Git及TortoiseGit使用说明文档

# 引言

## Git和SVN区别

Git 不仅仅是个版本控制系统，它也是个内容管理系统(CMS)，工作管理系统等。

如果你是一个具有使用 SVN 背景的人，你需要做一定的思想转换，来适应 Git 提供的一些概念和特征。

Git 与 SVN 区别点：

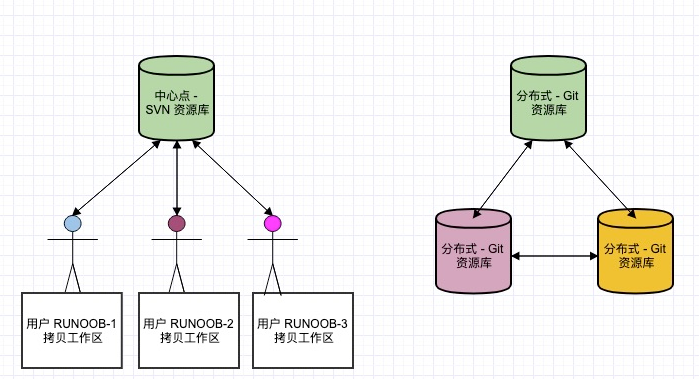
1、**Git 是分布式的**，SVN 是集中式的：这是 Git 和其它非分布式的版本控制系统，例如 SVN，CVS 等，最核心的区别。因为 Git 是分布式的，所以 Git 支持离线工作，在本地可以进行很多操作，而 SVN 必须联网才能正常工作。。

2、Git 把内容按元数据方式存储，而 SVN 是按文件：所有的资源控制系统都是把文件的元信息隐藏在一个类似 .svn、.cvs 等的文件夹里。

3、Git 分支和 SVN 的分支不同：Git 分支是指针指向某次提交，而 SVN 分支是拷贝的目录。这个特性使 Git 的分支切换非常迅速，且创建成本非常低。

4、Git 没有一个全局的版本号，而 SVN 有：目前为止这是跟 SVN 相比 Git 缺少的最大的一个特征。

5、Git 的内容完整性要优于 SVN：Git 的内容存储使用的是 SHA-1 哈希算法。这能确保代码内容的完整性，确保在遇到磁盘故障和网络问题时降低对版本库的破坏。

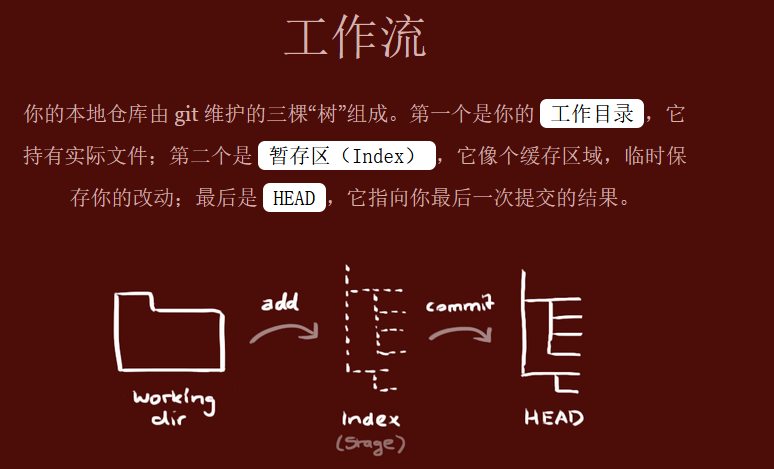


## Git核心概念

工作区(Workspace)是电脑中实际的目录。

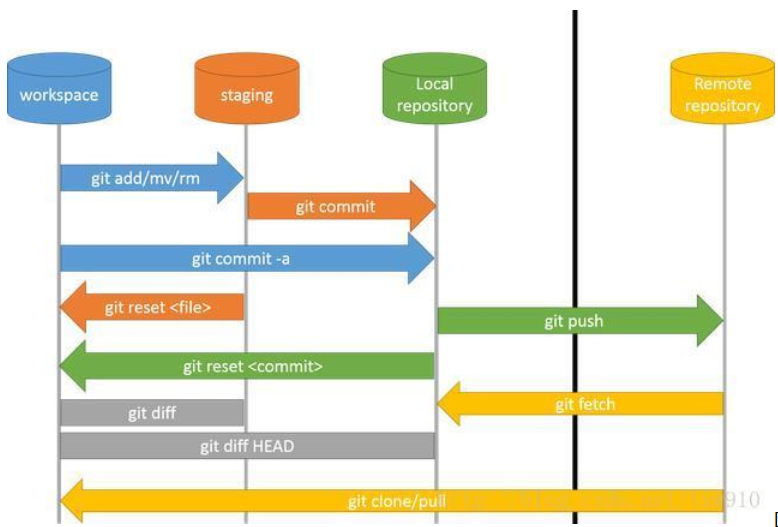
暂存区(Index)类似于缓存区域，临时保存你的改动。

仓库区(Repository)，分为本地仓库和远程仓库。



通常提交代码分为几步：

1. git add从工作区提交到暂存区
2. git commit从暂存区提交到本地仓库
3. git push或git svn dcommit从本地仓库提交到远程仓库



## Git 工作流程

一般工作流程如下：

克隆 Git 资源作为工作目录。

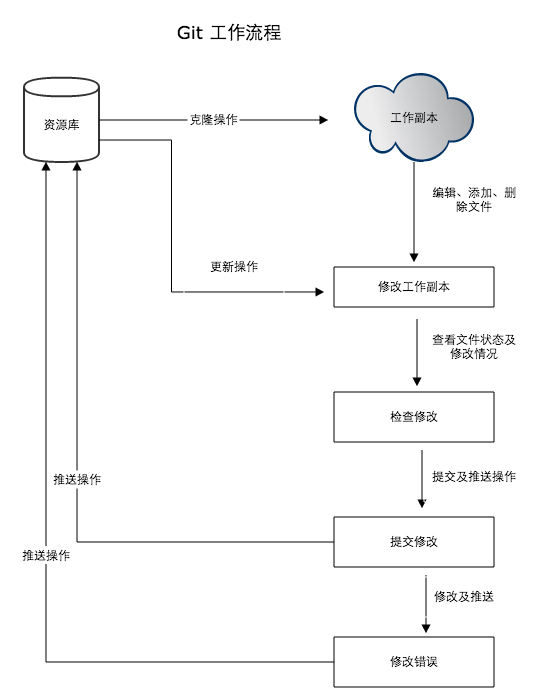
在克隆的资源上添加或修改文件。

如果其他人修改了，你可以更新资源。

在提交前查看修改。

提交修改。

在修改完成后，如果发现错误，可以撤回提交并再次修改并提交。



# 准备工作1：Git配置

## Gitlab账号注册

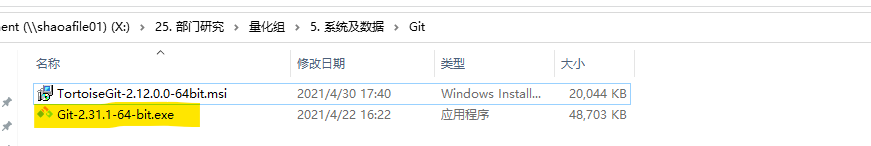
Gitlab地址：<http://10.198.60.61:8010/>

到该网站使用邮箱注册一个个人账户

## Git的安装和配置（无需管理员权限）：

1. **安装包地址**

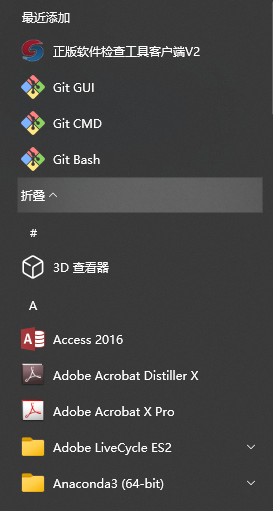
\25. 部门研究\量化组\5. 系统及数据\Git\Git-2.31.1-64-bit.exe



