Gitlab 安装和使用

Name：曲中岭

Email:zlingqu@126.com
Q Q :441869115

第一章 部署准备

1.1 目的

安装 gitlab-ce 社区版，用于版本管理。

1.2 规划

OS ：CentOS_6.9 x64
IP ：172.16.6.20
gilab-ce ：11.8.0

1.3 简介

GitLab 由以下服务构成:

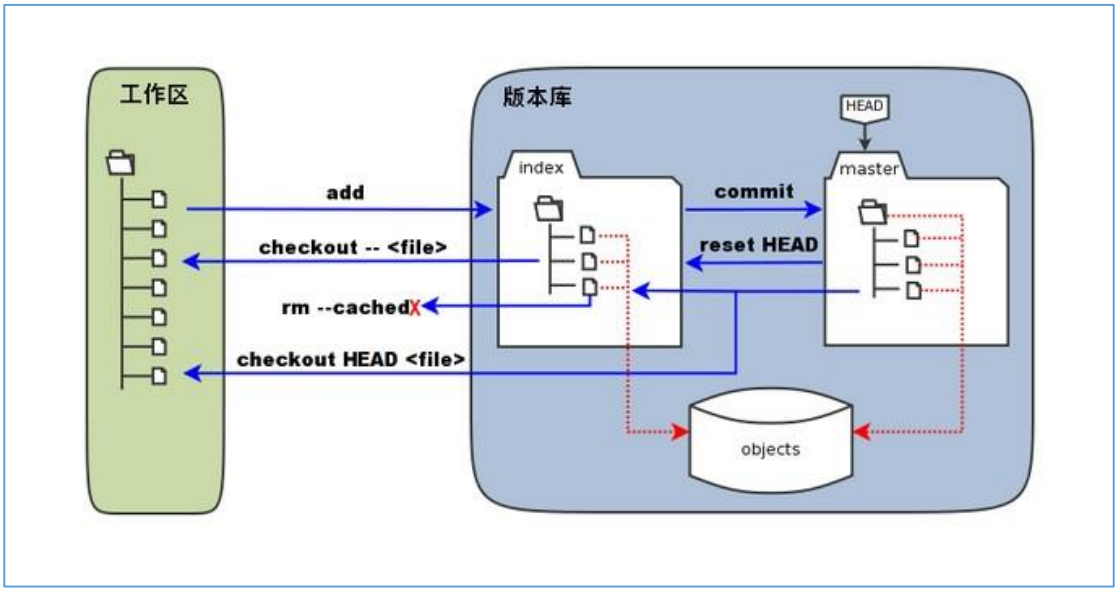
- nginx: 静态 web 服务器
- gitlab-shell: 用于处理 Git 命令和修改 authorized keys 列表
- gitlab-workhorse: 轻量级的反向代理服务器
- logrotate：日志文件管理工具
- postgresql：数据库
- redis：缓存数据库
- sidekiq：用于在后台执行队列任务（异步执行）
- unicorn：An HTTP server for Rack applications，GitLab Rails 应用是托管在这个服务器上面的。

1.4 系统包含

主配置文件: /etc/gitlab/gitlab.rb
GitLab 文档根目录: /opt/gitlab
默认存储库位置: /var/opt/gitlab/git-data/repositories
GitLab Nginx 配置文件路径: /var/opt/gitlab/nginx/conf/gitlab-http.conf
Postgresql 数据目录: /var/opt/gitlab/postgresql/data

1.5 git 架构

工作区：就是你在电脑里能看到的目录。
暂存区：英文叫 stage，或 index。一般存放在 ".git 目录下" 下的 index 文件（.git/index）中，所以我们把暂存区有时也叫作索引（index）。
版本库：工作区有一个隐藏目录.git，这个不算工作区，而是 Git 的版本库。



图中左侧为工作区，右侧为版本库。在版本库中标记为 "index" 的区域是暂存区（stage，index），标记为 "master" 的是 master 分支所代表的目录树。

图中我们可以看出此时 "HEAD" 实际是指向 master 分支的一个"游标"。所以图示的命令中出现 HEAD 的地方可以用 master 来替换。

图中的 objects 标识的区域为 Git 的对象库，实际位于 ".git/objects" 目录下，里面包含了创建的各种对象及内容。

当对工作区修改（或新增）的文件执行 "git add" 命令时，暂存区的目录树被更新，同时工作区修改（或新增）的文件内容被写入到对象库中的一个新的对象中，而该对象的 ID 被记录在暂存区的文件索引中。

当执行提交操作（git commit）时，暂存区的目录树写到版本库（对象库）中，master 分支会做相应的更新。即 master 指向的目录树就是提交时暂存区的目录树。

第二章 gitlab 安装和使用

2.1 添加源

添加 gitlab 源，我这里使用清华大学的源：

```
cat >> /etc/yum.repos.d/gitlab-ce.repo << EOF
[gitlab-ce]
name=Gitlab CE Repository
baseurl=https://mirrors.tuna.tsinghua.edu.cn/gitlab-ce/yum/el$releasever/
gpgcheck=0
enabled=1
EOF
```

2.2 安装

安装依赖

```
yum install cronie openssl-clients
```

安装 gitlab-ce

```
yum install gitlab-ce
```

自动解决其他依赖

安装后程序的主目录在：/opt/gitlab

2.3 配置域名

配置访问的域名，这个用于会初始化到 nginx 配置中。

```
[root@localhost gitlab]# grep ^external_url /etc/gitlab/gitlab.rb
external_url 'http://gitlab.lx2.com'
```

2.4 加载配置

```
gitlab-ctl reconfigure
```

加载配置，见下图

```
[root@localhost gitlab]#
[root@localhost gitlab]# gitlab-ctl reconfigure
Starting Chef Client, version 13.6.4
resolving cookbooks for run list: ["gitlab"]
Synchronizing Cookbooks:
  - gitlab (0.0.1)
  - package (0.1.0)
  - postgresql (0.1.0)
  - redis (0.1.0)
  - mattermost (0.1.0)
  - registry (0.1.0)
  - consul (0.1.0)
  - gitaly (0.1.0)
  - letsencrypt (0.1.0)
  - nginx (0.1.0)
  - runit (4.3.0)
  - acme (3.1.0)
  - crond (0.1.0)
  - compat_resource (12.19.1)
Installing Cookbook Gems:
```

完成后会有如下提示

```
  * service[postgres-exporter] action nothing (skipped due to action :nothing)
Recipe: gitlab::postgres-exporter
  * runit_service[postgres-exporter] action enable
    * ruby_block[restart_service] action nothing (skipped due to action :nothing)
    * ruby_block[restart_log_service] action nothing (skipped due to action :nothing)
    * ruby_block[reload_log_service] action nothing (skipped due to action :nothing)
    * directory[/opt/gitlab/sv/postgres-exporter] action create (up to date)
    * template[/opt/gitlab/sv/postgres-exporter/run] action create (up to date)
    * directory[/opt/gitlab/sv/postgres-exporter/log] action create (up to date)
    * directory[/opt/gitlab/sv/postgres-exporter/log/main] action create (up to date)
    * template[/opt/gitlab/sv/postgres-exporter/log/run] action create (up to date)
    * template[/var/log/gitlab/postgres-exporter/config] action create (up to date)
    * directory[/opt/gitlab/sv/postgres-exporter/env] action create (up to date)
    * ruby_block[Delete unmanaged env files for postgres-exporter service] action run (skipped due to only_if)
    * template[/opt/gitlab/sv/postgres-exporter/check] action create (skipped due to only_if)
    * template[/opt/gitlab/sv/postgres-exporter/finish] action create (skipped due to only_if)
    * directory[/opt/gitlab/sv/postgres-exporter/control] action create (up to date)
    * link[/opt/gitlab/init/postgres-exporter] action create (up to date)
    * file[/opt/gitlab/sv/postgres-exporter/down] action delete (up to date)
    * directory[/opt/gitlab/service] action create (up to date)
    * link[/opt/gitlab/service/postgres-exporter] action create (up to date)
    * ruby_block[wait for postgres-exporter service socket] action run (skipped due to not_if)
     (up to date)
  * template[/var/opt/gitlab/postgres-exporter/queries.yaml] action create (up to date)
Recipe: gitlab::deprecate-skip-auto-migrations
  * file[/etc/gitlab/skip-auto-reconfigure] action create (skipped due to only_if)
  * ruby_block[skip-auto-migrations deprecation] action run (skipped due to only_if)

Running handlers:
Running handlers complete
Chef Client finished, 4/627 resources updated in 01 minutes 05 seconds
gitlab Reconfigured!
[root@localhost gitlab]#
```

后续如果修改了/etc/gitlab 中的配置文件，也可以执行这个命令，使配置生效

2.5 汉化

2.5.1 方法 1

安装 gitlab-ce

查看安装的 gitlab-ce 版本

```
cat /opt/gitlab/embedded/service/gitlab-rails/VERSION
```

比如我的版本是 11.7.5

下载汉化包，注意版本要与 gitlab-ce 一致，比如我这里使用最新的 11.8.0 版本

```
wget https://gitlab.com/xhang/gitlab/repository/11-7-stable-zh/archive.tar.bz2 -O gitlab-11-7-stable-zh.tar.bz2
```

解压

```
tar xf  gitlab-11-7-stable-zh.tar.bz2
```

停止 gitlab-ce

```
gitlab-ctl stop
```

备份

```
cp -r /opt/gitlab/embedded/service/gitlab-rails{,.ori}
```

拷贝和覆盖

```
cp –rf gitlab-11-7-stable-zh/* /opt/gitlab/embedded/service/gitlab-rails
```

