

配置管理学习总结

1 Gerrit 服务器搭建

1.1 Gerrit 简介

Gerrit，一种免费、开放源代码的代码审查软件，使用网页界面。利用网页浏览器，同一个团队的软件程序员，可以相互审阅彼此修改后的程序代码，决定是否能够提交，退回或者继续修改。它使用 Git 作为底层版本控制系统。它分支自 Rietveld，作者为 Google 公司的 Shawn Pearce，原先是为了管理 Android 计划而产生。这个软件的名称，来自于荷兰设计师赫里特·里特费尔德 (Gerrit Rietveld)。最早它是由 Python 写成，在第二版后，改成用 Java 与 SQL。使用 Web Toolkit 来产生前端的 JavaScript。

1.2 Gerrit 搭建

1.2.1 安装

安装需要为 gerrit 设置一个专用户，之后所有的 gerrit 操作都需要在这个账户下操作。使用下列命令创建：

```
$sudo adduser gerrit
```

新建的用户是没有运行 sudo 的权限的，为了以后操作方便，我们需要将 gerrit 用户加入到 sudoers。

```
$sudo vim /etc/sudoers
```

找到下面两行：

```
# Allow members of group sudo to execute any command
```

```
%sudo ALL=(ALL:ALL) ALL
```

在后面加入：

```
gerrit ALL=(ALL:ALL) ALL
```

然后保存关闭。切换到 gerrit 用户：

```
$sudo su gerrit
```

安装包放置在了 10.20.25.126 服务器上，可以使用 ssh 协议下载，或者自己网上下载，文件名为 gerrit-

2.8.4.war。

```
$scp root@10.20.25.126:/work/gerrit-2.8.4.war /work/
```

```
$cd /work
```

上面也介绍了 gerrit 是用 Java 写的，所以运行需要依赖 Java 环境，如果没有装的请自行安装。本机所装的版本是：java version "1.6.0_31"。

```
$java -jar gerrit-2.8.4.war init -d review_site
```

之后会出现一些配置的对话，可以使用默认的配置，因为之后我们会通过修改文件的方式来修改配置。

运行完毕后会在工作目录生成一个新的文件夹 review_site。里面的目录结构如下：

```
bin  cache  data  db  etc  git  lib  logs  plugins  static  tmp
```

我们只需关注以下几个文件夹：

bin：用来存放 gerrit 的启动脚本。

db：用来存放 H2 数据库，当然只有使用了 gerrit 默认的数据库时才会用到。我们之后会配置成 mysql。

etc：用来存放配置文件。通过修改该文件夹里的配置文件可以重新配置 gerrit。

git：这个就是传说中的代码仓库。这个文件夹里存放的都是项目的镜像。

lib：gerrit 运行所需的 java 库。运行 mysql 所需要的两个驱动库就需要放在这个目录下。不然会连接不上数据库。

logs：gerrit 的运行一些信息会打印到该文件目录下的文件。

因为我们使用的是 mysql，所以还需要安装 mysql。如果本机没有安装 mysql 需要运行如下命令安装：

```
$sudo apt-get install mysql-client-5.5
```

```
$sudo apt-get install mysql-server-5.5
```

安装完成后还需要拷贝两个 .jar 文件到 review_site/lib 目录下：bcprov-jdk16-144.jar 和 mysql-connector-java-5.1.21.jar。着两个文件可以从 10.20.25.126 服务器的 gerrit 目录下拷贝过来，拷贝指令用：

```
$scp root@10.20.25.126:/work/gerrit/review_site/lib/bcprov-jdk16-144.jar /work/review_site/lib/
```

```
$scp root@10.20.25.126:/work/gerrit/review_site/lib/mysql-connector-java-5.1.21.jar /work/review_site/lib/
```

至此我们的安装完成。下面就需要重新配置 gerrit。

1.2.2 配置

配置 gerrit 需要修改 review_site 目录下的 gerrit.config 文件。

```
$cd /work/review_site/
```

```
$vim etc/gerrit.config
```

将文件的内容修改如下：

```
[gerrit]
```

```
basePath = git
```

```
canonicalWebUrl = https://10.20.25.126:8080 #web 端访问的地址，改成自己的 ip
```

```
[database]
```

```
type = mysql #数据库类型
```

```
hostname = localhost #数据库地址，使用本地
```

```
database = reviewdb #数据库名字
```

```
username = gerrit #数据库登录的用户名
```

```
[auth]
```

```
type = LDAP #认证方式，gerrit 支持多种认证方式，我们使用 LDAP
```

```
[ldap]
```

```
server = ldap://192.168.100.129:389 #ldap 服务器的地址
```

```
username = uid=searchsnake,dc=archermind,dc=com
```

```
accountBase = dc=archermind,dc=com
```

```
accountPattern = (uid=${username})
```

```
accountFullName = displayName
```

```
accountSshUserName = uid
```

```
accountEmailAddress = mail2
```

```
groupBase = dc=archermind,dc=com
```

