研发过程管理

版本管理工具 Git 操作培训

中国联通软件研究院

2020.04

课程简介

版本控制系统是保存文件多个版本的一种机制，可以有效管理软件开发项目中发生的一切变更，因此版本控制工具是作为软件开发项目管理的基础工具。

Git 是世界上最先进的分布式版本控制系统，《Git 操作培训课程》主要讲解版本控制工具 Git 的操作和天梯版本控制操作流程以及分支策略设计方案。通过本课程的学习，可以使学员掌握版本控制系统的概念，能够使用 Git、Gitflow 和天梯进行项目的版本控制。

目录

第 1 章 Git 简介

1.1 关于版本控制

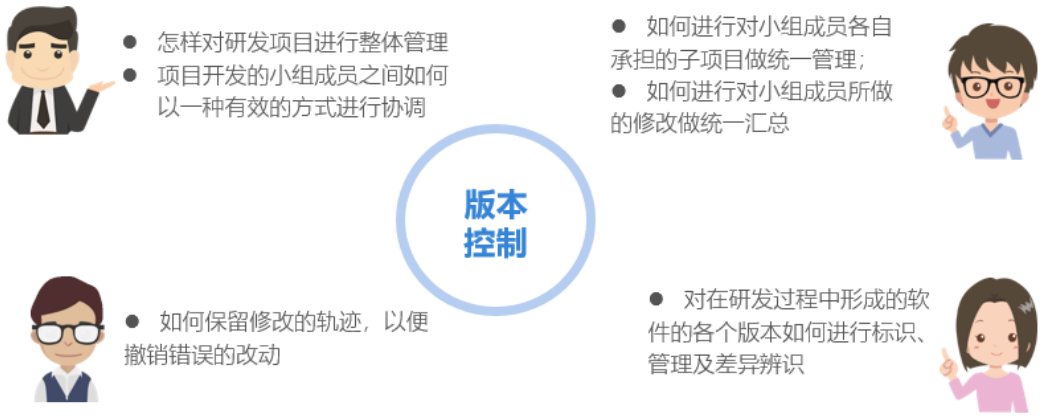
版本控制系统是保存文件多个版本的一种机制，配置管理往往作为版本控制的同义词。配置管理策略将决定如何管理项目中发生的一切变化。因此，它记录了系统以及应用程序的演进过程。另外，它也是对团队成员协作方式的管理。使用版本控制系统，当修改某个文件后，你仍旧可以访问该文件之前的任意一个修订版本。它也是我们共同合作交付软件时所使用的一种机制。

第一个流行的版本控制系统是一个 UNIX 下的专有工具，称为 SCCS（Source Code Control System，源代码控制系统），可以追溯到 20 世纪 70 年代。它被 RCS（Revision Control System，修订控制系统）和后来的 CVS（Concurrent Versions System，并发版本控制系统）所取代。虽然这三种系统的市场份额越来越小，但至今仍旧有人在使用。现在市面上有很多更好用的版本控制系统，既有开源的，也有商业版的，而且都是针对各种不同的应用环境设计的。一般来说，包括 Subversion、Mercurial 和 Git 在内的开源工具就可以满足绝大多数团队的需求。

本质上来讲，版本控制系统的目的有两个。首先，它要保留每个文件的所有版本的历史信息，并使之易于查找。这种系统还提供一种基于元数据（这些元数据用于描述数据的存储信息）的访问方式，使元数据与某个单个文件或文件集合相链接。其次，它让分布式团队（无论是空间上不在一起，还是不同的时区）可以愉快地协作。

为什么需要进行有效的版本控制呢？最关键的是它能回答下面这些问题：

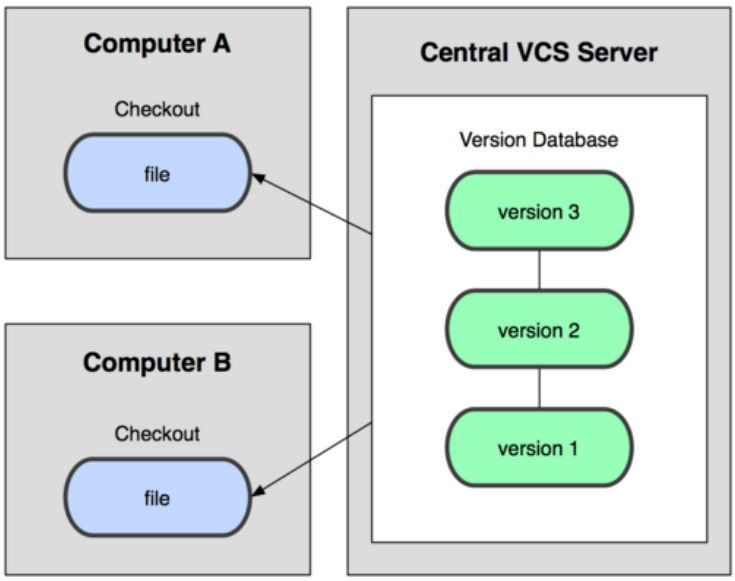
1) 对于我们开发的应用软件，某个特定的版本是由哪些文件和配置组成的？如何再现一份与生产环境一模一样的软硬件环境？
2) 什么时候修改了什么内容，是谁修改的，以及为什么要修改？因此，我们很容易知道应用软件在何时出了错，出错的过程，甚至出错的原因。



版本控制不仅仅针对源代码。每个与所开发的软件相关的产物都应被置于版本控制之下。开发人员不但要用它来管理和控制源代码，还要把测试代码、数据库脚本、构建和部署脚本、文档、库文件和应用软件所用的配置文件都纳入到版本控制之中，甚至把编译器以及工具集等也放在里面，以便让新加入项目的成员可以很容易地从零开始工作。

为了解决版本控制的这个问题，人们很久以前就开发了许多种不同类型的版本控制系统，包括本地版本控制系统、集中化的版本控制系统、分布式版本控制系统。

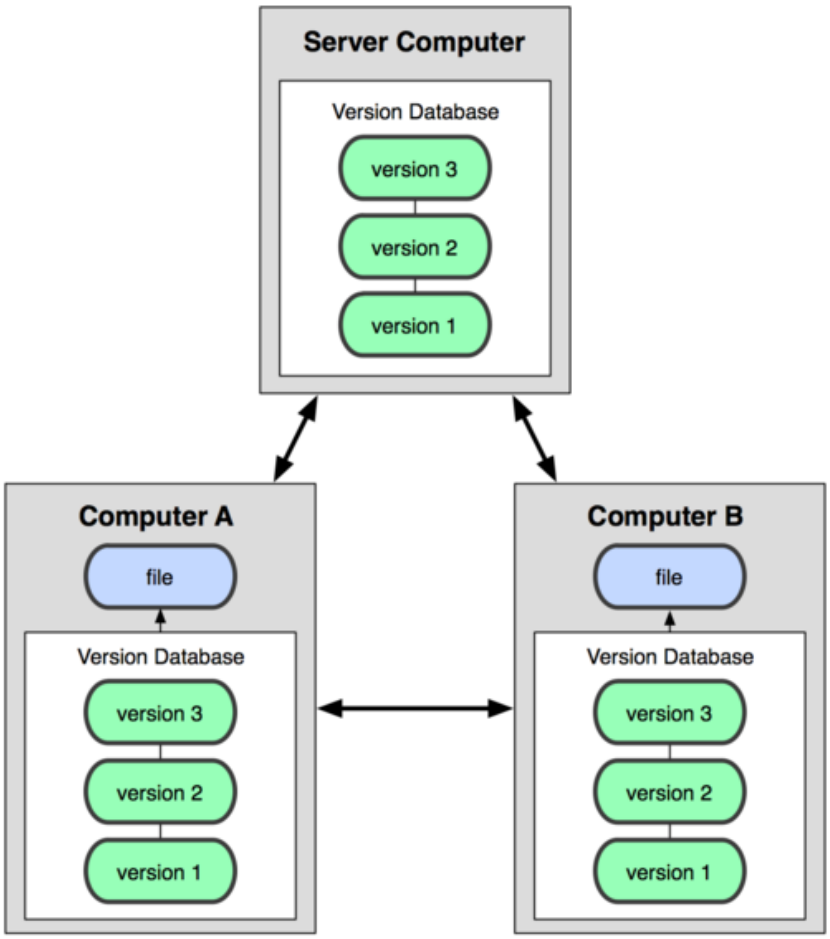
本地版本控制系统，大多都是采用某种简单的数据库来记录文件的历次更新差异，但是无法满足不同系统上的开发者协同工作的要求，于是，集中化的版本控制系统（Centralized Version Control Systems，简称 CVCS）应运而生。这类系统，诸如 CVS，Subversion 以及 Perforce 等，都有一个单一的集中管理的服务器，保存所有文件的修订版本，而协同工作的人们都通过客户端连到这台服务器，取出最新的文件或者提交更新。多年以来，这已成为版本控制系统的标准做法。如下图：



