Snakemake Vignette: Intro to Pipelines

Written By: Tim Boyarski, BSc
With assistance from: Lauren Chong, MSc
For: BC Cancer Agency – Lymphoid Cancer Research
Date of: August 31st, 2017

Introduction

The following vignette is to be used as an introduction to the understanding and development of Snakemake based pipelines using the newly designed LCR Snakemake Pipeline System. This document provides a step-wise guide how to create, configure, and execute a Snakemake pipeline.

The vignette is applicable for users regardless whether they are running the pipeline locally or on a high-performance computing cluster. It is intended for audiences who are new to Snakemake, including those who have already completed the online tutorial as provided by the language's author, which can be found here:

https://snakemake.readthedocs.io/en/latest/tutorial/tutorial.html

Table 1. Documenting the changes to this vignette document

Date	Author	Contact	Changes
2017-08-25	tboyarski	tim.boyarski@gmail.com	Final Version

Contents

Introduction 1

I. Terminology 3

II. Pre-Conditions 4

III. Snakemake module organization 5

IV. Workspace organization 6

V. Conda activation 8

VI. Workspace creation 9

1) Navigate to "py_startHERE" module to interact with the 'startHERE.py' 9

2) Create a workspace with fastqGen, fastqUtil, bamGen, bamUtil, bamMetrics 9

3) Change directories to enter the new workspace, open 'buildPipe.py', and correct misspelled modules 11

VII. Workspace configuration 12

1) Configure the variable "TYPE" in buildPipe.py 12

2) Configure the variable "REFFILE" in buildPipe.py 12

3) Configure the variable "snakeDIR" in buildPipe.py 13

4) Review 'buildPipe.py' 13

VIII. Building and configuring the Snakefile 14

5) Execute the python script 'buildPipe.py' 14

6) Edit the Snakefile input directive 15

7) A finalized 'Snakefile' should look similar to Figure 19 16

IX. Pipeline configuration 17

1) Add raw '.bam' files into 'input/rawBAM/' 17

2) Edit 'sampleFILEsingle.txt' to include your sample names. 18

3) Consider editing the JSON file to reflect cluster specifications. 19

4) Edit the YAML file, if desired. 19

X. Running a pipeline 21

1) Dry-run the pipeline. 21

2) Run pipeline. 22

XI. References 23

XII. Appendices 23

I. Terminology

- **Workspace**: This term refers to the current, or potential, directory where the pipeline is located.

- **Modules:** Typically start with the format of the output file, then they include the purpose(s), and lastly if needed, the software used by all sub-modules.

    - A vcfGenUtil_*varScan* module uses is able to generate or utilize '.vcf' files, and all its operations are done using the processing functionality of VarScan.

    The most commonly used categories used for naming are:
        - Gen - Ability to output files which are of a format than the inputs.
        - Util - Ability to manipulate files without changing their format or summaries files.
        - Annotate - Ability to annotate existing files without changing their format.
        - Metrics - Ability to output files providing metrics on existing data.

- **Sub-modules:** Typically, an action verb and a file format, sub-modules are named by what they specifically accomplish. If two rules have the same functionality but accomplish it using different software, then the sub-modules can be suffix'ed with an '_', and then the software name.

    - *mergeVCF – merges* '.vcf' files
    - *sortBAM_bioambam* – sorts '.bam' file using biobambam
    - *sortBAM_samtools* – sorts '.bam' file using samtools

****NOTE**** *The alignment sub-modules contained within "bamGen" do not follow this nomenclature.*

Complex modules are decomposed into smaller, rule-containing, sub-modules. We highly encourage that these small rules are separated into their own file. Rules are to be collectively gathered into the module-specific '_INCLUDE' file which should only contain comments, control flow of internal sub-modules, and the sub-module 'include:' statements. Control flow is achieved using conditionals and "ruleorder". This control flow allows for the simplistic swapping of equivalent rules running different software, and to set rule precedence when rules are ambiguous because they are capable of producing the same output file.

If not already available, the modules are to be copied from online from the "LCR-BCCRC/workflow_exploration" private repository. Please download the entire Snakemake directory as it contains the modules directory, and this vignette expects "modules" to be a sub-directory. A direct link to the repository can be found below:

Private: https://github.com/LCR-BCCRC/workflow_exploration

The private repository is actively maintained. For the purpose of transparency and accessibility, a public facing repository has been made available. It will not be actively maintained, and it reflects the state of the private code base at the end of development by the initial author. For those with access, please utilize the private repository link above and get the most up-to-date vignettes. In the event you are not apart of the BC Cancer Agency and do not have access to the Lymphoid Cancer Research Departments private repository, please refer to this static copy of the repository and vignettes, which can be found a public repository hosted on the GitHub account of the initial author:

Public: https://github.com/tboyarski/BCCRC-Snakemake

II. Pre-Conditions

1) This vignette was validated using the core software listed in Table 2 below. The vignette will likely also work with the most up-to-date versions of the software listed, however, this is not guaranteed.

Table 2. Software versions utilized during the creation of this vignette. Newer versions are likely not to have redacted existing functionality, as such, it is highly likely that this vignette is compatible with the most up-to-date versions of the software listed.

Software	Vignette Version	Theoretical Compatibility
CentOS	5	6
Python	3.5	≥3.5
Snakemake	3.10.1	≥3.10
DRMAA	0.7.6	≥0.7.6

2) The conda environment was created using the following '.YML' file. Except for the installation of Snakemake and Python, the dependencies listed are software required for the successful ***execution*** and processing of the finalized pipeline; however, they are not required for the ***generation*** of the Snakemake pipeline.

```
name: CentOS5-Compatible
channels:
    - bioconda
    - defaults
dependencies:
    - biobambam=2.0.39=0
    - bwa=0.7.13=0
    - java-jdk=8.0.92=0
    - picard=2.7.1=py35_1
    - samtools=0.1.18=0
    - snakemake=3.12.1=py35_1
    - snpeff=4.1l=0
    - varscan=2.4.0=0
    - drmaa=0.7.6=py35_0
    - python=3.5.1=5
```

3) All white space formatting must be performed exclusively with spaces. The system requires that users do not use tab-indentation when editing or writing code. This can be established on a UNIX system by setting the following conditions in your '.vimrc' file.

To use have tabs automatically converted into spaces:

```
set expandtab
```

To set the width of the tabs which are automatically converted into spaces:

```
set shiftwidth=4
set tabstop=4
```

III. Snakemake module organization

All Snakemake modules are to be stored within the modules sub-directory of the Snakemake system. 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 four files. The directory structure is outlined in Figure 1, and below this is an explanation as to the purpose of each of the four file categories.

```
[tboyarski@login4 modules]$ tree -L 2
.
|-- [tboyarsk      2193 Jun 30 19:49]  README.md
|-- [tboyarsk       512 Jul  5 18:17]  TEMPLATE
|   |-- [tboyarsk      2278 Jul 19 18:36]  README.md
|   |-- [tboyarsk       512 Jul  5 18:17]  RegTestReadMeTemplate
|   |-- [tboyarsk     12602 Jun 16 13:09]  SnakeTricks.md
|   |-- [tboyarsk       512 Apr 21 16:09]  tPile
|   |-- [tboyarsk      6263 Jul 10 20:40]  temp.py
|   |-- [tboyarsk      1642 Jul 21 12:02]  temp_masterINCLUDE
|   `-- [tboyarsk      6357 Jul 21 15:43]  temp_subINCLUDE
|-- [tboyarsk       512 Aug  2 17:27]  bamGen
|   |-- [tboyarsk      5036 Jun 30 17:56]  README.md
|   |-- [tboyarsk       512 Jul 10 10:26]  RegTestReadMe
|   |-- [tboyarsk      7649 Jul 13  8:55]  bamALIGN_bwa
|   |-- [tboyarsk      5718 Jul 12 18:34]  bamALIGN_star
|   |-- [tboyarsk     10104 Jul 13  9:37]  bamGen.py
|   |-- [tboyarsk      2437 Aug  2 17:27]  bamGen_INCLUDE
|   `-- [tboyarsk      2442 Jul 12 19:38]  sam2BAM
|-- [tboyarsk     65536 Jul  7 17:01]  bamMetrics
|   |-- [tboyarsk      2565 Jul  7 16:53]  README.md
|   |-- [tboyarsk     65536 Jul  7 20:57]  RegTestREADME
|   |-- [tboyarsk     12660 Jul 12 12:07]  bamMetrics.py
|   |-- [tboyarsk      1800 Jun  9 18:01]  bamMetrics_INCLUDE
|   |-- [tboyarsk      3225 Jul  7 16:44]  collectGCBias
|   |-- [tboyarsk      7331 Jul 10 20:32]  collectMERGE_ADAPTOR
|   |-- [tboyarsk      3959 Jul  7 16:48]  collectMultMetrics
|   |-- [tboyarsk      4552 Jul 12 12:24]  collectRNASeq
|   |-- [tboyarsk      2953 Jul  7 16:51]  collectWGS
|   |-- [tboyarsk      2721 Jul  7 16:52]  flagStats
|   `-- [tboyarsk      2624 Jul  7 16:53]  readLen
```

Figure 1. Tree structure of the beginning portion of 'modules/, as of August 8th, 2017. Exemplifies that each module contains the four core files. Please note that 'TEMPLATE/' contains the 'tPile/' directory for the module vignette..

