Xzbots及廖雪峰git

廖雪峰git：

Git bash常用命令：

<https://mp.weixin.qq.com/s/VNSLznUuo1jHfYl__emrAw>

* 安装及设置git

安装：sudo apt-get install git -y

设置：<https://www.cnblogs.com/superGG1990/p/6844952.html>

（生成密钥对那就是邮箱，未设置前没有权限下载github上code）

警告：搜索问题，在某个文件后面加上ip即可

* git remote add origin [git@github.com:yilu-zhang/code.git](mailto:git@github.com:yilu-zhang/code.git) ：先在github上创建远程库，最好同名，再使用该命令与之关联，yilu-zhang：为github账户名，添加后，远程库的名字就是origin，这是Git默认的叫法，也可以改成别的，但是origin这个名字一看就知道是远程库。
* git remote rm origin：删除远程关联，更改远程连接库需先删除再连接。
* git remote: 查看远程库的信息
* git remote -v : 查看更详细远程库的信息
* git push -u origin master：把本地库的所有内容推送到远程库上，用git push命令，实际上是把当前分支master推送到远程。
* git clone git@github.com:michaelliao/gitskills.git：从远程库克隆到本地，可以下载别人的开源库。
* git pull ： 抓取远程的新提交
* git pull --rebase origin master：进行代码合并。

注意：远程库有的本地没有时，上传会出错，这时需先将远程与本地合并。

* q : 当出现不自动返回输入命令时，按q返回
* mkdir code1：在当前文件夹下创建文件夹code1。
* cd code1：从当前文件夹跳到code1，空缺跳到根目录。
* pwd：查看当前所处文件夹。
* git init：命令把这个目录变成Git可以管理的仓库，如果你没有看到.git目录，那是因为这个目录默认是隐藏的，用ls -ah命令就可以看见。
* git config --global user.name "yiluzhang"：设置名称。
* git config --global user.email "861280152@qq.com":设置邮箱。
* 在Windows和Linux环境上安装Git并配置同一账户

<https://www.cnblogs.com/superGG1990/p/6844952.html>

（注意秘钥是id\_rsa.pub.pub）

<https://blog.csdn.net/seaee/article/details/51151148>

* git add：把文件添加到仓库，注意，可反复多次使用，添加多个文件。如git add readme.txt。
* git commit：把文件提交到仓库，如：gitcommit-m"wrote a readme file（说明）"；
* git status：仓库当前的状态；
* git diff:查看更改。git diff HEAD -- readme.txt命令可以查看工作区和版本库里面最新版本的区别。
* git log：查看日志，git log --pretty=oneline：查看简化的日志。
* git reset --hard HEAD^:版本回退，尝试回退时，最好先建立分支再回退。用HEAD表示当前版本，也就是最新的提交1094adb...（注意我的提交ID和你的肯定不一样），上一个版本就是HEAD^，上上一个版本就是HEAD^^，当然往上100个版本写100个^比较容易数不过来，所以写成HEAD~100。
* git reset --hard 1094a：回到回退前版本，只要上面的命令行窗口还没有被关掉，你就可以顺着往上找啊找啊，找到那个append GPL的commit id是1094adb...，于是就可以指定回到未来的某个版本：
* git reflog：记录你的每一次命令。
* git checkout -- file：命令中的--很重要，没有--，就变成了“切换到另一个分支”的命令，另外--两端空格不能忘

场景1：当你改乱了工作区某个文件的内容，想直接丢弃工作区的修改时，用命令git checkout -- file。

场景2：当你不但改乱了工作区某个文件的内容，还添加到了暂存区时，想丢弃修改，分两步，第一步用命令git reset HEAD <file>，就回到了场景1，第二步按场景1操作。

场景3：已经提交了不合适的修改到版本库时，想要撤销本次提交，参考[版本回退](https://www.liaoxuefeng.com/wiki/0013739516305929606dd18361248578c67b8067c8c017b000/0013744142037508cf42e51debf49668810645e02887691000)一节，不过前提是没有推送到远程库。

* ssh-keygen -t rsa -C "861280152@qq.com"：创建github密钥，使用默认设置按回车就好。
* 由于远程库是空的，我们第一次推送master分支时，加上了-u参数，Git不但会把本地的master分支内容推送的远程新的master分支，还会把本地的master分支和远程的master分支关联起来，在以后的推送或者拉取时就可以简化命令git push origin master ，把本地master分支的最新修改推送至GitHub。

注意：当你第一次使用Git的clone或者push命令连接GitHub时，会得到一个警告，需手动输入yes，否者会连接失败。

* rm <file>和git rm <file>:两者都是删除文件操作。前者不添加到暂存区，可按上面场景1恢复，也可使用git rm和git commit提交删除；后者会提交到暂存区，可按上面场景2恢复，也可使用git commit提交删除。
* git rm -r test： (删除文件夹)
* ls：用来显示目标列表，在Linux中是使用率较高的命令。ls命令的输出信息可以进行彩色加亮显示，以分区不同类型的文件。
* git branch：查看分支，当前分支前面会标一个\*号。
* git branch <name>：创建name分支。
* git checkout <name>：切换name分支。
* git checkout -b <name>：创建+切换到name分支，相当于以上两句。
* git merge <name>：合并某分支到当前分支。
* git branch -d <name>：删除分支。
* git branch -D <name> ：如果要丢弃一个没有被合并过的分支，先切换到另一个分支，可以通过它强行删除。
* git log --graph --pretty=oneline --abbrev-commit：查看各分支，git log --graph查看详细分支图。
* git merge --no-ff -m "merge with no-ff" dev：合并分支时，加上--no-ff参数就可以用普通模式合并，合并后的历史有分支，能看出来曾经做过合并，而fast forward合并就看不出来曾经做过合并。
* git stash：暂时隐藏，修复bug时，我们会通过创建新的bug分支进行修复，然后合并，最后删除；
* 当手头工作没有完成时，先把工作现场git stash一下，然后去修复bug，修复后，再git stash pop，回到工作现场。
* 工作现场还在，Git把stash内容存在某个地方了，但是需要恢复一下，有两个办法：
* 一是用git stash apply恢复，但是恢复后，stash内容并不删除，你需要用git stash drop来删除，可以多次stash，恢复的时候，先用git stash list查看，然后恢复指定的stash，用命令git stash apply stash@{0} ；
* 另一种方式是用git stash pop，恢复的同时把stash内容也删了：
* git rebase : 把本地未push的分叉提交历史整理成直线；目的是使得我们在查看历史提交的变化时更容易，因为分叉的提交需要三方对比。
* git tag <tagname> : 用于新建一个标签，默认为HEAD，git tag <tagname> commit id也可以指定一个commit id；
* git tag -a <tagname> -m "blablabla..." : 可以指定标签信息；
* git tag : 可以查看所有标签。
* git show <tagname> : 查看标签信息

注意：标签总是和某个commit挂钩。如果这个commit既出现在master分支，又出现在dev分支，那么在这两个分支上都可以看到这个标签。

* git push origin <tagname> : 推送一个本地标签；
* 命令git push origin --tags : 推送全部未推送过的本地标签；
* 命令git tag -d <tagname> : 删除一个本地标签；
* 创建的标签都只存储在本地，不会自动推送到远程。所以，打错的标签可以在本地安全删除。
* git push origin :refs/tags/<tagname> : 删除一个远程标签。
* git config --global color.ui true：让Git显示颜色，会让命令输出看起来更醒目
* git add -f App.class ： 强制添加被忽略的文件
* git check-ignore -v App.class ： 检查那个.gitignore哪个规则写错了
* git config --global alias.st status ：配置别名，用st就表示status，加上--global是针对当前用户起作用的，如果不加，那只针对当前的仓库起作用。
* cat .gitconfig：查看.gitconfig中配置内容，包括users和alias（别名），可直接在文件中增加/删除别名，注意使用该命令时先用cd ~切换到Git配置文件放在用户主目录
* git config --global --unset alias.sta ：删除全局配置别名cm
* git config --unset alias.cm ：删除当前工作区的配置别名cm
* cat .git/config ： 查看当前工作区配置内容

[alias]

st = status

lggr = log --graph --pretty=oneline --abbrev-commit

* git config --global core.autocrlf false：window提交文件出现“resynchronize”（重新同步）错误时，如果为真，则让git检查转换CRLF在行尾转换活动时是否可逆。Git将验证命令是否直接或间接修改工作树中的文件。例如，提交一个文件，然后签出同一个文件，应该会在工作树中生成原始文件。如果不是当前core设置的情况。独裁者，git将拒绝该文件。可以将变量设置为“warn”，在这种情况下，git只会警告不可逆转换，但会继续操作。

1. 小技巧及错误：
2. 点上下键可以选择其他窗口命令,即使窗口关闭后也能用；b、git bash中选中后不用点复制直接粘贴就行。
3. 本地新建的分支如果不推送到远程，对其他人就是不可见的；
4. 报错“command not found”：直接从word复制的文本格式不对
5. 添加远程库时，下划线或其他字母打错也是可以绑定的，然后就会造成报错
6. 团队协作：

从本地推送分支，使用git push origin branch-name，如果推送失败，先用git pull抓取远程的新提交；

在本地创建和远程分支对应的分支，使用git checkout -b branch-name origin/branch-name，本地和远程分支的名称最好一致；

如果git pull提示no tracking information，则说明本地分支和远程分支的链接关系没有创建，需建立本地分支和远程分支的关联，使用git branch --set-upstream branch-name origin/branch-name；

从远程抓取分支，使用git pull，如果有冲突，要先处理冲突。

另外，但是，并不是一定要把本地分支往远程推送，那么，哪些分支需要推送，哪些不需要呢？

master分支是主分支，因此要时刻与远程同步；

dev分支是开发分支，团队所有成员都需要在上面工作，所以也需要与远程同步；

bug分支只用于在本地修复bug，就没必要推到远程了，除非老板要看看你每周到底修复了几个bug；

feature分支是否推到远程，取决于你是否和你的小伙伴合作在上面开发。

总之，就是在Git中，分支完全可以在本地自己藏着玩，是否推送，视你的心情而定！

1. 图片、视频这些二进制文件，虽然也能由版本控制系统管理，但没法跟踪文件的变化，只能把二进制文件每次改动串起来，也就是只知道图片从100KB改成了120KB，但到底改了啥，版本控制系统不知道，也没法知道。不幸的是，Microsoft的Word格式是二进制格式，因此，版本控制系统是没法跟踪Word文件的改动的，前面我们举的例子只是为了演示，如果要真正使用版本控制系统，就要以纯文本方式编写文件。
2. 千万不要使用Windows自带的记事本编辑任何文本文件。原因是Microsoft开发记事本的团队使用了一个非常弱智的行为来保存UTF-8编码的文件，他们自作聪明地在每个文件开头添加了0xefbbbf（十六进制）的字符，你会遇到很多不可思议的问题，比如，网页第一行可能会显示一个“?”，明明正确的程序一编译就报语法错误，等等，都是由记事本的弱智行为带来的。建议你下载Notepad++代替记事本，不但功能强大，而且免费！记得把Notepad++的默认编码设置为UTF-8 without BOM即可。
3. 初始化一个Git仓库，使用git init命令。
4. 添加文件到Git仓库，分两步：

第一步，使用命令git add ，注意，可反复多次使用，添加多个文件；

第二步，使用命令git commit，完成。

1. 如果你使用Windows系统，为了避免遇到各种莫名其妙的问题，请确保目录名（包括父目录）不包含中文。
2. 工作区：一个被给git init的文件夹为一个工作区；

版本库：初始化后隐藏的.git文件，工作区文件共用一个版本库。包括暂存区（缓冲区，git add）和master（主干，HEAD为头指针，git commit）。三者关系可看作：工作区-暂存区-master。