这种做法带来了许多好处，特别是相较于老式的本地 VCS 来说。现在，每个人都可以在一定程度上看到项目中的其他人正在做些什么。而管理员也可以轻松掌控每个开发者的权限，并且管理一个 CVCS 要远比在各个客户端上维护本地数据库来得轻松容易。

事分两面，有好有坏。这么做最显而易见的缺点是中央服务器的单点故障。如果宕机一小时，那么在这一小时内，谁都无法提交更新，也就无法协同工作。要是中央服务器的磁盘发生故障，碰巧没做备份，或者备份不够及时，就会有丢失数据的风险。最坏的情况是彻底丢失整个项目的所有历史更改记录，而被客户端偶然提取出来的保存在本地的某些快照数据就成了恢复数据的希望。但这样的话依然是个问题，你不能保证所有的数据都已经有人事先完整提取出来过。本地版本控制系统也存在类似问题，只要整个项目的历史记录被保存在单一位置，就有丢失所有历史更新记录的风险。

于是分布式版本控制系统（ Distributed Version Control System，简称 DVCS ）面世了。在这类系统中，像 Git，Mercurial，Bazaar 以及 Darcs 等，客户端并不只提取最新版本的文件快照，而是把代码仓库完整地镜像下来。这么一来，任何一处协同工作用的服务器发生故障，事后都可以用任何一个镜像出来的本地仓库恢复。因为每一次的提取操作，实际上都是一次对代码仓库的完整备份，如下图：



1.2 Git 是什么

1.2.1 Git 简史

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。

到了 2005 年，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。他们对新的系统制订了若干目标：

- 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）

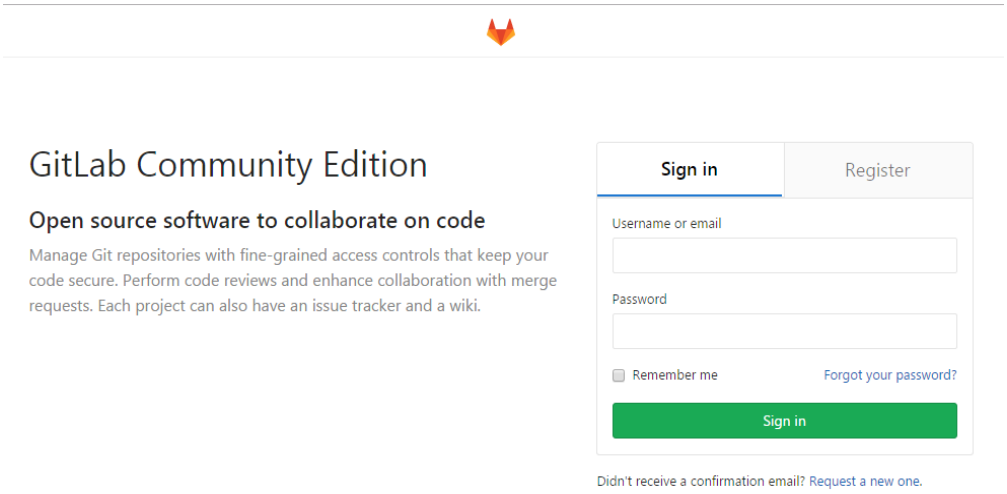
自诞生于 2005 年以来，Git 日臻成熟完善，在高度易用的同时，仍然保留着初期设定的目标。它的速度飞快，极其适合管理大项目，它还有着令人难以置信的非线性分支管理系统，可以应付各种复杂的项目开发需求。

- Git是一个开源的分布式版本控制系统
  - ✓ 由Linux之父 Linus Torvalds为了帮助管理 Linux 内核开发而开发
  - ✓ 用于敏捷高效地处理任何或小或大的项目
  - ✓ 目前世界上最先进的分布式版本控制系统 (没有之一)

1.2.2 Gitlab 简介

Gitlab 使用 Git 作为代码管理工具，并在此基础上搭建起来的 web 服务；实现了一个自托管的 Git 项目仓库，可以通过 WEB 界面访问公开的或者私人项目；具有类似

Github 的功能：浏览源代码、管理缺陷、注释、管理团队对仓库的访问、浏览提交过的版本。



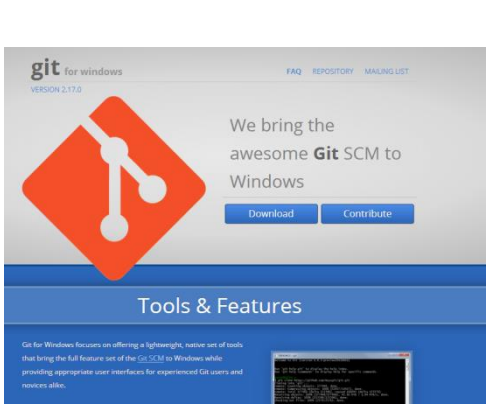
第 2 章 环境安装

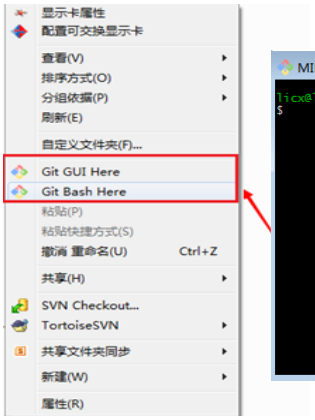
2.1 Git 安装

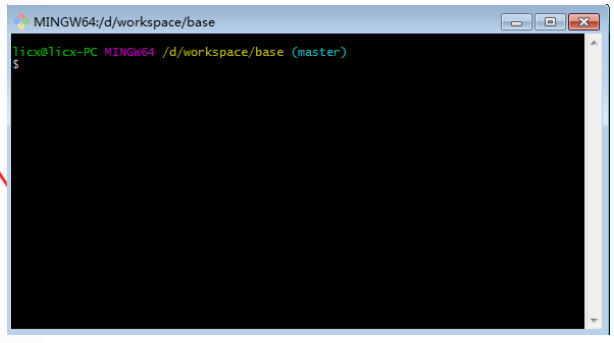
在 Windows 上安装 Git 很简单，下载安装包：https://git-for-windows.github.io/，根据本地环境选择 32 位或 64 为位的安装包，下载完成后，只需按默认选项安装即可。

验证是否安装成功：

- 进入一个目录，右击鼠标，出现“Git GUI here”和“Git Bash here”
- 点击“Git Bash here”后，若出现第三幅图，说明安装成功



