关于 **repo** 的使用

目的：

本文档主要用于指导客户在自己的服务器上建立 Git 公共代码库，并使用 repo 进行下载和管理，通过 git 分布式版本控制系统来实现团队开发代码、合并 AW 发布代码同时进行。

1、建立自己的公共代码库

从全志服务器下载镜像仓库。存放于公共服务器上，假设客户自己的公共服务器为 Aserver，IP 地址为 192.168.2.7

负责人下载镜像仓库，并把镜像仓库作为公共代码库，其他开发人员则可以从这个公共代码库下载代码。

下载镜像仓库方法如下：
repo init 时加上--mirror 参数，即可下载到镜像仓库。

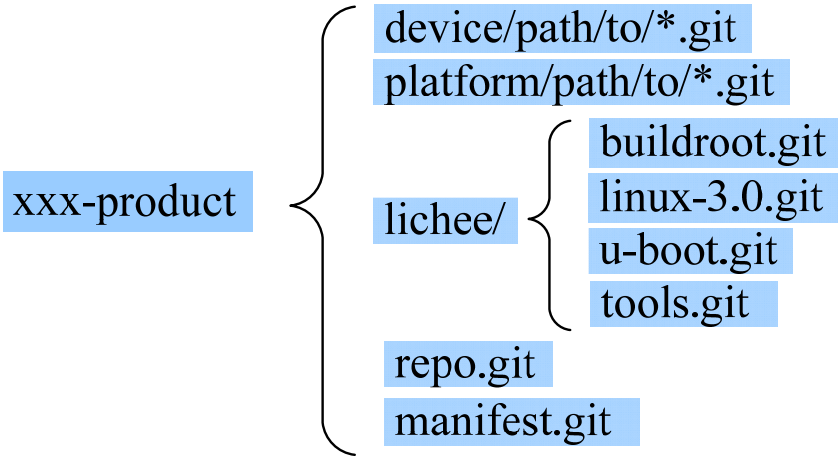
```
$ mkdir android4.0
$ cd android4.0
$ repo init -u
ssh://username@61.143.53.198/git_repo/homlet/manifest.git --mirror
$ repo sync

$ cd ..
$ mkdir lichee
$ cd lichee
$ repo init -u
ssh://username@61.143.53.198/git_repo/homlet/manifest.git -m
lichee.xml --mirror
$ repo sync
 # 如下载过程中显示错误，中止后，再次执行repo sync命令直到下载完成
```

下载完成后，将 android4.0 目录下生成的 device、platform、repo.git 这 3 个文件夹移动到 Aserver 服务器上 /git_repo/homlet/目录下（假设为公共代码库的位置）。其他剩余文件夹没有用处可以删除。

lichee 下面生成 buildroot.git、linux-3.0.git、tools.git、u-boot.git 这 4 个目录移动到 Aserver 服务器上 /git_repo/homlet/lichee/目录下。

整个仓库目录结构如下：



开发人员要从自己的代码库下载代码，还需要创建下载代码用的 manifest.git 仓库。如客户之前已经有 manifest.git 仓库，则只需添加两个 xml 文件（一个用于下载 android、一个用于下载 lichee）。如没有的话，可以从全志服务器下载一个镜像仓库，如下：

```
$ cd /git_repo/homlet
$ git clone --mirror
ssh://username@61.143.53.198/git_repo/homlet/manifest.git
```

下载完成后，将 manifest.git 目录也移动到/git_repo/homlet/目录下。

此时客户自己的代码库就算建立起来了。

2、负责人创建自己的开发分支

公共代码库中已存在一个 apollo-dev 分支，建议客户不要在此分支上做开发，而将此分支作为与全志服务器同步代码更新的分支。客户自己创建一个新的分支用于自己的独立开发，当全志发布新版本，同步更新后，再将全志的 apollo-dev 分支合并到自己的开发分支上即可。

创建步骤如下：
负责人先从自己的公共代码库下载代码。假设客户已经从自己的公共代码库下载成功。分支创建如下：

```
 # 已经从自己的公共代码库下载成功代码
$ cd android4.0
$ repo start apollo-dev --all
$ repo start develop --all    # 先在本地仓库创建develop分支
$ repo branch   # 查看是否创建成功
$ repo forall -c git push homlet develop:develop
```

```
   # 将自己的开发分支推送到服务器公共代码库中

$ cd ..
$ cd lichee
$ repo sync
$ repo start apollo-dev --all
$ repo start develop --all
$ repo branch
$ repo forall -c git push homlet develop:develop
    # 在lichee中也一样，创建develop分支并推送到公共代码库中
```

