**Git**

# Git的历史

同生活中的许多伟大事件一样，Git 诞生于一个极富纷争大举创新的年代。Linux 内核开源项目有着为数众广的参与者。绝大多数的 Linux 内核维护工作都花在了提交补丁和保存归档的繁琐事务上（1991－2002年间）。到 2002 年，整个项目组开始启用分布式版本控制系统 BitKeeper 来管理和维护代码。

到 2005 年的时候，开发 BitKeeper 的商业公司同 Linux 内核开源社区的合作关系结束，他们收回了免费使用 BitKeeper 的权力。这就迫使 Linux 开源社区（特别是 Linux的缔造者 Linus Torvalds ）不得不吸取教训，只有开发一套属于自己的版本控制系统才不至于重蹈覆辙。他们对新的系统订了若干目标：

•速度

•简单的设计

•对非线性开发模式的强力支持（允许上千个并行开发的分支）

•完全分布式

•有能力高效管理类似 Linux 内核一样的超大规模项目（速度和数据量）

下载地址：

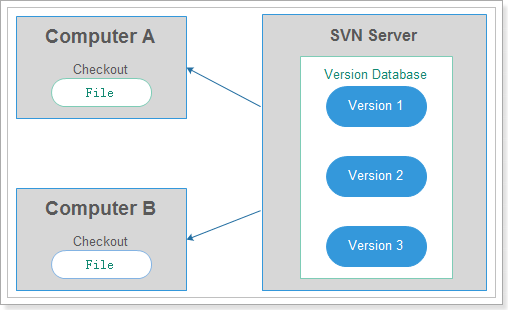
https://www.kernel.org/pub/software/scm/git/

# Git与Svn对比

## Svn原理分析

svn属于集中式版本管理控制系统，系统中保存了所有文件的修订版本，而协同工作人员通过连接svn服务器，提取出最新的文件，获取提交更新。

下图就是标准的集中式版本控制工具管理方式：



集中管理方式在一定程度上看到其他开发人员在干什么，而管理员也可以很轻松掌握每个人的开发权限。

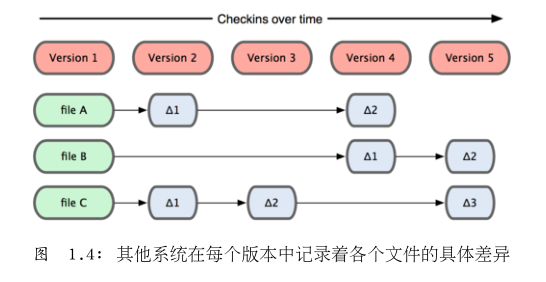
但是相较于其优点而言，集中式版本控制工具缺点很明显：

1. 服务器单点故障
2. 容错性差

## Git原理分析

### Svn记录具体差异

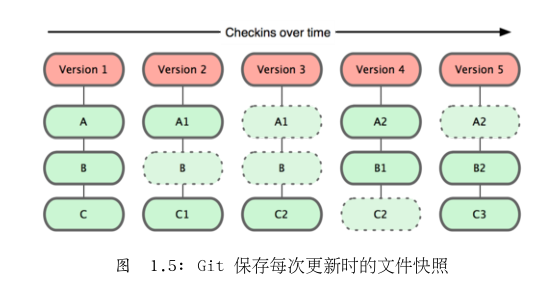
Git 和其他版本控制系统的主要差别在于，Git 只关心文件数据的整体是否发生变化，而大多数其他系统则只关心文件内容的具体差异。这类系统（CVS，Subversion，Perforce，Bazaar等等）每次记录有哪些文件作了更新，以及都更新了哪些行的什么内容，请看图 1.4。



### Git记录整体变化

Git 并不保存这些前后变化的差异数据。实际上，Git 更像是把变化的文件作快照后，记录在一个微型的文件系统中。每次提交更新时，它会纵览一遍所有文件的指纹信息并对文件作一快照，然后保存一个指向这次快照的索引。

为提高性能，若文件没有变化，Git 不会再次保存，而只对上次保存的快照作一连接。Git 的工作方式就像图 1.5 所示。



这是 Git 同其他系统的重要区别。它完全颠覆了传统版本控制的套路，并对各个环节的

实现方式作了新的设计。Git 更像是个小型的文件系统，但它同时还提供了许多以此为基础

的超强工具，而不只是一个简单的 VCS。稍后在第三章讨论 Git 分支管理的时候，我们会

再看看这样的设计究竟会带来哪些好处。

### 操作本地执行

Git 不用跑到外面的服务器上去取数据回来，而直接从本地数据库读取后展示给你看。所以任何时候你都可以马上翻阅，无需等待。

如果想要看当前版本的文件和一个月前的版本之间有何差异，Git 会取出一个月前的快照和当前文件作一次差异运算，而不用请求远程服务器来做这件事，或是把老版本的文件拉到本地来作比较。

