# 什么是Git？

## Git的定义

Git来源于Linux。Linux的内核全球有很多开发者贡献代码（原因是BItKeeper最开始是免费的，后面要收费了）（用一个不开源的框架版本管理一个开源的框架，大牛感觉到了侮辱）

所以Git天生就是分布式的。

什么是开源？--不单单是免费， 是开放，基于类库和包是开源的，你使用我的类库和包，也必须是开源的。

企业不做开源的原因：

## 与SVN有何区别？

1）Git是分布式的，SVN不是:

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。好处是跟其他同事不会有太多的冲突，自己写的代码放在自己电脑上，一段时间后再提交、合并，也可以不用联网在本地提交；

2）GIT与SVN版本机制不一样：

Git 只关心文件数据的整体是否发生变化，而SVN这类版本控制系统则只关心文件内容的具体差异。这类系统（如SVN）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，然而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链接。

3）GIT不需要联网：

SVN 断开网络或者断开VPN就无法commit代码，但是Git 可以先commit到本地仓库。用SVN的话，没有网络或者断开VPN时，你当然也可以继续在本地开发，但是无法commit代码，因为SVN 每次commit都必须联网，长时间不commit代码会丢失大量开发进程的历史纪录。有个比喻是：不能commit就像用word写文档不能save一样危险。而且有网络的情况下每一次commit都会花上数秒甚至更长时间。但用 Git 的话，就算你在飞机或者火车上，都可以非常愉快地频繁提交更新，因为是在本地仓库commit所以几乎不需要时间，而且commit一定要频繁，不然无法记录你的改动，如果你一天commit一次，中间的修改你就找不回来，然后等到了有网络的时候再将版本纪录和代码一起上传到远程仓库。

4）Git 的内容完整性要优于SVN：

因为Git 在commit（存储在本地）或者push（上传到远程仓库）之前，通过对文件的内容或目录的结构计算出一个 SHA-1哈希值，作为指纹字符串进行内容的校验，并将此结果作为数据的唯一标识和索引，在远处仓库接受到commit的文件之后，会再计算一遍哈希值然后跟传递过来的哈希值做比较，如果不一致，说明文件在传输时变得不完整，或者磁盘损坏导致文件数据损坏。另外在 Git 数据库中的东西都是用此哈希值来作索引，而不是靠文件名；

中心化：银行（中心）、

去中心化：比特币、分布式账本（捐钱：红十字会） 银行都在做这一块。

Commit 完整性 40位 SH1的加密（不可逆）

## 为什么要用Git？

1) git每台电脑都有一个版本库，可以在本地做版本管理；

2) 速度快。git的速度远超大部分版本管理系统，包括svn

3) 强大的分支管理功能

4) 活跃的开源社区，如最著名的github

## Git与GitHub

gitHub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。

gitHub除了git代码仓库托管及基本的 Web管理界面以外，还提供了订阅、讨论组、文本渲染、在线文件编辑器、协作图谱（报表）、代码片段分享（Gist）等功能。摘自百度：作为开源代码库以及版本控制系统，Github拥有超过900万开发者用户。随着越来越多的应用程序转移到了云上，Github已经成为了管理软件开发以及发现已有代码的首选方法。

# 安装和配置

除了官方版本，也有其他软件公司将Git包装成GUI操作界面类型的程序(官方版本主要是使用指令模式进行操作)。虽然GUI的图形化操作界面看起来比较吸引人，但是建议还是先从Git的指令模式开始学习，这样才能够清晰的了解Git的工作细节。玩转了指令模式，界面工具自然能够水到渠成。

### Git安装、部署和运行

下载 Windows版 git（windows环境默认安装即可）。

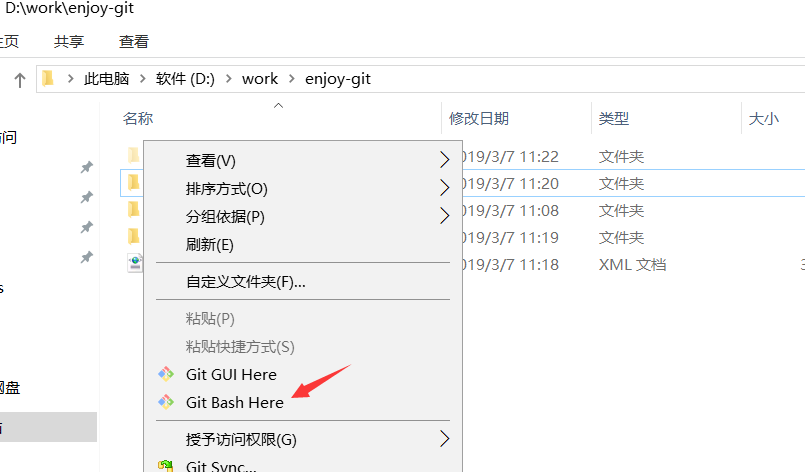
下载地址：<https://git-scm.com/downloads>

图形化工具SourceTree：<https://www.sourcetreeapp.com/>

https://www.sourcetreeapp.com/

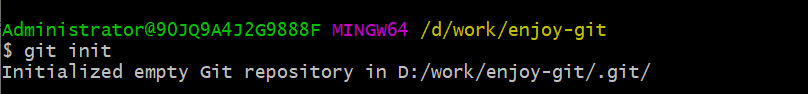
### GIT指令模式

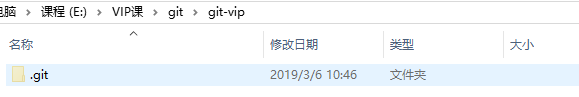
安装完git后，在任一一个文件夹(我自己的一个简单项目:D:\work\enjoy-git),右键进入git bash



#### Git项目存入版本库

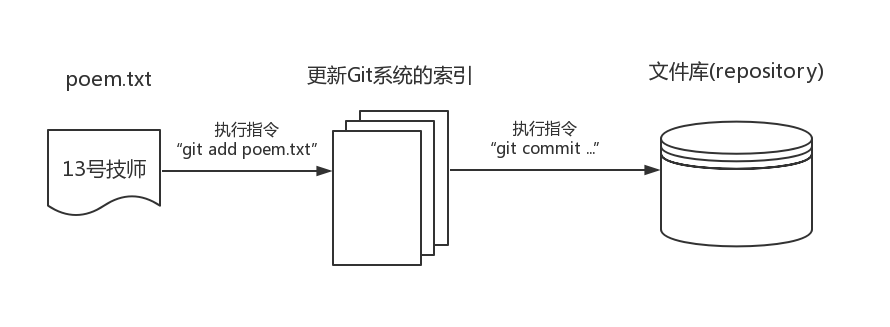
git init 让git开始管理这个文件夹，在同级目录下会出现一个隐藏的.git文件。



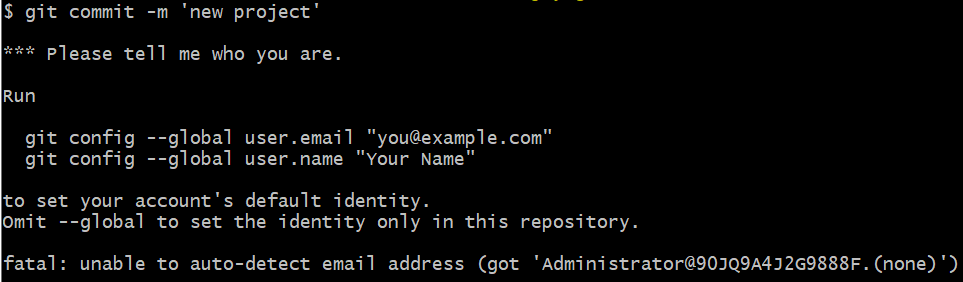




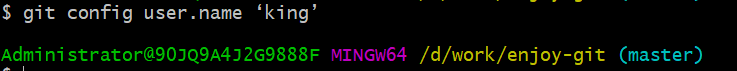
#### Git管理文件



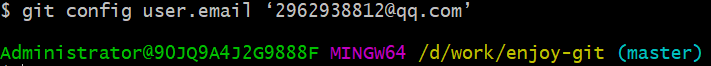
记得提交前设置下操作者的姓名和email ,不然会报错提交不了.



git config user.name ‘king’

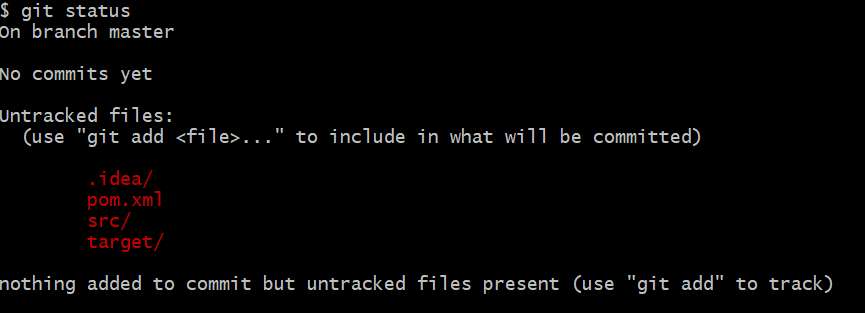


git config user.email ‘2962938812@qq.com’



##### 提交版本

git status 检查版本状态



**提交本地**(可以两个步骤，也可以一个步骤)

两个步骤：

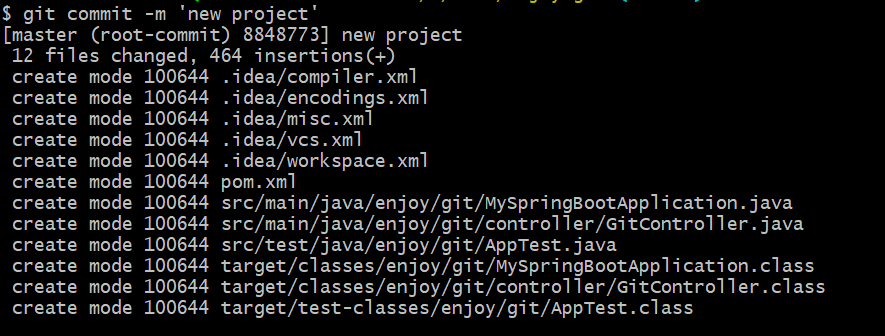
git add . 或git add -a或git add filename （.代表所有新增、修改 -a 新增、修改、删除）

加入缓存区（.或者-a表示全部，也可单独文件）

git commit -m ‘new project’ 提交版本 ‘版本提交描述’

