Snakemake Module Vignette: Intro to Pipelines

The following documentation is to be used as an introduction to the understanding and development of Snakemake based pipeline modules which are run on a Sun Grid Engine (CentOS5) and support in Python (3.5.1).

Written By: Tim Boyarski, BSc

With assistance from: Lauren Chong, MSc

For: BC Cancer Agency – Lymphoid Cancer Research

Date of: March 28th, 2017

Contents

[I. Snakemake module organization 2](#_Toc478562672)

[II. Single Module Design - Intro 3](#_Toc478562673)

[III. Single Module Design - INCLUDE 4](#_Toc478562674)

[IV. Single Module Design - PY 8](#_Toc478562675)

[V. Single Module Design - README 11](#_Toc478562676)

[VI. Multiple Module Design - Intro 13](#_Toc478562677)

[VII. Multiple Module Design - INCLUDE 14](#_Toc478562678)

[VIII. Multiple Module Design - PY 16](#_Toc478562679)

[IX. Multiple Module Design - README 16](#_Toc478562680)

[X. References 16](#_Toc478562681)

[XI. Appendices 16](#_Toc478562682)

## Snakemake module organization

All Snakemake modules are to be stored within the modules directory. Inside the directory, modules are to be developed with the intent of high cohesion. This means that we must attempt to keep modules small and focused. Each module will have at minimum three files (.md, .py, and \_INCUDE). An example structure is outlined in Figure 1 and Figure 2.

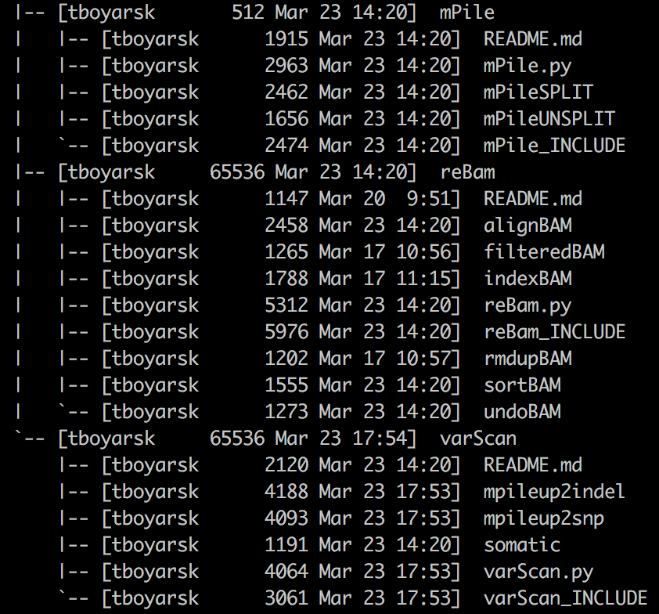


Figure . Tree structure of the module directory on Genesis

Snakemake/modules

/module0

/README.md

/module0.py

/module0\_INCLUDE

/module1

/README.md

/module1.py

/module1\_INCLUDE

/subModule1A

/subModule12B

…

/moduleN

/README.md

/moduleN.py

/moduleN\_INCLUDE

Figure . Hypothetical structure of modules directory

Naming: Modules should be named after their purpose. A *varScan* module uses varScan. A *reBAM* module re-assembles and processes ‘.BAM’ files. The modules themselves do not need to have the word ‘mod’ in them, it’s just used in the example above by chance. Sub-modules are then named by what they specifically accomplish. *sortBAM* sorts BAM files, *indexBAM* indexes ‘.BAM’ files, etc.

Multi-Rule Composition: Complex modules are decomposed into smaller events. We highly encourage that these small events are separated into their own Snakefile. They are all included into the core ‘\_INCLUDE’ file. For multi-part composition, the ‘\_INCLUDE’ should only contain the ‘include’ statements and a description about the submodules included. *When using submodules, there should be no rules in the ‘\_INCLUDE’ file.*

Core files:

* **README.md** – A read-me file written in markdown formatting.
* **moduleN.py** – A python script which accepts the ‘.YAML’ file, ‘JSON’ file, and Snakefile to be used. Acts as a script to populate the ‘.YAML’ file, ‘JSON’ file, and Snakefile.
* **moduleN\_INCLUDE** – The Snakefile to be included into the pipeline.
  + - **Single-Rule**: Contains only 1 rule.
    - **Multi**-**Rule**: Contains no rules. Contains submodule ‘include’ statements and a significant description for each submodule.
* **moduleN+1** – Contains only 1 rule. Used when module has multiple rules, each are isolated.

## Single Module Design - Intro

A key focus of module design is cohesion, keeping modules small and focused.

* Primary calls, those which run programs, should all be separated into a different module.
* Secondary calls, those which support a primary calls, include: print statements, directory creation, input redirection, parsing commands, and other administrative functions.

The following guide explains how to build the non-chromosome splitting mPile Snakemake module.

\*\*WARNING: Please note this module already exists, and has already been expanded beyond a single module design. Have a colleague assist in setting up the module workspace such that following this tutorial will not replace production ready files. This vignette assumes that the mPile module does not exist, even though it does.

1. **In the modules directory, copy and rename the Template/ directory.**

Figure 3 shows the contents inside the modules directory at the time of writing this document. The Template directory to be copied has been indicated by a yellow box. The new directory will be named “mPile”.

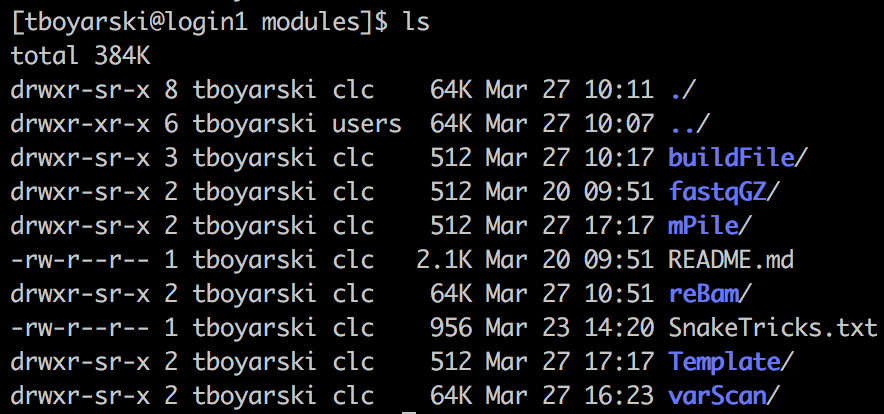


Figure . Genesis screenshot of the modules directory. Highlighted in the yellow box is the 'Template' directory users have been instructed to copy.