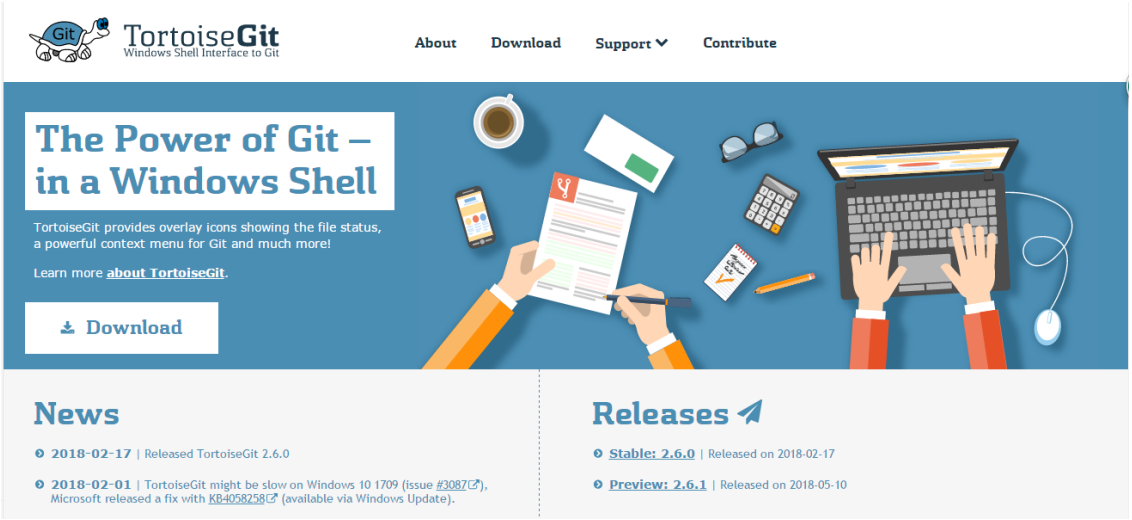




2.2 客户端工具 TortoiseGit 安装

TortoiseGit 是非常优秀的开源的版本库客户端，为操作 git 提供了方便友好的客户端图形界面。只支持 Windows 下载安装： https://tortoisegit.org/download/，Mac 下推荐使用 sourcetree2.3。

安装说明：因为 TortoiseGit 只是一个程序壳,必须依赖一个 Git Core,所以安装前请确定已完成 git 安装和配置。



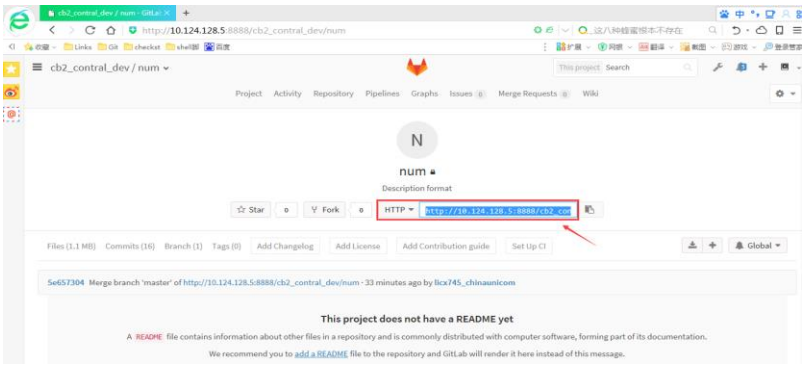
第 3 章 TortoiseGit 基础操作

Git 是命令行操作模式，TortoiseGit 界面化操作模式，不用记 Git 相关命令就可以直接操作，对于初次接触 Git 的用户来说使用界面化操作模式更易上手，所以本课程主要使用客户端工具 TortoiseGit 进行版本控制的常用操作介绍。

3.1 Git Clone

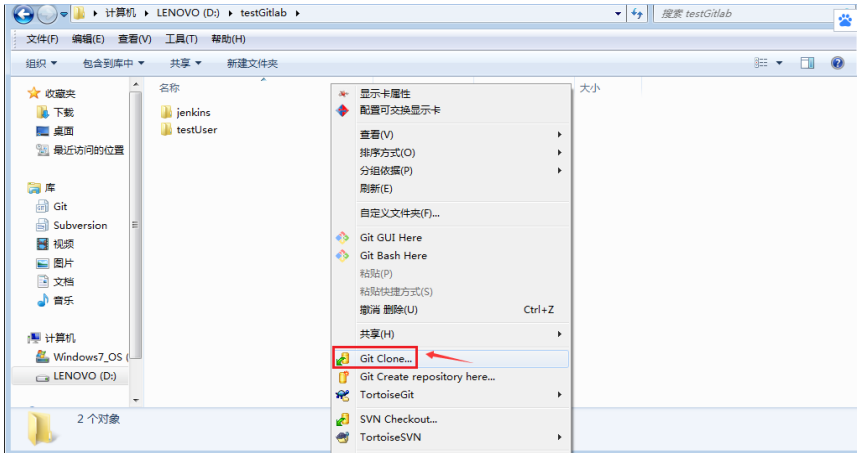
将已有的 Git 仓库克隆出一个新的镜像仓库到本地：

1) 先获取该工程的 Git 地址，可登录 Gitlab 获取：

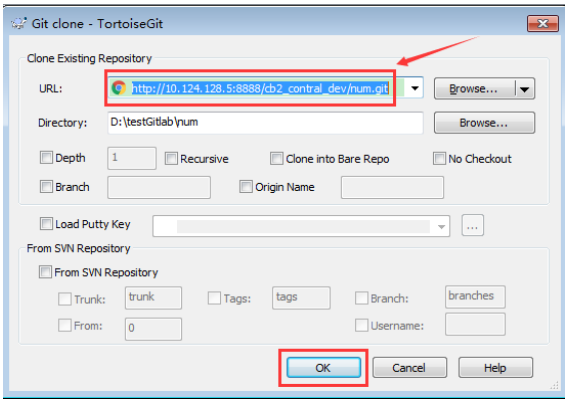


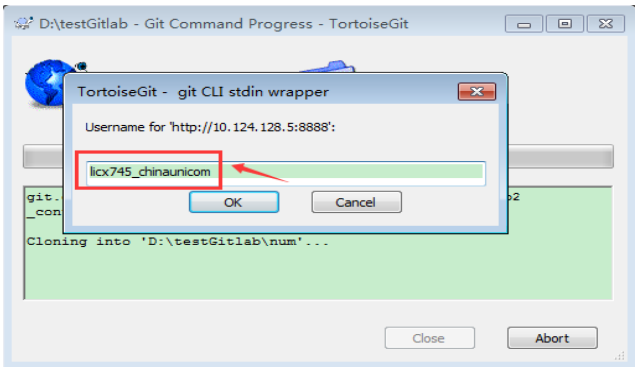
2) 在目标文件夹内，右击鼠标-->”Git Clone”，目标文件夹即你要将这个版本

库存放的本地路径：



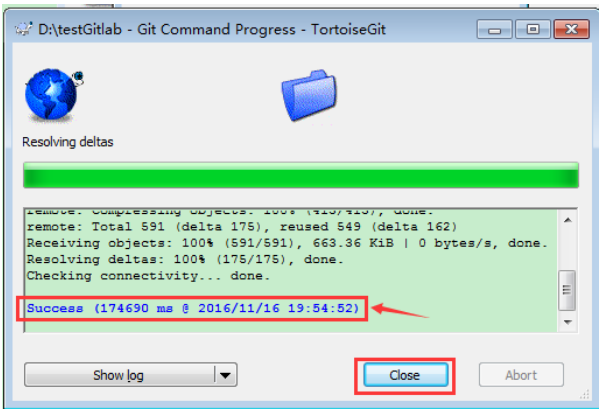
3） 在 Git Clone 窗口，仓库 URL 处会显示已复制的工程地址，点击“OK”继续，

