

git

点滴学习，随时记录

[git x分支强制覆盖master分支方法](#)

[git删除未跟踪文件](#)

[git分支管理策略](#)

[将其他分支文件或提交合并到当前分支](#)

[搭建自己git服务](#)

[git 连接远程仓库方法](#)

[git修改远程仓库地址](#)

[一台电脑上不同的Git仓库账号使用不同的公私钥设置](#)

[Gerrit代码Review入门实战](#)

git x分支强制覆盖master分支方法

1、删除本地master分支

2、将本地x分支名称改为master分支

3、强制推送本地master分支到远程

```
1 git push origin master --force
```

git删除未跟踪文件

git删除未跟踪文件

1、删除 untracked files

```
1 git clean -f
```

2、连 untracked 的目录也一起删掉

```
1 git clean -fd
```

3、连 .gitignore 中的 untrack 文件/目录也一起删掉 (慎用)

```
1 git clean -xfd
```

4、在用上述 git clean 前，墙裂建议加上 -n 参数来先看看会删掉哪些文件，防止重要文件被误删

```
1 git clean -nxfd  
2 git clean -nf  
3 git clean -nfd
```

git分支管理策略

个人在项目中使用git分支管理策略介绍

主分支Master

首先，代码库应该有一个、且仅有一个主分支Master。项目的正式版本，都在这个主分支上发布。它是自动建立的，版本库初始化以后，默认就是在Master分支进行开发。

功能开发分支 feature/*

临时性分支

功能分支，它是为了开发某种特定功能，从master分支上面分出来的。开发完成后，并入release；

功能分支的名字，可以采用feature/w_*的形式命名。即 feature/开发者标识_开发的功能

- 1 基于master分支创建一个功能分支：
- 2 开发完成后，将功能分支合并到release分支：
- 3 删除feature分支：

版本预发布分支 release/*

永久分支，命名 release/版本号

预发布分支，它是指发布正式版本之前（即合并到Master分支之前），我们可能需要有一个预发布的版本进行测试。

预发布分支最初是从master分支上面分出来的，将开发完成的feature分支合并到release分支进行测试，测试完成后将release分支合并到master分支进行发布

- 1 创建一个预发布分支：
- 2 将各feature分支合并到release分支
- 3 对release分支进行测试
- 4 确认没有问题后，合并到master分支：

bug修复分支 fixbug/*

临时性分支，命名 fixbug/开发者标识_修改的功能，

软件正式发布以后，难免会出现bug。这时就需要创建一个分支，进行bug修补。

修补bug分支是从Master分支上面分出来的。修补结束以后，再合并进Master和release分支。它的命名，可以采用fixbug/*的形式。

- 1 基于master创建一个修补bug分支：
- 2
- 3 修补结束后，合并到master分支：
- 4
- 5 再合并到release分支：

6

7 最后，删除“修补bug分支”：

将其他分支文件或提交合并到当前分支

将其他分支文件合并到当前分支

如将dev分支x.php合并到master分支为例

```
1 1、将分支切换到master分支  
2 git checkout master  
3  
4 2、合并文件，将dev分支上 x.php文件追加补丁到dev分支上 x.php文件。你可以接受或者拒绝补丁内容。  
5 git checkout --patch dev x.php
```

将其他分支提交合并到当前分支

如将dev分支某次提交合并到master分支为例

```
1 1、将分支切换到master分支  
2 git checkout master  
3  
4 3、查看要合并的提交的commitID  
5 git log  
6  
7 2、合并提交,如要合并的commitID: 7fcb3defff  
8 git cherry-pick 7fcb3defff  
9  
10 4、推送到master远程  
11 git push
```

搭建自己git服务

Gogs

一款极易搭建的自助 Git 服务 比gitlab更轻量级

站点: <https://gogs.io/>

git 连接远程仓库方法

方案一：本地创建项目根目录，然后与远程Git关联，之后的操作一样：

```
1 #创建新文件夹
2 mkdir xxx
3 #进入
4 cd xxx
5 #初始化Git仓库
6 git init
7 #提交改变到缓存
8 git commit -m 'description'
9 #本地git仓库关联GitHub仓库
10 git remote add origin git@github.com:han1202012/TabHost_Test.git
11 #提交到GitHub中
12 git push -u origin master
```

方案二：方案二就是不用关联Git仓库，直接从Git中克隆源码到本地，项目根目录也不用创建；

```
1 #从GitHub上克隆项目到本地
2 git clone git@github.com:han1202012/NDKHelloWorld.git #注意克隆的时候直接
   在仓库根目录即可，不用再创建项目根目录 ；
3 #添加文件
4 git add ./* # 将目录中所有文件添加;
5 #提交缓存
6 git commit -m '提交';
7 #提交到远程GitHub仓库
8 git push -u origin master
```

git修改远程仓库地址

方法有三种：

修改命令

```
1 git remote set-url origin [要改成的git url]
```

先删后加

```
1 git remote rm origin  
2 git remote add origin [要改成的git url]
```

直接修改config文件

```
1 vim .git/config  
2  
3 # 修改config文件中git远程仓库地址  
4  
5 [remote "origin"]  
6     url = [要改成的git url]
```

一台电脑上不同的Git仓库账号使用不同的公私钥设置

使用场景：

如我有多个git仓库账号，如2个bitbucket账号，一个github账号。我想让不同的git仓库账号使用不同的公私钥对，同时若多个bitbucket或多个github账号配置同一个公钥是不允许的，一个公钥只能配置在一个github或bitbucket账号中。要想解决如上问题，解决方法如下：

解决思路

生成多对私钥/公钥，注意密钥文件命名避免重复
git操作时，可以区分两个账户，推送到相应的仓库
设置不同的Host对应同一个HostName，但密钥不同
取消Git全局用户名/邮箱设置，为每个仓库单独设置用户名/邮箱

