Connect系统

---代码仓库

1.0

1.0

修订记录

|  |  |  |  |
| --- | --- | --- | --- |
| **版本** | **日期** | **修订人** | **修订内容** |
| 1.0 | 2022/6/12 | 曾毅 | 初稿 |
|  |  |  |  |

目 录

[Connect系统 1](#_Toc24837)

[---代码仓库 1](#_Toc18787)

[1.0 1](#_Toc17705)

[1. 目的 5](#_Toc32488)

[2. 介绍 6](#_Toc10086)

[3. GIT代码仓库 7](#_Toc22001)

[3.1 代码仓库 7](#_Toc5803)

[3.2 版本控制 7](#_Toc25111)

[4. Gerrit系统 9](#_Toc16119)

[4.1 介绍 9](#_Toc29466)

[4.2 仓库托管和权限管理 9](#_Toc7641)

[4.3 代码提交检查 12](#_Toc4567)

[4.4 持续集成 12](#_Toc15387)

[4.5 自动化测试 12](#_Toc16843)

[5. Gerrit实践：普通用户 13](#_Toc7606)

[5.1 配置 13](#_Toc5039)

[5.1.1 邮箱地址 13](#_Toc25221)

[5.1.2 HTTP Credentials 14](#_Toc14836)

[5.1.3 SSH keys 15](#_Toc9540)

[5.2 提交流程 18](#_Toc5675)

[5.2.1 提交流程 19](#_Toc21476)

[5.2.2 克隆仓库 19](#_Toc11929)

[5.2.3 本地调试和提交 20](#_Toc21318)

[5.2.4 提交到远程仓库 20](#_Toc13708)

[5.2.5 审核 21](#_Toc4550)

[5.2.6 审核通过提交 22](#_Toc22567)

[5.3 驳回后再次提交 23](#_Toc1866)

[5.3.1 提交一个修改 23](#_Toc30910)

[5.3.2 审核并驳回 24](#_Toc22198)

[5.3.3 修改后再次提交 25](#_Toc22788)

[5.3.4 审核并提交 27](#_Toc9191)

[5.4 废弃流程 28](#_Toc6794)

[5.4.1 提交一个修改 28](#_Toc2728)

[5.4.2 废弃 29](#_Toc28777)

[5.4.3 回退 29](#_Toc17988)

[5.5 切换分支并提交 31](#_Toc29453)

[5.5.1 克隆仓库并切换分支 31](#_Toc22168)

[5.5.2 本地调试和提交分支 31](#_Toc20340)

[5.5.3 提交到远程仓库分支 32](#_Toc5849)

[5.6 Git常用操作 33](#_Toc5481)

# 目的

本文档用于介绍Connect系统代码仓库使用方法。

# 介绍

为了达成研发部门的目标，软件部门需要控制代码质量，一般来说要控制代码质量需要进行软件的版本控制和代码提交审核。

软件的版本控制在总体设计中有详细描述，而本文档通过GIT工具对总体设计的方案进行实施，另外本公司通过Gerrit系统维护公司代码仓库并进行代码提交审核，在后续基于Gerrit系统会集成jenkins进行持续进程和自动化测试。

# GIT代码仓库

根据《总体设计》目前交换机有三个代码仓库。

## 代码仓库

#### Connect仓库

Connect仓库主要存放我司业务代码和编译平台，业务代码按组件包的形式进行管理，所有组件包基于CPPF编译平台引导编译。

#### Netforward-sdk-6.1.7仓库

此仓库主要存放楠菲微的SDK代码，目前楠菲微提供了一套编译平台可以一次性编译出bootloader，kernel和sdk库，我司在此基础上编写了一套很轻的打包脚本，用于将编译生成的各种目标文件按照不同的作用进行提取和打包。

Netforward仓库一般情况下不用编译，项目主管会将编译好的boot，kernel和sdk lib提交到对应的connect分支中。

如果需要调试kernel和sdk中的功能则需要单独编译。

#### Switch仓库

Switch仓库比较杂，由于目前文件服务器还未搭建，我们也会把资料上传一份进行备份。

## 版本控制

#### 分支目录

根据《总体设计》介绍，一个组件包有以下代码分支，



这些分支的含义为，

Trunk目录为组件包的主干版本，主干版本不允许研发代码直接提交。

Develop目录为项目开发目录，当一个项目开始时从trunk目录拉出develop目录进行开发，开发过程中的代码都提交到develop目录中。

Branch目录为特别分支目录，当功能出现较大分歧时需要通过trunk开出branch分支进行单独演进。

Maintain目录为项目维护目录，当研发部门在develop目录中完成单元测试和集成测试并通过了测试部门的准入测试，这时develop目录中的代码将合入trunk目录，然后从trunk目录中拉出maintain目录进行系统测试，后续对BUG的修订都提交到maintain目录中。Maintain目录中的BUG修订周期性的同步回trunk主干。另外产品发布后长期的BUG修订和版本发布都在maintain目录中进行。

Tag目录主要用于标记项目节点，当系统测试通过软件版本允许发布时基于当前的maintain目录会打出相应的tag目录。

Custom目录主要用于客户定制的临时版本。

#### GIT分支管理

我司采用GIT进行版本控制，当对应组件的GIT仓库建立时master分支就是Trunk主干版本，默认情况下大家都不能提交代码到trunk（master）中，只有当对应项目进入内测时才将develop分支的代码同步到trunk分支。

当一个项目开始，项目经理根据项目分析需要的组件并选择对应的版本，给出组件和版本列表，根据列表对应的组件开出develop分支，develop分支总是以”develop”单词开头，加上项目名字或者项目编号组成，比如”develop-connect-1.0”，当项目进入维护阶段对应的分支将进行回收。

如果需要建立Branch分支，Branch分支的命名规则为”branch”+”特定说明词汇”，比如”branch-arch-2.0”。

Maintain分支的命名规则为，”maintain”+”项目名字或编号”，maintain分支在项目开发过程中有一个develop分支与之对应，比如当fiber-1.0项目进入内测后，对应组件包有一个develop分支：develop-connect-1.0和一个maintain分支：maintain-connect-1.0，但是在这时develop分支已经不再使用，BUG的修订和代码的改动都应该在maintain分支上进行。

Tag分支根据Gerrit系统进行标记，Tag分支的命名规则为”tag”+”项目名字或编号”+日期。

Custom分支的命名规则为”custom”+”特定说明词汇”，比如”custom-tcl-switch”。

#### 分支销毁

仓库中有几种类型的分支在恰当的情况下需要被销毁。

#### Develop分支

当项目内测结束发布正式版本后，接下来将进入漫长的维护期，整个维护期的工作都在maintain分支上进行，这时develop分支需要申请销毁。

#### Branch分支

Branch分支是为特定研发用的分支，当研究完成合入主干或者证明相关研究不再需要时，需要申请销毁对应分支。

#### Custom

Custom分支主要为某个客户临时定制，长远来看客户的功能应该能够被评估合入主干。Custom分支存在的时间不能太长，应该在接下来的项目中合入主干。

# Gerrit系统

## 介绍

我们使用Gerrit系统有三个目的，第一，用于代码仓库的托管和权限管理；第二，用于代码提交检查；第三，用于持续集成和自动化测试。

第一点和第二点是当前我们必须要做到的，第三点在后续逐步增加。

## 仓库托管和权限管理

由于我们使用GIT工作管理代码，那么需要一个服务器托管GIT代码仓库。现有的GIT代码托管平台比较多，有gitosis，github，gerrit等，我们目前计划用gerrit作为仓库托管，主要考虑gerrit相对比较成熟而且免费，后续还可以用于持续集成。

#### 仓库建立和维护

#### 仓库建立和销毁

当需要建立一个新的代码仓库时，高级工程师根据本文档第三章节的约束对仓库进行命名并组织讨论建立仓库的目的和必要性，讨论以后如果建立仓库的请求被接受，将需求提交给Gerrit管理员，管理员根据提交的请求建立Gerrit仓库并进行人员配置。

注意，仓库建立申请必须用正式邮件。

所有代码仓库一旦建立不做销毁，如果确定已经废弃不用即可。

#### 仓库维护

仓库建立以后会指定仓库的Owner，Owner对仓库的日常管理负责。

#### 分支建立和分支销毁

一般来说高级工程师和项目经理都有可能申请代码分支，高级工程师作为某些仓库的负责人，当进行新特性研究或者其他目的时建立新分支，而项目经理在立项时评估代码仓库在对应项目中是否需要开出分支，如有则提交申请建立分支。

根据3.2章节仓库版本控制介绍，在分支需要被销毁的情况下销毁分支。

#### 权限管理

仓库创建以后工程师要能够使用仓库还需要创建自己的账号并且申请相应的权限。

#### 账号管理

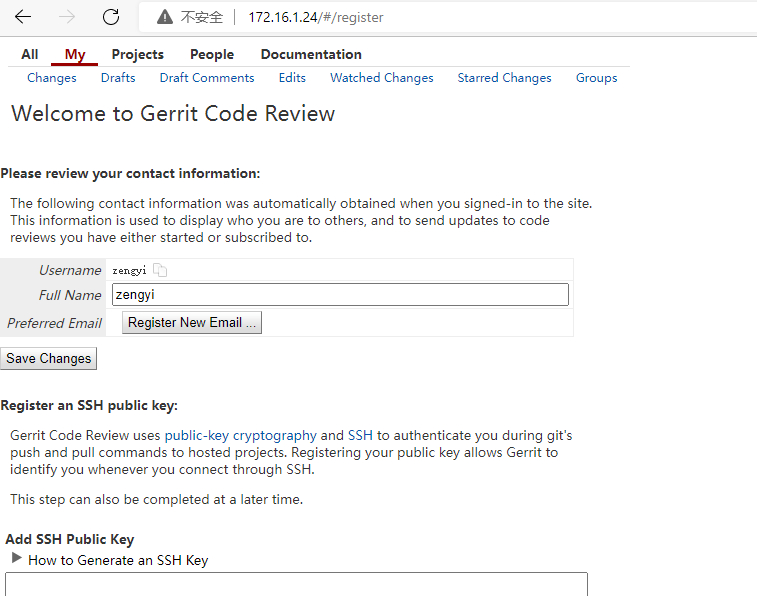
Gerrit账号需要向Gerrit管理员申请，账号密码和员工入职的邮箱名字相同（注：是@前面部分，zengyi@micronet.ltd）。

