

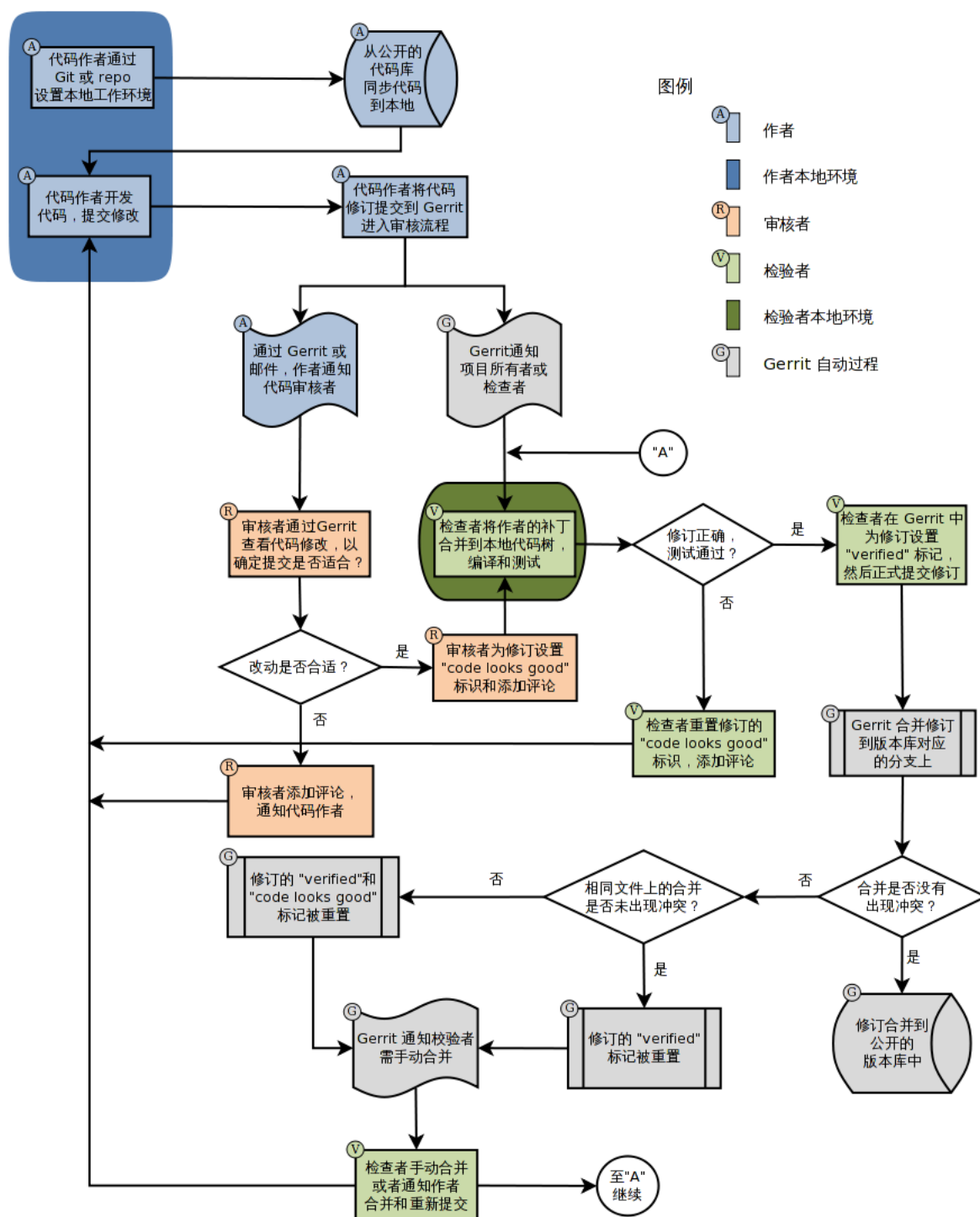
第32章 Gerrit 代码审核服务器

谷歌 Android 开源项目在 Git 的使用上有两个重要的创新，一个是为多版本库协同而引入的 repo，这在前面第 25 章已经详细讨论过。另外一个重要的创新就是 Gerrit —— 代码审核服务器。Gerrit 为 Git 引入的代码审核是强制性的，也就是说除非特别的授权设置，向 Git 版本库的推送必须要经过 Gerrit 服务器，修订必须经过代码审核的一套工作流之后，才可能经批准并纳入正式代码库中。

首先贡献者的代码通过 git 命令（或 repo 封装）推送到 Gerrit 管理下的 Git 版本库，推送的提交转化为一个一个的代码审核任务，审核任务可以通过 refs/changes/ 下的引用访问到。代码审核者可以通过 Web 界面查看审核任务、代码变更，通过 Web 界面做出通过代码审核或打回等决定。测试者也可以通过 refs/changes/ 之下的引用获取修订然后对其进行测试，如果测试通过就可以将该评审任务设置为校验通过（verified）。最后经过了审核和校验的修订可以通过 Gerrit 界面中的提交动作合并到版本库对应的分支中。

Android 项目网站上有一个代码贡献流程图¹，详细地介绍了 Gerrit 代码审核服务器的工作流程。翻译后的工作流程图见图 32-1。

¹ <http://source.android.com/source/life-of-a-patch.html>



摘自：<http://source.android.com/source/life-of-a-patch.html>

图 32-1: Gerrit 代码审核 workflow

32.1 Gerrit 的实现原理

Gerrit 更准确地说应该称为 Gerrit2。因为 Android 项目最早使用的评审服务器 Gerrit 不是今

天这个样子的。最早版本的 Gerrit 是用 Python 开发运行于 Google App Engine 上的，从 Python 之父 Guido van Rossum 开发的 Rietveld 分支而来。在这里要讨论的 Gerrit 实为 Gerrit2，是用 Java 语言实现的¹。

1. SSH 协议的 Git 服务器

Gerrit 本身基于 SSH 协议实现了一套 Git 服务器，这样就可以对 Git 数据推送进行更为精确的控制，为强制审核的实现建立了基础。

Gerrit 提供的 Git 服务的端口并非标准的 22 端口，默认是 29418 端口。这个端口是可以被发现的，当访问 Gerrit 的 Web 界面时可以得到这个端口。对 Android 项目的代码审核服务器，访问 https://review.source.android.com/ssh_info 就可以查看到 Git 服务的服务器域名和开放的端口。下面用 curl 命令查看网页的输出。

```
$ curl -L -k http://review.source.android.com/ssh_info
review.source.android.com 29418
```

2. 特殊引用 refs/for 和 refs/changes

Gerrit 的 Git 服务器，禁止用户向 refs/heads 命名空间下的引用执行推送（除非特别的授权），即不允许用户直接向分支进行提交。为了让开发者能够向 Git 服务器提交修订，Gerrit 的 Git 服务器只允许用户向特殊的引用 refs/for/<branch-name> 下执行推送，其中 <branch-name> 即为开发者的工作分支。向 refs/for/<branch-name> 命名空间下推送并不会在其中创建引用，而是为新的提交分配一个 ID，称为 review-id，并为该 review-id 的访问建立如下格式的引用 refs/changes/nn/<review-id>/m，其中：

- ❑ review-id 是 Gerrit 为评审任务顺序而分配的全局唯一的号码。
- ❑ nn 为 review-id 的后两位数，位数不足用零补齐。即 nn 为 review-id 除以 100 的余数。
- ❑ m 为修订号，该 review-id 的首次提交修订号为 1，如果该修订被打回，重新提交修订号会自增。

¹ <http://code.google.com/p/gerrit/wiki/Background>

3. Git 库的钩子脚本 hooks/commit-msg

为了保证已经提交审核的修订通过审核入库后，如果被别的分支拣选（cherry-pick）后再推送至服务器时不会产生新的重复的评审任务，Gerrit 设计了一套特殊的方法，即要求每个提交在提交说明中包含 `Change-Id` 键值对作为标签，该标签在首次生成时使用特殊的哈希算法以保障其唯一性。执行拣选操作时，提交说明会被保持，即来自原始提交说明中的 `Change-Id` 键值对也会保持不变，这样当新提交推送到 Gerrit 服务器时，Gerrit 会发现新的提交包含了已经处理过的 `Change-Id`，就不再为该修订创建新的评审任务和 `review-id`，而直接将提交入库。

为了使得 Git 提交中包含唯一的 `Change-Id`，Gerrit 提供了一个钩子脚本，将该脚本拷贝到开发者的本地 Git 库的钩子脚本目录中，即脚本文件 `.git/hooks/commit-msg`。这个钩子脚本在用户提交时，自动在提交说明中创建 `Change-Id` 键值对。至于如何实现 `Change-Id` 值的唯一性，可以参考该脚本。

当 Gerrit 获取到用户向 `refs/for/<branch-name>` 推送的提交中包含 “`Change-Id: I...`” 的格式时，如果该 `Change-Id` 之前没有见过，会创建一个新的评审任务并分配新的 `review-id`，并在 Gerrit 的数据库中保存 `Change-Id` 和 `review-id` 的关联。

如果用户的提交因为某种原因被打回重做，开发者修改之后重新推送到 Gerrit 时就要注意在提交说明中使用相同的 `Change-Id`（使用 `--amend` 提交即可保持提交说明），以免创建新的评审任务。还要在推送时将当前分支推送到 `refs/changes/<nn>/<review-id>/<m>` 中，是为该评审任务的一个新的修订，其中 `<nn>` 和 `<review-id>` 和之前提交的评审任务的修订号相同，`<m>` 则要人工选择一个新的修订号。

以上说起来很复杂，但是在实际操作中只要使用 `repo` 这一工具，就相对容易多了。

4. 其余一切交给 Web

Gerrit 另外一个重要的组件就是 Web 服务器，通过 Web 服务器实现对整个评审工作流的控制。关于 Gerrit 工作流，请参见本章开头出现的 Gerrit 工作流程图。

想要感受一下 Gerrit 的魅力？请直接访问 Android 项目的 Gerrit 网站：

<https://review.source.android.com/>，您会看到如图 32-2 的界面。

AllDocumentationOpenMergedAbandoned

status:merged

Search

RegisterSign In

open source project

Search for status:merged

| ID | Subject | Owner | Project | Branch | Updated | V | R |
|-----------|---|-----------------|----------------------------------|----------------------|---------|---|---|
| I76d7fcf7 | ARM: tegra: usb_phy: Correct utmi power off sequence (MERGED) | Benoit Goby | kernel/tegra | linux-tegra-2.6.36 | 9:05 AM | ✓ | ✓ |
| Ibf3ec6a0 | Add transient visibility mode for empty containers (MERGED) | Tor Norbye | platform/sdk | master | 8:38 AM | ✓ | ✓ |
| I1ee128e0 | Fix ADT to build with the new layoutlib API. (MERGED) | Xavier Ducrohet | platform/sdk | master | 8:21 AM | ✓ | ✓ |
| Ice6324c0 | New layoutlib API. (MERGED) | Xavier Ducrohet | platform/sdk | master | 8:07 AM | ✓ | ✓ |
| If976cc25 | ARM: tegra: dvfs: Fix dvfs disable config option (MERGED) | Colin Cross | kernel/tegra | linux-tegra-2.6.36 | 7:54 AM | ✓ | ✓ |
| I0496cf37 | ARM: tegra: dvfs: Add lock to dvfs_reg (MERGED) | Colin Cross | kernel/tegra | linux-tegra-2.6.36 | 7:43 AM | ✓ | ✓ |
| I9e3a3cc8 | ARM: tegra: dvfs: Fix locking on external dvfs calls (MERGED) | Colin Cross | kernel/tegra | linux-tegra-2.6.36 | 7:07 AM | ✓ | ✓ |
| I401ab558 | ARM: tegra: dvfs: Add config options to disable dvfs (MERGED) | Colin Cross | kernel/tegra | linux-tegra-2.6.36 | 7:07 AM | ✓ | ✓ |
| I643375c3 | fix getDeviceIdentity and getImeiSV for f5521fg modules. (MERGED) | Johan Lindström | platform/vendor/st-ericsson/u300 | ericsson-mbm-devices | 6:53 AM | ✓ | ✓ |
| Iff0473dc | [ARM] tegra: cleanup empty functions in mach/tpb.h (MERGED) | Erik Gilling | kernel/tegra | linux-tegra-2.6.36 | 6:05 AM | ✓ | ✓ |
| Id3579a09 | video: add short video mode decode to fblmon | Erik Gilling | kernel/tegra | linux-tegra-2.6.36 | 6:05 AM | ✓ | ✓ |

图 32-2: Android 项目代码审核网站

点击菜单中的 “Merged” 即可显示已经通过评审且合并到代码库中的审核任务。图 32-3 中显示的就是 Andorid 一个已经合并到代码库中的历史评审任务。

Change I51bcb5b3: Handle instrumentation time output that contains a bracket.

android.comhttps://review.source.android.com/#change,16993

Change I51bcb5b3: Handle in...

open source project

Change I51bcb5b3: Handle instrumentation time output that contains a bracket.