1. **Enter the renamed module directory and rename the files to reflect the module name.**

Only multiple module designs require the “temp\_masterINCLUDE” file, as such it can be deleted. Files containing rules, like the one we are about to write, use the template file “temp\_subINCLUDE”. For clarification in naming, please refer to Table 1. The python file and the rule containing Snakefile should be named in accordance with the module directory name, this results in the files being named “mPile.py” and “mPile\_INCLUDE”, respectively.

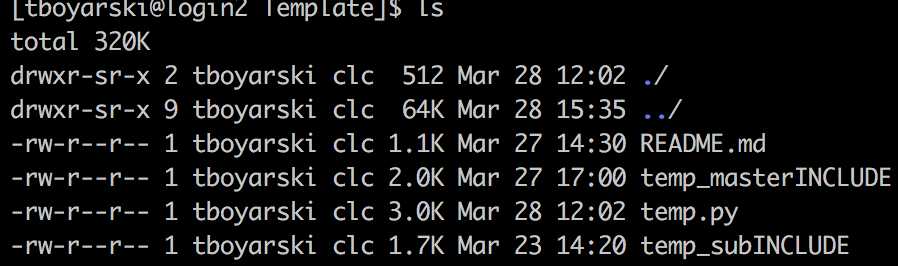


Figure . Genesis screenshot of the contents of the Template directory.

|  |  |  |
| --- | --- | --- |
| Template Name | Action | Final Name |
| README.md | No action | README.md |
| temp\_masterINCLUDE | Deleted | N/A |
| temp.py | Renamed | mPile.py |
| temp\_subINCLUDE | Renamed | mPile\_INCLUDE |

Table . A conversion table to assist in when renaming files from their template names. Names were chosen in accordance with the development of the mPile module.

### Single Module Design - INCLUDE

The “\_INCLUDE” module is the file which is provided to the Snakefile pipeline. It gives the pipeline access to the rules of the module. In this case, for a single module, the rules are contained within this “\_INCLUDE” file. In any module directory, there should only ever be a single “\_INCLUDE” file.

1. **Using VIM, open the “\_INCLUDE” file.**

Open “mPile\_INCLUDE”.

1. **Highlight the fields to be edited by moving the cursor to “XXXXXXXX” on Line 13, and pressing “Shift + #”**

Red highlighting should appear as seen in Figure 5. These are the fields that will need to be edited. The following steps will refer to the line numbers in Figure 5. The coloured boxes below are to help clarify variable usage.

Rule’s Log or Parameter Variable

.YAML Configuration File Variable



Figure . Genesis screenshot of the starting "mPile\_INCLUDE" file, after performing Step 3. The highlighted fields are areas of text which will need to be replaced. The highlighting can be turned off using the VIM command ":noh".

1. **Go to Line 2 and Line 3, insert your name and the date.**
2. **Go to Line 5, edit the python call.**

Typically, this should just involve replacing the highlighted X’s with the module name, as seen in Figure 6.

../../Desktop/Screen%20Shot%202017-03-30%20at%209.22.19%20AM.png

Figure . VIM screenshot of Line 5 from the mPile\_INCLUDE module. Allows for easy access when building a pipeline build file.

1. **Go to Line 6, Line 7, and Line 8, insert the input file type, output file type, and the module’s purpose.**
2. **Go to Line 13, edit the name of the module to be the same as the file prefix.**
3. **Go to Line 15, edit the input for the module.**

A lot of the power in Snakemake and Make comes from their ability to specify starting input files, targeting with precision and ambiguity simultaneously. With this power comes a complexity.

VARIABLES

FUNCTIONS VARIABLES

EXAMPLES VARIABLES

Named Wildcards: Indicated by double {} E.G. {{namedWildcard}}

Wildcards are automatically determined via regex parsing of the argument used to run your pipeline.

Inline Variables: Indicated by single {} E.G. {rndVAR}

Inline variables can be used as place holders when populate with data, like from the YAML config file.

They can be of the following formats:

Strings … rndVAR=“Strings are formatted in quotes”

Lists … rndVAR=[“Lists”, “follow”, “python”, “formatting”]

YAML Parameters … rndVAR=config[“paramNAME”]

Input Function: Functions can be used in the input directive, provided they have a return value. Functions called from the input directive are automatically given the Snakemake Wildcard Object. As such, the method signature of must accept a formal parameter. For consistency, please always name the first formal argument “wildcards”. This is not used in this module, as such, the concept will not be further explained nor will examples be given.

Expand Function: The expand function is the most commonly used Snakemake function. It is used to generate the sets of input and output files required. Variables passed into this function perform a full union. For more information on the expand function, please refer to the following link to the [Snakemake documentation](http://snakemake.readthedocs.io/en/latest/tutorial/basics.html?highlight=expand()#step-5-calling-genomic-variants).

Using:

X = “output”

Y = “mutect”

Wildcard1 = “Pfeiffer”

Z = [“1”,”2”]

What does this become?

$expand(“{var1}/{var2}/{{wildcard1}}\_{var3}.fa”, var1=X, var2=Y, var3=Z)

It becomes:

$output/mutect/Pfeiffer\_1.fa, output/mutect/Pfeiffer\_2.fa

The resultant mPile input call should be the same as Figure 7.