#### 账号和权限

一般来说Gerrit管理员会根据工程师的职位直接将其加入某个工作组，如果工程师需要其他的代码仓库权限，需要直接向仓库owner申请将其加入。

#### 代码拉取

账号建立完成以后，工程师通过浏览器输入“http://172.16.1.24/”登陆Gerrit系统，如果在成都办公室则需要添加静态路由“route add -p 172.16.1.0 MASK 255.255.255.0 172.31.4.44”然后才能访问，

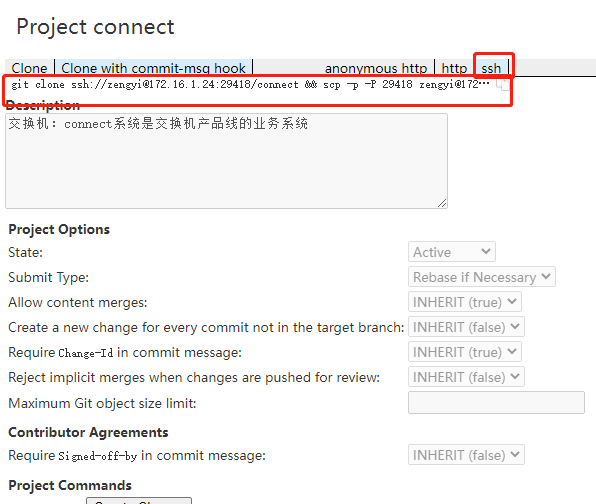


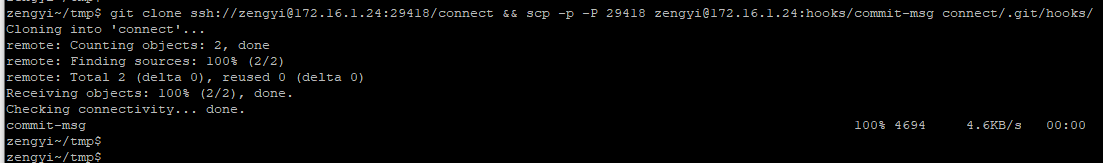
通过点击仓库部分，可以看到工程师可以看到的仓库，



点击某个仓库进入，仓库的路径要用红线框中的路径，因为提交代码时需要根据相关hook创建commit ID，



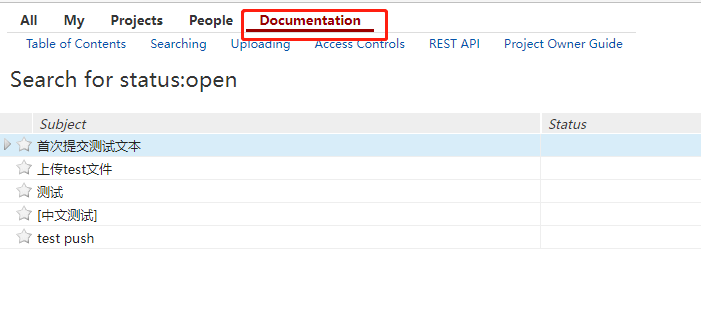




关于SSH KEY在下一章描述。

#### 其他学习资料

Documentation中有关于Gerrit的学习资料，作为工程师需要进行简单了解，



## 代码提交检查

代码检查是使用Gerrit的主要原因，在Gerrit系统中我们主要分为两个角色，一个是代码提交者，另外一个是代码审核者。

工程师和高级工程师都是代码提交者，而只有高级工程师才能担任代码审核者。

## 持续集成

//TBD

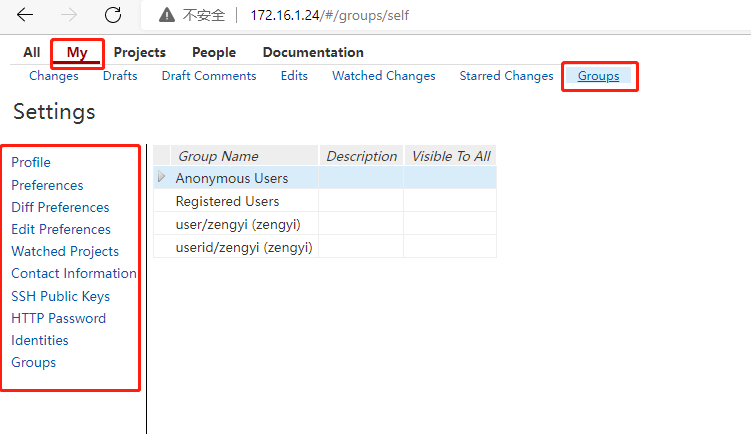
## 自动化测试

//TBD

# Gerrit实践：普通用户

## 配置

Gerrit用户在如下红框中进行个人信息配置，

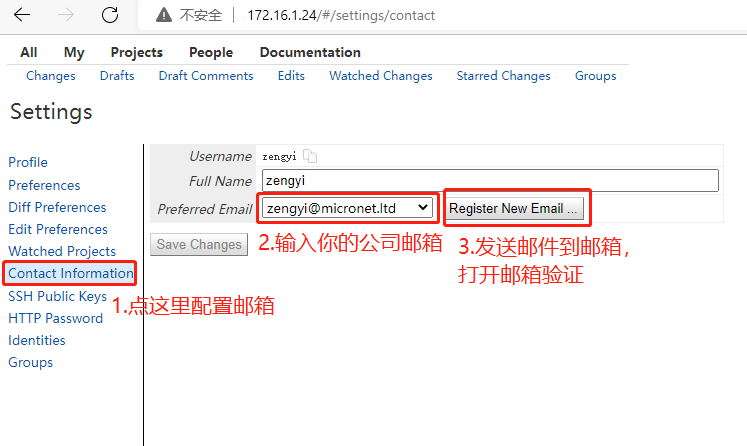


个人信息比较多，细节需要由大家自行了解，这里主要介绍工作中必须要用到的配置，

### 邮箱地址

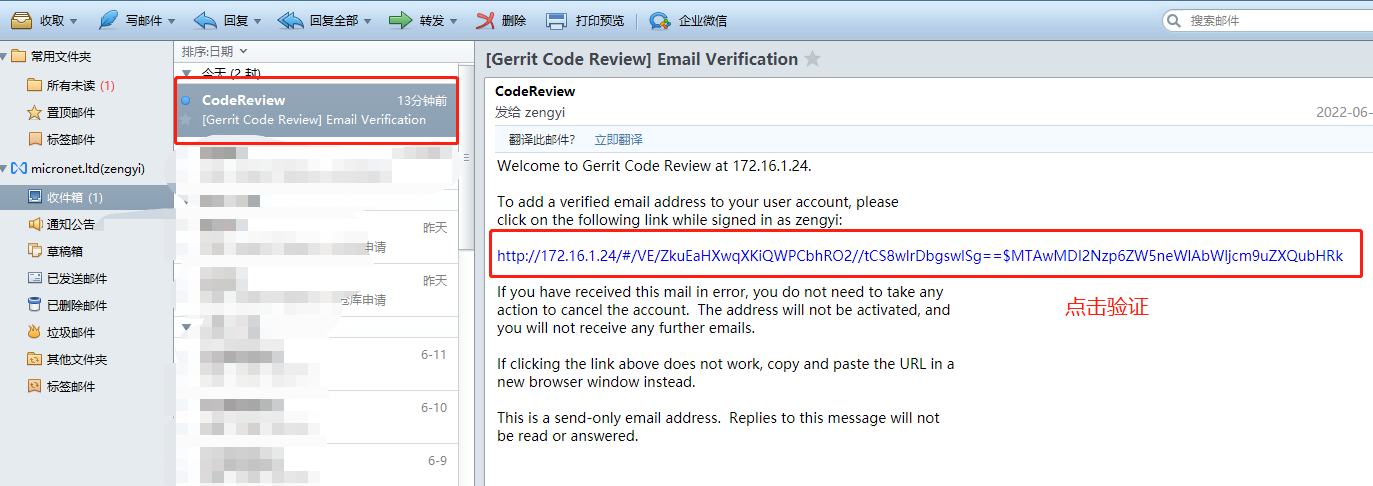
每个人的邮箱地址在入职时已经创建，邮箱地址在Gerrit系统中必须认证，否则无法提交代码。邮箱配置需要先填写邮箱地址，再认证。