（参考资料 <https://zhuanlan.zhihu.com/p/123195804>， 注意：生成SSH步骤暂且跳过）

安装时相关参数勾选为默认即可。

安装完成后，右键菜单栏会出现Git GUI和Git Bash



1. **Git与Gitlab相联系**

Git是一个分布式版本控制系统，而Gitlab是一个集成了git的服务，它可以以网页或者客户端的形式，帮助用户把git本地的数据提交到远程的服务器里。

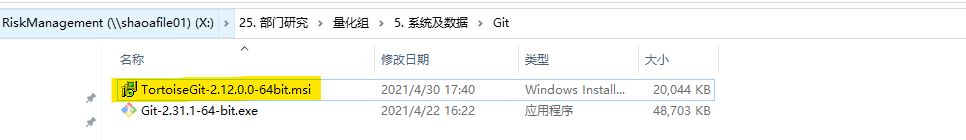
**配置Gitlab用户名和邮箱**

从开始菜单打开Git bash，配置Gitlab用户名和账号（需要先注册好Gitlab账号）

|  |
| --- |
| *# 配置用户名*  **git config --global user.name "username"**  **//（ "username"是自己的账户名）**  *# 配置邮箱*  **git config --global user.email "username@email.com"**  *//("username@email.com"注册账号时用的邮箱)*  配置完成后输入  **git config --global --list**  来查看配置的结果  https://pic3.zhimg.com/80/v2-471336d64983b20cfd3f08389aef35e6_720w.png |

# 准备工作2：TortoiseGit安装及配置

安装包地址：25. 部门研究\量化组\5. 系统及数据\Git\ TortoiseGit-2.12.0.0-64bit.msi



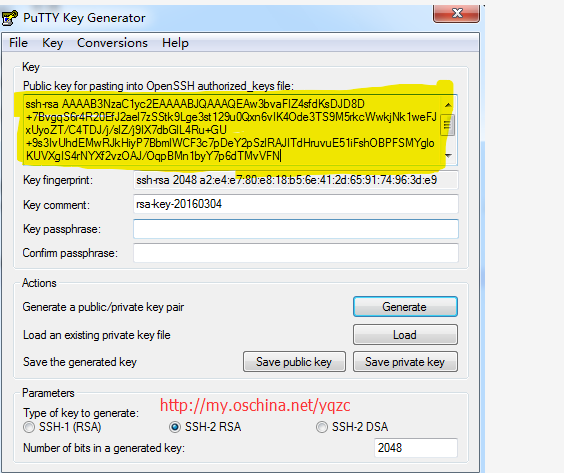
1. 安装步骤参考如下链接(需要管理员权限， 安装到“完成”即可，密钥配置步骤参考下一个链接)

