以下是今天学习 repo 的一些操作描述和理解：

repo 的安装：curl https://storage.googleapis.com/git-repo-downloads/repo > ~/bin/repo
由于国内 Google 访问受限，所以上述命令不一定能下载成功。
将上述 URL 换成清华大学开源软件镜像：https://mirrors.tuna.tsinghua.edu.cn/git/git-repo

repo 关注的是当前 git 库的数量、名称、路径等；
repo 通过一个 git 库来管理项目清单文件，这个库叫 manifests，在这个目录下可以看到.git 文件夹和 default.xml 文件；

```
majc0806@majc0806-OptiPlex-3020:~/repotest/.repo/manifests$ ll
总用量 32
drwxrwxr-x 3 majc0806 majc0806  4096 8月   17 17:02 ./
drwxrwxr-x 5 majc0806 majc0806  4096 8月   17 17:02 ../
-rw-rw-r-- 1 majc0806 majc0806 17892 8月   17 17:02 default.xml
drwxrwxr-x 2 majc0806 majc0806  4096 8月   17 17:02 .git/
```

打开 repo 这个可执行文件，发现是一个可执行的 Python 脚本，但实际上 repo 是一系列脚本的集合，这些脚本也是通过 git 库来维护的，这个 git 库名字叫 repo。在位置在 project/.repo/repo，进入该文件夹下可以看到许多 Python 脚本和一个.git 文件夹；

```
majc0806@majc0806-OptiPlex-3020:~/repotest/.repo/repo$ ll
总用量 584
drwxrwxr-x 7 majc0806 majc0806  4096 8月   17 17:02 ./
drwxrwxr-x 5 majc0806 majc0806  4096 8月   17 17:02 ../
-rw-rw-r-- 1 majc0806 majc0806  4439 8月   17 17:02 color.py
-rw-rw-r-- 1 majc0806 majc0806  6355 8月   17 17:02 color.pyc
-rw-rw-r-- 1 majc0806 majc0806  7736 8月   17 17:02 command.py
-rw-rw-r-- 1 majc0806 majc0806  8704 8月   17 17:02 command.pyc
-rw-rw-r-- 1 majc0806 majc0806 11358 8月   17 17:02 COPYING
drwxrwxr-x 2 majc0806 majc0806  4096 8月   17 17:02 docs/
-rw-rw-r-- 1 majc0806 majc0806  2660 8月   17 17:02 editor.py
-rw-rw-r-- 1 majc0806 majc0806  2779 8月   17 17:02 editor.pyc
-rw-rw-r-- 1 majc0806 majc0806  3107 8月   17 17:02 error.py
-rw-rw-r-- 1 majc0806 majc0806  5807 8月   17 17:02 error.pyc
-rw-rw-r-- 1 majc0806 majc0806    50 8月   17 17:02 .flake8
drwxrwxr-x 8 majc0806 majc0806  4096 8月   17 17:02 .git/
-rw-rw-r-- 1 majc0806 majc0806   121 8月   17 17:02 .gitattributes
```

在使用 repo init 一个项目时，就会从远程把 manifests 和 repo 这两个 git 库拷贝到本地；

```
majc0806@majc0806-OptiPlex-3020:~/repotest/.repo$ ll
总用量 20
drwxrwxr-x  5 majc0806 majc0806 4096 8月   17 17:02 ./
drwxrwxr-x  3 majc0806 majc0806 4096 8月   17 17:02 ../
drwxrwxr-x  3 majc0806 majc0806 4096 8月   17 17:02 manifests/
drwxrwxr-x 10 majc0806 majc0806 4096 8月   17 17:02 manifests.git/
lrwxrwxrwx  1 majc0806 majc0806   21 8月   17 17:02 manifest.xml -> manifests/default.xml
drwxrwxr-x  7 majc0806 majc0806 4096 8月   17 17:02 repo/
```

.repo/manifests/default.xml 文件的内容理解：

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

  <remote  name="aosp"
           fetch=".." />
  <default revision="refs/tags/android-4.0.4_r2.1"
           remote="aosp"
           sync-j="4" />