操作

1、创建多对公私钥

注意多对公私钥创建时，`-f` 和 `-C` 参数要不同。

创建方法如下：

```
1 ssh-keygen -t rsa -f ~/.ssh/id_rsa -C "xxx.yyyy@gmail.com"
2 解释几个参数：
3
4 -t 指定密钥类型，默认是 rsa，可以省略
5 -f 指定密钥文件存储文件名（注：~/.ssh/是密钥目录；id_rsa是密钥名，密钥名可以随意）
6 -C 设置注释文字，比如邮箱（1、这里的C是大写的 2、一定要关联你自己的GitHub的注册邮箱）
7 接着它会提示你输入两次密码（该密码是你push文件的时候要输入的密码，而不是GitHub管理者的密码），
8 你也可以不输入密码（推荐），直接按回车。那么push的时候就不需要输入密码，可以直接提交到GitHub上：
```

2、更改本地的SSH配置

用于区分账号，设置不同秘钥

在`~/.ssh`文件夹下新建config文件并编辑，配置Host（自定义账号域名）和HostName（实际git账号域名）、秘钥对应关系。

如果我们有一个bitbucket账号，一个github账号，设置方法如下

```
1 vim ~/.ssh/config
2
3 #第一个ssh密钥
4 Host github.com
5 HostName github.com
```

```
6 PreferredAuthentications publickey
7 User git
8 IdentityFile ~/.ssh/id_rsa_github
9 IdentitiesOnly yes
10
11 #第二个ssh密钥
12 Host bitbucket.org
13 HostName bitbucket.org
14 PreferredAuthentications publickey
15 User git
16 IdentityFile ~/.ssh/id_rsa_bitbucket
17 IdentitiesOnly yes
```

如果我们有两个bitbucket账号，设置方法如下

```
1 vim ~/.ssh/config
2
3 #第一个ssh密钥
4 Host git1.bitbucket.org
5 HostName bitbucket.org
6 PreferredAuthentications publickey
7 User git
8 IdentityFile ~/.ssh/id_rsa_bitbucket_1
9 IdentitiesOnly yes
10
11 #第二个ssh密钥
12 Host git2.bitbucket.org
13 HostName bitbucket.org
14 PreferredAuthentications publickey
15 User git
16 IdentityFile ~/.ssh/id_rsa_bitbucket_2
17 IdentitiesOnly yes
```

上面git1、git2 为自定义Host前缀。

以上配置文件参数说明：

```
1 # Host: 对识别的模式，进行配置对应的的主机名和ssh文件
2 # HostName: 登录主机的主机名
3 # PreferredAuthentications: 设置登录方式，publickey公钥，改成password则要输入密码
4 # IdentityFile: 私钥全路径名
5 # User: 登录名
6 # IdentitiesOnly 这个配置yes，表示只使用这里的key，防止使用默认的
```

3、将两个密钥对中的公钥分别加到两个Git账号配置下

4、克隆新的项目

以两个bitbucket账号为例：

一般情况下，我们是通过如下的方式克隆一个项目：

```
git clone git@bitbucket.org:your-account/your-prj.git
```

在我们有两个密钥的情况下（使用非默认密钥时），我们需要对这个语句中的域名部分做一下修改：

```
git clone git@xxx.bitbucket.org/your-prj.git
```

注意：这里的xxx是你更改本地的SSH配置第二步中的Host，比如的：

```
git clone git@git2.bitbucket.org:your-account/your-prj.git
```

这样就可以clone成功了。

若是一个github账号，一个bitbucket账号，那么git仓库地址不需要改变。

5、取消全局 用户名/邮箱设置，每个仓库单独设置

取消全局 用户名/邮箱 配置

```
1 git config --global --unset user.name  
2 git config --global --unset user.email
```

为每个仓库单独设置 用户名/邮箱

```
1 cd YourRepoPath  
2 git config user.name "You Name"  
3 git config user.email name@example.com
```

其实很简单，就是去掉--global参数就行了

参考文章：

<https://www.artjay.me/2019/more-ssh/>

Gerrit代码Review入门实战

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。

1、Why Code Review

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。如果这个人体假或辞职了，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。

