# 需要搭建 linux 系统

# git搭建

因为 **gitHub** 上的项目是公开的，不适合公司内部项目放在上面，而私人的需要收费，这绝非是我们愿意的。所以找了个跟 **gitHub** 很相似，但是又免费的 **gitLab** 。现在将搭建 **gitLab** 过程记录一下留作参考。

## gitLab介绍

[GitLab](https://about.gitlab.com/) 是利用 **Ruby on Rails** 一个开源的版本管理系统，实现一个自托管的Git项目仓库，可通过Web界面进行访问公开的或者私人项目。它拥有与Github类似的功能，能够浏览源代码，管理缺陷和注释。可以管理团队对仓库的访问，它非常易于浏览提交过的版本并提供一个文件历史库。团队成员可以利用内置的简单聊天程序(Wall)进行交流。它还提供一个代码片段收集功能可以轻松实现代码复用，便于日后有需要的时候进行查找。

## 依赖组件

ruby 1.9.3+，MySQL，git，redis， Sidekiq。最低配置CPU 1G，RAM 1G+swap可以支持100用户。

## 连接SSH

1. 首先要确定 **ubuntu** 上安装并且启动了 **ssh** 服务,然后还要确保 **linux** 系统上的防火墙要关闭, **win8** 的防火墙可以正常开启
2. **SSH** (仿真终端工具有PUTTY，SecureCRT 以及Fterm等）

有人说首先要在 **Virtual Network Editor** 做 **NAT** 的端口映射,其实是不需要的;

**ps** -**e**|**grep** ssh

如果结果中只有 **ssh-agent** ,那说明 **ssh-server** 还没有启动,一般 **ubuntu** 是默认安装 **ssh** 的,要是没有安装,可以通过

sudo apt-get **install** openssh-**server**

3.安装ssh服务

**sudo** /etc/init.d/ssh start

初始化root密码

**ubuntn** 刚安装时候 **root** 还没有密码，设置一个，使用命令

**sudo** passwd

## 安装gitlib整合包

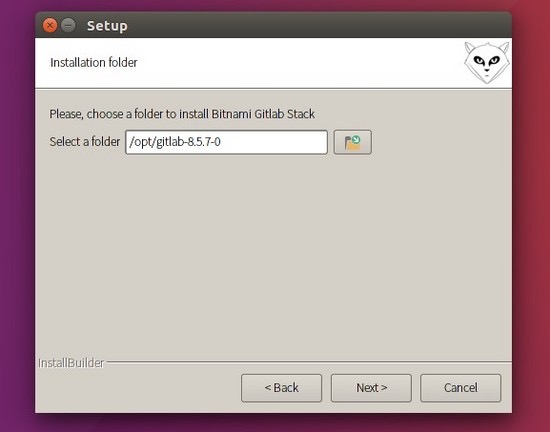
[官方有安装包](https://about.gitlab.com/downloads/) 与脚本下载， [官方安装指南](https://about.gitlab.com/installation/) 。同样GITHUB上有个社区非官方的 [安装指南](https://github.com/gitlabhq/gitlab-recipes) 。

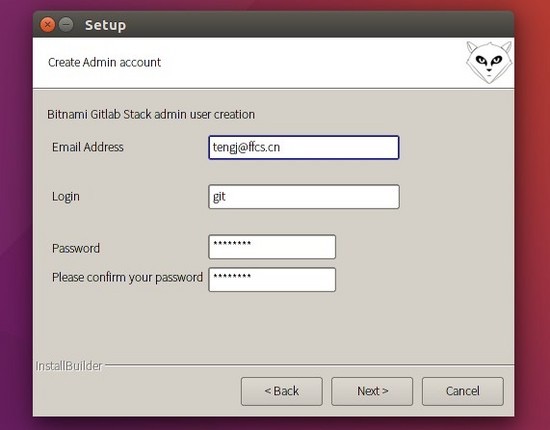
1. 但这儿里推荐 bitnami 下载打包安装版本 <https://bitnami.com/stack/gitlab/installer> ,省去很多时间。他们也提供相关 [WIKI](http://wiki.bitnami.com/Applications/BitNami_GitLab)
2. 如果没有权限，先授权账号：

