Repo 数据结构脚本搭建过程

1、 gitlab 手动创建汇总清单

Gitlab 上面创建　　git@192.168.1.49:root/manifest.git

里面存放 manifest.list 文件

manifest.list 文件内容为各个工程名或者项目组的 manifest.git 的 url

格式如下：

git@192.168.1.49:example1/manifest.git

git@192.168.1.49:example2/manifest.git

git@192.168.1.49:example3/manifest.git

git@192.168.1.49:example4/manifest.git

git@192.168.1.49:example5/manifest.git

git@192.168.1.49:example6/manifest.git

其中，example1,example2,example3,example4,example5,example6 是项目组组名

2、 gitlab 手动创建项目组或者工程名的 manifest.git： 如 git@192.168.1.49:example1/manifest.git

里面存放 default.xml，repo 清单库

格式如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<manifest>
  <remote    name="origin" fetch="."/>
  <default revision="master" remote="origin" sync-j="4" />
  <project path="A" name="A.git" review="http://192.168.1.49:9999"/>
  <project path="A/b" name="b.git" review="http://192.168.1.49:9999"/>
  <project path="A/d" name="d.git" review="http://192.168.1.49:9999"    />
</manifest>
```

3、 根据 path.list 生成对应的 default.xml(path.list 由项目负责人来提供)，通过脚本生成 default.xml 并上传到清单库上

生成脚本： default.xm.sh

生成命令：

bash –x　default.xml.sh git@192.168.1.49:example1/manifest.git　gerrit gerrit@szprize.com

- 其中依次传递三个参数：
- $1　目标清单库---- git@192.168.1.49:example1/manifest.git
