# Gerrit 配置使用教程

| 版本 | 修改内容 | 修订人 | 修订时间 |
| --- | --- | --- | --- |
| **V1.0** | 创建 | 赵守生 | **2011** 年 **7** 月 **29** 日 |
| | | | |
| | | | |
| | | | |
| | | | |

# 目　录

# 1. 快速配置

## 1.1 创建 apache 帐号名

产生一个登录帐号，ethanzhao

$htpasswd -c ethanzhao_account.txt ethanzhao

该命令创建了一个 ethanzhao_password.txt 的"用户:密码"对，用于登录 Gerrit。


请将此文件拷贝到\\192.168.1.24\public\gerrit_account 里，邮件通知系统管理员加入访问权限。目前请将发邮件给我 ethan.zhao@ebensz.com 负责加入，指出 apache2 帐号文件的位置。

## 1.2 创建 ssh 密钥

1） ssh-keygen -t rsa

一路回车后产生了 ssh 密钥，位于~/.ssh/id_rsa, id_rsa.pub

2） 创建~/.ssh/config，内容如下

```
Host et
    User ethanzhao      <-此名称必须与 gerrit 登录名保持一致。
    Port 29418
    Hostname 192.168.1.21
IdentityFile /home/ethan/.ssh/id_rsa    <-此为私钥实际的路径
```

其中 Host 为一个短称，可起易记字符，将来可以使用此名称代替 ssh://ethan@192.168.1.21 使用。

## 1.3 登录 Gerrit WebUI

打开浏览器，使用自己的帐号登录 http://192.168.1.21，弹出登录框后输入帐号密码登录。

## 1.4 注册邮件，加入公钥

1） 登录后，按下图注册 email 和添加自己的公钥。


这些信息都要可以后期从 Settings 菜单中后期加入，如：


2） 检查邮箱，你将收到一封由 gitadmin@ebensz.com 发送的 Gerrit 邮件帐号激活码，将其复制到浏览器后打开。

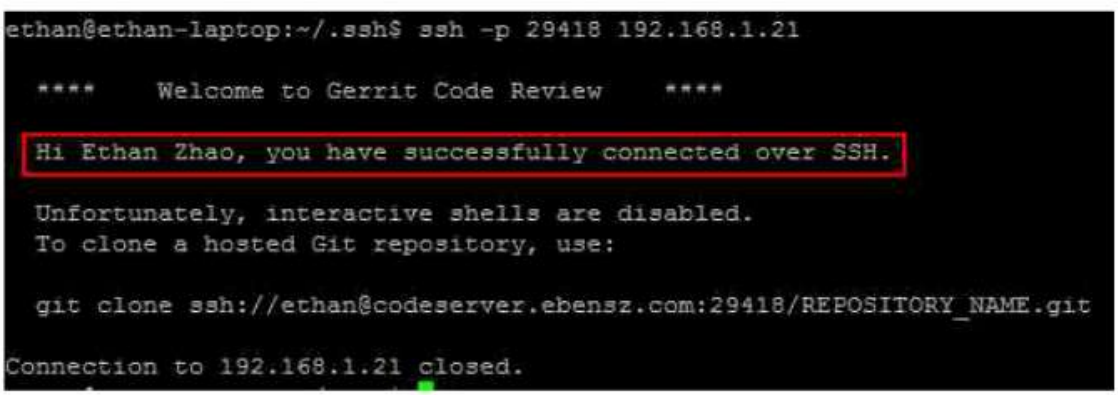
3） 此时你会看到右上角有 Anonymous Coward, :-)，原因是你未登记全名。