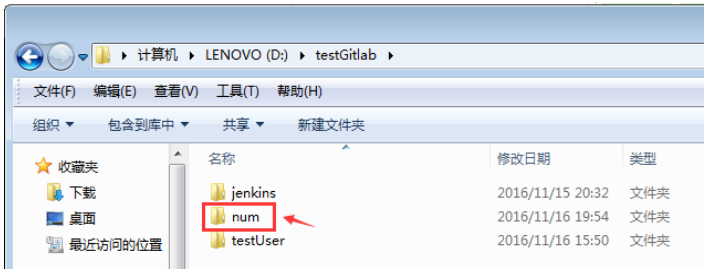




首次克隆某 Gitlab 服务器上的代码时，会提示输入 Git 账号和密码，所以在克隆代码前确定已申请 Git 账号并添加权限。

4） Clone 成功后会显示“succes”，点击“close”完成，目标文件夹内会出现刚才 clone 的工程：

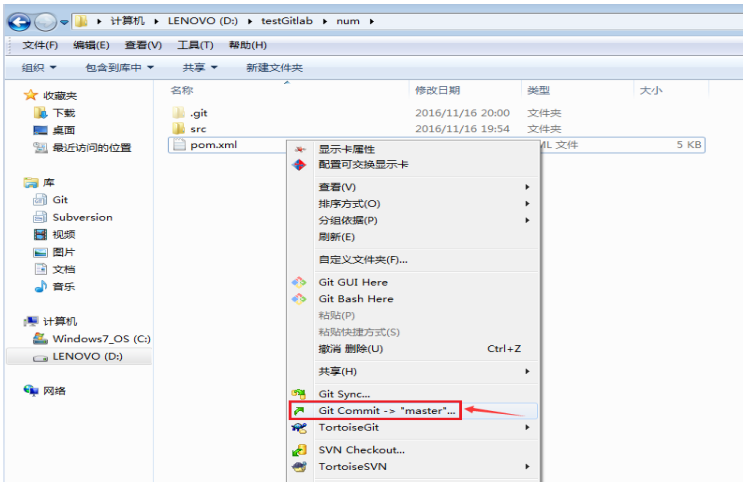




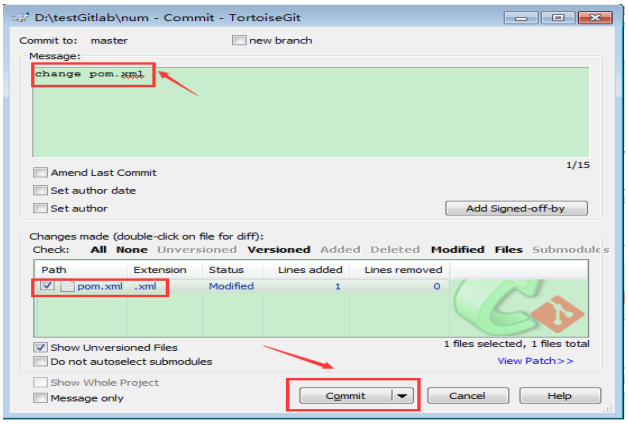
3.2 Git Commit

commit 是提交代码到本地仓库。比如，在本地修改了一个文件，首先需要把变更提交到本地仓库，再推送到远端仓库。

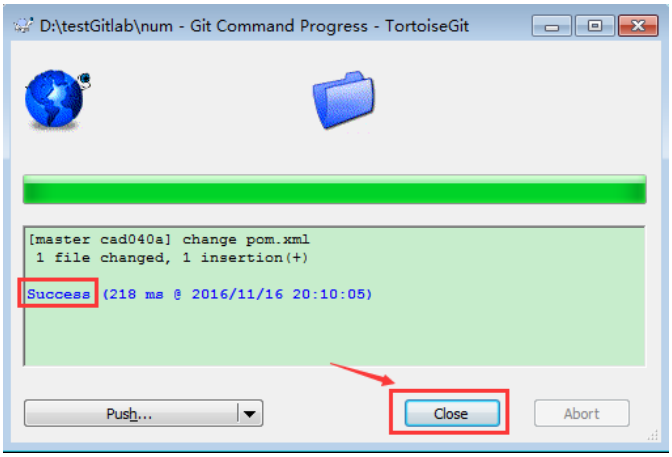
1) 在文件夹空白处，右击鼠标-->Git Commit，选择要提交到本地哪个分支：