Change-Id: I51bcb5b3aaa1520b25519b0b8b4679691c4b7fe

Owner: Brett Chabot

Project: platform/sdk

Branch: master

Topic:

Uploaded: Sep 4, 2010 3:57 AM

Updated: Sep 6, 2010 6:25 AM

Status: Merged

Handle instrumentation time output that contains a bracket.

Bug 2975380

Change-Id: I51bcb5b3aaa1520b25519b0b8b4679691c4b7fe

Permalink

Reviewer

Verified

Code Review

Brett Chabot

✓

Verified

Omari Stephens

+1

Looks good to me, but someone else must approve

Xavier Ducrohet

✓

Looks good to me, approved

Included in

Dependencies

Patch Set 1

5fb1e79b01166519211c5603c5106ab023d

gitweb

Patch Set 2

d042a5b41f07cd020bc26a2b6745b7cd95d5a7

gitweb

Author

Brett Chabot <bretchabot@android.com>

Sep 4, 2010 3:58 AM

Committer

Brett Chabot <bretchabot@android.com>

Sep 4, 2010 3:58 AM

repo download

checkout

pull

cherry-pick

patch

Download

repo download platform/sdk 16993/2

图 32-3: Android 项目的 16993 号评审

从图 32-3 中可以看出：

- URL 中显示的评审任务编号为 16993。

Git 权威指南——自排版

5

- ❑ 该评审任务的 `Change-Id` 以字母 `I` 开头，包含了一个唯一的 40 位 SHA1 哈希值。
- ❑ 整个评审任务有三个人参与，一个人进行了检查（`verify`），两个人进行了代码审核。
- ❑ 该评审任务的状态为已合并：“`merged`”。
- ❑ 该评审任务总共包含两个补丁集：`Patch set 1` 和 `Patch set 2`。
- ❑ 补丁集的下载方法是：`repo download platform/sdk 16993/2`。

如果使用 `repo` 命令获取补丁集是非常方便的，因为封装后的 `repo` 屏蔽掉了 Gerrit 的一些实现细节，例如补丁集在 Git 库中的存在位置。如前所述，补丁集实际保存在 `refs/changes` 命名空间下。使用 `git ls-remote` 命令，从 Gerrit 维护的代码库中可以看到补丁集对应的引用名称。

```
$ git ls-remote \
    ssh://review.source.android.com:29418/platform/sdk \
    refs/changes/93/16993*
5fb1e79b01166f5192f11c5f509cf51f06ab023d      refs/changes/93/16993/1
d342ef5b41f07c0202bc26e2bfff745b7c86d5a7      refs/changes/93/16993/2
```

接下来就来介绍一下 Gerrit 服务器的部署和使用方法。

32.2 架设 Gerrit 的服务器

1. 下载 war 包

Gerrit 是由 Java 开发的，被封装为一个 war 包：`gerrit.war`，安装非常简洁。如果需要从源码编译出 war 包，可以参照相关文档¹。不过最简单的就是从 Google Code 上直接下载编译好的 war 包。

从下面的地址下载 Gerrit 的 war 包：

<http://code.google.com/p/gerrit/downloads/list>。在下载页面会有一个文件名类似

`Gerrit-x.x.x.war` 的 war 包，这个文件就是 Gerrit 的全部。示例中使用的是 2.1.5.1 版本，把下载的 `Gerrit-2.1.5.1.war` 包重命名为 `Gerrit.war`。下面的介绍就是基于这个版本。

2. 数据库选择

Gerrit 需要数据库来维护账户信息、跟踪评审任务等。目前支持的数据库类型有 PostgreSQL、

¹ <http://gerrit.googlecode.com/svn/documentation/2.1.5/dev-readme.html>

MySQL 及嵌入式的 H2 数据库。

选择使用默认的 H2 内置数据库是最简单的，因为这样无须任何设置。如果想使用更为熟悉的 PostgreSQL 或 MySQL，则需要预先建立数据库。

对于 PostgreSQL，在数据库中创建一个用户 `gerrit`，并创建一个数据库 `reviewdb`。

```
createuser -A -D -P -E gerrit
createdb -E UTF-8 -O gerrit reviewdb
```

对于 MySQL，在数据库中创建一个用户 `gerrit` 并为其设置口令（不要真如下面那样将口令设置为 “secret”），并创建一个数据库 `reviewdb`。

```
$ mysql -u root -p

mysql> CREATE USER 'gerrit'@'localhost' IDENTIFIED BY 'secret';
mysql> CREATE DATABASE reviewdb;
mysql> ALTER DATABASE reviewdb charset=latin1;
mysql> GRANT ALL ON reviewdb.* TO 'gerrit'@'localhost';
mysql> FLUSH PRIVILEGES;
```

3. 以一个专用用户帐号执行安装

在系统中创建一个专用的用户帐号如：`gerrit`。以该用户身份执行安装，将 Gerrit 的配置文件、内置数据库、`war` 包等都自动安装在该用户主目录下的特定目录中。

```
$ sudo adduser gerrit
$ sudo su gerrit
$ cd ~gerrit
$ java -jar gerrit.war init -d review_site
```

在安装过程中会提出一系列问题。

❑ 创建相关目录。

默认 Gerrit 在安装用户主目录下创建 `review_site`，并把相关文件安装在这个目录之下。

Git 版本库的根路径默认位于此目录之下的 `git` 目录中。

```
*** Gerrit Code Review 2.1.5.1
***

Create '/home/gerrit/review_site' [Y/n]?

*** Git Repositories
***
```

```
Location of Git repositories [git]:
```

❑ 选择数据库类型。

选择 H2 数据库是简单的选择，无须额外的配置。

```
*** SQL Database
***

Database server type [H2/?]:
```

❑ 设置 Gerrit Web 界面认证的类型。

默认为 openid，即使用任何支持 OpenID 的认证源（如 Google、Yahoo!）进行身份认证。此模式支持用户自建帐号，用户通过 OpenID 认证源的认证后，Gerrit 会自动从认证源获取相关属性如用户全名和邮件地址等信息创建帐号。Android 项目的 Gerrit 服务器即采用此认证模式。

如果有可用的 LDAP 服务器，那么 ldap 或 ldap_bind 也是非常好的认证方式，可以直接使用 LDAP 中的已有帐号进行认证，不过此认证方式下 Gerrit 的自建帐号功能是关闭的。此安装示例选择的的就是 LDAP 认证方式。

HTTP 认证也是可选的认证方式，此认证方式需要配置 Apache 的反向代理，并在 Apache 中配置 Web 站点的口令认证，通过口令认证后 Gerrit 在创建帐号的过程中会询问用户的邮件地址并发送确认邮件。

```
*** User Authentication
***

Authentication method [OPENID/?]: ?
Supported options are:
  openid
  http
  http_ldap
  ldap
  ldap_bind
  development_become_any_account
Authentication method [OPENID/?]: ldap
LDAP server [ldap://localhost]:
LDAP username :
Account BaseDN : dc=foo,dc=bar
Group BaseDN [dc=foo,dc=bar]:
```

❑ 发送邮件设置。

默认使用本机的 **SMTP** 发送邮件。

```
*** Email Delivery
***

SMTP server hostname      [localhost]:
SMTP server port          [(default)]:
SMTP encryption           [NONE/?]:
SMTP username             :
```

❑ Java 相关设置。

使用 **OpenJava** 和 **Sun Java** 均可。**Gerrit** 的 **war** 包要复制到 **review_site/bin** 目录中。

```
*** Container Process
***

Run as                    [gerrit]:
Java runtime              [/usr/lib/jvm/java-6-sun-1.6.0.21/jre]:
Copy gerrit.war to /home/gerrit/review_site/bin/gerrit.war [Y/n]?
Copying gerrit.war to /home/gerrit/review_site/bin/gerrit.war
```

❑ SSH 服务相关设置。

Gerrit 的基于 **SSH** 协议的 **Git** 服务非常重要，默认的端口为 **29418**。换成其他端口也无妨，因为 **repo** 可以自动探测到该端口。

```
*** SSH Daemon
***

Listen on address         [*]:
Listen on port            [29418]:

Gerrit Code Review is not shipped with Bouncy Castle Crypto v144
  If available, Gerrit can take advantage of features
  in the library, but will also function without it.
Download and install it now [Y/n]?
Downloading http://www.bouncycastle.org/download/bcprov-jdk16-144.jar ...
OK
Checksum bcprov-jdk16-144.jar OK
Generating SSH host key ... rsa... dsa... done
```

❑ HTTP 服务相关设置。

默认启用内置的 **HTTP** 服务器，端口为 **8080**，如果该端口被占用（如 **Tomcat**），则需要更换为其他端口，否则服务启动失败。如下例就换成了 **8081** 端口。

```
*** HTTP Daemon
```

```

***

Behind reverse proxy      [y/N]? y
Proxy uses SSL (https://) [y/N]? y
Subdirectory on proxy server [/]: /gerrit
Listen on address         [*]:
Listen on port            [8081]:
Canonical URL              [https://localhost/gerrit]:

Initialized /home/gerrit/review_site

```

4. 启动 Gerrit 服务

Gerrit 服务正确安装后，运行 Gerrit 启动脚本来启动 Gerrit 服务。

```

$ /home/gerrit/review_site/bin/gerrit.sh start
Starting Gerrit Code Review: OK

```

服务正确启动之后，会看到 Gerrit 服务打开两个端口，这两个端口是在 Gerrit 安装时指定的。您的输出和下面的示例可能略有不同。

```

$ sudo netstat -ltnp | grep -i gerrit
tcp        0      0 0.0.0.0:8081          0.0.0.0:*            LISTEN
26383/GerritCodeRev
tcp        0      0 0.0.0.0:29418         0.0.0.0:*            LISTEN
26383/GerritCodeRev

```

5. 设置 Gerrit 服务开机自动启动

Gerrit 服务的启动脚本支持 start、stop、restart 参数，可以作为 init 脚本开机自动执行。

```

$ sudo ln -snf \
    /home/gerrit/review_site/bin/gerrit.sh \
    /etc/init.d/gerrit.sh
$ sudo ln -snf ../init.d/gerrit.sh /etc/rc2.d/S90gerrit
$ sudo ln -snf ../init.d/gerrit.sh /etc/rc3.d/S90gerrit

