# SNAPPY Pipeline Documentation

***Release master***

**Manuel Holtgrewe**

**Sep 20, 2023**

# PIPELINE USERS DOCS

This is the documentatation for the CUBI Pipeline. This documentation is split into four parts:

**Pipeline User Docs** Documentation for **pipeline users**. Start here to learn about the pipeline. This section starts at *Quickstart*.

**Pipeline Step Docs** Documentation for the individual pipeline steps. This includes a general description, description of the related configuration settings, and a documentation of generated output files and input workflow steps. This section starts at *Pipeline Step Introduction*.

**Pipeline Developers Docs** Documentation for **pipeline developers**. After you are proficient in using the pipeline, continue reading here if you want to fix, change, or extend the pipeline. This section starts at *Developer's Introduction*.

**API Documentation** This is the entry point for the API.

**Project Info** House-keeping information about the project, such as instructions for developer setup, author list, changelog etc. Start at *How To: Release*).

---

**Note:** Where to Start?

Even if you want to modify the pipeline, it's best to read the user documentation first (BIH users start a *Quickstart*, other users refer to *Installation*) as you need to be able to run the pipeline to test your changes and additions.

---

CHAPTER

ONE

# QUICKSTART

This chapter gives the minimal number of commands required for setting up the pipeline on the BIH cluster.

**Note:** This describes the setup as a pipeline user. If you want to know about the setup as a pipeline developer, see *Installation*.

## 1.1 Install (Mini)conda

First, install miniconda, e.g., into `$HOME/miniconda3`.

```
$ wget -O /tmp/Miniconda3-latest-Linux-x86_64.sh \
    https://repo.continuum.io/miniconda/Miniconda3-latest-Linux-x86_64.sh
$ bash /tmp/Miniconda3-latest-Linux-x86_64.sh -b -p $HOME/miniconda3
```

**Note:** What is conda/miniconda?

Conda is a Python-based package manager that can also package binary files (such as Bioinformatics software). Miniconda is a minimal Conda installation.

If anything goes wrong with your Miniconda installation, you can always just remove `$HOME/miniconda3` and start anew.

Now, make sure it is available in your `PATH` environment variable.

```
$ export PATH=$HOME/miniconda3/bin:$PATH
```

## 1.2 Install Snappy Pipeline

The recommended way of installing the CUBI pipeline is via `pip`.

Replace the *X.Y.Z* in the definition of `VERSION` below with the version you find in the `README.rst` file of the project on the CUBI GitHub.

```
$ VERSION=vX.Y.Z
$ pip install git+ssh://git@github.com:bihealth/snappy-pipeline.git@v${VERSION}
↪#egg=snappy_pipeline
```

Or see `README.rst` for a more detailed installation guide and the environment setup step.

CHAPTER

TWO

# INSTALLATION

**Note:** If you are on the BIH cluster, first read *Quickstart* as this also explains the temporary directory.

## 2.1 Prerequisites

The CUBI pipeline requires Python >=3.7 (e.g., from a Miniconda3 installation).

More recent versions also work but other requirements as Snakemake might make it depend on a more recent Python version.

For cluster execution, you need a Snakemake profile available.

## 2.2 Installing a Release

This is the recommended way if you just want to use the pipeline, simply read *Quickstart*.

## 2.3 Installing as a Developer

It is highly recommended to have a Miniconda installation for the development as this allows for easily resetting everything. You can of course clone the code anywhere you like.

```
$ mkdir -p ~/Development/pipeline_dev
$ cd ~/Development/pipeline_dev
$ git clone git@github.com:bihealth/snappy-pipeline.git
$ cd snappy_pipeline
$ pip install -e .
$ pip install -r requirements/dev.txt
```

It's also a good idea to install some packages required for testing through conda:

```
$ conda env update --name root --file environment.yaml
```

(If you do not do this, please make sure that you have git-lfs in your PATH through other means)

### 2.3.1 Running the Tests

To run the tests, you need to add the packages in `requirements/test.txt`.

```
$ cd ~/Development/pipeline_dev
$ py.test
```

### 2.3.2 Running the Style Checks

```
$ cd ~/Development/pipeline_dev
$ flake8
```

### 2.3.3 Developer Documentation

Make sure to also read the "Pipeline Developer Docs" section, starting with *Developer's Introduction*.

CHAPTER

THREE

# USAGE

As a user, you will mostly interface with the CUBI pipeline system using the `snappy-snake` program.

This program is a wrapper around Snakemake and provides the following features:

- a number of pre-packaged, well-tested workflows (pipeline steps) that are
- driven by configuration and sample sheet files and
- can be shared over multiple projects; and a
- easy-to-use command line interface.

Here is how to get command line help:

```
$ snappy-snake --help
```

CHAPTER

FOUR

# OVERVIEW

This chapter gives you the big picture of the CUBI pipeline system. The audience is people who already have experience with Bioinformatics pipeline/workflow systems and see the benefit of such systems (e.g., GNU Make, Snakemake, bpipe, etc.) over shell files over interactive bash commands. You are part of the audience if you agree that automation is key for effective, efficient, and reproducible Bioinformatics analysis as this is a requirement for important key requirements such as provenance tracking.

Up to a certain point, automation in Bioinformatics workflows is a no-brainer as the same steps always repeat themselves. After this point, the tasks might become very project specific and not benefit from generic, shared automation much. One example is report generation where most of the code cannot be re-used in different projects. Here, different means should be used (e.g., using Rmarkdown documents).

The CUBI pipeline system is aimed at the steps upstream of this "certain point".

## 4.1 Motivation

Generally, the aim was to achieve the following properties in a pipeline system:

**Re-use.** Ability to re-use common Bioinformatics analysis steps. Mostly, these are shell snippets with calls to standard Bioinformatics tools with some glue and conversion code thrown in.

**Configurability.** Allow for good configuration by configuration files. No paths should be hard-coded in the system but instead come from a configuration file. Further, the important parameters that might need tweaking should be exposed through the configuration.

**Sensible Default Parameters.** Provide sensible defaults for configuration. Ideally, use auto-tuning of parameters (e.g., call BWA-ALN for short and single reads, BWA-MEM for long, paired reads).

**Good Documentation.** Provide good documentation of the pipeline system. Widespread re-use improves the pay-off of good documentation.

**Logging software versions.** Log the version of the pipeline and tools to allow analyses to be repeated with the same program versions in the future. At the very least, knowing the versions used can help explain (slight) differences in results.

**Versioning of pipeline code.** Use semantic versioning for result files. Output paths should not change or disappear between minor versions.

**Robustness.** Pipeline execution failure should be prevented (e.g., all required parameters to called tools should be present) and technical weaknesses should be worked around (e.g., by allowing restarting of jobs).

**Restartability.** If the pipeline is stopped or when new input data sets are added, do not repeat unnecessary work. Further, if an intermediate file changes, the dependent files should be updated. (This is similar to what GNU Make does.)