  <project path="build" name="platform/build">
    <copyfile src="core/root.mk" dest="Makefile" />
  </project>
  <project path="abi/cpp" name="platform/abi/cpp" />
  <project path="bionic" name="platform/bionic" />
  <project path="bootable/bootloader/legacy" name="platform/bootable/bootloader/legacy" />
  <project path="bootable/diskinstaller" name="platform/bootable/diskinstaller" />
  <project path="bootable/recovery" name="platform/bootable/recovery" />
```

<remote>：描述了远程仓库的基本信息。name 描述的是一个远程仓库的名称，通常我们看到的命名是 origin;fetch 用作项目名称的前缘，在构造项目仓库远程地址时使用到;review 描述的是用作 code review 的 server 地址。

<default>：default 标签定义的属性将作为**<project>**标签的默认属性，在**<project>**标签中，也可以重写这些属性。属性 revision 表示当前的版本，也就是我们俗称的分支;属性 remote 描述的是默认使用的远程仓库名称，即**<remote>**标签中 name 的属性值;属性 sync-j 表示在同步远程代码时，并发的任务数量，配置高的机器可以将这个值调大。

<project>：每一个 repo 管理的 git 库，就是对应到一个**<project>**标签，path 描述的是项目相对于远程仓库 URL 的路径，同时将作为对应的 git 库在本地代码的路径;name 用于定义项目名称，命名方式采用的是整个项目 URL 的相对地址。 譬如，AOSP 项目的 URL 为 https://android.googlesource.com/，命名为 **platform/build** 的 git 库，访问的 URL 就是 https://android.googlesource.com/platform/build

仓库目录保存的是历史信息和修改记录，工作目录保存的是当前版本的信息。一般来说，一个项目的 Git 仓库目录（默认为.git 目录）是位于工作目录下面的，但是 Git 支持将一个项目的 Git 仓库目录和工作目录分开来存放。 对于 repo 管理而言，既有分开存放，也有位于工作目录存放的:

- **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，也会保留工作目录的.git，但里面所有的文件都是到.repo 的链接。这样，即做到了分开存放，也兼容了在工作目录下的所有 git 命令。

既然.repo 目录下保存了项目的所有信息，所有要拷贝一个项目时，只是需要拷贝这个目录就可以了。repo 支持从本地已有的.repo 中恢复原有的项目。

```
majc0806@majc0806-OptiPlex-3020:~/repotest/.repo/manifests/.git$ ll
总用量 16
drwxrwxr-x 2 majc0806 majc0806 4096 8月   17 17:02 ./
drwxrwxr-x 3 majc0806 majc0806 4096 8月   17 17:02 ../
lrwxrwxrwx 1 majc0806 majc0806   26 8月   17 17:02 config -> ../../manifests.git/config
lrwxrwxrwx 1 majc0806 majc0806   31 8月   17 17:02 description -> ../../manifests.git/description
-rw-rw-r-- 1 majc0806 majc0806   24 8月   17 17:02 HEAD
lrwxrwxrwx 1 majc0806 majc0806   25 8月   17 17:02 hooks -> ../../manifests.git/hooks/
-rw-rw-r-- 1 majc0806 majc0806  145 8月   17 17:02 index
lrwxrwxrwx 1 majc0806 majc0806   24 8月   17 17:02 info -> ../../manifests.git/info/
lrwxrwxrwx 1 majc0806 majc0806   24 8月   17 17:02 logs -> ../../manifests.git/logs/
lrwxrwxrwx 1 majc0806 majc0806   27 8月   17 17:02 objects -> ../../manifests.git/objects/
lrwxrwxrwx 1 majc0806 majc0806   31 8月   17 17:02 packed-refs -> ../../manifests.git/packed-refs
lrwxrwxrwx 1 majc0806 majc0806   24 8月   17 17:02 refs -> ../../manifests.git/refs/
lrwxrwxrwx 1 majc0806 majc0806   28 8月   17 17:02 rr-cache -> ../../manifests.git/rr-cache/
lrwxrwxrwx 1 majc0806 majc0806   27 8月   17 17:02 shallow -> ../../manifests.git/shallow
lrwxrwxrwx 1 majc0806 majc0806   23 8月   17 17:02 svn -> ../../manifests.git/svn/
```

使用：

`repo init -u <URL> [<OPTIONS>]`

-u：指定 `manifests` 这个远程 `git` 库的 `URL`，`manifests` 库是整个项目的清单。默认情况，这个 `git` 库只包含了 `default.xml` 一个文件，其内容可以参见 Android 的样本

-m， –manifest-name：指定所需要的 manifests 库中的清单文件。默认情况下，会使用 maniftests/default.xml

-b， –manifest-branch：指定 manifest.xml 文件中的一个版本，，也就是俗称的“分支”

运行该命令后，会在当前目录下新建一个.repo 子目录

`repo sync [PROJECT_LIST]`

下载远程代码，并将本地代码更新到最新，这个过程称为“同步”。如果不使用任何参数，那么会对所有 repo 管理的进行同步操作;也可以 PROJECT_LIST 参数，指定若干要同步的 PROJECT。 根据本地 git 库代码不同，同步操作会有不同的行为：

•当本地的 git 库是第一次触发同步操作时，那么，该命令等价于 `git clone`，会将远程 git 库直接拷贝到本地

•当本地已经触发过同步操作时，那么，该命令等价于 `git remote update && git rebase origin/<BRANCH>`，<BRANCH>就是当前与本地分支所关联的远程分支 代码合并可能会产生冲突，当冲突出现时，只需要解决完冲突，然后执行 `git rebase --continue` 即可

