**Git分布式版本控制工具**

# 概述

## 1.1开发中的实际场景

场景一：备份 小明负责的模块就要完成了，就在即将Release之前的一瞬间，电脑突然蓝屏，硬盘光荣牺牲！几个月来的努力付之东流

场景二：代码还原 这个项目中需要一个很复杂的功能，老王摸索了一个星期终于有眉目了，可是这被改得面目全非的代码已经回不到从前了。什么地方能买到哆啦A梦的时光机啊？

场景三：协同开发 小刚和小强先后从文件服务器上下载了同一个文件：Analysis.java。小刚在Analysis.java文件中的第30行声明了一个方法，叫count()，先保存到了文件服务器上；小强在Analysis.java文件中的第50行声明了一个方法，叫sum()，也随后保存到了文件服务器上，于是，count()方法就只存在于小刚的记忆中了

场景四：多版本同步开发不同期上线 老许是一位项目经理，根据产品经理的要求，需要同步开发多个功能，但是会在不同的时间节点上线，针对同一工程和项目，应该怎么解决这个问题？

场景五：追溯问题代码的编写人和编写时间！ 老王是另一位项目经理，每次因为项目进度挨骂之后，他都不知道该扣哪个程序员的工资！就拿这次来说吧，有个Bug调试了30多个小时才知道是因为相关属性没有在应用初始化时赋值！可是二胖、王东、刘流和正经牛都不承认是自己干的！

## 1.2版本控制器的方式

1. 集中式版本控制工具

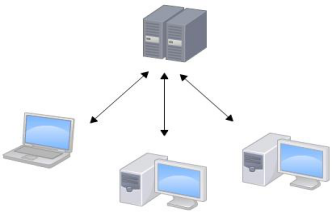
集中式版本控制工具，版本库是集中存放在中央服务器的，team里每个人work时从中央服务器下载代码，是必须联网才能工作，局域网或互联网。个人修改后然后提交到中央版本库。 举例：SVN和CVS

1. 分布式版本控制工具

分布式版本控制系统没有“中央服务器”，每个人的电脑上都是一个完整的版本库，这样工作的时候，无需要联网了，因为版本库就在你自己的电脑上。多人协作只需要各自的修改推送给对方，就能互相看到对方的修改了。 举例：Git

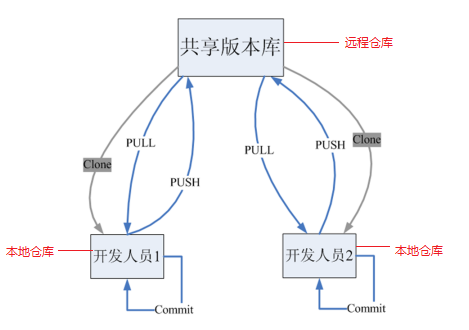
## 1.3 SVN

SVN是一个开放源代码的版本控制系统，是Apache Subversion的缩写。SVN是集中式的管理。SVN必须有一个服务器版本库就放在一个中央服务器，所有开发人员都是与服务器进行交互的。 优点：具有很强的权限控制 缺点：严重依赖中央服务器



## 1.4 Git

Git是分布式的，Git不需要有中心服务器，我们每台电脑拥有的东西都是一样的。我们使用Git并且有个中心服务器，仅仅是为了方便交换大家的修改，但是这个服务器的地位和我们每个人的PC是一样的。我们可以把它当做一个开发者的pc就可以就是为了大家代码容易交流不关机用的。没有它大家一样可以工作，只不过“交换”修改不方便而已。 git是一个开源的分布式版本控制系统，可以有效、高速地处理从很小到非常大的项目版本管理。Git是Linus Torvalds 为了帮助管理 Linux 内核开发而开发的一个开放源码的版本控制软件。 同生活中的许多伟大事物一样，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多的参与者。 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一个专有的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 Linux 内核社区免费使用 BitKeeper 的权力。 这就迫使 Linux 开源社区（特别是 Linux 的缔造者 Linus Torvalds）基于使用 BitKeeper 时的经验教训，开发出自己的版本系统。 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（允许成千上万个并行开发的分支） 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）



备注：

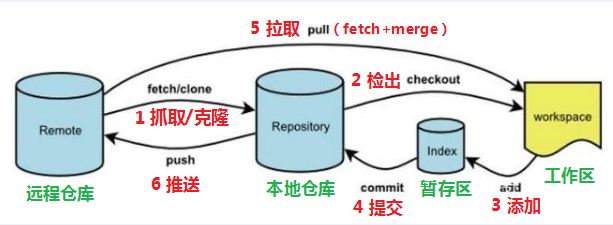
​ Clone：克隆，就是将远程仓库复制到本地

​ Push：推送，就是将本地仓库代码上传到远程仓库

​ Pull：拉取，就是将远程仓库代码下载到本地仓库

​ commit：提交，将工作区代码提交到本地仓库

## 1.5 Git工作流程图

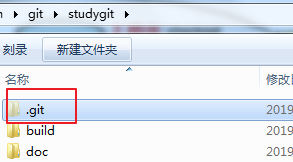


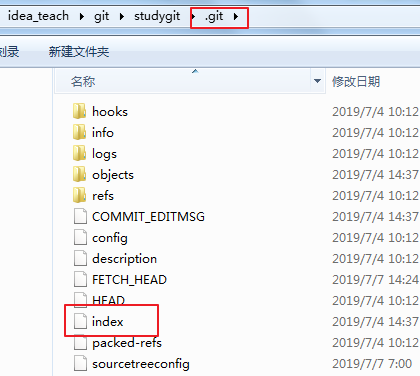
基本概念：

1.远程仓库 ： 在局域网或互联网上的一个主机，存放代码库的主机或平台，比如gitee.com(码云)

2.本地仓库： 在本地主机上的一个代码库，可以独立存在，也可以与远程仓库进行关联

凡是一个文件夹，包含.git隐藏文件夹（git工作目录），说明此文件目录使用git版本管理。.git文件目录中存储了很多配置信息、日志信息和文件版本信息、工作区、暂存区信息等。.git文件夹中有很多文件，其中有一个index文件就是暂存区，也可以叫做stage。暂存区是一个临时保存修改文件的地方。





1.工作区：对任何文件的修订(增删改)，都先放在工作区，工作区不与任何仓库分支进行关联

2.暂存区：把修订的文件，从工作区经过add（添加）后与某一个仓库分支进行关联，只有进入缓存区的文件才能commit(提交)到本地仓库。

3.分支：代码存放在仓库，默认是主分支(master)，可以在主分支基础上创建很多子分支，比如develop（开发）、bugfix（bug修复）等。

命令如下：

1.clone（克隆）: 从远程仓库中克隆代码到本地仓库

​ fetch (抓取) ： 从远程库，抓取到本地仓库，不进行任何的合并动作，一般操作比较少。

2.checkout （检出）:从本地仓库中检出一个仓库分支然后进行修订

3.add（添加）: 在提交前先将代码提交到暂存区

4.commit（提交）: 提交到本地仓库。本地仓库中保存修改的各个历史版本

5.pull (拉取) ： 从远程库拉到本地库，自动进行合并(merge)，然后放到到工作区，相当于fetch+merge

6.push（推送） : 修改完成后，需要和团队成员共享代码时，将代码推送到远程仓库

## 1.6 工作目录下的状态

Git工作目录下的文件存在两种状态：

untracked 未跟踪（未被纳入版本控制）

tracked 已跟踪（被纳入版本控制）

Unmodified 未修改状态

Modified 已修改状态

Staged 已暂存状态

这些文件的状态会随着我们执行Git的命令而发生变化

# 2 Git远程仓库

## 2.1 常用的托管服务[远程仓库]

前面我们已经知道了Git中存在两种类型的仓库，即本地仓库和远程仓库。那么我们如何搭建Git远程仓库呢？我们可以借助互联网上提供的一些代码托管服务来实现，其中比较常用的有GitHub、码云、GitLab等。 gitHub（ 地址：https://github.com/ ）是一个面向开源及私有软件项目的托管平台，因为只支持Git 作为唯一的版本库格式进行托管，故名gitHub 码云（地址： https://gitee.com/ ）是国内的一个代码托管平台，由于服务器在国内，所以相比于GitHub，码云速度会更快 GitLab （地址： https://about.gitlab.com/ ）是一个用于仓库管理系统的开源项目，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，一般用于在企业、学校等内部网络搭建git私服。

## 2.2 注册码云

要想使用码云的相关服务，需要注册账号（地址： https://gitee.com/signup ）



## 2.3 创建远程仓库





仓库创建好之后，可以查看仓库的信息



点击克隆/下载中的复制可以获取该仓库的地址

该仓库的地址为：https://gitee.com/wangshuang736/studygit.git

## 2.4 邀请其他开发人员

已经在码云上创建了自己的远程仓库，目前仓库成员只有自己一个人（身份为管理员）。在企业实际开发中，一个项目往往是由多个人共同开发完成的，为了使多个参与者都有权限操作远程仓库，就需要邀请其他项目参与者成为当前仓库的成员。

1. 进入仓库管理中心，点击仓库成员管理



1. 直接添加成员，输入成员在gitee的邮箱地址



3.输入邮箱后，自动提示码云用户信息，确定后点击添加



确定成员信息，点击提交



查看仓库成员



被邀请者登录，可收到添加进仓库的私信



# 3 分支

几乎所有的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把你的工作从开发主线上分离开来进行重大的Bug修改、开发新的功能，以免影响开发主线。