<https://blog.csdn.net/xc_zhou/article/details/93137856>

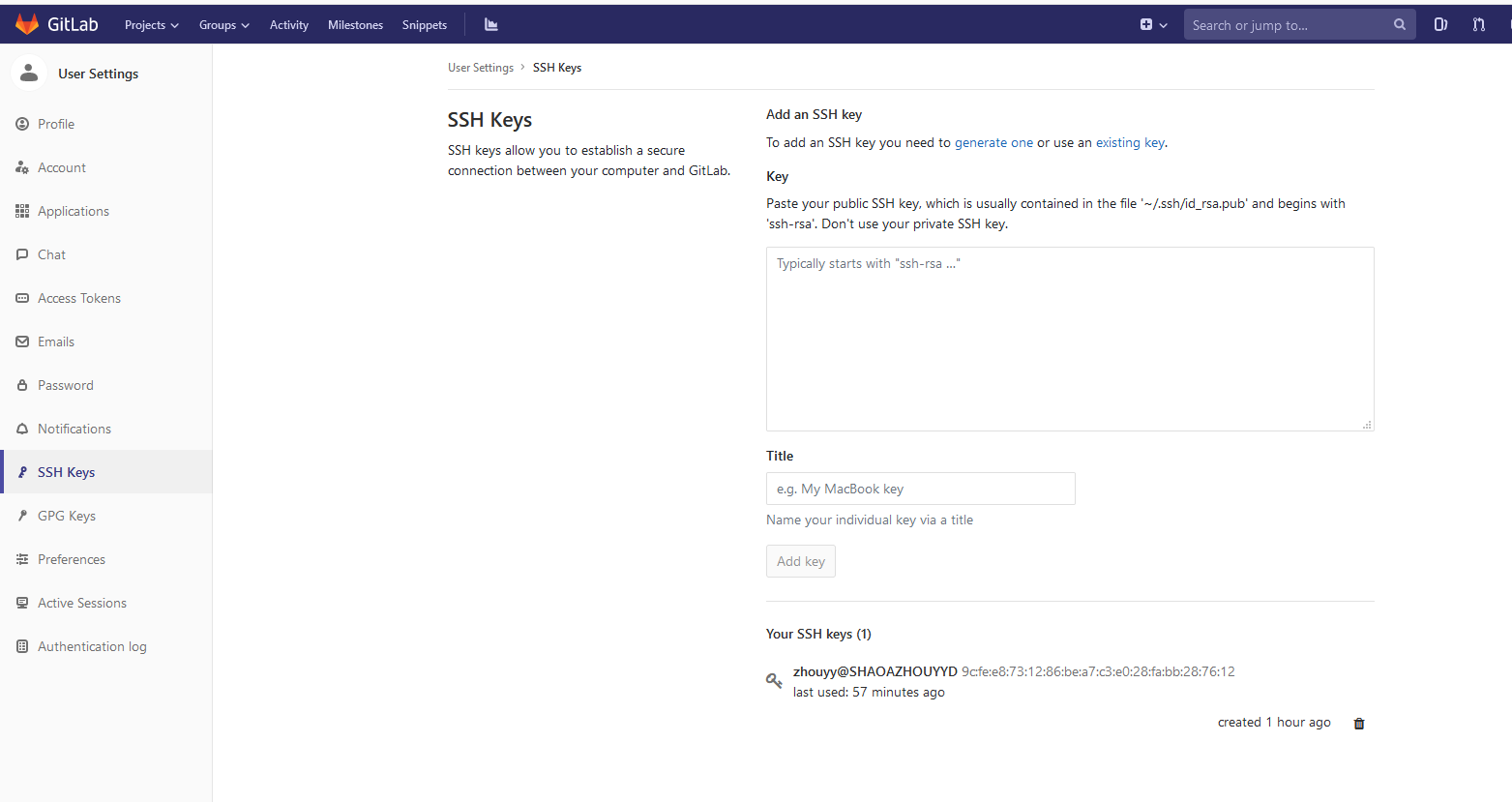
1. 安装完成后通过如下方式进行秘钥配置

<https://www.cnblogs.com/kid526940065/p/8574697.html>

1. 配置好后将步骤2过程中生成的公钥添加到Gitlab，步骤如下：



标黄部分全部复制，粘贴到GitHub管理平台中，在个人账户的设置中找到如下界面



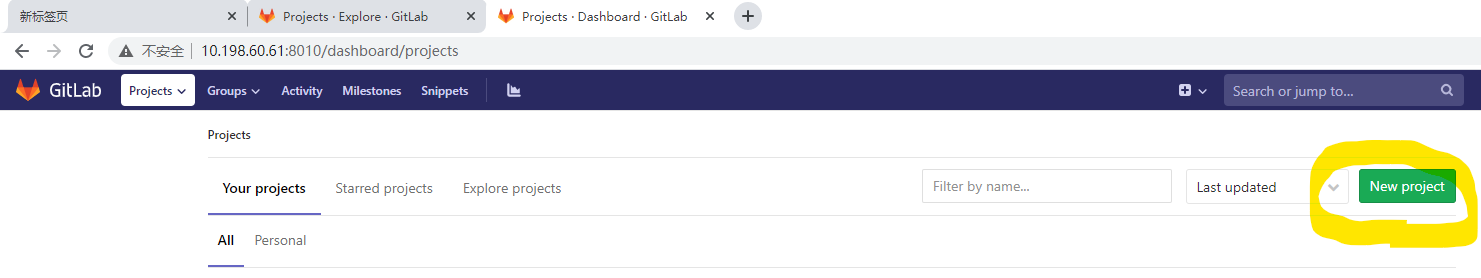
在Key窗口粘贴，添加成功如下所示



（为什么GitHub需要SSH Key呢？因为GitHub需要识别出你推送的提交确实是你推送的，而不是别人冒充的，而Git支持SSH协议，所以，GitHub只要知道了你的公钥，就可以确认只有你自己才能推送。）