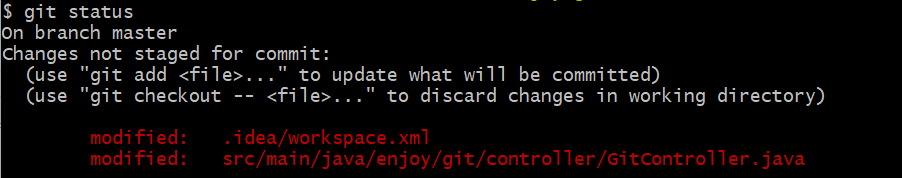
一个步骤（两个步骤合一）

git commit -am ‘new project’ 加入缓存区并提交

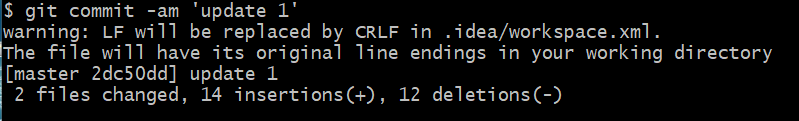


如果其中一个文件GitController.java内容进行了修改，我们再次提交

git status ---先检查下状态(显示变动信息)



git commit -am ’update 1’



补充说明：

1.如果只执行git init后没有执行git commit的话，可以使用一个命令把索引信息删除

git rm --cached 文件名

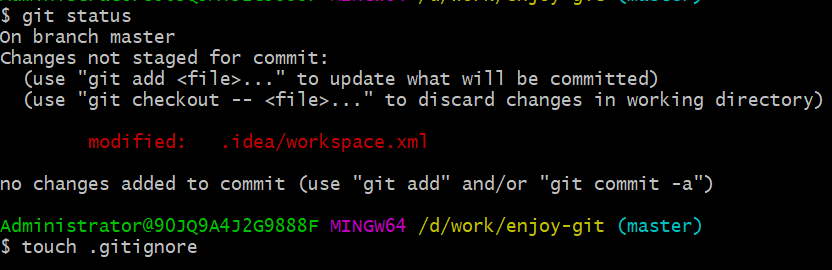
1. 如果你修改了文件名，使用git commit -am 的话，可以也有对应的文件修改记录。

虽然git mv 原来的文件名 新文件名 可以做到 ，但是提交中一样可以做到，所以mv基本上使用得比较少

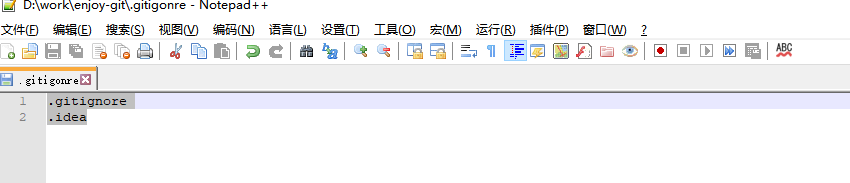
##### 忽略和排除

忽略那些不需要管理的文件夹或文件（比如此项目中.idea文件就需要忽略掉）

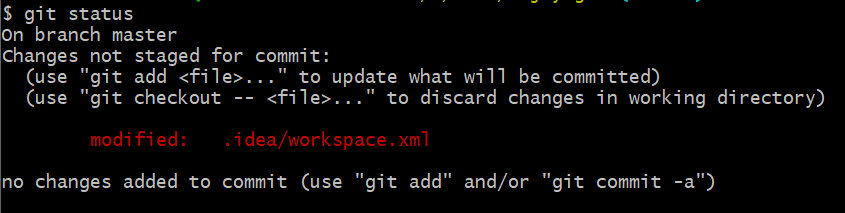
Touch .gitignore 创建一个.gitignore文件



输入一下内容，因为.gitignore本身就需要忽略，所以此文件需要忽略



我们再次检查发现为什么 .idea文件为什么没被忽略了



原因是.gitignore只能忽略那些原来没有被追踪的文件，如果某些文件已经被纳入了版

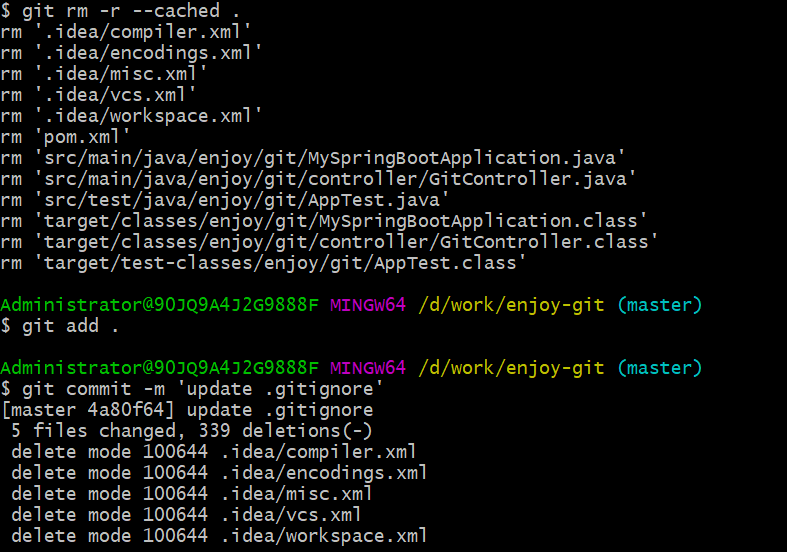
本管理中，则修改.gitignore是无效的。那么解决方法就是先把本地缓存删除（改变成

未被追踪状态），然后再提交：

git rm -r --cached .

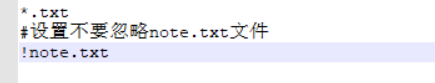
git add .

git commit -m 'update .gitignore'



补充说明：

1. “.gitignore”文件的影响范围是它所在的文件夹和所有的子文件夹
2. 每一个文件夹沟可以创建自己的“.gitignore”文件，如果它上层的文件夹也有“.gitignore”文件，这个文件夹也会受到它的影响。
3. “.gitignore”文件中可以使用“#”字符开头表示注释，文件夹路径是用“/”字符，文件名可以使用通配符“\*”，另外还可以使用“!”字符表示排除。



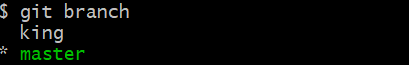
#### Git分支管理

##### 创建分支

git branch king 创建一个 名字为king的分支



git branch 不加任何参数，查看分支情况(master代表主版本)



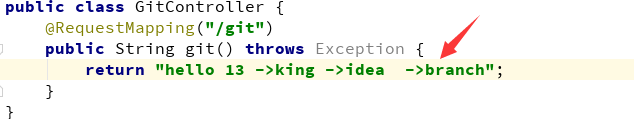
##### 切换分支

git checkout king 切换分支(king)

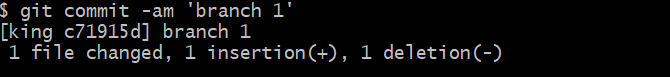


##### 切换提交

在分支king上修改



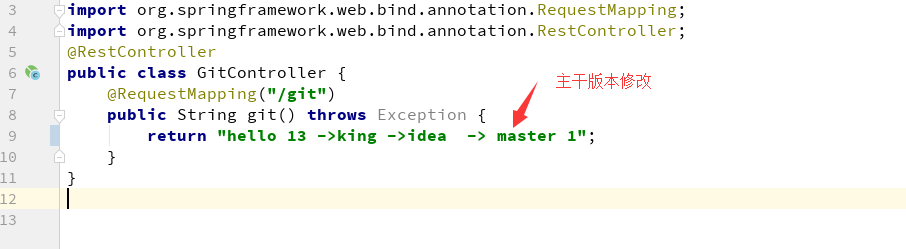
在分支king上提交 git commit -am 'branch 1'