2) 在 commit 窗口填写 commit message，即说明本次提交了哪些内容的变更，勾选 commit 的文件，点击“commit”继续：



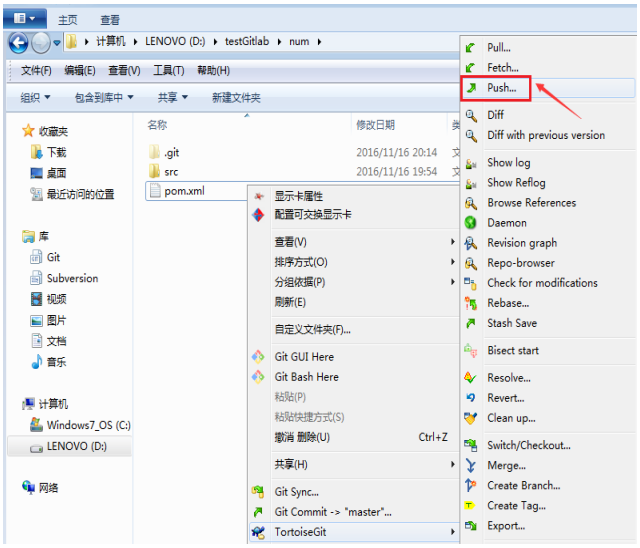
3) Commit 成功后显示“Success”，点击“Close”完成：



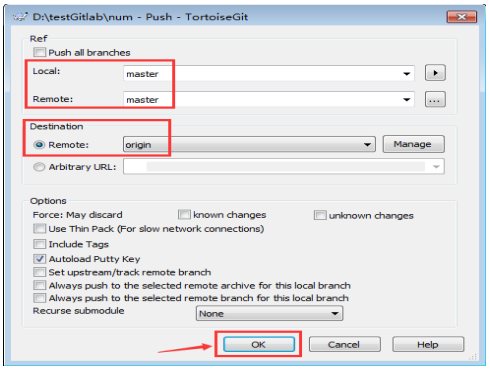
3.3 Git Push

Push 是推送代码变更到远程仓库，

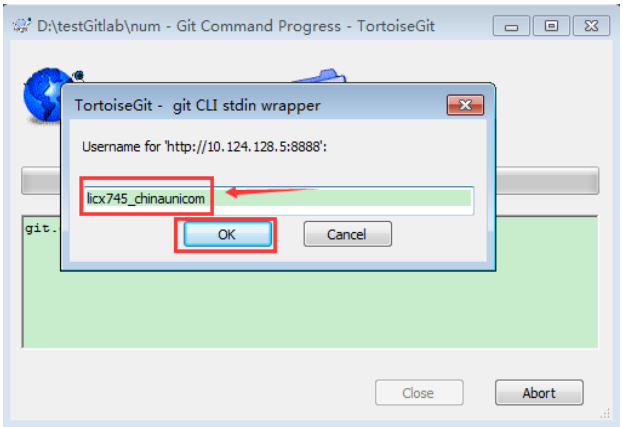
1）在文件夹空白处，右击鼠标-->TortoiseGit-->Push：



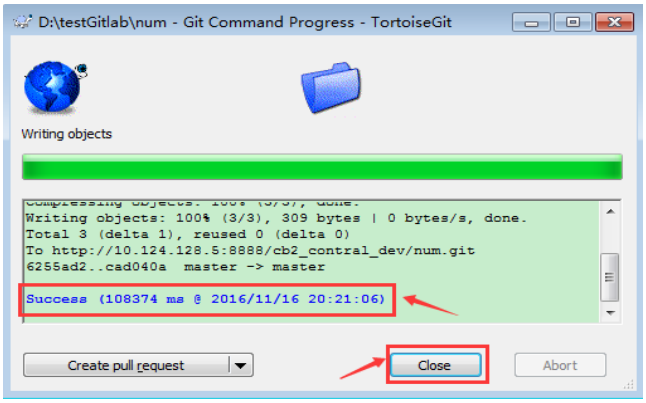
2）Push 窗口会显示将本地分支推送到远程分支，点击“OK”继续：



3）输入用户名和密码，点击“OK”继续：



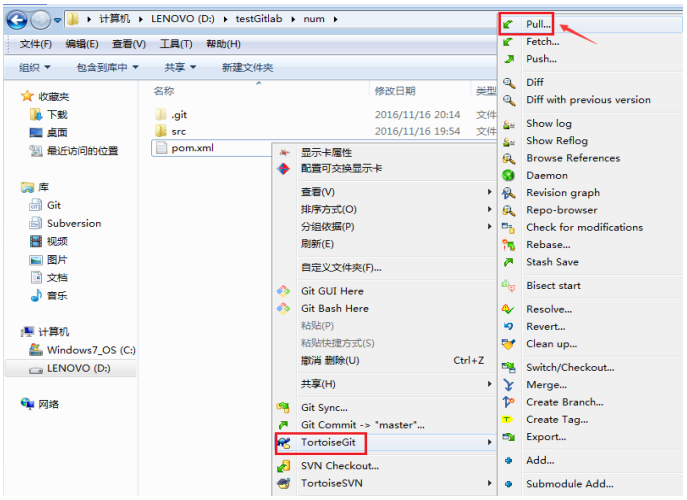
4）Push 成功后会提示“Success”，点击“Close”完成：



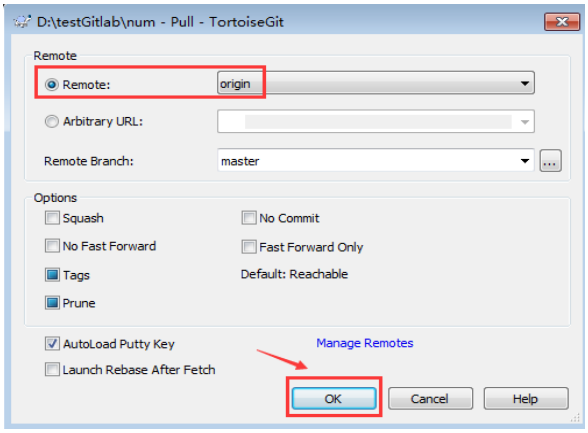
3.4 Git Pull

Git Pull 是拉取远程代码，将本地代码与远端代码仓库同步。

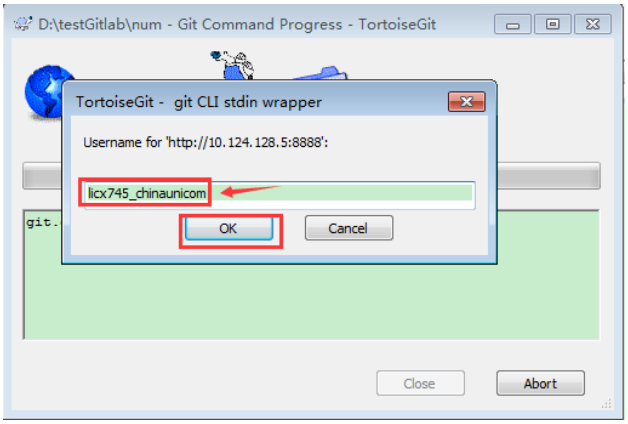
1）在文件夹空白处，右击鼠标-->TortoiseGit-->Pull：



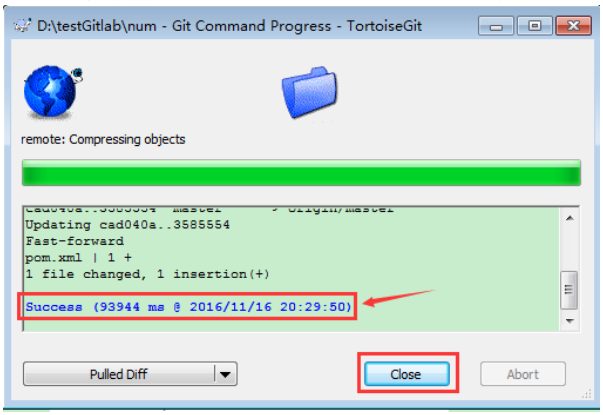
2）Pull 窗口会显示拉取的远程分支，点击“OK”继续：



3）输入用户名和密码，点击“OK”继续：



4）Pull 成功会提示“Success”， 点击“Close”完成：



3.5 Git 分支

几乎每一种版本控制系统都以某种形式支持分支。使用分支意味着你可以从开发主线上分离开来，然后在不影响主线的同时继续工作。

我们来看一个简单的分支与合并的例子，实际工作中大体也会用到这样的工作流程：

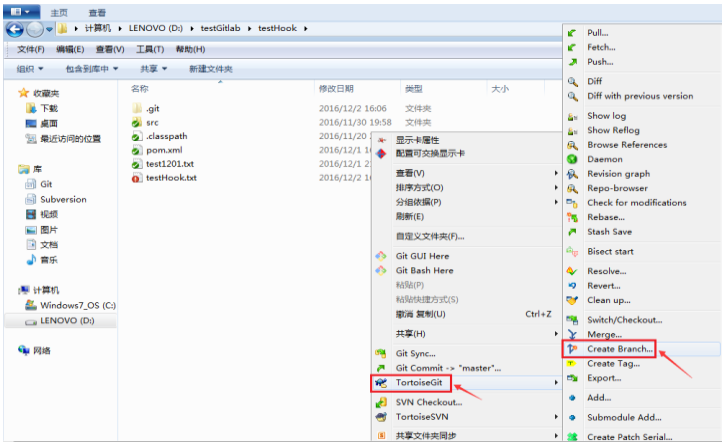
- 开发某个网站。
- 为实现某个新的需求，创建一个分支。
- 在这个分支上开展工作。

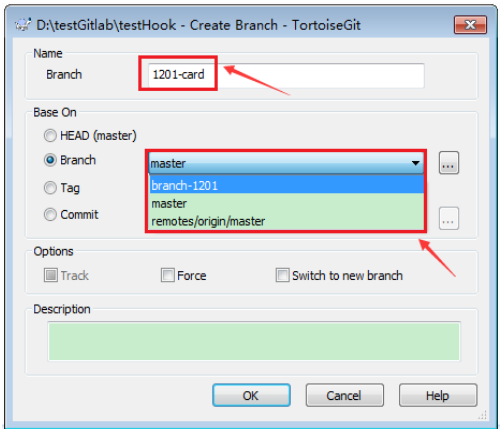
假设此时，你突然接到一个电话说有个很严重的问题需要紧急修补，那么可以按照下面的方式处理：

- 返回到原先已经发布到生产服务器上的分支。
- 为这次紧急修补建立一个新分支，并在其中修复问题。
- 通过测试后，回到生产服务器所在的分支，将修补分支合并进来，然后再推送到生产服务器上。
- 切换到之前实现新需求的分支，继续工作。

