# Software Reuse and Git Submodules
## Software Engineering for Scientists

A central tenet of software design is reusability. If you go through the trouble of writing robust and tested code that opens and parses a file, performs a statistical test, or creates a plot of the data, then it is in your interest to reuse that code as often as possible. Reusing code requires you to decouple it from a specific use, which typically begins with moving the code to its own module. Once the code is in its own module, it can be used by any other script through the `import` statement.

Once you have the code in a module, it can be tempting to just copy the file from one repo to another, but this is a bad practice and should be avoided. Bug fixes and feature updates must be applied to each instance, which is difficult to manage and the copies of the code quickly become out of sync. A better practice is to use Git submodules.

Submodules are also useful when using third-party methods in your projects. While using third-party-code is very good practice, it can be challenging when that project is also under active development. Suppose you clone some repo and begin developing software that uses that code, and after a few months, you share your code with a collaborator. Since you follow best practices, your README clearly states that your code depends on some other repository, and your collaborator clones that repo. However, there is an error when they try and reproduce your results. After a lot of heartache, you discover that the repo you depend on changed since your last clone and the two projects are now out of sync. At this point, you have to fix your code to catch up with the new changes. The frustrating thing is that this will almost certainly happen again in the future.

Suppose we want to analyze the fires in the Amazon from the data set at https://www.kaggle.com/gustavomodelli/forest-fires-in-brazil To get a sense of what the data look like we want to open the file and find some statistics about the fires. In particular, we want to know the max fires per year. Instead of writing a new method to open and read the file, we can reuse the code from an existing `utils` repo https://github.com/swe4s/utils

After cloning the fire repo https://github.com/swe4s/fire, we add the `utils` repo as a submodule.

```
$ git checkout -b new_submodule
$ git submodule add https://github.com/swe4s/utils.git
```

This command will create two staged objects (added but  not committed) in your repo

1. `.gitmodules` file that holds the submodules local path and repo URL
2. `utils` directory with a full clone of the utils repo

```
$ 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:   utils
```

`.gitmodules`

```
[submodule "utils"]
	path = utils
	url = https://github.com/swe4s/utils.git
```

According to `git status`, `utils` is a file, which offers some insight into how git treats submodules. While the submodule is a folder in your repo, git tracking of the parent repo does not extend into the submodule. Git only tracks the state of its submodules at their commit level. This is a very important aspect of git submodules. Submodules are tied to a specific commit. Updates made to submodules have no effect on the parent repo. If we make a change to `utils` that advance its commit history, all of its parent repos will still be tied to the older commit and the change will have no effect.

To check which commit a submodule is set at run:

```
$ git submodule
 0812f2a799e91dc65607f9ee1a95ca5f07b721c6 utils (heads/master)
```

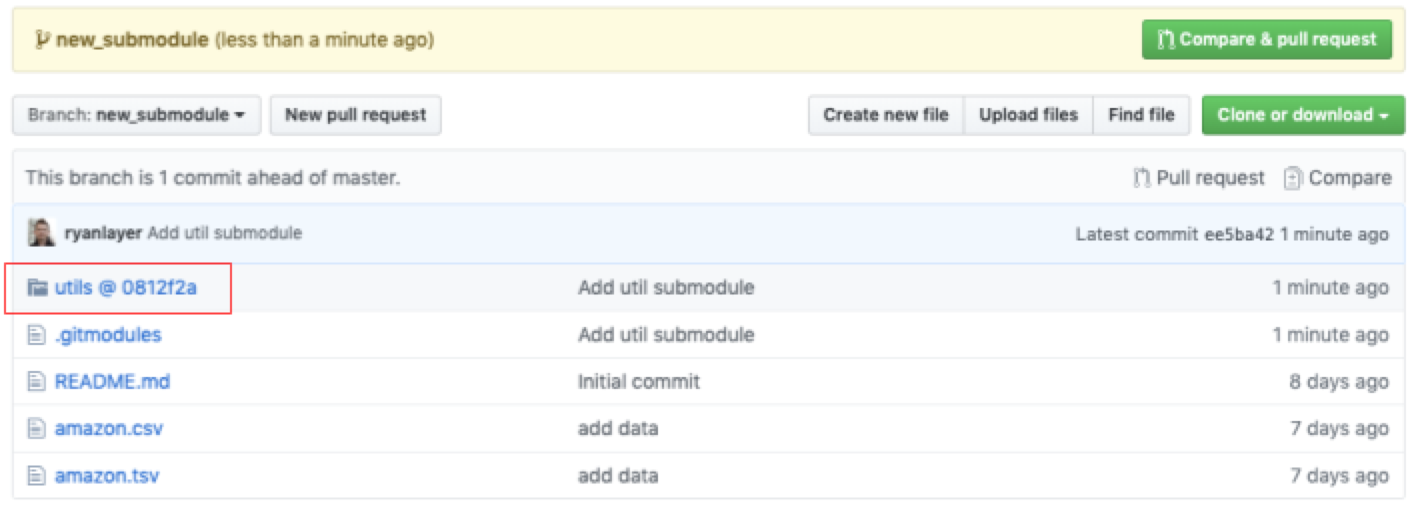
If we go into the `utils` directory, all git commands will be run with respect to the `utils` repo. We have effectively left the parent repo. If we check the log we see the `utils` history and the top commit matches the parent repo's submodule commit.

```
$ cd utils/
$ git log
commit 0812f2a799e91dc65607f9ee1a95ca5f07b721c6 (HEAD -> master,
origin/master, origin/HEAD)
Author: Ryan Layer <ryan.layer@gmail.com>
Date:   Mon Oct 14 15:48:53 2019 -0600

    add file col reading

commit b66d558c5b0878a90403702acc0b1eb414c0f9c4
Author: Ryan Layer <ryan.layer@gmail.com>
Date:   Mon Oct 14 12:50:04 2019 -0600

    Initial commit
```

