**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](mailto: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](mailto:git@192.168.1.49:example1/manifest.git)

git@192.168.1.49:example2/manifest.git

git@192.168.1.49:example3/manifest.git

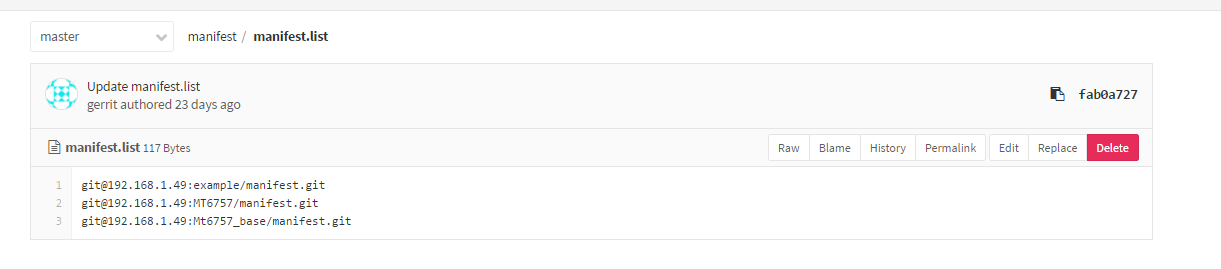
git@192.168.1.49:example4/manifest.git

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

以下是我们是现在的汇总清单库



1. 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](mailto: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>

以下是我们项目具体的子仓库清单列表：



1. 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.xml.sh

生成命令：

bash –x default.xml.sh [git@192.168.1.49:example1/manifest.git](mailto:git@192.168.1.49:example1/manifest.git) gerrit [gerrit@szprize.com](mailto: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](mailto: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](mailto:gerrit@szprize.com) (git push 需要用到)

1. 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

1. 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](mailto:git@192.168.1.49:example1/manifest.git) prize-bs14 [prize-bs14@szprize.com](mailto: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](mailto: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](mailto:gerrit@szprize.com) (git push 需要用到)

1. 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

1. 负责人从gitlab克隆项目，执行：

* repo init –u [**git@192.168.1.49:MT6757/manifest.git**](mailto:git@192.168.1.49:MT6757/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

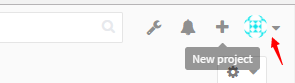
1. 通过gerrit页面，审核修改部分通过，submit并merge，确认gitlab数据同步到位
2. 开发人员进行开发

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>

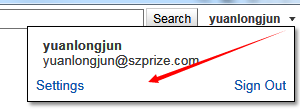
（默认第一次登录需要您重置密码，你可以再次输入密码：123456789作为你的密码，确认后

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](http://192.168.1.49:8081/" \l "/settings/ssh-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](mailto:name@szprize.com)

