

# docker私有镜像仓库

有时候使用 Docker Hub 这样的公共仓库可能不方便，用户可以创建一个本地仓库供私人使用。

本节介绍如何使用本地仓库。

[docker-registry](#) 是官方提供的工具，可以用于构建私有的镜像仓库。但是面临一些维护问题。比如某些镜像删除以后空间默认是不会回收的，需要一些命令去回收空间然后重启 Registry。

Nexus 是第三方镜像仓库工具，相比官方的工具性能要好点。在企业中把内部的一些工具包放入 Nexus 中是比较常见的做法，最新版本 Nexus3.x 全面支持 Docker 的私有镜像。所以使用 [Nexus3.x](#) 一个软件来管理 Docker , Maven , Yum , PyPI 等是一个明智的选择。

## 启动 Nexus 容器

```
docker run -d --name nexus3 --restart=always \  
-p 8088:8081 \  
-p 5001:5001 \  
--mount src=nexus-data,target=/nexus-data \  
sonatype/nexus3
```

如果本地没有镜像，首次运行需等待 3-8 分钟，你可以使用 `docker logs nexus3 -f` 查看日志：

```
2021-04-26 21:04:18,636+0000 INFO [jetty-main-1] *SYSTEM  
org.eclipse.jetty.server.Server - Started @169543ms  
2021-04-26 21:04:18,637+0000 INFO [jetty-main-1] *SYSTEM  
org.sonatype.nexus.bootstrap.jetty.JettyServer -  
-----  
  
Started Sonatype Nexus OSS 3.30.1-01
```

如果你看到以上内容，说明 Nexus 已经启动成功，你可以使用浏览器打开 `http://YourIP:8088` 访问 Nexus` 了。

首次运行请通过以下命令获取初始密码：

```
docker exec nexus3 cat /nexus-data/admin.password  
  
9266139e-41a2-4abb-92ec-e4142a3532cb
```

首次启动 Nexus 的默认帐号是 `admin`，密码则是上边命令获取到的，点击右上角登录，首次登录需更改初始密码。

登录之后可以点击页面上方的齿轮按钮按照下面的方法进行设置。

## 创建仓库

创建一个私有仓库的方法：`Repository->Repositories` 点击右边菜单 `Create repository` 选择 `docker (hosted)`

- **Name:** 仓库的名称
- **HTTP:** 仓库单独的访问端口 (例如: **5001**)
- **Hosted -> Deployment policy:** 请选择 **Allow redeploy** 否则无法上传 Docker 镜像。

其它的仓库创建方法请各位自己摸索, 还可以创建一个 `docker (proxy)` 类型的仓库链接到 DockerHub 上。再创建一个 `docker (group)` 类型的仓库把刚才的 `hosted` 与 `proxy` 添加在一起。主机在访问的时候默认下载私有仓库中的镜像, 如果没有将链接到 DockerHub 中下载并缓存到 Nexus 中。

## 添加访问权限

菜单 `Security->Realms` 把 Docker Bearer Token Realm 移到右边的框中保存。

添加用户规则: 菜单 `Security->Roles -> Create role` 在 `Privileges` 选项搜索 `docker` 把相应的规则移动到右边的框中然后保存。

添加用户: 菜单 `Security->Users -> Create local user` 在 `Roles` 选项中选中刚才创建的规则移动到右边的窗口保存。

## Docker主机访问镜像仓库

配置非https仓库地址, 因为 Docker 默认不允许非 `HTTPS` 方式推送镜像。我们可以通过 Docker 的配置选项来取消这个限制, 或者查看下一节配置能够通过 `HTTPS` 访问的私有仓库。

对于使用 `systemd` 的系统, 请在 `/etc/docker/daemon.json` 中写入如下内容 (如果文件不存在请新建该文件)

```
# vi /etc/docker/daemon.json

{
  "registry-mirrors": [
    "https://docker.mirrors.ustc.edu.cn",
    "https://registry.docker-cn.com",
    "https://hub-mirror.c.163.com",
    "https://mirror.ccs.tencentyun.com",
    "https://reg-mirror.qiniu.com"
  ],
  "insecure-registries": [
    "192.168.21.142:5001"
  ]
}
```

重启docker服务

```
[root@localhost ~]# systemctl restart docker
```

标记镜像

```
docker tag autotpsite:v1 192.168.21.142:5001/autotpsite:v1
```

登录仓库

```
docker login 192.168.21.142:5001
```

推送镜像

```
docker push 192.168.21.142:5001/autotpsite:v1
```

另一台机器测试拉取镜像

```
docker pull 192.168.21.142:5001/autotpsite:v1
```

## Jenkins分布式环境搭建

Jenkins分布式使用方法



节点可以作为jenkins服务器控制的一台个人电脑或者服务器，节点上不需要安装jenkins服务，只需要有java运行环境就可以。

CentOS7安装java环境

安装之前先检查一下系统有没有自带open-jdk

```
rpm -qa |grep java
rpm -qa |grep jdk
rpm -qa |grep gcj
```

