Git使用教程

1. **gitlab安装和卸载**

**Ubuntu16.04.4**

**安装**

官方文档地址：

<https://about.gitlab.com/install/#stq=remove&stp=1>

sudo apt-get update

sudo apt-get install -y curl openssh-server ca-certificates

sudo apt-get install -y postfix

curl https://packages.gitlab.com/install/repositories/gitlab/gitlab-ce/script.deb.sh | sudo bash

sudo apt-get install gitlab-ce

或

sudo EXTERNAL\_URL="http://gitlab.example.com" apt-get install gitlab-ce

EXTERNAL\_URL：设置gitlab以后访问的地址



注：此处gitlab存在ce和ee版本，ce是社区版，ee是企业版，此处我们安装ce版本，ce版本比ee版本的协议宽容度更高，也更不容易出错。

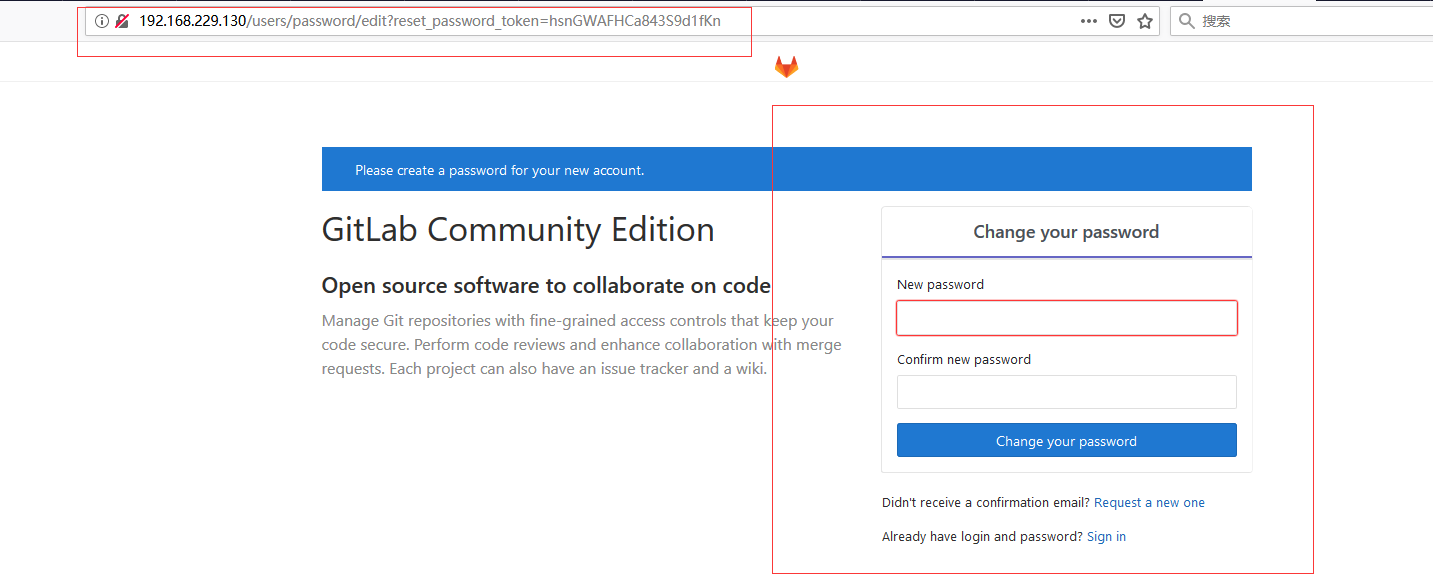
安装成功后，我们需要初始化配置

gitlab-ctl reconfigure

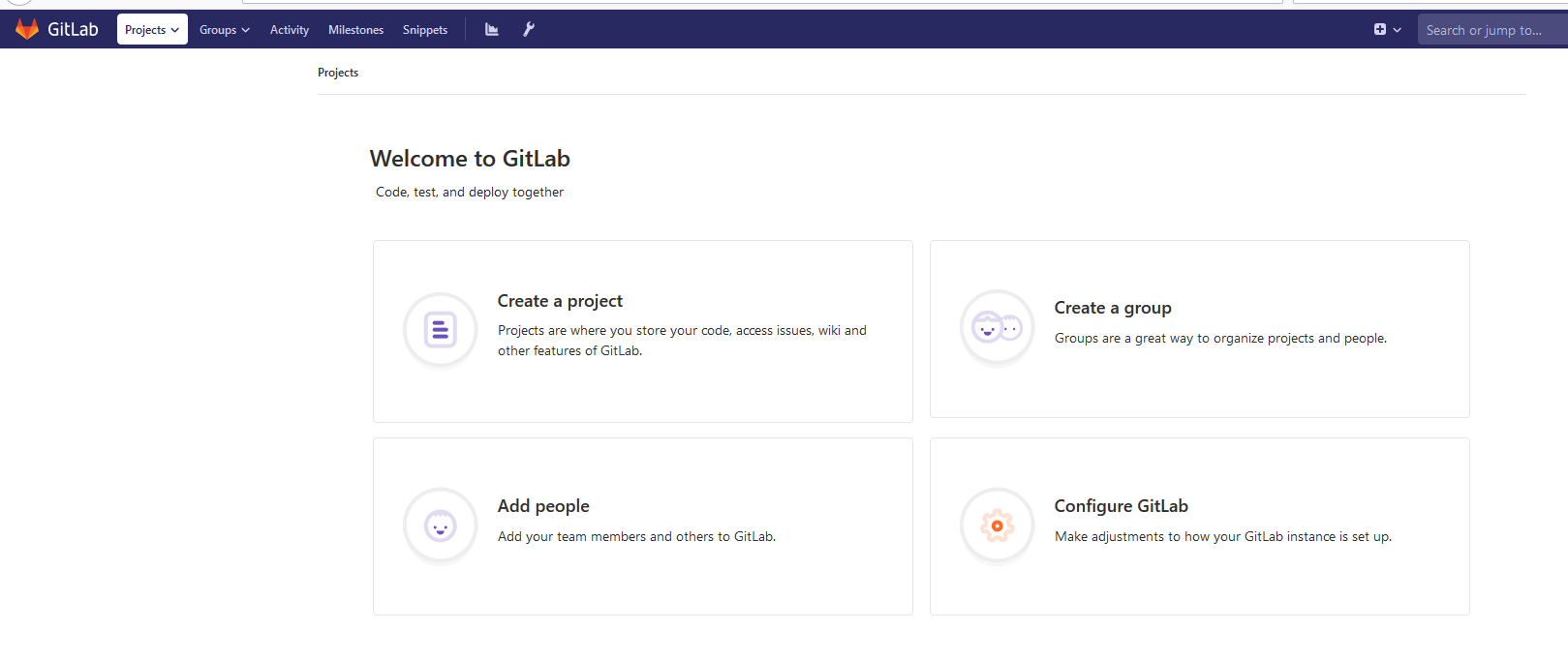
启动gitlab

gitlab-ctl start

gitlab默认占用的端口是80，因此根据ip地址就可以进行访问了，如果是跨机器访问则必须开放端口号，如下图说明gitlab安装成功。



上图是设置root账户的密码，我们设置的是gitlab66，设置完毕后，gitlab会转到登录页面，此时输入账户和密码就可以了



关闭gitlab

gitlab-ctl stop

**卸载**

前提：必须在Gitlab运行状态下才能卸载

Remove services

sudo gitlab-ctl uninstall

Clean any data generated by usage of the package

sudo gitlab-ctl cleanse

You may also want to remove any accounts you configured. To do so

sudo gitlab-ctl remove-accounts

Then remove the package using

sudo dpkg -P gitlab-ce

Furthermore, gitlab-ce uses these directories (as described here)

/opt/gitlab holds application code for GitLab and its dependencies.

