`git submodule` 命令用于初始化，更新或检查子模块。

使用语法

```
git submodule [--quiet] add [<options>] [--] <repository> [<path>]
git submodule [--quiet] status [--cached] [--recursive] [--] [<path>…]
git submodule [--quiet] init [--] [<path>…]
git submodule [--quiet] deinit [-f|--force] (--all|[--] <path>…)
git submodule [--quiet] update [<options>] [--] [<path>…]
git submodule [--quiet] summary [<options>] [--] [<path>…]
git submodule [--quiet] foreach [--recursive] <command>
git submodule [--quiet] sync [--recursive] [--] [<path>…]
git submodule [--quiet] absorbgitdirs [--] [<path>…]
```

使用场景

基于公司的项目会越来越多，常常需要提取一个公共的类库提供给多个项目使用，但是这个 library 怎么和 git 在一起方便管理呢？

我们需要解决下面几个问题：

- 如何在 git 项目中导入 library 库?
- library 库在其他的项目中被修改了可以更新到远程的代码库中?
- 其他项目如何获取到 library 库最新的提交?
- 如何在 clone 的时候能够自动导入 library 库?

解决以上问题，可以考虑使用 git 的 submodule 来解决。

submodule 是什么?

`git submodule` 是一个很好的多项目使用共同类库的工具，它允许类库项目做为 repository,子项目做为一个单独的 git 项目存在父项目中，子项目可以有自己的独立的 `commit`，`push`，`pull`。而父项目以 Submodule 的形式包含子项目，父项目可以指定子项目 header，父项目中会的提交信息包含 Submodule 的信息，再 clone 父项目的时候可以把 Submodule 初始化。

示例

以下是一些示例 -

1.添加子模块

在本例中，我们将会添加一个名为 “DbConnector” 的库。

```
$ git submodule add http://github.com/chaconinc/DbConnector
Cloning into 'DbConnector'...
remote: Counting objects: 11, done.
remote: Compressing objects: 100% (10/10), done.
remote: Total 11 (delta 0), reused 11 (delta 0)
Unpacking objects: 100% (11/11), done.
Checking connectivity... done.
```

默认情况下，子模块会将子项目放到一个与仓库同名的目录中，本例中是“DbConnector”。 如果你想要放到其他地方，那么可以在命令结尾添加一个不同的路径。

如果这时运行 `git status`，你会注意到几个东西。

```
$ git status
On branch master
Your branch is up-to-date with 'origin/master'.

Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    new file:   .gitmodules
    new file:   DbConnector
```

首先应当注意到新的 `.gitmodules` 文件。 该置文件保存了项目 URL 与已经拉取的本地目录之间的映射：

```
$ cat .gitmodules
[submodule "DbConnector"]
    path = DbConnector
    url = http://github.com/chaconinc/DbConnector
```

如果有多个子模块，该文件中就会有多条记录。 要重点注意的是，该文件也像 `.gitignore` 文件一样受到(通过)版本控制。 它会和该项目的其他部分一同被拉取推送。 这就是克隆该项目的人知道去哪获得子模块的原因。

在 `git status` 输出中列出的另一个是项目文件夹记录。 如果你运行 `git diff`，会看到类似下面的信息：

```
$ git diff --cached DbConnector
diff --git a/DbConnector b/DbConnector
new file mode 160000
index 0000000..c3f01dc
--- /dev/null
+++ b/DbConnector
```

```
@@ -0,0 +1 @@
+Subproject commit c3f01dc8862123d317dd46284b05b6892c7b29bc
```

虽然 *DbConnector* 是工作目录中的一个子目录，但 Git 还是会将它视作一个子模块。当你不在那个目录中时，Git 并不会跟踪它的内容， 而是将它看作该仓库中的一个特殊提交。

如果你想看到更漂亮的差异输出，可以给 `git diff` 传递 `--submodule` 选项。

```
$ git diff --cached --submodule
diff --git a/.gitmodules b/.gitmodules
new file mode 100644
index 0000000..71fc376
--- /dev/null
+++ b/.gitmodules
@@ -0,0 +1,3 @@
+[submodule "DbConnector"]
+       path = DbConnector
+       url = http://github.com/chaconinc/DbConnector
Submodule DbConnector 0000000...c3f01dc (new submodule)
```

