# 1.git、repo、gerrit、github

**git**

git是什么？ Git是分布式版本控制系统。SVN是集中式版本控制系统。

**repo**

repo命令行工具对Git命令进行了封装，可以管理多个git项目，从而更好地对代码进行集中式管理。

repo是Google用Python脚本写的调用Git的脚本，用于下载、管理Android项目的软件仓库。

**gerrit**

Gerrit是一个建立在Git版本控制系统之上，基于Web的代码审查工具。

Gerrit是一种免费、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。

代码审核（Code Review）是软件研发质量保障机制中非常重要的一环，但在实际项目执行过程中，却因为种种原因被Delay甚至是忽略。在实践中，免费、开放源代码的代码审查软件Gerrit是一个很好的选择。

它在传统的源码管理协作流程中强制性引入代码审核机制，通过人工代码审核和自动化代码验证过程，将不符合要求的代码屏蔽在代码库之外，确保核心代码多人校验、多人互备和自动化构建核验。

Gerrit适用性：几乎任何需要正式发布的项目都应当使用Gerrit来进行代码审查，如果Team中有新人，必须使用Gerrit确保代码质量。

Gerrit上进行代码审查，确认入库

Verify：工程里面接入了jenkins自动verify，结果可在上图红框内展示verify结果。

review代码，提交入库。

Gerrit官网：https://www.gerritcodereview.com/

**github**

GitHub是一个面向开源及私有软件项目的托管平台，因为只支持git 作为唯一的版本库格式进行托管，故名GitHub。

GitHub于2008年4月10日正式上线，除了Git代码仓库托管及基本的 Web管理界面以外，还提供了订阅、讨论组、文本渲染、在线文件编辑器、协作图谱（报表）、代码片段分享（Gist）等功能。

# 1.集成gerrit开发环境的代码提交

git提交后，使用git log查看提交记录，如下：

$ git log

commit 13d381265ffff88088e1af88d0e2c2c1143743cd

Author: John Doe <john.doe@example.com>

Date: Thu Dec 16 10:15:48 2010 +0100

another commit

commit ca45e125145b12fe9681864b123bc9daea501bf7

Author: John Doe <john.doe@example.com>

Date: Thu Dec 16 10:12:54 2010 +0100

one commit

提交记录包含commit SHA、Author、Date、Comment等信息。

如果集成gerrit开发环境，那么git log信息会增加一条Change-Id。

$ git log

commit 13d381265ffff88088e1af88d0e2c2c1143743cd

Author: John Doe <john.doe@example.com>

Date: Thu Dec 16 10:15:48 2010 +0100

another commit

Change-Id: I93478acac09965af91f03c82e55346214811ac79

commit ca45e125145b12fe9681864b123bc9daea501bf7

Author: John Doe <john.doe@example.com>

Date: Thu Dec 16 10:12:54 2010 +0100

one commit

Change-Id: I93478acac09965af91f03c82e55346214811ac79

commit SHA是一个40位的字符串，Change-Id是一个41位的字符串。

每一次git commit都会为这次提交生成一个commit SHA；

使用git commit --amend会生成一个新的commit SHA。-- 只要使用git commit，不管git commit还是git commit --amend都会生成一个新的commit SHA。

对于集成gerrit的开发环境，git commit会生成一个新的Change-Id，但是git commit --amend不会生成一个新的Change-Id，而是沿用之前git commit时生成的Change-Id。

使用git commit --amend提交gerrit服务器，也就是说使用相同的Change-Id提交，会使用原有的gerrit提交，但生成一个新的Patch Set。

🡪如果想要修改提交到gerrit服务器的提交，但不想生成一个新的提交，那就使用相同的Change-Id。Change-Id可以手动修改？

# 1.Gerrit的使用及常用git内容

<https://blog.csdn.net/dovar_66/article/details/60873639>

**1.Gerrit是什么？**

Gerrit实际上一个Git服务器，它为在其服务器上托管的Git仓库提供一系列权限控制，以及一个用来做Code Review是Web前台页面。当然，其主要功能就是用来做Code Review。

2.配置ssh

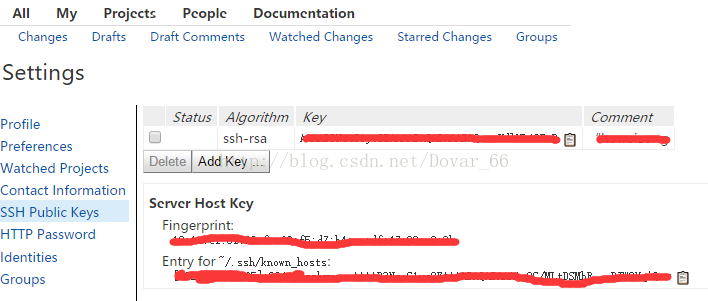
1.使用gitbash查看是否已经有了ssh密钥：cd ~/.ssh

2.生成ssh密钥：

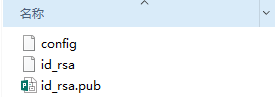
$ ssh-keygen -t rsa -C “你想设置的用户名”

3.生成密钥后要求设置密码，可按回车键跳过

4.登录网址后，将生成的.ssh文件夹下的公钥id\_rsa.pub中的内容粘贴到setting--SSH Public Keys中即可



5.你可能还需要给ssh添加一个配置文件config



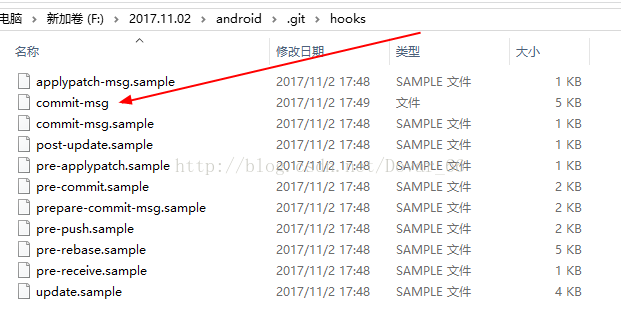
config文件中具体需要写入什么配置请询问你们搭建gerrit环境的同事

**3.gerrit的日常使用**

使用git命令拉取代码，git命令从gerrit项目中获取，获取clone地址时选中“clone with commit-msg hook”和“SSH”。打开gitbash进行代码下载。



下载完成后注意检查commit-msg文件是否下载成功，如果没下载到，请自行前往gerrit下载，并放到项目目录下，如下图



commit-msg文件是用于为每次commit自动生成唯一的changeId。

Push命令：git push origin Head:refs/for/branch  这里的branch是指你想要提交到的分支名

4.使用中可能遇到的一些问题：

1.**gerrit通过给每个独立的commit加上一个唯一的changId来保证有序的代码提交和review，多次commit携带相同的changid视为对同一次commit的修改**。如果commit没有携带changid的话则无法push成功，所以当push失败的时候第一件事应该是检查当前的commit是否都带有changid。如果仅最后一次commit没有changid，可使用git commit --amend 然后直接：wq退出编辑即可，此命令的作用就是将最新一次commit指向上一次带有changid的commit,如果含多条commit都没有changid的话，可以checkout这些commit然后逐条添加changid，或者rebase到最后一次有效的commit然后重新编辑提交。

2.vi编辑器，按esc退出编辑模式，然后冒号:wq即可

3.可能遇到由于某次commit携带的用户信息与gerrit网站上配置的不同而导致push失败的，使用git更换本地账号**：**

        命令分别为：git config  --global user.name 你的目标用户名；

               git config  --global user.email 你的目标邮箱名;

