Chris Jean
WordPress developer for Automattic, Linux fan, and all-around nerd

Git Submodules: Adding, Using, Removing, Updating

I've spent a little more than a month working with Git now. I can honestly say that while there are many things that I like about Git, there are just as many things that I personally find to be a pain in the butt.

Submodules specifically have managed to be a thorn in my side on many occasions. While the concept of submodules is simple, figuring out how to actually work with them can be a chore. I say "figuring out" because not everything about working with submodules is well documented. I'll cover two of the more difficult things to figure out: removing and updating submodules from your repository.

What are Submodules?

The concept of submodules is brilliant. It essentially allows you to attach an external repository inside another repository at a specific path. In order to illustrate the value of submodules, it will probably be helpful for me to explain how I am using them.

My profession is working with WordPress themes. Basically, I develop feature enhancements to the themes. I develop the code for these enhancements in modules that are completely contained in their own folder. This allows for the code to be easily added to other themes and also simplifies code updates/improvements as the code for specific features is consistent across all the themes that use that specific module.

Each theme that we produce is kept in its own Git repository. In addition, I've created a separate repository for each one of these feature modules. Rather than actually putting the feature module code directly into the theme repositories, I simply add the needed feature module repositories as submodules.

For example, we have a theme called FlexxBold. FlexxBold currently includes a total of seven submodules: billboard, contact-page-plugin, featured-images, feedburner-widget, file-utility, flexx-layout-editor, and tutorials. Since I'm using submodules, the code can be pulled directly from the relevant submodule repositories rather than requiring me to manually update each individual theme repository.

As I mentioned before, not everything in Git is easy to work with. There are four main functions you will need to understand in order to work with Git submodules. In order, you will need to know how to add, make use of, remove, and update submodules. I'll cover each of those uses below.

Adding Submodules to a Git Repository

Fortunately, adding a submodule to a git repository is actually quite simple. For example, if I'm in the repository working directory of a new theme called SampleTheme and need to add the billboard repository to the path lib/billboard, I can do so with the following command:

```
[user@office SampleTheme]$ git submodule add git@mygithost:billboard lib/billboard
Initialized empty Git repository in ~/git_dev/SampleTheme/lib/billboard/.git/
remote: Counting objects: 1006, done.
remote: Compressing objects: 100% (978/978), done.
remote: Total 1006 (delta 631), reused 0 (delta 0)
Receiving objects: 100% (1006/1006), 408.22 KiB, done.
Resolving deltas: 100% (631/631), done.
```

There are three main parts to this command:

- `git submodule add` – This simply tells Git that we are adding a submodule. This syntax will always remain the same.

- `git@mygithost:billboard` – This is the external repository that is to be added as a submodule. The exact syntax will vary depending on the setup of the Git repository you are connecting to. You need to ensure that you have the ability to clone the given repository.
- `lib/billboard` – This is the path where the submodule repository will be added to the main repository.

Let's check to see how the repository is doing.

```
[user@office SampleTheme]$ git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       new file:   .gitmodules
#       new file:   lib/billboard
#
```

Notice how the supplied path was created and added to the changes to be committed. In addition, a new file called .gitmodules was created. This new file contains the details we supplied about the new submodule. Out of curiosity, let's check out the contents of that new file:

```
[user@office SampleTheme]$ cat .gitmodules
[submodule "lib/billboard"]
path = lib/billboard
url = git@mygithost:billboard
```

Being able to modify this file later will come in handy later.

All that is left to do now is to commit the changes and then push the commit to a remote system if necessary.

Using Submodules

Having submodules in a repository is great and all, but if I look in my repository, all I have is an empty folder rather than the actual contents of the submodule's repository. In order to fill in the submodule's path with the files from the external repository, you must first initialize the submodules and then update them.

Note: This has changed in newer versions of Git. Now the submodule's repository will be cloned with master checked out. If that repository also has submodules, then your submodule's submodules will have to be populated by following the steps below from within your project's submodule directory (confusing yet?).

For instance, if you are working in project called `phone-app`, add a submodule called `graphics-lib`, and `graphics-lib` has a submodule called `renderer`, when you add `graphics-lib` to `phone-app`, the `phone-app/graphics-lib` directory will be populated as a cloned repo but the `phone-app/graphics-lib/renderer` directory will be empty. To populate `phone-app/graphics-lib/renderer`, first change directories to `phone-app/graphics-lib` and follow the instructions below.

