

# 咖啡兔

- [首页](#)
- [博客](#)
- [Activiti](#)
- [Activiti实战](#)
- [分类](#)
- [标签](#)
- [关于我](#)
- [GitHub](#)
- [微博（我才是咖啡兔）](#)
- [Activiti论坛](#)

27 Mar 2012

[Comments](#)

## Git Submodule使用完整教程

自从看了蒋鑫的《[Git权威指南](#)》之后就开始使用Git Submodule功能，团队也都熟悉了怎么使用，多个子系统（模块）都能及时更新到最新的公共资源，把使用的过程以及经验和容易遇到的问题分享给大家。

Git Submodule功能刚刚开始学习可能觉得有点怪异，所以本教程把每一步的操作的命令和结果都用代码的形式展现给大家，以便更好的理解。

### 1. 对于公共资源各种程序员的处理方式

每个公司的系统都会有一套统一的系统风格，或者针对某一个大客户的多个系统风格保持一致，而且如果风格改动后要同步到多个系统中；这样的需求几乎每个开发人员都遇到，下面看看各个层次的程序员怎么处理：

假如对于系统的风格需要几个目录：css、images、js。

- 普通程序员，把最新版本的代码逐个复制到每个项目中，如果有N个项目，那就是要复制N x 3次；如果漏掉了某个文件夹没有复制…@（&#@#。
- 文艺程序员，使用Git Submodule功能，执行：git submodule update，然后冲一杯咖啡悠哉的享受着。

引用一段《[Git权威指南](#)》的话： 项目的版本库在某些情况虾需要引用其他版本库中的文件，例如公司积累了一套常用的函数库，被多个项目调用，显然这个函数库的代码不能直接放到某个项目的代码中，而是要独立为一个代码库，那么其他项目要调用公共函数库该如何处理呢？分别把公共函数库的文件拷贝到各自的项目中会造成冗余，丢弃了公共函数库的维护历史，这显然不是好的方法。

### 2. 开始学习Git Submodule

“工欲善其事，必先利其器”！

既然文艺程序员那么轻松就搞定了，那我们就把过程一一道来。

说明：本例采用两个项目以及两个公共类库演示对submodule的操作。因为在一写资料或者书上的例子都是一个项目对应1~N个lib，但是实际应用往往并不是这么简单。

#### 2.1 创建Git Submodule测试项目

##### 2.1.1 准备环境

```
1 → henryyan@hy-hp ~ pwd
2 /home/henryyan
3 mkdir -p submd/repos
```

创建需要的本地仓库：

```
1 cd ~/submd/repos
2 git --git-dir=lib1.git init --bare
3 git --git-dir=lib2.git init --bare
4 git --git-dir=project1.git init --bare
5 git --git-dir=project2.git init --bare
```

初始化工作区：

```
1 mkdir ~/submd/ws
2 cd ~/submd/ws
```

##### 2.1.2 初始化项目

初始化project1：

```
1 → henryyan@hy-hp ~/submd/ws git clone ../repos/project1.git
```

```

2 Cloning into project1...
3 done.
4 warning: You appear to have cloned an empty repository.
5
6 → henryyan@hy-hp ~/submd/ws cd project1
7 → henryyan@hy-hp ~/submd/ws/project1 git:(master) echo "project1" > project-infos.txt
8 → henryyan@hy-hp ~/submd/ws/project1 git:(master) X ls
9 project-infos.txt
10
11 → henryyan@hy-hp ~/submd/ws/project1 git:(master) X git add project-infos.txt
12 → henryyan@hy-hp ~/submd/ws/project1 git:(master) X git status
13 # On branch master
14 #
15 # Initial commit
16 #
17 # Changes to be committed:
18 #   (use "git rm --cached <file>..." to unstage)
19 #
20 #   new file:   project-infos.txt
21 #
22 → henryyan@hy-hp ~/submd/ws/project1 git:(master) X git commit -m "init project1"
23 [master (root-commit) 473a2e2] init project1
24 1 files changed, 1 insertions(+), 0 deletions(-)
25 create mode 100644 project-infos.txt
26 → henryyan@hy-hp ~/submd/ws/project1 git:(master) git push origin master
27 Counting objects: 3, done.
28 Writing objects: 100% (3/3), 232 bytes, done.
29 Total 3 (delta 0), reused 0 (delta 0)
30 Unpacking objects: 100% (3/3), done.
31 To /home/henryyan/submd/ws/../../repos/project1.git
32 * [new branch]      master -> master
33 </file>

```

初始化project2:

```

1 → henryyan@hy-hp ~/submd/ws/project1 cd ..
2 → henryyan@hy-hp ~/submd/ws git clone ../repos/project2.git
3 Cloning into project2...
4 done.
5 warning: You appear to have cloned an empty repository.
6
7 → henryyan@hy-hp ~/submd/ws cd project2
8 → henryyan@hy-hp ~/submd/ws/project2 git:(master) echo "project2" > project-infos.txt
9 → henryyan@hy-hp ~/submd/ws/project2 git:(master) X ls
10 project-infos.txt
11
12 → henryyan@hy-hp ~/submd/ws/project2 git:(master) X git add project-infos.txt
13 → henryyan@hy-hp ~/submd/ws/project2 git:(master) X git status
14 # On branch master
15 #
16 # Initial commit
17 #
18 # Changes to be committed:
19 #   (use "git rm --cached <file>..." to unstage)
20 #
21 #   new file:   project-infos.txt
22 #
23 → henryyan@hy-hp ~/submd/ws/project2 git:(master) X git commit -m "init project2"
24 [master (root-commit) 473a2e2] init project2
25 1 files changed, 1 insertions(+), 0 deletions(-)
26 create mode 100644 project-infos.txt
27 → henryyan@hy-hp ~/submd/ws/project2 git:(master) git push origin master
28 Counting objects: 3, done.
29 Writing objects: 100% (3/3), 232 bytes, done.
30 Total 3 (delta 0), reused 0 (delta 0)
31 Unpacking objects: 100% (3/3), done.
32 To /home/henryyan/submd/ws/../../repos/project2.git
33 * [new branch]      master -> master
34 </file>

```

### 2.1.3 初始化公共类库

初始化公共类库lib1:

```

1 → henryyan@hy-hp ~/submd/ws git clone ../repos/lib1.git
2 Cloning into lib1...
3 done.
4 warning: You appear to have cloned an empty repository.
5 → henryyan@hy-hp ~/submd/ws cd lib1
6 → henryyan@hy-hp ~/submd/ws/lib1 git:(master) echo "I'm lib1." > lib1-features
7 → henryyan@hy-hp ~/submd/ws/lib1 git:(master) X git add lib1-features
8 → henryyan@hy-hp ~/submd/ws/lib1 git:(master) X git commit -m "init lib1"

```

```

9 [master (root-commit) c22aff8] init lib1
10 1 files changed, 1 insertions(+), 0 deletions(-)
11 create mode 100644 lib1-features
12 → henryyan@hy-hp ~/submd/ws/lib1 git:(master) git push origin master
13 Counting objects: 3, done.
14 Writing objects: 100% (3/3), 227 bytes, done.
15 Total 3 (delta 0), reused 0 (delta 0)
16 Unpacking objects: 100% (3/3), done.
17 To /home/henryyan/submd/ws/../../repos/lib1.git
18 * [new branch] master -> master

```

初始化公共类库lib2:

```

1 → henryyan@hy-hp ~/submd/ws/lib1 git:(master) cd ..
2 → henryyan@hy-hp ~/submd/ws git clone ../repos/lib2.git
3 Cloning into lib2...
4 done.
5 warning: You appear to have cloned an empty repository.
6 → henryyan@hy-hp ~/submd/ws cd lib2
7 → henryyan@hy-hp ~/submd/ws/lib2 git:(master) echo "I'm lib2." > lib2-features
8 → henryyan@hy-hp ~/submd/ws/lib2 git:(master) X git add lib2-features
9 → henryyan@hy-hp ~/submd/ws/lib2 git:(master) X git commit -m "init lib2"
10 [master (root-commit) c22aff8] init lib2
11 1 files changed, 1 insertions(+), 0 deletions(-)
12 create mode 100644 lib2-features
13 → henryyan@hy-hp ~/submd/ws/lib2 git:(master) git push origin master
14 Counting objects: 3, done.
15 Writing objects: 100% (3/3), 227 bytes, done.
16 Total 3 (delta 0), reused 0 (delta 0)
17 Unpacking objects: 100% (3/3), done.
18 To /home/henryyan/submd/ws/../../repos/lib2.git
19 * [new branch] master -> master