当然，你还可以直接修改git的配置文件的方式来进行修改：打开全局的.gitconfig文件的命令为：vi ~/.gitconfig; 然后在文件中直接修改即可。

# 1.git修改前一次提交的方法(注意保持Change-Id不变)

<https://blog.csdn.net/jfkidear/article/details/17795781>

**基本概念**

如果发现上一次提交的内容存在问题，就需要修改了。

基本概念

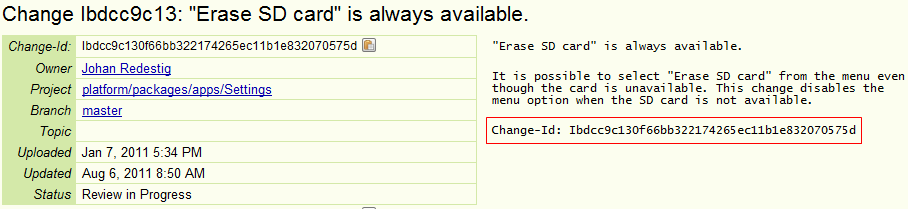
Change:

**Gerrit中的一个Change就是一个Review任务，它对应一个commit。** 每个commit，应该是为了一个目的的完整修改。如果某一次修改不完全，就需要**修正该commit**。

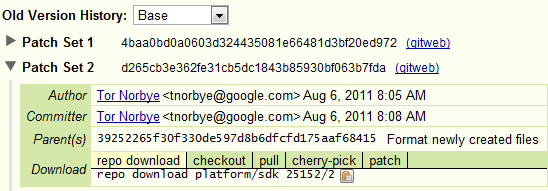
**每一次修正之前的commit，重新提交时，都应该保持Change-Id不变，这样就不会产生新的Change，而是在原有的Change下产生一个新的Patch Set。**

所有的Patch Set中，只有最新的一个是真正有用的，能够合并的。

**图1：Change和Change-Id**



**图2：Patch Set**



**修改前一次提交的方法**

**方法一：用  --amend 选项**

1. #修改需要修改的地方。
2. git add .
3. git commit –amend

注：这种方式可以比较方便的保持原有的Change-Id，推荐使用。

**方法二：先reset，再修改**

这是可以完全控制上一次提交内容的方法。但在与Gerrit配合使用时，需特别注意保持同一个commit的多次提交的Change-Id是不变的。

否则，就需要Abondon之前的Change，产生一些垃圾不说，操作得不对，会使得简单的事情复杂化，甚至无法合并。

1. git reset HEAD^
2. #重新修改
3. git add .
4. git commit -m “MSG”

特别注意：为了保持提交到Gerrit的Change不变，**需要复制对应的Change-Id到commit msg的最后，可以到Gerrit上对应的Change去复制**，参见图1。

也可以使用gerrit网页的download，下载已经提交的patch，然后本地将这个patch合并，再修改，修改完成之后提交，按照上述方法**复制对应的Change-Id到commit msg的最后**，最好执行git push。

**方法三：只是修改作者**

如果email不对，会无法提交到Gerrit，所以这个命令也可能用到。

1. git commit –amend –author=<user-email>

注：如果该email地址从未有过成功的提交，这个修改会不成功。在别的分支做一次成功提交之后，就可以修改了。

# 1. git commit 中的change-id

<https://blog.csdn.net/FIRE_TRAY/article/details/49227341>

**为什么要使用change-id**

保证已经提交审核的修订通过审核入库后，被别的分支 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 设计了一套方法，即要求每个提交包含唯一的 Change-Id，这个 Change-Id 因为出现在日志中，当执行 cherry-pick 时也会保持，Gerrit 一旦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 task-id，而直接将提交入库。

**解决“ERROR：missing Change-Id in commit message”**

在执行git push origin Head:refs/for/xxxx时有时会报出这样的错误，也就是在commit Message仅仅包含如“feature：xxxx”等title这样的信息而缺少Change-Id。

**手动解决**

**执行git commit --amend，在title下面空出一行后，将错误提示中的change-id复制到Message中，再push**。**注意，一定要空出一行，否则Git会把其作为title的一部分处理。**

**利用commit-msg hook自动生成change-id**

将如下的的hook脚本复制到Git项目中.git/hooks下，并命名为“commit-msg”。一般情况 .git/hooks 下会包含一个叫“commit-msg.sample”的文件，可以把它删除。添加完成后，执行chmod u+x .git/hooks/commit-msg保证hook可执行。

**commit-msg hook脚本**

#!/bin/sh

# From Gerrit Code Review 2.6

#

# Part of Gerrit Code Review (http://code.google.com/p/gerrit/)

#

# Copyright (C) 2009 The Android Open Source Project

#

# Licensed under the Apache License, Version 2.0 (the "License");

# you may not use this file except in compliance with the License.

# You may obtain a copy of the License at

#

# http://www.apache.org/licenses/LICENSE-2.0

#

# Unless required by applicable law or agreed to in writing, software

# distributed under the License is distributed on an "AS IS" BASIS,

# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.

# See the License for the specific language governing permissions and

# limitations under the License.

#

unset GREP\_OPTIONS

CHANGE\_ID\_AFTER="Bug|Issue"

MSG="$1"

# Check for, and add if missing, a unique Change-Id

#

add\_ChangeId() {

clean\_message=`sed -e '

/^diff --git a\/.\*/{

s///

q

}

/^Signed-off-by:/d

/^#/d

' "$MSG" | git stripspace`

if test -z "$clean\_message"

then

return

fi

# Does Change-Id: already exist? if so, exit (no change).

if grep -i '^Change-Id:' "$MSG" >/dev/null

then

return

fi

id=`\_gen\_ChangeId`

T="$MSG.tmp.$$"

AWK=awk

if [ -x /usr/xpg4/bin/awk ]; then

# Solaris AWK is just too broken

AWK=/usr/xpg4/bin/awk

fi

# How this works:

# - parse the commit message as (textLine+ blankLine\*)\*

# - assume textLine+ to be a footer until proven otherwise

# - exception: the first block is not footer (as it is the title)

# - read textLine+ into a variable

# - then count blankLines

# - once the next textLine appears, print textLine+ blankLine\* as these

# aren't footer

# - in END, the last textLine+ block is available for footer parsing

$AWK '

BEGIN {

# while we start with the assumption that textLine+

# is a footer, the first block is not.

isFooter = 0

footerComment = 0

blankLines = 0

}

# Skip lines starting with "#" without any spaces before it.

/^#/ { next }

# Skip the line starting with the diff command and everything after it,

# up to the end of the file, assuming it is only patch data.

# If more than one line before the diff was empty, strip all but one.

/^diff --git a/ {

blankLines = 0

while (getline) { }

next

}

# Count blank lines outside footer comments

/^$/ && (footerComment == 0) {

blankLines++

next

}

# Catch footer comment

/^\[[a-zA-Z0-9-]+:/ && (isFooter == 1) {

footerComment = 1

}

/]$/ && (footerComment == 1) {

footerComment = 2

}

# We have a non-blank line after blank lines. Handle this.

(blankLines > 0) {

print lines

for (i = 0; i < blankLines; i++) {

print ""