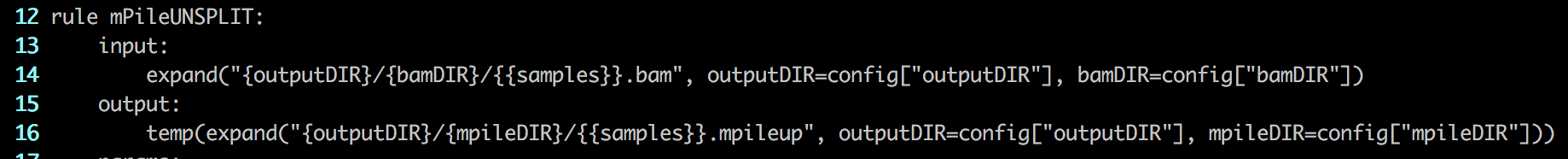


Figure . VIM screenshot of Line 14 of the mPileUNSPLIT module. This module is named differently from the vignette because it has since been converted to a submodule.

1. **Go to Line 17, edit the output for the module.**

Aside from two exceptions, the output directive is designed the same as the input directive.

Exception 1: The output directive cannot use functions

Exception 2: The output directive has 2 file protection tags

* + 1. temp(fileName) – This file will be deleted when it is no longer required by the pipeline.
    2. protected(filename) – This file is protected from being deleted by the pipeline.

The resultant mPile output call should be the same as Figure 8.

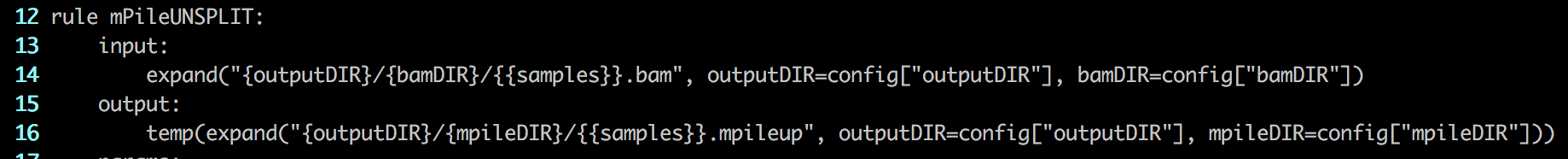


Figure . VIM screenshot of Line 16 of the mPileUNSPLIT module. This module is named differently from the vignette because it has since been converted to a submodule.

1. **Go to Line 19, edit the name of the log file.**

This should be the same as the rule name on Line 13 and the prefix of the file name.

1. **Go to Lines 20, edit and add more parameters as needed.**

Any additionally required parameters which were not specifically list in the ‘.YAML’ configuration file can be specified here. Python calls can be utilized here as well as reference can be made to the “.YAML” file. For ease of reading it is encouraged that users combine related arguments together, as seen in Figure 9. This is especially encouraged when there are five or more arguments to be passed to the program.

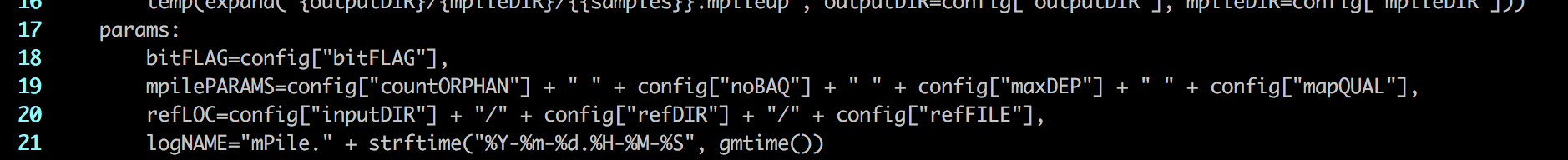


Figure . VIM screenshot of Line 19 and Line 20 of the mPileUNSPLIT module. This module is named differently from the vignette because it has since been converted to a submodule. The variable mpilePARAMS is a union of all the parameters which will have to be passed to the program mpileup. This makes the resultant bash shell call short and therefore easier to read.

1. **Go to Line 22, edit the name of the log directory.**

This should be the same as the name of the directory this module is contained within.

1. **Go to Line 24, add the run call(s).**

Run calls execute in python. As such, to make shell calls when using the “run” directive. The resulting call should look like Figure 10.

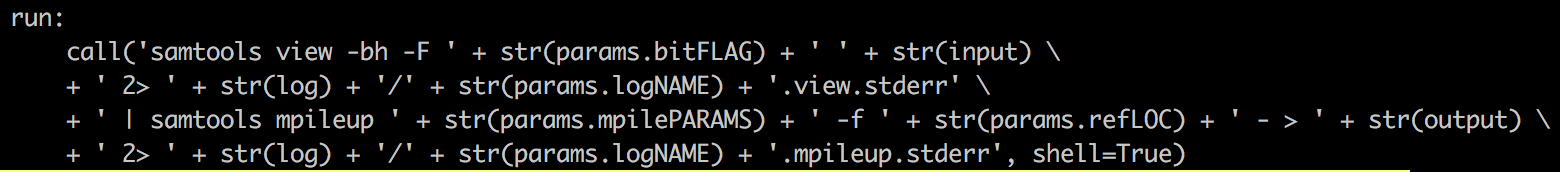
****

Figure . VIM screenshot of mPileUNSPLIT run directive and the argument it calls. The standard error for both portions of the call is redirected to log files.

1. **Delete the extra hints at the bottom.**

At the time of reading this, they may or may not exist. They are considered everything after the long-dashed line. Keep the long-dashed line. They are currently offered during the development phase for ease of access when creating new modules. An example of a finished module is available in Figure 11.



Figure . VIM screenshot of the mPileUNSPLIT module on Genesis. This module was the original mPile\_INCLUDE, but expansion required the creation of submodules. mPile\_INCLUDE was renamed to mPileUNSPLIT when it was converted to a submodule.

### Single Module Design - PY

The “.py” python script is the file which helps auto-populate the constructed Snakefile, ‘.YAML’, and ‘.JSON’ files. It provides the rules, calls, and the default arguments for the calls. There should only ever be one “.py” file.