```

## 2.2 为主项目添加Submodules

### 2.2.1 为project1添加lib1和lib2

```

1 → henryyan@hy-hp ~/submd/ws/lib2 git:(master) cd ../project1
2 → henryyan@hy-hp ~/submd/ws/project1 git:(master) ls
3 project-infos.txt
4 → henryyan@hy-hp ~/submd/ws/project1 git:(master) git submodule add ~/submd/repos/lib1.git libs/lib1
5 Cloning into libs/lib1...
6 done.
7 → henryyan@hy-hp ~/submd/ws/project1 git:(master) X git submodule add ~/submd/repos/lib2.git libs/lib2
8 Cloning into libs/lib2...
9 done.
10 → henryyan@hy-hp ~/submd/ws/project1 git:(master) X ls
11 libs project-infos.txt
12 → henryyan@hy-hp ~/submd/ws/project1 git:(master) X ls libs
13 lib1 lib2
14
15 → henryyan@hy-hp ~/submd/ws/project1 git:(master) X git status
16 # On branch master
17 # Changes to be committed:
18 #   (use "git reset HEAD <file>..." to unstage)
19 #
20 #   new file:   .gitmodules
21 #   new file:   libs/lib1
22 #   new file:   libs/lib2
23 #
24
25 # 查看一下公共类库的内容
26
27 → henryyan@hy-hp ~/submd/ws/project1 git:(master) cat libs/lib1/lib1-features
28 I'm lib1.
29 → henryyan@hy-hp ~/submd/ws/project1 git:(master) cat libs/lib2/lib2-features
30 I'm lib2.
31 </file>

```

好了，到目前为止我们已经使用git submodule add命令为project1成功添加了两个公共类库（lib1、lib2），查看了当前的状态发现添加了一个新文件(.gitmodules)和两个文件夹(libs/lib1、libs/lib2)；那么新增的.gitmodules文件是做什么用的呢？我们查看一下文件内容便知晓了：

```

1 n@hy-hp ~/submd/ws/project1 git:(master) X cat .gitmodules
2 [submodule "libs/lib1"]
3   path = libs/lib1
4   url = /home/henryyan/submd/repos/lib1.git
5 [submodule "libs/lib2"]
6   path = libs/lib2
7   url = /home/henryyan/submd/repos/lib2.git

```

原来如此，.gitmodules记录了每个submodule的引用信息，知道在当前项目的位置以及仓库的所在。

好的，我们现在把更改提交到仓库。

```
1 → henryyan@hy-hp ~/submd/ws/project1 git:(master) X git commit -a -m "add submodules[lib1,lib2]"
2 [master 7157977] add submodules[lib1,lib2] to project1
3 3 files changed, 8 insertions(+), 0 deletions(-)
4 create mode 100644 .gitmodules
5 create mode 160000 libs/lib1
6 create mode 160000 libs/lib2
7
8 → henryyan@hy-hp ~/submd/ws/project1 git:(master) git push
9 Counting objects: 5, done.
10 Delta compression using up to 2 threads.
11 Compressing objects: 100% (4/4), done.
12 Writing objects: 100% (4/4), 491 bytes, done.
13 Total 4 (delta 0), reused 0 (delta 0)
14 Unpacking objects: 100% (4/4), done.
15 To /home/henryyan/submd/ws/../../repos/project1.git
16 45cbbcb..7157977 master -> master
```

假如你是第一次引入公共类库的开发人员，那么项目组的其他成员怎么Clone带有Submodule的项目呢，下面我们再clone一个项目讲解如何操作。

2.3 Clone带有Submodule的仓库

模拟开发人员B……

```
1 → henryyan@hy-hp ~/submd/ws/project1 git:(master) cd ~/submd/ws
2 → henryyan@hy-hp ~/submd/ws git clone ../repos/project1.git project1-b
3 Cloning into project1-b...
4 done.
5 → henryyan@hy-hp ~/submd/ws cd project1-b
6 → henryyan@hy-hp ~/submd/ws/project1-b git:(master) git submodule
7 -c22aff85be91eca442734dcb07115ffe526b13a1 libs/lib1
8 -7290dce0062bd77df1d83b27dd3fa3f25a836b54 libs/lib2
```

看到submodules的状态是hash码和文件目录，但是注意前面有一个减号：-，含义是该子模块还没有检出。

OK，检出project1-b的submodules……

```
1 → henryyan@hy-hp ~/submd/ws/project1-b git:(master) git submodule init
2 Submodule 'libs/lib1' (/home/henryyan/submd/repos/lib1.git) registered for path 'libs/lib1'
3 Submodule 'libs/lib2' (/home/henryyan/submd/repos/lib2.git) registered for path 'libs/lib2'
4 → henryyan@hy-hp ~/submd/ws/project1-b git:(master) git submodule update
5 Cloning into libs/lib1...
6 done.
7 Submodule path 'libs/lib1': checked out 'c22aff85be91eca442734dcb07115ffe526b13a1'
8 Cloning into libs/lib2...
9 done.
10 Submodule path 'libs/lib2': checked out '7290dce0062bd77df1d83b27dd3fa3f25a836b54'
```

读者可以查看：.git/config文件的内容，最下面有submodule的注册信息！

验证一下类库的文件是否存在：

```
1 → henryyan@hy-hp ~/submd/ws/project1-b git:(master) cat libs/lib1/lib1-features libs/lib2/lib2-features
2 I'm lib1.
3 I'm lib2.
```

上面的两个命令(git submodule init & update)其实可以简化，后面会讲到！

2.3 修改Submodule

我们在开发人员B的项目上修改Submodule的内容。

先看一下当前Submodule的状态：

```
1 → henryyan@hy-hp ~/submd/ws/project1-b git:(master) cd libs/lib1
2 → henryyan@hy-hp ~/submd/ws/project1-b/libs/lib1 git status
3 # Not currently on any branch.
4 nothing to commit (working directory clean)
```

为什么是Not currently on any branch呢？不是应该默认在master分支吗？别急，一一解答！

Git对于Submodule有特殊的处理方式，在一个主项目中引入了Submodule其实Git做了3件事情：

- 记录引用的仓库
- 记录主项目中Submodules的目录位置



- 记录引用Submodule的commit id

在project1中push之后其实就是更新了引用的commit id，然后project1-b在clone的时候获取到了submodule的commit id，然后当执行git submodule update的时候git就根据gitlink获取submodule的commit id，最后获取submodule的文件，所以clone之后不在任何分支上；但是master分支的commit id和HEAD保持一致。

查看~/submd/ws/project1-b/libs/lib1的引用信息：

```
1 → henryyan@hy-hp ~/submd/ws/project1-b/libs/lib1 cat .git/HEAD
2 c22aff85be91eca442734dcb07115ffe526b13a1
3 → henryyan@hy-hp ~/submd/ws/project1-b/libs/lib1 cat .git/refs/heads/master
4 c22aff85be91eca442734dcb07115ffe526b13a1
```

现在我们要修改lib1的文件需要先切换到master分支：

```
1 → henryyan@hy-hp ~/submd/ws/project1-b/libs/lib1 git checkout master
2 Switched to branch 'master'
3 → henryyan@hy-hp ~/submd/ws/project1-b/libs/lib1 git:(master) echo "add by developer B" >> lib1-fea
4 → henryyan@hy-hp ~/submd/ws/project1-b/libs/lib1 git:(master) X git commit -a -m "update lib1-fea
5 [master 36ad12d] update lib1-features by developer B
6 1 files changed, 1 insertions(+), 0 deletions(-)
```

在主项目中修改Submodule提交到仓库稍微繁琐一点，在git push之前我们先看看project1-b状态：

```
1 → henryyan@hy-hp ~/submd/ws/project1-b git:(master) X git status
2 # On branch master
3 # Changes not staged for commit:
4 #   (use "git add <file>..." to update what will be committed)
5 #   (use "git checkout -- <file>..." to discard changes in working directory)
6 #
7 #    modified:   libs/lib1 (new commits)
8 #
9 no changes added to commit (use "git add" and/or "git commit -a")
10 </file></file>
```

libs/lib1 (new commits)状态表示libs/lib1有新的提交，这个比较特殊，看看project1-b的状态：

```
1 → henryyan@hy-hp ~/submd/ws/project1-b git:(master) X git diff
2 diff --git a/libs/lib1 b/libs/lib1
3 index c22aff8..36ad12d 160000
4 --- a/libs/lib1
5 +++ b/libs/lib1
6 @@ -1,1 @@
7 -Subproject commit c22aff85be91eca442734dcb07115ffe526b13a1
8 +Subproject commit 36ad12d40d8a41a4a88a64add27bd57cf56c9de2
```

