

Personal Project (Module) Development with OMFIT

OMFIT is a phenomenal tool for fusion research, especially on DIII-D which many of the modules are built for. This “tutorial” hopefully removes some of the frustration during the startup of a project using OMFIT to create a framework for how to develop your personal scripts, functions, and workflows in a git environment.

This document assumes you are setup with OMFIT (see [First Time Running OMFIT](#)). You have an account and have installed OMFIT either as a personal installation on iris or your personal computer. Here are [instructions](#) for doing so. See “Running a personal copy at an institution”. This document also assumes you have some general familiarity with the nomenclature of OMFIT. You must read the OMFIT First Commit Tutorial through the “Get your own copy of the OMFIT-source repository”

Some resources that should be used in parallel with this:

[OMFIT First Commit Tutorial](#)

[OMFIT Youtube Module Development](#)

[GitHub Documentation](#)

Bold items are command lines

The basic package of analysis is done in a module. This is a contained set of scripts that only rely on themselves or their stated dependencies and should run consistently once fully developed. (emphasis on *should*)

Running OMFIT on IRIS with a personal installation.

cd into the OMFIT-source directory

Make sure you are on the unstable branch and it is up to date with origin

git checkout unstable

```
[chabanr@irisc ~]$ cd OMFIT-source
[chabanr@irisc OMFIT-source]$ git checkout unstable
Switched to branch 'unstable'
Your branch is up to date with 'origin/unstable'.
Cleared temporary .pyc files
Use git hooks for this branch
[chabanr@irisc OMFIT-source]$ !█
```

Start a new local branch:

git checkout -b new_project

```
[chabanr@irisc OMFIT-source]$ git checkout -b new_project
Switched to a new branch 'new_project'
Cleared temporary .pyc files
Use git hooks for this branch
[chabanr@irisc OMFIT-source]$ █
```

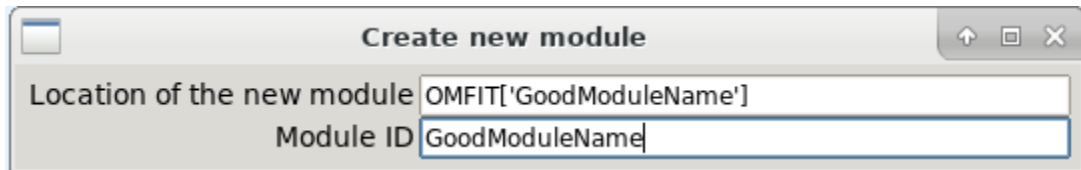
Immediately push your branch and create the upstream branch. The upstream branch refers to the thing that you will push and pull to (the merge is built in here). For example, if you are working on your personal laptop, push some changes, and then switch to IRIS and pull, those changes come from the upstream branch. [NOTE: while the upstream branch does NOT need to have the same name as your local branch I recommend you do so it is easier to keep track of. This breaks down for more complex projects with multiple developers but by the time you are working on that you won't need this tutorial ;)]

git push --set-upstream origin new_branch

```
[chabanr@irisc OMFIT-source]$ git push --set-upstream origin new_project
agent key RSA SHA256:e37Cjgjk0C+K9iUyQG5HSFnAb5pHgycj2oXFnX7Ahsg returned incorrect signature type
Total 0 (delta 0), reused 0 (delta 0), pack-reused 0
remote:
remote: Create a pull request for 'new_project' on GitHub by visiting:
remote:   https://github.com/gafusion/OMFIT-source/pull/new/new_project
remote:
To github.com:gafusion/OMFIT-source.git
* [new branch]      new_project -> new_project
Branch 'new_project' set up to track remote branch 'new_project' from 'origin'.
[chabanr@irisc OMFIT-source]$ █
```

Open OMFIT
omfit

From OMFIT go to file > new module , give it a good name and hit enter



The “Module ID” section is how OMFIT will save this module within the source code. Whatever you enter here will be the root of your module. For example this would save under */home/yourusername/OMFIT-source/modules/GoodModuleName*

Your module will be loaded in the workspace with some defaults



Open a separate terminal from where you were running omfit and run **git status**

```
[chabanr@irisb OMFIT-source]$ git status
On branch new_project
Your branch is up to date with 'origin/new_project'.

nothing to commit, working tree clean
[chabanr@irisb OMFIT-source]$
```

Hey, where is my GoodModuleName? I worked hard to make that!