1. **Using VIM, open “mPile.py”, formerly temp\_subINCLUDE.**
2. **Edit the comments.**

Take the time to edit add your name, the date, the input, the output. Typically, the purpose can be left unmodified, the default description is sufficient. Editing it may alter the line numbers being referenced.

1. **Highlight the fields to be edited by moving the cursor to “XXXXXXXXX” on Line 11, and pressing “Shift + #”.**

To remove this, type “:noh” into VIM, or, eliminate them by filling in all the fields.

1. **Go to Line 18, edit the value of the variable.**

This name should be the same as the name of the directory this module is contained within, as seen in Figure 12. The module name is used multiple times throughout when populating the Snakefile, the ‘.YAML’, and the ‘.JSON,

files. The name is used for user reporting. When the python script is run, the outputs describe its actions.

../../Desktop/Screen%20Shot%202017-03-29%20at%2011.07.26%20AM.png

Figure . VIM screenshot of the module name used.

1. **To preserve line number references, will start from the bottom.**

Certain sections will require multiple lines of code to be organize, as such, this will push down the lines below. By starting from the bottom, we will not affect the lines until we start altering that line’s location.

1. **Go to line 73, edit the call used for this module.**

The statements required for all rules should be listed here. Use the output statements for the rules as templates. Convert the output statement {{samples}} from a wild card. It should become {samples} and have an

accompanying declaration at the end, “samples=config[“sample]. The template is shown in Figure 13, it is converted to the proper call, as seen in Figure 14.

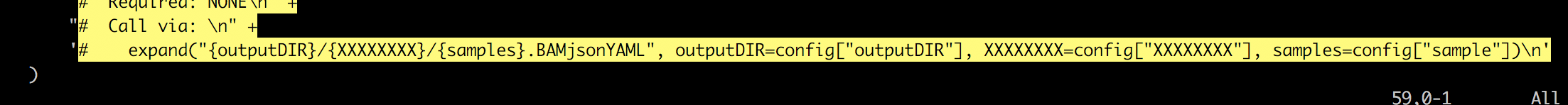


Figure . VIM screenshot of the code to be replaced inside the python file being built.

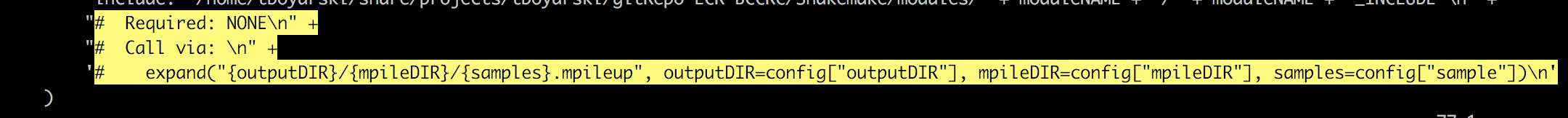


Figure . VIM screenshot of the code which replaced the template text for mPile.py

1. **Go to line Lines 68 and 69, reflect here what was listed in Lines 5 and 6 of this file.**

This output is used to provide context to what the module does when being viewed from the Snakefile.

1. **Go to Line 57, if using submodules edit this, otherwise, do not change.**

In a single main module, everything is named the same to make variable population easier, as seen in Figure 15. Otherwise, if there are submodules to this unit, add the names of the rules within each submodule, as seen in Figure 16.

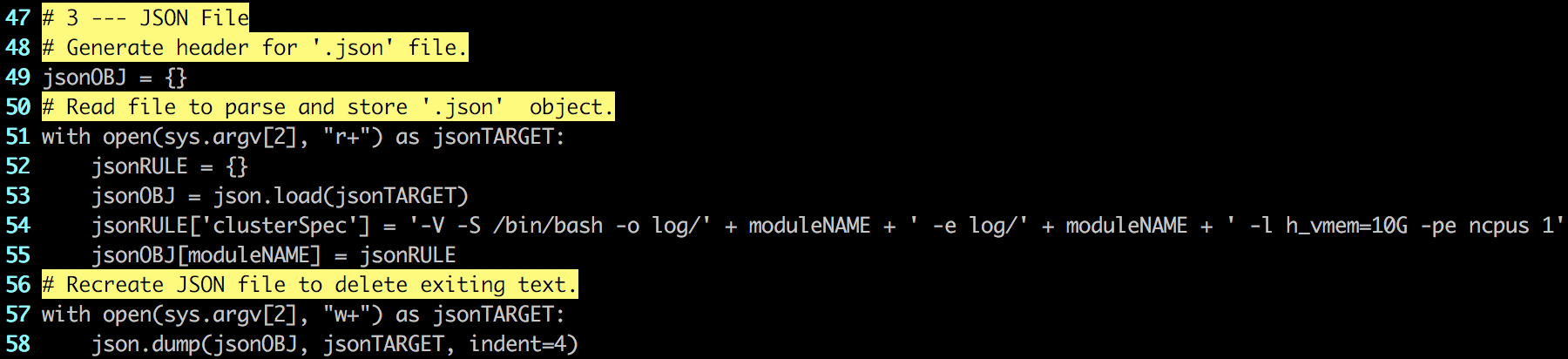


Figure . Genesis screenshot of the template structure provided. If only a single module is used, nothing needs to be changed as the "moduleNAME" variable listed at the top of the file will populate all the necessary fields.

