

# Jenkins Phabricator 持续集成

一、 简介.....	2
1. 持续集成.....	2
2. Jenkins.....	2
3. Phabricator.....	2
4. 目标.....	2
二、 搭建.....	2
1. 所需的软件.....	2
2. 环境搭建.....	2
1) 安装 Jenkins 和 Phabricator.....	3
2) Phabricator 配置.....	3
3) Jenkins 配置.....	3
三、 Jenkins 和 Phabricator 集成.....	11
1. 安装配置.....	11
2. 使用.....	13
四、 Reference.....	18

## 一、 简介

### 1. 持续集成

一种软件开发实践，即团队的成员经常集成他们的工作，通常每个成员每天至少集成一次，这导致每天发生多次集成。每次集成都通过自动化的构建(包括测试)来验证，从而尽快地检测出集成错误。许多团队发现这个过程会大大减少集成问题，让团队能够更快地开发内聚的软件。本质上说，持续集成就是通过快速的反馈降低风险。

### 2. Jenkins

Jenkins 是一个用 java 编写的持续集成工具。它为软件开发提供持续集成服务。它基于服务器，并运行于 servlet 容器，例如 Apache Tomcat。它支持版本控制工具，包括 CVS, Subversion, Git, Mercurial, Perforce, Clearcase 和 RTC，并能执行基于 Apache Ant 和 Apache Maven 的项目。另外，它能够通过多种方法开始构建，例如，由版本控制系统触发，由 cron-like 机制调度，由远程命令触发，或者通过一个 URL。

### 3. Phabricator

Phabricator 是一套基于网络的软件开发协作工具，包括代码审查工具、库浏览器、Herald 变化检测工具、Bug 跟踪、Phriction Wiki。Phabricator 集成了 Git、Mercurial、和 Subversion 版本控制工具。它在 Apache License 2 协议下发布的自由软件。

### 4. 目标

我们的目标是采用 Git + Phabricator+Jenkins 模式，实现 Git 托管代码，Phabricator 进行代码审查、Bug 跟踪，Jenkins 集成构建的流程。通过 Phabricator 和 Jenkins 的强大功能，完善代码管理和编译，测试以及部署，提高软件质量并加速软件开发。

## 二、 搭建

### 1. 所需的软件

为了搭建一个完整的持续集成系统，我们需要如下软件：

Apache: Internet 上最流行的 Web 服务器软件，仅支持静态网页

PHP: 在服务器端执行的嵌入 HTML 文档的脚本语言

MariaDB/MySQL: 数据库管理系统

Git: 版本控制系统的一个免费开源客户端

Jenkins: 持续构建工具

Phabricator: Code review 工具

以下分别是这些软件的安装配置和使用方法的简要介绍。

### 2. 环境搭建

### 1) 安装 Jenkins 和 Phabricator

具体内容查看安装脚本 [install-jenkins-phabricator.sh](#)。

### 2) Phabricator 配置

安装完成后，打开 <http://<your-server>>，首先注册一个管理员账户。

按照官方配置指南 [Configuration Guide](#)，需要进行多项配置，这里只介绍几个比较重要的，其余可以省略或自行参考官方指南。

#### ➤ [Accounts and Registration](#)

打开 <http://<your-server>/auth>，添加一个或多个登录系统。登录方法称为 AuthenticationProvider，例如以下：

- Username/Password: 用户使用用户名和密码进行登录或注册
- LDAP: 用户使用 LDAP 认证进行登录或注册
- OAuth: 用户使用支持 OAuth2 的提供商账户（例如 Github、Facebook 或 Google）进行登录或注册。

#### ● 恢复管理员账户

如果你意外的锁住了你的管理员用户，可以使用 `bin/auth` 脚本进行恢复，要恢复管理员账户，运行：

```
phabricator/$ ./bin/auth recover <username>
```

- 使用 Web 控制台管理账户

打开 <http://<your-server>/people/> 管理账户

- 手动创建新账户

这里有两种方式创建用户，第一种是通过 web 用户界面 <http://<your-server>/people/>，另外一种是通过 CLI 使用 `accountadmin` 可执行文件：

```
phabricator/$ ./bin/accountadmin
```

#### ➤ [Configuring Outbound Mail](#)

Phabricator 可以通过多种方式发送邮件，称为“Adapter”。这里使用 External SMTP，配置已经在安装脚本中写好，只需要设置好对应的参数，不再需要手动配置。如果需要使用其它邮件发送方式，可以查看官方文档。

#### ➤ [Diffusion User Guide: Repository Hosting](#)

Phabricator 支持 Git、Subversion 等版本控制工具，可以通过 HTTP 和 SSH 对其中托管的库进行读写访问验证。这里我们使用 Git 版本控制工具并通过 SSH 进行访问。配置已经在安装脚本中完成，但每个 Phabricator 用户还需要在 Phabricator 中添加公钥。步骤如下：