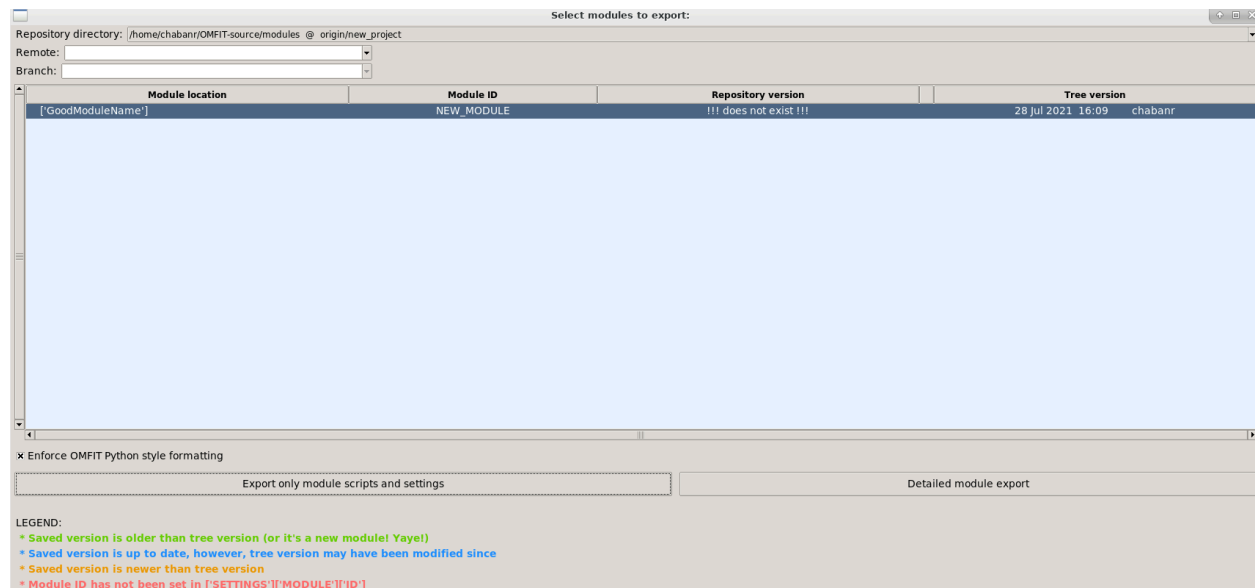
This module does not exist because we are editing a *project module* rather than a *source module*. To remedy this, we will:

1. Export the module to our local branch then
2. Convert our session to Developer Mode so that we are editing the files directly in the source area.

This is covered more in depth in the “Export modules changes from your OMFIT session” section of the OMFIT 1st commit tutorial. I strongly recommend you go check that out.

