**版本控制**

## 一、VSS

**1 VSS 的全称为 Visual Source Safe 。作为 Microsoft Visual Studio 的一名成员**，它主要任务就是负责项目文件的管理，几乎可以适用任何软件项目。管理软件开发中各个不同版本的源代码和文档，占用空间小并且方便各个版本代码和文档的获取,对开发小组中对源代码的访问进行有效的协调。VSS作为一款历史悠久的版本管理工具，在早期扛起了版本管理系统方面的大气，能帮助解决一部分版本控制方面的问题，也在一定程度上帮助解决代码共享方面的难题。

**2 缺点**

（1）文件大多会以独占的形势进行锁定。如果一个人在修改的时候其他人没有办法进行修改。

（2）VSS只支持Windows版本，且只兼容微软的开发工具。

（3）文件存储，服务器必须共享文件夹，对文件的安全性没有足够保障。

## 二、SVN

**1 SVN是Subversion的简称，是一个开放源代码的版本控制系统**，相较于RCS、CVS，它采用了分支管理系统，它的设计目标就是取代CVS。互联网上很多版本控制服务已从CVS迁移到Subversion。说得简单一点SVN就是用于多个人共同开发同一个项目，共用资源的目的。SVN是一个开源的版本控制系统。和VSS相比，除开最基本的代码和文件管理功能外，主要的革新是提供了分支的功能，从而解决了VSS文件独占的问题。大幅提升了开发人员的工作效率，谁写完代码，随时可以提交到自己的分支上，最后对所有分支进行合并，

解决冲突即可。相比VSS而言，在工作模式上有了翻天覆地的改变。

**2 优点**

（1）管理方便，逻辑明确，操作简单，上手快。

（2）易于管理，集中式服务器更能保证安全性。

（3）代码一致性非常高。

（4）有良好的目录级权限控制系统。

**3 劣势**

（1）对服务器性能要求高，数据库容量经常暴增，体量大。

（2）必须联网。如果不能连接到服务器上，基本上不可以工作，如果服务器不能连接上，就不能提交，还原，对比等等。

（3）不适合开源开发。

（4）分支的管控方式不灵活

**4 简单的对比**

　　SVN的工作机制在某种程度上就像一颗正在生长的树：

一颗有树干和许多分支的树

分支从树干生长出来，并且细的分支从相对较粗的树干中长出

一棵树可以只有树干没有分支（但是这种情况不会持续很久，随着树的成长，肯定会有分支啦，^^）

一颗没有树干但是有很多分支的树看起来更像是地板上的一捆树枝

如果树干患病了，最终分支也会受到影响，然后整棵树就会死亡

如果分支患病了，你可以剪掉它，然后其他分支还会生长出来的哦！

如果分支生长太快了，对于树干它可能会非常沉重，最后整棵树会垮塌掉

当你感觉你的树、树干或者是分支看起来很漂亮的时候，你可以给它照张相，这样就就可以记得它在那时是多么的赞。

**5 Trunk**

　　Trunk是放置稳定代码的主要环境，就好像一个汽车工厂，负责将成品的汽车零件组装在一起。

　　以下容将告诉你如何使用SVN trunk：

除非你必须处理一些容易且能迅速解决的BUG，或者你必须添加一些无关逻辑的文件（比如媒体文件：图像，视频，CSS等等），否则永远不要在trunk直接做开发

不要因为特殊的需求而去对先前的版本做太大的改变，如何相关的情况都意味着需要建立一个branch（如下所述）

不要提交一些可能破坏trunk的内容，例如从branch合并

如果你在某些时候偶然间破坏了trunk，bring some cake the next day (”with great responsibilities come… huge cakes”)

**6 Branches**

一个branch就是从一个SVN仓库中的子树所作的一份普通拷贝。通常情况它的工作类似与UNIX系统上的符号链接，但是你一旦在一个SVN branch里修改了一些文件，并且这些被修改的文件从拷贝过来的源文件独立发展，就不能这么认为了。当一个branch完成了，并且认为它足够稳定的时候，它必须合并回它原来的拷贝的地方，

