

[Chris Jean](#)

[Linux, WordPress, programming, anime, and other stuff](#)

- [Home](#)
- [Linux](#)
- [Development](#)
- [Random Ramblings](#)

# Git Submodules: Adding, Using, Removing, Updating

by [Chris Jean](#)

April 20th, 2009

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 lib/billboard**". 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](#).

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](#)

- o  [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. 😊

[Reply](#)

2.  [Paddy](#) says:

[July 26, 2009 at 9:34 am](#)

Yup, it was very useful indeed for a newbie with Git Sub modules .....This is rocking tutorial.

Thank you for sharing!

[Reply](#)

- o  [gaarai](#) says:

[July 27, 2009 at 10:22 pm](#)

Glad to share Paddy.

[Reply](#)

3. [Thoughts on Clojure Package Management - Digital Digressions by Stuart Sierra](#) says:

[August 31, 2009 at 11:14 am](#)

[...] I concluded that it's just too early. Clojure is a scarcely two years old. It just released "1.0" this year, and is still developing rapidly. The libraries are evolving equally rapidly. If you want to build a project using, say, Compojure, the best way to do it is with Git submodules. [...]

[Reply](#)

4.  [Joseph Sofaer](#) says:

[October 5, 2009 at 5:43 pm](#)

That was really helpful.


[Reply](#)

5.  [Spencer Hill](#) says:

[May 19, 2010 at 2:27 pm](#)

Yeah man. SUPER clear. Had this done in like a second.

[Reply](#)

6.  [Jose Luis de la Fuente](#) says:

[August 10, 2010 at 11:46 pm](#)

Thanks a lot man!! Specially for the Removing submodules section. Has been very useful for me 😊

[Reply](#)

7. [Team Learnings for August 17th « Rubinauts Blog](#) says:

[August 19, 2010 at 12:34 pm](#)

[...] a git submodule is a little trickier than adding one, found some detailed instructions here. It boils down to the [...]

[Reply](#)

8.  [Wills](#) says:

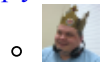
[August 24, 2010 at 8:01 am](#)

Thanks lot,

I have a question.

If I create a tag/branch in the superproject (main), that tag/branch name automatically pushed to the submodules? how ?

[Reply](#)



o [gaarai](#) says:

[August 30, 2010 at 4:08 pm](#)

Submodules know nothing about any possible “superprojects”. Repos with submodules simply store where the submodule repo is from and which commit of that repo is referenced. When you init and update a repo’s submodules, it simply pulls down the commit version specified. This is why you have to manually update the submodules to newer versions.

Basically, a submodule is nothing more than a bookmark that says to pull down a specific commit of another repo at a specific location and to put it in a specific directory.

[Reply](#)



9. [Alex](#) says:

[August 24, 2010 at 1:43 pm](#)

Thanks a lot.

Especially for “Updating Submodules” section.

[Reply](#)

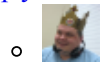


10. [Madhu](#) says:

[September 6, 2010 at 6:17 am](#)

Is there a way to update the submodules also at the time of cloning the whole project or we have to do it one by one updating the submodules?

[Reply](#)



o [gaarai](#) says:

[September 21, 2010 at 8:28 am](#)

I do neither. I wrote a simple script for me to quickly run after cloning a repo that takes care of all of this for me. You can find the script [here](#). Note: It is written in Perl.

The main reason that I wrote this is that I have submodules that also have submodules. Remembering which submodules I had to switch to to do subsequent inits and updates on was a pain, so I wrote this to handle it all for me. It works well for non-nested submodules as well.

[Reply](#)



11. [Juan Delgado](#) says:

[September 21, 2010 at 7:44 am](#)

Hi there,

Nice tutorial. One thing to add though! We’ve found out that after setting up the module to its master branch (so it’s not headless and we can run easy pulls), updating the submodule was making it headless again, which is a pain.


We solved this by doing this:

```
git config submodule.lib/billboard.update merge
```

Which as per documentation is one of the ways of preventing this from happening (the other one would be using rebase).

Hope this helps!

[Reply](#)

- o  [gaarai](#) says:  
[September 21, 2010 at 8:18 am](#)

Thanks for the info Juan.