重新启动

```
gitlab-ctl start
```

2.5.2 方法 2

安装好 gitlab-ce

查看安装的 gitlab-ce 版本

```
cat /opt/gitlab/embedded/service/gitlab-rails/VERSION
```

比如我的版本是 11.7.5，就 git 下来对应的中文补丁包

```
git clone https://gitlab.com/xhang/gitlab.git -b v11.7.5
```

对比不同

```
cd gitlab
git diff origin/11-7-stable origin/11-7-stable-zh > /tmp/11.7.5.diff
```

停止 gitlab-ce

```
gitlab-ctl stop
```

修改不同处

```
cd /tmp
patch -d/opt/gitlab/embedded/service/gitlab-rails -p1 -f < 11.7.diff
```

重新启动

```
gitlab-ctl start
```

2.5.3 观察

汉化后首次登陆:

设置

2.6 邮件通知

2.6.1 配置邮件

[root@localhost gitlab]# egrep -v "^$|^#" /etc/gitlab/gitlab.rb

```
external_url 'http://gitlab.lx2.com'
gitlab_rails['gitlab_email_enabled'] = true
gitlab_rails['gitlab_email_from'] = 'developer@lx2.com'
gitlab_rails['gitlab_email_display_name'] = '管理员'
gitlab_rails['smtp_enable'] = true
gitlab_rails['smtp_address'] = "smtp.mxhichina.com"
gitlab_rails['smtp_port'] = 465
gitlab_rails['smtp_user_name'] = "developer@lx2.com"
gitlab_rails['smtp_password'] = "*****"
gitlab_rails['smtp_domain'] = "lx2.com"
gitlab_rails['smtp_authentication'] = "login"
gitlab_rails['smtp_enable_starttls_auto'] = false
gitlab_rails['smtp_tls'] = true
```

```
[root@localhost gitlab]# egrep -v "^$|^#" /etc/gitlab/gitlab.rb
external_url 'http://gitlab.lx2.com'
gitlab_rails['gitlab_email_enabled'] = true
gitlab_rails['gitlab_email_from'] = 'developer@lx2.com'
gitlab_rails['gitlab_email_display_name'] = '管理员'
gitlab_rails['smtp_enable'] = true
gitlab_rails['smtp_address'] = "smtp.mxhichina.com"
gitlab_rails['smtp_port'] = 465
gitlab_rails['smtp_user_name'] = "developer@lx2.com"
gitlab_rails['smtp_password'] = "[redacted]"
gitlab_rails['smtp_domain'] = "lx2.com"
gitlab_rails['smtp_authentication'] = "login"
gitlab_rails['smtp_enable_starttls_auto'] = false
gitlab_rails['smtp_tls'] = true
[root@localhost gitlab]#
```

分两部分：

1） 邮件发送部分

如下图，在 53 行附近，可直接修改原有内容，也可以添加

```
51
52 ### Email Settings
53 gitlab_rails['gitlab_email_enabled'] = true
54 gitlab_rails['gitlab_email_from'] = 'developer@lx2.com'
55 gitlab_rails['gitlab_email_display_name'] = '管理员'
56 # gitlab_rails['gitlab_email_reply_to'] = 'noreply@example.com'
57 # gitlab_rails['gitlab_email_subject_suffix'] = ''
58
59 ### GitLab user privileges
60 # gitlab_rails['gitlab_default_can_create_group'] = true
61 # gitlab_rails['gitlab_username_changing_enabled'] = true
```

gitlab_rails['gitlab_email_reply_to']： 邮件抄送，我这里没有配置

gitlab_rails['gitlab_email_display_name']：显示的发件人

2） smtp 部分

如下图，在 509 行附近，可以修改原有内容，也可以直接添加

```
506 ###! Docs: https://docs.gitlab.com/omnibus/settings/smtp.html
507 ###! **Use smtp instead of sendmail/postfix.**
508
509 gitlab_rails['smtp_enable'] = true
510 gitlab_rails['smtp_address'] = "smtp.mxhichina.com"
511 gitlab_rails['smtp_port'] = 465
512 gitlab_rails['smtp_user_name'] = "developer@lx2.com"
513 gitlab_rails['smtp_password'] = "[redacted]"
514 gitlab_rails['smtp_domain'] = "lx2.com"
515 gitlab_rails['smtp_authentication'] = "login"
516 gitlab_rails['smtp_enable_starttls_auto'] = false
517 gitlab_rails['smtp_tls'] = true
518
519 ###! **Can be: 'none', 'peer', 'client_once', 'fail_if_no_peer_cert'**
520 ###! Docs: http://api.rubyonrails.org/classes/ActionMailer/Base.html
521 # gitlab_rails['smtp_openssl_verify_mode'] = 'none'
522
523 # gitlab_rails['smtp_ca_path'] = "/etc/ssl/certs"
```

图中是阿里企业邮的配置，其他邮箱配置可参考官方文档

https://docs.gitlab.com.cn/omnibus/settings/smtp.html

配置好后使用如下命令使配置生效。

```
gitlab-ctl reconfigure
```

2.6.2 测试

如下测试结果，表示配置可用，其中我们可以看到默认抄送人是 noreply@gitlab.lx2.com

```
gitlab-rails console
irb(main):007:0> Notify.test_email('quzhongling@lx2.com','ttt','aeidafd').deliver_now
```

```
irb(main):006:0>
irb(main):007:0> Notify.test_email('quzhongling@lx2.com','ttt','aeidafd').deliver_now
Notify#test_email: processed outbound mail in 1.3ms
Sent mail to quzhongling@lx2.com (695.3ms)
Date: Fri, 01 Mar 2019 16:40:02 +0800
From: =?UTF-8?B?566h55CG5ZGY?= <developer@lx2.com>
Reply-To: =?UTF-8?B?566h55CG5ZGY?= <noreply@gitlab.lx2.com>
To: quzhongling@lx2.com
Message-ID: <5c78efe265f50_58dd3fbb57ddc6cc75170@localhost.localdomain.mail>
Subject: ttt
Mime-Version: 1.0
Content-Type: text/html;
 charset=UTF-8
Content-Transfer-Encoding: 7bit
Auto-Submitted: auto-generated
X-Auto-Response-Suppress: All

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN" "http://www.w3.org/TR/REC-html40/loose.dtd">
<html><body><p>aeidafd</p></body></html>

=> #<Mail::Message:70073498403380, Multipart: false, Headers: <Date: Fri, 01 Mar 2019 16:40:02 +0800>, <From: "管理员" <developer@lx2.com>>, <Reply-To: "管理员" <noreply@gitlab.lx2.com>>, <To
: quzhongling@lx2.com>, <Message-ID: <5c78efe265f50_58dd3fbb57ddc6cc75170@localhost.localdomain.mail>>, <Subject: ttt>, <Mime-Version: 1.0>, <Content-Type: text/html; charset=UTF-8>, <Content
-Transfer-Encoding: 7bit>, <Auto-Submitted: auto-generated>, <X-Auto-Response-Suppress: All>>
irb(main):008:0>
irb(main):009:0>
```

配置好邮箱后，会有一些通知什么的会自动通知，比如

第三章 gitlab 使用

3.1 git 常用命令

3.1.1 cofig

配置用户名和邮箱

```
git config --global user.name "曲中岭"
git config --global user.email quzhongling@lx2.com
```

配置提交方式

```
git config --global push.default simple
```

查看配置

```
git config --list
```

```
[root@tidb6 test-1]# git config --list
user.name=曲中岭
user.email=quzhongling@lx2.com
push.default=simple
core.repositoryformatversion=0
core.filemode=true
core.bare=false
core.logallrefupdates=true
remote.origin.url=git@gitlab.lx2.com:quzl/test-1.git
remote.origin.fetch=+refs/heads/*:refs/remotes/origin/*
branch.master.remote=origin
branch.master.merge=refs/heads/master
[root@tidb6 test-1]#
```

查看配置

```
cat ~/.gitconfig
```

```
[root@tidb6 ~]# cat ~/.gitconfig
[user]
        name = 曲中岭
        email = quzhongling@lx2.com
[push]
        default = simple
[root@tidb6 ~]#
```

3.1.2 init/clone

创建目录并初始 git 仓库

```
mkdir test
cd test
git init
```

指定目录初始化 git 仓库

```
git init test2
```

版本克隆，自动创建一个和仓库同名的目录作为 git 仓库，-b 指定分支，默认是 master
使用 ssh 协议：

```
git clone git@gitlab.lx2.com:quzl/test-1.git
git clone git@gitlab.lx2.com:quzl/test-1
git clone git@gitlab.lx2.com:quzl/test-1.git -b bate3
```

使用 http 协议，配置的证书无效，需要输入登陆的账户密码

```
git clone http://quzl@gitlab.lx2.com/quzl/test-1.git
```

```
[root@es-3 abc]#
[root@es-3 abc]# git clone http://quzl@gitlab.lx2.com/quzl/test-1.git
正克隆到 'test-1'...
Password for 'http://quzl@gitlab.lx2.com':
remote: Enumerating objects: 156, done.
remote: Counting objects: 100% (156/156), done.
remote: Compressing objects: 100% (64/64), done.
remote: Total 156 (delta 67), reused 137 (delta 54)
接收对象中: 100% (156/156), 13.24 KiB | 0 bytes/s, done.
处理 delta 中: 100% (67/67), done.
[root@es-3 abc]#
[root@es-3 abc]# ls
test-1
[root@es-3 abc]#
```