**Ease of use.** Help the users not shoot themselves in the foot too badly (e.g., prevent accidentally overwriting already existing files). Easy local and cluster execution. At least provide sensible defaults for resource requirements, ideally auto-configured from input data.

## 4.2 Definitions

For clarity, this documentation uses the following definitions for separating the code for pipeline steps and the actual execution of code.

**pipeline** Code for performing a set of Bioinformatics tasks in an automated fashion.

**project** A project corresponds to a directory in the file system. A project is an **instance** of a pipeline, in that the different available pipeline parts are plugged together by configuration and the executed.

**(pipeline) step** Program code (Snakefiles, scripts etc.) for performing a certain "encapsulated" set of tasks. Examples are read mapping, variant calling, and variant annotation.

**(pipeline) step instance** A project's folder on the file system, with *configuration*, where a pipeline step is executed. The instance shares the pipeline step code with all other intances of the same type.

**working directory** A directory on the disk for a step instance.

## 4.3 An Example Project

The above part of this chapter is quite abstract. Let us draw some pictures and go from the abstract description to a concrete example. We will use a simple NGS somatic variant calling pipeline for matched tumor/normal pairs, setup for WES or WGS processing.

### 4.3.1 Components of a CUBI Pipeline Project

The following figure shows the different components that are involved for running the CUBI pipeline.

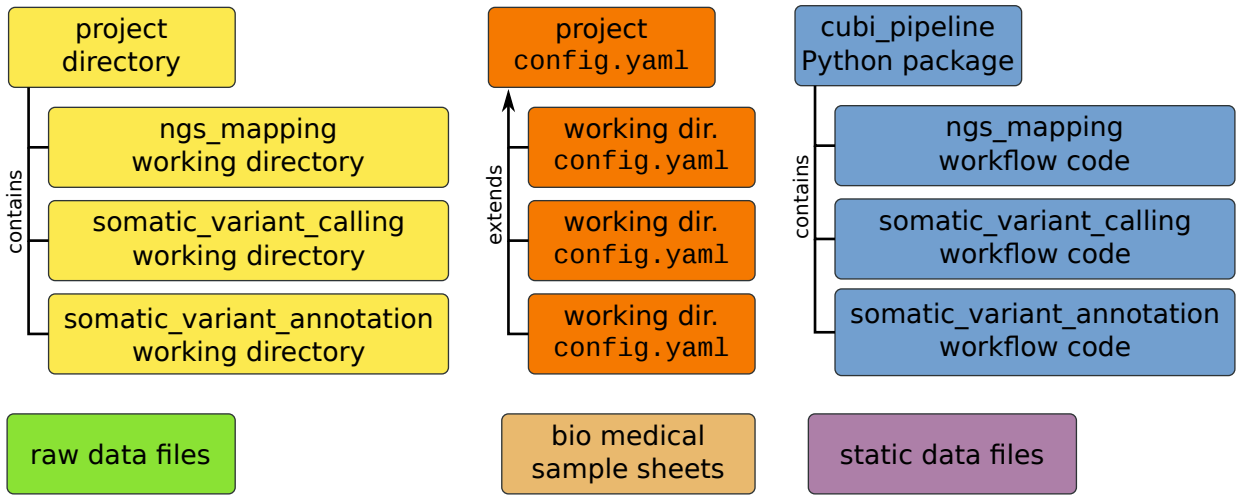


Fig. 1: Overview of the different components for running the CUBI pipeline. Boxes of the same color indicate that the represented entities belong together.

The different parts are as follows

- The blue-colored boxes represent the `snappy_pipeline` Python package that contains the `snappy-snake` executable and the code for the different pipeline steps.
- The yellow-colored boxes represent the project directory with the different sub directories for the step instances. For each step that is to be executed (with a given configuration set), a directory is created. In the given example, there is only one directory (and thus instance) for each step.
- The orange-colored boxes represent the configuration. There is a project-wide `config.yaml` file that defines project-wide defaults. Each step instance can then override certain settings, similar to how sub-classing in OOP works. One read mapping step instance may use GRCh37 for the reference and another instance might use GRCh38 (not shown in this example).
- The purple-colored box represents static data such as the reference sequence, annotations, databases such as dbSNP or dbNSFP. These static data files are created and maintained independently of the individual projects.
- The green box represents the raw input data, e.g., a directory containing the FASTQ reads for each sample. While, of course, raw data can be shared over projects, the data directories are usually under control of the project manager while the static data is under control of the maintainer of the static data project of **Cubit**.
- The brown box represents the bio-medical sample sheets with metadata that describe the data sets of the experiment and also (at least) parts of the experimental setup.

The number of steps might seem intimidating at first, but you will quickly get used to this arrangement. After all, the configuration is closely related to the directories. Further, static data and raw data paths are just put into the configuration once and otherwise you do not have to deal with it. Also, there is UI support for generating and updating the bio-medical sample sheet files.

### 4.3.2 Components of a Pipeline Step Instance Excecution

The following figure shows the components involved when executing a pipeline step (in this case, the NGS read mapping step).

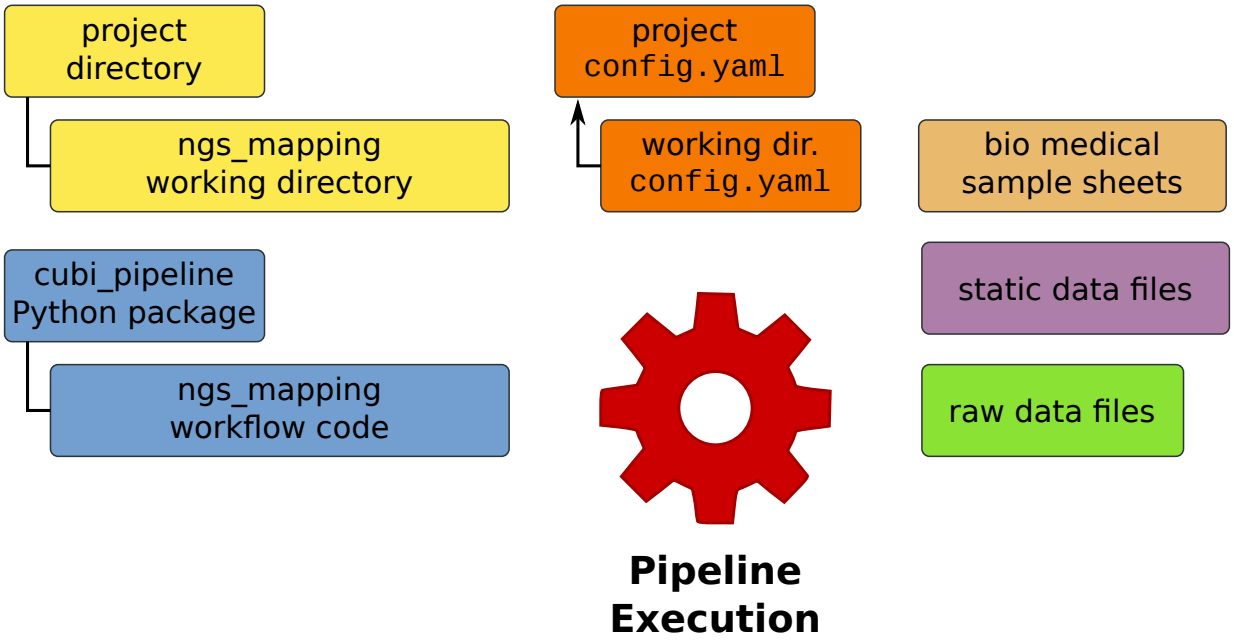


Fig. 2: Overview of the components involved when executing a pipeline step in a working directory.