}

lines = ""

blankLines = 0

isFooter = 1

footerComment = 0

}

# Detect that the current block is not the footer

(footerComment == 0) && (!/^\[?[a-zA-Z0-9-]+:/ || /^[a-zA-Z0-9-]+:\/\//) {

isFooter = 0

}

{

# We need this information about the current last comment line

if (footerComment == 2) {

footerComment = 0

}

if (lines != "") {

lines = lines "\n";

}

lines = lines $0

}

# Footer handling:

# If the last block is considered a footer, splice in the Change-Id at the

# right place.

# Look for the right place to inject Change-Id by considering

# CHANGE\_ID\_AFTER. Keys listed in it (case insensitive) come first,

# then Change-Id, then everything else (eg. Signed-off-by:).

#

# Otherwise just print the last block, a new line and the Change-Id as a

# block of its own.

END {

unprinted = 1

if (isFooter == 0) {

print lines "\n"

lines = ""

}

changeIdAfter = "^(" tolower("'"$CHANGE\_ID\_AFTER"'") "):"

numlines = split(lines, footer, "\n")

for (line = 1; line <= numlines; line++) {

if (unprinted && match(tolower(footer[line]), changeIdAfter) != 1) {

unprinted = 0

print "Change-Id: I'"$id"'"

}

print footer[line]

}

if (unprinted) {

print "Change-Id: I'"$id"'"

}

}' "$MSG" > "$T" && mv "$T" "$MSG" || rm -f "$T"

}

\_gen\_ChangeIdInput() {

echo "tree `git write-tree`"

if parent=`git rev-parse "HEAD^0" 2>/dev/null`

then

echo "parent $parent"

fi

echo "author `git var GIT\_AUTHOR\_IDENT`"

echo "committer `git var GIT\_COMMITTER\_IDENT`"

echo

printf '%s' "$clean\_message"

}

\_gen\_ChangeId() {

\_gen\_ChangeIdInput |

git hash-object -t commit --stdin

}

# 1.如何在gerrit中修改已提交的change

<https://blog.csdn.net/u010698836/article/details/52189630>

在gerrit审核中，经常会遇到开发人员提交的代码审核不通过的情况。

那么开发人员这时有两个选择：1）按照要求修改代码，然后重新提交一次代码。2）修改原来的Change。

第一种方法会产生多次commit，而这些commit实际上是没有太多意义的，所以不推荐使用。

所以推荐第二种方法。下面讲解如何做：

## 安装 commit-msg hook

**安装gerrit的commit-msg hook的目的是为了能够在每次提交的时候在你的本地产生一个Change-Id，这个Change-Id是将gerrit的Change和你的commit联系起来的纽带。**

1. *# 到项目的根目录下执行*
2. curl -Lo .git/hooks/commit-msg http://your-gerrit-server/gerrit/tools/hooks/commit-msg
3. chmod u+x .git/hooks/commit-msg

**这样的话，你每次提交的时候，这个hook都会在commit message的后面添加一行Change-Id：**

1. $ git log -1
2. commit 29a6bb1a059aef021ac39d342499191278518d1d
3. Author: A. U. Thor <author@example.com>
4. Date: Thu Aug 20 12:46:50 2009 -0700
5. Improve foo widget by attaching a bar.
6. We want a bar, because it improves the foo by providing more
7. wizbangery to the dowhatimeanery.
8. Bug: *#42*
9. Change-Id: Ic8aaa0728a43936cd4c6e1ed590e01ba8f0fbf5b

## 修改已经提交至gerrit的commit

1. $ git checkout 有问题的commit
2. $ <修改>
3. $ git commit --amend
4. $ git push origin HEAD:refs/for/master
5. Counting objects: 5, done.
6. Delta compression using up to 8 threads.
7. Compressing objects: 100% (2/2), done.
8. Writing objects: 100% (3/3), 546 bytes, done.
9. Total 3 (delta 0), reused 0 (delta 0)
10. remote: Processing changes: updated: 1, done
11. remote:
12. remote: Updated Changes:
13. remote: http://gerrithost:8080/68
14. remote:
15. To ssh://gerrithost:29418/RecipeBook.git
16. \* [new branch] HEAD -> refs/for/master

注意到上面的Updated Changes字样了吗？这说明我们是更新了gerrit上的Change，而不是一般情况下的New Changes。

然后你到gerrit上查看自己的Change，是不是有变化了？

## windows下怎么弄？

commit-msg hook在windows下可能无法起作用，但我们依然有办法解决这问题。

我们还是按照前面讲的步骤来：

1. $ git checkout 有问题的commit
2. $ <修改>
3. $ git commit --amend

注意，这个时候你需要手工添加Change-Id了，到gerrit上找到自己的Change的Change-Id：

上图的最后一样就是Change-Id，然后手工添加到commit注释的最后一行上：

1. Change-Id: I347f61e90f259c78fcaaa8367b804941005a9b2b

然后

1. $ git push origin HEAD:refs/for/master
2. Counting objects: 5, done.
3. Delta compression using up to 8 threads.
4. Compressing objects: 100% (2/2), done.
5. Writing objects: 100% (3/3), 546 bytes, done.
6. Total 3 (delta 0), reused 0 (delta 0)
7. remote: Processing changes: updated: 1, done
8. remote:
9. remote: Updated Changes:
10. remote: http://gerrithost:8080/68
11. remote:
12. To ssh://gerrithost:29418/RecipeBook.git

\* [new branch] HEAD -> refs/for/master

# 1. missing change-Id的解决办法

套路部分为解决问题的步骤,心法部分为所用到的知识点的梳理.

## 场景:

你用 [**git**](http://lib.csdn.net/base/git) push 向 gerrit 提交了待审核代码,一切都很顺利,你脑袋里冒出了"代码头上加了'佛祖保佑'果然有效"的想法.

此时 git 打印出如下提示,你的内心OS同步打印 "心情 -5" :

|  |
| --- |
| remote: Resolving deltas: 100% (14/14)  remote: Processing changes: refs: 1,done  remote: ERROR: missing Change-Idincommit message footer  remote:  remote: Hint: To automatically insert Change-Id,installthe hook:  remote:   gitdir=$(git rev-parse --git-dir);scp-p -P 29418 liux@gerrit.xxxxx.com:hooks/commit-msg${gitdir}/hooks/  remote: And then amend the commit:  remote:   git commit --amend  remote:  To ssh://liux@121.xx.xx.xx:29418/kaiba\_admin\_yunying.git   ! [remote rejected] HEAD -> refs/for/master(missing Change-Idincommit message footer)  error: failed to push some refs to'ssh://liux@121.xx.xx.xx:29418/sample\_project.git' |

## 套路:

### 大前提: commit-msg 文件必须已经在该项目中存在.

使用ls命令检查该文件是否存在:

|  |
| --- |
| $ cd project\_dir  $ ls .git/hooks/commit-msg |

如果该文件不存在,则按照 git push 时产生的提示信息,获取该文件:

|  |
| --- |
| $ gitdir=$(git rev-parse --git-dir);scp-p -P 29418 liux@gerrit.xxxxx.com:hooks/commit-msg${gitdir}/hooks/ |

上面的命令可以直接从 git push 产生的错误信息中复制出来.

如果要手敲该命令,别忘了把用户名换成自己的。

### 方法一: 使用 amend 选项生成 Change-Id:

如果缺失 Change-Id 的是最后一个 (head) commit, 使用以下命令即可解决问题:

|  |
| --- |
| $ git commit --amend |

该命令会打开默认的 commit message 编辑器,一般是 vi.