At this point, these changes exist only in your local workspace. To add the submodule to your local repo the new files must be committed, and then pushed to the remote repo.

```
$ git commit -m "Add util submodule"
[master 27f416a] Add util submodule
 2 files changed, 4 insertions(+)
 create mode 100644 .gitmodules
 create mode 160000 utils
$ git push origin new_submodule
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 8 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 446 bytes | 446.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
 * [new branch]      new_submodule -> new_submodule
```

You can now see the submodule and the commit it is tied to the `fire` branch on GitHub.


new_submodule (less than a minute ago)

| | | |
|---|---|---|
| utils @ 0812f2a | Add util submodule | 1 minute ago |
| .gitmodules | Add util submodule | 1 minute ago |
| README.md | Initial commit | 8 days ago |
| amazon.csv | add data | 7 days ago |
| amazon.tsv | add data | 7 days ago |

This branch is 1 commit ahead of master.

ryanlayer Add util submodule — Latest commit ee5ba42 1 minute ago

## Cloning repos with submodules

When a repo with submodules is cloned, the folders for the submodules are empty. To get the contents of the submodules, you must run `git submodule update --init`. This is **VERY IMPORTANT** that this step is added to your .travis.yml file for any repo with submodules. Otherwise, your tests will fail.

```
$ git clone https://github.com/swe4s/submodules.git
$ ls
README.md   amazon.csv  amazon.tsv  utils
$ ls utils/
$ git submodule update --init
Submodule 'utils' (https://github.com/swe4s/utils.git) registered for path 'utils'
Cloning into 'submodules/utils'...
Submodule path 'utils': checked out '0812f2a799e91dc65607f9ee1a95ca5f07b721c6'
```

## Changes to submodules

With this submodule in place, we can now use it to find the max number of wildfires with `get_max.py`

`get_max.py`

```
from utils import file_io

file_name = "amazon.tsv"
col_nums = [3]

data = file_io.get_file_columns(file_name, col_nums)

print(max(data[0]))
```

Unfortunately it doesn't work. There are characters in `amazon.tsv` that are not compatible with how `file_io.get_file_columns` in the `util` submodule.

```
$ python get_max.py
Traceback (most recent call last):
  File "get_max.py", line 7, in <module>
    data = file_io.get_file_columns(file_name, col_nums)
  File "/Users/rl/Box
Sync/Research/teaching/swe4s/modules/submodules/utils/file_io.py", line 5, in
get_file_columns
    for l in open(file_name):
  File "/Users/rl/miniconda3/envs/swe4s/lib/python3.6/codecs.py", line 321, in decode
    (result, consumed) = self._buffer_decode(data, self.errors, final)
UnicodeDecodeError: 'utf-8' codec can't decode byte 0xe7 in position 1539: invalid
continuation byte
```

We can go into the `utils` directory/repo and make changes to that repo. Remember, once we move into the `utils` directory we have left the parent repo and entered the submodule. As long as we are in the `utils` directory, all git commands will be with respect to `utils` repo.

To fix this error, we need to specify how the file is encoded when we open it. The default is `utf-8`, but it needs to be moved to the more inclusive encoding `ISO-8859-1`
`utils/file_io.py`

```
def get_file_columns(file_name, col_nums):
    cols = []

    for l in open(file_name, encoding = "ISO-8859-1"):
        A = l.rstrip().split()
        col = []
        for i in col_nums:
            col.append(A[i])
        cols.append(col)
    return cols
```

If we make this change, then move back to the parent repo we can see that something has changed.

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

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)
  (commit or discard the untracked or modified content in submodules)

      modified:   utils (modified content)

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

If we go back into `utils` and commit this change, then check the status from the parent repo we now see that there are new commits, not just modified content. This is a warning that the commit that the parent repo expects (0812f2a) does not match the current commit of the submodule (c16f835e).

```
$ cd utils
$ git add file_io.py
$ git commit -m "fix encoding"
[detached HEAD c16f835] fix encoding
 1 file changed, 1 insertion(+), 1 deletion(-)
$ cd ..
$ git status
On branch new_submodule
Your branch is up to date with 'origin/new_submodule'.

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:   utils (new commits)

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

To move the parent repo’s commit state, run `git add` on the submodule.

```
$ git add utils
git status
On branch new_submodule
Your branch is up to date with 'origin/new_submodule'.

Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

      modified:   utils
$ git submodule
 c16f835e997cedcfe20853a6ea234eff84762ec0 utils (heads/master-1-gc16f835)
```