从状态中可以看出libs/lib1的commit id由原来的c22aff85be91eca442734dcb07115ffe526b13a1更改为36ad12d40d8a41a4a88a64add27bd57cf56c9de2

注意：如果现在执行了git submodule update操作那么libs/lib1的commit id又会还原到c22aff85be91eca442734dcb07115ffe526b13a1，

这样的话刚刚的修改是不是就丢死了呢？不会，因为修改已经提交到了master分支，只要再git checkout master就可以了。

现在可以把libs/lib1的修改提交到仓库了：

```
1 → henryyan@hy-hp ~/submd/ws/project1-b git:(master) X cd libs/lib1
2 → henryyan@hy-hp ~/submd/ws/project1-b/libs/lib1 git:(master) git push
3 Counting objects: 5, done.
4 Writing objects: 100% (3/3), 300 bytes, done.
5 Total 3 (delta 0), reused 0 (delta 0)
6 Unpacking objects: 100% (3/3), done.
7 To /home/henryyan/submd/repos/lib1.git
8 c22aff8..36ad12d master -> master
```

现在仅仅只完成了一步，下一步要提交project1-b引用submodule的commit id：

```
1 → henryyan@hy-hp ~/submd/ws/project1-b/libs/lib1 git:(master) cd ~/submd/ws/project1-b
2 → henryyan@hy-hp ~/submd/ws/project1-b git:(master) X git add -u
3 → henryyan@hy-hp ~/submd/ws/project1-b git:(master) X git commit -m "update libs/lib1 to latest
4 [master c96838a] update libs/lib1 to latest commit id
5 1 files changed, 1 insertions(+), 1 deletions(-)
6 → henryyan@hy-hp ~/submd/ws/project1-b git:(master) git push
7 Counting objects: 5, done.
8 Delta compression using up to 2 threads.
9 Compressing objects: 100% (3/3), done.
10 Writing objects: 100% (3/3), 395 bytes, done.
11 Total 3 (delta 0), reused 0 (delta 0)
12 Unpacking objects: 100% (3/3), done.
13 To /home/henryyan/submd/ws/../../repos/project1.git
14 7157977..c96838a master -> master
```

OK, 大功高成, 我们完成了Submodule的修改并把libs/lib1的最新commit id提交到了仓库。

接下来要看看project1怎么获取submodule了。

## 2.4 更新主项目的Submodules

好的, 让我们先进入project1目录同步仓库:

```
1 → henryyan@hy-hp ~/submd/ws/project1-b git:(master) cd ../project1
2 → henryyan@hy-hp ~/submd/ws/project1 git:(master) git pull
3 remote: Counting objects: 5, done.
4 remote: Compressing objects: 100% (3/3), done.
5 remote: Total 3 (delta 0), reused 0 (delta 0)
6 Unpacking objects: 100% (3/3), done.
7 From /home/henryyan/submd/ws/../../repos/project1
8 7157977..c96838a master -> origin/master
9 Updating 7157977..c96838a
10 Fast-forward
11  libs/lib1 |      2 +-
12  1 files changed, 1 insertions(+), 1 deletions(-)
13 → henryyan@hy-hp ~/submd/ws/project1 git:(master) X git status
14 # On branch master
15 # Changes not staged for commit:
16 #   (use "git add <file>..." to update what will be committed)
17 #   (use "git checkout -- <file>..." to discard changes in working directory)
18 #
19 #    modified:   libs/lib1 (new commits)
20 #
21 no changes added to commit (use "git add" and/or "git commit -a")
22 </file></file>
```

我们运行了git pull命令和git status获取了最新的仓库源码, 然后看到了状态时modified, 这是为什么呢?

我们用git diff比较一下不同:

```
1 → henryyan@hy-hp ~/submd/ws/project1 git:(master) X git diff
2 diff --git a/libs/lib1 b/libs/lib1
3 index 36ad12d..c22aff8 160000
4 --- a/libs/lib1
5 +++ b/libs/lib1
6 @@ -1,1 @@
7 -Subproject commit 36ad12d40d8a41a4a88a64add27bd57cf56c9de2
8 +Subproject commit c22aff85be91eca442734dcb07115ffe526b13a1
```

从diff的结果分析出来时因为submodule的commit id更改了, 我们前面刚刚讲了要在主项目更新submodule的内容首先要提交submdoule的内容, 然后再更新主项目中引用的submodulecommit id; 现在看到的不同就是因为刚刚更改了project1-b的submodule commit id; 好的, 我来学习一下怎么更新project1的公共类库。

follow me.....

```
1 → henryyan@hy-hp ~/submd/ws/project1 git:(master) X git submodule update
2 → henryyan@hy-hp ~/submd/ws/project1 git:(master) X git status
3 # On branch master
4 # Changes not staged for commit:
5 #   (use "git add <file>..." to update what will be committed)
6 #   (use "git checkout -- <file>..." to discard changes in working directory)
7 #
8 #    modified:   libs/lib1 (new commits)
9 #
10 no changes added to commit (use "git add" and/or "git commit -a")
11 </file></file>
```

泥马, 为什么没有更新? git submodule update命令不是更新子模块仓库的吗?

别急, 先听我解释: 因为子模块是在project1中引入的, git submodule add ~/submd/repos/lib1.git libs/lib1命令的结果, 操作之后git只是把lib1的内容clone到了project1中, 但是没有在仓库注册, 证据如下:

```
1 → henryyan@hy-hp ~/submd2/ws/project1 git:(master) X cat .git/config
2 [core]
3   repositoryformatversion = 0
4   filemode = true
5   bare = false
6   logallrefupdates = true
7 [remote "origin"]
8   fetch = +refs/heads/*:refs/remotes/origin/*
9   url = /home/henryyan/submd/ws/../../repos/project1.git
10 [branch "master"]
11   remote = origin
12   merge = refs/heads/master
```

我们说过git submodule init就是在.git/config中注册子模块的信息, 下面我们试试注册之后再更新子模块:

```

1 → henryyan@hy-hp ~/submd/ws/project1 git:(master) X git submodule init
2 Submodule 'libs/lib1' (/home/henryyan/submd/repos/lib1.git) registered for path 'libs/lib1'
3 Submodule 'libs/lib2' (/home/henryyan/submd/repos/lib2.git) registered for path 'libs/lib2'
4 → henryyan@hy-hp ~/submd/ws/project1 git:(master) X git submodule update
5 remote: Counting objects: 5, done.
6 remote: Total 3 (delta 0), reused 0 (delta 0)
7 Unpacking objects: 100% (3/3), done.
8 From /home/henryyan/submd/repos/lib1
9   c22aff8..36ad12d  master    -> origin/master
10 Submodule path 'libs/lib1': checked out '36ad12d40d8a41a4a88a64add27bd57cf56c9de2'
11
12 → henryyan@hy-hp ~/submd/ws/project1 git:(master) cat .git/config
13 [core]
14   repositoryformatversion = 0
15   filemode = true
16   bare = false
17   logallrefupdates = true
18 [remote "origin"]
19   fetch = +refs/heads/*:refs/remotes/origin/*
20   url = /home/henryyan/submd/ws/../../repos/project1.git
21 [branch "master"]
22   remote = origin
23   merge = refs/heads/master
24 [submodule "libs/lib1"]
25   url = /home/henryyan/submd/repos/lib1.git
26 [submodule "libs/lib2"]
27   url = /home/henryyan/submd/repos/lib2.git
28
29 → henryyan@hy-hp ~/submd/ws/project1 git:(master) cat libs/lib1/lib1-features
30 I'm lib1.
31 add by developer B