这时什么都不用修改,直接保存退出即可 (:wq).

再次查看 git log,就会发现缺失的 Change-Id 已经被补上了. 再次 git push 即可.

### 方法二: 如果缺失 Change-Id 的不是最后一个 commit, 可用 reset 方法:

比如,如果缺失 Change-Id 的 commit 是 git log 中的第二个 commit, 则可以用 git reset 命令将本地分支回退到该 commit.

(但其实用 git reset 找回 Change-Id 是普通青年才干的事情,文艺青年有更优雅的办法.见方法三)

首先执行 git log, 找出缺失了 Change-Id 的 commit,并复制其 commit-id:

|  |
| --- |
| $ git log  commit 8e1cad33bcd98e175cba710b1eacfd631a5dda41  Author: liux <liux@xxxx.cn>  Date:   Mon Dec 19 17:43:00 2016 +0800        testcommit"with amended commit message"        Change-Id: I9d2af0cc31423cf808cd235de0ad02abf451937d    commit 1a9096a34322885ac101175ddcac7dab4c52665d  Author: liux <liux@xxxx.cn>  Date:   Mon Dec 19 15:23:36 2016 +0800        testcommit-msg hook    ...... |

发现是第二个 commit 缺失 Change-Id. 将代码 reset 到这个 commit, 并执行 amend:

|  |
| --- |
| $ git reset 1a9096a34322885ac101175ddcac7dab4c52665d  $ git commit --amend |

注: 上面的 git reset 用法不会毁灭你写的代码,放心执行即可.

这时 git log 可以发现该 commit 已经补全了 change-Id.

下一步是把 git reset 撤消掉的代码重新 commit, 然后 push 即可:

|  |
| --- |
| $ git add ......  $ git commit -m "你的提交日志"  $ git push review HEAD:refs/for/master |

### 方法三: 使用交互式 rebase 找回任意提交位置的 Change-Id:

前面方法二中给出的例子是第二个提交缺失 Change-Id,这时用 git reset 还可以解决问题.

但如果你在一个方案上已经工作了一个月,生成了100个本地 commit,提交时才发现 git log 中第99个 commit 缺失 Change-Id. 如果这时还用 git reset 来找回 Change-Id ......

不要香菇,不要蓝瘦.文艺青年表示有办法优雅的解决问题: 交互式 rebase.

**第一步,找到缺失 Change-Id 的那个 commit:**

|  |
| --- |
| $ git log  commit 8aaaa749db4a5b105aa746659c5cd266ac82fffe  Author: liux <liux@xxxx.cn>  Date:   Mon Dec 19 17:43:24 2016 +0800        I am commit message 3        Change-Id: Ic89d5ce6ce4de70d1dcb315ce543c86a2b3ac003    commit 8e1cad33bcd98e175cba710b1eacfd631a5dda41  Author: liux <liux@xxxx.cn>  Date:   Mon Dec 19 17:43:00 2016 +0800        I am commit message 2        Change-Id: I9d2af0cc31423cf808cd235de0ad02abf451937d    commit 1a9096a34322885ac101175ddcac7dab4c52665d  Author: liux <liux@xxxx.cn>  Date:   Mon Dec 19 15:23:36 2016 +0800        I am commit message 1    commit d714bcde0c14ba4622d28952c4b2a80882b19927  Author: shangsb <shangsb@czfw.cn>  Date:   Wed Dec 14 09:20:52 2016 +0800        这是一个提交        Change-Id: I629b2bedff95491875f63634ad3da199612735b6  ...... |

发现是 "I am commit message 1" 这个提交没有 Change-Id.

**第二步,编辑交互式 rebase 的命令文件:**

执行 git rebase -i, 参数为 该提交的上一个提交的 commit-id (本例中为 "表单" 那个提交):

|  |
| --- |
| $ git rebase -i d714bcde0c14ba4622d28952c4b2a80882b19927 |

这个命令会打开默认的编辑器,一般为 vi. 内容如下:

|  |
| --- |
| pick 1a9096a I am commit message 1  pick 8e1cad3 I am commit message 2  pick 8aaaa74 I am commit message 3  # Rebase d714bcd..8aaaa74 onto d714bcd  #  # Commands:  #  p, pick = use commit  #  r, reword = use commit, but edit the commit message  #  e, edit = use commit, but stop for amending  #  s, squash = use commit, but meld into previous commit  #  f, fixup = like "squash", but discard this commit's log message  #  x, exec = run command (the rest of the line) using shell  #  # These lines can be re-ordered; they are executed from top to bottom.  #  # If you remove a line here THAT COMMIT WILL BE LOST.  #  # However, if you remove everything, the rebase will be aborted.  #  # Note that empty commits are commented out |

可以将这个文件理解为 git rebase 的内嵌脚本.其命令写法已经在下面的注释里给出了.

这里不赘述,仅给出最终要将该文件编辑成什么样子:

|  |
| --- |
| reword 1a9096a I am commit message 1  pick 8e1cad3 I am commit message 2  pick 8aaaa74 I am commit message 3  # Rebase d714bcd..8aaaa74 onto d714bcd  #  # Commands:  #  p, pick = use commit  #  r, reword = use commit, but edit the commit message  #  e, edit = use commit, but stop for amending  #  s, squash = use commit, but meld into previous commit  #  f, fixup = like "squash", but discard this commit's log message  #  x, exec = run command (the rest of the line) using shell  #  # These lines can be re-ordered; they are executed from top to bottom.  #  # If you remove a line here THAT COMMIT WILL BE LOST.  #  # However, if you remove everything, the rebase will be aborted.  #  # Note that empty commits are commented out |

即: 将缺失了 Change-Id 的 commit 前面的 "pick" 改为 "reword" 即可. 保存退出 (:wq)

注1: 上述文件中 commit 的顺序是和 git log 显示的顺序相反的: git log 为最新的在最前; 上述文件为 最新的在最后.

注2: 如果进入该模式后,却不确定该怎么改,这时不要担心,直接退出编辑则什么都不会发生 (:q!)

注3: 如果没有搞清楚运作机制,就要注意,除了按需把 pick 改为 reword 外,不要做其他改动.尤其注意不要删除任何行 (被删除的那行对应的提交将丢失).

注4: 你应该已经发现,有多个 commit 缺失 Change-Id 的情况也可以用该方法一次性处理.

**第三步,逐个编辑 commit-msg:**

上一步打开的文件保存退出后,git会逐个打开被你标注了 reword 的提交日志页面.

不需要修改任何东西,逐个保存退出即可 (一路 :wq).

**第四步,再次提交:**

用 git log 查看提交日志,会发现缺失的 Change-Id 都生成了. 愉快的提交代码吧!

|  |
| --- |
| $ git push review HEAD:refs/for/master |

## 心法:

### gerrit 的 Change-Id 机制:

首先要明确, Change-Id 是 gerrit (代码审核平台)的概念, 与 git (版本管理) 是没有关系的.

简单来说, Change-Id 是 gerrit 用以追踪具体提交的机制. 这里不贴网上已有的解释,举两个栗子大家体会下:

1. 你已经用 git push 将代码提交 gerrit 审核了,这时你发现代码中有疏漏,修改了一下,执行 git commit --amend, 再次推送还可以成功. 这就是因为 gerrit 检查到两次 push 的 commit 有同一个 change-id, 就认为是同一个提交,因此可以 amend.