也就是说：如果原来是从trunk中拷贝的，就应该回到trunk去，或者合并回它原来拷贝的父级branch。

　　以下内容将告诉你如何使用SVN branches：

如果你需要修改你的应用程序，或者为它开发一个新的特性，请从trunk中创建一个新的branch，然后基于这个新的分支进行开发

除非是因为必须从一个branch中创建一个新的子branch，否则新的branch必须从trunk创建

当你创建了一个新branch，你应当立即切换过去。如果你没有这么做，那你为什么要在最初的地方创建这个分支呢？

**7 Tags**

　　从表面上看，SVN branches和SVN tags没有什么差别，但是从概念上来说，它们有许多差别。其实一个SVN tags就是上文所述的“为这棵树照张相”：一个trunk或者一个branch修订版的命名快照。

　　以下内容将告诉你如何使用SVN tags：

作为一个开发者，永远不要切换至、取出，或者向一个SVN tag提交任何内容：一个tag好比某种“照片”，并不是实实在在的东西，tags只可读，不可写。

在特殊或者需要特别注意的环境中，如：生产环境（production）、？（staging）、测试环境（testing）等等，只能从一个修复过的（fixed）tag中checkout和update，永远不要commit至一个tag。

对于上述提及到的环境，可以创建如下的tags：“production”，“staging”，“testing”等等。你也可以根据软件版本、项目的成熟程度来命名tag：“1.0.3”，“stable”，“latest”等等。

当trunk已经稳定，并且可以对外发布，也要相应地重新创建tags，然后再更新相关的环境（production, staging, etc）

**8 工作流样例**

　　假设你必须添加了一个特性至一个项目，且这个项目是受版本控制的，你差不多需要完成如下几个步骤：

使用SVN checkout或者SVN switch从这个项目的trunk获得一个新的工作拷贝（branch）

使用SVN切换至新的branch

完成新特性的开发（当然，要做足够的测试，包括在开始编码前）

一旦这个特性完成并且稳定（已提交），并经过你的同事们确认，切换至trunk

合并你的分支至你的工作拷贝（trunk），并且解决一系列的冲突

重新检查合并后的代码

如果可能的话，麻烦你的同事对你所编写、更改的代码进行一次复查（review）

提交合并后的工作拷贝至trunk

如果某些部署需要特殊的环境（生成环境等等），请更新相关的tag至你刚刚提交到trunk的修订版本

使用SVN update部署至相关环境

## 三、Git

**1 Git是一款免费、开源的分布式版本控制系统**，用于敏捷高效地处理任何或小或大的项目。作为一个开源的分布式版本控制系统，可以有效、高速的处理从很小到非常大的项目版本管理。分布式相比于集中式的最大区别在于开发者可以提交到本地，每个开发者通过克隆（git clone），在本地机器上拷贝一个完整的Git仓库。git(分布式版本控制系统)，每台电脑有自己的版本库

（仓库，repository），‘中央服务器’方便交换修改，安装后需设置名字和Email地址。

**2 Git的优缺点如下**

（1）适合分布式开发，每一个个体都可以作为服务器。每一次Clone就是从服务器上pull到了所有的内容，包括版本信息。

（2）公共服务器压力和数据量都不会太大。

（3）速度快、灵活，分支之间可以任意切换。

（4）任意两个开发者之间可以很容易的解决冲突，并且单机上就可以进行分支合并。

（5）离线工作，不影响本地代码编写，等有网络连接以后可以再上传代码，并且在本地可以根据不同的需要，本地新建自己的分支。

**3 创建版本库**

（1）mkdir learngit #创建空目录，即工作区

（2）git init #初始化形成git可管理的仓库，.git目录是Git跟踪管理版本库的

（只能跟踪文本文件的改动，图像、视频等二进制文件不能跟踪变化）

**4 版本库文件操作**

（1）git add readme.txt #把文件添加到Git仓库 （添加到暂存区Stage）

（2）git commit -m "a readme file" #把文件提交到仓库，-m指定说明文字（把Stage内容提交到当前分支）

（3）git status #查看仓库当前状态

（4）git diff readme.txt #查看修改

（5）git log #查看历史改动记录（git log --pretty=online #查看精简记录）

（6）git reset --hard HEAD^ #版本回退（上上版本为HEAD^ 往上100版本为HEAD~100，HEAD指定当前版本）（git reset --hard 3628164(版本号，前几位即可)）hard含义

（7）git reflog #查看每次的命令，可显示commit id，结合版本回退

（8）git checkout -- readme.txt #撤销修改，回到最近一次git commit或git add时的状态（git reset HEAD readme.txt # 可以撤销暂存区的修改放回工作区）