使用 git 协议，并不是都支持，验证发现 gitlab 不支持

```
git clone git://gitlab.lx2.com/quzl/test-1.git
git clone git://gitlab.lx2.com/quzl/test-1
```

克隆到指定目录

```
git clone <repo> <directory>
```

3.1.3 add/commit

提交内容到暂存区

```
git add .
git add a.txt
#如果和缓冲区相比，工作区删除了内容，需要加-A 或者--all
git add . –A
git add . --all
```

提交修改到本地库

```
git commit -m "add file "
将 add、commit 合成一步执行，只对修改有效
git commit -am "add file "
```

3.1.4 reset

reset 使暂存区的目录树被重写，暂存区被 master 分支指向的目录树所替换，但是工作区不受影响。比如使用了 add 更新了暂存区，可以用这个命令达到取消 add 操作的目的。

```
git reset HEAD
git reset master
git reset HEAD a.txt
```

3.1.5 rm

从暂存区删除文件，或者清空暂存区，工作区则不做出改变。-r 递归删除。也可用于取消 add 操作。

```
git rm --cached a.txt
git rm --cached –r .
```

3.1.6 checkout

用暂存区全部或指定的文件替换工作区的文件，危险操作，因为未 add 的动作将被覆盖

```
git checkout .
git checkout -- <file>
```

用 HEAD 指向的 master 分支中的全部或者部分文件替换暂存区和以及工作区中的文件，危险操作，不会删除新增文件，因为未 add、未 commit 的动作将会被覆盖

```
git checkout HEAD .
git checkout HEAD <file>
```

3.1.7 revert

版本回滚，通过日志查看到 commit-id，回滚，提交到远程

```
git log
git revert a3a1c93792ebd82695dd2fb0b4e4de2415dbebdd
git push
```

```
222[root@tidb6 test-1]# git log
commit a3a1c93792ebd82695dd2fb0b4e4de2415dbebdd
Author: 曲中岭 <quzhongling@lx2.com>
Date:   Thu Mar 7 16:46:00 2019 +0800

    更新 g.txt

commit bdf5737d7a9613bac0d1b4fa28c8be528b281708
Author: 曲中岭 <quzhongling@lx2.com>
Date:   Thu Mar 7 16:35:20 2019 +0800

    更新 g.txt
```

```
[root@tidb6 test-1]#
[root@tidb6 test-1]# git revert a3a1c93792ebd82695dd2fb0b4e4de2415dbebdd
[bate4 8a2679b] Revert "更新 g.txt"
 1 file changed, 1 insertion(+), 2 deletions(-)
[root@tidb6 test-1]#
[root@tidb6 test-1]#
[root@tidb6 test-1]# cat g.txt
ddd
hello
aaa
111[root@tidb6 test-1]# git push
Counting objects: 5, done.
Delta compression using up to 2 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 317 bytes | 0 bytes/s, done.
Total 3 (delta 1), reused 0 (delta 0)
remote:
remote: To create a merge request for bate4, visit:
remote:   http://gitlab.lx2.com/quzl/test-1/merge_requests/new?merge_request%5Bsource_branch%5D=bate4
remote:
To git@gitlab.lx2.com:quzl/test-1.git
   a3a1c93..8a2679b  bate4 -> bate4
[root@tidb6 test-1]#
```

3.1.8 status/diff

查看当前状态，-s 简略输出

```
git status
git status -s
```

比较的是工作区和暂存区的差别，--stat 显示摘要，下同

```
git diff
git diff --stat
```

比较的是暂存区和版本库的差别

```
git diff --cached
git diff --cached  HEAD
git diff --cached  master
```

可以查看工作区和版本库的差别

```
git diff HEAD
git diff bate1
```

3.1.9 rm/mv

删除工作区和缓存区，使用这个命令后，可以直接 commit，不用 add 了

```
git rm a.txt
```

只删除缓存区

```
git rm --cached a.txt
```

强制删除，当工作区被修改或者工作区和暂存区都被修改时使用-f 强制删除

```
git rm -f a.txt
```

删除目录 ，-r 递归删除

```
git rm -r dir1
```

移动或重命名一个文件、目录、软连接。同时修改工作区和暂存区

```
git mv b.txt b.txt.bak
```

3.1.10 branch

查看本地分支

```
git branch
```

查看远程分支

```
git branch -r
```

查看所有分支，包括本地和远程

```
git branch -a
```

创建分支

```
git branch bate1
```

切换分支

```
git checkout bate1
```

切换分支，当提交当前分支工作，默认不允许切换，可使用如下方法切换

创建+切换分支

```
git checkout -b bate1
# -b 参数表示创建并切换，相当于前面两条命令
```

删除分支

```
git branch –d bate1
```

删除远程分支

```
git push --delete origin bate2
git push origin :bate2
```

3.1.11 fetch

fetch 所有分支到本地，但不会 merge

```
git fetch
```

```
[root@tidb6 test-1]# git fetch
remote: Enumerating objects: 10, done.
remote: Counting objects: 100% (10/10), done.
remote: Compressing objects: 100% (5/5), done.
remote: Total 6 (delta 2), reused 0 (delta 0)
Unpacking objects: 100% (6/6), done.
来自 gitlab.lx2.com:quzl/test-1
   6a34d82..a56356a  master     -> origin/master
   bdf5737..a3a1c93  bate4      -> origin/bate4
[root@tidb6 test-1]#
```

指定分支进行 fetch

```
git fetch bate4
```

```
[root@tidb6 test-1]# git diff origin/bate4
[root@tidb6 test-1]#
[root@tidb6 test-1]# git fetch
来自 gitlab.lx2.com:quzl/test-1
   0ac35f4..96324c8  bate4      -> origin/bate4
[root@tidb6 test-1]#
[root@tidb6 test-1]# git diff origin/bate4
diff --git a/g.txt b/g.txt
index 04b0532..ed3c51d 100644
--- a/g.txt
+++ b/g.txt
@@ -1,3 +1,2 @@
 ddd
 hello
-aaa
\ No newline at end of file
[root@tidb6 test-1]#
```

fetch 远程的 master 分支到本地的 branch2，如果不存在 branch2 则会自动创建

```
git fetch :branch2
git fetch master:branch2
```

对比差异，解决冲突

```
git diff origin/bate4
git diff branch2
```

git pull = git fetch + git merge
但使用后者更加安全，虽然使用 git pull 可以自动合并并提示有冲突

推送到远程分支

```
git push origin master
git push origin bate1
git push
```

推送到远程仓库的新分支

```
git push --set-upstream origin bate1
```

3.1.12 merge

合并某分支到当前分支

```
git merge master
```

合并分支，在 master 分支上操作，将 bate1 合并到 master

```
git checkout master
git merge bate1
```

冲突解决，比如 bate1、bate2 同时修改了同一个文件的同一行，依次合并到 master 会冲突，需要手动解决冲突

```
[root@tidb6 test-1]#
[root@tidb6 test-1]# git merge bate3
更新 9886d58..a078df5
Fast-forward
 d.txt | 3 +++
 1 file changed, 3 insertions(+)
 create mode 100644 d.txt
[root@tidb6 test-1]#
[root@tidb6 test-1]# git merge bate4
自动合并 d.txt
冲突（添加/添加）：合并冲突于 d.txt
自动合并失败，修正冲突然后提交修正的结果。
[root@tidb6 test-1]#
```

手动解决的方式就是手动编辑 vim 该文件

```
hello
<<<<<<< HEAD
bate3
hello------
=======
hello
bate4
hello------
hello
>>>>>>> bate4
~
~
```

比如我选择以 bate4 为准，就修改为 bate4 内容即可

hello

hello

bate4

hello------

hello

备注：bate3 和 bate4 中的内容

```
[root@tidb6 test-1]#
[root@tidb6 test-1]# git checkout bate3
切换到分支 'bate3'
[root@tidb6 test-1]#
[root@tidb6 test-1]# cat d.txt
hello
bate3
hello------
[root@tidb6 test-1]#
[root@tidb6 test-1]#
[root@tidb6 test-1]# git checkout bate4
切换到分支 'bate4'
[root@tidb6 test-1]#
[root@tidb6 test-1]# cat d.txt
hello
hello
bate4
hello------
hello
[root@tidb6 test-1]#
```

另一个例子：
bate3：e.txt 内容
master
bate3
master

bate4：e.txt 内容
master
bate4
master

冲突内容：

```
master
<<<<<<< HEAD
bate3
=======
bate4
>>>>>>> bate4
master
~
```

如果在尾部追究了内容，冲突如下：

```
hello
master
bate
master
hello
<<<<<<< HEAD
=======
hello
>>>>>>> bate4
~
~
```

另外：
冲突解决后，push 到远程仓库后的注释信息显示，如下，d.txt 我解决时以 bate4 为准，所以显示 bate4 提交时的注释信息，e.txt 我没有选择 bate3、也没选择 bate4，所以显示的是我冲突解决后的 commit 信息

d.txt	add d.txt in bate4
e.txt	e.txt冲突解决

3.1.13 push

语法

```
git push [alias] [branch]
```

把本地库的修改推送到远程服务器

```
git push
git push origin master
```

删除远程分支

```
git push --delete origin bate2
git push origin :bate2
```

3.1.14 log

一般查看，默认按照时间逆序排列

```
git log
```

显示简洁版本

```
git log –oneline
```

显示拓扑图

```
git log --oneline –graph
```

逆序显示

```
git log –reverse
```

显示指定用户提交的，并只显示 5 条

```
git log --author=Linus -5
```

指定日期

```
git log --oneline --before={3.weeks.ago} --after={2010-04-18} --no-merges
```

