参考博客:

```
https://blog.csdn.net/z69183787/article/details/53336948
```

Git tag

一.Tag用来做什么?

Tag即标签，用以给项目仓储打标签，通常用作里程碑标识，以方便项目进度、发布版本管理及规划。

Git tag可以用来创建标签，列出标签表，删除标签及用以验证带有GPG签名的带标签的项目。

二.git tag常见用法

1.创建Tag

```
git tag v1.0
```

标记当前位置为tag，tag名为v1.0

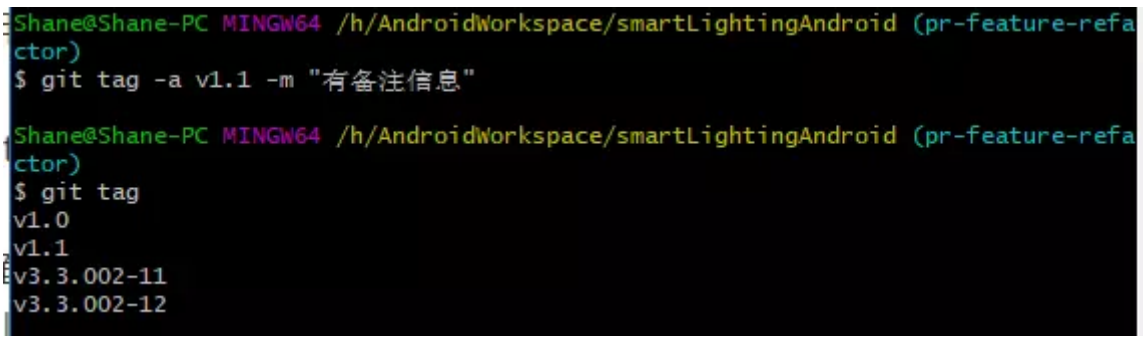
```
git tag -m "Say something" v1.0
```

标记当前位置为tag，tag描述为”Say something”，tag名为v1.0

上面创建一个名为`v1.0`的tag。使用`git tag`命令可以看到新增加的tag

还可以加上`-a`参数来创建一个带备注的tag，备注信息由`-m`指定。如果你未传入`-m`则创建过程系统会自动为你打开编辑器让你填写备注信息。

```
git tag -a tagName -m "my tag"
```



2.列出tag表

git tag (-l)

列出tag表，只显示标签名

git tag -ln

列出tag表，并显示其message（描述）

3.删除tag

git tag -d [tag name]

删除某个tag

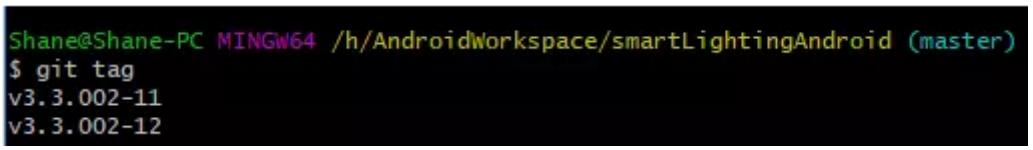
4.验证tag

git tag -v [tag name]