3.5.1 新建分支

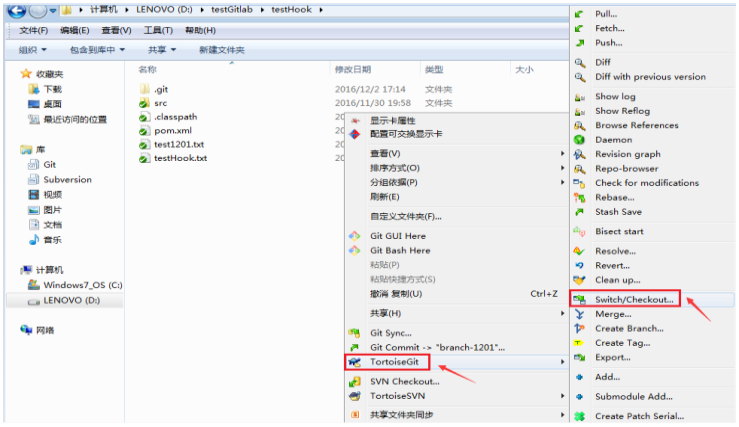
在工程目录下，右击鼠标-->TortoiseGit-->Create Branch，填写分支名称等信息：



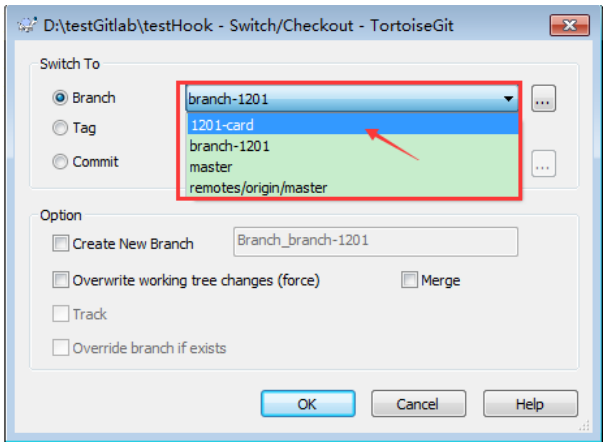


3.5.2 切换分支

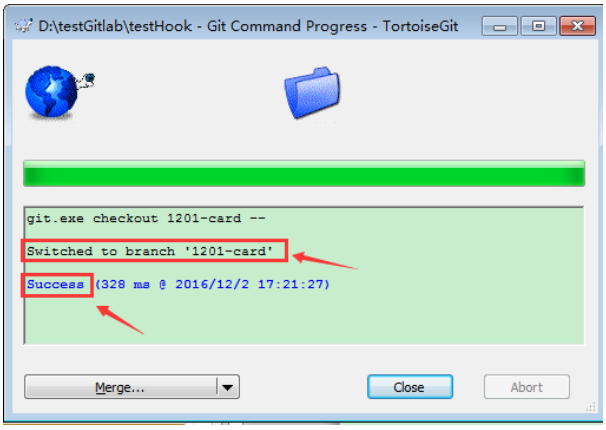
1) 在工程目录下，右击鼠标-->TortoiseGit-->Switch/Checkout：



2) 选择要转换到的分支，然后点击“OK”完成分支切换



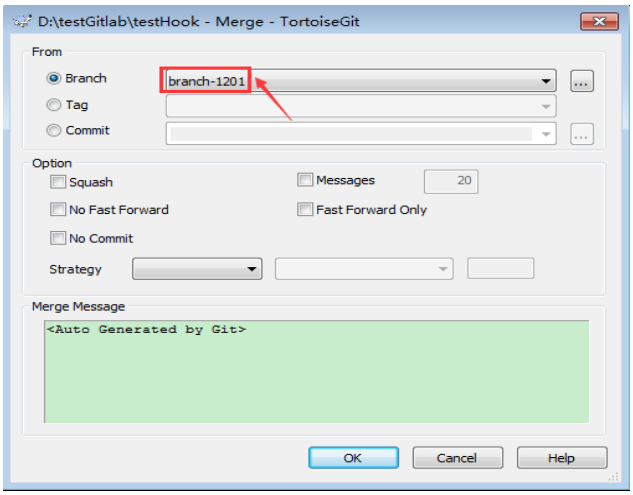
3) 成功后显示的信息如下：



3.5.3 合并分支

分支合并首先要切换到要合并到的分支，然后执行 merge 操作，将要合并的分支 merge 过来。本例将 branch-1201 合并到 1201-card 分支：

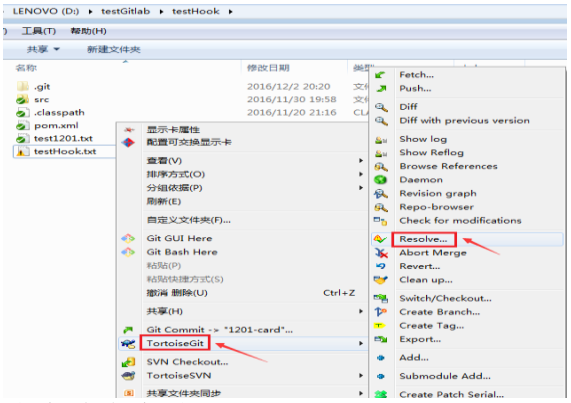
1) 首先切换到 1201-card 分支
2) 右击鼠标-->TortoiseGit-->Merge
3) 在 Merge 窗口中从 Branch 后的下拉列表中选择要 merge 过来的分支，本例为 branch-1201，Merge Message 这块可以写也可以默认，然后点击“OK”完成分支合并。



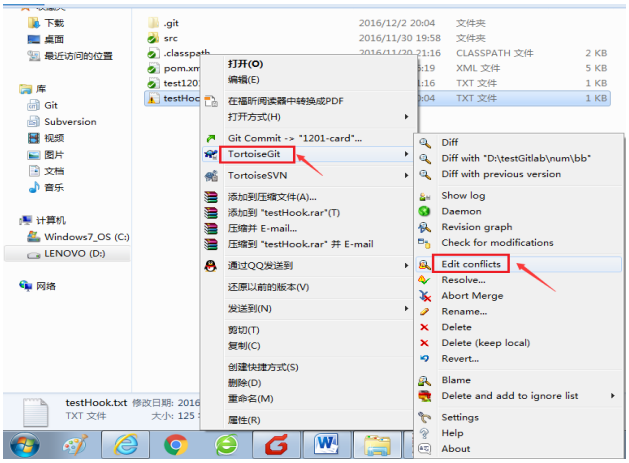
3.5.4 解决合并冲突

在合并分支的操作中，有可能会发生冲突，提示“CONFLICT”并且发生冲突的文件会有黄色三角和黑色叹号的标识。解决冲突的方法有两个：

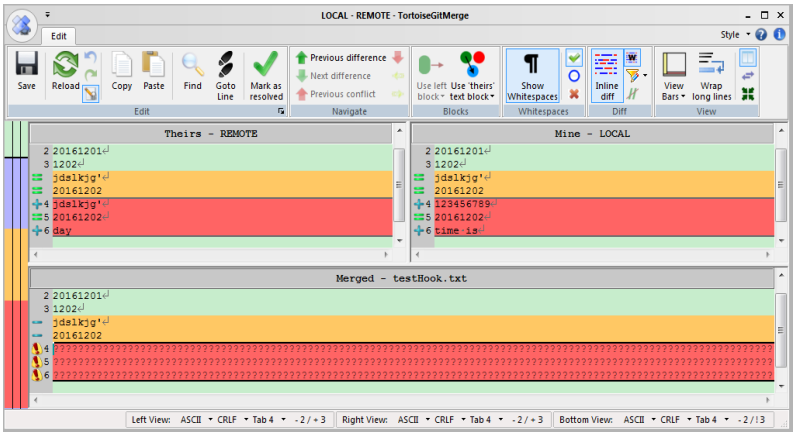
1) 打开冲突的文件，本例为 testHook.txt，删除冲突标识，并修改为需要的内容，点击保存，然后右击冲突文件-->TortoiseGit-->Reslove…，将冲突文件标记为冲突解决，最后提交即可



2) 右击冲突文件-->TortoiseGit-->Edit conflict:



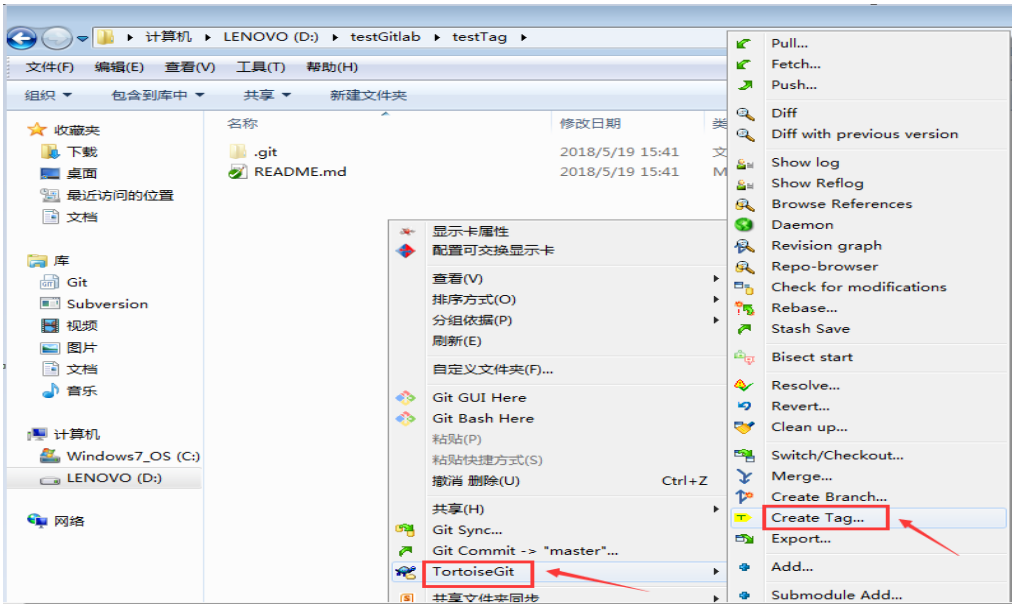
在冲突解决窗口中选择最终的代码:



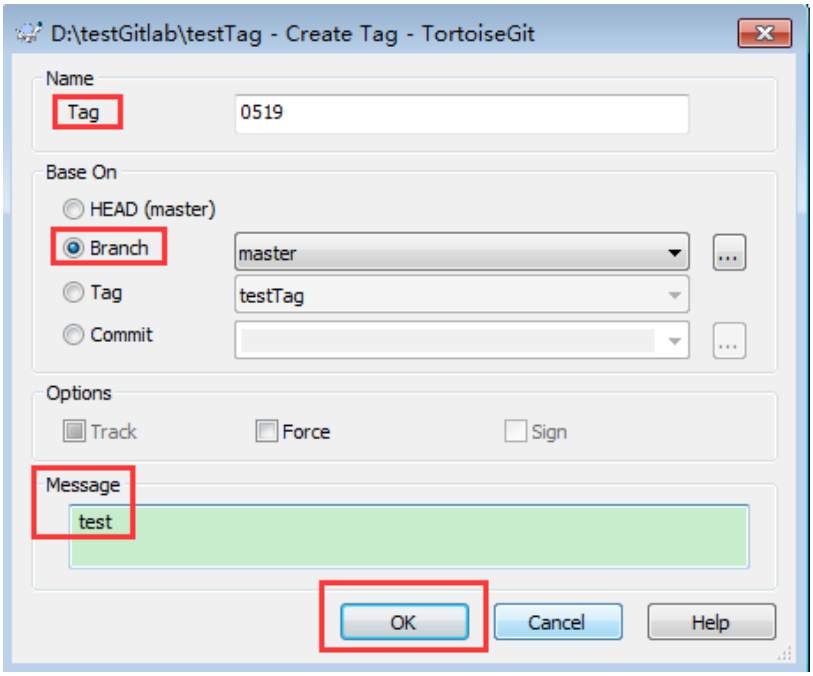
3.5.5 打 Tag

打 Tag 标志着一个成熟的版本，相当于里程碑的作用，方便后续代码回滚和追溯。

1) 右击鼠标--- TortoiseGit---Create Tag…



2) 在新建 Tag 窗口，填写 tag 的名字、打 tag 的分支以及 message，点击 OK，完成新建 tag：



第 4 章 天梯 Git 分支策略设计方案

4.1 分支策略

对于一个完整的工程，共包括三类分支：

- **特性分支**

1) 命名规范：task_天梯研发任务 ID 号_迭代日期（例：task_54321_190520）

2) 权限规范：研发人员有权限操作

3) 管理规范：以任务为单位，创建研发任务时自动拉取，一个工程可存在多个

task 分支。

*注：研发人员需要在天梯平台创建研发任务，研发任务自动创建对应的特性分支，在特性分支上进行相应的需求开发工作。

- release 分支（版本分支）

1)命名规范：release__迭代结束时间（如 release_190520）

2)权限规范：研发人员创建迭代研发任务时自动创建

3)管理规范：以迭代为单位，一个工程可存在多个版本分支。

- Master 分支

1)命名规范：master

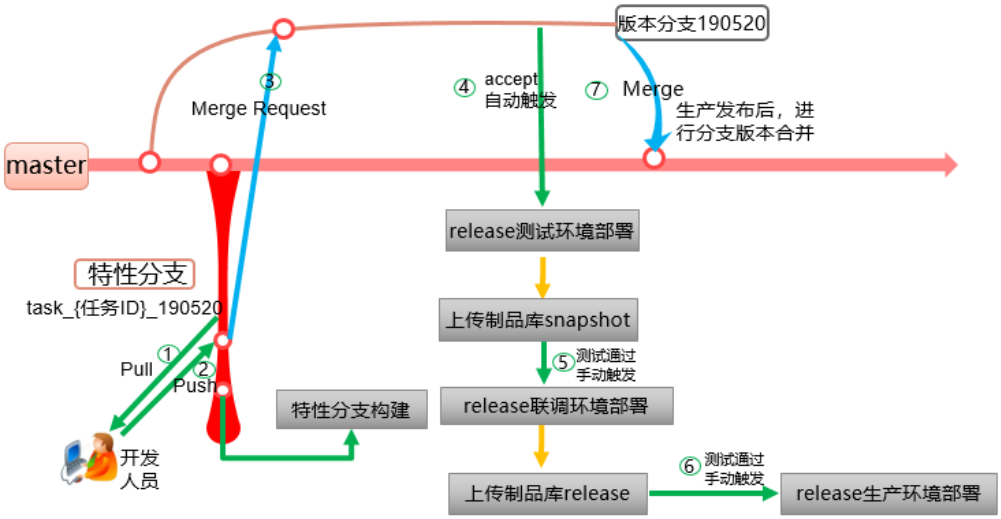
2)权限规范：master 或管理组有权限操作，研发人员可通过天梯平台进行创建。

3)管理规范：一个工程只存在一个 master 分支。

4.2 天梯版本发布 Git 流程

Master 作为工程的主分支，首先一个需求拆分多个研发任务，然后根据每个研发任务从 master 主分支拉取特性分支 feature 分支，进入到开发环节。可以通过 push commit 信息、打 tag 提交开发完成的代码，在每次 commit 信息被 push 到远端分支后，都会自动触发编译、单元测试、质量检查，保证每次被提交的代码都是可用的。当 tag 信息被 push 到远程分支后，触发开发环境的自动部署，进入开发环境自测阶段，自测没问题了，可以提交合并到 release 分支的 merge 申请，接下来会把 release 分支发布至测试环境，测试环境测试通过后，将 release 分支合并至主分支 master，并且打一个可以发布生产的 tag，进入准生产、生产环境的发布。这里要注意的是，从准生产到生产是不会有再一次的编译打包的，因为经过验证可以发布的代码，不应该多次打包，以免打包的过程中出现不可预期的问题，那么我们在从准生产到生产的过程中，只会修改配置文件，而不会对编译好的二进制包做任何的动作。

