The 2016 Git User's Survey is now up! ***12 September — 20 October 2016.***

Please devote a few moments of your time to fill out the simple questionnaire. It will help the Git community understand your needs, what you like about Git (and what you don't), and help us improve it in general. The results will be published at the GitSurvey2016 wiki page.


--distributed-even-if-your-workflow-isnt

Search entire site...

Download this book in PDF, mobi, or ePub form for free.

This book is translated into Deutsch, 简体中文, 正體中文, Français, 日本語, Nederlands, Русский, 한국어, Português (Brasil) and Čeština.

Partial translations available in Arabic, Español, Indonesian, Italiano, Suomi, Македонски, Polski and Türkçe.

Translations started for Azərbaycan dili, Беларуская, Català, Esperanto, Español (Nicaragua), فارسی, हिन्दी, Magyar, Norwegian Bokmål, Română, Српски, ภาษาไทย, Tiếng Việt, Українська and Ўзбекча.

The source of this book is hosted on GitHub.
Patches, suggestions and comments are welcome.

Chapters ▾

2nd Edition

7.11 Git 工具 - 子模块

子模块

有种情况我们经常会遇到：某个工作中的项目需要包含并使用另一个项目。 也许是第三方库，或者你独立开发的，用于多个父项目的库。 现在问题来了：你想要把它们当做两个独立的项目，同时又想在一个项目中使用另一个。

我们举一个例子。 假设你正在开发一个网站然后创建了 Atom 订阅。 你决定使用一个库，而不是写自己的 Atom 生成代码。 你可能不得不通过 CPAN 安装或 Ruby gem 来包含共享库中的代码，或者将源代码直接拷贝到自己的项目中。 如果将这个库包含进来，那么无论用何种方式都很难定制它，部署则更加困难，因为你必须确保每一个客户端都包含该库。 如果将代码复制到自己的项目中，那么你做的任何自定义修改都会使合并上游

的改动变得困难。

Git 通过子模块来解决这个问题。 子模块允许你将一个 Git 仓库作为另一个 Git 仓库的子目录。 它能让你将另一个仓库克隆到自己的项目中，同时还保持提交的独立。

开始使用子模块

我们将要演示如何在一个被分成一个主项目与几个子项目的项目上开发。

我们首先将一个已存在的 Git 仓库添加为正在工作的仓库的子模块。 你可以通过在 `git submodule add` 命令后面加上想要跟踪的项目 URL 来添加新的子模块。 在本例中，我们将会添加一个名为 “DbConnector” 的库。

```
$ git submodule add https://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`，你会注意到几件事。

```
$ 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 = https://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。

Note

由于 .gitmodules 文件中的 URL 是人们首先尝试克隆/拉取的地方，因此请尽可能确保你使用的URL 大家都能访问。 例如，若你要使用的推送 URL 与他人的拉取 URL 不同，那么请使用他人能访问到的 URL。 你也可以根据自己的需要，通过在本地执行 `git config submodule.DbConnector.url <私有URL>` 来覆盖这个选项的值。 如果可行的话，一个相对路径会很有帮助。

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 = https://github.com/chaconinc/DbConnector
Submodule DbConnector 0000000...c3f01dc (new submodule)
```

当你提交时，会看到类似下面的信息：

```
$ git commit -am 'added DbConnector module'
[master fb9093c] added DbConnector module
 2 files changed, 4 insertions(+)
 create mode 100644 .gitmodules
 create mode 160000 DbConnector
```

注意 DbConnector 记录的 `160000` 模式。 这是 Git 中的一种特殊模式，它本质上意味着你是将一次提交记作一项目录记录的，而非将它记录成一个子目录或者一个文件。

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 https://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' (https://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 https://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' (https://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'
```