### 数据完整性

在保存到 Git 之前，所有数据都要进行内容的校验和（checksum）计算，并将此结果作为数据的唯一标识和索引。换句话说，不可能在你修改了文件或目录之后，Git 一无所知。

这项特性作为 Git 的设计哲学，建在整体架构的最底层。所以如果文件在传输时变得不完整，或者磁盘损坏导致文件数据缺失，Git 都能立即察觉。

Git 使用 SHA-1 算法计算数据的校验和，通过对文件的内容或目录的结构计算出一个SHA-1 哈希值，作为指纹字符串。该字串由 40 个十六进制字符（0-9 及 a-f）组成，看起来就像是：

*24b9da6552252987aa493b52f8696cd6d3b00373*

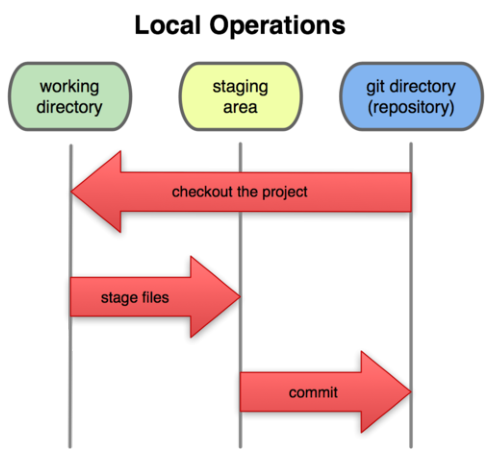
Git 的工作完全依赖于这类指纹字串，所以你会经常看到这样的哈希值。实际上，所有保存在 Git 数据库中的东西都是用此哈希值来作索引的，而不是靠文件名。

### 三种状态

现在请注意，接下来要讲的概念非常重要。

对于任何一个文件，在 Git 内都只有三种状态：已提交（committed），已修改（modified）和已暂存（staged）。已提交表示该文件已经被安全地保存在本地数据库中了；已修改表示修改了某个文件，但还没有提交保存；已暂存表示把已修改的文件放在下次提交时要保存的清单中。

由此我们看到 Git 管理项目时，文件流转的三个工作区域：Git 的本地数据目录，工作目录以及暂存区域。



1. 如果是 git 目录中保存着的特定版本文件，就属于已提交状态；
2. 如果作了修改并已放入暂存区域，就属于已暂存状态；
3. 如果自上次取出后，作了修改但还没有放到暂存区域，就是已修改状态。

# Git服务器搭建

Git服务器可以搭建在windows,也可以搭建在Linux中，由于windows中copssh登录git服务器软件已经收费，免费版本有很多缺点。因此把git服务器搭建在Linux中。

## 依赖环境

Git需要很多依赖环境：因此安装git需要先安装下列软件：

[root@itcast-01 ~]# yum install curl-devel expat-devel gettext-devel openssl-devel zlib-devel perl-devel gcc-c++

## 安装git

上传git安装包：

安装包：git-2.9.3.tar.gz

tar -xvf git-2.9.3.tar.gz

安装命令：(进入git解压目录)