[sendemail] #配置邮件信息，即 Gerrit 系统使用的邮箱

```
enable = true  
smtpServer = smtp.archermind.com  
smtpServerPort = 25  
smtpUser = mailman  
from = mailman@archermind.com
```

[container]

```
user = gerrit  
javaHome = /usr/lib/jvm/java-6-openjdk-amd64/jre
```

[sshd]

```
listenAddress = *:29418
```

[httpd]

```
listenUrl = proxy-https://*:8080/
```

[cache]

```
directory = cache
```

上面配置了数据库类型为 mysql，并且指定了数据库的用户名和数据库的名字，所以下一步就是初始化数据库。

```
$mysql -u root -p
```

```
mysql>create database reviewdb;
```

```
mysql>insert into mysql.user(Host,User>Password)           #创建用户
```

```
    >values
```

```
    >('localhost','gerrit',password('archermind'));
```

```
mysql>flush privileges;
```

```
mysql>grant all privileges on reviewdb.* to gerrit@localhost identified by 'archermind';   #授权
```

```
mysql>flush privileges;
```

```
mysql>exit;
```

```
$mysql -u gerrit -p           #验证以下能否登录
```

最后一步配置修改 review_site/etc/secure.config 文件，加入以下内容：

```
[database]
```

```
password = archermind
```

```
[sendemail]
```

```
smtpPass = "ArcherMai209#man"
```

```
[ldap]
```

```
password = print,5screen      #LDAP 服务器登录的密码
```

这三个都是密码，一个是数据库的密码，一个是邮箱的密码，最后一个是 LDAP 服务器的密码。

配置完后重新初始化 Gerrit 目录：

```
$cd /work
```

```
$java -jar gerrit-2.8.4.war init -d review_site
```

1.2.3 启动

启动的脚本放在 review_site/bin 文件夹下，运行下面的命令启动：

```
$cd /work/review_site/bin
```

```
./gerrit.sh start
```

当看到 **Starting Gerrit Code Review: OK** 时表示启动成功。如果失败了，通过 logs 下的 error_log 文件查找原因。

运行下面命令可以查看监听端口信息：

```
$sudo netstat -ltnp | grep -i gerrit
```

每次开机需要手动启动 Gerrit 太麻烦，如果放在远程电脑更不方便，所以需要将 Gerrit 设置成开机自启动。

```
$sudo ln -snf /work/review_site/bin/gerrit.sh /etc/init.d/gerrit.sh
```

```
$sudo ln -snf /etc/init.d/gerrit.sh /etc/rc2.d/s90gerrit
```

```
$sudo ln -snf /etc/init.d/gerrit.sh /etc/rc3.d/s90gerrit
```

在/etc/default/目录下创建文件 gerritcodereview 文件，写入下面的内容：

```
GERRIT_SITE=/work/review_site
```

```
NO_START=0
```

现在打开网页就可以看到 gerrit 的 web 端了。网址是配置文件里的 canonicalWebUrl。

1.2.4 创建管理员账户

gerrit 会将第一个创建的用户作为管理员用户，ccount ID 为 1000000 的就是管理员。LDAP 认证方式会自动将登录的用户加入到数据库，第一个登录的用户将自动设为管理员，ccount ID 为 1。

当然如果使用 OPENID 认证方式是可以注册用户的，直接在 Web 端注册即可。

2 git 与 gerrit 的协调工作

2.1 git 一些基本命令介绍

git 是常用的版本控制工具，关于它的知识推荐两本书《Git Pro》和《Git 权威指南》。这里通过一个简单的本地代码仓库来介绍命令和一些基础知识，这些命令能够满足我们的日常开发。

第一个要配置的是你个人的用户名称和电子邮件地址。这两条配置很重要，每次 Git 提交时都会引用这两条信息，说明是谁提交了更新，所以会随更新内容一起被永久纳入历史记录：

```
$ git config --global user.name "Wang Bo"
$ git config --global user.email "bo.wang@archermind.com"
$ cd work
$ mkdir example
$ cd example/
$ git init
$ ls -a
```

可以看到该文件夹下生成了一个隐藏文件夹.git，这个文件夹实际上就是我们所称为的 git 仓库，所有的信息都存储在该目录下。我们新建的 example 文件夹称为工作树。

```
archermind@archermind:~$ cd /work
archermind@archermind:/work$ mkdir example
archermind@archermind:/work$ cd example/
archermind@archermind:/work/example$ git init
Initialized empty Git repository in /work/example/.git/
archermind@archermind:/work/example$ ls -a
.  .git
archermind@archermind:/work/example$
```

```
$ touch README
```

`$git status`

这时你会看到 git 输出一些如下图的信息：

```
archermind@archermind:/work/example$ touch README
archermind@archermind:/work/example$ git status
# On branch master
#
# Initial commit
#
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#       README
nothing added to commit but untracked files present (use "git add" to track)
archermind@archermind:/work/example$
```

