GitLab配置使用简介

（版本：1.0.0.0）

中国金融认证中心

2019年7月9日

版权声明：本文档的版权属于中国金融认证中心，任何人或组织未经许可，

不得擅自修改、拷贝或以其它方式使用本文档中的内容

文档修订记录

本文档会随时保持更新，请与中国金融认证中心索要最新版本

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **版本** | **内容** | **日期** | **编写** | **审核** |
| 1.0.0.0 | 创建 | 2019/7/9 | 杨晓宇 |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

注：对该文件内容增加、删除或修改须填写此修订记录，详细记载变更信息，以保证其可追溯性。

目录

[1 引言 1](#_Toc69803703)

[2 安装配置 1](#_Toc69803704)

[2.1 安装 1](#_Toc69803705)

[2.1.1 创建repo文件 1](#_Toc69803706)

[2.1.2 安装依赖 1](#_Toc69803707)

[2.1.3 安装gitlab-ce 2](#_Toc69803708)

[2.2 Docker安装gitlab 2](#_Toc69803709)

[2.2.1 配置代理 3](#_Toc69803710)

[2.3 配置 4](#_Toc69803711)

[2.3.1 URL配置 4](#_Toc69803712)

[2.3.2 邮件配置 4](#_Toc69803713)

[2.3.3 LDAP配置 5](#_Toc69803714)

[2.3.4 HTTPS配置 6](#_Toc69803715)

[2.4 启动 6](#_Toc69803716)

[3 使用 7](#_Toc69803717)

[3.1 用户配置 7](#_Toc69803718)

[3.1.1 登录 7](#_Toc69803719)

[3.1.2 变更邮箱 7](#_Toc69803720)

[3.1.2.1 变更公共和Commit邮箱 9](#_Toc69803721)

[3.1.2.2 变更通知邮箱 10](#_Toc69803722)

[3.1.3 添加SSH公钥 10](#_Toc69803723)

[3.1.4 用户提权 11](#_Toc69803724)

[3.2 创建分组 11](#_Toc69803725)

[3.2.1 创建分组 11](#_Toc69803726)

[3.2.2 添加组成员 12](#_Toc69803727)

[3.2.3 切换组所有者权限 13](#_Toc69803728)

[3.3 工程配置 14](#_Toc69803729)

[3.3.1 添加工程 14](#_Toc69803730)

[3.3.2 添加工程成员 17](#_Toc69803731)

[4 命令行工具 18](#_Toc69803732)

[5 工程示例 19](#_Toc69803733)

[6 GitLab runner 22](#_Toc69803734)

[7 Git功能拾遗 29](#_Toc69803735)

[7.1 创建远程分支 29](#_Toc69803736)

[7.2 跟踪远程分支 30](#_Toc69803737)

[7.3 删除远程分支 31](#_Toc69803738)

[7.4 删除本地分支 31](#_Toc69803739)

[7.5 切换URL 32](#_Toc69803740)

[7.6 子模块（submodule） 33](#_Toc69803741)

[7.7 下载指定分支代码 33](#_Toc69803742)

[7.8 比较不同分支某个文件差异 33](#_Toc69803743)

[7.9 使用指定分支模块指定文件覆盖当前分支同名文件 33](#_Toc69803744)

[7.10 撤销本地commit 34](#_Toc69803745)

[7.11 撤销已暂存未提交的文件 34](#_Toc69803746)

[7.12 删除untracked文件 34](#_Toc69803747)

[7.13 配置比较工具 35](#_Toc69803748)

[7.14 git stash 35](#_Toc69803749)

# 引言

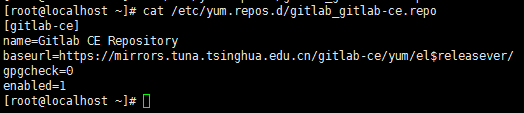
Gitlab 是一个提供代码托管、提交审核和问题跟踪的代码管理平台。基于Git实现的在线代码仓库托管软件，可以用gitlab搭建一个类似于Github一样的系统，一般用于在企业、学校等内部网络搭建git私服。

# 安装配置

## 安装

### 创建repo文件

新建/etc/yum.repos.d/gitlab\_gitlab-ce.repo文件，内容为：



[gitlab-ce]

name=Gitlab CE Repository

baseurl=https://mirrors.tuna.tsinghua.edu.cn/gitlab-ce/yum/el$releasever/

gpgcheck=0

enabled=1

### 安装依赖

sudo yum install -y curl policycoreutils-python openssh-server cronie openldap-clients

sudo systemctl enable sshd

sudo systemctl start sshd

sudo firewall-cmd --permanent --add-service=http

sudo systemctl reload firewalld

sudo yum install postfix

sudo systemctl enable postfix

sudo systemctl start postfix

### 安装gitlab-ce

sudo yum install gitlab-ce -y

## Docker安装gitlab

<https://docs.gitlab.com/omnibus/docker/>

可以使用Docker安装gitlab，在支持docker的服务器上执行如下命令：

docker run --detach \

--hostname gitlabdev \

--publish 443:443 --publish 80:80 --publish 10022:22 \

--name gitlab \

--restart always \

--volume /srv/gitlab/config:/etc/gitlab \

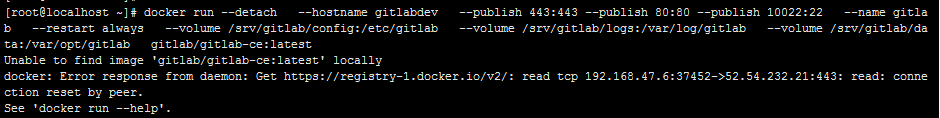
--volume /srv/gitlab/logs:/var/log/gitlab \

--volume /srv/gitlab/data:/var/opt/gitlab \

gitlab/gitlab-ce:latest

如果有如下错误，则需要配置代理。

docker: Error response from daemon: Get https://registry-1.docker.io/v2/: read tcp 192.168.47.6:49272->34.195.246.183:443: read: connection reset by peer.



### 配置代理

<https://github.com/moby/moby/issues/32270>

1. 为docker服务创建 systemd drop-in 目录

 mkdir -p /etc/systemd/system/docker.service.d

1. 编辑/etc/systemd/system/docker.service.d/http-proxy.conf文件，添加如下内容：

Environment="HTTP\_PROXY=http://192.168.113.7:1080"

对于HTTPS的代理

编辑/etc/systemd/system/docker.service.d/https-proxy.conf文件，添加如下内容：

Environment="HTTPS\_PROXY=https://192.168.113.7:1080"

1. 执行sudo systemctl daemon-reload
2. 重启docker， sudo systemctl restart docker
3. 验证配置是否加载：systemctl show --property=Environment docker

## 配置

配置文件路径/etc/gitlab/gitlab.rb：

### URL配置

查找external\_url，配置url为本地ip。

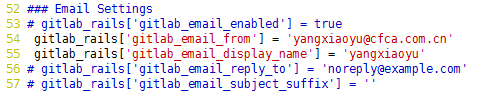


### 邮件配置

配置通知邮服务通知邮件地址：

gitlab\_rails['gitlab\_email\_from'] = 'yangxiaoyu@cfca.com.cn'

gitlab\_rails['gitlab\_email\_display\_name'] = 'yangxiaoyu'



配置邮件服务器信息：

gitlab\_rails['smtp\_enable'] = true

gitlab\_rails['smtp\_address'] = "mail.cfca.com.cn"

gitlab\_rails['smtp\_port'] = 465

gitlab\_rails['smtp\_user\_name'] = "email@cfca.com.cn"

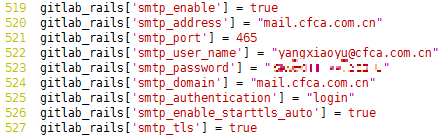
gitlab\_rails['smtp\_password'] = "password"

gitlab\_rails['smtp\_domain'] = "mail.cfca.com.cn"

gitlab\_rails['smtp\_authentication'] = "login"

gitlab\_rails['smtp\_enable\_starttls\_auto'] = true

gitlab\_rails['smtp\_tls'] = true



上述的gitlab\_email\_from和smtp\_user\_name应保持一致，正式系统中需要申请专用邮箱。

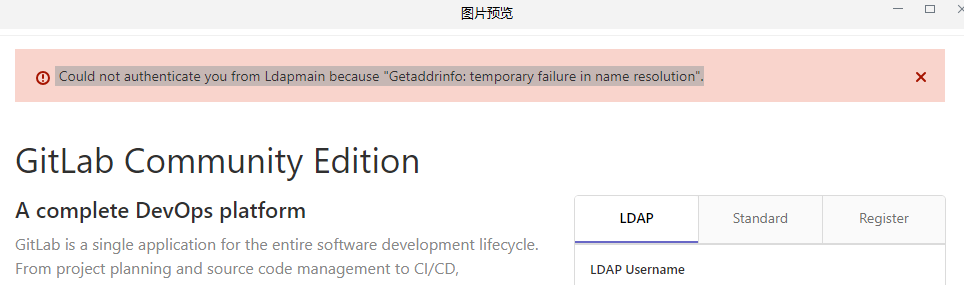
如果邮件不能正常发出可能是smtp\_address无法解析,可以通过配置/etc/hosts文件解决,添加一行: 192.168.254.103 mail.cfca.com.cn

### LDAP配置



LDAP测试命令行：

ldapsearch -D "CN=gitlab,OU=Configurator,DC=yz,DC=cfca,DC=com,DC=cn" -w gitlab\_1234 -p 389 -h 192.168.117.1 -b "OU=CSP,OU=dev,DC=yz,DC=cfca,DC=com,DC=cn" -s sub "CN=yangxiaoyu"



如果出现”Could not authenticate you from Ldapmain because “Getaddrinfo: temporary failure in name resolution”问题，很可能是因为网络配置问题，导致无法解析AD域服务器地址，可以重启docker服务或者解决地址解析问题。