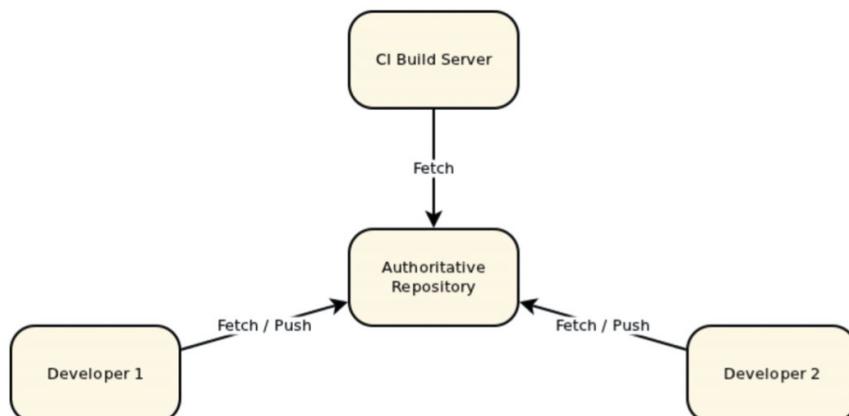
2、Gerrit（盖瑞特）简介

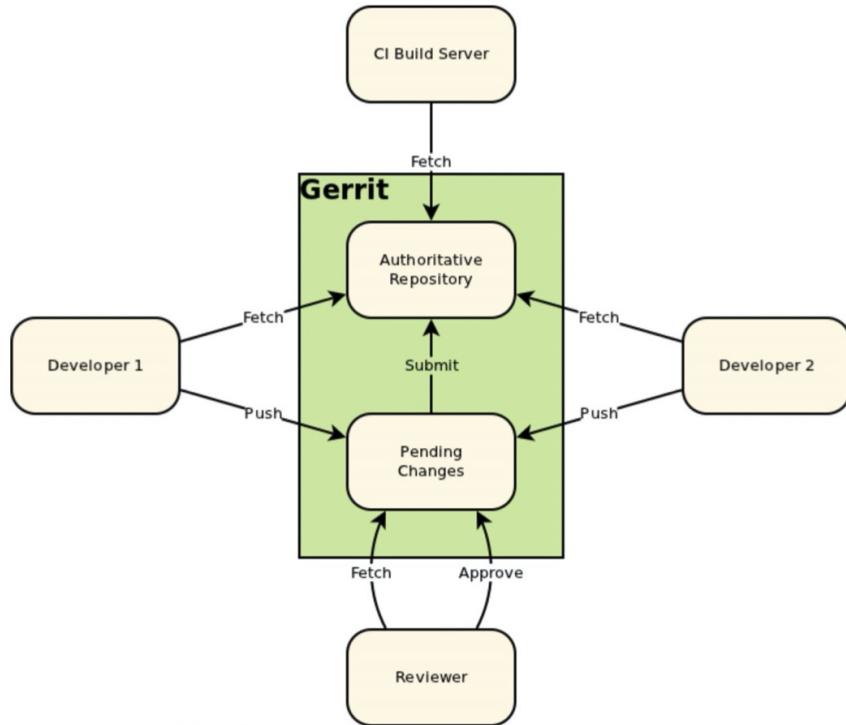
Gerrit，是一种开放[源代码](#)的基于web平台的代码评审工具，基于 Git 版本控制系统。它在传统的源码管理协作流程中强制性引入代码审核机制，通过人工代码审核和自动化代码验证过程，将不符合要求的代码屏蔽在代码库之外，确保核心代码多人校验、多人互备和自动化构建核验。

- | 作者是 Google 公司的 Shawn Pearce，最初是为了管理 Android 项目而产生的。
- | 它最初是由 Python 编写，在第二版后改用 Java 与 SQL。
- | 名字来源于一名荷兰设计师及建筑师 Gerrit Thomas Rietveld(赫里特·托马斯·里特费尔德)的名字。

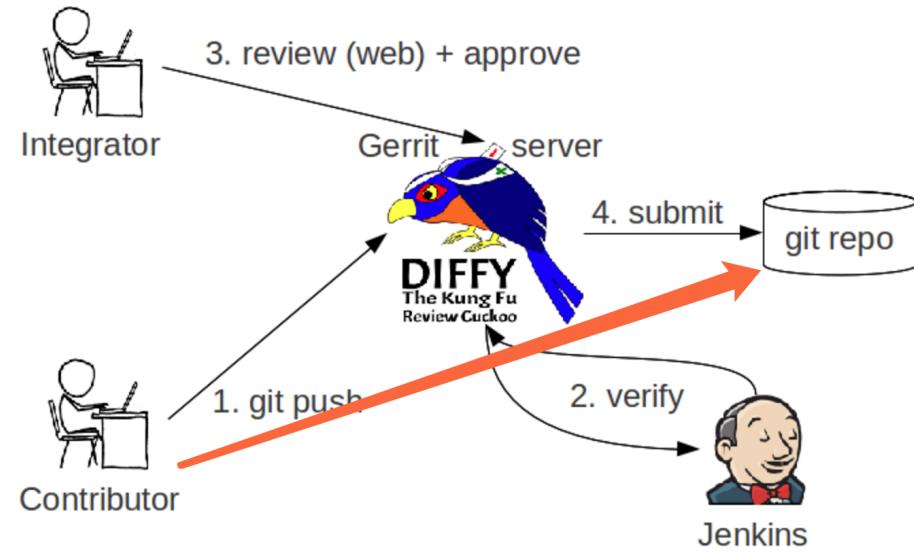
Gerrit工作流

git





或：



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：

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。
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。

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效果

The screenshot shows two main sections of the Gerrit interface.

Top Section: A list of open pull requests (status:open). The columns include Subject, Status, Owner, Project, Branch, Updated, Size, CR, and V. The list includes various users like 赵伟, liub, maozl, etc., across branches like develop, master, and dev. Most pull requests have a green checkmark in the CR column, except for a few which are red or yellow.

Bottom Section: A code review session for the file `CrashStatController.java`. The code editor shows a diff between two versions of the file. The changes are highlighted with red and green highlights. The code itself is Java, dealing with Elasticsearch aggregations and histogram builders.

```

67     }
68     FiltersAggregationBuilder statAgg = filters("stat");
69     for (String v : versions) {
70       if (StringUtil.equals(v, "all_")) {
71         statAgg.filter(v, termQuery("client_id", appid));
72       } else {
73         statAgg.filter(v, termQuery("version", v));
74       }
75     }
76   statAgg.subAggregation(dateHistogram("every_day").field("report_time"),
77     requestBuilder.addAggregation(statAgg);
78   SearchResponse searchResponse = requestBuilder.execute().actionGet();
79
80   Aggregations aggregations = searchResponse.getAggregations();
81   Map<Long, Map<String, Stat>> resultStat = new HashMap<>();
82
83   Filters stat = aggregations.get("stat");
84   for (Filters.Bucket bucket : stat.getBuckets()) {
85     Histogram date_histogram = bucket.getAggregations().get("every_day");
86     String version = bucket.getKeyAsString();
87     for (Histogram.Bucket bucket1 : date_histogram.getBuckets()) {
88       Stat stat1 = getOrCreateStat(resultStat, key.getMillis(), versi
89       stat1.setDocCount(bucket1.getDocCount());
90       stat1.setTotal(bucket1.getDocCount());
91       Cardinality contUre = bucket1.getAggregations().get("count_us
92       stat1.setEffect(contUre.getValue())
93       .setVersion(version)
94     }
95   }
96   Histogram date_histogram = bucket.getAggregations().get("every_day");
97   String version = bucket.getKeyAsString();
98   for (Histogram.Bucket bucket1 : date_histogram.getBuckets()) {
99     Stat stat1 = getOrCreateStat(resultStat, key.getMillis(), versi
100    stat1.setTotal(bucket1.getDocCount());
101    Cardinality contUre = bucket1.getAggregations().get("count_us
102    stat1.setEffect(contUre.getValue())
103    .setVersion(version)
104  }