[root@itcast-01 git-2.9.3]# make prefix=/usr/local all

[root@itcast-01 git-2.9.3]# make prefix=/usr/local install

## 创建git用户

添加Linux的用户，用户名为git的用户：

[root@itcast-01 ~]# useradd git

设置git用户的密码

[root@itcast-01 ~]# passwd git

切换到git用户

[root@itcast-01 ~]# su - git

## 初始化仓库

初始化仓库，仓库名为taotao.git：

[git@itcast-01 root]$ cd

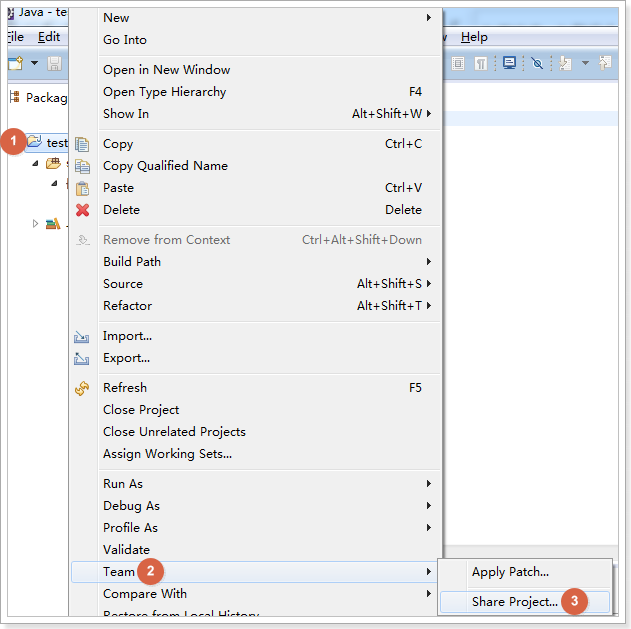
[git@itcast-01 ~]$ git init --bare taotao.git