此时公共代码库中就有了 apollo-dev 和 develop 两个分支。开发人员可下载并进行开发。

3、开发人员从自己的公共代码库下载代码

客户的开发人员从自己代码库下载代码，速度快并节省时间。

开发人员下载之前，需要修改 repo 引导脚本，repo 引导脚本是用于下载 repo.git 仓库。将 repo 引导脚本中的地址“REPO_URL='ssh://username@61.143.53.198/git_repo/repo.git'”修改为客户自己的 repo.git 仓库下载地址。

（1）、假设开发人员 A 是在服务器 Aserver 上做开发，即开发代码和公共代码库在同一台服务器。开发人员 A 下载代码，登录到 Aserver 服务器上之后使用如下命令：

```
$ mkdir android4.0
$ cd android4.0
$ repo init -u /git_repo/homlet/manifest.git
$ repo sync
$ repo start apollo-dev --all
$ repo forall -c git checkout -b develop homlet/develop
$ repo branch

$ cd ..
$ mkdir lichee
$ cd lichee
$ repo init -u /git_repo/homlet/manifest.git -m lichee.xml
$ repo sync
$ repo start apollo-dev --all
$ repo forall -c git checkout -b develop homlet/develop
$ repo branch
```

（2）、假设开发人员 A 不是在 Aserver 服务器上做开发，而是登录其他服务器做开发（或者在自己的 PC 上做开发），从 Aserver 下载代码时使用如下命令：

```
$ mkdir android4.0
$ cd android4.0
$ repo init -u
ssh://developerA@192.168.2.7/git_repo/homlet/manifest.git
$ repo sync
$ repo start apollo-dev --all
$ repo forall -c git checkout -b develop homlet/develop
$ repo branch

$ cd ..
$ mkdir lichee
$ cd lichee
$ repo init -u
ssh://developerA@192.168.2.7/git_repo/homlet/manifest.git -m
lichee.xml
$ repo sync
$ repo start apollo-dev --all
$ repo forall -c git checkout -b develop homlet/develop
$ repo branch
```

开发人员有了 develop 分支，则可以进行一系列的开发工作。

4、开发人员本地提交，并推送到代码库

在自己的开发分支上修改代码并在本地提交，确认提交无误后，将提交推送到公共代码库中。

（1）、开发人员 A 在具体的目录中进行开发提交，并推送到公共库里。如：

```
$ cd android4.0/device/softwinner/apollo-mele
$ git status
  # 修改完成的代码
$ git add .
$ git commit -m “some message”
$ git push homlet develop
```

（2）、开发人员 A 在本地修改代码，使用 repo 进行本地提交并推送提交到代码库中。

```
$ cd android4.0
$ repo checkout develop
$ repo status
  # 修改完成的代码
$ repo forall -c git add .
$ repo forall -c git commit -m “some message”
$ repo forall -c git push homlet develop
```

开发人员 A 已将自己的提交推送到公共库中，开发人员 B 要同步开发人员 A 的代码，直接

从代码库进行更新：

（1）、只更新某一个仓库

```
$ cd android4.0/device/softwinner/apollo-mele
$ git checkout develop
$ git pull homlet develop
```

（2）、更新所有的仓库

```
$ cd android4.0
$ repo checkout develop   # 切换到develop分支
$ repo branch    # 查看分支是否切换成功
$ repo forall -c git pull homlet develop  # 将A的提交更新到B本地
```

lichee 部分的开发过程方法一致，这里不再赘述。

5、开发人员比较代码差异

开发人员需要比较修改了哪些地方，有如下几种方法：

（1）、比较 develop 和 apollo-dev 分支，可看到全志发布的 apollo-dev 分支和本地开发分支 develop 之间的区别。

```
$ cd android4.0
$ repo forall -c git diff -a apollo-dev develop

$ cd ../lichee
$ repo forall -c git diff -a apollo-dev develop
```

（2）、比较不同版本之间的区别，假设全志发布了两个版本，要查看新版本修改了什么，直接对两个 tag 进行比较。使用如下命令：

```
$ cd android4.0
$ repo forall -c git diff -a homlet-apollo-android-v0.1
homlet-apollo-android-v0.2

$ cd ../lichee
$ repo forall -c git diff -a homlet-apollo-android-v0.1
homlet-apollo-android-v0.2
```