选择 Settings|Contact Information，填上命名，同时已激活的邮件已经在 Preferred Email 中列出了。


### 1.5 测试你的客户端

ssh -p 29418 192.168.1.21

如果你的公钥添加成功，你将获得如下的信息




### 1.6 查看项目和组

点击 Admin->Projects，便列出了当前 Gerrit 管理的 Projects 列表。


点击 Admin->Groups，显示所有的组，用户也可以在此创建一个新组。

Groups

| Group Name | Description |
|---|---|
| Administrators | Gerrit Site Administrators |
| Anonymous Users | Any user, signed-in or not |
| Developers | Users are the developer |
| Integrators | Member(s) of this Group has(have) the final say to approval changes. |
| Non-Interactive Users | Users who perform batch actions on Gerrit |
| Project Owners | Any owner of the project |
| Registered Users | Any signed-in user |

Create New Group

Create Group

## 2. 快速上手的操作流程

### 2.1 通过 SSH 获取一个项目

git clone ssh://192.168.1.21:29418/test.git

```
ethan@ethan-laptop:~$ git clone ssh://192.168.1.21:29418/test.git
Initialized empty Git repository in /home/ethan/test/.git/
remote: Counting objects: 222, done
remote: Finding sources: 100% (222/222)
remote: Total 222 (delta 9), reused 62 (delta 9)
Receiving objects: 100% (222/222), 31.85 KiB, done.
Resolving deltas: 100% (9/9), done.
```

### 2.2 拷贝 commit-msg 和 gerrit-cherry-pick

scp -p -P 29418 192.168.1.21:hooks/commit-msg .git/hooks

scp -p -P 29418 192.168.1.21:bin/gerrit-cherry-pick ~/bin

如果~/bin 未加入环境变更中，请修改.bashrc，加入 export PATH=~/bin:$PATH;

该 commit-msg 主要用来产生 Change-Id 行，详细的解释见下文。

### 2.3 开发并提交一个待评审的变更

$ cd test

$ echo a >test.c 或 vi test.c 进行更改。

$ git add test.c

$ git commit -am ' test.c added'

$ git push origin HEAD:refs/for/master

如果通知到给出评审者或相关人员，请使用

git push --receive-pack='git receive-pack --reviewer=gitadmin@ebensz.com --cc=ethan.zhao@ebensz.com' origin HEAD:refs/for/master，关于这一点，后面将解释并介绍一种比较好的办法来简化这个操作。

refs/for/{branch}是一个神奇的分支，所有提待评审的提交都是提交到这种分支上，如上句是提交到 master 上待评审，如果是 experiment 分支，则是 refs/for/experiment 如果使用

### 2.4 评审人及相关人收到邮件通知

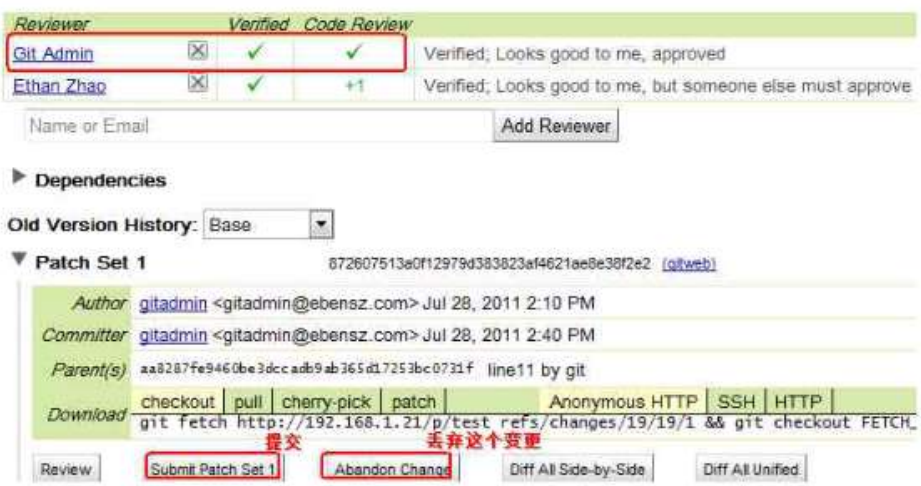

## 2.5 评审人或相关人评审变更

此时 gitadmin 登录 Gerrit WebUI，通过 All-Open，会查看到这个待评审的变更。点击该变更后显示出详情如下：


具有评审权限的人，点击 Review 按钮后进入评审界面：


## 2.6 提交变更

此时回到 Change 详情面时会有如下的 Verified 和 Code Review 的绿色勾选，则表示该变更审核通过，可以提交，只有有提交权限的人才可以提交变更。




