密级状态：绝密(　　)　　秘密(　　)　　内部(　)　　公开(√)

REPO 镜像服务器搭建和管理_V2.2

（技术部，MID 组）

文件状态： [　] 正在修改 [√] 正式发布	当前版本：	V2.2
	作　　者：	Cody Xie
	完成日期：	2013-12-31
	审　　核：	胡卫国，张小珠
	完成日期：	2013-12-31

福州瑞芯微电子有限公司

Fuzhou　Rockchips　Semiconductor　Co . , Ltd

版 本 历 史

版本号	作者	修改日期	修改说明	备注
V0.9	Cody Xie	2013-7-15	初稿	
V2.0	Cody Xie	2013-12-18	更新 repo 管理说明	
V2.1	Cody Xie	2013-12-24	添加常见问题 FAQ	
V2.2	Cody Xie	2013-12-31	修正部分笔误，添加部分难以理解地方的说明。	

目 录

1 概述

本文档简单描述如何搭建服务器及如何使用 repo，如有问题请与 fae@rock-chips.com 联系，反馈给文档作者完善内容。阅读本文档前，请先理解 git 相关命令及概念。

2 镜像服务器建立

2.1 GITOLITE 搭建

2.1.1 服务器端操作：

a) 创建 git 账户：

```
sudo adduser --system --shell /bin/bash --group git
sudo passwd git
```

b) 下载源码： git clone https://github.com/sitaramc/gitolite.git

c) 以“git”账户登录 Linux 系统/ 或者直接：

```
su - git
```

d) 确保“~/.ssh/authorized_keys”为空或者不存在。

e) 拷贝服务器管理员的公钥到“$HOME/YourName.pub”。

f) 执行：

```
mkdir -p $HOME/bin
```

g) 执行下列命令安装，不同版本安装方法不同，请参考源码中的文档：

```
gitolite/install -to $HOME/bin
```

h) 执行：

```
$HOME/bin/gitolite setup -pk YourName.pub （管理员的公钥）
```

2.1.2 给 GIT 用户添加下载权限

如果没有就在$HOME 帐号下建一个.shh 隐藏目录，并且把管理员帐号下面的 id_rsa 复制到刚建立的.ssh 下

```
cp /home/xx/.ssh/id_rda /home/git/.shh/ -rf
```

切到 ROOT 帐号下

```
sudo su
```

更改组权限

```
chown git:git id_rsa*
```

退出 ROOT

```
exit
```

登录 git 帐号

```
su - git
```

改权限

```
sudo chmod 600 .ssh/id_rsa*
```

2.1.3 客户端操作：

本节介绍如何使用 gitolite-admin 来管理服务器权限。相关概念资料，请搜索 progit 文档阅读理解。

a) 克隆 gitolite 管理仓库：

```
git  clone  ssh://git@host/gitolite-admin.git
```

说明:/git@host

host 表示服务器的 IP 地址或域名

例如，服务器的 IP 是 192.168.1.200

那 git 管理仓库是 git clone ssh://git@192.168.1.200/gitolite-admin.git

b) 添加用户公钥

```
cp  username.pub  keydir/username.pub
```

c) 添加管理员用户

```
vi  conf/gitolite.conf

@admin  =  admin1  admin2  admin3

repo  gitolite-admin

RW+     =     @admin
```

d) 提交生效

每次修改 conf/gitolite.conf 内容都要提交才会生效

git add conf/gitolite.conf

git commit “add usergroup”

git push origin master

2.2 REPO 镜像

2.2.1 初始化镜像仓库

2.2.1.1 服务器端操作：

获取原始 repo 工具：

su - git

git clone ssh://git@www.rockchip.com.cn/repo/rk/tools/repo

初始化镜像仓库：

cd repositroies

mkdir repo ; cd repo （repo 目录名字可以换，后面相应目录名字都得换）

~/repo/repo init --mirror --repo-url ssh://git@www.rockchip.com.cn/repo/rk/tools/repo -u ssh://git@www.rockchip.com.cn/repo/rk/platform/manifest -b android-4.4 -m rk3188_android4.4.1.xml