Module Core files:

- **README.md** – A read-me file written in markdown format.
- **moduleN.py** – A python script which accepts the names of the YAML file, JSON file, and Snakefile. It will act as a script to populate the YAML file, JSON file, and Snakefile.
- **moduleN_INCLUDE** – The Snakemake file to be included into the pipeline. It contains no rules; rather, it contains submodule 'include:' statements and a significant description for each submodule contained. It may also contain control flow in the form of python conditionals or Snakemake "ruleorder".
- **subModuleA** – Isolation of a single rule, with the potential to possess supporting functions for the Snakemake input directive or for python processing elsewhere during execution.

IV. Workspace organization

The code was developed in such a manner that the working directory may be located anywhere. Before we discuss how to create workspace with the assistance of py_startHERE, py_buildFILES, and buildPipe.py, we will first establish what a workspace consists of.

Aside from the input source files provided by the user, all files and folders within the workspace will be created by python scripts or by Snakemake. An actual workspace generated using buildPipe.py is shown in Figure 2. The standardized starting structure of a hypothetical workspace directory is outlined in Figure 3.

****NOTE**** *Users are expected to provide their own input files, typically in the format of '.bam' files.*

```
|-- [tboyarsk       160 Aug  8 10:39]  README.md
|-- [tboyarsk      3526 Aug  8 10:40]  Snakefile
|-- [tboyarsk      3046 Aug  8 10:40]  buildPipe.py
|-- [tboyarsk       142 Aug  8 10:39]  clean.sh
|-- [tboyarsk       512 Aug  8 10:40]  input
|   |-- [tboyarsk      2306 Aug  8 10:40]  config.json
|   |-- [tboyarsk      5776 Aug  8 10:40]  config.yaml
|   |-- [tboyarsk       247 Aug  8 10:39]  help.txt
|   |-- [tboyarsk       201 Aug  8 10:39]  inputPartList.txt
|   |-- [tboyarsk       512 Aug  8 10:39]  rawBam
|   |   |-- [tboyarsk       512 Aug  8 10:39]  Parts
|   |   |   |-- [tboyarsk        81 Aug  8 10:39]  Part1-Pfeiffer2.bam -> /home/lchong/share/projects/lchong/DLBCL_cell_lines/WGSS/gsc_bam/MD903_subset.bam
|   |   |   |-- [tboyarsk        81 Aug  8 10:39]  Part1-Pfeiffer3.bam -> /home/lchong/share/projects/lchong/DLBCL_cell_lines/WGSS/gsc_bam/MD903_subset.bam
|   |   |   |-- [tboyarsk        81 Aug  8 10:39]  Part2-Pfeiffer2.bam -> /home/lchong/share/projects/lchong/DLBCL_cell_lines/WGSS/gsc_bam/MD903_subset.bam
|   |   |   |-- [tboyarsk        81 Aug  8 10:39]  Part2-Pfeiffer3.bam -> /home/lchong/share/projects/lchong/DLBCL_cell_lines/WGSS/gsc_bam/MD903_subset.bam
|   |   |   |-- [tboyarsk        81 Aug  8 10:39]  Part3-Pfeiffer2.bam -> /home/lchong/share/projects/lchong/DLBCL_cell_lines/WGSS/gsc_bam/MD903_subset.bam
|   |   |   |-- [tboyarsk        81 Aug  8 10:39]  Part3-Pfeiffer3.bam -> /home/lchong/share/projects/lchong/DLBCL_cell_lines/WGSS/gsc_bam/MD903_subset.bam
|   |   |   `-- [tboyarsk        81 Aug  8 10:39]  Part4-Pfeiffer2.bam -> /home/lchong/share/projects/lchong/DLBCL_cell_lines/WGSS/gsc_bam/MD903_subset.bam
|   |   |-- [tboyarsk        81 Aug  8 10:39]  Pfeiffer2.bam -> /home/lchong/share/projects/lchong/DLBCL_cell_lines/WGSS/gsc_bam/MD903_subset.bam
|   |   `-- [tboyarsk        81 Aug  8 10:39]  Pfeiffer3.bam -> /home/lchong/share/projects/lchong/DLBCL_cell_lines/WGSS/gsc_bam/MD903_subset.bam
|   |-- [tboyarsk        33 Aug  8 10:39]  sampleFILEpair.txt
|   `-- [tboyarsk        20 Aug  8 10:39]  sampleFILEsingle.txt
`-- [tboyarsk       512 Aug  8 10:40]  log
    `-- [tboyarsk     65536 Aug  8 10:40]  bamMetrics
        |-- [tboyarsk       512 Aug  8 10:40]  collectAlignmentSummaryMERGE
        |-- [tboyarsk       512 Aug  8 10:40]  collectGCBias
        |-- [tboyarsk       512 Aug  8 10:40]  collectInsertSizeMerge
        |-- [tboyarsk       512 Aug  8 10:40]  collectMultMetrics
        |-- [tboyarsk       512 Aug  8 10:40]  collectRNASeq
        |-- [tboyarsk       512 Aug  8 10:40]  collectRNASeqMERGE
        |-- [tboyarsk       512 Aug  8 10:40]  collectWGS
        |-- [tboyarsk       512 Aug  8 10:40]  collectWGSMERGE
        |-- [tboyarsk       512 Aug  8 10:40]  flagStats
        `-- [tboyarsk       512 Aug  8 10:40]  readLen
```

Figure 2. Starting tree structure of workspace immediately after its creation using py_startHERE. The variety of linkages is to demonstrate what is acceptable and encouraged as to promote the sharing of files and stringent use of disk space. The sym-linked files in 'rawBam/' and 'Parts/' are provided as example files to allow the directed acyclic graph to be generated easily from. Users are expected to replace the contents of 'rawBam/' with input files specific to their own project.

Workspace Core files:

- **README.md** – A read-me file written in markdown format.
- **Snakefile** – Snakemake driver file; responsible for executing the pipeline.
- **buildPipe.py** – Convenience script to build the workspace directory.
- **clean.sh** – Convenience script to clean up the workspace directory. See Appendix 1.
- **input/** – Location for all files which exist prior to the start of pipeline execution.
- **input/rawBam** – Location for all your input '.bam' files.
- **input/config.yaml** – Pipeline specific configuration variables.
- **input/config.json** – Cluster specific configuration variables.
- **Input/inputPartList.txt** – List of the composition of multi-part samples to be merged (optional).
- **Input/sampleFILE<type>.txt** – List of either single (single) or paired (pair), tumor-normal samples.
- **log/** – Location for all log files created during pipeline execution.
- **log/module0** – Location for log directory of sub-modules contained within the module.
- **log/module0/sub-module** – Location for all log files created by sub-module during pipeline execution.

```
myWorkSpace/
      README.md
      Snakefile
      buildPipe.py
      clean.sh
      /input
            /config.yaml
            /config.json
            /inputPartList.txt
            /sampleFILEsingle.txt
            /sampleFILEpair.txt
            /rawBam
                  /Sample1.bam
                  /SampleN.bam
                  /Parts
                        /Part1.bam
                        /PartN.bam
      /log
      /shellCalls.txt
      /runStats.txt
            /module1
                  /sub-module1
                  /…
                  /sub-moduleN
            /moduleN
                  /sub-module1
                  /…
                  /sub-moduleN
      /output
            /module1
                  /sub-module1
                  /…
                  /sub-moduleN
            /moduleN
                  /sub-module1
                  /…
                  /sub-moduleN
```

Figure 3. Core structure of a hypothetical workspace after running a Snakemake pipeline. Module names typically do not contain numbers, but for the purposes of the vignette are numbered to exemplify any number of different modules or sub-modules.

V. Conda activation

Snakemake pipelines have been developed to be run under the environmental management of an Anaconda environment. The conda environment can be started on either a cluster or a local machine. If possible, please use the existing Anaconda environment available on the Genesis cluster.

