Repo

repo为了方便管理git库而开发的python脚本。

.repo/manifests

<?xml version="1.0" encoding="UTF-8"?>

<manifest>

<remote name="aosp"

fetch=".."

review="https://android-review.googlesource.com/" />

<default revision="master"

remote="aosp"

sync-j="4" />

<project path="build" name="platform/build" groups="pdk,tradefed" >

<copyfile src="core/root.mk" dest="Makefile" />

</project>

<project path="abi/cpp" name="platform/abi/cpp" groups="pdk" />

<project path="art" name="platform/art" groups="pdk" />

...

<project path="tools/studio/translation" name="platform/tools/studio/translation" groups="notdefault,tools" />

<project path="tools/swt" name="platform/tools/swt" groups="notdefault,tools" />

</manifest>

<remote>：描述远程仓库信息，name描述名称（xml属性），fetch项目名前缘，review：审查的地址

<default>：定义属性，project标签的默认属性，属性revision表示当前版本，sync -j 表示同步远程代码时，并发任务数量。

<project>：每个repo管理的git库，就对应到每个project标签，path描述项目相对远程仓库url的路径。同时将作为对应git库在本地代码路径，name定义项目名称。

修改：

假如需要修改替换一些git库，需要修改default.xml。但直接定制会导致下一个更新项目清单造成冲突。repo提供了一个灵活方式local\_manifest:所有定制是遵循default.xml规范。文件可自定义。将新建的.repo/local\_manifests子目录即可。repo会遍历 .repo/local\_manifests目录下的所有\*.xml文件，最终与default.xml合并成一个总项目清单文件manifest.xml。

REPO脚本库：.repo/repo

repo是对git封装，提供一套repo命令集，init和sync等，repo自动化实现也都在git库里面，在第一次初始化时，repo会远程下载git库到本地。

仓库目录和工作目录：

仓库目录保存历史信息，修改记录等，工作目录保存的是当前版本信息。git仓库目录位于工作目录下，但git支持仓库和工作目录分开放。repo管理既有分开，也有在工作目录下。

Manifests:仓库目录有两份拷贝，一份位于工作目录（.repo/manifests)的 .git目录下，另一份独立存放于.repo/manifests.git

repo:仓库目录位于工作目录(.repo/repo)的git目录下

project：所有被管理git库的仓库目录都是分开存放的，位于.repo/projects目录下。同时保留工作目录.git，但文件都归于.repo链接。

在下载一个项目时，仅需要拷贝目录即可，repo会支持本地已有的.repo中恢复原有的项目。

*/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\** 使用介绍 *\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**/

repo cmd option

repo init -u <url> <options>

-u : 制定manifests这个远程git仓库的URL，manifests库是整个项目清单。git默认包含default.xml一个文件

-m : (-manifests -name)指定所需的manifests库。默认使用manifests/default.xml

-b ：（-manifests -branch指制定manifests.xml文件中一个版本，分支。

--trace：可以查看如何生成。

repo --trace init -u url -b branch -m manifests

在当前目录进行初始化repo ，clone两个git库，一个-u指定了manifests，另一个是manifests.git 另一个是默认的repo 创建manifests/.git目录，将manifests切换branch指定分支，在.repo目录下-m创建软链接，链接到指定的清单文件，默认manifests/default.xml

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*不常用的参数\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

下载安卓源码：-repo --url 指定远程仓库url，默认googlesource但是国内受限，不允许访问。

-repo-branch：这个与manifests一样，repo有版本差异，通过参数指定下载repo这个远程分支。

- no – repo – verify 下载repo库，对repo源码检查，通过 -url指定第三方远程库，可能检查不通过。

Repo sync

repo sync [project\_list]

下载远程代码，并且讲本地更新，（同步），根据本地git库不同，会产生不同行为：

1、本地git库第一次处罚同步操作，该命令等同git clone

2、本地已触发过同步，等价git remote update && git rebase origin/<branch>，branch是本地分支关联的远程分支，代码合并可能产生冲突，只要解决冲突，然后git rebase –continue

sync的命令参数：

-j： 开启多线程同步操作，jn n为数量，看电脑系统核数线程数。默认4

-c ： - curren – branch： 只同步特定远程分支。默认会同步全部远程分支

-d ： -detach ：脱离当前本地分支，切换manifest中设定分支，实际操作里，这个参数很有用，当第一次sync后，切换到dev分支进行开发，如果不带该参数sync，则触发本地dev分支和manifests设定的远程分支进行合并，导致很多sync失败