```

服务自动启动脚本 /etc/init.d/gerrit.sh 需要通过 /etc/default/gerritcodereview 提供一些默认的配置。以下面的内容来创建该文件。

```

GERRIT_SITE=/home/gerrit/review_site
NO_START=0

```

6. Gerrit 认证方式的选择

如果是开放的 Gerrit 服务，使用 OpenId 认证是最好的方法，就像谷歌 Android 项目的代码审核服务器配置的那样。任何人只要在可以作为 OpenId 提供者的网站上（如 Google、Yahoo! 等）拥有帐号，就可以直接通过 OpenId 注册，Gerrit 会在用户登录 OpenId 提供者网站成功后，自动获取（经过用户的确认）用户在 OpenId 提供者站点上的部分注册信息（如用户全名或邮件地址）在 Gerrit 上自动为用户创建帐号。

如果架设有 LDAP 服务器，并且用户帐号都在 LDAP 中进行管理，那么采用 LDAP 认证也是非常好的方法。登录时提供的用户名和口令通过 LDAP 服务器验证之后，Gerrit 会自动从 LDAP 服务器中获取相应的字段属性为用户创建帐号。创建帐号的用户全名和邮件地址因为来自于 LDAP，因此不能在 Gerrit 更改，但是用户可以注册新的邮件地址。我在配置 LDAP 认证时遇到了一个问题就是创建帐号的用户全名是空白的，这是因为没有正确设置 LDAP 的相关字段。如果 LDAP 服务器使用的是 OpenLDAP，Gerrit 会从 displayName 字段获取用户全名，如果使用 Active Directory 则用 givenName 和 sn 字段的值拼接形成用户全名。

Gerrit 还支持使用 HTTP 认证，这种认证方式需要架设 Apache 反向代理，在 Apache 中配置 HTTP 认证。用户若要访问 Gerrit 网站，首先需要通过 Apache 配置的 HTTP Basic Auth 认证，当 Gerrit 发现用户已经登录后，会要求用户确认邮件地址。当用户确认邮件地址后，再填写其他必须的字段完成帐号注册。HTTP 认证方式的缺点除了在口令文件管理上需要管理员手工维护比较麻烦之外，还有一个缺点就是用户一旦登录成功后，想退出登录或更换其他用户帐号登录会变得非常麻烦，除非关闭浏览器。关于用户切换有一个小窍门：例如 Gerrit 登录 URL 为 `https://server/gerrit/login/`，则用浏览器访问 `https://nobody:wrongpass@server/gerrit/login/`，即用错误的用户名和口令覆盖掉浏览器缓存的认证用户名和口令，这样就可以重新认证了。

在后面的 Gerrit 演示和介绍中，为了设置帐号的方便，使用了 HTTP 认证，因此下面再介绍一下 HTTP 认证的配置方法。

7. 配置 Apache 代理访问 Gerrit

默认 Gerrit 的 Web 服务端口为 8080 或 8081，通过 Apache 的反向代理就可以使用标准的 80（HTTP）或 443（HTTPS）来访问 Gerrit 的 Web 界面。

```
ProxyRequests Off
ProxyVia Off
ProxyPreserveHost On

<Proxy *>
    Order deny,allow
    Allow from all
</Proxy>

ProxyPass /gerrit/ http://127.0.0.1:8081/gerrit/
```

如果要配置 Gerrit 的 HTTP 认证,则还需要在上面的配置中插入 HTTP Basic Auth 认证的设置。

```
<Location /gerrit/login/>
    AuthType Basic
    AuthName "Gerrit Code Review"
    Require valid-user
    AuthUserFile /home/gerrit/review_site/etc/gerrit.passwd
</Location>
```

在上面的配置中,指定了口令文件的位置:

/home/gerrit/review_site/etc/gerrit.passwd。可以用 htpasswd 命令维护该口令文件。

```
$ touch /home/gerrit/review_site/etc/gerrit.passwd
$ htpasswd -m /home/gerrit/review_site/etc/gerrit.passwd jiangxin
New password:
Re-type new password:
Adding password for user jiangxin
```

至此为止, Gerrit 服务安装完成。在正式使用 Gerrit 之前,先来研究一下 Gerrit 的配置文件,以免安装过程中遗漏或因错误的设置而影响使用。

32.3 Gerrit 的配置文件

Gerrit 的配置文件保存在部署目录下的 etc/gerrit.conf 文件中。如果对安装时的配置不满意,可以手工修改配置文件,重启 Gerrit 服务即可。

全部采用默认配置时的配置文件:

```
[gerrit]
    basePath = git
    canonicalWebUrl = http://localhost:8080/
```

```
[database]
    type = H2
    database = db/ReviewDB
[auth]
    type = OPENID
[sendemail]
    smtpServer = localhost
[container]
    user = gerrit
    javaHome = /usr/lib/jvm/java-6-openjdk/jre
[sshd]
    listenAddress = *:29418
[httpd]
    listenUrl = http://*:8080/
[cache]
    directory = cache
```

如果采用 **LDAP** 认证，下面的配置文件片断配置了一个支持匿名绑定的 **LDAP** 服务器配置。

```
[auth]
    type = LDAP
[ldap]
    server = ldap://localhost
    accountBase = dc=foo,dc=bar
    groupBase = dc=foo,dc=bar
```

如果采用 **MySQL** 而非默认的 **H2** 数据库，下面的配置文件显示了相关配置。

```
[database]
    type = MYSQL
    hostname = localhost
    database = reviewdb
    username = gerrit
```

LDAP 绑定或与数据库连接的用户口令保存在 `etc/secure.config` 文件中。

```
[database]
    password = secret
```

下面的配置将 **Web** 服务架设在 **Apache** 反向代理的后面。

```
[httpd]
    listenUrl = proxy-https://*:8081/gerrit
```

32.4 Gerrit 的数据库访问

之所以要对数据库访问多说几句，是因为在 Web 界面往往无法配置对 Gerrit 的一些设置，需要直接修改数据库，而大部分用户在安装 Gerrit 时都会选用内置的 H2 数据库，可能大部分用户并不了解如何操作 H2 数据库。

实际上无论选择何种数据库，Gerrit 都提供了两种数据库操作的命令行接口。第一种方法是在服务器端调用 `gerrit.war` 包中的命令入口，另外一种方法是远程 SSH 调用接口。

对于第一种方法，需要在服务器端执行，而且如果使用的是 H2 内置数据库还需要先将 Gerrit 服务停止。先以安装用户的身份进入 Gerrit 部署目录下，再执行命令调用 `gerrit.war` 包，如下：

```
$ java -jar bin/gerrit.war gsql
Welcome to Gerrit Code Review 2.1.5.1
(H2 1.2.134 (2010-04-23))

Type '\h' for help. Type '\r' to clear the buffer.

gerrit>
```

当出现“`gerrit>`”提示符时，就可以输入 SQL 语句操作数据库了。

第一种方式需要登录到服务器上，而且操作 H2 数据库时还要预先停止服务，显然很不方便。但是这种方法也有存在的必要，就是不需要认证，尤其是在管理员帐号尚未建立之前就可以查看和更改数据库。

当在 Gerrit 上注册了第一个帐号时，即拥有了管理员帐号，正确为该帐号配置公钥之后，就可以访问 Gerrit 提供的 SSH 登录服务。Gerrit 的 SSH 协议提供访问数据库的第二种方法。下面的命令就是用管理员公钥登录 Gerrit 的 SSH 服务器，操作数据库。虽然演示用的是本机地址（localhost），但是操作远程服务器也是可以的，只要拥有管理员权限。

```
$ ssh -p 29418 localhost gerrit gsql
Welcome to Gerrit Code Review 2.1.5.1
(H2 1.2.134 (2010-04-23))

Type '\h' for help. Type '\r' to clear the buffer.

gerrit>
```

运行命令 `gerrit gsql` 连接 Gerrit 的 SSH 服务。当连接上数据库管理接口后，便出现“`gerrit>`”提示符，在该提示符下可以输入 SQL 命令。下面的示例中，使用的数据库的后端为 H2

内置数据库。

可以输入 `show tables` 命令显示数据库列表。

```
gerrit> show tables;
```

| TABLE_NAME | TABLE_SCHEMA |
|-----------------------------|--------------|
| ACCOUNTS | PUBLIC |
| ACCOUNT_AGREEMENTS | PUBLIC |
| ACCOUNT_DIFF_PREFERENCES | PUBLIC |
| ACCOUNT_EXTERNAL_IDS | PUBLIC |
| ACCOUNT_GROUPS | PUBLIC |
| ACCOUNT_GROUP_AGREEMENTS | PUBLIC |
| ACCOUNT_GROUP_MEMBERS | PUBLIC |
| ACCOUNT_GROUP_MEMBERS_AUDIT | PUBLIC |
| ACCOUNT_GROUP_NAMES | PUBLIC |
| ACCOUNT_PATCH_REVIEWS | PUBLIC |
| ACCOUNT_PROJECT_WATCHES | PUBLIC |
| ACCOUNT_SSH_KEYS | PUBLIC |
| APPROVAL_CATEGORIES | PUBLIC |
| APPROVAL_CATEGORY_VALUES | PUBLIC |
| CHANGES | PUBLIC |
| CHANGE_MESSAGES | PUBLIC |
| CONTRIBUTOR_AGREEMENTS | PUBLIC |
| PATCH_COMMENTS | PUBLIC |
| PATCH_SETS | PUBLIC |
| PATCH_SET_ANCESTORS | PUBLIC |
| PATCH_SET_APPROVALS | PUBLIC |
| PROJECTS | PUBLIC |
| REF_RIGHTS | PUBLIC |
| SCHEMA_VERSION | PUBLIC |
| STARRED_CHANGES | PUBLIC |
| SYSTEM_CONFIG | PUBLIC |
| TRACKING_IDS | PUBLIC |

(27 rows; 65 ms)

输入 `show columns` 命令显示数据库的表结构。

```
gerrit> show columns from system_config;
```

| FIELD | TYPE | NULL | KEY | DEFAULT |
|----------------------------|--------------|------|-----|---------|
| REGISTER_EMAIL_PRIVATE_KEY | VARCHAR(36) | NO | | '' |
| SITE_PATH | VARCHAR(255) | YES | | NULL |
| ADMIN_GROUP_ID | INTEGER(10) | NO | | 0 |
| ANONYMOUS_GROUP_ID | INTEGER(10) | NO | | 0 |
| REGISTERED_GROUP_ID | INTEGER(10) | NO | | 0 |

```

WILD_PROJECT_NAME      | VARCHAR(255) | NO   |      | ''
BATCH_USERS_GROUP_ID   | INTEGER(10)  | NO   |      | 0
SINGLETON                | VARCHAR(1)   | NO   | PRI  | ''
(8 rows; 52 ms)

```

关于 H2 数据库更多的 SQL 语法，请参考：

<http://www.h2database.com/html/grammar.html>。

下面开始介绍 Gerrit 的使用。

32.5 立即注册为 Gerrit 管理员

第一个 Gerrit 账户自动成为权限最高的管理员，因此 Gerrit 安装完毕后的第一件事情就是立即注册或登录，以便初始化管理员帐号。下面的示例是在本机（localhost）以 HTTP 认证方式架设的 Gerrit 审核服务器。第一次访问的时候会弹出非常眼熟的 HTTP Basic Auth 认证界面，如图 32-4。