首先填写自己在公司的邮箱地址，



如果发送失败，联系管理员（或者你的上级）。

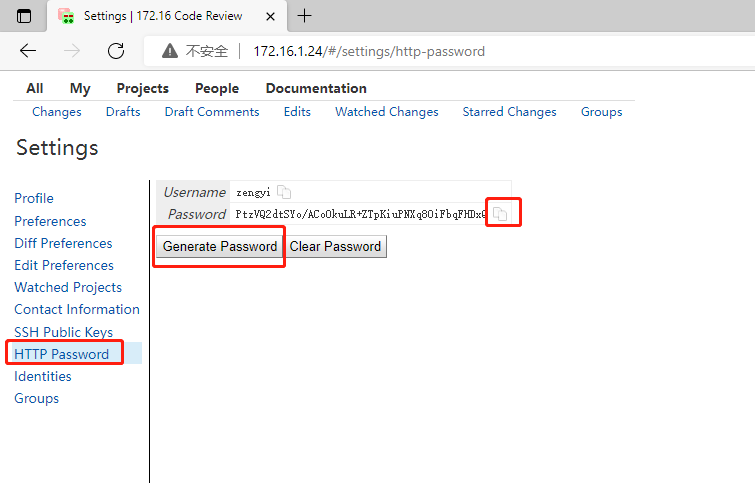
邮件发送成功后，打开邮箱，



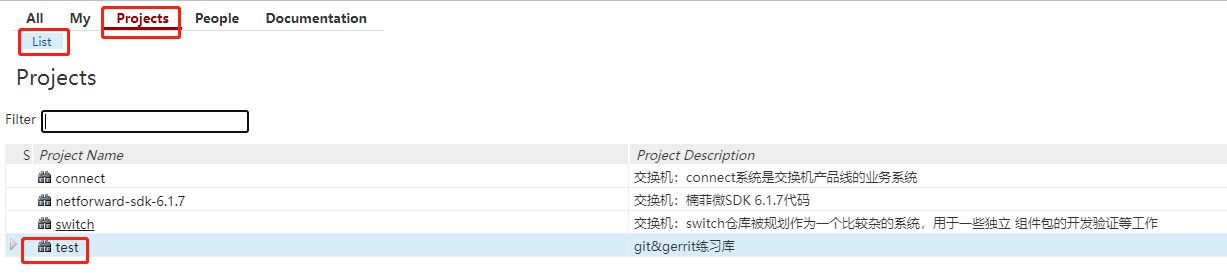
### HTTP Credentials

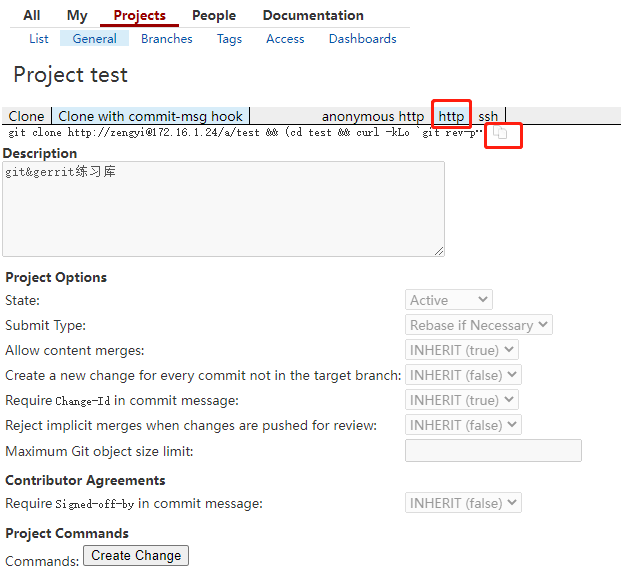
HTTP Credentials用于使用HTTP访问git仓库时输入密码。

第一步，产生一个新的密码，



第二步，选择仓库和链接，





第三步，访问仓库并输入产生的新密码，



### SSH keys

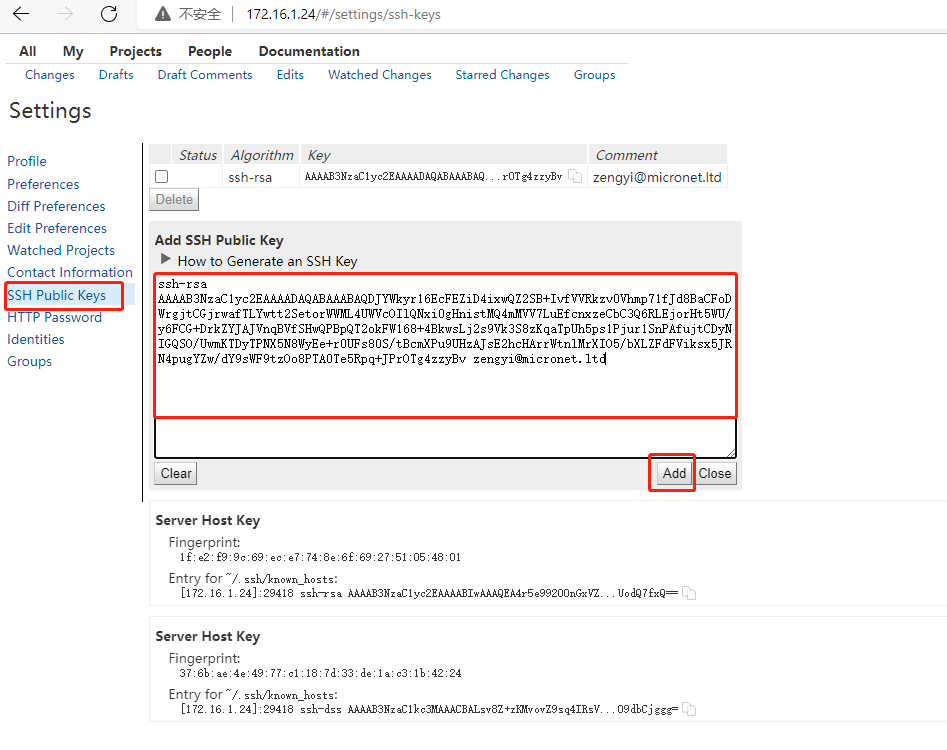
相比使用HTTP，更常用的是使用SSH访问git仓库。使用SSH访问仓库首先需要根据自身的信息产生keys，并把public keys放到服务器中。

第一步，根据自身信息产生密钥，

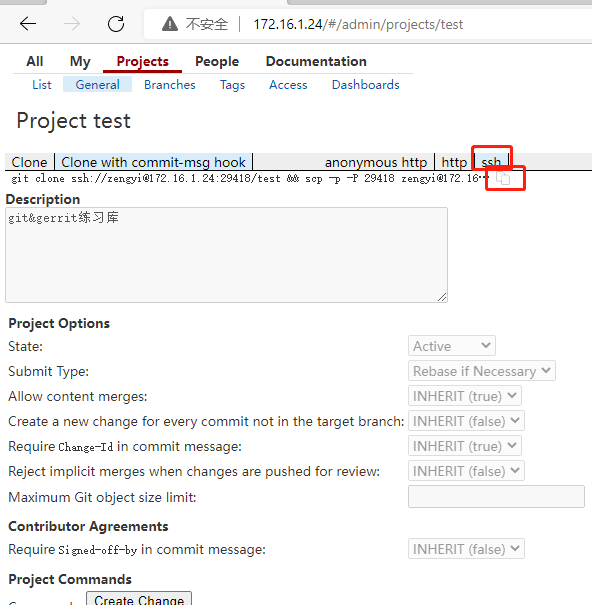
|  |
| --- |
| [zengyi]$ssh-keygen -C "zengyi@micronet.ltd"  Generating public/private rsa key pair.  Enter file in which to save the key (/home/zengyi/.ssh/id\_rsa):  Created directory '/home/zengyi/.ssh'.  Enter passphrase (empty for no passphrase):  Enter same passphrase again:  Your identification has been saved in /home/zengyi/.ssh/id\_rsa  Your public key has been saved in /home/zengyi/.ssh/id\_rsa.pub  The key fingerprint is:  SHA256:69HpshFFOWPbF5xHvaZLmCvgrEEX+0QS1u2c575S9r8 zengyi@fibercentury.com  The key's randomart image is:  +---[RSA 3072]----+  | .. .+ |  | ..\* + o|  | o o.\* . o.|  | . ..+ o .o |  | .S. +ooo |  | oo+ +o= |  | .o c.o =.o |  | . .\*o= o.. . |  | oo ++o .o.E+|  +----[SHA256]-----+  [zengyi]$ |

第二步，将公钥拷贝到服务器，

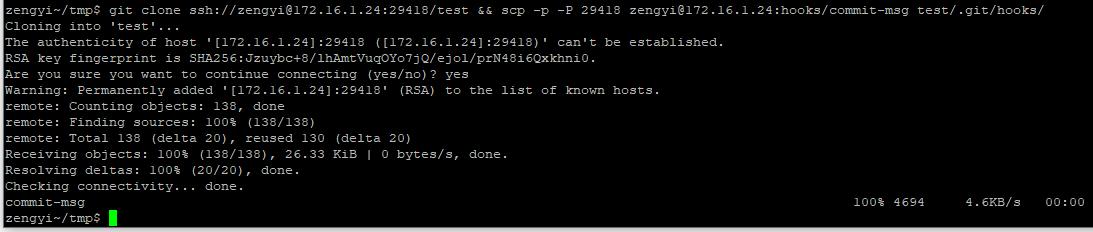
|  |
| --- |
| [zengyi]$cd ~/.ssh  [zengyi]$cat ./id\_rsa.pub  ssh-rsa  zengyi@micronet.com  [zengyi]$ |



第三步，选择仓库和链接，



第四步，根据链接访问仓库，



因为私钥在本地ls ~/.ssh中，所以不需要输入密码，比较方便。

如果希望一个账号的公钥和密码在多台电脑上使用，那么需要把对用的公钥和私钥都拷贝到相关电脑。

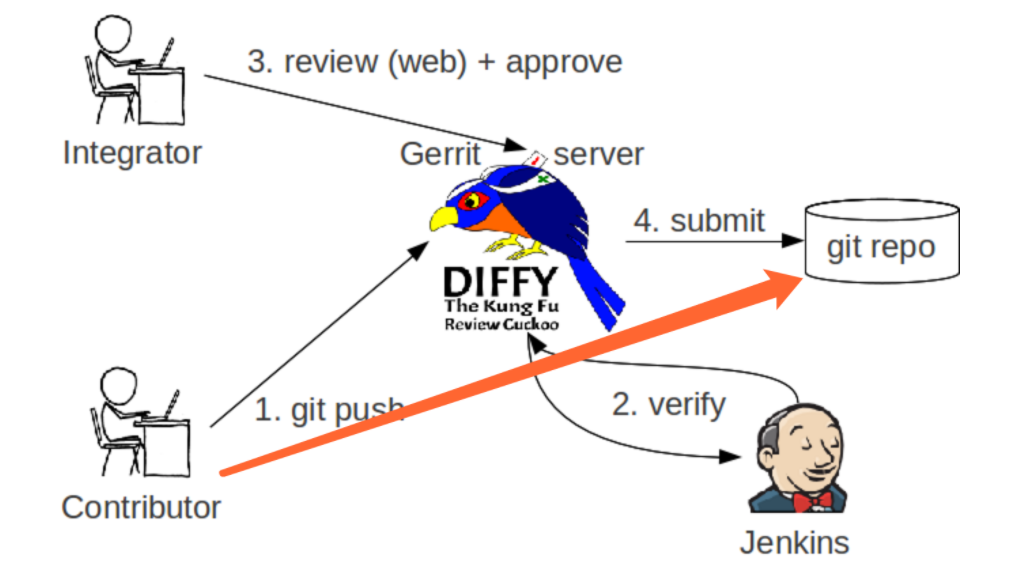
## 提交流程

我们知道Git是一个分布式仓库，在Git的通常流程中工程师首先在本地完成本地仓库的提交，典型操作为，

|  |
| --- |
| git add ./\*  git commit –m “notes” |

当完成本地仓库提交以后，再通过”git push”将本地修改提交到公司代码仓库。

而Gerrit流程与Git的通常流程差不多，只是在将本地修改提交到公司代码仓库中时，多了一些步骤。下图介绍了Gerrit的提交流程，其中Contributor就是代码修改者负责代码提交，而Integrator就是代码审阅者在一些公司专门设有负责代码集成的岗位。