2. git push 将代码提交到 gerrit 审核,到 gerrit 网站一看,大红字标着 Can Not Merge 字样. 我想常用 gerrit 的同学肯定都遇到过这问题. 之前我的做法是, git reset 后,更新代码,再重新提交. 现在的做法是,不用 git reset 了,直接 git commit --amend, 删掉 commit log 中的 change-id 那行,然后wq保存退出.这时 gerrit 的那个钩子脚本会再生成一个不同的 change-id ,这时再更新代码,重新提交即可成功. 这里只简要介绍该方法,具体步骤将在 代码冲突 场景中详解.

Change-Id 的生成机制请继续向下看.

### git 的 hook 机制:

钩子(hooks)是一些在$GIT-DIR/hooks目录的脚本, 在被特定的事件(certain points)触发后被调用。当git init命令被调用后, 一些非常有用的示例钩子脚本被拷到新仓库的hooks目录中; 但是在默认情况下它们是不生效的。 把这些钩子文件的".sample"文件名后缀去掉就可以使它们生效。

hook机制可以理解为回调.各个钩子其实就是一段 bash 脚本,各钩子脚本的名字都是固定的.可以查看git项目根目录下的 .git/hooks 这个文件夹,看看都有哪些可用的钩子.

|  |
| --- |
| $ cd project\_dir  $ ls .git/hooks/  applypatch-msg.sample  commit-msg.sample   pre-applypatch.sample  prepare-commit-msg.sample  pre-rebase.sample  commit-msg             post-update.sample  pre-commit.sample      pre-push.sample            update.sample |

如果有自己感兴趣的 git 事件要处理,修改相应的钩子脚本罗辑即可.然后把 .sample 后缀去掉,钩子就生效了.

在 gerrit 的 Change-Id 生成机制中,其实 gerrit 就是利用了 commit-msg 的钩子,在我们提交代码后,按一定规则去修改了我们的提交日志,在其末尾添加了这么一行:

 Change-Id: .......

这个钩子脚本是什么时候被加入我们的项目中的呢? 其实就是你在 git push 出错时 gerrit 网站给你的提示中的那句命令:

|  |
| --- |
| $ gitdir=$(git rev-parse --git-dir); scp -p -P 29418 liux@gerrit.kaiba315.com:hooks/commit-msg ${gitdir}/hooks/ |

执行该命令即可得到生成 Change-Id 的钩子脚本. 这条命令做了以下事情:

|  |
| --- |
| // git rev-parse --git-dir 这条命令将找到该项目的 git 目录,并将其赋值给 gitdir 这个变量.  // 一般就是项目根目录下的 .git/ 这个目录.  $ gitdir=$(git rev-parse --git-dir)    // 执行 scp 命令,从 gerrit 代码服务器将钩子脚本文件 commit-msg 下载到项目的钩子目录下 (一般是 .git/hooks/)  $ scp -p -P 29418 liux@gerrit.kaiba315.com:hooks/commit-msg ${gitdir}/hooks/ |

查看该脚本,会发现它是用 awk 命令处理了 .git/COMMIT\_EDITMSG 这个文件.

所以如果想手动生成 Change-Id ,只要执行下面命令,就可以生成一个可用的 Change-Id:

|  |
| --- |
| $ cd project\_dir  $ echo "some commit" > /tmp/test\_generate\_change\_id  $ .git/hooks/commit-msg /tmp/test\_generate\_change\_id  $ cat /tmp/test\_generate\_change\_id  some commit    Change-Id: Ic89d5ce6ce4de70d1dcb315ce543c86a2b3ac003 |

### 利用 git commit --amend 重新生成 Change-Id 的原理:

git commit --amend , 看名字就知道,是对某个 commit 做出修改的.这种修改既可以包含文件修改,也可以仅包含提交日志修改.

我们用 --amend 对 commit 做出修改后, commit-msg 的钩子会被重新触发, Change-Id 就会被生成出来.

用交互式 git rebase 来生成 Change-Id 也是同一个道理.

另:

通过总结历次缺失 Change-Id 的例子,发现基本我们自己通过 git commit 生成的提交都会很顺利的生成 Change-Id.

通过 git merge, git revert 等命令由 git 自己生成的 commit 则有较高概率会缺失 Change-Id.

嗯,我们发现了一个伟大的定律! 然并卵... 并不知道怎么解决这个问题.

因此提倡尽量用 git rebase 代替 git merge 来更新代码.

事实上, git rebase 更新代码 相较 git merge 更新代码,有诸多优势,只是略复杂些.强烈建议用 git rebase 方式更新代码.

### git rebase -i:

// TODO: 待续...

# 1.Gerrit code review

<https://www.cnblogs.com/chenpingzhao/p/9337117.html>

代码审核（Code Review）是软件研发质量保障机制中非常重要的一环，但在实际项目执行过程中，却因为种种原因被Delay甚至是忽略。在实践中，给大家推荐一款免费、开放源代码的代码审查软件Gerrit。

### 一、 Code Review是什么

Code Review最直观的解释即看代码。常规的做法为自己看，有时代码逻辑问题可能自己看不出来，需要找同事一起看，在大家知识体系相对平均的情况下可能需要花钱专门的公司帮助查看。

Code Review需要看哪些？对于刚入职场或者刚接触到Coding的新人来说，代码风格是比较重要的一块。除此之外，编码规范及代码结构写法，框架和工具的选型，具体项目的业务逻辑，安全隐患，性能问题等都可以通过review的方式发现。Code Review从前往后大致分为结对编程，提交代码后，测试之前，发版之前，发版之后等几个阶段，越往后，Code Review的效果越差，修复的成本也越来越高。

为什么一定要做入库前Code Review？

首先，代码审查的最大的功用是纯社会性的。如果你在编程，而且知道将会有同事检查你的代码，你编程态度就完全不一样了。你写出的代码将更加整洁，有更好的注释和程序结构。

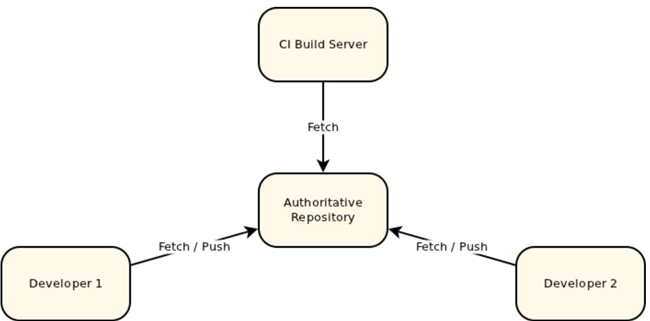
其次，偷懒是人的天性，从节约成本的角度考虑，大家一般会选择在测试之前无限制的Delay Code Review。入库前做Code Review便是成本和效果之间最佳平衡点，它能及时发现问题，进行修改后确保代码质量。

最后，代码审查能传播知识。在很多开发团队里，经常每个人负责一个核心模块，每个人都只关注自己的模块。除非是同事的模块影响了自己的程序，他们从不相互交流。这种情况的后果是，每个模块只有一个人熟悉里面的代码。如果这个人休假或辞职了，其他人则束手无策。通过代码审查，至少会有两个人熟悉这些程序——作者，以及审查者。审查者并不能像程序的作者一样对程序十分了解，但至少他会熟悉程序的设计和架构，这是极其重要的。