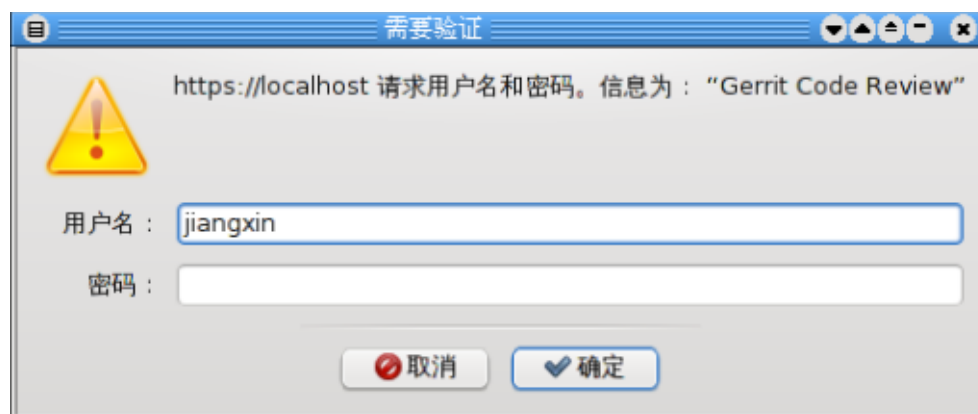


图 32-4: HTTP Basic Auth 认证界面

输入正确的用户名和口令登录后，系统自动创建 ID 为 1000000 的帐号，该帐号是第一个注册的帐号，会被自动赋予管理员的身份。因为使用的是 HTTP 认证，用户的邮件地址等个人信息尚未确定，因此登录后首先进入到个人信息设置界面，如图 32-5。

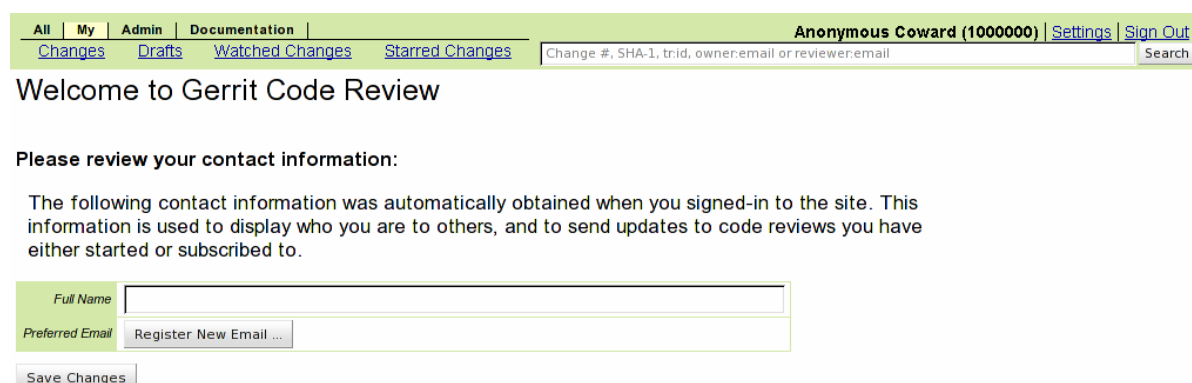


图 32-5: Gerrit 第一次登录后的个人信息设置界面

在图 32-5 中可以看到在菜单中有 “Admin” 菜单项，说明当前登录的用户被赋予了管理员权限。在图 32-5 的联系方式确认对话框中有一个注册新邮件地址的按钮，点击该按钮弹出邮件地址录入对话框，如图 32-6。

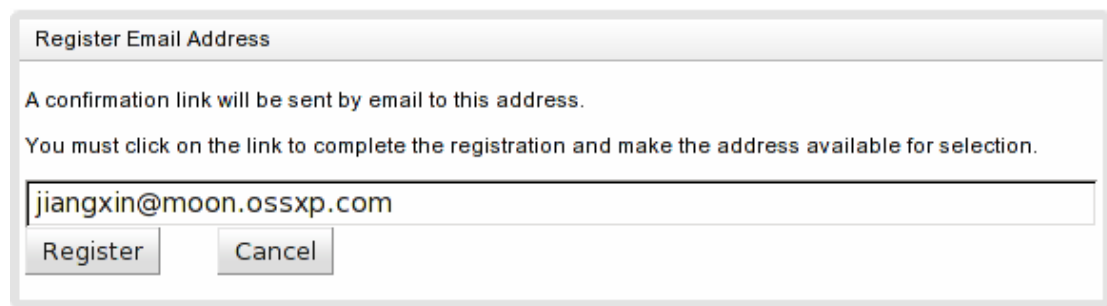
A dialog box titled "Register Email Address". It contains the text: "A confirmation link will be sent by email to this address. You must click on the link to complete the registration and make the address available for selection." Below the text is a text input field containing the email address "jiangxin@moon.ossxp.com". At the bottom are two buttons: "Register" and "Cancel".

图 32-6: 输入个人的邮件地址

必须输入一个有效的邮件地址以便能够收到确认邮件。这个邮件地址非常重要，因为 Git 代码提交时，在提交说明中出现的邮件地址需要和这个地址一致。填写了邮件地址后会收到一封确认邮件，点击邮件中的确认链接会重新进入到 Gerrit 帐号设置界面，如图 32-7。

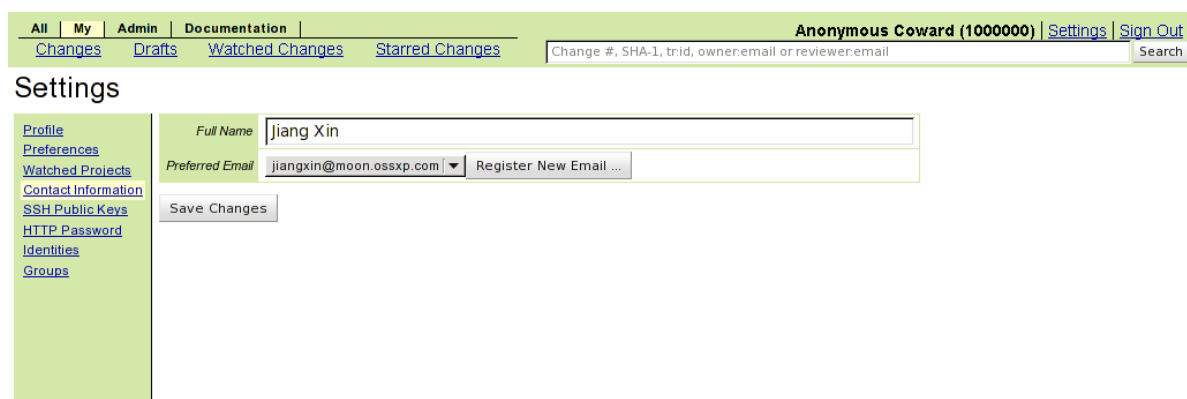
A screenshot of the Gerrit web interface. The top navigation bar includes links for "All", "My", "Admin", and "Documentation". The "Admin" link is highlighted. Below the navigation bar, there are links for "Changes", "Drafts", "Watched Changes", and "Starred Changes". The user's name "Anonymous Coward (1000000)" is displayed in the top right corner, along with "Settings" and "Sign Out" links. The main content area is titled "Settings" and contains a sidebar with links for "Profile", "Preferences", "Watched Projects", "Contact Information", "SSH Public Keys", "HTTP Password", "Identities", and "Groups". The "Profile" section is active, showing fields for "Full Name" (jiang Xin) and "Preferred Email" (jiangxin@moon.ossxp.com). There is a "Register New Email ..." button next to the email field and a "Save Changes" button at the bottom.

图 32-7: 邮件地址确认后进入 Gerrit 界面

在 Full Name 字段输入用户名，点击保存更改后，右上角显示的 “Anonymous Coward” 就会显示为用户登录的姓名和邮件地址。

接下来需要做的最重要的一件事就是配置公钥（如图 32-8）。通过该公钥，注册用户可以通过 SSH 协议向 Gerrit 的 Git 服务器提交，如果具有管理员权限还能够远程管理 Gerrit 服务器。



图 32-8: Gerrit 的 SSH 公钥设置界面

在文本框中粘贴公钥。关于如何生成和管理公钥，请参见 “第 29 章 使用 SSH 协议” 的相关内容。

点击 “Add” 按钮，完成公钥的添加。添加的公钥就会显示在列表中（如图 32-9）。一个用户可以添加多个公钥。



图 32-9: 用户的公钥列表

点击左侧的 “Groups”（用户组）菜单项，可以看到当前用户所属的分组，如图 32-10。



图 32-10: Gerrit 用户所属的用户组

第一个注册的用户同时属于三个用户组，一个是管理员用户组（Administrators），另外两个分别是 Anonymous Users（任何用户）和 Registered Users（注册用户）。

32.6 管理员访问 SSH 的管理接口

在 Gerrit 个人配置界面中设置了公钥之后，就可以连接 Gerrit 的 SSH 服务器执行命令，示例使用的是本机 localhost，其实远程 IP 地址一样可以。只是对于远程主机需要确认端口不要被防火墙拦截，Gerrit 的 SSH 服务器使用特殊的端口，默认是 29418。

任何用户都可以通过 SSH 连接执行 `gerrit ls-projects` 命令查看项目列表。下面的命令没有输出，是因为项目尚未建立。

```
$ ssh -p 29418 localhost gerrit ls-projects
```

可以执行 `scp` 命令从 Gerrit 的 SSH 服务器中拷贝文件。

```
$ scp -P 29418 -p -r localhost:/ gerrit-files

$ find gerrit-files -type f
gerrit-files/bin/gerrit-cherry-pick
gerrit-files/hooks/commit-msg
```

可以看出 Gerrit 服务器提供了两个文件可以通过 `scp` 下载，其中 `commit-msg` 脚本文件应该放在用户本地 Git 库的钩子目录中以便在生成的提交中包含唯一的 `Change-Id`。这在之前的 Gerrit 原理中介绍过。

除了普通用户可以执行的命令外，管理员还可以通过 SSH 连接执行 Gerrit 相关的管理命令。例如之前介绍的管理数据库：

```
$ ssh -p 29418 localhost gerrit gsql
Welcome to Gerrit Code Review 2.1.5.1
(H2 1.2.134 (2010-04-23))

Type '\h' for help. Type '\r' to clear the buffer.

gerrit>
```

此外管理员还可以通过 SSH 连接执行帐号创建，项目创建等管理操作，可以执行下面的命令查看帮助信息。

```
$ ssh -p 29418 localhost gerrit --help
gerrit COMMAND [ARG ...] [--] [--help (-h)]
```

```
--          : end of options
--help (-h) : display this help text

Available commands of gerrit are:

approve
create-account
create-group
create-project
flush-caches
gsq1
ls-projects
query
receive-pack
replicate
review
set-project-parent
show-caches
show-connections
show-queue
stream-events

See 'gerrit COMMAND --help' for more information.
```

更多的帮助信息，还可以参考 Gerrit 版本库中的帮助文件：<Documentation/cmd-index.html>。

32.7 创建新项目

一个 Gerrit 项目对应于一个同名的 Git 库，同时拥有一套可定制的评审流程。创建一个新的 Gerrit 项目就会在对应的版本库根目录下创建 Git 库。管理员可以使用命令行创建新项目。

```
$ ssh -p 29418 localhost gerrit create-project --name new/project
```

执行 `gerrit ls-projects` 命令可以看到新项目已经成功创建。

```
$ ssh -p 29418 localhost gerrit ls-projects
new/project
```

在 Gerrit 的 Web 管理界面也可以看到新项目已经建立，如图 32-11。

| All | My | Admin | Documentation | Jiang Xin <jiangxin@moon.ossxp.com> Settings Sign Out | |
|--------------------------------------|--|---|-------------------------------|---|-------------------------------------|
| Groups | Projects | Change #, SHA-1, tr.id, owner:email or reviewer:email | | | <input type="text" value="Search"/> |
| Projects | | | | | |
| Project Name | Description | | | | |
| ► -- All Projects -- | Rights inherited by all other projects | | | | |
| new/project | | | | | |