```

上面的结果足以证明刚刚的推断，所以记得当需要更新子模块的内容时请先确保已经运行过git submodule init。

## 2.5 为project2添加lib1和lib2

这个操作对于读到这里的你来说应该是轻车熟路了，action:

```

1 → henryyan@hy-hp ~/submd/ws/project1 git:(master) cd ~/submd/ws/project2
2 → henryyan@hy-hp ~/submd/ws/project2 git:(master) git submodule add ~/submd/repos/lib1.git libs/
3 Cloning into libs/lib1...
4 done.
5 → henryyan@hy-hp ~/submd/ws/project2 git:(master) X git submodule add ~/submd/repos/lib2.git lib
6 zsh: correct 'libs/lib2' to 'libs/lib1' [nyae]? n
7 Cloning into libs/lib2...
8 done.
9 → henryyan@hy-hp ~/submd/ws/project2 git:(master) X ls
10 libs project-infos.txt
11 → henryyan@hy-hp ~/submd/ws/project2 git:(master) X git submodule init
12 Submodule 'libs/lib1' (/home/henryyan/submd/repos/lib1.git) registered for path 'libs/lib1'
13 Submodule 'libs/lib2' (/home/henryyan/submd/repos/lib2.git) registered for path 'libs/lib2'
14
15 → henryyan@hy-hp ~/submd/ws/project2 git:(master) X git status
16 # On branch master
17 # Changes to be committed:
18 #   (use "git reset HEAD <file>..." to unstage)
19 #
20 #   new file:   .gitmodules
21 #   new file:   libs/lib1
22 #   new file:   libs/lib2
23 #
24 → henryyan@hy-hp ~/submd/ws/project2 git:(master) X git commit -a -m "add lib1 and lib2"
25 [master 8dc697f] add lib1 and lib2
26 3 files changed, 8 insertions(+), 0 deletions(-)
27 create mode 100644 .gitmodules
28 create mode 160000 libs/lib1
29 create mode 160000 libs/lib2
30 → henryyan@hy-hp ~/submd/ws/project2 git:(master) git push
31 Counting objects: 5, done.
32 Delta compression using up to 2 threads.
33 Compressing objects: 100% (4/4), done.
34 Writing objects: 100% (4/4), 471 bytes, done.
35 Total 4 (delta 0), reused 0 (delta 0)
36 Unpacking objects: 100% (4/4), done.
37 To /home/henryyan/submd/ws/../../repos/project2.git
38   6e15c68..8dc697f  master -> master
39
40 </file>

```

我们依次执行了添加submodule并commit和push到仓库，此阶段任务完成。



## 2.6 修改lib1和lib2并同步到project1和project2

假如开发人员C同时负责project1和project2，有可能在修改project1的某个功能的时候发现lib1或者lib2的某个组件有bug需要修复，这个需求多模块和大型系统中经常遇到，我们应该怎么解决呢？

假如我的需求如下：

- 在lib1中添加一个文件：README，用来描述lib1的功能
- 在lib2中的lib2-features文件中添加一写文字：学习Git submodule的修改并同步功能

### 2.6.1 在lib1中添加一个文件：README

```
1 → henryyan@hy-hp ~/submd/ws/project2 git:(master) cd libs/lib1
2 → henryyan@hy-hp ~/submd/ws/project2/libs/lib1 git:(master) echo "lib1 readme contents" > README
3 → henryyan@hy-hp ~/submd/ws/project2/libs/lib1 git:(master) X git add README
4 → henryyan@hy-hp ~/submd/ws/project2/libs/lib1 git:(master) X git commit -m "add file README"
5 [master 8c666d8] add file README
6 1 files changed, 1 insertions(+), 0 deletions(-)
7 create mode 100644 README
8 → henryyan@hy-hp ~/submd/ws/project2/libs/lib1 git:(master) git push
9 Counting objects: 4, done.
10 Delta compression using up to 2 threads.
11 Compressing objects: 100% (2/2), done.
12 Writing objects: 100% (3/3), 310 bytes, done.
13 Total 3 (delta 0), reused 0 (delta 0)
14 Unpacking objects: 100% (3/3), done.
15 To /home/henryyan/submd/repos/lib1.git
16 36ad12d..8c666d8 master -> master
```

前面提到过现在仅仅只完成了一部分，我们需要在project2中再更新lib1的commit id:

```
1 → henryyan@hy-hp ~/submd/ws/project2 git:(master) X git status
2 # On branch master
3 # Changes not staged for commit:
4 #   (use "git add <file>..." to update what will be committed)
5 #   (use "git checkout -- <file>..." to discard changes in working directory)
6 #
7 #    modified:   libs/lib1 (new commits)
8 #
9 no changes added to commit (use "git add" and/or "git commit -a")
10 → henryyan@hy-hp ~/submd/ws/project2 git:(master) X git add libs/lib1
11 → henryyan@hy-hp ~/submd/ws/project2 git:(master) X git commit -m "update lib1 to latest commit"
12 [master ce1f3ba] update lib1 to latest commit id
13 1 files changed, 1 insertions(+), 1 deletions(-)
14 </file></file>
```

我们暂时不push到仓库，等待和lib2的修改一起push。

### 2.6.2 在lib2中的lib2-features文件添加文字

```
1 → henryyan@hy-hp ~/submd/ws/project2 git:(master) cd libs/lib2
2 → henryyan@hy-hp ~/submd/ws/project2/libs/lib2 git:(master) echo "学习Git submodule的修改并同步功能" > lib2-features
3 → henryyan@hy-hp ~/submd/ws/project2/libs/lib2 git:(master) X git add lib2-features
4 → henryyan@hy-hp ~/submd/ws/project2/libs/lib2 git:(master) X git commit -m "添加文字: 学习Git submodule的修改并同步功能"
5 [master e372b21] 添加文字: 学习Git submodule的修改并同步功能
6 1 files changed, 1 insertions(+), 0 deletions(-)
7 → henryyan@hy-hp ~/submd/ws/project2/libs/lib2 git:(master) git push
8 Counting objects: 5, done.
9 Delta compression using up to 2 threads.
10 Compressing objects: 100% (2/2), done.
11 Writing objects: 100% (3/3), 376 bytes, done.
12 Total 3 (delta 0), reused 0 (delta 0)
13 Unpacking objects: 100% (3/3), done.
14 To /home/henryyan/submd/repos/lib2.git
15 7290dce..e372b21 master -> master
16
17 → henryyan@hy-hp ~/submd/ws/project2/libs/lib2 git:(master) echo "学习Git submodule的修改并同步功能" > lib2-features
18 → henryyan@hy-hp ~/submd/ws/project2/libs/lib2 git:(master) X git add lib2-features
19 → henryyan@hy-hp ~/submd/ws/project2/libs/lib2 git:(master) X git commit -m "添加文字: 学习Git submodule的修改并同步功能"
20 [master e372b21] 添加文字: 学习Git submodule的修改并同步功能
21 1 files changed, 1 insertions(+), 0 deletions(-)
22 → henryyan@hy-hp ~/submd/ws/project2/libs/lib2 git:(master) git push
23 Counting objects: 5, done.
24 Delta compression using up to 2 threads.
25 Compressing objects: 100% (2/2), done.
26 Writing objects: 100% (3/3), 376 bytes, done.
27 Total 3 (delta 0), reused 0 (delta 0)
28 Unpacking objects: 100% (3/3), done.
```



```

29 To /home/henryyan/submd/repos/lib2.git
30 7290dce..e372b21 master -> master
31 → henryyan@hy-hp ~/submd/ws/project2/libs/lib2 git:(master) cd -
32 ~/submd/ws/project2
33 → henryyan@hy-hp ~/submd/ws/project2 git:(master) X git status
34 # On branch master
35 # Your branch is ahead of 'origin/master' by 1 commit.
36 #
37 # Changes not staged for commit:
38 #   (use "git add <file>..." to update what will be committed)
39 #   (use "git checkout -- <file>..." to discard changes in working directory)
40 #
41 #    modified:   libs/lib2 (new commits)
42 #
43 no changes added to commit (use "git add" and/or "git commit -a")
44 → henryyan@hy-hp ~/submd/ws/project2 git:(master) X git add libs/lib2
45 → henryyan@hy-hp ~/submd/ws/project2 git:(master) X git commit -m "update lib2 to latest commit"
46 [master df344c5] update lib2 to latest commit id
47 1 files changed, 1 insertions(+), 1 deletions(-)
48 → henryyan@hy-hp ~/submd/ws/project2 git:(master) git status
49 # On branch master
50 # Your branch is ahead of 'origin/master' by 2 commits.
51 #
52 nothing to commit (working directory clean)
53 → henryyan@hy-hp ~/submd/ws/project2 git:(master) git push
54 Counting objects: 8, done.
55 Delta compression using up to 2 threads.
56 Compressing objects: 100% (6/6), done.
57 Writing objects: 100% (6/6), 776 bytes, done.
58 Total 6 (delta 0), reused 0 (delta 0)
59 Unpacking objects: 100% (6/6), done.
60 To /home/henryyan/submd/ws/../../repos/project2.git
61 8dc697f..df344c5 master -> master
62 </file></file>

```

## 2.7 同步project2的lib1和lib2的修改到project1

现在project2已经享受到了最新的代码带来的快乐，那么既然project1和project2属于同一个风格，或者调用同一个功能，要让这两个(可能几十个)项目保持一致。

```

1 → henryyan@hy-hp ~/submd/ws/project2 git:(master) cd ../project1
2 → henryyan@hy-hp ~/submd/ws/project1 git:(master) git pull
3 Already up-to-date.