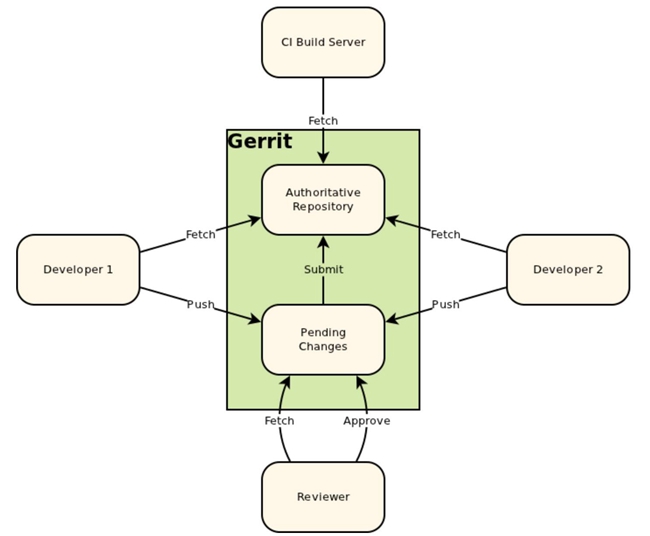
### 二、 Gerrit简介

Gerrit是Google为Android系统研发量身定制的一套免费开源的代码审核系统，它在传统的源码管理协作流程中强制性引入代码审核机制，通过人工代码审核和自动化代码验证过程，将不符合要求的代码屏蔽在代码库之外，确保核心代码多人校验、多人互备和自动化构建核验。

Gerrit之前的系统架构：



Gerrit之后的系统架构：

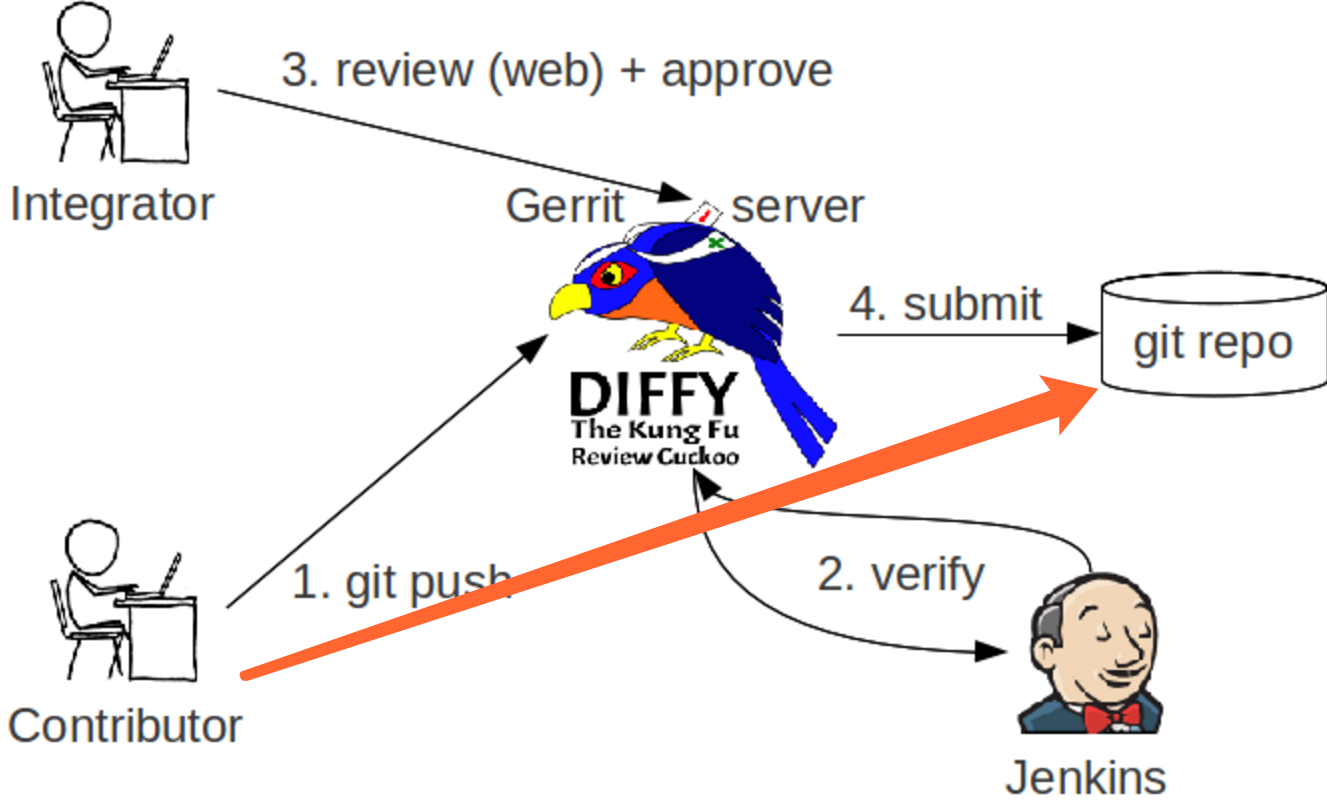


通过Gerrit机制将代码做分隔。

Gerrit适用性

几乎任何需要正式发布的项目都应当使用Gerrit来进行代码审查，如果Team中有新人，必须使用Gerrit确保代码质量。

#### 工作流程



使用过git的同学，都知道，当我们git add --> git commit --> git push 之后，你的代码会被直接提交到repo，也就是代码仓库中，就是图中橘红色箭头指示的那样。

那么gerrit就是上图中的那只鸟，普通成员的代码是被先push到gerrit服务器上，然后由代码审核人员，就是左上角的integrator在web页面进行代码的审核(review)，可以单人审核，也可以邀请其他成员一同审核，当代码审核通过(approve)之后，这次代码才会被提交(submit)到代码仓库(repo)中去。

无论有新的代码提交待审核，代码审核通过或被拒绝，代码提交者(Contributor)和所有的相关代码审核人员(Integrator)都会收到邮件提醒。  
gerrit还有自动测试的功能，和主线有冲突或者测试不通过的代码，是会被直接拒绝掉的，这个功能似乎就是右下角那个老头(Jenkins)的任务。

整个流程就是这样。 在使用过程中，有两点需要特别注意下：

1. 当进行commit时，必须要生成一个Change-Id，否则，push到gerrit服务器时，会收到一个错误提醒。
2. 提交者不能直接把代码推到远程的master主线(或者其他远程分支)上去。这样就相当于越过了gerrit了。 gerrit必须依赖于一个refs/for/\*的分支。

假如我们远程只有一个master主线，那么只有当你的代码被提交到refs/for/master分支时，gerrit才会知道，我收到了一个需要审核的代码推送，需要通知审核员来审核代码了。  
当审核通过之后，gerrit会自动将这条分支合并到master主线上，然后邮件通知相关成员，master分支有更新，需要的成员再去pull就好了。而且这条refs/for/master分支，是透明的，也就是说普通成员其实是不需要知道这条线的，如果你正确配置了sourceTree，你也应该是看不到这条线的。

这两点很重要！！这两点很重要！！这两点很重要！！

### 三、Gerrit安装

#### 3.1. 环境准备

* ①. Linux，Gerrit需要Linux环境，ubuntu 或者  centos，这里使用Ubuntu；
* ②. JDK，使用1.7版本就行
* ③. MySQL，其实这个非必须，Gerrit自带的有H2数据库
* ④. nginx/apache，作为认证和反向代理服务器；
* ⑤. Maven, 在安装的过程中会下载一些jar文件；
* ⑥. Git，用来拉取代码