在整个的流程中，没有太多的手工动作，只保留必要的代码提交，代码审核等动作，其他的动作都是通过自动化流程工具来完成。这样能够保证项目参与人员只关注自己的任务本身，而不被流程牵扯太多的精力。



① Pull 拉取代码
② Push 特性分支，触发构建（代码检查、质量门禁）
③ 邮件通知需求下任务完成，经校验，可提出 Merge Request
④ accept 触发 release 分支测试环境构建部署（代码检查/质量门禁）
⑤ 测试环境验证通过，拉取 snapshot 软件包部署联调环境
⑥ 联调环境验证通过，拉取 release 软件包部署生产环境
⑦ 生产发布，合并 release 至 master

第 5 章 天梯版本控制实操

5.1 创建 Git 账号

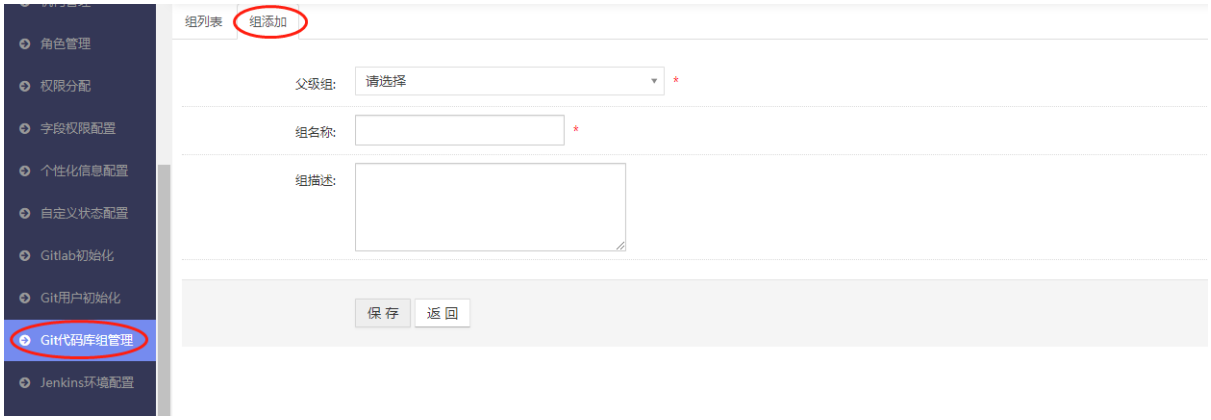
在实际项目中，由于平台权限控制，为团队开发人员创建 Git 账号是由项目管理员来操作。

点击“配置”菜单，从“未分配用户”找到申请 Git 账号的人员名字，勾选后，点击“批量创建”，提示“创建成功”，到“已分配用户”查看对应 Git 账号登录名。

5.2 Git 组管理

5.2.1 创建 Git 组

Git 组可以帮助项目有效管理人员和工程的操作权限。点击“配置”菜单，选择“Git 代码库组管理”，填写组信息，保存即完成 Git 组创建。按照天梯的管理规范，每个项目有一个根组，后来创建的组都属于根组的子组。



5.2.2 Git 组分配用户

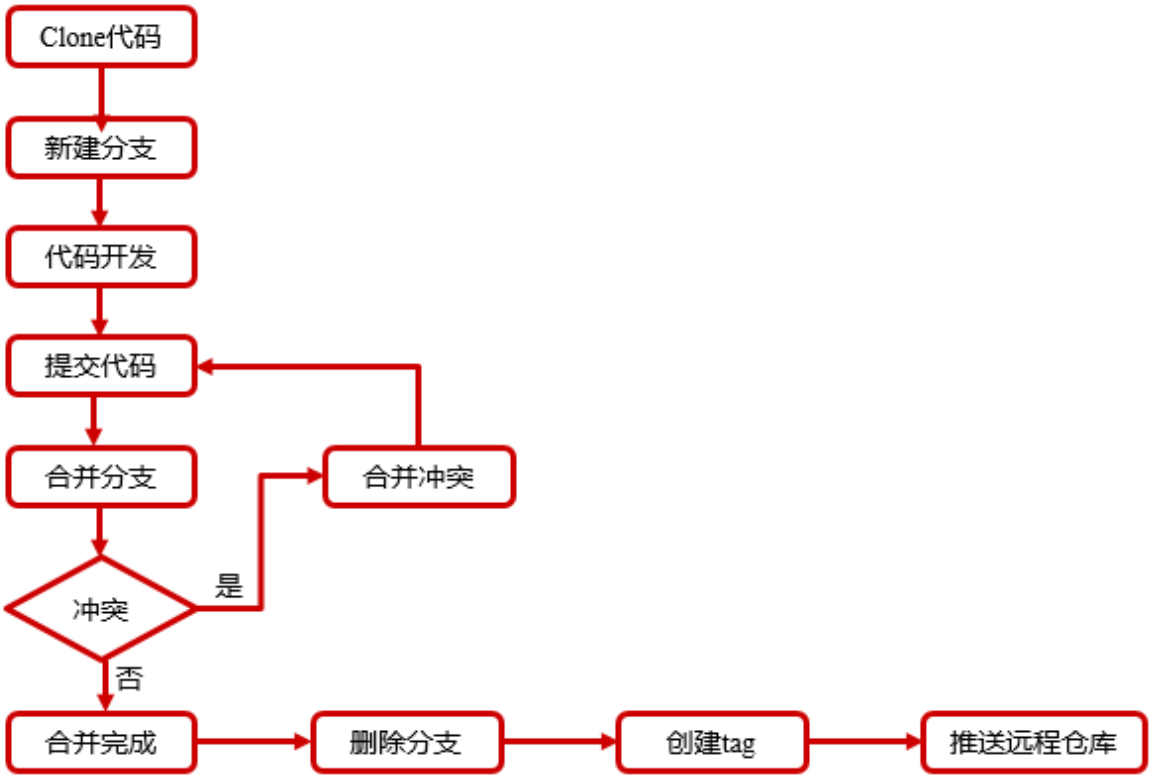
把人员分配进组里，此人员的 Git 账号才拥有对该组下工程的操作权限，并且才可以在该组下创建工程。分配完成以后，默认为开发者 developer 权限，可满足普通开发成员的日常操作，若要获取对工程 master 分支的操作权限，需要升级权限，在天梯进行“升级组管理”操作。

5.3 Git 工程实操

点击“代码与应用”->代码库列表，使用 TortoiseGit 克隆工程到本地，结合对 Git 的学习，进行演练。


仪表盘
工作
代码与应用
代码库列表
合并请求列表
批量合并分支
构建与发布
测试
支持
按代码名称筛选
新建代码库
代码库
项目地址
创建时间
DevOpsManagement/usercenter
查看信息
http://10.124.128.2:8888/DevOpsMan
2020-04-29 02:39:23.0
DevOpsManagement/ctservice
查看信息
http://10.124.128.2:8888/DevOpsMan
2020-04-10 01:23:28.0
DevOpsManagement/drweb
查看信息
http://10.124.128.2:8888/DevOpsMan
2020-04-09 07:17:57.0
DevOpsManagement/notice_center
查看信息
http://10.124.128.2:8888/DevOpsMan
2020-04-03 02:19:10.0
DevOpsManagement/notice_service
查看信息
http://10.124.128.2:8888/DevOpsMan
2020-03-26 03:35:45.0
DevOpsManagement/pm
查看信息
http://10.124.128.2:8888/DevOpsMan
2020-02-27 07:57:56.0

演练流程：


Clone代码
新建分支
代码开发
提交代码
合并分支
合并冲突
冲突
是
否
合并完成
删除分支
创建tag
推送远程仓库