# GitLab使用

## 在GitLab新建项目

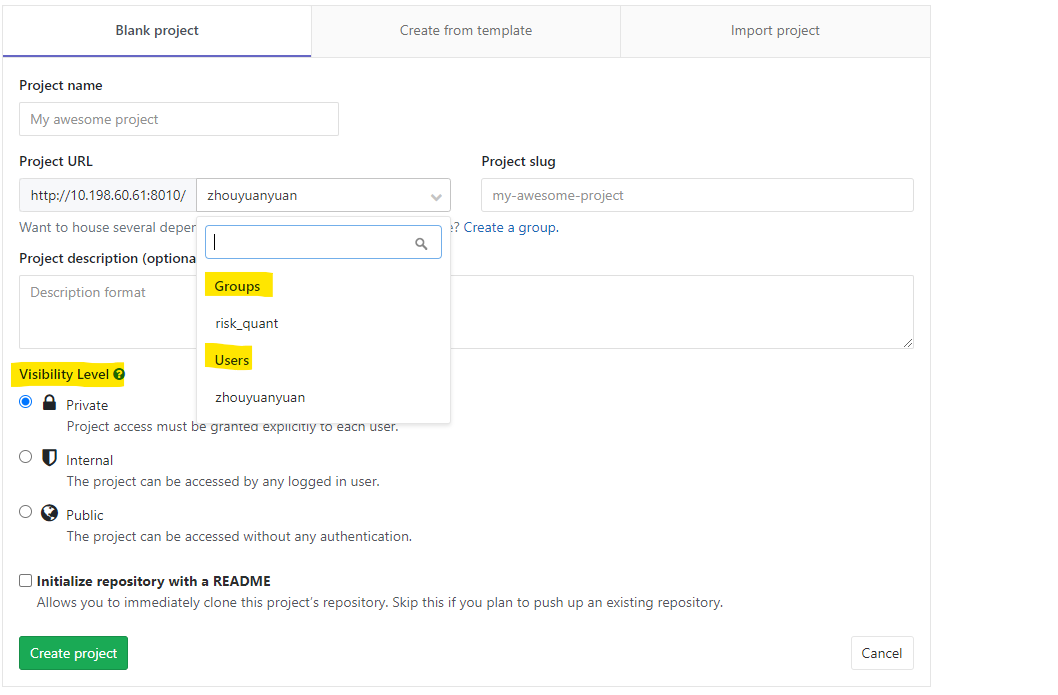
****

创建时可以选择在自己用户下创建或者某个群组内创建，并设置可见性

(私有库：只有被赋予权限的用户可见

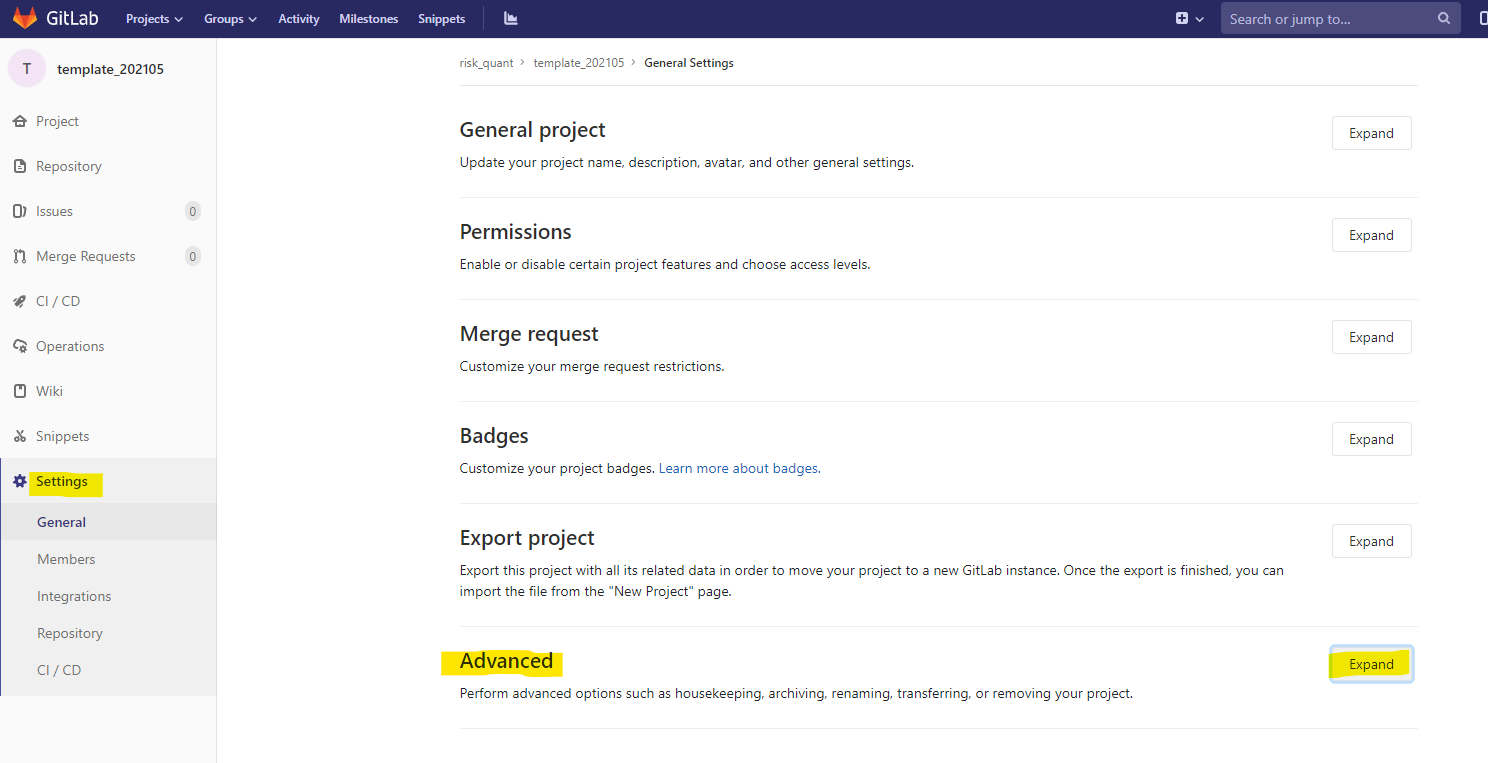
内部库：登录用户可以下载

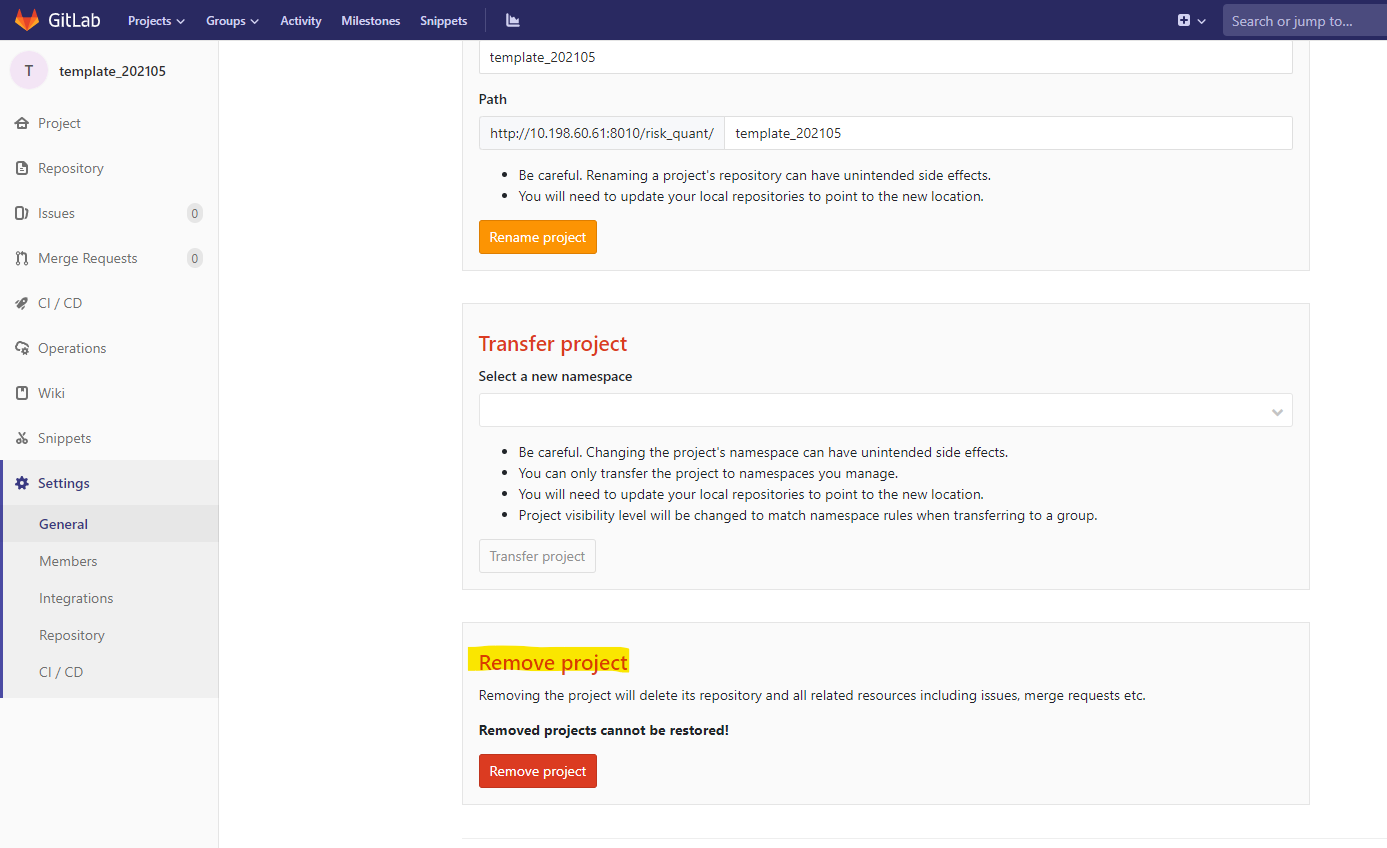
公开库：所有人可以下载)