如果没有输入信息表示没有安装。

如果安装可以使用 `rpm -qa | grep java | xargs rpm -e --nodeps` 批量卸载所有带有Java的文件  
这句命令的关键字是java

首先检索包含java的列表

```
yum list java*
```

检索1.8的列表

```
yum list java-1.8*
```

安装1.8.0的所有文件

```
yum install java-1.8.0-openjdk* -y
```

使用命令检查是否安装成功

```
java -version
```

到此安装结束了。这样安装有一个好处就是不需要对path进行设置，自动就设置好了

1.打开公网地址<http://devops.sqtest.online:7073/>

输入用户名和密码登录



## 欢迎来到 Jenkins!

登录

☐

保持登录状态

点击任意一个节点机，进入节点列表

Jenkins

新建任务  
用户列表  
构建历史  
编辑视图  
项目关系  
检查文件指纹  
我的视图  
打开 Blue Ocean  
Lockable Resources  
凭据  
新建视图

构建队列 (20)

构建执行状态

node-hpc (未在线)  
node-sq46-lq (未在线)

点击进入节点列表

S	W	名称 ↓	上次成功	上次失败	上次持续时间
		couser44_pipeline_demo2	1月13天 - #9	1月14天 - #7	1天5小时
		demossionder	没有	无	无
		devops_node	27天 - #12	26天 - #13	52秒
		devops_test	25天 - #4	无	1分10秒
		firstPipeline	没有	无	无
		firstPipeline	9天12小时 - #16	10天 - #14	1分31秒
		firstProject	没有	无	无
		henry_course_combat	4天11小时 - #3	4天11小时 - #2	15秒
		huawei_day2	16天 - #1	16天 - #4	23秒
		linxiao_project	12天 - #1	无	4分35秒
		myProject	9天12小时 - #21	12天 - #4	44秒

再点击

Jenkins > 节点列表 > node-hpc

返回列表  
状态  
删除节点  
配置从节点  
构建历史  
负载统计  
日志  
打开 Blue Ocean

构建执行状态

点击

## 代理 node-hpc

连接中断

```
java.nio.channels.ClosedChannelException
    at org.jenkinsci.remoting.protocol.NetworkLayer.onRecvClosed
    at org.jenkinsci.remoting.protocol.impl.NIONetworkLayer.read
    at org.jenkinsci.remoting.protocol.IOHub$OnReady.run(IOHub.j
    at jenkins.util.ContextResettingExecutorService$1.run(Context
    at jenkins.security.ImpersonatingExecutorService$1.run(Imper
    at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadP
    at java.util.concurrent.ThreadPoolExecutor$Worker.run(Thread
    at java.lang.Thread.run(Thread.java:748)
```

节点连接Jenkins的方式如下:

- 在浏览器中启动节点
- 在命令行中启动节点

```
java -jar agent.jar -jnlpUrl http://devops.sqtest.online:7073/791d4f871b1bfa3c204cea822d50566fa19a5e2ecfad64539b2df8e36bd192lworkspace"
```

2.选择左下角代理节点，选择新建节点。

Jenkins > Nodes

返回工作台  
新建节点

构建队列

队列中没有构建任务

构建执行状态

agent-1  
1 空闲

S	名称 ↓	架构	时钟差异	响应时间
	agent-1	Linux (amd64)	1.2 秒 落后	556ms
	master	Linux (amd64)	已同步	0ms
	node-xiaoze	Windows 10 (amd64)	已同步	2611ms
	testnode1		N/A	N/A
	win10-haiwen		N/A	N/A
获取到的数据		8 分 47 秒	8 分 47 秒	8 分 47 秒