### HTTPS配置

生成证书和私钥（略）

配置external\_url为<https://ip>

配置证书和私钥：

修改配置文件

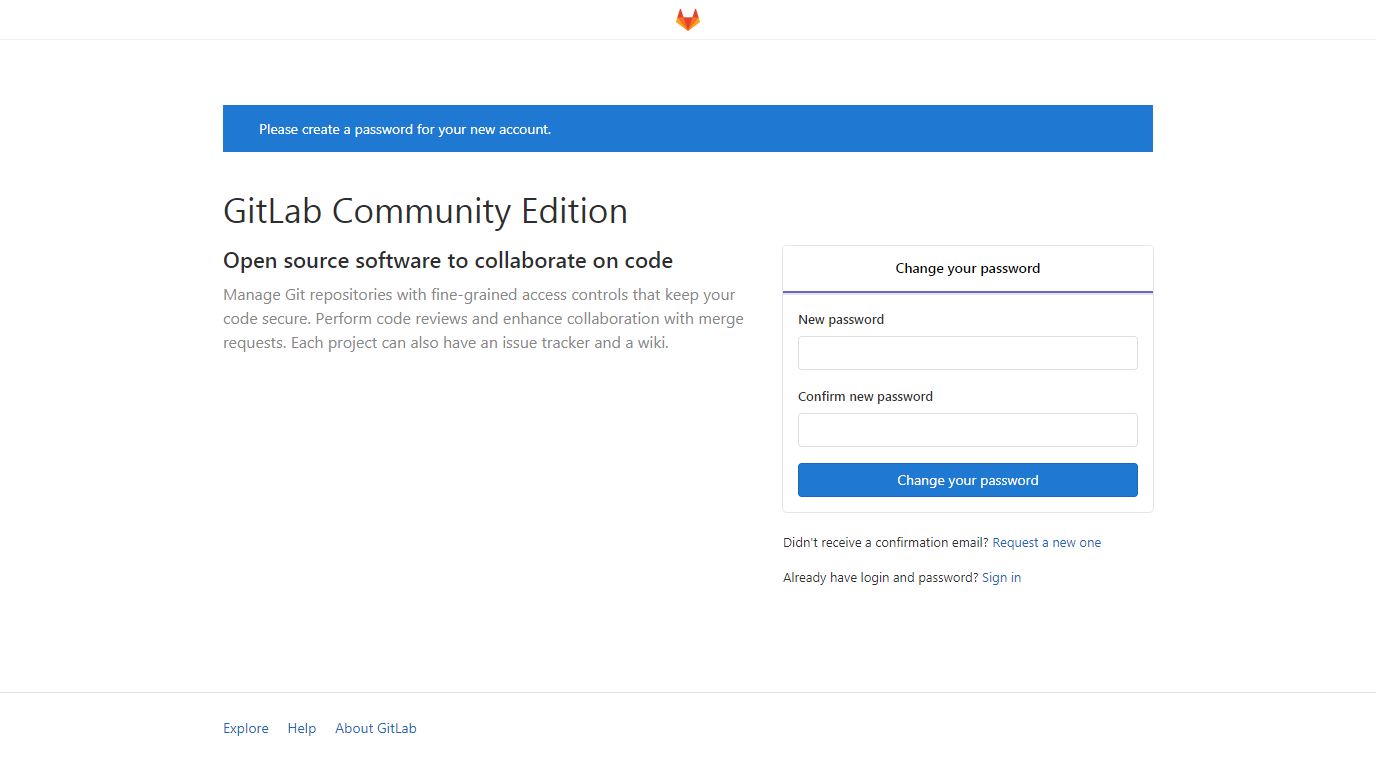


## 启动

gitlab-ctl reconfigure更具配置文件生成配置

gitlab-ctl start启动服务

登录<http://192.168.41.157>访问界面，第一次登录时要求输入密码。

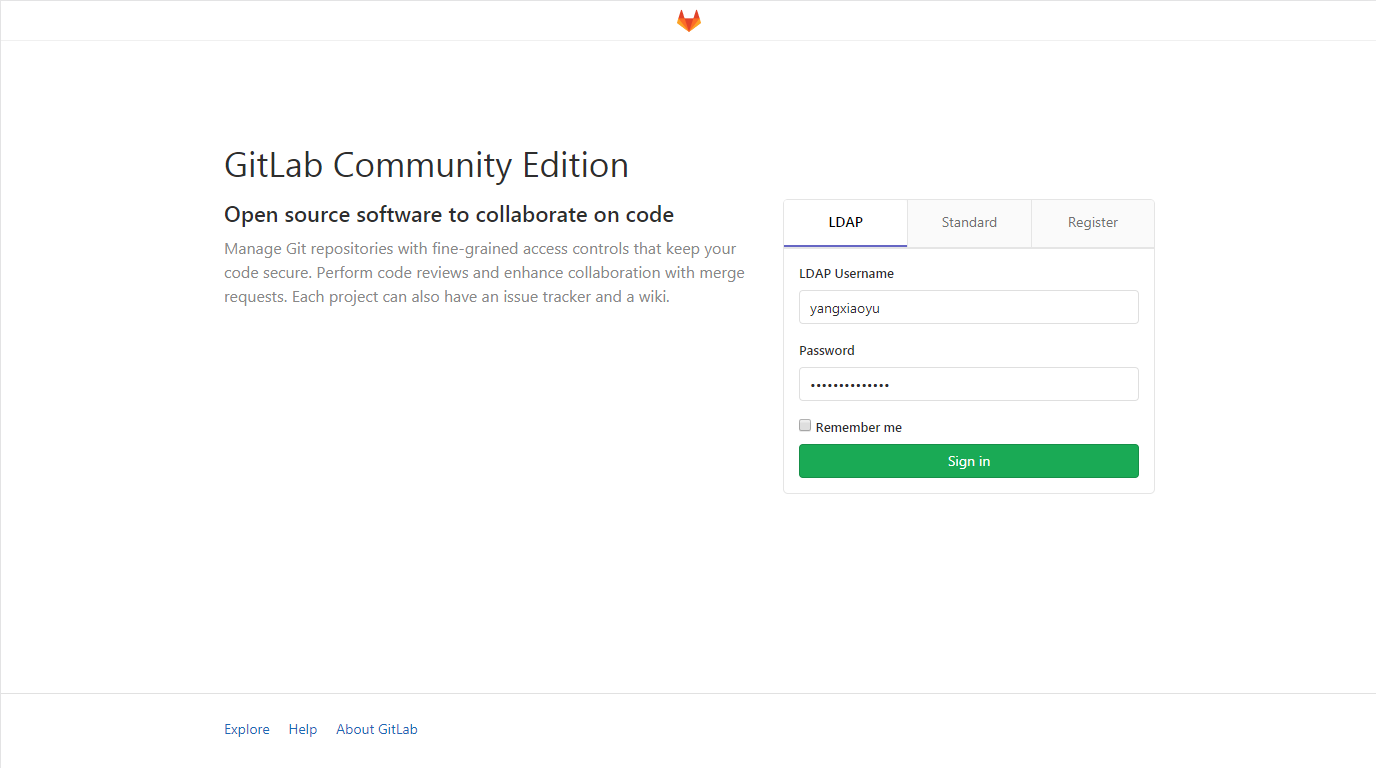


# 使用

## 用户配置

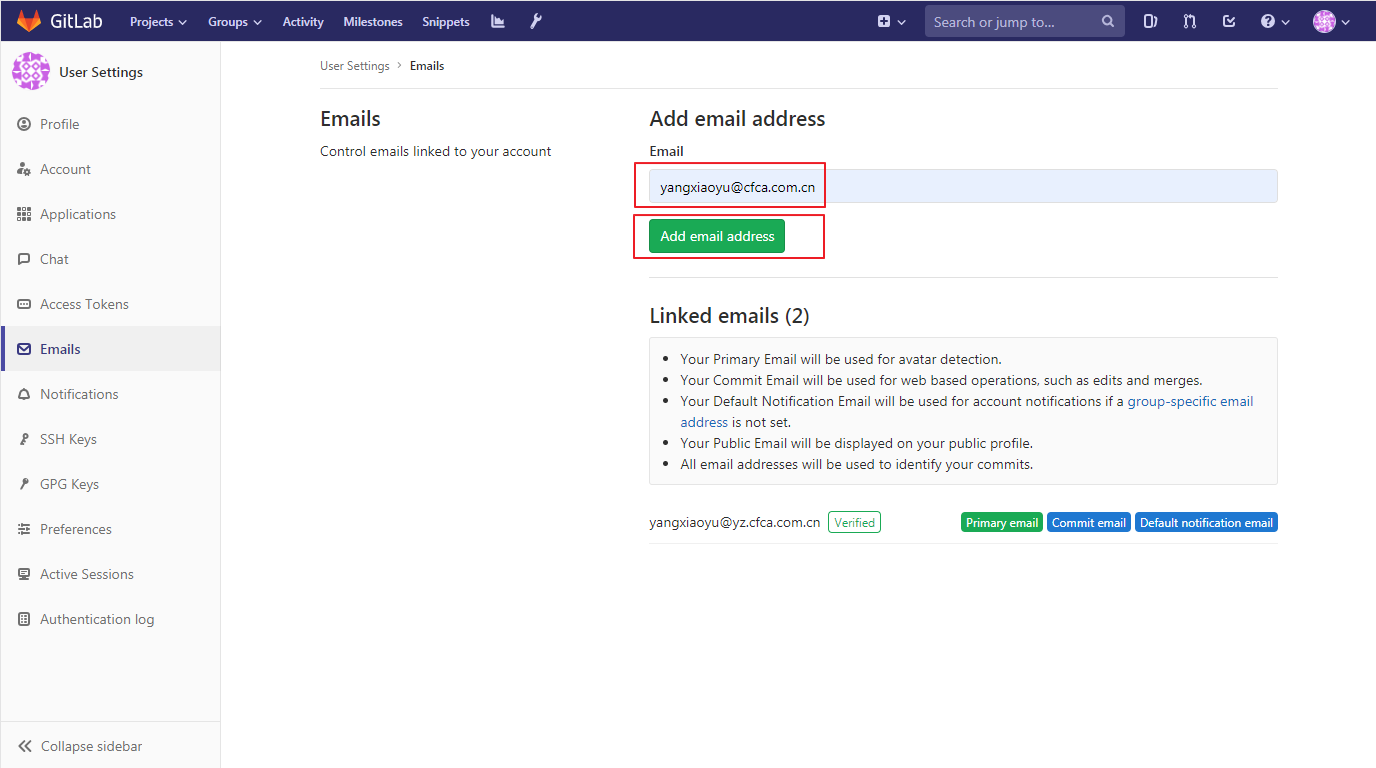
### 登录

普通用户可以使用yz域账号密码登录，如下所示：

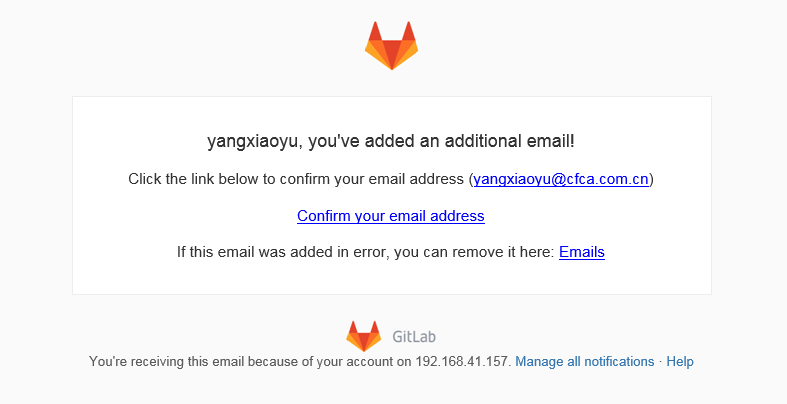


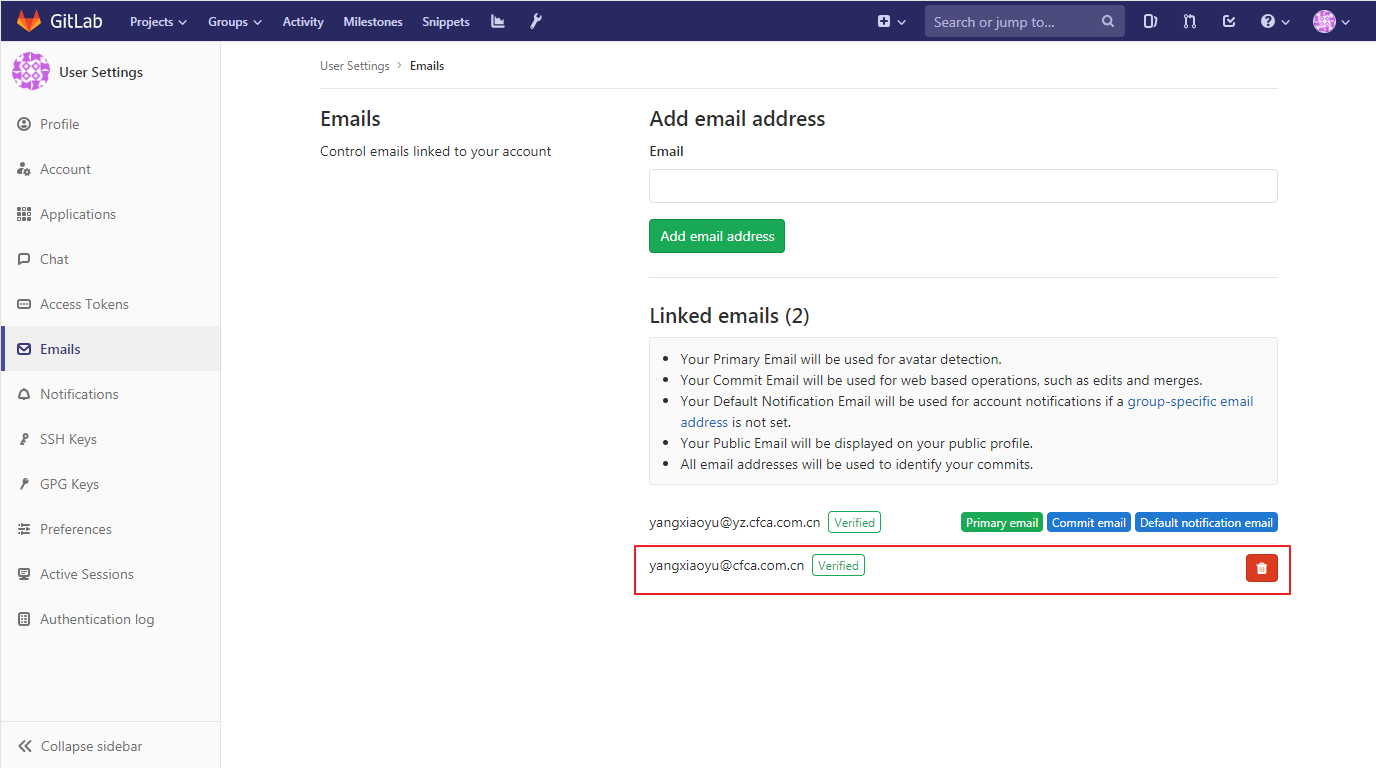
### 变更邮箱

由于添加了yz域认证，GitLab默认添加了yz域的邮箱（即email@yz.cfca.com.cn）。需要变更该邮箱为cfca.com.cn的邮箱。

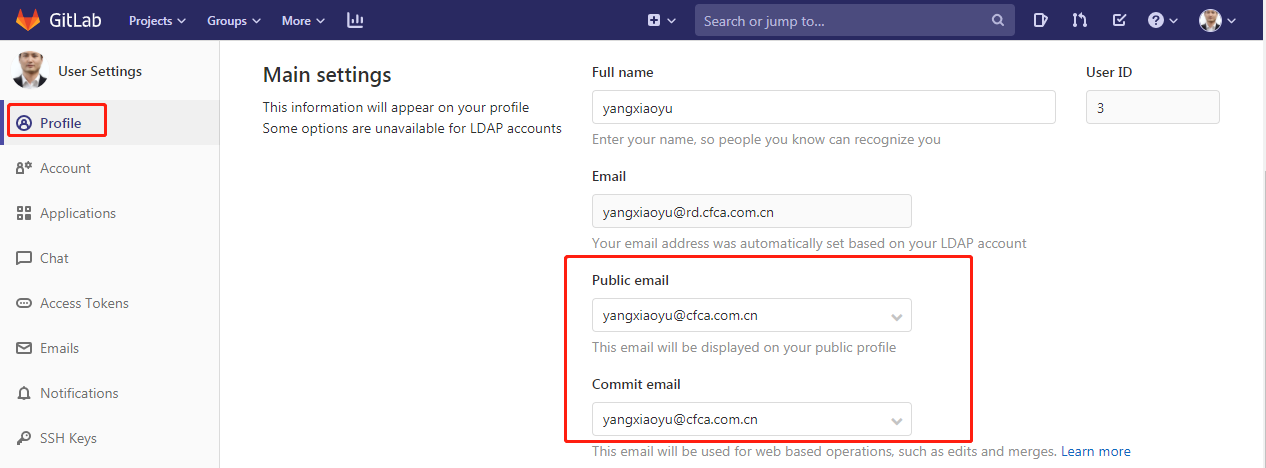