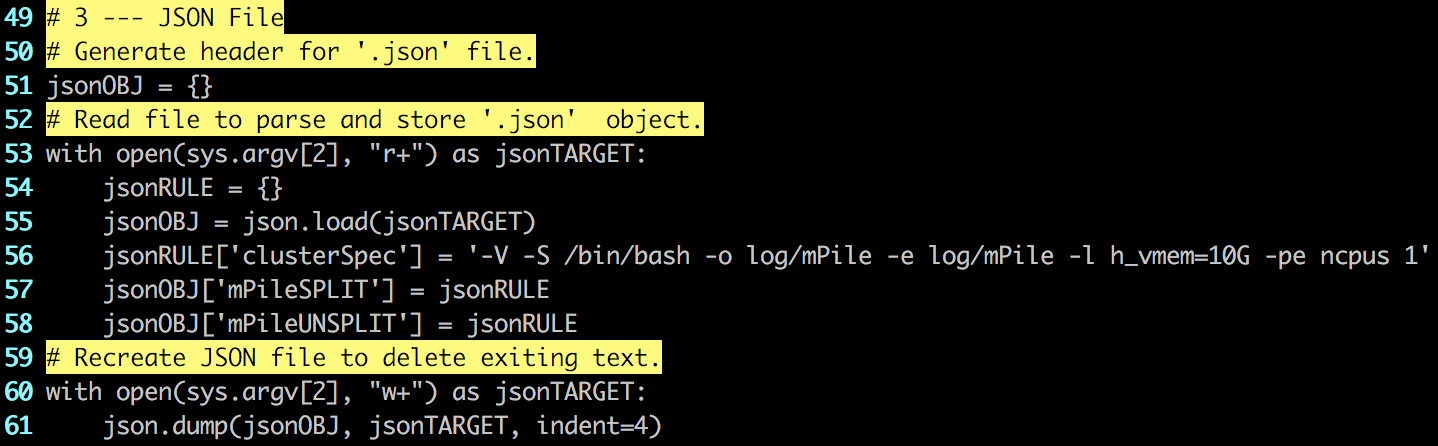


Figure . Genesis screenshot of the JSON File section from the module mPile. mPile has two modules, mPileSPLIT and mPileUNSPLIT, they are added to the JSON object on Line 57 and Line 58 respectively.

1. **Go to Lines 42 and 45, add your parameters here.**

The parameters listed here should follow the format of line 42, and should be added to the Line 45 “.write()” statement. Add all the parameters that were used in the coding of the modules, as seen in Figure 17. This python script will populate a file in “.YAML” format. Existing examples can be see below.

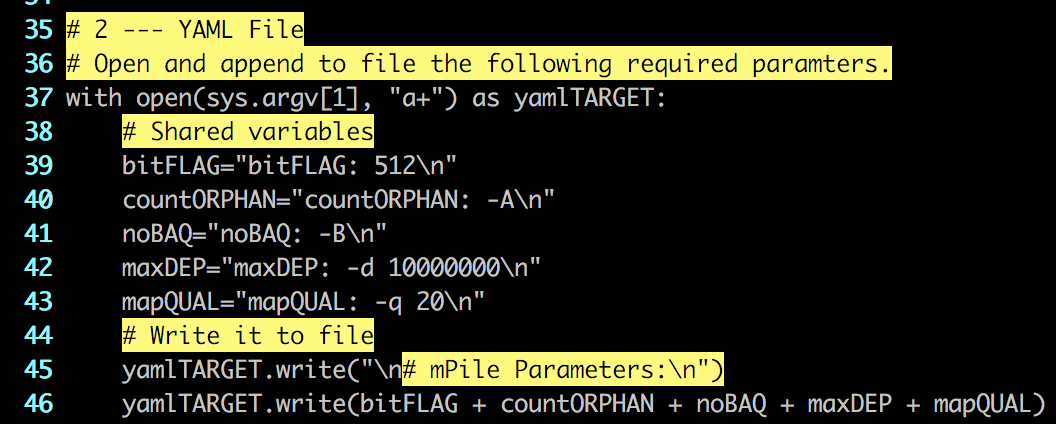


Figure . Genesis screenshot of the YAML file section of the module mPILE. Both modules use the same parameters.