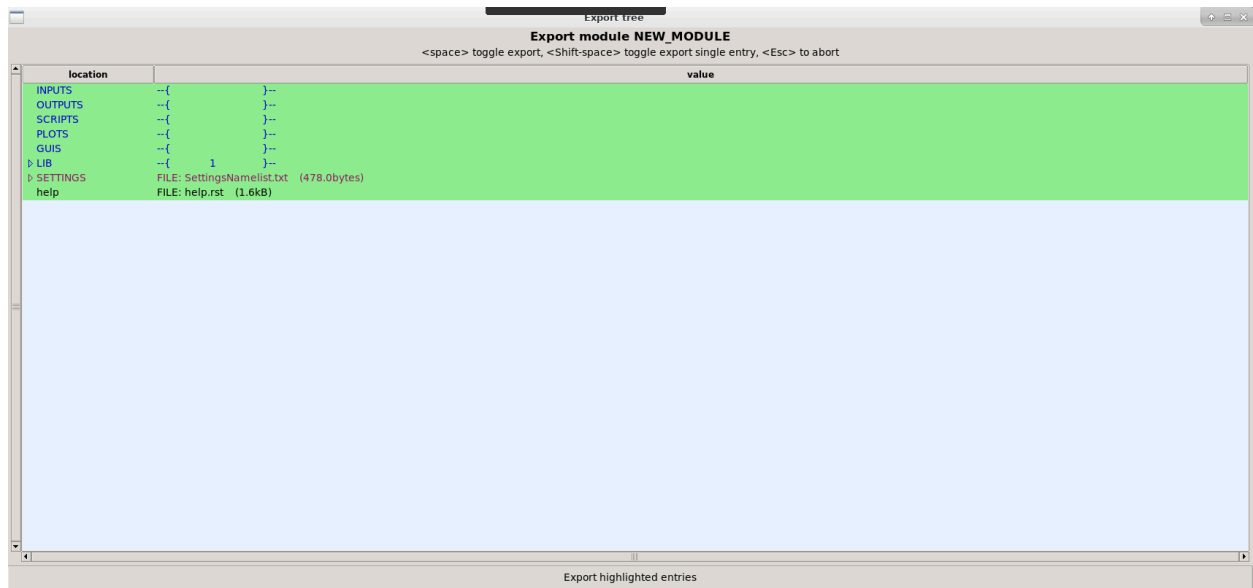
File > Export Modules

From the “repository directory:” dropdown menu select
/home/yourusername/OMFIT-sourcemodules @ origin/new_project



Note that the “repository version” does not exist. This is because OMFIT is reading all the modules in *OMFIT-source/modules* (the repository version) and the tree version (what you have open in your project) differ.

click Detailed module export



Green background means it will be exported. Click the big button at the bottom. You will know you are successful by this:

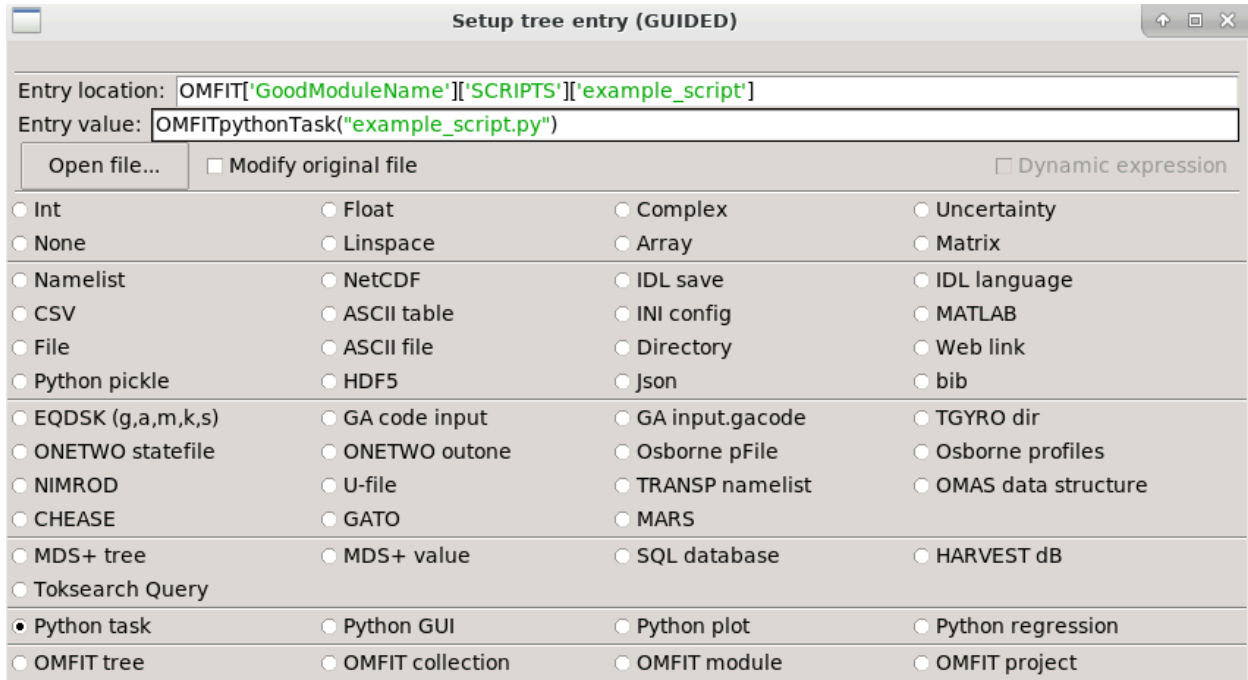
```
Moving files to save directory: /home/chabanr/OMFIT-source/modules/GoodModuleName /local-s  
cratch/chabanr/OMFIT/OMFIT_2021-07-28_16_07_05_850785/project/tmpSaveDir_2021-07-28_16_49_3  
2_814633/GoodModuleName
```