Anaconda environments are specific to the version of Anaconda being run. When loaded, these environments can take up to a few minutes to setup. Users can check what environments are available by using the call:

```
$ conda info -e
```

Specific Anaconda environments are required depending on which operating system you are on. For this vignette, we will be on the Genesis server, running CentOS5, we will be using the following Anaconda environment, which can be activated directly or indirectly, as follows:

```
$ source ~/share/usr/anaconda/4.3.0/bin/activate CentOS5-Compatible

OR … (Depending how your path variables are setup)

$ source activate CentOS5-Compatible
```

There is no explicit reporting about the activation of the conda environment. Instead, users are notified to their active conda environment by a prefix to their bash prompt, as highlighted in the yellow box in Figure 4.

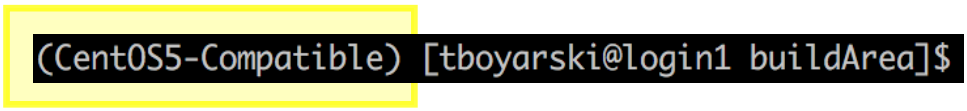

Figure 4. Text based notification of the Conda environment currently active for a single session within a single terminal window. The yellow box indicates the name of the active conda environment.

VI. Workspace creation

To ensure consistency across workspaces which utilize this Snakemake system, the module "py_startHERE" was created to automate workspace creation. "py_startHERE" is considered to be the starting point for any user, beginner or expert, intending to use the system. A user-friendly error response script has been provided to inform users as to the names of the modules which are currently activate, and subsequently, those names which can be provided as inputs to the "py_startHERE" python script 'startHERE.py'. This python script can use used without providing the list of module names; however, this will result in users having to later manually accomplish exactly what buildPipe.py would have done. The manual addition of module to buildPipe.py is not covered in the vignette.

1) **Navigate to "py_startHERE" module to interact with the 'startHERE.py'**
The python script within the module is setup to take a number of arguments to generate a workspace. By providing no formal arguments, the standard interaction instructions for the script are displayed. They provide examples of how to interact with the file, as well, they advise the user of the Snakemake system's current module composition.

```
$ python startHERE.py
```

```
(CentOS5-Compatible) [tboyarski@login4 py_startHERE]$ python startHERE.py
**ERROR** Invalid number of arguments.
        Please provide absolute path for desintation of the directory, and also the new directory's name.
                E.g. python startHERE.py /home/tboyarski/share/projects/tboyarski NewAwesomeProject


        If you're feeling really confident, you can also just add the module names to the end of the call.
                E.g. python startHERE.py /home/tboyarski/share/projects/tboyarski PMBCL1 processBam mpileup varScan annotateVcf utils bamMetrics


        Here are the modules currently listed. They may not all work, this just queries the directory and edits a bit.

                        vcfGenUtil_varScan              fastqUtil               vcfUtil         bamMetrics              py_buildFile
                        bamGen          fastqGen                bamUtil         vcfAnnotate             mpileupGen
                        genericUtil             py_startHERE            starFusion
```

Figure 5. Example of the user-prompt provide to the user intending to use this Snakemake system to generate a pipeline.

2) **Create a workspace with fastqGen, fastqUtil, bamGen, bamUtil, bamMetrics**
Users are to provide the python script 'startHERE.py' an absolute location, the name of the new workspace, and the modules which are to be included in the workspace. The order in which the modules are listed will determine the order in which the module descriptions and variables are written to the file configuration files. Absent of identically named variables or modules, as this would create over-writing issues, the ordering of the modules within any of the files does not matter and has no consequence. This script will create and name a new workspace, "myWorkSpace". Inside it will populate the directory with the required files, one of which being 'buildPipe.py'.

To avoid accidentally over-writing a valid workspace, the system is setup to only create new directories, this is exemplified in Figure 6. When an erroneous module name is provided to the python script, the script will print an error message, and it will still write the incorrectly spelled module call into 'buildPipe.py'. Figure 7 demonstrates that by providing the correct arguments, the script is able to create the workspace, providing user-feedback while doing so. Figure 8 demonstrates the error message when providing incorrectly spelled modules.

For the purposes of this vignette, please use the following shell call from inside the module "py_startHERE":

```
$ python startHERE.py ~/share/projects/tboyatrski myWorkspace fastqGen
    fastqUtil bamGen bamUtil bamMetrics
```

```
(CentOS5-Compatible) [tboyarski@login3 py_startHERE]$ python startHERE.py ~/share/projects/tboyarski/ myWorkSpace fastqfastqGen fastqUtil bamGen bamUtil bamMetrics
Creating workspace directory myWorkSpace at location:

       /home/tboyarski/share/projects/tboyarski/

Traceback (most recent call last):
  File "startHERE.py", line 103, in <module>
    mkdir(buildDirectory)
FileExistsError: [Errno 17] File exists: '/home/tboyarski/share/projects/tboyarski//myWorkSpace'
```

Figure 6. Screenshot of the error message provided when trying to use 'startHERE.py' to create a workspace directory inside of a directory which already contains a directory of that name. This demonstrates that the 'startHERE.py' does not over-write directories.

```
(CentOS5-Compatible) [tboyarski@login3 py_startHERE]$ python startHERE.py ~/share/projects/tboyarski/ myWorkSpace fastqGen fastqUtil bamGen bamUtil bamMetrics
Creating workspace directory myWorkSpace at location:

       /home/tboyarski/share/projects/tboyarski/

Module found: fastqGen
Module found: fastqUtil
Module found: bamGen
Module found: bamUtil
Module found: bamMetrics
Build complete!
```

Figure 7. Example of the startHERE.py generation of a workspace. All modules listed were reported as correctly found, and the build is confirmed as complete for improved user-experience.

```
(CentOS5-Compatible) [tboyarski@login4 py_startHERE]$ python startHERE.py /home/tboyarski/share/projects/tboyarski myWorkSpace2
bamMetric
Creating workspace directory myWorkSpace2 at location:

       /home/tboyarski/share/projects/tboyarski

**Warning** Module not found: bamMetric
**Warning** Still added string, please check buildPipe.py to validate path to module: bamMetric
Build complete!
```

Figure 8. Example of the error message provided by startHERE.py when it is unable to locate one or more of the modules which were requested to be included in the building of the new workspace at the location provided.

****WARNING**** *If a module is incorrectly spelled, users may either delete and re-create the workspace with the correct name, or, users must navigate to the file "myWorkSpace/buildPipe.py", and they must ensure all the modules listed within the file are correct.*

3) **Change directories to enter the new workspace, open 'buildPipe.py', and correct misspelled modules**
An incorrectly spelled module name is still written two times in the file buildPipe.py. Both writes are to the same line within the file; as such, it is very easy to correct misspelled module names. Later on, if buildPipe.py is run with an incorrectly spelled module name, an error message is provided, as displayed in Figure 9.

```
(CentOS5-Compatible) [tboyarski@login4 myWorkSpace2]$ python buildPipe.py
buildFile.py    Using: single column format.
buildFile.py    Sampling: input/sampleFILEsingle.txt
buildFile.py    Creating: input/config.yaml
                Creating: input/config.json
                Creating: Snakefile
buildHeader.py  Creating: log/
        ...Calling genSupportingFileList
        ...Calling genChrList
python: can't open file '/extscratch/clc/projects/tboyarski/gitRepo-LCR-BCCRC/Snakemake/modules/bamMetric/bamMetric.py': [Errno 2] No such file or directory
```

Figure 9. Error message provided by buildPipe.py when internally there is a reference call to a module which does not exist. The error message displayed here indicates the exact file, 'bamMetric.py', which is erroneously listed.

The correct generation of buildPipe.py is considered part of the workspace creation, as such, we will briefly discuss correcting module spelling errors inside a generated 'buildPipe.py' file. We will reframe from fully discussing the configuration of 'buildPipe.py', as that is performed in the section "Workspace creation".

The buildPipe.py file can be easily edited in "VIM", screenshots of the code before and after the changes are listed in Figure 10. In this scenario, the user intended to request the module "bamMetrics", but instead requested "bamMetric". We will change the name "bamMetric" to "bamMetrics" within buildPipe.py.

