# Repo仓库搭建

1. **repo**

repo是一个基于git的仓库管理工具，是一个python脚本。对于包含有多个git仓库的工程，特别像android那样那么多git的工程，repo提供了非常好的解决方法。Repo可以管理很多的git仓库，可以统一上传等操作，并且可以自动化部分android的开发流程。

1. **repo引导器**

repo包含两部分，一个为python脚本的repo，通常放在~/bin目录，如果通过命令的方式安装repo的，repo一般在usr/bin目录下，这样我们就不需要再下载repo放到~/bin目录下；一个为repo.git，是一个git仓库，位于由repo管理的工程的.repo目录下的repo文件夹。

Python脚本的repo类似于一个引导器，通过以下命令下载并设置为可执行：

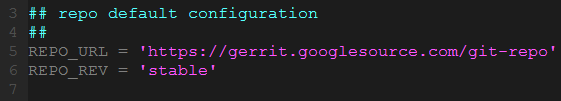
curl https://storage.googleapis.com/git-repo-downloads/repo > ~/bin/repo

chmod u+x ~/bin/repo

设置repo为可执行权限之后，设置~/bin到PATH，运行repo，会出现如下提示：



这是因为当前目录下不存在.repo/repo文件夹。Repo脚本里面有两个变量，分别为REPO\_URL和REPO\_REV，这两个变量默认的值为：

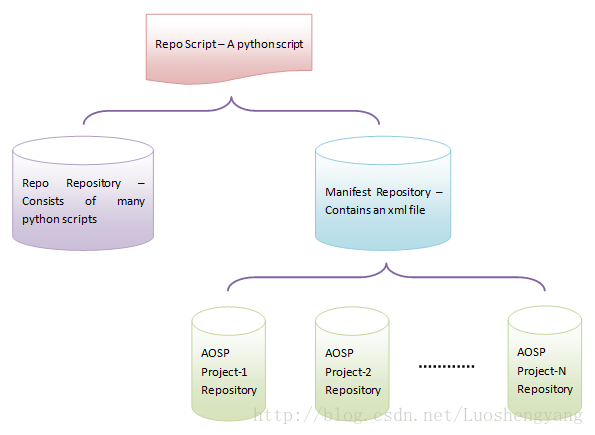


运行repo init的时候，如果发现当前目录没有.repo/repo，repo默认从该地址获取repo，放到当前目录的.repo/repo下面。

1. **repo管理的工程**

**一**个完整的repo工程必须由两部分组成，一个是上一节所说的repo，另一个是manifests.git。manifest.git提供repo工程所需要的所有git仓库的地址，repo根据manifest.git提供的xml文件，从xml文件解析各个仓库的地址，然后从这些地址获取git仓库，最后检出这些仓库到工作区。

一个repo工程包含有repo脚本、repo仓库、manifest仓库和工程所需要的project git仓库，下面一张图非常好地勾勒了repo的体系框架，如下图所示：

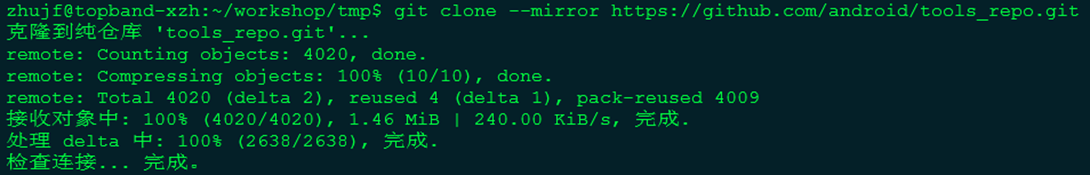


通常，我们建立仓库，需要准备好repo和manifest.git两个仓库，如果不自己准备repo，运行repo init的时候，repo会自动从默认的地址获取repo仓库。因为repo的默认地址是google的，有时候很难访问，所以，一般情况下都是自己准备好一个repo仓库，然后，运行repo init的时候，指定repo仓库地址即可。manifest.git仓库存放我们工程所有git仓库的地址，因此，需要根据仓库的地址，准备好xml文件。