The other way to verify everything is working is by running **git status** in a new terminal while in the OMFIT-source directory:

```
[chabanr@irisb OMFIT-source]$ git status  
On branch new_project  
Your branch is up to date with 'origin/new_project'.  
  
Untracked files:  
  (use "git add <file>..." to include in what will be committed)  
    modules/GoodModuleName/  
  
nothing added to commit but untracked files present (use "git add" to track)  
[chabanr@irisb OMFIT-source]$
```

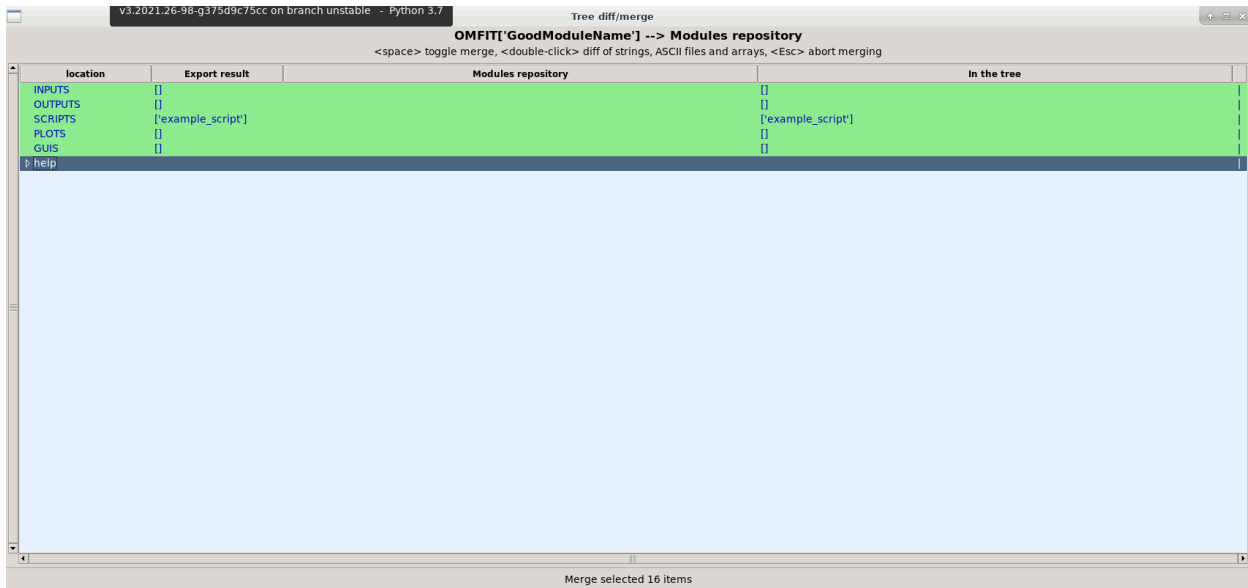
Every time you add a new script you must export it using the method above before it will be trackable by git OR before you can enter developer mode