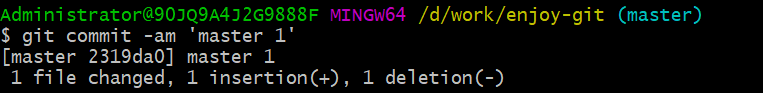
##### 分支的冲突和解决

切回主干版本修改内容



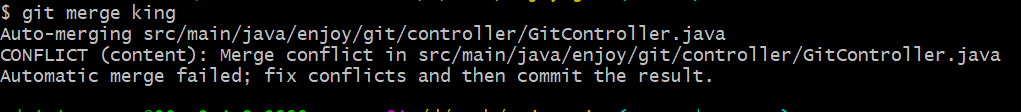


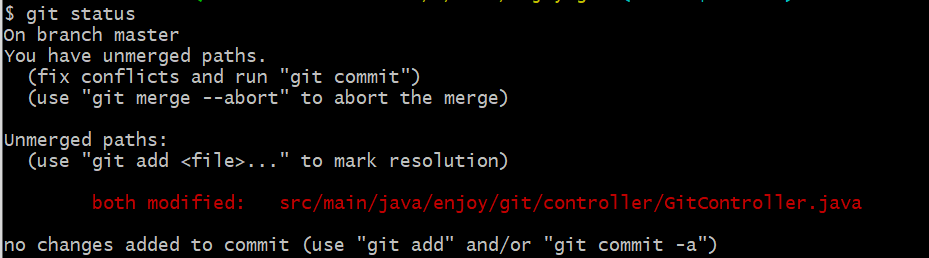
提交主干版本



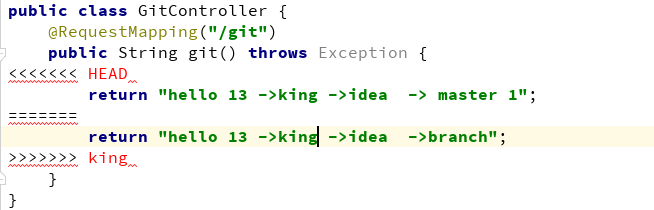
查看冲突

git merge king master版本与king分支

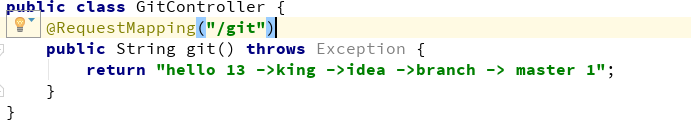


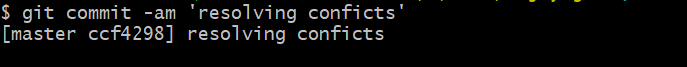


java类中冲突显示出来。



解决冲突（手工修改内容再次提交）---注意如果出现了冲突，就必须先解决冲突才能切换分支





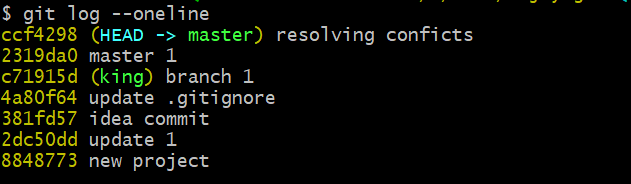
#### Git日志

git log



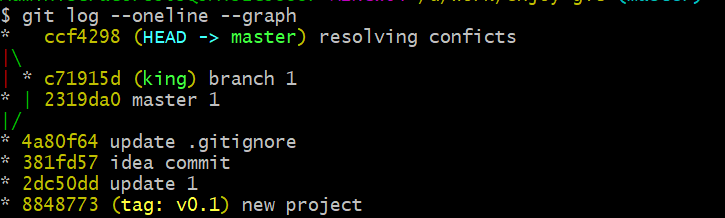
简化成一行

git log --oneline



简化成一行,同时显示分支

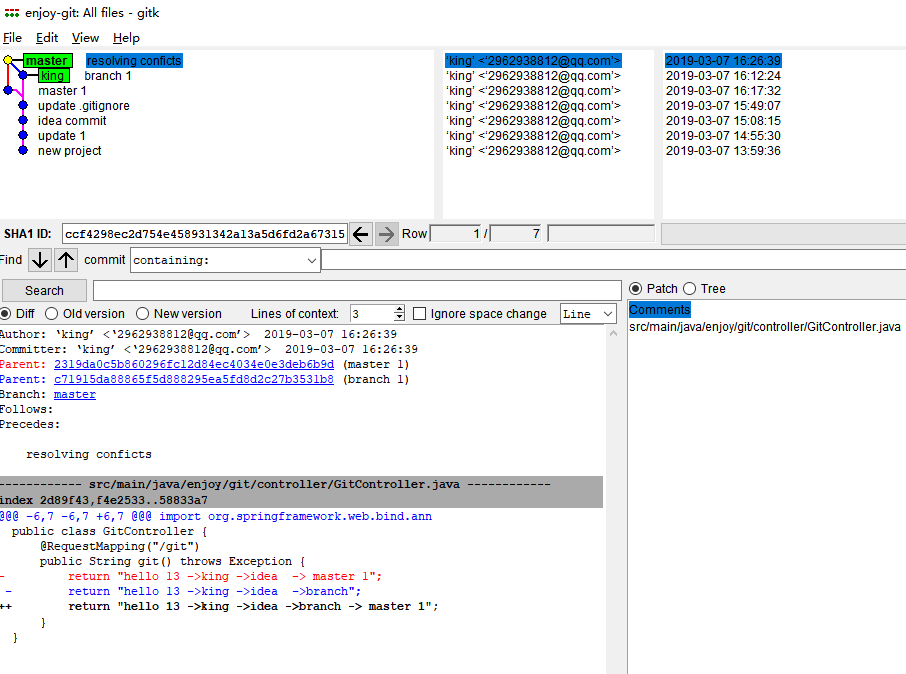
git log --oneline --graph



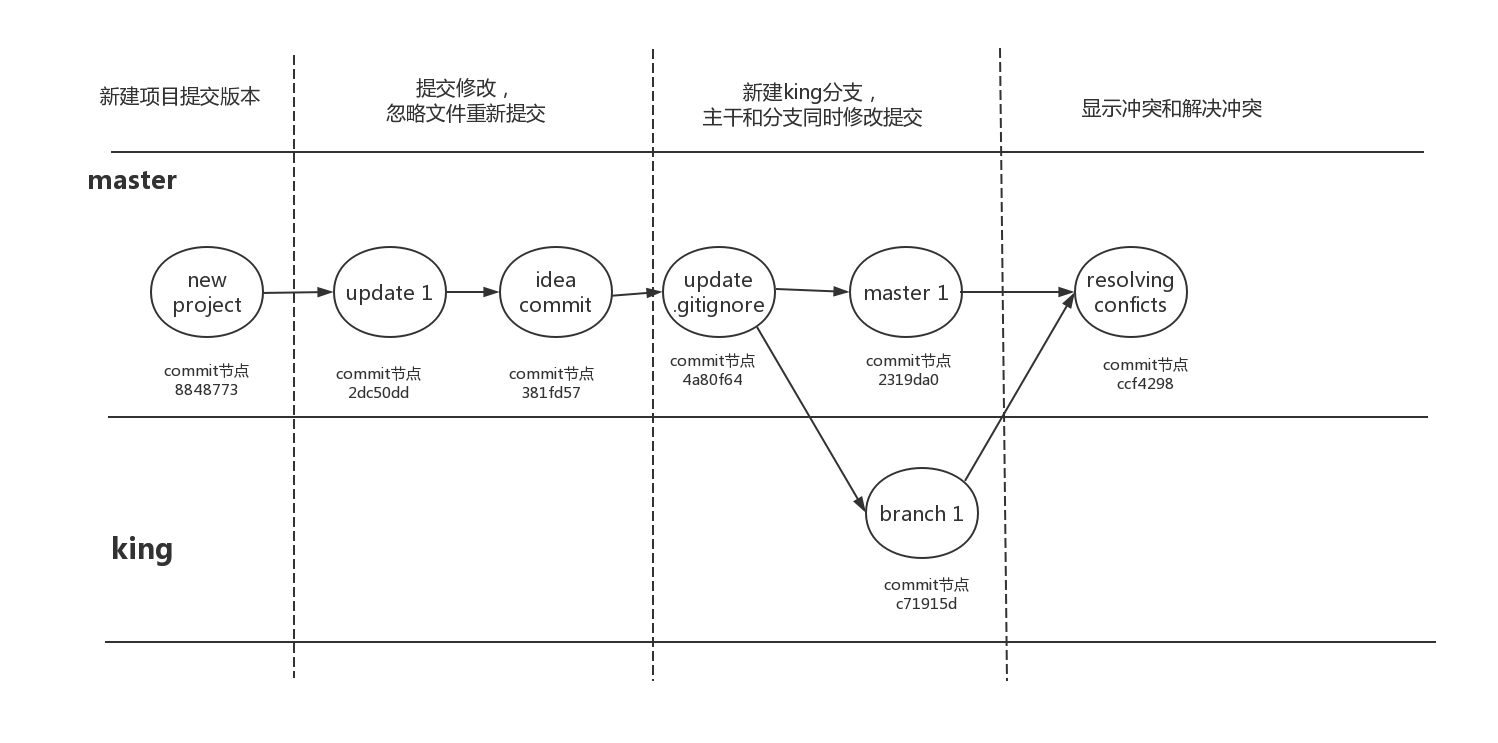
#### Git提交点

在命令行使用gitk可以展示简单的git界面。

commit的中的这些字母和数字就是commit节点的标识(比如最新的提交点是 ccf4298)



这个界面太low了，我来给大家画一张图总结一下之前的提交信息

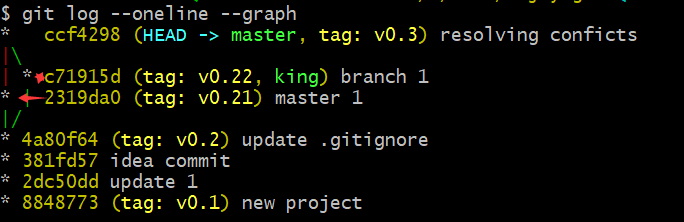


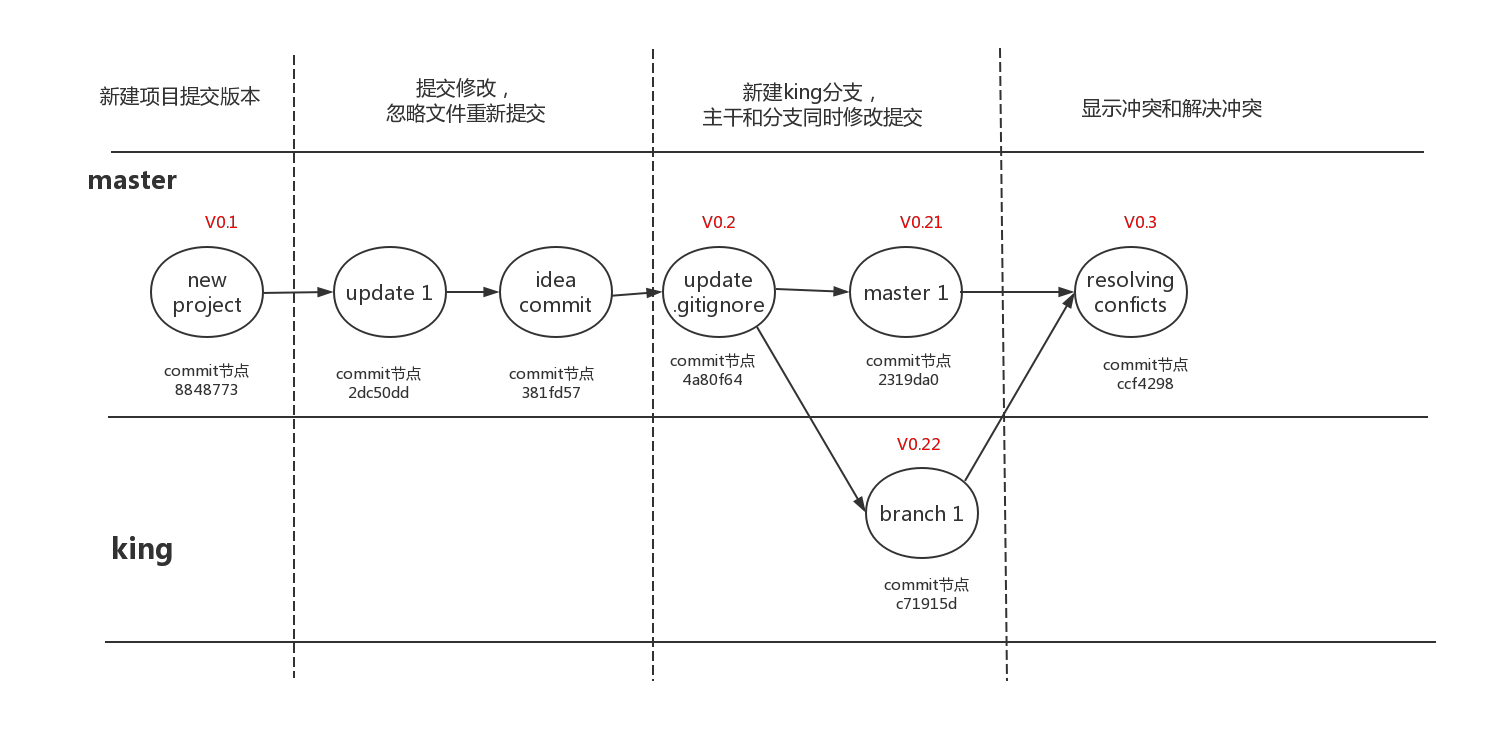
##### 标签

git tag -a v0.1 8848773 给第一个提交点打上标签



重要的修改点都打上tag

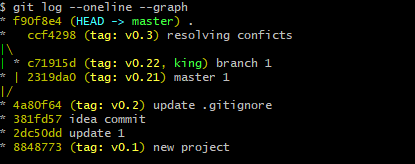




##### 恢复

在利用github实现多人合作程序开发的过程中，我们有时会出现错误提交的情况，此时我们希望能撤销提交操作，让程序回到提交前的样子，本文总结了两种解决方法：回退（reset）、反做（revert）