为减小关联，可以一个项目放在一个文件夹内，另外，可文件夹嵌套git init，两个版本库相互独立，互不干扰，但需两边都需要提交。

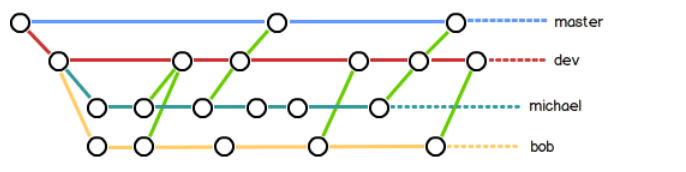
1. 你又理解了Git是如何跟踪修改的，每次修改，如果不用git add到暂存区，那就不会加入到commit中。
2. id\_rsa是私钥，不能泄露出去，id\_rsa.pub是公钥，可以放心地告诉任何人。
3. 最后友情提示，在GitHub上免费托管的Git仓库，任何人都可以看到喔（但只有你自己才能改）。所以，不要把敏感信息放进去。
4. 如果你不想让别人看到Git库，有两个办法，一个是交点保护费，让GitHub把公开的仓库变成私有的，这样别人就看不见了（不可读更不可写）。另一个办法是自己动手，搭一个Git服务器，因为是你自己的Git服务器，所以别人也是看不见的。这个方法我们后面会讲到的，相当简单，公司内部开发必备。
5. Git支持多种协议，包括https，地址是https://github.com/michaelliao/gitskills.git,但通过ssh支持的原生git协议速度最快。
6. SVN是早于git的版本管理软件。
7. 当Git无法自动合并分支时，就必须首先解决冲突。解决冲突后，再提交，合并完成。解决冲突就是把Git合并失败的文件手动编辑为我们希望的内容，再提交。
8. 在实际开发中，我们应该按照几个基本原则进行分支管理：

首先，master分支应该是非常稳定的，也就是仅用来发布新版本，平时不能在上面干活；

那在哪干活呢？干活都在dev分支上，也就是说，dev分支是不稳定的，到某个时候，比如1.0版本发布时，再把dev分支合并到master上，在master分支发布1.0版本；

你和你的小伙伴们每个人都在dev分支上干活，每个人都有自己的分支，时不时地往dev分支上合并就可以了。

自己总结：三线管理，master（版本管理）-dev（次总线），个人线（个人）。



1. 开发一个新feature，最好新建一个分支；
2. 在GitHub上，可以任意Fork开源仓库；自己拥有Fork后的仓库的读写权限；可以推送pull request给官方仓库来贡献代码。

19.使用GitHub时，国内的用户经常遇到的问题是访问速度太慢，有时候还会出现无法连接的情况（原因你懂的）。

如果我们希望体验Git飞一般的速度，可以使用国内的Git托管服务——[码云](https://gitee.com/)（[gitee.com](https://gitee.com/)）。

和GitHub相比，码云也提供免费的Git仓库。此外，还集成了代码质量检测、项目演示等功能。对于团队协作开发，码云还提供了项目管理、代码托管、文档管理的服务，5人以下小团队免费。

20.忽略某些文件时，需要编写.gitignore；

.gitignore文件本身要放到版本库里，并且可以对.gitignore做版本管理！

忽略文件的原则是：

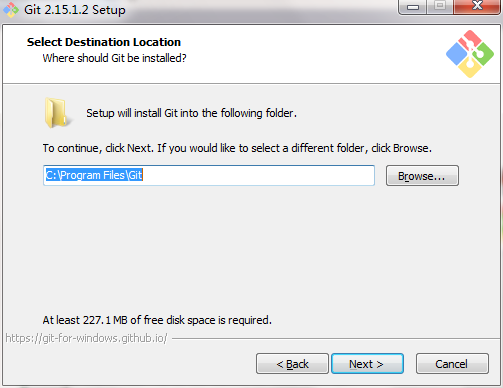
1. 忽略操作系统自动生成的文件，比如缩略图等；
2. 忽略编译生成的中间文件、可执行文件等，也就是如果一个文件是通过另一个文件自动生成的，那自动生成的文件就没必要放进版本库，比如Java编译产生的.class文件；
3. 忽略你自己的带有敏感信息的配置文件，比如存放口令的配置文件。

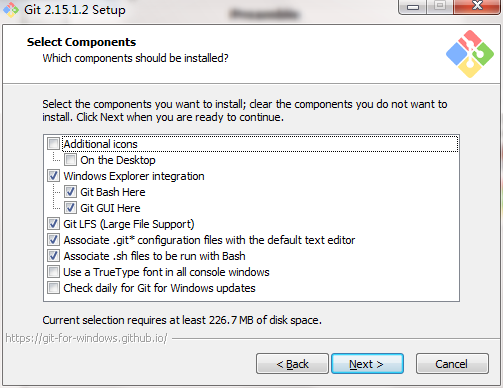
使用Windows的童鞋注意了，如果你在资源管理器里新建一个.gitignore文件，它会非常弱智地提示你必须输入文件名，但是在文本编辑器里“保存”或者“另存为”就可以把文件保存为.gitignore了。

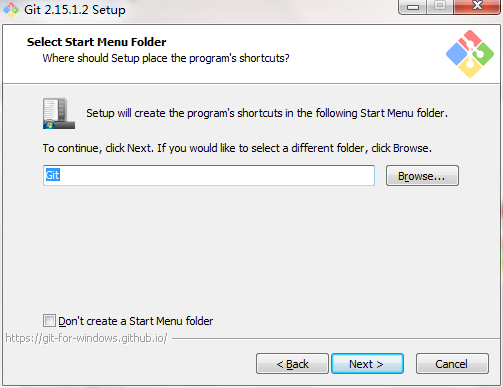
Xzbots git：

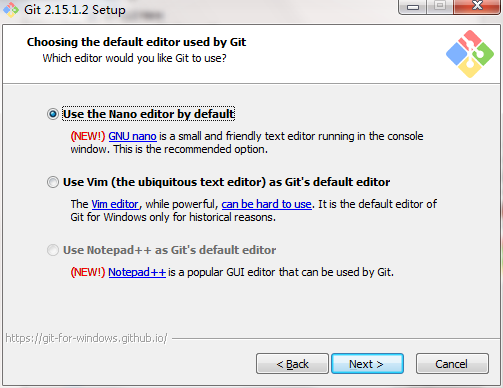
**软件安装**

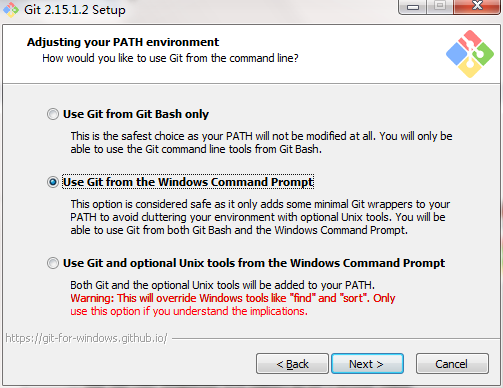


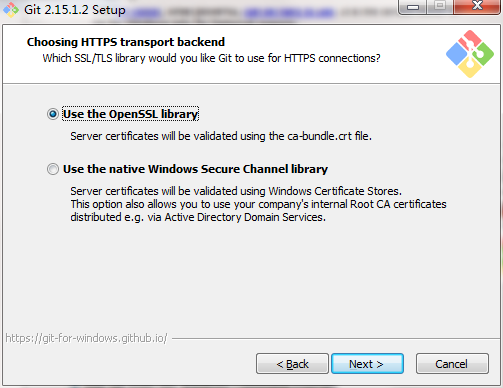


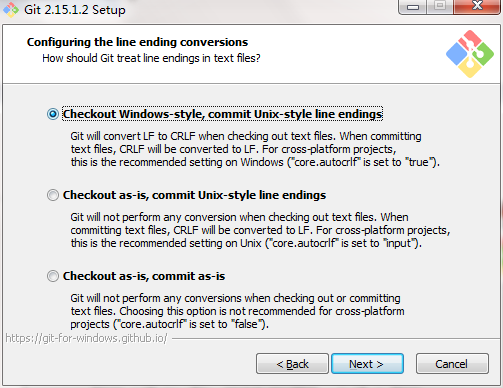


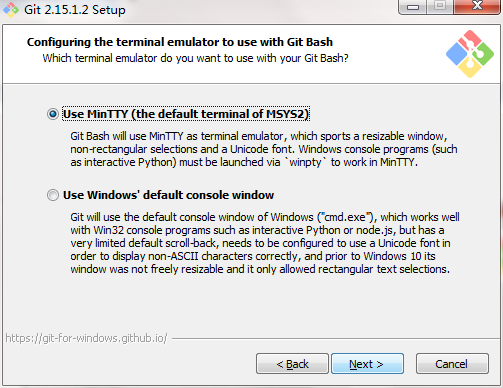


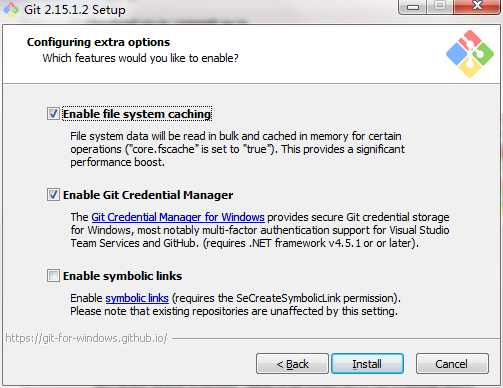


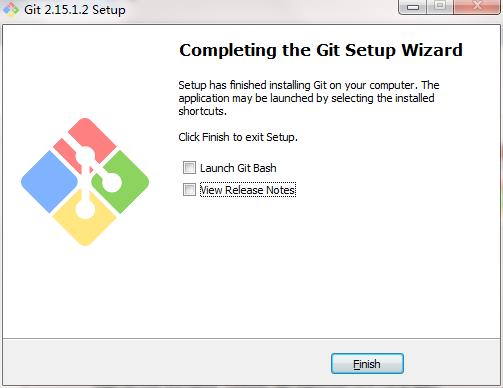












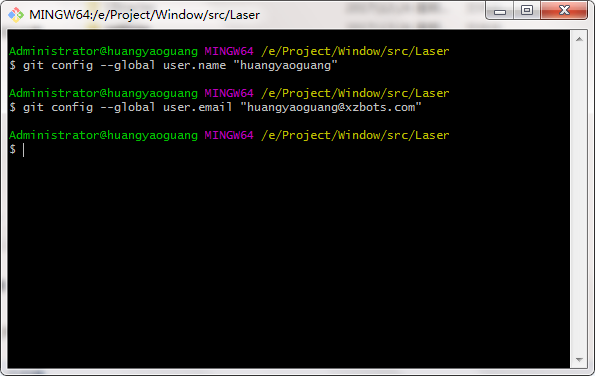
**软件配置**：

1. 在电脑任意一个地方点击右键-> Git Bash Here，在命令行输入以下命令，每条命令输入后都需要按回车键。

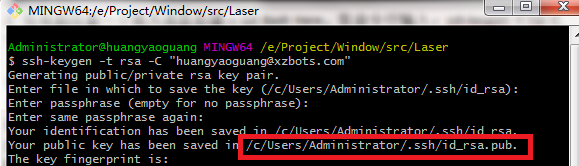
$ git config --global user.name "Your Name"

$ git config --global user.email ["email@example.com"](mailto:%22email@example.com%22)