A

```
54 #--------------------------------------------------------------------------------------------------------------------------------------------
55 # PYTHON SCRIPT #
56 #--------------------
57 # Module 0:  Call to generate .YAML and Snakefiles with header information.
58 call('python ' + snakeDIR + '/modules/py_buildFile/buildFile.py ' + TYPE + ' ' + SAMPLE + ' ' + REFFILE + ' ' +  YAMLFILE + ' ' + CLUSTERFILE + ' ' + SNAKEFILE, shell=True)
59
60 # Module 1:
61 call('python ' + snakeDIR + '/modules/bamMetric/bamMetric.py ' + YAMLFILE + ' ' + CLUSTERFILE + ' ' +  SNAKEFILE, shell=True)
62 #-----------------------------------------------------------------------------------------------------------------------------------------
```

B

```
54 #--------------------------------------------------------------------------------------------------------------------------------------------
55 # PYTHON SCRIPT #
56 #--------------------
57 # Module 0:  Call to generate .YAML and Snakefiles with header information.
58 call('python ' + snakeDIR + '/modules/py_buildFile/buildFile.py ' + TYPE + ' ' + SAMPLE + ' ' + REFFILE + ' ' +  YAMLFILE + ' ' + CLUSTERFILE + ' ' + SNAKEFILE, shell=True)
59
60 # Module 1:
61 call('python ' + snakeDIR + '/modules/bamMetrics/bamMetrics.py ' + YAMLFILE + ' ' + CLUSTERFILE + ' ' +  SNAKEFILE, shell=True)
62 #-----------------------------------------------------------------------------------------------------------------------------------------
```

Figure 10. A it can be seen that the module call on line 61 is to 'bamMetric/bamMetric.py'. We know the correct module name is in fact pluralized, and it is 'bamMetrics'. By changing 'bamMetric' to 'bamMetrics', in both positions, the module call will now successfully execute.

VII. Workspace configuration

The majority of the creation of the pipeline is automated into a script called buildPipe.py. The script buildPipe.py was built in the previous section. It is a result of interacting with "py_startHERE" and running the internal python script 'startHERE.py'. Automating the pipeline creation became increasingly important as the number of configurable variables needing to be included grew. A pipeline of any meaningful purpose will have at least a hundred different mandatory configurable variables. It became apparent that users could not be reasonably expected to identify and list all these variables. It was determined that automating this setup was of great convenience and value to the user. The following steps build from creation of the workspace. They explain how to create and configure the pipeline which will be contained within the workspace.

1) **Configure the variable "TYPE" in buildPipe.py**

This determination is made depending if the pipeline will be running unpaired samples (single), or paired tumor-normal (pair) samples. Because the pairing of tumor-normal samples is very strict, the processing of tumor-normal sample names is done differently. The variable "TYPE" may take one of two arguments, "single" or "pair". The correct assignment of this variable can be seen in Figure 11. It will also be used to determine which sample file will be read from.

```
 9 #-------------------------------------------------------
10 # LOCAL VARIABLES #
11 #------------------
12 # Formatting of the sample file.
13 TYPE="single"
14
15 # Creating a relative variable for the sample file.
16 SAMPLE="input/sampleFILE" + TYPE + ".txt"
```

Figure 11. Configuration of the 'TYPE' variable inside of the 'buildPipe.py' python script.

2) **Configure the variable "REFFILE" in buildPipe.py**

Previously the genome was set in the configuration files; now, the entire process is decided at the time of pipeline creation as to prevent the accidental mid-execution switching of the reference genome. In doing so, we also alleviate the responsibility of the user determining the corresponding chromosomal list formatted specific to the reference file. This is scripted and the chromosomal list is now parsed and added to the 'YAML' configuration file automatically. For ease of use, multiple reference genomes have been provided inside 'buildPipe.py'. Please select the genomes which corresponds to the operating system and aligner to be used. An example of a successfully configured "REFFILE", line 33, can be seen in Figure 12.

```
19 # ********************************************************************************
20 # ************************************ WARNING ************************************
21 # ********************************************************************************
22 #
23 # Store the absolute path of the reference genome file.
24 #
25 #----gphost01 ONLY--------
26 #REFFILE="/genesis/extscratch/clc/references/genomes/hg19.fa"
27
28
29 #----Genesis ONLY--------
30 # Generic GRCh37 Genome Reference:
31 #REFFILE="/reference/genomes/9606/hg19a/genome/GRCh37-lite.fa"
32 # BWA Alignement GRCh37 Genome Reference:
33 REFFILE="/genesis/extscratch/clc/references/genomes/gsc/bwa-0.7.5a/GRCh37-lite.fa"
34
35 # ********************************************************************************
36 # ************************************ WARNING ************************************
37 # ********************************************************************************
```

Figure 12. Configuration of the 'REFFILE' variable inside of the 'buildPipe.py' python script.

3) **Configure the variable "snakeDIR" in buildPipe.py**

Typically, users should not have to configure this variable. It is provided such that the referencing of all internal modules can be provided as an absolute reference. This is done by combing the absolute path to the Snakemake directory, with the internal module path. An example of a successfully configured "snakeDIR", line 49, can be seen in Figure 13.

```
48 # Absolute directory for the Snakemake code repository.
49 snakeDIR="/extscratch/clc/projects/tboyarski/gitRepo-LCR-BCCRC/Snakemake"
50 #-------------------------------------------------------------------------------
```

Figure 13. Configuration of the "snakeDIR" variable inside of the 'buildPipe.py' python script. This variable allows for the relocation of the entire repository, while still ensuring absolute paths are used to reference internal modules.

4) **Review 'buildPipe.py'**

The final result of buildPipe.py should be similar to Figure 14.

+ buildPipe.py

```
#-------------------------------------------------------------------------------
# Author: tboyarski
# Date: 2017-08-11 @ 12-54
#-------------------------------------------------------------------------------
# PYTHON PACKAGES #
#-------------------
# Required library to make shell calls.
from subprocess import call
#-------------------------------------------------------------------------------
# LOCAL VARIABLES #
#-------------------
# Formatting of the sample file.
TYPE="single"

# Creating a relative variable for the sample file.
SAMPLE="input/sampleFILE" + TYPE + ".txt"


# ********************************************************************************
# ************************************** WARNING **************************************
# ********************************************************************************
#
# Store the absolute path of the reference genome file.
#
#----gphost01 ONLY--------
REFFILE="/genesis/extscratch/clc/references/genomes/hg19.fa"


#----Genesis ONLY--------
# Generic GRCh37 Genome Reference:
#REFFILE="/reference/genomes/9606/hg19a/genome/GRCh37-lite.fa"
# BWA Alignement GRCh37 Genome Reference:
REFFILE="/genesis/extscratch/clc/references/genomes/gsc/bwa-0.7.5a/GRCh37-lite.fa"

# ********************************************************************************
# ************************************** WARNING **************************************
# ********************************************************************************

# Path and name of the YAML Snakemake pipeline configuration file.
YAMLFILE="input/config.yaml"

# Path and name of the JSON Snakemake pipeline cluster configuration file.
CLUSTERFILE="input/config.json"

# Path and name of the Snakefile for respective pipeline project.
SNAKEFILE="Snakefile"

# Absolute directory for the Snakemake code repository.
snakeDIR="/extscratch/clc/projects/tboyarski/gitRepo-LCR-BCCRC/Snakemake"
#-------------------------------------------------------------------------------



#-------------------------------------------------------------------------------
# PYTHON SCRIPT #
#-------------------
# Module 0:  Call to generate .YAML and Snakefiles with header information.
call('python ' + snakeDIR + '/modules/py_buildFile/buildFile.py ' + TYPE + ' ' + SAMPLE + ' ' + REFFILE + ' ' +  YAMLFILE + ' ' + CLUSTERFILE + ' ' + SNAKEFILE, shell=True)

# Module 1:
call('python ' + snakeDIR + '/modules/fastqGen/fastqGen.py ' + YAMLFILE + ' ' + CLUSTERFILE + ' ' +  SNAKEFILE, shell=True)

# Module 2:
call('python ' + snakeDIR + '/modules/fastqUtil/fastqUtil.py ' + YAMLFILE + ' ' + CLUSTERFILE + ' ' +  SNAKEFILE, shell=True)

# Module 3:
call('python ' + snakeDIR + '/modules/bamGen/bamGen.py ' + YAMLFILE + ' ' + CLUSTERFILE + ' ' +  SNAKEFILE, shell=True)

# Module 4:
call('python ' + snakeDIR + '/modules/bamUtil/bamUtil.py ' + YAMLFILE + ' ' + CLUSTERFILE + ' ' +  SNAKEFILE, shell=True)

# Module 5:
call('python ' + snakeDIR + '/modules/bamMetrics/bamMetrics.py ' + YAMLFILE + ' ' + CLUSTERFILE + ' ' +  SNAKEFILE, shell=True)
#-------------------------------------------------------------------------------
```

Figure 14. A finalized version of an example 'buildPipe.py' file. The file reflects what the user is expected to have by this section in the vignette.