****

## 编辑或删除项目（谨慎删除）

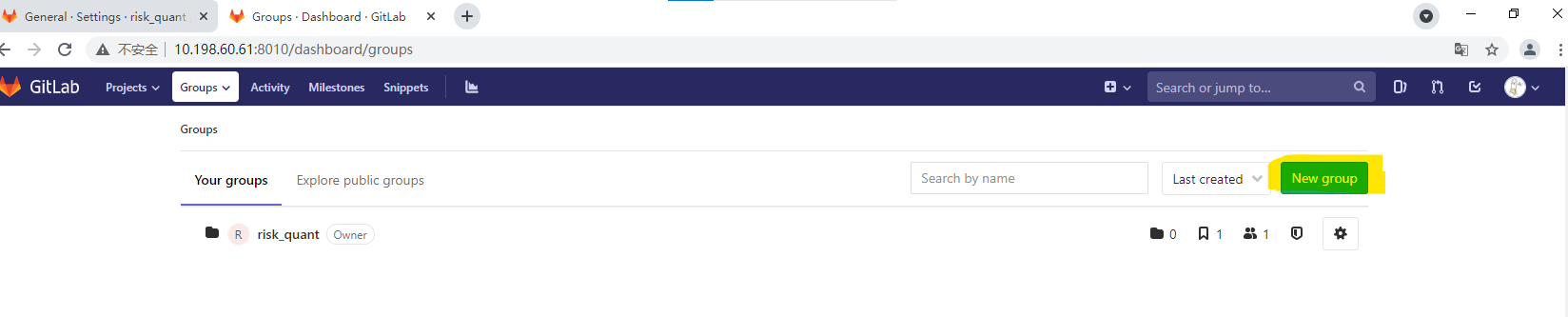
到项目setting下，按照标黄步骤所示操作





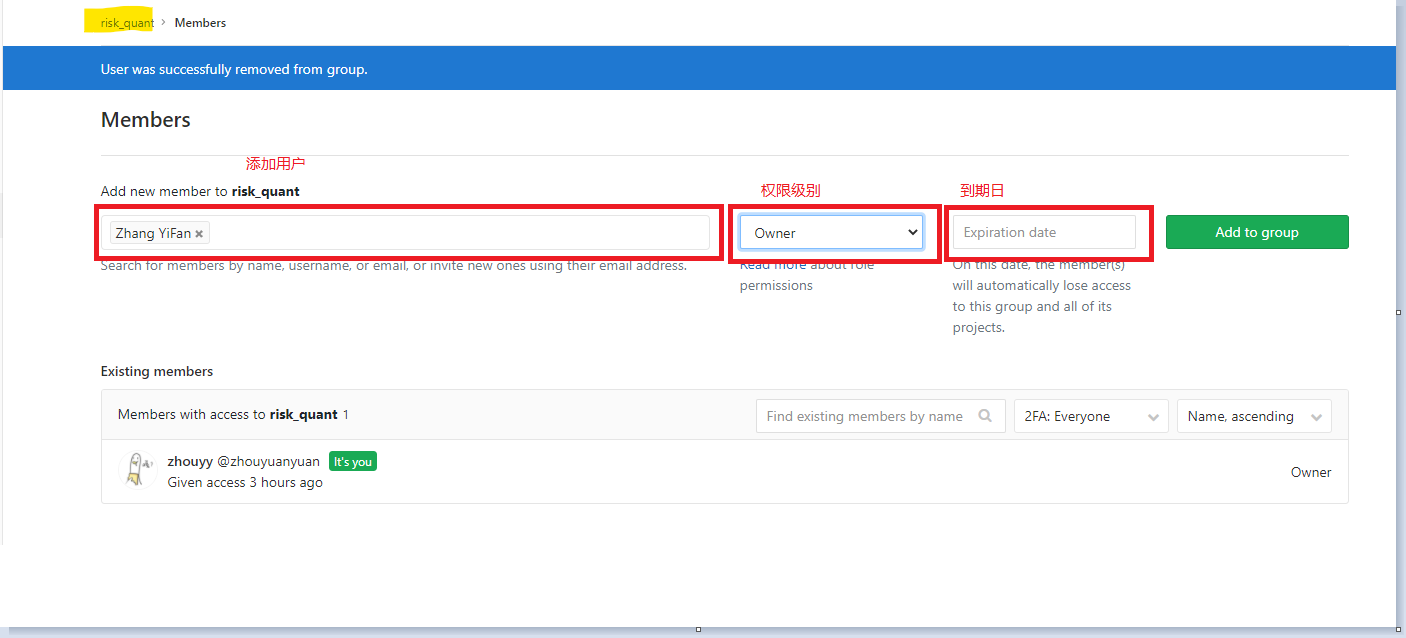
## 组管理

## 新建组（已新建risk\_quant）

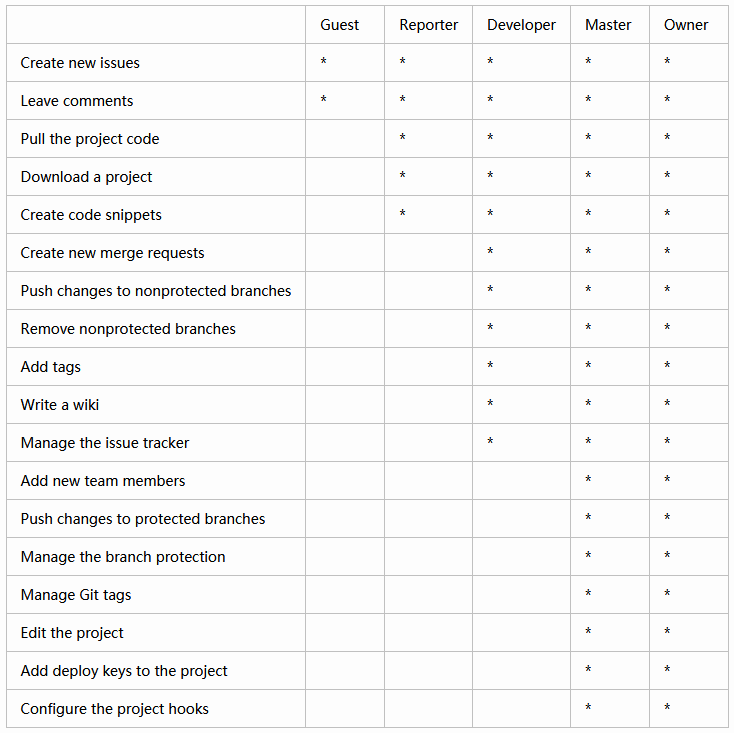


## 添加组成员

进入所在group（如risk\_quant）

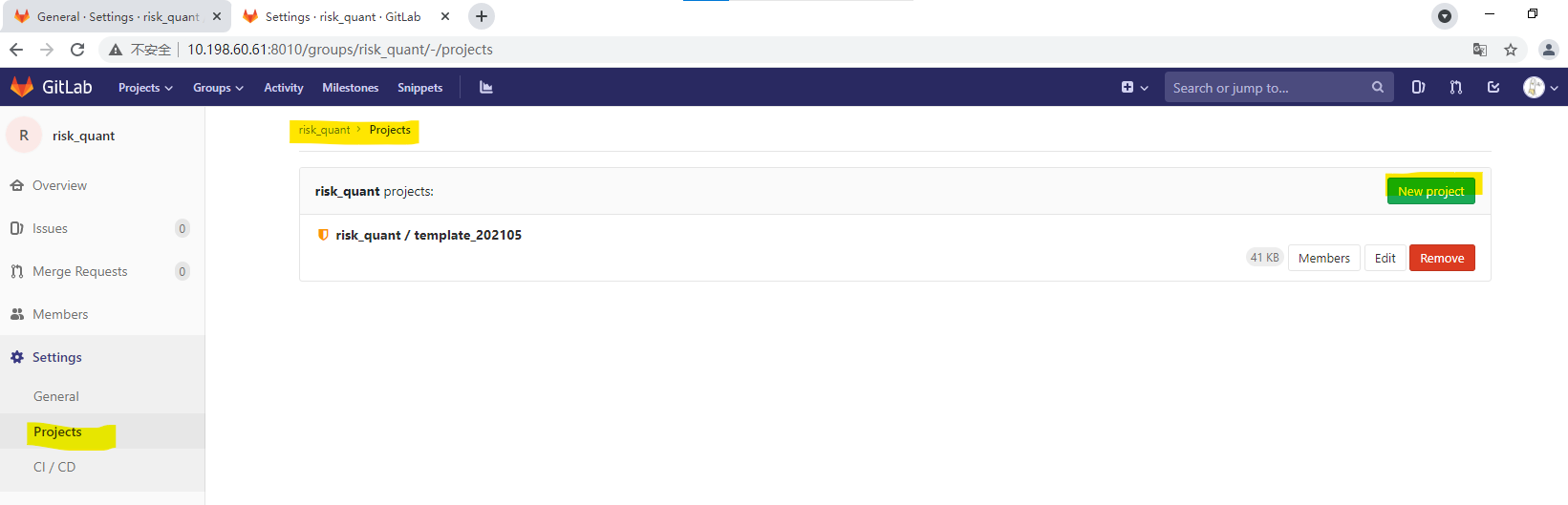


下表列出了部分Guest,Reporter,Developer,Master,Owner对应的权限，详细内容可点击read more查看



## 从组内添加项目

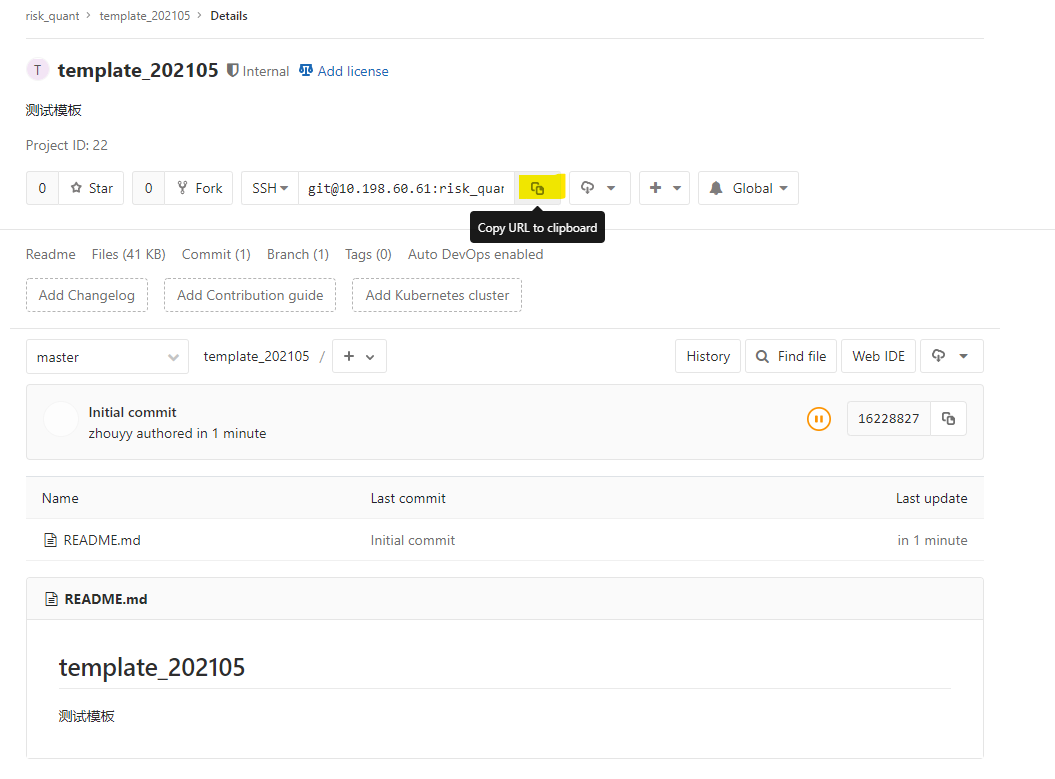
从组里添加项目可免去再添加项目用户的步骤



# TortoiseGit使用

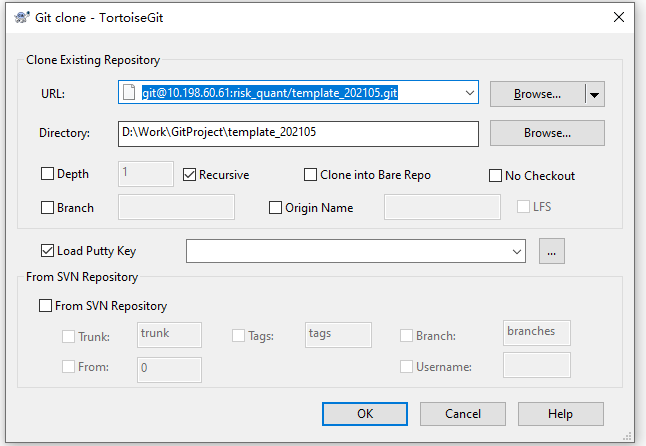
## 克隆（clone）远程仓库到本地

1. 假如我们已经在分组risk\_quant中新建了项目template\_202105, 先在GitLab中复制路径，如下所示：

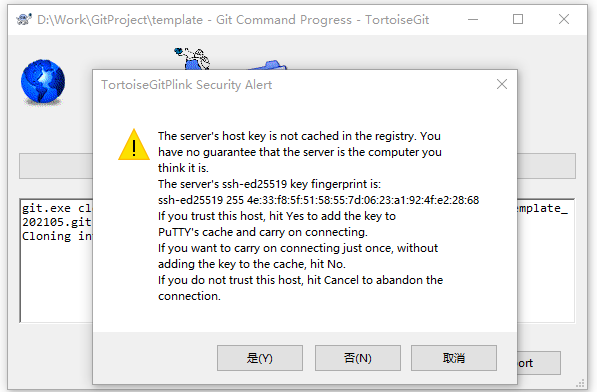