验证某个tag的GPG签名

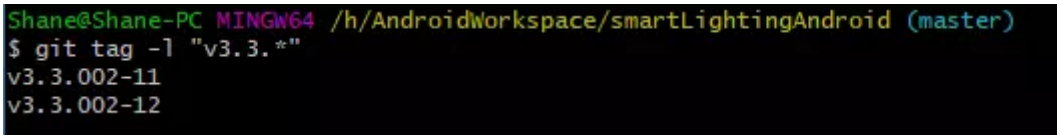
5.列出已有的tag

```
1 git tag
```



显示tag

6.加上-l命令可以使用通配符来过滤tag

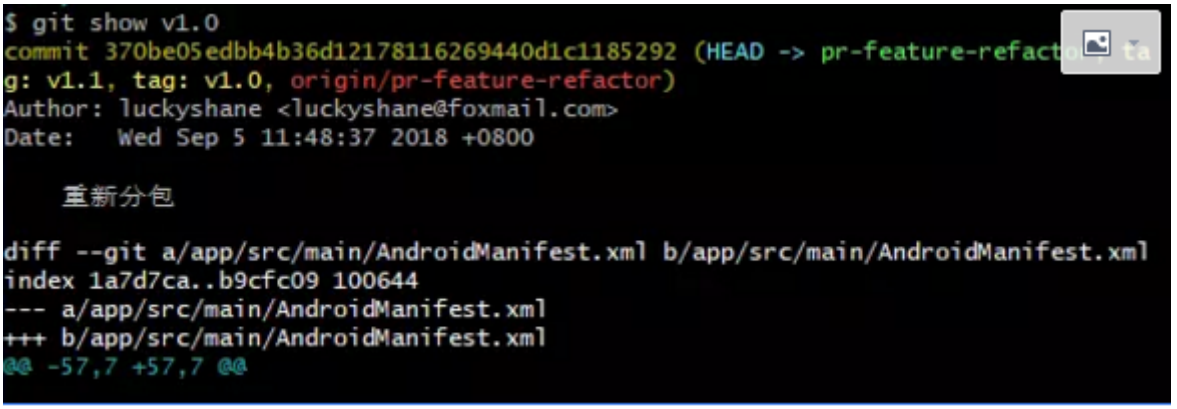


7.查看tag详细信息

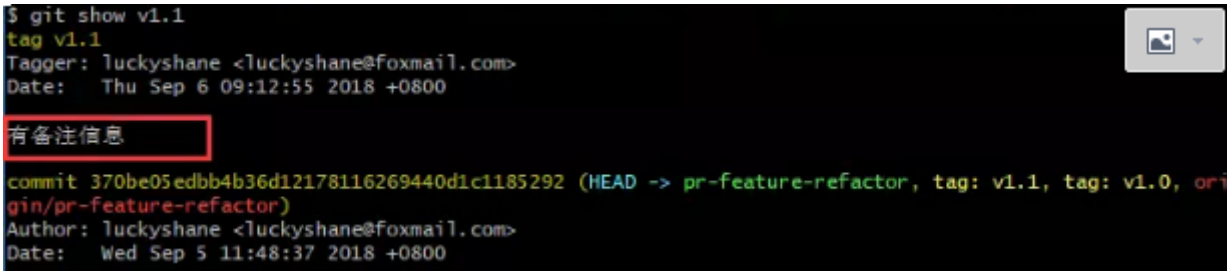
git show命令可以查看tag的详细信息，包括commit号等。

```
1 git show tagName
```

查看v1.0tag的详细信息



查看带备注的v1.1的详细信息

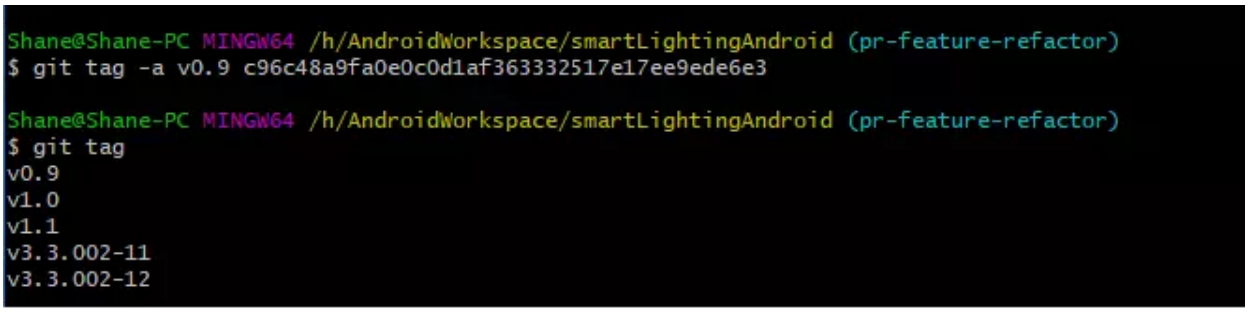


tag最重要的是有git commit号，后期我们可以根据这个commit号来回溯代码。

8.给指定的某个commit号加tag

打tag不必要在head之上，也可在之前的版本上打，这需要你知道某个提交对象的校验和（通过git log获取，取校验和的前几位数字即可）。

```
1 git tag -a v1.2 9fceb02 -m "my tag"
```