图 32-11: Gerrit 中项目列表

在项目列表中可以看到除了新建的 `new/project` 项目之外还有一个名为“-- All Projects --”的项目，其实它并非一个真实存在的项目，只是为了项目授权管理的方便——在“-- All Projects --”中建立的项目授权能够被其他项目共享。

在服务器端也可以看到在 Gerrit 部署中，版本库根目录下已经有同名的 Git 版本库被创建。

```
$ ls -d /home/gerrit/review_site/git/new/project.git
/home/gerrit/review_site/git/new/project.git
```

这个新的版本库刚刚初始化尚未包括任何数据。是否可以通过 `git push` 向该版本库推送一些初始数据呢？下面用 Gerrit 的 SSH 协议克隆该版本库，并尝试向其推送数据。

```
$ git clone ssh://localhost:29418/new/project.git myproject
Cloning into myproject...
warning: You appear to have cloned an empty repository.

$ cd myproject/

$ echo hello > readme.txt

$ git add readme.txt

$ git commit -m "initialized."
[master (root-commit) 15a549b] initialized.
1 files changed, 1 insertions(+), 0 deletions(-)
create mode 100644 readme.txt
$ git push origin master
Counting objects: 3, done.
Writing objects: 100% (3/3), 222 bytes, done.
Total 3 (delta 0), reused 0 (delta 0)
To ssh://localhost:29418/new/project.git
 ! [remote rejected] master -> master (prohibited by Gerrit)
error: failed to push some refs to 'ssh://localhost:29418/new/project.git'
```

向 Gerrit 的 Git 版本库推送失败，远程 Git 服务器返回错误信息：“prohibited by Gerrit”。这是因为 Gerrit 默认不允许直接向分支推送，而是需要向 `refs/for/<branch-name>` 的特殊

引用进行推送以便将提交转换为评审任务。

但是是否可以将版本库的历史提交不经审核，直接推送到 Gerrit 维护的 Git 版本库中呢？是的，只要通过 Gerrit 的管理界面为该项目授权：允许某个用户组（如 Administrators 组）的用户可以直接向分支推送。（注意该授权在推送完毕后尽快撤销，以免被滥用）

Gerrit 的界面对用户非常友好（如图 32-12）。例如在添加授权的界面中，只要在用户组的输入框中输入前几个字母，就会弹出用户组列表以供选择。



图 32-12：添加授权的界面

添加授权完毕后，项目 “new/project” 的授权列表就会出现新增的为 Administrators 管理员添加的 “+2: Create Branch” 授权，如图 32-13。

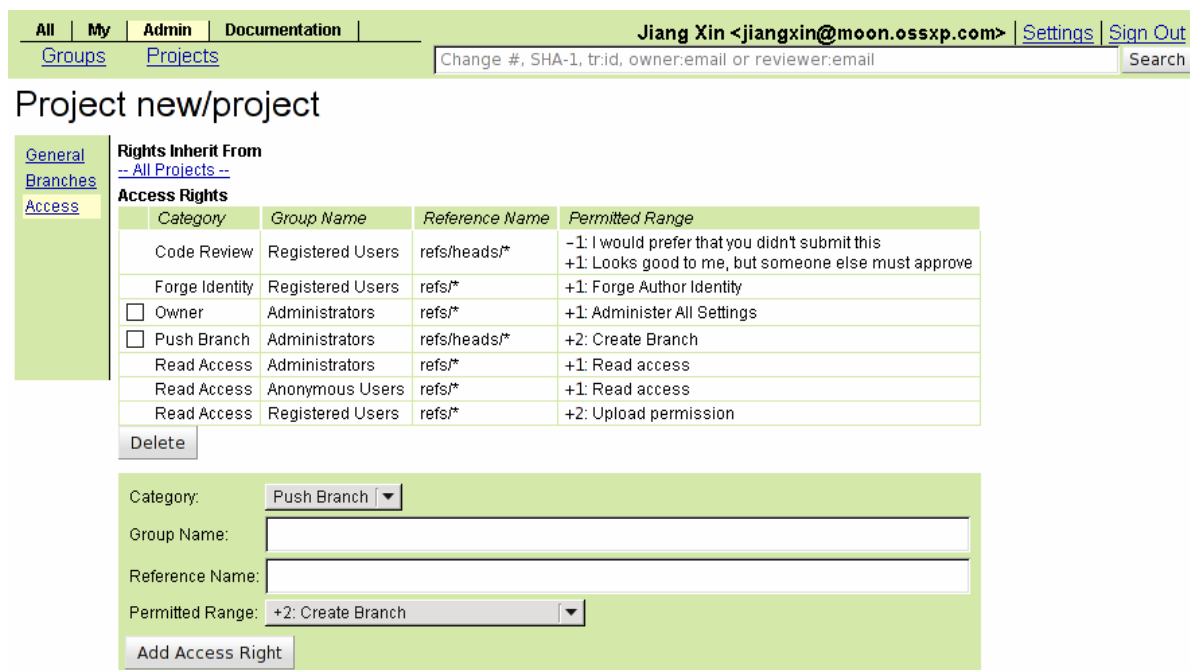


图 32-13: 添加授权后的授权列表

因为已经为管理员分配了直接向 `refs/heads/*` 引用推送的授权，这样就能够向 Git 版本库推送数据了。再执行一次推送任务看看能否成功。

```
$ git push origin master
Counting objects: 3, done.
Writing objects: 100% (3/3), 222 bytes, done.
Total 3 (delta 0), reused 0 (delta 0)
To ssh://localhost:29418/new/project.git
 ! [remote rejected] master -> master (you are not committer jiangxin@ossxp.com)
error: failed to push some refs to 'ssh://localhost:29418/new/project.git'
```

推送又失败了，但是服务器端返回的错误信息却不同。上一次出错返回的是“prohibited by Gerrit”，而这一次返回的错误信息是“you are not committer”。

这是为什么呢？看看提交日志：

```
$ git log --pretty=full
commit 15a549bac6bd03ad36e643984fed554406480b2c
Author: Jiang Xin <jiangxin@ossxp.com>
Commit: Jiang Xin <jiangxin@ossxp.com>

initialized.
```

提交者 Committer 为“Jiang Xin <jiangxin@ossxp.com>”，而 Gerrit 中注册的用户邮件地址是“jiangxin@moon.ossxp.com”，两者之间不一致，导致 Gerrit 再一次拒绝了提交。如果再到 Gerrit 中看一下 new/project 的权限设置，会看到这样一条授权：

| Category | Group Name | Reference Name | Permitted Range |
|----------------|------------------|----------------|---------------------------|
| ===== | ===== | ===== | ===== |
| Forge Identity | Registered Users | refs/* | +1: Forge Author Identity |

这条授权的含义是：提交中的 **Author** 字段不进行邮件地址是否注册的检查，但是要对 **Commit** 字段进行邮件地址检查。如果增加一个更高级别的“Forge Identity”授权，也可以忽略对 **Committer** 的邮件地址的检查，但是尽量不要对授权进行非必须的改动，因为在提交的时候使用注册的邮件地址是一个非常好的实践。

下面就通过 `git config` 命令修改提交时所用的邮件地址，和 **Gerrit** 注册时用的地址保持一致。然后用 `--amend` 参数重新执行提交以便让修改后的提交者邮件地址在提交中生效。

```
$ git config user.email jiangxin@moon.ossxp.com

$ git commit --amend -m initialized
[master 82c8fc3] initialized
Author: Jiang Xin <jiangxin@ossxp.com>
1 files changed, 1 insertions(+), 0 deletions(-)
create mode 100644 readme.txt

$ git push origin master
Counting objects: 3, done.
Writing objects: 100% (3/3), 233 bytes, done.
Total 3 (delta 0), reused 0 (delta 0)
To ssh://localhost:29418/new/project.git
* [new branch]      master -> master
```

看，这次提交成功了！之所以成功，是因为提交者的邮件地址更改了。看看重新提交的日志，可以发现 **Author** 和 **Commit** 的邮件地址不同，而 **Commit** 字段的邮件地址和注册时使用的邮件地址相同。

```
$ git log --pretty=full
commit 82c8fc3805d57cc0d17d58e1452e21428910fd2d
Author: Jiang Xin <jiangxin@ossxp.com>
Commit: Jiang Xin <jiangxin@moon.ossxp.com>

    initialized
```

注意，版本库初始化完成之后，应尽快删除为项目新增的“Push Branch”类型的授权，对新的提交强制使用 **Gerrit** 的评审流程。

32.8 从已有的 Git 库创建项目

如果已经拥有很多版本库，希望从这些版本库创建 Gerrit 项目，如果像上面介绍的那样一个地创建项目，再执行 `git push` 命令推送已经包含历史数据的版本库，将是十分麻烦的事情。那么有没有什么简单的办法呢？可以通过下面的步骤实现多项目的快速创建。

首先将已有版本库创建到 Gerrit 的版本库根目录下。注意版本库名称将会成为项目名（除去 `.git` 后缀），而且创建（或克隆）的版本库应为裸版本库，即使用 `--bare` 参数创建。

例如在 Gerrit 的 Git 版本库根目录下创建名为 `hello.git` 的版本库。下面的示例中我偷了一下懒，直接从 `new/project` 克隆到 `hello.git` 。

```
$ git clone --mirror \
    /home/gerrit/review_site/git/new/project.git \
    /home/gerrit/review_site/git/hello.git
```

这时查看版本库列表，却看不到新建立的名为 `hello.git` 的 Git 库出现在项目列表中。

```
$ ssh -p 29418 localhost gerrit ls-projects
new/project
```

可以通过修改 Gerrit 数据库来注册新项目，即连接到 Gerrit 数据库，输入 SQL 插入语句。

```
$ ssh -p 29418 localhost gerrit gsql
Welcome to Gerrit Code Review 2.1.5.1
(H2 1.2.134 (2010-04-23))

Type '\h' for help. Type '\r' to clear the buffer.

gerrit> INSERT INTO projects
    -> (use_contributor_agreements ,submit_type ,name)
    -> VALUES
    -> ('N' , 'M' , 'hello');
UPDATE 1; 1 ms
gerrit>
```

注意 SQL 语句中的项目名称是版本库名称除去 `.git` 后缀的部分。在数据库插入数据后，再来查看项目列表就可以看到新注册的项目了。

```
$ ssh -p 29418 localhost gerrit ls-projects
hello
new/project
```

可以登录到 Gerrit 项目对新建立的项目进行相关设置。例如修改项目的说明，项目的提交策略，是否要求提交说明中必须包含“Signed-off-by”信息等，如图 32-14。



图 32-14：项目基本设置

这种通过修改数据库从已有版本库创建项目的方法适合创建大批量的项目。下面就对新建立的 hello 进行一次完整的 Gerrit 评审流程。

32.9 定义评审 workflow

刚刚安装好的 Gerrit 的评审 workflow 并不完整，还不能正常地开展评审工作，需要对项目授权进行设置以定制适合的评审 workflow。

默认安装的 Gerrit 中只内置了四个用户组，如表 32-1 所示。

表 32-1：Gerrit 内置用户组

| 用户组 | 说明 |
|-----------------------|------------------|
| Administrators | Gerrit 管理员 |
| Anonymous Users | 任何用户，登录或未登录 |
| Non-Interactive Users | Gerrit 中执行批处理的用户 |
| Registered Users | 任何登录用户 |

未登录的用户只属于 Anonymous Users，登录用户则同时拥有 Anonymous Users 和 Registered Users 的权限。对于管理员则还拥有 Administrators 用户组权限。