## 2.7 冲突发生时

一个示例：

用户 git 修改 test.c 并完成一次提交，该提交待评审。

```
line4
line5 by git
```

$git commit -am 'line5 by git'

$git push origin HEAD:refs/for/master

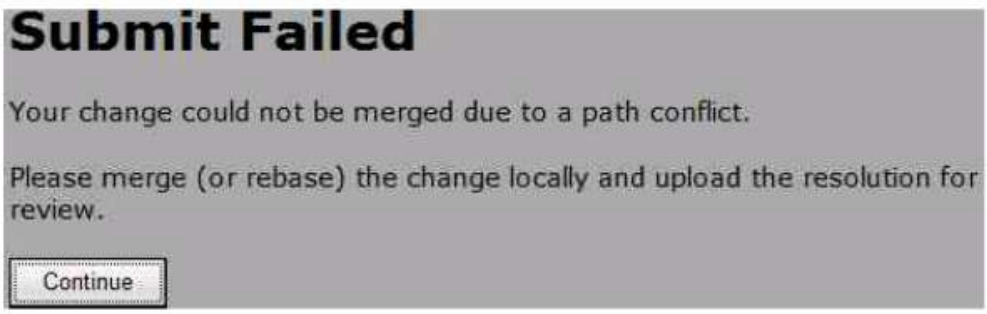
因 git 提交待评审，服务器仓库无变化，Ethan 与服务器更新时并不知晓 git 的变更。其修改文件如下，并也完成了一次提交。

line4
line5 by ethan

$git commit -am 'line5 by ethan'

$git push origin HEAD:refs/for/master

Reviewer 首先评审 git 的提交，通过后提交到了服务器上。后来再评审了 Ethan 的内容也没有问题，也准备提交，这时由于冲突，服务器拒绝并给出建议：




用户需要在客户端解决冲突(stash or rebase)，并重新提交。

# 3. 基本操作

## 3.1 创建一个项目到 Gerrit

### 3.1.1 客户端使用 ssh 创建

ssh -p 29418 ssh://192.168.1.21 gerrit create-project --name test --empty-commit

注意#--empty-commit 是必要的，否则无法使用 checkout master [bug?]

### 3.1.2 服务器本地创建方法

服务器端，使用 review_site 站点的户主（如 gerrit2）直接在仓库位置下创建 test2.git 仓库。

git --git-dir=/home/gerrit2/review_site/git/test2.git init

目前发现直接创建的项目在设置 Admin->Projects->Access 时会出现如下的访问错误


分析原因主要是此时创建的项目 test2.git 并未注册到 Gerrit 的数据库 reviewdb 上，此时点击 Admin->Projects->test2->Access 时会出现如下错误对话框，原因是未将 test2 注册到 reviewdb 数据里。

对于这个问题，Gerrit2.2.1 与 2.1 有不同的解决方法。

1） 2.1.7 以前版本：方法是使用一条 Insert 语句将其注册到 projects 表中。

- 当使使用内嵌 H2 数据库

```
ssh -p 29418 192.168.1.21 gerrit gsql
INSERT INTO projects
(use_contributor_agreements
 ,submit_type
 ,name)
VALUES
```

('N'
,'M'
,'test2');

- 当使使用内嵌 mysql 数据库时

  mysql -u root -p

  mysql> use reviewdb;

  mysql>

  INSERT INTO projects(project_id,use_contributor_agreements,submit_type,name) VALUES(nextval_project_id()),'N','M','./test2');

2） 2.2.1 有一个 bug，创建后不能使用 Admin|Projects|test2|Access 设权限，解决方法如下（参见 http://code.google.com/p/gerrit/issues/detail?id=1000）。