登录邮箱点击确认链接

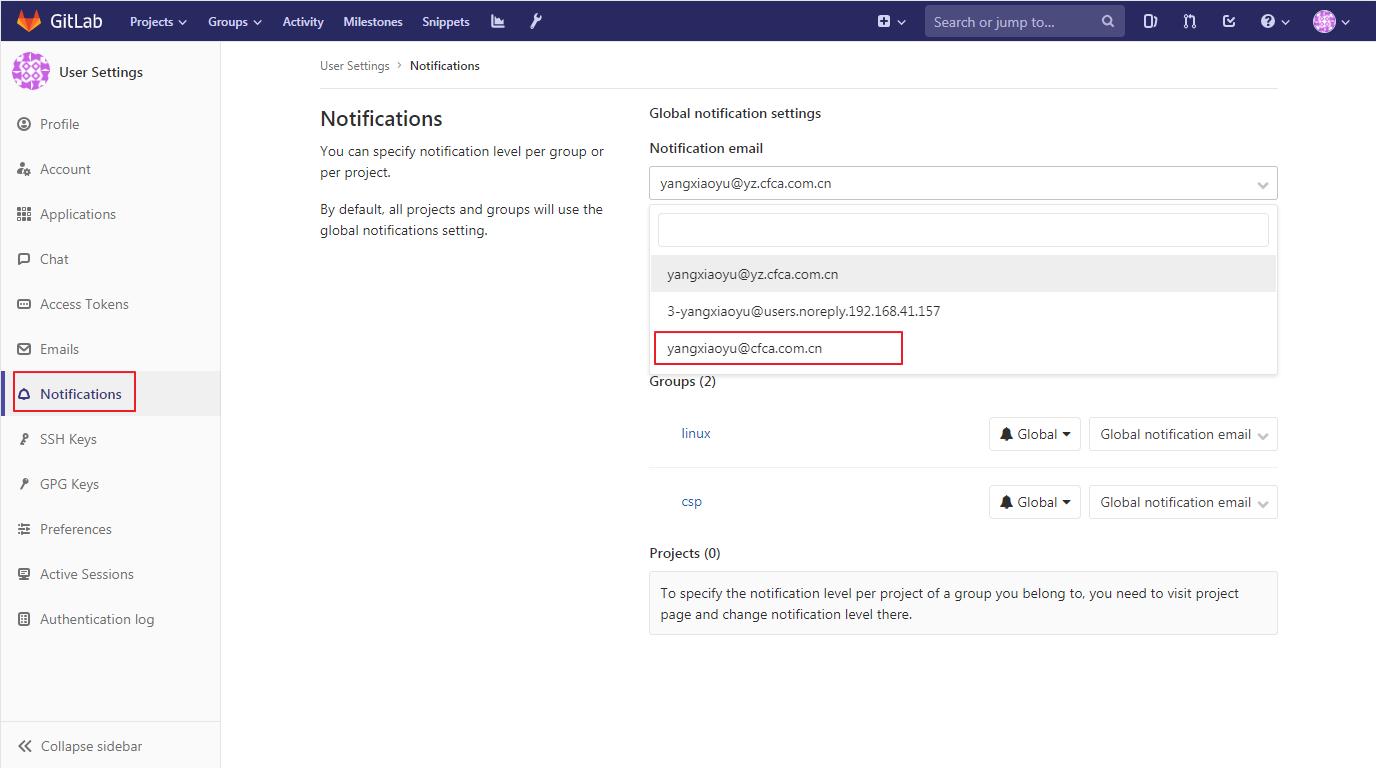




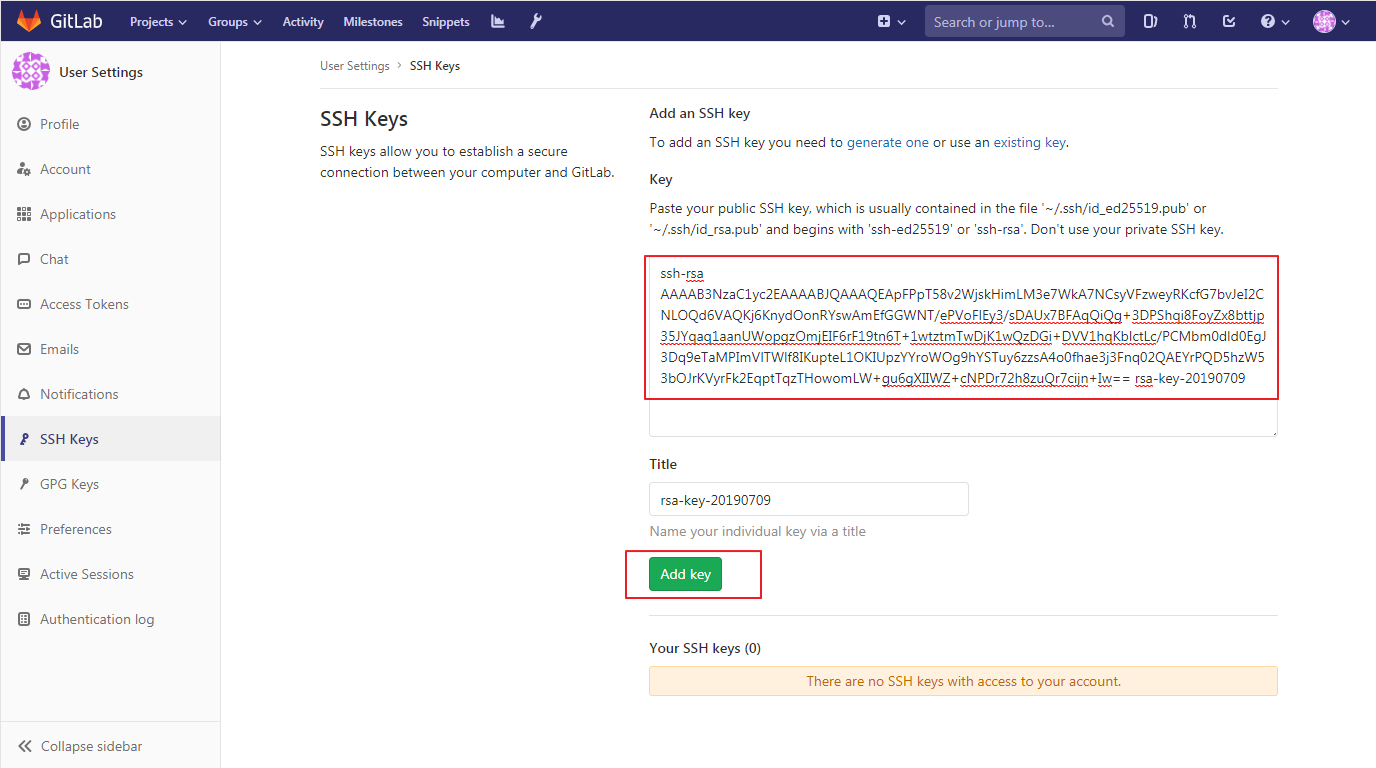
#### 变更公共和Commit邮箱



#### 变更通知邮箱



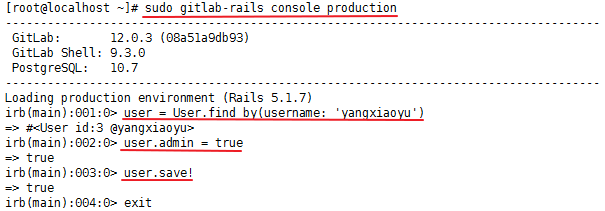
### 添加SSH公钥



### 用户提权

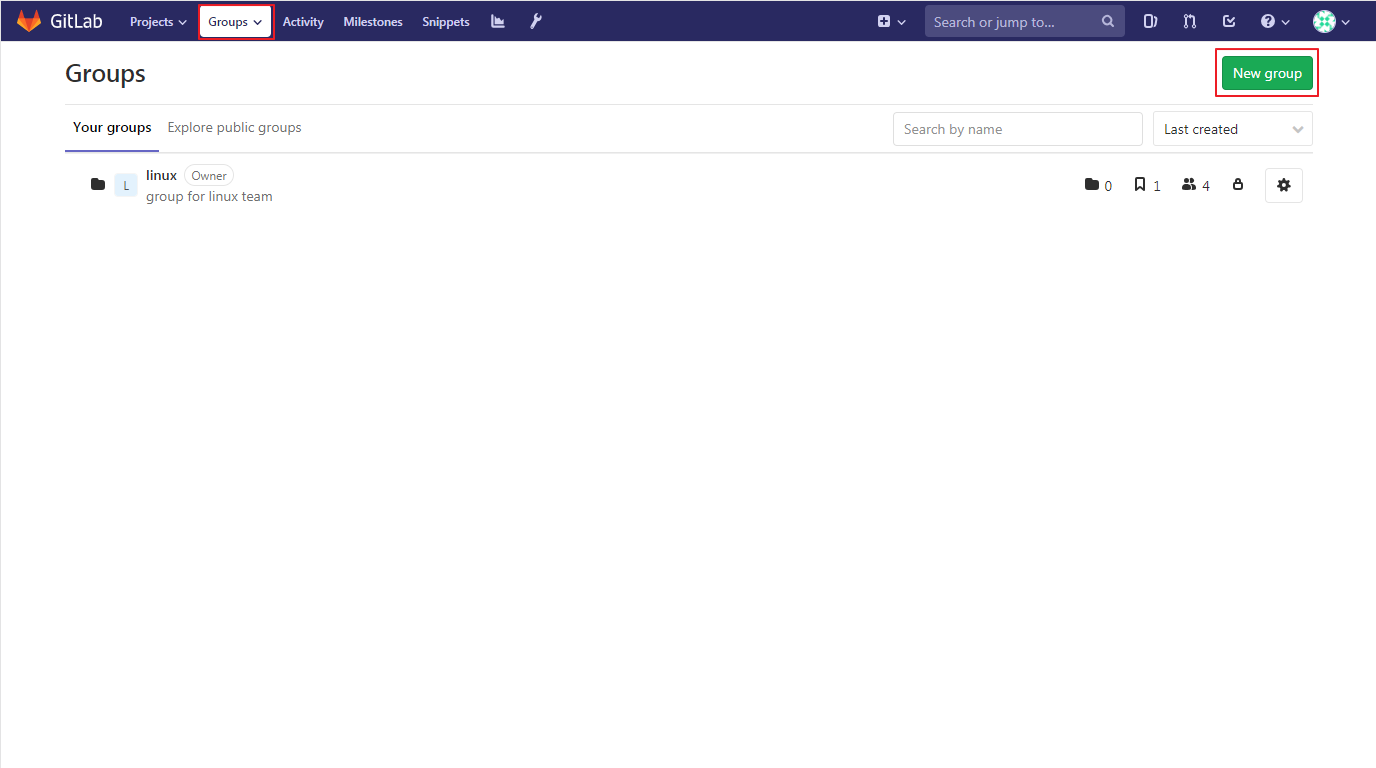
添加普通账号为管理员账号：

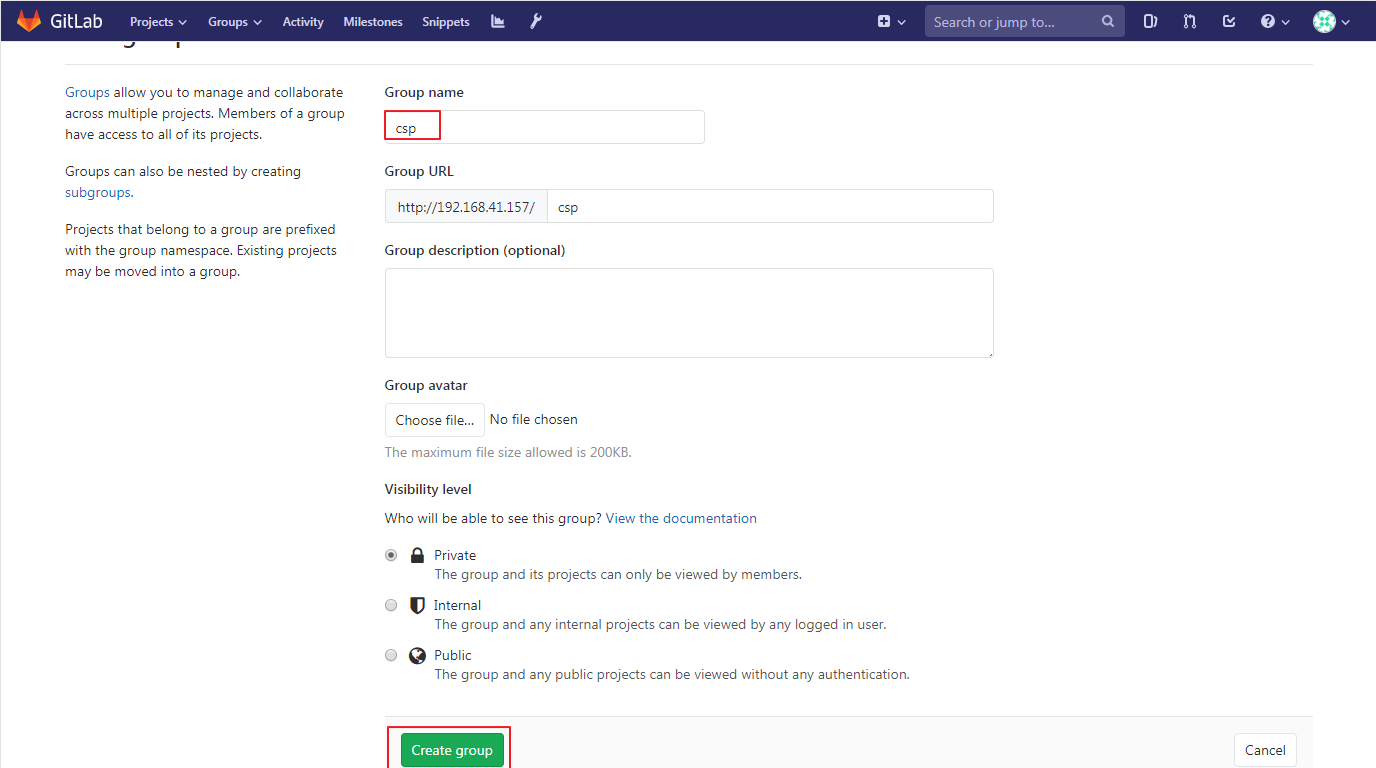
|  |  |
| --- | --- |
| sudo gitlab-rails console production | 打开gitlab rails console终端 |
| user = User.find\_by(username: 'yangxiaoyu') | 通过用户名查找用户 |
| user.admin = true | 赋予admin权限 |
| user.save! | 保存更改 |
| exit | 退出终端 |
| sudo gitlab-ctl reconfigure | 重新生成配置并生效 |