3.填写节点名称，选择固定节点，点击确定。

节点名称

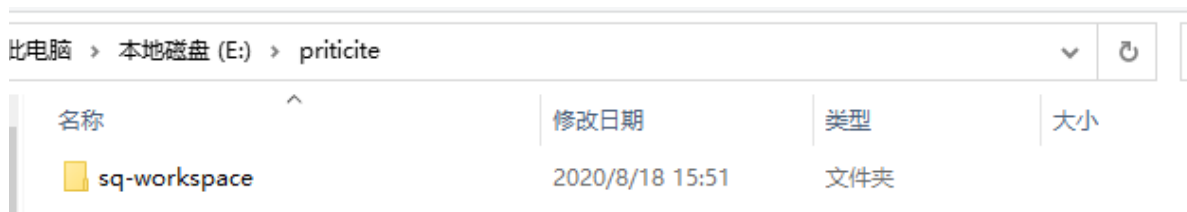
☒ 固定节点

添加一个普通、固定的节点到Jenkins。之所以叫做“固定”，是因为Jenkins没给这种节点提供更高级的集成方式，例如：动态配置。没有其他类型能选择的话可以选择该类型；例如，你在添加不受Jenkins管理的物理机、虚拟机等等。

☐ 复制现有节点

要复制的任务名称

4.电脑任意路径新建一个文件夹作为远程工作目录。



5.进入文件夹的复制目录路径。



6.配置节点，然后点击保存。

名字

描述

执行器数量

远程工作目录  本机新建的远程工作目录

标签

用法  选择only build jobs，只有这个节点的任务才会运行

启动方式  选择通过java web 启动代理

☐ 禁用工作目录

自定义工作目录

内部数据目录

☐ 当工作目录缺失时失败

☐ Use WebSocket

可用性

7.点击高级，填写Tunnel连接映射的端口号，别忘了开头的冒号。

☐ 禁用工作目录

自定义工作目录

内部数据目录

☐ 当工作目录缺失时失败

☐ Use WebSocket

Tunnel 连接位置

JVM 选项

可用性

**Tips:**如果你用的Jenkins服务器是自己用容器搭建的，端口这里不需要设置，只需要将容器的50000端口映射出来，jenkins节点默认连接的是50000端口。

8.点击节点进入，点击下载agent.jar包，最好跟你的工作目录同级目录方便查找。

S	名称 ↓	架构	时钟差异	响应时间
	<a href="#">agent-1</a>	Linux (amd64)	已同步	555ms
	<a href="#">master</a>	Linux (amd64)	已同步	0ms
	<a href="#">node-sq</a>		N/A	N/A



## Agent node-sq

临时断开此节点

节点连接Jenkins的方式如下：

- Launch 在浏览器中启动节点
- 在命令行中启动节点

```
java -jar agent.jar -jnlpUrl http://devops.sqtest.online:7073/computer/node-sq/slave-agent.jnlp -secret ee4fe33bdf31368b6dab18a8394024372250345402736890d6bc291bd50343 -workDir "E:\priticite\sq-workspace"
```

Run from agent command line, with the secret stored in a file:

```
echo ee4fe33bdf31368b6dab18a8394024372250345402736890d6bc291bd50343 > secret-file
java -jar agent.jar -jnlpUrl http://devops.sqtest.online:7073/computer/node-sq/slave-agent.jnlp -secret @secret-file -workDir "E:\priticite\sq-workspace"
```

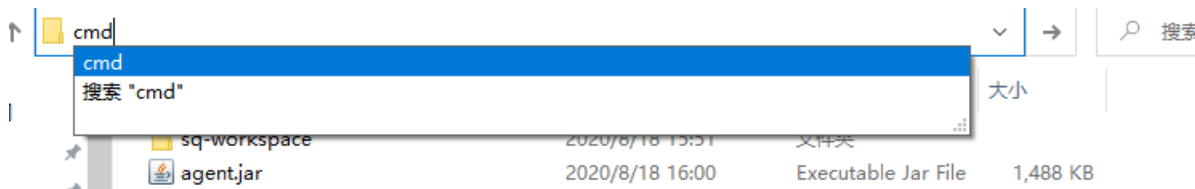
## 关联到node-sq的项目

无

此电脑 > 本地磁盘 (E:) > priticite >

名称	修改日期	类型	大小
sq-workspace	2020/8/18 15:51	文件夹	
agent.jar	2020/8/18 16:00	Executable Jar File	1,488 KB

9.打开命令行，切入agent.jar所在目录，在命令行中启动节点。




复制第一条命令



## 代理 node-sq

临时断开此

节点连接Jenkins的方式如下:

-  Launch 在浏览器中启动节点
- 在命令行中启动节点

```
java -jar agent.jar -jnlpUrl http://devops.sqtest.online:7073/computer/node-sq/slave-agent.jnlp -secret  
ec4fe33bdf31368b6dab18a8394024372250345402736890d6bcaf291bd50343 -workDir "E:\priticite\sq-workspace"
```