.repo/repo/repo sync

备注：下载可能遇到问题：

下载代码失败有几个原因

一个没有.bashrc 文件

需要从其它帐户下 COPY 过来并且权限改成自己的组，并且要用 keychain 管理

具体使用方法如下：

1. 安装 keychain 软件包：

$sudo aptitude install keychain

2. 配置使用密钥：

$vim ~/.bashrc

增加下面这行：

eval `keychain --eval ~/.ssh/id_rsa`

其中，id_rsa 是私钥文件名称。

执行 source ~/.bashrc 后，成功会有如下提示：

```
* Found existing ssh-agent (22251)
* Adding 1 ssh key(s)...
Enter passphrase for /home/git/.ssh/id_rsa:
* Error: Problem adding; giving up
KeyChain 2.6.8; http://www.gentoo.org/proj/en/keychain/
Copyright 2002-2004 Gentoo Foundation; Distributed under the GPL

* Initializing /home/git/.keychain/lushen-sh file...
* Initializing /home/git/.keychain/lushen-csh file...
* Initializing /home/git/.keychain/lushen-fish file...
* Starting ssh-agent
* Warning: /home/git/.ssh/id_rsa.pub missing; can't tell if /home/git/.ssh/id_rsa is loaded
```

创建仓库组权限：

```
.repo/repo/repo list -n > projects.txt
sed -i 's/^repo//' projects.txt （repo 指的是前面创建文件夹的名字，list 会把那个加上）
sed -i 's/^/@rk\ = \ repo\//' projects.txt
```

2.2.1.2 客户端操作：

将 projects.txt 内容拷贝复制到 gitolite-admin/conf/gitolite.conf 中：

```
cd path_to_gitolite-admin （指的是 gitolite-admin 的路径）
echo "" >> conf/gitolite.conf (添加个空行)
cat projects.txt >> conf/gitolite.conf （ps：之前有个笔误写成 echo，请修改）
```

添加组权限，在文件最后（所有“@rk = repo/…”后面）添加如下内容：

```
vi conf/gitolite.conf
    @usergroup = user1 user2 user3
    @rk = repo/platform/manifest
    repo @rk
        R   = @usergroup
        RW+ = @admin
```

2.3 MANIFEST 创建

添加 manifest 工程：

```
vi  conf/gitolite.conf
    repo  repo/platform/manifest
        R     =  @usergroup
        Rw    =  @admin
cd  ../
git  clone  ssh://git@host/repo/platform/manifest.git
cd  manifest
```

拷贝一份原始 manifest.xml 并上传：

```
cp  original.xml  rk3188_android-4.4.xml(可以是其他名字)
```

备注：original.xml 指的是同步下来的 xml， /home/git/repositories/repo/.repo/manifests 目录下面的 xml。

复制完成后，打开改名后的 xml 文件，看一下最上面几行：如果发现下面的红色部分路径是../..的话，请修改为 .. 退出并保存。如果不修改将会导致客户端无法同步代码。fetch 指向的相对地址是 init 时使用-u 后的参数指定的 manifest 仓库地址的相对路径。

举个例子说明下为什么这么改：假如使用的 manifest 工程路径是/home/git/repositories/repo/platform/manifest.git，其他工程的路径，如/home/git/repositories/repo/rk/device/rockchip/rksdk.git，这个工程是 rk 仓库，路径是 device/rockchip/rksdk，rk 仓库的路径就是 manifest 的上级的上级的 rk 目录。

```
<manifest>
  <remote   name="aosp"
            fetch="../.."
            sync-c="true"  />
  <remote   name="rk"
            fetch="../.."
            sync-c="true"  />
  <default  revision="refs/tags/android-4.4.2_r1"
            remote="aosp"
            sync-j="4"
```

sync-c="true" />

git add rk3188_android-4.4.xml

git commit -m “add initial manifest for android-4.4”

git push origin master:android-4.4(分支名，可以是其他名字)