The different parts are as follows:

- The working directory `project/ngs_mapping`.
- The step-level configuration in `project/ngs_mapping/config.yaml`.
- The project-level configurations in `project/.snappy_pipeline/config.yaml` (by convention).
- The `snappy_pipeline` Python package installed centrally.
- The bio-medical sample sheets with the data sets to use. (The project-wide configuration files point at these files.)
- The static data files setup by the Cubit administrator (here, it would be the reference FASTA path and the read mapper index location).
- The raw data files to be processed by the pipeline step (here, it would be the sample FASTQ files).

### 4.3.3 How FASTQ files are found

In its `data_sets` section, the project-level configuration file provides search paths and search patterns to find the input FASTQ files. `snappy` internally combines these paths & search patterns with the sample-specific path information provided in the sample sheet. In the end, FASTQ files retained for processing are files which paths match:

```
<configuration search path>/<sample-specific folder>/../<search pattern>
```

The search will loop over provided search paths & search patterns. Paired reads files are coupled by similarity of their path. Note that when the `Folder` entry is absent from the sample sheet, the library name is used instead.

However, this default behaviour can be overriden using the `path_link_in` option (which is available only for steps that use FASTQ files as input). When this configuration option is not empty, `snappy` will use it instead of the list of search paths defined in the `data_set` part. It will also ignore the folder information, and rely instead on the library names to search FASTQ files. The search path becomes:

```
<path_link_in>/<library_name>/../<search_pattern>
```

This mechanism enables steps that generate FASTQ files on output, for example adapter trimming. In that case, the input of the mapping step can be redirected towards the ouput of the adapter trimming step using this method.

### 4.3.4 Overview of the Somatic Variant Pipeline

The following figure shows an overview the simple somatic variant calling pipeline used in the example.

The configuration, static data files, and bio-medical sample sheets are used for the input of all pipeline steps. The raw data files are used for the input of the NGS mapping. The resulting read alignments are used as the input for the somatic variant calling. The resulting somatic variant files are then used as the input for the somatic variant annotation.

Within each step the following actions are performed:

1. The reads are first mapped to a reference genome, yielding BAM files contaning the read alignments. (Additional text files with the alignment reports are also generated at this step, but this pipeline does not use these files in the downstream steps.)
2. Then, the pairs of BAM alignments for the matched tumor/normal samples for each individual are given to a somatic variant caller that produces a VCF file with the list of somatic variants for each patient.
3. Finally, variant annotations are added to indicate whether each event is present in the snp databases specified in the configuration (e.g., dbSNP or COSMIC) and functional mutation impact predictions are also added using the tool specified in the configuration (e.g., using MutationTaster).

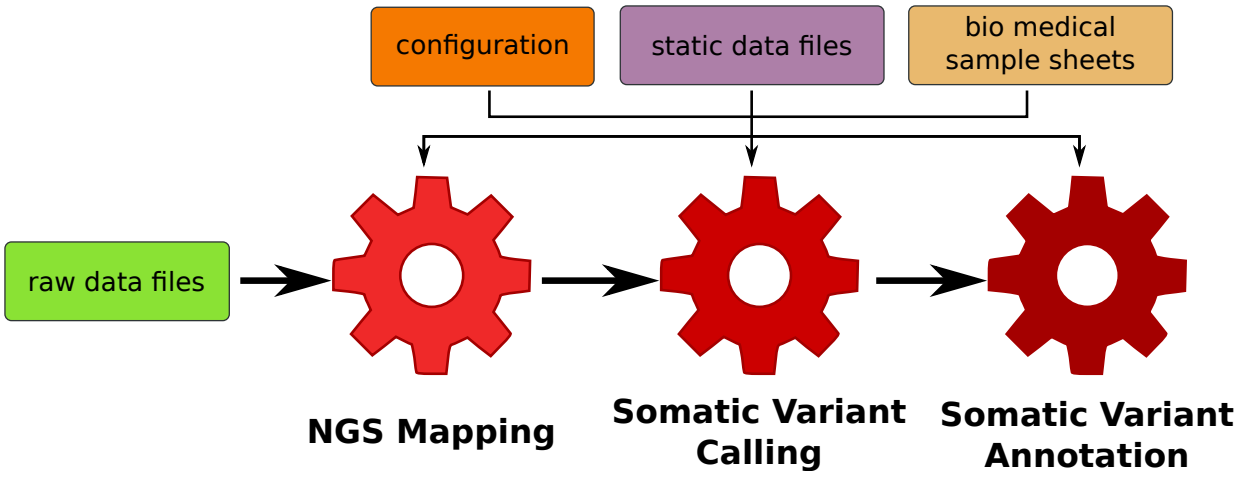


Fig. 3: Overview of the steps in somatic variant calling pipeline.

### 4.3.5 The Matched Cancer Data Schema

For the somatic variant calling, the matched cancer study bio-medical data sheet schema is used. It is described in full in the BioMed Sheets project. Here, we give a summary so this document is self-contained.

- The study contains a number of patients/donors, and each individual is associated with a normal and a tumor sample.
- From each sample, an WES library is generated and sequenced; for each library, there is a directory with the library name, storing the FASTQ files from sequencing.

### 4.3.6 Project Directory Setup

The project directory is setup with the following helper tool:

```
$ snappy-start-project --directory somatic_project
[...]
Do not forget to fill out your README.md file!

SUCCESS: all done, have a nice day!

$ tree -a somatic_project
somatic_project/
+-- .snappy_pipeline/
|   `-- config.yaml
`-- README.md
```

The `config.yaml` file is setup with common configuration for the pipeline steps. The template used uses the paths specific to the Cubit installation on the BIH cluster. In the far future, custom templates will be used for this and the generic files will contain "TODO" entries for changes.