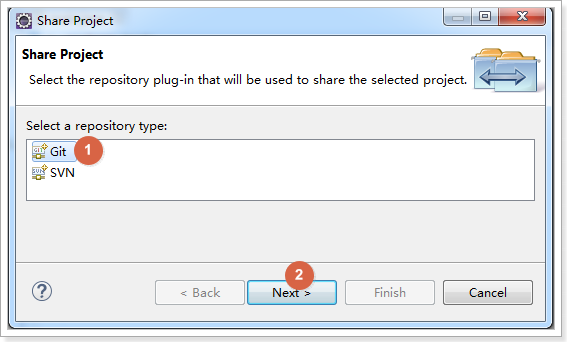
始化空的 Git 仓库于 /home/git/taotao.git/

# Eclipse的Git使用

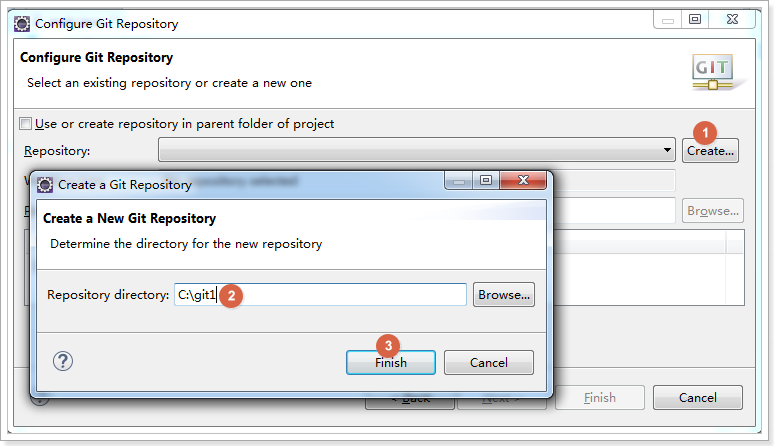
## 检入代码

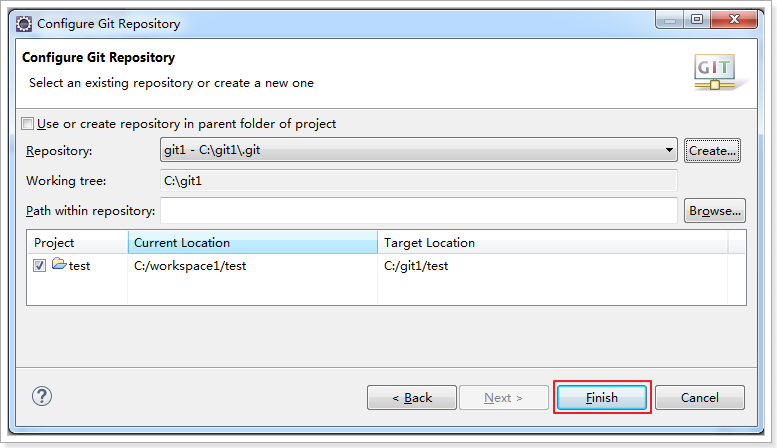
### 设置使用Git



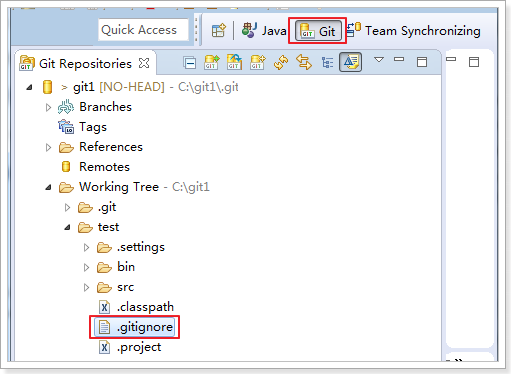


### 设置git文件存放路径

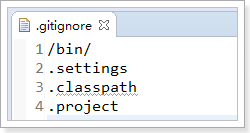