```

看看上面的结果对吗？为什么lib1和lib2更新了但是没有显示new commits呢？说到这里我记得刚刚开始学习的时候真得要晕死了，Git跟我玩捉迷藏游戏，为什么我明明提交了但是从project1更新不到任何改动呢？

帮大家分析一下问题，不过在分析之前先看看当前(project1和project2)的submodule状态：

```

1 # project2 的状态，也就是我们刚刚修改后的状态
2 → henryyan@hy-hp ~/submd/ws/project2 git:(master) git submodule
3 8c666d86531513dd1aebdf235f142adbac72c035 libs/lib1 (heads/master)
4 e372b21dfffa611802c282278ec916b5418acebc2 libs/lib2 (heads/master)
5
6 # project1 的状态，等待更新submodules
7 → henryyan@hy-hp ~/submd/ws/project1 git:(master) git submodule
8 36ad12d40d8a41a4a88a64add27bd57cf56c9de2 libs/lib1 (remotes/origin/HEAD)
9 7290dce0062bd77df1d83b27dd3fa3f25a836b54 libs/lib2 (heads/master)

```

两个项目有两个区别：

- commit id各不相同
- libs/lib1所处的分支不同

### 2.7.1 更新project1的lib1和lib2改动

我们还记得刚刚在project2中修改的时候把lib1和lib2都切换到了master分支，目前project1中的lib1不在任何分支，我们先切换到master分支：

```

1 → henryyan@hy-hp ~/submd/ws/project1 git:(master) cd libs/lib1
2 → henryyan@hy-hp ~/submd/ws/project1/libs/lib1 git checkout master
3 Previous HEAD position was 36ad12d... update lib1-features by developer B
4 Switched to branch 'master'
5 Your branch is behind 'origin/master' by 1 commit, and can be fast-forwarded.
6 → henryyan@hy-hp ~/submd/ws/project1/libs/lib1 git:(master) git pull
7 remote: Counting objects: 4, done.
8 remote: Compressing objects: 100% (2/2), done.
9 remote: Total 3 (delta 0), reused 0 (delta 0)
10 Unpacking objects: 100% (3/3), done.

```

```

11 From /home/henryyan/submd/repos/lib1
12 36ad12d..8c666d8 master -> origin/master
13 Updating c22aff8..8c666d8
14 Fast-forward
15  README | 1 +
16  lib1-features | 1 +
17  2 files changed, 2 insertions(+), 0 deletions(-)
18  create mode 100644 README
19 → henryyan@hy-hp ~/submd/ws/project1/libs/lib1 git:(master)

```

果不其然，我们看到了刚刚在project2中修改的内容，同步到了project1中，当然现在更新了project1的lib1，commit id也会随之变动：

```

1 → henryyan@hy-hp ~/submd/ws/project1/libs/lib1 git:(master) cd ../../
2 → henryyan@hy-hp ~/submd/ws/project1 git:(master) X git status
3 # On branch master
4 # Changes not staged for commit:
5 #   (use "git add <file>..." to update what will be committed)
6 #   (use "git checkout -- <file>..." to discard changes in working directory)
7 #
8 #   modified:   libs/lib1 (new commits)
9 #
10 no changes added to commit (use "git add" and/or "git commit -a")
11 → henryyan@hy-hp ~/submd/ws/project1 git:(master) X git diff
12 diff --git a/libs/lib1 b/libs/lib1
13 index 36ad12d..8c666d8 160000
14 --- a/libs/lib1
15 +++ b/libs/lib1
16 @@ -1,1 @@
17 -Subproject commit 36ad12d40d8a41a4a88a64add27bd57cf56c9de2
18 +Subproject commit 8c666d86531513dd1aebdf235f142adbac72c035
19 </file></file>

```

现在最新的commit id和project2目前的状态一致，说明真的同步了；好的，现在可以使用相同的办法更新lib2了：

```

1 → henryyan@hy-hp ~/submd/ws/project1 git:(master) X cd libs/lib2
2 → henryyan@hy-hp ~/submd/ws/project1/libs/lib2 git:(master) git pull
3 remote: Counting objects: 5, done.
4 remote: Compressing objects: 100% (2/2), done.
5 remote: Total 3 (delta 0), reused 0 (delta 0)
6 Unpacking objects: 100% (3/3), done.
7 From /home/henryyan/submd/repos/lib2
8 7290dce..e372b21 master -> origin/master
9 Updating 7290dce..e372b21
10 Fast-forward
11  lib2-features | 1 +
12  1 files changed, 1 insertions(+), 0 deletions(-)

```

### 2.7.2 更新project1的submodule引用

在2.7.1中我们更新了project1的lib1和lib2的最新版本，现在要把最新的commit id保存到project1中以保持最新的引用。

```

1 → henryyan@hy-hp ~/submd/ws/project1 git:(master) X git status
2 # On branch master
3 # Changes not staged for commit:
4 #   (use "git add <file>..." to update what will be committed)
5 #   (use "git checkout -- <file>..." to discard changes in working directory)
6 #
7 #   modified:   libs/lib1 (new commits)
8 #   modified:   libs/lib2 (new commits)
9 #
10 no changes added to commit (use "git add" and/or "git commit -a")
11 → henryyan@hy-hp ~/submd/ws/project1 git:(master) X git commit -a -m "update lib1 and lib2 commi
12 [master 8fcca50] update lib1 and lib2 commit id to new version
13 2 files changed, 2 insertions(+), 2 deletions(-)
14 → henryyan@hy-hp ~/submd/ws/project1 git:(master) git push
15 Counting objects: 5, done.
16 Delta compression using up to 2 threads.
17 Compressing objects: 100% (3/3), done.
18 Writing objects: 100% (3/3), 397 bytes, done.
19 Total 3 (delta 0), reused 0 (delta 0)
20 Unpacking objects: 100% (3/3), done.
21 To /home/henryyan/submd/ws/../../repos/project1.git
22 c96838a..8fcca50 master -> master
23 </file></file>

```

### 2.8 更新project1-b项目的子模块(使用脚本)

```

1 → henryyan@hy-hp ~/submd/ws/project1-b git:(master) git pull
2 remote: Counting objects: 5, done.
3 remote: Compressing objects: 100% (3/3), done.

```

```
4 remote: Total 3 (delta 0), reused 0 (delta 0)
5 Unpacking objects: 100% (3/3), done.
6 From /home/henryyan/submd/ws/../../repos/project1
7   c96838a..8fcca50  master    -> origin/master
8 Updating c96838a..8fcca50
9 Fast-forward
10  libs/lib1 |      2 +-
11  libs/lib2 |      2 +-
12  2 files changed, 2 insertions(+), 2 deletions(-)
13 → henryyan@hy-hp  ~/submd/ws/project1-b git:(master) X git status
14 # On branch master
15 # Changes not staged for commit:
16 #   (use "git add <file>..." to update what will be committed)
17 #   (use "git checkout -- <file>..." to discard changes in working directory)
18 #
19 #    modified:   libs/lib1 (new commits)
20 #    modified:   libs/lib2 (new commits)
21 #
22 no changes added to commit (use "git add" and/or "git commit -a")
23 </file></file>
```

Git提示lib1和lib2有更新内容，这个判断的依据来源于submodule commit id的引用。

现在怎么更新呢？难道还是像project1中那样进入子模块的目录然后git checkout master，接着git pull？

而且现在仅仅才两个子模块、两个项目，如果在真实的项目中使用的话可能几个到几十个不等，再加上N个submodule，自己算一下要怎么更新多少个submodules？

例如笔者现在做的一个项目有5个web模块，每个web模块引用公共的css、js、images、jsp资源，这样就有20个submodule需要更新！！

工欲善其事，必先利其器，写一个脚本代替手动任务。

### 2.8.1 牛刀小试

```
1 → henryyan@hy-hp  ~/submd/ws/project1-b git:(master) X grep path .gitmodules | awk '{ print $3 }'?
2 → henryyan@hy-hp  ~/submd/ws/project1-b git:(master) X cat /tmp/study-git-submodule-dirs
3  libs/lib1
4  libs/lib2
```