6、合并全志发布的新版本代码

此过程仍然由负责人来进行操作。假设全志发布了版本 homlet-apollo-android-v0.5，要实现同步，步骤如下：

（1）、负责人从客户自己的公共代码库下载 android 和 lichee 代码,（直接在原来创建 develop 分支时下载的那个目录下即可）。添加远程仓库：

```
$ cd android4.0
$ repo remote add quanzhi ssh://XinWu@221.4.213.95/git_repo/homlet/
  # 将全志服务器上的发布仓库添加为远程仓库

$ cd ../lichee
$ repo remote add quanzhi
ssh://MeLE@221.4.213.95/git_repo/homlet/lichee/
```

（2)、 查看远程仓库是否添加正确

进入android4.0/device/softwinner/apollo-mele目录，使用命令git remote -v 查看远程仓库的地址。

如：quanzhi 后面的地址是：

ssh://MeLE@221.4.213.95/git_repo/homlet/device/softwinner/apollo-mele.git

则表示添加远程仓库成功

进入lichee/linux-3.0 目录，使用命令git remote -v 查看远程仓库的地址。

如：quanzhi 后面的地址是：

ssh://MeLE@221.4.213.95/git_repo/homlet/lichee/linux-3.0.git

则表示添加远程仓库成功。

如地址错误，删除远程仓库重新添加。删除远程仓库的命令为：

```
$ cd android4.0
$ repo remote rm quanzhi

$ cd ../lichee
$ repo remote rm quanzhi
```

（3)、远程仓库添加成功，获取远程仓库更新,同时能获取到全志发布的tag:

```
$ cd android4.0
$ repo status   # 查看是否有修改，确保为干净状态再继续。
$ repo forall -c git fetch quanzhi
$ repo checkout apollo-dev
$ repo branch   # 查看是否全部切换到apollo-dev分支
$ repo forall -c git merge quanzhi/apollo-dev
  # 本地的apollo-dev分支同步更新，和全志发布的apollo-dev分支内容一致

$ cd ../lichee
$ repo forall -c git fetch quanzhi
$ repo status   # 查看是否有修改
$ repo checkout apollo-dev
$ repo branch   # 查看是否全部切换到apollo-dev分支
$ repo forall -c git merge quanzhi/apollo-dev
  # 本地的apollo-dev分支同步更新，和全志发布的apollo-dev分支内容一致
```

（4）、将同步完成的内容推送到自己的公共代码库中

```
$ cd android4.0
$ repo forall -c git push homlet apollo-dev
$ repo forall -c git push homlet homlet-apollo-android-v0.5 # 全志发布的tag也推送到自己的公共代码库中

$ cd ../lichee
$ repo forall -c git push homlet apollo-dev
$ repo forall -c git push homlet homlet-apollo-android-v0.2
```

推送成功后，其他开发人员A 或B在本地使用如下命令可更新本地的apollo-dev分支。

```
$ cd android4.0
$ repo status   # 查看状态，是否有未提交的修改,确保为干净状态再更新
$ repo checkout apollo-dev
$ repo branch
$ repo sync

$ cd ../lichee
$ repo status  # 查看状态，是否有未提交的修改,确保为干净状态后再更新
$ repo checkout apollo-dev
$ repo branch
$ repo sync
```

开发人员本地的apollo-dev分支更新完成，提交历史和全志发布的apollo-dev分支一致。

（5）、将全志发布版本代码合并到自己的开发分支上

此过程为一个整合过程，应由负责人进行操作。

```
$ cd android4.0
$ repo status   # 查看状态，确保为干净状态再合并
$ repo checkout develop  # 切换到自己的开发分支上
$ repo branch
$ repo forall -c git merge apollo-dev

$ cd ../lichee
$ repo status   # 查看状态，确保为干净状态再合并
$ repo checkout develop   # 切换到自己的开发分支上
$ repo branch
$ repo forall -c git merge apollo-dev
```

合并过程中，如提示出现冲突，需要负责人手动解决，耐心修改代码。完成后，使用命令 git add filename 来标记为已解决状态，所有的冲突都解决后，在本地做提交，完成一次合并。

（6）、将合并完成后的开发分支推送到公共代码库

合并完成后，确保合并结果没有问题。则可以推送到公共代码库中。

```
$ cd android4.0
$ repo status   # 查看状态，确保为干净状态
$ repo checkout develop  # 切换到自己的开发分支上
$ repo branch
$ repo forall -c git pull homlet develop  # 推送之前应该先从公共代码库更新一下，以免推送不成功
$ repo forall -c git push homlet develop

$ cd ../lichee
$ repo status   # 查看状态，确保为干净状态
$ repo checkout develop  # 切换到自己的开发分支上
$ repo branch
$ repo forall -c git pull homlet develop  # 推送之前应该先从公共代码库更新一下，以免推送不成功
$ repo forall -c git push homlet develop
```