```

整体上来说，个推使用的标准配置为Gerrit+Jenkins+Sonar，整个系统搭建完成后得到的效果为：100% Code Style问题避免入库，80% 设计问题避免入库，40% 逻辑错误避免入库，20% 安全隐患避免入库，100% 人员互备。

3、Gerrit入门实战

Gerrit部署和运行

JDK环境配置

java -jar gerrit-2.12.war init -d review_site

```

Generating SSH host key ... rsa(simple)... done

*** HTTP Daemon
*** 

Behind reverse proxy      [y/N]?
Use SSL (https://)          [y/N]?
Listen on address          [*]:
Listen on port              [8080]: 8381
Canonical URL              [http://getui:8381/]: http://121.199.40.127:8381/

*** Plugins
*** 

Installing plugins.
Install plugin singleusergroup version v2.12 [y/N]?
Install plugin commit-message-length-validator version v2.12 [y/N]?
Install plugin reviewnotes version v2.12 [y/N]?
Install plugin replication version v2.12 [y/N]?
Install plugin download-commands version v2.12 [y/N]?
Initializing plugins.
No plugins found with init steps.

Initialized /root/gerrit/review_site
Executing /root/gerrit/review_site/bin/gerrit.sh start
Starting Gerrit Code Review: OK
Waiting for server on 121.199.40.127:8381 ... OK
Opening http://121.199.40.127:8381/#/admin/projects/ ...FAILED
Open Gerrit with a JavaScript capable browser:
  http://121.199.40.127:8381/#/admin/projects/

```

Gerrit用户角色和权限

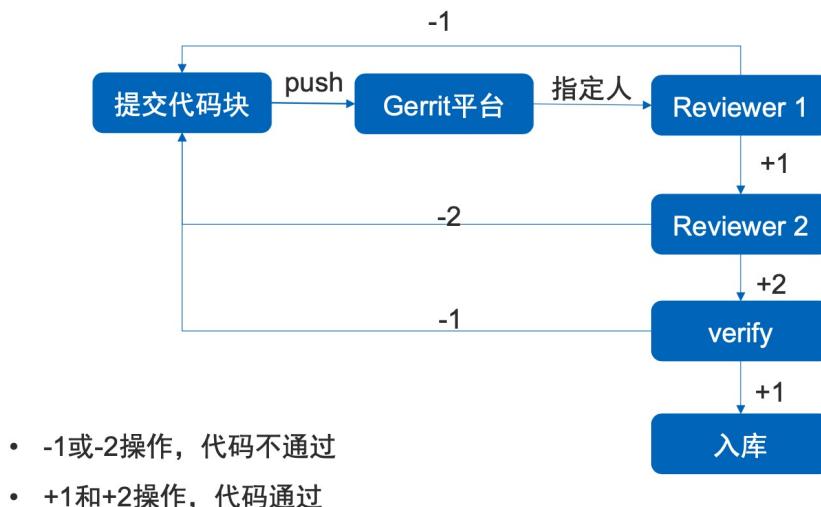
Gerrit用户角色全新可以由管理者进行配置。

Gerrit用户角色:

- 1 admin: 拥有最高权限
- 2 verify: 确认+1权限, 入库权限
- 3 reviewer: 代码评审权限, 包含+1/-1, +2/-2的权限
- 4 developer: push权限

第一个登录gerrit的网页的用户为admin管理员用户, 创建dev帐号、review帐号和verify帐号, 创建dev、review和verify用户组并添加相应用户。

Gerrit使用流程:



Gerrit相关配置

接下来还有很多内容需要设置一下。我们依次来看看吧。

profile

注意设置Username, 代码同步时需要用到。

初次登录时，Full Name 和 Email Address 字段都是空的，要设置一下，没有设置之前，右上角显示的用户名也是 admin

Preferences

Preferences 页面用于配置gerrit的web页面，我一般把时间格式改成习惯的24小时制，同时确保 Email Notification 处于开启状态，这个默认就是开启的；然后在 Show Change Number In Changes Tables 打上勾，这样能清楚地看到每个审核的编号，邮件里面显示的也是这个编号。

Watched Projects

这里有必要先说一下gerrit的两个默认项目：

我们点击左上方的菜单栏 Projects -> List，就能看到两个默认的项目 All-Projects 和 All-Users，这两个工程是两个基础的工程，我们新建的工程默认都是继承自 All-Projects 的权限。关于权限部分我们在后面的章节详细介绍。

所以在 Watched Projects 菜单中，就是当前用户要监听的项目，当这些项目发生变化时，你会收到邮件提醒，如果你选择了 All-Projects，那么就意味着你要监听所有的工程，因为所有工程都会默认继承自 All-Projects。

注意，后面的那些选项，勾选了某一项，就表示仅仅给当前项发送邮件提醒，保险期间，我们就全部勾上就好了。

Contact Information

The screenshot shows the Gerrit web interface under the 'Contact Information' tab. The left sidebar includes options like Profile, Preferences, Diff Preferences, Edit Preferences, Watched Projects, Contact Information (which is selected and highlighted with a red box), SSH Public Keys, HTTP Password, Identities, and Groups. The main content area displays fields for Username (admin), Full Name (xwlpeng), and Preferred Email (lpeng@microwu.com). A 'Register New Email ...' button is also present. A 'Save Changes' button is at the bottom. The top navigation bar shows 'All My Projects People Plugins Documentation' with 'Open Merged Abandoned' filters.