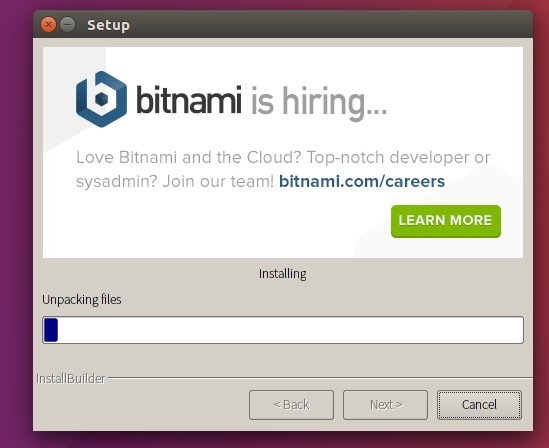
chmod 777 bitnami-gitlab-8.5.7-0-linux-x64-installer.run

1. 然后切换 root 用户执行：
2. ./bitnami-gitlab-8.5.7-0-linux-x64-installer.run
3. 执行后就会弹出图形化界面，一步步确认就行，如下图









### 启动命令

第一次安装好了就默认启动了 **gitlab** ，如果要手动启动的话有两种办法

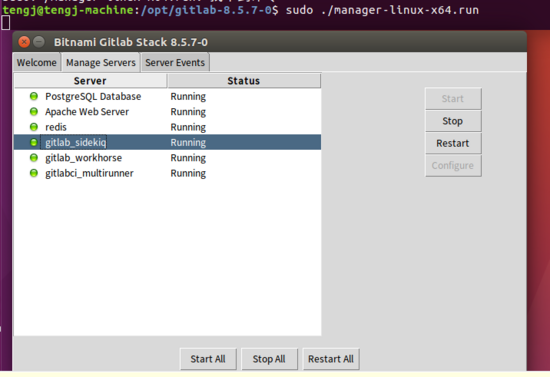
找到安装好的路径 **/opt/gitlab-8.5.7-0**

需要 **root** 权限执行命令：

sudo ./ctlscript.sh **start**

也可以查看`GUI`管理控制台，在`Manage Servers中`选择`**Start** All`启动所有

sudo ./manager-linux-x64.run

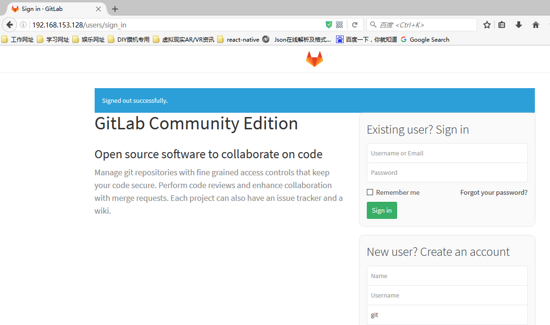


### 登陆系统

启动后，就可以访问系统页面了，这里先通过ifconfig命令获取当前的ip地址



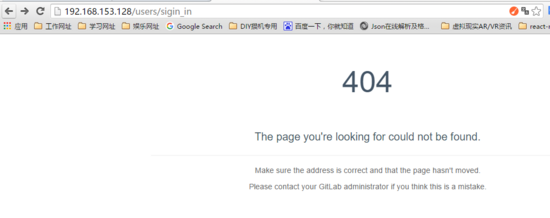
我的是 **192.168.153.128** ，于是就可以在外面 **window** 系统浏览器访问