查看全局（伪项目 “-- All Projects --”）的初始权限设置，会看到如表 32-2 一样的授权表格。

表 32-2：Gerrit 授权表格

| 编号 | 类别 | 用户组名称 | 引用名称 | 权限范围 |
|----|----|-------|------|------|
|----|----|-------|------|------|

| | | | | |
|---|----------------|------------------|--------------|---|
| 1 | Code Review | Registered Users | refs/heads/* | -1: I would prefer that you didn't submit this |
| | | | | +1: Looks good to me, but someone else must approve |
| 2 | Forge Identity | Registered Users | refs/* | +1: Forge Author Identity |
| 3 | Read Access | Administrators | refs/* | +1: Read access |
| 4 | Read Access | Anonymous Users | refs/* | +1: Read access |
| 5 | Read Access | Registered Users | refs/* | +2: Upload permission |

对此表格中的授权解读如下：

- ❑ 对于匿名用户：根据第 4 条授权策略，匿名用户能够读取任意版本库。
 - ❑ 对于注册用户：根据第 5 条授权策略，注册用户具有比第四条授权高一个等级的权限，即注册用户除了具有读取版本库权限外，还可以向版本库的 `refs/for/<branch-name>` 引用推送，产生评审任务的权限。
- 之所以这种可写的权限也放在“Read Access”类别中，是因为 Git 的写操作必须建立在拥有读权限之上，因此 Gerrit 将其与读取都放在“Read Access”归类之下，只不过高一个级别。
- ❑ 对于注册用户：根据第 2 条授权策略，在向服务器推送提交的时候，忽略对提交中 `Author` 字段的邮件地址检查。这个在之前已经讨论过。
 - ❑ 对于注册用户：根据第 1 条授权策略，注册用户具有代码审核的一般权限，即能够将评审任务设置为“+1”级别（看起来不错，但需要通过他人认可），或者将评审任务标记为“-1”（评审任务没有通过不能提交）。
 - ❑ 对于管理员：根据第 3 条策略，管理员能够读取任意版本库。

上面的授权策略仅仅对评审流程进行了部分设置。如：提交能够进入评审流程，因为登录用户（注册用户）可以将提交以评审任务方式上传；注册用户可以将评审任务标记为“+1：看起来不错，但需其他人认可”。但是没有人有权限可以将评审任务提交逐一合并到正式版本库中，即没有人能够对评审任务做最终的确认及提交，因此评审流程是不完整的。

有两种方法可以实现对评审最终确认的授权，一种是赋予特定用户 `Verified` 类别中的“+1: Verified”的授权，另外一个方法是赋予特定用户 `Code Review` 类别中更高级别的授权：“+2: Looks good to me, approved”。要想实现对经过确认的评审任务的提交，还需要赋予特定用户 `Submit` 类别中的“+1: Submit”授权。

下面的示例中，创建两个新的用户组 **Reviewer** 和 **Verifier**，并为其赋予相应的授权。

可以通过 **Web** 界面或命令行创建用户组。如果通过 **Web** 界面添加用户组，选择“**Admin**”菜单下的“**Groups**”子菜单，如图 32-15。



图 32-15: Gerrit 用户组创建

输入用户组名称后，点击“**Create Group**”按钮，进入创建用户组后的设置页，如图 32-16。

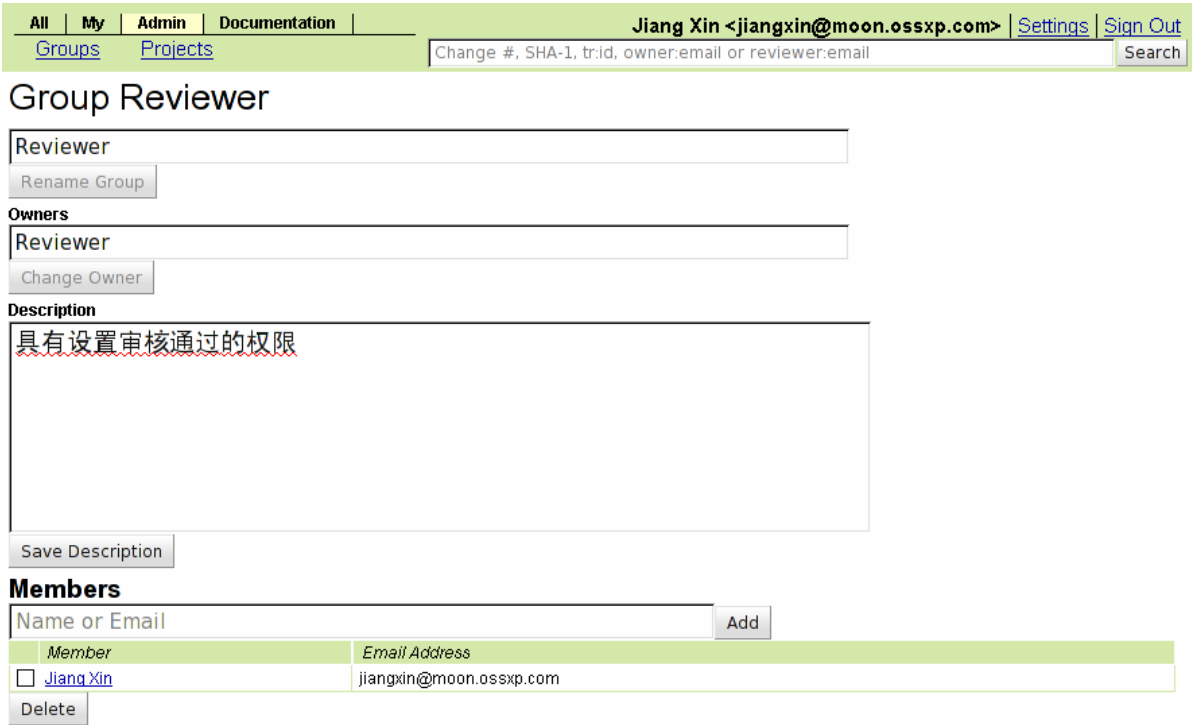


图 32-16: Gerrit 用户组设置页

注意到在用户设置页面中有一个 **Owners** 字段名称和用户组名称相同，实际上这是 **Gerrit** 关于用户组的一个特别的功能。一个用户组可以设置另外一个用户组为本用户组的 **Owners**，属于 **Owners** 用户组的用户实际上相当于本用户组的管理者，可以添加用户、修改用户组名称等。不过一般最常用的设置是使用同名的用户组作为 **Owners**。

在用户组设置页面的最下面，是用户组用户分配对话框，可以将用户分配到用户组中。

图 32-17 是添加了两个新用户组后的用户组列表：



图 32-17: Gerrit 用户组列表

接下来要为新的用户组授权，需要访问“Admin”菜单下的“Projects”子菜单，点击对应的项目进入权限编辑界面。为了简便起见，选择“-- All Projects --”，对其授权的更改可以被其他的所有项目共享。图 32-18 是为 Reviewer 用户组建立授权过程的页面。

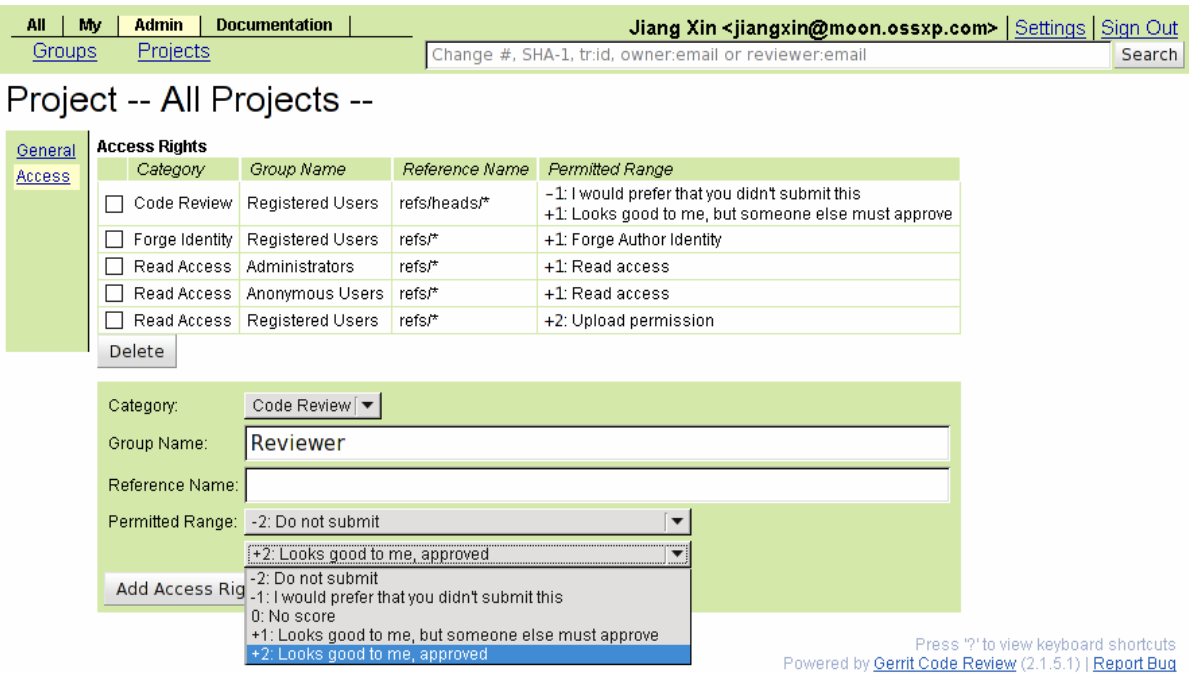


图 32-18: 为 Reviewer 用户组建立授权

分别为两个新建的用户组分配授权，如表 32-3 所示。编号从 6 开始，是因为这里补充的授权是建立在前面的默认授权列表的基础上的。

表 32-3: 新用户组权限分配表

| 编号 | 类别 | 用户组名称 | 引用名称 | 权限范围 |
|----|----|-------|------|------|
|----|----|-------|------|------|

| | | | | |
|---|-------------|----------|--------|--------------------------------|
| 6 | Code Review | Reviewer | refs/* | -2: Do not submit |
| | | | | +2: Looks good to me, approved |
| 7 | Verified | Verifier | refs/* | -1: Fails |
| | | | | +1: Verified |
| 8 | Submit | Verifier | refs/* | +1: Submit |

这样，就为 Gerrit 所有的项目设定了可用的评审 workflow。

32.10 Gerrit 评审 workflow 实战

分别再注册两个用户帐号 `dev1@moon.ossxp.com` 和 `dev2@moon.ossxp.com`，两个用户分别属于 `Reviewer` 用户组和 `Verifier` 用户组。这样 Gerrit 部署中就拥有了三个用户帐号，用帐号 `jiangxin` 进行代码提交，用 `dev1` 帐号对任务进行代码审核，用 `dev2` 帐号对审核任务进行最终的确认。

32.10.1 开发者在本地版本库中工作

`Repo` 是 Gerrit 的最佳伴侣，凡是需要和 Gerrit 版本库交互的工作都封装在 `repo` 命令中。关于 `repo` 的用法在上一部分的 `repo` 多版本库协同的章节中已经详细介绍了。这里只介绍开发者如何只使用 `git` 命令来和 Gerrit 服务器交互。这样也可以更深入地理解 `repo` 和 Gerrit 整合的机制，具体操作过程如下。

- (1) 首先克隆 Gerrit 管理的版本库，使用 Gerrit 提供的运行于 29418 端口的 SSH 协议。

```
$ git clone ssh://localhost:29418/hello.git
Cloning into hello...
remote: Counting objects: 3, done
remote: Compressing objects: 100% (3/3)
Receiving objects: 100% (3/3), done.
```