- 生成密钥

运行命令 `ssh-keygen -t rsa -C "Comment message"`

- 上传到 Phabricator

登录到 Phabricator，打开 <http://<your-server>/settings/panel/ssh>，点击 Upload Public Key，将刚才生成的密钥公钥内容 `~/.ssh/id_rsa.pub` 复制进去。

### 3) Jenkins 配置

Jenkins 安装完成后，进入 <http://<your-server>:8080/manage> 配置 Jenkins，在这个页面可以完成配置系统，全局安全设置，从磁盘重新加载配置信息，管理插件，显示系统信息，显示系统日志，显示统计信息，命令行工具，脚本控制台，管理节点，管理认证，关于 Jenkins，

管理旧数据，管理用户和关闭 jenkins 等任务。以下是各个任务的简要介绍。

在[配置系统](#)中，可能由于没有安装某些插件而缺少相应的配置选项，只需要在[插件管理](#)部分安装插件即可。对于我们的目标，我们首先需要安装 [gitplugin](#), [git clientplugin](#), [git serverplugin](#),[git parameter plugin](#),[build authorization token root plugin](#), [build with parameters](#), [post build script plugin](#), [phabricator differential plugin](#) 和 [cobertura plugin](#)。安装方法可以查看[插件管理](#)部分

a) Configure System

Jenkins 的一些基本配置内容可以查看文件/etc/sysconfig/jenkins

系统配置最上面显示了 Jenkins 所在目录

点击后面的 advances 可以配置工作目录和日志保存目录

Workspace root Directory: 设置每次构建工作区的目录

Build Record Root Directory: 设置每次构建日志的目录

Home directory	/var/lib/jenkins
Workspace Root Directory	<input type="text" value="\${ITEM_ROOTDIR}/workspace"/>
Build Record Root Directory	<input type="text" value="\${ITEM_ROOTDIR}/builds"/>
System Message	<div><div></div><div>[Plain text] <a href="#">Preview</a></div></div>
# of executors	<input type="text" value="2"/>
Labels	<input type="text"/>
Usage	<input type="text" value="Utilize this node as much as possible"/>
Quiet period	<input type="text" value="5"/>
SCM checkout retry count	<input type="text" value="0"/>
<input type="checkbox"/> Restrict project naming	

工程名称设置，可以要求工程名称符合某种格式

<input checked="" type="checkbox"/> Restrict project naming	
Naming Strategy	<div><div>Strategy</div><div><div><input type="radio"/> Default</div><div><input checked="" type="radio"/> Pattern</div></div><div>Name Pattern <input type="text" value=".*"/></div><div>Description <input type="text"/></div><div>force existing <input type="checkbox"/></div></div>

全局属性：这里可以添加一些环境变量，之后在任务中可以引用，还可以添加一些工具，只需要填写工具的名称和所在目录。

## Global properties

### ☒ Environment variables

List of key-value pairs

name

value

Add

### ☒ Tool Locations

List of tool locations

Name

Home

Add

**Maven:** 可以设置所使用的 Maven 来源，默认选项或者自己设置 maven 所在路径

## Maven Configuration

Default settings provider

Default global settings provider

如果我们之前已经安装了 JDK，那么这里使用默认选项即可，也可以指定 JAVA\_HOME 的路径。

## JDK

JDK installations

☒ JDK  
Name

JAVA\_HOME

☐ Install automatically

Add JDK

List of JDK installations on this system

## Git

Git installations

☒ Git  
Name

Path to Git executable

☐ Install automatically

上面由于可执行程序 git 已经在系统 PATH 路径中，所以可以不使用完整路径。  
后面的 ANT 和 Maven 安装类似，这里不再讲述。截图如下：

Ant

Ant installations

Ant Name

Default

ANT\_HOME

/usr/share/ant

☐ Install automatically

Add Ant

List of Ant installations on this system

Maven

Maven installations

Maven Name

Maven 3

MAVEN\_HOME

/opt/apache/apache-maven-3.2.1

☐ Install automatically

Jenkins URL: 设置 Jenkins 的网络地址  
Email Add: 设置 jenkins 的 email 地址。

Jenkins Location

Jenkins URL

http://10.60.1.147:8080/

System Admin e-mail address

Jenkins Server <luoyuanhao@software.ict.ac.cn>

Git plugin 和 CVS 使用默认配置

Git plugin

Global Config user.name Value

Global Config user.email Value

Create new accounts base on author/commmitter's email

☐

CVS

Default Compression Level

3 (Recommended) ▼

Private Key Location

/home/jenkins/.ssh/id\_rsa

Private Key Password

.....