For example: I am adding a new script: **right click on scripts and select "edit tree entry"** and **create a new key after ['SCRIPTS']** then hit enter



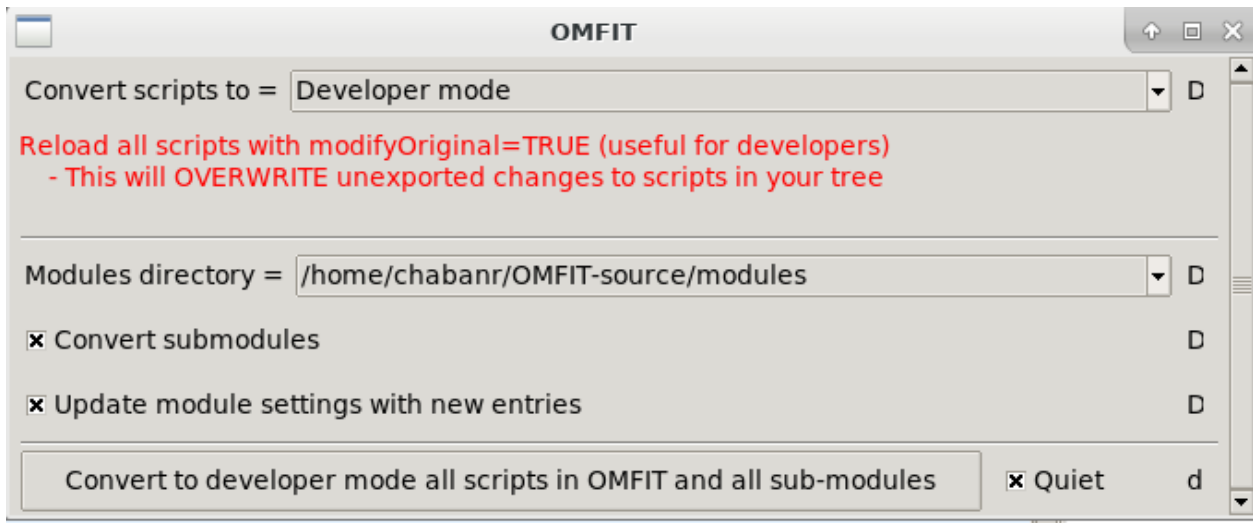
Most script-like things are stored as “Python task” and should have a “.py” at the end of the filename but *not* in their “entry location”

Follow the above steps to export it. (You will need to manually highlight everything this time)

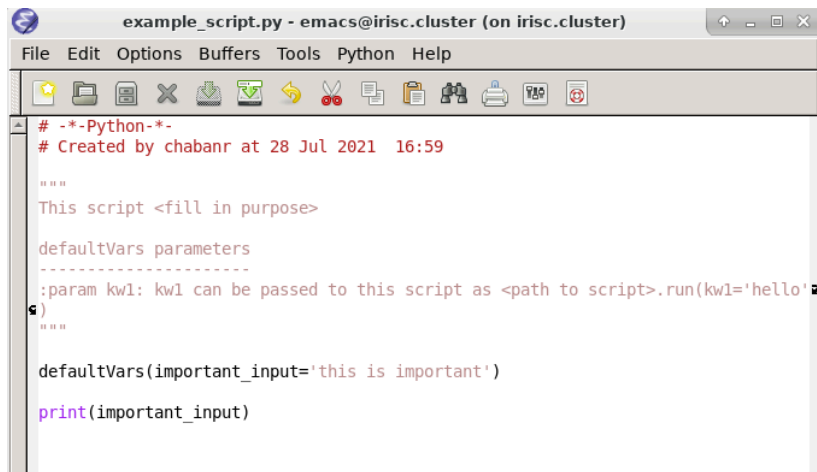
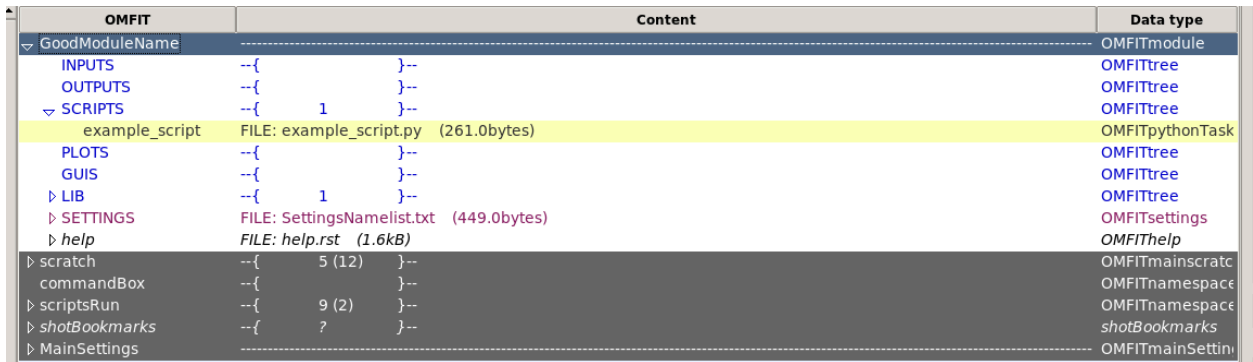