在包含子模块的项目上工作

现在我们有一份包含子模块的项目副本，我们将会同时在主项目和子模块项目上与队员协作。

拉取上游修改

在项目中使用子模块的最简模型，就是只使用子项目并不时地获取更新，而并不在你的检出中进行任何更改。我们来看一个简单的例子。

如果想要在子模块中查看新工作，可以进入到目录中运行 git fetch 与 git merge，合并上游分支来更新本地代码。

```
$ git fetch
From https://github.com/chaconinc/DbConnector
   c3f01dc..d0354fc  master     -> origin/master
$ git merge origin/master
Updating c3f01dc..d0354fc
Fast-forward
 scripts/connect.sh | 1 +
 src/db.c           | 1 +
 2 files changed, 2 insertions(+)
```

如果你现在返回到主项目并运行 git diff --submodule，就会看到子模块被更新的同时获得了一个包含新添加提交的列表。 如果你不想每次运行 git diff 时都输入 --submodle，那么可以将 diff.submodule 设置为 “log” 来将其作为默认行为。

```
$ git config --global diff.submodule log
$ git diff
Submodule DbConnector c3f01dc..d0354fc:
  > more efficient db routine
  > better connection routine
```

如果在此时提交，那么你会将子模块锁定为其他人更新时的新代码。

如果你不想在子目录中手动抓取与合并，那么还有种更容易的方式。 运行 `git submodule update --remote`，Git 将会进入子模块然后抓取并更新。

```
$ git submodule update --remote DbConnector
remote: Counting objects: 4, done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 4 (delta 2), reused 4 (delta 2)
Unpacking objects: 100% (4/4), done.
From https://github.com/chaconinc/DbConnector
   3f19983..d0354fc  master     -> origin/master
Submodule path 'DbConnector': checked out 'd0354fc054692d3906c85c3af05ddce39a1c0644'
```

此命令默认会假定你想要更新并检出子模块仓库的 `master` 分支。 不过你也可以设置为想要的其他分支。 例如，你想要 DbConnector 子模块跟踪仓库的 “stable” 分支，那么既可以在 `.gitmodules` 文件中设置（这样其他人也可以跟踪它），也可以只在本地的 `.git/config` 文件中设置。 让我们在 `.gitmodules` 文件中设置它：

```
$ git config -f .gitmodules submodule.DbConnector.branch stable

$ git submodule update --remote
remote: Counting objects: 4, done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 4 (delta 2), reused 4 (delta 2)
Unpacking objects: 100% (4/4), done.
From https://github.com/chaconinc/DbConnector
   27cf5d3..c87d55d  stable -> origin/stable
Submodule path 'DbConnector': checked out 'c87d55d4c6d4b05ee34fbc8cb6f7bf4585ae6687'
```

如果不用 `-f .gitmodules` 选项，那么它只会为你做修改。但是在仓库中保留跟踪信息更有意义一些，因为其他人也可以得到同样的效果。

这时我们运行 `git status`，Git 会显示子模块中有 “新提交”。

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

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

  modified:   .gitmodules
  modified:   DbConnector (new commits)

no changes added to commit (use "git add" and/or "git commit -a")
```

如果你设置了配置选项 `status.submodulesummary`，Git 也会显示你的子模块的更改摘要：

```
$ git config status.submodulesummary 1

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

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
```

```
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   .gitmodules
        modified:   DbConnector (new commits)

Submodules changed but not updated:

* DbConnector c3f01dc...c87d55d (4):
  > catch non-null terminated lines
```

这时如果运行 `git diff`，可以看到我们修改了 .gitmodules 文件，同时还有几个已拉取的提交需要提交到我们自己的子模块项目中。

```
$ git diff
diff --git a/.gitmodules b/.gitmodules
index 6fc0b3d..fd1cc29 100644
--- a/.gitmodules
+++ b/.gitmodules
@@ -1,3 +1,4 @@
 [submodule "DbConnector"]
        path = DbConnector
        url = https://github.com/chaconinc/DbConnector
+       branch = stable
 Submodule DbConnector c3f01dc..c87d55d:
  > catch non-null terminated lines
  > more robust error handling
  > more efficient db routine
  > better connection routine
```

这非常有趣，因为我们可以直接看到将要提交到子模块中的提交日志。 提交之后，你也可以运行 `git log -p` 查看这个信息。