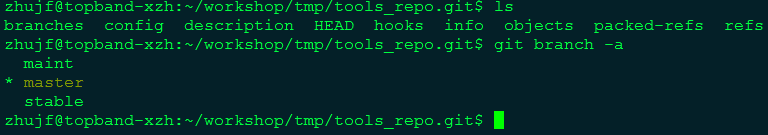
1. **repo仓库创建**

这一节将创建自己的repo仓库，这样，运行repo init的时候，就不需要从google的网站获取repo，而是从自己的服务器获取repo。

首先，我们先clone一份repo工具，可以从这里clone，如下所示：



git clone –mirror不同于git clone，带—mirror表示获取下来的是一个bare和带有所有分支的仓库，如下图所示：



假如已经在服务器建立了一个repo仓库，运行如下命令，把repo tool仓库push到服务器的repo仓库：

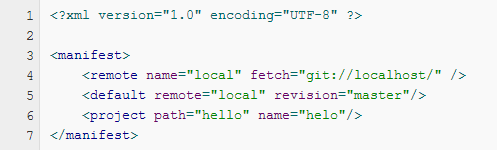
git push –mirror [服务器repo仓库地址]；

至此，repo仓库创建完成。

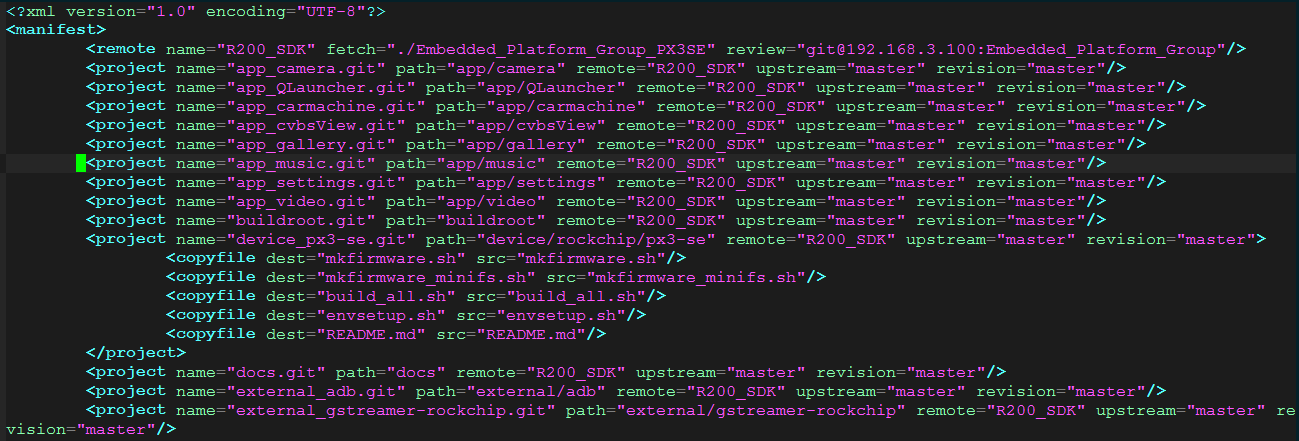
1. **manifests.git仓库创建**

manifests.git是仓库的集合，工程的所有git仓库的地址都在manifest.git仓库下，通常manifests.git可以包含多个工程的xml文件，在repo init的时候，指定使用哪个xml文件即可，这样，通个一个manifests.git仓库就可以管理非常多工程。

最简单的一个manifests.git仓库是，只包含一个default.xml文件，如果repo init的时候，没有指定使用哪个xml文件，那么repo默认使用default.xml文件。一个简单的default.xml内容如下：



下面以px3se的xml为例，解释如何创建自己的xml文件



一个xml文件至少包含remote和project元素，其中,元素remote表示服务器地址，xml文件可以创建多个remote，下面的元素project通过remote属性，指定使用哪个remote。

元素remote通常包含name、fetch和review属性，各个属性的含义如下所示：

* 1. name: 远程git服务器的名字，元素project通过remote属性，指定从哪个git服务器获取，获取project到本地后，该project的up stream为name，地址为元素remote的地址；
  2. alias：远程git服务器的别名，如果指定了，则会覆盖name的设定。在manifest中，各个remote元素的name不可以重名，但alias可以重名；
  3. fetch：定义了git仓库在属性review的路径下，如果没有review，fetch为仓库地址；
  4. review：定义了审核代码的服务器主机，repo upload命令需要用到该属性；

remote元素的fetch和review属性有些repo不一样，有些repo获取代码的时候，从fetch属性值获取，但，在px3se项目，review为git远程服务器的前缀，fetch为git仓库的路径，所以，如果遇到不能获取git仓库的，尝试修改这两个值。

px3se项目的review属性值为git@192.168.3.100:Embedded\_Platform\_Group，fetch属性值为./Embedded\_Platform\_Group\_PX3SE，表示git远程服务器在git@192.168.3.100:Embedded\_Platform\_Group的Embedded\_Platform\_Group\_PX3SE下。

manifest的xml有一个default元素，px3se项目的xml文件目前没有设置该元素，有待以后完善，但该元素属于一个非常重要的元素。Default元素设定xml包含的project的默认属性值，如果project没有设定该属性，那么默认采用default元素设定的属性。remote元素的各个属性含义如下：