First, we need to initialize the submodule(s). We can do that with the following command:

```
[user@office SampleTheme]$ git submodule init
Submodule 'lib/billboard' (git@mygithost:billboard) registered for path 'lib/billboard'
```

Then we need to run the update in order to pull down the files.

```
[user@office SampleTheme]$ git submodule update
Initialized empty Git repository in ~/git_dev/SampleTheme/lib/billboard/.git/
remote: Counting objects: 26, done.
remote: Compressing objects: 100% (22/22), done.
remote: Total 26 (delta 5), reused 0 (delta 0)
Receiving objects: 100% (26/26), 17.37 KiB, done.
Resolving deltas: 100% (5/5), done.
Submodule path 'lib/billboard': checked out '1c407cb2315z0847facb57d79d680f88ca004332'
```

Looking in the lib/billboard directory now shows a nice listing of the needed files.

Removing Submodules

What happens if we need to remove a submodule? Maybe I made a mistake. It could also be that the design of the project has changed, and the submodules need to change with it. No problem, I'll simply run `git submodule rm [submodule path]` and everything will be great, right?

```
[user@office SampleTheme]$ git submodule rm lib/billboard
error: pathspec 'rm' did not match any file(s) known to git.
Did you forget to 'git add'?
 b8ff8f68eb56938b9b4bf993619218fa848c5848 lib/billboard (1.2.25)
```

Unfortunately, this is wrong. Git does not have a built in way to remove submodules. Hopefully this will be resolved in the future, because we now have to do submodule removal manually.

Sticking with the example, we'll remove the lib/billboard module from SampleTheme. All the instructions will be run from the working directory of the SampleTheme repository. In order, we need to do the following:

1. Remove the submodule's entry in the .gitmodules file. Since lib/billboard is the only submodule in the SampleTheme repository, we can simply remove the file entirely by running `git rm .gitmodules`. If lib/billboard isn't the only submodule, the .gitmodules file will have to be modified by hand. Open it up in vi, or your favorite text editor, and remove the three lines relevant to the submodule being removed. In this case, these lines will be removed:

   ```
   [submodule "lib/billboard"]
   path = lib/billboard
   url = git@mygithost:billboard
   ```

2. Remove the submodule's entry in the .git/config. This step isn't strictly necessary, but it does keep your config file tidy and will help prevent problems in the future. The submodule's entry in .git/config will only be present if you've run `git submodule init` on the repository. If you haven't, you can skip this step. In this example, the following lines will be removed:

   ```
   [submodule "billboard"]
   url = git@mygithost:billboard
   ```

3. Remove the path created for the submodule. This one is easy. Simply run `git rm --cached [plugin path]`. In this example, I will run `git rm --cached lib/billboard`. As I've seen noted elsewhere, do not put a trailing slash as the command will fail. For example, if I run `git rm --cached lib/billboard/`, I get an error: `fatal: pathspec 'lib/billboard/' did not match any files`.

   ```
   [user@office SampleTheme]$ git rm --cached lib/billboard
   rm 'lib/billboard'
   ```

Updating Submodules

There is an aspect about submodules that some may not realize when first working with Git submodules. When you add the submodule, the most recent commit of the submodule is stored in the main repository's index. That means that as the code in the submodule's repository updates, the same code will still be pulled on the repositories relying on the submodule.

This makes a lot of sense when you consider how your code will have been tested and verified (or at least should be) against a specific version of the submodule code, but the main repository's code may not work well with new submodule updates before the changes are tested.

Unfortunately, like removing submodules, Git does not make it clear how to update a submodule to a later commit. Fortunately though, it's not that tough.

1. Initialize the repository's submodules by running `git submodule init` followed by `git submodule update`.

```
[user@office SampleTheme]$ git submodule init
Submodule 'lib/billboard' (git@mygithost:billboard) registered for path 'lib/billboard'
[user@office SampleTheme]$ git submodule update
Initialized empty Git repository in ~/git_dev/SampleTheme/lib/billboard/.git/
remote: Counting objects: 26, done.
remote: Compressing objects: 100% (22/22), done.
remote: Total 26 (delta 5), reused 0 (delta 0)
Receiving objects: 100% (26/26), 17.37 KiB, done.
Resolving deltas: 100% (5/5), done.
Submodule path 'lib/billboard': checked out '1c407cb2315z0847facb57d79d680f88ca004332'
```

2. Change into the submodule's directory. In this example, `cd lib/billboard`.

```
[user@office SampleTheme]$ cd lib/billboard
[user@office SampleTheme/lib/billboard]$
```