（9）rm readme.txt #删除工作区文件

1） git rm readme.txt #从版本库中删除文件 git commit #提交删除操作

2）git checkout -- readme.txt #恢复删除（git checkout 用版本库的版本替换工作区的版本）

**5 远程仓库：**

（1）在GitHub上创建新的仓库

（2）git remote add origin git@github.com:\*/learngit.git #把本地仓库关联到Github仓库，远程库的名称是origin

（3）git push -u origin master #把本地库内容推送到远程库上（第一次推送加-u,实现关联和推送，之后直接使用git push origin master）

（4） git clone git@github.com:\*/learngit.git( Git支持多种协议，包括https，但通过ssh支持的原生git协议速度最快)

**6 分支管理：(master分支是主分支，HEAD指向master，master指向提交)**

（1）git checkout -b dev #创建新的分支dev并切换到dev

1)git branch dev #创建dev分支

2)git checkout dev #切换到dev分支）

（2）git branch #查看当前分支

（3）git checkout master #切换到master分支

1）git merge dev #合并分支成果（Fast forward模式）

2）git merge --no-ff -m "merge with no-ff" dev #禁用Fast forward模式的合并 ，会生成一个新的commit

（4）git branch -d dev #删除分支

（5）git log --graph --pretty=online --abbrev-commit #查看合并情况，git log --graph可以看到分支合并图

**7 bug分支：**

（1）git stash #储存当前工作现场（git checkout mater #确定出bug的分支

git checkout -b issue-101 #在出bug的分支创建临时分支

git checkout master #修复完成切换到原分支

git merge –no-ff -m “merged bug fix 101” issue-101 #合并分支

git branch -d issue -101） #删除分支

（2）git stash list #查看存储的现场

（3）git stash pop #恢复stash内容并删除

等价于：

1）git stash apply #恢复内容

2）git stash drop #删除stash）

**8 feature分支**

（1）git branch -D feature-vulcan #强制删除一个没有被合并过的分支

（2）git remote #查看远程库信息（git remote -v #查看远程库详细信息）

（3）git push origin master #推送指定分支

（4）git checkout -b dev origin/dev #创建远程origin的dev分支到本地

（5）git pull #把最新的提交抓取下来

（6）git branch --set-upstream dev origin/dev #设置dev和origin/dev的链接

**9 标签管理**

（1）git tag v1.0 #对当前分支打标签，默认打在最新提交的commit上

git tag v0.9 6224937 #对指定commit打标签

git tag -a v0.1 -m “v0.1 released” 2628164 #-a指定标签名，-m指定说明文字

git tag -s v0.2 -m “v0.2 released” fec145a #-s用私钥签名一个标签,采用PGP签名，需安装gpg）

（2）git tag #查看所有标签

（3）git show v0.9 #查看标签信息

（4）git tag -d v0.1 #删除本地的标签信息

（5）git push origin :refs/tags/v0.9 #删除远程标签信息

（6）git push origin v1.0 #推送标签到远程（git push origin --tags #一次性推送全部尚未推送到远程的本地标签）

**10 git/gitlab/github**

**git**是一个版本管理工具，是可以在你电脑不联网的情况下，只在本地使用的一个版本管理工具，其作用就是可以让你更好的管理你的程序，比如你原来提交过的内容，以后虽然修改了，但是通过git这个工具，可以把你原来提交的内容重现出来，这样对于你后来才意识到的一些错误的更改，可以进行还原。

关于github，这是一个网站，就是每个程序员自己写的程序，可以在github上建立一个网上的仓库，你每次提交的时候可以把代码提交到网上，这样你的每次提交，别人也都可以看到你的代码，同时别人也可以帮你修改你的代码，这种开源的方式非常方便程序员之间的交流和学习。

**github**是一个非常适合程序员交流的网站，很多国际上的技术大牛都在github上有自己的开源代码，其他人只要申请个账号就可以随意的看到这些大牛写的程序。同时国内的很多互联网公司如百度，阿里等，也在github上公布有开源的代码，感兴趣的程序员也可以自己查找着看一些。想看国内大公司的代码，可以看连接：https://www.zhihu.com/question/20194937