2.4 镜像服务器测试

在客户端，先下载 repo 工具，如前所述，按照如下命令测试：

git clone ssh://git@host/repo/rk/tools/repo （使用搭建的服务器里的 repo 工具，host 是搭建的服务器域名或 ip）

mkdir test_repo; cd test_repo

../repo/repo init --repo-url=ssh://git@host/repo/rk/tools/repo -u ssh://git@host/repo/platform/manifest -b android-4.4 -m rk3188_android-4.4.xml

备注：

--repo-url=ssh://git@host/repo/rk/tools/repo —— 这个路径是新搭建的服务器的 repo 的路径。

-u ssh://git@host/repo/platform/manifest —— 这个路径是新搭建的服务器的 manifest 工程路径。

-b android-4.4 —— 这个是 manifest 工程中提交的分支名

-m rk3188_android-4.4.xml —— 这个是使用的 xml 文件名，是前面提交的名字，可以根据产品来命名，可以提交多个 manifest，修改里面的分支名，来管理不同产品。

如有提示权限问题，依照前述 gitolite 添加权限即可。

3 服务器管理建议

a) 在产品有必要做修改的工程创建产品分支，如下所示，rk30/mid/4.4_r1/release 是 rk 的分支，common_product 是公司通用修改提交的分支，product_a 是具体产品。

b) 出产品固件，必须保存一份当前代码的版本信息的 manifest.xml 文件，一般在开发者的代码工程做，并将此 manifest 提交到服务器做保存。

repo manifest -r -o rk3188_kitkat_rel_vX.XX(rk 发布版本号)_product-name(产品名)_131218(日期).xml

c) 每次更新 sync 前必须保存一份 manifest.xml 文件，这个必须确实执行，不然不好排查问题。

repo manifest -r -o rk3188_kitkat_rel_vX.XX(rk 发布版本号)_product-name(产品名)_131218(日期).xml

4 REPO 命令使用

4.1 Repo 介绍

Repo is a tool that we built on top of Git. Repo helps us manage the many Git repositories, does the uploads to our revision control system, and automates parts of the Android development workflow. Repo is not meant to replace Git, only to make it easier to work with Git in the context of Android. The repo command is an executable Python script that you can put anywhere in your path.

repo 使用简单手册：http://source.android.com/source/version-control.html

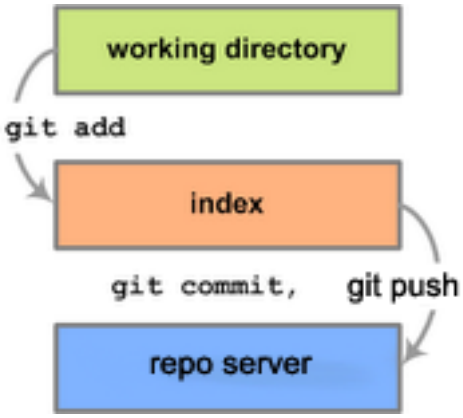
repo 代码工程地址：https://code.google.com/p/git-repo/

4.2 repo 基本命令

4.2.1 基本工作流程：

1. 使用 repo start 开始一个分支开发。
2. 编辑代码。
3. 使用 git add 缓存更改。
4. 使用 git commit 提交更改。

5. 使用 git push 仓库名 本地分支名:远程分支名 上传本地代码到服务器。



4.2.2 同步更新代码

使用 sync 命令，执行的是 git fetch --update-head-ok 命令。这个可以更新当前分支，并自动将你的本地修改的提交 rebase 到最上。注意：如果当前分支有本地提交或者与服务器有冲突，或者不是用 repo start 建立的分支，repo 可能自动切换到*<no branch> 。因此，请尽量用 repo start 建立开发分支。

同步整个工程

repo sync

同步指定工程：

repo sync project1 project2…

解决同步 sync 冲突问题：

sync 有可能出现提示冲突，并提示使用“git rebase --abort”，“git rebase --continue”，“git rebase --skip”等 3 个命令解决冲突。