"Your Name"和["email@example.com"](mailto:%22email@example.com%22)分别用个人的姓名和公司邮箱代替

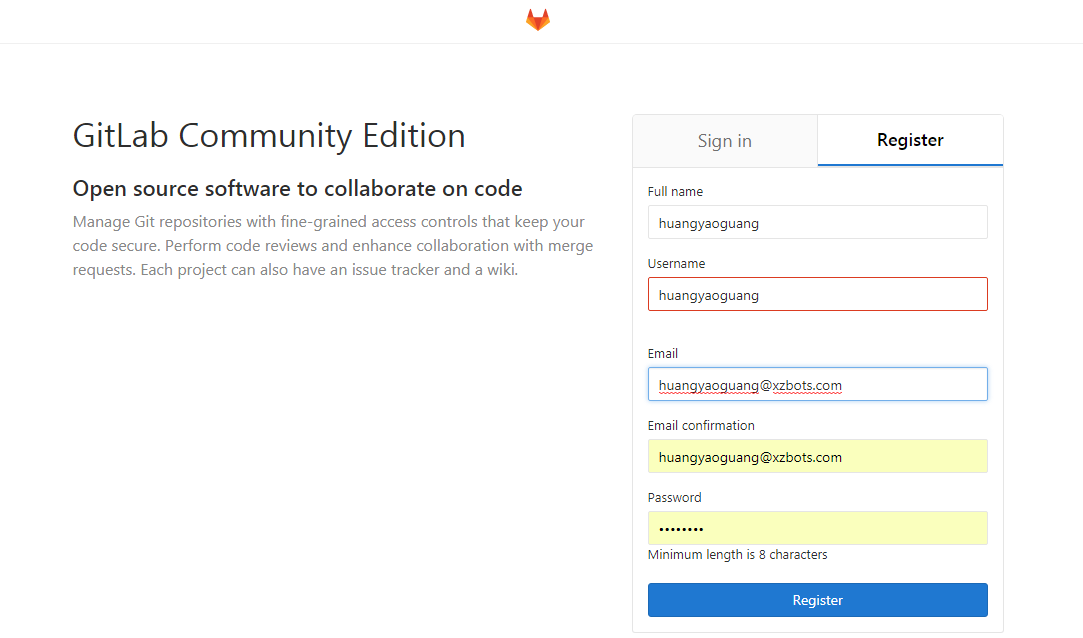


2. 在电脑任意一个地方点击右键-> Git Bash Here，在命令行输入：ssh-keygen -t rsa -C "XXX@xzbots.com"，然后连续按回车键即可。须记住id\_rsa.pub的路径，后续需要将id\_rsa.pu里的内容上传到服务器。

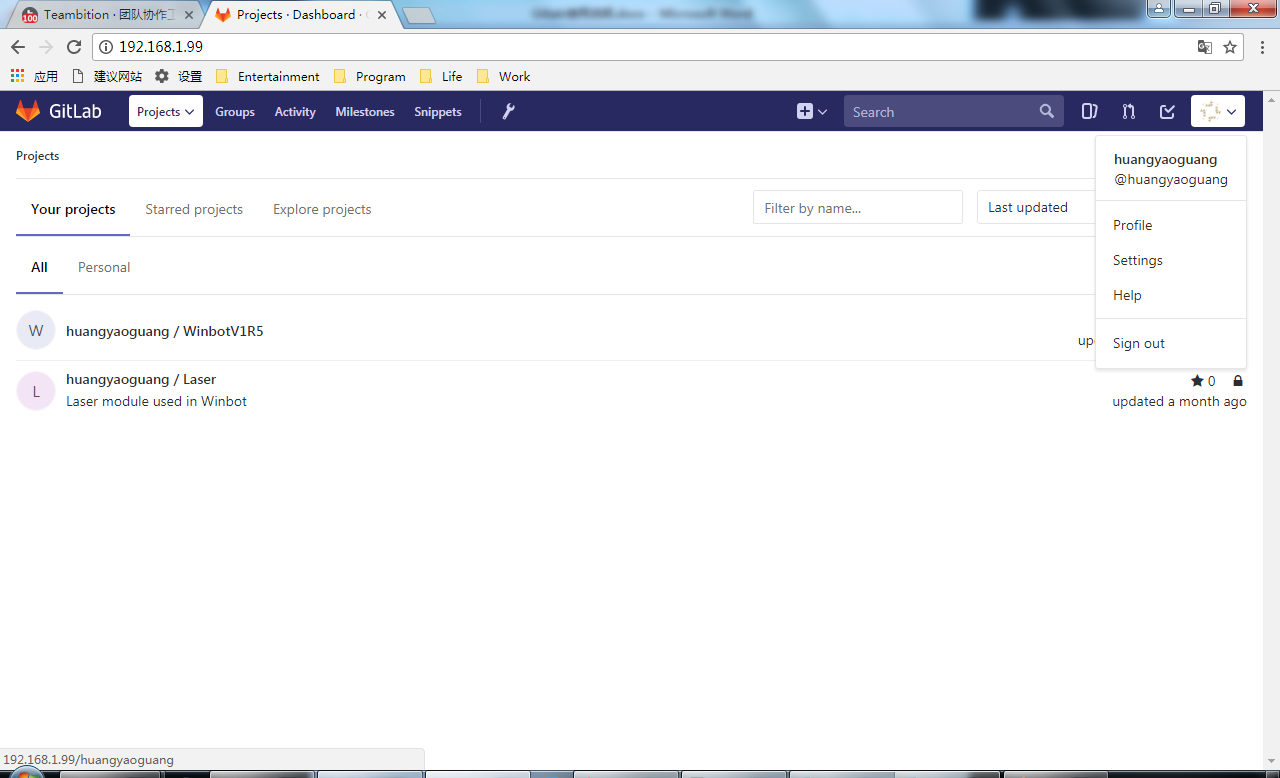


**Gitlab注册与设置**

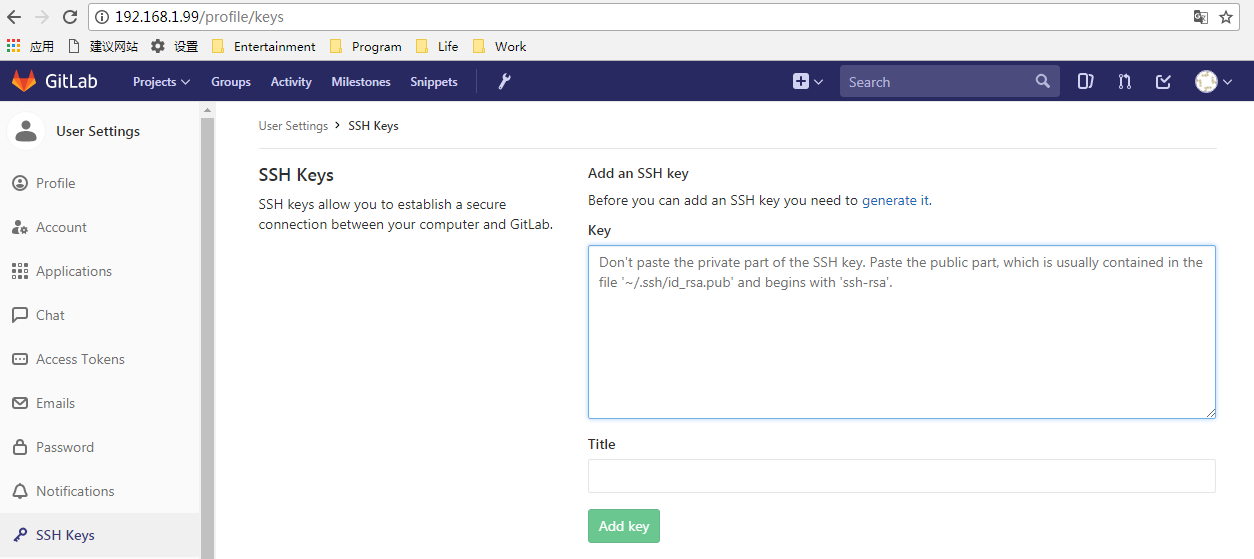
在浏览器输入“192.168.1.99”，然后点击Register，分别输入姓名，用户名和邮箱以及密码，邮箱使用公司邮箱。



点击右上角图标，然后点击Settings

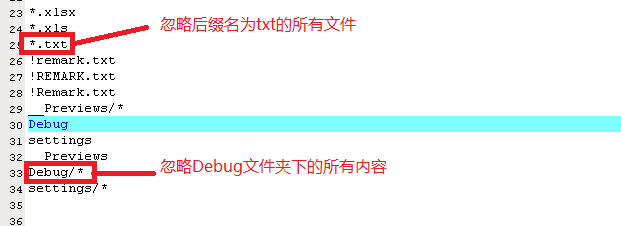


在左侧标题栏点击SSH-KEY，将之前生成的id\_rsa.pub用记事本打开，将里面的内容全部拷贝到Key底下的文本框，点击Add key，即可。



**Git本地使用**

1. 在待上传代码的文件夹里建立.gitignore，或者将现有的.gitignore拷贝到待上传代码的文件夹里，修改.gitignore的内容，将需要忽略（不需要上传到服务器的文件，通常是程序编译过程中产生的中间编译文件或者工程的历史文件等）的文件后缀名或者文件夹添加到.gitignore里面。添加忽略的文件或文件夹，可加快备份的时间，减少备份占用的硬盘空间。在.gitignore里，\*表示通配符，“\*.后缀名”表示忽略所有对应后缀名的文件；“文件夹/”表示忽略文件夹下的所有内容；文件或者文件夹前面加“！”，表示例外情况，即不忽略相应的文件或者文件夹。