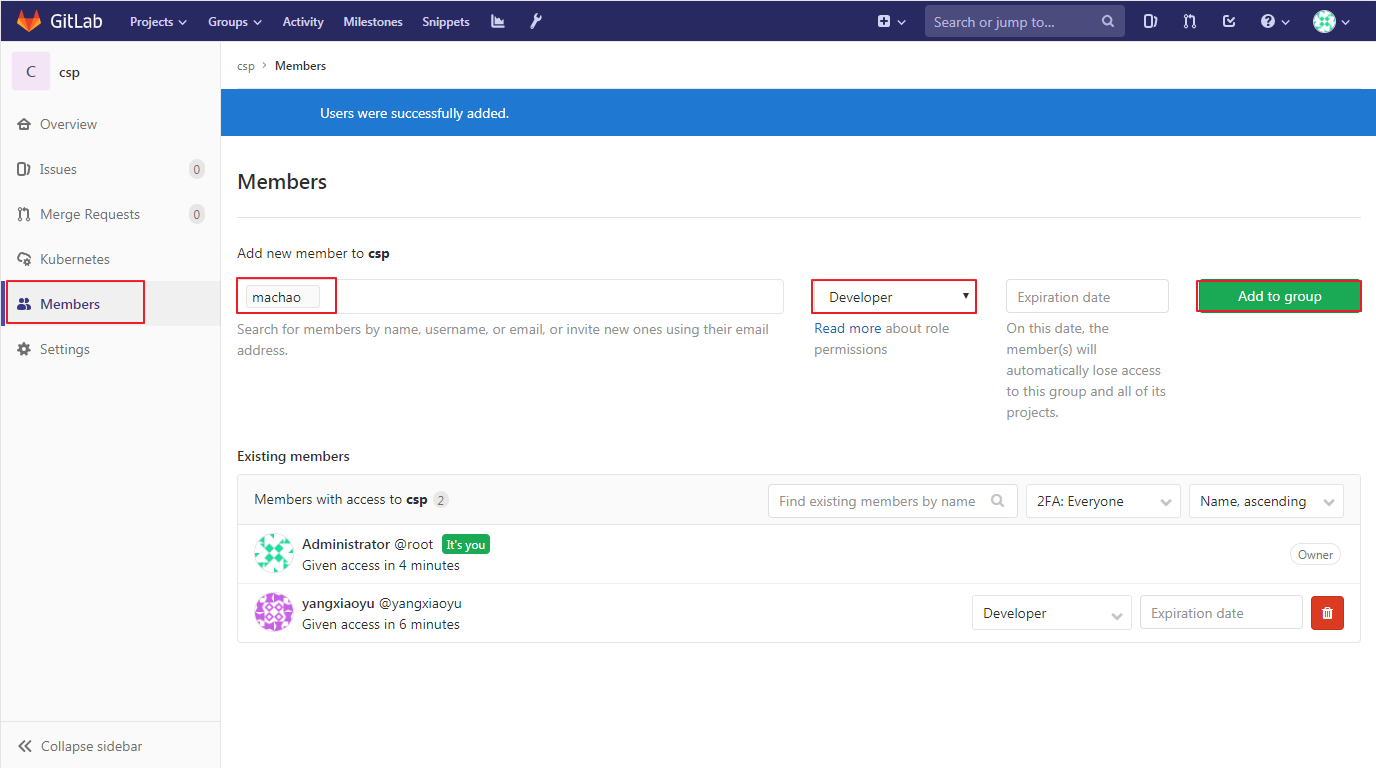
## 创建分组

### 创建分组

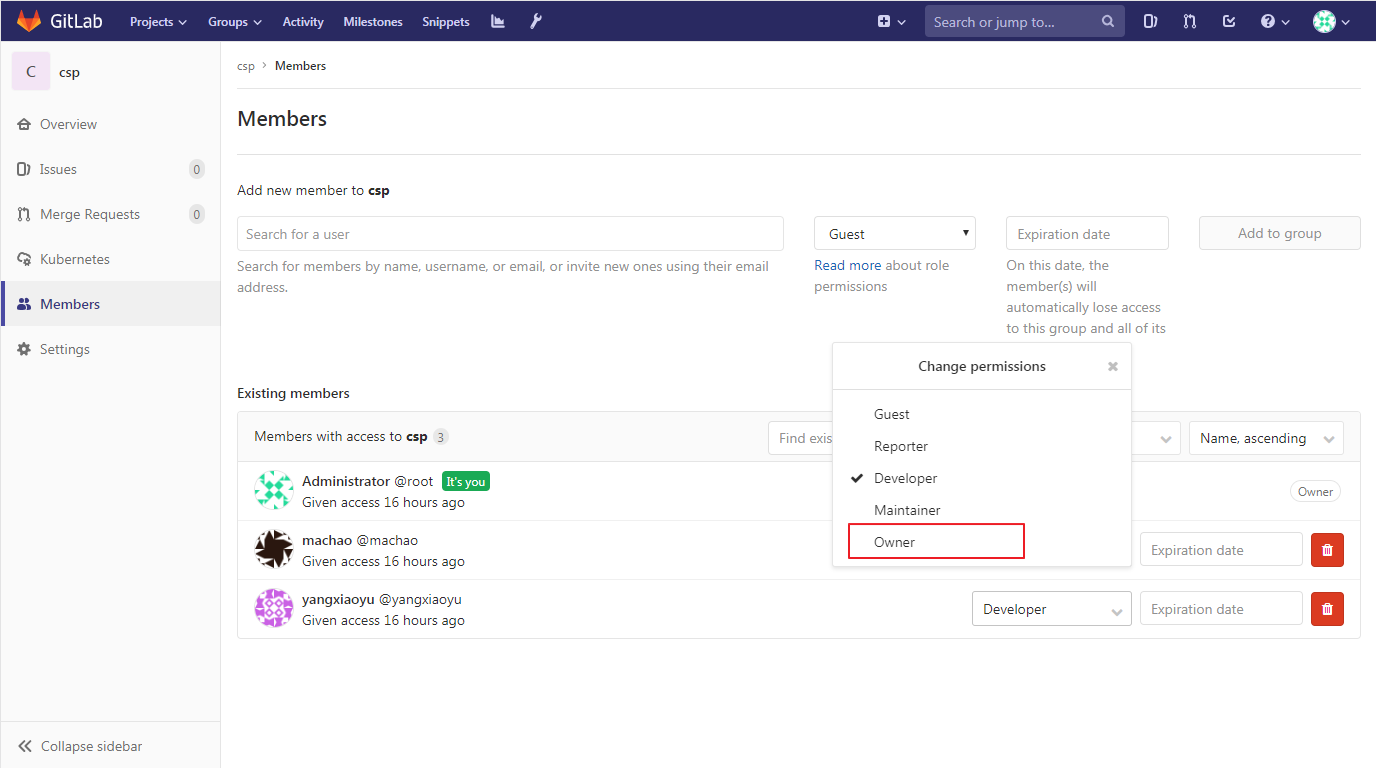


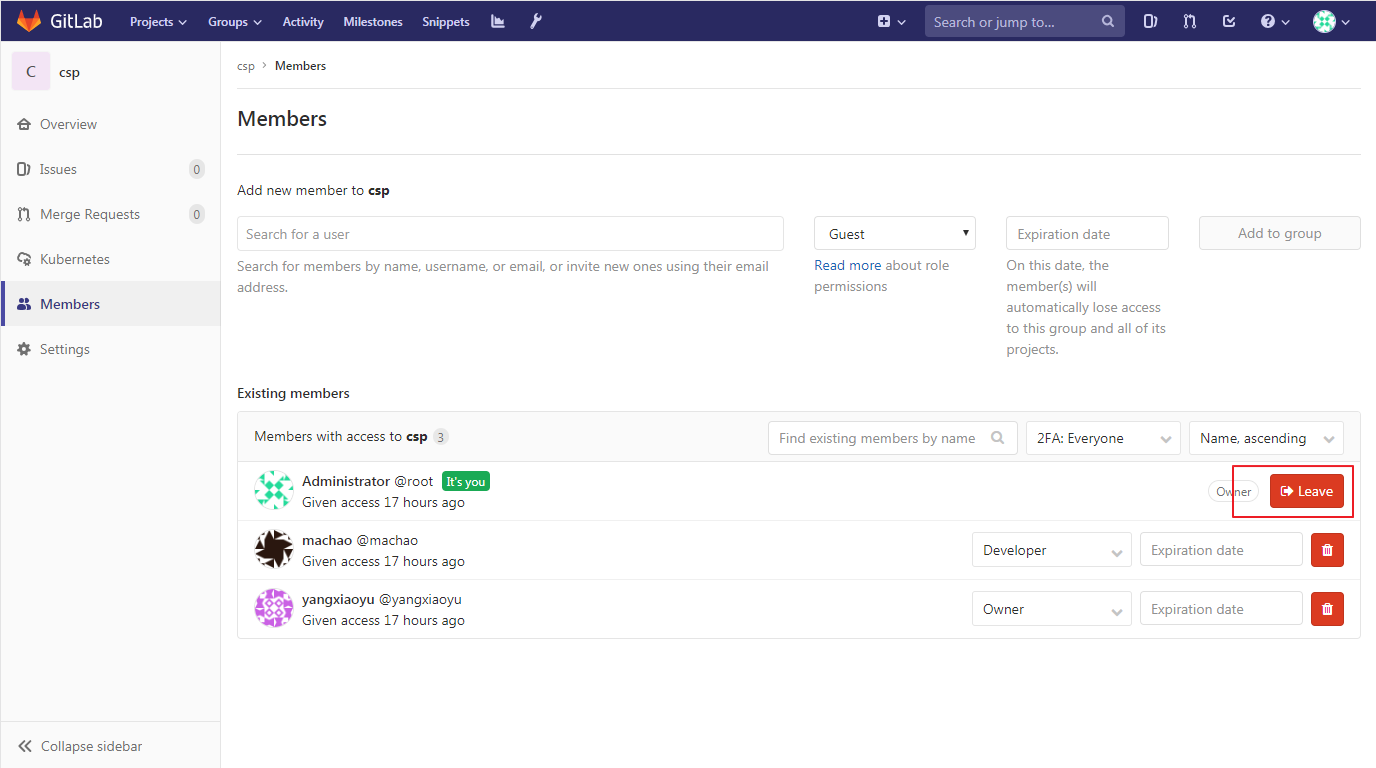


### 添加组成员



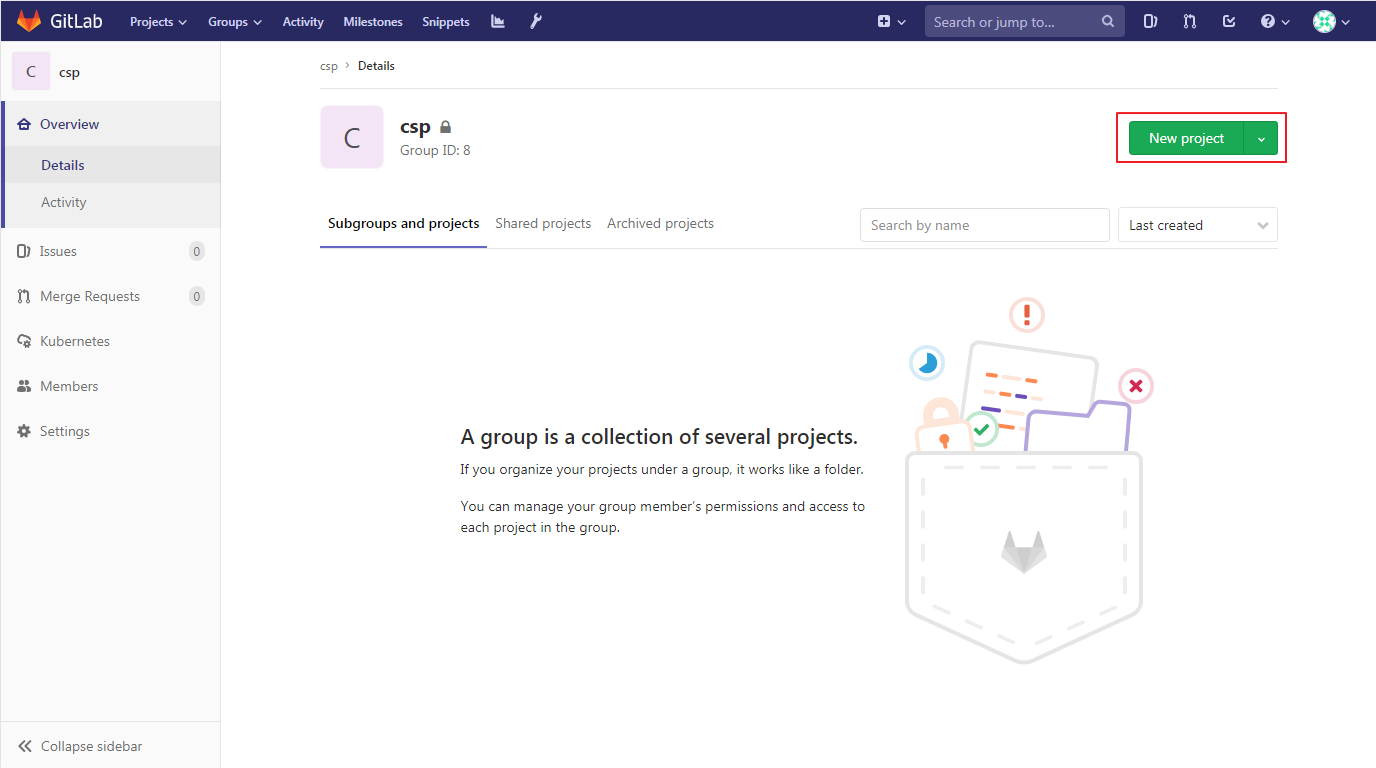
### 切换组所有者权限

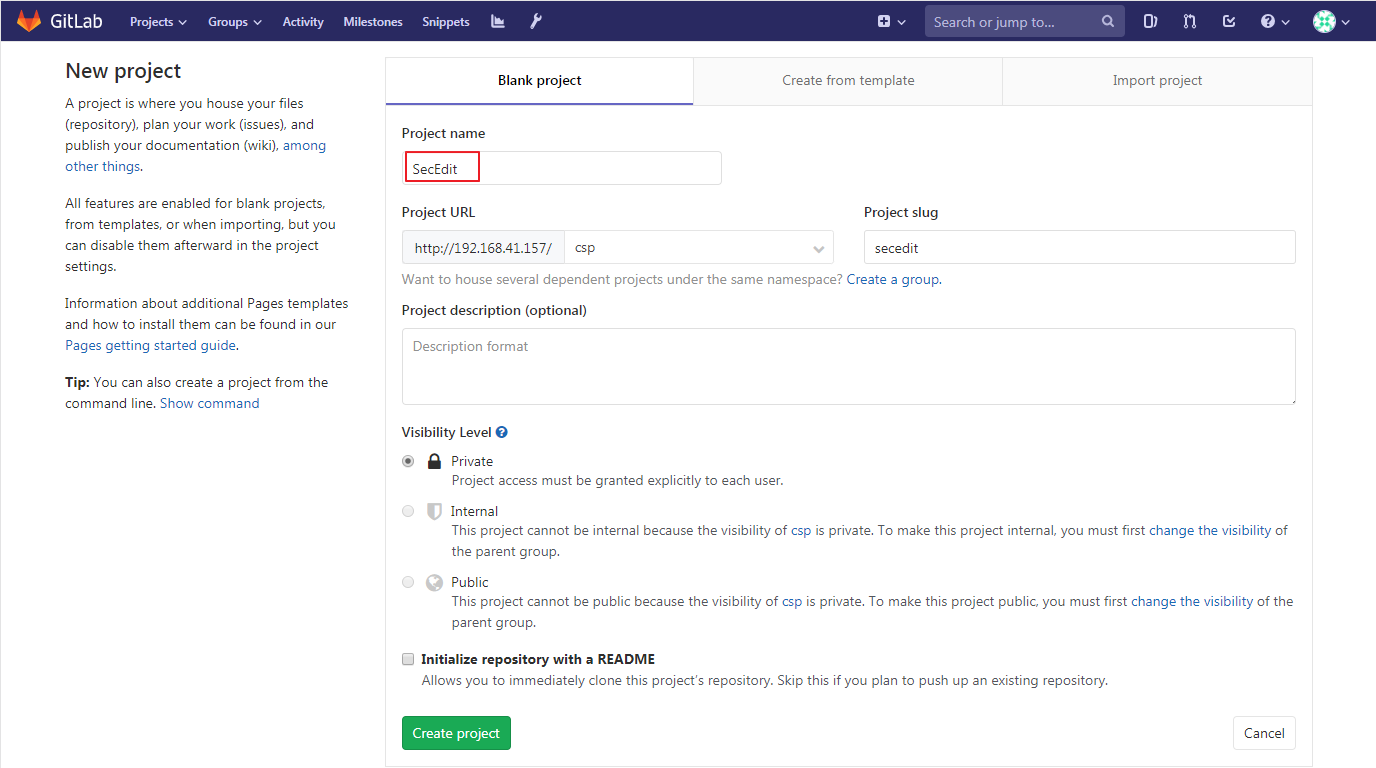


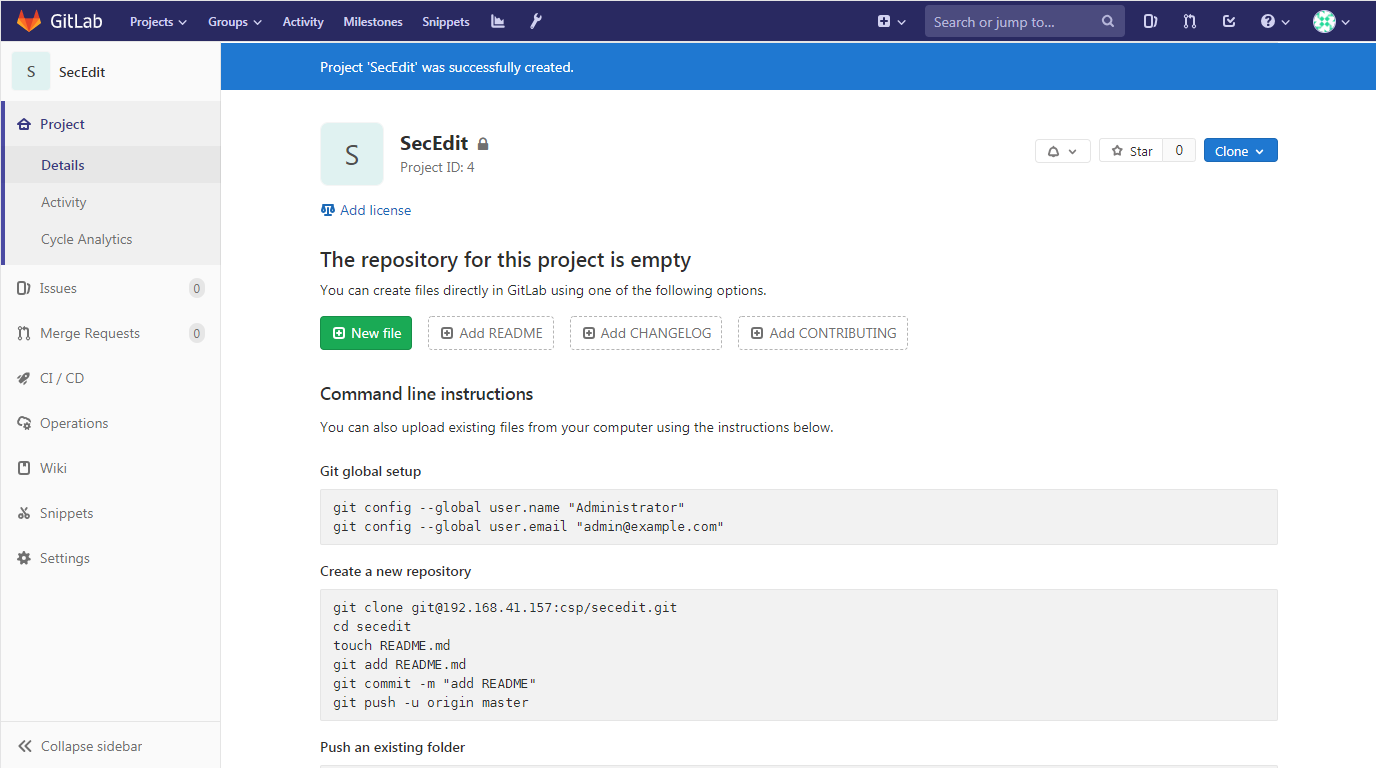


## 工程配置

### 添加工程







创建后会有一下命令行提示，可以根据命令行进行下一步操作：

Git全局变量设置

git config --global user.name "Administrator"

git config --global user.email "admin@example.com"