粘贴命令到命令行


```
选择C:\Windows\System32\cmd.exe - java -jar agent.jar -jnlpUrl http://devops.sqtest.online:7073/computer/node-xiaoze/slave-agent.jnl...  
Microsoft Windows [版本 10.0.18363.1016]  
(c) 2019 Microsoft Corporation。保留所有权利。  
D:\>java -jar agent.jar -jnlpUrl http://devops.sqtest.online:7073/computer/node-xiaoze/slave-agent.jnlp -secret db7d2210  
16fed8a04b9ffd3f92d9e562658e6a5acafbbf527124fd4da6dfdb69 -workDir "D:\jenkins-workspace"  
八月 18, 2020 3:37:49 下午 org.jenkinsci.remoting.engine.WorkDirManager.initializeWorkDir
```

10.出现connected代表连接成功

```
选择C:\Windows\System32\cmd.exe - java -jar agent.jar -jnlpUrl http://devops.sqtest.online:7073/computer/node-xiaoze/slave-agent.jnl...  
八月 18, 2020 3:37:50 下午 hudson.remoting.jnlp.Main createEngine  
信息: Setting up agent: node-xiaoze  
八月 18, 2020 3:37:50 下午 hudson.remoting.jnlp.Main$CuiListener <init>  
信息: Jenkins agent is running in headless mode.  
八月 18, 2020 3:37:50 下午 hudson.remoting.Engine startEngine  
信息: Using Remoting version: 4.3  
八月 18, 2020 3:37:50 下午 org.jenkinsci.remoting.engine.WorkDirManager initializeWorkDir  
信息: Using D:\jenkins-workspace\remoting as a remoting work directory  
八月 18, 2020 3:37:50 下午 hudson.remoting.jnlp.Main$CuiListener status  
信息: Locating server among [http://devops.sqtest.online:7073/]  
八月 18, 2020 3:37:50 下午 org.jenkinsci.remoting.engine.JnlpAgentEndpointResolver resolve  
信息: Remoting server accepts the following protocols: [JNLP4-connect, Ping]  
八月 18, 2020 3:37:50 下午 org.jenkinsci.remoting.engine.JnlpAgentEndpointResolver resolve  
信息: Remoting TCP connection tunneling is enabled. Skipping the TCP Agent Listener Port availability check  
八月 18, 2020 3:37:50 下午 hudson.remoting.jnlp.Main$CuiListener status  
信息: Agent discovery successful  
Agent address: devops.sqtest.online  
Agent port: 8072  
Identity: 91:f4:b9:8c:00:e1:66:2a:77:8f:db:d9:ab:9c:d0:4f  
八月 18, 2020 3:37:50 下午 hudson.remoting.jnlp.Main$CuiListener status  
信息: Handshaking  
八月 18, 2020 3:37:50 下午 hudson.remoting.jnlp.Main$CuiListener status  
信息: Connecting to devops.sqtest.online:8072  
八月 18, 2020 3:37:50 下午 hudson.remoting.jnlp.Main$CuiListener status  
信息: Trying protocol: JNLP4-connect  
八月 18, 2020 3:37:50 下午 hudson.remoting.jnlp.Main$CuiListener status  
信息: Remote identity confirmed: 91:f4:b9:8c:00:e1:66:2a:77:8f:db:d9:ab:9c:d0:4f  
八月 18, 2020 3:37:51 下午 hudson.remoting.jnlp.Main$CuiListener status  
信息: Connected  
八月 18, 2020 7:53:29 下午 hudson.remoting.jnlp.Main$CuiListener status
```

同时节点显示在线状态



 新建节点

## 构建队列

队列中没有构建任务

## 构建执行状态

 **agent-1** (未在线)

 **node-sq** (未在线)

 **win10-haiwen**

1 空闲

2 空闲

 [age](#)

 [mas](#)

 [nod](#)

 [win](#)

获取

由于之前设置节点的时候选择了只运行绑定此节点的任务（这样为了避免别人用你的电脑当肉鸡），所以配置任务时需要设置运行的节点。

在任务的配置界面,填上之前节点定义的标签（标签可以个性化定义-保证唯一，否则别人也可以通过相同的标签选到你的节点）

**General** 源码管理 构建触发器 构建环境 构建 构建后操作

☐ 丢弃旧的构建

☐ 参数化构建过程

☐ 关闭构建

☐ 在必要的时候并发构建

☒ 限制项目的运行节点

标签表达式

Label win10 is serviced by 1 node. Permissions or other restrictions provided by plugins may prevent this job from running on those nodes.