在开发中，一般有如下分支使用原则与流程：

**master （生产） 分支**

线上分支，主分支，中小规模项目作为线上运行的应用对应的分支；

**test（测试）分支**

从master创建的分支，一般作为测试部门的测试分支，进行预发测试。测试完成后，需要合并到master分支，进行发版上线，中小规模项目可省略此分支；

**develop（开发）分支**

从test创建的分支，如果开发没有test分支，是从master创建的分支，一般作为开发部门的主要开发分支，如果没有其他并行开发不同期上线要求，都可以在此版本进行开发，阶段开发完成后，需要是合并到test分支继续测试，如果没有test分支，可直接合并到master分支。

**develop\_xxx分支**

从develop创建的分支，一般是同期并行开发，但不同期上线时创建的分支，一般是合并到develop分支。

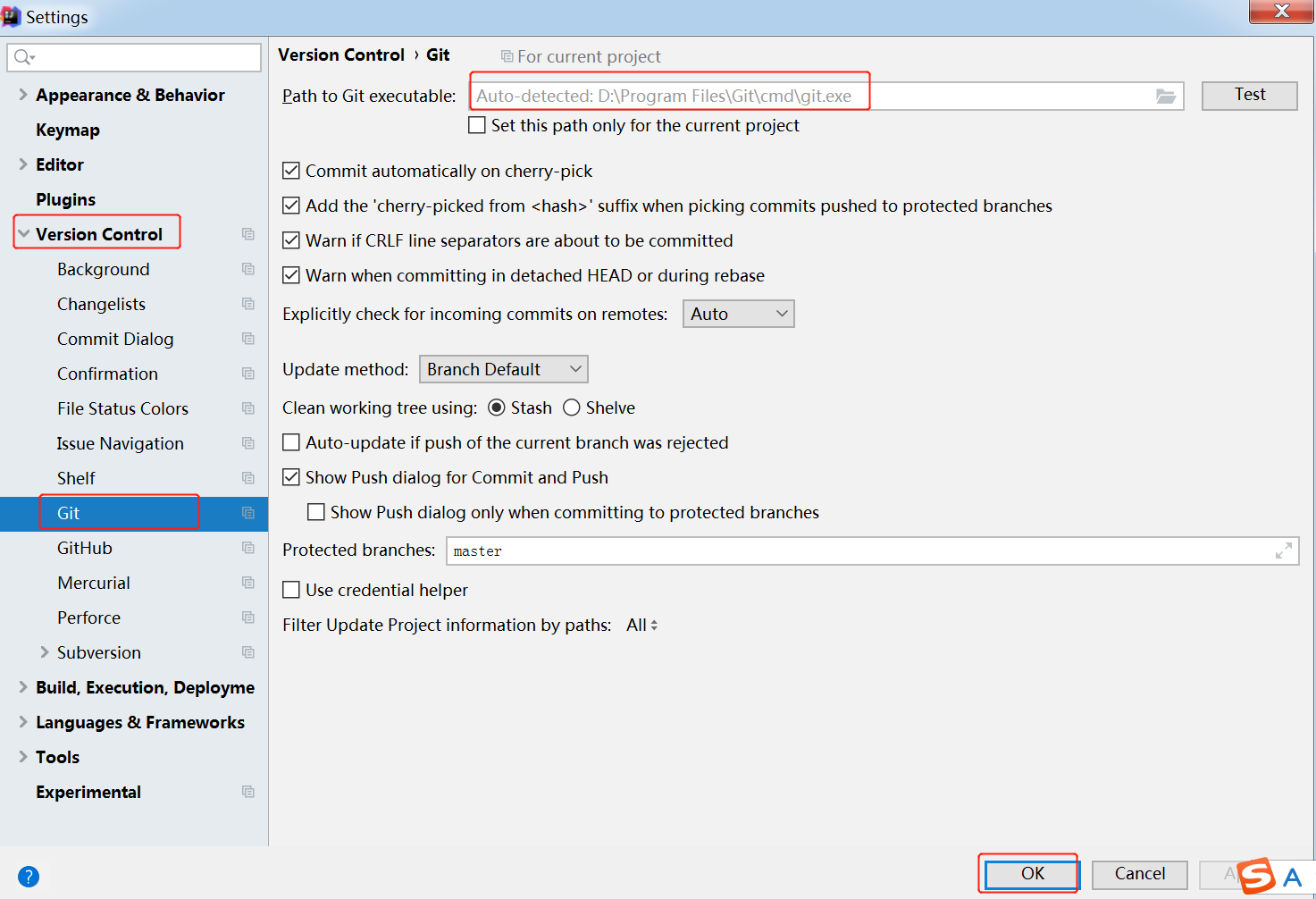
**hot**fix(bugfix)分支

从master派生的分支，一般作为线上bug修复使用，修复完成后需要合并到master、test、develop分支。

# 4 在IDEA中使用Git

## 4.1 在IDEA中配置Git

安装好IntelliJ IDEA后，如果Git安装在默认路径下，那么idea会自动找到git的位置，如果更改了Git的安装位置则需要手动配置下Git的路径。选择File→Settings打开设置窗口，找到Version Control下的git选项：