```
$ git log -p --submodule
commit 0a24cfc121a8a3c118e0105ae4ae4c00281cf7ae
Author: Scott Chacon <schacon@gmail.com>
Date:   Wed Sep 17 16:37:02 2014 +0200

    updating DbConnector for bug fixes

diff --git a/.gitmodules b/.gitmodules
index 6fc0b3d..fd1cc29 100644
--- a/.gitmodules
+++ b/.gitmodules
@@ -1,3 +1,4 @@
 [submodule "DbConnector"]
        path = DbConnector
        url = https://github.com/chaconinc/DbConnector
+       branch = stable
Submodule DbConnector c3f01dc..c87d55d:
  > catch non-null terminated lines
  > more robust error handling
  > more efficient db routine
  > better connection routine
```

当运行 `git submodule update --remote` 时，Git 默认会尝试更新所有子模块，所以如果有很多子模块的话，你可以传递想要更新的子模块的名字。

在子模块上工作

你很有可能正在使用子模块，因为你确实想在子模块中编写代码的同时，还想在主项目上编写代码（或者跨子模块工作）。 否则你大概只能用简单的依赖管理系统（如 Maven 或 Rubygems）来替代了。

现在我们将通过一个例子来演示如何在子模块与主项目中同时做修改，以及如何同时提交与发布那些修改。

到目前为止，当我们运行 `git submodule update` 从子模块仓库中抓取修改时，Git 将会获得这些改动并更新子目录中的文件，但是会将子仓库留在一个称作 “游离的 HEAD” 的状态。 这意味着没有本地工作分支（例如 “master”）跟踪改动。 所以你做的任何改动都不会被跟踪。

为了将子模块设置得更容易进入并修改，你需要做两件事。 首先，进入每个子模块并检出其相应的工作分支。 接着，若你做了更改就需要告诉 Git 它该做什么，然后运行 `git submodule update --remote` 来从上游拉取新工作。 你可以选择将它们合并到你的本地工作中，也可以尝试将你的工作变基到新的更改上。

首先，让我们进入子模块目录然后检出一个分支。

```
$ git checkout stable
Switched to branch 'stable'
```

然后尝试用 “merge” 选项。 为了手动指定它，我们只需给 `update` 添加 `--merge` 选项即可。 这时我们将会看到服务器上的这个子模块有一个改动并且它被合并了进来。

```
$ git submodule update --remote --merge
remote: Counting objects: 4, done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 4 (delta 2), reused 4 (delta 2)
Unpacking objects: 100% (4/4), done.
From https://github.com/chaconinc/DbConnector
   c87d55d..92c7337  stable     -> origin/stable
Updating c87d55d..92c7337
Fast-forward
 src/main.c | 1 +
 1 file changed, 1 insertion(+)
Submodule path 'DbConnector': merged in '92c7337b30ef9e0893e758dac2459d07362ab5ea'
```

如果我们进入 DbConnector 目录，可以发现新的改动已经合并入本地 `stable` 分支。 现在让我们看看当我们对库做一些本地的改动而同时其他人推送另外一个修改到上游时会发生什么。

```
$ cd DbConnector/
$ vim src/db.c
$ git commit -am 'unicode support'
[stable f906e16] unicode support
 1 file changed, 1 insertion(+)
```

如果我们现在更新子模块，就会看到当我们在本地做了更改时上游也有一个改动，我们需要将它并入本地。

```
$ git submodule update --remote --rebase
First, rewinding head to replay your work on top of it...
Applying: unicode support
Submodule path 'DbConnector': rebased into '5d60ef9bbebf5a0c1c1050f242ceeb54ad58da94'
```

如果你忘记 `--rebase` 或 `--merge`，Git 会将子模块更新为服务器上的状态。并且会将项目重置为一个游离的 HEAD 状态。

```
$ git submodule update --remote
Submodule path 'DbConnector': checked out '5d60ef9bbebf5a0c1c1050f242ceeb54ad58da94'
```

即便这真的发生了也不要紧，你只需回到目录中再次检出你的分支（即还包含着你的工作的分支）然后手动地合并或变基 `origin/stable`（或任何一个你想要的远程分支）就行了。

如果你没有提交子模块的改动，那么运行一个子模块更新也不会出现问题，此时 Git 会只抓取更改而并不会覆盖子模块目录中未保存的工作。