(1) All projects 里添加 refs/meta/config 的 Push 权限 （可选）

(2) 以建立 review_site 的用户身份 (gerrit2) 克隆：git clone /home/git/repositories/test2.git/

(3) 修改'.git/config' 加入 'fetch = +refs/meta/*:refs/meta/*'行到'[remote "origin"]'部分。

(4) 创建 project.config,内容为

[access "refs/*"]

owner = group Registered Users

(5) 创建文件 groups，内容为（注意 Registered-Users 和 Registered Users 之间为 Tab 而非空格）

global:Registered-Users Registered Users

(6) git add .; git commit -am 'add project access'

(7) git push origin HEAD:refs/meta/config

(8) ~/review_site/bin/gerrit.sh restart

(9) 登录 webUI，移除 Owner 为 Registered Users

可以用这种方法处理多个 project 的访问权限问题,如

git push /home/gerrit2/review_site/git/repotest/project1.git HEAD:refs/meta/config

git push /home/gerrit2/review_site/git/repotest/project2.git HEAD:refs/meta/config

## 3.2 查询变更

典型的查询变更界面是通过 All 菜单到达的。例如单击 All 下的 Merged，则搜索所有已经合并后变更。此项操作等同于在右上角搜索框内输入 status:merged 查询的操作。


### 3.2.1 基本搜索操作符

owner:"Ethan Zhao"

project:"test"

has:star (或者 is:starred)


is:watched：当变更与当前用户之一设置的 water 过滤器匹配时

Settings->Watched Projects:设置 test 为 watched,则输入“is:watched”可搜索出处理 watched 中的项目变更。

3.2.2 Boolean 操作符

可以使用“-”，“AND”，“OR”来确定搜索条件。

“-“ 取反，相当于 NOT 的意思，例如 “-is:starred”

OR 取或，例如：owner:"git" OR owner:"Ethan"

AND 取与，例如：owner:"Ethan Zhao" AND reviewer:"git"

3.2.3 Label 操作符

label:CodeReview=+2

label:Verified-1

label 操作符用于查找那些处于 code review 过程中的变更所获得的分数，根据服务器的配置可能有不同的标签，但 CodeReview 和 Verified 是两个默认怎不自带的标签。

## 3.3 上传变更

3.3.1 Change-Id

Gerrit 提供了一个 commit-msg hook，每次当 git commit 时它都会自动创建并插入一个独一无二的 Change-Id 行。其安装方法如前述。

scp -p -P 29418 192.168.1.21:hooks/commit-msg .git/hooks

安装成功后，每次提交后的 log 如下，多出一个 Change-Id 的行

```
git@codeserver:~/test$ git log -1
commit 17946c10693dbb13c7f8b9c34cc99058ecf78680
Author: gitadmin <gitadmin@ebensz.com>
Date:   Thu Jul 28 14:10:12 2011 +0800

    line12 by git

    Change-Id: Iff9877290841866b5f3c7c3d202f7d074f4afb4c
```

Gerrit 识别该 Change-Id， Gerrit 会自动更新相同的 Change-Id 的提交内容（amend, rebase，squash，cherry-pick）

Amending a commit：使用 git commit －amend 提交时保留 Change-Id 不变，Gerrit 会自动更新该提交内容

Rebasing a commit：Rebase 一个提交时，保留 Change-Id 不变，Gerrit 会自动更新使用 rebased 提交

Squashing commits：压缩多个提交到一起，保留其中一个 Change-Id 行，Gerrit 会识别出来。用户需要在 Gerrit Web 界面里手动丢弃其它的变更。

Cherry-picking a commit：当进行 cherry pick 时，保留 Change-Id，这样 Gerrit 视 cherry-picked commit 为一个已有变更的替换，这对于某个项目为 fast-forward-only 合并策略会非常有用。

3.3.2 git push 提交待审核的变更

git push 创建待审核的变更到到 Gerrit：这种方式的操作是 git push URL refs/for/'branch'