## 4.2 在IDEA中操作Git

### 4.2.1 初始化远程仓库 [项目leader操作]

创建项目远程仓库



初始化仓库并推送到远程

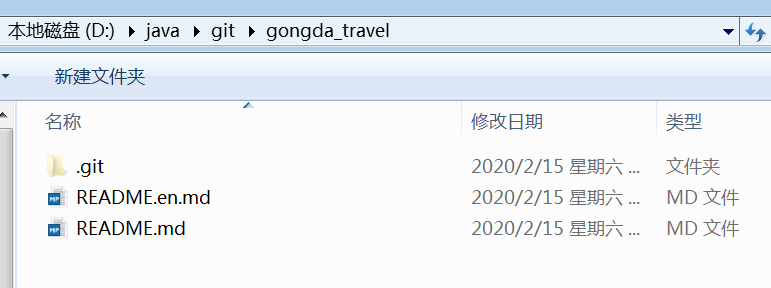
主要初始化忽略文件及项目初始化代码，并推送到远程仓库

1.clone远程仓库到本地

选定本地磁盘一个文件夹，使用git命令完成此操作，比如d:/java/git

clone完成后，如图所示





.git是隐藏文件

2.切换到开发分支

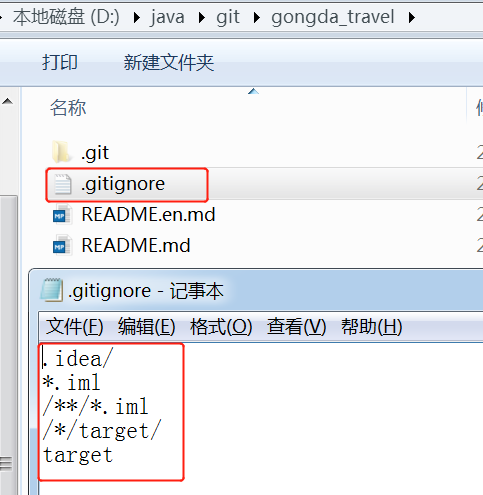
查看当前分支



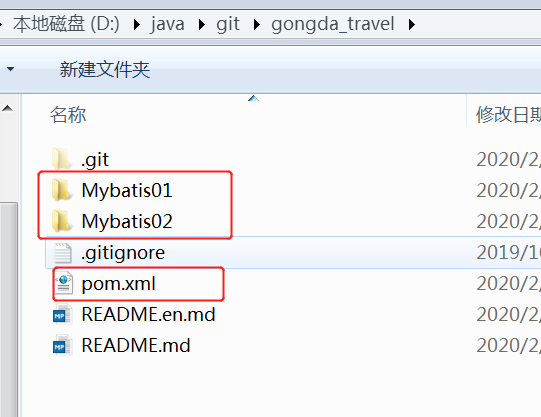
检出develop分支，并确定是在开发分支



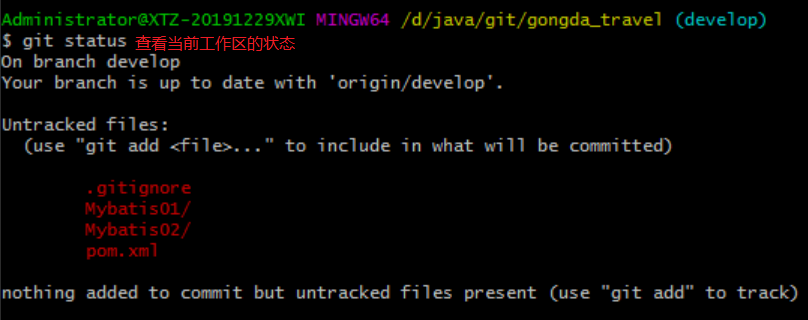
添加忽略文件

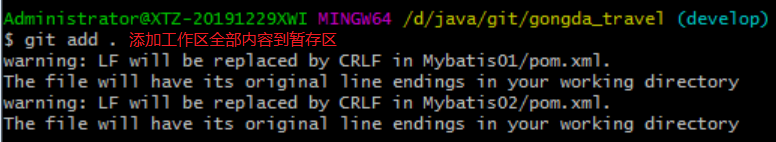


把初始化项目代码放到git仓库中



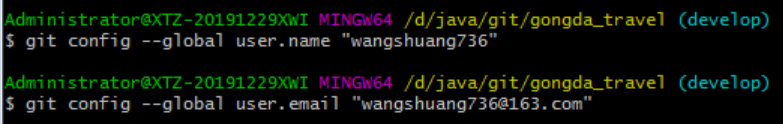
通过git命令完成添加、提交、推送





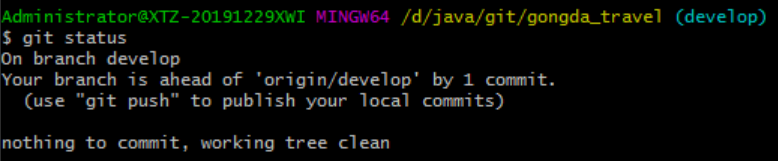


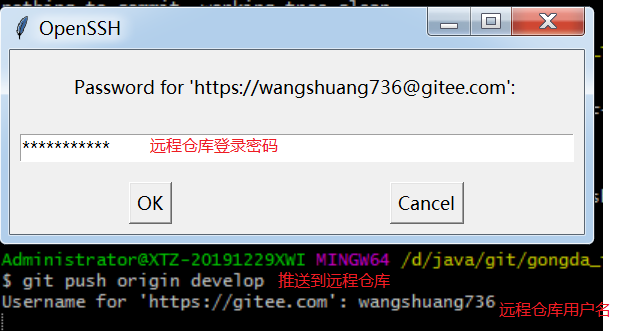
配置用户名和邮箱





查看当前状态，没有需要提交的内容了

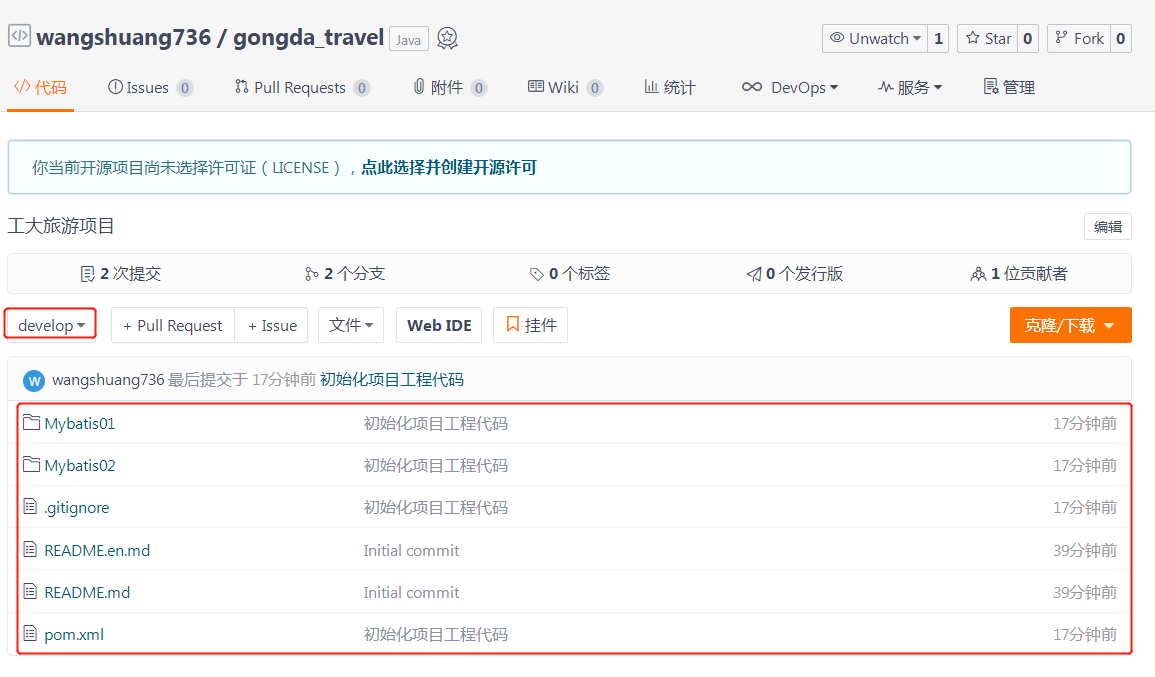






当前用户如有权限，会推送成功

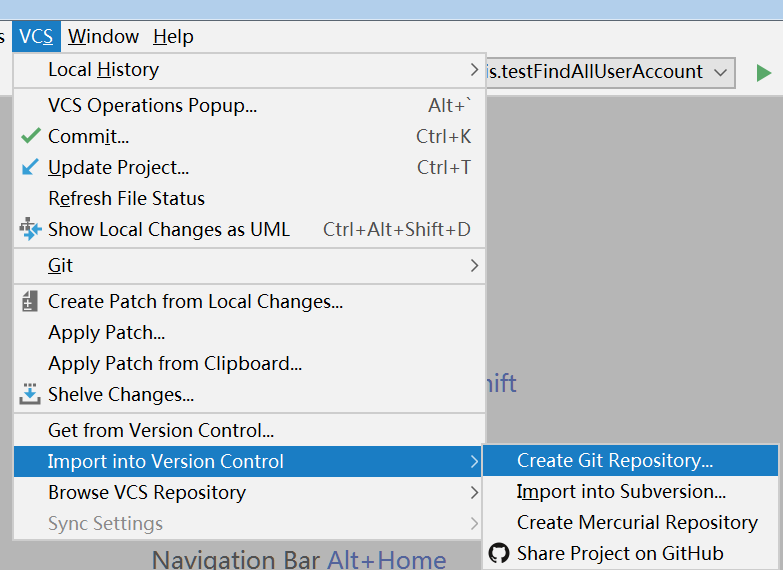