我们通过分析.gitmodules文件得出子模块的路径，然后就可以根据这些路径进行更新。

### 2.8.2 上路

```
1 → henryyan@hy-hp  ~/submd/ws/project1-b git:(master) X mkdir bin
2 → henryyan@hy-hp  ~/submd/ws/project1-b git:(master) X vi bin/update-submodules.sh
```

把下面的脚本复制到bin/update-submodules.sh中：

```
1 #!/bin/bash
2 grep path .gitmodules | awk '{ print $3 }' > /tmp/study-git-submodule-dirs
3
4 # read
5 while read LINE
6 do
7     echo $LINE
8     (cd ./$LINE && git checkout master && git pull)
9 done < /tmp/study-git-submodule-dirs
```

稍微解释一下上面的脚本执行过程：

- 首先把子模块的路径写入到文件/tmp/study-git-submodule-dirs中；
- 然后读取文件中的子模块路径，依次切换到master分支(修改都是在master分支上进行的)，最后更新最近改动。

### 2.8.3 2013-01-19更新

网友@紫煌给出了更好的办法，一个命令就可以代替上面的bin/update-submodules.sh的功能：

```
1 git submodule foreach git pull
```

此命令也脚本一样，循环进入（enter）每个子模块的目录，然后执行foreach后面的命令。

该后面的命令可以任意的，例如 git submodule foreach ls -l 可以列出每个子模块的文件列表

### 2.8.3 体验工具带来的便利

```
1 → henryyan@hy-hp  ~/submd/ws/project1-b git:(master) X git submodule
2 +36ad12d40d8a41a4a88a64add27bd57cf56c9de2 libs/lib1 (heads/master)
3 +7290dce0062bd77df1d83b27dd3fa3f25a836b54 libs/lib2 (heads/master)
```



```

4
5 # 添加执行权限
6 → henryyan@hy-hp ~/submd/ws/project1-b git:(master) X chmod +x ./bin/update-submodules.sh
7 → henryyan@hy-hp ~/submd/ws/project1-b git:(master) X ./bin/update-submodules.sh
8 libs/lib1
9 Already on 'master'
10 remote: Counting objects: 4, done.
11 remote: Compressing objects: 100% (2/2), done.
12 remote: Total 3 (delta 0), reused 0 (delta 0)
13 Unpacking objects: 100% (3/3), done.
14 From /home/henryyan/submd/repos/lib1
15 36ad12d..8c666d8 master -> origin/master
16 Updating 36ad12d..8c666d8
17 Fast-forward
18  README | 1 +
19  1 files changed, 1 insertions(+), 0 deletions(-)
20  create mode 100644 README
21 libs/lib2
22 Switched to branch 'master'
23 remote: Counting objects: 5, done.
24 remote: Compressing objects: 100% (2/2), done.
25 remote: Total 3 (delta 0), reused 0 (delta 0)
26 Unpacking objects: 100% (3/3), done.
27 From /home/henryyan/submd/repos/lib2
28 7290dce..e372b21 master -> origin/master
29 Updating 7290dce..e372b21
30 Fast-forward
31  lib2-features | 1 +
32  1 files changed, 1 insertions(+), 0 deletions(-)
33 → henryyan@hy-hp ~/submd/ws/project1-b git:(master) X git submodule
34 8c666d86531513dd1aebdf235f142adbac72c035 libs/lib1 (heads/master)
35 e372b21dffa611802c282278ec916b5418acebc2 libs/lib2 (heads/master)
36
37 → henryyan@hy-hp ~/submd/ws/project1-b git:(master) X git status
38 # On branch master
39 # Untracked files:
40 #   (use "git add <file>..." to include in what will be committed)
41 #
42 #   bin/
43 nothing added to commit but untracked files present (use "git add" to track)
44 </file>

```

更新之后的两个变化：

- git submodule的结果和project2的submodule commit id保持一致；
- project1-b不再提示new commits了。

现在可以把工具添加到仓库了，当然你可以很骄傲的分享给其他项目组的同事。

```

1 → henryyan@hy-hp ~/submd/ws/project1-b git:(master) X git add bin/update-submodules.sh
2 → henryyan@hy-hp ~/submd/ws/project1-b git:(master) X git commit -m "添加自动更新submodule的快捷脚
3 [master 756e788] 添加自动更新submodule的快捷脚本^^
4 1 files changed, 9 insertions(+), 0 deletions(-)
5 create mode 100755 bin/update-submodules.sh
6 → henryyan@hy-hp ~/submd/ws/project1-b git:(master) git push
7 Counting objects: 5, done.
8 Delta compression using up to 2 threads.
9 Compressing objects: 100% (3/3), done.
10 Writing objects: 100% (4/4), 625 bytes, done.
11 Total 4 (delta 0), reused 0 (delta 0)
12 Unpacking objects: 100% (4/4), done.
13 To /home/henryyan/submd/ws/../../repos/project1.git
14 8fcca50..756e788 master -> master

```

## 2.9 新进员工加入团队，一次性Clone项目和Submodules

一般人使用的时候都是使用如下命令：

```

1 git clone /path/to/repos/foo.git
2 git submodule init
3 git submodule update

```

新员工不耐烦了，嘴上不说但是心里想：怎么那么麻烦？

上面的命令简直弱爆了，直接一行命令搞定：

```

1 git clone --recursive /path/to/repos/foo.git

```

-recursive参数的含义：可以在clone项目时同时clone关联的submodules。

git help 对其解释:

```
--recursive, --recurse-submodules
After the clone is created, initialize all submodules within, using their default settings. This is equivalent to running git
submodule update --init --recursive immediately after the clone is finished. This option is ignored if the cloned repository
does not have a worktree/checkout (i.e. if any of --no-checkout/-n, --bare, or --mirror is given)
```

2.9.1 使用一键方式克隆project2

```
1 → henryyan@hy-hp ~/submd/ws git clone --recursive ../repos/project2.git project2-auto-clone-submodules
2 Cloning into project2-auto-clone-submodules...
3 done.
4 Submodule 'libs/lib1' (/home/henryyan/submd/repos/lib1.git) registered for path 'libs/lib1'
5 Submodule 'libs/lib2' (/home/henryyan/submd/repos/lib2.git) registered for path 'libs/lib2'
6 Cloning into libs/lib1...
7 done.
8 Submodule path 'libs/lib1': checked out '8c666d86531513dd1aebdf235f142adbac72c035'
9 Cloning into libs/lib2...
10 done.
11 Submodule path 'libs/lib2': checked out 'e372b21dffa611802c282278ec916b5418acebc2'
```

舒服.....

3. 移除Submodule

牢骚：搞不明白为什么git不设计一个类似：git submodule remove的命令呢？

我们从project1.git克隆一个项目用来练习移除submodule：

```
1 → henryyan@hy-hp ~/submd/ws git clone --recursive ../repos/project1.git project1-remove-submodules
2 Cloning into project1-remove-submodules...
3 done.
4 Submodule 'libs/lib1' (/home/henryyan/submd/repos/lib1.git) registered for path 'libs/lib1'
5 Submodule 'libs/lib2' (/home/henryyan/submd/repos/lib2.git) registered for path 'libs/lib2'
6 Cloning into libs/lib1...
7 done.
8 Submodule path 'libs/lib1': checked out '8c666d86531513dd1aebdf235f142adbac72c035'
9 Cloning into libs/lib2...
10 done.
11 Submodule path 'libs/lib2': checked out 'e372b21dffa611802c282278ec916b5418acebc2'
12 → henryyan@hy-hp ~/submd/ws cd !$
13 → henryyan@hy-hp ~/submd/ws cd project1-remove-submodules
```

3.1 Step by

1、删除git cache和物理文件夹

```
1 → henryyan@hy-hp ~/submd/ws/project1-remove-submodules git:(master) git rm -r --cached libs/
2 rm 'libs/lib1'
3 rm 'libs/lib2'
4 → henryyan@hy-hp ~/submd/ws/project1-remove-submodules git:(master) X rm -rf libs
```

2、删除.gitmodules的内容（或者整个文件） 因为本例只有两个子模块，直接删除文件：

```
1 → henryyan@hy-hp ~/submd/ws/project1-remove-submodules git:(master) X rm .gitmodules
```

如果仅仅删除某一个submodule那么打开.gitmodules文件编辑，删除对应submodule配置即可。

3、删除.git/config的submodule配置 源文件：

```
[core]
  repositoryformatversion = 0
  filemode = true
  bare = false
  logallrefupdates = true
[remote "origin"]
  fetch = +refs/heads/*:refs/remotes/origin/*
  url = /home/henryyan/submd/ws/../../repos/project1.git
[branch "master"]
  remote = origin
  merge = refs/heads/master
[submodule "libs/lib1"]
  url = /home/henryyan/submd/repos/lib1.git
[submodule "libs/lib2"]
  url = /home/henryyan/submd/repos/lib2.git
```