工程师首先把代码提交到本地，然后开始Gerrit提交流程，

1. 工程师(Contributor)首先把代码push到Gerrit服务器的代码评审分支（本章后续有介绍）中；
2. Gerrit探测到分支有代码提交触发持续集成验证，在图中Gerrit触发Jenkins进行集成验证，Jenkins将验证的结果反馈给Gerrit，如果验证失败，工程师需要修订代码重新提交。Jenkins是一个独立的持续集成工具，Gerrit是代码托管和代码评审工具，两个工具是完全独立的。考虑到平台搭建工作量，前期我们第二步省略；
3. 具有代码审核权限的工程师(Integrator)在Contributor push代码以后会收到代码review提示，根据提示工程师审核代码并给其打分，如果审核不通过结果将反馈给提交工程师，提交工程师根据反馈重新修改代码然后再次提交，如果审核通过将进入下一个流程；
4. 当Integrator审核代码通过并且Jenkins验证结果通过，Gerrit服务器自动把这次提交的内容从评审分支（内部不可见）提交到正式仓库，这时其他账号通过pull可以获取修改以后的内容。

接下来将通过一个完整的仓库用例介绍常用的操作。

我们将使用switch仓库作为介绍，在下面的例子中假定zengyi既有代码审核权限又有代码提交权限。

### 提交流程

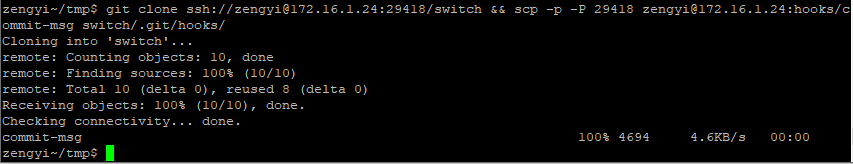
假设zengyi（提交者）需要创建一个个人目录并在目录中创建自己的任务清单。

### 克隆仓库

首先zengyi登陆Gerrit系统，找到switch仓库并用对应链接将仓库克隆到本地，





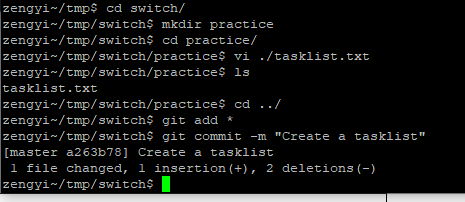


### 本地调试和提交

第一次使用git记得配置用户名和邮箱地址，



本地提交，

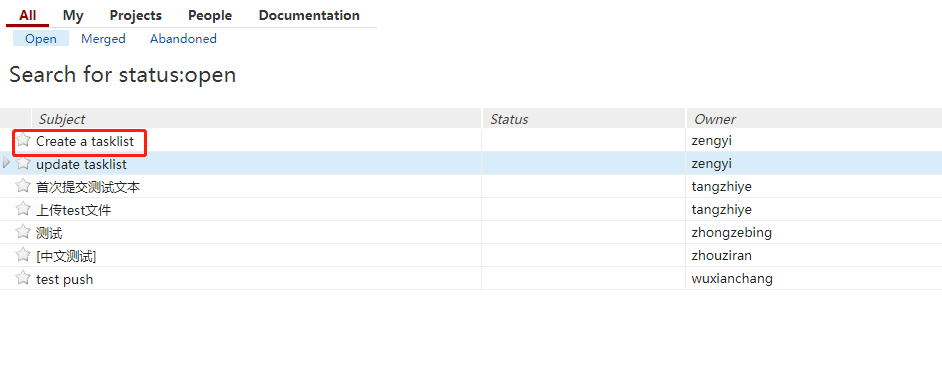


### 提交到远程仓库

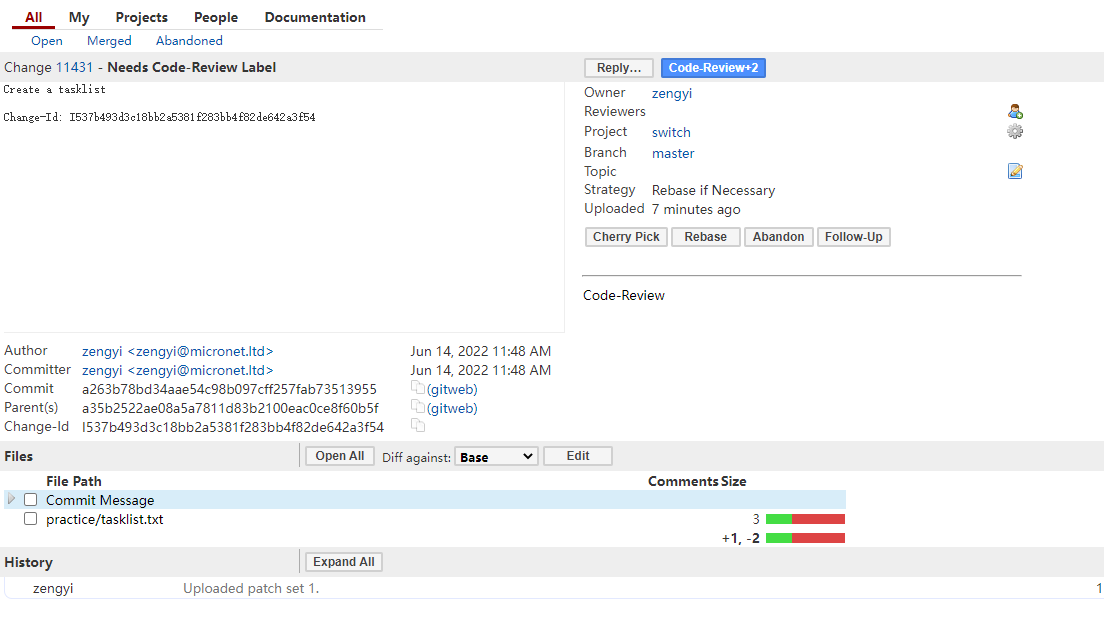
首先PUSH到远程仓库，

|  |
| --- |
| zengyi~/tmp/switch$ git push origin HEAD:refs/for/master  Counting objects: 4, done.  Writing objects: 100% (4/4), 342 bytes | 0 bytes/s, done.  Total 4 (delta 0), reused 0 (delta 0)  remote: Processing changes: new: 1, refs: 1, done  remote:  remote: New Changes:  remote: http://172.16.1.24/11431 Create a tasklist  remote:  To ssh://zengyi@172.16.1.24:29418/switch  \* [new branch] HEAD -> refs/for/master  zengyi~/tmp/switch$ |