远程仓库中内容有变化



项目已经push到远程仓库，所有的开发人员可以从远程仓库中克隆代码了，下面是开发人员的操作方法

### 4.2.2 创建本地仓（不推荐使用）

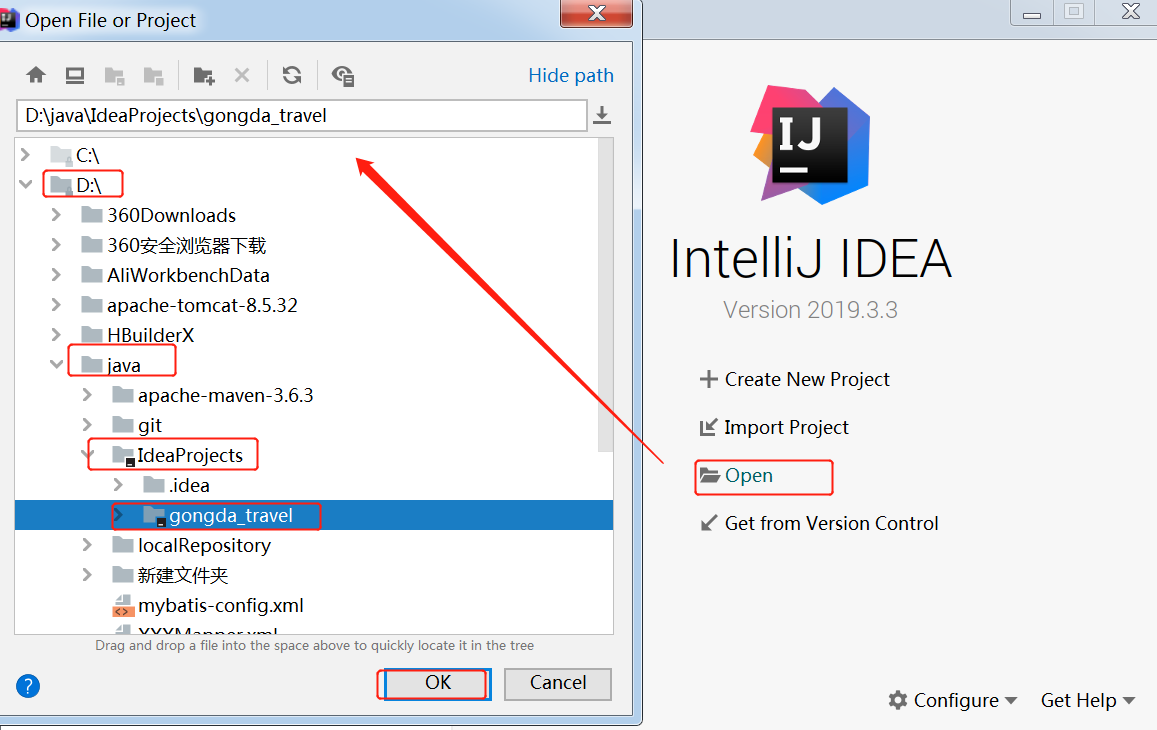
先在本地创建本地仓库，然后与远程仓库关联 ，不建议按此方式操作。

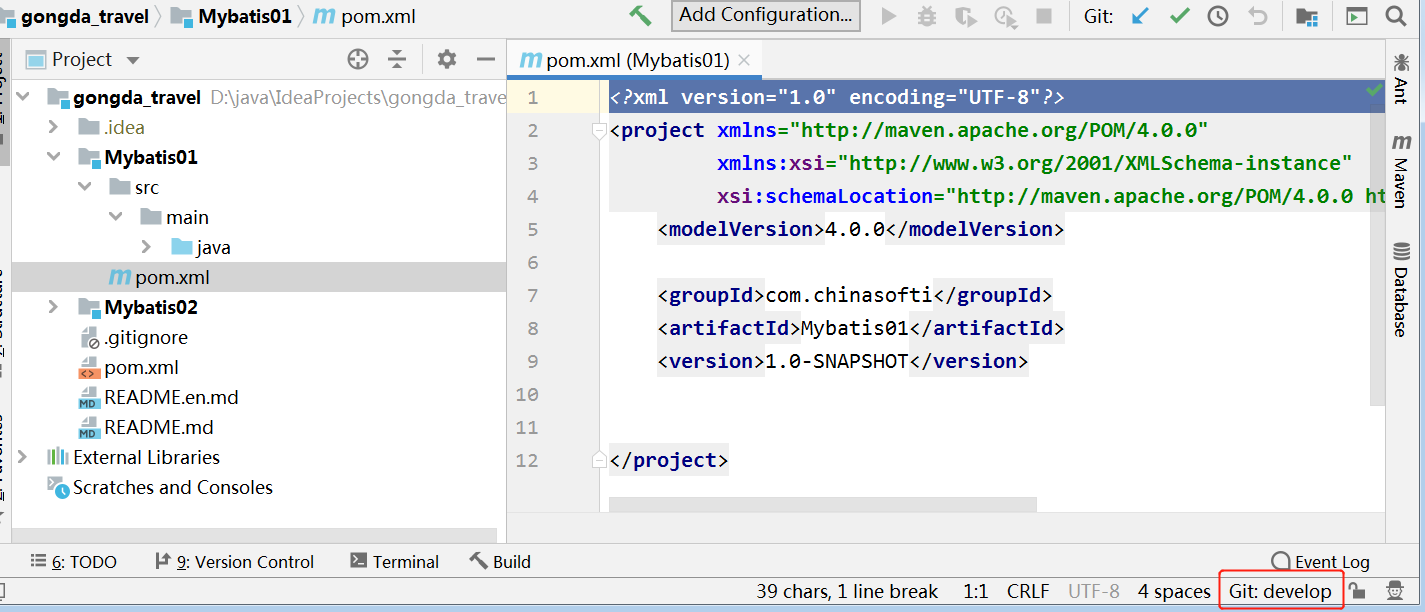


### 4.2.3克隆远程仓库到本地 （推荐使用）

与前面的方法相同，只是这里的路径是IDEA的工作空间，即把工程克隆到IDEA工作空间，然后用IDEA打开，注意是develop分支

### 4.2.4 打开clone的本地仓库项目



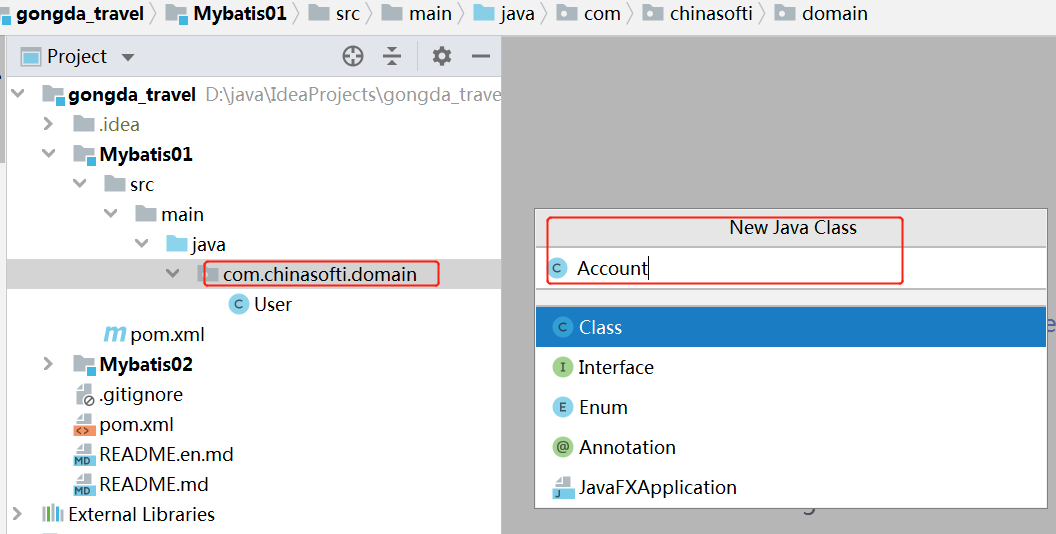


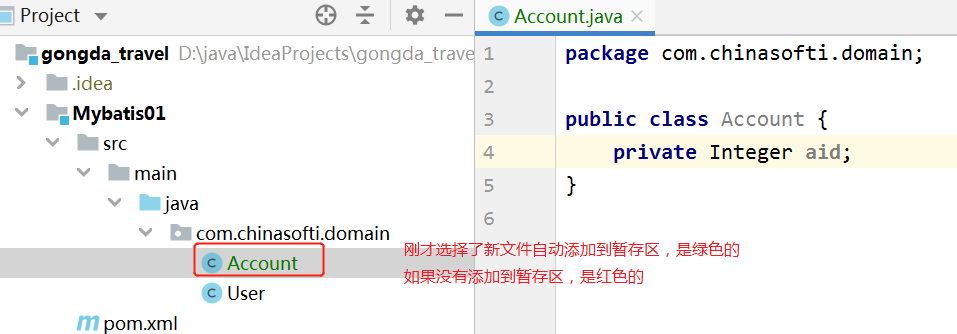
IDEA自动识别Git分支，现在是develop分支

### 4.2.5 本地仓库常规操作

1.新增文件

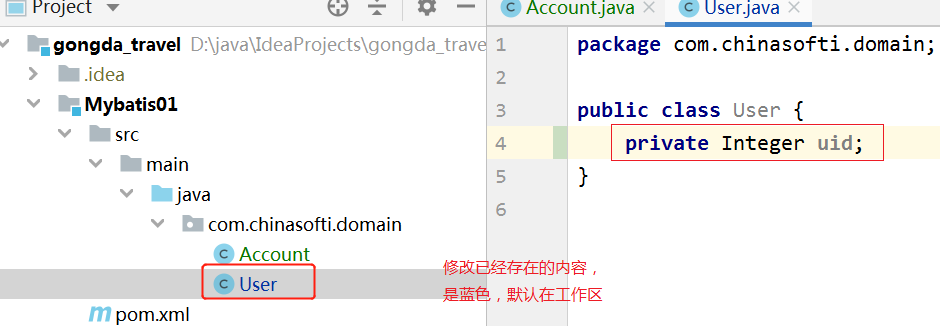
比如在某个包新增一个Java文件，如图所示：



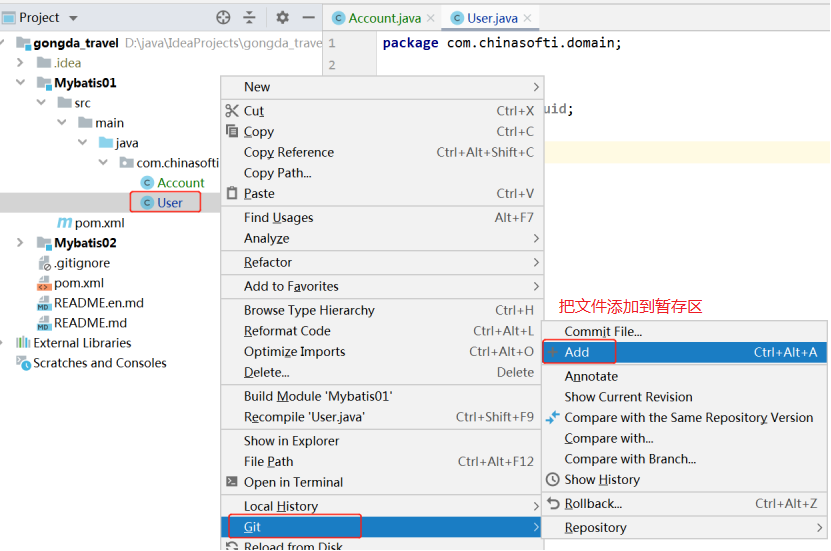


2.编辑文件

比如修改某一个包下已存在的文件内容，如图所示：

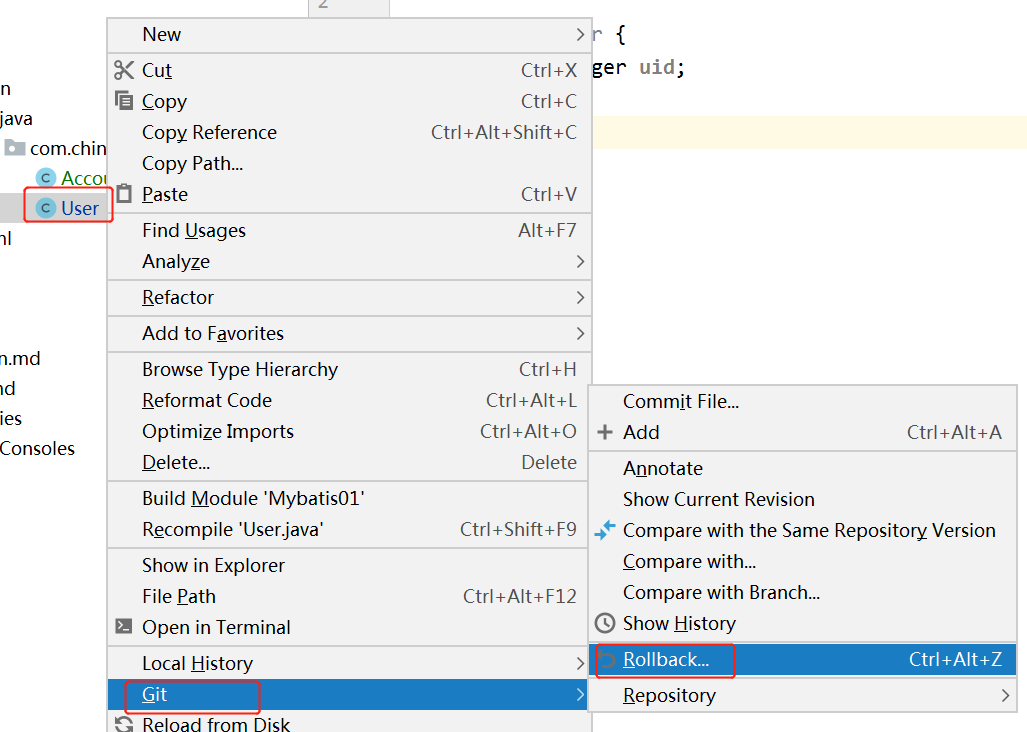


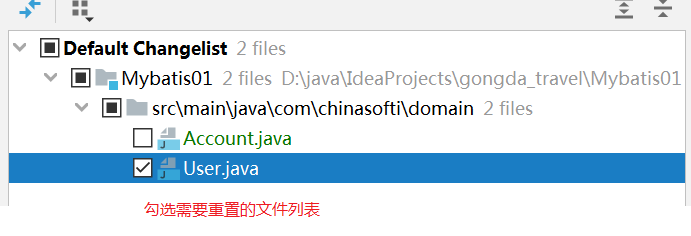
正常编辑文件，默认放在工作区，如果需要添加到暂存区，选择文件，右键菜单，选择git-Add