直接提交变更到仓库（Bypass Gerrit）

### 3.3.3 git push 直接提交变更到仓库

git push 可以绕过审核（Bypass review）以达到创建分支、标签和提交等

refs/heads/*： 直接 push 创建分支。push 用户需要打开 Push 分支权限

refs/tags/*:直接 push 创建标签。push 用户需要打开 Push 标签权限 Push Annotated Tag

refs/*：直接 push 提交

### 3.3.4 使用 repo upload 创建待审核变更

## 3.4 命令行工具

git upload-pack：Git 服务器端命令，当客户端使用 git fetch 时，就连接到服务器端的 upload-pack 进程。

git receive-pack：Git 服务器端命令，当客户端使用 git push 时，连接到远端的 receiver-pack 协商通信

gerrit approve（或者 gerrit review）:验证，同意和/或提交一个一个补丁集

gerrit ls-projects：查询所有使用者可以看到的 project

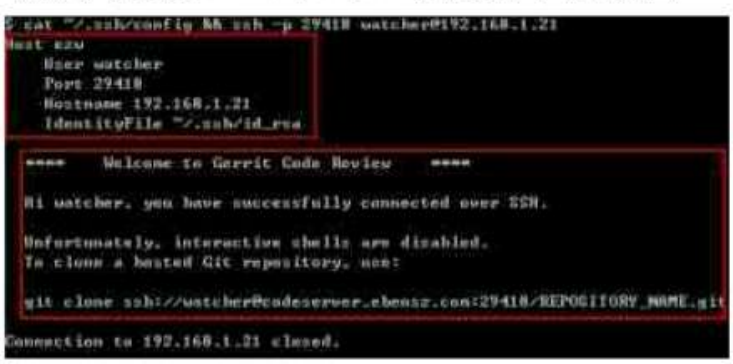
## 3.5 管理员工具

gerrit create-account：创建一个新的内部用户帐号。

例如 cat ~/ez_watcher.pub | ssh -p 29418 192.168.1.21 gerrit create-account --ssh-key – watcher

这样使用 ez_watcher.pub 创建了一个用户 watcher。

在客户端测试（先要建立一个~/.ssh/config 的文件 ，内容见图）




但使用 HTTP 鉴权情况下，该方式创建的帐号能获取代码，进行开发。但沿无法登录 HTTP 界面，因为 apache2 使用自己的用户帐号。

gerrit create-project：创建一个 Gerrit 管理的项目

ssh -p 29418 ssh://192.168.1.21 gerrit create-project --name test --empty-commit

gerrit create-group：创建一个组。下面的示例创建一个 Integrators 的组，包含成员 git 和 Ethan

ssh -p 29418 192.168.1.21 gerrit create-group --member git --member Ethan Integrators

gerrit gsql：已激活的数据库管理接口，可执行包括 SELECT, UPDATE, INSERT, DELETE 和 ALTER 等所有的 SQL 语句。例如 ssh git gerrit gsql（即 ssh -p 29418 git@192.168.1.21 gerrit gsql）

gerrit show-connections: 显示 ssh -p 29418 git@192.168.1.21 gerrit show-connections

```
git@codeserver:~/test$ ssh -p 29418 git@192.168.1.21 gerrit show-connections
Session     Start     Idle   User            Remote Host
-------------------------------------------------------------------
9d335ee1 12:02:25 00:00:00  git             codeserver.ebenssz.com
--
```

## 3.6 权限控制（主要译自官方文档）

Gerrit 的权限控制是基于组策略的。每个用户隶属于一个或多个组。Gerrit 权限不对用户直接开放。

1）系统组

Administrators：该组内用户可以执行 Admin 菜单下的 Project 和 Group 所有设置行为。该组内用户并不直接拥有项目代码审核批准或提交权限。在这一点上，它与其它通用组里的用户没什么两样。

Anonymous Users：所有用户都自动为该组成员，未登录用户也属于该组。所有授权给该组的权限将被所有用户继承。Administrators 或项目 Owner 可以授予访问权限给这个组，以使该组内成员可以不必要登录即可查看变更变化。通常仅仅授权为 Read。

