

Git 进阶指导手册

杨岚岚

2015.12.15

目录

一. ssh 认证	1
1.1 生成 ssh 公钥	1
1.2 ssh 设置主机别名	2
1.3 ssh-agent 密钥管理	3
二. git 命令的使用	3
2.1 git 用户设置	3
2.2 git 提交代码	4
2.3 git 合并代码	4
2.3.1 git 分支合并	4
2.3.2 git cherry-pick 合并指定提交	5
2.3.3 git patch 补丁的使用方法	5
2.4 git 本地与服务器协同工作	6
三. Gerrit 的使用	8
3.1 加入 ssh 公钥	8
3.2 测试 gerrit 下的 ssh 连接	8
3.3 下载 git 仓库	9
3.4 下载钩子脚本	9
3.5 提交代码到 gerrit 审核服务器	9

一. ssh 认证

1.1 生成 ssh 公钥

ssh 服务的安装方式如下：

```
sudo apt-get install openssh-client  
sudo apt-get install openssh-server
```

生成 ssh 公钥密钥对，

```
ssh-keygen -f ~/.ssh/username
```

此处的 username 需要与邮箱名一致，例如我的 username 为 lanlan.yang, 生成 ssh 公钥的命令为 ssh-keygen -f ~/.ssh/lanlan.yang, 执行命令后，可以设置密码，

也可以直接回车，即设置为空密码。

```
ssh-keygen -f ~/.ssh/lanlan.yang
Generating public/private rsa key pair.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/gerrit2/.ssh/lanlan.yang.
Your public key has been saved in /home/gerrit2/.ssh/lanlan.yang.pub.
The key fingerprint is:
e1:9f:b3:84:a3:69:47:88:65:a1:7c:67:cc:bd:a9:0a gerrit2@ubuntu
The key's randomart image is:
+--[ RSA 2048]-----+
|           |
|      .    |
|   .. +..   |
|    o +.=..  |
|     = +S   o  |
|    ..oo.    |
|   E  .o.=    |
|    ..ooo o   |
|    .+o  .    |
+-----+
```

cd ~/.ssh，可以看到生成了两个文件，lanlan.yang.pub 和 lanlan.yang，其中 lanlan.yang.pub 是 ssh 公钥，需要发送给配置管理员开通权限，私钥 lanlan.yang 自己保留。

ssh 公钥认证现在存在的普遍问题是，由于本机存在多对 ssh 公钥，导致系统访问服务器错误选择了未被授权的私钥，即便配置管理员已经开通了权限，但是依然无法下载代码。很多新同事刚安装 ubuntu 系统就发现.ssh 目录下存在多对 ssh 公钥，可能大家拿到的 ubuntu 系统本身就存在 ssh 公钥对。最简单的解决方法，是删除本地多余的公钥对。

1.2 ssh 设置主机别名

修改 ssh 配置文件，设置主机别名，也可以解决本机存在多对 ssh 公钥导致访问服务器失败的问题。

设置方法如下：

在~/.ssh 目录新建文件 config，然后在文件中添加以下内容

```
host szgit_master
    user szgit
    port 22
    hostname 10.240.2.41
    identityfile ~/.ssh/lanlan.yang
```

下面对 config 文件中的参数进行说明。

szgit_master 是主机别名。
szgit 是 ubuntu 登陆的用户名。
22 是 ssh 端口号。
10.240.2.41 是要访问的代码服务器的 IP。
~/ssh/lanlan.yang 完整的客户端私钥路径。

下面举例说明主机别名的使用方法，
不使用主机别名时，我们下载仓库代码的命令为：
git clone szgit@10.240.2.41: msm8909.git
使用主机别名后，下载仓库代码的命令为：
git clone szgit_master:msm8909.git
使用主机别名时访问 szgit_master 服务器，会根据 config 文件，使用
~/ssh/lanlan.yang 私钥去访问。

1.3 ssh-agent 密钥管理

重新安装 ubuntu 系统，或误删了~/ssh 目录里 ssh 公钥对，就需要重新生成 ssh 公钥对，并将 ssh 公钥发给配置管理员更新公钥，这种情况下可能依然访问不了已经开通权限的 git 仓库，此时开发工程师可以尝试在本机执行以下两条命令来解决问题。

```
ssh-agent bash
ssh-add ~/ssh/当前用的私钥(比如：lanlan.yang)
```

ssh-agent 是为解决每次登陆远程机器都需要输入 passphrase 的问题而引入的。ssh-agent 启动后，可通过 ssh-add 将私钥加入 agent。ssh-add 会提示用户输入 passphrase 以解密私钥，然后将解密后的私钥纳入 agent 管理。agent 可同时管理多个私钥。如果 ssh-agent 中有多个私钥，会依次尝试，直到认证通过或遍历所有私钥。