```
$ git submodule update --remote
remote: Counting objects: 4, done.
```

```
remote: Compressing objects: 100% (3/3), done.
remote: Total 4 (delta 0), reused 4 (delta 0)
Unpacking objects: 100% (4/4), done.
From https://github.com/chaconinc/DbConnector
   5d60ef9..c75e92a  stable     -> origin/stable
error: Your local changes to the following files would be overwritten by checkout:
        scripts/setup.sh
Please, commit your changes or stash them before you can switch branches.
Aborting
Unable to checkout 'c75e92a2b3855c9e5b66f915308390d9db204aca' in submodule path 'DbConnector'
```

如果你做了一些与上游改动冲突的改动，当运行更新时 Git 会让你知道。

```
$ git submodule update --remote --merge
Auto-merging scripts/setup.sh
CONFLICT (content): Merge conflict in scripts/setup.sh
Recorded preimage for 'scripts/setup.sh'
Automatic merge failed; fix conflicts and then commit the result.
Unable to merge 'c75e92a2b3855c9e5b66f915308390d9db204aca' in submodule path 'DbConnector'
```

你可以进入子模块目录中然后就像平时那样修复冲突。

发布子模块改动

现在我们的子模块目录中有一些改动。 其中有一些是我们通过更新从上游引入的，而另一些是本地生成的，由于我们还没有推送它们，所以对任何其他人都不可用。

```
$ git diff
Submodule DbConnector c87d55d..82d2ad3:
  > Merge from origin/stable
  > updated setup script
  > unicode support
  > remove unnecessary method
  > add new option for conn pooling
```

如果我们在主项目中提交并推送但并不推送子模块上的改动，其他尝试检出我们修改的人会遇到麻烦，因为他们无法得到依赖的子模块改动。 那些改动只存在于我们本地的拷贝中。

为了确保这不会发生，你可以让 Git 在推送到主项目前检查所有子模块是否已推送。 `git push` 命令接受可以设置为 “check” 或 “on-demand” 的 `--recurse-submodules` 参数。 如果任何提交的子模块改动没有推送那么 “check” 选项会直接使 `push` 操作失败。

```
$ git push --recurse-submodules=check
The following submodule paths contain changes that can
not be found on any remote:
  DbConnector

Please try

        git push --recurse-submodules=on-demand

or cd to the path and use

        git push

to push them to a remote.
```

如你所见，它也给我们了一些有用的建议，指导接下来该如何做。 最简单的选项是进入每一个子模块中然后手动推送到远程仓库，确保它们能被外部访问到，之后再次尝试这次推送。

另一个选项是使用 “on-demand” 值，它会尝试为你这样做。

```
$ git push --recurse-submodules=on-demand
Pushing submodule 'DbConnector'
Counting objects: 9, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (8/8), done.
Writing objects: 100% (9/9), 917 bytes | 0 bytes/s, done.
Total 9 (delta 3), reused 0 (delta 0)
To https://github.com/chaconinc/DbConnector
   c75e92a..82d2ad3  stable -> stable
Counting objects: 2, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (2/2), 266 bytes | 0 bytes/s, done.
Total 2 (delta 1), reused 0 (delta 0)
To https://github.com/chaconinc/MainProject
   3d6d338..9a377d1  master -> master
```

如你所见，Git 进入到 DbConnector 模块中然后在推送主项目前推送了它。 如果那个子模块因为某些原因推送失败，主项目也会推送失败。

合并子模块改动

如果你其他人同时改动了一个子模块引用，那么可能会遇到一些问题。 也就是说，如果子模块的历史已经分叉并且在父项目中分别提交到了分叉的分支上，那么你需要做一些工作来修复它。

如果一个提交是另一个的直接祖先（一个快进式合并），那么 Git 会简单地选择之后的提交来合并，这样没什么问题。

不过，Git 甚至不会尝试去进行一次简单的合并。 如果子模块提交已经分叉且需要合并，那你会得到类似下面的信息：