### 设置忽略的文件

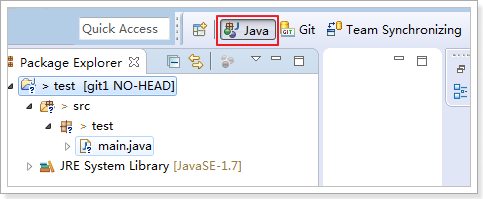


如下图，加入检入时需要忽略的文件

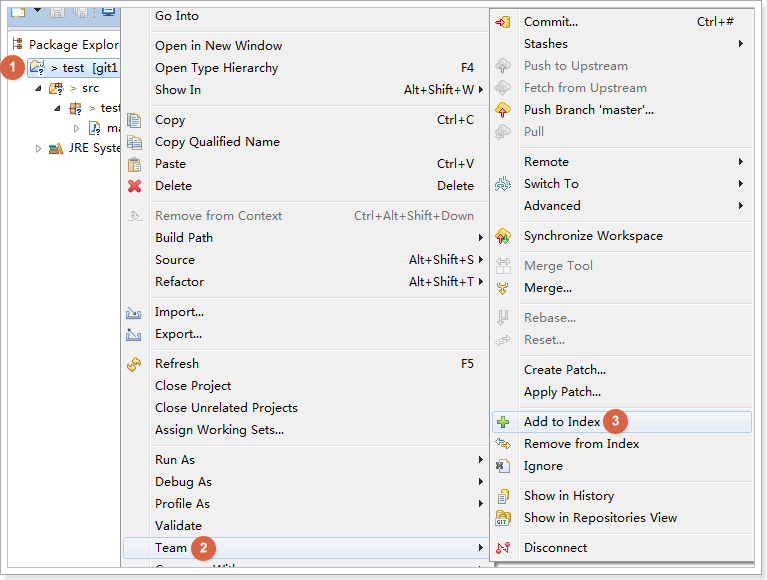


### 提交代码到本地(暂存区)

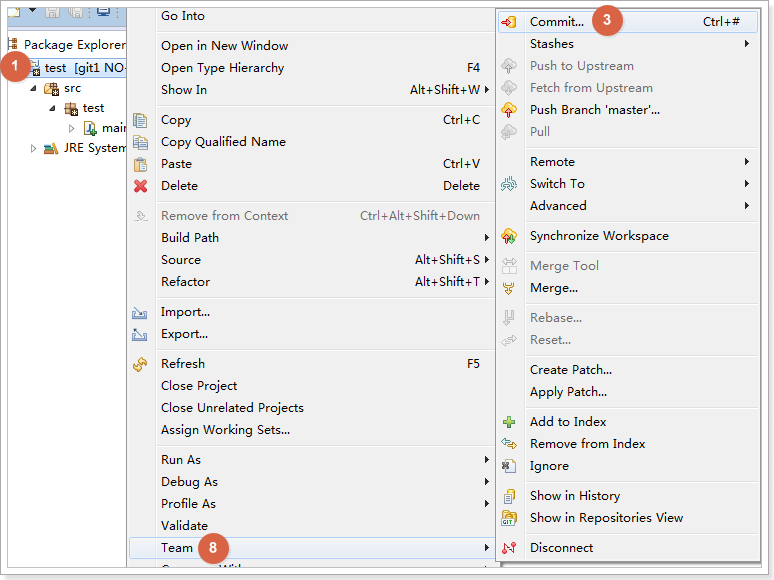
回到java视图

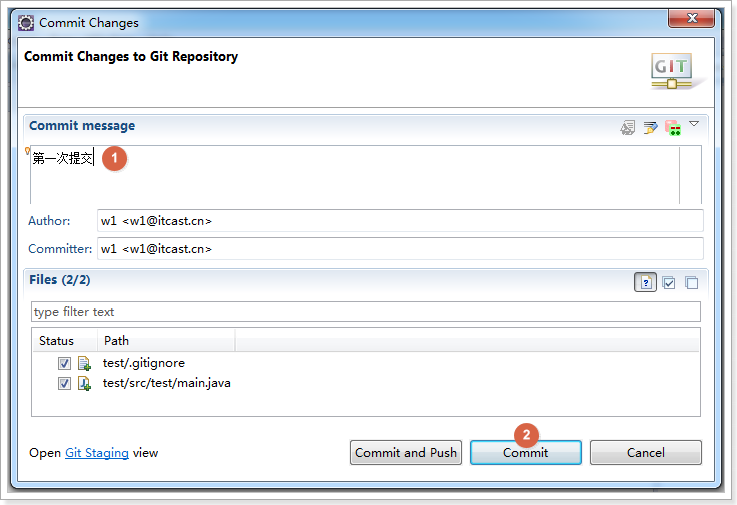


添加代码到git中

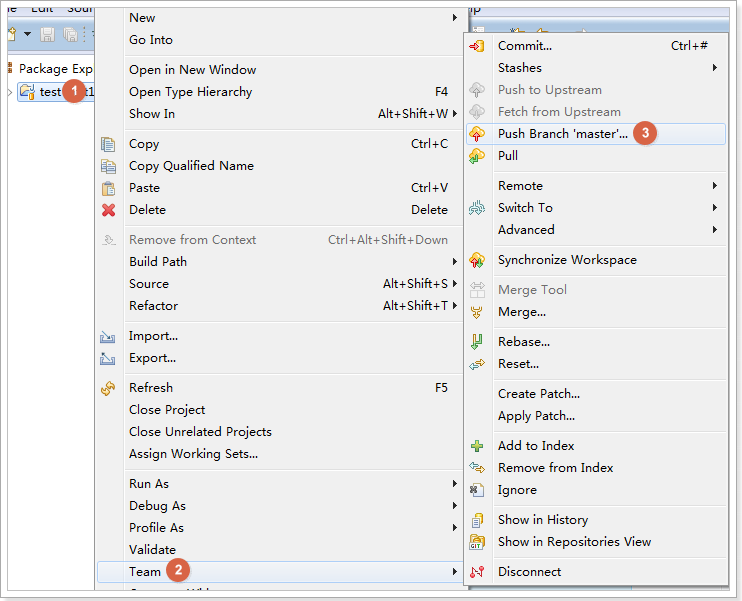


提交代码到本地

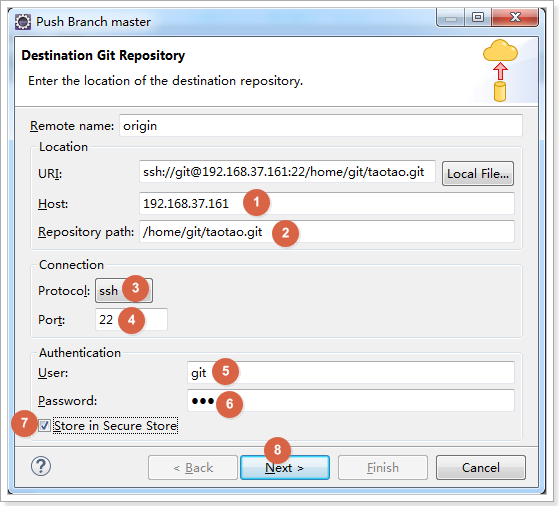