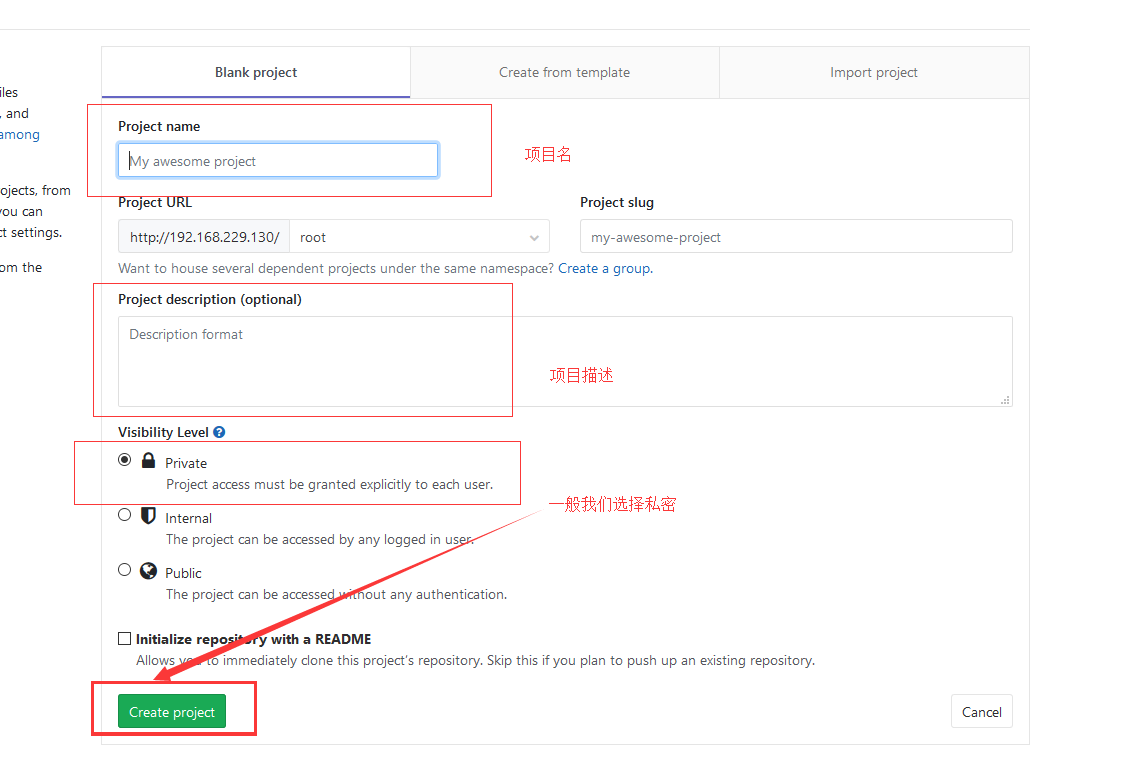
/var/opt/gitlab holds application data and configuration files that gitlab-ctl reconfigure writes to.

/etc/gitlab holds configuration files for omnibus-gitlab. These are the only files that you should ever have to edit manually.

/var/log/gitlab contains all log data generated by components of omnibus-gitlab.

1. **创建gitlab项目**

当我们安装好gitlab后，创建项目



我们在创建好项目后，此时还没有任何代码，我们需要在本地将代码上传到gitlab进行管理。

步骤1：在本地首先我们需要安装git，Ubuntu安装命令

apt-get install git

步骤2：对git进行基础配置

git config --global user.name 'root'

git config --global user.email '982452937@qq.com'

--global表示全局的，也就是对所有的项目有效

我们要和gitlab进行交互可以是http方式，这种方式每次都要输入密码，因此一般都会选择ssh方式，但是ssh方式一个机器只能配置一个账户

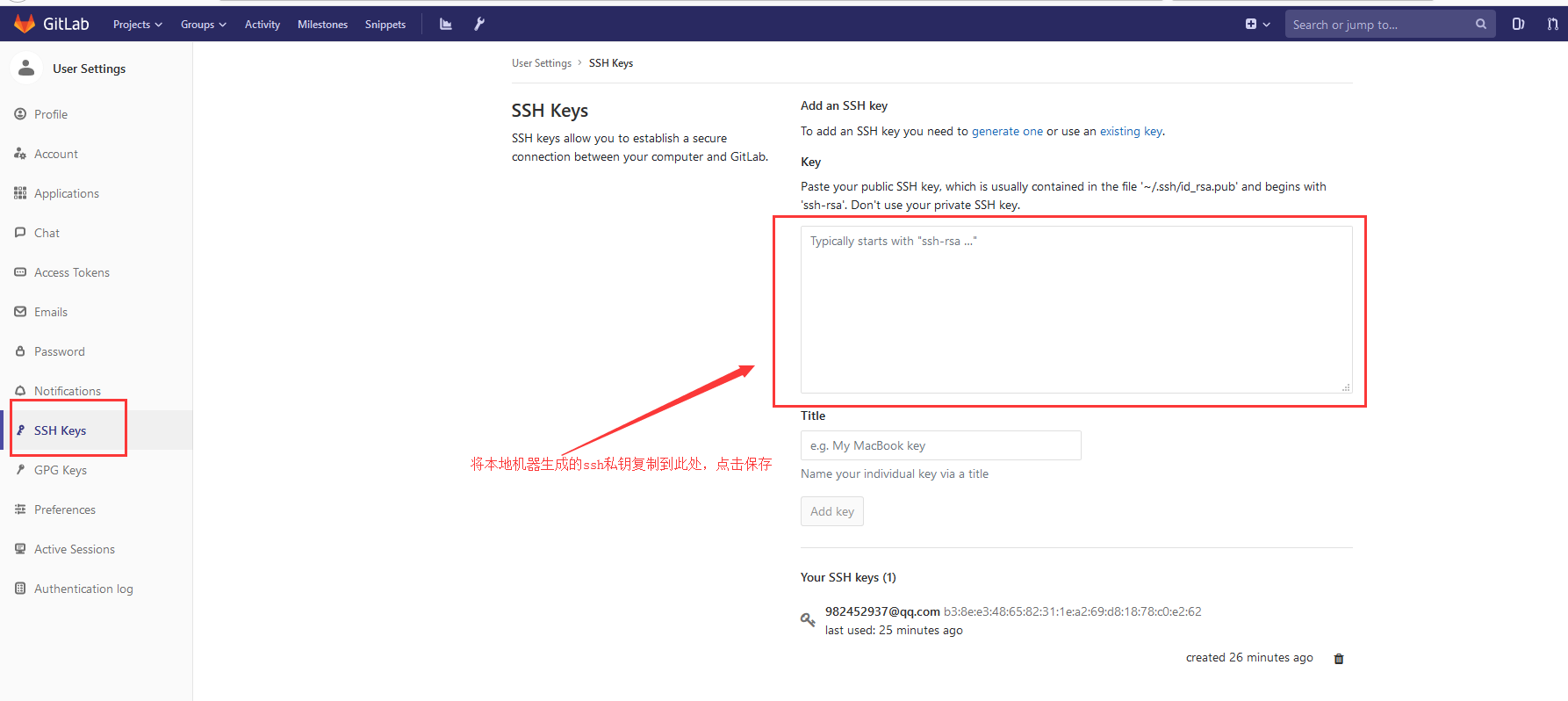
**配置ssh**

在本地机器执行命令

ssh-keygen -o -t rsa -C "982452937@qq.com" -b 4096

拷贝如下文件的内容到gitlab上

vim /root/.ssh/id\_rsa.pub



本地配置远程地址

查看远程地址：git remote –v

添加远程地址

git remote add origin [git@192.168.229.130:root/springmvc.git](mailto:git@192.168.229.130:root/springmvc.git)

origin是该远程地址的别名，此名可以随便取

注：此处地址千万不能加上机器的端口号，因为gitlab端口号默认是80，因此不能在这加上端口号，否则访问不了

步骤3：可以正常进行代码提交了

创建一个文件夹project

mkdir project

进入此文件夹：cd project

本地库初始化：git init

然后将需要提交的过程代码放置到project文件夹下。

git status

git add .