3.1.15 stash

储藏当前的工作状态，如下图，储藏后，已经跟踪的文件也是不可见了。

```
[root@tidb6 test-1]# git status
# 位于分支 bate4
# 要提交的变更:
#   (使用 "git reset HEAD <file>..." 撤出暂存区)
#
#       新文件:     h.txt
#
[root@tidb6 test-1]#
[root@tidb6 test-1]#
[root@tidb6 test-1]#
[root@tidb6 test-1]# git stash
Saved working directory and index state WIP on bate4: cf9ad62 add g.txt
HEAD 现在位于 cf9ad62 add g.txt
[root@tidb6 test-1]#
[root@tidb6 test-1]# git stash list
stash@{0}: WIP on bate4: cf9ad62 add g.txt
[root@tidb6 test-1]#
[root@tidb6 test-1]# git status
# 位于分支 bate4
无文件要提交，干净的工作区
[root@tidb6 test-1]#
[root@tidb6 test-1]#
[root@tidb6 test-1]#
[root@tidb6 test-1]# ls
a.txt  b.txt  c.txt  Dockerfile  d.txt  e.txt  f.txt  g.txt  README.md
[root@tidb6 test-1]#
```

储藏，把所有未提交（已被跟踪）的修改都保存起来，用于后续恢复当前工作目录。

```
git stash
```

查看有哪些储藏

```
git stash list
```

恢复隐藏，默认是第 0 个，即最新的那个

```
git stash apply
git stash apply stash@{1}
```

删除储藏记录

```
git stash drop stash@{0}
git stash clear
```

3.1.16 remote

查看关联的远程仓库

```
git remote
```

查看关联的远程仓库的详细信息

```
git remote -v
```

添加远程仓库的关联，默认是 origin，说白了就是使用 origin 指代对应的仓库。使用场景更多是：限制 gitlab/github 中创建空的仓库，本地仓库建立关联，至于能否 push，要看权限，能

否 pull，也要看权限。

```
git remote add xxx git@gitlab.lx2.com:quzl/test-1
```

```
[root@es-3 test-1]# git remote add xxx git@gitlab.lx2.com:quzl/test-1
[root@es-3 test-1]#
[root@es-3 test-1]# git remote
origin
xxx
[root@es-3 test-1]# git remote -v
origin  git@gitlab.lx2.com:quzl/test-1.git (fetch)
origin  git@gitlab.lx2.com:quzl/test-1.git (push)
xxx     git@gitlab.lx2.com:quzl/test-1 (fetch)
xxx     git@gitlab.lx2.com:quzl/test-1 (push)
[root@es-3 test-1]#
[root@es-3 test-1]#
```

建立关联后，可使用如下命令，将本地的 master 与远程的 master 建立关联，第一次推送需要加-u，后续可以不加

```
git push -u origin master
```

关联信息可在 .git/config 中看到关联的相关信息

```
[root@es-3 test-1]# cat .git/config
[core]
        repositoryformatversion = 0
        filemode = true
        bare = false
        logallrefupdates = true
[remote "origin"]
        url = git@gitlab.lx2.com:quzl/test-1.git
        fetch = +refs/heads/*:refs/remotes/origin/*
[branch "master"]
        remote = origin
        merge = refs/heads/master
[remote "xxx"]
        url = git@gitlab.lx2.com:quzl/test-1
        fetch = +refs/heads/*:refs/remotes/xxx/*
[root@es-3 test-1]#
```

删除关联

```
git remote remove xxx
git remote rm xxx
```

3.1.17 tag

查看标签

```
git tag
```

打标签

```
git tag v1.0
git tag -a V1.2
git tag -a V1.2 -m 'release 1.2'
```

给某一个 commit 打标签，找到历史 commit-id，再打

```
git log --pretty=oneline --abbrev-commit
git tag v0.9 3acc674
```

查看标签信息，包括历史 commit 信息

```
git show V1.2
```

删除标签

```
git tag -d V1.3
```

推送到远程，包括标签信息，可以只推送某个标签，也可以--tags，推送所有标签

```
git push origin v1.0
git push --tags
```

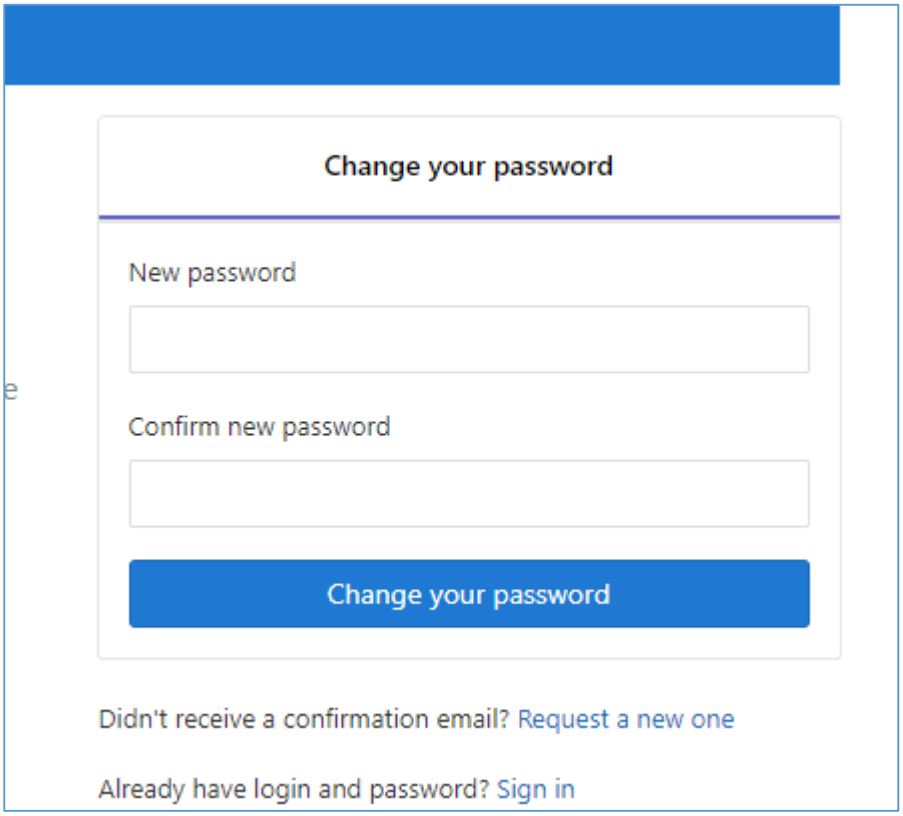
删除远程标签信息

```
git push origin :refs/tags/V1.2
```