比如我已经修改了控制类的名字，并且已经提交(f90f8e4)这个提交点。



在Git里，有一个HEAD指针指向当前分支（只有一个分支的情况下会指向master，而master是指向最新提交）

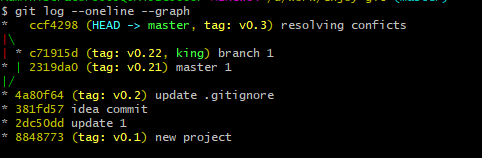
git reset的作用是修改HEAD的位置，即将HEAD指向的位置改变为之前存在的某个版本。（比如需要回到到V0.3）

使用“git reset --hard 目标版本号”命令将版本回退



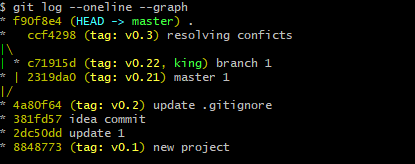
查看一下日志：

git log --oneline --graph

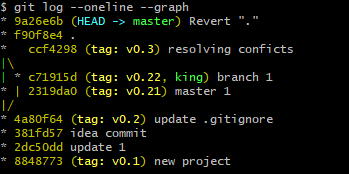


git revert的作用通过反做创建一个新的版本，这个版本的内容与我们要回退到的目标版本一样，但是HEAD指针是指向这个新生成的版本，而不是目标版本。

比如我已经修改了控制类的名字，并且已经提交(f90f8e4)这个提交点。



使用“git revert HEAD”撤销前一次的commit



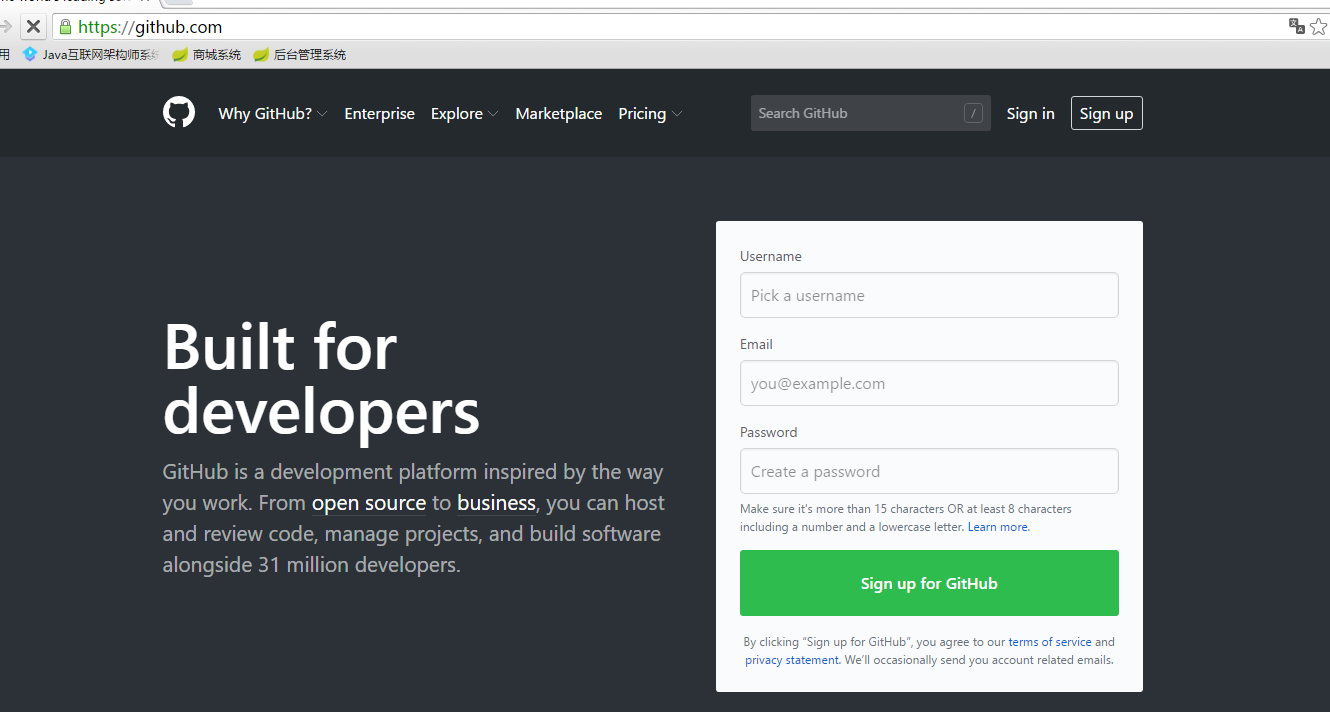
看log信息，revert已经把新建了一个提交点9a26e6b，这个点的内容和ccf4298提交点的内容是一样。区别就是新建了一个commit点

