Git Submodules

Christian Külker

2024-06-17

Contents

Since storage is so cheap, copying around is a habit. With **git sub-modules** it can be avoided.

There are basically two motivations for using git submodules:

1. Using an other project inside the working tree and maintain both history independently

2. Splicing (linking) many subprojects together, to form a large project. This can overcome limitations of repository size, git access and git transfer size.

The following document assumes 1.

1 Using Git Submodules

Example: https://github.com/chenkie/aurelia-jwt-auth

This includes the 'server' subdirectory which points to

https://github.com/auth0-blog/nodejs-jwt-authentication-sample/tree/63b35acb9b2576e107ae6339c27a2dcf2

To get this, clone the original repository and update the git sub-modules:

```
mkdir test
cd test
git clone https://github.com/chenkie/aurelia-jwt-auth.git
cd server
git submodule update --init
```

If you just want to update one submdoule

```
git submodule update --init pandoc
Submodule 'pandoc' (https://github.com/ckuelker/pankyll-pandoc.git)
 ↪  registered\
 for path 'pandoc'
Cloning into '/tmp/PTRE1/pankyll-theme-rankle-example/pandoc'...
Submodule path 'pandoc': checked out
 ↪  'e04b90aee4817eb8075c5534f528d6235a1ddde3'
```

2 Adding A Git Submodule

From the **root of the project**:

```
git submodule add <remote_url> <destination_folder>
git commit -m "Added the submodule to the project."  <destination_folder>
git commit -m "Added the submodule to the project." .gitmodules
git push
```

Example:

This example adds and commits a git sub-module from `github.com` to a repository as a read only module (`ro`). While `github.com` provides read/write (rw) access to ones own

git repositories it is easier to administer each repository by itself. Therefore it is advisable to add the read only (ro) repository.

- read/write (rw): git@github.com:ckuelker/pankyll-theme-newspaper.git
- read only (ro): https://github.com/ckuelker/pankyll-theme-newspaper.git

```
git submodule add https://github.com:ckuelker/pankyll-theme-newspaper.git\
 themes/pankyll-theme-newspaper
Cloning into '/srv/g/g.c8i.org/https-christian.kuelker.info/themes/\
pankyll-theme-newspaper'...
remote: Enumerating objects: 279, done.
remote: Counting objects: 100% (279/279), done.
remote: Compressing objects: 100% (166/166), done.
remote: Total 279 (delta 146), reused 222 (delta 91), pack-reused 0
Receiving objects: 100% (279/279), 46.23 KiB | 0 bytes/s, done.
Resolving deltas: 100% (146/146), done.
git commit themes/pankyll-theme-newspaper
git commit .gitmodules
```

3 How does this work?

The first `git submodule add` command creates the file `.gitmodules` and an entry in `.git/config`. Every other `git submodule add` will add entry to `.gitmodules` and `.git/config`.

The file `.gitmodules` is located at the same level as `.git`. This file contains the git sub-module information, in this case the sub-module is located in the directory `server`.

```
[submodule "server"]
        path = server
        url = https://github.com/auth0/nodejs-jwt-authentication-sample
```

The directory is empty.

After the execution of:

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

The directory is populated. One file `.git` marks the module in its content.

```
1  gitdir: ../.git/modules/server
```

The . `/git/config` file on the main repository gets updated:

```
[submodule "server"]
        url = https://github.com/auth0/nodejs-jwt-authentication-sample
```

In case the latter step failed this can be done for all sub-modules by `init` or for one sub module via `init PATH`. Example for one module called `server`:

```
git submodule init server
```

4 Showing the remote repository commit

```
git submodule status
    -63b35acb9b2576e107ae6339c27a2dcf2f3aed04 server
```

If more than one submodule is present all will be shown.

```
git submodule status
 b2f25b9952f139d3884f214ef2e6965fd742af54 content (v0.2.0-1-gb2f25b9)
 e04b90aee4817eb8075c5534f528d6235a1ddde3 pandoc (v0.1.1)
 d9ede740610048adf2a51cb1683fc3ea9048e892 themes/pankyll-theme-rankle
  ↪  (v0.1.6)
```

If only one is of interest use the directory.

```
git submodule status pandoc
 e04b90aee4817eb8075c5534f528d6235a1ddde3 pandoc (v0.1.1)
```

See also list sub-modules below.

5 Update A Sub-Module

Sub-modules are occasionally updated. We assume that the sub-module directory is on the to level of the repository. If this is not the case, replace `DIRECTORY` with `PATH` or enter the sub-modules root directory.

For a sub-module with changes included, a `checkout` will do.

```
git -C DIRECTORY checkout VERSION
git commit -m 'Updated sub-module DIRECTORY to VERSION' DIRECTORY
git submodule status
```

If the repository contains more than one submodule the submodule can be specified, so that just that submodule is addressed.

```
git submodule status DIRECTORY
e04b90aee4817eb8075c5534f528d6235a1ddde3 DIRECTORY
```

For a sub-module with remote changes, this remote changes need to fetched first and the sub-module will have the indicator `(new commits)`:

```
git submodule update --remote
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:   DIRECTORY (new commits)
git commit -m 'Updated sub-module DIRECTORY to VERSION' DIRECTORY
git submodule status
```

If you want to work on just one specify the DIRECTORY.

```
git submodule update --remote pandoc
Submodule path 'pandoc': checked out
 ↪  '7e778b2c185404fcae3ec04611e2dfc7cfd97efc
```

If you have more than one submodule, lets say a, b and c that is a also a directory, the following loop may help. However, this also might fail with a detached HEAD.

```
export pfx=`pwd`;for i in a b c;do cd $pfx/$i;\
git checkout `git rev-parse --abbrev-ref HEAD`;\
git pull;done
```

6 Commit A Changed Sub-module

A git sub-module is basically a directory with a git working tree included into the main repository. On top of that the git sub-module has its status. The status is the last commit of the sub-module that is known to the main repository and that is usually the commit that is made known by a commit to the main repository after the `git submodule add` command.