Known Hosts Location

/home/jenkins/.ssh/known\_hosts

Authentication

Add

Email 配置: 设置发送邮件的 smtp 服务器, 应特别注意, 缺省邮箱后缀  
点击“Advanced”可以配置 smtp 认证, 包括用户名, 用户密码, 缺省编码等。

Test configuration by sending test e-mail: 通过发送邮件测试配置是否成功, 点击 Test Configuration, 如果成功会显示 Email was successfully sent, 否则下面会报错, 出错时请认真看下面的出错提示, 可能错误是 SMTP server 没有配置对, 默认是 localhost, 或者是某些服务没有打开, 或者是端口已经被其他程序占用, 默认端口是 25, 可以不填写, 另外重启一下也可能就成功。

**E-mail Notification**

SMTP server

Default user e-mail suffix

☒ Use SMTP Authentication

User Name

Password

Use SSL ☐

SMTP Port

Reply-To Address

Charset

☒ Test configuration by sending test e-mail

Test e-mail recipient

Email was successfully sent

到这里之后点击 **apply** 或者 **save** 保存设置，如果出现错误，根据错误提示进行更正即可。

#### b) Configure global security

这里主要完成一些关于系统安全和访问权限的配置，勾选 **Enable security** 才可以配置，否则不启用其机制。前面的一般使用默认配置，如下图



## Configure Global Security

☒ Enable security

TCP port for JNLP slave agents ☐ Fixed :  ☒ Random ☐ Disable

Disable remember me ☐

Access Control

**Security Realm**

☐ Delegate to servlet container

☒ Jenkins' own user database

☒ Allow users to sign up

☐ LDAP

☐ Unix user/group database

**Authorization**

☐ Anyone can do anything

☐ Legacy mode

☒ Logged-in users can do anything

☐ Matrix-based security

☐ Project-based Matrix Authorization Strategy

Disable remember me:如果勾选，每次登陆都需要输入用户名和密码

Delegate to servlet container: 使用容器的用户，tomcat 也可以添加用户，这里不讨论。

Jenkins's own user database: 使用 jenkins 自身的账户，需要勾选 Allow users to sign up 以允许注册

上面还有用户访问权限设置，有下面一些选项：

Anyone can do anythins:任何人可以做任何事

Legacy mode:如果是管理员，则拥有所有权限，其他用户或匿名用户都只有读权限

Logged-in users can do anythin: 登录用户可以做任何事

Matrix-based security: 基于矩阵的安全，功能看点击后面蓝色问号后的提示，见下图：

☐ Matrix-based security

In this scheme, you can configure who can do what by using a big table.

Each column represents a permission. Hover the mouse over the permission names to get more information about what they represent.

Each row represents a user or a group (often called 'role', depending on the security realm.) This includes a special user 'anonymous', which represents unauthenticated users, as well as 'authenticated', which represents all authenticated users (IOW, everyone except anonymous users.) Use the text box below the table to add new users/groups/roles to the table, and click the [x] icon to remove it from the table.

Permissions are additive. That is, if an user X is in group A, B, and C, then the permissions that this user actually has are the union of all permissions given to X, A, B, C, and anonymous.

(from [Matrix Authorization Strategy Plugin](#))

Project-based Matrix Authorization strategy: 基于工程的矩阵授权策略，和上面基于矩阵的区别不大，只是基于工程的可以对各个工程进行不同的权限配置，而上面的是对所有 jobs 都一样。

☐ Project-based Matrix Authorization Strategy

This mode is an extension to "Matrix-based security" that allows additional ACL matrix to be defined for each project separately (which is done on the job configuration screen.)

This allows you to say things like "Joe can access project A, B, and C but he can't see D." See the help of "Matrix-based security" for the concept of matrix-based security in general.

ACLs are additive, so the access rights granted below will be effective for all the projects.

(from [Matrix Authorization Strategy Plugin](#))

Markup Formatter

Safe HTML

Treats the text as HTML and uses it as is without any translation, but removes potentially unsafe elements like <script>.