On Branch master 说名下面的信息来自主分支。git 工作有三种状态，第一种状态是未更新，第二种是暂存，第三种是提交。由于 Git 在仓库中添加文档时并非是简单的文档复制过去，势必要将所添加文档进行一番处理，生成 Git 仓库所能接受的数据格式，Git 称这个过程为“take a snapshot”（生成快照）。所谓的快照就是将你修该的内容加入到暂存区域：

`$git add README`

`$git status`

信息改变了：

```
archermind@archermind:/work/example$ git add README
archermind@archermind:/work/example$ git status
# On branch master
#
# Initial commit
#
# Changes to be committed:
#   (use "git rm --cached <file>..." to unstage)
#
#       new file:   README
archermind@archermind:/work/example$
```

所生成的快照被存放在一个临时存储区，称为索引。使用 `git -commit` 命令可将索引提交至仓库中，这个过程称为提交。每一次提交意味着版本在进行一次更新。输入简约的版本更新说明是非常必要的，对于简短的版本更新说明可以使用“-m”选项，如 `git commit -m “你的版本更新信息”`：

`$git commit -s -m "Create README"`

`$git status`

此时会显示：

nothing to commit (working directory clean)

这说明暂存的区域内容已经提交到代码仓库，工作目录很干净。可以通过下面的指令来查看仓库的提交信息：

`$git log`

```
archermind@archermind:/work/example$ git log
commit 2d6d089a35f6a9eef3afca495142d606802c8e73
Author: Wang Bo <bo.wang@archermind.com>
Date: Tue Jul 1 10:02:39 2014 +0800

    Create README

Signed-off-by: Wang Bo <bo.wang@archermind.com>
archermind@archermind:/work/example$
```

Git 使用 SHA-1 算法计算数据的校验和，通过对文件的内容或目录的结构计算出一个 SHA-1 哈希值,作为指纹字符串。该字符串由 40 个十六进制字符(0-9 及 a-f)组成，看起来就像是：

`2d6d089a35f6a9eef3afca495142d606802c8e73`

Git 的工作完全依赖于这类指纹字符串，所以你会经常看到这样的哈希值。实际上，所有保存在 Git 数据库中的东西都是用此哈希值来作索引的，而不是靠文件名。

`$echo 'hello world' >README`

`$git add .`

`$git diff --cached`

diff 是用来查看暂存起来的数据与上次提交的数据之间的差异，输入上述指令后可以看到如下信息：

```
diff --git a/README b/README
index e69de29..3b18e51 100644
--- a/README
+++ b/README
@@ -0,0 +1 @@
+hello world
```

上面的命令是本地开发所需掌握的基本命令。总结一下上面的工作流程：



2.2 远程代码仓库

上面只是介绍了本地建仓库开发的基本命令，很多时候，一个项目由团队里的多个人完成，所以代码仓库一般放在远程服务器上。Gerrit 之前也介绍过了，它可以用来管理代码仓库，所以下一步就是建立起代码仓库与 gerrit 的联系，让他们相互配合，完成我们的开发任务。上面是介绍了 gerrit 的服务器搭建方法，10.20.25.126 服务器已经使用上面的方法搭建好了，我们将利用该服务器进行下面内容的讲解。

```
$cd /work
```

```
$git clone example/.git --mirror
```

运行上面的命令会生成一个 example.git 的文件夹，这个文件夹就是仓库的镜像。我们把它压缩放到服务器上
面：

```
$tar -zcvf example.tgz example.git
```

```
$scp example.tgz root@10.20.25.126:/work
```

```
$ssh root@10.20.25.126
```

```
$su gerrit
```

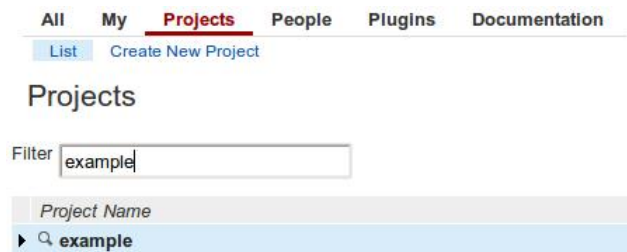
```
$tar -zxvf /work/example.tgz -C /work/gerrit/review_site/git/
```

```
$/work/gerrit/review_site/bin/gerrit.sh restart
```