- $2　用户名--- gerrit　(git push 需要用到)
- $3　用户邮箱----　gerrit@szprize.com　(git push 需要用到)

4、 root 用户 ,根据清单库上 default.xml,在 gitlab 和 gerrit 数据底层仓库生成对应的子仓库,同时 gitlab web 页面和 gerrit 页面会出现对应的仓库，修改 gerrit 的配置文件 replication，形成数据从 gerrit 向 gitlab 同步关系

bash –x repo_create_gerrit.sh

5、 通过负责人用户上传初始数据到 gitlab,并在初始目录增加.review 文件(gerrit 服务器信息)和.testr.conf(jenkins 触发脚本)

执行命令:

Bash –x repo_upload.sh git@192.168.1.49:example1/manifest.git prize-bs14 prize-bs14@szprize.com

- 其中依次传递三个参数：
- $1 目标清单库---- git@192.168.1.49:example1/manifest.git
- $2 用户名--- gerrit (git push 需要用到)
- $3 用户邮箱---- gerrit@szprize.com (git push 需要用到)

6、 由于数据直接上传到 gitlab 上面后，gerrit 和 gitlab 的原始数据不一样，所以得用 root 用户，重新同步数据从 gitlab 到 gerrit 里面去。

- 先删掉 gerrit 关于其具体的项目 example 的项目
- 执行 repo_create_gerrit 脚本

  bash –x repo_create_gerrit.sh

7、 负责人从 gitlab 克隆项目，执行：

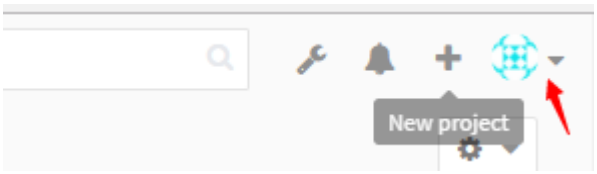
- repo init –u **git@192.168.1.49:Mt6757_base/manifest.git**
- repo sync
- 创建 master 分支, repo start master –all
- repo branch
- 增加 README,commit 形成第一个 commit 点
- 修改 README,创建 gerrit 需要的钩子 id,使用 git commit –amend
- Git review

8、 通过 gerrit 页面，审核修改部分通过，submit 并 merge，确认 gitlab 数据同步到位

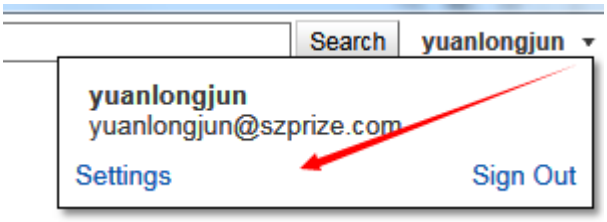
9、 开发人员进行开发

首先，增加 ssh-key

- 登录你的编译服务器