```
$ git pull
remote: Counting objects: 2, done.
remote: Compressing objects: 100% (1/1), done.
remote: Total 2 (delta 1), reused 2 (delta 1)
Unpacking objects: 100% (2/2), done.
From https://github.com/chaconinc/MainProject
   9a377d1..eb974f8  master     -> origin/master
Fetching submodule DbConnector
warning: Failed to merge submodule DbConnector (merge following commits not found)
Auto-merging DbConnector
CONFLICT (submodule): Merge conflict in DbConnector
Automatic merge failed; fix conflicts and then commit the result.
```

所以本质上 Git 在这里指出了子模块历史中的两个分支记录点已经分叉并且需要合并。 它将其解释为 “merge following commits not found”（未找到接下来需要合并的提交），虽然这有点令人困惑，不过之后我们会解释为什么是这样。

为了解决这个问题，你需要弄清楚子模块应该处于哪种状态。 奇怪的是，Git 并不会给你多少能帮你摆脱困境的信息，甚至连两边提交历史中的 SHA-1 值都没有。 幸运的是，这很容易解决。 如果你运行 `git diff`，就会得到试图合并的两个分支中记录的提交的 SHA-1 值。

```
$ git diff
diff --cc DbConnector
index eb41d76,c771610..0000000
--- a/DbConnector
+++ b/DbConnector
```

所以，在本例中，`eb41d76` 是我们的子模块中大家共有的提交，而 `c771610` 是上游拥有的提交。 如果我们进入

子模块目录中，它应该已经在 eb41d76 上了，因为合并没有动过它。 如果不是的话，无论什么原因，你都可以简单地创建并检出一个指向它的分支。

来自另一边的提交的 SHA-1 值比较重要。 它是需要你来合并解决的。 你可以尝试直接通过 SHA-1 合并，也可以为它创建一个分支然后尝试合并。 我们建议后者，哪怕只是为了一个更漂亮的合并提交信息。

所以，我们将会进入子模块目录，基于 `git diff` 的第二个 SHA 创建一个分支然后手动合并。

```
$ cd DbConnector

$ git rev-parse HEAD
eb41d764bccf88be77aced643c13a7fa86714135

$ git branch try-merge c771610
(DbConnector) $ git merge try-merge
Auto-merging src/main.c
CONFLICT (content): Merge conflict in src/main.c
Recorded preimage for 'src/main.c'
Automatic merge failed; fix conflicts and then commit the result.
```

我们在这儿得到了一个真正的合并冲突，所以如果想要解决并提交它，那么只需简单地通过结果来更新主项目。

```
$ vim src/main.c ❶
$ git add src/main.c
$ git commit -am 'merged our changes'
Recorded resolution for 'src/main.c'.
[master 9fd905e] merged our changes

$ cd .. ❷
$ git diff ❸
diff --cc DbConnector
index eb41d76,c771610..0000000
--- a/DbConnector
+++ b/DbConnector
@@@ -1,1 -1,1 +1,1 @@@
- Subproject commit eb41d764bccf88be77aced643c13a7fa86714135
 -Subproject commit c77161012afbbe1f58b5053316ead08f4b7e6d1d
++Subproject commit 9fd905e5d7f45a0d4cbc43d1ee550f16a30e825a
$ git add DbConnector ❹

$ git commit -m "Merge Tom's Changes" ❺
[master 10d2c60] Merge Tom's Changes
```

❶

首先解决冲突

❷

然后返回到主项目目录中

❸

再次检查 SHA-1 值

❹

解决冲突的子模块记录

❺

提交我们的合并

这可能会让你有点儿困惑，但它确实不难。

有趣的是，Git 还能处理另一种情况。 如果子模块目录中存在着这样一个合并提交，它的历史中包含了的两边的提交，那么 Git 会建议你将它作为一个可行的解决方案。 它看到有人在子模块项目的某一点上合并了包含这两次提交的分支，所以你可能想要那个。

这就是为什么前面的错误信息是 “merge following commits not found”，因为它不能 这样 做。 它让人困惑是因为谁能想到它会尝试这样做?

如果它找到了一个可以接受的合并提交，你会看到类似下面的信息：