Now onto step 2: Currently we are editing a module that is stored only within our un-named project. Any files we create would be stored in that temporary scratch area where OMFIT puts things until they are saved. By entering developer mode we tell OMFIT to edit all the scripts directly in the source code directory.

click **Develop > manage modules scripts...** select **Developer Mode** from the first dropdown and click the big button at the bottom



It will look like nothing has happened. However if I make a new script in the scripts section it will turn yellow.

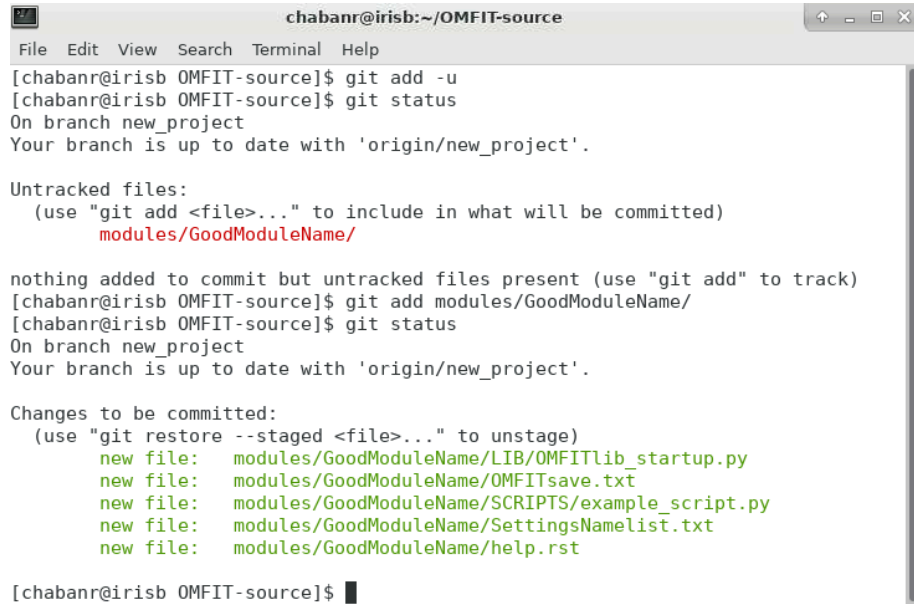


Put some code in the file:

Go to the other terminal and add your new module.

git add modules/GoodModuleName

Check with **git status**



```
chabanr@irisb:~/OMFIT-source
File Edit View Search Terminal Help
[chabanr@irisb OMFIT-source]$ git add -u
[chabanr@irisb OMFIT-source]$ git status
On branch new_project
Your branch is up to date with 'origin/new_project'.

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

nothing added to commit but untracked files present (use "git add" to track)
[chabanr@irisb OMFIT-source]$ git add modules/GoodModuleName/
[chabanr@irisb OMFIT-source]$ git status
On branch new_project
Your branch is up to date with 'origin/new_project'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
  new file:   modules/GoodModuleName/LIB/OMFITlib_startup.py
  new file:   modules/GoodModuleName/OMFITsave.txt
  new file:   modules/GoodModuleName/SCRIPTS/example_script.py
  new file:   modules/GoodModuleName/SettingsNameList.txt
  new file:   modules/GoodModuleName/help.rst

[chabanr@irisb OMFIT-source]$
```