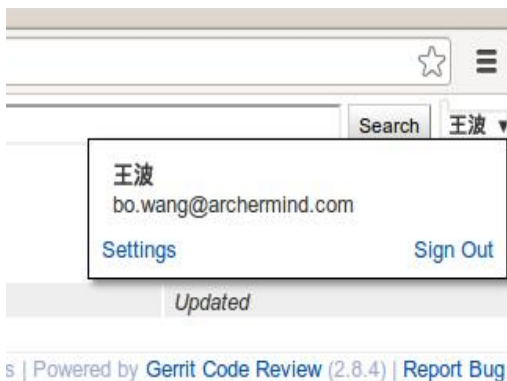
```
$exit
```

```
$exit
```

重启了 gerrit 之后我们就可以在 web 端看到刚刚添加进去的项目：



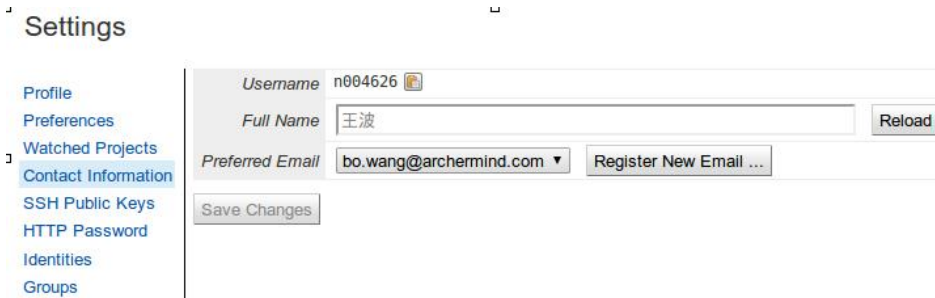
在浏览器中输入服务器网址 <http://10.20.25.126:8080>，用户名、密码和邮箱的用户、密码相同（不知道密码的去找系统管理部重置密码）。登录后点击右上角的名字，选择 setting：



Settings



记住 username。如果 Email Address 为空，点击左边的 Contact Information。



点击 Register New Email...注册邮箱，别填错了。注册完之后会收到系统的邮件，需要点击邮件中的连接来激活。

```
$sudo adduser <username>
```

其中<username>就是之前让大家记住的哪个 Username。新建完成后输入如下命令：

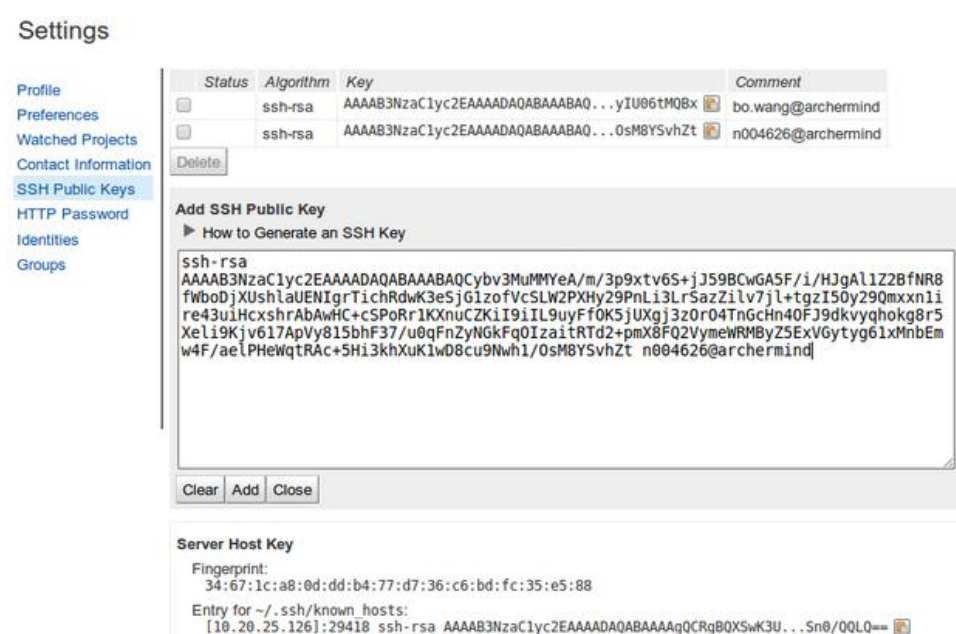
```
$sudo su <username>
```

```
$ssh-keygen
```

一路按 Enter 键直至生成完毕。

```
$cat ~/.ssh/id_*.pub
```

将打印出的信息复制。打开 Gerrit，点击 SSH Public Keys。将复制的 Key 粘贴进去：



OK，key 已经上传就可以下载刚才新建的项目了。注意，下载和开发等工作都是需要在刚建的用户上进行的。

因为 git 在 push 和 clone 的时候需要指明远端的用户名，如果没有用户名则会默认使用本机的用户名，为了开发方便，所以才建了和 gerrit 远端账户相同的本地用户。

```
$cd ~/
```

```
$git clone ssh://10.20.25.126:29418/example
```

此时会在主目录下多出一个文件夹 example，进入文件夹你会发现和我们之前在/work 目录下建的一模一样。

```
$cd example/
```

```
$git branch
```