9.将tag同步到远程服务器

同提交代码后，使用`git push`来推送到远程服务器一样，`tag`也需要进行推送才能到远端服务器。

使用`git push origin [tagName]`推送单个分支。

```
1 git push origin v1.0
```

推送本地所有tag，使用`git push origin --tags`。

10.切换到某个tag

跟分支一样，可以直接切换到某个tag去。这个时候不位于任何分支，处于游离状态，可以考虑基于这个tag创建一个分支。

```
Shane@Shane-PC MINGW64 /h/AndroidWorkspace/smartLightingAndroid (master)
$ git tag
v0.9
v1.0
v1.1
v3.3.002-11
v3.3.002-12

Shane@Shane-PC MINGW64 /h/AndroidWorkspace/smartLightingAndroid (master)
$ git checkout v0.9
Note: checking out 'v0.9'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by performing another checkout.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -b with the checkout command again. Example:

  git checkout -b <new-branch-name>

HEAD is now at c96c48a 移除冗余文件
```

11.删除某个tag

- 本地删除

```
1 git tag -d v0.1.2
```

```
Shane@Shane-PC MINGW64 /h/AndroidWorkspace/smartLightingAndroid (master)
$ git tag -d v0.9
Deleted tag 'v0.9' (was 5b1f879)
```

- 远端删除

git push origin :refs/tags/<tagName>

```
1 git push origin :refs/tags/v0.1.2
```

GPG

GPG是加密软件，可以使用GPG生成的公钥在网上安全的传播你的文件、代码。

为什么说安全的？以Google所开发的repo为例，repo即采用GPG验证的方式，每个里程碑tag都带有GPG加密验证，假如在里程碑v1.12.3处你想要做修改，修改完后将这个tag删除，然后又创建同名tag指向你的修改点，这必然是可以的。但是，在你再次clone你修改后的项目时，你会发现，你对此里程碑tag的改变不被认可，验证失败，导致你的修改在这里无法正常实现。这就是GPG验证的作用，这样就能够保证项目作者（私钥持有者）所制定的里程碑别人将无法修改。那么，就可以说，作者的代码是安全传播的。

为什么会有这种需求？一个项目从开发到发布，再到后期的更新迭代，一定会存在若干的稳定版本与开发版本（存在不稳定因素）。作为项目发起者、持有者，有权定义他（们）所认可的稳定版本，这个稳定版本，将不允许其他开发者进行改动。还以Google的repo项目为例，项目所有者定义项目开发过程中的点A为稳定版v1.12.3，那么用户在下载v1.12.3版本后，使用的肯定是A点所生成的项目、产品，就算其他开发者能够在本地对v1.12.3进行重新指定，指定到他们修改后的B点，但是最终修改后的版本给用户用的时候，会出现GPG签名验证不通过的问题，也就是说这样的修改是不生效的。

可能还是不太好理解，说一个我曾经遇过的坑，repo在某个版本有个小错误，我在这个版本修复错误后，提交到内网的服务器给同事用（我重新定义这个版本的指向），最后发现同事在更新项目后，出现版本GPG签名验证错误，不能正常使用。

说了这么多，希望大家能够理解GPG签名验证的意义所在。下边就说一说，如何在你的项目中使用这种方式。

生成GPG Key

gpg *--gen-key*

然后根据提示选择你要的签名

```
1. gpg (GnuPG) 1.4.11; Copyright (C) 2010 Free Software Foundation, Inc.
2. This is free software: you are free to change and redistribute it.
3. There is NO WARRANTY, to the extent permitted by law.
4.
5. 请选择您要使用的密钥种类：
6.    (1) RSA and RSA (default)
7.    (2) DSA and Elgamal
8.    (3) DSA (仅用于签名)
9.    (4) RSA (仅用于签名)
10. 您的选择?
```