VIII. Building and configuring the Snakefile

At this stage, the user now has a finalized buildPipe.py file. The following steps will walk the user through using 'buildPipe.py' and it will explain the files that are created as a result. This vignette will provide insight as to interpreting the output generated by these scripts. Execution of 'buildPipe.py' results in the generation and population of a Snakefile and the required pipeline specific '.YAML' and '.JSON' configuration files.

5) **Execute the python script 'buildPipe.py'**

The script will generate the configuration files and driver script for the workspace. 'buildPipe.py' will interact with each module listed. It will generate the log directories of the respective modules which were included in the build file you created. Everything that is created by this python script is mentioned in output statements; the statements are not recorded. The statement's purpose is to allow the user to validate what they told the script to do. It clearly reports the file from which the call was made, the action being performed, and the actionable target, as demonstrated in Figure 15. An example of the output which should be provided when running 'buildPipe.py' and following this vignette is provided in Figure 16.

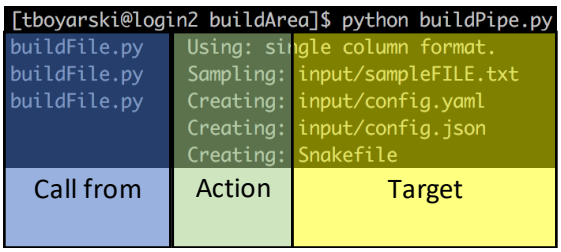

Figure 15. Example of the output provided when running the buildPipe.py script.

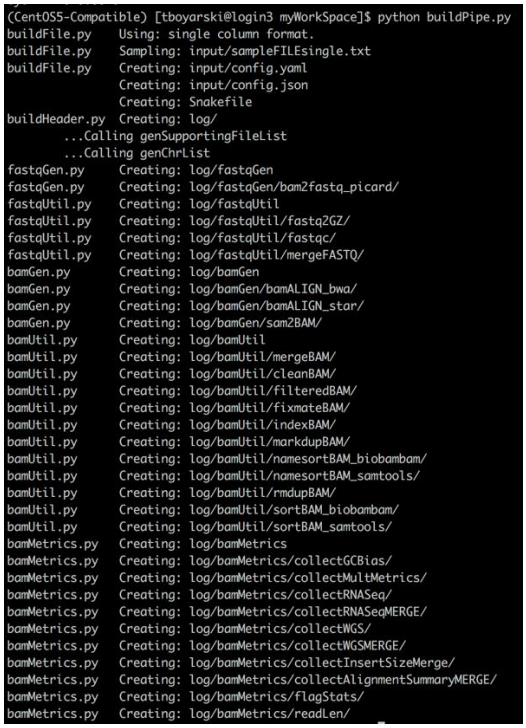

Figure 16. Example of the output provided when running buildPipe.py to assembly a workspace containing the 'bamMetrics' module. Reporting of the sub-module specific log directories can be seen following the setup of the input directory, configuration files, and log directory.

6) **Edit the Snakefile input directive**

When the Snakefile is generated, a dummy-value is provided in the input-field-line of the input-directive. The user must determine to which point the Snakefile pipeline is to progress. They will replace the code with a desired endpoint, which is the output of one of its sub-modules. This results in the Snakefile trying to generate this file, and all its dependencies. Nearly all intermediate files are removed after use. If any are designed, they should be listed in the rule 'all', in addition to the desired output file. The ordering of desired outputs, and potential intermediates does not matter, rather, the only restriction is that each file request be separated by a comma. An example of replacing this value is provided in image A of Figure 17. Its correct replacement is demonstrated in image B of Figure 17. In this case, we desired the output generated by the "bamMetrics" sub-module GCBias (Line 113) and have not requested any intermediate files.

****WARNING**** *Users must remove the '#' and indent 8 spaces; tabs are* ***not*** *allowed.*

Figure 17. Two VIM screenshots taken of the automatically generated Snakefile. This top image, A, is prior to the replacement of the dummy-value, the second image, B, as after the replacement of the dummy-value. Line 13 content in image A may differ from what is seen.

7) **A finalized 'Snakefile' should look similar to Figure 19**

A complete Snakefile will often be 200+ lines. This is to provide enough documentation such that the modules themselves do not need to be interacted with. The user can just read the information each module has printed into the 'Snakefile' and they can choose from within these options.

****NOTE**** *Figure 19 only shows the first 70 lines of a 125 line Snakefile.*

```
1  #-------------------------------------------------------------------------------------------------------------------------------------------------
2  # Author: tboyarski
3  # Date: 2017-08-11.14-56-49
4  # Call using: time snakemake --jobs 100 --cluster-config input/config.json --jobname "{cluster.jobName}.{jobid}" --drmaa "{cluster.clusterSpec} --runStats log/
   runStats.txt"
5  #-------------------------------------------------------------------------------------------------------------------------------------------------
6  # PYTHON PACKAGES #
7  #-----------------
8  # Used by some modules which require paired tumor-normal samples
9  from pandas import read_table
10 # Used by some modules which require paired tumor-normal samples
11 from io import StringIO
12 #-------------------------------------------------------------------------------------------------------------------------------------------------
13 # SNAKEMAKE CONFIGURATION #
14 #-------------------------
15 # Global config:
16 configfile: "input/config.yaml"
17 #-------------------------------------------------------------------------------------------------------------------------------------------------
18
19
20 #-------------------------------------------------------------------------------------------------------------------------------------------------
21 # SNAKEMAKE RULE #
22 #-----------------
23 # Global rule to pull all output files:
24 rule all:
25     input:
26         # Single: Normal Runs
27         expand("{outputDIR}/{metricsDIR}/{samples}.collectGCBias.txt", outputDIR=config["outputDIR"], metricsDIR=config["metricsDIR"], samples=config["sample"]),
28 #-------------------------------------------------------------------------------------------------------------------------------------------------
29
30
31 #-------------------------------------------------------------------------------------------------------------------------------------------------
32 #**** fastqGen ****:
33 #  Included:
34 #      bam2fastq:              Fastq generation from a '.bam' file.
35 #  Files:
36 #      input:      X.bam
37 #      output:     X.1.fastq
38 #                  X.2.fastq
39 include: "/extscratch/clc/projects/tboyarski/gitRepo-LCR-BCCRC/Snakemake/modules/fastqGen/fastqGen_INCLUDE"
40 #  Required: NONE
41 #  Call via:
42 #bam2fastq_picard:     expand("{outputDIR}/{fastqDIR}/{samples}.{readDirection}.fastq", outputDIR=config["outputDIR"], fastqDIR=config["fastqDIR"],
   samples=config["sample"], readDirection=["1", "2"])
43 #-------------------------------------------------------------------------------------------------------------------------------------------------
44
45
46 #-------------------------------------------------------------------------------------------------------------------------------------------------
47 #***** fastqUtil *****
48 #  Included:
49 #      fastq2GZ:                   Run perl QC script and convert a '.bam' file into a '.fastq.zip'
50 #      fastqc:                     Compress a '.fastq' file into a '.fastq.gz' file
51 #      mergeFASTQ_ADAPTOR:         Produce a single '.fastq' file from a list of '.fastq' files.
52 include: "/extscratch/clc/projects/tboyarski/gitRepo-LCR-BCCRC/Snakemake/modules/fastqUtil/fastqUtil_INCLUDE"
53 #  Call via:
54 #fastq2GZ:            expand("{outputDIR}/{fastqDIR}/{samples}.{readDirection}.fastq.gz", outputDIR=config["outputDIR"], fastqDIR=config["fastqDIR"],
   samples=config["sample"], readDirection=["1", "2"])
55 #fastqc:              expand("{outputDIR}/{fastqcDIR}/{samples}_fastqc.zip", outputDIR=config["outputDIR"], fastqcDIR=config["fastqcDIR"],
   samples=config["sample"])
56 #mergeFASTQ:          expand("{outputDIR}/{fastqDIR}/{samples}.{readDirection}.fastq{compressionSuffix}", outputDIR=config["outputDIR"],
   fastqDIR=config["fastqDIR"], samples=config["sample"], readDirection=["1","2"], compressionSuffix=[".gz"])
57 #-------------------------------------------------------------------------------------------------------------------------------------------------
58
59
60 #-------------------------------------------------------------------------------------------------------------------------------------------------
61 #**** bamGen ****:
62 #  Included:
63 #      bamALIGN_bwa:           Generate a '.bam' file from paired '.fastq' files, via BWA.
64 #      bamALIGN_star:          Generate a '.bam' file from paired '.fastq' files, via STAR.
65 #      sam2BAM:                Generate a '.bam' file from a '.sam' file, via samtools.
66 include: "/extscratch/clc/projects/tboyarski/gitRepo-LCR-BCCRC/Snakemake/modules/bamGen/bamGen_INCLUDE"
67 #  Required: NONE
```

Figure 18. VIM screenshot on Genesis of a production ready Snakefile (A.K.A. Snakemake pipeline file). The module call from Line 113 is used on Line 27 to dictate that the pipeline should produce GCBias metadata files.