创建新的repository

git clone git@192.168.41.157:linux/sslgw.git

cd sslgw

touch README.md

git add README.md

git commit -m "add README"

git push -u origin master

上传已经存在的文件夹

cd existing\_folder

git init

git remote add origin git@192.168.41.157:linux/sslgw.git

git add .

git commit -m "Initial commit"

git push -u origin master

上传已经存在的repository

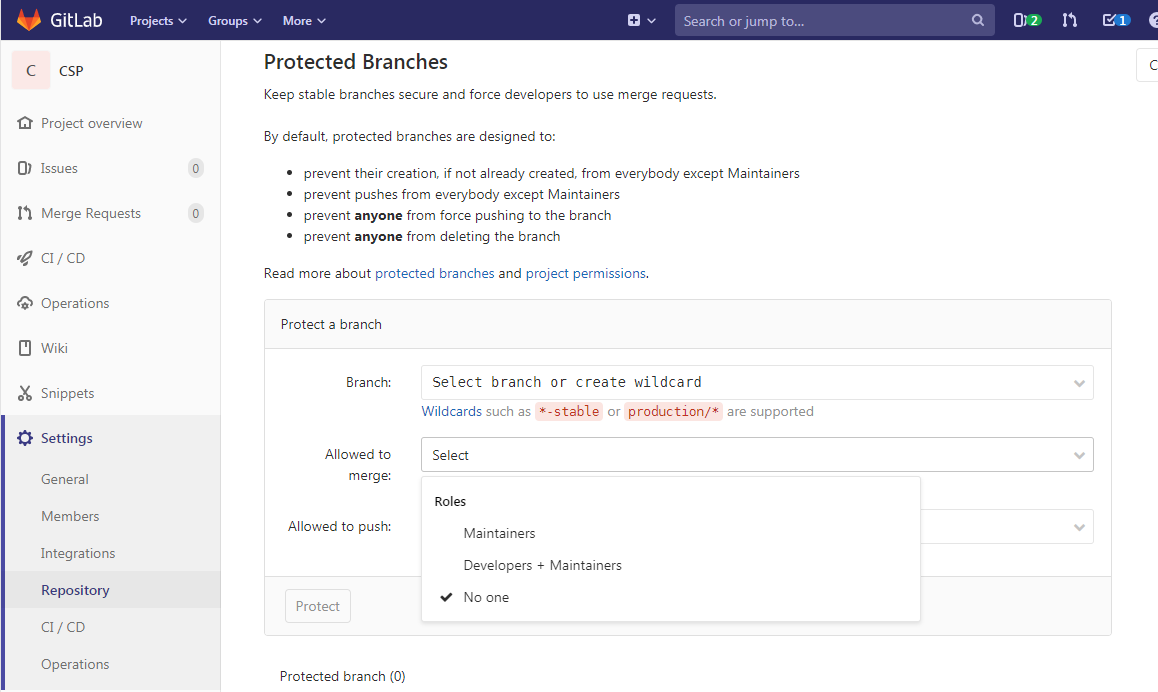
cd existing\_repo

git remote rename origin old-origin

git remote add origin git@192.168.41.157:linux/sslgw.git

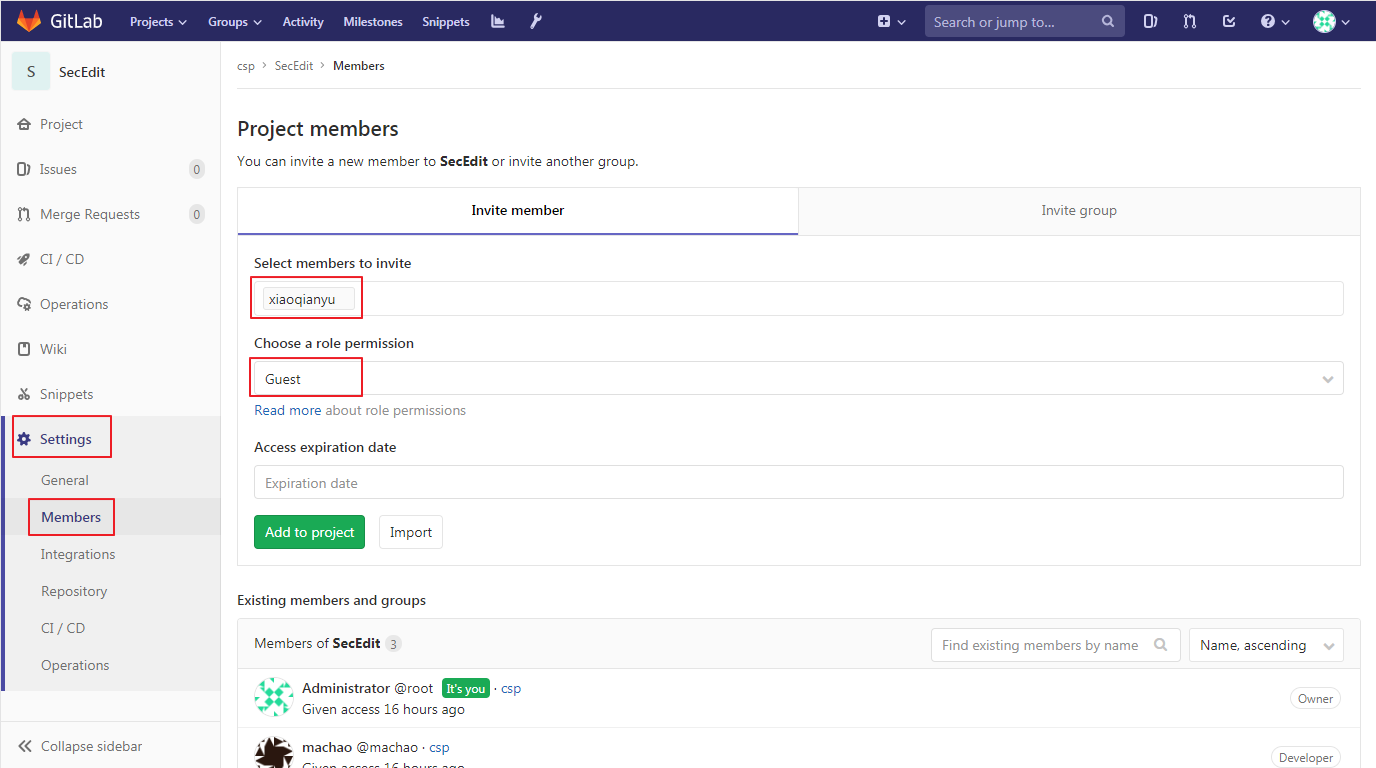
git push -u origin --all

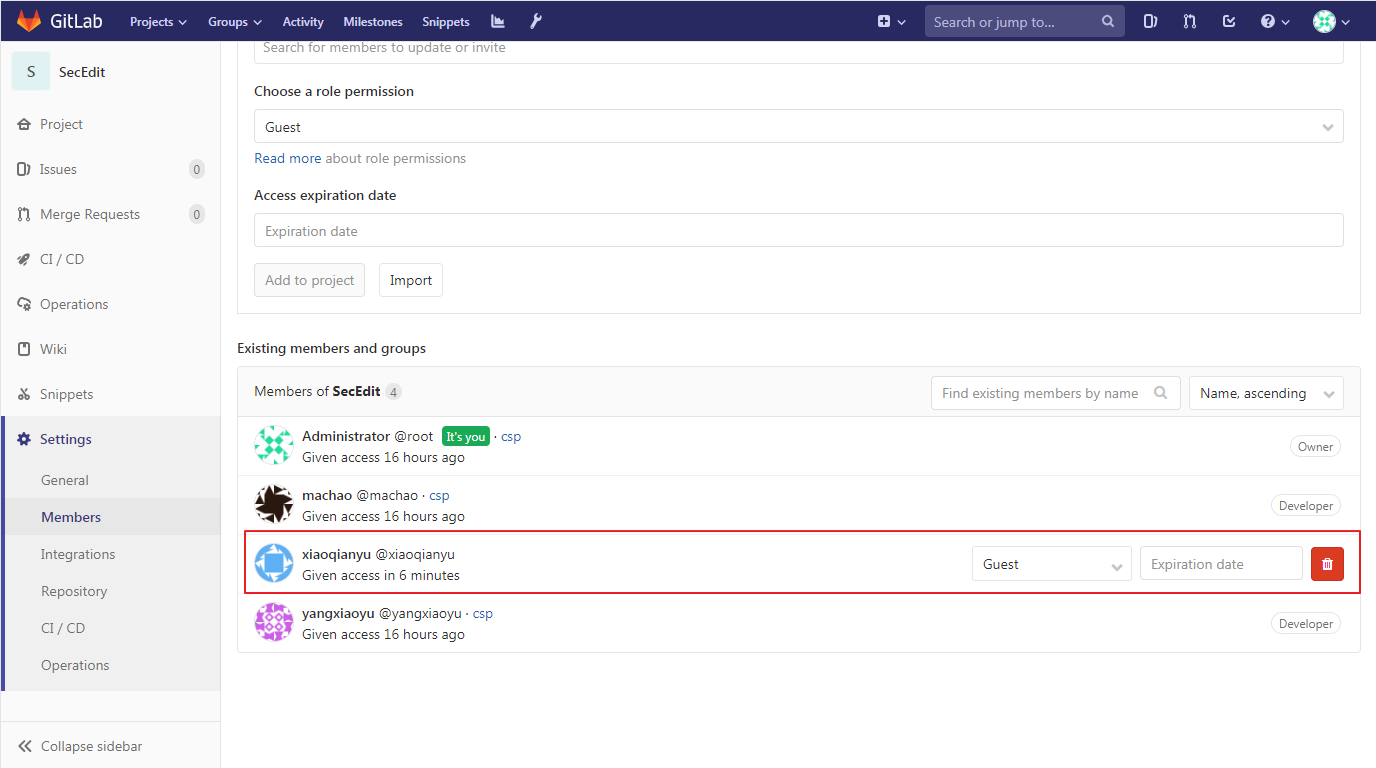
git push -u origin –tags



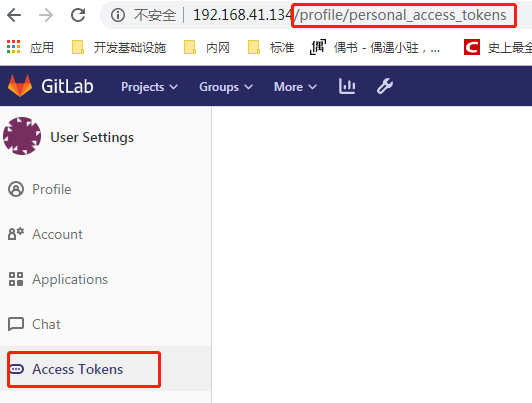
如果在git push -u origin master出现remote rejected错误，可能是因为master分支被设置为保护分支。可以在工程的setting-》Repository-》Protected Branches中Allowed to merge中选择可以允许merge的角色。