```
repo upload [PROJECT_LIST]
```
从字面意思理解，upload 就是要上传，将本地的代码上传到远程服务器。upload 命令首先会找出本地分支从上一次同步操作以来发生的改动，然后会将这些改动生成 Patch 文件，上传至 *Gerrit* 服务器。 如果没有指定 PROJECT_LIST，那么 upload 会找出所有 git 库的改动;如果某个 git 库有多个分支，upload 会提供一个交互界面，提示选择其中若干个分支进行上传操作。

upload 并不会直接将改动合并后远程的 git 库，而是需要先得到 Reviewer 批准。Reviewer 查看改动内容、决定是否批准合入代码的操作，都是通过 Gerrit 完成。 Gerrit 服务器的地址是在 manifests 中指定的：打开.repo/manifest.xml，<remote>这个 XML TAG 中的 review 属性值就是 Review 服务器的 URL:

```
<remote name="aosp"
        fetch=".."
        review="https://android-review.googlesource.com/" />
```

```
repo download <TARGET> <CHANGE>
```
upload 是把改动内容提交到 Gerrit，download 是从 Gerrit 下载改动。与 upload 一样，download 命令也是配合 Gerrit 使用的。

- **<TARGET>**：指定要下载的 PROJECT，譬如 *platform/frameworks/base*, *platform/packages/apps/Mms*
- **<CHANGE>**：指定要下载的改动内容。这个值不是 Commit-ID，也不是 Change-ID，而是一个 Review 任务 URL 的最后几位数字。 譬如，AOSP 的一个 Review 任务 https://android-review.googlesource.com/#/c/23823/，其中 **23823** 就是<CHANGE>。

```
repo forall [PROJECT_LIST] -c <COMMAND>
```
对指定的 git 库执行**-c** 参数制定的命令序列。在管理多个 git 库时，这是一条非常实用的命令。PROJECT_LIST 是以空格区分的，譬如：

```
$ repo forall frameworks/base packages/apps/Mms -c "git status"
```
表示对 *platform/frameworks/base* 和 *platform/packages/apps/Mms* 同时执行 `git status` 命令。如果没有指定 PROJECT_LIST，那么，会对 repo 管理的所有 git 库都同时执行命令。

该命令的还有一些其他参数：

- **-r, –regex**： 通过指定一个正则表达式，只有匹配的 PROJECT，才会执行指定的命令
- **-p**：输出结果中，打印 PROJECT 的名称

```
repo prune [<PROJECT_LIST>]
```
删除指定 PROJECT 中，已经合并的分支。当在开发分支上代码已经合并到主干分支后，使用该命令就可以删除这个开发分支。

随着时间的演进，开发分支会越来越多，在多人开发同一个 git 库，多开发分支的情况会愈发明显，假设当前 git 库有如下分支：

```
* master
```

```
  dev_feature1_201501   # 已经合并到 master
  dev_feature2_201502   # 已经合并到 master
  dev_feature3_201503   # 正在开发中，还有改动记录没有合并到 master
```
那么，针对该 git 库使用 prune 命令，会删除 dev_feature1_201501 和 dev_feature2_201502。

定义删除无用的分支，能够提交团队的开发和管理效率。prune 就是删除无用分支的”杀手锏“。

`repo start <BRANCH_NAME> [<PROJECT_LIST>]`
在指定的 PROJECT 的上，切换到<BRANCH_NAME>指定的分支。可以使用**–all** 参数对所有的 PROJECT 都执行分支切换操作。 该命令实际上是对 `git checkout` 命令的封装，<BRANCH_NAME>是自定义的，它将追踪 manifest 中指定的分支名。

当第一次 sync 完代码后，可以通过 start 命令将 git 库切换到开发分支，避免在匿名分支上工作导致丢失改动内容的情况。

`repo status [<PROJECT_LIST>]`

status 用于查看多个 git 库的状态。实际上，是对 `git status` 命令的封装。