如果您知道如何解决冲突，使用如下命令：

a) 手工编辑 conflicts（即提示冲突）的文件，并使用：
   vim some_comflict_file
   git add some_conflict_file
b) 解决所有冲突后，执行：
   git commit
   git rebase --continue

如果不知道如何解决，请先使用如下命令：

a) 退出 rebase 状态：

```
git rebase --abort
```

b) 手工去相应工程用 git 命令合并。

4.2.3 创建使用开发分支

创建分支：

```
repo start branch_name
```

检查分支是否创建：

```
repo status
```

查看当前所有工程分支：

```
repo branch
```

git 命令中关于分支的操作：

检出新分支：

```
git checkout -b branch_name
```

检出基于远程分支的跟踪分支：

```
git checkout -b branch_name -t repository/remote_branch_name
```

删除分支：

```
git branch -d branch_name / git branch -D branch_name
```

比较分支提交差别，分支名也可以是 HEAD，commitid 等。

```
git log branch_name1..branch_name2
```

4.2.4 查看本地修改

查看所有工程本地修改：

```
repo diff
```

如下所示，显示工程名字以及工程本地工作区和暂存区的差别：

```
xkd@rk-mid:~/rk3188_kitkat$ .repo/repo/repo diff

project hardware/rk29/sensor/
diff --git a/st/MmaSensor.cpp b/st/MmaSensor.cpp
index 65c8b0f..875fbcc 100755
--- a/st/MmaSensor.cpp
+++ b/st/MmaSensor.cpp
@@ -541,11 +541,11 @@ void MmaSensor::processEvent(int code, int value)
     switch (code) {
         case EVENT_TYPE_ACCEL_X:
             mPendingMask |= 1<<Accelerometer;
-            mPendingEvents[Accelerometer].acceleration.x = value * ACCELERATION_RATIO_ANDROID_TO_HW;
+            mPendingEvents[Accelerometer].acceleration.y = value * ACCELERATION_RATIO_ANDROID_TO_HW;
             break;
         case EVENT_TYPE_ACCEL_Y:
             mPendingMask |= 1<<Accelerometer;
-            mPendingEvents[Accelerometer].acceleration.y = value * ACCELERATION_RATIO_ANDROID_TO_HW;
+            mPendingEvents[Accelerometer].acceleration.x = (-1) * value * ACCELERATION_RATIO_ANDROID_TO_HW;
             break;
         case EVENT_TYPE_ACCEL_Z:
             mPendingMask |= 1<<Accelerometer;
```

查看工程状态：

repo status

如下所示，显示当前本地内容，“-m”指修改未提交，“--”指不在版本库管理的文件。

```
xkd@rk-mid:~/rk3188_kitkat$ .repo/repo/repo status
project device/common/                          branch rk30/mid/4.4_r1/release
project device/rockchip/rksdk/                  (*** NO BRANCH ***)
 --     parameter/parameter-crypto-768m
project external/wlan_loader/                   branch rk29/mid/4.0.1_r1/release
project hardware/rk29/hwcomposer_rga/           (*** NO BRANCH ***)
 --     rm_cpp_current.sh
project hardware/rk29/libgralloc_ump/           (*** NO BRANCH ***)
 --     rm_cpp_current.sh
project hardware/rk29/sensor/                   (*** NO BRANCH ***)
 -m     st/MmaSensor.cpp
project kernel/                                 branch develop-3.10
 --     defconfig
 --     drivers/video/logo/logo_android_bmp.c
 --     drivers/video/logo/logo_sunset_bmp.c
 --     include/linux/version.h
 --     kernel.img
 --     scripts/bmptologo
 --     scripts/kconfig/lex.zconf.c
```