1. 选一个你喜欢的文件夹作为本地仓库，在空白位置处，右击鼠标，在菜单中选择【Git Clone】，然后在弹窗URL位置粘贴上刚刚复制的URL路径：

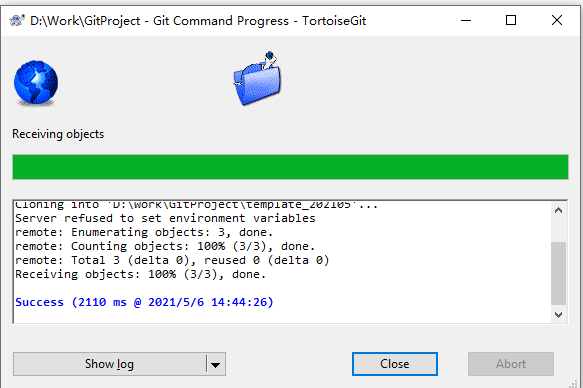




第一次运行可能若出现如下窗口，点击是。



完成后项目就克隆到了本地:





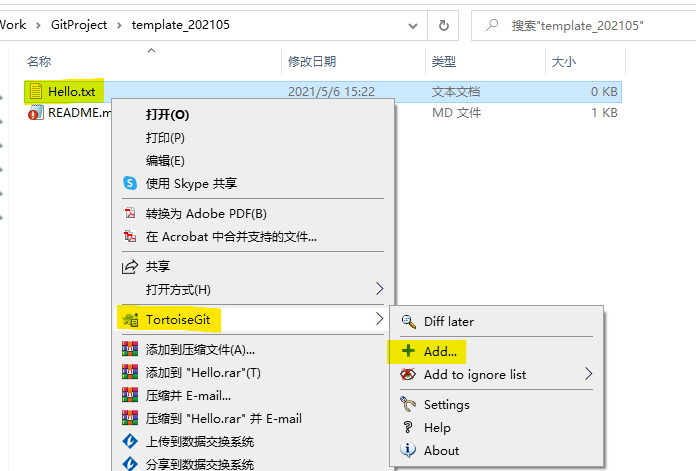


若是看不到角标可以查看该教程：[TortoiseGit状态图标不能正常显示的解决办法](https://www.cnblogs.com/xiesong/p/5761352.html)，或使用重启大法。

## 将代码或文件提交（commit）到服务器

场景：项目中新增、修改、删除了某些文件，将修改完的版本上传至服务器

1. 当我们在项目中添加新文件，需要右键->TortoiseGit->Add，之后该文件才会被添加到本地库。

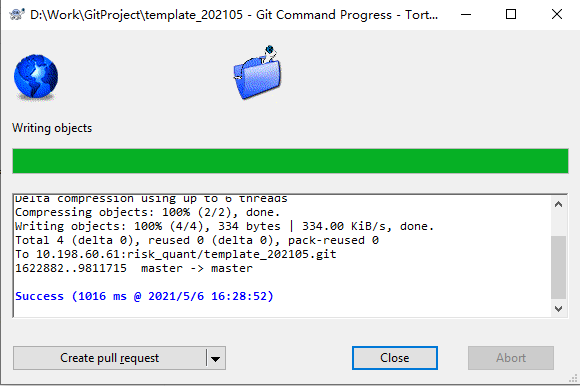
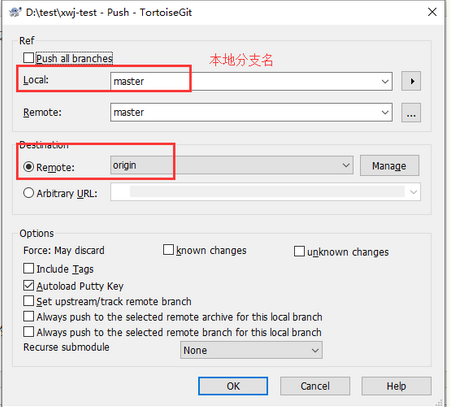


1. 当我们修改了部分已有文件，文件前面的角标状态会变成感叹号：
2. 接下来我们将本次所有修改commit到版本库：在空白处右键->Git Commit -> ‘master’