☐ 静默期



☐ 重试次数


保存设置后运行该任务，就会在指定的节点运行了。

构建队列

队列中没有构建任务

  [testjob](#)


  [testpipelin](#)

  [testuser1](#)

图标: 小虫大

构建执行状态

 agent-1 (未在线)

 node-sq (未在线)

 win10-haiwen

1 空闲

2 [testjob1](#)

#15

## Jenkins-pipeline语法

传统方法: 自由风格

进阶方法: pipeline流水线脚本

Jenkins 2.0开始加入的功能

pipeline: 用代码定义一切软件的生产过程, 构建-单元测试-部署-自动化测试-性能-安全-交付

Jenkins提供的语法来定义软件生产过程

## 脚本式语法

### 基本语法

node定义脚本任务执行在那台机器--必须要有

```
node('机器的标签') #如果不写执行任务的机器, 默认在master上面运行
{
    待执行的任务
}
```

```
node('haiwen'){
    echo '执行pipeline测试'
}
```

stage表示某个环节, 对应的是视图中的小方块, 可以自定义环节名称和其中要执行的代码(可以没有, 但是建议有)

```
stage('环节名称'){
    待执行的任务
}
```

```

node('haiwen'){
    stage('阶段1'){
        echo '执行pipeline测试'
    }
    stage('阶段2'){
        echo '执行pipeline测试'
    }
    stage('阶段3'){
        echo '执行pipeline测试'
    }
    stage('阶段4'){
        echo '执行pipeline测试'
    }
}

```

node和stage可以相互嵌套

```

stage('阶段1'){
    node{
        sh "echo '执行pipeline测试' "
    }
    node('haiwen'){
        stage('阶段2'){
            bat "echo '执行pipeline测试' "
        }
        stage('阶段3'){
            bat "echo '执行pipeline测试'"
        }
    }
}

```

## JenkinsFile管理流水线脚本

流水线

定义

Pipeline script from SCM

SCM: Git

Repositories

Repository URL: git@gitee.com:shonekey666/autotpsite.git

Credentials: haiwen (haiwen)

Branches to build

指定分支 (为空时代表any): \*/master

源代码浏览器: (自动)

Additional Behaviours: Jenkinsfile

保存 应用

Jenkinsfile文件所在的仓库地址

添加节点的git密钥

注意文件路径

将Jenkinsfile文件提交到git仓库，Jenkins会读取仓库

```
node('haiwen_linux'){ //node表示任务执行所在的机器--通过节点标签指定
    stage('单元测试'){
        echo '执行单元测试'
    }
    stage('构建镜像'){
        echo '执行构建镜像'
    }
    stage('推送镜像'){
        echo '执行推送镜像'
    }
    stage('更新服务'){
        sh 'touch testfile'
        sh "echo '执行更新服务' > testfile"
        writeFile encoding: 'utf-8', file: 'testfile2', text: '文本写入测试\n'
    }
}
```

## Pipeline部署实战

compose.yml

```
version: "3" # 表示compose版本，有1, 2, 3 最新是3 必须用字符串表示
services:
    #定义服务-容器启动相关参数
    frontend: #服务的名称 可以自定义
        image: nginx:latest # 指定容器启动所需的镜像
        container_name: mynginx # 指定容器名称
        ports: # 端口映射 可以映射多个端口
            - 80:80 # 端口映射不要加空格
        volumes: # 目录挂载 可以挂载多个
            - /root/conf:/etc/nginx
            - /root/html:/usr/share/nginx/html
        depends_on: # 表示依赖的服务，在依赖的服务成功启动后才会启动
            - backend

    backend:
        image: autotpsite:v1
        container_name: autotpsite
        ports:
            - 8081:8081
```

Jenkinsfile

```
node('Linux_haiwen'){
    stage('单元测试'){
        checkout scm
        echo '执行单元测试'
    }
    stage('Build镜像'){
        echo '开始Build镜像'
        sh 'docker build -t autotpsite:v1 .'
    }
    stage('Push镜像'){
```

```

echo '开始Push镜像'
sh 'docker tag autotpsite:v1 192.168.21.142:5001/autotpsite:v1'
sh 'docker login 192.168.21.142:5001'
sh 'docker push 192.168.21.142:5001/autotpsite:v1'
}
stage('更新服务'){
    try {
        echo '开始更新服务'
        sh 'docker-compose -f compose.yml up -d'
    }
    catch (Exception e) {
        echo '更新服务失败'
        println(e) //这里捕获异常打印
        sh 'docker-compose down'
    }
}
}
}

```

## gitee配置WebHook

### jenkins端

项目配置>构建触发器>Generic Webhook Trigger

当前Jenkins平台所有webhook都是一样的: [http://JENKINS\\_URL/generic-webhook-trigger/invoke](http://JENKINS_URL/generic-webhook-trigger/invoke)

因此需要自定义token,目的是为了区分其他也配置了Generic Webhook Trigger的Jenkins任务

如果不配置token会导致其他配置了Generic Webhook Trigger的任务也会被触发,同时也是为了安全性。

token尽量配置成随机的字符串

保存

### gitee端

进入远程仓库, 选择管理>WebHooks

URL部分填写:

```

http://devops.sqtest.online:7073/generic-webhook-trigger/invoke?
token=#test#@dev~ops!