查看是否有未合并的分支：

repo overview

如下所示，显示出工程名字，未合并提交号。

```
xkd@rk-mid:~/rk3188_kitkat$ .repo/repo/repo overview
Deprecated. See repo info -o.
Projects Overview

project external/wlan_loader/
* rk29/mid/4.0.1_r1/release         ( 1 commit , Tue Dec 17 15:27:27 2013 +0800)
                                      - 13564690 add something.txt
```

4.2.5 暂存工作区内容

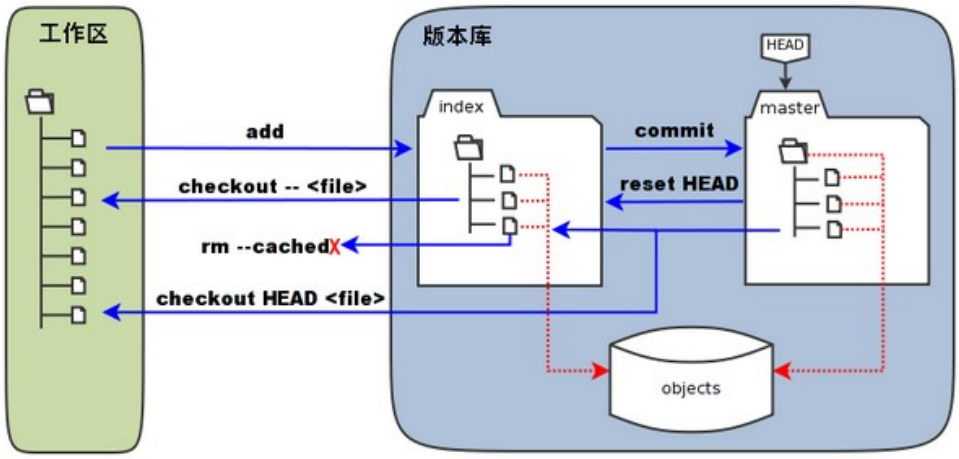
这里介绍几种方法临时保存修改中的内容，首先，先看下什么是工作区，暂存区，版本库。这里引用网络的一段说明：

工作区：我们会想当然的认为，当前仓库所在目录就是我们的工作区，其实这是不完全正确的。在当前仓库中，新增，更改，删除文件这些动作，都发生在工作区里面。

暂存区：英文叫 stage, 或 index。在版本库.git）目录下，有一个 index 文件。它实际上就是一个包含文件索引的目录树，像是一个虚拟的工作区。在这个虚拟工作区的目录树中，记录了文件名、文件的状态信息（时间戳、文件长度等），文件的内容并不存储其中，而是保存在 Git 对象库（.git/objects）中，文件索引建立了文件和对象库中对象实体之间的对应。如果当前仓库，有文件更新，并且使用 git

add 命令，那么这些更新就会出现在暂存区中。

版本库：当前仓库下，如果没有任何的提交，那么版本库就是对应上次提交后的内容。下面这个图展示了工作区、版本库中的暂存区和版本库之间的关系。



图中左侧为工作区，右侧为版本库。在版本库中标记为 "index" 的区域是暂存区（stage, index），标记为 "master" 的是 master 分支所代表的目录树。

图中我们可以看出此时 "HEAD" 实际是指向 master 分支的一个“游标”。所以图示的命令中

出现 HEAD 的地方可以用 master 来替换。

图中的 objects 标识的区域为 Git 的对象库，实际位于 ".git/objects" 目录下，里面包含了创建的各种对象及内容。

当对工作区修改（或新增）的文件执行 "git add" 命令时，暂存区的目录树被更新，同时工作区修改（或新增）的文件内容被写入到对象库中的一个新的对象中，而该对象的 ID 被记录在暂存区的文件索引中。

当执行提交操作（git commit）时，暂存区的目录树写到版本库（对象库）中，master 分支会做相应的更新。即 master 指向的目录树就是提交时暂存区的目录树。