二. git 命令的使用

2.1 git 用户设置

git 用户账号和邮箱地址建议大家按照邮箱名称来设置，如果任意设置，有可能导致团队协作时授权出现问题。

```
git 账户名称设置: git config --global user.name lanlan.yang
git 用户邮箱设置: git config --global user.email lanlan.yang@ck-telecom.com
```

2.2 git 提交代码

git 本地仓库分工作区，暂存区和版本库共三个区域。一次完整的 git 提交，是先使用 `git add filename` 命令，将本地所做的修改添加到暂存区，然后执行 `git commit -m “注释”` 命令，提交到版本库。

Git 的一次提交对应一个功能，一般情况下，每一次提交改动的文件不会太多，如果某一次提交改动的文件比较多，使用 `git add .` 或 `git add --all` 命令可以将所有工作区内的改动添加到暂存区，然后 `git commit` 提交到版本库。

提交后检视代码，发现注释写得不准确，使用 `git commit --amend -m “修改后的注释”` 命令，来更正注释。

2.3 git 合并代码

2.3.1 git 分支合并

(1)git merge 命令

`git merge branchA`，此命令用来合并本地的分支，假设我们现在的工作分支是 `master`，则将本地分支 `branchA` 的内容合并到工作分支 `master` 上来。

(2)git pull 命令

`git pull origin branchA`，此命令是将远程代码服务器上的分支 `branchA` 的内容合并到本地的分支上来，它的作用等同于执行 `git fetch origin branchA` 和 `git merge origin/branchA` 这两个命令。

(3)git rebase 命令

`git rebase` 有点类似 `git merge`，但两者又有不同，`git merge` 是那种横冲直撞型的，所有的提交一起合并，冲突也要一起解决，`git rebase` 则是一个提交一个提交的合并，一旦有冲突，解决之后才能继续合并下一个提交。

`git rebase` 常用的命令有：

- `git rebase 分支名`，
- `git rebase --continue`，
- `git rebase --skip`，
- `git rebase --abort`。

下面详细说明，`git rebase` 合并分支的用法。

`git rebase branchA`，执行此命令则是将 `branchA` 上的补丁一个个的打到当前的工作分支，假设工作分支为 `master`，如果没有冲突发生，则 `git rebase` 会完成所有的合并操作后自动结束。如果冲突发生，`git rebase` 操作就会中断，并提示用户，有冲突发生，我们有三种选择继续进行下面的工作。第一种是手动解决冲突之后，执行 `git add` 命令，然后 `git rebase --continue`，继续进行合并；第二种是跳过此冲突不解，继续后面的合并操作，需执行 `git rebase --skip` 命令；最后一种是直接结束此次 `git rebase` 合并操作，需执行 `git rebase --abort` 命令。

2.3.2 git cherry-pick 合并指定提交

git cherry-pick commitid, 只要存在在本地的 git 版本库中, 任意一条 git 提交, 我们都可以通过它的哈希码, 将它合并到工作分支上来, 使用此命令时, 同样有可能出现冲突, 那么就需要手动解决冲突。