#### Git远程仓库

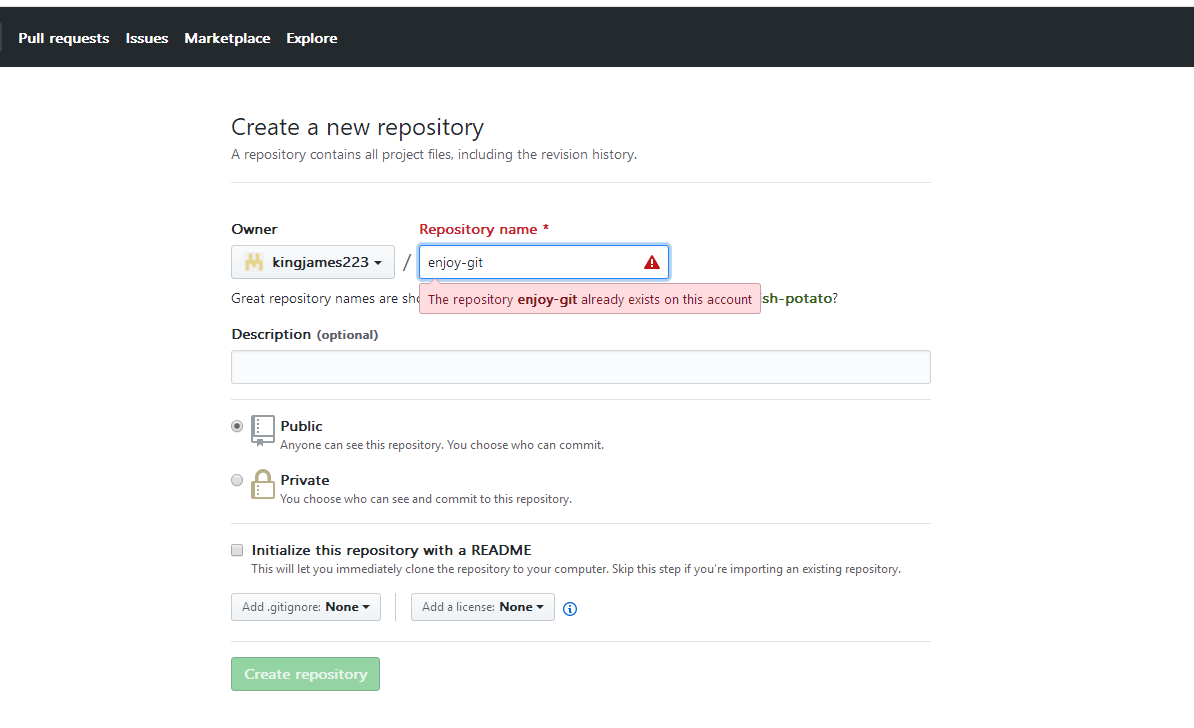
##### 注册GitHub账户

远程仓库使用大家都非常熟悉的GitHub,首先大家必须在github上注册一个账户，建议大家使用QQ邮箱即可完成注册(具体过程自行完成即可)。

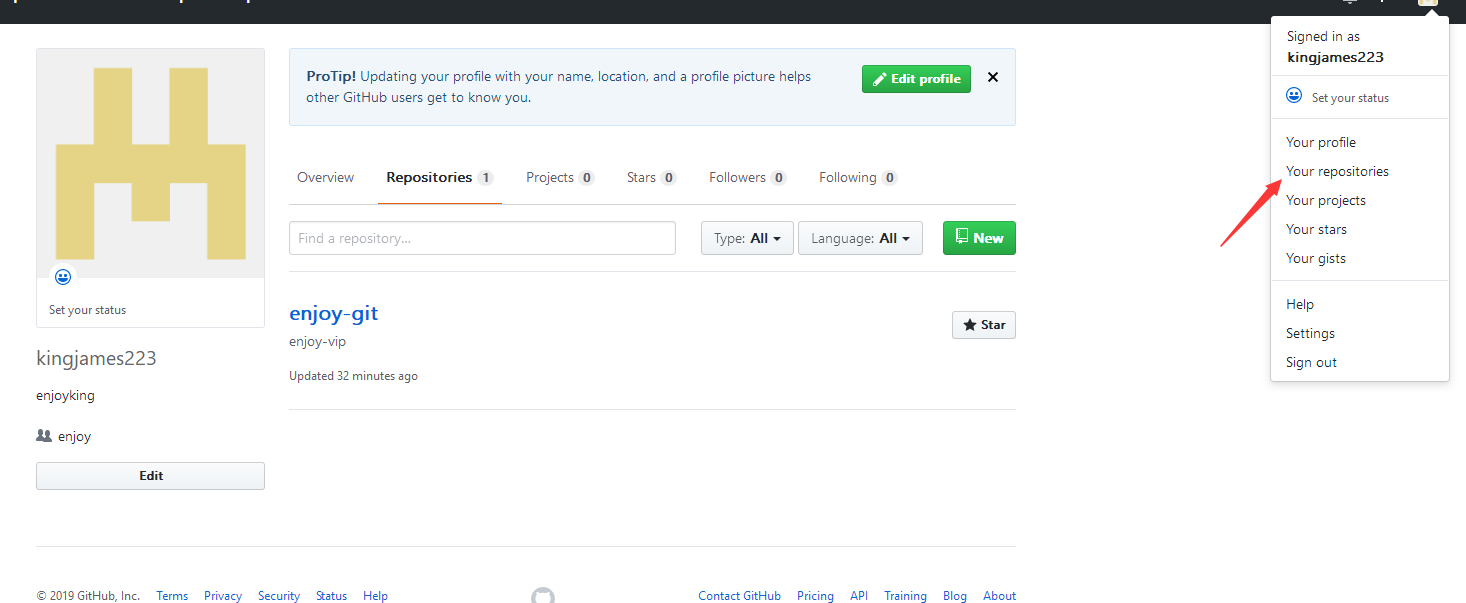
https://github.com/

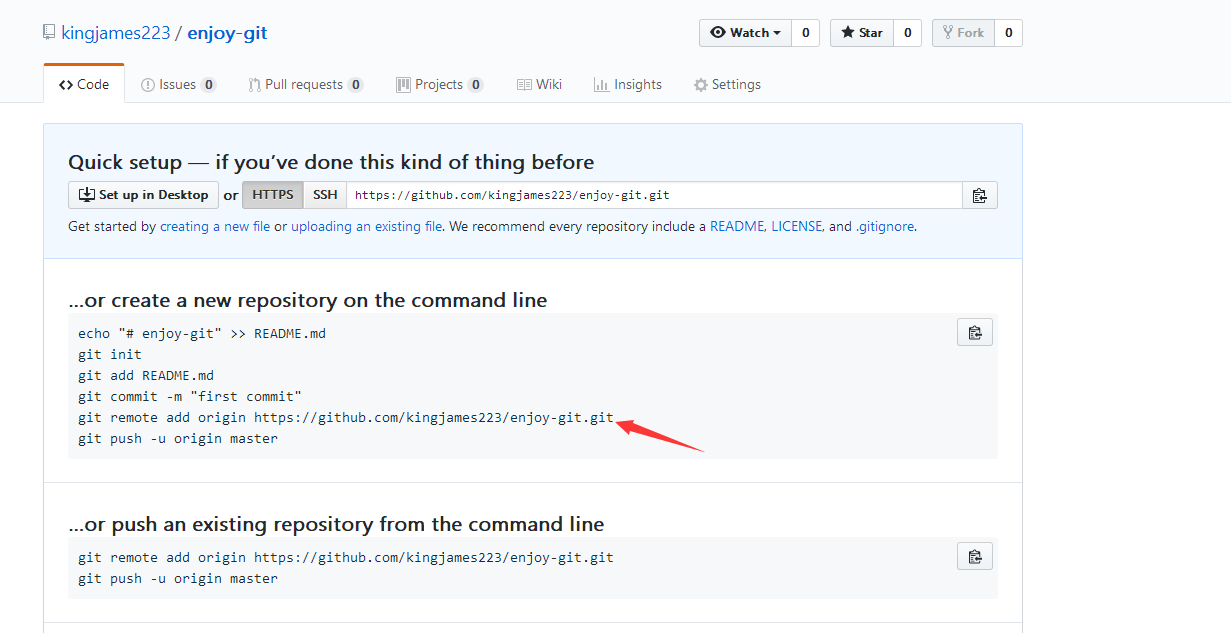


新建一个和本地一样的工程，“enjoy-git”



回到仓库，查询出enjoy-git仓库远程地址



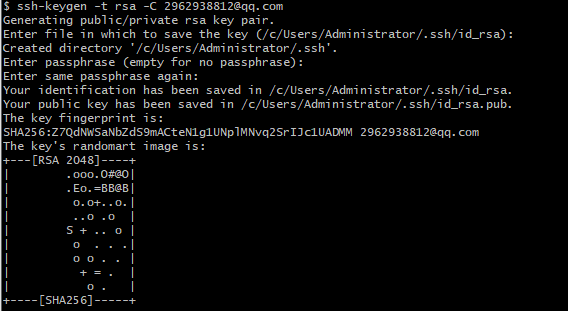


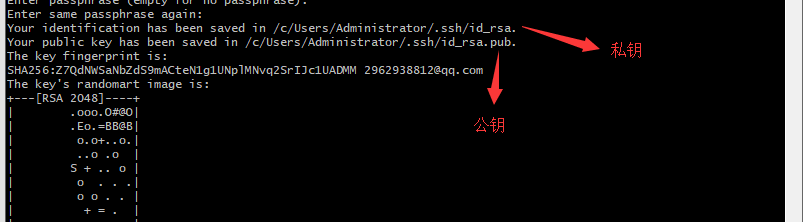
##### 生成和配置RAS秘钥

注意在本地连通github首先必须通过RAS加密方式通讯，所以本地需要生成一个RSA的秘钥。命令如下：

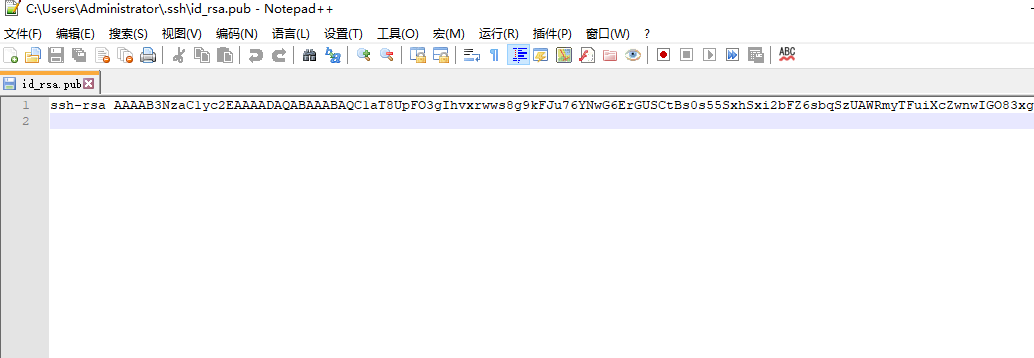
ssh-keygen -t rsa -C [2962938812@qq.com](mailto:2962938812@qq.com)