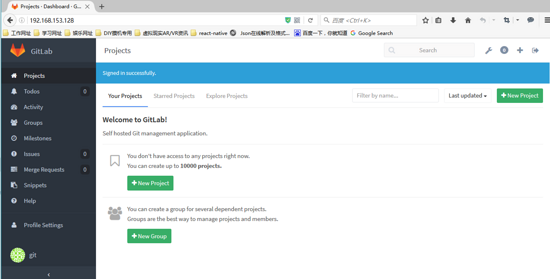


使用我之前配置的 **git** 登陆,我之前安装的时候配置的管理员账号是 **git** ,密码12345678，

这里我登陆后发现跳转到 **404** 页面，把地址 **http://192.168.153.128/users/sign\_in**

后面的 **users/sign\_in** 删掉回车就行,就去后界面如下：





### 创建项目前的设置

如果你是自己一个人玩就直接创建项目即可，因为我这边是给团队使用的，所以需要创建一下用户组，以及添加组成员，配置权限等，下面来介绍这些配置

配置服务器邮件

添加人员或者自己注册的时候需要发邮件激活，所以要先配置服务器，我在 **bitnami** 提供的 [wiki](https://wiki.bitnami.com/Applications/BitNami_GitLab#How_to_configure_the_email_settings_of_GitLab.3f) 里面找到了办法，原文如下：

*How to configure the email settings of GitLab?*

You can configure the SMTP settings during the installation process. If you are using the Virtual Machine or AMI, you can configure it manually. For example, these are the options to configure it using a GMail account:

/opt/bitnami/apps/gitlab/htdocs/config/environments/production.rb

config.action\_mailer.raise\_delivery\_errors = true

config.action\_mailer.delivery\_method = :smtp

config.action\_mailer.perform\_deliveries = true

config.action\_mailer.smtp\_settings = {

:address => "smtp.gmail.com",

:port => 587,

:domain => "gmail.com",

:authentication => :plain,

:user\_name => "your\_account@gmail.com",

:，password => "your\_password",

:enable\_starttls\_auto => true

}

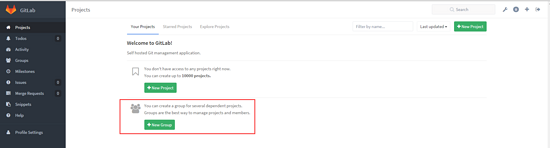
简单的说就是修改 **/opt/bitnami/apps/gitlab/htdocs/config/environments/production.rb** 文件，里面的内容， **address** ， **domain** ， **user\_name** ， **password** 这几个配置成你自己的邮箱，记得你的邮箱必须要支持 **smtp** 。 **port** 端口我默认也是587，不需要修改。

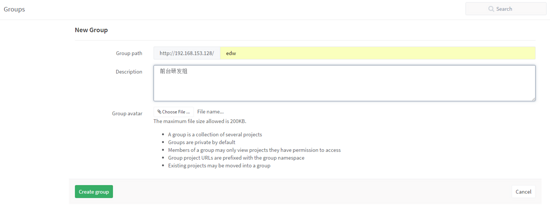
配置好了邮箱，就要测试一下，我们在 **http://192.168.153.128/users/sign\_in** 首页，注册一个额用户，然后我收到邮件激活，然后就可以登陆系统了。