2.点击鼠标右键，选择 Git Bash Here

3.在GIt Bash中输入 git init ，然后回车。

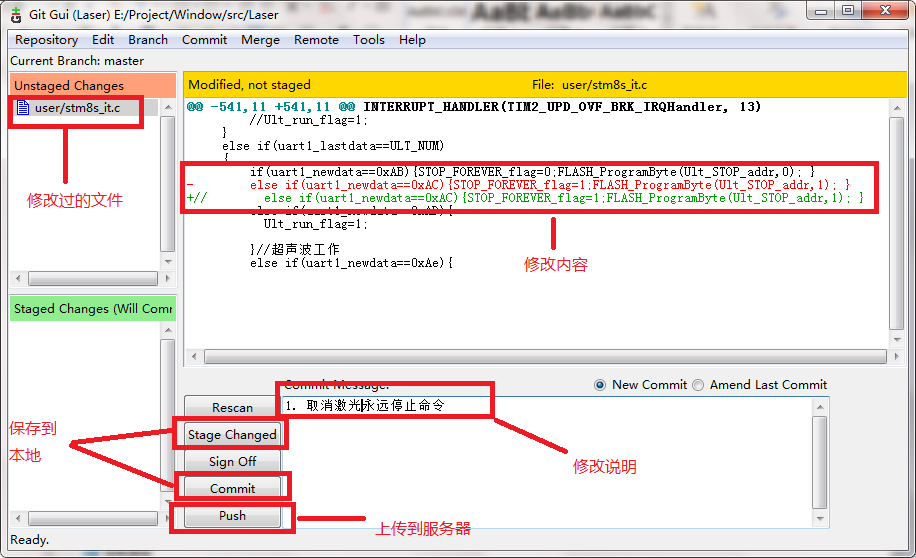


每次版本作出修改时，都需要保存修改。在保存之前，先检查修改的地方，确保无误后才提交修改。

5.在文件夹右键，选择 Git GUI Here

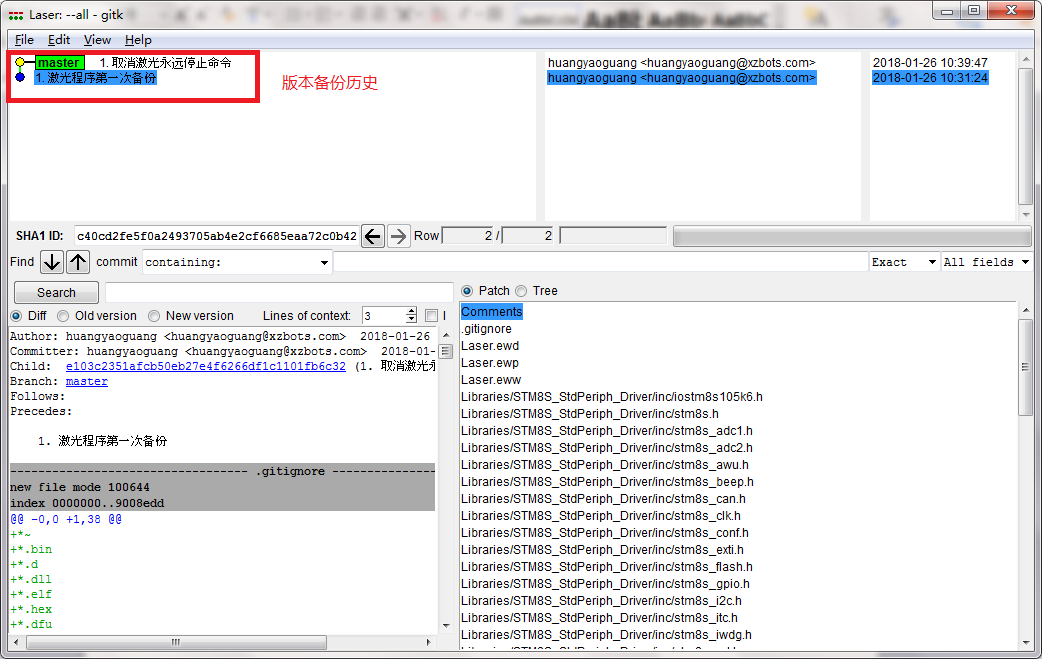
6. 先后点击Unstaged Changes里的各个修改过的文件，检查修改内容。（“-”表示删除，“+”表示增加）

7. 在Commit Messages中记录修改的说明，例如修复的Bug，增加的功能等等，然后点击Stage Changed，再点击Commit，即可。Push按键用于将版本上传到服务器，使用前续配置远端服务器。对于仅在本地使用的版本备份，则可不需要上传到服务器。需要上传到服务器的相关版本管理，文档下面会有相应的介绍。

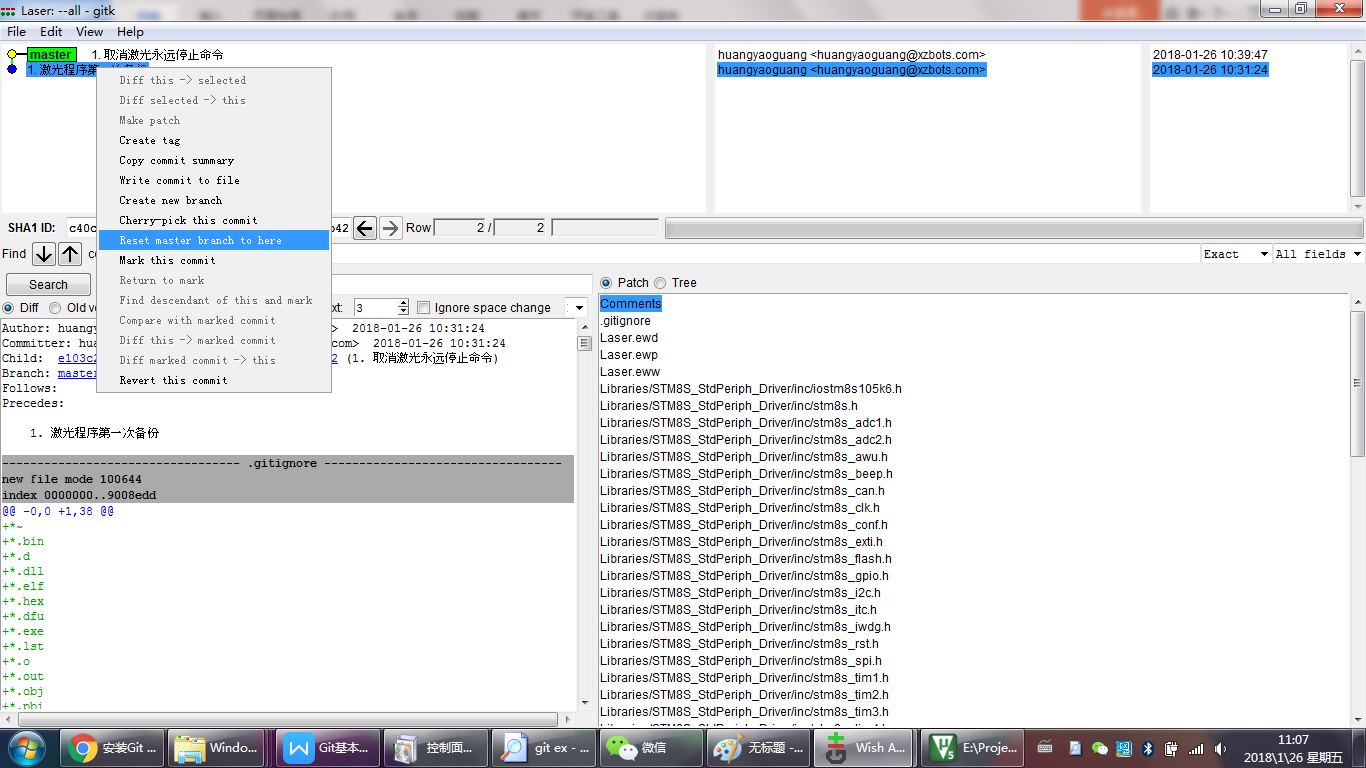


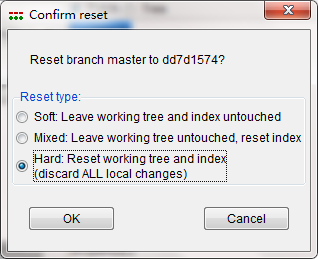
**本地版本管理**

1. 历史版本查看。右键->Git GUI->Repository->Visualize All Brance History，点击下图红框中各个修改条目，则可看到之前的备份修改记录

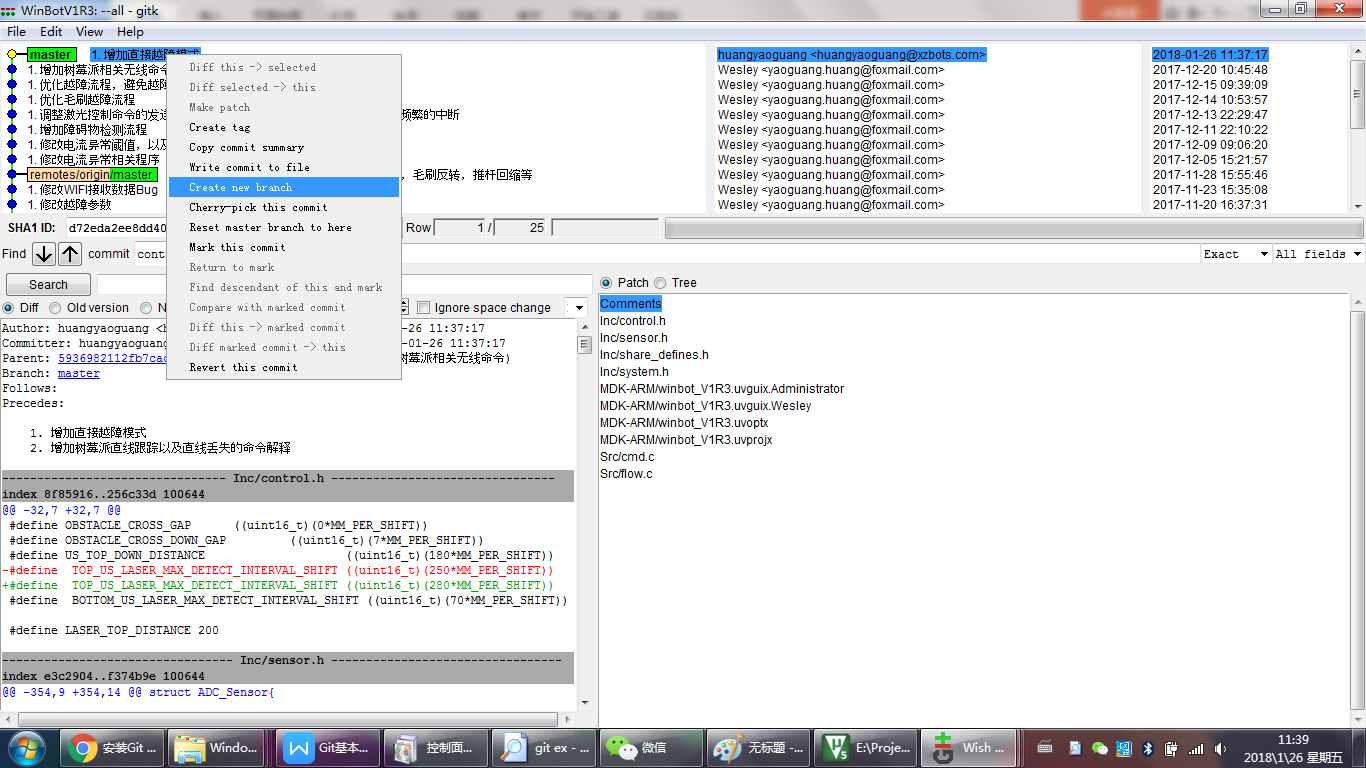


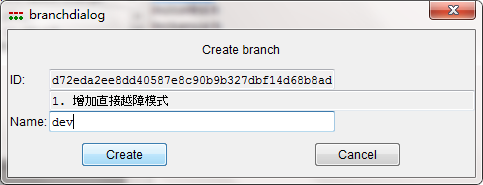
1. 版本回退。右键点击相应的修改记录，选择Reset master branch to here。在弹出的对话框选择Hard，则在此修改记录之前的修改将会被清除。