输入后一直回车即可完成。

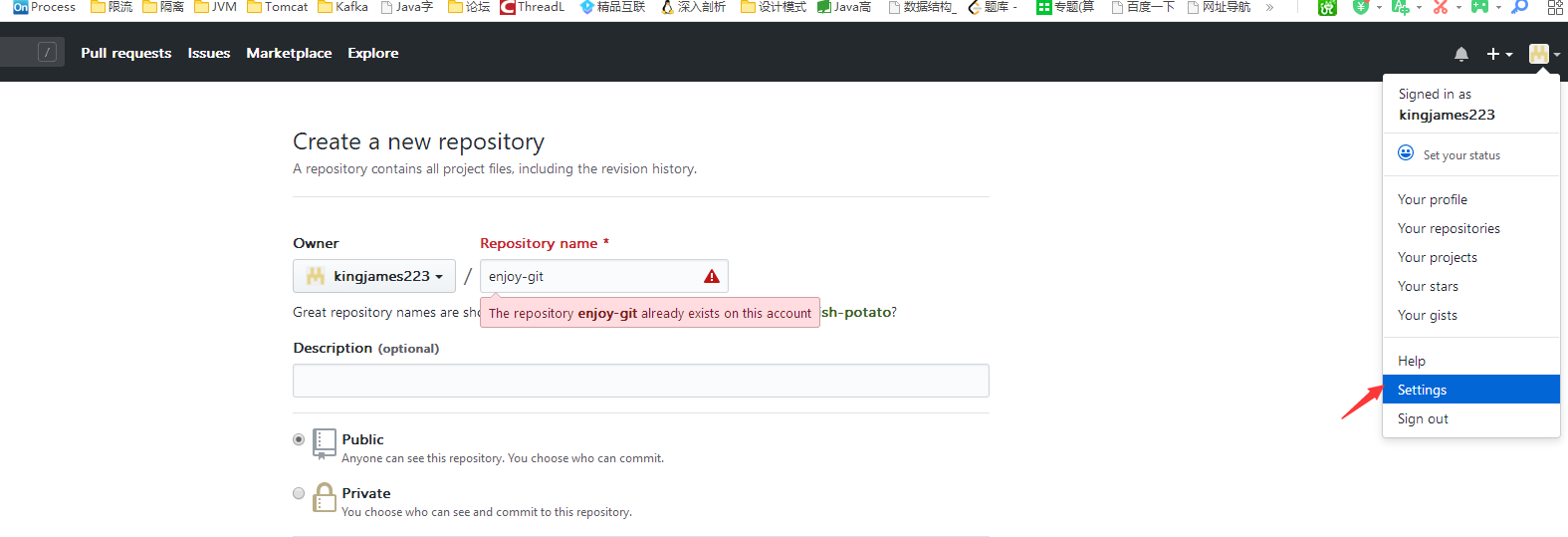




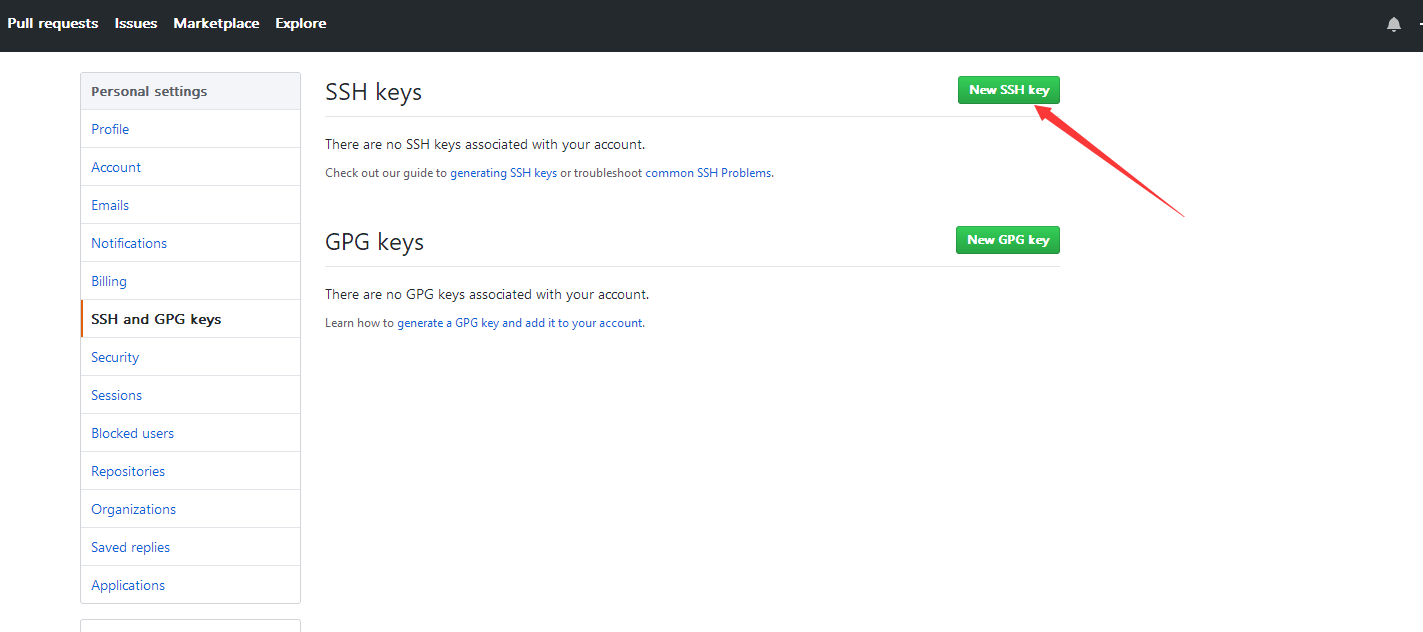
根据上图的公钥地址寻找到对应的公钥:



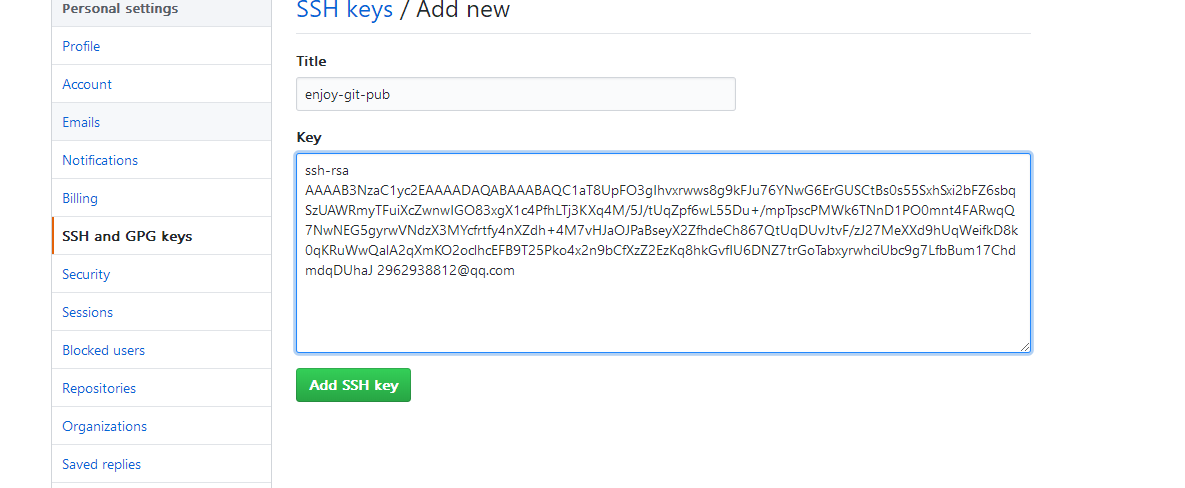
同时配置github上对应的公钥

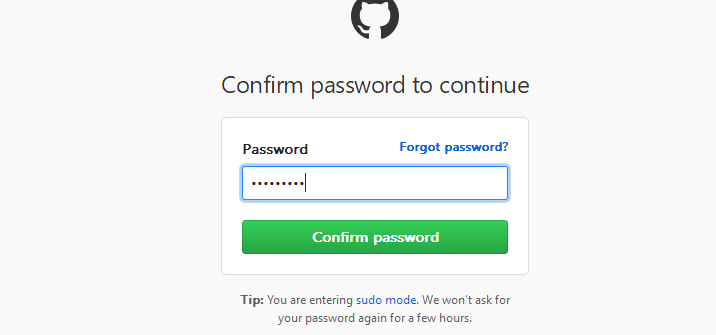


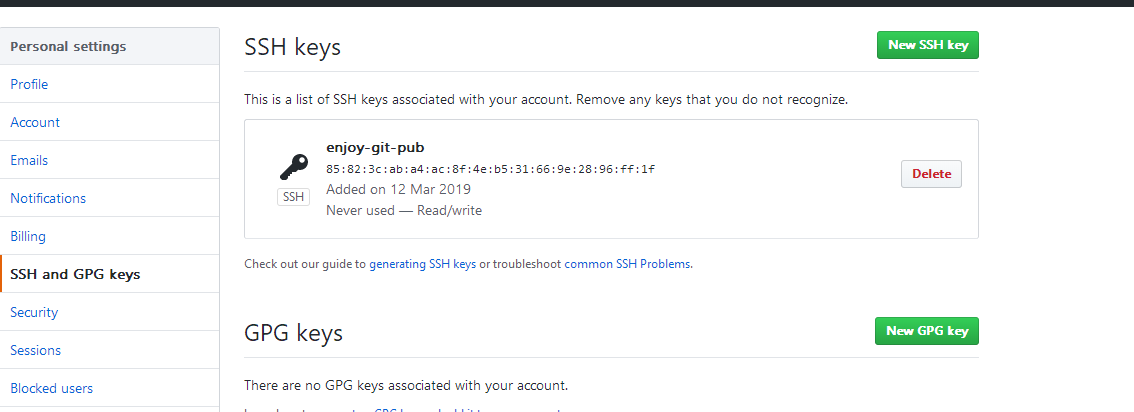
选择SSH keys，需要new一个SSH key



把刚才本地生成公钥粘贴至key中且保存(保存后会提示输入密码，github的登录密码)







##### 本地与远程仓库通讯（使用https的方式，第一次提交时需要输入密码）

经过上面的注册GitHub账户过程中新建的一个enjoy-git库中知道了远程通讯的github地址为 https://github.com/kingjames223/enjoy-git.git

1. 本地新建一个远程连接 enjoy

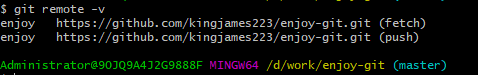
git remote add enjoy <https://github.com/kingjames223/enjoy-git.git>



1. 查看远程连接 git remote



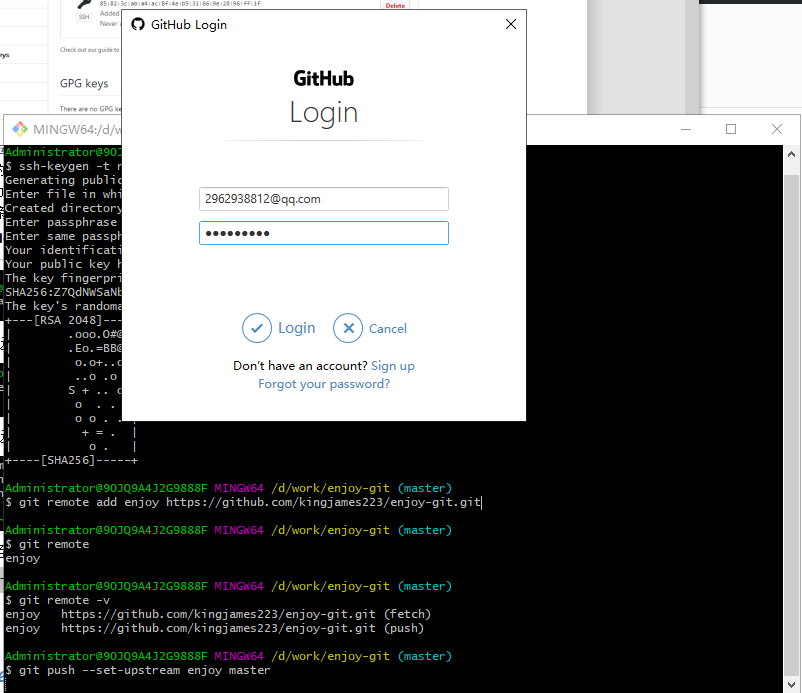
1. 查看远程连接详情 git remote -v



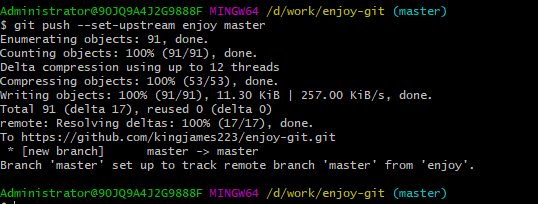
上传本地项目至GitHub (主版本)