选择加密种类

```
1. RSA 密钥长度应在 1024 位与 4096 位之间。
2. 您想要用多大的密钥尺寸？(2048)
3.
4. 请设定这把密钥的有效期限。
```

5. 0 = 密钥永不过期
6. <n> = 密钥在 n 天后过期
7. <n>w = 密钥在 n 周后过期
8. <n>m = 密钥在 n 月后过期
9. 密钥的有效期限是？(0)

选择密钥有效期

对以上信息进行确认，然后输入身份，最终确认，等待生成GPG Key。

1. 您需要一个用户标识来辨识您的密钥；本软件会用真实姓名、注释和电子邮件地址组合
2. 成用户标识，如下所示：
3. “Heinrich Heine (Der Dichter) <heinrichh@duesseldorf.de>”
4.
5. 真实姓名： Keven Liu
6. 电子邮件地址： airk908@gmail.com
7. 注释： GPG key for Keven
8. 您选定了这个用户标识：
9. “Keven Liu (GPG key for Keven) <airk908@gmail.com>”
10.
11. 更改姓名(N)、注释(C)、电子邮件地址(E)或确定(O)/退出(Q)？ O
12. 您需要一个密码来保护您的私钥。
13.
14. 我们需要生成大量的随机字节。这个时候您可以多做些琐事(像是敲打键盘、移动
15. 鼠标、读写硬盘之类的)，这会让随机数字发生器有更好的机会获得足够的熵数。

生成过程中可能出现类似提示：

1. 随机字节不够多。请再做一些其他的琐事，以使操作系统能搜集到更多的熵！
2. (还需要174字节)

疯狂的敲打键盘吧，不过看好还需要多少，悠着点。

稍等片刻，GPG Key就能生成好了，验证一下是否生成成功：

gpg --list-keys

如果输出类似信息就代表属于你的GPG Key生成成功了：

```
1 /home/keven/.gnupg/pubring.gpg
```

```
1 pub 2048R/AF26C87F 2013-09-30
2 uid Keven Liu (Gpg key for Keven Liu<airk908@gmail.com>) <airk908@gmail.c
om>
```

```
3  sub 2048R/C9A00F19 2013-09-30
```

/home/keven/.gnupg/pubring.gpg
上边显示的是公钥，顺便也看一下与之匹配的私钥生成如何：
gpg --list-secret-keys
如果成功，会显示类似信息，不过文件位置应该是/home/keven/.gnupg/pubring.gpg（keven是我的用户名，你的机器上该是你的）

使用GPG加密你的tag

git tag -s "My tag message" v1.0
是的，将-m 换做-s就是加密签名了
不过，好像很多人会出现错误，比如：

1. gpg: WARNING: using insecure memory!
2. gpg: please see http://www.gnupg.org/faq.html for more information
3. gpg: skipped `Keven Liu <airk908@gmail.com>': secret key not available
4. gpg: signing failed: secret key not available
5. error: gpg failed to sign the tag
6. fatal: unable to sign the tag

这个错误一Google的话都是教你怎么生成一个GPG Key，其实不然，git tag有一条help这样写到：
-u, --local-user <key-id> 使用另一个Key签名此tag
所以，这个错误可以这样解决：
git tag -u "Keven Liu" -s "My tag message" v1.0
这样，你的tag就用刚刚生成的GPG Key签名了。可以查看一下项目的tag验证信息：
git tag -v v1.0
终端输出：

1. object c88d710635a97e6a634d2a1b25e3eba2f8a3574e
2. type commit
3. tag v1.0
4. tagger Keven Liu <airk908@gmail.com> 1380552288 +0800
5.
6. My first tag with gpg key of my own
7. gpg: 于 2013年09月30日 星期一 22时45分05秒 CST 创建的签名，使用 RSA，钥匙号 AF36C27F

8. gpg: 完好的签名，来自于“Keven Liu (Gpg key for Keven Liu<airk908@gmail.com>) <airk908@gmail.com>”

至此，带有GPG签名验证的tag就做好了，其他开发者将无法修改你所签名的这个tag。