```
$ git merge origin/master
warning: Failed to merge submodule DbConnector (not fast-forward)
Found a possible merge resolution for the submodule:
 9fd905e5d7f45a0d4cbc43d1ee550f16a30e825a: > merged our changes
If this is correct simply add it to the index for example
by using:

  git update-index --cacheinfo 160000 9fd905e5d7f45a0d4cbc43d1ee550f16a30e825a "DbConnector"

which will accept this suggestion.
Auto-merging DbConnector
CONFLICT (submodule): Merge conflict in DbConnector
Automatic merge failed; fix conflicts and then commit the result.
```

它会建议你更新索引，就像你运行了 `git add` 那样，这样会清除冲突然后提交。不过你可能不应该这样做。你可以轻松地进入子模块目录，查看差异是什么，快进到这次提交，恰当地测试，然后提交它。

```
$ cd DbConnector/
$ git merge 9fd905e
Updating eb41d76..9fd905e
Fast-forward

$ cd ..
$ git add DbConnector
$ git commit -am 'Fast forwarded to a common submodule child'
```

这些命令完成了同一件事，但是通过这种方式你至少可以验证工作是否有效，以及当你在完成时可以确保子模块目录中有你的代码。

子模块技巧

你可以做几件事情来让用子模块工作轻松一点儿。

子模块遍历

有一个 `foreach` 子模块命令，它能在每一个子模块中运行任意命令。 如果项目中包含了大量子模块，这会非常有用。

例如，假设我们想要开始开发一项新功能或者修复一些错误，并且需要在几个子模块内工作。 我们可以轻松地保存所有子模块的工作进度。

```
$ git submodule foreach 'git stash'
Entering 'CryptoLibrary'
No local changes to save
```

```
Entering 'DbConnector'
Saved working directory and index state WIP on stable: 82d2ad3 Merge from origin/stable
HEAD is now at 82d2ad3 Merge from origin/stable
```

然后我们可以创建一个新分支，并将所有子模块都切换过去。

```
$ git submodule foreach 'git checkout -b featureA'
Entering 'CryptoLibrary'
Switched to a new branch 'featureA'
Entering 'DbConnector'
Switched to a new branch 'featureA'
```

你应该明白。 能够生成一个主项目与所有子项目的改动的统一差异是非常有用的。

```
$ git diff; git submodule foreach 'git diff'
Submodule DbConnector contains modified content
diff --git a/src/main.c b/src/main.c
index 210f1ae..1f0acdc 100644
--- a/src/main.c
+++ b/src/main.c
@@ -245,6 +245,8 @@ static int handle_alias(int *argcp, const char ***argv)

       commit_pager_choice();

+      url = url_decode(url_orig);
+
       /* build alias_argv */
       alias_argv = xmalloc(sizeof(*alias_argv) * (argc + 1));
       alias_argv[0] = alias_string + 1;
Entering 'DbConnector'
diff --git a/src/db.c b/src/db.c
index 1aaefb6..5297645 100644
--- a/src/db.c
+++ b/src/db.c
@@ -93,6 +93,11 @@ char *url_decode_mem(const char *url, int len)
        return url_decode_internal(&url, len, NULL, &out, 0);
 }

+char *url_decode(const char *url)
+{
+       return url_decode_mem(url, strlen(url));
+}
+
 char *url_decode_parameter_name(const char **query)
 {
        struct strbuf out = STRBUF_INIT;
```

在这里，我们看到子模块中定义了一个函数并在主项目中调用了它。 这明显是个简化了的例子，但是希望它能让你明白这种方法的用处。

有用的别名

你可能想为其中一些命令设置别名，因为它们可能会非常长而你又不能设置选项作为它们的默认选项。 我们在 Git 别名 介绍了设置 Git 别名，但是如果你计划在 Git 中大量使用子模块的话，这里有一些例子。