```

格式是

```

http://JENKINS_URL/generic-webhook-trigger/invoke?token=你所设置的token

```

选择触发事件, 比如Push, 那么当仓库收到更新推送时会触发远程构建

更新保存, 测试一下, 发现任务构建成功

## 注意

因为Gitee是公网环境，所以此时Jenkins所在的服务器也必须处于公网环境，否则Gitee的请求是无法发送到Jenkins对应的任务的，当然更无法实现自动触发任务构建了。

## 附录

### GIT安装（已安装的忽略）

在 Windows 平台上安装 Git 同样轻松，有个叫做 msysGit 的项目提供了安装包，可以到 GitHub 的页面上下载 exe 安装文件并运行：

安装包下载地址：<https://gitforwindows.org/>

官网慢，可以用国内的镜像：<https://npm.taobao.org/mirrors/git-for-windows/>。



完成安装之后，就可以使用命令行的 git 工具（已经自带了 ssh 客户端）了，另外还有一个图形界面的 Git 项目管理工具。

在开始菜单里找到"Git"->"Git Bash"，会弹出 Git 命令窗口，你可以在该窗口进行 Git 操作。

### Git 配置

Git 提供了一个叫做 git config 的工具，专门用来配置或读取相应的工作环境变量。

这些环境变量，决定了 Git 在各个环节的具体工作方式和行为。

配置个人的用户名称和电子邮件地址：

```
$ git config --global user.name "haiwen"
$ git config --global user.email haiwen@test.com
```

## 查看配置信息

要检查已有的配置信息，可以使用 `git config --list` 命令：

```
$ git config --list
```

## 生成SSH Key(密钥)

```
$ ssh-keygen -t rsa -C "你的邮箱"
```

此处会提示 `Enter file in which to save the key (/Users/shutong/.ssh/id_rsa)`：这样一段内容,让我们输入文件名，这里不用管，直接按 `enter` 键就好了。

之后会有提示你是否需要设置密码，如果设置了每次使用Git都会用到密码，一般都是直接不写为空，直接 `enter` 就好了。

上述操作执行完毕后，在 `~/.ssh/` 目录会生成 `id-rsa` (私钥)和 `id-rsa.pub` (公钥)

## 同步本地仓库到Gitee

大家都知道国内访问 Github 速度比较慢，很影响我们的使用。

如果你希望体验到 Git 飞一般的速度，可以使用国内的 Git 托管服务——[Gitee \(gitee.com\)](https://gitee.com)。

Gitee 提供免费的 Git 仓库，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，Gitee 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。

接下来我们学习一下如何使用 Gitee。

由于我们的本地 Git 仓库和 Gitee 仓库之间的传输是通过SSH加密的，所以我们需要配置验证信息。

### 1、我们先在 [Gitee](https://gitee.com) 上注册账号并登录后，然后上传自己的 SSH 公钥。

我们在 Git Github 章节已经生成了自己的 SSH 公钥，所以我们只需要将用户主目录下的 `~/.ssh/id_rsa.pub` 文件的内容粘贴 Gitee 上。

选择右上角用户头像 -> 设置，然后选择 "SSH公钥"，填写一个便于识别的标题，然后把用户主目录下的 `.ssh/id_rsa.pub` 文件的内容粘贴进去：



成功添加后如下图所示：

## SSH公钥

使用SSH公钥可以让你在你的电脑和码云通讯的时候使用安全连接（Git的Remote要使用SSH地:

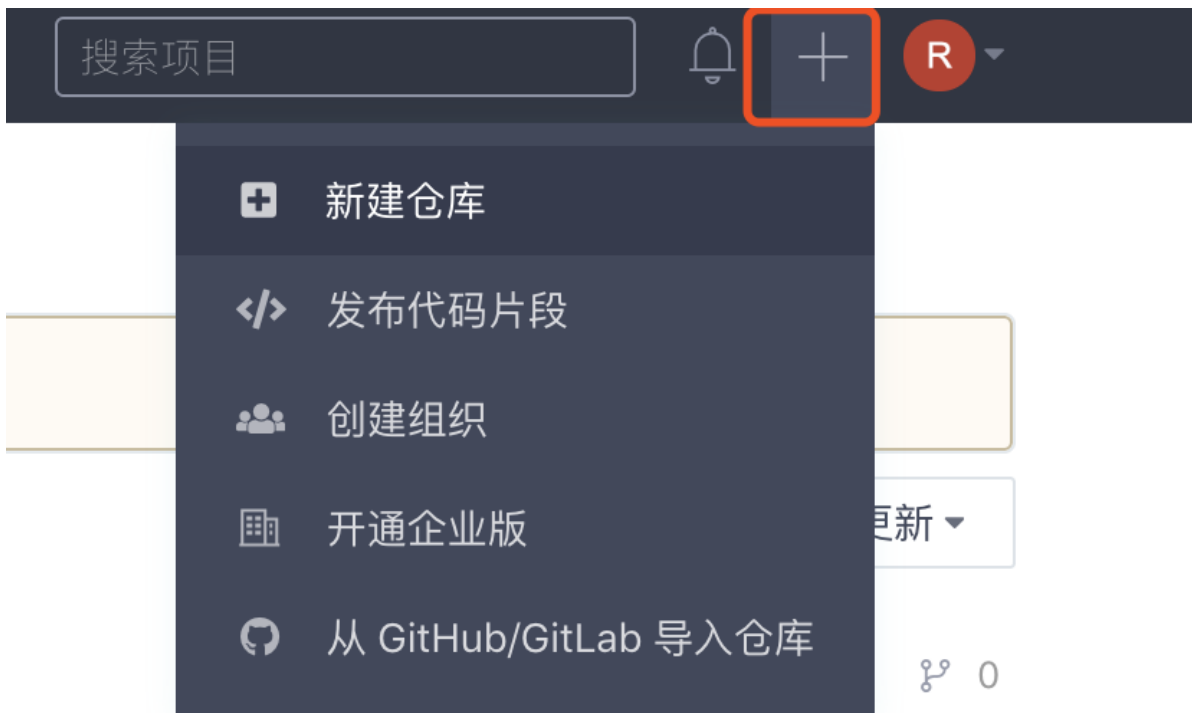
您当前的SSH公钥数: 1

RUNOOB SSH Key SHA256: T8aRJzyLdem5a26LhYA3CB+c [redacted] 添加于 刚刚

接下来我们创建一个项目。

点击右上角的 + 号，新建仓库：





然后添加仓库信息：

仓库名称 ✓

runoob-test

归属 路径

RUNOOB / runoob-test

仓库地址: <https://gitee.com/imnoob/runoob-test>

仓库介绍 非必填

RUNOOB 测试

是否开源

☐ 私有 ☒ 公开

任何人都可以访问该仓库的代码和其他任何形式的资源

选择语言 添加 .gitignore 添加开源许可证 ⓘ

JavaScript 请选择 .gitignore 模板 Apache-2.0

☐ 使用Readme文件初始化这个仓库

☐ 使用Issue模板文件初始化这个仓库 ⓘ

创建成功后看到如下信息：

📄 1 次提交      🔗 1 个分支      🏷️ 0 个标签      📦 0 个发行版

master ▾    + Pull Request    + Issue    文件 ▾    Web IDE    📌 挂件

 RUNOOB 最后提交于 1 分钟前 Initial commit

 LICENSE      Initial commit

接下来我们看下连接信息：

📦 0 个发行版      👤 1 位贡献者

克隆/下载 ▾

HTTPS    **SSH**    SVN    SVN+SSH

git@gitee.com:imnoob/runoob-test.git    复制

📄 下载ZIP

## 企业级软件开发协作工具

代码托管 项目管理 文档协作 完备安全策略

[了解更多](#)

项目名称最好与本地库保持一致。

然后，我们在本地库上使用命令 **git remote add** 把它和 Gitee 的远程库关联：

```
git remote add origin git@gitee.com:imnoob/runoob-test.git
```

之后，就可以正常地用 **git push** 和 **git pull** 推送了！

如果在使用命令 **git remote add** 时报错：

```
git remote add origin git@gitee.com:imnoob/runoob-test.git
fatal: remote origin already exists.
```

这说明本地库已经关联了一个名叫 origin 的远程库，此时，可以先用 **git remote -v** 查看远程库信息：

```
git remote -v
origin    git@github.com:tianqixin/runoob.git (fetch)
origin    git@github.com:tianqixin/runoob.git (push)
```

可以看到，本地库已经关联了 origin 的远程库，并且，该远程库指向 GitHub。

我们可以删除已有的 GitHub 远程库：

```
git remote rm origin
```

再关联 Gitee 的远程库（注意路径中需要填写正确的用户名）：

```
git remote add origin git@gitee.com:imnoob/runoob-test.git
```