“git submodule update” clones the submodule repo and then checks out the commit of that repo represented internally in the parent repo. Whenever you checkout a commit, the repo becomes headless. So a submodule repo being headless after an update is not a surprise to me.


In the system I built to manage packaging of theme and plugin zips from the repos, I have some automated code that manages getting the appropriate submodule commits before packaging the files. Whenever I push up a new tag for a repo, all repos that use it as a submodule will automatically be updated to use the new commit, these individual repos will have their history files and version numbers updated, and then the repo updates will be committed and pushed along with a new tag (possibly setting off another run of updates).

So, using the submodule init/update system as-is on all development platforms ensures that the repos that we pull down will match what is available to our customers/users (short of any untagged changes in the parent). It also prevents anyone from accidentally committing untagged commits for the submodules to parent repos, which I can see happening easily in your setup.

We have to remember to checkout the master branch whenever we do development on the submodules, but as this is a standard practice at all times, there aren't any problems when a developer uses a different system without special config setups that remove the need to checkout the master branch to do updates.

In short, to each their own git config setup. 😊


[Reply](#)

-  [Wouter Samaey](#) says:  
[July 19, 2011 at 10:00 am](#)

This post seems very interesting, but unfortunately you lost me.

Are you trying to say that you should always use tags, so you can keep working headless ?


[Reply](#)

-  [Chris Jean](#) says:  
[July 19, 2011 at 10:07 am](#)

We don't work headless; we simply have to checkout the master branch each time we need to work on one of the submodules.

Our solution is likely to be specific to our own system. If this approach doesn't work or doesn't make sense for your workflow, then I recommend that you find one that does.

[Reply](#)

12.  [naja](#) says:  
[November 5, 2010 at 7:40 pm](#)

thanks!

[Reply](#)

13. [How to update submodules in git « The Missing Readme](#) says:



[November 6, 2010 at 12:49 am](#)

[...] for anyone who wants to learn git), but using submodules is a bit confusing. I've found that Chris Jean's Blog and the chapter on submodules in Pro Git both explain different parts of it well (I recommend [...])

[Reply](#)



14. [Ruud](#) says:

[November 24, 2010 at 5:48 am](#)

Thank you! Very clear...

[Reply](#)



15. [Lillem4n](#) says:

[November 26, 2010 at 6:48 pm](#)

Great tutorial! You succeeded where all other google hits failed; now I can work with submodules! \o/ tyvm

[Reply](#)

16. [DANIEL.DRZIMOTTA.COM > Git Submodules: Adding, Using, Removing, Updating :: Chris Jean](#) says:

[November 30, 2010 at 12:35 pm](#)

[...] Git Submodules: Adding, Using, Removing, Updating :: Chris Jean. This was written by Daniel Drzimotta. Posted on Tuesday, November 30, 2010, at 11:24 am. Filed under Uncategorized. Tagged git, submodules. Bookmark the permalink. Follow comments here with the RSS feed. Post a comment or leave a trackback. [...]

[Reply](#)



17. [Peter](#) says:

[December 23, 2010 at 6:12 am](#)

Hi,

I have a GIT repository and inside several SUBMODULES.

The problem is everytime someone updates submodule, I try to update my files by calling:

GIT SUBMODULE UPDATE

But what happens is it deletes all my uncommitted files which is very annoying. I tried to commit and push the files first but it doesn't let me push:

error: failed to push some refs to 'ssh://dev.ancreative.co.uk/var/git/library/Blocks.git' To prevent you from losing history, non-fast-forward updates were rejected Merge the remote changes before pushing again. See the 'non-fast-forward' section of 'git push --help' for details.

So I try to call git submodule update but then this deletion happens.

Anyone has any idea what can cause this problem? Thanks a lot.

[Reply](#)



o [Chris Jean](#) says:

[December 23, 2010 at 7:34 am](#)

Your workflow is wrong here. `git submodule update` will always remove any changes you have. So, if you have changes in a submodule, you don't want to run `git submodule update`.

When you have submodule code that you are modifying and others may be modifying, you have to start working with that specific repo as a regular repo (which it still is) rather than as a submodule.