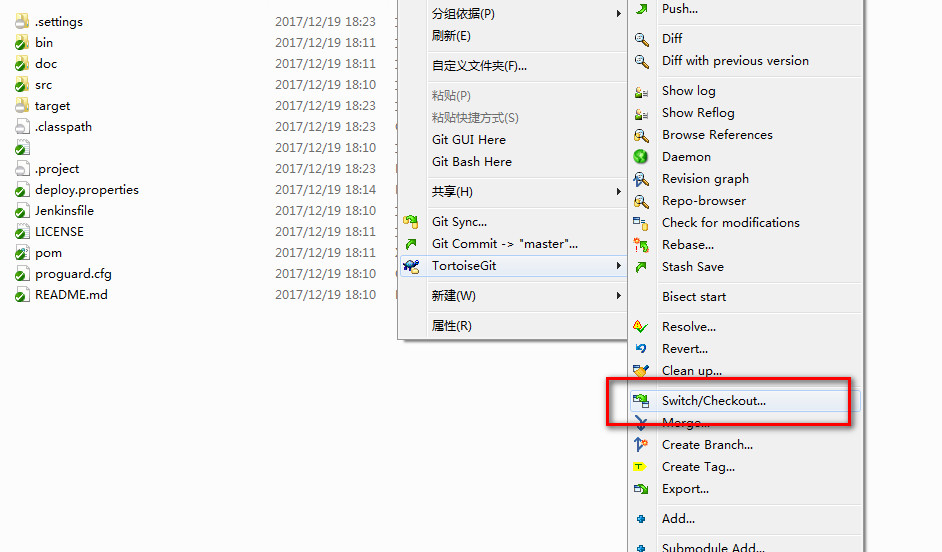
总结来说，git可以认为是一个软件，能够帮你更好的写程序，github则是一个网站，这个网站可以帮助程序员之间互相交流和学习。

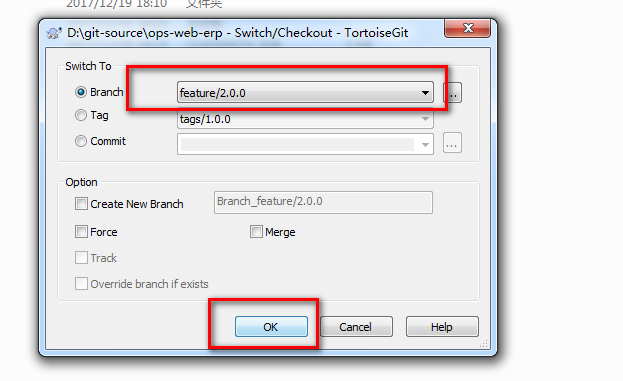
**github，gitlab等产品都是第三方基于git这项技术开发的，github是程序员交流的网站，gitlab一般是公司内网搭建的。**

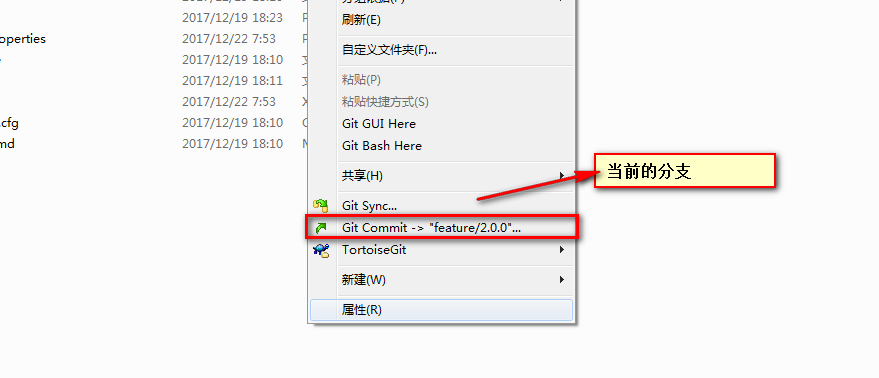
**11 git客户端合并代码操作**

**说明：2.0发布需要拉出一个realease/2.0.0 分支，先从feature2.0.0同步到dev,再从dev同步到realease/2.0.0, 构建和部署realease/2.0.0环境，运营验证问题，修复的代码提交到realease/2.0.0，最后上线那一天，先从realease/2.0.0合并代码到dev ,再从realease/2.0.0合并代码到master.**