☐ Disable syntax highlighting

Disable syntax highlighting: 是否启用 html 语法高亮

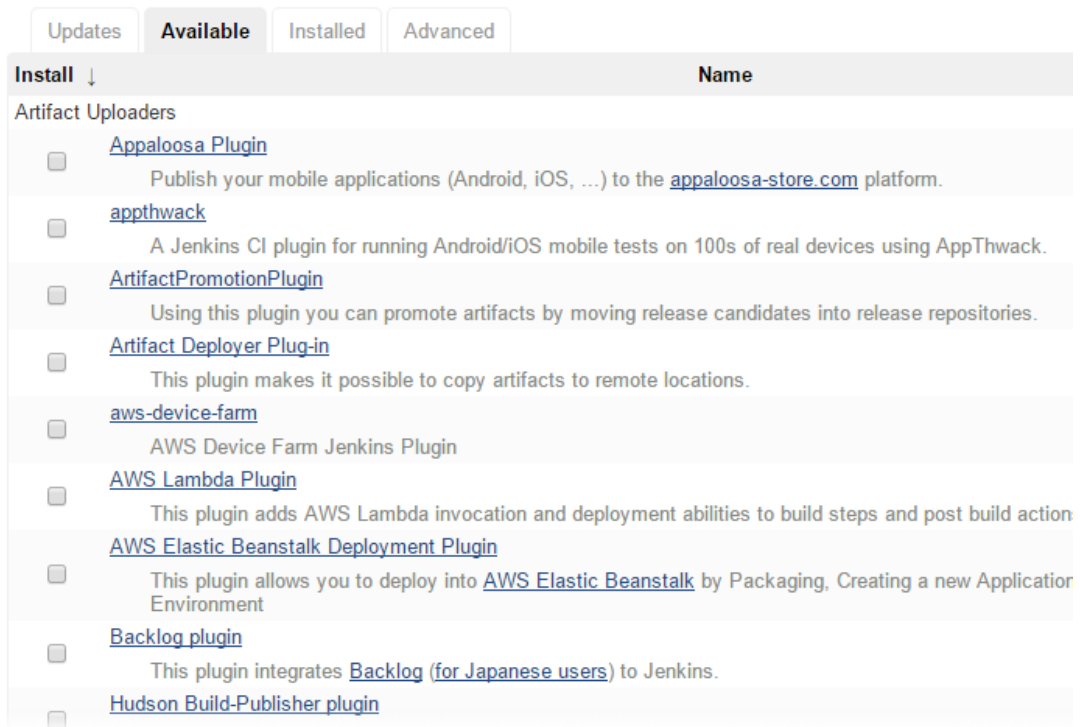
c) Reload Configuration from Disk

重新从磁盘加载配置信息。

d) Manage plugins

管理各种插件，包括安装，升级，删除等等。





如上图所示，Updates 表示已安装可升级插件，Available 表示可用但未安装插件，Installed 表示已安装插件，Advanced 里面可以从本地安装插件。如果 Available 界面没有任何可用更新，可能是更新网址不正确。进入 Advanced 界面更改更新网址为：<https://updates.jenkins-ci.org/current/update-center.json>

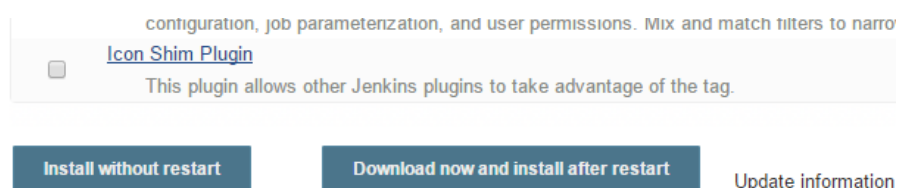
### Update Site

URL

点击 Check Now 检查是否有可用更新和可用插件。

Available 列表里有很多可选插件，插件是按照类型分类，按字母序排序的。类型包括有 Artifact Uploaders 上传文件插件，Authentication and User Management 认证和用户管理类插件，Build Notifiers 构建提示插件，构建报告插件等等。如下图所示，点击类型名可以展开此类型下所有可用的插件，每个插件都有简要的功能介绍。

勾选所有想要安装的插件，下拉到最后，可以点击安装，安装完成后重启 jenkins 即可，如下图：



e) System information

这里主要显示了一些系统属性，环境变量，已安装的插件汇总等信息。

f) System log

主要显示了系统的日志

g) Load statistics

一些统计数据，例如节点数目，正在运行的节点数，构建队列长度等信息。

h) Jenkins CLI

命令行接口，需要下载 `jenkins-cli.jar` 文件。将在 [jenkins cli 命令行使用简介.doc](#) 中详细介绍。

You can access various features in Jenkins through a command-line tool. See [the Wiki](#) for more and run it as follows:

```
java -jar jenkins-cli.jar -s http://10.60.1.147:8080/ help
```

## Available Commands



<a href="#">add-job-to-view</a>	Adds jobs to view.
<a href="#">build</a>	Builds a job, and optionally waits until its completion.
<a href="#">cancel-quiet-down</a>	Cancel the effect of the "quiet-down" command.
<a href="#">clear-queue</a>	Clears the build queue.
<a href="#">connect-node</a>	Reconnect to a node.
<a href="#">console</a>	Retrieves console output of a build.
<a href="#">copy-job</a>	Copies a job.
<a href="#">create-job</a>	Creates a new job by reading stdin as a configuration XML file.
<a href="#">create-node</a>	Creates a new node by reading stdin as a XML configuration.
<a href="#">create-view</a>	Creates a new view by reading stdin as a XML configuration.
<a href="#">delete-builds</a>	Deletes build record(s).
<a href="#">delete-job</a>	Deletes job(s).
<a href="#">delete-node</a>	Deletes node(s).
<a href="#">delete-view</a>	Deletes view(s).
<a href="#">disable-job</a>	Disables a job.
<a href="#">disconnect-node</a>	Disconnects from a node.
<a href="#">enable-job</a>	Enables a job.
<a href="#">get-job</a>	Dumps the job definition XML to stdout.
<a href="#">get-node</a>	Dumps the node definition XML to stdout.
<a href="#">get-view</a>	Dumps the view definition XML to stdout.
<a href="#">groovy</a>	Executes the specified Groovy script.