2.3.3 git patch 补丁的使用方法

git 补丁的生成方式有 git format-patch 和 git diff 两种, 不同的方式生成的补丁, 打补丁的方式也不同。

(1) git format-patch 与 git apply 的使用

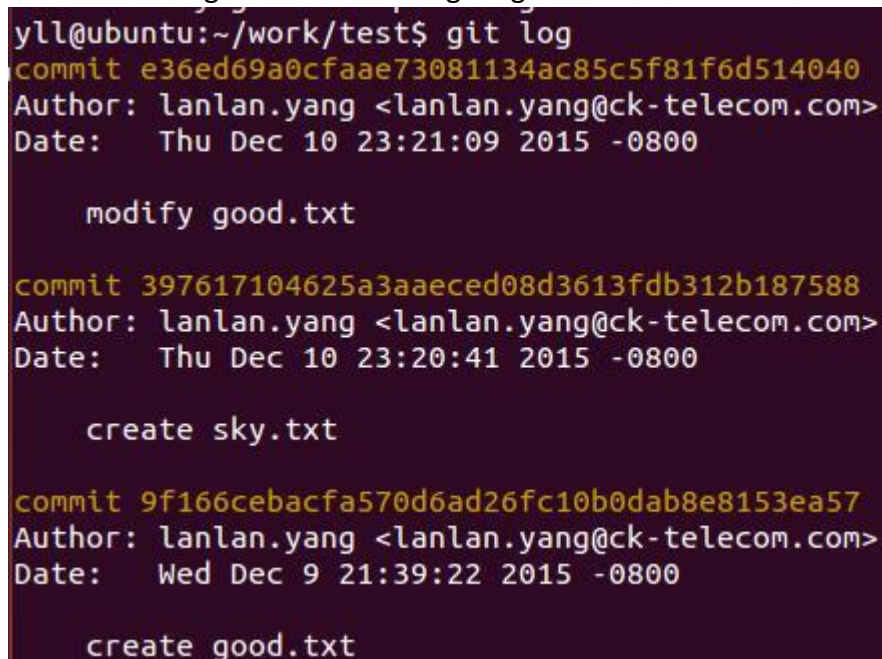
git format-patch 常见使用方法如下所示:

git format-patch -N commitid 或 git format-patch -N HEAD^^...^

git format-patch 生成的每一个 patch 都是针对每一条提交的。-N 是生成的 patch 数目, 指定的 commitid 为第 N 个 patch, 在此在之前的提交为第 N-1 个 patch, 依次类推。如果我们只需要针对某一个提交生成一个 patch, 直接执行命令:

git format-patch -1 commitid 即可。

举例说明, 在 git 仓库 test 中, git log 查看历史提交记录如下:



```
yll@ubuntu:~/work/test$ git log
commit e36ed69a0cf...
Author: lanlan.yang <lanlan.yang@ck-telecom.com>
Date: Thu Dec 10 23:21:09 2015 -0800

    modify good.txt

commit 397617104625a3a...
Author: lanlan.yang <lanlan.yang@ck-telecom.com>
Date: Thu Dec 10 23:20:41 2015 -0800

    create sky.txt

commit 9f166cebaf570d6ad26fc10b0dab8e8153ea57
Author: lanlan.yang <lanlan.yang@ck-telecom.com>
Date: Wed Dec 9 21:39:22 2015 -0800

    create good.txt
```

图 2.1

执行 git format-patch -1 HEAD 命令, 自动生成生成 0001-modify-good.txt.patch, 补丁的命名是编号+提交注释+.patch 后缀, 我们可以直观地看出这个补丁是针对哪个提交生成的。

执行 git format-patch -2 HEAD^命令, 自动生成

0001-create-sky.txt.patch

0002-modify-good.txt.patch

一共两个补丁。

通过 `git format-patch` 命令生成的补丁，在打补丁时可以直接使用命令 `git apply` 补丁名

此命令是直接新增一条提交。如果生成多条补丁，打补丁时要按照序号，依次打上去。即先执行命令 `git apply 0001-create-sky.txt.patch`，然后执行命令 `0002-modify-good.txt.patch`，由上图 2.1 可以看出，补丁前面的序号就是按照代码提交的先后顺序。

(2) `git diff` 与 `patch -pN` 的使用

`git diff` 可以针对连续几次的提交内容生成一个 `patch`。

使用方法是 `git diff commitid1 commit2 >*.patch`

或 `git diff HEAD^...^ HEAD^...^ >*.patch`

使用 `git diff` 命令时，要把提交时间早的 `commitid` 放在前面。`git diff` 生成的 `patch`，在打 `patch` 时使用以下命令

`patch -pN < *.patch`

在打 `patch` 时，参数 `N` 要根据执行打补丁命令的目录级数决定，如果在 `git` 仓库的根目录下打补丁，参数为 `-p1`，第二级目录打补丁，则为 `-p2`，依此类推。通过 `patch -pN < *.patch` 命令打补丁只是修改了本地工作区的文件，还需执行 `git add` 将修改的文件添加到暂存区，然后再使用 `git commit` 命令才可完成整个打补丁的操作。