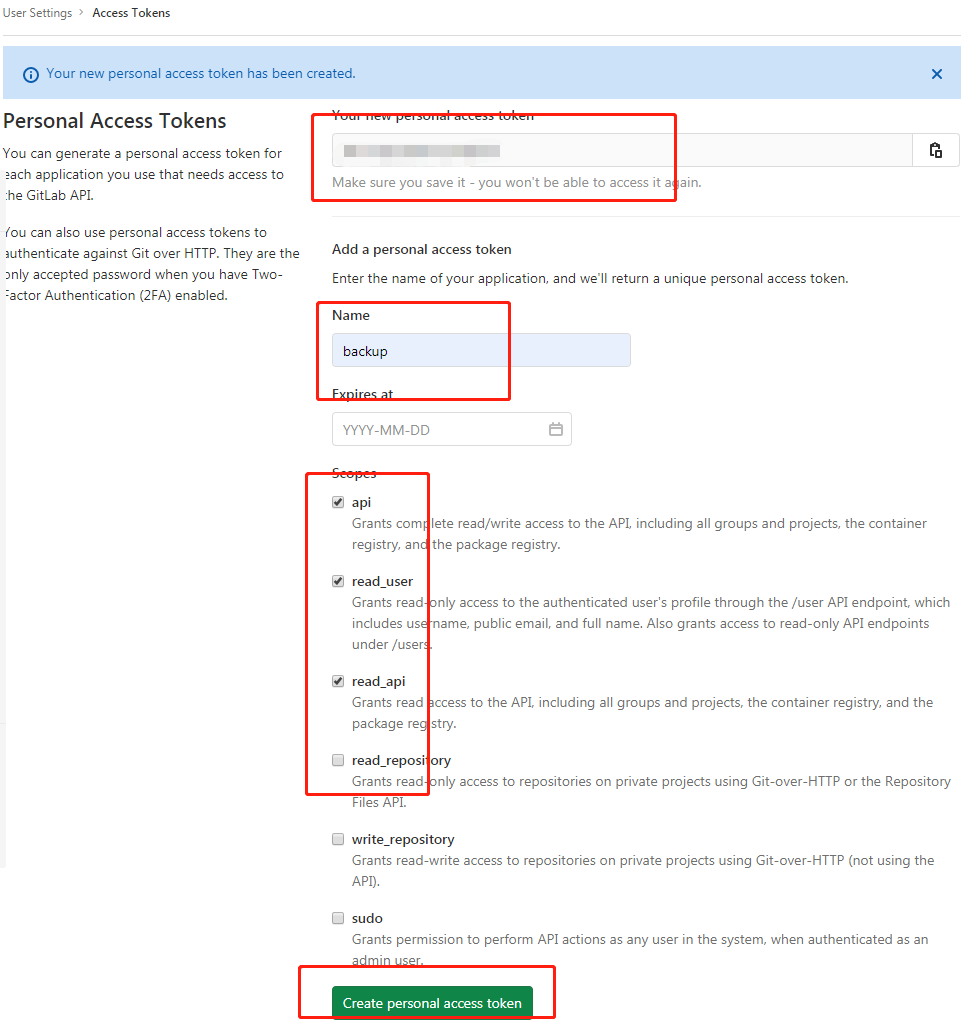
### 添加工程成员





# 命令行工具







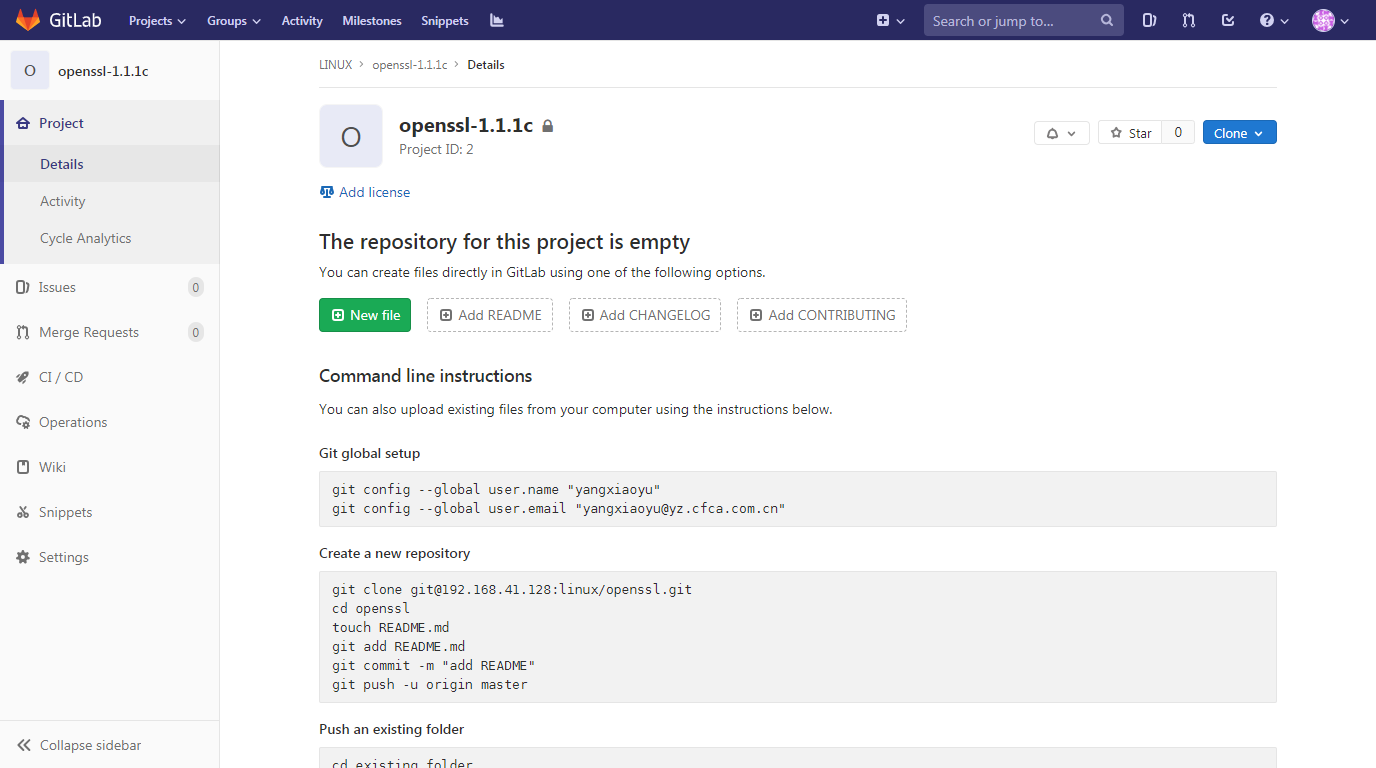
root@gitlabrd:/# export GITLAB\_API\_ENDPOINT=http://127.0.0.1/api/v4

root@gitlabrd:/# export GITLAB\_API\_PRIVATE\_TOKEN=实际生成的token

root@gitlabrd:/# export GITLAB\_API\_HTTPARTY\_OPTIONS="{verify: false}"

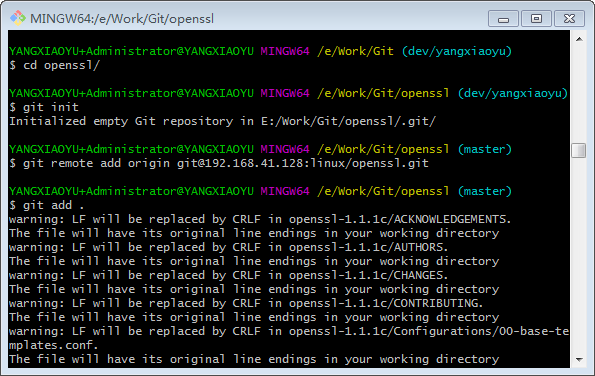
# 工程示例

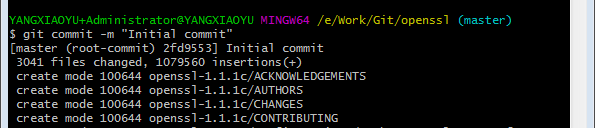
添加一个新工程例如openssl-1.1.1c，添加后如下图所示：

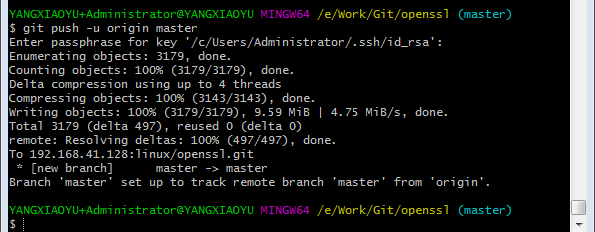


我们需要先把openssl-1.1.1c源码上传，根据页面提示：

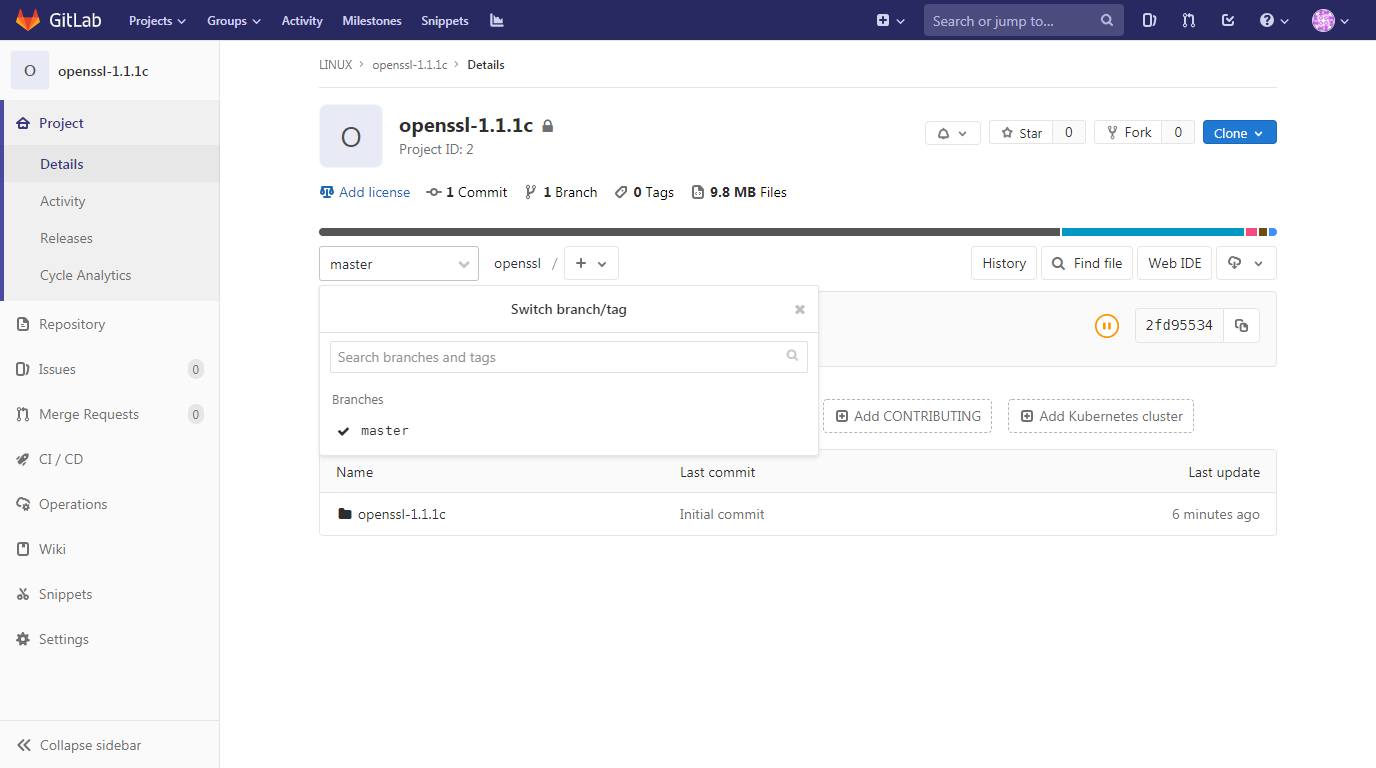








上传后如下：



# GitLab runner

<https://docs.gitlab.com/runner/>

安装

<https://docs.gitlab.com/runner/install/linux-repository.html>

添加GitLab官方库

curl -L https://packages.gitlab.com/install/repositories/runner/gitlab-runner/script.rpm.sh | sudo bash