i) Script console

脚本控制台，通过执行脚本和 jenkins 交互。

j) Manage Nodes

节点管理，我们暂时只有一个 master 节点，不介绍多节点的使用。

S	Name ↓	Architecture	Clock Difference	Free Disk Space	Free Swap Space	Free Temp Space
	<a href="#">master</a>	Linux (amd64)	In sync	496.01 GB	 0 B	496.01 GB
	Data obtained	10 min	10 min	10 min	10 min	10 min

k) Manager credential

增加，修改或者删除认证，可用于 jenkins 和 jenkins job 和第三方服务的验证。

#### l) About jenkins

介绍了 jenkins 的版本信息，还有所依赖的第三方库列表和 license。

#### m) Manager Old Data

管理旧数据，具体介绍看 jenkins 的解释

#### n) Manager Users

管理用户，这个要在 Configure Global security 中选中允许用户注册才可以使用

☒ Enable security

TCP port for JNLP slave agents ☐ Fixed :  ☒ Random ☐ Disable

Disable remember me ☐

Access Control

Security Realm

☐ Delegate to servlet container

☒ Jenkins' own user database

☒ Allow users to sign up


☐ LDAP

☐ Unix user/group database

这里显示了已经注册的用户，已登录的用户后面没有红色删除标记，但可以删除其他用户，点击后面蓝色的配置标记可以进入相应用户的配置。

## Users

These users can log into Jenkins. This is a sub set of [this list](#), which also contains auto-created users who really just m: no direct Jenkins access.

User Id	
 <a href="#">luoyuanhao</a>	<a href="#">luoyuanhao</a>

#### o) Prepare for shutdown

关掉 jenkins

## 三、 Jenkins 和 Phabricator 集成

### 1. 安装配置

集成 Jenkins 和 Phabricator 需要在 Jenkins 中安装 PhabricatorDifferential Plugin 插件，配置过程如下：

在 Phabricator 中新建 robot 用户 Jenkins

### Create New User

Choose the type of user account to create. For a deta

**Account Type**

- ☐ **Create Standard User**  
Create a standard user account.
- ☒ **Create Bot User**  
Create a new user for use with automated scripts.
- ☐ **Create Mailing List User**  
Create a mailing list user to represent an existing, exte

### Create New User

You are creating a new **bot** user account.

**Username**

**Real Name**

**Email**

新建用户完成后，进入 Jenkins 用户界面 <http://<your-server>/p/jenkins>，点击右边的 **Edit Settings**，再点击 **Conduit API Tokens**，点击 **Generate API Token**，复制生成的 Token。进入到 Jenkins 系统配置页面 <http://<your-server>:8080/configure>，找到 **Phabricator** 部分，添加 **Credentials**，填写 **PhabricatorURL** 并将之前生成的 Token 粘贴到此处。

### Add Credentials

**Kind**

**Phabricator URL**

**Description**

**Conduit Token**

由于 Jenkins 中使用的系统用户是 jenkins，也需要把 jenkins 用户的 SSH Public Key 上传到 Phabricator。方法如下：

切换到 Jenkins 用户并生成密钥：

su Jenkins

ssh-keygen -t rsa -C "jenkins@phabricator server"

```
[root@nobida147 ~]# su jenkins
[jenkins@nobida147 root]$ ssh-keygen -t rsa -C "jenkins@nobida147"
```

然后进入 <http://<your-server>/settings/panel/ssh> 点击 **Upload Public Key**，将刚才生成的密钥上传到 Phabricator

## Upload SSH Public Key

Name	<input type="text" value="jenkins@nobida147"/>	Required
Public Key	<div>LOKAIVK1J95g0omy7pwwsJCyVBQGkTT68qC0457f8X STtbKhMjR4MnNfeBNvtiQ1Kd8QGxK/DXYFBUOx5j 381XrpnKGCo+B4IOJ7Sy/rbfT0ckOdtNRn3Pcepafdd alRLz/zse3rdcPWan5yvH2AcY7watW8MKK0maFlGHa nwyUznDk9jdmLEFcmqzxd4wxxUl0IG9pOCbq4rr62/g KZpF5PP/AvJzwlx1TC5hL7FxGhUqdpIbFsrK3CwudO4 hwWgFvCdG/pE2rc+mNeZla6YWKwfe4rl2nn10G8Tix 6bp3IsPlei6XA3OZd3gMniheg09ELST6fviP jenkins@nobida147</div>	Required

## 2. 使用

在 Phabricator 新建 Repository 处 <http://<your-server>/diffusion/new> 新建一个 Repository。

### Repository Name and Location

Choose a human-readable name for this repository, like "Code". You can change this later.