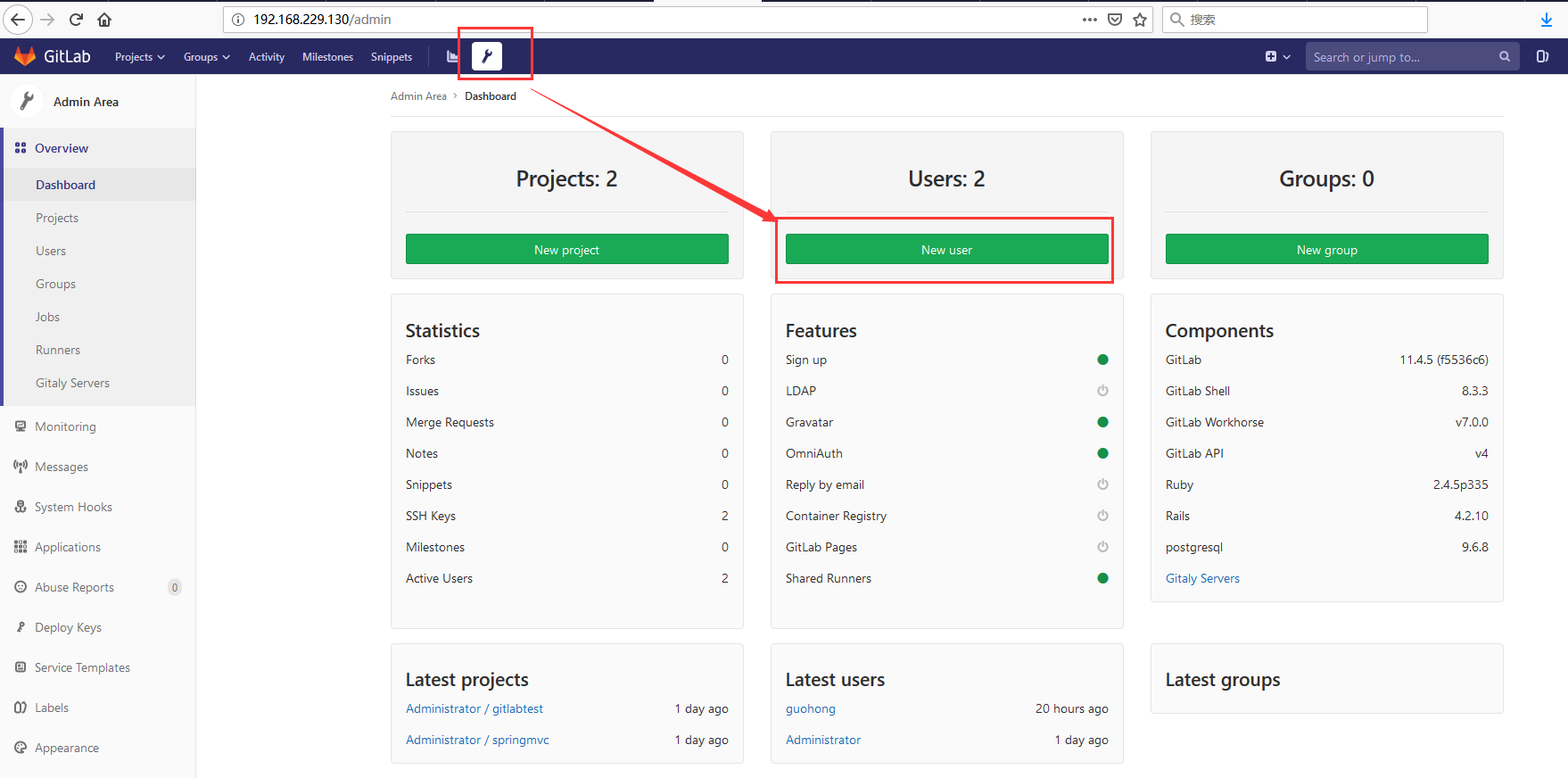
git commit –m ‘测试’