IX. Pipeline configuration

We will now edit the recently generated configuration files. Aside from the Snakefile, all configuration files are location in the 'input/' workspace sub-directory.

1) **Add raw '.bam' files into 'input/rawBAM/'**

The user may populate the input directories using symlinks or by moving over the files to this directory. For the purpose of this vignette, the sample files 'Pfeiffer2.bam' and Pfeiffer3.bam' will be sufficient. To avoid the resource intensive task of moving files across networks or disks, it is recommended the user utilize use symbolic links for raw inputs whenever possible. The resulting setup may appear similar to Figure 19.

****NOTE**** *The 'Parts/' directory located within 'rawBam/' is to exemplify the organization of '.bam' files if they are provided in segments. '.bam' files are likely to be considered segmental if they are replicates or sub-sections an larger sample. Regardless if they are segmental or complete, '.bam' files are to be located in the 'rawBam/' directory. The 'Parts/' directory is just an easy way to contain and subsequently delete template material which will be rarely used. Currently, segmental input '.bam' files are considered very uncommon.*

```
(CentOS5-Compatible) [tboyarski@login3 myWorkSpace]$ tree
.
|-- [tboyarsk       160 Aug 11 12:54]  README.md
|-- [tboyarsk     10629 Aug 11 14:58]  Snakefile
|-- [tboyarsk      3590 Aug 11 14:14]  buildPipe.py
|-- [tboyarsk       142 Aug 11 12:54]  clean.sh
|-- [tboyarsk       512 Aug 11 14:56]  input
|   |-- [tboyarsk      5998 Aug 11 14:56]  config.json
|   |-- [tboyarsk     13261 Aug 11 14:56]  config.yaml
|   |-- [tboyarsk       247 Aug 11 12:54]  help.txt
|   |-- [tboyarsk       201 Aug 11 12:54]  inputPartList.txt
|   |-- [tboyarsk       512 Aug 11 12:54]  rawBam
|   |   |-- [tboyarsk       512 Aug 11 12:54]  Parts
|   |   |   |-- [tboyarsk        81 Aug 11 12:54]  Part1-Pfeiffer2.bam -> /home/lchong/share/projects/lchong/DLBCL_cell_lines/WGSS/gsc_bam/MD903_subset.bam
|   |   |   |-- [tboyarsk        81 Aug 11 12:54]  Part1-Pfeiffer3.bam -> /home/lchong/share/projects/lchong/DLBCL_cell_lines/WGSS/gsc_bam/MD903_subset.bam
|   |   |   |-- [tboyarsk        81 Aug 11 12:54]  Part2-Pfeiffer2.bam -> /home/lchong/share/projects/lchong/DLBCL_cell_lines/WGSS/gsc_bam/MD903_subset.bam
|   |   |   |-- [tboyarsk        81 Aug 11 12:54]  Part2-Pfeiffer3.bam -> /home/lchong/share/projects/lchong/DLBCL_cell_lines/WGSS/gsc_bam/MD903_subset.bam
|   |   |   |-- [tboyarsk        81 Aug 11 12:54]  Part3-Pfeiffer2.bam -> /home/lchong/share/projects/lchong/DLBCL_cell_lines/WGSS/gsc_bam/MD903_subset.bam
|   |   |   |-- [tboyarsk        81 Aug 11 12:54]  Part3-Pfeiffer3.bam -> /home/lchong/share/projects/lchong/DLBCL_cell_lines/WGSS/gsc_bam/MD903_subset.bam
|   |   |   `-- [tboyarsk        81 Aug 11 12:54]  Part4-Pfeiffer2.bam -> /home/lchong/share/projects/lchong/DLBCL_cell_lines/WGSS/gsc_bam/MD903_subset.bam
|   |   |-- [tboyarsk        81 Aug 11 12:54]  Pfeiffer2.bam -> /home/lchong/share/projects/lchong/DLBCL_cell_lines/WGSS/gsc_bam/MD903_subset.bam
|   |   `-- [tboyarsk        81 Aug 11 12:54]  Pfeiffer3.bam -> /home/lchong/share/projects/lchong/DLBCL_cell_lines/WGSS/gsc_bam/MD903_subset.bam
|   |-- [tboyarsk        33 Aug 11 12:54]  sampleFILEpair.txt
|   `-- [tboyarsk        20 Aug 11 12:54]  sampleFILEsingle.txt
`-- [tboyarsk       512 Aug 11 14:56]  log
    |-- [tboyarsk       512 Aug 11 14:56]  bamGen
    |   |-- [tboyarsk       512 Aug 11 14:56]  bamALIGN_bwa
    |   |-- [tboyarsk       512 Aug 11 14:56]  bamALIGN_star
    |   `-- [tboyarsk       512 Aug 11 14:56]  sam2BAM
    |-- [tboyarsk     65536 Aug 11 14:56]  bamMetrics
    |   |-- [tboyarsk       512 Aug 11 14:56]  collectAlignmentSummaryMERGE
    |   |-- [tboyarsk       512 Aug 11 14:56]  collectGCBias
    |   |-- [tboyarsk       512 Aug 11 14:56]  collectInsertSizeMerge
    |   |-- [tboyarsk       512 Aug 11 14:56]  collectMultMetrics
    |   |-- [tboyarsk       512 Aug 11 14:56]  collectRNASeq
    |   |-- [tboyarsk       512 Aug 11 14:56]  collectRNASeqMERGE
    |   |-- [tboyarsk       512 Aug 11 14:56]  collectWGS
    |   |-- [tboyarsk       512 Aug 11 14:56]  collectWGSMERGE
    |   |-- [tboyarsk       512 Aug 11 14:56]  flagStats
    |   `-- [tboyarsk       512 Aug 11 14:56]  readLen
    |-- [tboyarsk     65536 Aug 11 14:56]  bamUtil
    |   |-- [tboyarsk       512 Aug 11 14:56]  cleanBAM
    |   |-- [tboyarsk       512 Aug 11 14:56]  filteredBAM
    |   |-- [tboyarsk       512 Aug 11 14:56]  fixmateBAM
    |   |-- [tboyarsk       512 Aug 11 14:56]  indexBAM
    |   |-- [tboyarsk       512 Aug 11 14:56]  markdupBAM
    |   |-- [tboyarsk       512 Aug 11 14:56]  mergeBAM
    |   |-- [tboyarsk       512 Aug 11 14:56]  namesortBAM_biobambam
    |   |-- [tboyarsk       512 Aug 11 14:56]  namesortBAM_samtools
    |   |-- [tboyarsk       512 Aug 11 14:56]  rmdupBAM
    |   |-- [tboyarsk       512 Aug 11 14:56]  sortBAM_biobambam
    |   `-- [tboyarsk       512 Aug 11 14:56]  sortBAM_samtools
    |-- [tboyarsk       512 Aug 11 14:56]  fastqGen
    |   `-- [tboyarsk       512 Aug 11 14:56]  bam2fastq_picard
    `-- [tboyarsk       512 Aug 11 14:56]  fastqUtil
        |-- [tboyarsk       512 Aug 11 14:56]  fastq2GZ
        |-- [tboyarsk       512 Aug 11 14:56]  fastqc
```

Figure 19. Example of the directory structure on Genesis after running a 'buildPipe.py'. A variety of mock inputs and part files are provided.

2) **Edit 'sampleFILEsingle.txt' to include your sample names.**
The default values provided inside the file should reflect the names of the provided '.bam' files. If these sample names so not match, then remove the dummy sample names provided, and by entering the correct sample names on new lines inside the file 'sampleFILEsingle.txt'. This is exemplified below:

```
Pfeiffer2
Pfeiffer3
```

We will briefly explain the formatting of 'sampleFILEpair.txt' and 'inputPartList.txt'; however, please note that neither of these files are required for this vignette. Both of these files utilize the first like as a header for the internal content. At no point are more than two items listed per line.