这里是当前账号的配置，你可以在这里把full-name填写完全，这样右上角也会同步更新你刚设置好的名称。

注意这里的Preferred Email有两种设置方法：

1. 如果你的邮件服务器配置完成了，可以点击Register New Email，你就会收到一封确认邮件，确认之后就能设置好了。
2. 通过SSH在命令行中进行配置，通过命令行进行配置是最方便快捷，也是最优先推荐的方法。不过因为SSH命令行的方式需要配置SSH公钥，所以我们这里先留空，一会通过命令行来配置。

SSH Public Keys

The screenshot shows the Gerrit web interface under the 'SSH Public Keys' tab. The left sidebar includes Profile, Preferences, Diff Preferences, Edit Preferences, Watched Projects, Contact Information (selected and highlighted with a red box), SSH Public Keys (highlighted with a red box), HTTP Password, Identities, and Groups. The main content area shows a table with one row: Status (ssh-rsa), Algorithm (AAAAB3NzaC1yc2EAAQABAAAQ...), Key (fBPCyp8RvY...), and Comment (comment). Below the table is a 'Add SSH Public Key' section with a 'How to Generate an SSH Key' link. At the bottom are 'Clear', 'Add', and 'Close' buttons. The top navigation bar shows 'All My Projects People Plugins Documentation' with 'Open Merged Abandoned' filters.

这里需要把你的公钥内容拷贝出来，然后粘贴到对话框中。

Groups

The screenshot shows the Gerrit web interface under the 'Groups' tab. The left sidebar includes Profile, Preferences, Diff Preferences, Edit Preferences, Watched Projects, Contact Information, SSH Public Keys, HTTP Password, Identities (selected and highlighted with a red box), and Groups. The main content area shows a table with three rows: Group Name (Administrators, Anonymous Users, Registered Users), Description (Gerrit Site Administrators), and Visible To All (checkboxes checked for Administrators and Registered Users). The top navigation bar shows 'All My Projects People Plugins Documentation' with 'Open Merged Abandoned' filters.

最后来看一下gerrit的分组。图片里面是gerrit默认的几个分组，我们需要知道的是 Administrator 就是管理员分组， Anonymous Users 指的是所有添加到gerrit数据库中的成员都默认加入的一个组。之后我们还可以建立新的分组，加入新的成员等等。

示例：

接下来我们来做一个演示，看看一个新的成员是如何被添加到gerrit服务器中，然后他们又是如何协同工作的。

这里一共涉及到两个角色，一个是管理员，一个是普通成员。

管理员设置SSH

在之前的文中我们提到过，gerrit自带的H2数据库就完全够用了，对成员的管理，邮件添加等操作，均可以通过SSH来完成。那第一步我们就来看一下管理员如何才能远程SSH到gerrit服务器。

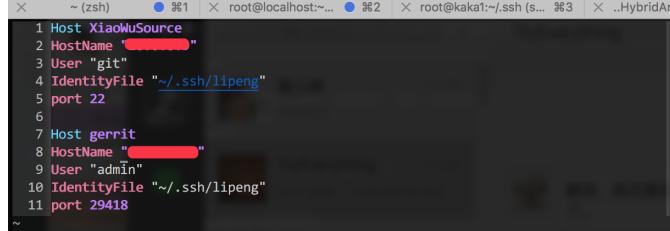
首先确保在之前，已经成功把你的公钥添加到了web页面账户中。

接下来，需要修改之前`~/.ssh/`文件夹下面的config文件，我们拿我的config文件做为示例，做个讲解。

我们还是先进入到`~/.ssh/`文件夹中

```
{17-01-18 11:27}MichaelLees-MacBook-Pro:~/ssh KaKa% ll  
total 40  
-rw-r--r-- 1 KaKa staff 173B Jan 13 14:55 config  
-rw-r--r-- 1 KaKa staff 4.8K Jan 16 15:42 known_hosts  
-r----- 1 KaKa staff 1.6K Jun 8 2016 lipeng  
-r----- 1 KaKa staff 402B Jan 13 14:33 lipeng.pub  
{17-01-18 11:27}MichaelLees-MacBook-Pro:~/ssh KaKa% vim lipeng.pub  
{17-01-18 13:32}MichaelLees-MacBook-Pro:~/ssh KaKa% vim config
```

然后查看一下config文件：`vim config`



```
1 Host XiaoWuSource  
2 HostName "████████"  
3 User "git"  
4 IdentityFile "~/.ssh/lipeng"  
5 port 22  
6  
7 Host gerrit  
8 HostName "████████"  
9 User "admin"  
10 IdentityFile "~/.ssh/lipeng"  
11 port 29418
```

我们看到这里面有两个Host部分，我们重点来看第2个Host部分，这个是我们新建的，用于连接到gerrit服务器的配置。

照猫画虎，对于我们之前建立在`192.168.1.100`的gerrit服务器来说，你的Host配置可能如下：

1	Host gerrit
2	HostName "192.168.1.100"
3	User "admin"
4	IdentityFile "~/.ssh/id_rsa"
5	port 29418

User要和我们在gerrit服务器上注册的名称保持一致(不是full-name)，认证文件注意要和公钥对应的私钥文件，端口要填写gerrit服务的端口号，这里是默认的`29418`。

配置完config文件，我们就可以SSH到gerrit了，我们来尝试一下吧：

```
[17-01-18 13:52}MichaelLees-MacBook-Pro:~/ssh KaKa% ssh gerrit -l admin  
**** Welcome to Gerrit Code Review ****  
Hi xw.lipeng, you have successfully connected over SSH.  
Unfortunately, interactive shells are disabled.  
To clone a hosted Git repository, use:  
git clone ssh://admin@████████:29418/REPOSITORY_NAME.git  
Connection to ██████████ closed.
```

我们在管理员的机器上，输入`ssh gerrit -l admin`命令，就可以得到gerrit服务器的响应，只不过因为我们禁用了shell，所以连接很快断开了，没关系，这样证明做为管理员，已经可以通过命令行对gerrit服务器进行一系列的操作了。