git push origin master

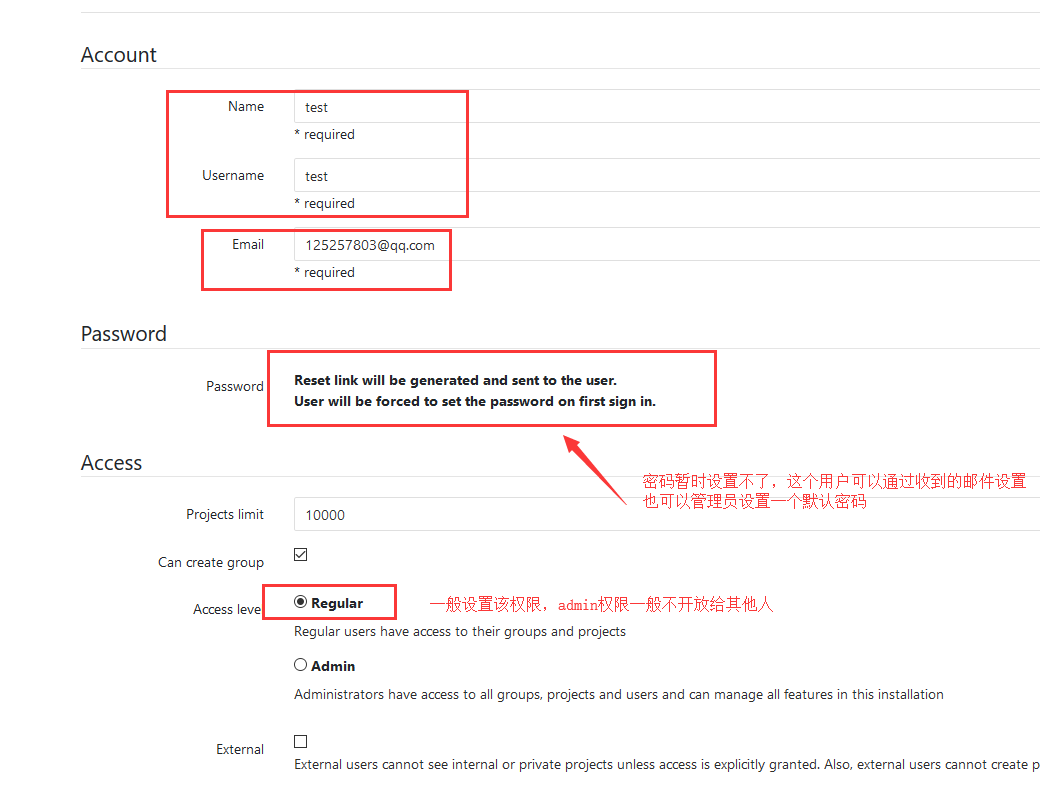
此处的master是远程分支名，origin就是远程地址的别名。

1. **gitlab添加账户**

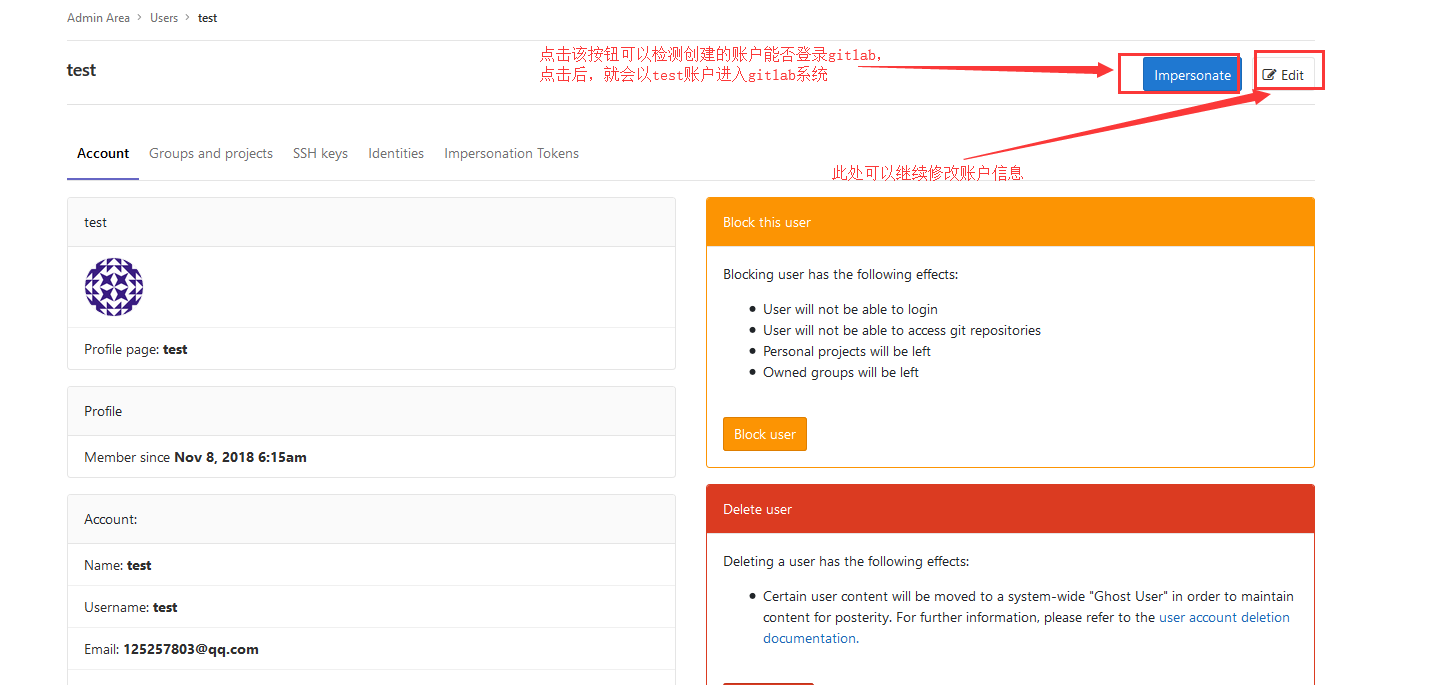
首先使用admin账户登录到gitlab系统



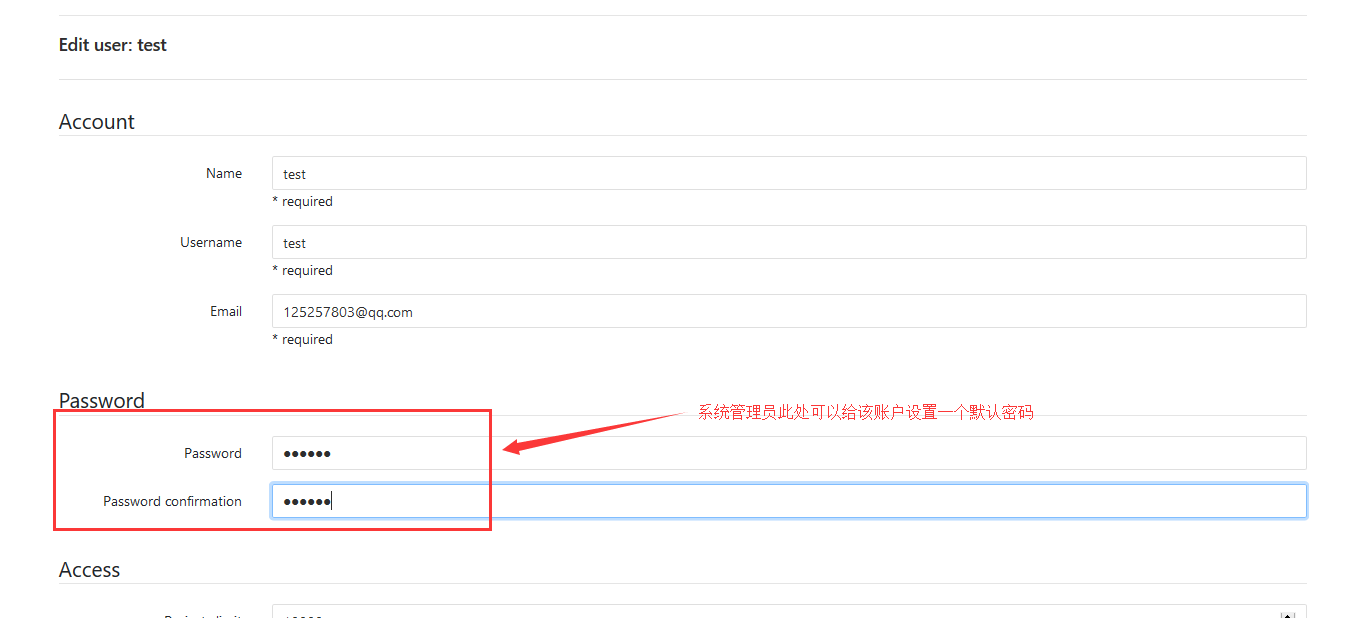
只需要填写如下信息，然后点击保存



创建成功后如下图所示：



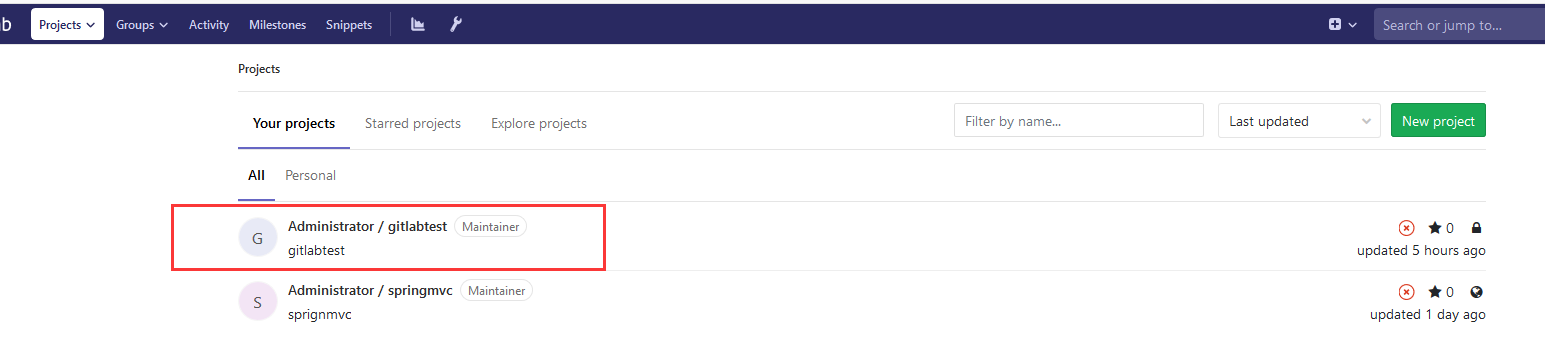
点击EDIT按钮，修改账户信息，此时我们就可以初始化该账户的密码了。该账户可以拿着该密码登录gitlab系统，然后系统会提示他进行修改密码。