2.克隆含有子模块的项目

接下来我们将会克隆一个含有子模块的项目。 当你在克隆这样的项目时，默认会包含该子模块目录，但其中还没有任何文件：

```
$ git clone http://github.com/chaconinc/MainProject
Cloning into 'MainProject'...
remote: Counting objects: 14, done.
remote: Compressing objects: 100% (13/13), done.
remote: Total 14 (delta 1), reused 13 (delta 0)
Unpacking objects: 100% (14/14), done.
Checking connectivity... done.
$ cd MainProject
$ ls -la
total 16
drwxr-xr-x   9 schacon  staff  306 Sep 17 15:21 .
drwxr-xr-x   7 schacon  staff  238 Sep 17 15:21 ..
drwxr-xr-x  13 schacon  staff  442 Sep 17 15:21 .git
-rw-r--r--   1 schacon  staff   92 Sep 17 15:21 .gitmodules
drwxr-xr-x   2 schacon  staff   68 Sep 17 15:21 DbConnector
-rw-r--r--   1 schacon  staff  756 Sep 17 15:21 Makefile
drwxr-xr-x   3 schacon  staff  102 Sep 17 15:21 includes
```

```
drwxr-xr-x   4 schacon  staff  136 Sep 17 15:21 scripts
drwxr-xr-x   4 schacon  staff  136 Sep 17 15:21 src
$ cd DbConnector/
$ ls
$
```

其中有 DbConnector 目录，不过是空的。 你必须运行两个命令：git submodule init 用来初始化本地配置文件，而 `git submodule update` 则从该项目中抓取所有数据并检出父项目中列出的合适的提交。

```
$ git submodule init
Submodule 'DbConnector' (http://github.com/chaconinc/DbConnector) registered for path 'DbConnector'
$ git submodule update
Cloning into 'DbConnector'...
remote: Counting objects: 11, done.
remote: Compressing objects: 100% (10/10), done.
remote: Total 11 (delta 0), reused 11 (delta 0)
Unpacking objects: 100% (11/11), done.
Checking connectivity... done.
Submodule path 'DbConnector': checked out 'c3f01dc8862123d317dd46284b05b6892c7b29bc'
```

现在 *DbConnector* 子目录是处在和之前提交时相同的状态了。

不过还有更简单一点的方式。 如果给 `git clone` 命令传递 `--recursive` 选项，它就会自动初始化并更新仓库中的每一个子模块。

```
$ git clone --recursive http://github.com/chaconinc/MainProject
Cloning into 'MainProject'...
remote: Counting objects: 14, done.
remote: Compressing objects: 100% (13/13), done.
remote: Total 14 (delta 1), reused 13 (delta 0)
Unpacking objects: 100% (14/14), done.
Checking connectivity... done.
Submodule 'DbConnector' (http://github.com/chaconinc/DbConnector) registered for path 'DbConnector'
Cloning into 'DbConnector'...
remote: Counting objects: 11, done.
remote: Compressing objects: 100% (10/10), done.
remote: Total 11 (delta 0), reused 11 (delta 0)
Unpacking objects: 100% (11/11), done.
Checking connectivity... done.
```

```
Submodule path 'DbConnector': checked out 'c3f01dc8862123d317dd46284b05b6892c7b29bc'
```

3. 删除 Submodule

git 并不支持直接删除 Submodule 需要手动删除对应的文件:

```
cd pod-project


git rm --cached pod-library
rm -rf pod-library
rm .gitmodules
更改 git 的配置文件 config:
vim .git/config
```

可以看到 Submodule 的配置信息：

```
[submodule "pod-library"]
  url = git@github.com:jjz/pod-library.git
```

删除 submodule 相关的内容,然后提交到远程服务器:

```
git commit -a -m 'remove pod-library submodule'
```