当执行 "git reset HEAD" 命令时，暂存区的目录树会被重写，被 master 分支指向的目录树所替换，但是工作区不受影响。

当执行 "git rm --cached <file>" 命令时，会直接从暂存区删除文件，工作区则不做出改变。

当执行 "git checkout ." 或者 "git checkout -- <file>" 命令时，会用暂存区全部或指定的文件替换工作区的文件。这个操作很危险，会清除工作区中未添加到暂存区的改动。

当执行 "git checkout HEAD ." 或者 "git checkout HEAD <file>" 命令时，会用 HEAD 指向的 master 分支中的全部或者部分文件替换暂存区和以及工作区中的文件。

4.2.5.1 方法 1：提交工作区到暂存区

```
git add somefile
```

4.2.5.2 方法 2：缓存当前工作区内容

```
git stash
git stash list
git stash pop apply
```

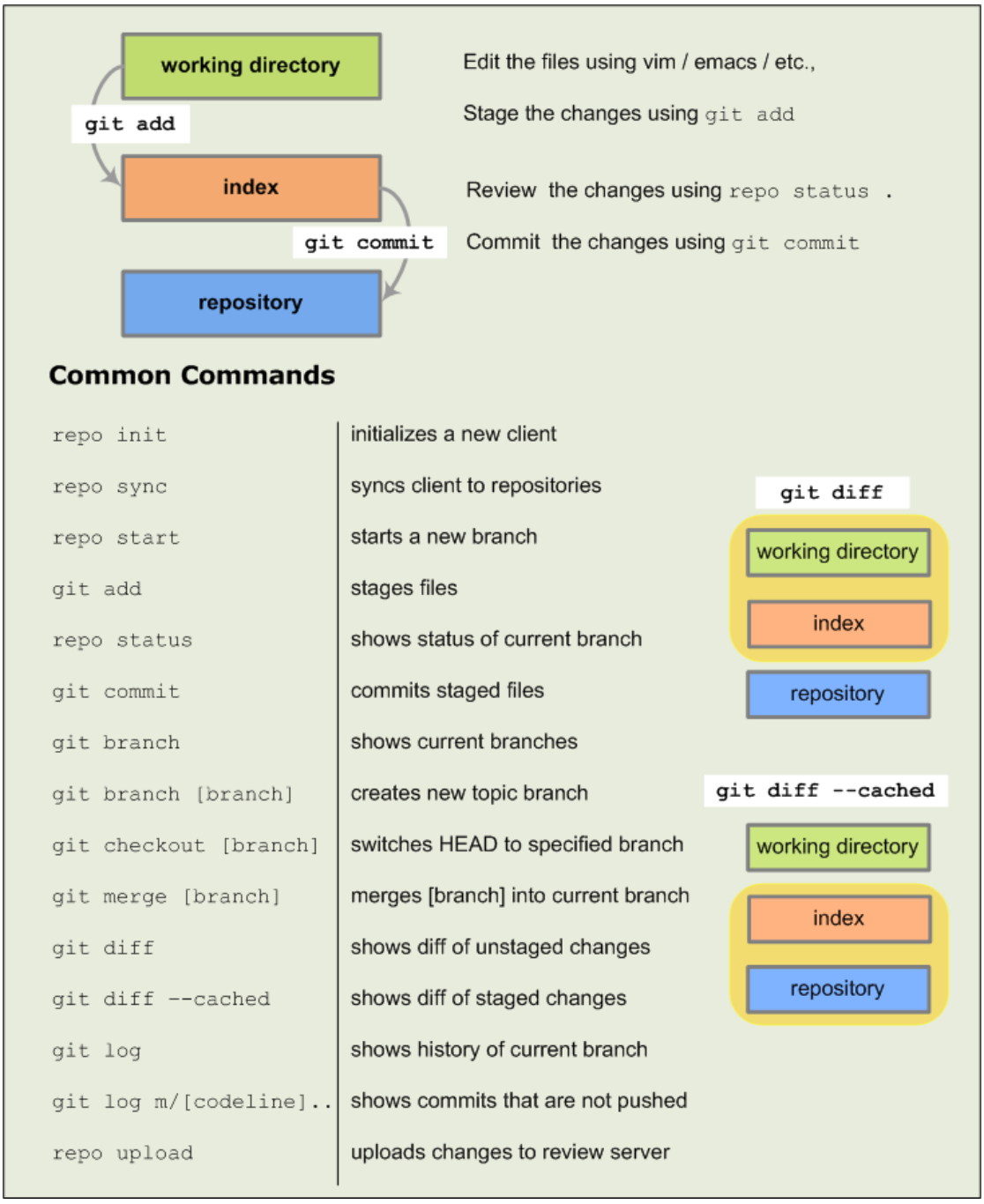
4.2.5.3 方法 3：提交并新建临时分支

```
git commit -am “temp commit”

git branch temp_branch

git checkout new_develop_branch
```