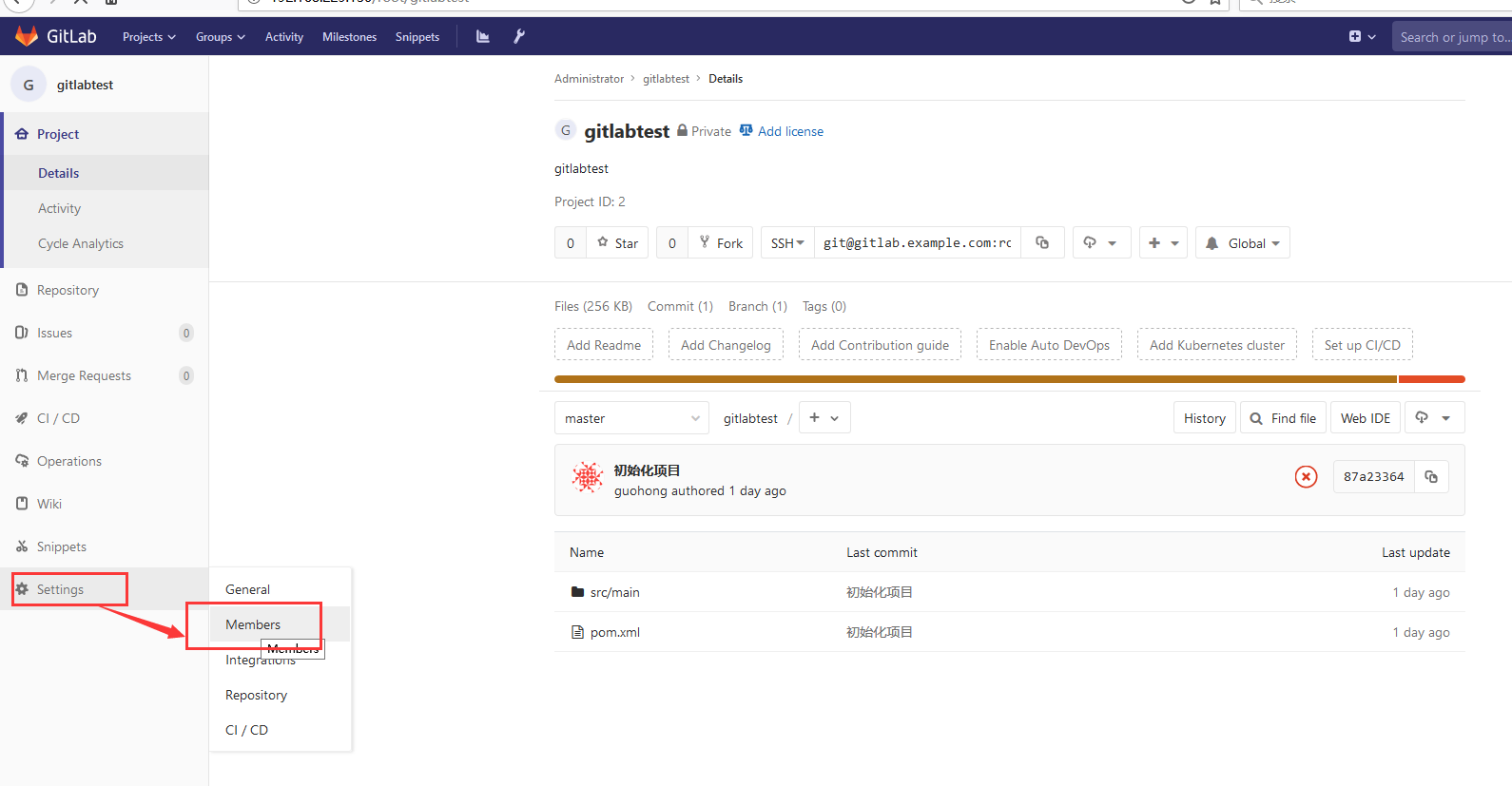


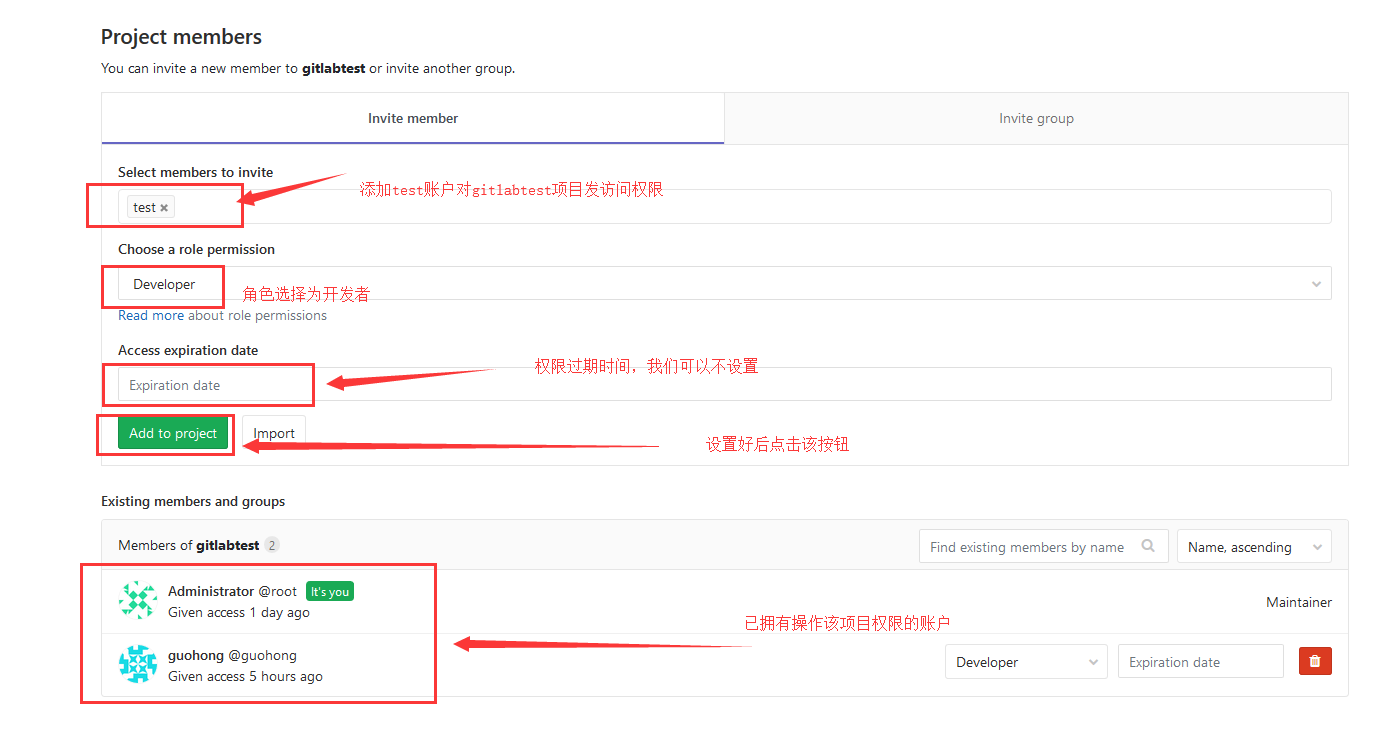
1. **gitlab开放某个账户的项目访问权限**

目标：为test账户配置访问操作gitlabtest项目的权限

选中项目，点进去







此处说下permission的控制问题：

Developer：只能clone，提交到master分支会报错

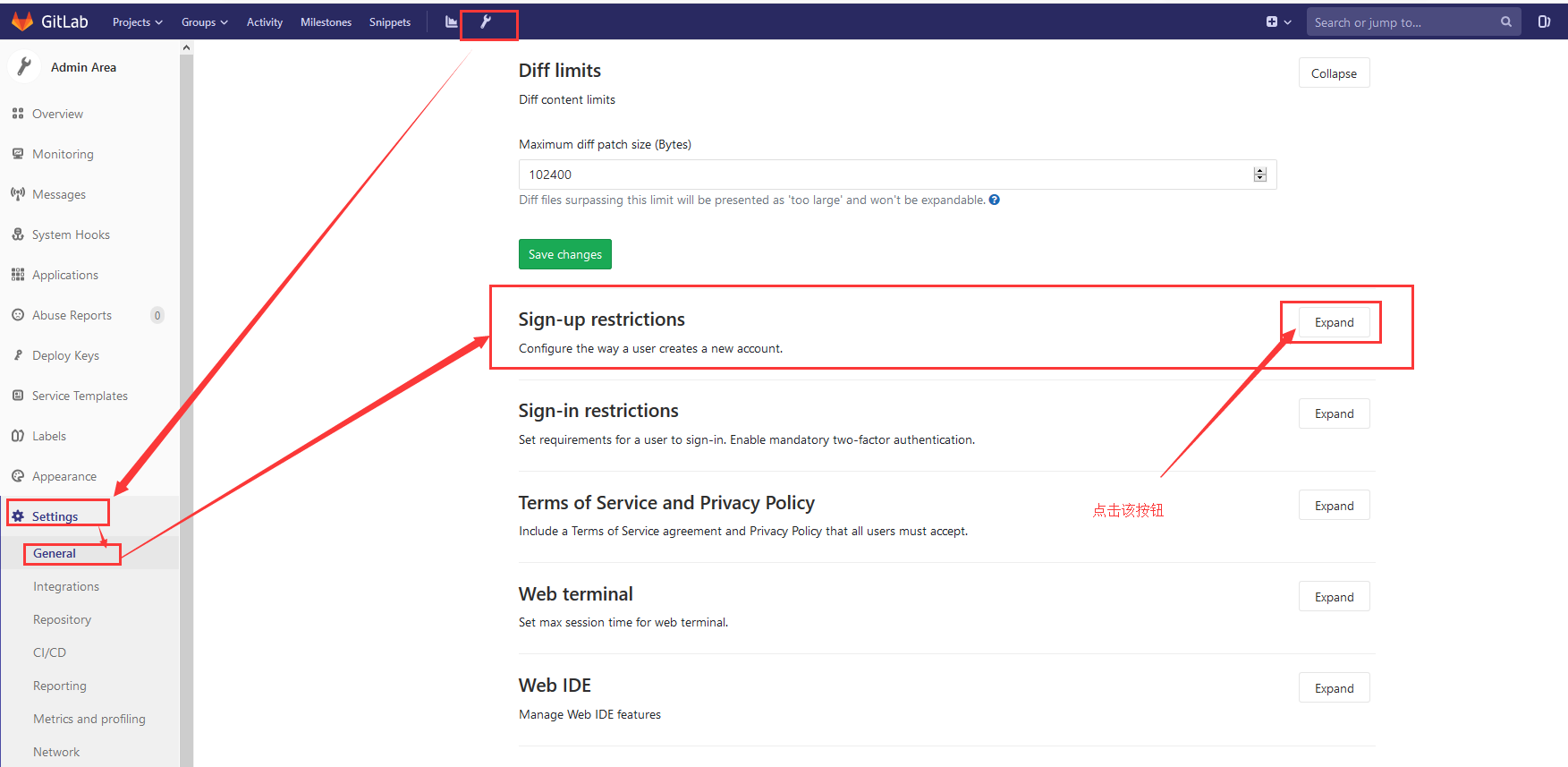
Maintainer：可以clone也可以push到master

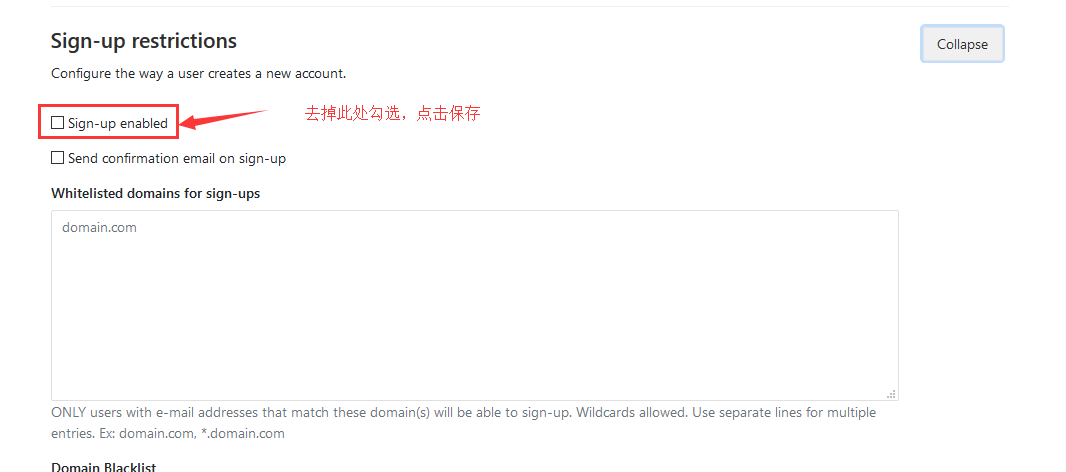
可以参考文档：https://www.cnblogs.com/minseo/p/9012195.html