```
$ git config alias.sdiff '!'"git diff && git submodule foreach 'git diff'"
$ git config alias.spush 'push --recurse-submodules=on-demand'
$ git config alias.supdate 'submodule update --remote --merge'
```

这样当你想要更新子模块时可以简单地运行 `git supdate`，或 `git spush` 检查子模块依赖后推送。

子模块的问题

然而使用子模块还是有一些小问题。

例如在有子模块的项目中切换分支可能会造成麻烦。 如果你创建一个新分支，在其中添加一个子模块，之后切换到没有该子模块的分支上时，你仍然会有一个还未跟踪的子模块目录。

```
$ git checkout -b add-crypto
Switched to a new branch 'add-crypto'

$ git submodule add https://github.com/chaconinc/CryptoLibrary
Cloning into 'CryptoLibrary'...
...

$ git commit -am 'adding crypto library'
[add-crypto 4445836] adding crypto library
 2 files changed, 4 insertions(+)
 create mode 160000 CryptoLibrary

$ git checkout master
warning: unable to rmdir CryptoLibrary: Directory not empty
Switched to branch 'master'
Your branch is up-to-date with 'origin/master'.

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

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        CryptoLibrary/

nothing added to commit but untracked files present (use "git add" to track)
```

移除那个目录并不困难，但是有一个目录在那儿会让人有一点困惑。 如果你移除它然后切换回有那个子模块的分支，需要运行 `submodule update --init` 来重新建立和填充。

```
$ git clean -fdx
Removing CryptoLibrary/

$ git checkout add-crypto
Switched to branch 'add-crypto'

$ ls CryptoLibrary/

$ git submodule update --init
Submodule path 'CryptoLibrary': checked out 'b8dda6aa182ea4464f3f3264b11e0268545172af'

$ ls CryptoLibrary/
Makefile        includes        scripts         src
```

再说一遍，这真的不难，只是会让人有点儿困惑。

另一个主要的告诫是许多人遇到了将子目录转换为子模块的问题。 如果你在项目中已经跟踪了一些文件，然后想要将它们移动到一个子模块中，那么请务必小心，否则 Git 会对你发脾气。 假设项目内有一些文件在子目录中，你想要将其转换为一个子模块。 如果删除子目录然后运行 `submodule add`，Git 会朝你大喊：

```
$ rm -Rf CryptoLibrary/
$ git submodule add https://github.com/chaconinc/CryptoLibrary
'CryptoLibrary' already exists in the index
```

你必须要先取消暂存 `CryptoLibrary` 目录。 然后才可以添加子模块：

```
$ git rm -r CryptoLibrary
```

```
$ git submodule add https://github.com/chaconinc/CryptoLibrary
Cloning into 'CryptoLibrary'...
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.
```

现在假设你在一个分支下做了这样的工作。 如果尝试切换回的分支中那些文件还在子目录而非子模块中时 - 你会得到这个错误：

```
$ git checkout master
error: The following untracked working tree files would be overwritten by checkout:
  CryptoLibrary/Makefile
  CryptoLibrary/includes/crypto.h
  ...
Please move or remove them before you can switch branches.
Aborting
```

你可以通过 `check -f` 来强制切换，但是要小心，如果其中还有未保存的修改，这个命令会把它们覆盖掉。

```
$ git checkout -f master
warning: unable to rmdir CryptoLibrary: Directory not empty
Switched to branch 'master'
```

当你切换回来之后，因为某些原因你得到了一个空的 `CryptoLibrary` 目录，并且 `git submodule update` 也无法修复它。 你需要进入到子模块目录中运行 `git checkout .` 来找回所有的文件。 你也可以通过 `submodule foreach` 脚本来为多个子模块运行它。

要特别注意的是，近来子模块会将它们的所有 Git 数据保存在顶级项目的 `.git` 目录中，所以不像旧版本的 Git，摧毁一个子模块目录并不会丢失任何提交或分支。

拥有了这些工具，使用子模块会成为可以在几个相关但却分离的项目上同时开发的相当简单有效的方法。

prev | next