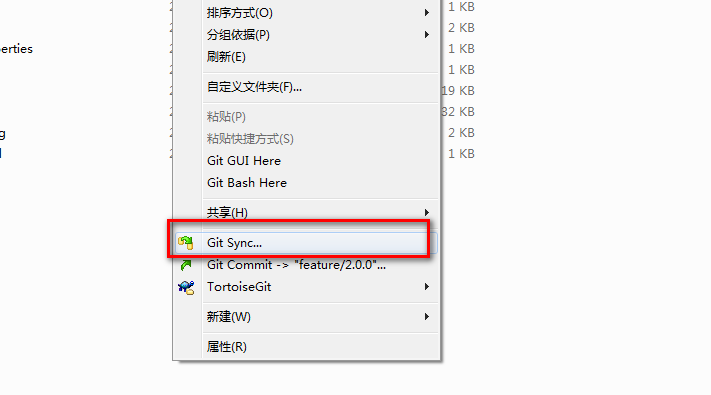
## 1 git客户端，切换检出，选择feature2.0，确定

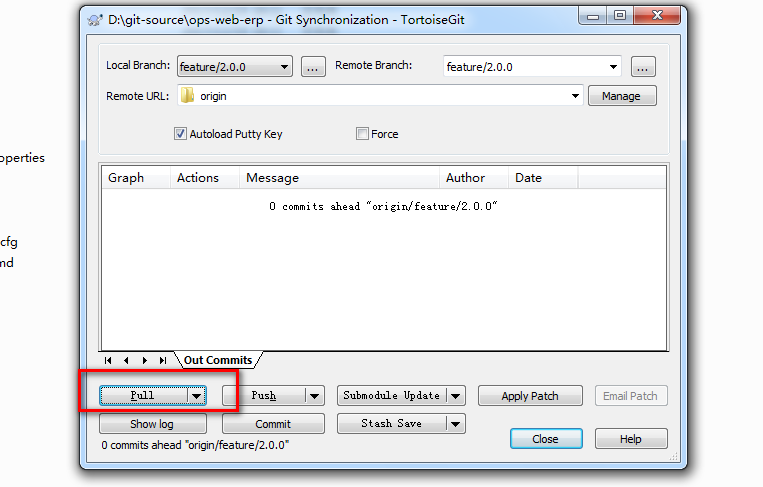






## 2 git同步，pull ,保证本地和远程都是最新的一致





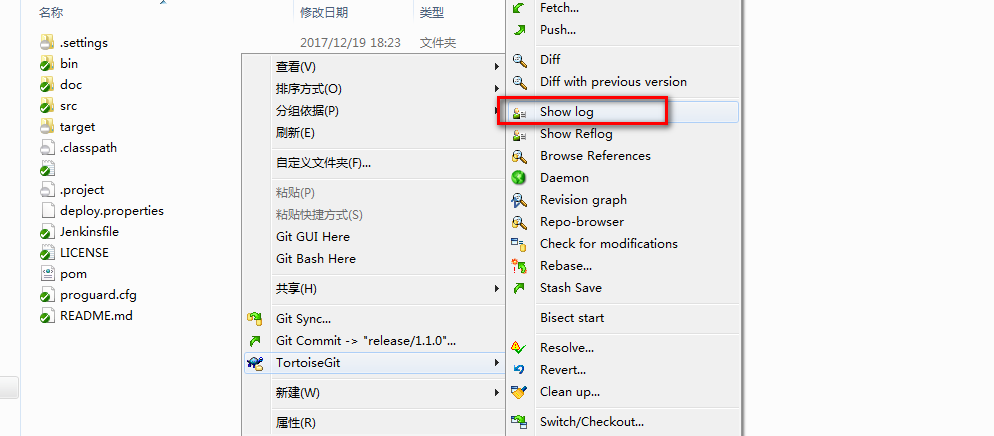
## 3 git客户端，切换检出，选择dev ，确定

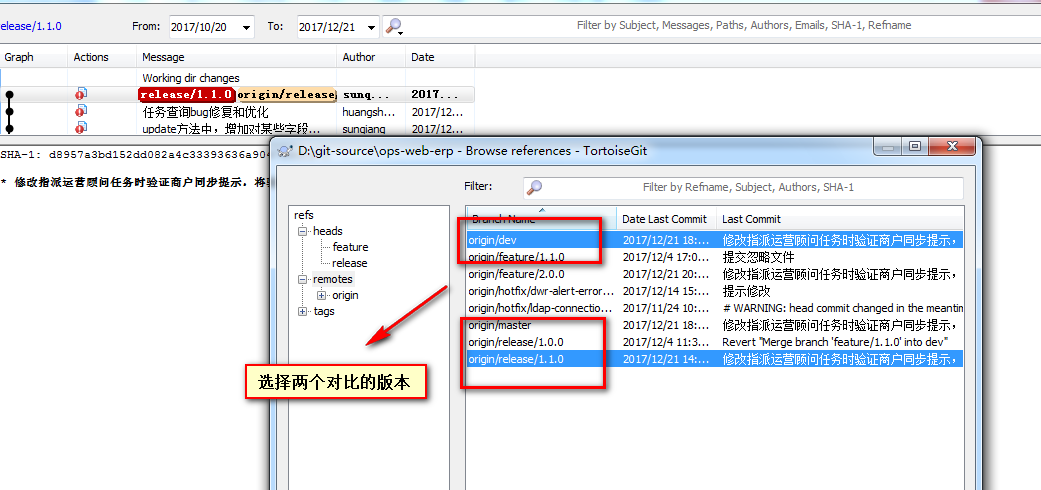
同第1步处理方式

## 4 git同步，pull,保证本地和远程都是最新的一致

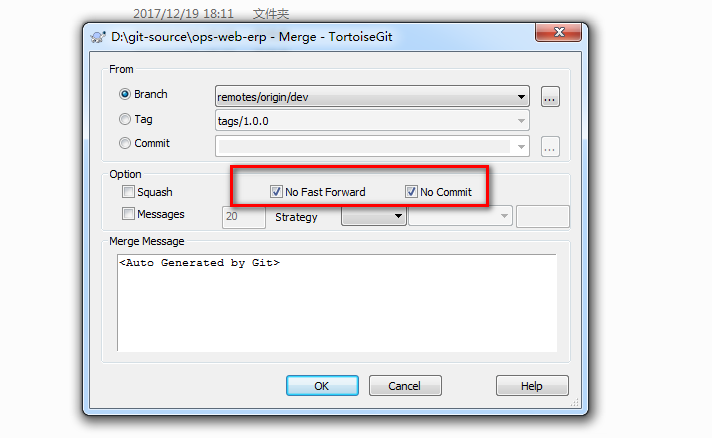
同第2步处理方式