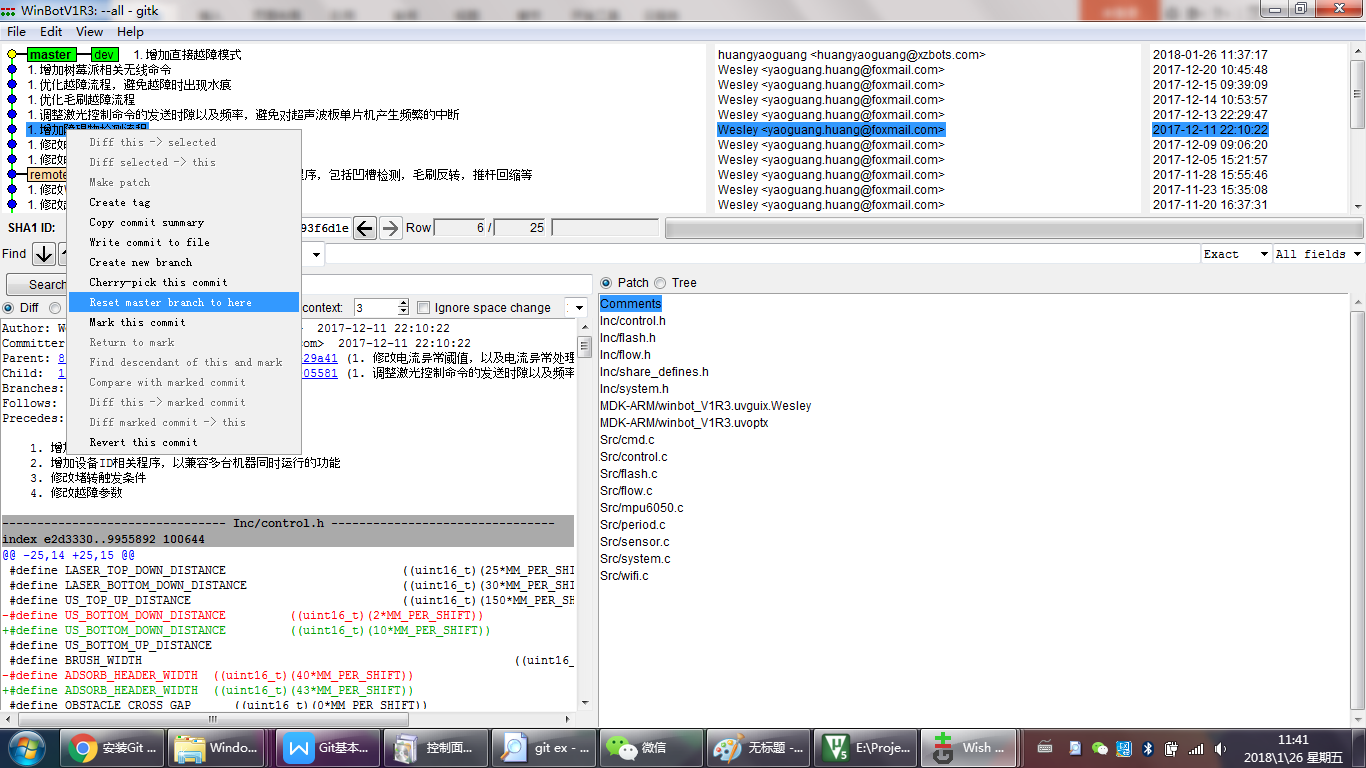


1. 如果回退是临时的，或者想保留最新的修改记录，则可按照一下操作进行：点击最新的修改记录，建立分支。

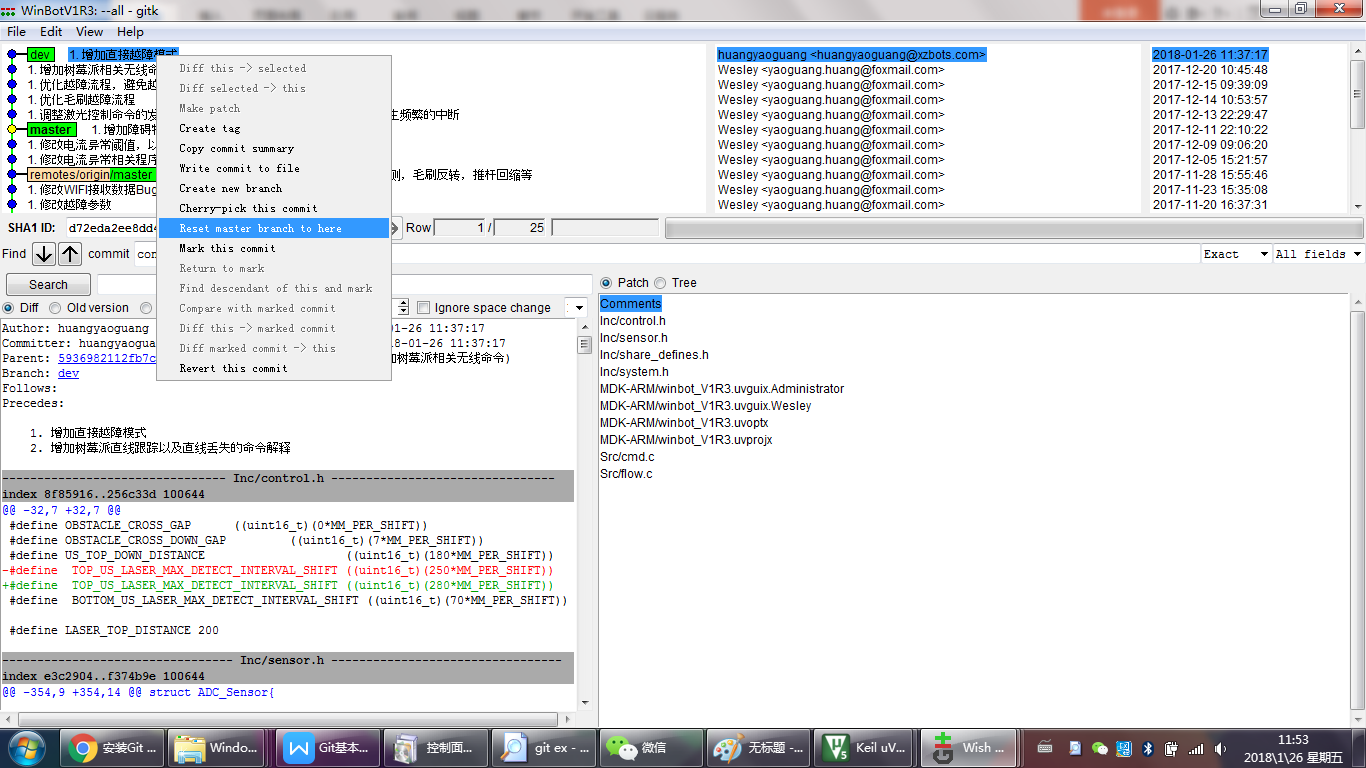




然后按照版本回退的方法，选择Hard模式回退到以前的修改记录。



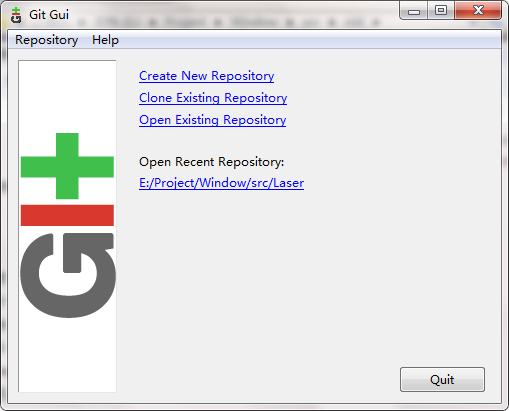
当需要还原到最新的备份记录时，则选择最新的备份记录，右键选择Reset master branch to here，选择Hard模式回到最新的版本。



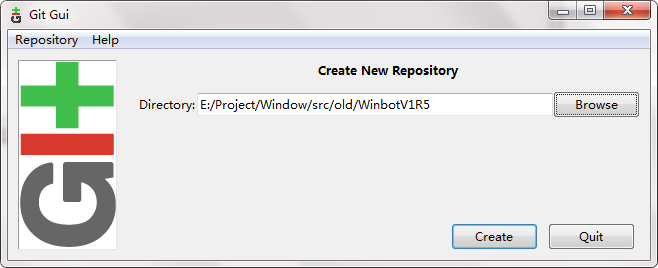
公司正式的项目，例如软件，PCB机械制图等，都需要管理员首先在Gitlab上创建相应的项目，并将项目相关的人员添加到项目的成员中，随后将项目对应的链接发给相关的人员。

注意：Gitlab生成的链接需要修改才能使用，例如Gitlab生成的链接为[git@192.168.1.99:huangyaoguang/WinbotV1R5.git](mailto:git@192.168.1.99:huangyaoguang/WinbotV1R5.git)，实际在Git中配置的地址应该为ssh://git@192.168.1.99/huangyaoguang/WinbotV1R5.git

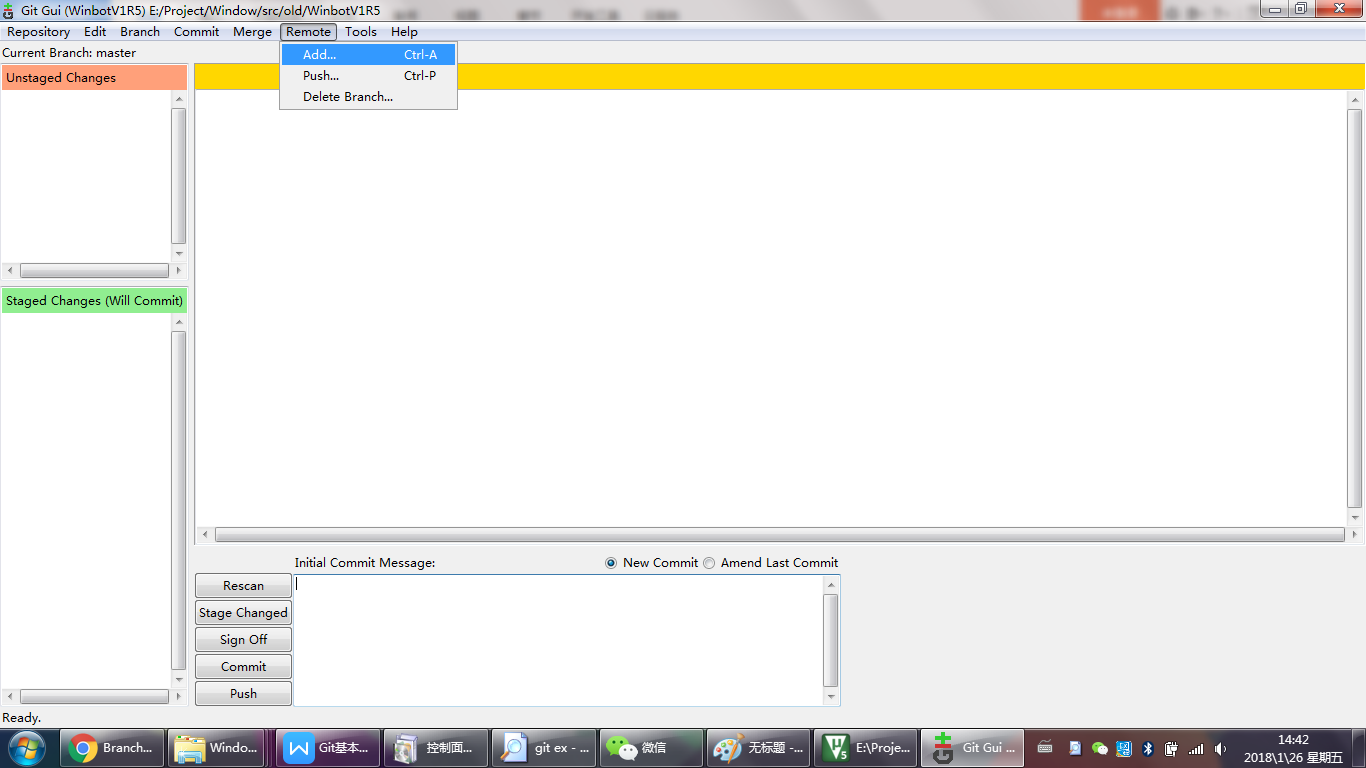
项目成员收到链接后，为新项目新建文件夹，然后右键->Git GUI Here，点击Create New Repository



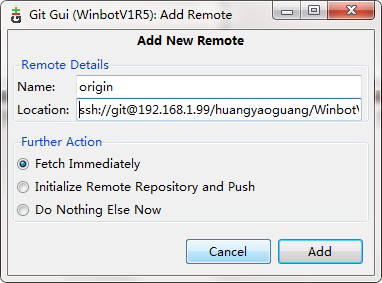
选择刚才新建的文件夹，然后点击Create。

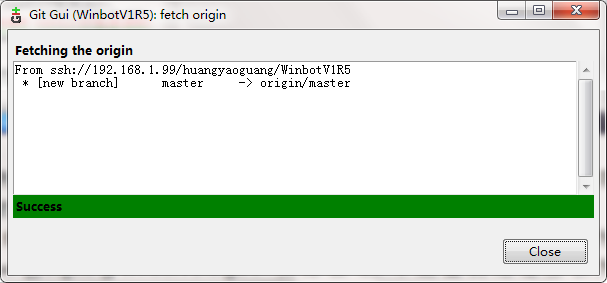


点击Remote->add...