- (2) 然后拷贝 Gerrit 服务器提供的 `commit-msg` 钩子脚本。

```
$ cd hello
$ scp -P 29418 -p localhost:/hooks/commit-msg .git/hooks/
```

- (3) 别忘了修改 Git 配置中提交者的邮件地址，以便和 Gerrit 中注册的地址保持一致。不使用 `--global` 参数调用 `git config` 可以只对本版本库的提交设定提交者邮件。

```
$ git config user.email jiangxin@moon.ossxp.com
```

- (4) 然后修改 `readme.txt` 文件并提交。注意提交的时候使用了“-s”参数，目的是在提交说明中加入“Signed-off-by:”标记，这在 Gerrit 提交中可能是必须的。

```
$ echo "gerrit review test" >> readme.txt
$ git commit -a -s -m "readme.txt hacked."
[master c65ab49] readme.txt hacked.
1 files changed, 1 insertions(+), 0 deletions(-)
```

- (5) 查看一下提交日志，会看到其中有特殊的标签。

```
$ git log --pretty=full -1
commit c65ab490f6d3dc36429b8f1363b6191357202f2e
Author: Jiang Xin <jiangxin@moon.ossxp.com>
Date: Mon Nov 15 17:50:08 2010 +0800

    readme.txt hacked.

Change-Id: Id7c9d88ebf5dac2d19a7e0896289de1ae6fb6a90
Signed-off-by: Jiang Xin <jiangxin@moon.ossxp.com>
```

提交说明中出现了“Change-Id:”标签，这个标签是由钩子脚本“commit-msg”自动生成的。

至于这个标签的含义，在前面 Gerrit 的实现原理中已经介绍过。

好了，准备把这个提交推送到服务器上吧。

32.10.2 开发者向审核服务器提交

由 Gerrit 控制的 Git 版本库不能直接提交，因为正确设置的 Gerrit 服务器，会拒绝用户直接向 `refs/heads/*` 推送。

```
$ git status
# On branch master
# Your branch is ahead of 'origin/master' by 1 commit.
#
nothing to commit (working directory clean)

$ git push
Counting objects: 5, done.
Writing objects: 100% (3/3), 332 bytes, done.
Total 3 (delta 0), reused 0 (delta 0)
To ssh://localhost:29418/hello.git
! [remote rejected] master -> master (prohibited by Gerrit)
error: failed to push some refs to 'ssh://localhost:29418/hello.git'
```

直接推送就会遇到“prohibited by Gerrit”的错误。

正确的做法是向特殊的引用推送，这样 Gerrit 会自动将新提交转换为评审任务。

```
$ git push origin HEAD:refs/for/master
Counting objects: 5, done.
Writing objects: 100% (3/3), 332 bytes, done.
Total 3 (delta 0), reused 0 (delta 0)
To ssh://localhost:29418/hello.git
 * [new branch]      HEAD -> refs/for/master
```

看到了么，向 refs/for/master 推送成功。

32.10.3 审核评审任务

以 Dev1 用户登录 Gerrit 网站，点击“**All**”菜单下的“**Open**”标签，可以看到新提交到 Gerrit 的状态为 Open 的评审任务，如图 32-19。



图 32-19: Gerrit 评审任务列表

点击该评审任务，显示关于此评审任务的详细信息，如图 32-20。

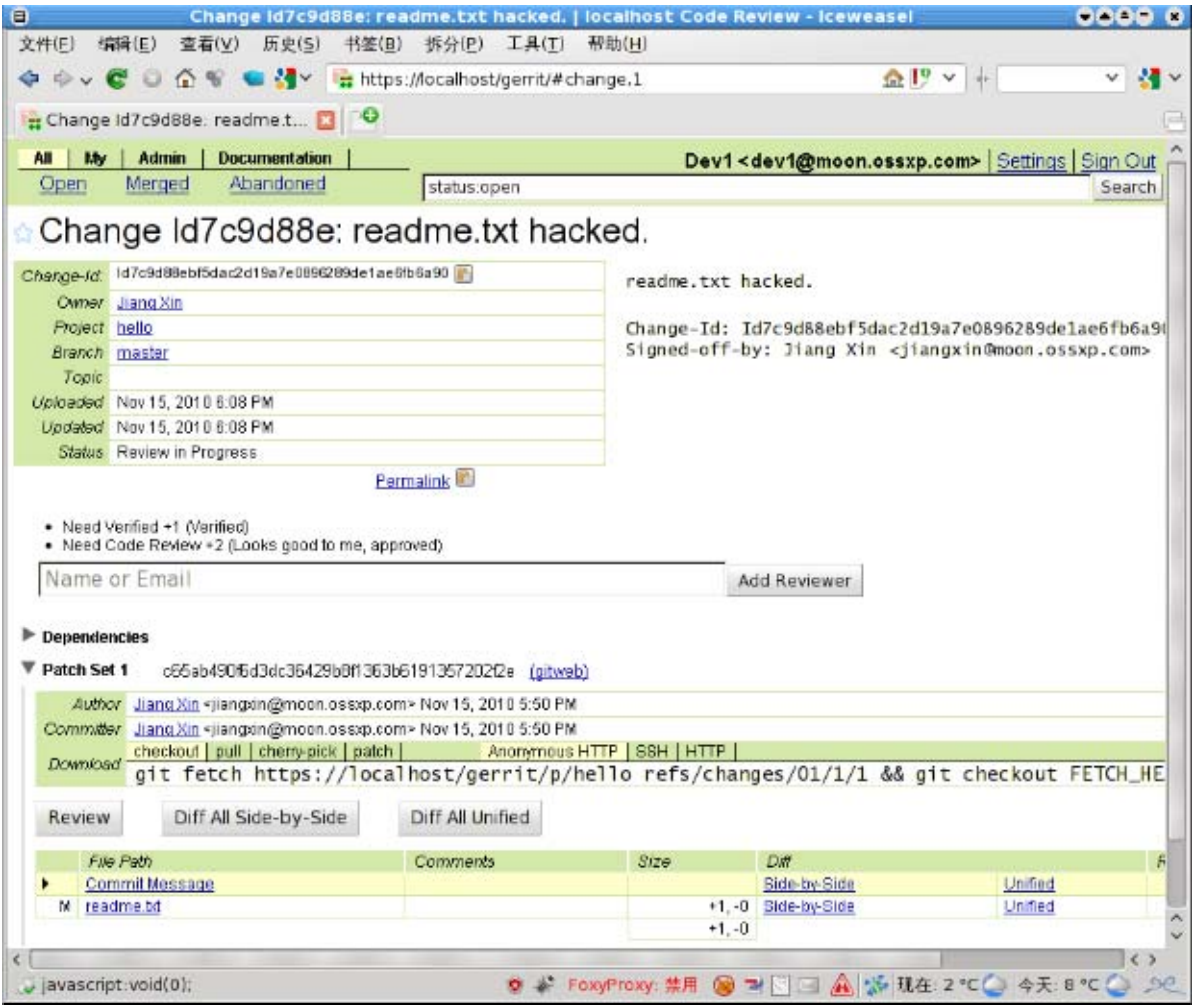


图 32-20: Gerrit 评审任务概述

从 URL 地址栏可以看到该评审任务的评审编号为 1。目前该评审任务有一个补丁集(Patch Set 1)，可以点击 “Diff All Side-by-Side” 查看变更集，以决定该提交是否应该被接受。作为测试，先让此次提交通过代码审核，于是以 Dev1 用户身份点击 “Review” 按钮。点击 “Review” 按钮后，弹出代码评审对话框，如图 32-21。



图 32-21: Gerrit 任务评审对话框

选择 “+2: Looks good to me, approved.”，点击按钮 “Publish Comments” 以通过评审。注意因为没有给 Dev1 用户（Reviewer 用户组）授予 Submit 权限，因此此时 Dev1 还不能将此审核任务提交。

Dev1 用户做出通过评审的决定后，代码提交者 jiangxin 会收到一封邮件，如图 32-22。

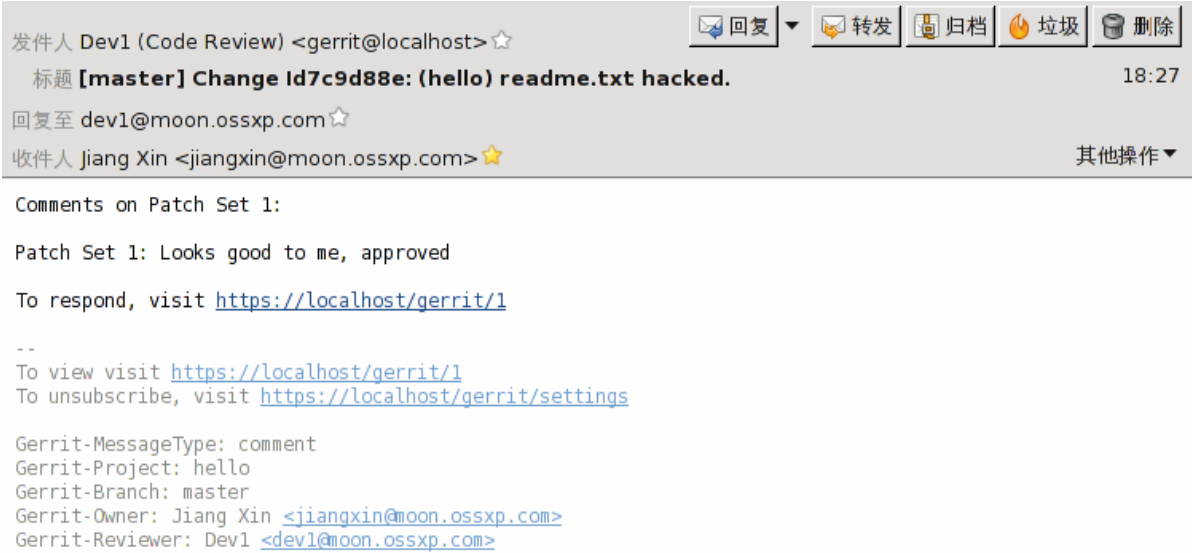


图 32-22: Gerrit 通知邮件

32.10.4 评审任务没有通过测试

下面以 Dev2 帐号登录 Gerrit，查看处于打开状态的评审任务，如图 32-23。会看到评审任务 1 的代码评审已经通过，但是尚未进行测试检查（Verify）。于是 Dev2 下载该补丁集，在本机进行测试。



图 32-23: Gerrit 评审任务显示

假设测试没有通过，Dev2 用户点击该评审任务的 “Review” 按钮，重置该任务的评审状态，如图 32-24。



图 32-24: Gerrit 评审任务未通过

注意到图 32-24 中 Dev2 用户的评审对话框有三个按钮，多出的“Publish and Submit”按钮是因为 Dev2 拥有 Submit 授权。Dev2 用户在上面的对话框中选择了“-1: Fails”，点击“Publish Comments”按钮，该评审任务的评审记录被重置，同时提交者和其他评审参与者会收到通知邮件，如图 32-25。