删除后:

```
[core]
  repositoryformatversion = 0
  filemode = true
```

```

bare = false
logallrefupdates = true
[remote "origin"]
    fetch = +refs/heads/*:refs/remotes/origin/*
    url = /home/henryyan/submd/ws/../../repos/project1.git
[branch "master"]
    remote = origin
    merge = refs/heads/master
```

4、提交更改

```

1  → henryyan@hy-hp  ~/submd/ws/project1-remove-submodules git:(master) X git status
2  # On branch master
3  # Changes to be committed:
4  #   (use "git reset HEAD <file>..." to unstage)
5  #
6  #   deleted:    libs/lib1
7  #   deleted:    libs/lib2
8  #
9  # Changes not staged for commit:
10 #   (use "git add/rm <file>..." to update what will be committed)
11 #   (use "git checkout -- <file>..." to discard changes in working directory)
12 #
13 #   deleted:    .gitmodules
14 #
15 → henryyan@hy-hp  ~/submd/ws/project1-remove-submodules git:(master) X git add .gitmodules
16 → henryyan@hy-hp  ~/submd/ws/project1-remove-submodules git:(master) X git commit -m "删除子模块lib1和lib2"
17 [master 5e2ee71] 删除子模块lib1和lib2
18   3 files changed, 0 insertions(+), 8 deletions(-)
19   delete mode 100644 .gitmodules
20   delete mode 160000 libs/lib1
21   delete mode 160000 libs/lib2
22 → henryyan@hy-hp  ~/submd/ws/project1-remove-submodules git:(master) git push
23 Counting objects: 3, done.
24 Delta compression using up to 2 threads.
25 Compressing objects: 100% (2/2), done.
26 Writing objects: 100% (2/2), 302 bytes, done.
27 Total 2 (delta 0), reused 0 (delta 0)
28 Unpacking objects: 100% (2/2), done.
29 To /home/henryyan/submd/ws/../../repos/project1.git
30   756e788..5e2ee71  master -> master
31 </file></file></file>
```

4. 结束语

对于Git Submodule来说在刚刚接触的时候多少会有点凌乱的赶紧，尤其是没有接触过svn:externals。

本文从开始创建项目到使用git submodule的每个步骤以及细节、原理逐一讲解，希望到此读者能驾驭它。

学会了Git submdoule你的项目中再也不会出现大量重复的资源文件、公共类库，更不会出现多个版本，甚至一个客户的多个项目风格存在各种差异。

本文的写作来源于工作中的实践，如果您对于某个做法有更好的办法还请赐教，希望能留下您宝贵的意见。

原创文章，转载请注明出处！ [Git Submoudle使用完整教程--咖啡兔](#)

原创文章，转载请注明：转载自：[Git Submodule使用完整教程](#)



25

21

喜欢

推荐

您可能也喜欢：



在Ubuntu(Debian)上安装最新版Git-咖啡兔 - HenryYan



新版Activiti Modeler发布以及教程 - 咖啡兔 - HenryYan



CAS单点登录(SSO)完整教程(2012-02-01更新) - 咖啡兔 -



咖啡兔 - CAS单点登录(SSO)完整教程(2012-02-01更新) - HenryYan



咖啡兔 - 在Ubuntu 12.04 LTS上安装Git1.7.10-rc4(中文文化) - HenryYan



咖啡兔 - Hudson教程系列 - HenryYan



工欲善其事，必先利其器 - 咖啡兔 - HenryYan



Activiti Explorer中文汉化 - 咖啡兔 - HenryYan

[无觅关联推荐\[?\]](#)





命令行乱到无法阅读

2015年11月5日    [← 回复](#)    [♥ 顶 \(2\)](#)    [➔ 转发](#)



咖啡兔



jiechic    2012年3月30日

1楼

好长，没看完，意思是不是，从其他仓库读取开发版公用库文件？

内容比较多，没办法；就是为了解决重复资源的应用并且保持一致的版本。

2012年3月30日    [← 回复](#)    [♥ 顶 \(2\)](#)    [➔ 转发](#)

45 条评论

6 条腾讯微博

最新    最早    最热



星辉Sunny



绿茶    2015年11月5日

1楼

命令行乱到无法阅读

不能同意更多！

4月17日    [← 回复](#)    [♥ 顶](#)    [➔ 转发](#)



richard\_ma

看完了，感觉作者写的很详细，多谢了

3月15日    [← 回复](#)    [♥ 顶](#)    [➔ 转发](#)



carlhan



carlhan    2月17日

1楼

谢谢 很好的文章。  
没有全部操作完。

但我的感受 怎么还不如这样：  
把project1和lib1都clone到工作目录ws中。然后在project1里面创建软链接libs/lib1 -> ../lib1。project1中添加libs/lib1到.gitignore中，在project1的README中说明或者提供一个脚本 更新lib1。

这样project1和lib1之间是分开的。每个repo commit都很方便。

2月17日    [← 回复](#)    [♥ 顶](#)    [➔ 转发](#)



carlhan

谢谢 很好的文章。  
没有全部操作完。

但我的感受 怎么还不如这样：  
把project1和lib1都clone到工作目录ws中。然后在project1里面创建软链接libs/lib1 -> ../lib1。project1中添加libs/lib1到.gitignore中，在project1的README中说明或者提供一个脚本 更新lib1。

2月17日    [← 回复](#)    [♥ 顶](#)    [➔ 转发](#)



JustForFun-GameWorld

写的这么认真，例子很好，佩服。如果最后能有个系统的总结就更好了。👍

2月3日    [← 回复](#)    [♥ 顶](#)    [➔ 转发](#)



victor



如此玄妙    2015年11月23日

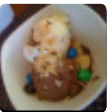
1楼

非常好的文章， 更新所有module 可以用这个啊

```
git submodule update --recursive
其他脚本都是不需要的
```

楼主是说其他的project如果用到相同的lib，才需要脚本的，不是说同一个project的更新

1月27日    [← 回复](#)    [♥ 顶](#)    [➔ 转发](#)



如此玄妙

非常好的文章， 更新所有module 可以用这个啊

```
git submodule update --recursive
其他脚本都是不需要的
```

2015年11月23日    [← 回复](#)    [♥ 顶](#)    [➔ 转发](#)

绿茶



命令行乱到无法阅读

2015年11月5日   回复   顶(2)   转发



呵呵哒

看到在project项目里面修改submodule项目就凌乱了，project项目负责人只管更新代码就好了，submodule应该由别人去管理吧

2015年10月4日   回复   顶   转发



test

test comment

2015年9月6日   回复   顶   转发



利源铝业

太啰嗦了，发文章出来也不过滤一下自己的SHELL环境，提交信息也弄上来浪费版面。。。

2015年8月7日   回复   顶   转发



烟尘

为什么我用 update 和 init 一点反应都没有，找了好久到不知道怎么更新啊

2015年7月28日   回复   顶   转发



clarkhan



咖啡兔 2014年1月3日

使用Git 1.7.11+版本的请不要再学习Submodule了，直接用Subtree

1楼

subtree 就是单纯的代码副本，带了历史记录吧。用了 - squash连历史记录都没了。而且每次pull、push操作也都要跟上仓库、分支，不是更玛法吗？

2015年6月9日   回复   顶   转发



leptune



awaken 2013年10月5日

请问你是如何实现在bash prompt里显示git (branchname)的？

1楼

其实博主的shel终端不是bash，是zsh。。。

2014年8月28日   回复   顶   转发



lihw

我觉得2.8 中，使用：

git submodule init

git submodule update

就可以更新到最新的版本了，没必要 git checkout, git pull.