Further, a project-wide `README.md` file is setup in which you can place documentation on the project.

```
$ cd somatic_project
$ head .snappy_pipeline/config.yaml
```

(continues on next page)

(continued from previous page)

```
# CUBI Pipeline Project "somatic_project" Configuration
#
# created: 2017-02-03T12:57:17.302044

# Step Configuration␣
↪================================================================================
#
# Configuration for paths with static data.  This has been preconfigured for the paths␣
↪on the BIH
# cluster.
#
static_data_config:
```

### 4.3.7 Working Directories for Step Instances

Next, we create the different step instances that we want to use using `snappy-start-step`. Note that this will extend the `.snappy_pipeline/config.yaml` file if there is no configuration entry for the given step. A different name for the instance can be given using the `--step` parameter.

Adding the `ngs_mapping` step creates the required directory and configuration files pointing to the global configuration for extension. Note how the difference in the project-wide configuration (and all other files created or modified) is displayed in the script's output.

See *NGS Mapping* for the default configuration of the `ngs_mapping` step. For all configuration settings that have no default and are marked with a `# required` comment (case insensitive), these markers are copied to the project configuration so you know which settings to adjust.

```
$ cd somatic_project
$ snappy-start-step --step ngs_mapping
[...]
INFO: applying the following change:

--- a/.snappy_pipeline/config.yaml  2017-02-03T12:47:32.246833
+++ b/.snappy_pipeline/config.yaml  2017-02-03T12:49:29.811706
@@ -22,7 +22,12 @@
 # Configuration for the individual steps.  These can be filled by the snappy-start-step␣
↪command
 # or initialized already with snappy-start-project.
 #
-step_config: {}
+step_config:
+  ngs_mapping:
+    bwa:
+      path_index:  # REQUIRED
+    star:
+      path_index:  # REQUIRED

 # Data Sets␣
↪============================================================================================
 #
[...]
```

(continues on next page)

(continued from previous page)

```
$ tree ngs_mapping
ngs_mapping/
|-- config.yaml
|-- pipeline_job.sh
`-- sge_log

$ cat ngs_mapping/config.yaml
pipeline_step:
name: ngs_mapping
version: 1

$ref: 'file://../.snappy/config.yaml'
```

Similarly, adding `somatic_variant_calling` adds configuration for somatic variant calling.

```
$ snappy-start-step --step somatic_variant_calling
[...]
INFO: applying the following change:

--- a/.snappy/config.yaml   2017-02-03T13:11:10.023648
+++ b/.snappy/config.yaml   2017-02-03T13:11:20.806588
@@ -29,6 +29,10 @@
     star:
     path_index: REQUIRED  # REQUIRED

+  somatic_variant_calling:
+    path_ngs_mapping: ../ngs_mapping  # REQUIRED
+    scalpel:
+      path_target_regions:  # REQUIRED
 # Data Sets␣
↪=============================================================================================
 #
 # Define data sets.  The search paths and patterns are given per data set.
[...]

$ tree somatic_variant_calling
somatic_variant_calling
+-- sge_log/
`-- config.yaml
```

The same is true for adding `somatic_variant_annotation`.

```
$ snappy-start-step --step somatic_variant_annotation
[...]
INFO: applying the following change:

--- a/.snappy_pipeline/config.yaml  2017-02-03T13:11:20.807090
+++ b/.snappy_pipeline/config.yaml  2017-02-03T13:12:22.693821
@@ -33,6 +33,10 @@
     path_ngs_mapping: ../ngs_mapping  # REQUIRED
     scalpel:
     path_target_regions:  # REQUIRED
```

(continues on next page)

(continued from previous page)

```
+  somatic_variant_annotation:
+    path_somatic_variant_calling: ../somatic_variant_calling  # REQUIRED
+    oncotator:
+      path_corpus: REQUIRED  # REQUIRED
 # Data Sets␣
↪==========================================================================================
 #
 # Define data sets.  The search paths and patterns are given per data set.
@@ -50,4 +54,5 @@
 #       - /fast/projects/medgen_genomes/2017-01-09_acheiropodia
 #     type: germline_variants
 #
-data_sets: {}
+data_sets                   # REQUIRED
+: {}
[...]
$ tree somatic_variant_annotation
somatic_variant_annotation
+-- sge_log/
`-- config.yaml
```

### 4.3.8 Adding Sample Sheets

**Note:** The following does not work yet but should in the future

**TODO**

For matched cancer studies, the most simple way of creating a sample sheet is starting from the shortcut TSV. The following creates a sample sheet TSV shortcut. This is then converted into a JSON bio-med sample sheet.

```
$ cat <<"EOF" | sed $'s/[ \t]\+/\t/g' > .snappy_pipeline/01_data_set.tsv
[Metadata]
schema          cancer_matched
schema_version  v1
title           Example matched cancer tumor/normal study
description     The study has two patients, P001 has one tumor sample, P002 has two

[Data]
patientName sampleName  isTumor    libraryType folderName
P001    N1  N   WES P001-N1-DNA1-WES1
P001    T1  Y   WES P001-T1-DNA1-WES1
P001    T1  Y   mRNA_seq    P001-T1-RNA1-mRNA_seq1
P002    N1  N   WES P002-N1-DNA1-WES1
P002    T1  Y   WES P002-T1-DNA1-WES1
P002    T1  Y   WES P002-T1-RNA1-mRNA_seq1
P002    T2  Y   WES P002-T2-DNA1-WES1
P002    T2  Y   mRNA_seq    P002-T2-RNA1-mRNA_seq1
EOF
$ biomedsheets -t matched_cancer \
    --input .snappy_pipeline/01_data_set.tsv \
```

(continues on next page)

(continued from previous page)

```
    --output .snappy_pipeline/01_data_set.json
$ head .snappy_pipeline/01_data_set.json
[TODO]
```

**Note:** Updating entries in data set TSV files does not work yet and requires a re-starting from scratch. As the data set primary keys are part of the file names, changing the PK of sample or library will require cleaning all output files and re-running the whole pipeline. Overall, it is better to only use the JSON sheet files and the corresponding tools and helpers.

Now, we have to register the data set in the configuration. Ensure that the `data_sets` entry look as follows. Replace `<path-to-demo-dir>` with the path to the `demo` directory of the `snappy_pipeline` project.

```
data_sets:
  first_batch:
    file: 01_first_batch.tsv
    search_patterns:
      # Note that currently only "left" and "right" key known
      - {'left': '*/L???/*_R1.fastq.gz', 'right': '*/L???/*_R2.fastq.gz'}
    search_paths: ['<path-to-demo-dir>/input/01_first_batch']
    type: matched_cancer
```

The full configuration format will be described elsewhere. It is notable, however, that there also is an optional `naming_scheme` property for each batch. Using this, you can select between naming based on secondary ID and pk (`secondary_id_pk`) and secondary ID alone (`only_secondary_id`).

### 4.3.9 Executing the Project's Pipeline

After executing the steps from above, our pipeline is ready to use. Each pipeline step instance will automatically run each predecessor within the pipeline. Thus, it is enough to execute the pipeline in the `somatic_variant_annotation` step.