1. **When finished, “mPile.py” should be the same as Figure 18.**

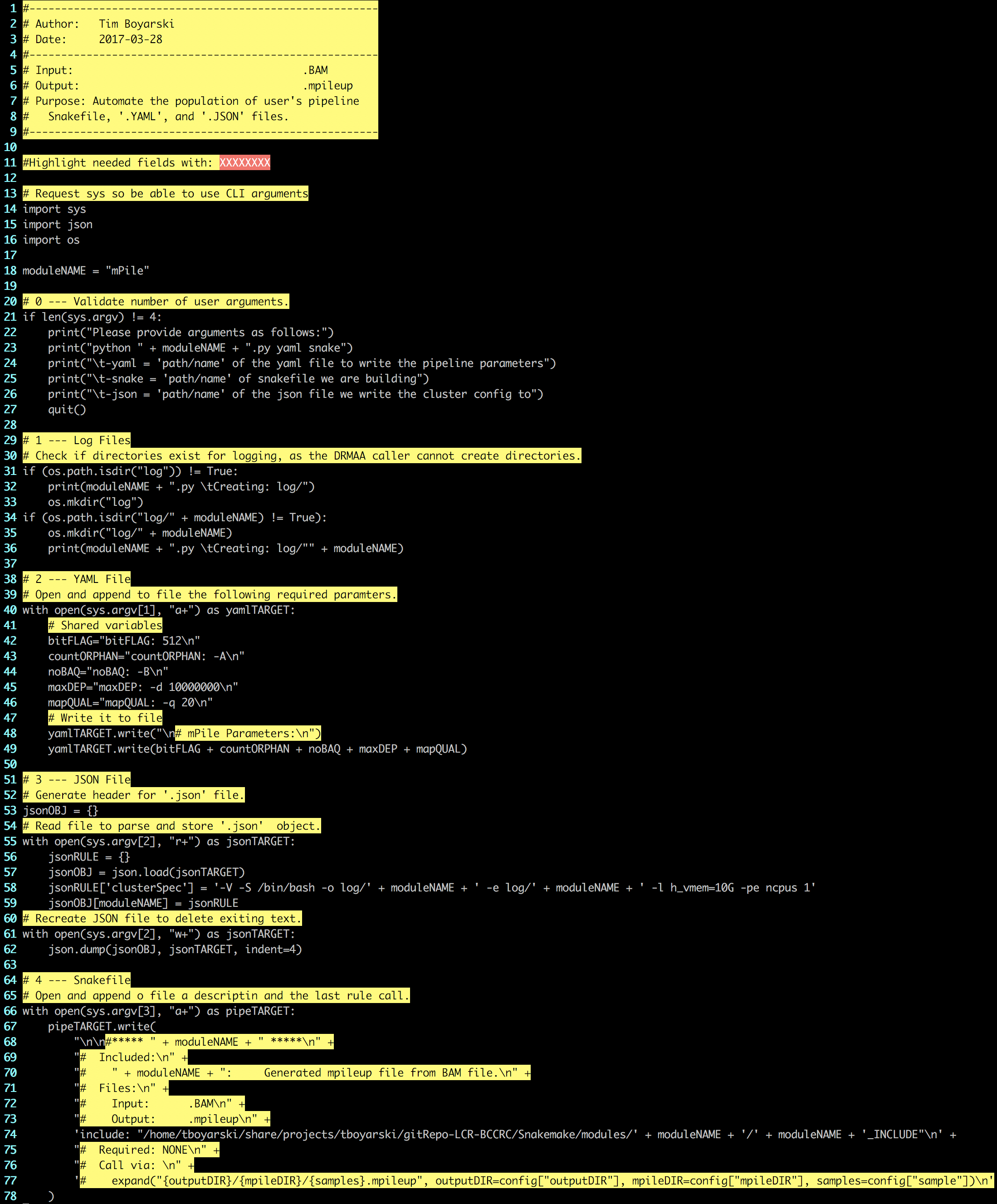


Figure . Final result of the "mPile.py" script, as required for this Vignette. The actual file on Genesis has since been altered as mPile was turned into multi-modules.

### Single Module Design - README

The README.md file is used as a point of reference for users on Genesis, and when viewing the repository on GitHub. The comments can be used as guidance on what to write.

1. **Module (Snakemake)**

Provide the name of the module and the program language in which its written in. This description describes the type of module, not what it does.

1. **Modules**

List each of the modules and provide a quick description as to the purpose of each. Full length descrtipitons should be contained in the specific module file.

1. **Logging**

Just change the directory in which it is being stored. The directory will be named the same as this module’s.

1. **Global Directories**

All of the global directories and their descriptions are already listed, for ease of access. Delete the ones that this module will not require.

1. **Global Parameters**

All of the global parameters and their descriptions are already listed, for ease of access. Delete the ones that this module will not require. Note the four-column table layout, as exemplified in Figure 19 and Figure 20.

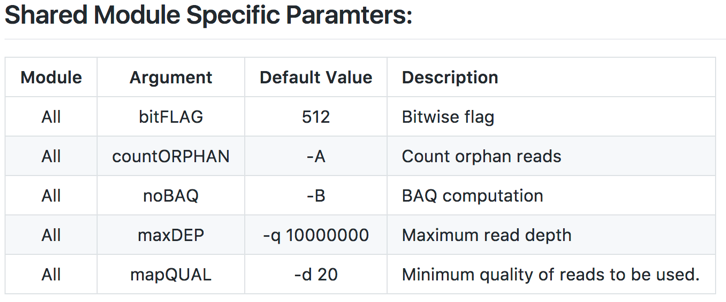


Figure . GitHub screenshot of the mPile module README.md. The image shows how the table format code is represented visually.

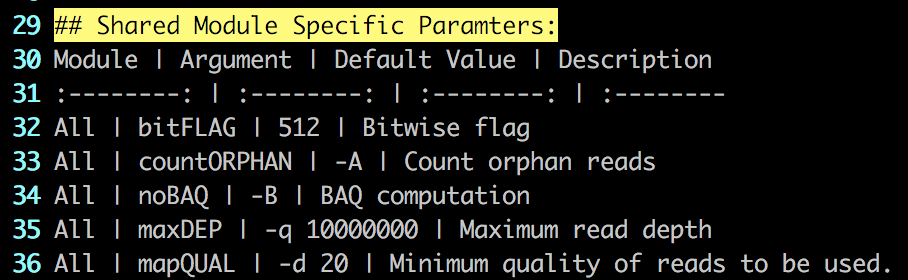


Figure . Genesis screenshot of the mPile module README.md. This code shows structure of the four-column markdown table. Column positions are determined by the colon's in Line 31.

1. **Module Specific Parameters**

The module specific parameters and their descriptions are to be listed here. Follow the formatting of Step 6.