2014年7月14日   回复   顶   转发



咖啡兔



文刀旋子 2014年5月22日

请问下你们现在用 submodule 还是 subtree

1楼

submodule

2014年5月22日   回复   顶   转发



文刀旋子

请问下你们现在用 submodule 还是 subtree

2014年5月22日   回复   顶   转发



台上莫漫夸-眼前何足算

submodule就是在你的git里面记录一下你引用到的其他模块的信息。

over。。。。

啥，这就over了。

对，木有错。就那么简单。

本质上和你手动copy一个reposit过来到你的工程目录，再拿个小本本记下来这个reposit是从哪里偷来的，木有啥区别。

只是，大家用多了发现这样不方便，于是就，给git一个submodule的命令。来便于操作。

-----

以上虽然有很多的胡扯部分，但是本质就是那样。下面分几点说明

-----

1、

你的工程和你submodule，没有啥关系。你只是用到人家的代码而已，为了便于管理git集成submodule命令，这个命令完全可以用其他方式（比如脚本）来代替。

submodule就一个git项目，该怎么更新同步就怎么更新同步，别老想着从你的工程去管理别人的submodule，考虑问题的思路就不对。

（我需要从linux内核项目来敲命令跟新Apache项目吗？？？？，需要吗？？？，我有可能2个都用啊）

虽然：1、submodule的仓库也是在你工程的.git/modules的文件夹下面

2、提供submodule这个便捷的命令

2、

你项目建立的时候一定只是引用某个版本的submodule（对应到commit），而不是某个分支。所以默认都是detached from XXXXX（commit id）；

因为，很可能人家提供submodule的人，过几天提交的文件会导致你当前工程完全不可用。。。。

3、上面默认的是最普遍的方式，但是也可能你的submodule就是自己维护的，可控！

那么这几个命令完全可以搞定（记不住木有关系，大不了多敲几行普通命令而已，俺就最头痛记命令）

（note：从overflow上面偷来的）

```
# add submodule to track master branch
git submodule add -b master [URL to Git repo];

# update your submodule
git submodule update --remote
```

2014年2月20日    ↩ 回复    ❤ 顶    ➡ 转发



天天Naitiz



咖啡兔 2014年1月3日  
使用Git 1.7.11+版本的请不要再学习Submodule了，直接用Subtree

1楼

subtree怎么感觉是拷贝一个副本过来主工程呢，好像不太好用。

2014年2月20日    ↩ 回复    ❤ 顶    ➡ 转发



咖啡兔

使用Git 1.7.11+版本的请不要再学习Submodule了，直接用Subtree

2014年1月3日    ↩ 回复    ❤ 顶    ➡ 转发



咖啡兔



fanhe 2013年12月23日  
我来说几句。  
博主shell的提示符太个性不利于阅读的人，一般来说提示符最简单至少都需要一个'\$'或者'#'，但是博主的是空格，看得太累了，  
无法快速地找出来真正执行的命令，实在没有看下去的欲望啊

1楼

多谢你的建议，后面的文章我会注意的

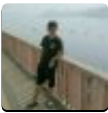
2013年12月23日    ↩ 回复    ❤ 顶    ➡ 转发



fanhe

我来说几句。  
博主shell的提示符太个性不利于阅读的人，一般来说提示符最简单至少都需要一个'\$'或者'#'，但是博主的是空格，看得太累了，  
无法快速地找出来真正执行的命令，实在没有看下去的欲望啊

2013年12月23日    ↩ 回复    ❤ 顶    ➡ 转发



SadieYuCN

Thank you very much !

2013年10月30日    ↩ 回复    ❤ 顶    ➡ 转发





lite3\_

终于看完了，这篇很有用啊。

2013年10月24日   回复   顶   转发



will111

awaken   2013年10月5日

1楼

请问你是如何实现在bash prompt里显示git (branchname)的?

[Showing your Git branch in your shell prompt](http://aaroncrane.co.uk/2009/03/git\_branch\_prompt/)

2013年10月18日   回复   顶   转发



awaken

请问你是如何实现在bash prompt里显示git (branchname)的?

2013年10月5日   回复   顶   转发



咖啡兔

葫芦和舒克   2013年7月31日

1楼

最近在用submodule做前端的架构，前端目前的工作仍处于需求不断的变化阶段，尤其是UI和交互面临这分久必合，合久必分的一种状态，所以在用GIT管理公用UI库时主要解决公用模块的统一维护和个性化定制的需求，读完你这篇文章后submodule更像一个软连接的实现，实现了统一维护，各自发布的需求。但是如果子模块有BUG，且他的OWNER不在时，需要FORK代码来提交新的修改，所以个性化时的处理比较麻烦。看了一下，还有subtree这个概念，能以子树合并的方式解决此类需求。谢谢你的笔记

你可以使用不同的branch管理个性化

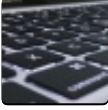
2013年7月31日   回复   顶   转发



葫芦和舒克

最近在用submodule做前端的架构，前端目前的工作仍处于需求不断的变化阶段，尤其是UI和交互面临这分久必合，合久必分的一种状态，所以在用GIT管理公用UI库时主要解决公用模块的统一维护和个性化定制的需求，读完你这篇文章后submodule更像一个软连接的实现，实现了统一维护，各自发布的需求。但是如果子模块有BUG，且他的OWNER不在时，需要FORK代码来提交新的修改，所以个性化时的处理比较麻烦。看了一下，还有subtree这个概念，能以子树合并的方式解决此类需求。谢谢你的笔记

2013年7月31日   回复   顶   转发



mac\_xiaoxu

太长大多，看起来有点乱

2013年7月5日   回复   顶   转发



yl

牛13! 学习了

2013年7月2日   回复   顶   转发



will

读完了，很好，实践出真知，很赞成博主的写作风格。（我今天cp了一个git init后的目录的文件到git管理的已有项目下，把.git也一起cp进去了于是就：`is in submodule`）。`.git/conf`中没有记录，`.git submodule`也不存在。郁闷了，上网搜到这里。学习了。

2013年6月19日   回复   顶   转发



赵庆

用 `git clone --recursive /path/to/repos/foo.git` 后会在当前文件夹下生成foo目录，单如果把它该成其他名字后，模块，不工作了，我用`git submodue init` 和 `git submodule update` 都不能同步这个改变。

2013年3月31日   回复   顶   转发



jovan

织 信息量好大，慢慢学习，赞一个.

2013年3月20日   回复   顶   转发



Bill Z

帮你更新一下， 我的git版本是：`git version 1.7.10.4`

- 现在不需要你说的 `git submodue add git-url local_dir` 后还要`git submodule init` 后才能`git submodule update` 的了
- submoude `.git`的位置现在是放在顶层的`.git`的目录的`modules`目录下面，这会影响到你说的删除submodule方法（在删除模块的时候还得到这个`modues`目录去删对应的模块了）
- 就是奇怪，还是没看到一个一次干净删除模块的命令。

我刚刚开始review这个功能，能多大程度帮助项目。

qq:1517537094

2013年1月11日

回复

顶

转发



Bill Z

学习，练习中。。。

2013年1月11日

回复

顶

转发



胡阳

收藏了，慢慢看

2013年1月7日


回复

顶

转发



txworking

紫煌 2013年1月1日

1楼

现在我更新submodule统一用：``git submodule foreach git pull``。  
``git submodule update``似乎除了和git clone配合的那种情况外就根本没用了。

但是这样submodule就直接到最新版本了，和主项目记录的commit id不一致。  
我用git pull & git submodule update倒是都同步了，但是submodule会变成no branch状态，这个不知道为啥。

2013年1月6日


回复

顶

转发



咖啡兔

紫煌 2013年1月1日

1楼

现在我更新submodule统一用：``git submodule foreach git pull``。  
``git submodule update``似乎除了和git clone配合的那种情况外就根本没用了。

学习了，之前我还自己写脚本更新每个submodule，我会把这个命令更新到博文

2013年1月2日

回复

顶

转发



紫煌

现在我更新submodule统一用：``git submodule foreach git pull``。  
``git submodule update``似乎除了和git clone配合的那种情况外就根本没用了。

2013年1月1日

回复

顶

转发



leslie

写得非常好，很顺，谢谢.

2012年12月5日

回复

顶

转发



ycsunjane@gmail.com

讲解很明晰，谢谢！我认为只要了解commit id就很好理解了submodule了！

2012年11月23日


回复

顶

转发



咖啡兔

Neo4j 2012年9月10日

1楼

写得真的很好，谢谢你。👍

感谢支持！

2012年9月10日

回复

顶

转发



Neo4j

写得真的很好，谢谢你。👍

2012年9月10日


回复

顶

转发



咖啡兔

jiechic 2012年3月30日

1楼

好长，没看完，意思是不是，从其他仓库读取开发版公用库文件？

内容比较多，没办法；就是为了解决重复资源的应用并且保持一致的版本。

2012年3月30日

回复

顶(2)

转发



jiechic

好长，没看完，意思是不是，从其他仓库读取开发版公用库文件？

2012年3月30日

回复

顶

转发



说点什么吧...



发布

多说强力驱动

Copyright © 2012~2013 咖啡兔 | Henry Yan. Hosted by [GitHub](#) and powered by [Jekyll](#). Templates from [Michael Bleigh](#).