For running, locally use:

```
$ cd somatic_variant_annotation
$ snappy-snake -p --step somatic_variant_annotation
```

For running with Snakemake profile on the cluster, use the `--snappy-pipeline-use-profile` parameter.

```
$ cd somatic_variant_annotation
$ snappy-snake -p --step somatic_variant_annotation --snappy-pipeline-use-profile "cubi-
↪v1"
```

CHAPTER

FIVE

# PIPELINE STEP INTRODUCTION

This part contains the links into the pipeline step documentation. The pipeline steps themselves are documented in the source code for easier syncing between the step code and documentation. For your convenience, the pipeline step documentation appears here in addition to the API documentation.

The first chapter *Generic Pipeline Step Description* gives an overview of the overall structure of a pipeline step. The following chapters each document one implemented pipeline step.

CHAPTER

SIX

# GENERIC PIPELINE STEP DESCRIPTION

Generally, each pipeline step takes some **input**, processes it in a **work** directory, and then creates an **output** directory with the pipeline step's result. Each pipeline step is implemented as a Snakemake workflow and a **step instance** corresponds to a Snakemake working directory on the file system.

## 6.1 File System Layout

The overall layout for a pipeline step instance is as follows:

```
working_dir_name/
+-- [input/]
+-- work/
+-- output/
`-- config.yaml
```

### 6.1.1 Directory `input/`

An optional input directory. This is directory is only created if files are to be linked into the directory that are not generated by another workflow. For example, the `ngs_mapping` pipeline step links in **variable** data the input FASTQ files into the `input/` directory.

Note that **static data** (such as reference, read mapper indices, annotation, etc., all that can be statically configured) is not linked into the `input/` directory. In contrast, the `variant_calling` step does not need an `input/` directory as it only works on the read alignments generated by the `ngs_mapping` step.

### 6.1.2 Directory `work/`

This is the working directory that contains all results of the pipeline, including logs as well as intermediary and final results. Intermediary results should be marked by the Snakemake `temp()` directive but there is no guarantee that temporary files are removed after the pipeline step finishes. Also note that you as a user have to consider the directory structure and file names in `work/` as **unstable**.

In short: in `work/`, the pipeline step authors can do whatever they want, including changing it between minor versions.

### 6.1.3 Directory `output/`

This is the "public" output directory. It contains a **stable** directory structure with **stable** names. The `output/` directory contains no files but rather **symlinks** into the `work/` directory.

By convention, the directories and file names should mirror the ones in `work/` (and thus form a subset) for simplicity. However, in order to keep semantic versioning, this convention might be broken to keep paths in the `output/` directory stable when something in `work/` changes.

## 6.2 Step Instance Configuration `config.yaml`

Each step instance must have a configuration file `config.yaml`. The file contains a YAML or JSON-formatted directory structure and typically looks as follows.

```
pipeline_step:
  name: ngs_mapping
  version: 1

$ref: 'file://../.snappy/config.yaml'
```

Consider the second part first. Here, JSON Pointer notation is used for referencing and loading the file `../.snappy_pipeline/config.yaml` at the root of YAML file. This file contains the basic configuration for all pipeline step instances in a project. The configuration file `config.yaml` in the pipeline step instance directory can then override settings as fit. These settings are placed into the YAML file and on loading of the `config.yaml` file, the configuration settings of both the including and the included file will be merged. The settings of the including file overriding the settings from the included files.

Consider the first part now. Here, it is simply configured that the pipeline step to be executed is named `ngs_mapping` and version 1 is assumed to be present. The versioning allows the pipeline step to check whether there are incompatibilities in the pipeline step implementation version and the version used when writing the step instance configuration.

---

**Note: Background Data Sets**

These data sets are available for use as background data. The provided data can be sparser (e.g., only NGS library for normal samples in an otherwise matched cancer/normal study).

The execution of `cubi-snake` in a directory will not automatically generate these files. Rather, they are only generated when used in a pipeline step such as `somatic_targeted_cnv_calling`.

---

CHAPTER

SEVEN

# ADAPTER TRIMMING

Implementation of the `adapter_trimming` step

The adapter_trimming step performs adapter & quality trimming of reads (DNA or RNA). The tools are highly configurable, and provide feedback of the success of the operation.

## 7.1 Step Input

For each library defined in all sample sheets, the instances of this step will search for the input files according to the configuration. The found read files will be linked into `work/input_links/{library_name}` (status quo, not a output path, thus path not guaranteed to be stable between minor versions).

The search paths can be overridden using the step configuration option `path_link_in`. `path_link_in` is a general features that enables pre-processing steps, typically before mapping.

### 7.1.1 Data Set Configuration

Consider the following data set definition from the main configuration file.

```
data_sets:
  first_batch:
    file: 01_first_batch.tsv
    search_patterns:
      # Note that currently only "left" and "right" key known
      - {'left': '*/L???/*_R1.fastq.gz', 'right': '*/L???/*_R2.fastq.gz'}
    search_paths: ['../input/01_first_batch']
```

Here, the data set `first_batch` is defined. The sample sheet file is named `01_first_batch.tsv` and looked for in the relative path to the configuration file. The input search will be start in the (one, but could be more than one) path `../input/01_first_batch` (relative to the directory containing the configuration file). The sample sheet provides a `folderName` `extraInfo` entry for each NGS library. This folder name is searched for (e.g., `P001-N1-DNA1-WES`). Once such a folder is found, the patterns in the values of the dict `search_patterns` are used for locating the paths of the actual files.

Currently, the only supported keys in the `search_patterns` dict are `"left"` and `"right""` (the latter can be omitted when only searching for single-end reads).

Consider the following example:

```
../input/
  `-- 01_first_batch
      |-- P001-N1-DNA1-WES1
      |   `-- 42KF5AAXX
      |       `-- L001
      |           |-- P001-N1-DNA1-WES1_R1.fastq.gz
      |           |-- P001-N1-DNA1-WES1_R1.fastq.gz.md5
      |           |-- P001-N1-DNA1-WES1_R2.fastq.gz
      |           `-- P001-N1-DNA1-WES1_R2.fastq.gz.md5
      [...]
```

Here, the folder `01_first_batch` will be searched for a directory named `P001-N1-DNA1-WES`. After finding, the relative paths `42KF5AAXX/L001/P001-N1-DNA1-WES1_R1.fastq.gz` and `42KF5AAXX/L001/P001-N1-DNA1-WES1_R2.fastq.gz` will be found and used for the left/right parts of a paired read set.

### 7.1.2 Overriding data set confguration with `path_link_in`

When the config option `path_link_in` is set, it takes precedence on the search paths defined in the data set configuration.

The searching for input files will follow the same rules as defined in the data set configuration, except that the base path for the search provided by one single path defined in the configuration of the step.