1. remote：该属性值为前面定义的其中一个remote元素的name属性值，用于指定使用哪一个git服务器；
2. revision：git分支的名字，表示从project的git仓库里面的哪个分支获取代码，例如，master或者regs/heads/master；
3. sync\_j：在repo sync的时候，默认有多少的并行线程；
4. sync\_c：如果设置为ture，则只同步指定的分支（revision属性指定），而不是所有的ref内容；
5. sync\_s：如果设置为true，则会同步git 的子项目

xml文件的project元素指定该工程的git仓库，一个project元素指定一个需要clone的git仓库，project元素各个属性的含义如下：

1. name：唯一标识project的名字，同时，用于生成该project的git仓库的URL，生成方法是，${remote\_fetch}/${project\_name}；
2. path：可选的路径。指定git clone project出来，代码在本地存放的位置。如果没有指定，则代码存放在以name属性值为目录名的路径；
3. remote：指定从哪个remote元素获取，属性值为目标remote元素的name属性值；
4. revision：指定需要获取的git提交点，可以是master，refs/heads/master，tag或者SHA-1值，如果为SHA-1（commit点），upstream不能省略，repo将会从upstream取revision；
5. groups：可不设。指定project所属的组，以空格或者逗号分隔多个组名。所有的project都自动属于”all”组，每一个project自动属于name和path属性值的组。例如<project name=”monkeys” path=”barrel-of”/>，该project自动属于default、monkeys和barrel-of组。如果一个project属于notdefault组，那么，该project在repo sync的时候，不会下载该project；
6. sync\_c：可不设。如果设置为true，则只会同步指定的分支（revision属性指定），而不是所有的ref内容；
7. sync\_s：可不设。如果设置为true，则会同步git的子项目；
8. upstream：在哪个git分支可以找到revision；

上面了解了xml文件的格式，下一步创建manifests.git仓库。首先，新建一个manifests.git目录，进入该目录，运行git init命令，初始化为一个git仓库，如下所示：

mkdir manifests.git;

cd manifests.git;

git init;

把准备好的xml文件放到manifests.git目录后，进行如下操作：

git add .;

git commit –am “commit xml file”;

git push –mirror [存放manifests.git仓库的服务器地址];

在这里，如果是代码管理工具是git-lab，那么，首先在git-lab下创建manifests仓库，然后复制仓库地址，把仓库地址放到上面的git push命令即可。

至此，manifests.git仓库创建完成。

1. **project仓库创建**

根据项目的git仓库数量，如果git仓库数量少，可以直接人工在git-lab上面一个一个创建git仓库，然后，根据仓库的地址和仓库在工程中的位置，添加到xml文件即可。

如果project git过多，如android拥有500多个仓库的工程，通过人工一个一个创建仓库是不切实际的，建议采用的方法是，编写脚本，读取android工程中的.repo/project.list文件，在本地创建好一个完整的android bare仓库，然后想方设法，把含有android所有git的bare仓库添加到git-lab中。

假设，通过上一步已经在git-lab中创建了整个工程所需要的project git仓库。如果git仓库很少，可以在本地，在每个git仓库运行git push –mirror [project git对应的服务器地址]，把每个仓库的代码都推送到服务器。

如果git仓库非常多，那么，建议的方法是，编写一个脚本，遍历本地工程的所有git仓库，然后，进入每个git仓库的目录，运行git push –mirror [project git对应的服务器地址]，这样效率非常高。

1. **获取repo工程**

通过前面的章节，已经完成了repo、manifests.git和project各个仓库的建立和推送到服务器，那么接下来的工作是，如何在本地获取已经创建好的repo工程。

获取的方法非常简单，只需要运行两个命令即可，本节以px3se为例获取repo工程，获取方法如下：

repo init --repo-url=git@192.168.3.100:Embedded\_Platform\_Group/repo.git

-u git@192.168.3.100:Embedded\_Platform\_Group/manifest.git -m px3se\_linux\_sdk.xml;

repo sync;

其中，repo-url指定我们前面所创建的repo仓库地址，-u指定的是manifests.git仓库地址，-m指定manifests.git仓库中的xml文件。如果manifests.git仓库管理非常多的工程，那么-m选项非常有用。

1. **常见问题**
2. fatal: branch 'stable' has not been signed。

最后查实必须要有stable分支和tag信息才能正常进行下面的步骤。

1. Please, commit your changes or stash them before you can switch branches. Aborting。

这个其实是Git提示的信息，解决方法就是到每个库中执行一个git reset --hard就好了。参考Switching a branch after aborting current changes in git。起因就是库已经错乱了，可能源于不熟悉repo，乱同步了不同的库造成的。朱炯福