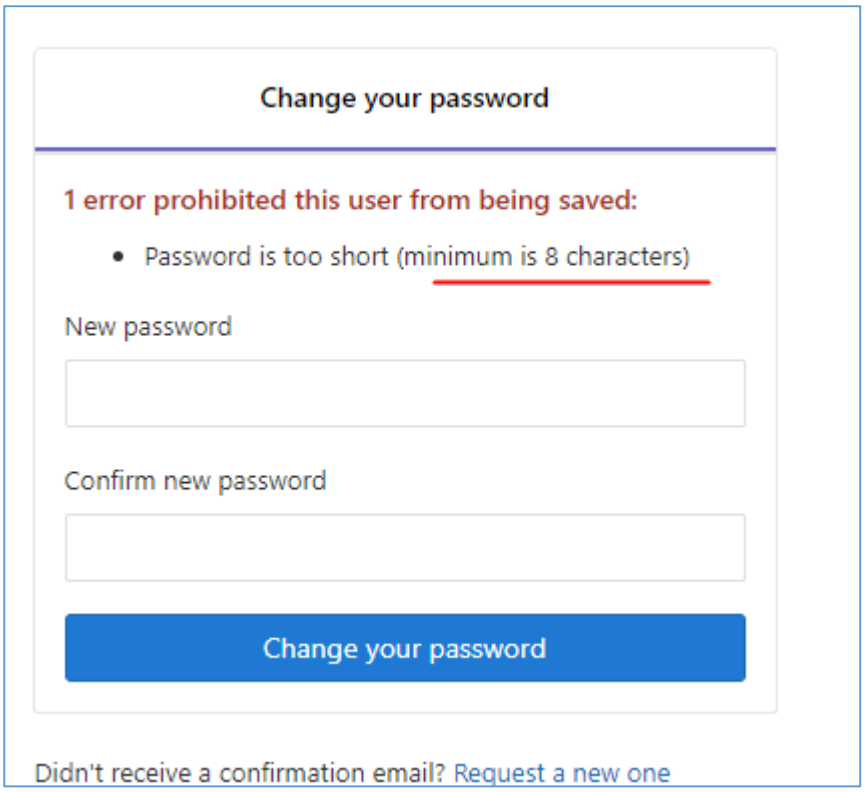
3.2 gitlab 用户管理

3.2.1 用户登陆

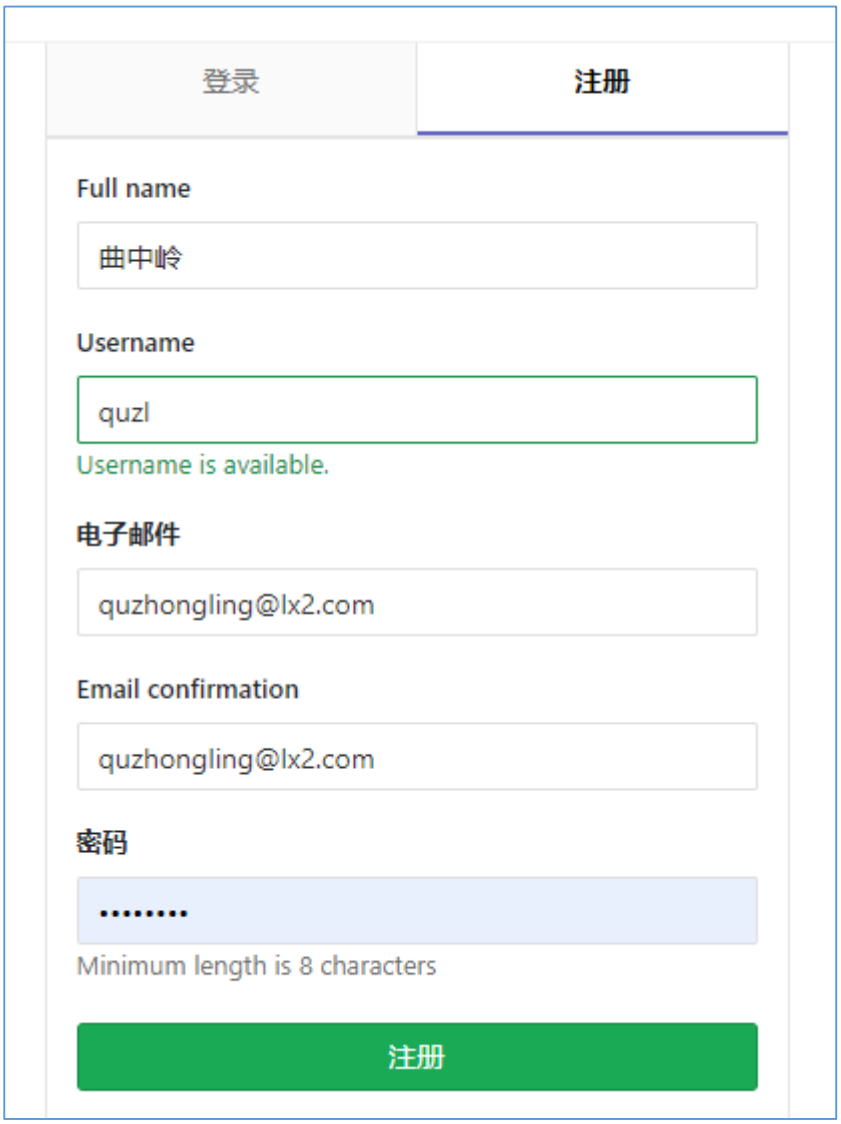
如下图，首次登陆需要修改密码，用户默认使 root



密码长度至少 8 位，否认会有如下报错


Change your password
1 error prohibited this user from being saved:
Password is too short (minimum is 8 characters)
New password
Confirm new password
Change your password
Didn't receive a confirmation email? Request a new one

3.2.2　用户注册



3.2.3 配置公钥

用户注册完成后，在自己的机器上使用 ssh-kegen 等生产密钥对，登陆 web 添加 ssh 密钥

3.2.4 角色权限说明

1）Guest(匿名用户) - 创建项目、写留言薄

2）Reporter（报告人）- 创建项目、写留言薄、拉项目、下载项目、创建代码片段

3）Developer（开发者）- 创建项目、写留言薄、拉项目、下载项目、创建代码片段、创建合并请求、创建新分支、推送不受保护的分支、移除不受保护的分支 、创建标签、编写 wiki

Master（管理者）- 创建项目、写留言薄、拉项目、下载项目、创建代码片段、创建合并请求、创建新分支、推送不受保护的分支、移除不受保护的分支 、创建标签、编写 wiki、增加团队成员、推送受保护的分支、移除受保护的分支、编辑项目、添加部署密钥、配置项目钩子

4）Owner（所有者）- 创建项目、写留言薄、拉项目、下载项目、创建代码片段、创建合并请求、创建新分支、推送不受保护的分支、移除不受保护的分支 、创建标签、编写 wiki、增加团队成员、推送受保护的分支、移除受保护的分支、编辑项目、添加部署密钥、配置项目钩子、开关公有模式、将项目转移到另一个名称空间、删除项目

3.3 gitlab 服务端操作

3.3.1 运维管理相关命名

```
# 查看版本
cat /opt/gitlab/embedded/service/gitlab-rails/VERSION

# 检查 gitlab
gitlab-rake gitlab:check SANITIZE=true --trace

# 实时查看日志
gitlab-ctl tail

# 数据库关系升级
gitlab-rake db:migrate

# 清理 redis 缓存
gitlab-rake cache:clear

# 升级 GitLab-ce 版本
yum update gitlab-ce
```

```
# 升级 PostgreSQL 最新版本
gitlab-ctl pg-upgrade
```

3.3.2 服务管理相关命令

```
# 启动所有 gitlab 组件:
gitlab-ctl start

# 停止所有 gitlab 组件:
gitlab-ctl stop

# 停止所有 gitlab postgresql 组件:
gitlab-ctl stop postgresql

# 停止相关数据连接服务
gitlab-ctl stop unicorn
gitlab-ctl stop sidekiq

# 重启所有 gitlab 组件:
gitlab-ctl restart

# 重启所有 gitlab-workhorse 组件:
gitlab-ctl restart  gitlab-workhorse

# 查看服务状态
gitlab-ctl status

# 生成配置并启动服务
gitlab-ctl reconfigure
```

3.3.3 日志相关命令

```
# 实时查看所有日志
gitlab-ctl tail

# 实时检查 redis 的日志
gitlab-ctl tail redis

# 实时检查 postgresql 的日志
gitlab-ctl tail postgresql

# 检查 gitlab-workhorse 的日志
gitlab-ctl tail gitlab-workhorse
```

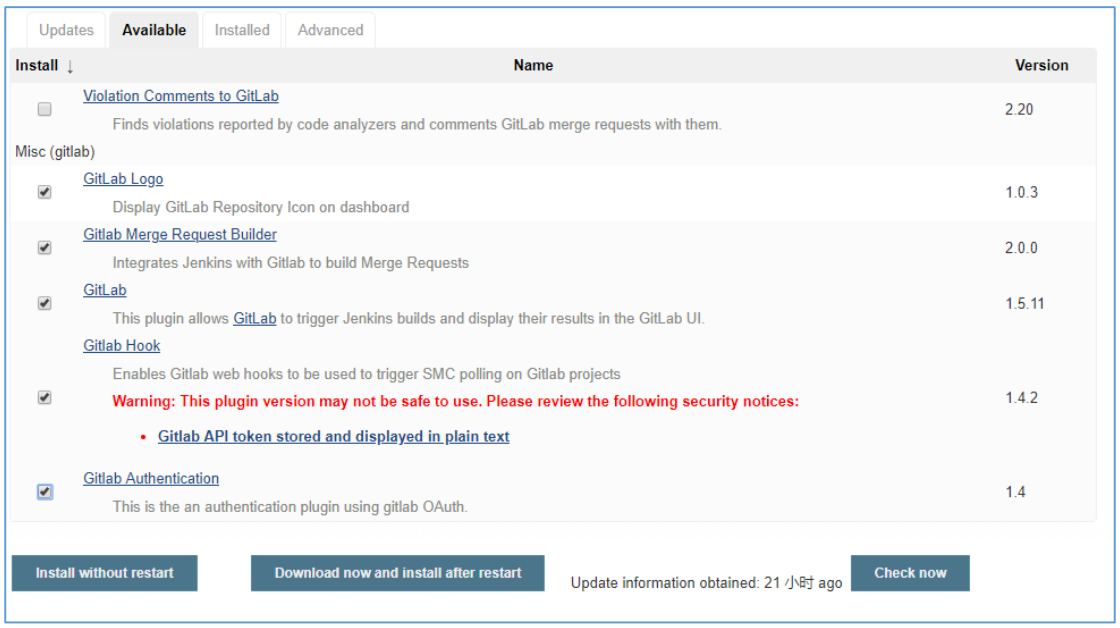
```
# 检查 logrotate 的日志
gitlab-ctl tail logrotate

# 检查 nginx 的日志
gitlab-ctl tail nginx

# 检查 sidekiq 的日志
gitlab-ctl tail sidekiq

# 检查 unicorn 的日志
gitlab-ctl tail unicorn
```

第四章 Jenkins 自动构建

4.1 jenkins 配置

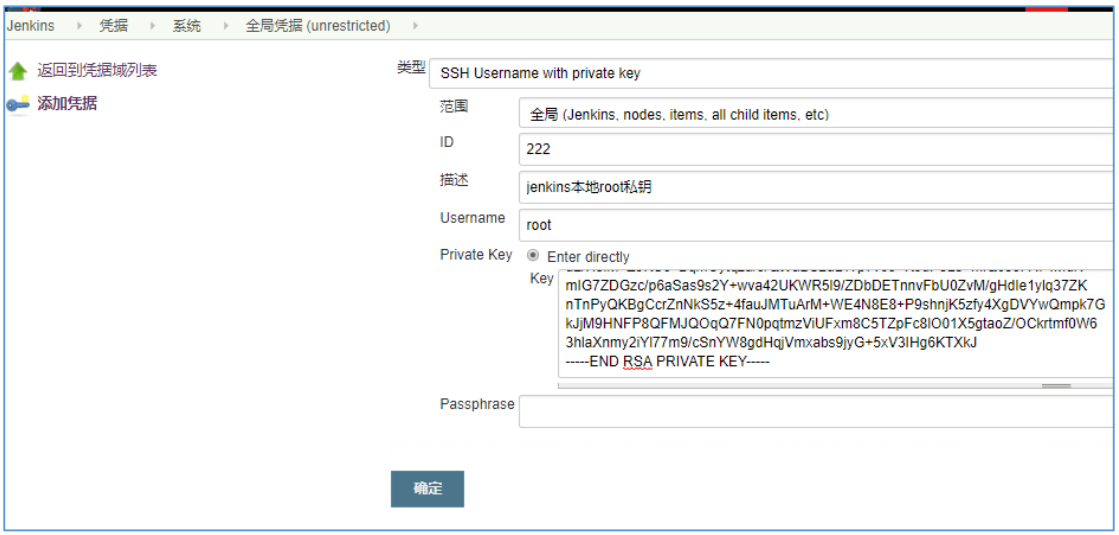
4.1.1 安装插件

如下图，安装 gitlab、gitlab logo 等插件



4.1.2 添加凭证（选做）

Jenkins 从 gitlab 拉取代码等，使用 ssh 协议，需要做信任，jenkins 本机需要生成公私密钥对，公钥加到项目的 key 里面，私钥信息需要添加一个凭证，如下图