#### Mixing Single-End and Paired-End Reads

By default, it is checked that for each `search_pattern`, the same number of matching files has to be found, otherwise directories are ignored. The reason is to reduce the number of possible errors when linking in files. You can change this behaviour by specifying `mixed_se_pe: True` in the data set information. Then, it will be allowed to have the matches for the `right` entry to be empty. However, you will need to consistently have either SE or PE data for each library; it is allowed to mix SE and PE libraries within one project but not to have PE and SE data for one library.

Note that mixing single-end and paired-end reads is not (yet) supported when overriding the data set configuration by setting a value to the configuration option `path_link_in`.

## 7.2 Step Output

Adapter trimming will be performed for all NGS libraries in all sample sheets. For each combination of tool library, a directory `{tool}/{lib_name}-{lib_pk}/out` will be created. Therein, trimmed fastq files will be created.

The input structure and file names will be maintained on output. For example, it might look as follows for the example from above:

```
output/
+-- bbduk
|   `-- out
|       `-- P001-N1-DNA1-WES1
|           |-- 42KF5AAXX
|           |   `-- L001
|           |       |-- P001-N1-DNA1-WES1_R1.fastq.gz
|           |       |-- P001-N1-DNA1-WES1_R1.fastq.gz.md5
|           |       |-- P001-N1-DNA1-WES1_R2.fastq.gz
```

(continues on next page)

(continued from previous page)

```
|           |       `-- P001-N1-DNA1-WES1_R2.fastq.gz.md5
|           `-- .done
[...]
```

## 7.3 Default Configuration

The default configuration is as follows.

```
# Default configuration adapter_trimming
step_config:
  adapter_trimming:
    path_link_in: ""  # OPTIONAL Override data set configuration search paths for FASTQ␣
↪files
    tools: [bbduk, fastp]  # REQUIRED, available: 'bbduk' and 'fastp'.
    bbduk:
      adapter_sequences: []  # REQUIRED
      # - /fast/work/groups/cubi/projects/biotools/static_data/app_support/bbtools/39.01/
↪resources/adapters.fa
      # - /fast/work/groups/cubi/projects/biotools/static_data/app_support/bbtools/39.01/
↪resources/phix174_ill.ref.fa.gz
      # Note: The author recommends setting tpe=t & tbo=t when adapter trimming paired␣
↪reads.
      num_threads: 8

      # Non-default parameters from https://www.biostars.org/p/268221/
      # & https://github.com/ewels/MultiQC/issues/1146#issuecomment-607980076

      # Input parameters:
      interleaved: auto    # (int) t/f overrides interleaved autodetection.
      qin: auto            # Input quality offset: 33 (Sanger), 64, or auto.
      copyundefined: f     # (cu) Process non-AGCT IUPAC reference bases by making all
                           # possible unambiguous copies.  Intended for short motifs
                           # or adapter barcodes, as time/memory use is exponential.

      # Output parameters:
      nzo: t               # Only write statistics about ref sequences with nonzero hits.
      qout: auto           # Output quality offset: 33 (Sanger), 64, or auto.
      statscolumns: 3      # (cols) Number of columns for stats output, 3 or 5.
                           # 5 includes base counts.
      rename: f            # Rename reads to indicate which sequences they matched.
      refnames: f          # Use names of reference files rather than scaffold IDs.
      trd: f               # Truncate read and ref names at the first whitespace.
      ordered: f           # Set to true to output reads in same order as input.

      # Histogram output parameters:
      gcbins: auto         # Number gchist bins.  Set to 'auto' to use read length.
      maxhistlen: 6000     # Set an upper bound for histogram lengths; higher uses
                           # more memory.  The default is 6000 for some histograms
                           # and 80000 for others.
```

(continues on next page)

(continued from previous page)

```
    # Histograms for mapped sam/bam files only:
    histbefore: t        # Calculate histograms from reads before processing.
    idbins: 100          # Number idhist bins.  Set to 'auto' to use read length.

    # Processing parameters:
    k: 21                # Kmer length used for finding contaminants.  Contaminants
                         # shorter than k will not be found.  k must be at least 1.
                         # bbduk default: 27
    rcomp: t             # Look for reverse-complements of kmers in addition to
                         # forward kmers.
    maskmiddle: t        # (mm) Treat the middle base of a kmer as a wildcard, to
                         # increase sensitivity in the presence of errors.
    minkmerhits: 1       # (mkh) Reads need at least this many matching kmers
                         # to be considered as matching the reference.
    minkmerfraction: 0.0 # (mkf) A reads needs at least this fraction of its total
                         # kmers to hit a ref, in order to be considered a match.
                         # If this and minkmerhits are set, the greater is used.
    mincovfraction: 0.0  # (mcf) A reads needs at least this fraction of its total
                         # bases to be covered by ref kmers to be considered a match.
                         # If specified, mcf overrides mkh and mkf.
    hammingdistance: 1   # (hdist) Maximum Hamming distance for ref kmers (subs only).
                         # Memory use is proportional to (3*K)^hdist.
                         # bbduk default: 0
    qhdist: 0            # Hamming distance for query kmers; impacts speed, not memory.
    editdistance: 0      # (edist) Maximum edit distance from ref kmers (subs
                         # and indels).  Memory use is proportional to (8*K)^edist.
    hammingdistance2: 0  # (hdist2) Sets hdist for short kmers, when using mink.
    qhdist2: 0           # Sets qhdist for short kmers, when using mink.
    editdistance2: 0     # (edist2) Sets edist for short kmers, when using mink.
    forbidn: f           # (fn) Forbids matching of read kmers containing N.
                         # By default, these will match a reference 'A' if
                         # hdist>0 or edist>0, to increase sensitivity.
    removeifeitherbad: t # (rieb) Paired reads get sent to 'outmatch' if either is
                         # match (or either is trimmed shorter than minlen).
                         # Set to false to require both.
    trimfailures: f      # Instead of discarding failed reads, trim them to 1bp.
                         # This makes the statistics a bit odd.
    findbestmatch: f     # (fbm) If multiple matches, associate read with sequence
                         # sharing most kmers.  Reduces speed.
    skipr1: f            # Don't do kmer-based operations on read 1.
    skipr2: f            # Don't do kmer-based operations on read 2.
    ecco: f              # For overlapping paired reads only.  Performs error-
                         # correction with BBMerge prior to kmer operations.

    # Trimming/Filtering/Masking parameters:
    # Note - if ktrim, kmask, and ksplit are unset, the default behavior is kfilter.
    # All kmer processing modes are mutually exclusive.
    # Reads only get sent to 'outm' purely based on kmer matches in kfilter mode.

    ktrim: r             # Trim reads to remove bases matching reference kmers.
                         # Values:
                         #   f (don't trim), [bbduk default]
```

(continues on next page)