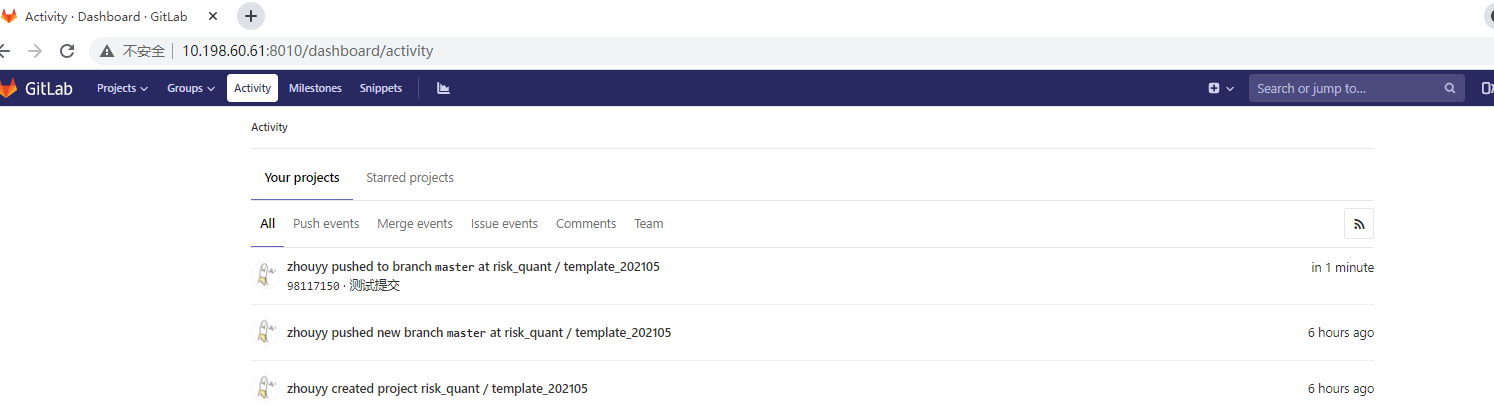
|  |
| --- |
| （双击文件名查看详细变动信息，可以看到本次修改删除了一行内容，且新增了一行hello git）  点击Commit后如下： |

1. Commit只是将代码提交到了本地版本库，服务器还没有同步，接下来将代码push到服务器：空白处右键，选择TortoiseGit->Git Push，出现如下弹框, 这里可以看到是本地哪个版本库提交到远端。至此，文件的整个提交过程就完成了。

（注，此步骤也可在上图最后一步commit之后直接点击push完成）



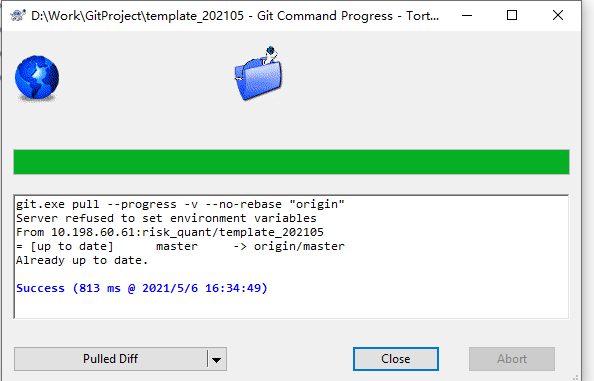
此时，到gitlab中activity页面，可以看到最新的提交记录：



## 拉取（pull）远端最新的修改内容

场景：当两个小伙伴一同修改同一个项目，小伙伴A想要获取B的最新修改内容（前提是B已经将最新版本push到了服务器）：

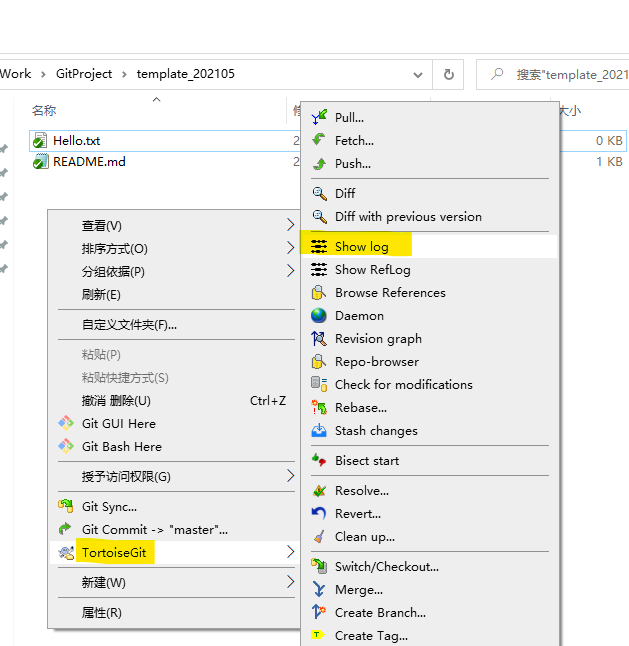
B可以通过空白处右键->Git Pull拉取，完成后如下。（注，在别人基础上修改，一定要先pull，否则可能出现conflict）