- 查看 sshkey，命令是 cat ~/.ssh/id_rsa.pub,把其内容复制。
- 打开 gitlab 网页：http://192.168.1.49,输入用户名（你的姓名拼音），密码是：123456789（默认第一次登录需要您重置密码，你可以再次输入密码：123456789 作为你的密码，确认后会要求你重新登录的），登录后，点击右上方的三角打开（如下图）的 profile settings,寻找其中上的 **SSH keys 那栏的空白处**，把上一步的复制内容复制进去。点击 add key 保存



- 同样，打开 gerrit 网页：http://192.168.1.49:9999，输入用户名（你的姓名拼音），密码是：123456，点击右上角的下三角（如下图），打开 settings，里面寻找 **SSH Public Keys** 空白处，把复制的内容复制进去，点击 ADD 保存

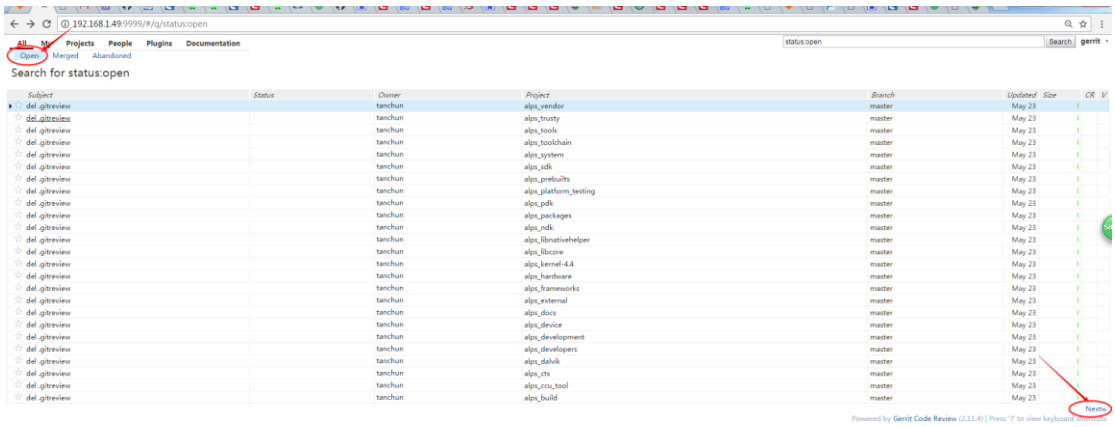


然后进行开发：

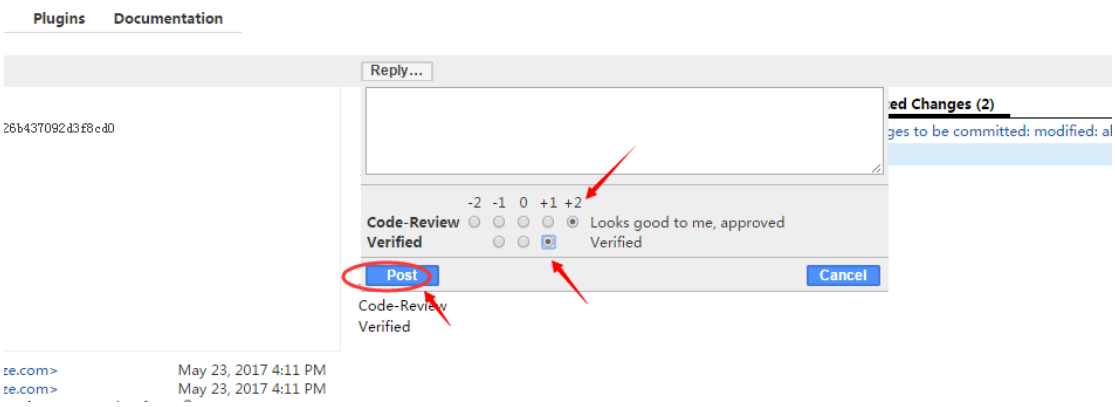
- mkdir platform_abc 新建一个文件夹，如 platform_abc(命名规则：“平台名称+”_”+”部门”三部分组成）
- cd platform_abc
- git config --global user.name name
- git cofnig --global user.email name@szprize.com
  （这两步需要提前做，不然 repo sync 时出现报错提示）
- repo init –u **git@192.168.1.49:Mt6757_base/manifest.git**
- repo sync
- repo start master –all （注意，同步下来的项目是在 no branch 上面，所以在一定要在本地新建分支 master，这步非常关键，否则会导致你所有的操作都有 no branch 上面，而无法进行代码提交或者会出现代码 commit 丢失这种情况）
- repo branch (查看是不是所有仓库都在 master 分支上)
- repo status (查看本地所有仓库是不是存在一些没有提交的点)，一般第一次时间会比较长一点，然后会在你的主目录里面，会有一些新的未提交的文件，你可以先 git add ,git commit 一下
- 接下来的操作跟平常的 git 的操作是一样，你可以去到具体的目录进开发工作，然后 git add ,git commit ,等开发工作完成后，使用 repo stauts（取代原来的 git status），来查看所有的提交记录，最后使用命令 repo upload,进行提交。