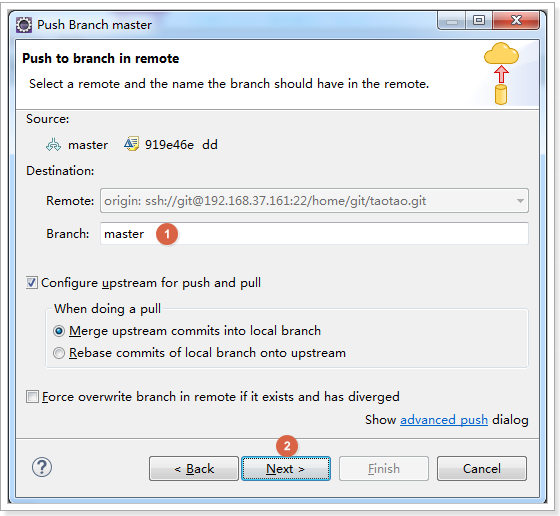


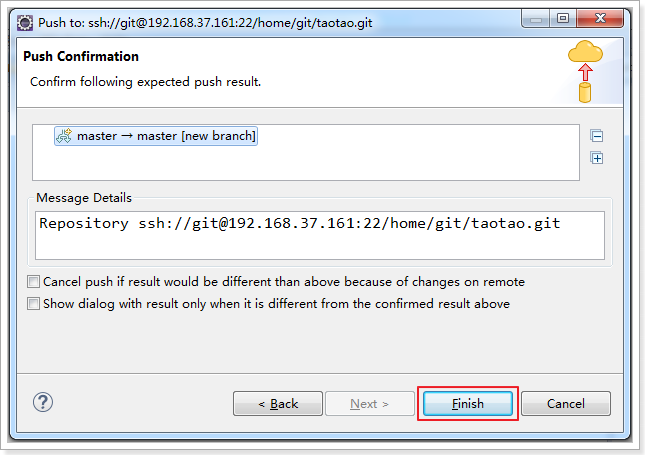
### 提交代码到Git服务器



设置git服务器地址：

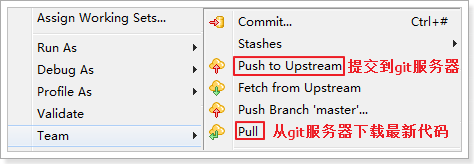






### 检入检出git服务器操作

当第一次检入到git服务器后，后续的代码检入检出git服务器的操作如下图即可。

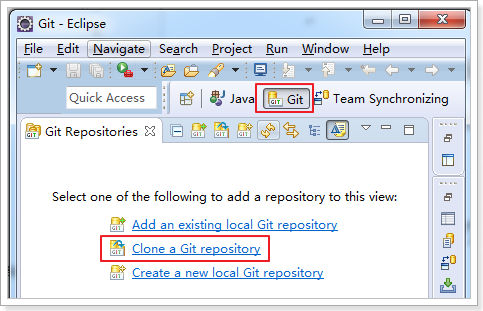


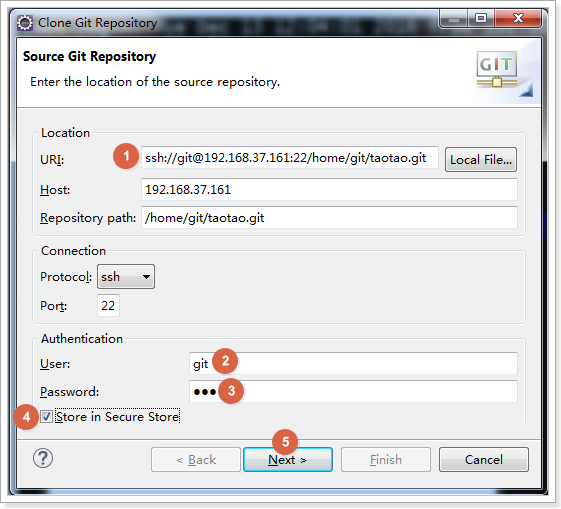