安装最新版GitLab Runner

sudo yum install gitlab-runner

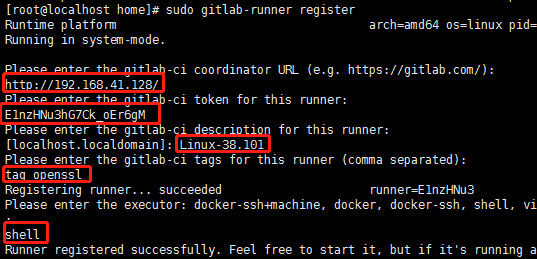
启动

sudo gitlab-runner start

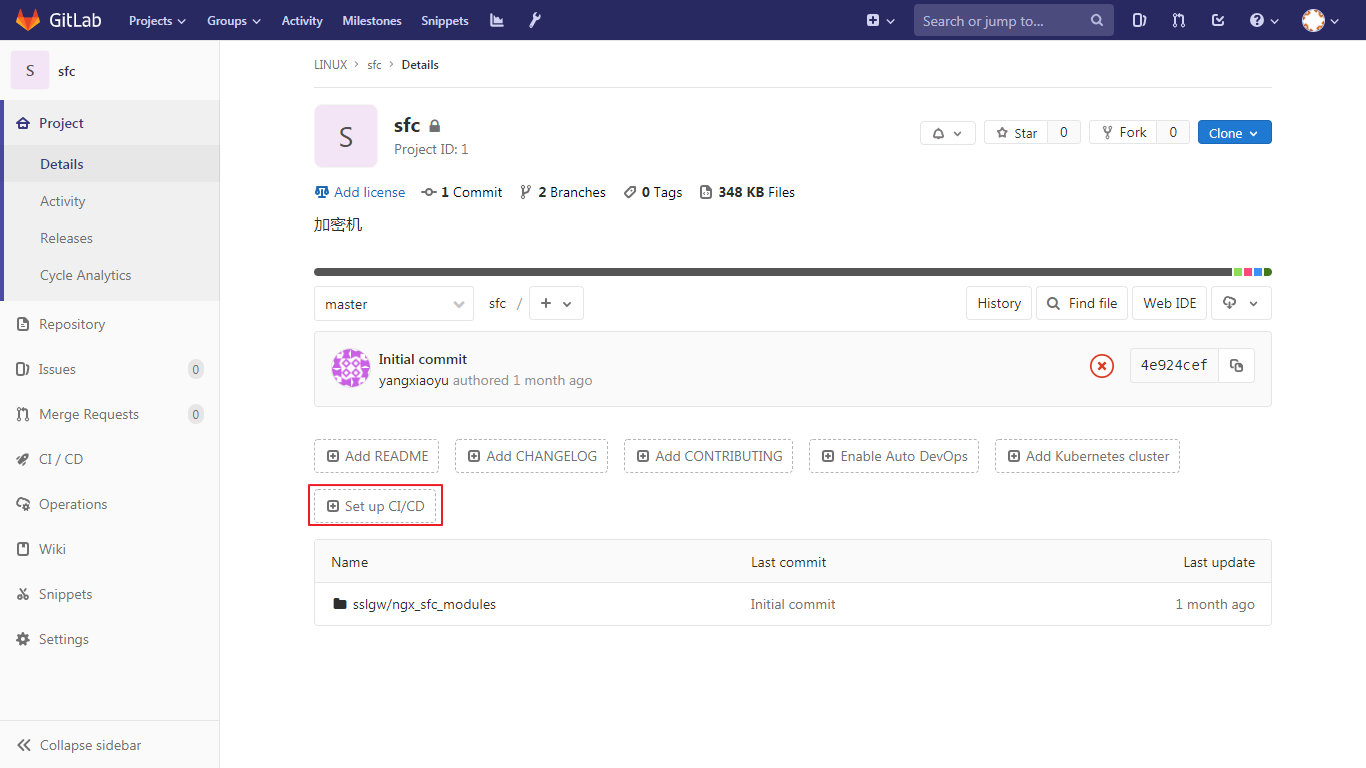
注册

sudo gitlab-runner register

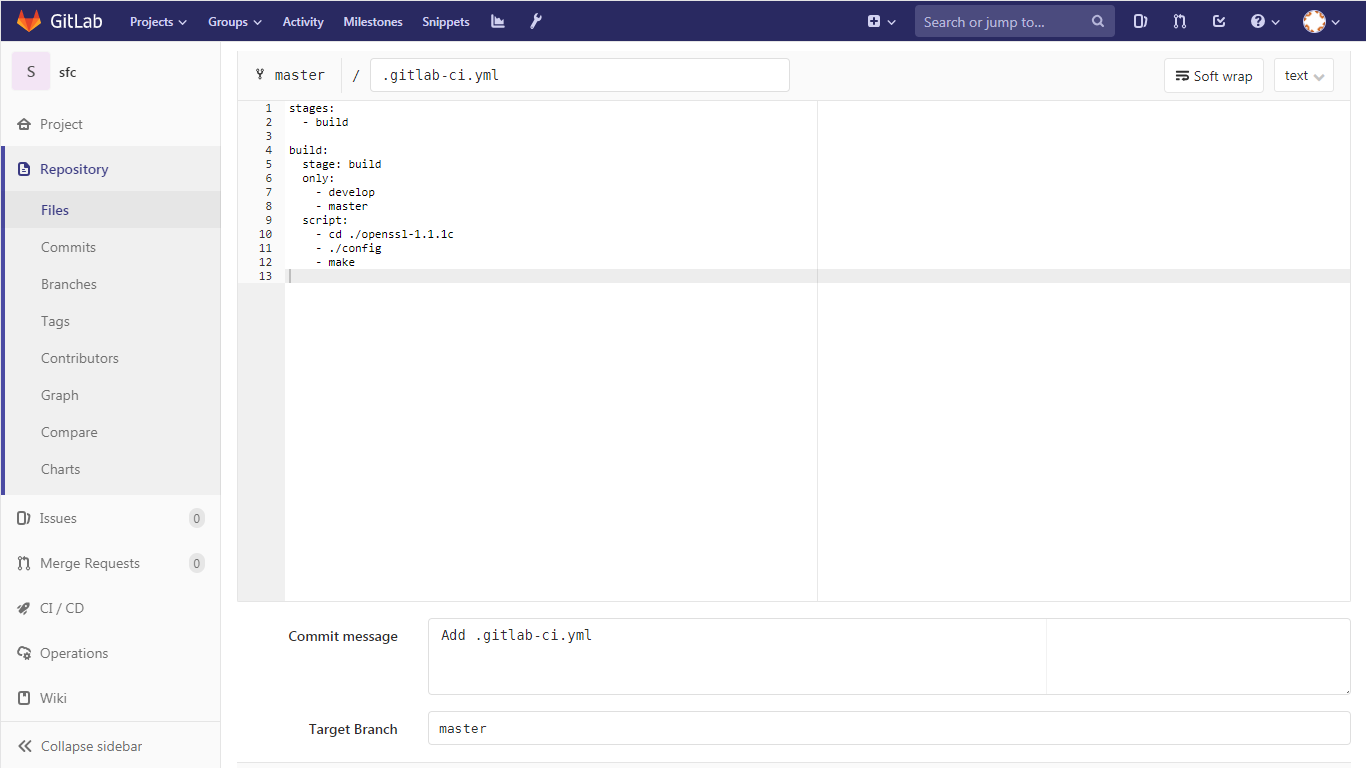
根据提示输入gitlab url，token，描述，tag，类型。

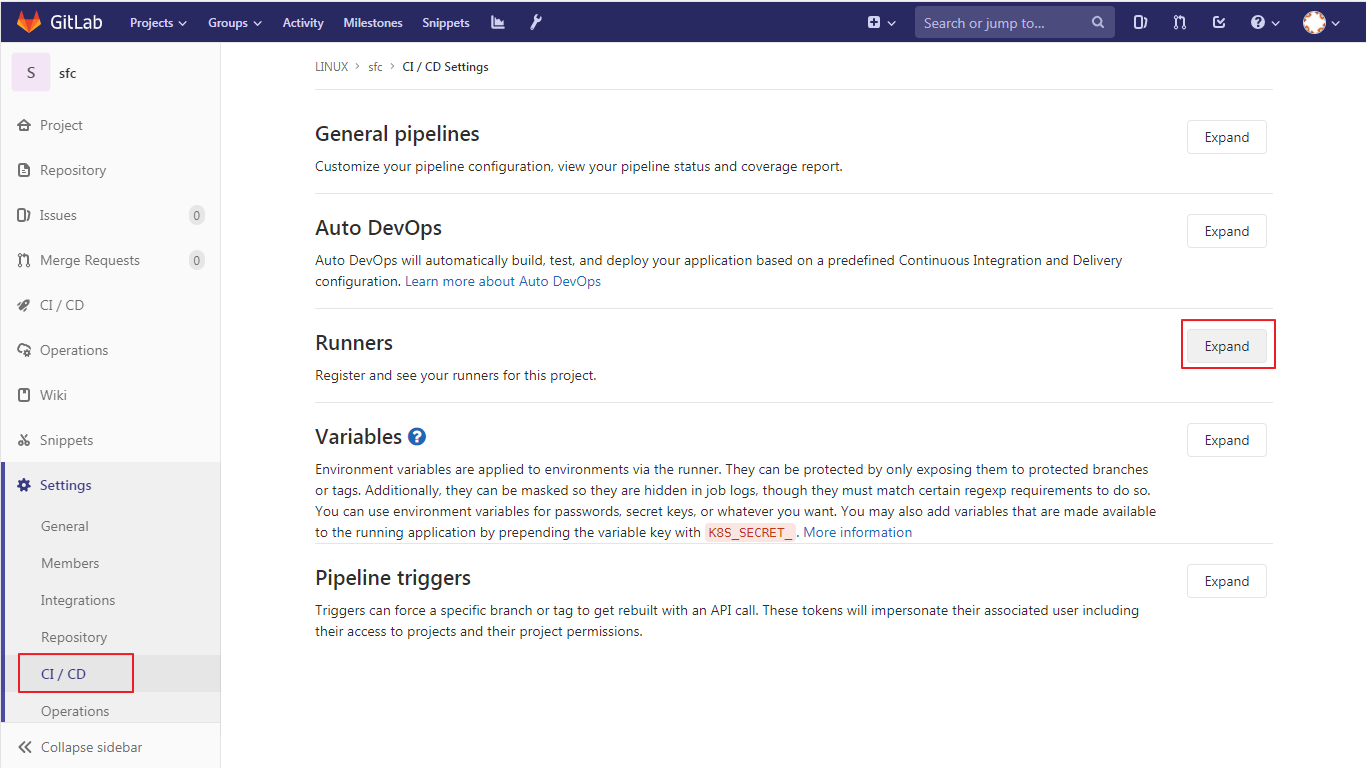


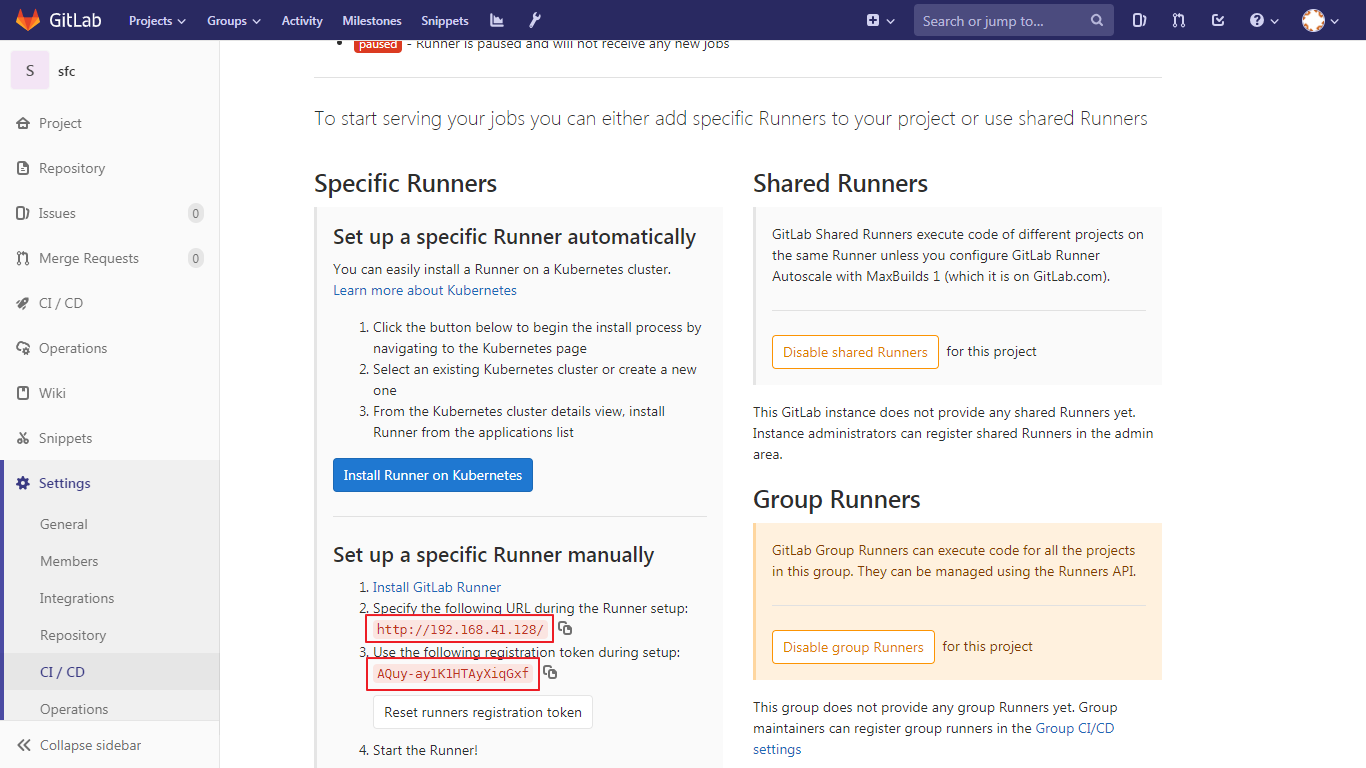
在gitlab管理界面上设置CI/CD

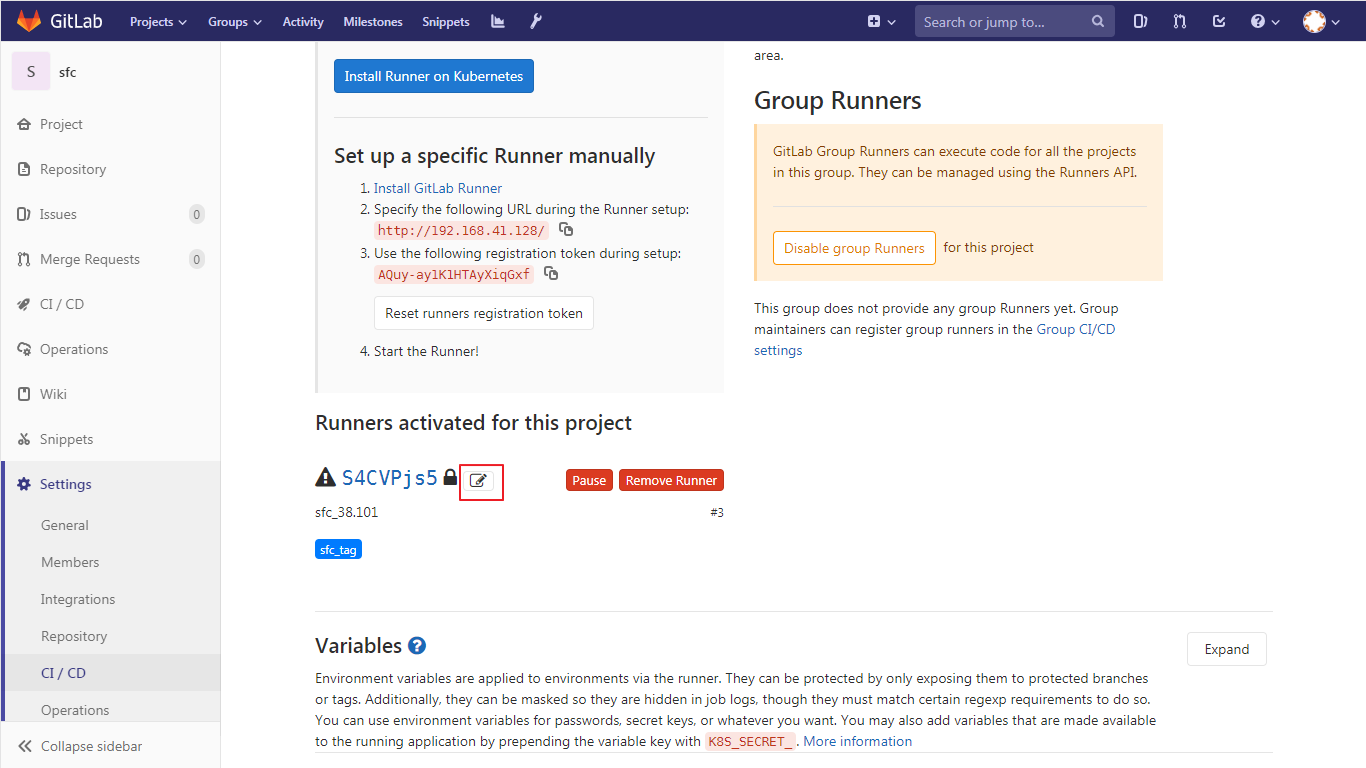


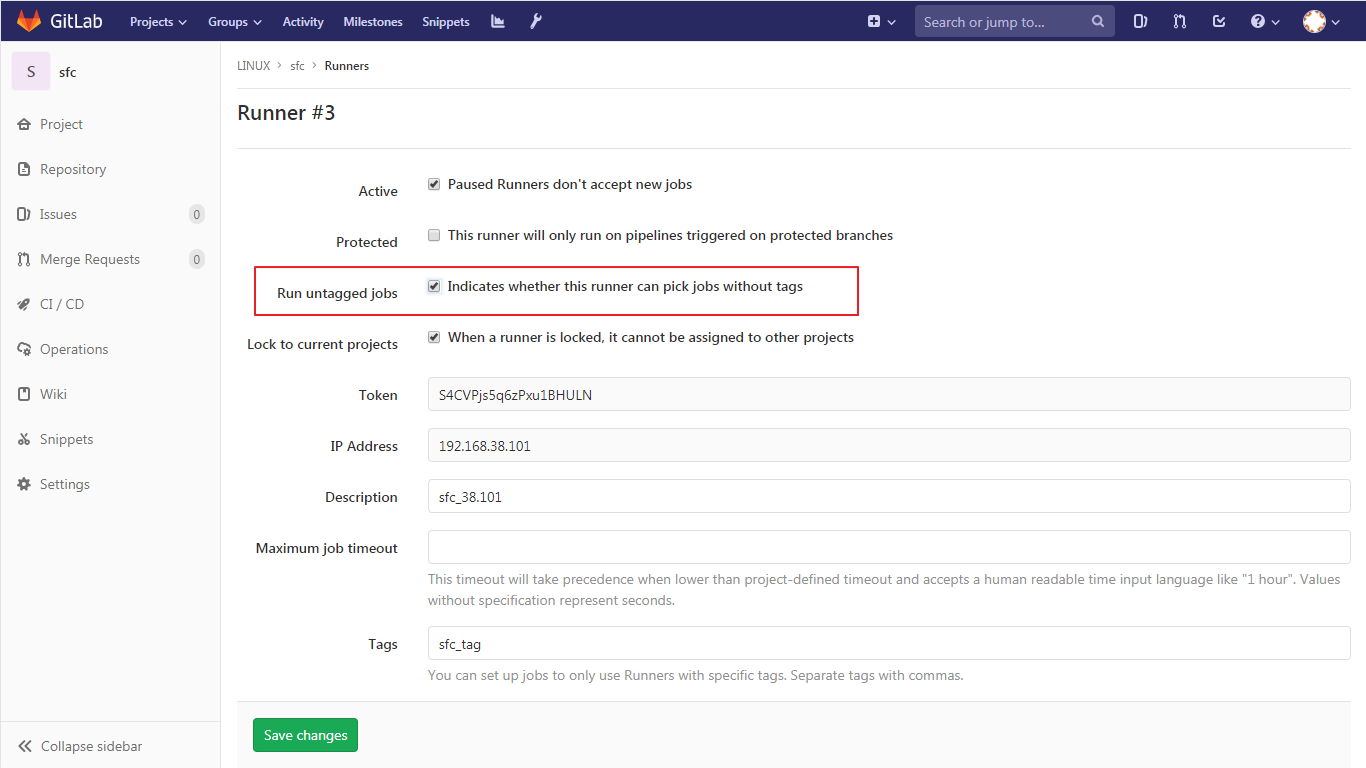
配置runner





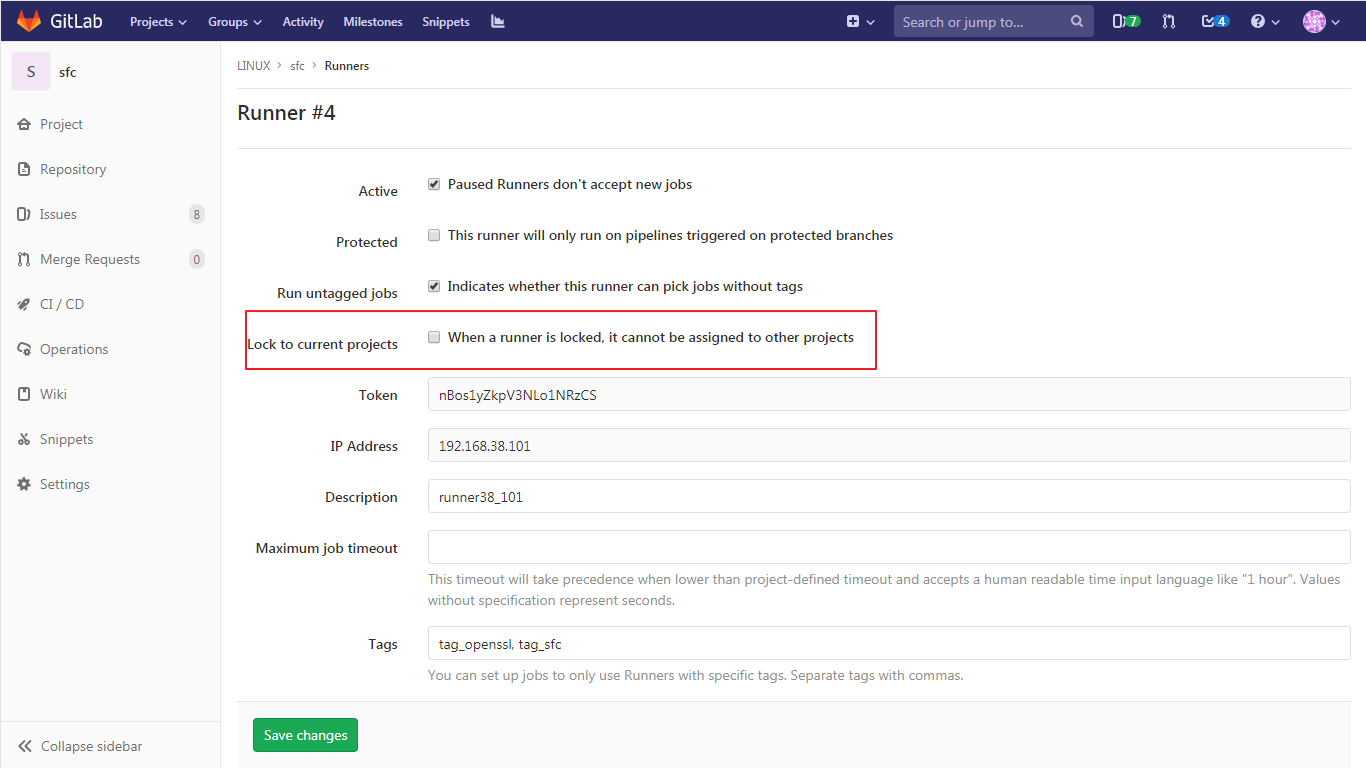




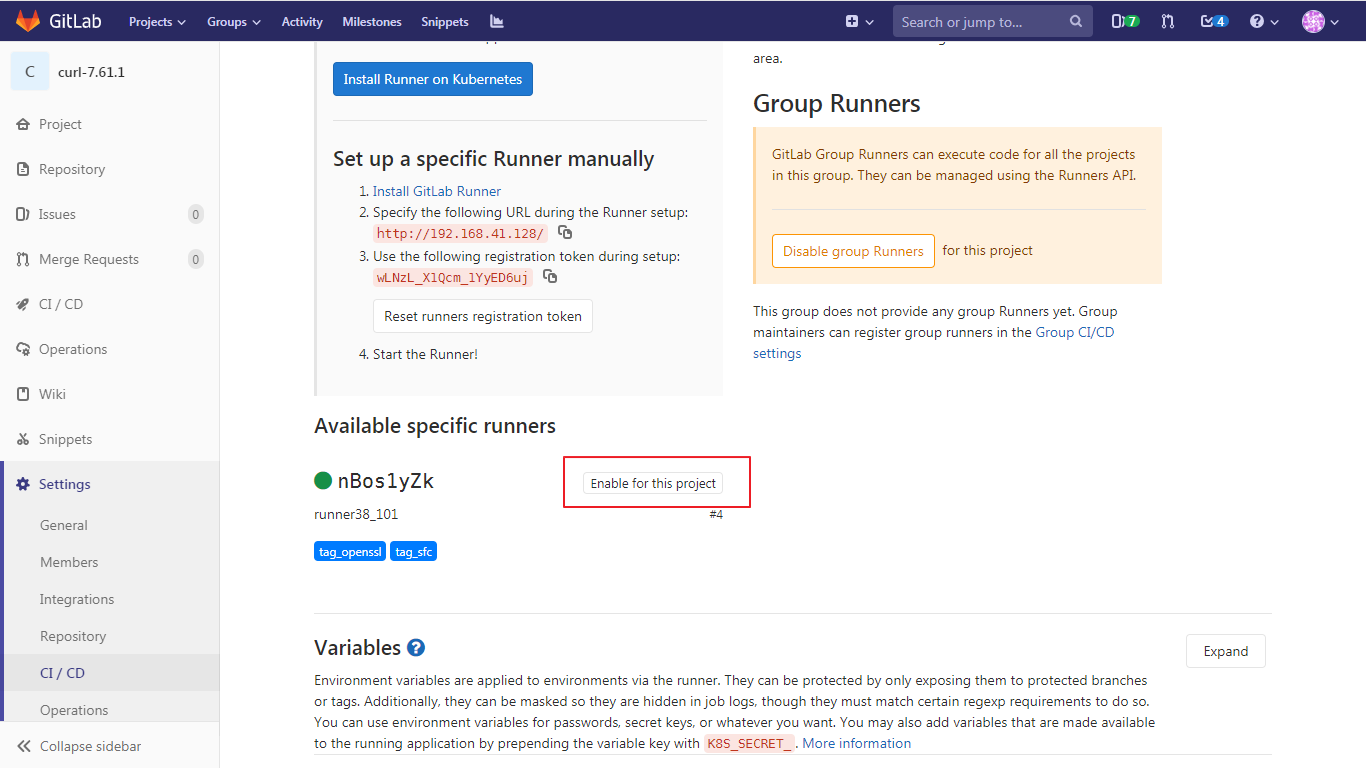


其他工程复用gitlab-runner

勾去“Lock to current projects”以便允许其他工程复用。



在其他项目的CI 配置中设置允许该Runner



在项目中引用子模块

如果在项目中使用第三方模块，比如在编译服务器密码机时使用到curl，或者protobuf，我们并不负责维护子模块，只是需要的时候同步更新子模块即可。

子模块添加：

git submodule add <url> <path>

其中，url为子模块的路径，path为该子模块存储的目标路径

执行成功后，git status会看到项目中修改了.gitmodules，并增加了一个新文件。

当使用gitlab-runner下载子模块时，限制了url只支持http(s)的路径，[不能使用git@xxx.git](mailto:不能使用git@xxx.git)路径。需要CI支持自动下载子模块时，需要在.gitlab-ci.yml文件中增加：

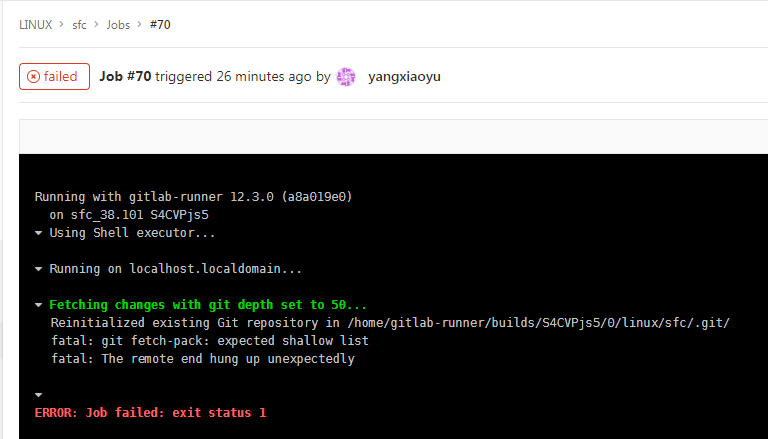
variables:

GIT\_SUBMODULE\_STRATEGY: normal

配置，参考[https://segmentfault.com/a/1190000010442764#articleHeader18](https://segmentfault.com/a/1190000010442764" \l "articleHeader18)

<https://docs.gitlab.com/ce/ci/git_submodules.html>

使用gitlab-runner出现错误



当出现上述错误可以考虑更新编译机器上的git。参考<https://blog.51cto.com/michaelkang/2387223?source=dra>。

1. 删除旧版本git

yum remove git -y

1. 安装第三方yum源

yum install https://centos7.iuscommunity.org/ius-release.rpm

centos7基础仓库只提供1.8.3版本，无法使用git2的新功能。

1. 安装新版本git

yum install git2u -y

查看版本号：git –version

1. 重新安装gitlab-runner

yum install gitlab-runner -y

# Git功能拾遗

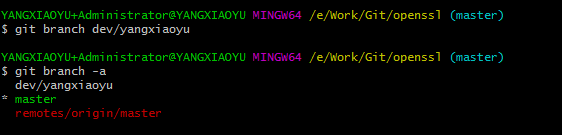
## 创建远程分支

本地新建了一个分支branch\_name，但是在远程没有。

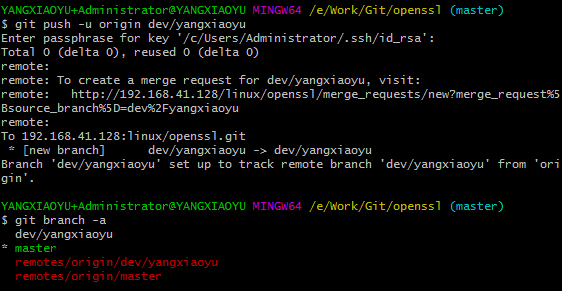
这时候 push 和 pull 指令就无法确定该跟踪谁，一般来说我们都会使其跟踪远程同名分支，所以可以利用 git push --set-upstream origin branch\_name 或者 git push -u origin branch\_name ，这样就可以自动在远程创建一个 branch\_name 分支，然后本地分支会 track 该分支。后面再对该分支使用 push 和 pull 就自动同步。

git branch dev/yangxiaoyu 创建本地分支

git branch -a 查看当前所有分支，包括远程和本地。



 git push -u origin dev/yangxiaoyu



git checkout dev/yangxiaoyu 切换到dev/yangxiaoyu分支。



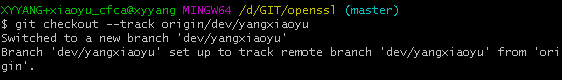
## 跟踪远程分支

如果远程新建了一个分支，本地没有该分支，可以利用“git checkout --track origin/branch\_name”，这时本地会新建一个分支名叫branch\_name，会自动跟踪远程的同名分支。

git branch -a 查看当前所有分支，包括远程和本地。



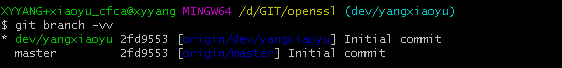
git checkout --track origin/dev/yangxiaoyu 自动跟踪远程的同名分支



我们再次查看可以看到新增了一个dev/yangxiaoyu的分支

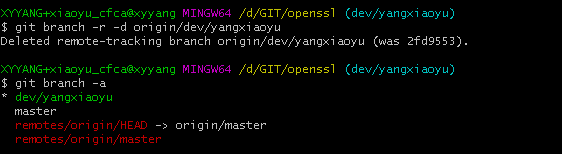