Registered Users：所有登录的用户均属于该组。当使用 OpenID 身份验证时，用户变成 Registered Users 非常容易，因此授权给这个组时应该要小心。典型的做法是将“Code Review -1..+1”分配给该组。允许他们对变更发表意见（vote a change），但不具备接受或拒绝一个变更的权限。Registered Users 对于所有有 Read 权限的项目的任何变更都有发表评论的权力。

Project Owners:被授予该组的访问权限总是在一个项目的上下文内被评估并转换成所有拥用该项目的用户的权限。

2）权限控制列表

用户属于多个组，则享受最大化权利的原则。

权限设置可以是一个常规的引用名空间或通配符表示的多个引用名空间，如 refs/heads/master 仅匹配 master 分支，或者 refs/heads/* 则匹配多个分支（refs/heads/master 和 refs/heads/test 等）。

引用名字也可以是个正式表达式（前缀\^），如\^refs/heads/[a-z]\{1,8\}表示匹配所有名称为小写且长度在 1~8 个字符之间的分支。“.”可以匹配任何单个字符。

引用名还可以自动包括当前用户，实现动态权限去匹配当前登录的用户。例如 refs/heads/sandbox/$\{username\}/*，允许用户 joe 使用 refs/heads/sandbox/joe/foo。

Gerrit 使用”访问权限所有集”来评估一个用户的引用级访问，来决定其的访问权限。如下表 Foo Leads 未被明确授予但仍然扔有 refs/heads/qa 的 Code Review 权限，（因为 refs/heads/*）

| Group | Reference Name | Category | Range |
|---|---|---|---|
| Registered Users | refs/heads/* | Code Review | -1..+1 |
| Foo Leads | refs/heads/* | Code Review | -2..+2 |
| QA Leads | refs/heads/qa | Code Review | -2..+2 |

Gerrit 也可以支持排它型独占访问控制，方法是前缀“-”号，如下表，Foo Leads 组的人将无法拥有“refs/heads/qa”的 Code Review 权限。

| Group | Reference Name | Category | Range |
|---|---|---|---|

| Group | Reference Name | Category | Range |
|---|---|---|---|
| Registered Users | refs/heads/* | Code Review | -1..+1 |
| Foo Leads | refs/heads/* | Code Review | -2..+2 |
| QA Leads | -refs/heads/qa | Code Review | -2..+2 |

为了使得 Foo Leads 组的成员有“refs/heads/qa”的 Code Review 权限，需要特别的增加一行，如下表

| Group | Reference Name | Category | Range |
|---|---|---|---|
| Registered Users | refs/heads/* | Code Review | -1..+1 |
| Foo Leads | refs/heads/* | Code Review | -2..+2 |
| QA Leads | -refs/heads/qa | Code Review | -2..+2 |
| Foo Leads | refs/heads/qa | Code Review | -2..+2 |

3）All-Projects 项目：

All Projects：该项目为 Gerrit 创建，默认为是所有后建项目的父项目，其默认权限如下。根据子项目继承父项目权限的规则。这些权限被后期建立的所有的项目继承。

只有 Administrators 组内成员可以修改这个项目的权限控制。

该项目的所有权不能被授予给其它组。否则被授权组将近乎拥有和 Administrators 组员相同的访问权限。默认的 All-Projects 权限如下：


## 3.7 变更提交评审时邮件通知

1）命令行方式

git push --receive-pack='git receive-pack --reviewer=gitadmin@ebensz.com --cc=ethan.zhao@ebensz.com' origin HEAD:refs/for/master

2）配置.git/config 内容如下：

```
[remote "origin"]
    fetch = +refs/heads/*:refs/remotes/origin/*
    url = ssh://192.168.1.21:29418/test.git
        receivepack = git receive-pack --reviewer=gitadmin@ebensz.com --cc=ethan.zhao@ebensz.com
    push = HEAD:refs/for/master
```

[branch "master"]
remote = origin
merge = refs/heads/master
即可使用“git push origin”完成提交

**Enjoy Gerrit...** ☺