(continued from previous page)

```
                        #   r (trim to the right),
                        #   l (trim to the left)
    kmask: ""           # Replace bases matching ref kmers with another symbol.
                        # Allows any non-whitespace character, and processes short
                        # kmers on both ends if mink is set.  'kmask: lc' will
                        # convert masked bases to lowercase.
    maskfullycovered: f # (mfc) Only mask bases that are fully covered by kmers.
    ksplit: f           # For single-ended reads only.  Reads will be split into
                        # pairs around the kmer.  If the kmer is at the end of the
                        # read, it will be trimmed instead.  Singletons will go to
                        # out, and pairs will go to outm.  Do not use ksplit with
                        # other operations such as quality-trimming or filtering.
    mink: 11            # Look for shorter kmers at read tips down to this length,
                        # when k-trimming or masking.  0 means disabled.  Enabling
                        # this will disable maskmiddle.
                        # bbduk default: 0 (disabled)
    qtrim: rl           # Trim read ends to remove bases with quality below trimq.
                        # Performed AFTER looking for kmers.  Values:
                        #   rl (trim both ends),
                        #   f (neither end),  [bbduk default]
                        #   r (right end only),
                        #   l (left end only),
                        #   w (sliding window).
    trimq: 25           # Regions with average quality BELOW this will be trimmed,
                        # if qtrim is set to something other than f.  Can be a
                        # floating-point number like 7.3.
                        # Very strict quality threshold, bbduk default: 6
    minlength: 35       # (ml) Reads shorter than this after trimming will be
                        # discarded.  Pairs will be discarded if both are shorter.
                        # bbduk default: 10
    mlf: 0              # (minlengthfraction) Reads shorter than this fraction of
                        # original length after trimming will be discarded.
    minavgquality: 0    # (maq) Reads with average quality (after trimming) below
                        # this will be discarded.
    maqb: 0             # If positive, calculate maq from this many initial bases.
    minbasequality: 0   # (mbq) Reads with any base below this quality (after
                        # trimming) will be discarded.
    maxns: -1           # If non-negative, reads with more Ns than this
                        # (after trimming) will be discarded.
    mcb: 0              # (minconsecutivebases) Discard reads without at least
                        # this many consecutive called bases.
    ottm: f             # (outputtrimmedtomatch) Output reads trimmed to shorter
                        # than minlength to outm rather than discarding.
    tp: 0               # (trimpad) Trim this much extra around matching kmers.
    tbo: f              # (trimbyoverlap) Trim adapters based on where paired
                        # reads overlap.
    strictoverlap: t    # Adjust sensitivity for trimbyoverlap mode.
    minoverlap: 14      # Require this many bases of overlap for detection.
    mininsert: 40       # Require insert size of at least this for overlap.
                        # Should be reduced to 16 for small RNA sequencing.
    tpe: f              # (trimpairsevenly) When kmer right-trimming, trim both
                        # reads to the minimum length of either.
```

(continues on next page)

(continued from previous page)

```
    forcetrimleft: 0     # (ftl) If positive, trim bases to the left of this position
                         # (exclusive, 0-based).
    forcetrimright: 0    # (ftr) If positive, trim bases to the right of this position
                         # (exclusive, 0-based).
    forcetrimright2: 0   # (ftr2) If positive, trim this many bases on the right end.
    forcetrimmod: 5      # (ftm) If positive, right-trim length to be equal to zero,
                         # modulo this number.
                         # bbduk default: 0
    restrictleft: 0      # If positive, only look for kmer matches in the
                         # leftmost X bases.
    restrictright: 0     # If positive, only look for kmer matches in the
                         # rightmost X bases.
    mingc: 0             # Discard reads with GC content below this.
    maxgc: 1             # Discard reads with GC content above this.
    # gcpairs: t           # Use average GC of paired reads.    Deprecated option?
    #                      # Also affects gchist.
    tossjunk: f          # Discard reads with invalid characters as bases.
    swift: f             # Trim Swift sequences: Trailing C/T/N R1, leading G/A/N R2.

    # Header-parsing parameters - these require Illumina headers:
    chastityfilter: f    # (cf) Discard reads with id containing ' 1:Y:' or ' 2:Y:'.
    barcodefilter: f     # Remove reads with unexpected barcodes if barcodes is set,
                         # or barcodes containing 'N' otherwise.  A barcode must be
                         # the last part of the read header.  Values:
                         #   t:     Remove reads with bad barcodes.
                         #   f:     Ignore barcodes.
                         #   crash: Crash upon encountering bad barcodes.
    barcodes: ""         # File of barcodes.
    xmin: -1             # If positive, discard reads with a lesser X coordinate.
    ymin: -1             # If positive, discard reads with a lesser Y coordinate.
    xmax: -1             # If positive, discard reads with a greater X coordinate.
    ymax: -1             # If positive, discard reads with a greater Y coordinate.

    # Polymer trimming:
    trimpolya: 0         # If greater than 0, trim poly-A or poly-T tails of
                         # at least this length on either end of reads.
    trimpolygleft: 0     # If greater than 0, trim poly-G prefixes of at least this
                         # length on the left end of reads.  Does not trim poly-C.
    trimpolygright: 8    # If greater than 0, trim poly-G tails of at least this
                         # length on the right end of reads.  Does not trim poly-C.
                         # bbduk default: don't trim polyG (trimpolyg=0)
    trimpolyg: 0         # This sets both left and right at once.
    filterpolyg: 8       # If greater than 0, remove reads with a poly-G prefix of
                         # at least this length (on the left).
    # Note: there are also equivalent poly-C flags.

    # Entropy/Complexity parameters:
    entropy: -1          # Set between 0 and 1 to filter reads with entropy below
                         # that value.  Higher is more stringent.
    entropywindow: 50    # Calculate entropy using a sliding window of this length.
    entropyk: 5          # Calculate entropy using kmers of this length.
    minbasefrequency: 0  # Discard reads with a minimum base frequency below this.
```

(continues on next page)