What you want to do is create a new branch for your local development (for this discussion, let's say that you call your dev branch "local-dev"). When commits are pushed from another source, commit your changes to the local-dev branch, switch to the master branch (`git checkout master`), pull down the changes (`git pull`), and switch back to your local-dev branch (`git checkout local-dev`). The final step to do depends on the way you want your repo's history to flow and how complex the commits you want to bring into your local-dev branch are. For simple changes, rebase against master (`git rebase master`). For complex changes (renamed or moved files, very large changes to files modified in both branches, etc), it is generally recommended that you use merge (`git merge master`).

If you are unfamiliar with all of this, I recommend that you read the following links:

[Git Book – Basic Branching and Merging](#)

[Git Book – Rebasing](#)

[Reply](#)



18. [Peter](#) says:

[December 23, 2010 at 9:31 am](#)

Thanks for super quick reply!

At the end I figured it out, after every submodule update they got switched to head so I added "git config submodule.[my.submodule.location].update.merge" after this all seems to be working fine. I'll have to check what "rebase master" means. Good job!

[Reply](#)

19. [Chris Jean – Git Submodules: Adding, Using, Removing, Updating – Yostivanich](#) says:

[January 22, 2011 at 8:09 pm](#)

[...] things to figure out: removing and updating submodules from your repository.via [Chris Jean – Git Submodules: Adding, Using, Removing, Updating](#). Git submodules is an incredibly cool feature, it's a great solution to including [...]

[Reply](#)



20. [Will](#) says:

[May 24, 2011 at 3:01 pm](#)

Just as a very slight simplification, the init/update pair of commands can be streamlined to

`git submodule update --init`

This will automatically initialize nay submodules if necessary, and then update them. You can simplify it further by replacing the `--init` with `-i`

[Reply](#)



21. [Jason](#) says:

[July 25, 2011 at 1:10 pm](#)

Nice tutorial however git submodule add also clones the submodule and check out the master branch. See:

<https://git.wiki.kernel.org/index.php/GitSubmoduleTutorial>

Jason

[Reply](#)

o




[Chris Jean](#) says:

[July 25, 2011 at 1:15 pm](#)

Yes it does, but this blog post is meant to help people get up and running with using Git submodules and is not meant as a comprehensive run-down of everything Git does internally that may or may not matter to the person using Git. Unfortunately, most of the “tutorials” I’ve read get mired in details and forget the point of actually doing something.

So, I’m not sure I understand your “however”. It would be nice if people would not only point out a short coming but why it is important given the context, which I assume that you think that it is.

[Reply](#)

-  *Jason* says:  
[July 25, 2011 at 2:23 pm](#)

Hi Chris,

I don’t really have an opinion as to whether it’s a shortcoming as I’m really can’t think of any alternative approach to how submodules would be implemented (IMO the current behavior works as one would expect). Rather, I’m merely referring to this sentence in the blog post:

“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.”


It is an empty folder only if you are cloning another repository which uses submodules, meaning you’ll need to run git submodule update in order to pull down the submodule contents. However if you add a submodule to the repository in the manner you state above, the contents of that repository will in fact be there.

So all I am saying is that newbies might be looking for some clarification:

- 1) If you add the submodule to a project you are working on, then that submodule’s contents will be cloned and checked out.
- 2) If you clone somebody else’s repository which uses submodules, then yes you will indeed need to run submodule init/update in order to retrieve the submodule contents. It is this behavior which newbies find confusing because (like me) the first time you clone a project using submodules, you start wondering why in the heck all of the “parts” aren’t there. 😊

Jason


[Reply](#)

-  *Chris Jean* says:  
[July 25, 2011 at 2:48 pm](#)

Gotcha. Things have changed since I posted this. I added a note.

Thanks for pointing it out.

[Reply](#)

-  *Jason* says:  
[July 26, 2011 at 9:20 am](#)

Cool. Unfortunately there remains no way to easily remove submodules. I wonder what the reasoning is for not offering a git submodule rm command? There must be something I’m missing regarding why this feature isn’t offered.