git push --set-upstream enjoy master 可简化git push -u enjoy master

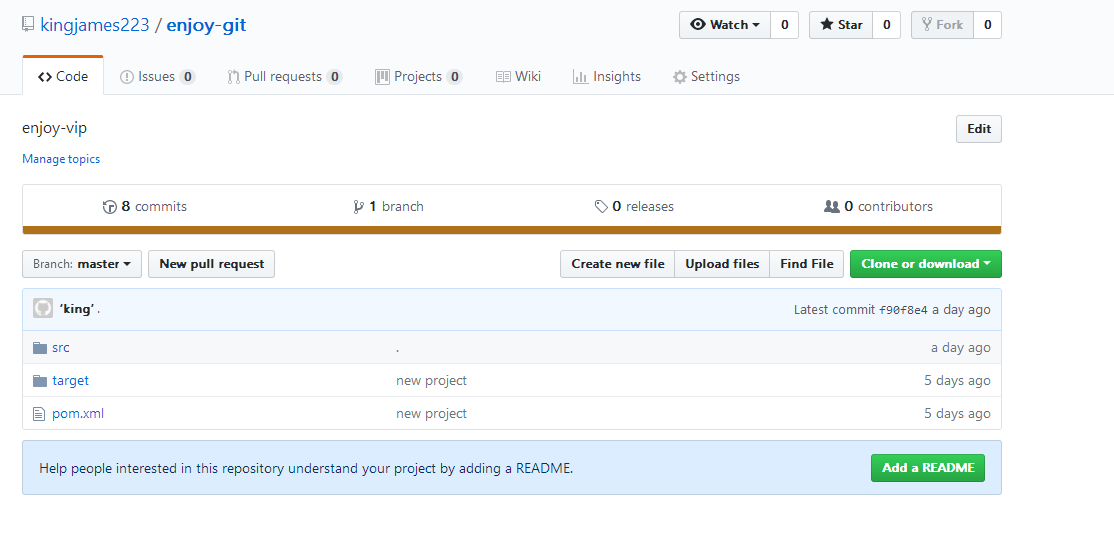
命令后需要输入github用户名和密码



上传成功后的本地命令窗口

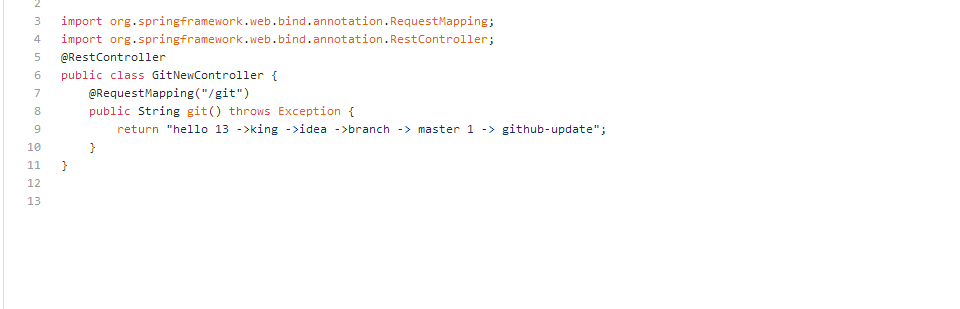


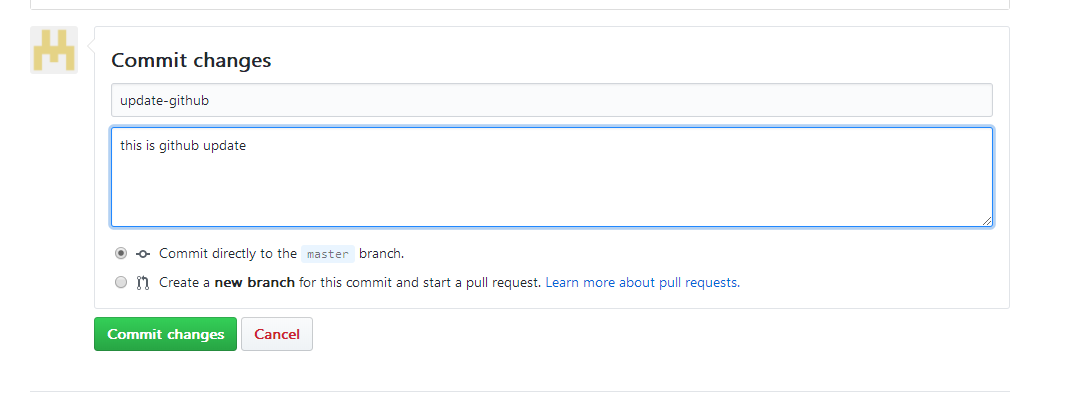
再次刷新下github上的项目(内容已经成功上传)



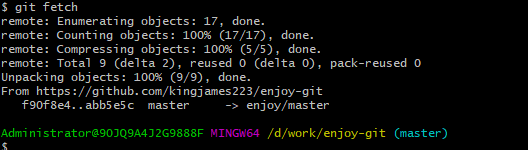
修改远程仓库的内容，同时同步至本地



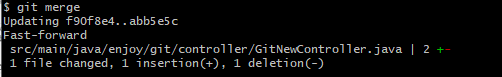


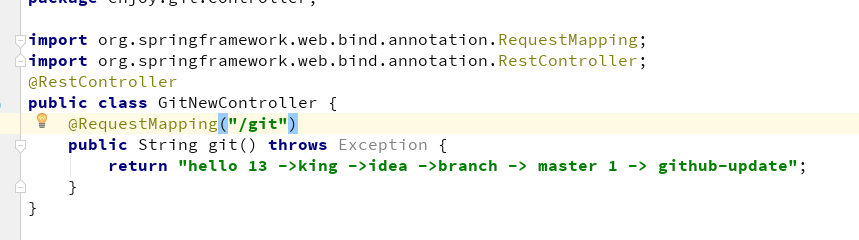


git fetch 提取远程仓库(如果服务器有修改，不会到本地)



git merge 合并冲突(如果只是本地或远程一端修改的话，就不会冲突)



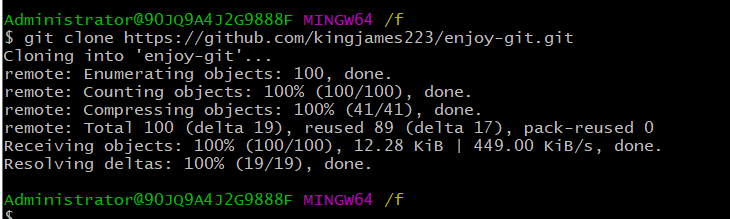


克隆，比如在f盘，我进行了一个

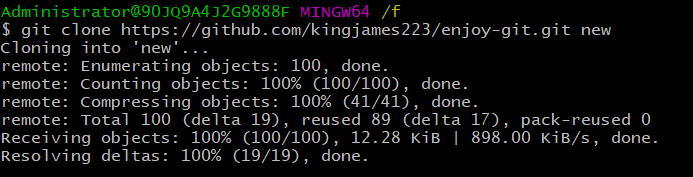
git clone github地址 ---在命令目录下会自动生成一个对应项目的目录

git clone github地址 文件夹 -- 会在文件夹中clone对应的内容

git clone https://github.com/kingjames223/enjoy-git.git



git clone <https://github.com/kingjames223/enjoy-git.git> new



克隆以后的项目push

git config --global push.default matching

#### Git Flow必备技能，规范团队Git操作

代码管理需要一个清晰的流程和规范

##### master分支

主分支

保持稳定

不允许直接往这个分支提交代码，只允许往这个分支发起merge request

只允许release分支和hotfix分支进行合流 , 所有在Master分支上的Commit应该Tag

##### develop分支

开发分支

相对稳定的分支

用于日常开发，包括代码优化、功能性开发

##### feature分支

特性分支

从develop分支拉取，用于下个迭代版本的功能特性开发

功能开发完毕合并到develop分支, 合并完分支后一般会删点这个Feature分支，但是我们也可以保留。

##### release分支

发布分支

从develop分支拉取

用于回归测试，bug修复

发布完成后打tag并合入master和develop, (记住：一旦打了Release分支之后不要从Develop分支上合并新的改动到Release分支)