注意：Commit提交的作用是把代码提交到本地。

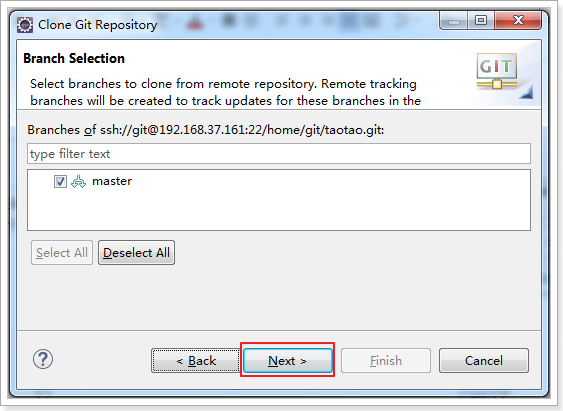
Push的作用是把本地的代码提交到远程的git服务器。

## 检出代码

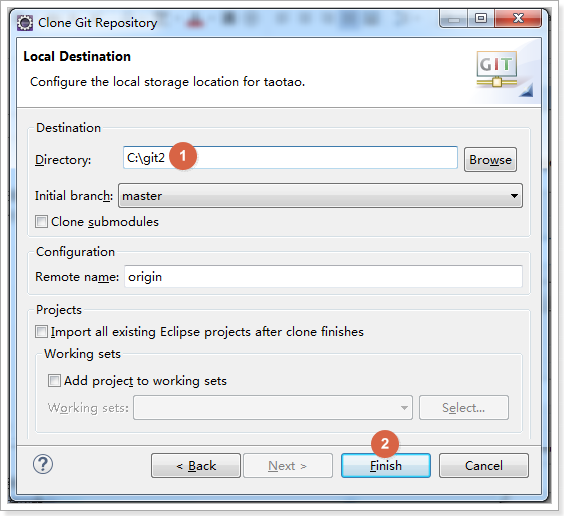
### 克隆git仓库到本地



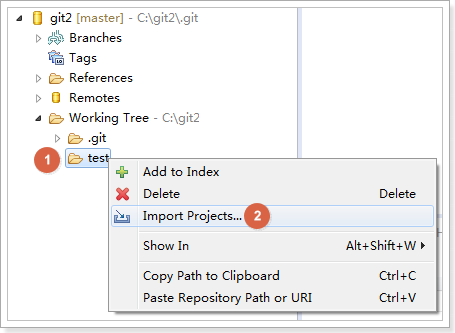


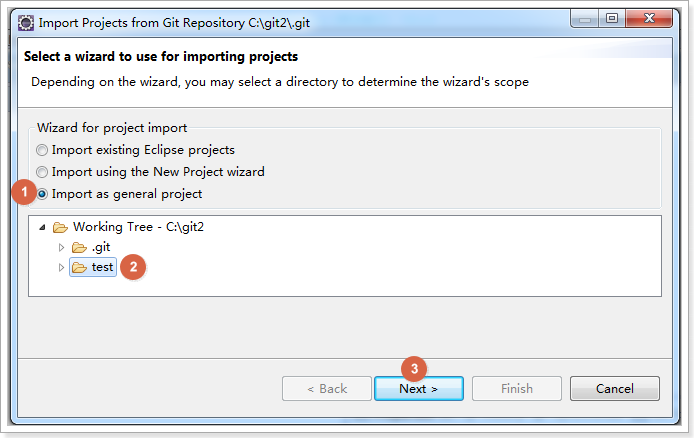


### 设置git文件存放路径

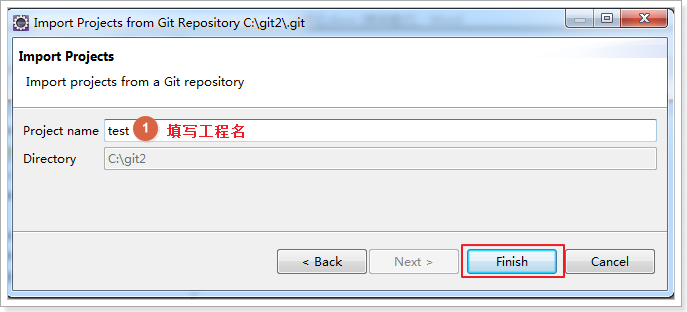