So after the sub-module changed, for example because the remote location of the

sub-module had a newer commit that was fetched and updated the directory of the sub-module a `git status` command will show that the sub-module was updated. If this new status of the sub-module is desired to stay around it can be made know to the main repository by making a `git commit` and eventually pushing it to the remote origin of the repository.

We assume the directory `content` is a git sub-module.

```
cd content
git commit -m 'update git submodule to v0.1.5-2-gf5648b5'
```

Or shorter:

```
git commit -m 'update git submodule to v0.1.5-2-gf5648b5' content
```

If we want we can distribute this:

```
git push
```

7 Activate A Sub-module

Usually all sub-modules are active. In case some sub-modules are big or have other reasons not always to be updated this selection of activation or deactivation might be useful:

A git sub-module is active when:

1. `submodule.NAME.active = true` (in `.git/config` : active = true)
2. sub-module path matches the specification of the path in `submodule.active`
3. `submodule.NAME.url` is set

A git sub-module is not active when:

1. `submodule.NAME.active = false` (in `.git/config` : active = false)

8 Remove A Sub-module

Depending on your git version, this can be simple or labour.

Modern git

```
git rm PATH_TO_SUB-MODULE
commit -m 'Removed git sub-module SUB-MODULE' PATH_TO_SUB-MODULE
```

This removes the entry in `.gitmodules` and `.git/config`. The info of the old SUB-MODULE is kept in `.git/modules` to be able to checkout older versions. If you for whatever reason want to remove all traces (some will remain in the git history though) do:

```
rm -rf git/modules/PATH_TO_SUB-MODULE
git config --remove-section submodule.PATH_TO_SUB-MODULE
```

Manually

1. Delete section from `.gitmodules`
2. Commit (and or add) `.gitmodules`: `git add .gitmodules;git commit -m 'rm    PATH_TO_SU`
3. Delete section from `.git/config`
4. Remove from working tree and index: `git rm --cached PATH_TO_SUB-MODULE`
5. Remove `rm -rf .git/modules/PATH_TO_SUB-MODULE` (Optional)
6. Commit `git commit -m 'Removed sub-module PATH_TO_SUB-MODULE'`
7. Delete untracked directory `rm -rf PATH_TO_SUB-MODULE`

Since 2013 this process is supported by `git submodule deinit` that should replace some steps:

1. Deinit `git submodule deinit -f -- PATH_TO_SUB-MODULE`
2. Remove `rm -rf .git/modules/PATH_TO_SUB-MODULE` (Optional)
3. Remove `git rm -f PATH_TO_SUB-MODULE`

9 List Sub-modules

To list git sub-modules (and display their status):

```
git clone https://github.com/ckuelker/pankyll-theme-rankle-example.git
cd pankyll-theme-rankle-example
git submodule
 b2f25b9952f139d3884f214ef2e6965fd742af54 content (v0.2.0-1-gb2f25b9)
 e04b90aee4817eb8075c5534f528d6235a1ddde3 pandoc (v0.1.1)
 d9ede740610048adf2a51cb1683fc3ea9048e892 themes/pankyll-theme-rankle
  ↪  (v0.1.6)
```

The version number in brackets are displayed when the last update of the sub-module matches a tag (release) of the sub-module repository.

This is the same as `git submodule status`. Sub-modules might have sub-modules. In this case use `--recusive`:

```
git clone https://github.com/ckuelker/pankyll-theme-rankle-example.git
cd pankyll-theme-rankle-example
git submodule status --recursive
 b2f25b9952f139d3884f214ef2e6965fd742af54 content (v0.2.0-1-gb2f25b9)
 e04b90aee4817eb8075c5534f528d6235a1ddde3 pandoc (v0.1.1)
 d9ede740610048adf2a51cb1683fc3ea9048e892 themes/pankyll-theme-rankle
  ↪  (v0.1.6)
 afecf2af5d897b763e5e8e28d46aad2f710ccad6
  ↪  themes/pankyll-theme-rankle/font/\
awesome (5.15.4-3-gafecf2af5)
 4a7162e18a6a01f2ce87dec05c4e4605d8167eb6
  ↪  themes/pankyll-theme-rankle/font/\
lato (v1.0.1)
 212dc424b9fdc758a35cc97a2bbfcb184d295b61
  ↪  themes/pankyll-theme-rankle/font/\
roboto (v0.10.0-3-g212dc42)
```

10 Links

- https://www.cloudbees.com/blog/git-detached-head

11 History

Version	Date	Notes
0.1.1	2024-06-17	Initial release
0.1.0	2022-06-01	First draft

12 Disclaimer of Warranty

13 Limitation of Liability

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE INFORMATION, DOCUMENTS OR PROGRAMS AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE INFORMATION, DOCUMENTS OR PROGRAMS (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE INFORMATION, DOCUMENTS OR PROGRAMS TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.