这条命令是列出分支，前面带*的表示当前分支。

```
$git checkout -b example_branch
```

```
$git branch
```

git checkout 是用来切换分支，加上-b 后表示新建分支，通过 git branch 可以看到当前分支已经切换到了 example_branch。

```
$echo 'This is example' >README
```

```
$git commit -a -s -m 'print some information to README'
```

```
$git push origin example_branch:refs/for/master
```

这条指令是将当前分支的修改推送到服务器，之后交给服务器去审核，审核通过了才可以并入 master 分支。

至于谁审核，这个需要用管理原账户去配置相应的权限，可以指定 review 的人。但是此时我们会发现如下的错误：

```
! [remote rejected] example_branch -> refs/for/master (missing Change-Id in commit message footer)
```

从错误上分析我们可以看出是提交的时候缺少了 Chang-Id。我们用 git log 查看信息的时候也没有显示这个信息，那么这个 id 从哪来？

通过查找资料我们会发现这个 ID 是钩子脚本自动生成的（关于钩子的作用和信息请自行查阅资料），钩子脚本存放的地址就是.git/hooks 目录。

```
$scp -P 29418 -p 10.20.25.126:/hooks/commit-msg .git/hooks/
```

上面的指令是用来下载钩子脚本的，下载完后运行：

```
$git reset --hard origin/master
```

```
$echo "This is example" >README
```

```
$git commit -a -s -m 'Now, we can see change-id'
```

```
$git log
```

```
commit d04c74017a50afb15758397f61939a76567a1627
Author: wangbo <bo.wang@archermind.com>
Date: Tue Jul 1 14:45:42 2014 +0800

    Now, we can see change-id

Change-Id: I91b4c513f9e003c167b3d8685169f523c7968f1e
Signed-off-by: wangbo <bo.wang@archermind.com>
```

上面命令的第一行是回退版本，会丢失信息的，慎用。现在有了 Chang-Id 就可以提交代码到远程仓库了。

`$git push origin example_branch:refs/for/master`

如果出现权限问题，那么请看下一节内容。如果顺利推上去了，那么说明自己已经有推送到该引用的权限了。

2.3 权限管理

用管理员账户登录 Gerrit，在 People 选项下选择新建分组，建如下三个分组：

Group PSD_Group

GeneralMembers

Group UUID

bf82bfc885ffa8d6712c2cc847b03056236dfa01

PSD_Group

Rename Group

Owners

PSD_Group

Change Owner

Description

PSD培训小组

Save Description

Group Options

☐ Make group visible to all registered users.

Save Group Options

Group review

GeneralMembers

Group UUID

3c207f46c507c0f3d34c4f6cb2592dc77c0c77f1

review

Rename Group

Owners

review

Change Owner

Description

具有设置审核通过的权限

Save Description

Group Options

☐ Make group visible to all registered users.

Save Group Options

Group Verified

General

Members

Group UUID

fa173e37c45459a99b27702fa8777f7b9f27fbc7

Verified

Rename Group

Owners

Verified

Change Owner

Description

设置评审设置通过

Save Description

Group Options

☐ Make group visible to all registered users.

Save Group Options

PSD_Group : 开发小组，即开发团队的人员。具有推送代码的权限。

Review : 具有审核代码权限的人员所在的组。

Verified : 具有确认和合并分支权限的人员所在的组。

选择 Projects 下的 All-Projects，选择 Access，按如下分配权限：

Reference: refs/*

Read

ALLOW Administrators

ALLOW Anonymous Users

Label Code-Review

-2 +2 review

Submit

ALLOW Verified

Exclusive

Reference: refs/for/*

Push

ALLOW PSD_Group

Exclusive

具有 review 权限的人可以在 All 下的 open 中看到需要 review 的代码。可以指定 review 的人，指定 review 的人会受到系统的邮件，review 的人员需要下载代码，编译，评估。然后确认是否使用这个修改。具有 Submit 权限的人才可以将代码合并到主分支。

AllMyProjectsPeoplePluginsDocumentation

OpenMergedAbandoned

Change-Id: I91b4c513f9e003c167b3d8685169f523c7968f1e

Owner 王波

Project example

Branch master

Topic

Uploaded Jul 1, 2014 2:46 PM

Updated Jul 1, 2014 2:46 PM

Submit Type Merge if Necessary

Status Review in Progress

Commit Message

Now, we can see change-id

Change-Id: I91b4c513f9e003c167b3d8685169f523c7968f1e

Signed-off-by: wangbo <bo.wang@archermind.com>

Need Code-Review

Add Reviewer

Dependencies

Reference Version: Base

Patch Set 1

d04c74017a50afb15758397f61939a76567a1627

Author wangbo <bo.wang@archermind.com> Jul 1, 2014 2:45 PM

Committer wangbo <bo.wang@archermind.com> Jul 1, 2014 2:45 PM

Parent(s) 2d6d089a35f6a9eef3afca495142d606802c8e73 Create README

Download checkout | pull | cherry-pick | patch | Anonymous HTTP | SSH | HTTP | git fetch ssh://n004626@10.20.25.126:29418/example refs/changes/35/35/1 && git checkout FETCH_HEAD

Review

Cherry Pick To

Abandon Change

File Path	Comments	Size	Diff	Reviewed
Commit Message			Side-by-SideUnified	
M README		+1, -0	Side-by-SideUnified	
		+1, -0	All Side-by-SideAll Unified	

Comments

王波

Uploaded patch set 1.