22.  *Diogo Melo* says:

[August 15, 2011 at 6:39 pm](#)

Excellent post, man. Thanks 😊

[Reply](#)

23.  *Nic* says:

[August 16, 2011 at 3:03 am](#)

First thanks for this great tutorial. ...like others said, this was the clearest description of the use of submodules I could find.

I'm trying to figure out whether I can use submodules AND make modifications to them – so far I haven't had any luck and maybe another approach is required for what I'm trying to do.

Basically I'd like to pull some libraries as submodules within my main repository and using your approach made this easy. However, I'd now like to make modifications to those libraries. I can't push those changes to the remote repos, since I have no access. ...I can only create local branches.

The changes I make are recognised by the main repository and I can commit and push them to my main remote repo. But if I now want to pull them onto the server, git pull will not recognise any changes and won't update.

Is what I'm trying to do even possible with submodules? Or are they purely for being updated via their respective remote repos and I can't really do any modifications to the code myself?

[Reply](#)



o *Chris Jean* says:

[August 19, 2011 at 10:40 am](#)

While I haven't done this myself, you might look at making a remote repository for yourself that is a clone of the repo that you want to modify. You can then add your remote repository as the source for your submodule.

You can then use whatever workflow you want when working with that remote repository. You can do pulls against the original source, merge as desired, and do your updates as desired. After committing these changes, you can then pull down the changes to your repository/repositories that use this repo as a submodule.

[Reply](#)

24.  *richtaur* says:

[September 28, 2011 at 5:10 pm](#)

Thanks for writing this, very helpful! Here's a tip someone taught me recently. You can combine commands:

```
git submodule update --init
```

Cheers!

[Reply](#)

25.  *phluks* says:

[October 19, 2011 at 4:36 am](#)

Yes! Nice.

I needed to get rid of a submodule 😊

[Reply](#)

26.  *Igor Santos* says:

[December 7, 2011 at 1:58 pm](#)

Hello!

I have two projects that should share a part of the code. So, I created a submodule. Both projects have the submodule with the master branch. Let's name them:

app > project 1

app-cli > project 2

models > the submodule (inside app/models and app-cli/models)

I've made some changes inside app/models, and inside that folder, I committed and pushed the changes.

After that I entered app-cli and tried to run "git submodules update", but it did nothing. I needed to enter app-cli/models and do a normal "git pull".

Is this expected? I did something wrong?

[Reply](#)



◦ [Chris Jean](#) says:

[December 7, 2011 at 2:11 pm](#)

This is as expected. From the man page for the git submodule update command:

Update the registered submodules, i.e. clone missing submodules and checkout the commit specified in the index of the containing repository. This will make the submodules HEAD be detached unless --rebase or --merge is specified or the key submodule.\$name.update is set to rebase or merge.

Remember when you did the "git submodule add" for the app and app-cli projects how it would have shown that ".gitmodules" was either added or modified and that the path to the submodule was a new file? This process stores the HEAD commit name of the submodule in the parent repository. When you updated the app/modules files, committed, and pushed the changes, doing a "git status" in the app directory should have shown that your module's path was modified. This is because it now has a new commit. In order to store that new commit name to the parent repository, you need to add the change and commit the change.

Now think about the quoted description above. When you do "git submodule update", a "git fetch" command will be run for the modules submodule and the commit stored in the parent repository will be checked out. As indicated in the documentation, this can result in a detached HEAD.

So yes, needing to do "git pull" (or "git merge" or "git rebase") to get the latest changes into your app-cli/modules directory is to be expected, and no, you did not do something wrong.

The biggest difficulty in understanding what the hell is going on with the "git submodule" commands is to understand what a submodule is and what a submodule is not. Writing this post was a way for me to improve my understanding. Essentially, a submodule is nothing more than a pointer to a specific commit of a repository that is connected with a path inside of another repository. A submodule really has nothing to do with files and nothing is automated. If you want a submodule's repository to update to the latest commit, you have to manually tell Git that you want that.

[Reply](#)



▪ [Igor Santos](#) says:

[May 27, 2012 at 12:44 pm](#)

Nice.

I was thinking that "submodule update" would discover the last commit for the submodules and then fetch the differences.