3. The submodule repositories added by `git submodule update` are "headless". This means that they aren't on a current branch. To fix this, we simply need to switch to a branch. In this example, that would be the master branch. We switch with the following command: `git checkout master`.

```
[user@office SampleTheme/lib/billboard]$ git status
# Not currently on any branch.
nothing to commit (working directory clean)
[user@office SampleTheme/lib/billboard]$ git checkout master
Previous HEAD position was b8ff8f6... re-ordering
Switched to branch 'master'
Your branch is behind 'origin/master' by 8 commits, and can be fast-forwarded.
[user@office SampleTheme/lib/billboard]$ git status
# On branch master
# Your branch is behind 'origin/master' by 8 commits, and can be fast-forwarded.
#
nothing to commit (working directory clean)
```

4. Next, we simply need to update the repository to ensure that we have the latest updates. We can do that with a pull: `git pull`.

```
[user@office SampleTheme/lib/billboard]$ git pull
remote: Counting objects: 31, done.
remote: Compressing objects: 100% (24/24), done.
remote: Total 24 (delta 15), reused 0 (delta 0)
Unpacking objects: 100% (24/24), done.
From mygithost:billboard
   b8ff8f6..5cab93f  master     -> origin/master
 * [new tag]         1.2.28     -> 1.2.28
From mygithost:billboard
 * [new tag]         1.2.26     -> 1.2.26
 * [new tag]         1.2.27     -> 1.2.27
Updating c547e0d..5cab93f
Fast-forward
 billboard.php                            |  109 ++++++++++++++-
 classes/view_gettingstarted.php          |  107 ++++++++++++++
 classes/view_gettingstarted_content.php  |   38 +++++
 css/admin.css                            |   26 ++++
 history.txt                              |   22 +++-
 js/admin.js                              |   17 +++
 lib/updater/get.php                      |   94 +++++++-----
 lib/updater/history.txt                  |    9 ++
 lib/updater/updater.php                  |  241 ++++++++++++++++++-------------------
 9 files changed, 519 insertions(+), 144 deletions(-)
 create mode 100644 classes/view_gettingstarted.php
 create mode 100644 classes/view_gettingstarted_content.php
 create mode 100644 css/admin.css
 create mode 100644 js/admin.js
 create mode 100644 lib/updater/history.txt
```

5. Now switch back to the root working directory of the repository. In our example, we are two directories in, so we run `cd ../..`.

```
[user@office SampleTheme/lib/billboard]$ cd ../..
[user@office SampleTheme]$
```

6. Everything is now ready to be commit and pushed back in (if there is a remote repository to push to that is). If you run `git status`, you'll notice that the path to the submodule is listed as modified. This is what you should expect to see. Simply add the path to be commit and do a commit. When you do the commit, the index will update the commit string for the submodule.

```
[user@office SampleTheme]$ git status
# On branch master
# Changed but not updated:
#   (use "git add ..." to update what will be committed)
#   (use "git checkout -- ..." to discard changes in working directory)
#
#       modified:   lib/billboard (new commits)
#
no changes added to commit (use "git add" and/or "git commit -a")
[user@office SampleTheme]$ git add lib/billboard
[user@office SampleTheme]$ git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD ..." to unstage)
#
#       modified:   lib/billboard
#
```

Closing Thoughts

I've learned a lot in my short time with Git. Expect to see more details about working with Git in the future. I have a series of pitfalls I've discovered that I should organize together and post about. I've also created some really slick scripts to help me automate numerous things when working with the repositories that I'd like to share.

If there is anything specific you'd like to know about Git, please add a comment or contact me.

Did I help you?

- Send me a tip via Paypal.
- Help with this site's hosting with my Linode referral code.

Categories : Development, Tips 'n Tricks

Comments

1.  **David James** says:
   June 10, 2009 at 5:22 pm

   Thank you very much for posting this. Especially the part about "Updating Submodules" which was not obvious from reading the documentation and lots of other information.

   In my opinion, the way that git handles submodules is not obvious, because I feel like it could have been done using several different API styles. Now that I've done it, it makes sense. It might even be one of the better ways of doing it. But it sure was confusing the first time around.

   Reply

   -  **gaarai** says:
     June 11, 2009 at 8:14 am

     You're welcome David. When I waded through the documentation available, it quickly became clear that there was not a one-stop-shop for handling submodules. I hope that I've done a good enough job. 🙂