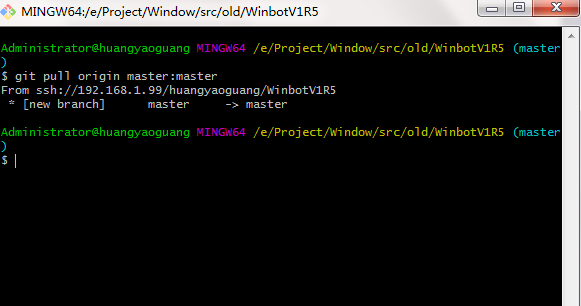


在弹出来的对话框中，name为服务器版本名字，通常填origin，Location为项目在Gitlab服务器的链接，填写完毕后点击Add。

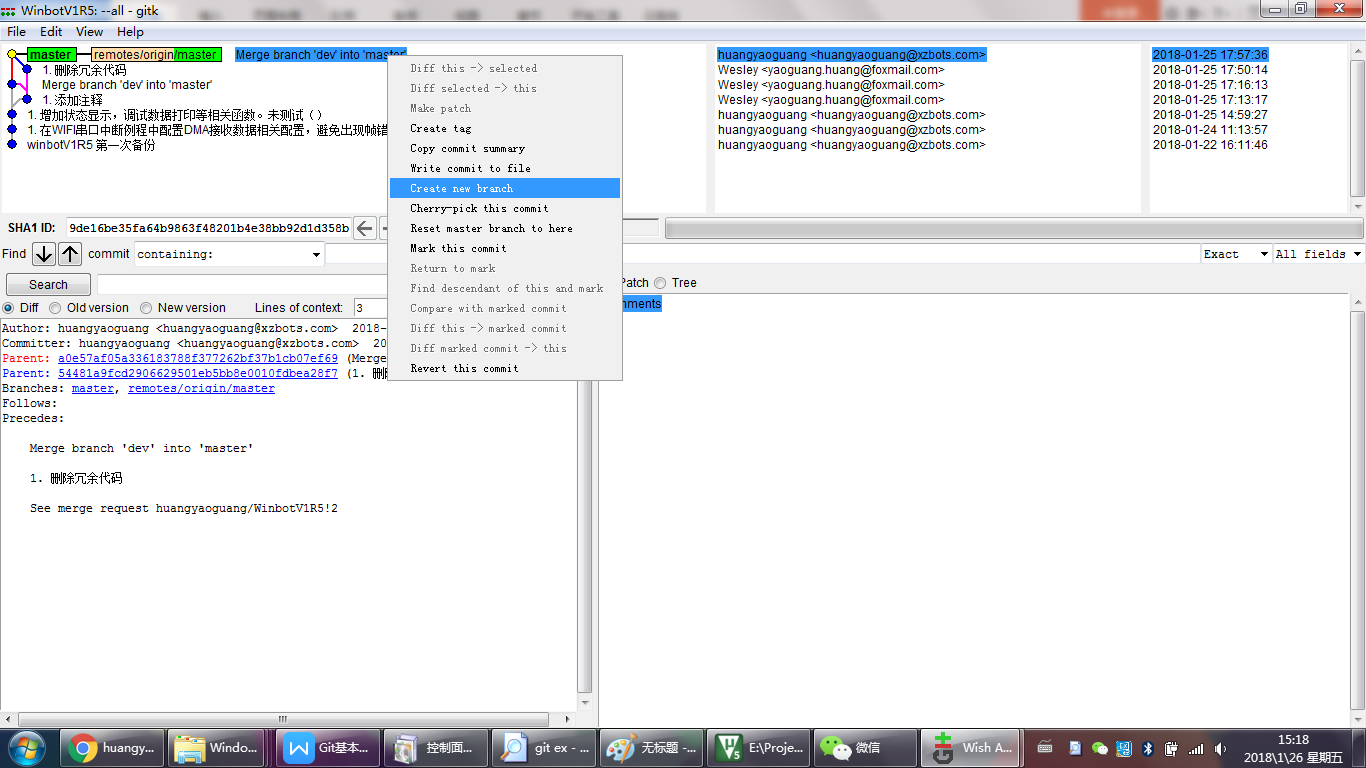


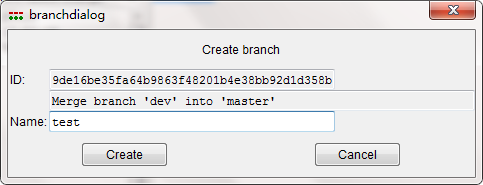


在Git Bash 输入 git pull origin master:master，然后回车。

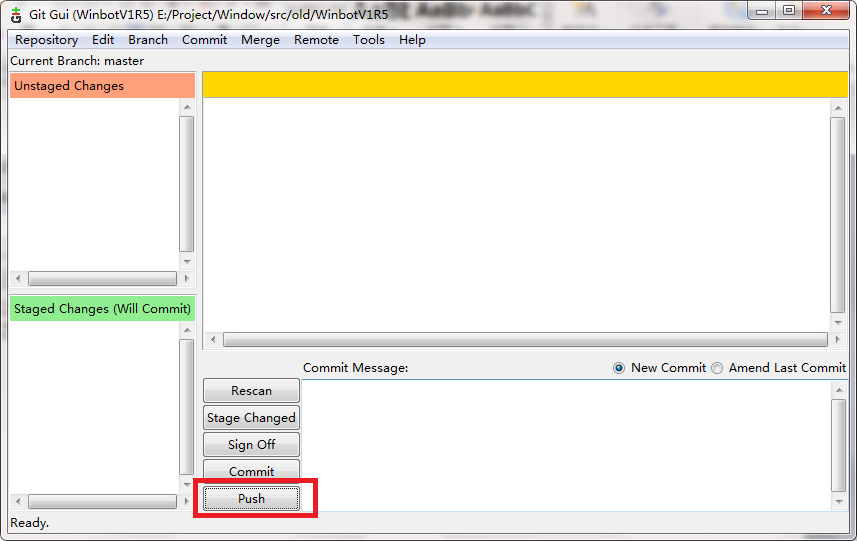


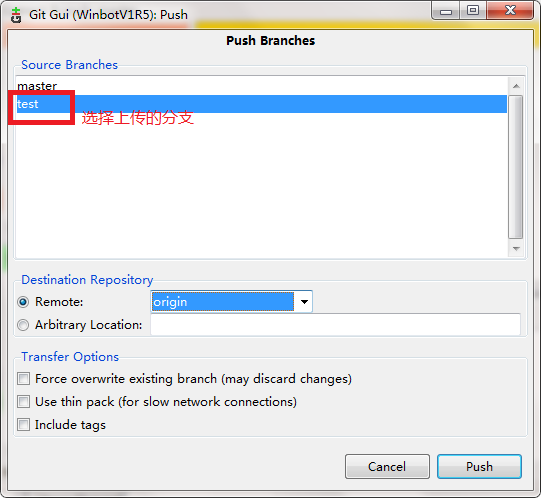
至此，已经将服务器上的版本拷贝到本地。为方便进行版本管理，所有本地的修改都要在分支上进行，当某个功能修改完成后，则在Gitlab上提交合并申请，管理员将新修改的内容合并到主干上。所以，当拷贝服务器的版本到本地后，参照文档之前介绍过的步骤在最新的修改记录上建立分支。