此时，我们再查看远程库信息：

```
git remote -v
origin    git@gitee.com:imnoob/runoob-test.git (fetch)
origin    git@gitee.com:imnoob/runoob-test.git (push)
```

现在可以看到，origin 已经被关联到 Gitee 的远程库了。

通过 git push 命令就可以把本地库推送到 Gitee 上。

有的小伙伴又要问了，一个本地库能不能既关联 GitHub，又关联 Gitee 呢？

答案是肯定的，因为 git 本身是分布式版本控制系统，可以同步到另外一个远程库，当然也可以同步到另外两个远程库。

使用多个远程库时，我们要注意，git 给远程库起的默认名称是 origin，如果有多个远程库，我们需要用不同的名称来标识不同的远程库。

仍然以 runoob-test 本地库为例，我们先删除已关联的名为 origin 的远程库：

```
git remote rm origin
```

然后，先关联 GitHub 的远程库：

```
git remote add github git@github.com:tianqixin/runoob-git-test.git
```

注意，远程库的名称叫 github，不叫 origin 了。

接着，再关联 Gitee 的远程库：

```
git remote add gitee git@gitee.com:imnoob/runoob-test.git
```

同样注意，远程库的名称叫 gitee，不叫 origin。

现在，我们用 git remote -v 查看远程库信息，可以看到两个远程库：

```
git remote -v
gitee    git@gitee.com:imnoob/runoob-test.git (fetch)
gitee    git@gitee.com:imnoob/runoob-test.git (push)
github   git@github.com:tianqixin/runoob.git (fetch)
github   git@github.com:tianqixin/runoob.git (push)
```

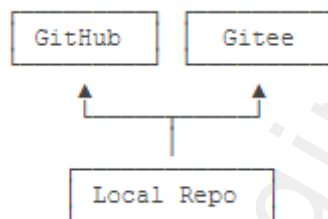
如果要推送到 GitHub，使用命令：

```
git push github master
```

如果要推送到 Gitee，使用命令：

```
git push gitee master
```

这样一来，我们的本地库就可以同时与多个远程库互相同步：



## Git日常基本操作命令

查看当前仓库状态

```
git status
```

初始化git仓库

```
git init    #生成一个隐藏的目录 .git 内部包含该仓库的版本信息
```

创建文件并添加内容

```
echo "print('hello jenkins')" > main_test.py
```

将文件添加到暂存区

```
git add main_test.py    #或者
git add .    #将该仓库下所有改动的文件添加到暂存区
```

git status

文件呈现红色，表示该文件在工作区

文件呈现绿色，表示该文件在暂存区

# GIT支线操作

## 场景1：撤销修改

```
#情况1: 文件没有提交到暂存区
echo "print 'test'" >> main_test.py #追加文件内容
git status #查看提示, 可以不写
git restore main_test.py #撤销修改, 保存到版本库的内容不会被撤销
```

```
#情况2: 文件已经提交到暂存区
echo "print 'test'" > main_test.py
git add . #添加到暂存区
cat main_test.py #发现文件被修改
git restore --staged main_test.py #把文件从暂存区拉回工作区
git restore main_test.py #撤销工作区的修改
```

## 场景2：读取之前的仓库存档

```
#回档操作, 返回上一个进度
#前情条件, 修改文件内容, 并提交到版本库
git log #查看当前仓库的改动记录
git reset --hard HEAD^ #表示返回上一个存档
```

```
git reflog #可以查看所有的改动记录, 查看存档坐标
git reset --hard 存档坐标 #实现成功穿梭到指定存档
```

## 远程仓库命令

### 查看远程仓库

```
$ git remote -v
origin git@gitee.com:shonekey666/test40_demo2.git (fetch)
origin git@gitee.com:shonekey666/test40_demo2.git (push)
#通常origin作为默认的远程仓库名称
```

### #删除远程仓库

```
git remote rm 远程仓库名
```

### 添加远程仓库

```
git remote add 远程仓库名 仓库的URL
```

## docker方式启动agent

```
docker run -i --rm --name agent1 --init -v agent1-workdir:/home/jenkins/agent
jenkins/agent java -jar /usr/share/jenkins/agent.jar -jnlpurl
http://devops.sqtest.online:7073/computer/agent-docker/slave-agent.jnlp -secret
f60e8d1c5be6ee57e0a9c9d894bb302187d976fbfca523790b3ba431c595728f -workDir
/home/jenkins/agent
```

## Groovy异常捕获语法

```
try {
    echo '执行pipeline测试'
    fail //这里制造失败
}
catch (Exception e) {
    println(e) //这里捕获异常打印
}
```