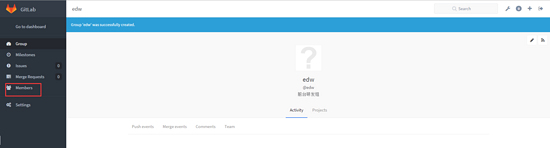
可以看到，非管理员登陆右上角就没有 **Admin Area** 功能

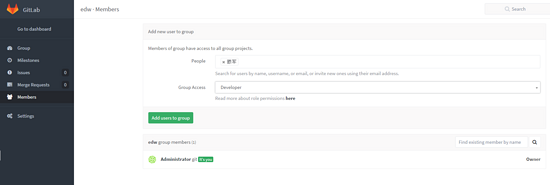
### 创建小组，添加组内成员

让组内成员自己注册账号激活后，我们就准备创建小组了，创建小组是为了方便管理，创建项目的时候直接指定小组，这样小组内的成员都可以看到项目



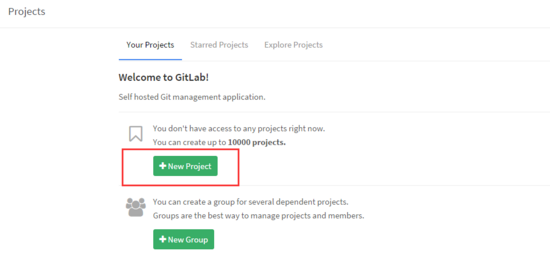


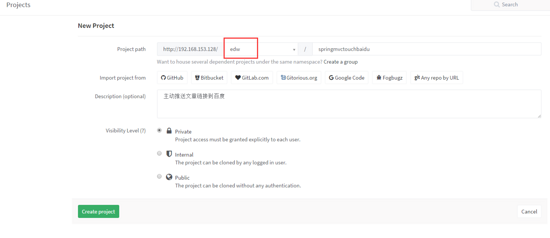


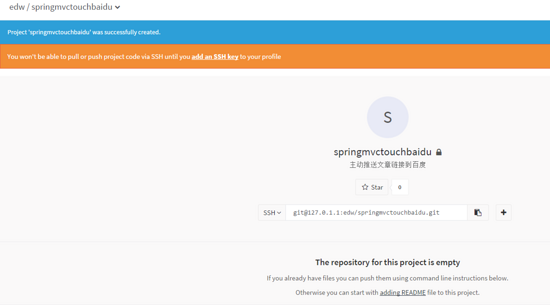


### 新增项目

小组创建好了就可以新增一个项目了,这里创建的时候选择 **edw** 这个创建好的小组，并且项目宣称 **private** 私有项目。这样只有小组内的成员可以下载这个项目

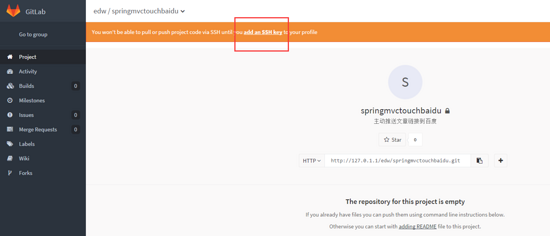




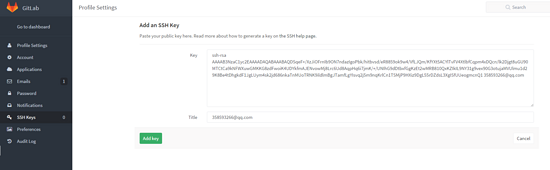


配置**SSH key**

这里切换到 **tengj** 用户登陆，发现已经有了 **springmvctouchbaidu** 项目

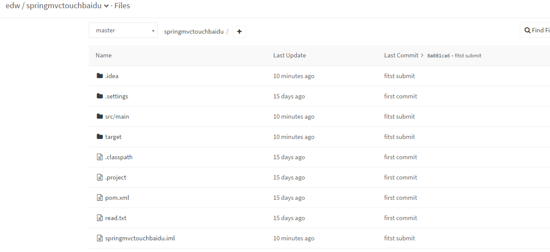


但是要下载到本地还需要配置 **SSH key** ,复制我们本地 **\.ssh\id\_rsa.pub** 的内容黏贴到下面图中指定的即可。



### 远程下载项目

这里模拟 **tengj** 下载远程项目，在这之前我已经用管理员账号把项目传上去了，如图



指定一个目录存放该项目，执行 **git clone** 命令：

$ git clone git@192.168.153.128:edw/springmvctouchbaidu.git

Cloning into 'springmvctouchbaidu'...

The authenticity **of** host '192.168.153.128 (192.168.153.128)' can't be established.

ECDSA key fingerprint is SHA256:1SgfW1DLmaZ9m/E82vVtWjBK1AaP+LMW5U4ng8nv6uE.