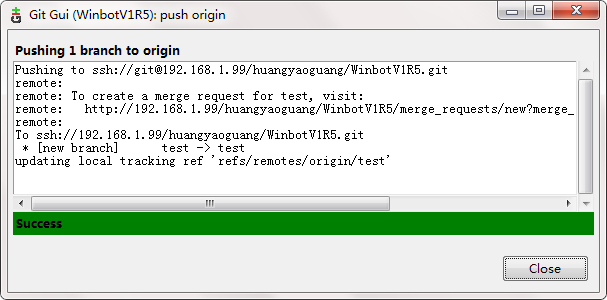


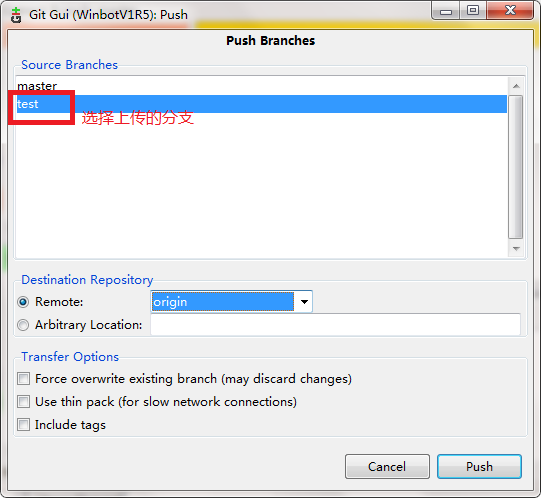


在本地建立分支之后，点击Push，在弹出的对话框选择上传的分支，再点击Push，则会在服务器上建立相应的分支。

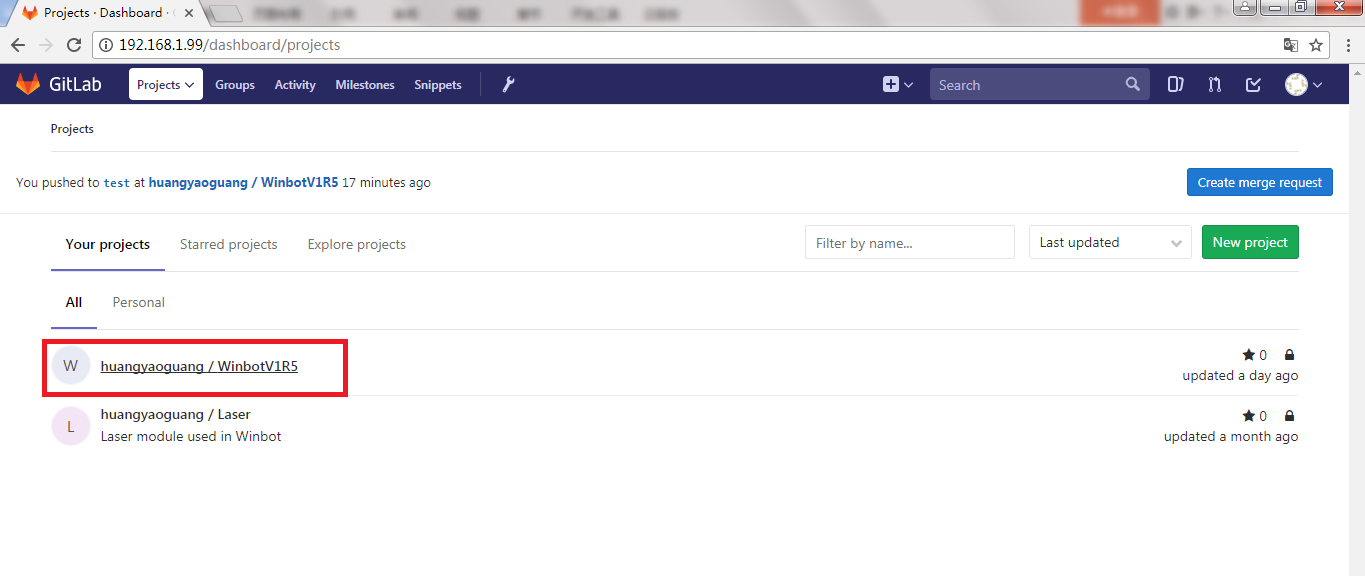




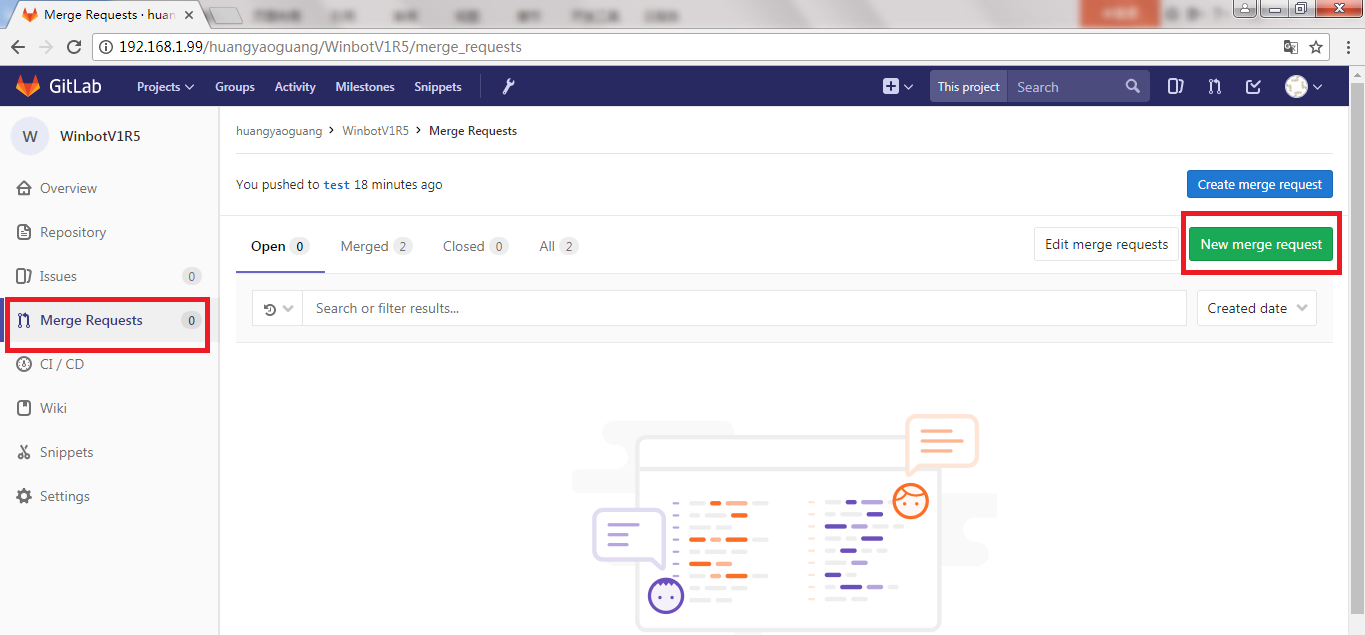




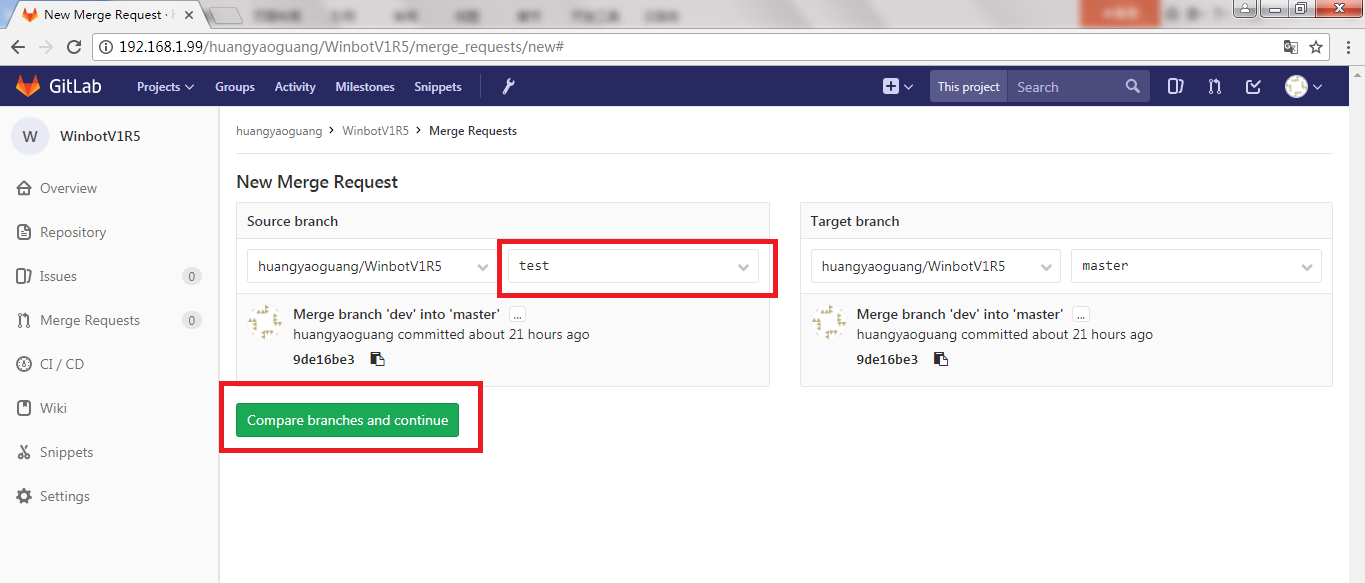
当项目完成一个新的功能或者实现阶段性成果时，则需要及时将分支合并到主干。登录Gitlab，选择相应的项目。