[Reply](#)

27.  [dragonx](#) says:

[December 8, 2011 at 9:32 pm](#)

Chris, I'm fairly new to git, I've been working with it a bit for a year, but never really getting past the basic checkin of changes, and occasional reverts. This post was very useful.

I just tried adding an open source python package as a submodule to my project. The tree structure of the submodule didn't work well as a subfolder, given the way it was built so that it could be installed via setup.py.

I ended up having it as a separate git repo, and I put a symlink in my main project.

Do you have any comments on why a submodule might be better than just a symlink to another repo?

[Reply](#)



◦ [Chris Jean](#) says:

[December 9, 2011 at 12:44 am](#)

A submodule has meaning outside of your local file system. A symlink does not.

[Reply](#)



28. [Ryan](#) says:

[December 10, 2011 at 1:40 am](#)

WordPress themes? Profession? Oh dear....

[Reply](#)



◦ [Chris Jean](#) says:

[December 10, 2011 at 7:55 am](#)

Perhaps "vocation" would have been a better word to use, but profession applies.

[Reply](#)



29. [Thiyagarajan Veluchamy](#) says:

[February 3, 2012 at 2:27 am](#)

Thanks, its helped me to understand the submodule concents.

[Reply](#)

30. [Como administrar um Git sob um Git | Rafeal's Blog](#) says:

[March 12, 2012 at 2:32 pm](#)

[...] <http://chrisjean.com/2009/04/20/git-submodules-adding-using-removing-and-updating/> - Link em que o Autor mostra com exemplos as possíveis funcionalidades; [...]

[Reply](#)

31. [Phonegap project structure using Git submodules | Truong-An \(Bi\) Thai](#) says:

[March 16, 2012 at 11:41 am](#)

[...] If You want a more general guide on how to create and work with Submodules, check out this: <http://chrisjean.com/2009/04/20/git-submodules-adding-using-removing-and-updating/> [...]

[Reply](#)

32. [git submodules](#) says:

[May 8, 2012 at 10:32 am](#)

[...] this post very helpful on git [...]

[Reply](#)33. *Pieter Jacobs* says:[June 6, 2012 at 8:13 am](#)

Thanks, this also really helped me to understand submodules etc.

[Reply](#)34. *Márcio Almada* says:[June 20, 2012 at 9:04 am](#)

This should go straight into Git official documentation. Thanks man!

[Reply](#)35. *張旭* says:[July 8, 2012 at 7:30 am](#)

you let me how to delete a submodule!  
thank you so much!

你讓我知道如何刪除一個 git submodule!  
感謝你!

[Reply](#)36. *Tom Byrne » Generating Docs for Github Wiki from Haxe code* says:[July 9, 2012 at 10:12 pm](#)

[...] streamline the documentation process, I added the wiki repository as a submodule to the main repository, this means that the wiki files would always sit in the same relative [...]

[Reply](#)37. *Millisami* says:[August 1, 2012 at 4:15 am](#)

Great!

Thanks for clearing the sub-module of which I used to be afraid of earlier.

[Reply](#)38. *David Alsbury* says:[August 8, 2012 at 12:00 pm](#)

Very helpful. You probably saved me a several hours trying to make sense of submodules.

[Reply](#)◦ *Chris Jean* says:[August 13, 2012 at 9:27 am](#)

Glad to help.

[Reply](#)39. *Trevor Green* says:[August 16, 2012 at 4:47 pm](#)

I'm trying to use wordpress as a submodule.

So I added it, that worked.

Then I did "git checkout 3.4.1 in the directory. That worked.

But rather than a clean checkout I'm getting the following in versions.php (and maybe other files, that is just the first error).

```
<<<<<<>>>>> 95e5cafc6323c23766ba6da81d9111137a67a93f
```


Which appears to be showing a conflict between the master and the tagged version in the submodule.

That is confusing as the submodule should just be a pointed to a specific version right?

Diff shows nothing.

So I'm not sure what the problem is with my workflow but I'm getting garbage. Any thoughts?

[Reply](#)

-  *Trevor Green* says:  
[August 16, 2012 at 4:48 pm](#)

That didn't post right. I took out the greater than less thans..