Name	<input type="text" value="hello-world"/>
------	--

Human-readable repository name.

Choose a "Callsign" for the repository. This is a short, unique identifier. For example, you might use "M" for your mobile app repository and "A" for your API repository.

Callsigns must be UPPERCASE, and can not be edited after the repository is created.

Callsign	<input type="text" value="A"/>
----------	--------------------------------

Short UPPERCASE identifier.

1. Clone 该库至本地，执行 `git commit --allow-empty -m "initial empty commit"`
2. `git push origin master`

```
[root@nobida147 ~]# git clone ssh://git@10.60.1.147/diffusion/A/hello-world.git
Initialized empty Git repository in /root/hello-world/.git/
PHP Warning: Module 'apc' already loaded in Unknown on line 0
remote: Counting objects: 2, done.
remote: Total 2 (delta 0), reused 0 (delta 0)
Receiving objects: 100% (2/2), done.
[root@nobida147 ~]# cd hello-world/
[root@nobida147 hello-world]# git commit --allow-empty -m "Initial empty commit"
[master 8408726] Initial empty commit
[root@nobida147 hello-world]# git push origin master
PHP Warning: Module 'apc' already loaded in Unknown on line 0
Counting objects: 1, done.
Writing objects: 100% (1/1), 195 bytes, done.
Total 1 (delta 0), reused 0 (delta 0)
remote: PHP Warning: Module 'apc' already loaded in Unknown on line 0
To ssh://git@10.60.1.147/diffusion/A/hello-world.git
  17e85dc..8408726 master -> master
[root@nobida147 hello-world]#
```

在 Jenkins <http://<your-server>:8080/view/All/newJob> 中新建对应该 Diffusion 的 Job，名称为 hello-world，free-style

由于需要将构建结果返回给 Phabricator，这里需要添加两个 String 参数 DIFF\_ID 和 PHID：

☒ This build is parameterized

#### String Parameter

Name

Default Value

Description

[Plain text] [Preview](#)

#### String Parameter

Name

Default Value

Description

在源代码管理中，选择 Git，Repository URL 填写 git clone Phabricator Diffusion 时的后半部分，由于已经上传了 Jenkins 用户的公钥到 Phabricator，这里不需要认证。

☒ Git  
Repositories

Repository URL

Credentials  [Add](#)

Name

ID of the repository, such as `origin`, to uniquely identify this repository among other remote repositories. This is the same "name" that you use in your `git remote` command. If left empty, Jenkins will generate unique names for you.

You normally want to specify this when you have multiple remote repositories.

(from [Jenkins GIT plugin](#))

Refspec

[Add Repository](#) [Delete Repository](#)

上面的 name 表示库的 ID，例如 `origin`，用于和其它远端库区分。Refspec 用于指定获取远端的哪些 refs 以及和本地 refs 对应。如果留空，则和 `git fetch` 效果相同。

Branches to build

Branch Specifier (blank for 'any')

Specify the branches if you'd like to track a specific branch in a repository. If left blank, all branches will be examined for changes and built.

The safest way is to use the `refs/heads/<branchName>` syntax. This way the expected branch is unambiguous.

If your branch name has a `/` in it make sure to use the full reference above. When not presented with a full path the plugin will only use the part of the string right of the last slash. Meaning `foo/bar` will actually match `bar`

Possible options:

- `<branchName>`  
Tracks/checks out the specified branch. If ambiguous the first result is taken, which is not necessarily the expected one. Better use `refs/heads/<branchName>`.  
E.g. `master`, `feature1`,...
- `refs/heads/<branchName>`  
Tracks/checks out the specified branch.  
E.g. `refs/heads/master`, `refs/heads/feature1/master`,...
- `<remoteRepoName>/<branchName>`  
Tracks/checks out the specified branch. If ambiguous the first result is taken, which is not necessarily the expected one. Better use `refs/heads/<branchName>`.  
E.g. `origin/master`
- `remotes/<remoteRepoName>/<branchName>`  
Tracks/checks out the specified branch.  
E.g. `remotes/origin/master`
- `refs/remotes/<remoteRepoName>/<branchName>`  
Tracks/checks out the specified branch.