完成配置后，test账户就可以对gitlabtest项目进行操作了

1. **gitlab关闭注册功能**

本来一开始是打算让大家自己按照我写好的格式规范注册GitLab账号，但是老是有人不遵守规范最后还得我来一个一个的提醒，大大的影响工作效率。因此我决定将GitLab的注册功能屏蔽掉，如果有新人进公司需要GitLab账号统一由我这边分配账户给他们。





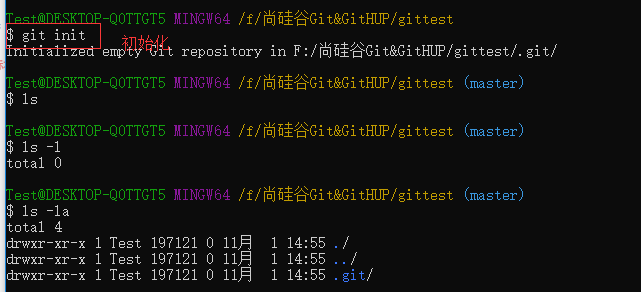
配置完成后，再请求gitlab登录页面的时候，就已经没有注册功能了

1. **git本地库初始化**

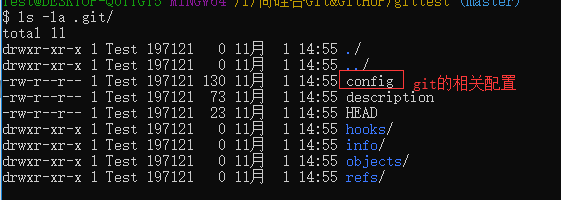
git安装完成后，本地初始化，比如我们想把某些文件给git管理，则需要打开git bash