-f ： -force -brokoken： 当git库sync失败，不中断同步操作，继续同步其他git库

-no -clone -bundle ： 在服务器发起请求时，为了做到尽快的响应速度，会用到内容分发到网络。同步会通过CDN(content delivery network）内容交付网络，就近与服务器建立连接，使用http或https的url/clone.bundle（远程镜像）初始化本地git库。

Upload

repo upload [project\_list]

上传，本地代码上传到元策划嗯服务器，upload命令会先找到本地分支从上一次同步操作以来，发生过的变动，生成patch文件，上传gerrit服务器。如果没有peoject\_list ，upload会找出所有git库改动，如果git库有多分支，则提供一个交互界面，提示选择若干分支操作。upload不会直接上传就合并到git库，要审核review批准。review查看内容选择合并，基本上通过gerrit完成。

reviewer：代码审阅人员。-re reviewer 指定审核员

commit-id：为了标识每个提交，引入commit-id，是一个SHA-1值，当提交内容一个checksum，用于验证提交内容完整性。

change-id gerrit针对每个review任务，引入一个change -id ，每个提交对应一个id，区分commit id 会用大写I开头。

path-set：当前需要review改动，一个change id关联多个commit id，通过path-set，表现的。当通过git commit –amend 命令修正上一次提交并上传时，commit id已经发生变化，但是仍可以保持change id不变，gerrit在原来review任务下，就出现新的patch set，修正多少次，就会出现多少个path。最新才是真正有用的，可合并的。

Download

repo download <target> <change>

upload是把改动内容提交gerrit，download是从gerrit下载改动

target：指定要下载的改动内容project

change：指要下载的改动内容。 review的url后几位数据。

Forall

repo forall [project\_list] -c <command>

指定git库执行-c制定的命令序列。project 用空格隔开

-r ： -regex ： 通过正则表达式，匹配project

-p： 输出结果，打印project名称

prune

repo prune [<project\_list>]

删除指定project已合并分支。当在开发分支上代码已经合并到主干分支后，使用命令可以删除该开发分支。

Start

repo start <branch\_name> [<project\_list>]

在指定project的上，切换到<branch\_name>指定分支，可以用-all参数对project都执行分支切换的操作。该命令是对git checkout命令封装

branch name 自定义，追踪manifests中指定分支名

当第一次sync代码后，可以通过start切换到开发分支，避免在匿名分支上工作导致丢失

status

repo status [<project\_list>]

status查看多个git库状态，git status一样

对项目清单文件进行定制

通过local\_manifest机制，避免直接修改default.xml，不会造成下次同步远程清单文件冲突。

local\_manifest.xml会与已有的default.xml融合成一个项目清单文件manifest.xml实现对git库进行替换和新增，通过以下命令可以到处清单文件。最终snapshot.xml是融合版本

repo manifest -o snapshot.xml -r

在编译之前，保存项目清单，可助问题溯源。回退版本验证，只需要基于snapshot上初始化上一个版本即可。

无法下载android源码问题

repo init 的时候，会远程下载manifest和repo两个git库，默认情况，两个git库都是写死在repo脚本里面，对aosp来说，这两个git库地址是google的，可以用-u -repo -url 参数，自定义库地址并且通 -no -repo -verify来绕过代码检查

repo init - -repo -url [path/to/repo] -u [path/to/manifest] -b [branch] - - no -repo

repo sync

更快的下载远程代码

repo sync -c 只下载manifest设定的分支，节省代码下载时间，以及本地磁盘空间。

repo manifest -o snapshot.xml -r

repo init -u [path/to/manifest] -b [another\_branch]

repo sync -c -d 不想被其他分支干扰，切换分支，这些命令相当于更新了manifest，而且之下在another branch的代码，本地只保存了两个分支，利用保存的snapshot.xml通过所有git库切换回原来的分支。

如果本地已有android源码，向下载另一份新的，通过reference参数，数分钟就可以下载完成

mkdir ～/android-new && cd ～/android-new

repo init – – reference =～/android-exsit -u [path/to/manifest] -b 【branch】

repo sync -c

避免在匿名分支工作

sync后，所有git库默认都在一个匿名分支上no branch ，容易由于五操作导致丢失代码修改，可用到以下命令所有git库切换到开发分支。

Repo start branch -all branch为分支名

使用upload提交代码

repo upload 在多个git库上修改代码，针对每个库单独提交繁琐，可以一并提交

repo upload - - review=“[R.E.viewer@google.com](mailto:R.E.viewer@google.com)”提交代码到指定reviewer邮箱中。

定期删除已合并的开发分支

git鼓励修复bug和开发feature时，使用新分支，创建分支代价小，速度快，已合并分支没有存在的价值，可以删除

repo prune [project\_list]

同时操作多个git库

repo forall -c “git branch | grep tmp | xargs git branch -D ；git branch”