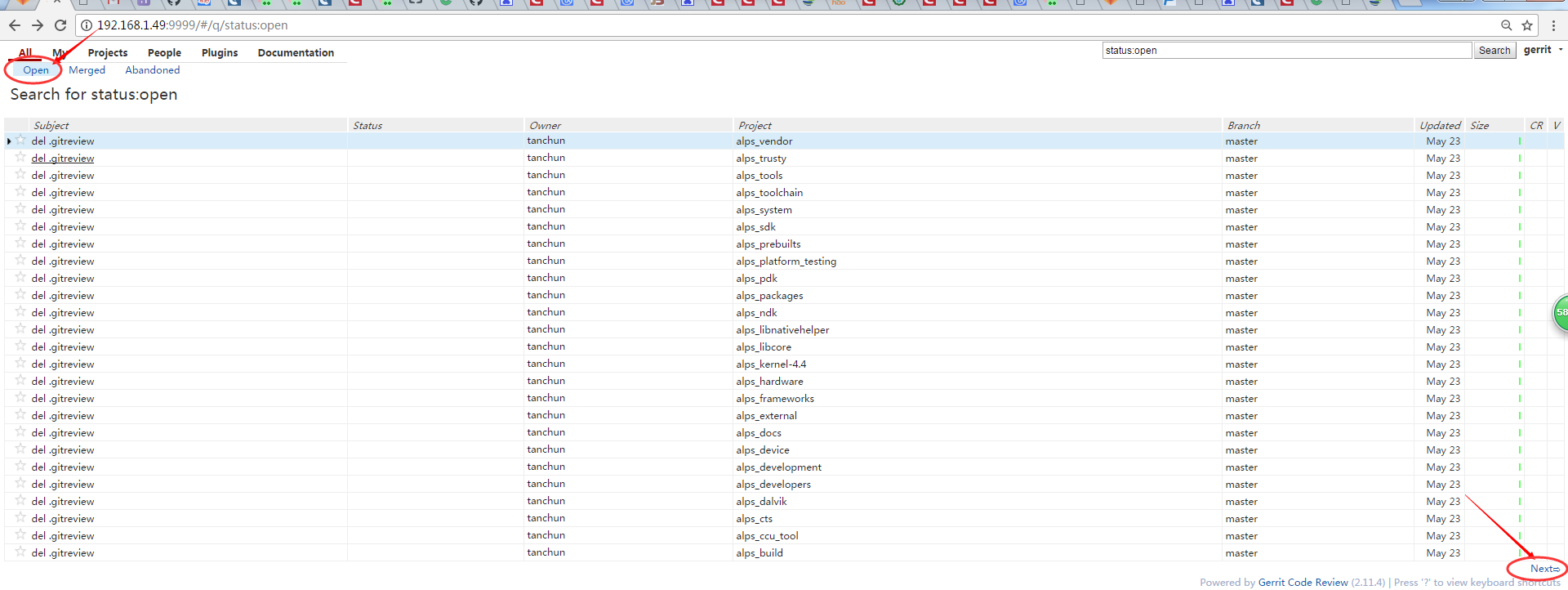
（这两步需要提前做，不然repo sync时出现报错提示）

* repo init –u [**git@192.168.1.49:Mt6757\_base/manifest.git**](mailto: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。

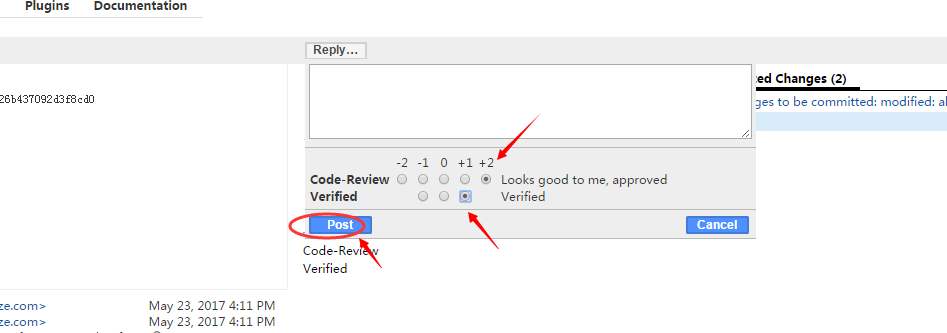
（repo upload执行后，它会自动弹出一个关于你的之前操作的一些commit清单，里面基本上带有#注释的，你可以把光标移到最左侧，使用vim的ctrl + V的块编辑，选中首字母#，按d，把#删掉，然后保存退出）

最后进行代码审核：

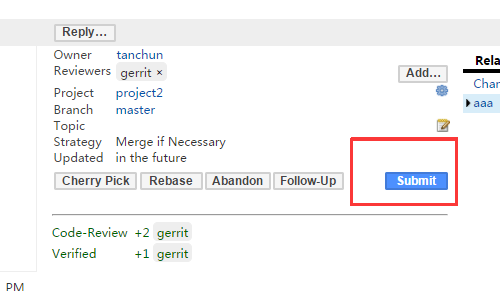
* 登录gerrit: <http://192.168.1.49:9999>,点击all下面的open,查看你提交的代码条目（如下图，如果仓库比较多请翻页）。



* 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：

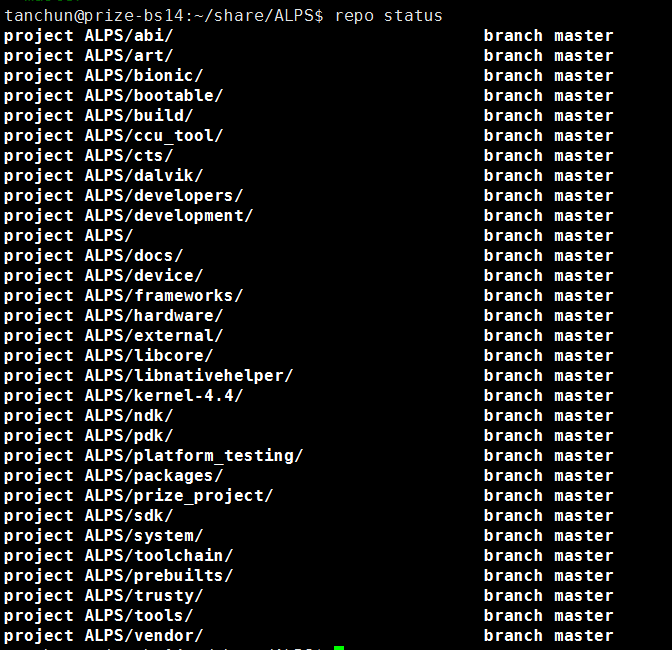
repo forall –c git pull (全部子仓库拉取更新)

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.