**Branches to build:** 指定要跟踪的分支，如果留空，则会检查构建所有分支。

在 **BuildTriggers** 部分，勾选 **Trigger builds remotely**，这样可以通过 URL 触发构建。也就是后面 Phabricator 发出一个 URLPOST 请求触发构建。这里填写一个验证令牌，用于 Phabricator 或 URL 触发时认证。记住下面 Jenkins 提示的 Use the following URL to trigger build remotely:

[http://<your-server>:8080/job/hello-world/buildWithParameters?token=TOKEN\\_NAME&param1=value1&param2=value2](http://<your-server>:8080/job/hello-world/buildWithParameters?token=TOKEN_NAME&param1=value1&param2=value2)

**Build Triggers**

☒ Trigger builds remotely (e.g., from scripts)

Authentication Token

Use the following URL to trigger build remotely: `JENKINS_URL/job/hello-world/build?token=TOKEN_NAME or /buildWithParameters?token=TOKEN_NAME`  
 Optionally append `&cause=Cause+Text` to provide text that will be included in the recorded build cause.

☐ Build after other projects are built

在 **Build** 部分，填写需要构建的具体内容，例如编译，单元测试，集成测试等。这里打印两个参数的值。

**Build**

☒ Execute shell

Command

[See the list of available environment variables](#)

在 **Post-buildActions** 部分，添加 **Post to Phabricator**，将构建结果返回给 Phabricator。

**Post-build Actions**

☒ **Post to Phabricator**

Comment on Success ☐

Post a differential comment on successful builds

Enable Uberalls ☒

Enable code coverage reporting

Read comment from file

Add additional context to Phabricator comment by outputting to this file

Preserve Formatting ☐

Preserve comment formatting

Maximum comment character length

Comment with console link on Failure ☐

Post a differential comment on failed builds with a link to the raw Console Output

在 Phabricator 中新建 Harbormaster build plan

<http://<your-server>/harbormaster/plan/edit>

**Harbormaster > Build Plans > New Build Plan**

**New Build Plan**

Plan Name

在该 plan 中添加 build step:

Edit Step: Make HTTP Request

Name

Make HTTP Request

Required

URI

http://10.60.1.147:8080/job/hello-world/buildWithParameters?token=hello-world-token&DIFF\_ID=\${buildable.diff}&PHID=\${target.phid}

Required

HTTP Method

POST

填写名称，由于该 build step 的 action 就是发送一个 URLPOST 请求，这里使用 MakeHTTP Request，然后在下面的 URI 中填写之前 Jenkins 提示的，因为 Jenkins 根据 PHID 向 Phabricator 返回构建结果，这里需要传递 PHID 作为参数：

[http://<your-server>:8080/job/hello-world/buildWithParameters?token=hello-world-token&DIFF\\_ID=\\${buildable.diff}&PHID=\\${target.phid}](http://<your-server>:8080/job/hello-world/buildWithParameters?token=hello-world-token&DIFF_ID=${buildable.diff}&PHID=${target.phid})

在 HTTP Method 部分选择 POST，在 WhenComplete 中选择 WaitFor Message，等待 Jenkins 返回结果再进行下一个 build step。点击 SaveBuild Step 保存构建步骤。

在下面还列出了其它 Phabricator 可以传递给 Jenkins 的参数以及参数解析。

The following variables can be used in most fields. To reference a variable, use `${name}` in a field.

Variable	Description
build.id	The ID of the current build.
buildable.commit	The commit identifier, if applicable.
buildable.diff	The differential diff ID, if applicable.
buildable.revision	The differential revision ID, if applicable.
repository.callsign	The callsign of the repository in Phabricator.
repository.phid	The PHID of the repository in Phabricator.
repository.staging.ref	The ref name for this change in the staging repository.
repository.staging.uri	The URI of the staging repository.
repository.uri	The URI to clone or checkout the repository from.
repository.vcs	The version control system, either "svn", "hg" or "git".
step.timestamp	The current UNIX timestamp.
target.phid	The PHID of the current build target.

在 Phabricator 中添加 Herald 规则

到 <http://<your-server>/herald/new> 创建 Herald 规则：

可以针对多种情况创建 Herald 规则，例如 commit，diff 或者 revision 等。最主要的就是这三个。这里我们为 Differential Revisions 创建一个 Herald 规则，当新建或者更新 revision 时触发 build step 从而发出 HTTP POST 请求让 Jenkins 执行构建。

Herald > Create Rule

Create Herald Rule

New Rule for

Commits

React to new commits appearing in tracked repositories.  
Commit rules can send email, flag commits, trigger audits, and run build plans

Differential Diffs

React to new diffs being uploaded, before writes occur.  
These rules can reject diffs before they are written to permanent storage, to protect sensitive information.

Differential Revisions

React to revisions being created or updated.  
Revision rules can send email, flag revisions, add reviewers, and run build plan

Manifest Tasks

React to tasks being created or updated.