### 导出工程

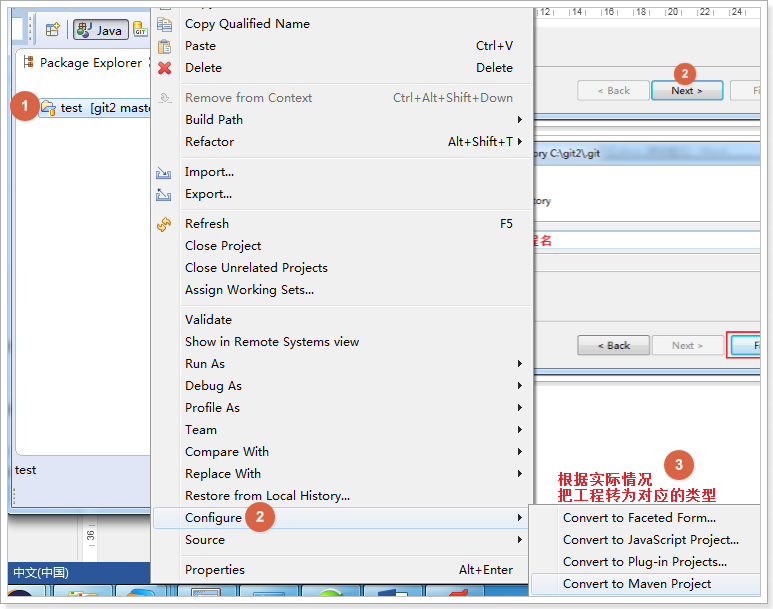




填写工程名

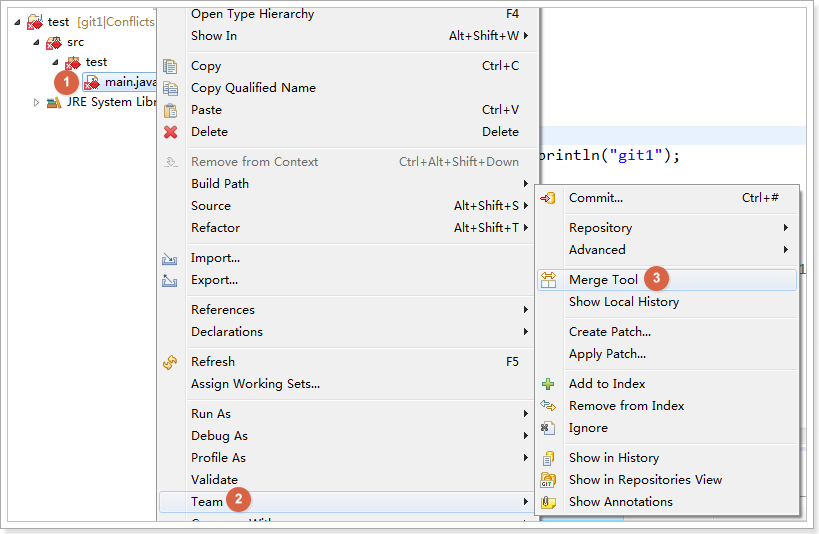


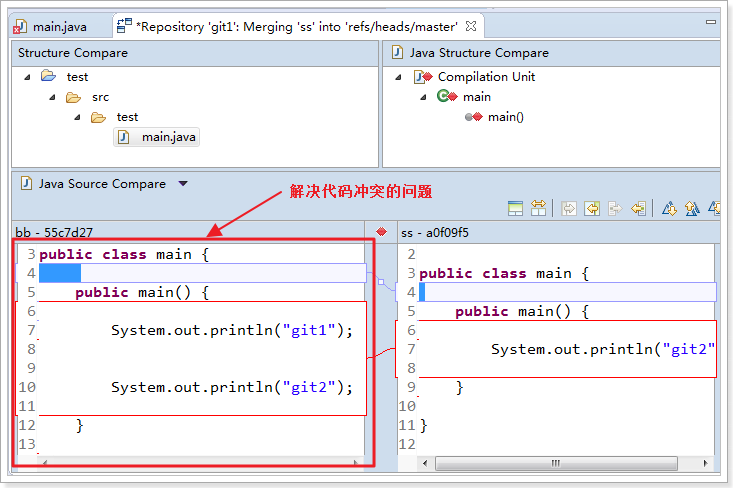
转换工程为对应的类型



## 解决冲突

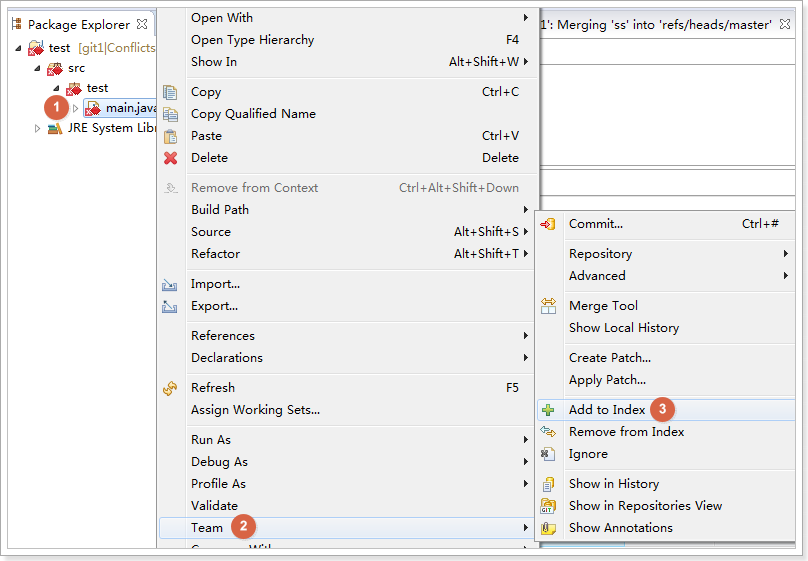
### 解决代码冲突





### 提交解决冲突后的代码

把解决冲突后的代码执行Add to Index



先Commit到本地

再Push到远程的git服务器