本地库初始化：git init



执行完成后，在隐藏文件.git里面会发现生成了如下文件

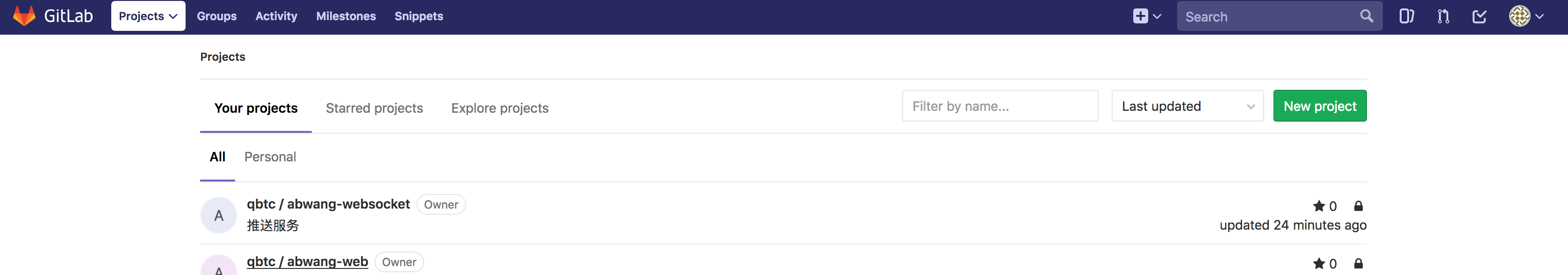


1. **拉代码**

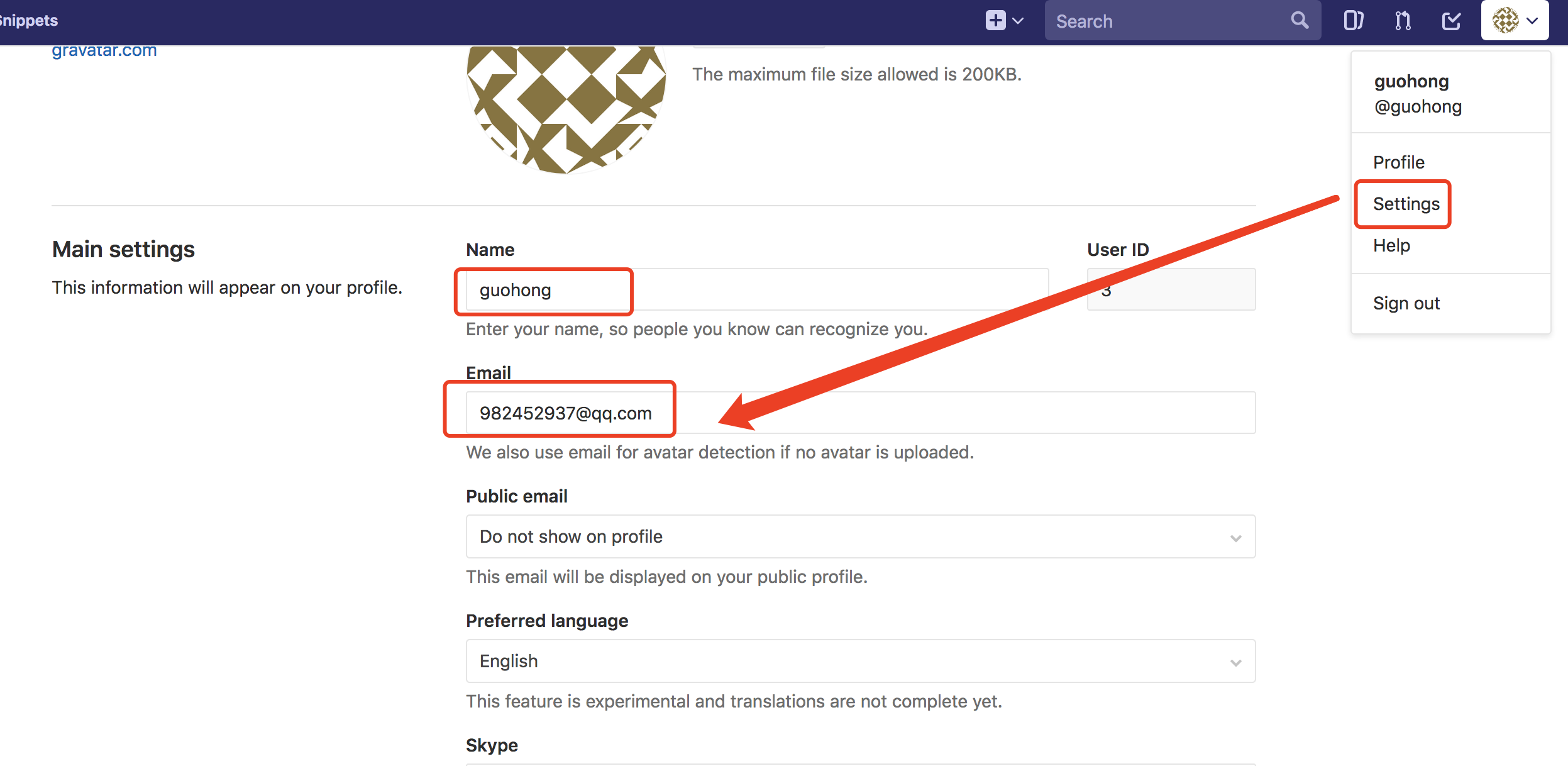
第一次拉代码时，需要进行key配置

操作系统mac

第一步，进入git网站注册账号。



第二步，生成key进行配置



打开终端，输入如下命令

git config --global user.name "guohong"

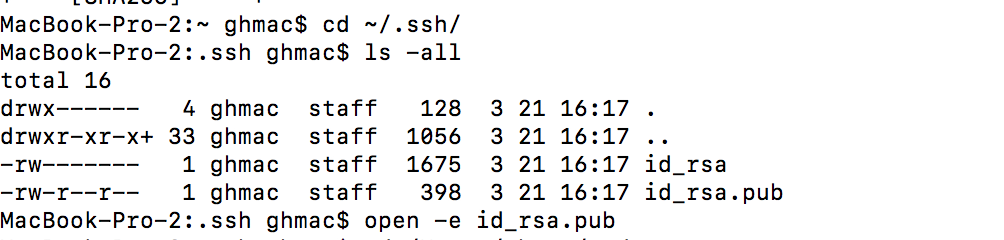
git config --global user.email [982452937@qq.com](mailto:982452937@qq.com)

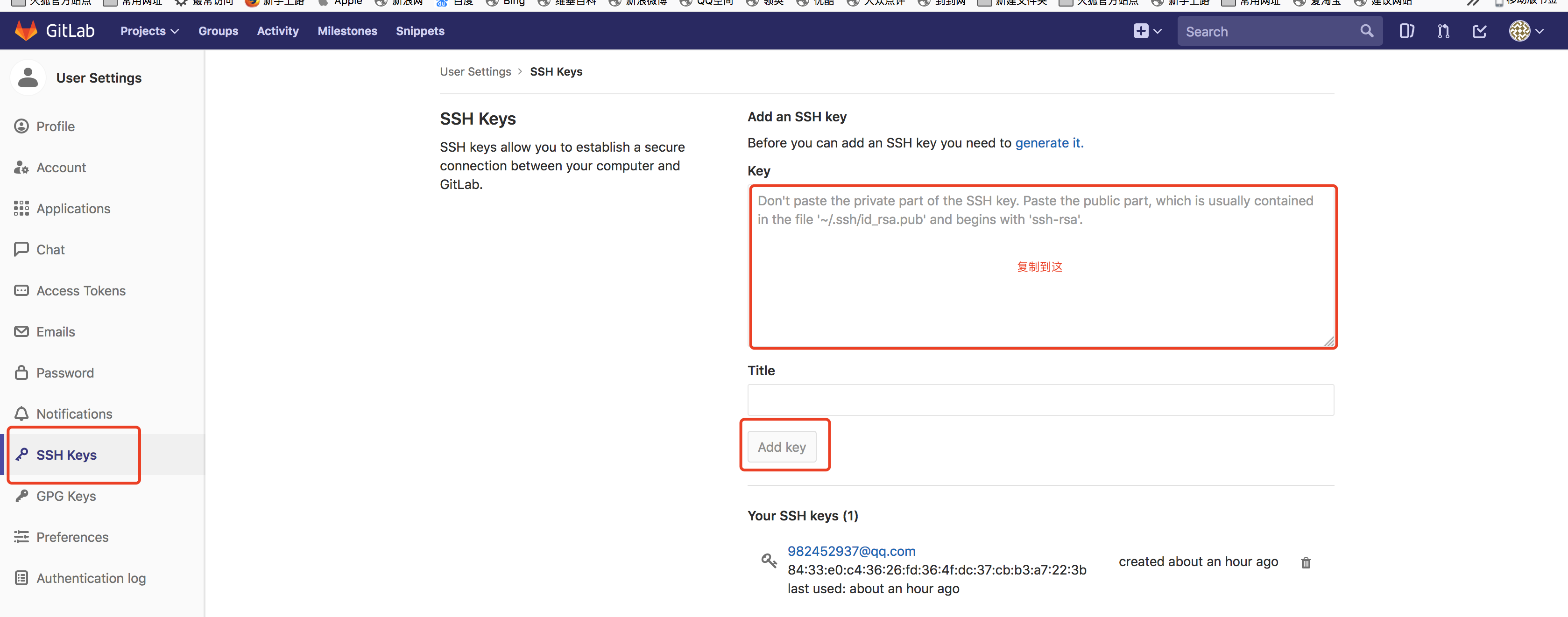
可以通过git config user.email命令查看设置的对不对

然后进行下面操作

ssh-keygen -t rsa -C [982452937@qq.com](mailto:982452937@qq.com)