git branch -vv 查看本地分支与远程分支的对于关系



## 删除远程分支

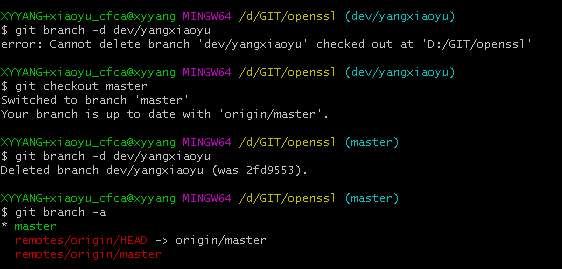
git branch -r -d origin/branch\_name 删除远程branch\_name分支



## 删除本地分支

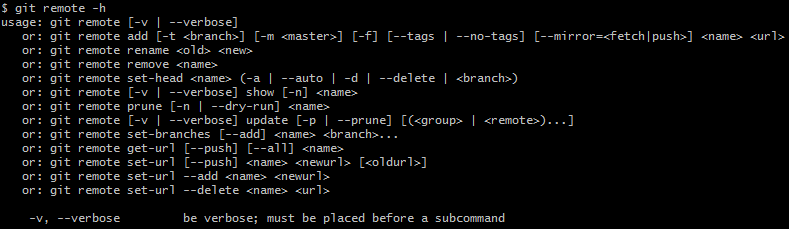
git branch -d branch\_name 删除本地branch\_name分支

当本地分支被checkout时，不能被删除。需先切换到其他分支再进行删除。



## 切换URL

Git可以使用SSH协议，也可以使用HTTP协议。使用SSH协议时通常需要配置秘钥，HTTP协议需要知道用户名密码。两者可以使用git remote命令进行切换。



查看当前remote

git remote -v



切换到HTTP协议：



切换后查看remote信息



## 子模块（submodule）

添加submodule

git submodule add 仓库地址 目标路径（在当前项目中的路径）

拉取代码时包含submodule

git clone <http://xxxx.git> --recurse-submodules

已下载的项目中没有下载submodule，重新下载submodule

git submodule init #初始化本地子模块配置

git submodule update

submodule有更新时拉取

git submodule update -remote

## 下载指定分支代码

git clone -b branch\_name <http://xxxx.git>

## 比较不同分支某个文件差异

git diff branch1 branch2 filepath

## 使用指定分支模块指定文件覆盖当前分支同名文件

git checkout branch\_name --filepath(filepath 可以是文件夹路径)

## 撤销本地commit

git reset HEAD~

## 撤销已暂存未提交的文件

即使用了git add file的文件,并未git commit的文件

git reset HEAD file

## 删除untracked文件

#删除untracked文件

git clean -f

#连untracked的目录一起删除

git clean -fd

#连gitignore的untracked文件目录一起删除（一般用来删除编译出来的.o之类的文件）

git clean -xfd

#用在上述git clean前，建议加上-n参数先看看会删除哪些文件，以防止重要文件被误删。

git clean -nxfd

git clean -nf

git clean -nfd

## 配置比较工具

git config --global diff.tool bc4

git config --global difftool.bc4.cmd '"D:\Program Files\Beyond Compare 4\BCompare.exe" "$LOCAL" "$REMOTE"'

git config --global merge.tool bc4

git config --global mergetool.prompt false

git config --global mergetool.bc4.cmd '"D:\Program Files\Beyond Compare 4\BCompare.exe" "$LOCAL" "$REMOTE" "$BASE" "$MERGED"'

git config --global mergetool.bc4.trustexitcode true

## git stash

（1）**git stash** save "save message"  : 执行存储时，添加备注，方便查找，只有git stash 也要可以的，但查找时不方便识别。

（2）**git stash list** ：查看stash了哪些存储

（3）**git stash show** ：显示做了哪些改动，默认show第一个存储,如果要显示其他存贮，后面加stash@{$num}，比如第二个 git stash show stash@{1}

（4）**git stash show -p** : 显示第一个存储的改动，如果想显示其他存存储，命令：git stash show  stash@{$num}  -p ，比如第二个：git stash show  stash@{1}  -p

（5）**git stash apply** :应用某个存储,但不会把存储从存储列表中删除，默认使用第一个存储,即stash@{0}，如果要使用其他个，git stash apply stash@{$num} ， 比如第二个：git stash apply stash@{1}

（6）**git stash pop** ：命令恢复之前缓存的工作目录，将缓存堆栈中的对应stash删除，并将对应修改应用到当前的工作目录下,默认为第一个stash,即stash@{0}，如果要应用并删除其他stash，命令：git stash pop stash@{$num} ，比如应用并删除第二个：git stash pop stash@{1}

（7）**git stash drop** stash@{$num} ：丢弃stash@{$num}存储，从列表中删除这个存储

（8）**git stash clear** ：删除所有缓存的stash