#### 3.2 java环境安装

下载：jdk-7u79-linux-x64.tar.gz <http://www.oracle.com/technetwork/java/javase/downloads/jdk7-downloads-1880260.html>

安装：sudo tar zxvf ./jdk-7u79-linux-x64.tar.gz -C /opt

配置：vim ~/.bashrc(针对当前用户) or vim /etc/profile(针对所有用户，推荐)

|  |  |
| --- | --- |
| 1  2  3  4 | export JAVA\_HOME=/opt/jdk1.7.0\_79  export JRE\_HOME=$JAVA\_HOME/jre  export CLASSPATH=$JAVA\_HOME/lib:$JRE\_HOME/lib:$CLASSPATH  export PATH=$JAVA\_HOME/bin:$JRE\_HOME/bin:$PATH |

验证：

|  |  |
| --- | --- |
| 1  2  3  4  5 | java -version    java version "1.7.0\_79"  Java(TM) SE Runtime Environment (build 1.7.0\_79-b15)  Java HotSpot(TM) 64-Bit Server VM (build 24.79-b02, mixed mode) |

#### 3.3 git环境

gerrit依赖，用来操作git repository

|  |  |
| --- | --- |
| 1 | sudo apt-get install git |

#### 3.4 下载gerrit

|  |  |
| --- | --- |
| 1 | wget  https://www.gerritcodereview.com/download/gerrit-2.12.4.war |

#### 3.5 apache2安装

|  |  |
| --- | --- |
| 1 | sudo apt-get install apache2 |

验证：

|  |  |
| --- | --- |
| 1 | service apache2 start |

#### 3.6 gerrit管理帐号

gerrit依赖，用来管理gerrit。

|  |  |
| --- | --- |
| 1  2 | sudo adduser gerrit  sudo passwd gerrit |

并将gerrit加入sudo权限

|  |  |
| --- | --- |
| 1 | sudo visudo |

gerrit  ALL=(ALL:ALL) ALL (为了测试方便，开最大权限）

#### 3.7 ****配置gerrit****

默认安装：

|  |  |
| --- | --- |
| 1 | java -jar gerrit-2.12.4.war init --batch -d ~/review\_site |

更新配置文件：sudo vim ~/review\_site/etc/gerrit.config

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34 | [gerrit]          basePath = git          canonicalWebUrl = http://47.120.74.47:8081/  [database]          type = h2          database = /home/gerrit/review\_site/db/ReviewDB  [index]          type = LUCENE  [auth]          type = http  [receive]          enableSignedPush = false  [sendemail]          smtpServer = smtp.163.com          smtpServerPort = 465          smtpEncryption = ssl          smtpUser = avcd@163.com          smtpPass = avcd123123123          sslVerify = false          from = code review<avcd@163.com>  [container]          user = gerrit          javaHome = /opt/jdk1.7.0\_79/jre  [sshd]          listenAddress = \*:29418  [httpd]          listenUrl = http://\*:8081/  [cache]          directory = cache    [http]          proxy = http://47.120.74.47:8090          proxyUsername = gerrit1 #proxy user & password          proxyPassword = 123456 |

#### 3.8 配置apache2反向代理

a、如果apache目录结构如下：

|  |  |
| --- | --- |
| 1 | apache2.conf conf-enabled magic mods-enabled ports.conf sites-enabled conf-available envvars mods-available sites-available |

开启SSL、Proxy、Rewrite等模块：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | cd /etc/apache2/mods-enabled  ln -s ../mods-available/proxy.load  ln -s ../mods-available/proxy.conf  ln -s ../mods-available/proxy\_http.load  ln -s ../mods-available/proxy\_balancer.conf  ln -s ../mods-available/proxy\_balancer.load  ln -s ../mods-available/rewrite.load  ln -s ../mods-available/ssl.conf  ln -s ../mods-available/ssl.load  ln -s ../mods-available/socache\_shmcb.load #  ln -s ../mods-available/slotmem\_shm.load # |

更新配置文件：sudo vim /etc/apache2/sites-enabled/gerrit-httpd.conf

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25 | ServerName 47.200.74.47  <VirtualHost \*:8090>      ProxyRequests Off      ProxyVia Off      ProxyPreserveHost On      AllowEncodedSlashes On      RewriteEngine On      RewriteRule ^/(.\*) http://47.200.74.47:8081/$1 [NE,P]        <Proxy \*>            Order deny,allow            Allow from all      </Proxy>        <Location /login/>          AuthType Basic          AuthName "Gerrit Code Review"          Require valid-user          AuthBasicProvider file          AuthUserFile /etc/apache2/passwords      </Location>        ProxyPass / http://127.0.0.1:8081/    </VirtualHost> |

b、如果apache目录结构如下：

|  |  |
| --- | --- |
| 1 | bin  build  cgi-bin  conf  error  htdocs  icons  include  lib  logs  man  manual  modules |

开启SSL、Proxy、Rewrite等模块：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | # Open LoadModule  LoadModule proxy\_module modules/mod\_proxy.so  LoadModule proxy\_http\_module modules/mod\_proxy\_http.so  LoadModule ssl\_module modules/mod\_ssl.so  LoadModule rewrite\_module modules/mod\_rewrite.so  # Gerrit config  Include conf/extra/gerrit-httpd.conf |

其中apache2/conf/extra/gerrit-httpd.conf内容同上，apache2/sites-enabled/gerrit-httpd.conf。

#### 3.9 配置gerrit账户密码

|  |  |
| --- | --- |
| 1  2  3 | touch /etc/apache2/passwords  htpasswd -b /etc/apache2/passwords admin 123456(管理员)  htpasswd -b /etc/apache2/passwords gerrit1 123456(普通用户) |

#### 3.10 启动gerrit&启动apache2

|  |  |
| --- | --- |
| 1  2 | sudo ~/review\_site/bin/gerrit.sh start  sudo /etc/init.d/apache2 start |

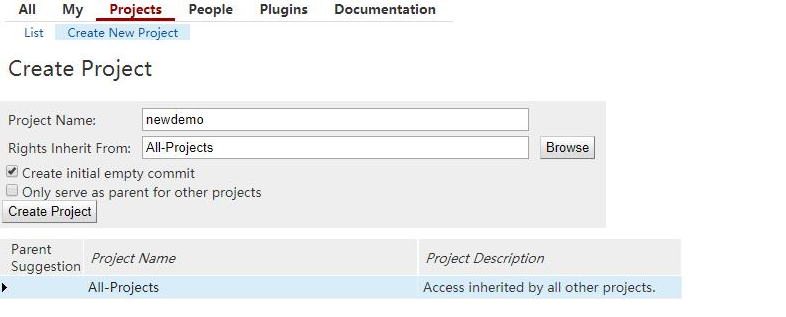
### 四、****如何使用gerrit****

前提：需要git使用端 / gerrit服务端配合使用。

#### 4.1 添加项目(gerrit 服务端)

|  |  |
| --- | --- |
| 1 | ssh -p 29418 gerrit1@47.200.74.47 gerrit create-project --empty-commit --name demo-project |

