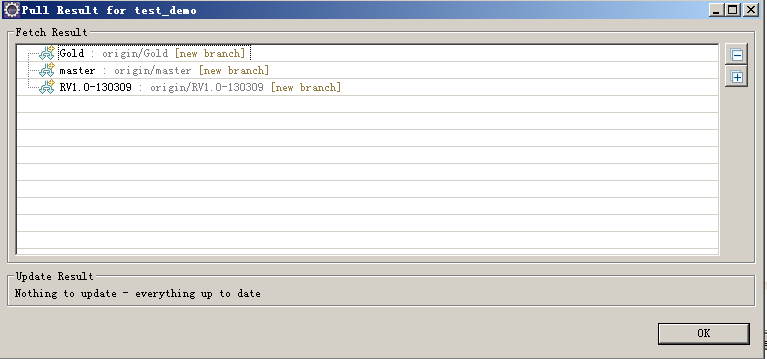


图7-21

Push操作：

在Git仓库处右键-Push，打开引导界面，选择远程仓库，Next。

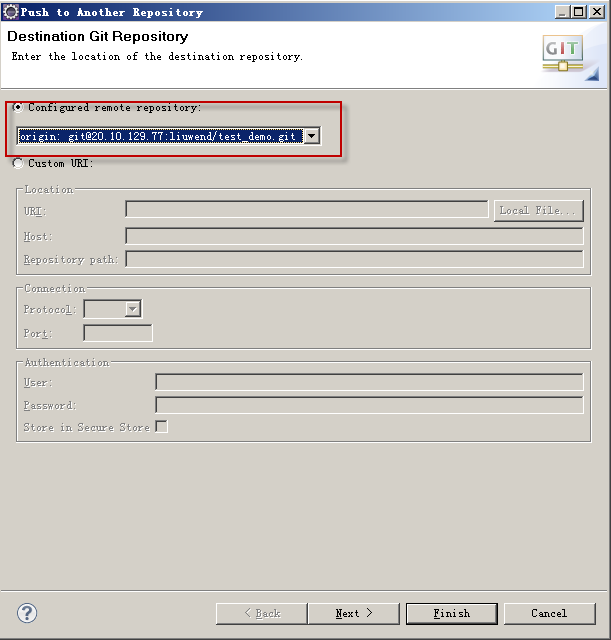


图7-22

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。

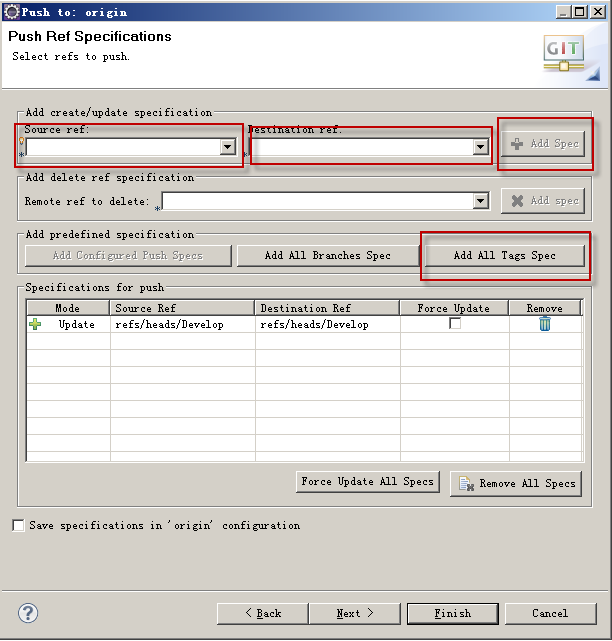


图7-23

结果如下：

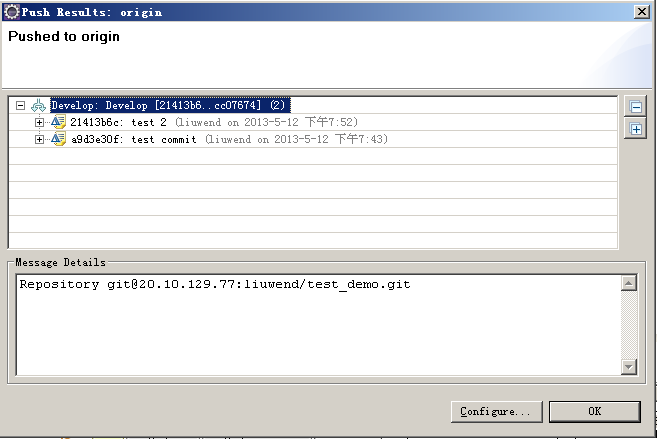


图7-24

查看提交日志：

在Git仓库处右键-Show In-History。

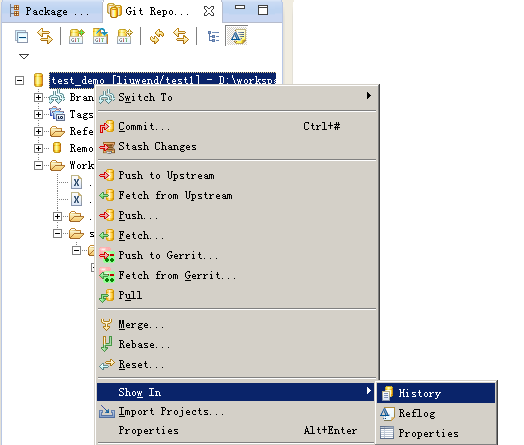


图7-25

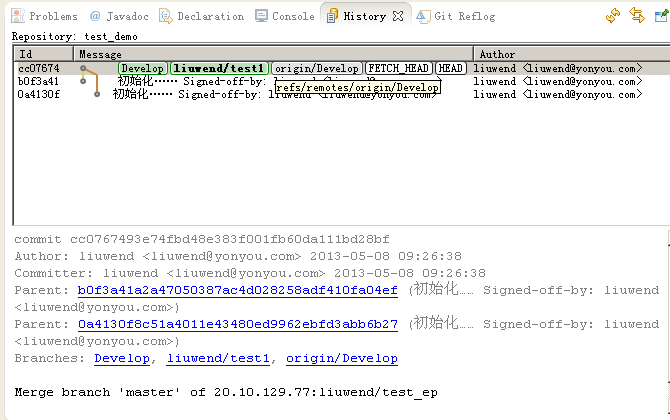


图7-26

（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）