ONE QUICK INTERJECTION

OMFIT uses the [black code formatting system](#). If you ever want to use a graphical git interface like [github desktop](#) or [sublime merge](#) as you progress with your work and knowledge of version control, you will get a lot of errors if you don't use black before you run a commit. To be fair you will get these errors with the standard command line interface of git too.

Here is an example of code that will run, but is improperly formatted according to black, and all sense of common code decency (that I placed in example_script)

```
#this is improperly done code here
x = 1234
y = 5

z = x
+ y
```

When I readded this to git (because I made changes) and tried to make the commit I got back an error:


```
chabanr@irisb:~/OMFIT-source/modules/GoodModuleName/SCRIPTS
File Edit View Search Terminal Help
Start new branch for example project

This is a tutorial on the workflow to begin a new project in python with OMFIT and
have version tracking through git.

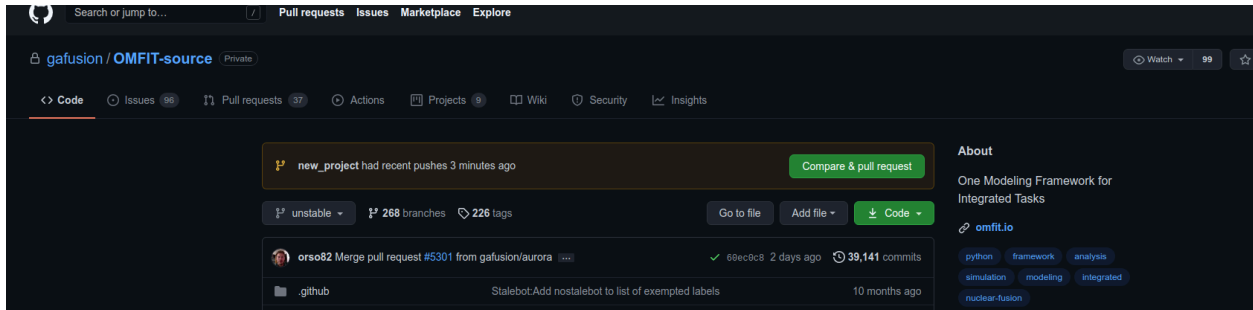
# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# On branch new_project
# Your branch is up to date with 'origin/new_project'.
#
# Changes to be committed:
#   new file:   ../LIB/OMFITlib_startup.py
#   new file:   ../OMFITsave.txt
#   new file:   example_script.py
#   new file:   ../SettingsNamelist.txt
#   new file:   ../help.rst
#
~
~
~
-- INSERT --
```

Here is the basics of editing in VIM

1. You cannot use the mouse at all to work interact with the VIM window.
2. Press the letter “i” for *insert* (thats when the bottom line will read --INSERT--)
3. Type your message normally
4. Hit *ESC* key and type “:wq” which stands for *write quit* (this appears in the bottom line)
5. If you have any issues you are probably going to need to ask someone or google it.

Making good commit messages is an important, undervalued, and underappreciated way of tracking your progress in projects. Commit small; commit often, and you will cause yourself less headache. I will leave you with [this article](#) you should peruse. Additionally, in the first commit tutorial you can find a section on how the commit message titles are displayed on the splash screen and release notes.

After you are done your terminal should look something like this:



Eventually you will want to merge your tools to the unstable branch. Or maybe you are expanding something someone else has already written. This is done through a **pull** request. In any case, once you have checked your code thoroughly (like maybe running some tests?) open a pull request with the “compare and pull request”. Type up a short description of why you want to add these changes to the unstable branch. OMFIT bots (morti) will check that your work will not break OMFIT for everyone and others will have the opportunity to comment on your code. Orso or Sterling will probably ask for changes or approve the request. Once merged you have contributed to OMFIT for everyone.