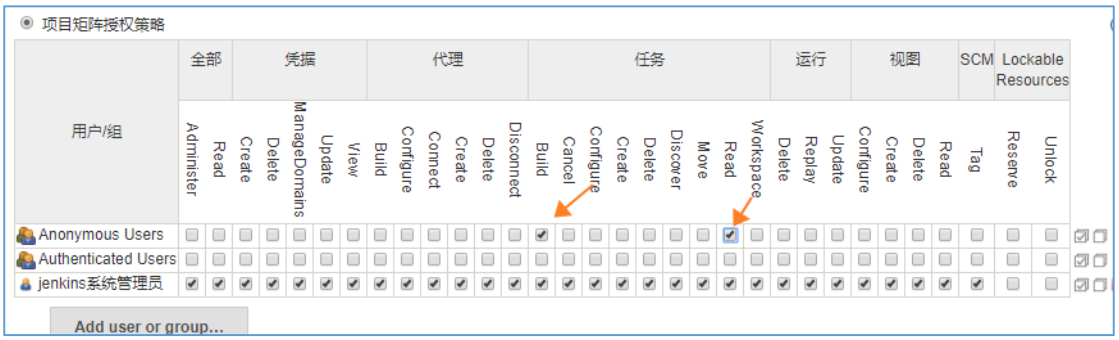
类型：SSH Username with private key
范围：全局
ID：可为空，自动生成
描述：随意
Username：可以不填，默认使用 root 填充。随意填写也 ok，因为我这里要使用的场景是 gitlab，而不是主机。如果是远程主机，需要填写对方的用户名，不一定是 root。
Private key:点击 Enter directly 后，直接输入私钥信息
Passphrase：输入生成私钥时的密码

这一步也可以不操作，即不添加这个凭证，让 job 自动使用当前用户的默认私钥，看个人喜欢。

如果这一步没有做，可以在配置 git 时选择“当前”选项。

4.1.3 匿名用户授权（选做）

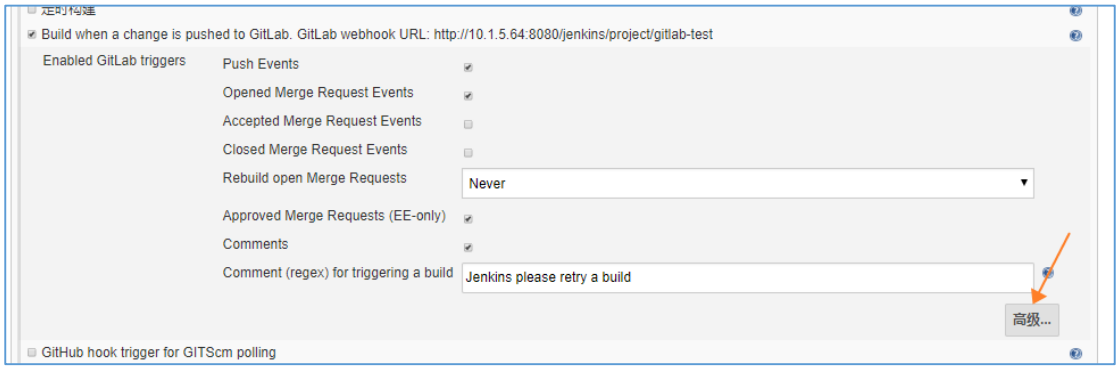
方法 1：设置匿名用户有权限进行构建和读取，如下图

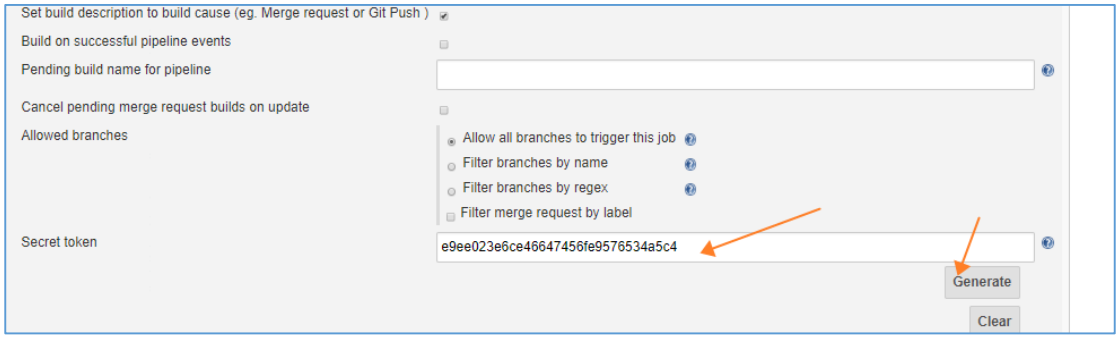


如果不设置，gitlab 的钩子将不生效，报错如下

方法 2：使用项目的 token
为了安全，我们不建议匿名用户具有任何权限

这样在设置触发器时如下图生成 token





并在 gitlab 配置 webhook 时加上此 token

4.2 gitlab 配置

4.2.1 允许本地网络使用 webhook

使用管理员（root）登陆，如下图，放开限制

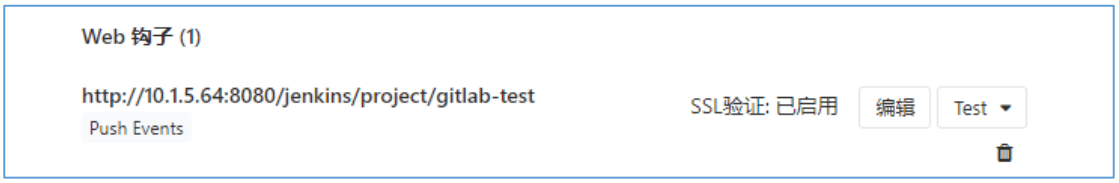
如果不进行这一步，当添加钩子时会报错，如下图所示

4.2.2 添加 webhook

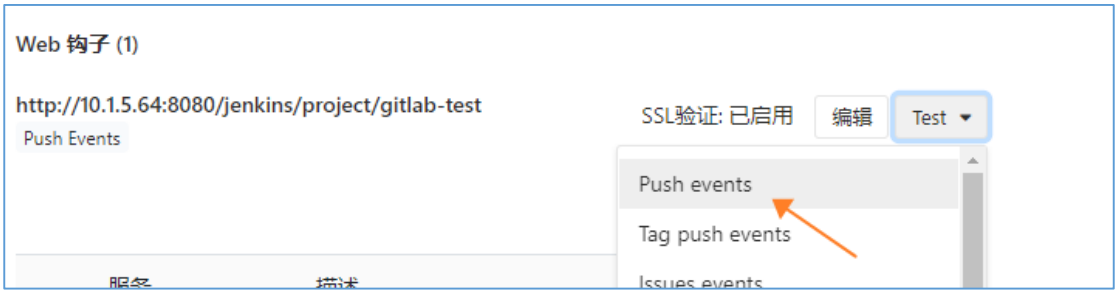
使用 quzl 账户登陆 gitlab，如下图添加钩子，该 url 即是 jenkins 中该项目的回调地址，格式固定，可仿照该格式进行书写

链接：http://10.1.5.64:8080/jenkins/project/gitlab-test，即回调地址
安全令牌：配置触发器时添加的。如果不启用令牌，需要授权匿名用户具有构建的权限。