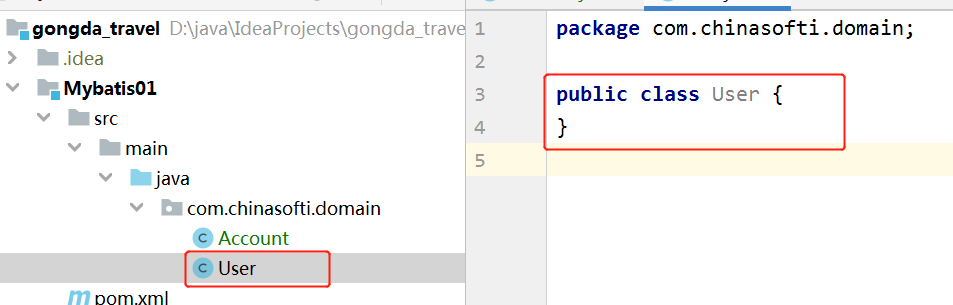
3.重置文件到修改前

比如修订了某一文件，需要重置到修改文件之前的状态，选择文件，右键菜单，选择git-rollback

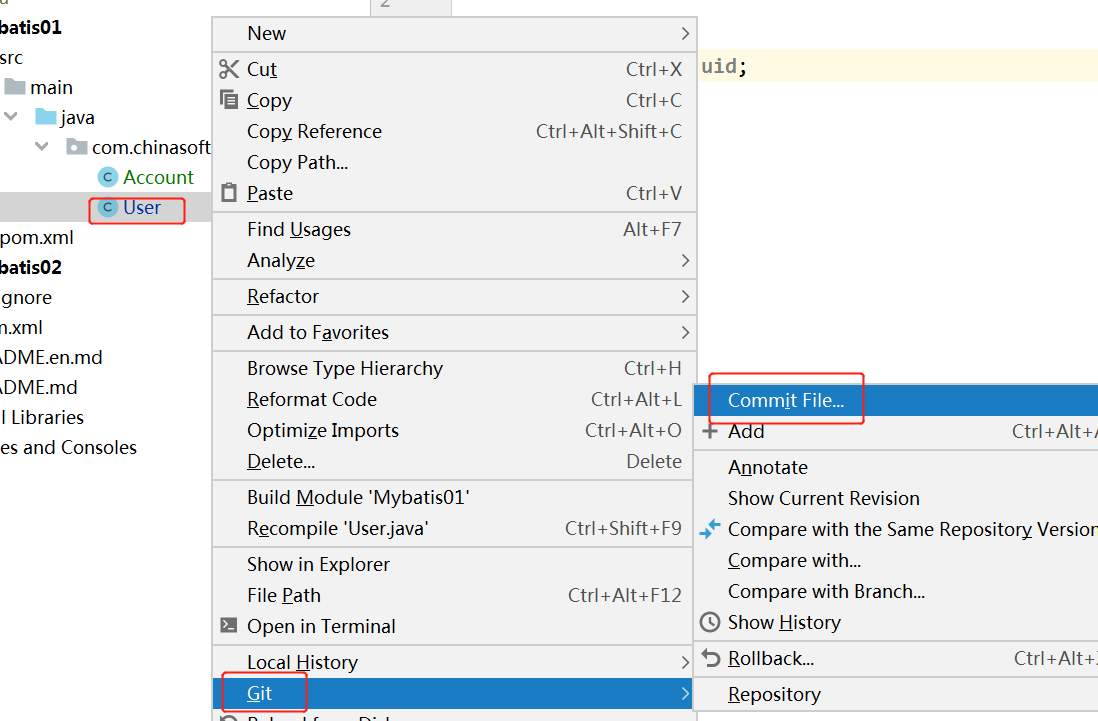


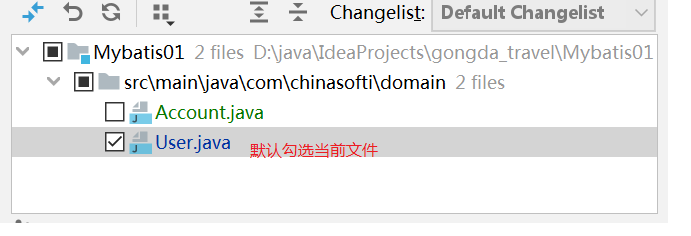


重置后，文件颜色自动消失，说明已重置到修改之前的状态。

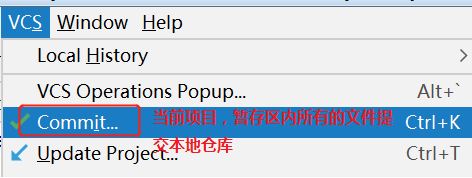
4.提交到本地仓库

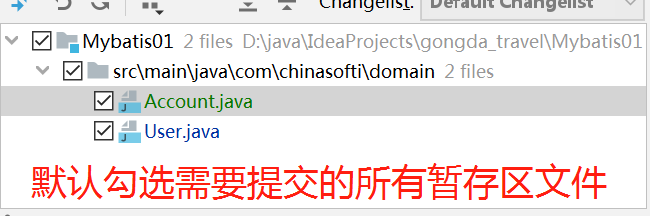
提交当前文件



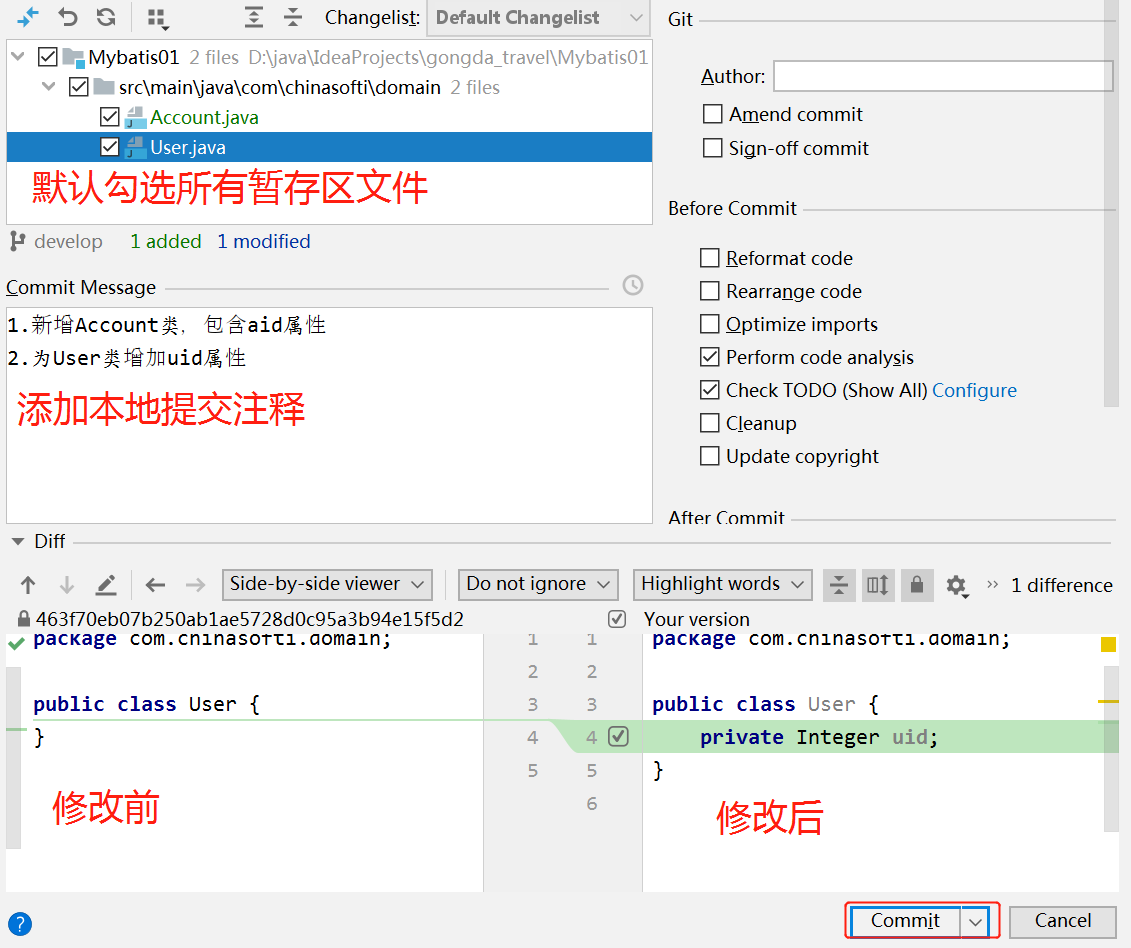


提交多个文件

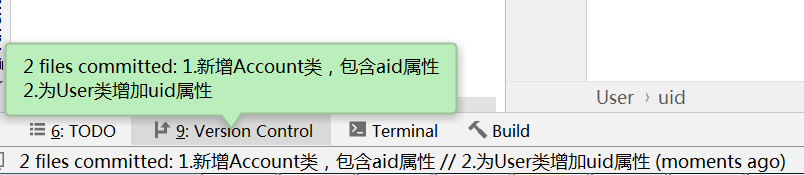




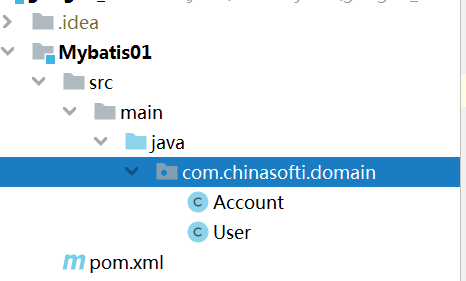
添加注释并提交



提交完成，IDEA弹出提交信息



提交到本地后颜色自动消失



### 4.2.6 推送到远程仓库

推送操作：

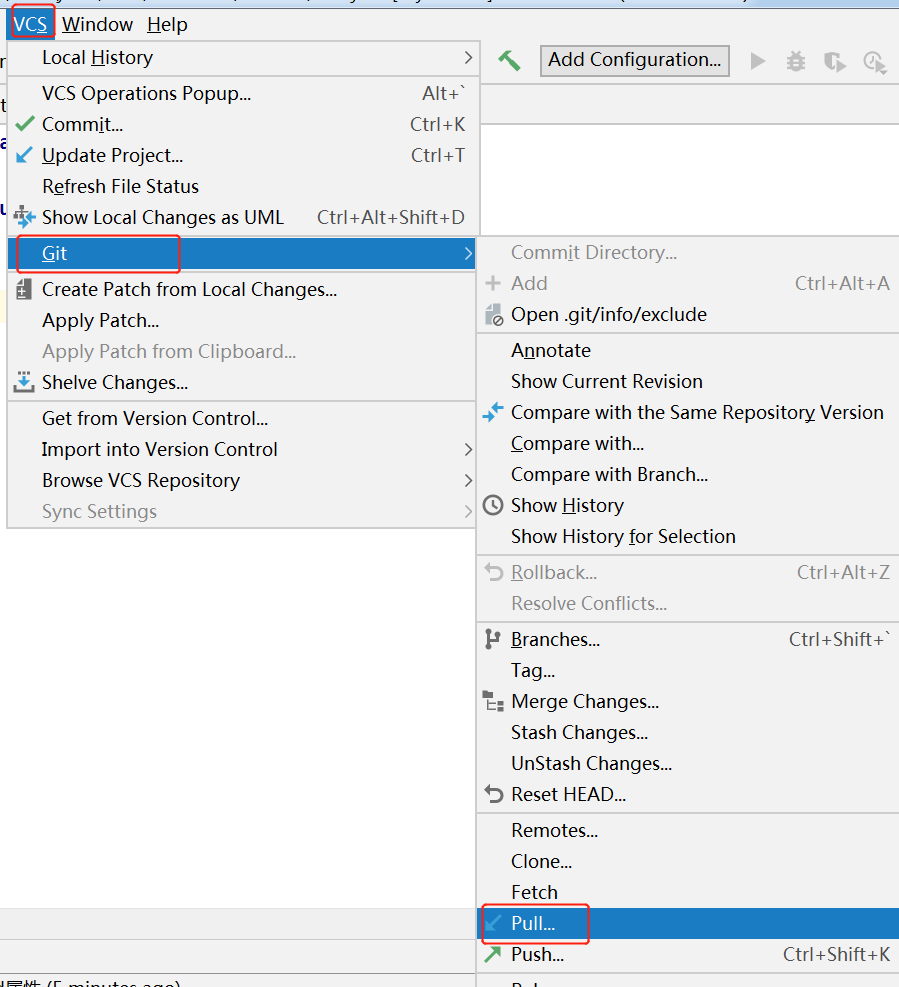
1.把工作区内容添加到暂存区；

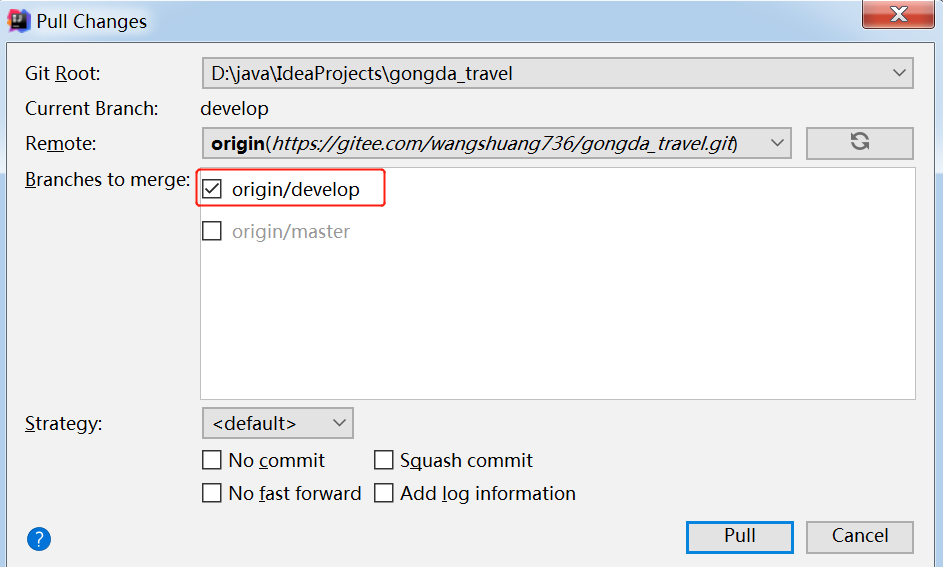
2.把暂存区内容提交到当前分支；