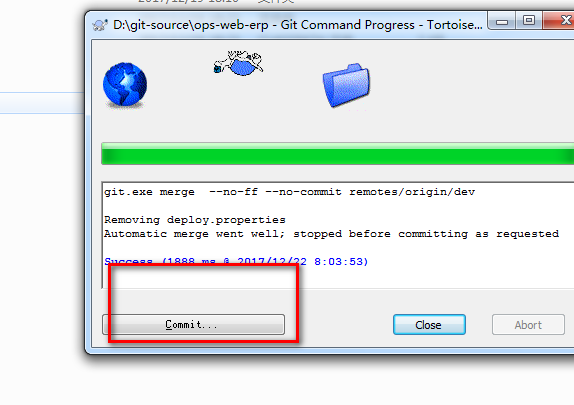
## 5 显示日志，比对版本的不同

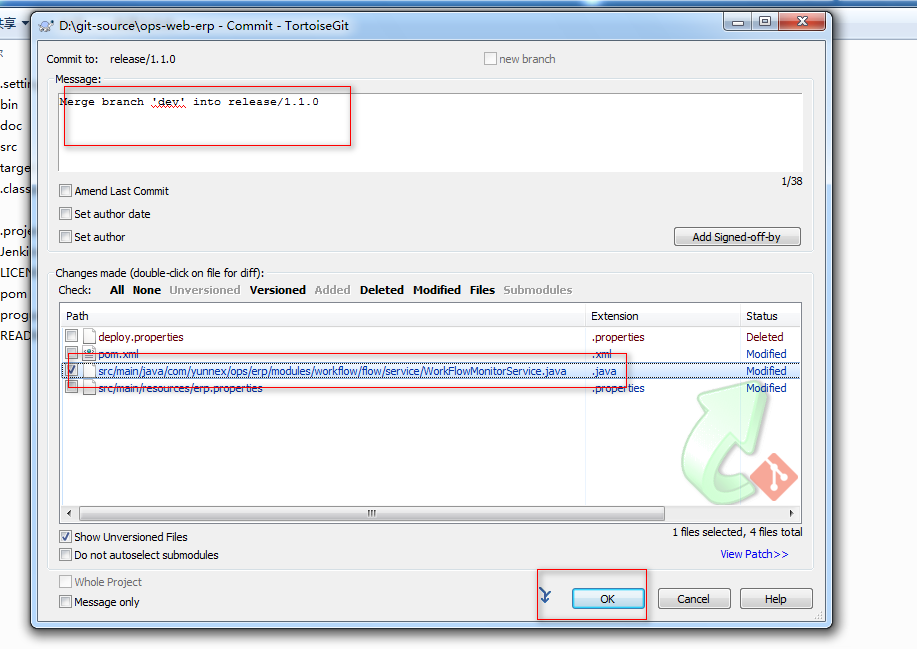




## 6 右键，合并，选择No Commit,No Fast Forward,点击commit，通过多选框选择要提交的文件，针对特殊的4个文件要认真比对

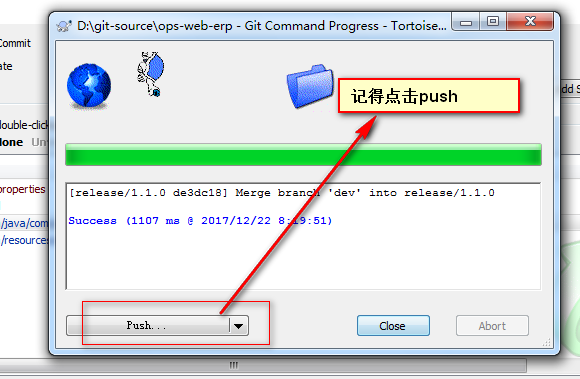




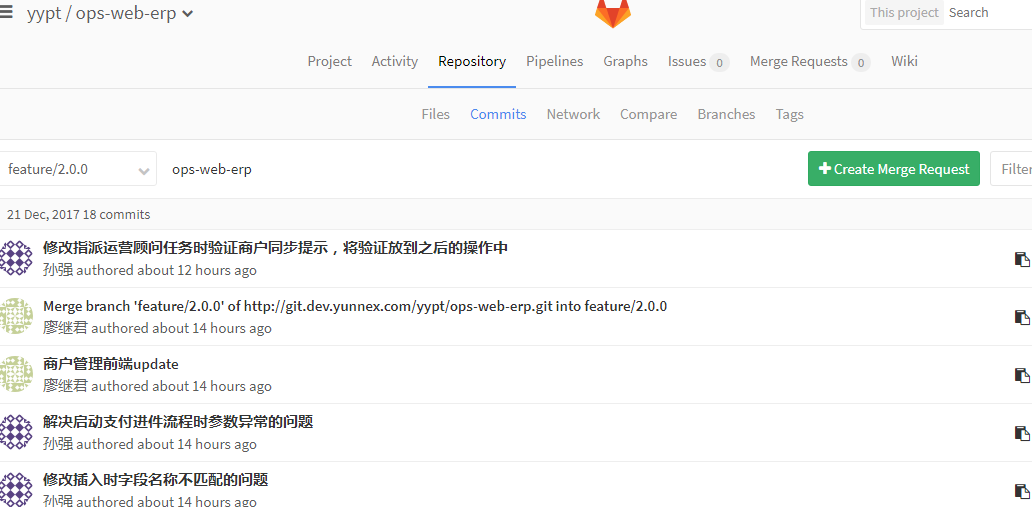


选择需要合并的文件

## 7 点击push，推送到远程



## 8 查看commit logs核对提交日志,通过网页或者git客户端查看



## 9 jenkins部署，检查生产包



<http://git.ops.yunnex.com/deployment/packages>



## 10 发邮件给运维，跟踪上线



## 11 Git客户端记住密码

打开项目中.git文件夹的config文件，添加

[credential]

     helper = store

命令，然后在上传代码时再输入一次用户名与密码，就可以保存输入信息

## 12 需要注意的几个文件

erp.properties 生产，研发，测试环境配置的不一样，要比对清楚在合并，如果需要新加了内容需要手动合并，把新加的配置项提交。

deploy.properties 这个文件一般不用合并

log4j.properties 生产环境级别是info，不用合并

SystemAuthorizingRealm.java 生产环境验证了ldap账号，测试换环境放开了，注意选择文件时去除这个文件