Congratulations! Good luck with your fusion code development!

How to merge unstable into your branch to stay up-to-date

Let’s say you are working with a module and you discover a bug you cannot fix. You open an issue on GitHub and it is resolved by someone with more experience than you. Yay! Now how do you get those changes into your branch you are developing? You merge unstable into your branch.

git merge unstable new_project

Depending on how long it has been you will get a short or LONG list of the changes everyone has made to the unstable (aka development) branch of OMFIT. Here is a sample of the output I got after merging unstable into my “isotope_ped” development branch:

```

modules/{STRAHL => ImpRad}/SCRIPTS/STRAHL_to_omfit_utilities.py | 0 +-
modules/{STRAHL => ImpRad}/SCRIPTS/VBtomo2.py | 0
modules/{STRAHL => ImpRad}/SCRIPTS/atomdat.py | 21 ++
modules/{STRAHL => ImpRad}/SCRIPTS/control.py | 2 +-
modules/{STRAHL => ImpRad}/SCRIPTS/device_defaults.py | 8 +-
modules/{STRAHL => ImpRad}/SCRIPTS/elms.py | 14 ++
modules/{STRAHL => ImpRad}/SCRIPTS/grid.py | 32 ++
modules/{STRAHL => ImpRad}/SCRIPTS/mds_gEQDSK.py | 11 +-
modules/{STRAHL => ImpRad}/SCRIPTS/nbi_cxr.py | 30 +-
modules/{STRAHL => ImpRad}/SCRIPTS/nete.py | 399 ++++++-----
modules/ImpRad/SCRIPTS/prepare.py | 47 ++
modules/{STRAHL => ImpRad}/SCRIPTS/reset.py | 0
modules/ImpRad/SCRIPTS/run.py | 331 ++++++-----
modules/ImpRad/SCRIPTS/run_steady.py | 224 ++++++
modules/{STRAHL => ImpRad}/SCRIPTS/source.py | 4 +-
modules/{STRAHL => ImpRad}/SCRIPTS/to omas.py | 24 +-

```

This manifests in your branch as a commit. So you will need to approve a commit message (a basic one is already written for you) :

Write your code in such a way that the user does not need to decide what happens when it is executed. Modularize your programming. My scripts are pretty straightforward with names like “calc_dimensionless”, “calc_Tsep”, “pedestal_analysis”. Crucially, all of these scripts take the same input: a string representing the data object for the shot that is present in the “INPUTS” and “OUTPUTS” section of the module.

Then I have a script titled “run_workflow” which will iterate through each script in the appropriate manner so that any script that is dependent on the output of another is run in the appropriate order. To add shots to my workflow one only needs to add the appropriate information to a dictionary in my code.

The “LIB” section of the OMFIT tree is for importing functions

Say you have a function like “my_adder(a, b)” and you want to call this in your scripts. Create a file in the LIB section of the OMFIT tree titled with the format “OMFITlib_XXXX” and then in any script you can use the line: **from OMFITlib_XXX import my_adder** and use the function just like you would any other python library.

Dont be afraid to break things

There are a lot of developers on OMFIT. If you break something (especially if it was simple), it will be discovered before it can be pushed to a place that will hurt people

Dont be afraid to ask for help

This has been a tough one for me personally. Ask the question. What might take you 4 hours to solve could take a 2 minute conversation with someone who knows and that’s just plain efficiency.

Read the documentation if it exists

Modules in OMFIT can date back to the days before python3 around 2012. Some modules have excellent documentation. Others not so much. If you find yourself getting stuck and not knowing where to go, try and find something to read.

Also get familiar with git. As everything we do moves deeper and deeper into the online world and you find yourself working on many different projects, you will need to know git. If you are bored or want to procrastinate there are worse ways than reading the [GitHub documentation](#)