Tumor and normal samples are to be included at a 1:1 ratio. Each tumor sample is to be considered paired to a normal (non-tumor) sample, as seen below in 'sampleFILEpair.txt':

```
tumor normal
Pfeiffer2T Pfeiffer2N
Pfeiffer3T Pfeiffer3N
```

Segmental '.bam' files are not required to follow a uniform ratio; each '.bam' sample can be comprised of a different number of segmental '.bam' files. Below represents the required formatting for 'inputPartList.txt':

```
groupName fileName
Pfeiffer2 Part1-Pfeiffer2
Pfeiffer2 Part2-Pfeiffer2
Pfeiffer2 Part3-Pfeiffer2
Pfeiffer2 Part4-Pfeiffer2
Pfeiffer3 Part1-Pfeiffer3
Pfeiffer3 Part2-Pfeiffer3
Pfeiffer3 Part3-Pfeiffer3
Pfeiffer3 Part4-Pfeiffer3
Pfeiffer3 Part5-Pfeiffer3
```

3) **Consider editing the JSON file to reflect cluster specifications.**
The JSON file, typically, does not need to be edited. Users may edit the JSON file with the intent of increasing the resources of a specific rule, or to change the logging directory of a specific rule, otherwise, very little should be changed in this file. Whitespace does not matter. Ordering is not maintain in a JSON file. The resulting JSON should appear similar to Figure 20. In the figure, only the first 20 lines are show. The JSON structure can be seen repeating in 4-line blocks. If setup correctly, it can then be assumed that there is a 4-line block corresponding to each sub-module for all the modules included within the pipeline.

```
1
2    "mergeBAM": {
3        "jobName": "{rule}_{wildcards.sampleMB}",
4        "clusterSpec": "-V -S /bin/bash -o log/bamUtil/mergeBAM -e log/bamUtil/mergeBAM -l h_vmem=10G -pe ncpus 1"
5    },
6    "mergeFASTQ": {
7        "jobName": "{rule}_{wildcards.sampleFM}_{wildcards.readDirection}_{wildcards.compressionSuffix}",
8        "clusterSpec": "-V -S /bin/bash -o log/fastqUtil/mergeFASTQ -e log/fastqUtil/mergeFASTQ -l h_vmem=10G -pe ncpus 1"
9    },
10   "readLen": {
11       "jobName": "{rule}_{wildcards.sampleRL}",
12       "clusterSpec": "-V -S /bin/bash -o log/bamMetrics/readLen -e log/bamMetrics/readLen -l h_vmem=10G -pe ncpus 1"
13   },
14   "collectRNASeqMERGE": {
15       "jobName": "{rule}_all.samples",
16       "clusterSpec": "-V -S /bin/bash -o log/bamMetrics/collectRNASeqMERGE -e log/bamMetrics/collectRNASeqMERGE -l h_vmem=10G -pe ncpus 1"
17   },
18   "rmdupBAM": {
19       "jobName": "{rule}_{wildcards.sampleRDB}",
20       "clusterSpec": "-V -S /bin/bash -o log/bamUtil/rmdupBAM -e log/bamUtil/rmdupBAM -l h_vmem=10G -pe ncpus 1"
```

Figure 20. Example of the 'config.json' file required by the Snakemake pipeline. This file provides the arguments required when submitting jobs to the cluster. The arguments '-o' and '-e' are for output redirection. The DRMAA arguments and job name are accessed by the SGE schedule at the beginning of every job submission.

4) **Edit the YAML file, if desired.**
The YAML file does not necessarily need to be edited, it is specifically designed to work with the default files written by 'buildPipe.py'. For the purposes of this vignette, the use of the default values is acceptable. The YAML file contains a number of parameters which are either global or rule specific. It is worth checking out this file to become familiar with the optional granularity of the pipeline. Typically this involves setting the annotation names of files, toggling if a pipeline is to process the BAM files by chromosome or as just a single file, which software is to be used to sort a BAM file, and other such options. The YAML file can often contain 100+ configurable variables, as such, Figure 21 only shows a subsection of the YAML file produced. For the sake of simplicity, users may edit line 33, such that there is only 1 sample listed.

****NOTE**** *Editing of line 3 is just to reduce the number of jobs which will be created by the pipeline. By reducing the number of jobs, we can decrease the complexity of the pipeline for this vignette. Once the vignette has been completed, come back to this step, add in the sample "Pfeiffer3" which was removed, and compare the differences in job count and file output.*

i/config.yaml

```yaml
#----------------------------------------------------------------------------------------------------------------------
# Author: tboyarski
# Date: 2017-08-11.14-56-49
#----------------------------------------------------------------------------------------------------------------------


###################################
# Global Parameters
###################################
#---------------------------------------------------------------- Parameters ----------------------------------------------
shellCallFile: log/shellCalls.txt
offCluster: False
fastqKEEP: True
inputPartList: input/inputPartList.txt
refFILE: /genesis/extscratch/clc/references/genomes/gsc/bwa-0.7.5a/GRCh37-lite.fa
supportingRefFILE: ['/genesis/extscratch/clc/references/genomes/gsc/bwa-0.7.5a/GRCh37-lite.dict', '/genesis/extscratch/clc/references/genomes/gsc/bwa-0.7.5a/
GRCh37-lite.fa.amb', '/genesis/extscratch/clc/references/genomes/gsc/bwa-0.7.5a/GRCh37-lite.fa.ann', '/genesis/extscratch/clc/references/genomes/gsc/bwa-0.7.5a/
GRCh37-lite.fa.bwt', '/genesis/extscratch/clc/references/genomes/gsc/bwa-0.7.5a/GRCh37-lite.fa.fai', '/genesis/extscratch/clc/references/genomes/gsc/bwa-0.7.5a/
GRCh37-lite.fa.pac', '/genesis/extscratch/clc/references/genomes/gsc/bwa-0.7.5a/GRCh37-lite.fa.sa']
chrLIST: ['1', '2', '3', '4', '5', '6', '7', '8', '9', '10', '11', '12', '13', '14', '15', '16', '17', '18', '19', '20', '21', '22', 'X', 'Y', 'MT', 'GL000207.
1', 'GL000226.1', 'GL000229.1', 'GL000231.1', 'GL000210.1', 'GL000239.1', 'GL000235.1', 'GL000201.1', 'GL000247.1', 'GL000245.1', 'GL000197.1', 'GL000203.1',
'GL000246.1', 'GL000249.1', 'GL000196.1', 'GL000248.1', 'GL000244.1', 'GL000238.1', 'GL000202.1', 'GL000234.1', 'GL000232.1', 'GL000206.1', 'GL000240.1',
'GL000236.1', 'GL000241.1', 'GL000243.1', 'GL000242.1', 'GL000230.1', 'GL000237.1', 'GL000233.1', 'GL000204.1', 'GL000198.1', 'GL000208.1', 'GL000191.1',
'GL000227.1', 'GL000228.1', 'GL000214.1', 'GL000221.1', 'GL000209.1', 'GL000218.1', 'GL000220.1', 'GL000213.1', 'GL000211.1', 'GL000199.1', 'GL000217.1',
'GL000216.1', 'GL000215.1', 'GL000205.1', 'GL000219.1', 'GL000224.1', 'GL000223.1', 'GL000195.1', 'GL000212.1', 'GL000222.1', 'GL000200.1', 'GL000193.1',
'GL000194.1', 'GL000225.1', 'GL000192.1']
#---------------------------------------------------------- Wildcard Constraint Regex -------------------------------------
sampleREGEX: '[^_|-|\/][0-9a-zA-Z-]*'
chrREGEX: '(_[0-9a-zA-Z\.]*(\.\d)?)?'
vcfProgramREGEX: '[0-9a-zA-Z]*'
varTypeREGEX: '[0-9a-zA-Z]*'
fastqReadDirectionREGEX: '(\.\d)?'
fastqCompressionSuffixREGEX: '(\.gz)?'
#---------------------------------------------------------- Directory -----------------------------------------------------
inputDIR: input
outputDIR: output
bamDIR: bam
fastqDIR: fastq
metricsDIR: metrics
mpileupDIR: mpileup
#----------------------------------------------------------                    Info ----------------------------------------
sample: ['Pfeiffer2', 'Pfeiffer3']
sampleFILE: input/sampleFILEsingle.tx
sampleFORMAT: single
#----------------------------------------------------------------------------------------------------------------------


###################################
# fastqGen Parameters
###################################
#---------------------------------------------------------- *Software* ----------------------------------------------------
fastqGen_samtoolsProg: samtools
fastqGen_picardProg: picard
#---------------------------------------------------------- *Shared Variables* --------------------------------------------
fastqGenDIR: fastqGen
#---------------------------------------------------------- bam2fastq_picard ----------------------------------------------
rawBamDIR: rawBam
fastqGen_fastqGen_samtoolsSortMem: 4000000000
fastqGen_picardValStringency: VALIDATION_STRINGENCY=LENIENT
fastqGen_picardMaxRec: MAX_RECORDS_IN_RAM=5000000
#----------------------------------------------------------------------------------------------------------------------


###################################
# fastqUtil Parameters
###################################
#---------------------------------------------------------- *Software* ----------------------------------------------------
fastqUtil_perlProg: perl
#---------------------------------------------------------- *Shared Variables* --------------------------------------------
fastqUtilDIR: fastqUtil
#---------------------------------------------------------- fastq2GZ ------------------------------------------------------
#---------------------------------------------------------- fastqc --------------------------------------------------------
fastqc: /genesis/extscratch/clc/usr/fastqc-0.10.1/fastqc.pl
fastqcDIR: fastqc
nogroupFlag: ''
#---------------------------------------------------------- mergeFASTQ ----------------------------------------------------
mergeFastqRootDIR: ''
```

Remove 'Pfeiffer3'

Figure 21. Genesis screenshot of the produced "config.yaml" file as generated by the python script "buildPipe.py". Due to consistent updates, the ".yaml" file produced may differ slightly from this image.