然后登陆Gerrit将看到自己的PUSH记录，

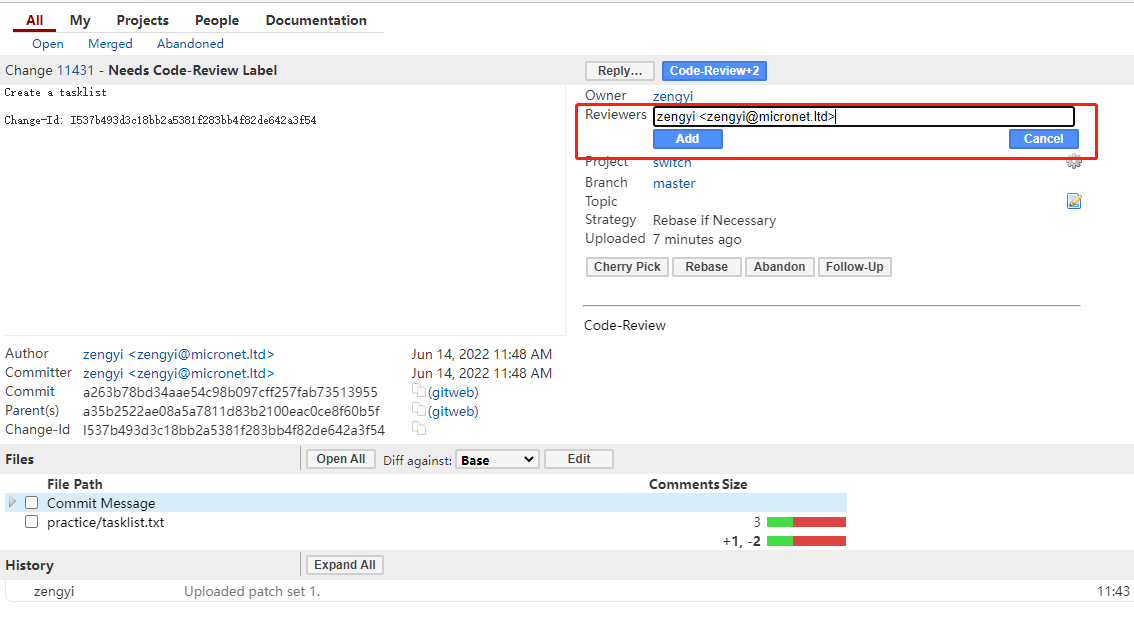


点击进入将看到本次提交的详细信息，

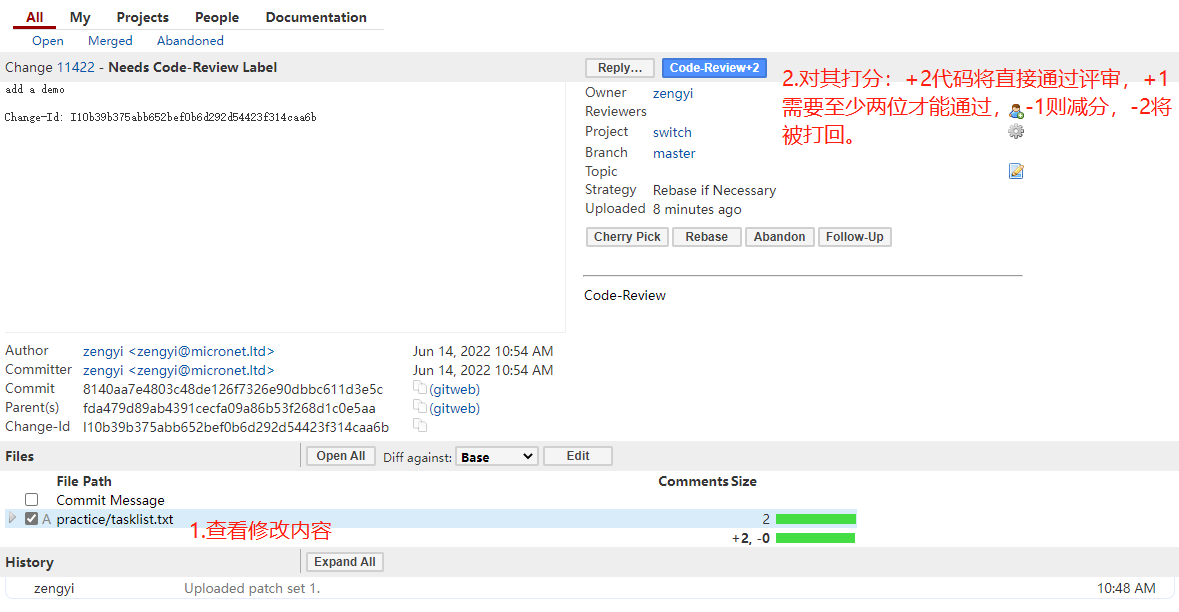


### 审核

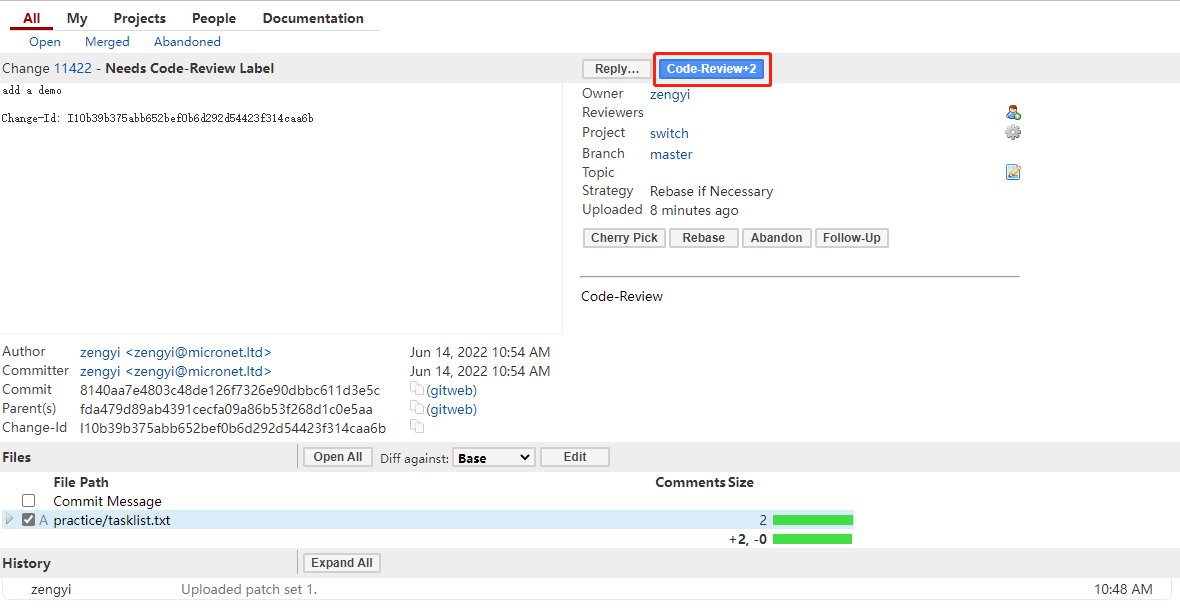
添加reviewer，



zengyi（审阅者）点击进入并对其打分，

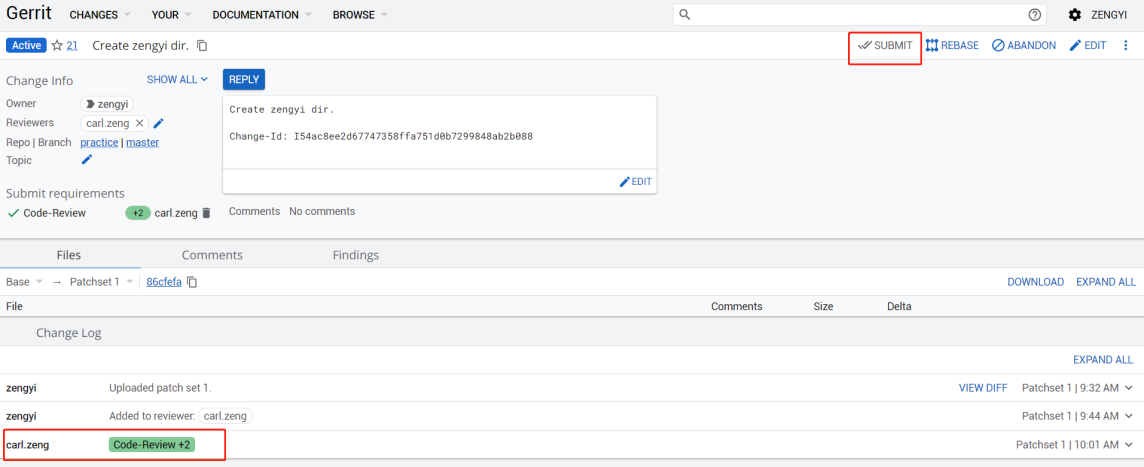


假设本次评审直接打+2分，即通过，

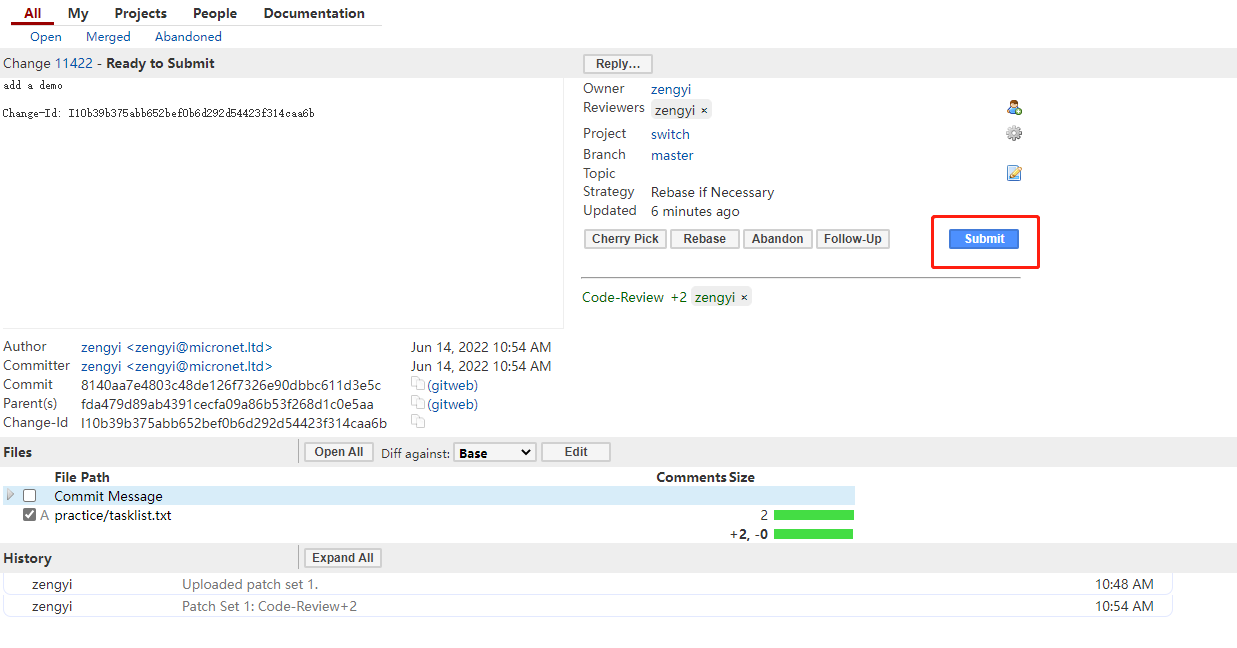


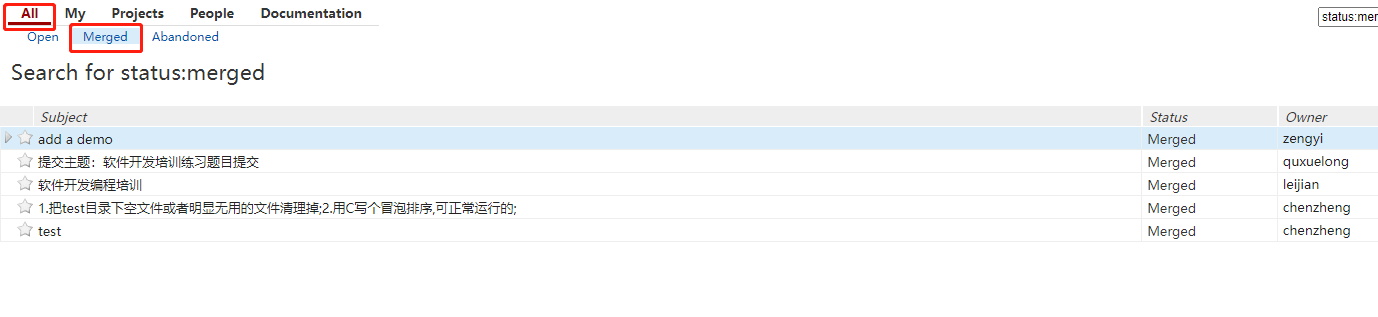
### 审核通过提交

审核通过后，在zengyi（提交者） 的账号中也可以看到评审状态，



由zengyi（提交者）点击submit将本次提交合到master分支中，





至此提交完成，其他账号可以取到最新的仓库信息。

## 驳回后再次提交

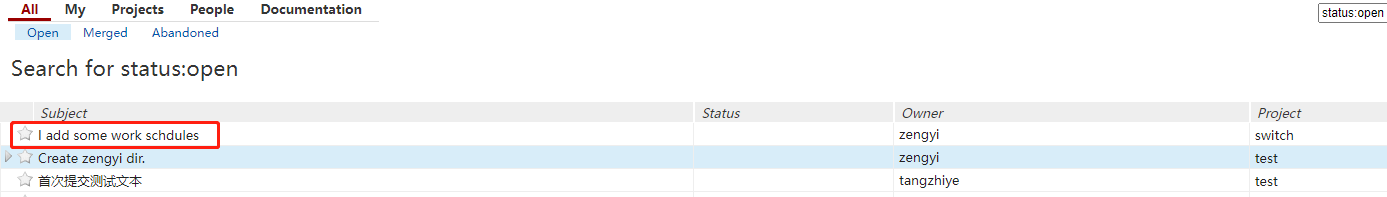
另外一种比较常见的情况是代码提交后被打回，被打回的代码经过修改重新提交，这时应该接着上一次的提交继续。