2.4 `git` 本地与服务器协同工作

团队协作需要共享代码时，常用的命令就是 `git clone`, `git fetch`, `git pull`, `git push` 等。

如下图 2.2，可以看到 `git` 本地版本库与代码服务器的关系。

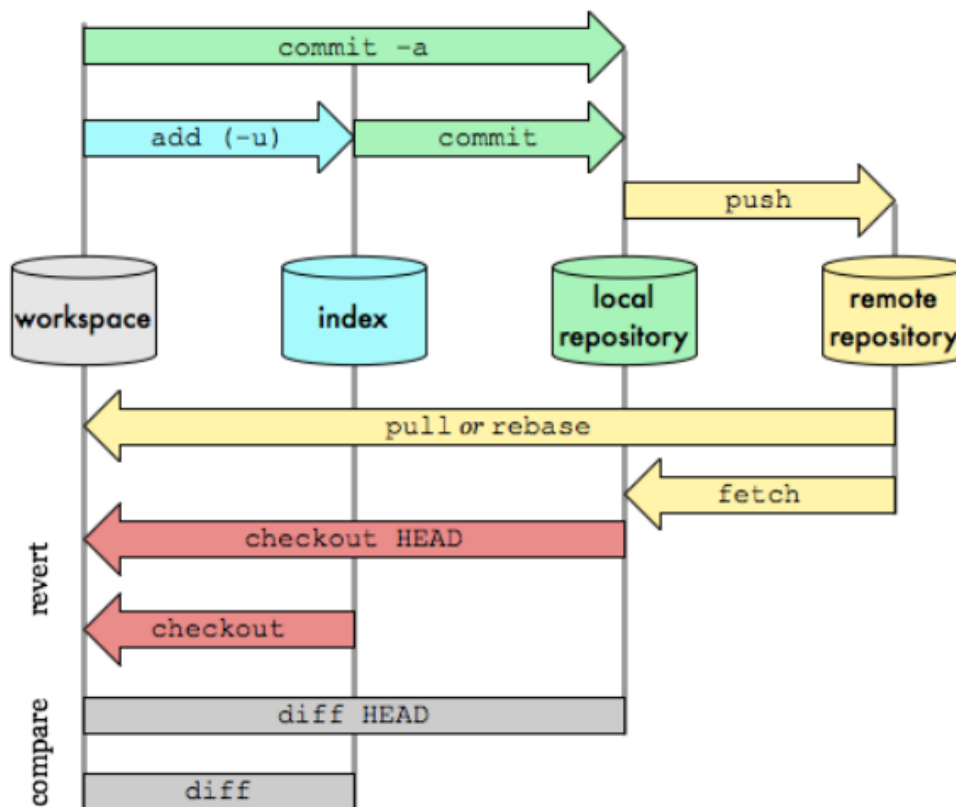


图 2.2

(1) `git clone` 命令是下载 `git` 仓库

使用方法有：

`git clone szgit@10.240.2.41:msm8909`

此命令是下载完整的 `git` 仓库 `msm8909`。

`git clone szgit@10.240.2.41:msm8909 -b`

`2_rlc01a-s00a_msm8909-la-1-2_amss_oem_standard_for_ckt`

此命令是只下载 `git` 仓库 `msm8909` 的分支

`2_rlc01a-s00a_msm8909-la-1-2_amss_oem_standard_for_ckt` 的内容到本地。

(2) `git fetch`

`git fetch origin master`

从服务器上的 `master` 分支取更新到本地，但并不会自动合并，如果要合并，还需执行 `git merge origin/master` 命令。

(3) `git pull`

`git pull origin master` 即是从服务器上的 `master` 分支获取代码更新到本地，同时将更新合并到本地的工作分支上来。

(4) `git push`

`git push origin master` 是将本地的工作分支推送到服务器上的 `master` 分支上去。

三. Gerrit 的使用

Gerrit 是为 Git 引入的代码审核服务器，其代码审核是强制性的，就是说，向 Git 版本库的推送必须要经过 Gerrit 服务器，修订代码必须经过代码审核的工作流程之后，经批准才能合并到正式版本库中。

3.1 加入 ssh 公钥

管理员开通 gerrit 的权限之后，我们登录到 gerrit 服务器之后的第一件事就加入 ssh 公钥。添加 ssh 公钥的方法是，登录用户之后，用户名下面有一个 settings 按钮，点击 settings 按钮，打开 settings 界面，选择 SSH Public Keys，打开下图 3.1