此时会生成rsa ssh key,打开这个key文件，复制到git网站





如上操作后，就可以进行拉代码了。

1. **常用命令**

(开始前将repo脚本放到linux模拟器的bin目录，例如D:\cygwin64\bin)

然后找一个你喜欢的地方创建本地仓库

mkdir daianlawork（创建本地工作空间）

cd daianlawork/

ssh-keygen（生成秘钥和公钥）

vi ~/.ssh/config（打开设置远程仓库地址）

vi ~/.ssh/id\_rsa.pub（打开复制公钥到web）

chmod 644 ~/.ssh/config（授权）

rm .repo/ rf

git config --global user.email 'chenxiang@daianla.cn'（设置邮箱）

git config --global user.name 'chenxiang'（设置用户名）

repo init -u gerritserver:manifests -m daianla.xml（初始化repo仓库）

repo sync（同步远程仓库）

repo start chenxiang --all (给所有工程创建chenxiang分支)

cd daianla-top/

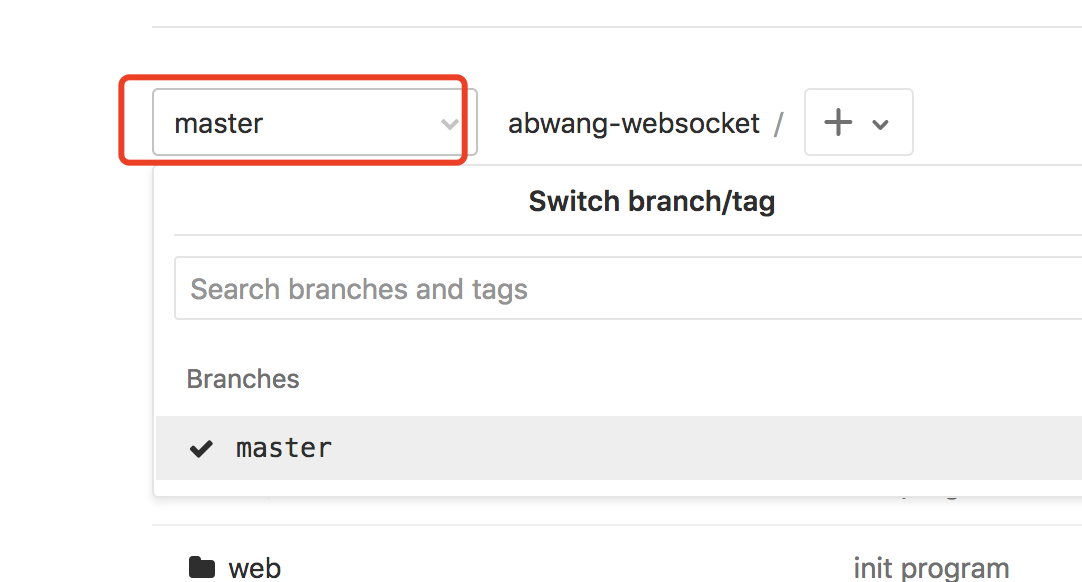
repo start chenxiang daianla-idcard(给指定工程创建分支)

git push origin HEAD:refs/for/master

refs/for/master是分支名

下图的分支名只有master

git push origin HEAD:master



--git 提交

git add --a(提交所有)

git add 文件名(提交单个文件)

git add xxx/\*(提交xx路径下的所有问题)

git commit -m 'ffff' 提交到本地库

git push origin HEAD:refs/for/master（将本地库的第一个提交提交到远程库）

将多次修改合并为一个提交

git add --a

git commit --amend

git push

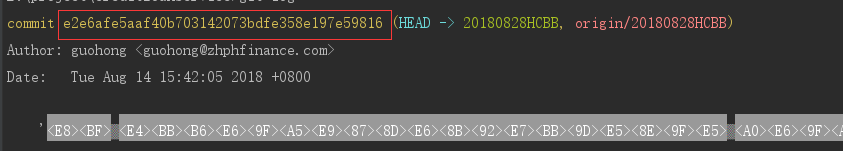
git pull origin 分支名

撤销add操作

git reset HEAD 文件名（如果不加文件名，则表示撤销本次add操作的所有文件）或者初次加入的时候可以使用git rm –-cache 文件名

撤销commit操作并保留修改

git reset –soft commitid号（可以通过git log查看这个号码）



注：其实就是--soft 、--mixed以及--hard是三个恢复等级。使用--soft就仅仅将头指针恢复，已经add的缓存以及工作空间的所有东西都不变。如果使用--mixed，就将头恢复掉，已经add的缓存也会丢失掉，工作空间的代码什么的是不变的。如果使用--hard，那么一切就全都恢复了，头变，aad的缓存消失，代码什么的也恢复到以前状态。

放弃本地修改，检出线上代码

git checkout 文件名

**解决git冲突**

方法1：

1.进入工程目录下 cd daianla-consumption-top/

2.git status命令查看git状态，会显示修改过的文件，如果有自己没有修改过的文件出现在目录，则将库上代码拉下来覆盖本地

3.git checkout consumption-app-web/src/main/java/cn/daianla/collect/controller/\*

方法2：

git pull 拉线上代码

git stash 备份本地修改代码

git pull 再次拉线上代码

git stash list 查看备份本地修改代码

git stash pop 从Git栈中读取最近一次保存的内容，恢复工作区的相关内容

git add --a

git commit -m '用户信息收集'

git push origin HEAD:refs/for/master

gitdir=$(git rev-parse --git-dir); scp -p -P 29418 guohong@192.168.6.202:hooks/commit-msg ${gitdir}/hooks/(第一次提交)

git commit --amend

git reset a2996daafd361a342ef35c8064f86eba23b90e09 恢复到修改前的版本位置 ，比如我一个提交落后版本，提交不了，可以恢复到以前版本，再pull，然后再依次提交

git教程：http://rogerdudler.github.io/git-guide/index.zh.html

1.查看 git状态：git status

2.查看git配置文件：git config -l

3.删除用户名：git config --global --unset-all user.name

4.添加用户名：git config --global --add user.name hong.guo

5.查看命令详情：git config

6.提交代码: git push

7.查看远程仓库地址：git remote -v

1. **分布式和集中式**

这里有几个概念：本地、服务器、中央服务器（远程服务器）。每一次commit是提交到本本机的服务器，这个不需要联网，正所谓的版本管理，就是要方便我们知道每一个版本，比如回到之前的某个版本（这是其一），而且回退到某个之前的版本，也是从本机的服务器拿的数据，这些都不需要联网。而 SVN 的每一次 commit 都需要联网，这就需要网络的等待。 Git只有在Push、pull 的时候需要联网，而我们平时更多的操作应是commit。再有就是，断网的情况下，SVN也能工作，但是由于没有版本控制的记录，当多人修改后就比较难以快速的合并，但是Git都在本地保存了版本记录，所以大家合并起来就方便得多了。