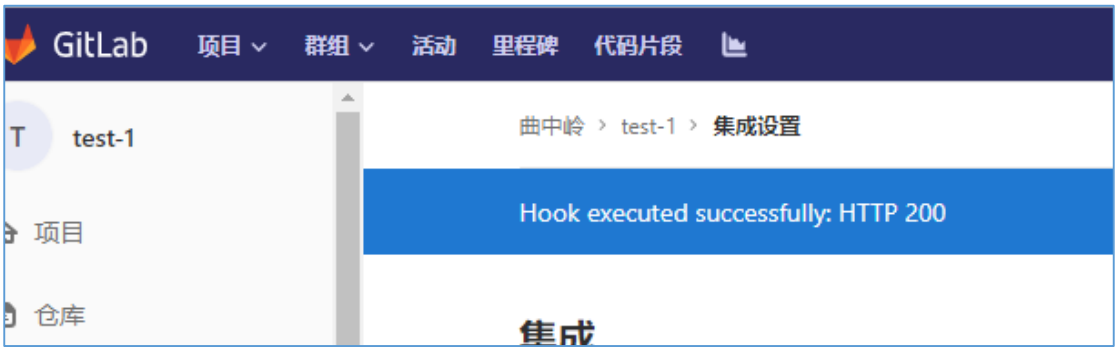
添加完成之后在该页面下方会显示已经添加的钩子



测试下是否可用



返回 200，表示正常使用

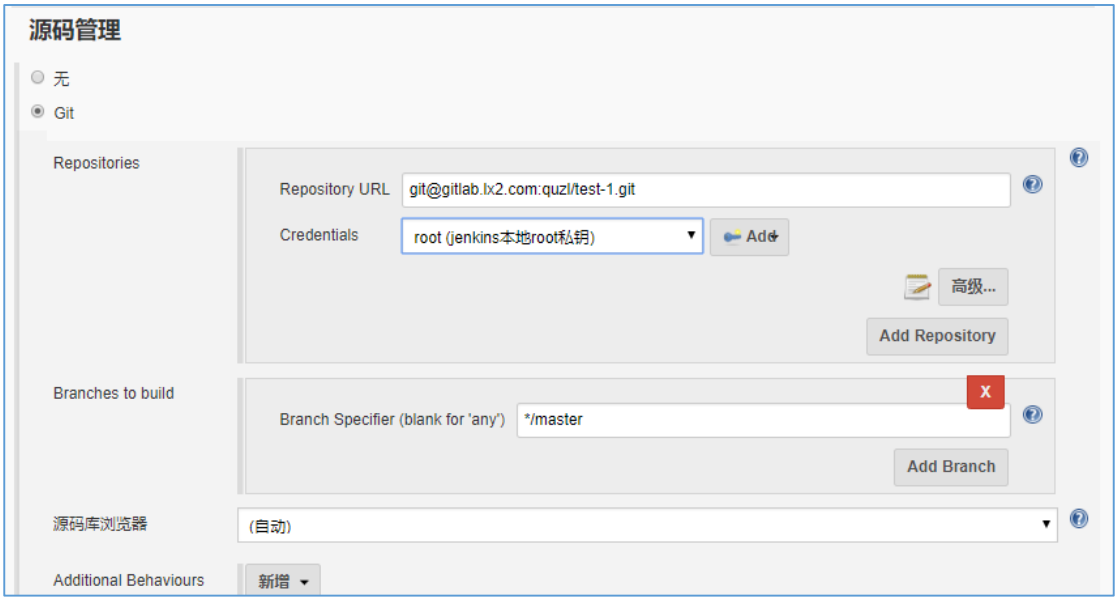


提醒：可以添加多个钩子

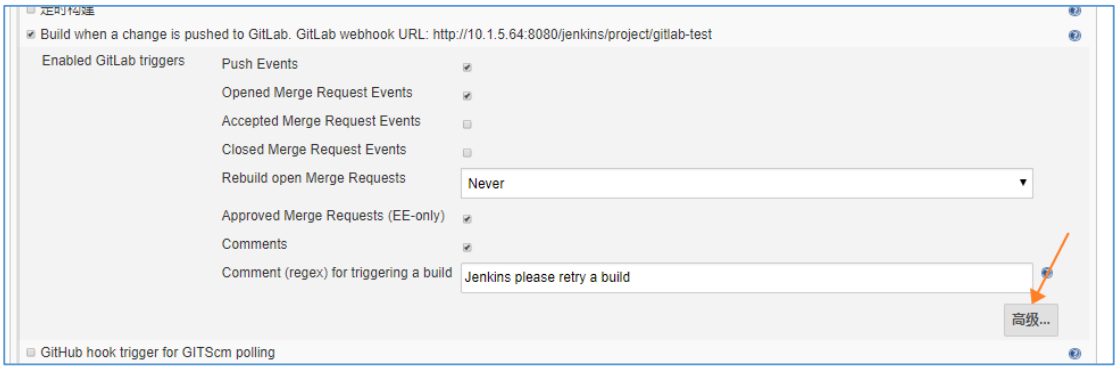
4.3 验证自动构建

4.3.1 jenkins 中 job 配置

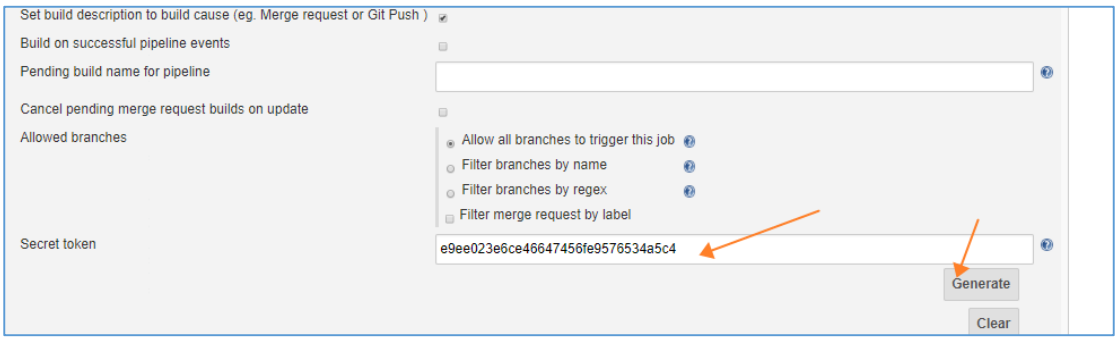
这里只列出关键配置，源码管理使用 git



触发器选择



生成 token，用于 gitlab 上的 webhook 配置

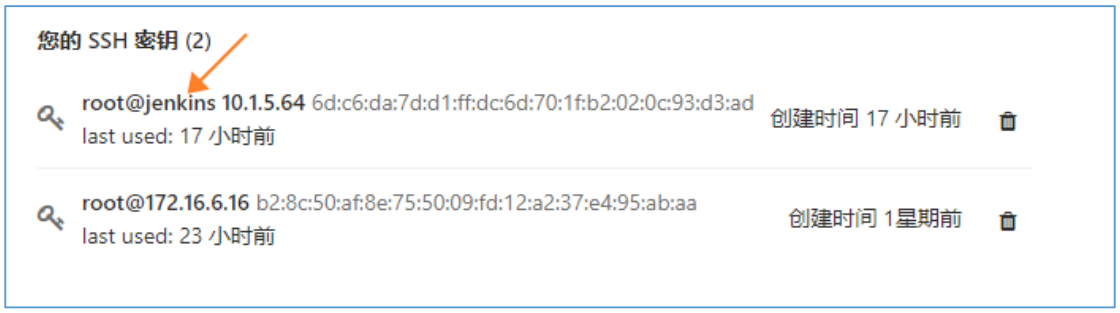


4.3.2 配置钩子

参考其他小节

4.3.3 添加 ssh 公钥证书

参考其他节，即将 jenkins 的公钥配置到对应的 gitlab 用户里面,如下图我新增了一个公钥



4.3.4 提交代码到 gitlab

如下图，更新了 test-1 的代码，并推送到 gitlab

```
[root@es-3 test-1]# vim b.txt
[root@es-3 test-1]#
[root@es-3 test-1]# git commit -am "update b.txt"
[master e2ef25a] update b.txt
 1 file changed, 1 insertion(+)
[root@es-3 test-1]#
[root@es-3 test-1]# git push
warning: push.default 未设置，它的默认值将会在 Git 2.0 由 'matching'
修改为 'simple'。若要不再显示本信息并在其默认值改变后维持当前使用习惯，
进行如下设置：

  git config --global push.default matching

若要不再显示本信息并从现在开始采用新的使用习惯，设置：

  git config --global push.default simple

参见 'git help config' 并查找 'push.default' 以获取更多信息。
（'simple' 模式由 Git 1.7.11 版本引入。如果您有时要使用老版本的 Git，
为保持兼容，请用 'current' 代替 'simple' 模式）

Counting objects: 5, done.
Delta compression using up to 2 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 295 bytes | 0 bytes/s, done.
Total 3 (delta 1), reused 0 (delta 0)
To git@gitlab.lx2.com:quzl/test-1.git
   8c2058c..e2ef25a  master -> master
[root@es-3 test-1]#
```

4.3.5 验证

gitlab 使用钩子触发自动构建，如下图 13、14 即是自动触发的，和 11、12 不同

Build History
构建历史
find
x
#14
2019-3-14 下午4:50
Started by GitLab push by 曲中岭
#13
2019-3-14 下午4:48
Started by GitLab push by 曲中岭
#12
2019-3-14 下午4:40
#11
2019-3-14 下午4:37