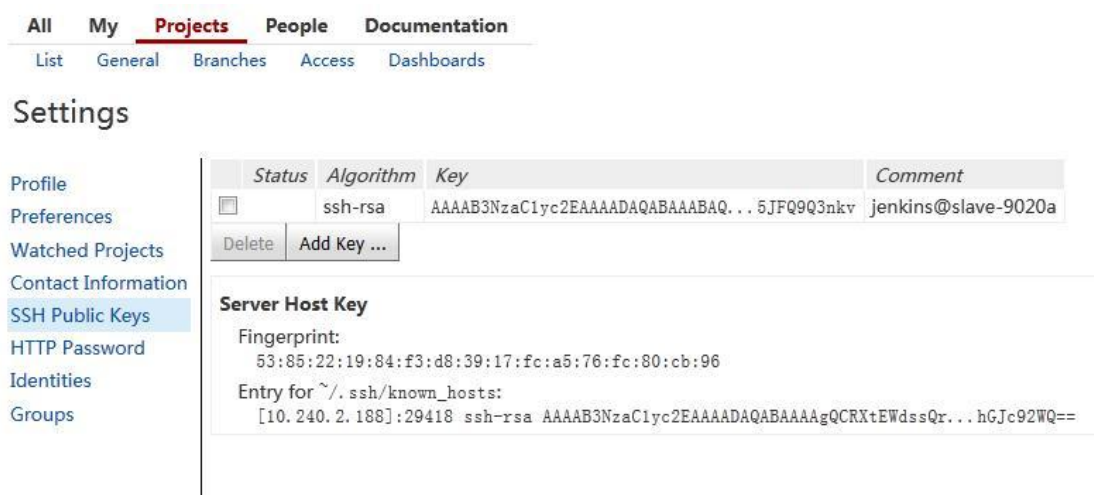


图 3.1

点击 Add Key 按钮，将自己的公钥中的内容拷贝上去。

3.2 测试 gerrit 下的 ssh 连接

添加公钥之后，我们先测试一下是否可以通过 ssh 公钥连接 gerrit 服务器。

执行命令 `ssh -p 29418 -i ~/.ssh/lanlan.yang 10.240.2.188 -l lanlan.yang`

其中 29418 是默认的 gerrit 的 ssh 端口号，10.240.2.188 即为要访问的 gerrit 服务器的 IP。

```
jenkins@slave-9020a:~$ ssh -p 29418 -i ~/.ssh/lanlan.yang 10.240.2.188 -l lanlan.yang
```

```
****      Welcome to Gerrit Code Review      ****
```

```
Hi lanlan.yang, you have successfully connected over SSH.
```


Unfortunately, interactive shells are disabled.

To clone a hosted Git repository, use:

```
git clone ssh://lanlan.yang@10.240.2.188:29418/REPOSITORY_NAME.git
```

Connection to 10.240.2.188 closed.

jenkins@slave-9020a:~\$

显示以上信息，则表示访问 **gerrit** 服务器成功。

如果嫌命令太长，可以设置 **ssh** 主机别名，设置方法在第一章中已经有过详细介绍，这里就不做说明了。

3.3 下载 git 仓库

Git 仓库的下载地址，可以登录 **gerrit** 后获得。打开 **gerrit** 页面，访问 Projects->List，从列表中选择 git 仓库 **test_yhy**，然后选择 **ssh**，可以看到仓库的克隆地址为：

```
git clone ssh://lanlan.yang@10.240.2.188:29418/test_yhy
```

通过上述命令，将 **test_yhy** 仓库下载到本地。

3.4 下载钩子脚本

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，将该钩子脚本拷贝到开发者的本地 GIT 库的钩子脚本目录中，即 **.git/hooks/commit-msg**。

下载钩子脚本的命令为，

```
scp -p -P 29418 lanlan.yang@10.240.2.188:hooks/commit-msg .git/hooks/commit-msg
```

3.5 提交代码到 gerrit 审核服务器

下载钩子脚本成功后，在本地仓库中提交代码，日志中会产生 **Change-Id** 行，如下所示为提交的日志信息。

```
jenkins@slave-9020a:/home/mysoftdata/yll/work/test_yhy$ git log
commit d5179c4b0b5de52df30aaf8e76da26f23fdb9aa
Author: lanlan.yang <lanlan.yang@ck-telecom.com>
Date: Tue Dec 15 13:54:47 2015 +0800
```

```
[test][test][test] create good.txt file
```

Change-Id: I96c620a69fc460d1a5f5dc20bf273493eb19aee6

推送代码到 **gerrit** 服务器时，需要注意的是，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。正确的做法是，向特殊的引用 **refs/for/*** 推送，这样 **Gerrit** 会自动将提交转化为评审任务。正确的推送代码的方法是：

git push origin HEAD:refs/for/master,

此处的 **master** 可以换成项目相关的分支名。待评审完成后，才会将提交合并到相应的分支 **master** 上去。