3.拉取远程仓库对应分支，如果有冲突，先解决冲突并提交保存

4.推送当前分支到远程仓库

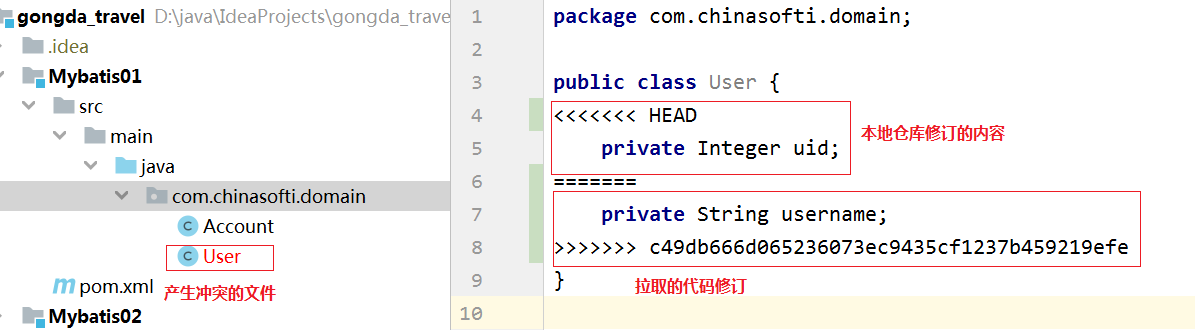
1）拉取远程仓库对应分支



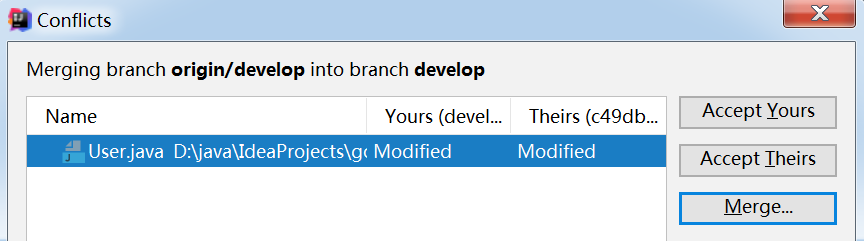


2）解决合并冲突

​ 如果从远程仓库拉取的代码中，同一文件在本地仓库也修改了，需要解决合并冲突，比如如图所示：

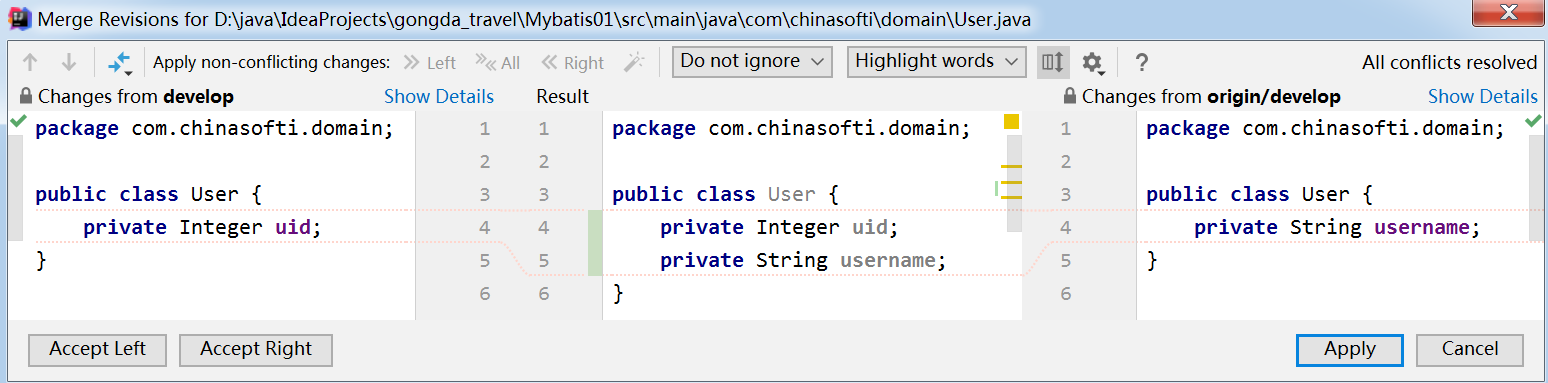


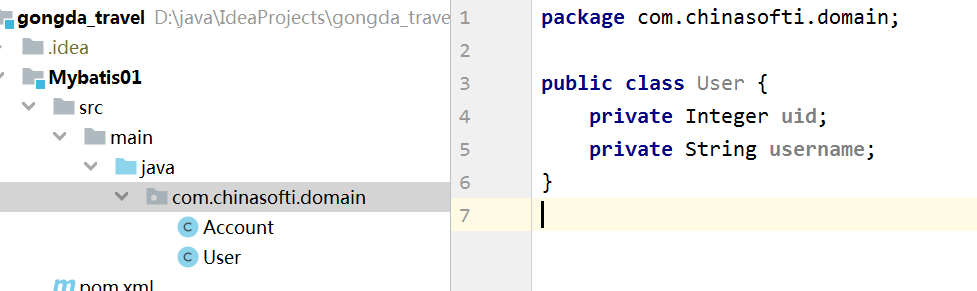
解决冲突（根据业务情况，进行代码合并，并删除冲突提示）



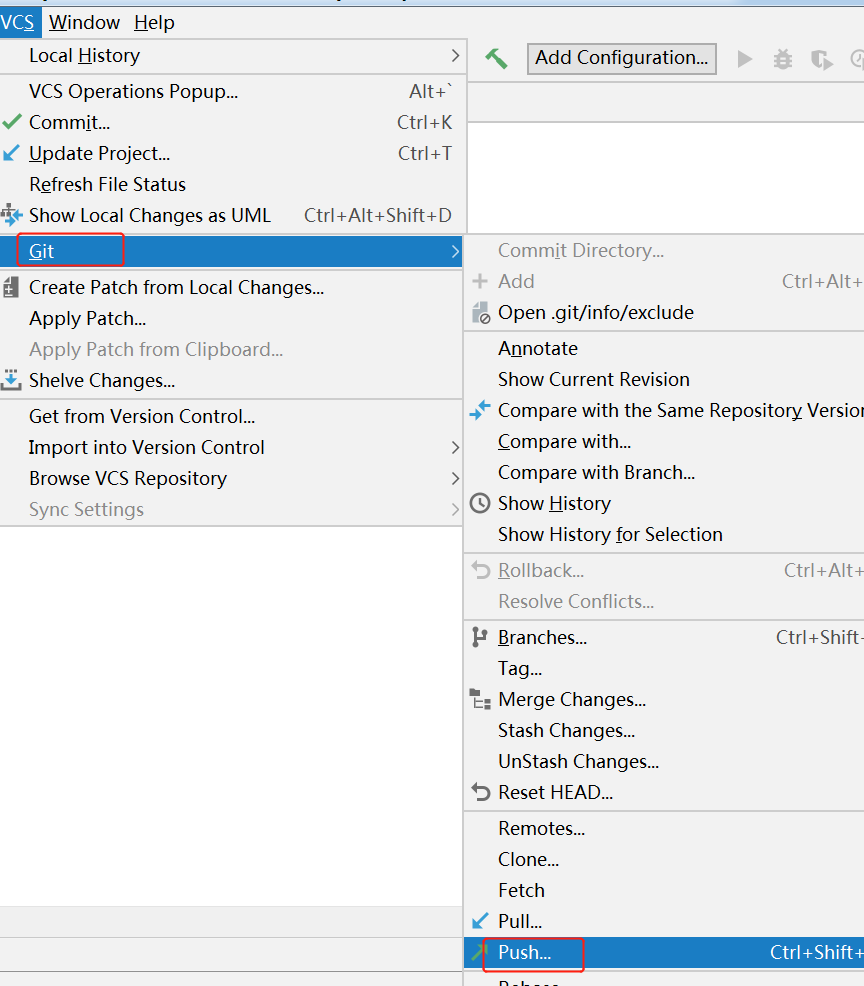
选择Merge，提交合并冲突

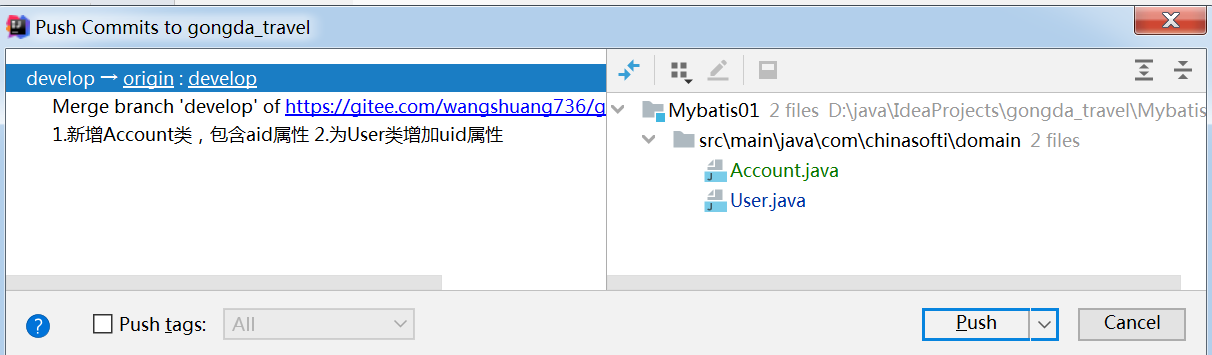


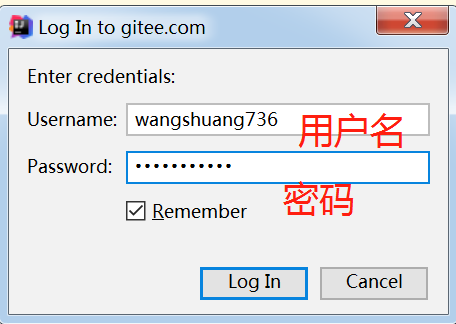


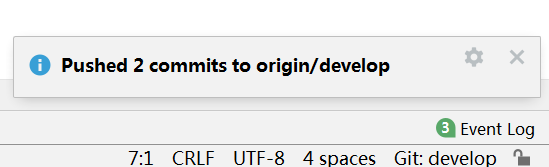


推送当前版本到远程仓库







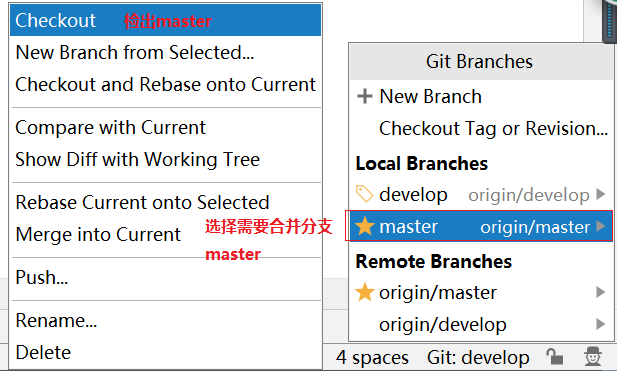


### 4.2.7 合并版本并推送到远程

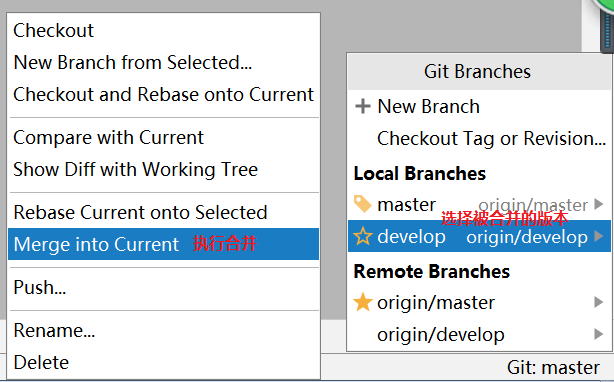