7、在服务器上开通gitweb服务器

首先要配置 apache 服务器的配置文件：
/etc/httpd/conf/httpd.conf

[root@localhost conf]# pwd
/etc/httpd/conf
[root@localhost conf]# ls
httpd.conf magic
[root@localhost conf]# pwd
/etc/httpd/conf
[root@localhost conf]#

要在最下面加上如下的语句，其他 IP 地址替换成自己服务器的地址。

```
<VirtualHost *:80>
    ServerName 192.168.110.123
    DocumentRoot /var/www/git
        <Directory /var/www/git>
            Options ExecCGI +FollowSymLinks +SymLinksIfOwnerMatch
            AllowOverride All
            order allow,deny
            Allow from all
            AddHandler cgi-script cgi
            DirectoryIndex gitweb.cgi
        </Directory>
</VirtualHost>
```

上面指定了服务器启动脚本的位置以及启动脚本
gitweb.cgi static
[root@localhost git]# pwd
/var/www/git
[root@localhost git]# ls
gitweb.cgi static
[root@localhost git]#

其次是对 gitweb 的配置文件进行修改：
/etc/gitweb.conf
$projectroot = "/home/git/repositories";
指明仓库的位置

[root@localhost repositories]# cd test.git
[root@localhost test.git]# ls
branches description HEAD info refs
config git-daemon-export-ok hooks objects
[root@localhost test.git]#
最后是运行：
service httpd start
ok，可以运行起来了
http://192.168.110.123

可参考网页：http://liubincm.blog.51cto.com/1828652/505731

8、在服务器上开通git协议下载

公共，非授权的只读访问要求我们在 HTTP 协议的基础上使用 Git 协议。主因在于速度。Git 协议更为高效，进而比 HTTP 协议更迅速，所以它能节省很多时间。

重申一下，这一点只适用于非授权、只读的访问。如果在防火墙之外的服务器上，该服务的使用应该局限于公诸于世的项目。假如是在防火墙之内，它也可以用于具有大量参与人员或者主机（长期整合资源或编译的服务器）的只读访问的项目，可以省去为逐一添加 SSH 公钥的麻烦。

第一步：服务器有防火墙，需要为该主机的 9418 端口打个允许通信的洞。在防火墙中添加 9418 端口。

第二步：查看是否能使用 git-daemon（一般情况下：git 安装成功，就可以使用 git-daemon 命令）。

```
[root@localhost /]# git daemon --help
usage: git daemon [--verbose] [--syslog] [--export-all]
           [--timeout=n] [--init-timeout=n] [--max-connections=n]
```

```
           [--strict-paths] [--base-path=path] [--base-path-relaxed]
           [--user-path | --user-path=path]
           [--interpolated-path=path]
           [--reuseaddr] [--detach] [--pid-file=file]
           [--[enable|disable|allow-override|forbid-override]=service]
           [--inetd | [--listen=host_or_ipaddr] [--port=n]
                      [--user=user [--group=group]]
           [directory...]
   #git-daemon命令可以使用
```

第三步：添加 git-daemon 服务，在/etc/init.d 中创建文件 git-daemon，内容如下：

```
#!/bin/bash
. /etc/init.d/functions
. /etc/rc.d/init.d/functions
export PATH=/git-1.7.5.4:$PATH
DAEMON='git-daemon'
ARGS='--base-path=/git_repo --reuseaddr --detach --user=git --group=git
--verbose --export-all /git_repo'
prog=git-daemon

start () {
        echo -n $"Starting ${prog}: "

        # start daemon
        $DAEMON $ARGS &
         RETVAL=$?
        echo
        [ $RETVAL = 0 ] && touch /var/lock/git-daemon
        return $RETVAL
}

stop () {
        # stop daemon
        echo -n $"Stopping ${prog}: "
        killproc $DAEMON
        RETVAL=$?
        echo
        [ $RETVAL = 0 ] && rm -f /var/lock/git-daemon

}
restart() {
        stop
        start
}
```

```
case $1 in
        start)
                start
        ;;
        stop)
                stop
        ;;
        restart)
                restart
        ;;
        status)
                status $DAEMON
                RETVAL=$?
        ;;
        *)

        echo $"Usage: ${prog} {start|stop|restart|status}"
        exit 3
esac
exit $RETVAL
```

第四步：由上诉文件内容可知，需要建立用户 git，将 git 用户的 shell 设置为 git-shell。然后再开启 git-daemon 服务，命令如下：

```
$ /etc/init.d/git-daemon start
```