HEAD

\$wp\_version = '3.4.1';

=====

\$wp\_version = '3.5-alpha-21535';

95e5cafc6323c23766ba6da81d9111137a67a93f

[Reply](#)

-  *Chris Jean* says:  
[August 21, 2012 at 9:06 am](#)

I've never had conflicts when doing a checkout from a clean working directory, so I really don't know what could cause that. Are you sure that the working directory was clean and that you didn't have any lingering files in the "changes to be committed" staging area?

[Reply](#)

40.  *Glitchdata* says:  
[August 23, 2012 at 1:17 am](#)

Thanks for this. It's good.

Linked your article here instead of writing our own. Cheers.

[http://wiki.glitchdata.com/index.php?title=Git:\\_Sub-Modules](http://wiki.glitchdata.com/index.php?title=Git:_Sub-Modules)


[Reply](#)

41. *git control | MY KNOWLEDGE LIBRARY* says:  
[October 10, 2012 at 5:00 am](#)

[...] <http://chrisjean.com/2009/04/20/git-submodules-adding-using-removing-and-updating/> [...]


[Reply](#)



42.  [James O'Brien](#) says:  
[December 13, 2012 at 7:44 pm](#)

Thanks. Helpful

[Reply](#)

43.  [Aru](#) says:  
[January 14, 2013 at 5:16 am](#)

Git submodule inside of a submodule- Not listing the submodules while doing --recursive cloning.

I have a repo A. Inside that i added a submodule repo B. Inside repo B i have repo C also.

When i am trying to clone repo A (git clone --recursive git@IPxxxxx:repo A) It is listing repo B.

But (i did cd B:) Inside repo B nothing is showing. In the actual situation it should display repo B and inside that repo c also should display.

Actually recursive submodules are not working here.


from my parent repo (ie repo A) I am following these steps.

```
$ git submodule add git@ip:B.git B
$ git submodule status
$ git status
$ git commit -m 'Added submodule B'
$ git push
$ git submodule init
$ git submodule update
```

After that for adding Repo C as submodule i executed the same steps from this location. "xx@xx-desktop:~/pjt1/A/B\$"


Pls correct me if any mistakes in the steps.

[Reply](#)

-  [Chris Jean](#) says:  
[January 16, 2013 at 1:23 pm](#)

I'm not quite sure what problem you are running into. To test your situation, I created four repos (A, B, C, D). I made D a submodule of C, C a submodule of B, and B a submodule of A. I then cloned A using `git clone --recursive A A-test`, and it properly populated all the submodules. So, if this did not work for you, I would have to assume it was something specific to the way you had your modules and submodules set up.


[Reply](#)

44.  [bernard](#) says:  
[January 18, 2013 at 4:52 am](#)

A way to get around all the problems with submodules (missing repos -> missing code -> need to fork all submoduled repos, renaming is a pain etc.) is using the subtree merge strategy (<http://www.kernel.org/pub/software/scm/git/docs/howto/using-merge-subtree.html>)

Much more "git-like", you never lose the "remote" code (because there is none), and merging back and forth is easy.

[Reply](#)

-  [Chris Jean](#) says:  
[January 18, 2013 at 9:35 am](#)

Fortunately, submodules work very well for me. There are a few rough edges that I think could be smoothed out (such as providing a command-driven approach to removing submodules), but overall, they do exactly what I need. Knowing this other approach is good to have in the back of my head though. Thanks for sharing it.

[Reply](#)

45. [Git Submodules - Bronson Quick](#) says:  
[January 22, 2013 at 5:35 am](#)

[...] <http://chrisjean.com/2009/04/20/git-submodules-adding-using-removing-and-updating/> [...]

[Reply](#)

## Leave a Reply

Name (required)

Mail (will not be published) (required)

Website

☐ Notify me of followup comments via e-mail

I believe that the free flow of information and ideas is key to the past and future development of mankind. Unless the content declares otherwise, the post content on this site is declared public domain ([CC0 1.0 Universal](#)) and can be used in any manner with or without attribution or permission. Of course, if you wish to give attribution back to me, that would be very nice. :)

This site is running [WordPress](#) with the [iThemes Builder](#) theme by [iThemes](#).