1. **When finished, “README.md” should look similar to Figure 21.**

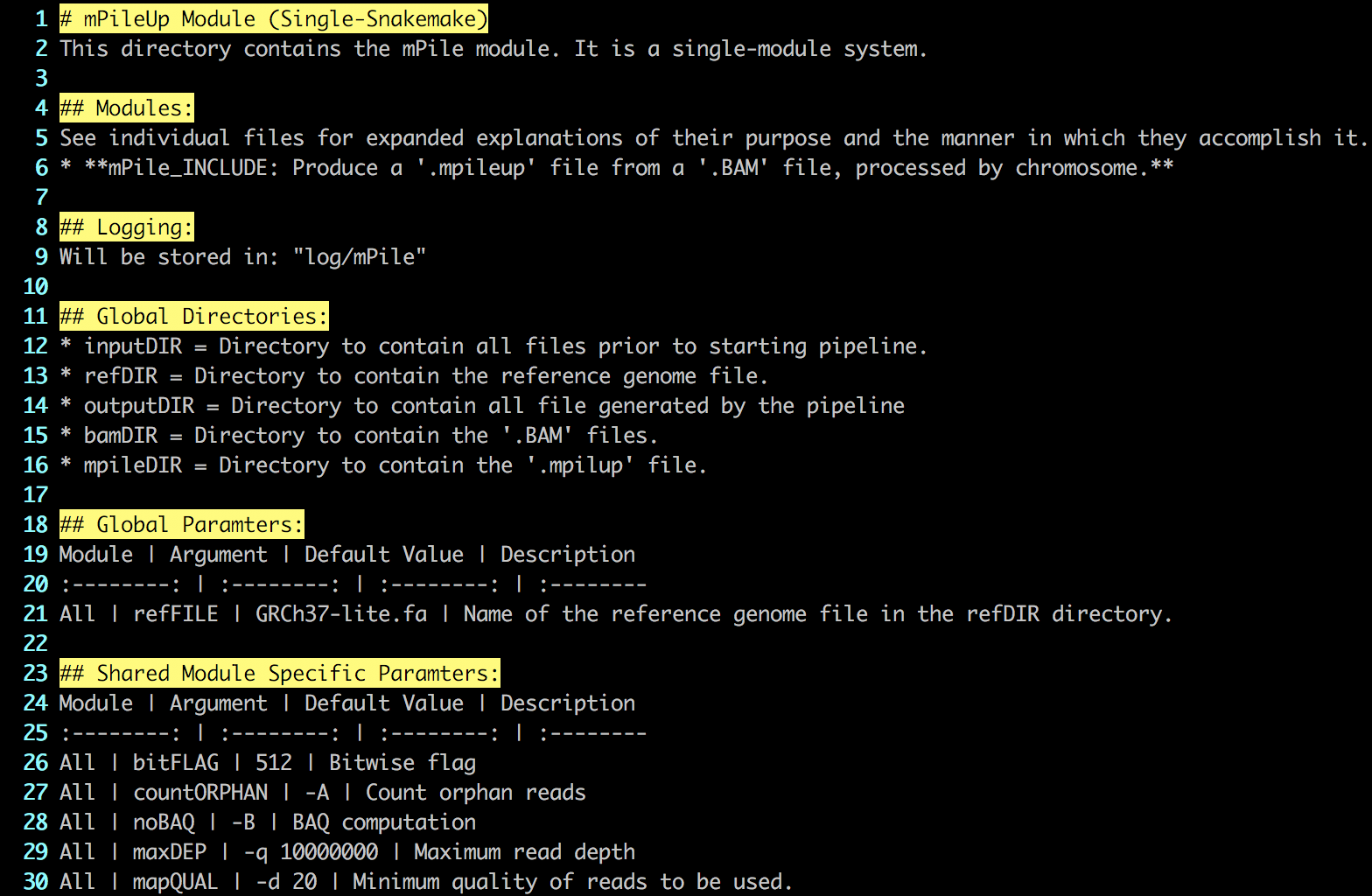


Figure . VIM screenshot of the README.md file for the mPile module. The current README.md on Genesis for the mPile module is now different due to expansion of the module.

## Multiple Module Design - Intro

A key focus of module design is cohesion, keeping modules small and focused.

* Primary calls should all be separated into a different module and are those which run programs.
* Secondary calls support a primary call include: print statements, directory creation, input redirection, parsing commands, and other administrative functions.

The following guide explains how to build the non-chromosome splitting mPile Snakemake module

1. **In the modules directory, copy “Template/” and name the copy with the module name.**

Figure 22 shows the contents inside the modules directory at the time of writing this document. The Template directory to be copied has been indicated by a yellow box.

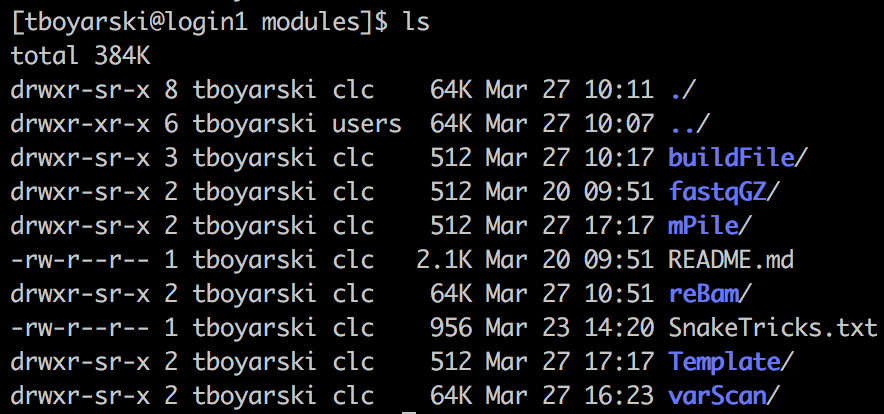
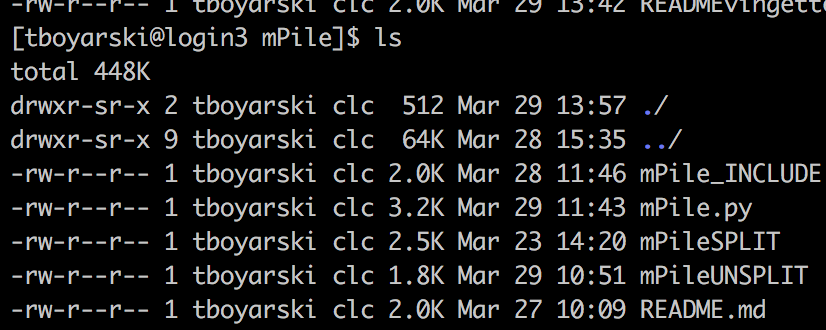


Figure . Genesis screenshot of the modules directory. Highlighted in the yellow box is the 'Template' directory users have been instructed to copy.

1. **Enter the renamed module directory and rename the files to reflect the module name.**

Files containing rules, like the one we are about to write, use the template file “temp\_subINCLUDE”. For clarification in naming, please refer to Table 1. The python file and the rule containing Snakefile should be named in accordance with the module directory name; this results in the files being named “mPile.py” and “mPile\_INCLUDE”, respectively. The starting and ending file structure is exemplified in Figure 23 and Figure 24 respectively.



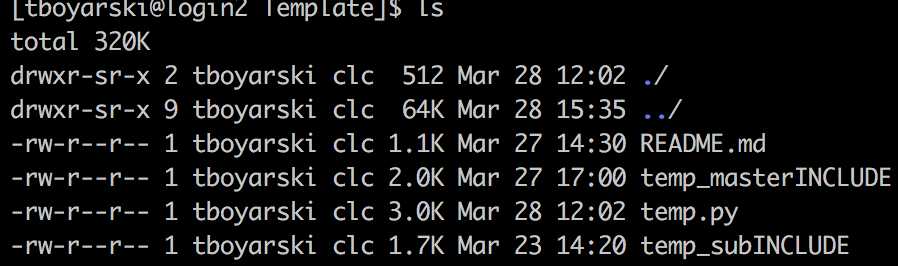


Figure . Genesis screenshot of the contents of the Template directory.