X. Running a pipeline

Your pipeline is now considered complete. There are a few more steps before we submit it to cluster.

1) **Dry-run the pipeline.**

Before running the pipeline officially on the cluster, it is recommended that users first perform a dry run of their pipeline using the argument '-n', as seen in Figure 22.

```
$ snakemake -n
```

Figure 22. Genesis screenshot of a Snakefile being dry-run. This is a linkage report which only determines anticipated outputs and jobs, it does not submit the jobs for processing.

2) **Run pipeline.**

If no errors are provided, we are then ready to make the cluster call to submit our pipeline. Snakemake behaviour is to execute the top most rule listed within the Snakefile, this behaviour is identical to that of GNU Make. Only the "rule all" should be executed. All other rules carry generic references, wildcards, and require a supporting YAML configuration file; as such, sub-module rules with this system are not to be called directly. Technically, more advanced users can interact with sub-module rules directly via command-line (CLI) arguments, but it will not be discussed within this vignette. The practice of providing CLI arguments to execute rules is cumbersome and strongly discouraged.

****NOTE**** *Snakemake determines working directory as from where the BASH call to Snakemake was made.*

Pre-Conditions:

a. Be in the root of the work space, where the "Snakefile" is located.
b. If the Snakemake file is not named "Snakefile", then the filename must be passed using "-s filename".

Local Calls:

```
$ snakemake -s Snakefile   ←  is equivalent to  → $ snakemake
$ snakemake -s LCR1Pipe    ←  is not equivalent to → $ snakemake
```

Execute a specific rule:

```
$ snakemake -s Snakefile all   ←  is equivalent to  → $ snakemake
```

Or it can be submitted to the cluster using a Snakemake DRMAA submission. Submitting to the cluster requires additional arguments. The call is saved for your convenience into the top (Line #4) of the generated Snakefile. Use the following bash command to have it printed out to terminal:

```
$ head -n 4 Snakefile
```

Copy and paste it into terminal and hit enter. The call should look like:

```
$ snakemake --jobs 100 --cluster-config input/config.json --jobname
    "{cluster.jobName}.{jobid}" --drmaa "{cluster.clusterSpec}" --stats
    log/runStats.txt
```

If successful, users will receive green and yellow text, as demonstrated in Figure 20. The call seen in Figure 20 was able to omit specifying the name of the Snakemake file because the file was named "Snakefile".

Figure 23. Genesis screenshot of a Snakemake pipeline file being submitted to the Sun Grid Engine job scheduler to be run. The "-s" argument is not provided because the Snakefile was named "Snakefile".

XI. References

Continuum Analytics (2017) Anaconda. [Available at: https://anaconda.org]

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/]

XII. 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
```

Snakemake Pipeline - buildPile.py Sequence Diagram

Please refer to Figure 21 on the next page.

Aside from the core build module (buildFile.py), only three processing modules were included (reBam, mPile, and varScan). The system is not limited by the number of modules it can accept, rather, it is limited by the number of modules which are able to be integrated with meaningful results. Addition of more modules would simply result in the continuation of the downward block cascade as seen in Figure 21.

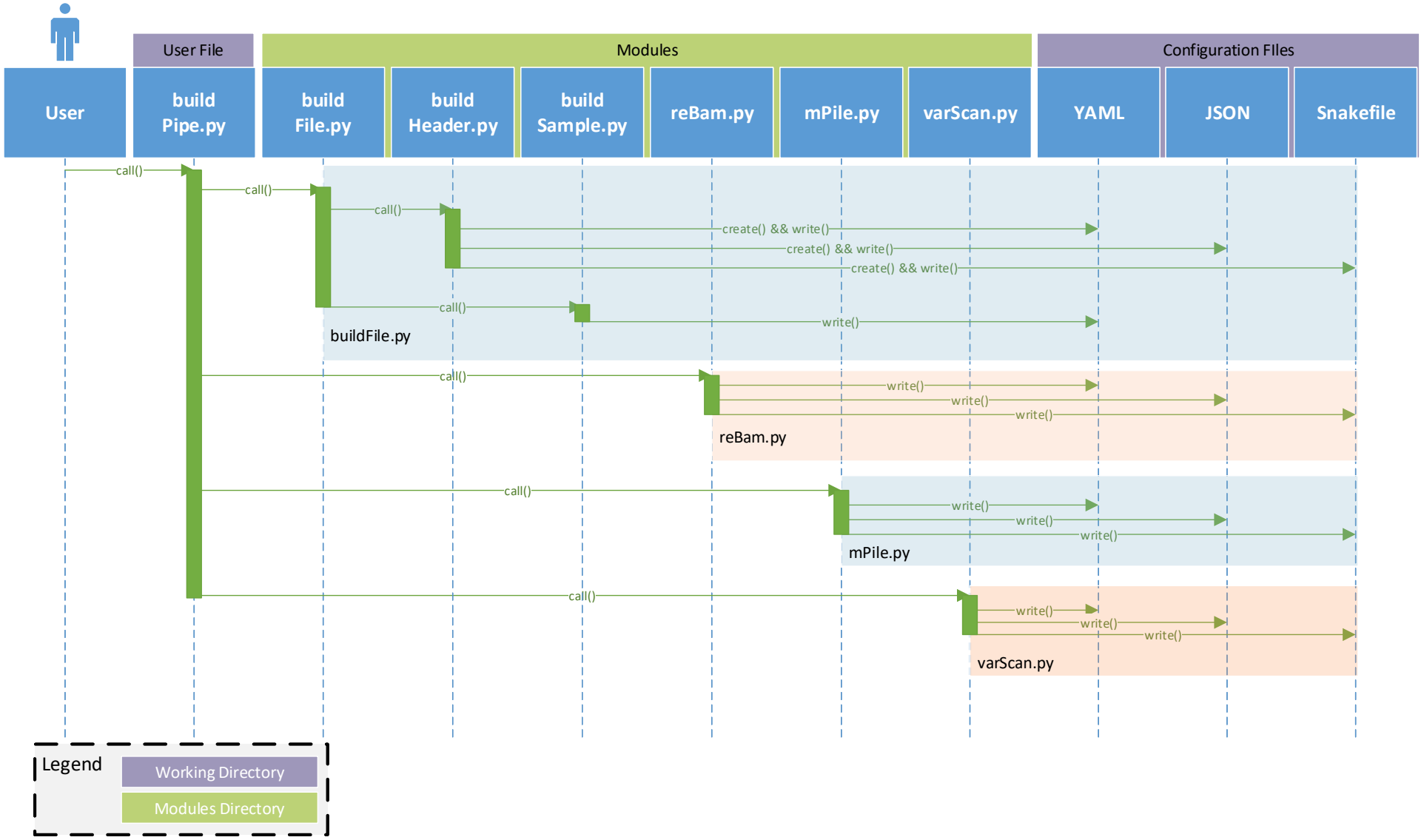

Figure 24. Visio sequence diagram of the user interaction with buildPipe.py. The diagram demonstrates the interactions amongst files in the build process of a the Snakemake pipeline. Note the modularity and low coupling of the system. The user directly interacts with only a single file. Each file that buildPipe.py interacts with is given the ability to direct write information to the three configuration files YAML, JSON, and Snakefile. Message labels are not factual method calls, rather they are meant to describe the actions being performed. As indicated by the legend, light purple background shading at the top of the diagram indicates the file would be within the current working directory. Light green background shading indicates the file is in the shared modules directory.