### 提交一个修改

增加zengyi（提交者）目录中tasklist中的内容并提交到远程仓库，

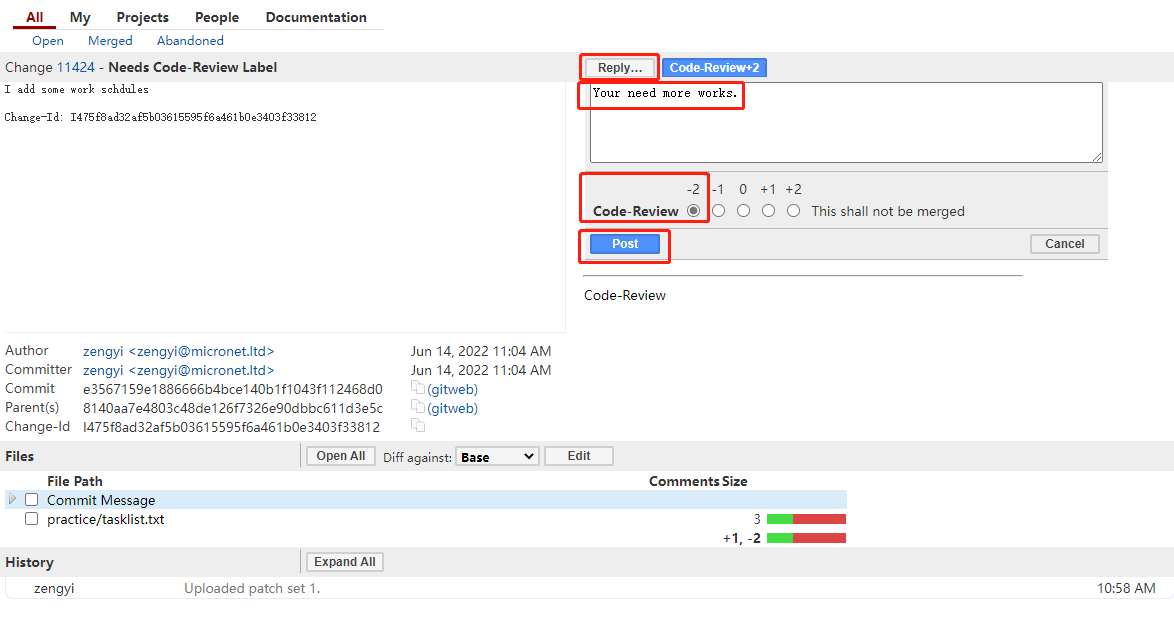
|  |
| --- |
| zengyi~/tmp/switch$ ls  practice  zengyi~/tmp/switch$ cd practice/  zengyi~/tmp/switch/practice$ ls  tasklist.txt  zengyi~/tmp/switch/practice$ vi ./tasklist.txt  zengyi~/tmp/switch/practice$ ls  tasklist.txt  zengyi~/tmp/switch/practice$ cd ..  zengyi~/tmp/switch$ ls  practice  zengyi~/tmp/switch$ git status  On branch master  Your branch is ahead of 'origin/master' by 1 commit.  (use "git push" to publish your local commits)  Changes not staged for commit:  (use "git add <file>..." to update what will be committed)  (use "git checkout -- <file>..." to discard changes in working directory)  modified: practice/tasklist.txt  no changes added to commit (use "git add" and/or "git commit -a")  zengyi~/tmp/switch$ git add -u  zengyi~/tmp/switch$ git commit -m "I add some work schdules"  [master e356715] I add some work schdules  1 file changed, 1 insertion(+), 2 deletions(-)  zengyi~/tmp/switch$ git push origin HEAD:refs/for/master  Counting objects: 4, done.  Writing objects: 100% (4/4), 356 bytes | 0 bytes/s, done.  Total 4 (delta 0), reused 0 (delta 0)  remote: Processing changes: new: 1, refs: 1, done  remote:  remote: New Changes:  remote: http://172.16.1.24/11424 I add some work schdules  remote:  To ssh://zengyi@172.16.1.24:29418/switch  \* [new branch] HEAD -> refs/for/master  zengyi~/tmp/switch$ |