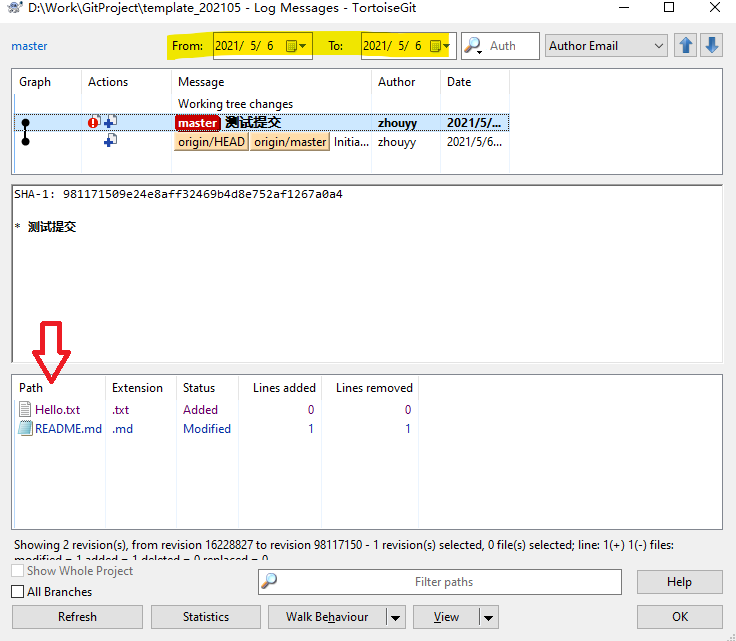
## 查看日志（show log）

场景：想要查看历史提交或回退代码

通过日志，可以很直观的看到提交相关记录。比如提交人、提交时间、提交了哪些文件等等。这些信息便于以后进行文件对比或者版本回滚



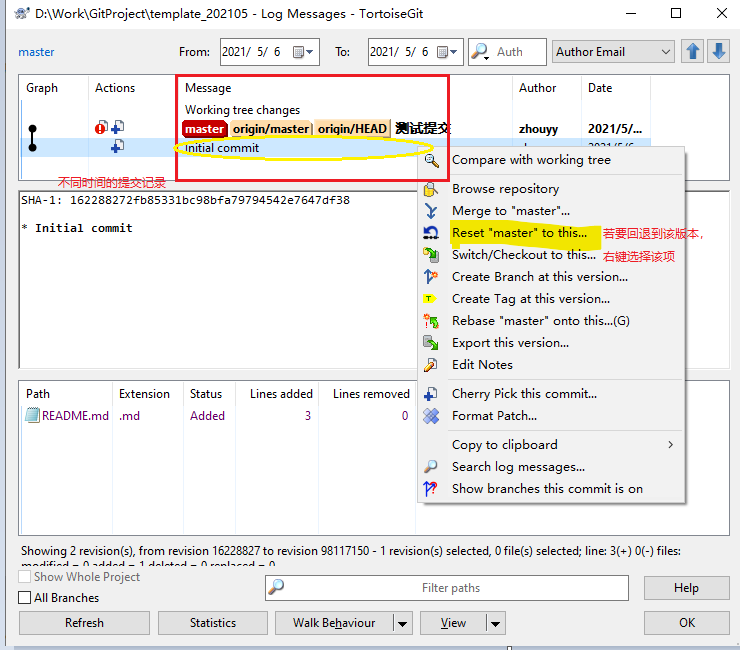
此时刷新后看到两个文件状态都已变成绿色，右键->show log可以查看项目历次修改信息（上方可过滤查询），并且也可以双击文件名的方式查看详细修改内容。



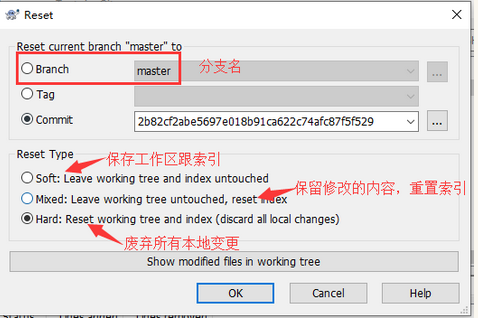
## 版本回滚(reset)

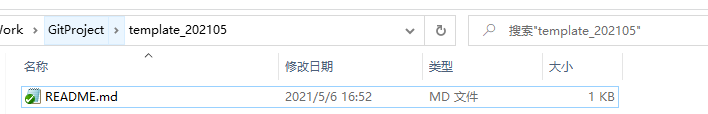
场景：当我们想将代码回退到之前某个版本

先查看日志，确定想回滚到哪个版本(这时就体现了commit时填的message的重要性)



选择之后，弹出如下弹框，在Reset Type下选择你回退的类型(楼主一般选择Hard：Reset working…..)，点击Ok即可



会发现，此时Hello.txt文件消失了，如下图

## 冲突解决

场景：与其他小伙伴共同开发一个功能，出现修改相同页面，其他开发人员修改后提交，自己也修改，导致无法拉取。

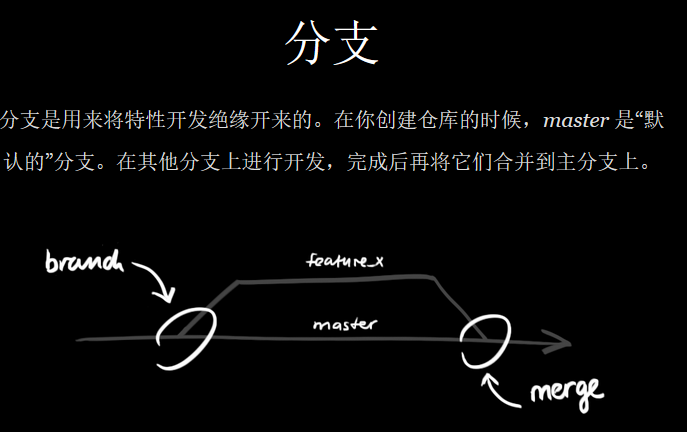
解决方案：<https://blog.csdn.net/myfmyfmyfmyf/article/details/105698827>

## 分支

场景：你代码写了很多，运行OK；但是突然想加个新功能进去，这个功能你也不知道能否正常运行，而且修改过程中，除了新加代码和文件进去，还会修改以前的代码。要是万一失败，修改回来也是一种很麻烦的事情。这种时候很多人就用备份方式。来看看git是怎么处理这个问题的：

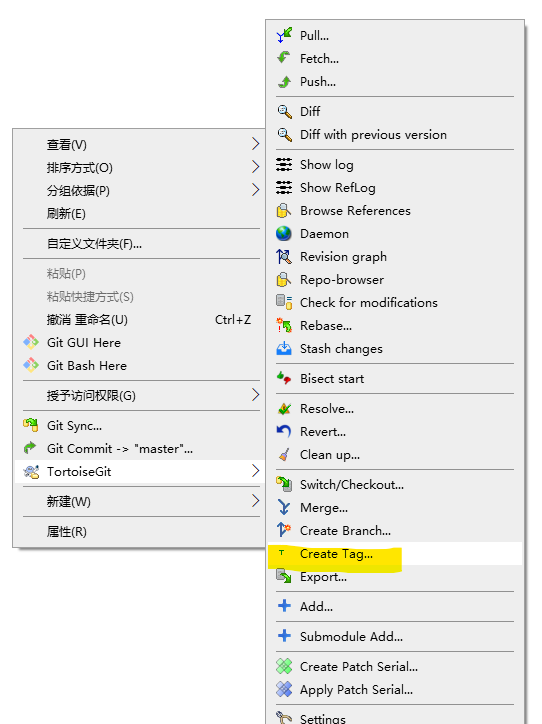
git的处理方式：当你想加一个新功能进去的时候，你可以新建一个分支，例如名字叫newbranch，然后在分支中把新功能加上去，如果OK，将代码合并到master分支上，如果新功能失败，切换回master分支上来，在newbranch写的代码，又全看不到了。

解决方案： <https://www.cnblogs.com/caicaizi/p/9809774.html>

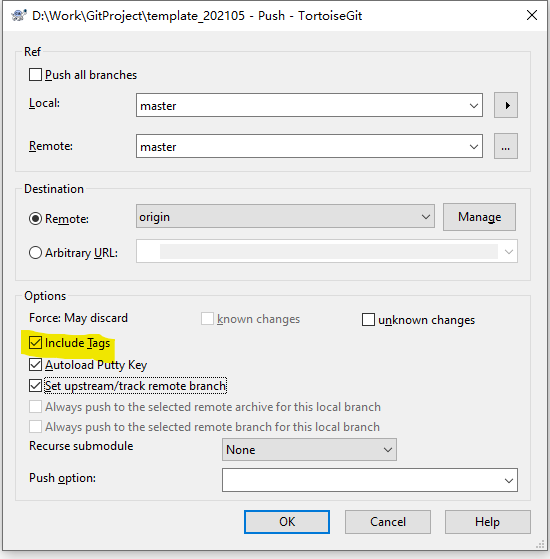


## Tag

所谓的Tag或是Release就是一个特别的版本，因为这个版本可能有特别的意义。例如：这个版本是特别的Milestone或是release 给客户的版本。其实，Tag与Release的作法与Branch完全相同。只是Branch可能会需要merge回原来的trunk中，而tag及 release大部分都不需要merge回trunk中。举例来说，今天我们的trunk做了一版，这个版本被认定是软件的1.0版。1.0版对于开发来 说是一个非常重要的里程碑。所以我们要特别为他做一个标记，亦即Tag。假设，这个 1.0版是要正式release给客户或是相关vendor，我们要可以为他做一个Release的标记。



在push时如果要推送标签，请勾选包含标签，如下图所示：



此时我们可以在gitlab上看到本次打了标签的版本：