4.2.6 常用命令图表



4.3 repo 小技巧

4.3.1 采用 repo 方式，那么多 git，怎么知道一次更新了哪些?

```
repo forall -c 'git fetch rk'
```

repo forall -c 'pwd; git log HEAD..remotes/m/android-4.4'

4.3.2 怎样直观的看到改了哪些文件?

repo forall -c 'pwd; git log --stat HEAD..remotes/m/android-4.4'

4.3.3 很久没有更新，怎样查看更新内容，合并需要的内容?

初略看下一段时间内的提交数，可以

.repo/repo/repo forall -c "git log --since=2013-09-11"

看这个会以某个时间起始点去查看

4.3.4 如果是一个 git，我用 git log; (查看当前) git pull (更新); git log (查看最新) 就能看到更新了哪些?

对应的操作是：

.repo/repo/repo manifest -r -o xxx-version.xml(保存当前版本信息)

.repo/repo/repo sync -n (相当于 git fetch)

.repo/repo/repo forall -p -c "git log HEAD..remotes/m/android-4.4" (相当于 git log)

.repo/repo/repo sync (相当于 git fetch && git merge 或 git merge)

4.3.5 查看 A 版本和 B 版本的差异?

前提是要备份当时版本的 manifest.xml，要有 A 和 B 的 xml

.repo/repo/repo manifest -r -o xxx-version.xml(保存当前版本信息)

通过如下命令，可以比较版本间的差异

cp A.xml .repo/manifest.xml

.repo/repo/repo sync -l

.repo/repo/repo forall -p -c "git tag A"

```
cp B.xml .repo/manifest.xml

.repo/repo/repo sync -l

.repo/repo/repo forall -p -c "git tag B"


.repo/repo/repo forall -p -c "git diff A B"
```

5 常见问题 FAQ

1. Q：初始化 repo 提示：fatal: git 1.7.2 or later required 怎么解决？

   A：请升级 git，使用 ppa 的源

   增加 ppa

   ```
   sudo apt-add-repository ppa:git-core/ppa
   sudo apt-get update
   sudo apt-get install git
   ```

   如果本地已经安装过 Git，可以使用升级命令：

   ```
   sudo apt-get dist-upgrade
   ```

2. Q：error: could not verify the tag 'v1.12.7' 怎么解决？

   A：请更新 repo 工具或者使用 RK 发布的 repo 工具。

   - 到 repo 的目录，使用 git pull 命令。
   - git clone ssh://git@www.rockchip.com.cn/repo/rk/tools/repo.git

3. Q：repo sync 时不断提示”Enter passphrase for key ‘/root/.ssh/id_rsa’: ”怎么解决？

   A：请参考 SDK 发布说明文档“使用 key-chain 管理密钥”一节内容，配置 keychain 工具，并重新登陆 shell 或直接“source ~/.bashrc”。请注意把公钥和私钥存放在同一路径。

4. Q：更新提示输入密码“git@www.rockchip.com.cn’s password:”，怎么解决？

   A：请确认

   - 是否已申请开通权限。
   - 本地使用的私钥“~/.ssh/id_rsa”是否正确。
   - 请确认在提示“Enter passphrase for key”时，输入是否正确。