添加普通成员

在管理员添加新的组员之前，我们需要先在普通成员的机器上生成ssh的公私钥，这里方便描述，我们把这个普通成员命名为test3。

在test3的电脑命令行中，生成利用`ssh-keygen`命令，生成公私钥。

```
[root@kaka1 ~]# cd ~/.ssh  
[root@kaka1 .ssh]# ll  
total 12  
-rw----- 1 root root 1675 Jan 16 09:02 id_rsa  
-rw-r--r-- 1 root root 392 Jan 16 09:02 id_rsa.pub  
-rw-r--r-- 1 root root 231 Jan 16 16:31 known_hosts  
[root@kaka1 .ssh]#
```

我们就使用默认的`id_rsa`命名好了。

接下来，test3成员需要把`id_rsa.pub`公钥发送给管理员，这样管理员才能正常把test3添加到gerrit用户组中。

我们假设管理员将test3的pub公钥放到了`~/home`目录下，也就是说，在管理员的电脑上，test3的公钥存放在`~/home/id_rsa.pub`文件，当然我们也可以重新把它命名为`test3.pub`，方便演示我这里就不做更名处理了。

接下来，管理员在命令行中输入如下的命令来完成添加普通成员的操作。注意：这个命令很强大很方便，可以一步到位地把成员的的名称，全名，邮箱以及ssh公钥认证全部设置好。

	\$ cat ~/home/id_rsa.pub ssh gerrit gerrit create-account --full-name test3
--	---

接下来我们来详细看一下这个命令：

- 符号把这个命令分成了两部分，第一部分的 `cat ~/home/id_rsa.pub` 表示把test3的公钥内容读入到输入流中
- `ssh gerrit`是我们之前在 `~/.ssh/config` 中配置好的gerrit服务器地址
- 又接着一个gerrit表示通过ssh中输入gerrit命令来进行相关操作
- `create-account` 表示要新建用户。注意，新建的用户名写在最后面，中间是其他参数
- `full-name` 就如同页面中的全名，我们这里命名为test3
- `email`表示该用户的email地址，我们填入 `test3@microwu.com`
- `ssh-key` – 注意，最后的 `-` 表示从输入流中读取ssh的公钥内容，也就是 `-` 符号之前我们读入的 test3用户公钥内容
- 最后面加上我们要create的用户名

这个命令执行完之后，管理员就把test3用户加入到了gerrit用户组中，并且设置了他的全称，邮件以及公钥文件，是不是一步到位，非常方便？？

这里我们回过头来，在管理员首次登陆web页面进行修改配置的时候，我们说过，管理员的邮箱可以通过命令行来设置，是的，同样通过ssh命令行：

```
1 $ ssh gerrit gerrit set-account --add-email admin@microwu.com admin
```

这个命令就表示为我们的 `admin` 用户添加email `admin@microwu.com`。执行完这个命令，再回到web界面上的用户设置界面，看看是不是管理员的email已经被设置好了？？

修改用户所在组

接下来我们看一下怎样修改test3用户所在的组吧。我们知道他已经处在 `Anonymous Users` 组中了，那我们想要新建一个组，就叫 `test_user` 吧，我们来看一下

All My Projects People Plugins Documentation Search term Changes Search xv.lipeng

List Groups Create New Group

Groups

Filter

Group Name	Description	Visible To All
Administrators	Gerrit Site Administrators	
Non-interactive Users	Users who perform batch actions on Gerrit	

Powered by Gerrit Code Review (2.13.5) | Press "?" to view keyboard shortcuts

我们在gerrit页面的顶部，点击 `People` → `list`，看一下默认的两个分组，`Administrator` 和 `Non-interactive Users`，这两个分组我们都能从字面上理解是什么意思。我们注意到 `Anonymous Users` 这个分组并没有显示在页面，因为它是匿名的嘛，所有的用户自动添加到这个分组中了。

All My Projects People Plugins Documentation Search term Changes Search xv.lipeng

List Groups Create New Group

Create Group

Create New Group test_user Create Group

Powered by Gerrit Code Review (2.13.5) | Press "?" to view keyboard shortcuts

选择 `Create New Group`，输入我们要添加的新分组 `test_user`

All My Projects People Plugins Documentation Search term Changes Search xv.lipeng

List Groups Create New Group

Group test_user

General Members Audit Log

Members Name or Email Add
Member Email Address
xw.lipeng ipeng@microwu.com

Included Groups Group Name Add
Group Name Description

Delete

Powered by Gerrit Code Review (2.13.5) | Press "?" to view keyboard shortcuts

新的分组中，我们看到管理员的账号被自动添加了进来

All My Projects People Plugins Documentation Search term Changes Search xv.lipeng

List Groups Create New Group

Group test_user

General Members Audit Log

Members Name or Email Add
test test2@test2@microwu.com dress
test3@test3@microwu.com microwu.com

Included Groups Group Name Add
Group Name Description

Delete

Powered by Gerrit Code Review (2.13.5) | Press "?" to view keyboard shortcuts

我们在 `Add` 搜索栏中输入 `test`，就会自动显示出来管理员之前在命令行中创建的test3用户，看到 `full-name` 和 `email` 了吧，都已经添加完成了！

test3用户已经添加到了test_user分组中了。

创建第一个项目，配置权限管理

添加project，选择 Inherit From All-Projects，当然也可以自定义Parent Project。

Project Name: getui-test
Rights Inherit From: All-Projects
Create initial empty commit:
Only serve as parent for other projects
Create Project
Parent Suggestion: Project Name: All-Projects
Project Description: Access inherited by all other projects.

添加Verified标签支持，这里修改All-Project项目的project.config，所有继承自All-Project的项目自动添加Verified标签，也可针对项目自定义是否verify。

clone | clone with commit-msg hook | Anonymous HTTP | SSH | HTTP
git clone ssh://lvgx@192.168.14.217:29418/All-Projects

Description
Access inherited by all other projects.