Add Comment

2:46 PM

当然你也可以给自己绕过评审的权限。具体的是修改项目的 Access。

3 多项目管理与 repo 工具。

3.1 repo 介绍以及基本命令的使用

安卓源码相当庞大，里面的一个独立功能就是一个项目，一个安卓源码有两三百个项目，每个项目都有一个 git 仓库，试想，要管理起来得多麻烦。于是就诞生了 repo 工具。谷歌开发 repo 不是为了替代 git，而是对 git 命令使用 python 脚本进行了封装，方便了开发。

关于 repo 有这么一则小故事：Android 之父安迪·鲁宾在回应乔布斯关于 Android 太开放导致开发维护更麻烦的言论时，在 Twitter1 上留了下面这段简短的话：

the definition of open: "mkdir android cd android repo init -
u git://android.git.kernel.org/platform/manifest.git repo sync make"

是的，就是 repo 让 Android 的开发变得如此简单。

关于 repo 的使用命令这里给一个链接：

<http://blog.csdn.net/skyflying2012/article/details/23742683>

希望后人能够将基本命令整理加到这部分内容来。

3.2 manifests.xml

repo sync 时会根据 manifests.xml 文件的内容下载代码，实际上它是指向.repo/manifests 里面的 default.xml 文件。这里简单介绍以下格式。

repo manifest XML 可以包含下面的元素。

(1) manifest: 最顶层的 XML 元素。

(2) remote 元素: 设置远程 git 服务器的属性，包括下面的属性

name: 远程 git 服务器的名字，直接用于 git fetch, git remote 等操作。

alias: 远程 git 服务器的别名，如果指定了，则会覆盖 name 的设定。在一个 manifest 中，name 不能重名，但 alias 可以重名。

fetch: 所有 projects 的 git URL 前缀。

review: 指定 Gerrit 的服务器名，用于 repo upload 操作。如果没有指定，则 repo upload 没有效果。

Example:

```
<remote fetch="ssh://git.example.com" name="test"review="gerrit.example.com"/>
```

(3) default 元素: 设定所有 projects 的默认属性值，如果在 project 元素里没有指定一个属性，则使用 default 元素的属性值。

remote: 之前定义的某一个 remote 元素中 name 属性值，用于指定使用哪一个远程 git 服务器。

revision: git 分支的名字，例如 master 或者 refs/heads/master。

sync_j: 在 repo sync 中默认并行的数目。

sync_c: 如果设置为 true , 则只同步指定的分支(revision 属性指定), 而不是所有的 ref 内容。

sync_s: 如果设置为 true , 则会同步 git 的子项目

Example:

```
<default remote="main" revision="platform/main"/>
```

(4) manifest-server 元素: 只能有一个该元素。它的 url 属性用于指定 manifest 服务的 URL , 通常是一个 XML RPC 服务。

它要支持一下 RPC 方法 :

GetApprovedManifest(branch, target): 返回一个 manifest 用于指示所有 projects 的分支和编译目标。target 参数来自环境变量 TARGET_PRODUCT 和 TARGET_BUILD_VARIANT , 组成\$TARGET_PRODUCT-\$TARGET_BUILD_VARIANT。

GetManifest(tag): 返回指定 tag 的 manifest

(5) project 元素 : 指定一个需要 clone 的 git 仓库。

name: 唯一的名字标识 project , 同时也用于生成 git 仓库的 URL。格式如下 :

`${remote_fetch}/${project_name}.git`

path: 可选的路径。指定 git clone 出来的代码存放在本地的子目录。如果没有指定 , 则以 name 作为子目录名。

remote: 指定之前在某 remote 元素中的 name。

revision: 指定需要获取的 git 提交点 , 可以是 master, refs/heads/master, tag 或者 SHA-1 值。

groups: 列出 project 所属的组 , 以空格或者逗号分隔多个组名。所有的 project 都自动属于"all"组。每一个 project 自动属于

name:'name' 和 path:'path'组。例如<project name="monkeys" path="barrel-of"/> , 它自动属于 default, name:monkeys, and path:barrel-of 组。如果一个 project 属于 notdefault 组 , 则 , repo sync 时不会下载。

sync_c: 如果设置为 true , 则只同步指定的分支(revision 属性指定), 而不是所有的 ref 内容。

sync_s: 如果设置为 true , 则会同步 git 的子项目。

upstream: 在哪个 git 分支可以找到一个 SHA1。用于同步 revision 锁定的 manifest(-c 模式)。该模式可以避免同步整个 ref 空间。

annotation: 可以有多个 annotation , 格式为 name-value pair。在 repo forall 命令中这些值会导入到环境变量中。

remove-project: 从内部的 manifest 表中删除指定的 project。经常用于本地的 manifest 文件 , 用户可以替换一个 project 的定义。