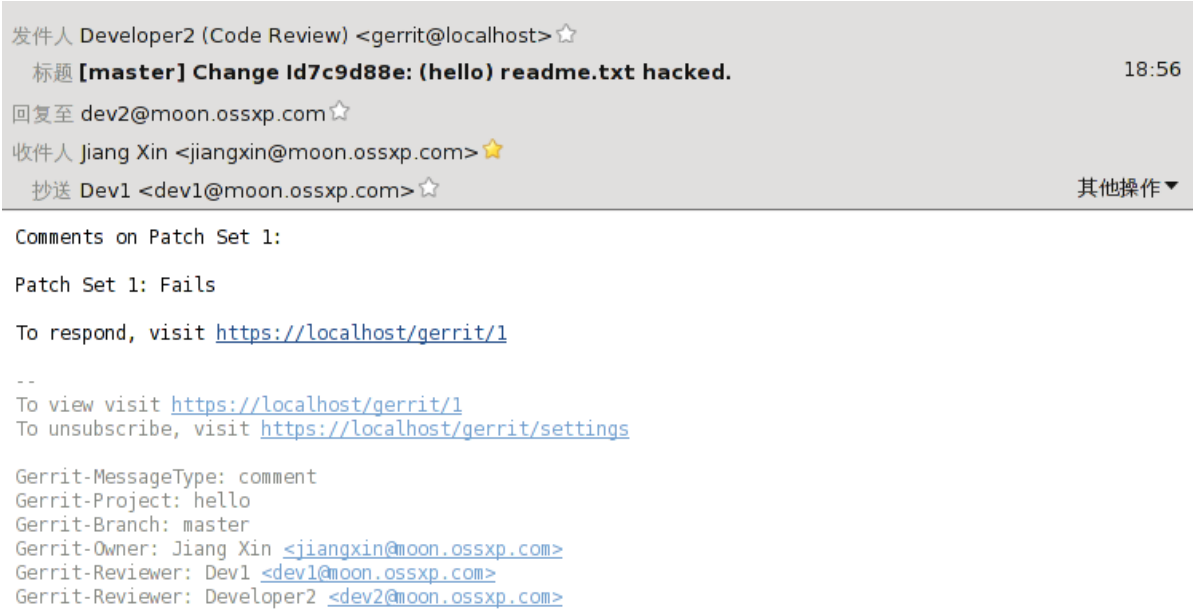


图 32-25: Gerrit 通知邮件：评审未通过

32.10.5 重新提交新的补丁集

提交者收到代码被打回的邮件，一定很难过。不过这恰恰说明了这个软件过程已经相当的完善，现在发现问题总比在集成测试时甚至被客户发现要好得多吧。

根据评审者和检验者的提示，开发者对代码进行重新修改。下面的 **Bugfix** 过程仅仅是一个简单的示例，**Bugfix** 没有这么简单的，对么？;-)

```
$ echo "fixed" >> readme.txt
```

重新修改后，需要使用 `--amend` 参数进行提交，即使用前次提交的日志重新提交，这一点非常重要。因为这样就会对原提交说明中的 “**Change-Id:**” 标签予以原样保留，当再将新提交推送到服务器时，**Gerrit** 不会为新提交生成新的评审任务编号，而是重用原有的任务编号，将新提交转化为老评审任务的新补丁集，具体操作过程如下。

- (1) 在执行 `git commit --amend` 时，可以修改提交说明，但是注意不要删除 **Change-Id** 标签，更不能修改它。

```
$ git add -u
$ git commit --amend

readme.txt hacked with bugfix.

Change-Id: Id7c9d88ebf5dac2d19a7e0896289de1ae6fb6a90
Signed-off-by: Jiang Xin <jiangxin@moon.ossxp.com>

# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
# On branch master
# Your branch is ahead of 'origin/master' by 1 commit.
#
# Changes to be committed:
#   (use "git reset HEAD^1 <file>..." to unstage)
#
# modified:   readme.txt
#
```

- (2) 提交成功后，执行 `git ls-remote` 命令会看到 **Gerrit** 维护的 **Git** 库中只有一个评审任务（编号 1），且该评审任务只有一个补丁集（Patch Set 1）。

```
$ git ls-remote origin
82c8fc3805d57cc0d17d58e1452e21428910fd2d      HEAD
c65ab490f6d3dc36429b8f1363b6191357202f2e      refs/changes/01/1/1
```

```
82c8fc3805d57cc0d17d58e1452e21428910fd2d      refs/heads/master
```

- (3) 把修改后的提交推送到 Gerrit 管理下的 Git 版本库中。注意依旧推送到

refs/for/master 引用中。

```
$ git push origin HEAD:refs/for/master
Counting objects: 5, done.
Writing objects: 100% (3/3), 353 bytes, done.
Total 3 (delta 0), reused 0 (delta 0)
To ssh://localhost:29418/hello.git
 * [new branch]      HEAD -> refs/for/master
```

- (4) 推送成功后，再执行 `git ls-remote` 命令，会看到唯一的评审任务（编号 1）有了两个补丁集。

```
$ git ls-remote origin
82c8fc3805d57cc0d17d58e1452e21428910fd2d      HEAD
c65ab490f6d3dc36429b8f1363b6191357202f2e      refs/changes/01/1/1
1df9e8e05fcf97a46588488918a476abd1df8121      refs/changes/01/1/2
82c8fc3805d57cc0d17d58e1452e21428910fd2d      refs/heads/master
```

32.10.6 新修订集通过评审

当提交者重新针对评审任务进行提交时，原评审任务的审核者会收到通知邮件，提醒有新的补丁集等待评审，如图 32-26。

```
发件人 Jiang Xin (Code Review) <gerrit@localhost> ☆
标题 [master] Change Id7c9d88e: (hello) readme.txt hacked. 19:06
回复至 jiangxin@moon.ossxp.com ☆
收件人 Developer2 <dev2@moon.ossxp.com> ☆, Dev1 <dev1@moon.ossxp.com> ☆ 其他操作 ▼

Hello Developer2, Dev1,

I'd like you to reexamine change Id7c9d88e.
Change Id7c9d88e (patch set 2) for master in hello:

readme.txt hacked with bugfix.

Change-Id: Id7c9d88ebf5dac2d19a7e0896289delaefb6a90
Signed-off-by: Jiang Xin <jiangxin@moon.ossxp.com>
---
M readme.txt
1 file changed, 2 insertions(+), 0 deletions(-)

git pull ssh://localhost:29418/hello refs/changes/01/1/2
--
Gerrit-MessageType: newpatchset
Gerrit-Project: hello
Gerrit-Branch: master
Gerrit-Owner: Jiang Xin <jiangxin@moon.ossxp.com>
Gerrit-Reviewer: Dev1 <dev1@moon.ossxp.com>
Gerrit-Reviewer: Developer2 <dev2@moon.ossxp.com>
Gerrit-Reviewer: Jiang Xin <jiangxin@moon.ossxp.com>
```

图 32-26: Gerrit 通知邮件：新补丁集

登录 Gerrit 的 Web 界面，可以看到评审任务 1 有了新的补丁集，如图 32-27。

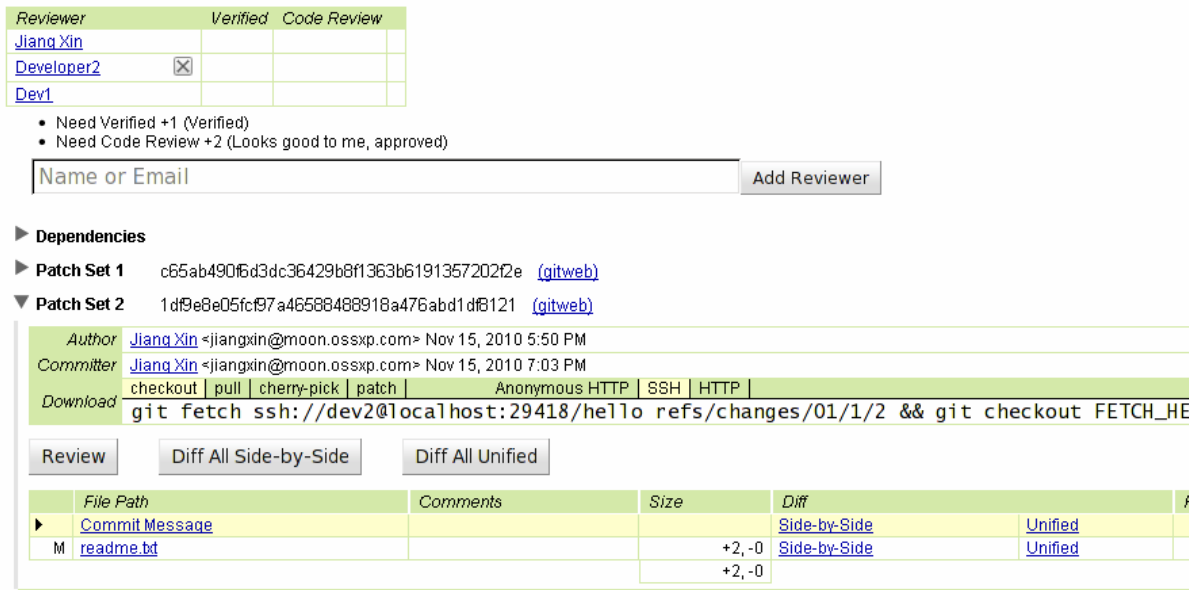


图 32-27: Gerrit 新补丁集显示

再经过代码审核和测试，这次 Dev2 用户决定让评审通过，点击了 “Publish and Submit” 按钮。Submit（提交）动作会将评审任务（refs/changes/01/1/2）合并到对应分支（master）中。图 32-28 显示的是通过评审完成合并的评审任务 1。



图 32-28: Gerrit 合并后的评审任务

32.10.7 从远程版本库更新

当 Dev1 和 Dev2 用户完成代码评审后，提交者会收到多封通知邮件。这其中最让人激动的就是代码被接受并合并到开发主线（master）中（如图 32-29），这令开发者感到多么荣耀啊。

```
发件人 Developer2 (Code Review) <gerrit@localhost> ☆
标题 [master] Change Id7c9d88e: (hello) readme.txt hacked with bugfix. 19:14
回复至 dev2@moon.ossxp.com ☆
收件人 Jiang Xin <jiangxin@moon.ossxp.com> ☆
抄送 Dev1 <dev1@moon.ossxp.com> ☆ 其他操作 ▼

Change Id7c9d88e by Jiang Xin submitted to master:

readme.txt hacked with bugfix.

Change-Id: Id7c9d88ebf5dac2d19a7e0896289de1ae6fb6a90
Signed-off-by: Jiang Xin <jiangxin@moon.ossxp.com>
---
M readme.txt
1 file changed, 2 insertions(+), 0 deletions(-)

Approvals:
Developer2: Verified
Dev1: Looks good to me, approved

--
To view visit https://localhost/gerrit/1
To unsubscribe, visit https://localhost/gerrit/settings

Gerrit-MessageType: merged
Gerrit-Project: hello
Gerrit-Branch: master
Gerrit-Owner: Jiang Xin <jiangxin@moon.ossxp.com>
Gerrit-Reviewer: Dev1 <dev1@moon.ossxp.com>
Gerrit-Reviewer: Developer2 <dev2@moon.ossxp.com>
Gerrit-Reviewer: Jiang Xin <jiangxin@moon.ossxp.com>
```

图 32-29: Gerrit 通知邮件：修订已合并

代码提交者执行 `git pull`，和 Gerrit 管理的版本库同步。

```
$ git ls-remote origin
1df9e8e05fcf97a46588488918a476abd1df8121      HEAD
c65ab490f6d3dc36429b8f1363b6191357202f2e      refs/changes/01/1/1
1df9e8e05fcf97a46588488918a476abd1df8121      refs/changes/01/1/2
1df9e8e05fcf97a46588488918a476abd1df8121      refs/heads/master

$ git pull
From ssh://localhost:29418/hello
   82c8fc3..1df9e8e  master    -> origin/master
Already up-to-date.
```

32.11 更多 Gerrit 参考

Gerrit 涉及的内容非常庞杂，还有诸如和 Gitweb、git-daemon 整合，Gerrit 界面定制等功能，

恕不在此一一列举。可以直接参考 Gerrit 网站上的帮助¹。

¹ <http://gerrit.googlecode.com/svn/documentation/>