Project Options
State: Active
Submit Type: Merge If Necessary
Automatically resolve conflicts: TRUE
Create a new change for every commit not in the target branch: FALSE
Require Change-ID in commit message: TRUE
Maximum Git object size limit:

Contributor Agreements
Require Signed-off-by in commit message: FALSE
Save Changes

Project Commands
Commands: Run GC | Create Change | **Edit Config**

```
0 *      value = +2 Looks good to me, approved
1 [label "Verified"]
2 *      function = MaxWithBlock
3 *
4 *      value = -1 Fails
5 *      value = 0 No score
6 *      value = +1 Verified
6 *      defaultValue = 0
```

创建用户组

All My Projects **People** Plugins Documentation

List Groups Create New Group

Group gexintest-dev

General **Members**

Name or Email	Add
[User icons]	[Email Address]

添加相关用户权限

All My Projects People Plugins Documentation

List General Branches Access Dashboards Create New Project

Search term

Project gixin-test

Edit Rights Inherit From: All-Projects History: (gitweb)

Reference: refs/*

Owner	ALLOW <input checked="" type="checkbox"/> project-verifier	<input type="checkbox"/> Exclusive
Read	ALLOW <input checked="" type="checkbox"/> Non-Interactive Users ALLOW <input checked="" type="checkbox"/> gexintest-dev ALLOW <input checked="" type="checkbox"/> gexintest-review ALLOW <input checked="" type="checkbox"/> gexintest-verify	<input type="checkbox"/> Exclusive
Abandon	ALLOW <input checked="" type="checkbox"/> gexintest-review	<input type="checkbox"/> Exclusive
Label Code-Review	-2 <input checked="" type="checkbox"/> +2 <input checked="" type="checkbox"/> gexintest-review	review 用户组 <input type="checkbox"/> Exclusive
Label Verified	-1 <input checked="" type="checkbox"/> +1 <input checked="" type="checkbox"/> Non-Interactive Users -1 <input checked="" type="checkbox"/> +1 <input checked="" type="checkbox"/> project-verifier	verify 用户组, 这里配置jenkins自动化用户 <input type="checkbox"/> Exclusive
Submit	ALLOW <input checked="" type="checkbox"/> gexintest-review	<input type="checkbox"/> Exclusive

Reference: refs/for/*

Push	ALLOW <input checked="" type="checkbox"/> gexintest-dev	develop 提交的for分支权限 <input type="checkbox"/> Exclusive
------	---	---

将代码库同步到本地 (SSH/Http)

HTTP 方式:

HTTP Password 密码在 账户 - -> Settings --> HTTP Password 处获取。

All My Projects People Plugins Documentation

List General Branches Access Dashboards Create New Project

Settings

- Profile
- Preferences
- Watched Projects
- Contact Information
- SSH Public Keys
- HTTP Password**
- Identities
- Groups

Username: lvgx

Password:

Generate Password | Clear Password

SSH方式:

添加SSH Public Key。

All My Projects People Plugins Documentation

List General Branches Access Dashboards Create New Project

Settings

- Profile
- Preferences
- Watched Projects
- Contact Information
- SSH Public Keys**
- HTTP Password
- Identities
- Groups

Status	Algorithm	Key	Comment
<input type="checkbox"/>	ssh-rsa	AAAAB3NzaC1yc2EAAAQABAAQ...rCmy1Lyw==	lvgx@getui.com

Add Key ...

Server Host Key

Fingerprint: SHA256:En...=C1yc2EAAAQABAAQDdzYu14pB...PdcGZI

Clone代码到本地

All My **Projects** People Plugins Documentation

List General Branches Access Dashboards Create New Project

提供http和ssh两种clone方式,自行测试

Project gixin-test

务必选中clone with commit-msg hook

clone | clone with commit-msg hook | Anonymous HTTP | SSH | HTTP |

```
git clone http://lvgx@192.168.14.217:8281/gixin-test && scp -p -P 29418 lvgx@192.168.14.217:hooks/cc
```

Description

```
xz@xzdeMacBook-Pro:~$ git clone ssh://lvgx@192.168.14.217:29418/gixin-test &&
scp -p -P 29418 lvgx@192.168.14.217:hooks/commit-msg gixin-test/.git/hooks/
Cloning into 'gixin-test'...
remote: Counting objects: 69, done
remote: Finding sources: 100% (69/69)
remote: Total 69 (delta 14), reused 64 (delta 14)
Receiving objects: 100% (69/69), 7.51 KiB | 0 bytes/s, done.
Resolving deltas: 100% (14/14), done.
Checking connectivity... done.
commit-msg
100% 4362 4.3KB/s 00:00
```

git clone 后面的scp 是hook钩子,对git push提交进行监听, 监听到后进行拦截操作, 拦截后进行后续审核。

commit-msg ,提供自动写入change-id 至git log内功能

提交第一个change

All My Projects People Plugins Documentation

status:open

Search for status:open

Subject	Status	Owner	Project	Branch	Updated	Size	CR	V
update Gerrit test	Open	lgx	gixin-test	master	11:24 AM			

Subject: update Gerrit test

Author: lgx<lgx@lgx.com> Jun 27, 2018 11:24 AM

Committer: lgx<lgx@lgx.com> Jun 27, 2018 11:24 AM

Commit: abd736e25565e0909e1ad9e650b60ec394951d

Parent(s): 0000a4c5cc00d5911930cfba5c50ca232636a

Change-ID: 3bd517170daaee77339fe41f275042b71f9740b

File Path: A xx_test

File: Commit Message

Comments Size: +1 0

Code-Review: Verified +1 fe-d

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。

gixin-test / xx_test

Patch Set: Base 1

Patch Set 1

xx_test

Reply... **Code-Review:2**

Owner: lgx
Reviewers: fe-d, lgx

Project: gixin-test
Branch: master
Topic:
Strategy: Merge If Necessary
Updated: 6 minutes ago