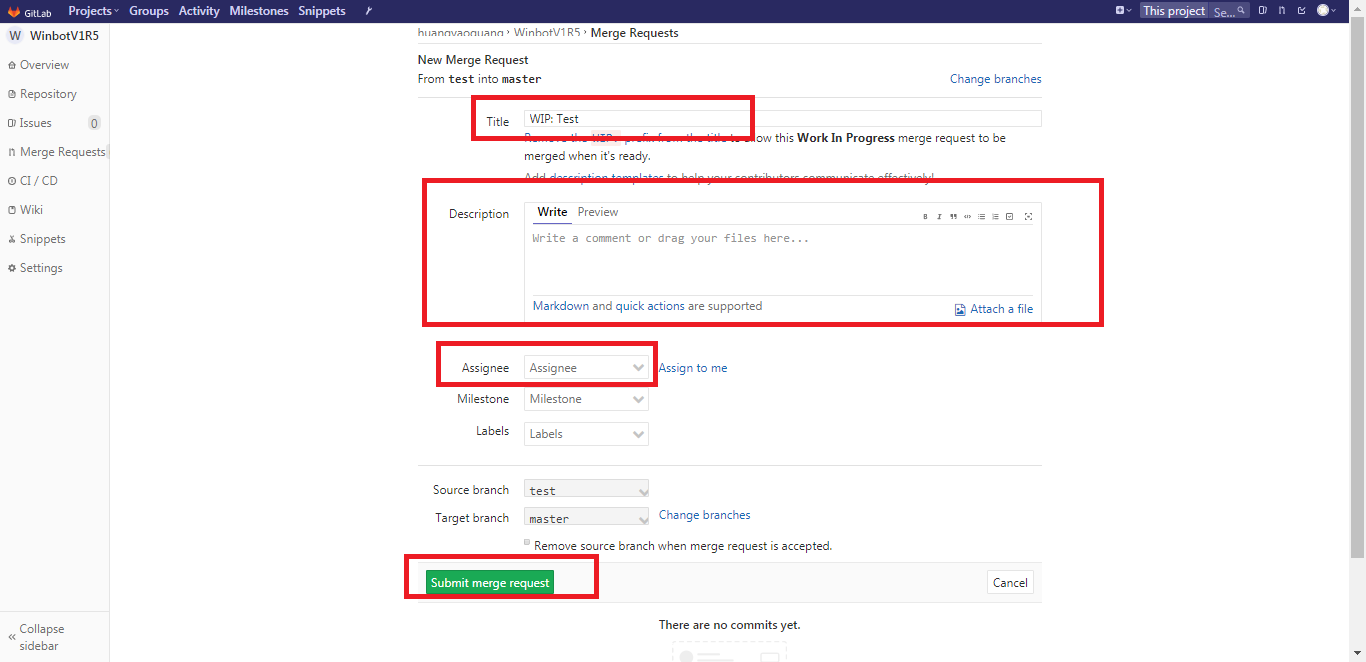
点击Merge Requests，然后再点击New Merge Request



选择合并的分支，点击Compare branches and continue。

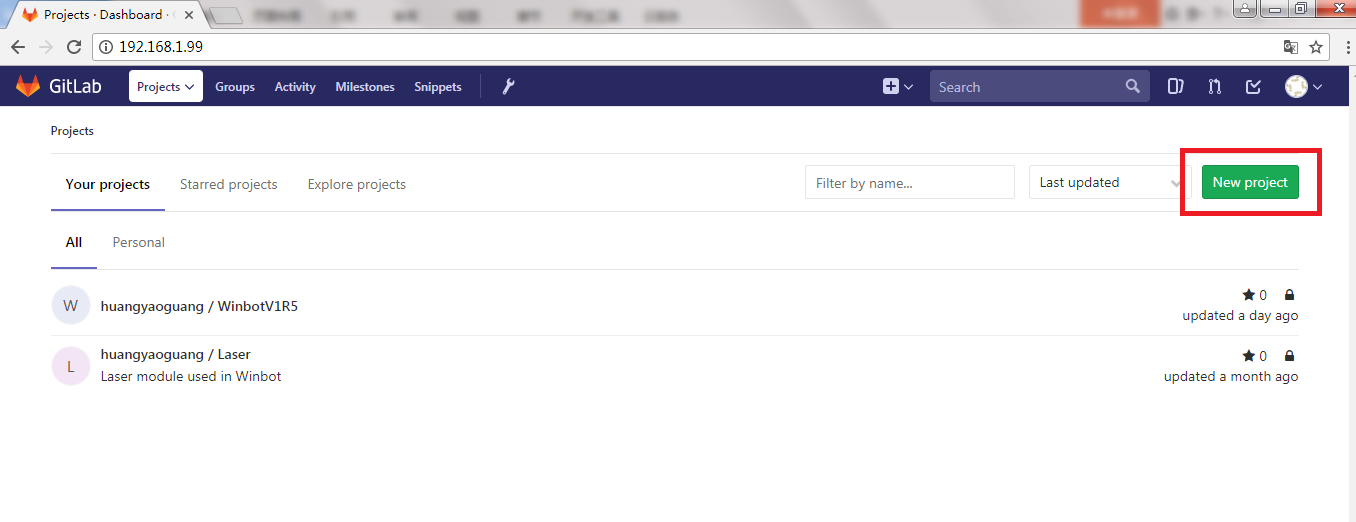


填写标题和说明，Assignee选择管理员，然后点击Submit Merge Request。

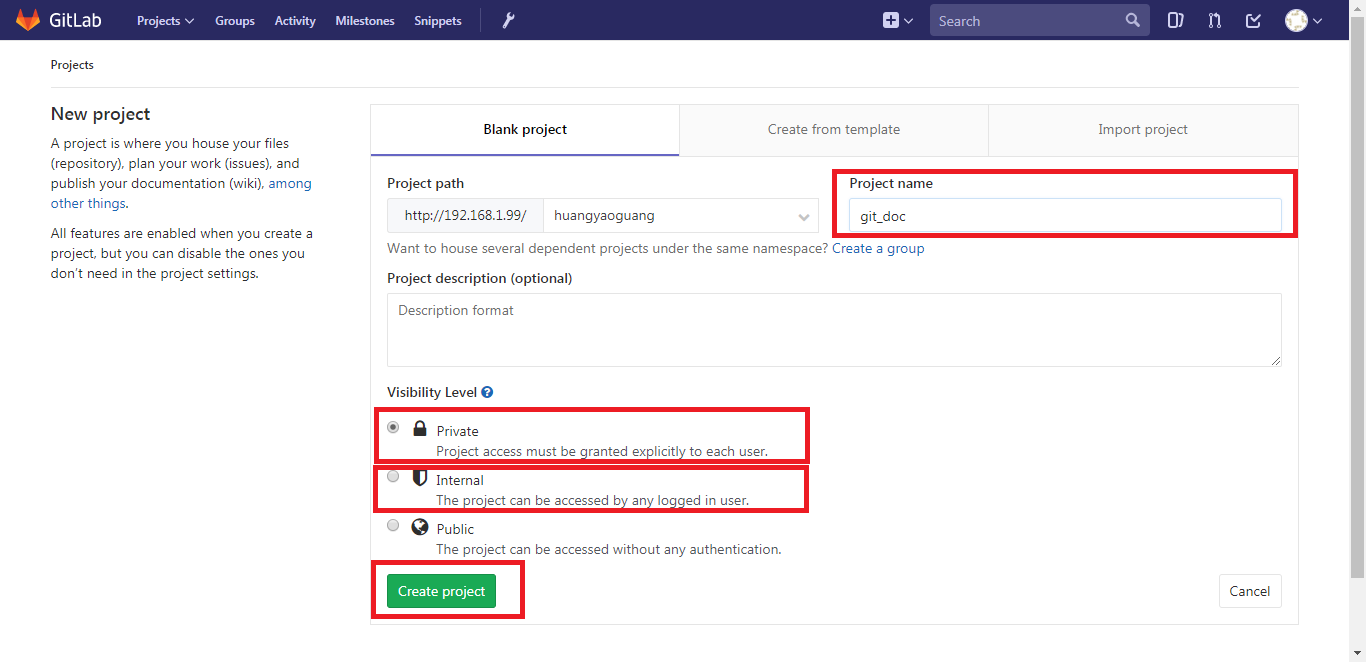


**在Gitlab上建立工程**

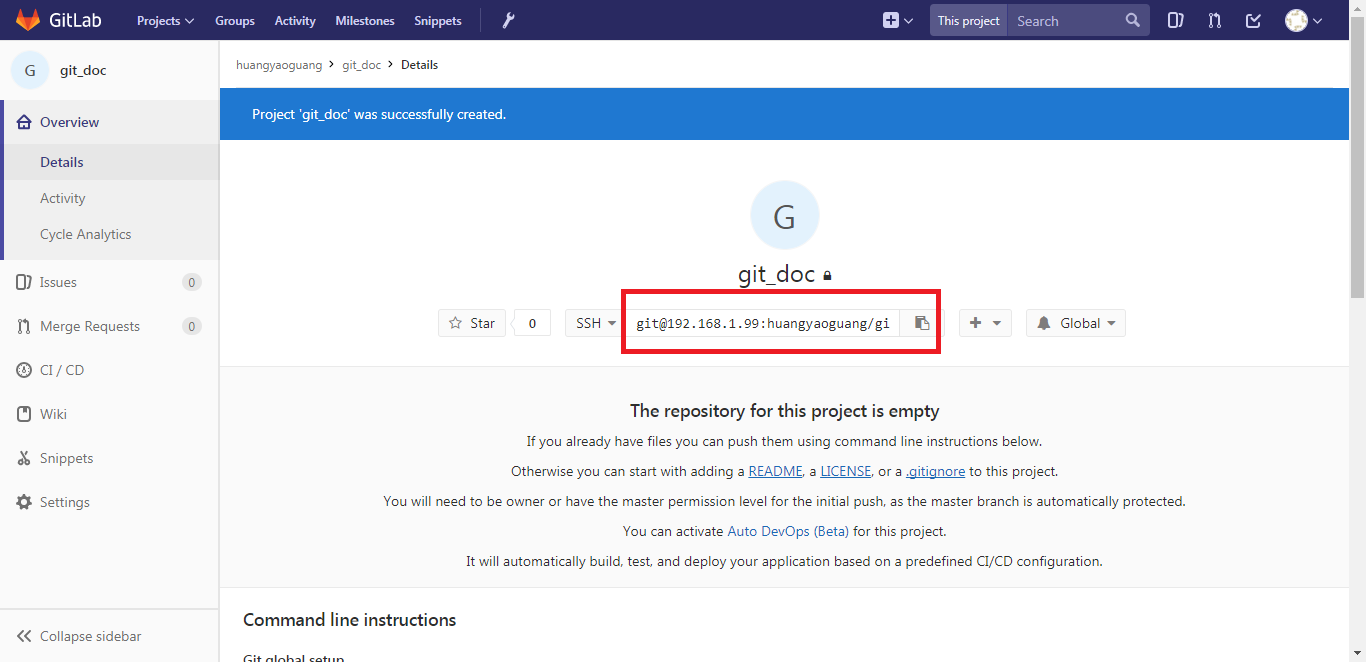
在Gitlab首页，点击New project



填写项目名字，根据项目性质选择可见等级，Internal表示该项目能够被登录Gitlab的所用用户看到，Private则只有相关的项目成员才能看到。

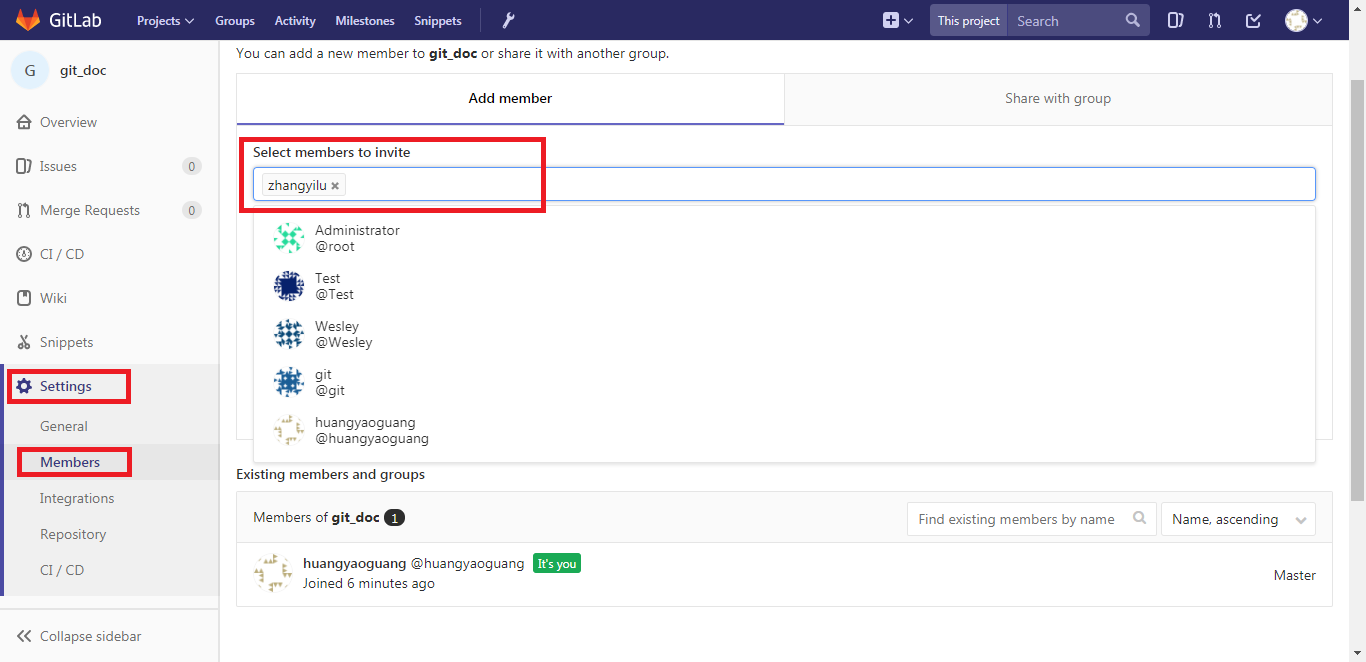


在下图红框位置可拷贝项目的链接地址，用来配置本地Git的远端服务器地址。注意：该链接需要修改才可使用。

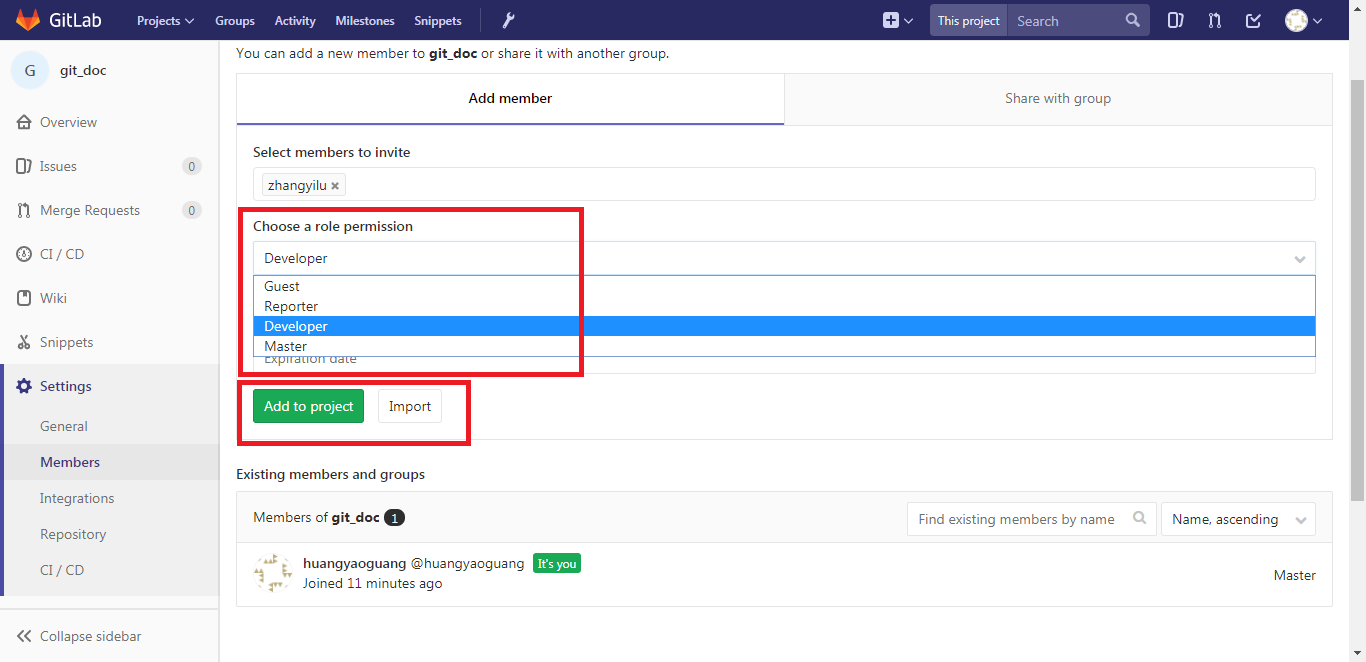


对于可见等级为Private的，根据实际需要添加项目成员。点击右侧Settings->Members

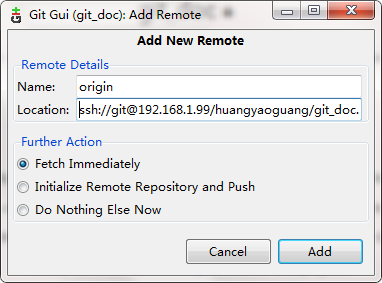
点击文本框Select members to invite，选择相应的成员即可。



选择相应成员的角色，如果成员只查看，不修改项目内容则选择Guest，研发人员选择Developer，管理员选择Master，根据人员不同角色先后将成员添加到项目。



若想将本地项目直接上传到服务器，在按照上文“本地使用”的步骤配置，然后配置远端服务器地址，过程与上文介绍一样，在Git GUI界面，点击Remote->add...，然后填写名字和链接地址，点击add。



回到主界面，点击Push，然后在弹出来的窗口再点击Push，即可将本地版本上传到服务器。