Are you sure you want to continue connecting (yes/no)? yes

Warning: Permanently added '192.168.153.128' (ECDSA) to the list **of** known hosts.

remote: 对象计数中: 339, 完成.

remote: 压缩对象中: 100% (242/242), 完成.

remote: Total 339 (delta 68), reused 339 (delta 68)

Receiving objects: 100% (339/339), 12.97 MiB | 14.44 MiB/s, done.

Resolving deltas: 100% (68/68), done.

Checking connectivity... done.

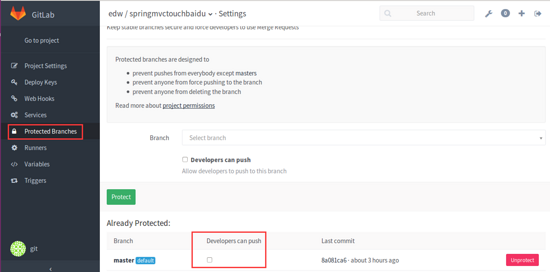
搞定，下载下来了。

### 配置权限

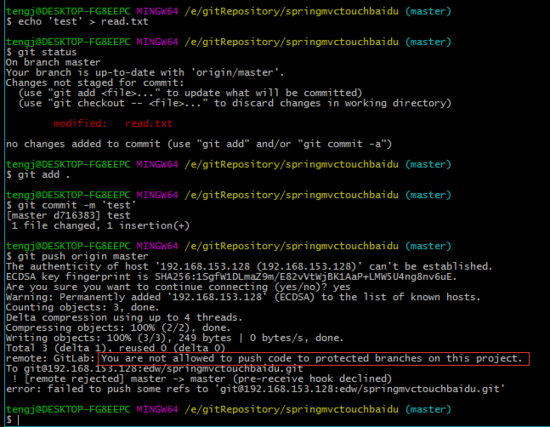
前面文章也说过，一般一个实际项目中 **master** 用来版本发布，一般不在这个分支上开发，所以除了 **git** 管理员用户可以提交代码到 **master** 外，其他组成员只能 **push** 代码到 **dev** 这个分支。

比如我添加 **tengj** 用户的时候选择的是 **Developer** 角色，只要在 **Protected Branches** 中这个勾不够上就行了。这样开发人员就无法 **push** 代码到 **master** 分支上。





下面模拟一下tengj用户提交代码：



总结

ok，花了三天时间终于搞定了，截图还是蛮累人的哈哈，算是比较详细的记录了我这次安装 **gitlab** 的过程，个人感觉比网络上的文章调理清晰多了。一开始查资料快累死了。希望对你有所帮助。

# CocoaPods发布框架到远程公有库

### 编写代码~上传远程仓库

### 创建podSpec

pod spec create 文件名称

//修改podspec文件内容，如下 假设项目名称Test01

s.name = "Test01"

s.version = "1.0.0"

s.summary = "Test01 is a test lib"

s.description = <<-DESC

"这是一个长的描述字数要比s.summary长，Test01 is a test lib"

DESC

s.homepage = "https://github.com/LouKit/Test01"

s.license = "Apache License, Version 2.0"

s.author = { "LK" => "loukit@qq.com" }

s.source = { :git => "https://github.com/LouKit/Test01.git", :tag => "#{s.version}" }

s.source\_files = "Classes", "Classes/\*\*/\*.{h,m}"

### 私有pod的验证

pod spec lint

pod lib lint 和 pod spec lint 命令的区别

pod lib lint是只从本地验证你的pod能否通过验证

pod spec lint是从本地和远程验证你的pod能否通过验证

我一般都是直接使用pod spec lint去验证pod有没有问题

使用pod spec lint去验证私有库能否通过验证时应该，应该要添加--sources选项，不然会出现找不到repo的错误

pod spec lint —sources='私有仓库repo地址,<https://github.com/CocoaPods/Specs'>

### CocoaPods本地私有库使用

创建podSpec文件 //pod spec create 文件名称  
修改podspec文件内容同‘CocoaPods发布框架到远程公有库’，其余:

# 本地库去掉地址即可

s.source = { :git => "", :tag => "#{s.version}" }

验证  
pod lib lint

出现警告

localhost:TestLog LouKit$ pod lib lint

TestLog (0.0.1)

WARN | homepage: The homepage has not been updated from default

WARN | url: There was a problem validating the URL http://EXAMPLE/TestLog.