第五章 Jenkins 集成 gitlab

5.1 gitlab 配置

5.1.1 生成 token

使用普通用户登陆 gitlab 添加一个 token 令牌，则该 token 拥有该用户的所有权限

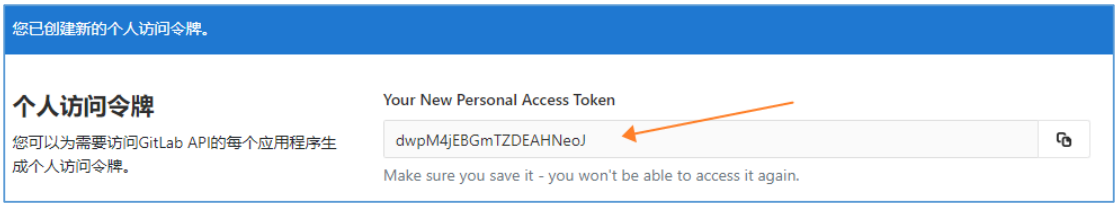
姓名：随意写，做到见名知意，比如用于 jenkins 的

Expires at：过期时间

范围：有三种，我这里选择 api，默认就是 api

创建完成后如下图，该 token 只显示一次，注意保存

dwpM4jEBGmTZDEAHNeoJ



5.1.2 token 使用举例

列出项目：

curl http://gitlab.lx2.com/api/v4/projects?private_token=dwpM4jEBGmTZDEAHNeoJ&per_page=10

```
root@jenkins:~#
root@jenkins:~# curl http://gitlab.lx2.com/api/v4/projects?private_token=dwpM4jEBGmTZDEAHNeoJ&per_page=10
[1] 1534
root@jenkins:~# [{"id":3,"description":"","name":"test-2","name_with_namespace":"曲中岭 / test-2","path":"test-2","pa
default_branch":null,"tag_list":[],"ssh_url_to_repo":"git@gitlab.lx2.com:quzl/test-2.git","http_url_to_repo":"http://
-2","readme_url":null,"avatar_url":null,"star_count":0,"forks_count":0,"last_activity_at":"2019-03-14T03:54:40.799Z",
"quzl","parent_id":null},"_links":{"self":"http://gitlab.lx2.com/api/v4/projects/3","issues":"http://gitlab.lx2.com/a
ojects/3/merge_requests","repo_branches":"http://gitlab.lx2.com/api/v4/projects/3/repository/branches","labels":"http
m/api/v4/projects/3/events","members":"http://gitlab.lx2.com/api/v4/projects/3/members"},"archived":false,"visibility
ive","avatar_url":"https://www.gravatar.com/avatar/8d229952a4eda5b3293e21ac73efeb83?s=80\u0026d=identicon","web_url":
ontainer_registry_enabled":true,"issues_enabled":true,"merge_requests_enabled":true,"wiki_enabled":true,"jobs_enabled
:true,"creator_id":2,"import_status":"none","open_issues_count":0,"public_jobs":true,"ci_config_path":null,"shared_wi
s_enabled":false,"only_allow_merge_if_all_discussions_are_resolved":false,"printing_merge_request_link_enabled":true,
"notification_level":3},"group_access":null}},{"id":2,"description":"","name":"test-1","name_with_namespace":"曲中岭
:"2019-03-01T08:53:53.758Z","default_branch":"master","tag_list":[],"ssh_url_to_repo":"git@gitlab.lx2.com:quzl/test-1
:"http://gitlab.lx2.com/quzl/test-1","readme_url":"http://gitlab.lx2.com/quzl/test-1/blob/master/README.md","avatar_u
44:14.369Z","namespace":{"id":2,"name":"quzl","path":"quzl","kind":"user","full_path":"quzl","parent_id":null},"_link
ab.lx2.com/api/v4/projects/2/issues","merge_requests":"http://gitlab.lx2.com/api/v4/projects/2/merge_requests","repo_
abels":"http://gitlab.lx2.com/api/v4/projects/2/labels","events":"http://gitlab.lx2.com/api/v4/projects/2/events","me
,"visibility":"private","owner":{"id":2,"name":"曲中岭","username":"quzl","state":"active","avatar_url":"https://www.
","web_url":"http://gitlab.lx2.com/quzl"},"resolve_outdated_diff_discussions":false,"container_registry_enabled":true
jobs_enabled":true,"snippets_enabled":true,"shared_runners_enabled":true,"lfs_enabled":true,"creator_id":2,"import_st
```

查看用户，不同的 token 拥有不同的权限，比如我第二个是 root 的 token，可查看全局的用户：

curl http://gitlab.lx2.com/api/v4/users?private_token=pCFLRiZypZoEYUyokBkS

```
root@jenkins:~# curl http://gitlab.lx2.com/api/v4/users?private_token=dwpM4jEBGmTZDEAHNeoJ&per_page=10
[1] 1548
root@jenkins:~# [{"id":2,"name":"曲中岭","username":"quzl","state":"active","avatar_url":"https://www.gravatar.com/avatar/8d229952a4
tp://gitlab.lx2.com/quzl"},{"id":1,"name":"Administrator","username":"root","state":"active","avatar_url":"https://www.gravatar.com/
on","web_url":"http://gitlab.lx2.com/root"}]
[1]+  Done                    curl http://gitlab.lx2.com/api/v4/users?private_token=dwpM4jEBGmTZDEAHNeoJ
root@jenkins:~#
root@jenkins:~#
root@jenkins:~#
root@jenkins:~#
root@jenkins:~# curl http://gitlab.lx2.com/api/v4/users?private_token=pCFLRiZypZoEYUyokBkS&per_page=10
[1] 1551
root@jenkins:~# [{"id":2,"name":"曲中岭","username":"quzl","state":"active","avatar_url":"https://www.gravatar.com/avatar/8d229952a4
tp://gitlab.lx2.com/quzl","created_at":"2019-03-01T08:43:00.655Z","bio":null,"location":null,"public_email":"","skype":"","linkedin"
_sign_in_at":"2019-03-14T08:13:54.802Z","confirmed_at":"2019-03-01T08:43:00.170Z","last_activity_on":"2019-03-15","email":"quzhongli
t":100000,"current_sign_in_at":"2019-03-15T01:44:16.070Z","identities":[],"can_create_group":true,"can_create_project":true,"two_fac
"is_admin":false},{"id":1,"name":"Administrator","username":"root","state":"active","avatar_url":"https://www.gravatar.com/avatar/e6
url":"http://gitlab.lx2.com/root","created_at":"2019-03-01T03:02:27.777Z","bio":"","location":"","public_email":"","skype":"","linke
st_sign_in_at":"2019-03-14T08:05:13.707Z","confirmed_at":"2019-03-01T03:02:27.487Z","last_activity_on":"2019-03-15","email":"admin@e
t":100000,"current_sign_in_at":"2019-03-15T03:40:34.406Z","identities":[],"can_create_group":true,"can_create_project":true,"two_fac
,"is_admin":true}]
[1]+  Done                    curl http://gitlab.lx2.com/api/v4/users?private_token=pCFLRiZypZoEYUyokBkS
root@jenkins:~#
```

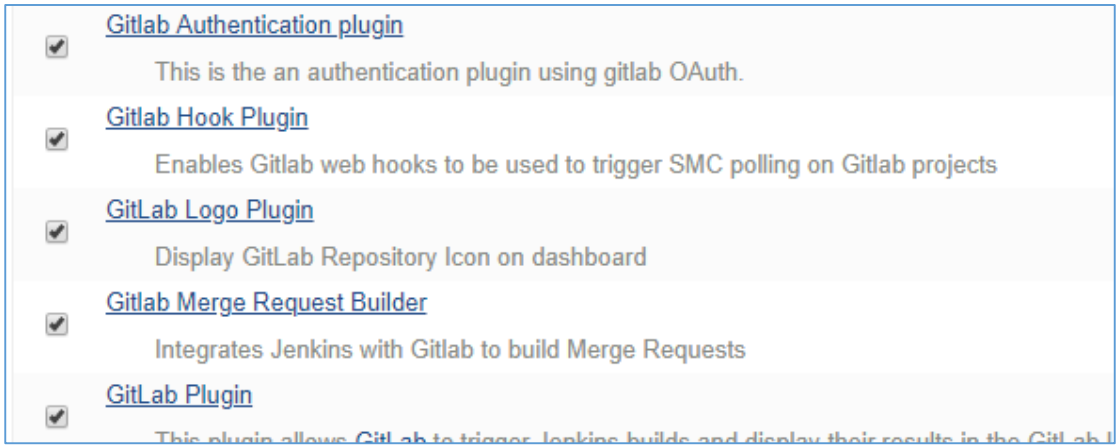
更多例子，参考官方文档

https://docs.gitlab.com/ee/api/projects.html

5.2 Jenkins 配置

5.1.3 安装插件

安装 gitlab 的相关插件



5.1.4 添加凭证

就像连接 svn 添加 svn 的凭证一样，添加一个 gitlab 的凭证，该凭证用于登陆 gitlab，可做普通用户名密码登陆所做的操作。

如下图：

如上图，添加凭证:

类型：Gitlab API token

范围：全局

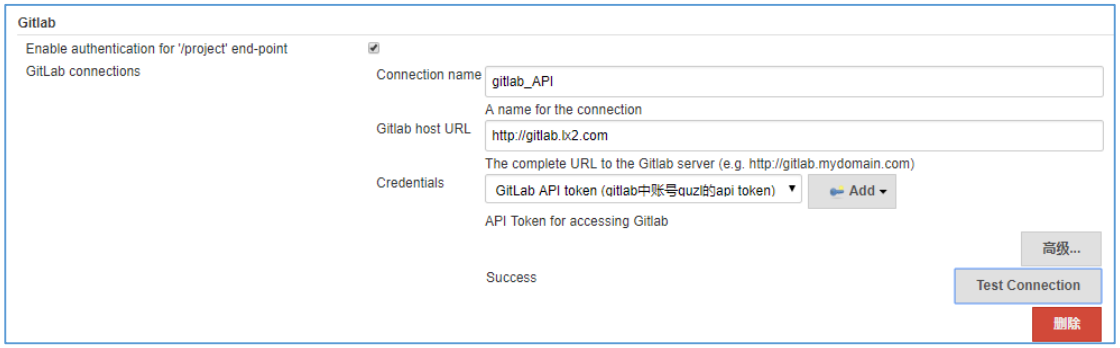
API token：输入刚刚生产的

ID：可以为空，自动生成

描述：随意

5.1.5 配置 gitlab

系统管理—>系统设置—>Gitlab



Connection name：随意
Gitlab host URL：填写 gitlab 的地址
Credentials: 选择上一步创建的凭证
点击 Test Connection，显示 Success 即表示成功，保存配置

参考文档:
http://www.hjqjk.com/2017/GitLab-install-config.html
https://www.jianshu.com/p/2400d9e57fd1