如果当前版本需要发布，需要把develop合并到master版本

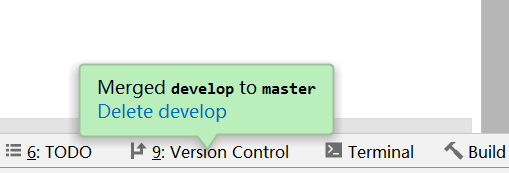
步骤：

1、切换到需要合并的版本，比如master

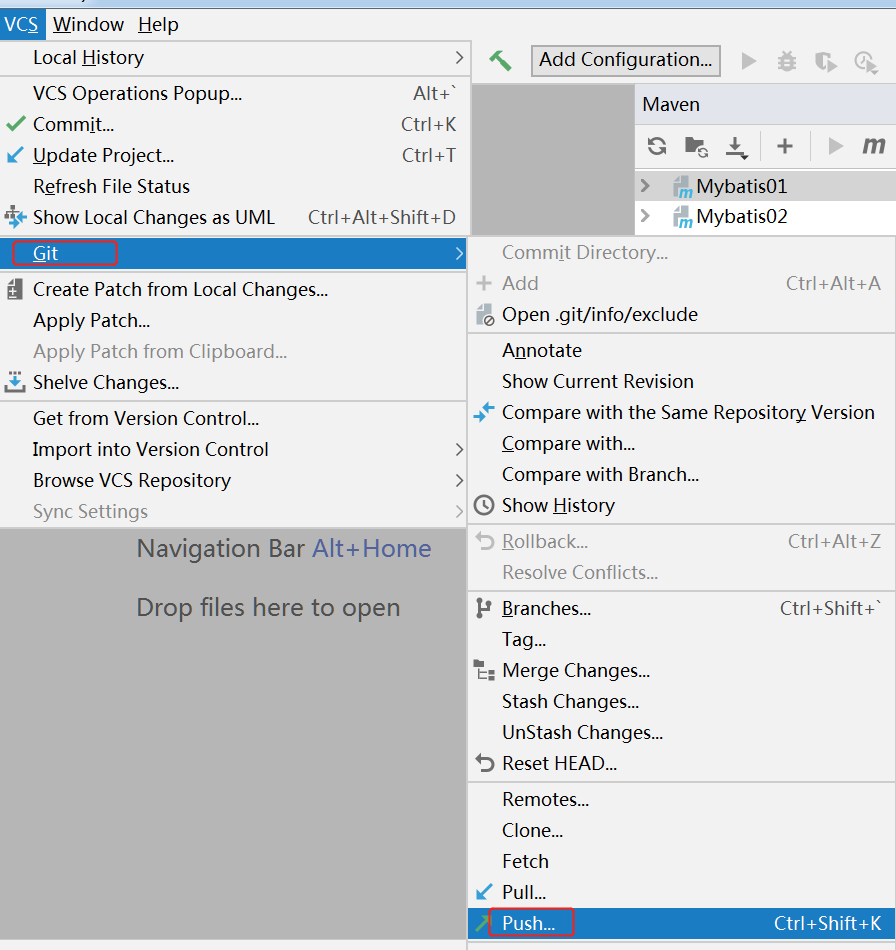


2、执行合并操作，使用当前版本合并其他版本到当前，比如master合并developer



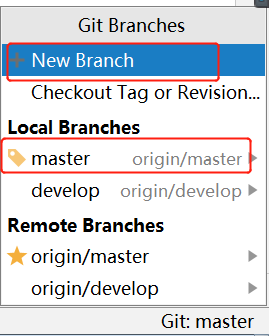


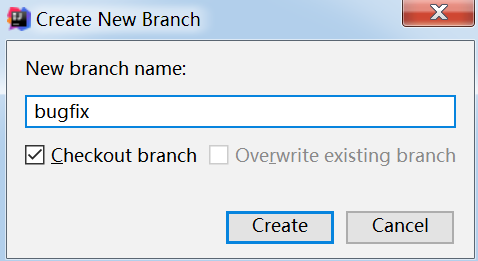
3、合并完成，推送到远程仓库



### 4.2.8 创建分支

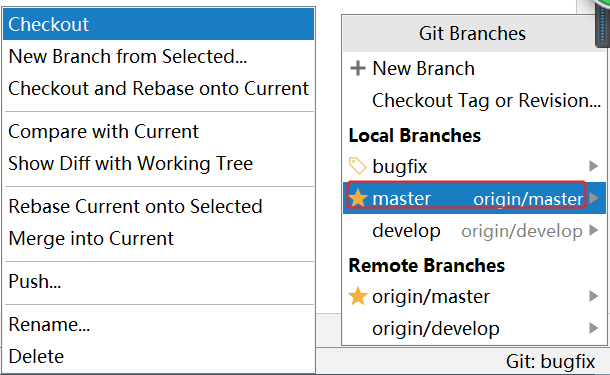
假如线上发生问题，需要解决线上问题，需要在master创建bugfix分支，解决线上问题。





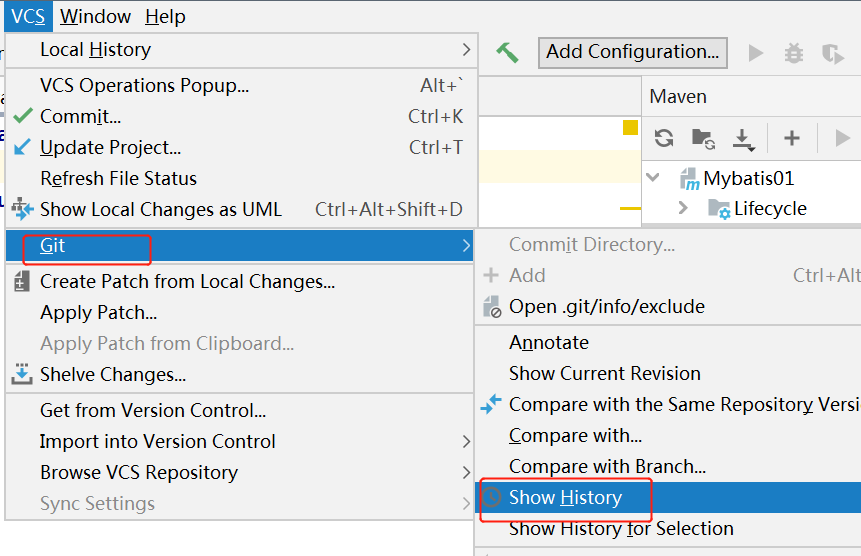
### 4.2.9切换分支

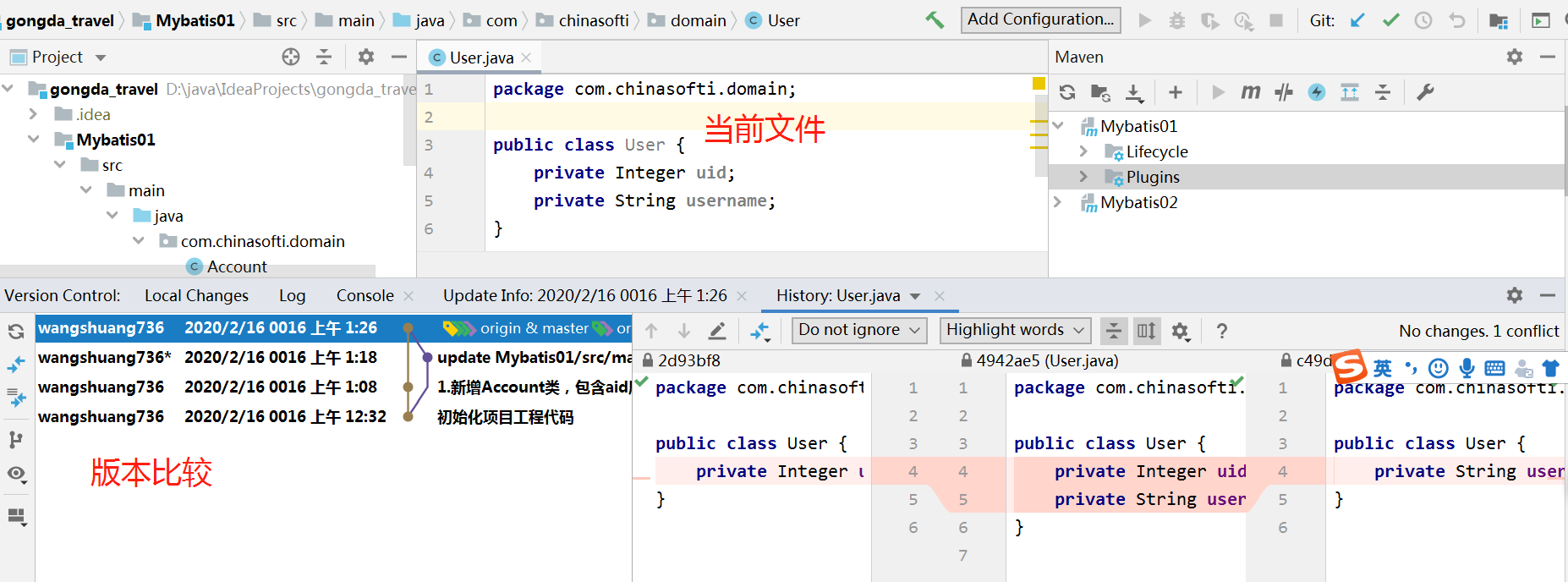
切换分支，只需要选中分支，然后选择checkout，就可以切换分支



### 4.2.10 版本比较

对代码修改后，可以点击对比按钮，对比差异





## 4.3 场景分析

基于我们后面的实战模式，我们做一个综合练习

​ 当前的开发环境如下，我们每个人都对这个项目已经开发一段时间，接下来我们要切换成团队开发模式。