Figure . Genesis screenshot of the contents of the mPile directory after the files have been named correctly.

|  |  |  |
| --- | --- | --- |
| Template Name | Action | Final Name |
| README.md | No action | README.md |
| temp\_masterINCLUDE | Renamed | mPile\_INCLUDE |
| temp.py | Renamed | mPile.py |
| temp\_subINCLUDE | Copied | mPileUNSPLIT |
| temp\_subINCLUDE | Renamed | mPileSPLIT |

Table . A conversion table to assist in when renaming files from their template names. Names were chosen in accordance with the development of the mPile module.

### Multiple Module Design - INCLUDE

Submodules can be built using the “Single Module Design – INCLUDE” guide. The only difference is the name of the file and the contained rule, as per Table 2 above. The two submodule names will be mPileUNSPLIT and mPileSPLIT. The rule name within the submodule should be the same as the file name, and cannot be the same as the directory name. The directory name is reserved to always carry the “\_INCLUDE” tag. Submodules will not carry the “\_INCLUDE” suffix.

1. **Using VIM, open “mPile\_INLCUDE” which was formerly “temp\_masterINCUDE”.**
2. **Go to Line 2 and 3, add your name and the date.**
3. **Go to Line 5, edit the path to reach the python file.**

The module should be inside a directory with the same name. Replace “XXXXXXX” with the name of the module.

1. **We will now start from the bottom, as to preserve number lines referenced.**

Certain sections will require multiple lines of code to be organize, as such, this will push down the lines below. By starting from the bottom, we will not affect the lines until we start altering that line’s location.

1. **Go to Line 20/21 and Line 25/26, add the module names mPileUNSPLIT and mPILESPLIT, respectively.**
2. **Go to Line 22/23 and Line 27/28, list the submodule specific variables.**

Provide a brief description of what the parameter represents. If there are no parameters, list None.

1. **Figure 25 demonstrates a finished two-submodule include section.**

If additional modules were needed, repeat the format and structure of the two modules just created. For readability, separate by a blank line.

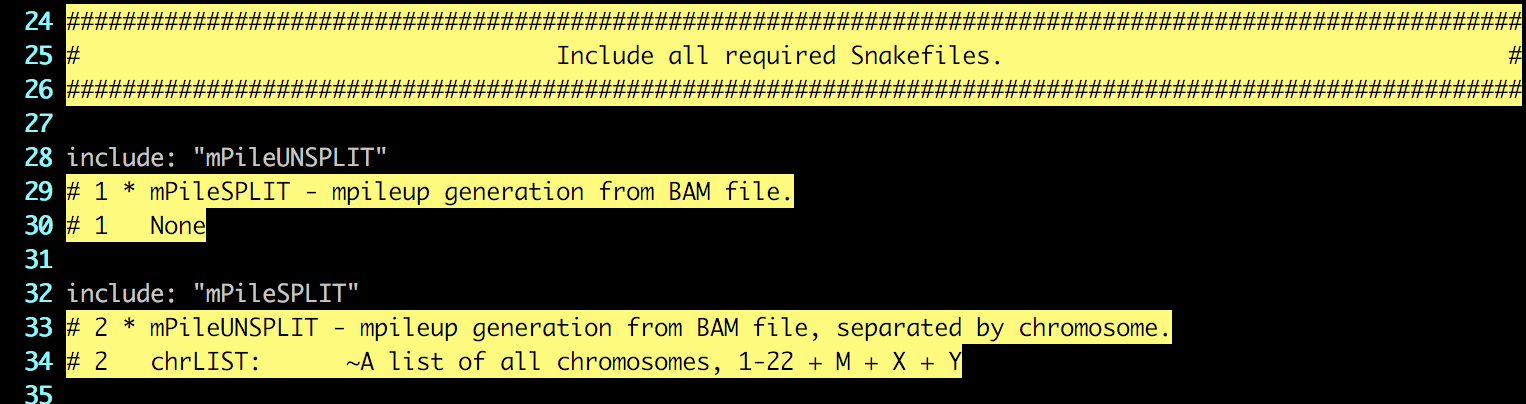


Figure . Genesis screenshot of the mPile\_INCLUDE file. The module mPileUNSPLIT, included on Line 28, does not have any exclusive variables. The module mPileSPLIT, included on line 32, is the only submodule which requires the list of chromosomes, chrLIST.

1. **Go to Line 13, list all parameters required by the module.**

These are the ‘.YAML’ configuration file parameters which are used, Global and Shared.

1. **Go to Line 8, update the purpose.**

This should describe the starting and ending points of the module and a very higher level.

1. **When finished, “mPile\_INCLUDE” should look similar to Figure 26.**



Figure . VIM screenshot of the finalized "mPILE\_INCLUDE". This image is of the Genesis file, as of March 28th, 2017.

### Multiple Module Design - PY

Same as Single Module Design – PY.

### Multiple Module Design - README

Same as Single Module Design – README.

## References

Köster, J., & Rahmann, S. (2012). Snakemake—a scalable bioinformatics workflow engine. *Bioinformatics*, *28*(19), 2520-2522. [Available at: https://pypi.python.org/pypi/snakemake]

Python Software Foundation (2017) Python. [Available at: https://www.python.org/]

## Appendices

Clean.sh

This file is used when developing modules. It assists in cleaning out the working environment of all produced files and outputs so that both the python build file and the Snakemake calls can be performed again. This file is a convenience function and otherwise serves no purpose for this pipeline.

#!/bin/bash

rm -rf output/

rm -rf log/

rm -rf .snakemake/

rm Snakefile

rm input/config.yaml

rm input/config.json