##### hotfix分支

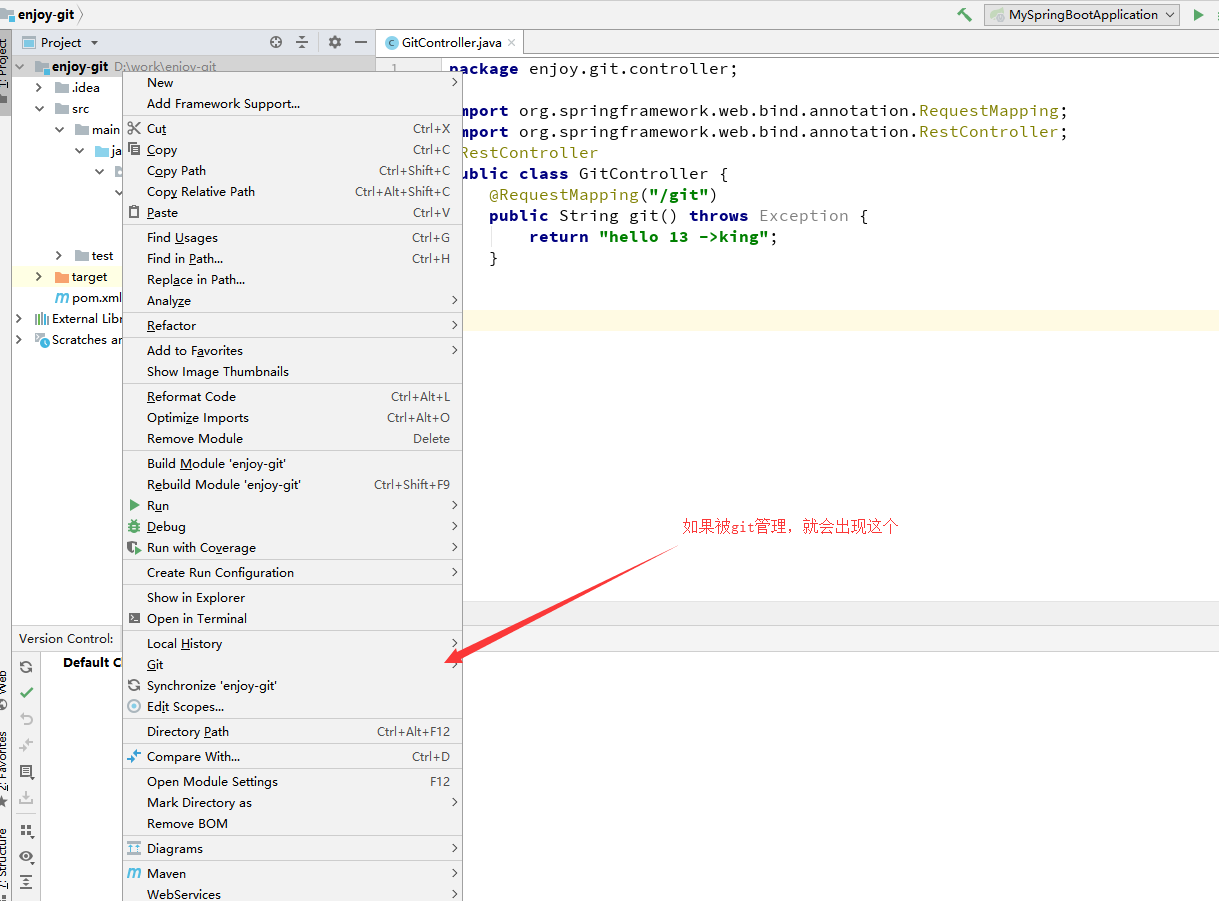
热更新分支

从develop分支拉取

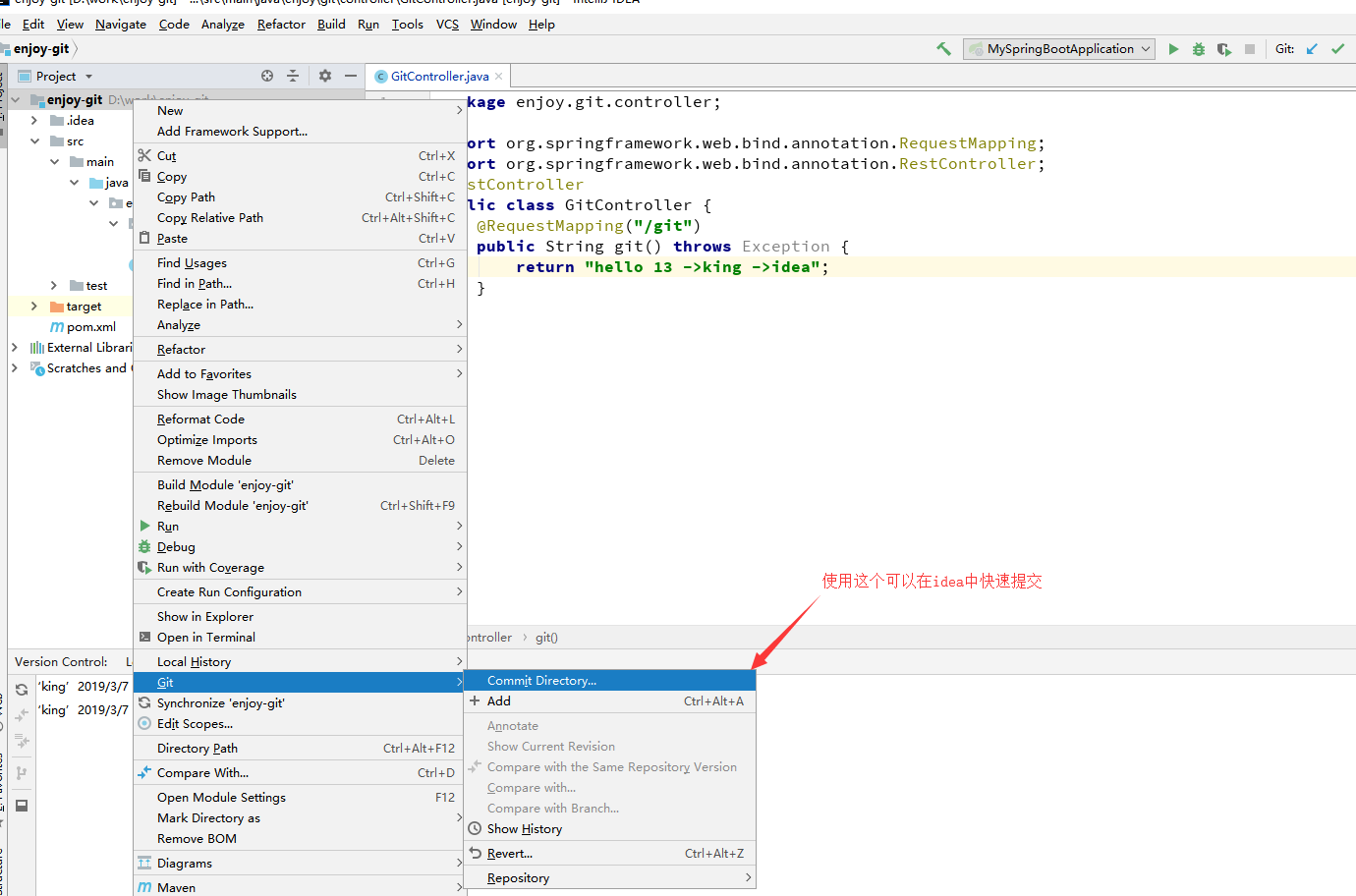
用于紧急修复上线版本的问题

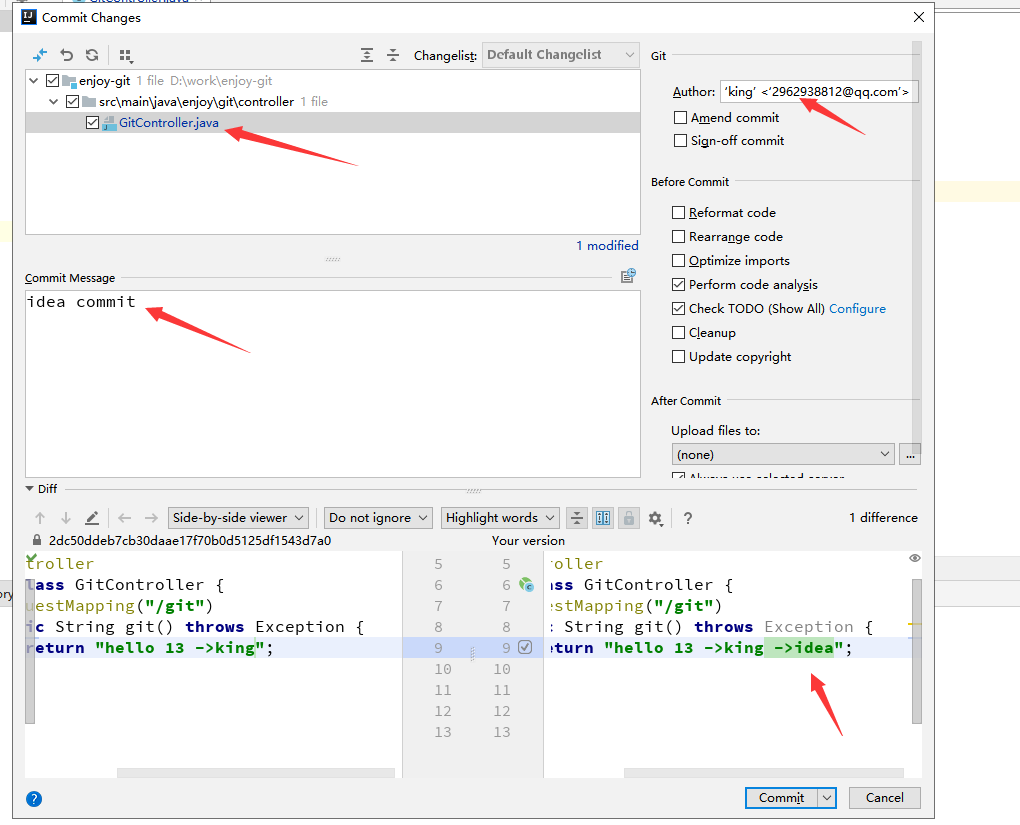
修复后打tag并合入master和develop

### GIT在idea中模式(eclipse下也是雷同)

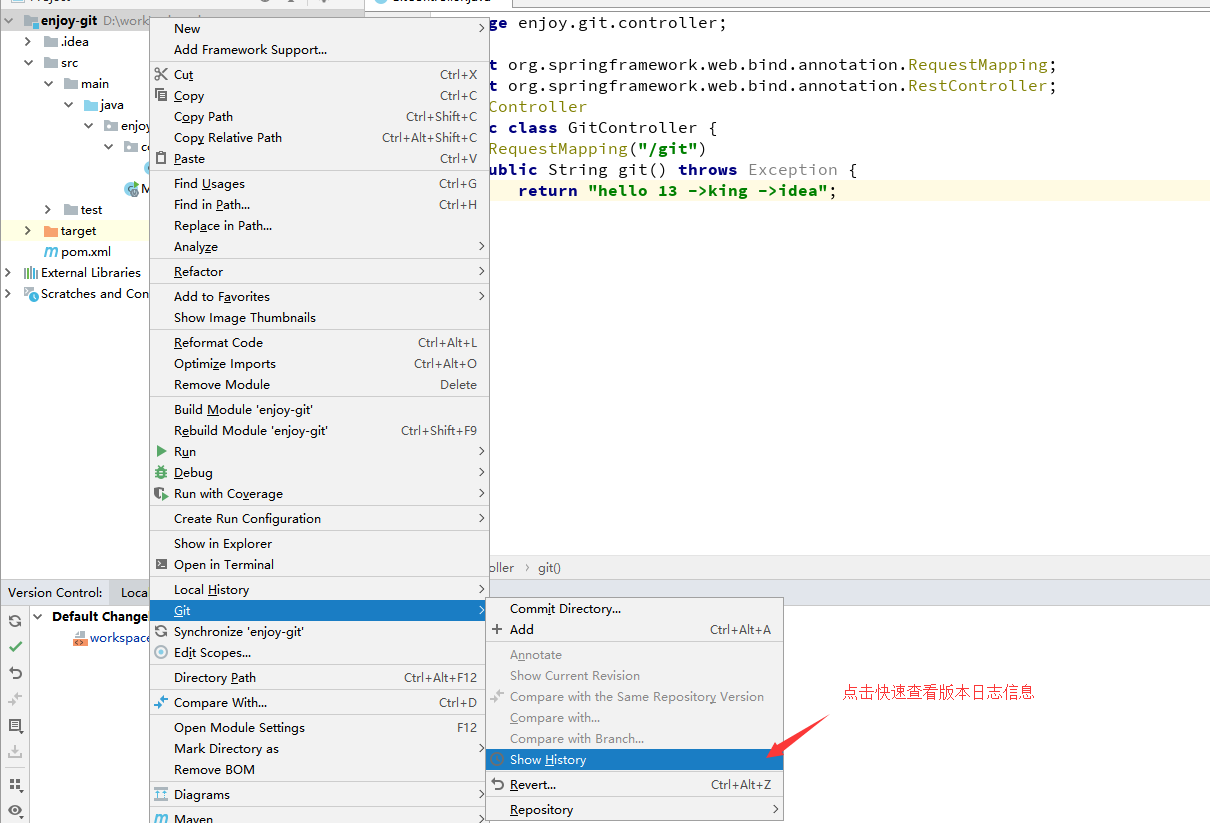


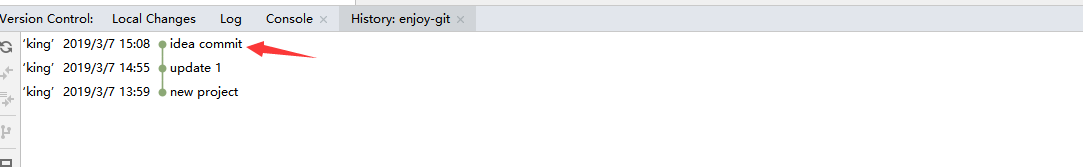
被管理后，就可进行版本提交，比如我修改了一个类，要进行提交





提交完后，也可以快速查看版本修改的日志信息





### GITLab

https://about.gitlab.com/install/#centos-7