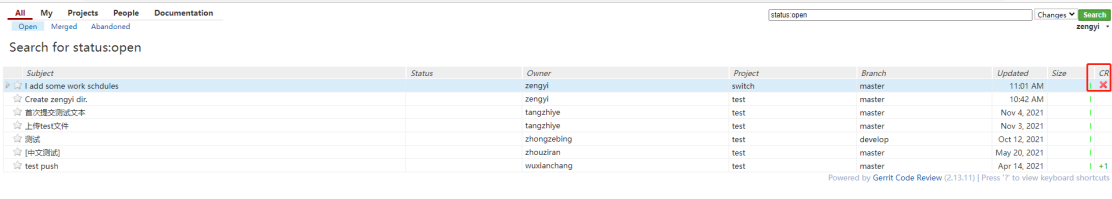
Gerrit已经看到这个改动，



### 审核并驳回

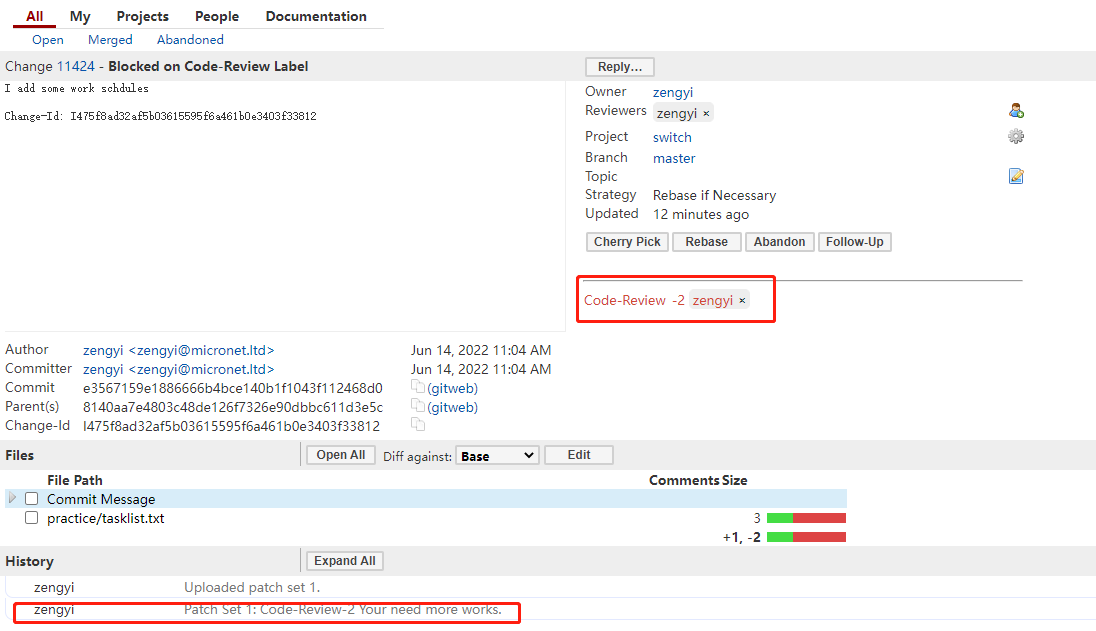
Zengyi（审阅者）看到表单进行审核并驳回，





### 修改后再次提交

Zengyi（提交者）看到了审阅结果，



修改并本地commit，注意这次commit使用—amend参数，

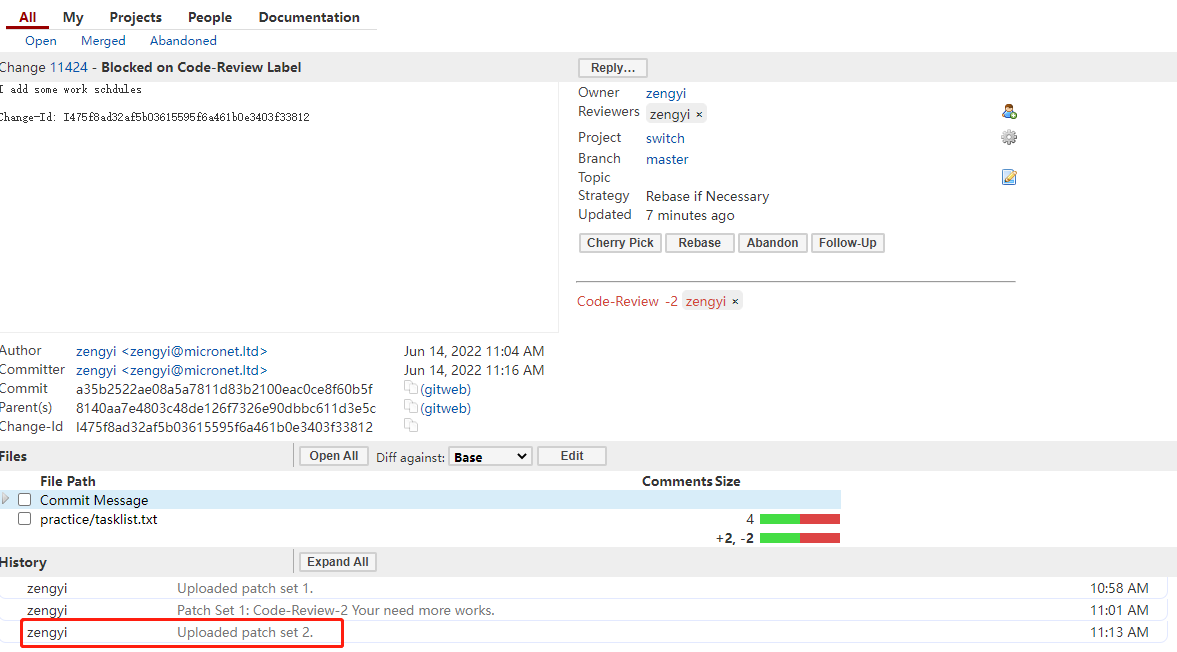
|  |
| --- |
| zengyi~/tmp/switch$ echo "I work to 12:00" >> ./practice/tasklist.txt  zengyi~/tmp/switch$ git add \*  zengyi~/tmp/switch$ git commit -amend --no-edit  error: did you mean `--amend` (with two dashes ?)  zengyi~/tmp/switch$ git commit --amend --no-edit  [master a35b252] I add some work schdules  Date: Tue Jun 14 11:04:21 2022 +0800  1 file changed, 2 insertions(+), 2 deletions(-)  zengyi~/tmp/switch$  zengyi~/tmp/switch$ git log  commit a35b2522ae08a5a7811d83b2100eac0ce8f60b5f  Author: zengyi <zengyi@micronet.ltd>  Date: Tue Jun 14 11:04:21 2022 +0800  I add some work schdules  Change-Id: I475f8ad32af5b03615595f6a461b0e3403f33812  … //more logs ignore.  zengyi~/tmp/switch$ |

可以通过git log看到使用了amend，CHANG-ID还是上一次的。接着重新push到远端仓库，

|  |
| --- |
| zengyi~/tmp/switch$ git push origin HEAD:refs/for/master  Counting objects: 4, done.  Writing objects: 100% (4/4), 374 bytes | 0 bytes/s, done.  Total 4 (delta 0), reused 0 (delta 0)  remote: Processing changes: updated: 1, refs: 1, done  remote:  remote: Updated Changes:  remote: http://172.16.1.24/11424 I add some work schdules  remote:  To ssh://zengyi@172.16.1.24:29418/switch  \* [new branch] HEAD -> refs/for/master  zengyi~/tmp/switch$ |

注意看，上一次是new这一次是updated。

在Gerrit上看到还是相同的表单，提交已经更新，

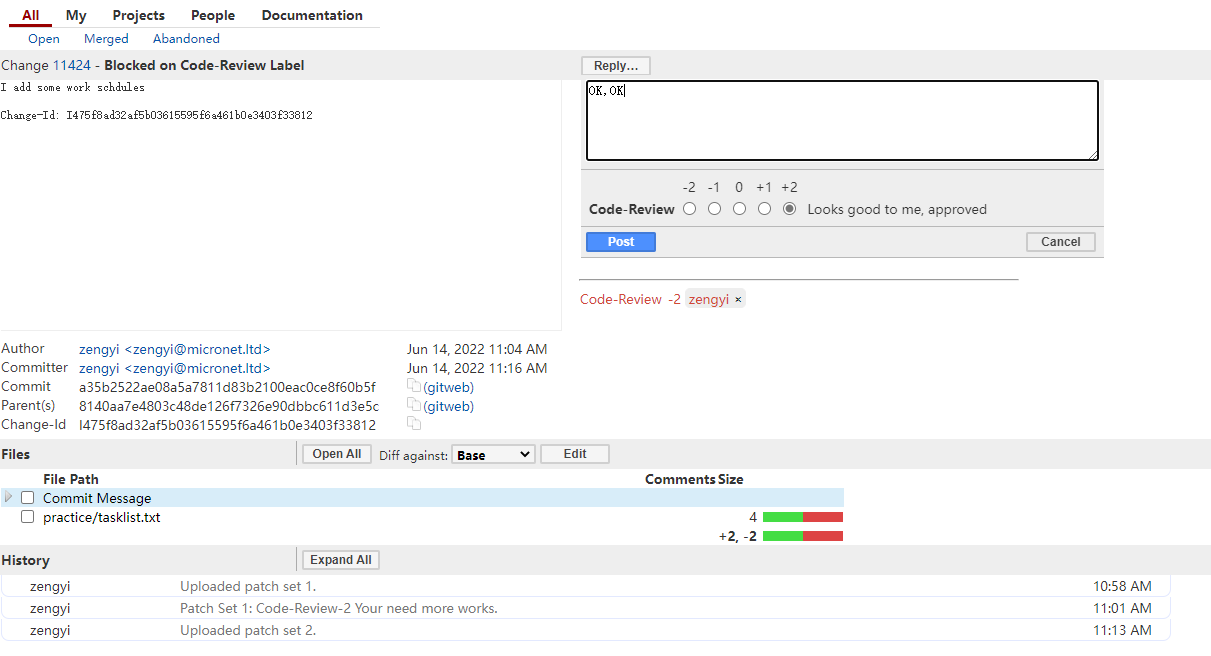


回复审阅者，

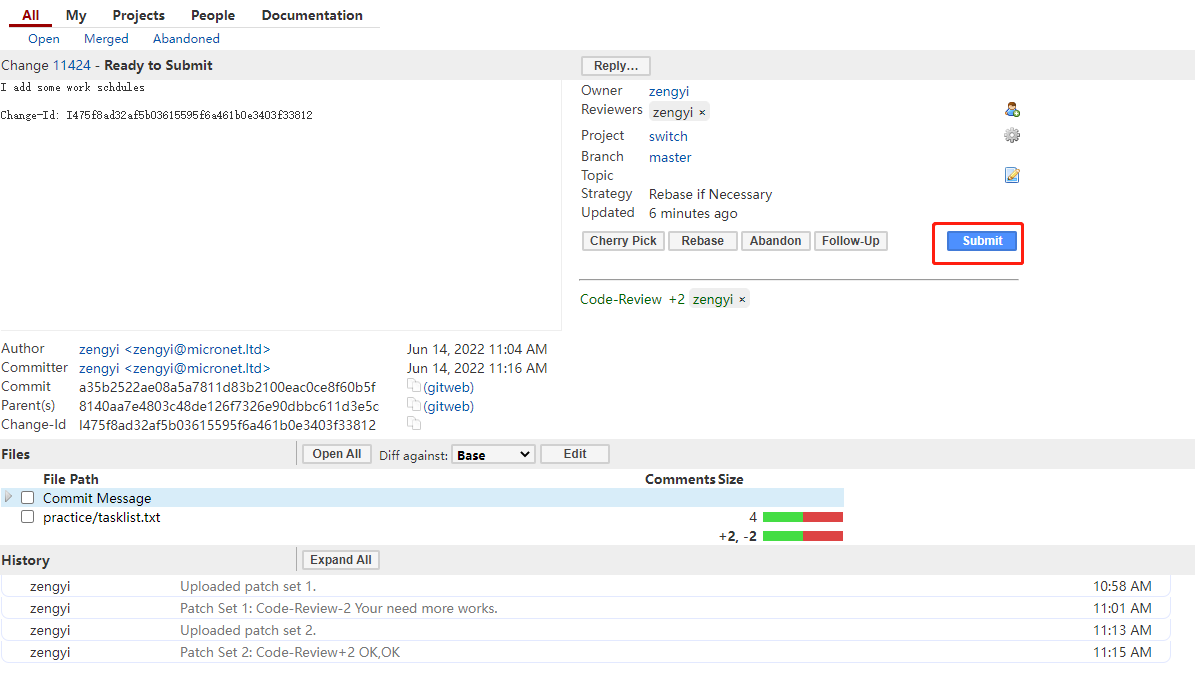
...