也就是我们由一个团队来完成这个项目实战的内容。团队有组长和若干组员组成（组长就是开发中的项目经理）。

所有操作都在IDEA中完成。

​ 练习场景如下：

​ 1、由组长，基于本项目创建本地仓库；创建远程仓库，推送项目到远程仓库。（真实环境：项目经理创建空的项目推送，然后逐步添加内容，提交并推送）

​ 2、每一位组员从远程仓库克隆项目到idea中,这样每位同学在自己电脑上就有了一个工作副本，可以正式的开始开发了。我们模拟两个组员(组员A、组员B)，克隆两个工作区。

​ 3、组员A修改工作区，添加到暂存区，提交到本地仓库。如果需要分享代码给另一个组员，推送到远程仓库。组员B可以直接从远程仓库获取代码，组员B可以获取组员A推送的代码。

​ 4、组员A和组员B对同一个文件进行修改，提交代码到本地，这时需要分享代码给其他组员；组员A推送代码到远程仓库；组员B也推送代码到远程仓库，由于两位组员修改了同一个文件，推送失败。

​ 解决方法：需要先获取远程仓库的代码到本地仓库，编辑冲突，提交并推送代码。

​ 5、组员A接到组长命令，要对某功能进行重大的实验性调整；

​ 操作思路：创建分支B1， 切换到B1分支，在B1分支上进行代码调整。B1分支达到了预期的目标，可以合并到原来的主干上；B1分支没有达到预期的目标，可以继续调整，也可以直接删除分支，切换到主分支继续开发。如果其他组员对主干上代码进行过调整，在合并时会产生冲突，需先解决冲突，再提交并推送。