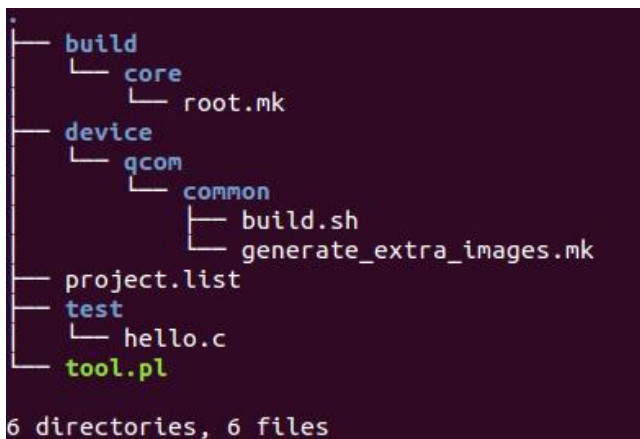
Example:

```
<project groups="aosp" path="device/driver/armv7"
    revision="600aab270ce712b62b268055737cabcded59bf04"/>
```

(6) include: 通过 name 属性可以引入另外一个 manifest 文件(路径相对与 manifest repository's root)。

3.3 repo+git+gerrit

(1) 新建工程目录。目录结构和文件如下 (高通平台源码的部分目录结构) :



```
├── build
│   └── core
│       └── root.mk
├── device
│   └── qcom
│       └── common
│           ├── build.sh
│           └── generate_extra_images.mk
├── project.list
├── test
│   └── hello.c
└── tool.pl
6 directories, 6 files
```

(2) 在 project.list 中添加如下内容 :

build

device/qcom/common

test

(3) 复制附录的脚本内容到 tool.pl 文件。

```
$chmod +x tool.pl
```

```
$vim tool.pl
```

\$DIR_NAME 是远端存放的目录。

\$fetch 是远端的地址。

\$review 是代码审核的地址，repo upload 会访问这个地址。

```
$/tool.pl
```

运行完会在目录下生成一个压缩包，将这个压缩包传到服务器并解压到 Gerrit 的 git 目录下。重启 Gerrit。方法和单个 git 仓库上传是一样的，此处不再赘述。

(4) 验证下载源码

```
$su <username>
```

```
$cd ~/
```

```
$mkdir Test
```

```
$cd Test
```

```
$curl http://mirror.core.archermind.com/android/aosp/repo >repo
```

```
$chmod +x repo
```

```
$/repo init -u ssh://10.20.25.126:29418/Test/manifests
```

```
$/repo sync
```

现在你的 Test 目录下存放的就是你的所有源码。可以用 repo 的相关命令来操作，也可以使用 git 命令来操作一个项目。

4 FAQ

Q：本地服务器搭建后登录，为什么网页端的用户名显示“？”？

A：因为数据库 mysql 默认的使用编码格式不是 utf8，导致不支持汉字。修改数据库的编码显示格式为 utf8 即可。

下面是一种解决方法，在创建数据库的时候：

```
mysql>CREATE DATABASE `test` CHARACTER SET 'utf8' COLLATE 'utf8_general_ci';
```

Q : gerrit 登录慢，为什么？

A : 我们搭建的服务器使用的是 LDAP 认证方式，登录时需要连接 LDAP 服务器进行认证，所以速度慢，登录在 10 秒到 1min 钟内都是正常的。本文提到的 LDAP 服务器是管理公司邮箱帐号的服务器，进行过优化，所以登录速度上很快。建议使用本文提到的 LDAP 服务器配置。

Q : git push 的时候为什么会出现权限问题？

A : 这边有两个地方会引起权限问题。一是 SSH Key 问题 (public key)，二是搭建的时候项目不归 gerrit 账户所有。对于第一种情况应做如下检查：

(1) 确保你是在和 gerrit 登录名相同的账户下操作的

(2) 确保 key 添加到了 gerrit

(3) 确保你添加的 key 是在和 gerrit 登录名相同的账户下生成的

(4) git 的邮箱一定要和 gerrit 上注册的邮箱相同，统一使用公司的邮箱，不然系统给你发邮件收不到。这个问题不会影响下代码，但是在 push 的时候会引起权限问题。

Q : 为什么使用 repo upload 命令的时候会出现权限问题？

A : 通过查找 repo 的帮助文件可以看到下面几句话：

review.URL.username:

Override the username used to connect to Gerrit Code Review. By default the local part of the email address is used.