### 审核并提交

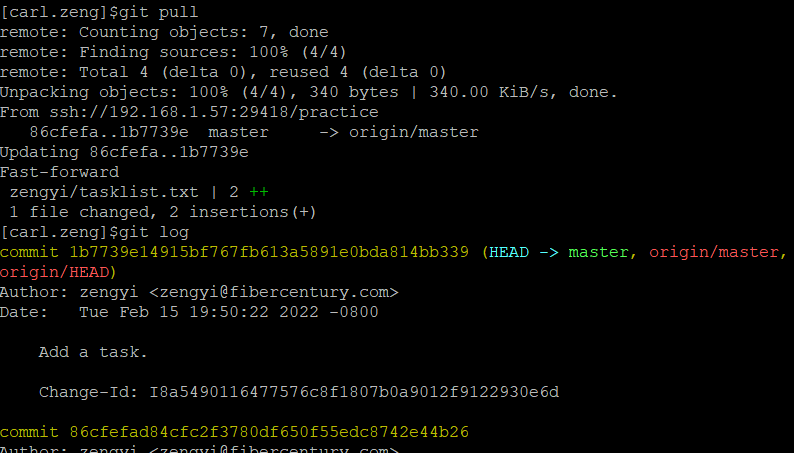
zengyi（审阅者）看到更新后的表单重新评审，



评审通过并提交，



在zengyi的本地仓库pull，可以看到已经更新，并且只有一次提交记录，



## 废弃流程

表单提交后有时因为提交错误，或者评审本次提交完全不可用，那么需要作废。

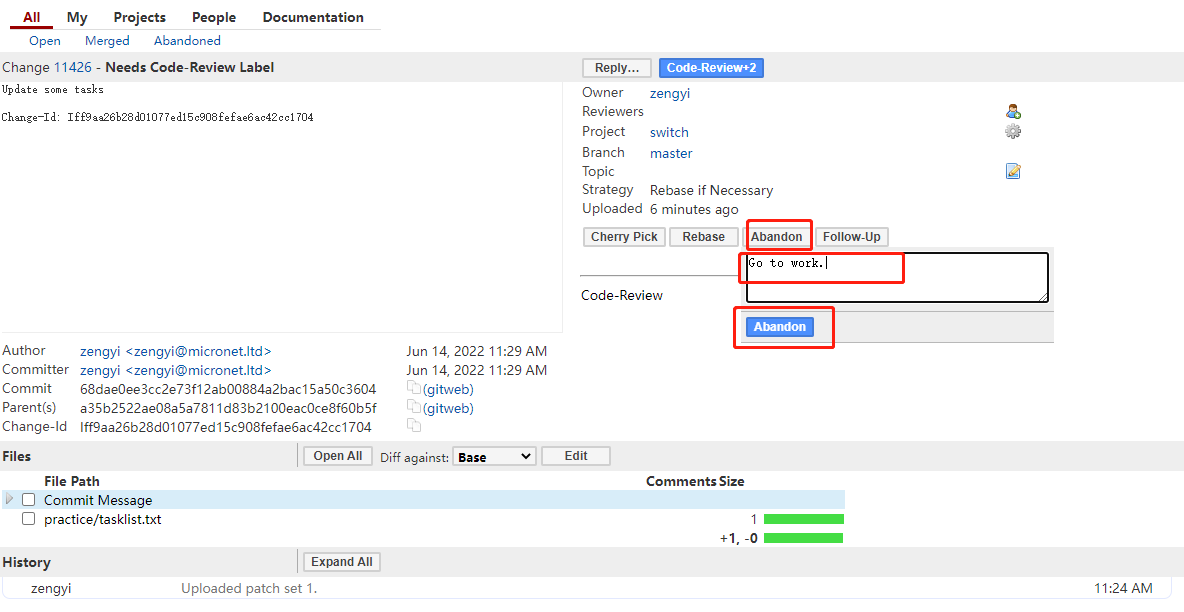
假设zengyi做了一个无用的操作，被审阅者作废，

### 提交一个修改

|  |
| --- |
| zengyi~/tmp/switch$ echo "I go to bed" >> ./practice/tasklist.txt  zengyi~/tmp/switch$ git add \*  zengyi~/tmp/switch$ git commit -m "Update some tasks"  [master 68dae0e] Update some tasks  1 file changed, 1 insertion(+)  zengyi~/tmp/switch$ git push origin HEAD:refs/for/master  Counting objects: 4, done.  Writing objects: 100% (4/4), 376 bytes | 0 bytes/s, done.  Total 4 (delta 0), reused 0 (delta 0)  remote: Processing changes: new: 1, refs: 1, done  remote:  remote: New Changes:  remote: http://172.16.1.24/11426 Update some tasks  remote:  To ssh://zengyi@172.16.1.24:29418/switch  \* [new branch] HEAD -> refs/for/master  zengyi~/tmp/switch$ |

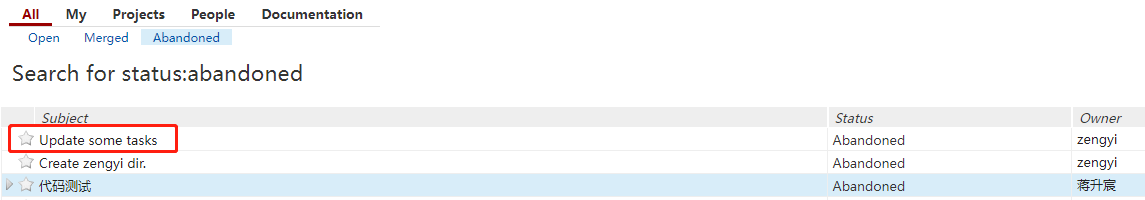
### 废弃

zengyi（审阅者）看到这个表单，直接废弃，



### 回退

Zengyi（提交者）在Gerrit中看到被废弃的表单和原因，并进行本地仓库回退，



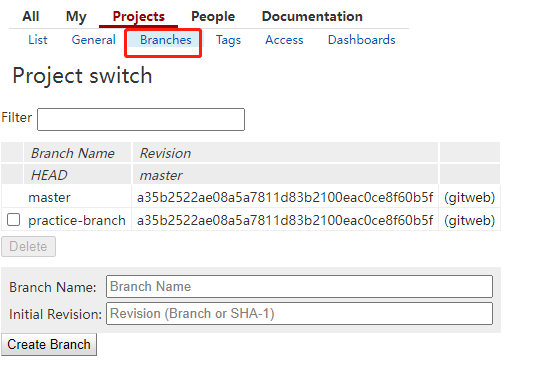
本地仓库回退，

|  |
| --- |
| zengyi~/tmp/switch$ git log  commit 68dae0ee3cc2e73f12ab00884a2bac15a50c3604  Author: zengyi <zengyi@micronet.ltd>  Date: Tue Jun 14 11:29:50 2022 +0800  Update some tasks  Change-Id: Iff9aa26b28d01077ed15c908fefae6ac42cc1704  commit a35b2522ae08a5a7811d83b2100eac0ce8f60b5f  Author: zengyi <zengyi@micronet.ltd>  Date: Tue Jun 14 11:04:21 2022 +0800  I add some work schdules  Change-Id: I475f8ad32af5b03615595f6a461b0e3403f33812  commit 8140aa7e4803c48de126f7326e90dbbc611d3e5c  Author: zengyi <zengyi@micronet.ltd>  Date: Tue Jun 14 10:54:37 2022 +0800  add a demo  Change-Id: I10b39b375abb652bef0b6d292d54423f314caa6b  commit fda479d89ab4391cecfa09a86b53f268d1c0e5aa  Author: Kang Zhiwei <kangzhiwei@micronet.ltd>  Date: Mon Jun 13 18:41:52 2022 +0800  Initial empty repository  zengyi~/tmp/switch$ git reset --hard HEAD^  HEAD is now at a35b252 I add some work schdules  zengyi~/tmp/switch$ git log  //Update some tasks 提交已经被回退  commit a35b2522ae08a5a7811d83b2100eac0ce8f60b5f  Author: zengyi <zengyi@micronet.ltd>  Date: Tue Jun 14 11:04:21 2022 +0800  I add some work schdules  Change-Id: I475f8ad32af5b03615595f6a461b0e3403f33812  commit 8140aa7e4803c48de126f7326e90dbbc611d3e5c  Author: zengyi <zengyi@micronet.ltd>  Date: Tue Jun 14 10:54:37 2022 +0800  add a demo  Change-Id: I10b39b375abb652bef0b6d292d54423f314caa6b  commit fda479d89ab4391cecfa09a86b53f268d1c0e5aa  Author: Kang Zhiwei <kangzhiwei@micronet.ltd>  Date: Mon Jun 13 18:41:52 2022 +0800  Initial empty repository  zengyi~/tmp/switch$ |

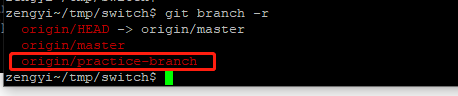
## 切换分支并提交

### 克隆仓库并切换分支

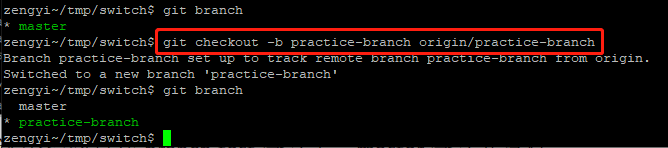
克隆仓库参考6.2.2，接着在Gerrit仓库的branch项中可以看到仓库有哪些分支，



仓库克隆到本地后，我们通过git branch –r查看远程分支，

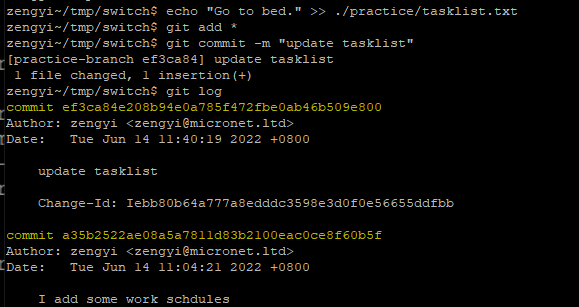


切换到此分支git checkout -b branch-test origin/branch-test

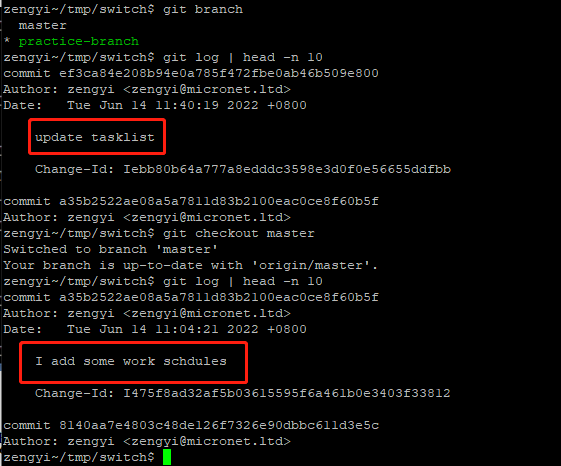


### 本地调试和提交分支

和6.2.3类似，



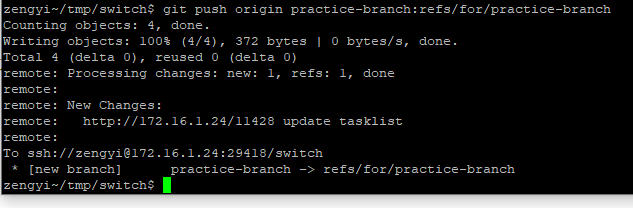
通过切换分支可以看到，这个提交只在branch-test分支上，master分支并没有，



### 提交到远程仓库分支

Push到远端分支要小心仓库的分支要给对：

git push origin practice-branch:refs/for/practice-branch



后面的动作就和6.2后续的动作一样了，主要在Gerrit上走流程。

## Git常用操作

Git的常用操作网上都有，这里不做分析，大家自行搜索学习。