WARN | license: Unable to find a license file

[!] TestLog did not pass validation, due to 3 warnings (but you can use `--allow-warnings` to ignore them).

[!] The validator for Swift projects uses Swift 3.0 by default, if you are using a different version of swift you can use a `.swift-version` file to set the version for your Pod. For example to use Swift 2.3, run:

`echo "2.3" > .swift-version`.

You can use the `--no-clean` option to inspect any issue.

警告可接受，无视！

创建[测试](http://lib.csdn.net/base/softwaretest)工程，并创建Podfile文件,进行安装本地库  
platform :ios, '9.0'

target 'Example' do

use\_frameworks!

#描述好本地相对路径

pod 'Test01',:path => '../Lib/Test01'

end

安装  
pod install

### CocoaPods远程私有库使用

创建一个专门用于存放spec文件的远程库

将远程库地址加入repo

pod repo add 名字XX 地址

创建本地模板库  
pod lib create spec文件名称 //记得修改podspec文件

创建远程库 用于存放 步骤3 存放的库工程  
git add .

git commit -m 'msg'

#查看是否有关联远程库，没有需要设置下

git remote

//没有设置关联

git remote add origin 地址

如：git remote add origin https://git.coding.net/LouKit/xxx.git

git push origin master

验证spec文件  
本地验证 pod lib lint

远程提交 必须远程验证 pod spec lint (刚刚步骤4提交时没有tag，所以这个步远程验证肯定过不了，需要提交tag)

提交tag:

git tag 0.1.0

git push --tags

//验证

pod spec lint

将spec 推送到步骤1的那个私有库  
pod repo push 名字XX spec文件名称.podspec //这里名字XX 必须对应步骤2 那个名字xx

修改宿主工程podfile  
source 'https://git.coding.net/LouKit/xxx.git'

source 'https://github.com/CocoaPods/Specs.git'

platform :ios, '9.0'

target 'Example' do

use\_frameworks!

pod 'spec文件名称'

end

安装  
pod install

### 维护阶段（扩充代码，版本号升级）

需重新提交代码 修改spec描述文件，大概步骤如下：

1. 重新提交代码

git add .

git commit -m 'msg'

git push origin master

2. 打tag 同时改spec描述文件

git tag 0.2.0

git push --tags

3. 提交pec文件

4. 更新spec文件到索引库

pod repo push 名字XX spec文件名称.podspec

问题 2. 依赖关系解决(比如制作库需要依赖AFN)

//直接spec文件描述即可

s.dependency 'AFNetworking', '~> 3.1.0'

问题 3. 依赖关系子库解决

(如一个库里有三个子库，现在只需用到其中一个，不要全部用到 ，即把一个库分成几个小库)

# 代表分离子库

#s.source\_files = 'xx/Classes/\*\*/\*' 这种写法是匹配所有的

s.subspec 'Category' do |c|

c.source\_files = 'xx/Classes/Category/\*\*/\*'

end

s.subspec 'Network' do |n|

n.source\_files = 'xx/Classes/Network/\*\*/\*'

n.dependency 'AFNetworking', '~> 3.1.0'

end

s.subspec 'Tool' do |t|

t.source\_files = 'xx/Classes/Tool/\*\*/\*'

end

然后使用这个子库过程podefile 写法

target 'Example' do

use\_frameworks!

pod 'xx/Category'

pod 'xx/Network'

end

到此 linux、git、cocoapods helloworld版本全部搭建完成。