  （repo upload 执行后，它会自动弹出一个关于你的之前操作的一些 commit 清单，里面基本上带有#注释的，你可以把光标移到最左侧，使用 vim 的 ctrl + V 的块编辑，选中首字母#，按 d，把#删掉，然后保存退出）

最后进行代码审核：

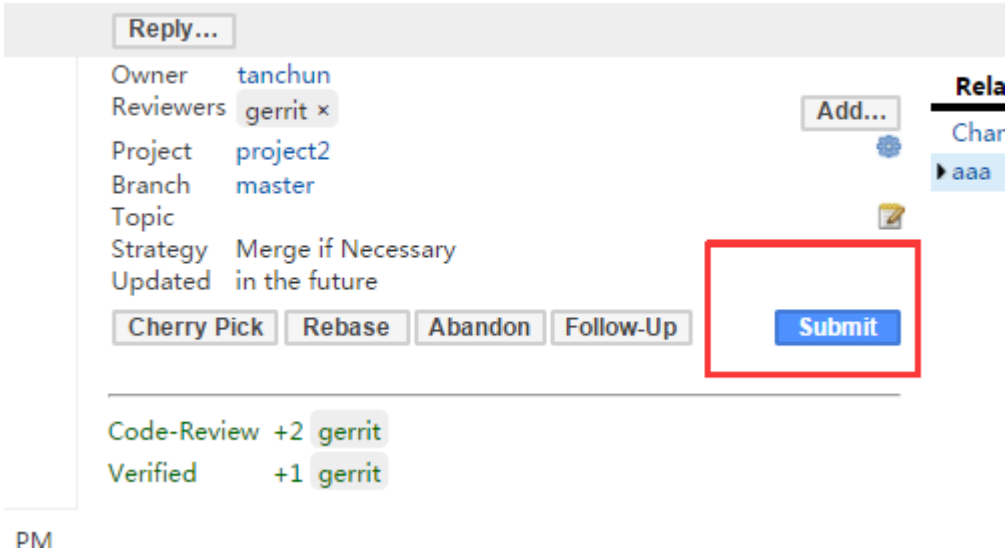
- 登录 gerrit: http://192.168.1.49:9999,点击 all 下面的 open,查看你提交的代码条目（如下图，如果仓库比较多请翻页）。



✧ 然后对代码进行审核打分，点击 post，如图



✧ 然后进行 submit 提交（注意，一般有权限的人才会出现 submit 的按钮，一般是项目的创建者，如下图）



其他一些常用的命令：

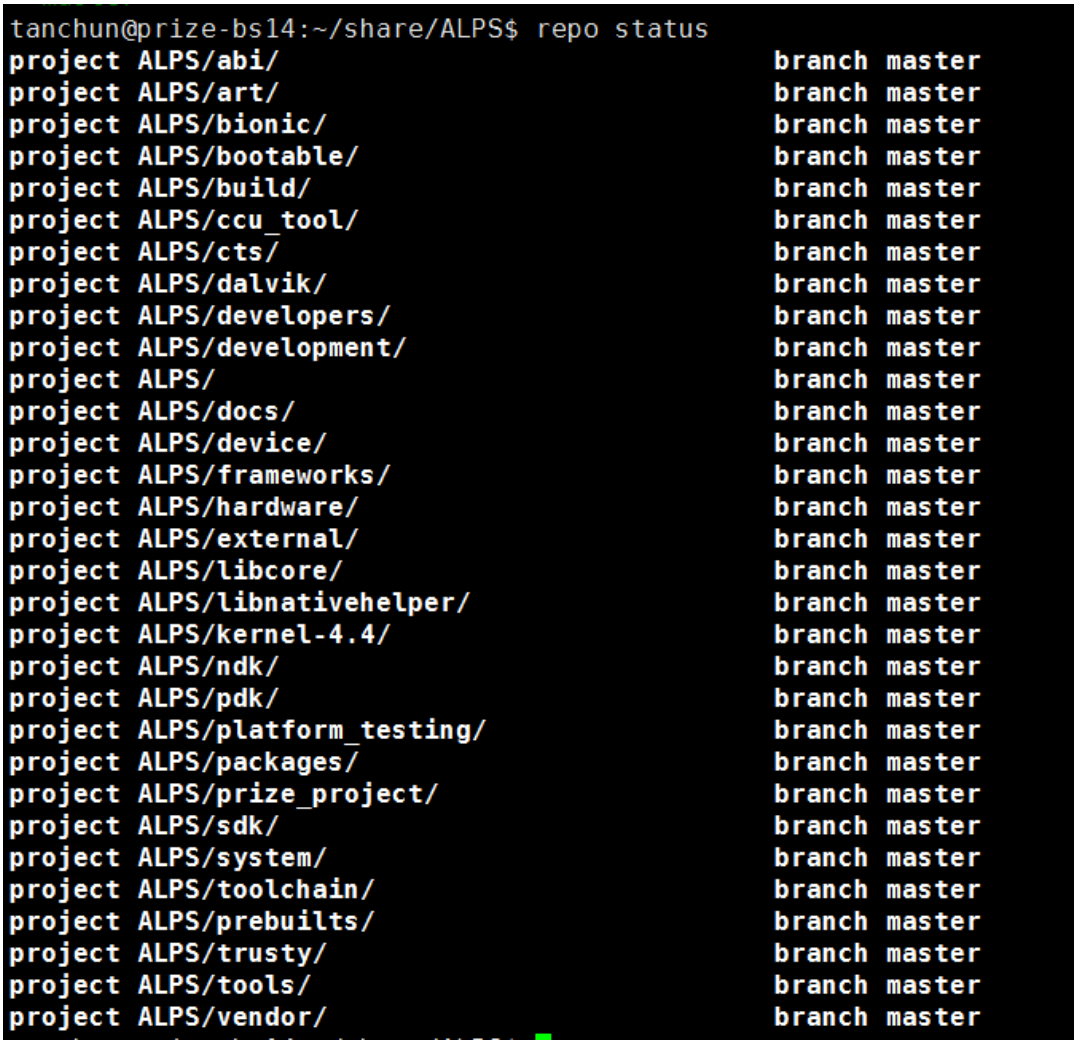
repo forall –c git pull　　(全部子仓库拉取更新)

repo forall –c git fetsh

如果是拉取单个子仓库的话：cd 到对应的目录，直接用 git pull ，git fetch,不过一般不建议

基本你可以理解为 repo forall+ git 命令的意思是我同时批量操作多个子仓库

另外，强调三点：第一，必须保证所有子仓库在 master 分支上，第二，提交前，必须先用 repo status 查看是否存在还没 commit 的文件，第三，最后用 repo upload 来上传



可能会遇到的问题：

问题：repo sync 出现类似下面的提示的话，可以忽略，查文档发现：git-lfs 是 gitlab 外置的一个工具，用于 Git 大文件存储（Git LFS），大型文件主要是高分辨率的图像和视频文件。而且经实测，虽然有提示但是是不影响项目编译的，所以可以忽略！

Fetching projects: 100% (30/30), done.
'mediatek/proprietary/hardware/libcamera_3a/libccu_lib/mt6757/prebuilt/libccu_bin/libccu_bin.pm': 1: git-lfs smudge --skip --
'mediatek/proprietary/hardware/libcamera_3a/libccu_lib/mt6757/prebuilt/libccu_bin/libccu_bin.pm': git-lfs: not found
error: external filter git-lfs smudge --skip -- %f failed -1
error: external filter git-lfs smudge --skip -- %f failed
Checking out files: 100% (68359/68359), done.
Syncing work tree: 100% (30/30), done.