The URL must match the review URL listed in the manifest XML file, or in the .git/config within the project. For example:

[remote "origin"]

```
url = git://git.example.com/project.git
```

```
review = http://review.example.com/
```

```
[review "http://review.example.com/"]
```

```
autoupload = true
```

```
autocopy = johndoe@company.com,my-team-alias@company.com
```

所以 upload 的时候使用的用户名默认的是邮箱地址，自然不对，出现 publickey 问题。

运行下面命令解决：

```
$git config --global review.http://10.20.25.126:8080.username <username>
```

其中<username>要和你的 Gerrit 帐号中的 username 相同。

添加配置后再使用 repo upload 时就不会出现问题了，直接推送到 Gerrit 进行审核。

Q：运行 \$mysql -u root -p 时出现问题：ERROR 2002 (HY000)，为什么？

A：是因为本机没有安装 mysql-serve-5.5。运行下面命令可解决：

```
$sudo apt-get install mysql-serve-5.5
```

Q：Gerrit 启动失败，\$cat logs/error_log 显示无法连接数据库，怎么解决？

A：出现上述问题有两个可能的原因：

1、配置出现错误，比如密码错误，Gerrit 账户名拼写错误等等。请仔细检查配置。

2、数据库连接器出现问题。运行下面命令解决：

```
$scp root@10.20.25.126:/work/gerrit/review_site/lib/bcprov-jdk16-144.jar /work/review_site/lib/
```

```
$scp root@10.20.25.126:/work/gerrit/review_site/lib/mysql-connector-java-5.1.21.jar /work/review_site/lib/
```

Q：以 mysql 搭建 Gerrit 服务器时，若初次登录显示为非法，再次登录方可进入，有时还无法得到管理员权限。

A：上述问题最直接的方法就是改用 Gerrit 默认的 H2 数据库。

5 参考资料

文档名称	版本	作者	出版单位	备注
《Pro git》				
《Git 权威指南》				

6 修改记录

版本	修改时间	修改者	备注
V1.0	2014-6-30	王波	最初版
V1.1	2014-8-14	王波	调整格式；修改了服务器端最新配置；增加 FAQ。

附录 A :

tool.pl 的内容 :

```
#!/usr/bin/perl -w
```

```
$DIR_NAME = "Test";
```

```
$fetch = "ssh://10.20.25.126:29418/";
```

```
$review = "http://10.20.25.126:8080/";
```

```
$curdir=`pwd`;
```

```
chomp $curdir;
```

```
$destdir = $curdir."/$DIR_NAME";
```

```
$list_path = $curdir."/project.list";
```

```
$project_info;
```

```
open (LFILE, $list_path) or die "Can't open project.list";
```

```
@list =<LFILE>;
```

```
foreach $git (@list)
```

```
{
```

```
    chomp $git;
```

```
    my $dir= $curdir."/".$git;
```

```
    chdir ($dir);
```

```
    $destgit = $destdir."/".$git.".git";
```

```
    if(-e ".git") {
```

```
        print "rm $dir/.git\n";
```

```
        system("rm -rf .git/");
```

```
    }
```

```
    system("git init");
```

```
    system("git add .");
```

```

system("git commit -a -m \"Source Base.\");

system("git clone $dir --mirror $destgit");

}

foreach $path(@list) {
    my $one_project_info;
    chomp $path;
    if($path eq "device/qcom/common") {
        $one_project_info = " <project name=\"$DIR_NAME/$path\" path=\"$path\"/>\n";
        $one_project_info = $one_project_info." <copyfile dest=\"build.sh\" src=\"build.sh\"/>\n";
        $one_project_info = $one_project_info." <copyfile
dest=\"vendor/qcom/build/tasks/generate_extra_images.mk\" src=\"generate_extra_images.mk\"/>\n";
        $one_project_info = $one_project_info." </project>\n";
    }elseif($path eq "build") {
        $one_project_info = " <project name=\"$DIR_NAME/$path\" path=\"$path\"/>\n";
        $one_project_info = $one_project_info." <copyfile dest=\"Makefile\" src=\"core/root.mk\"/>\n";
        $one_project_info = $one_project_info." </project>\n";
    }else {
        $one_project_info = " <project name=\"$DIR_NAME/$path\" path=\"$path\"/>\n";
    }
    $project_info = $project_info.$one_project_info;
}

$destdir = $curdir."/$DIR_NAME"/manifests";

system("mkdir $destdir");

open MFILE, ">$destdir/default.xml";

print MFILE << "EOF";

```



```
<?xml version="1.0" encoding="UTF-8"?>

<manifest>

  <remote fetch="$fetch" name="origin" review="$review"/>

  <default remote="origin" revision="master"/>

$project_info

</manifest>

EOF
```

```
chdir $destdir;

system("git init");

system("git add .");

system("git commit -a -m \"Create By perl\"");

system("git log");

chdir "../";

system("git clone manifests/.git --mirror manifests.git");

system("rm -rf manifests/");

chdir "../";

system("tar -zcvf $DIR_NAME.tgz $DIR_NAME");

system("rm -rf $DIR_NAME");
```