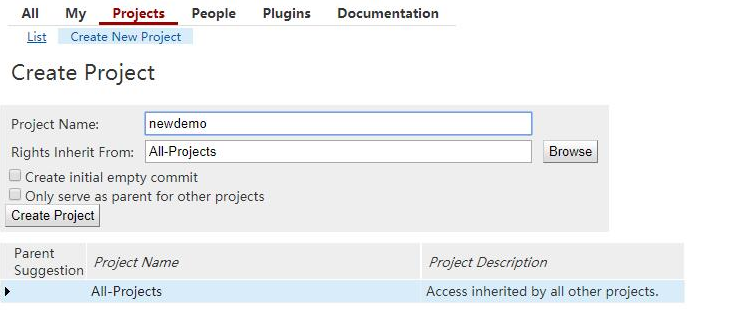
#建议采用管理界面添加



#### 4.2 使用gerrit添加已有项目：（适用于已有项目下移植到gerrit中）

ssh -p 29418 gerrit1@192.168.199.112 gerrit create-project --name exist-project #建议采用管理界面添加

或者使用gerrit管理界面



然后将已有项目与gerrit上建立的exist-project关联，即将已有代码库代码push到gerrit中进行管理。

|  |  |
| --- | --- |
| 1  2  3 | cd ~/exist-project    git push ssh://gerrit1@47.200.74.47:29418/exist-project \*:\* |

#### 4.3.生成sshkey(git使用端)

在开发账户中生成sshkey，用作与gerrit服务器连接。

|  |  |
| --- | --- |
| 1 | ssh-keygen -t rsa #生成sshkey |

#### 4.4.添加sshkey到gerrit服务器(gerrit 服务端)

此步骤与git流程类似，即将id\_rsa.pub内容上传到git repository，gerrit中帮我们管理git repository.

#### 4.5 拉取代码＆配置git hooks(git client端)

验证sshkey是否配置成功：ssh gerrit1@47.200.74.47 -p 29418

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | \*\*\*\*    Welcome to Gerrit Code Review    \*\*\*\*      Hi gerrit1, you have successfully connected over SSH.      Unfortunately, interactive shells are disabled.    To clone a hosted Git repository, use:      git clone ssh://gerrit1@47.200.74.47:29418/REPOSITORY\_NAME.git    Connection to 47.200.74.47 closed. |

拉取代码：

|  |  |
| --- | --- |
| 1 | git clone ssh://gerrit1@47.200.74.47:29418/REPOSITORY\_NAME.git |

更新githooks：

|  |  |
| --- | --- |
| 1 | gitdir=$(git rev-parse --git-dir); scp -p -P 29418 gerrit1@47.200.74.47:hooks/commit-msg ${gitdir}/hooks/ |

该过程用来在commit-msg中加入change-id，gerrit流程必备。

修改代码并提交，推送时与原有git流程不一致，采用 git push origin HEAD:refs/for/master 。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | [root@iZ62j8121e5Z abcd]# git push origin HEAD:refs/for/master  Counting objects: 3, done.  Compressing objects: 100% (2/2), done.  Writing objects: 100% (3/3), 710 bytes | 710.00 KiB/s, done.  Total 3 (delta 0), reused 0 (delta 0)  remote: Processing changes: new: 1, refs: 1, done  remote:  remote: New Changes:  remote:   http://47.200.74.47:8081/5 xxxx  remote:  To ssh://47.200.74.47:29418/abcd   \* [new branch]      HEAD -> refs/for/master |

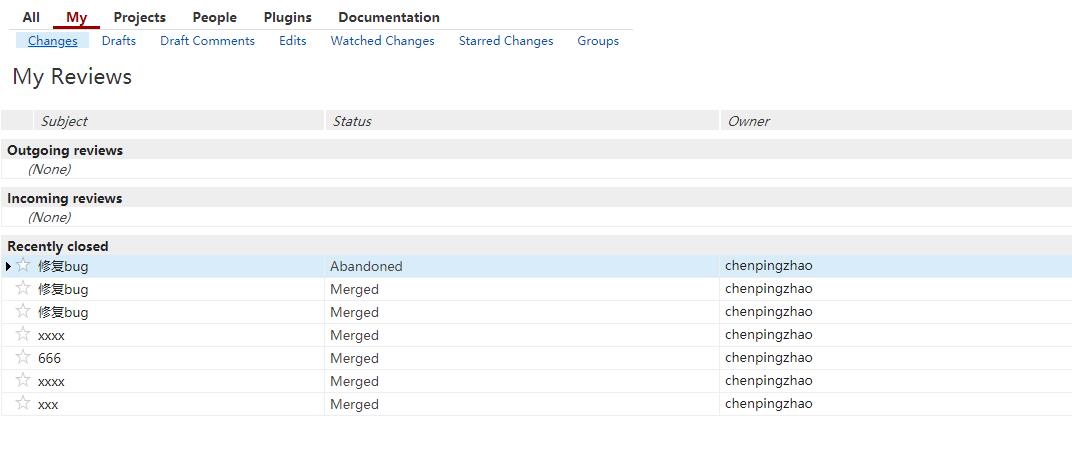
如果不加会提示一下错误：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | [root@iZ62j8121e5Z abcd]# git push origin HEAD:refs/for/master  Counting objects: 3, done.  Writing objects: 100% (3/3), 239 bytes | 239.00 KiB/s, done.  Total 3 (delta 0), reused 0 (delta 0)  remote: Processing changes: refs: 1, done  remote: ERROR: missing Change-Id in commit message footer  remote:  remote: Hint: To automatically insert Change-Id, install the hook:  remote:   gitdir=$(git rev-parse --git-dir); scp -p -P 29418 admin@47.200.74.47:hooks/commit-msg ${gitdir}/hooks/  remote: And then amend the commit:  remote:   git commit --amend  remote:  To ssh://47.200.74.47:29418/abcd   ! [remote rejected] HEAD -> refs/for/master (missing Change-Id in commit message footer)  error: failed to push some refs to 'ssh://admin@47.200.74.47:29418/abcd' |

### ****五.使用gerrit website完成code review****

当完成push后，可在gerrit管理界面看到当前提交code review的change。

查看某次提交的详细信息（审核者+2可通过本次提交，提交者可通过Abandon本次提交）：



如果审核者+2通过后，可提交该次commit.

### ****六 注意事项****

* 需要为每个使用者分配gerrit账号，不要都使用admin账号，因为admin账号可直接push master
* pull代码后需要配置githooks文件，以便在commit时自动生成change-id，否则无法push
* push代码时需要使用git push origin HEAD:refs/for/master(branch),gerrit默认关闭非admin账号的push direct权限
* push代码时需要commit email与gerrit account email一致，否则无法push成功，可选择关闭email notify，并开启forge user权限，或者通过修改gerrit数据库account email信息
* gerrit数据库与gitlab同步，需要安装replication插件，并开启该功能

参考链接：

Java SDK Download: <http://www.oracle.com/technetwork/java/javase/downloads/jdk7-downloads-1880260.html>

Gerrit Code Review - Releases Download: <https://gerrit-releases.storage.googleapis.com/index.html>

Gerrit Code Review -  Quick get started guide: <https://git.eclipse.org/r/Documentation/install-quick.html>

Gerrit代码审核流程 <http://www.worldhello.net/gotgit/images/gerrit-workflow.png>

Gerrit代码审核原理 <http://www.worldhello.net/gotgit/05-git-server/055-gerrit.html>

Gerrit代码审核权限管理 <https://gerrit-review.googlesource.com/Documentation/access-control.html#category_forge_committer>

Gerrit修改数据库email信息 <http://www.cnblogs.com/kevingrace/p/5624122.html>

Gerrit安装replication插件 <https://gerrit-review.googlesource.com/Documentation/cmd-plugin-install.html>