Cherry Pick | Rebase | Abandon | Follow-Up

Code-Review: Verified +1 fe-d

Submit

本地代码库更新, 获取最新入库代码

代码submit后通过git pull - - rebase 更新代码。

```

xz@xzdeMacBook-Pro:~/gixin-test$ git st
On branch master
Your branch is ahead of 'origin/master' by 1 commit.
  (use "git push" to publish your local commits)
nothing to commit, working directory clean
xz@xzdeMacBook-Pro:~/gixin-test$ git ls -2
a8d736e 2016-06-27 (HEAD -> master) update:gerrit test [lvgx]
080ea4d 2016-06-12 (origin/master) update:gerrit test 2 [daizi]
xz@xzdeMacBook-Pro:~/gixin-test$ git pull --rebase
From ssh://192.168.14.217:29418/gixin-test
  080ea4d..a8d736e master      -> origin/master
Current branch master is up to date.
xz@xzdeMacBook-Pro:~/gixin-test$ git st
On branch master
Your branch is up-to-date with 'origin/master'.
nothing to commit, working directory clean

```

Gerrit入门实战-初级修补

如果所有代码提交均被打回，可以进行暴力回滚：git reset <commit>，接着重新提交Gerrit，再进行Gerrit审查入库。

The screenshot shows a Gerrit commit page for a project named 'gixin-test'. The commit subject is 'updatereset test again'. The 'Follow-Up' section contains buttons for 'Cherry Pick', 'Rebase', and 'Abandon'. A red arrow points from the 'Abandon' button to the 'Follow-Up' section. Another red arrow points from the 'Follow-Up' section to the text '采用reset方式，需abandon本次commit'.

Gerrit入门实战-高级修补

如果单个提交打回，则可交互式回滚：git rebase -i <commit>，修改指定commit点：git commit --amend，完成所有commit点处理：git rebase --continue，然后重新提交Gerrit，最后Gerrit审查入库。

Rebase前

The screenshot shows a Gerrit commit page for a project named 'gixin-test'. The commit subject is 'update:rebase test'. The subject field is highlighted with a red box.

Rebase后

The screenshot shows a Gerrit commit page for a project named 'gixin-test'. The commit subject is 'update:rebase test, commit again'. The subject field is highlighted with a red box, and the 'Updated' timestamp '11:51 AM' is also highlighted with a red box.

rebase 在同一个点上修改，不会产生审核点，多个commit点同时存在是尤其有用。

```

xz@xzdeMacBook-Pro:~/gexin-test$ git rebase -i HEAD~2
Stopped at 88dd9da673487bf7d3... update:rebase test
You can amend the commit now, with
  git commit --amend
Once you are satisfied with your changes, run
  git rebase --continue
xz@xzdeMacBook-Pro:~/gexin-test$ git commit --amend
[detached HEAD 133045e] update: gexin-test -> refs/for/master
Date: Mon Jun 27 11:49:39 2016 +0800
1 file changed, 2 insertions(+)
xz@xzdeMacBook-Pro:~/gexin-test$ git st
interactive rebase in progress; onto 88dd9da673487bf7d3
Last command done (2 commands done):
  pick 40d7979 update:reset test again
  edit 88dd9da673487bf7d3--rebase test
No commands remaining.
You are currently editing a commit while rebasing branch 'master' on 'a8d736e'.
  (use "git commit --amend" to amend the current commit)
  (use "git rebase --continue" once you are satisfied with your changes)

nothing to commit, working directory clean
xz@xzdeMacBook-Pro:~/gexin-test$ git rebase --continue
Successfully rebased and updated refs/for/master.
xz@xzdeMacBook-Pro:~/gexin-test$ git push origin HEAD:refs/for/master
Counting objects: 3, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 334 bytes | 0 bytes/s, done.
Total 3 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1)
remote: Processing changes: updated: 1, refs: 1, done
remote: (W) 133045e: no files changed, message updated
remote:
remote: Updated Changes:
remote: http://192.168.14.217:8281/3099 update:rebase test , commit again
remote:
To ssh://lvgx@192.168.14.217:29418/gexin-test
 * [new branch]      HEAD -> refs/for/master

```

Gerrit经验谈

第一，Git别名绑定，添加别名字段，通过git review master这样简单语法提交到master源端分支，可以省去很多工作。修改系统目录或者项目目下的.gitconfig文件，添加

```

[alias]
review = !sh -c 'git push origin HEAD:refs/for/$1' -

```

也可通过git config --global alias.review 命令修改

```

[xz@xzdeMacBook-Pro:~/gexin-test$ git push origin HEAD:refs/for/master
Counting objects: 3, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 334 bytes | 0 bytes/s, done.
Total 3 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1)
remote: Processing changes: updated: 1, refs: 1, done
remote: (W) 133045e: no files changed, message updated
remote:
remote: Updated Changes:
remote: http://192.168.14.217:8281/3099 update:rebase test , commit again
remote:
To ssh://lvgx@192.168.14.217:29418/gexin-test
 * [new branch]      HEAD -> refs/for/master
[xz@xzdeMacBook-Pro:~/gexin-test$ git review master
Counting objects: 3, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 336 bytes | 0 bytes/s, done.
Total 3 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1)
remote: Processing changes: updated: 1, refs: 1, done
remote: (W) 0e349fa: no files changed, message updated
remote:
remote: Updated Changes:
remote: http://192.168.14.217:8281/3099 update:rebase test , commit again,3
remote:
To ssh://lvgx@192.168.14.217:29418/gexin-test
 * [new branch]      HEAD -> refs/for/master

```

第二，工具只是一部分，更重要的是人与人当面的沟通交流，大家讨论一个好的解决方案，才能更好的解决问题。没有交流，工具也就失去了意义。

最后，关于review积压问题，要避免提交积压，代码审核过程要及时完成，避免 Code Review流于形式。从个推实际使用效果看，Gerrit在核心代码质量控制、知识传承、团队培养等方面都具备很高的实用价值，推荐给广大开发团队用。