Rule Name 填写规则名称，在 Condition 部分添加条件，在 Action 部分添加满足上面的条件时需要执行的操作。例如当有 hello-world 库有 revision 创建或者更新，且 Author 是 luoyuanhao 时，执行 build plan，可以按照如下方式创建 Herald 规则：

**Edit Herald Rule**

Rule Name: Build hello-world

This Global rule triggers for Differential Revisions.

**Conditions**

When all of these conditions are met:

- Author is any of luoyuanhao
- Repository is any of rA hello-world

**Action**

Take these actions every time this rule matches:

- Run build plans Plan 1 Build hello-world

## 5. 创建 revision 触发 jenkins 构建

用户 luoyuanhao clone Phabricator 的 hello-world Repository，更新了代码并用 arc diff 创建 revision。

```
[root@nobida147 hello-world]# git checkout -b newbranch
Switched to a new branch 'newbranch'
[root@nobida147 hello-world]# echo "Add a line to readme" >> readme.md
[root@nobida147 hello-world]# git add .
[root@nobida147 hello-world]# git commit -m "add readme.md"
[newbranch 48e3a4f] add readme.md
1 files changed, 1 insertions(+), 0 deletions(-)
create mode 100644 readme.md
[root@nobida147 hello-world]# arc diff
```

```
1 add readme.md
2
3 Summary:add readme.md
4
5 Test Plan: run jenkins
6
7 Reviewers: testuser
8
9 Subscribers: luoyuanhao
0
```

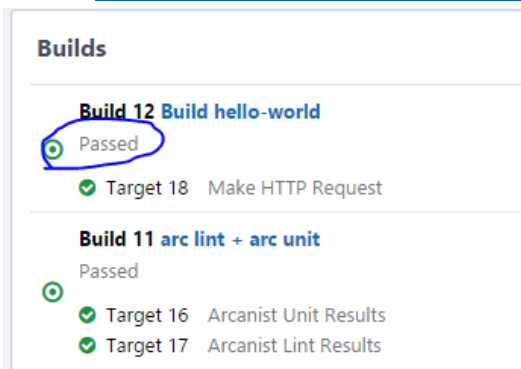
这时满足 Herald 规则 Runjenkins 的条件，就会 run build plan: build hello-world。build hello-world 其实就是向 Jenkins 发出一个带参数的 HTTP POST 请求。从而激活 Jenkins hello-world job。Jenkins hello-world 就从指定的 Git 库中拉取代码，并逐步执行 Build 部分指定的操作，

当所有 Build 部分都成功完成或者某个步骤出错时，进入 Post-build actions 部分，即将 build 结果返回给 Phabricator。

```
Started by remote host 10.60.1.147
Building in workspace /var/lib/jenkins/jobs/hello-world/workspace
> git rev-parse --is-inside-work-tree # timeout=10
Fetching changes from the remote Git repository
> git config remote.origin.url ssh://git@10.60.1.147/diffusion/A/hello-world
Fetching upstream changes from ssh://git@10.60.1.147/diffusion/A/hello-world.
> git --version # timeout=10
> git fetch --tags --progress ssh://git@10.60.1.147/diffusion/A/hello-world.
> git rev-parse refs/remotes/origin/master^{commit} # timeout=10
> git rev-parse refs/remotes/origin/origin/master^{commit} # timeout=10
Checking out Revision 840872684fcd1b19e7ef974b59715ee2e23aa68b (refs/remotes/
> git config core.sparsecheckout # timeout=10
> git checkout -f 840872684fcd1b19e7ef974b59715ee2e23aa68b
> git rev-list 840872684fcd1b19e7ef974b59715ee2e23aa68b # timeout=10
[workspace] $ /bin/sh -xe /tmp/hudson8889533578103163296.sh
+ echo 6
6
+ echo PHID-HMBT-5k3frjncrgrlajuskln
PHID-HMBT-5k3frjncrgrlajuskln
[phabricator:uberalls] No cobertura results found
[phabricator:process-build-result] No line coverage found, skipping...
[phabricator:process-build-result] No unit results available.
[phabricator:process-build-result] No coverage provider available.
[phabricator:harbormaster] Sending Harbormaster BUILD_URL via PHID: PHID-HMBT
[phabricator:process-build-result] Sending build result to Harbormaster with
[phabricator:comment-file] no files found by path: '.phabricator-comment'
Finished: SUCCESS
```

可以在 Jenkins Build 控制台输出查看构建的详细步骤。

可以在 <http://<your-server>/harbormaster/query/all> 处查看 Jenkins 返回的结果：



**Builds**

**Build 12 Build hello-world**

Passed

Target 18 Make HTTP Request

**Build 11 arc lint + arc unit**

Passed

Target 16 Arcanist Unit Results

Target 17 Arcanist Lint Results

## 四、 Reference

[Phabricator User Documentation](#)

[Use Jenkins](#)

[Jenkins The Definitive Guide](#)

[Jenkins Plugins](#)

[Phabricator Differential Plugin](#)