(continued from previous page)

```
      entropytrim: f       # Values:
                           #    f:  (false) Do not entropy-trim.
                           #    r:  (right) Trim low entropy on the right end only.
                           #    l:  (left) Trim low entropy on the left end only.
                           #    rl: (both) Trim low entropy on both ends.
      entropymask: f       # Values:
                           #    f:  (filter) Discard low-entropy sequences.
                           #    t:  (true) Mask low-entropy parts of sequences with N.
                           #    lc: Change low-entropy parts of sequences to lowercase.
      entropymark: f       # Mark each base with its entropy value.  This is on a scale
                           # of 0-41 and is reported as quality scores, so the output
                           # should be fastq or fasta+qual.
      # NOTE: If set, entropytrim overrides entropymask.

      # Cardinality estimation:
      cardinality: f       # (loglog) Count unique kmers using the LogLog algorithm.
      cardinalityout: f    # (loglogout) Count unique kmers in output reads.
      loglogk: 31          # Use this kmer length for counting.
      loglogbuckets: 2048  # Use this many buckets for counting.

    fastp:
      num_threads: 4

      trim_front1: 0                    # trimming how many bases in front for read1,␣
↪default is 0 (int [=0])
      trim_tail1: 0                     # trimming how many bases in tail for read1,␣
↪default is 0 (int [=0])
      max_len1: 0                       # if read1 is longer than max_len1, then trim␣
↪read1 at its tail to make it as long as max_len1. Default 0 means no limitation (int␣
↪[=0])
      trim_front2: 0                    # trimming how many bases in front for read2.␣
↪If it's not specified, it will follow read1's settings (int [=0])
      trim_tail2: 0                     # trimming how many bases in tail for read2.␣
↪If it's not specified, it will follow read1's settings (int [=0])
      max_len2: 0                       # if read2 is longer than max_len2, then trim␣
↪read2 at its tail to make it as long as max_len2. Default 0 means no limitation. If it
↪'s not specified, it will follow read1's settings (int [=0])
      dedup: False                      # enable deduplication to drop the duplicated␣
↪reads/pairs
      dup_calc_accuracy: 0              # accuracy level to calculate duplication (1~
↪6), higher level uses more memory (1G, 2G, 4G, 8G, 16G, 24G). Default 1 for no-dedup␣
↪mode, and 3 for dedup mode. (int [=0])
      dont_eval_duplication: True       # don't evaluate duplication rate to save time␣
↪and use less memory.
      trim_poly_g: True                 # force polyG tail trimming, by default␣
↪trimming is automatically enabled for Illumina NextSeq/NovaSeq data
      poly_g_min_len: 8                 # the minimum length to detect polyG in the␣
↪read tail. 10 by default. (int [=10])
      trim_poly_x: False                # enable polyX trimming in 3' ends.
      poly_x_min_len: 10                # the minimum length to detect polyX in the␣
↪read tail. 10 by default. (int [=10])
      cut_front: False                  # move a sliding window from front (5') to tail,
↪ drop the bases in the window if its mean quality < threshold, stop otherwise.
```

(continues on next page)

(continued from previous page)

```
      cut_tail: False                    # move a sliding window from tail (3') to front,
↪ drop the bases in the window if its mean quality < threshold, stop otherwise.
      cut_right: False                   # move a sliding window from front to tail, if␣
↪meet one window with mean quality < threshold, drop the bases in the window and the␣
↪right part, and then stop.
      cut_front_window_size: 4           # the window size option of cut_front, default␣
↪to cut_window_size if not specified (int [=4])
      cut_front_mean_quality: 20         # the mean quality requirement option for cut_
↪front, default to cut_mean_quality if not specified (int [=20])
      cut_tail_window_size: 4            # the window size option of cut_tail, default␣
↪to cut_window_size if not specified (int [=4])
      cut_tail_mean_quality: 20          # the mean quality requirement option for cut_
↪tail, default to cut_mean_quality if not specified (int [=20])
      cut_right_window_size: 4           # the window size option of cut_right, default␣
↪to cut_window_size if not specified (int [=4])
      cut_right_mean_quality: 20         # the mean quality requirement option for cut_
↪right, default to cut_mean_quality if not specified (int [=20])
      disable_quality_filtering: False   # quality filtering is enabled by default. If␣
↪this option is specified, quality filtering is disabled
      qualified_quality_phred: 15        # the quality value that a base is qualified.␣
↪Default 15 means phred quality >=Q15 is qualified. (int [=15])
      unqualified_percent_limit: 40      # how many percents of bases are allowed to be␣
↪unqualified (0~100). Default 40 means 40% (int [=40])
      n_base_limit: 5                    # if one read's number of N base is >n_base_
↪limit, then this read/pair is discarded. Default is 5 (int [=5])
      average_qual: 0                    # if one read's average quality score <avg_qual,
↪ then this read/pair is discarded. Default 0 means no requirement (int [=0])
      disable_length_filtering: False    # length filtering is enabled by default. If␣
↪this option is specified, length filtering is disabled
      length_required: 15                # reads shorter than length_required will be␣
↪discarded, default is 15. (int [=15])
      length_limit: 0                    # reads longer than length_limit will be␣
↪discarded, default 0 means no limitation. (int [=0])
      low_complexity_filter: False       # enable low complexity filter. The complexity␣
↪is defined as the percentage of base that is different from its next base (base[i] !=␣
↪base[i+1]).
      complexity_threshold: 30           # the threshold for low complexity filter (0~
↪100). Default is 30, which means 30% complexity is required. (int [=30])
      filter_by_index1: ""               # specify a file contains a list of barcodes␣
↪of index1 to be filtered out, one barcode per line (string [=])
      filter_by_index2: ""               # specify a file contains a list of barcodes␣
↪of index2 to be filtered out, one barcode per line (string [=])
      filter_by_index_threshold: 0       # the allowed difference of index barcode for␣
↪index filtering, default 0 means completely identical. (int [=0])
      correction: False                  # enable base correction in overlapped regions␣
↪(only for PE data), default is disabled
      overlap_len_require: 30            # the minimum length to detect overlapped␣
↪region of PE reads. This will affect overlap analysis based PE merge, adapter trimming␣
↪and correction. 30 by default. (int [=30])
      overlap_diff_limit: 5              # the maximum number of mismatched bases to␣
↪detect overlapped region of PE reads. This will affect overlap analysis based PE merge,
↪ adapter trimming and correction. 5 by default. (int [=5])
```

(continues on next page)

(continued from previous page)

```
      overlap_diff_percent_limit: 20      # the maximum percentage of mismatched bases␣
↪to detect overlapped region of PE reads. This will affect overlap analysis based PE␣
↪merge, adapter trimming and correction. Default 20 means 20%. (int [=20])
      umi: False                          # enable unique molecular identifier (UMI)␣
↪preprocessing
      umi_loc: ""                         # specify the location of UMI, can be (index1/
↪index2/read1/read2/per_index/per_read, default is none (string [=])
      umi_len: 0                          # if the UMI is in read1/read2, its length␣
↪should be provided (int [=0])
      umi_prefix: ""                      # if specified, an underline will be used to␣
↪connect prefix and UMI (i.e. prefix=UMI, UMI=AATTCG, final=UMI_AATTCG). No prefix by␣
↪default (string [=])
      umi_skip: 0                         # if the UMI is in read1/read2, fastp can skip␣
↪several bases following UMI, default is 0 (int [=0])
      overrepresentation_analysis: False  # enable overrepresented sequence analysis.
```

## 7.4 Available Adapter Trimming Tools

The following adpter trimming tools are currently available

- "bbduk"
- "fastp"

CHAPTER

EIGHT

# GERMLINE BUILD TARGET SEQUENCE GCNV MODEL

Implementation of the `helper_gcnv_model_targeted` step

The `helper_gcnv_model_targeted` step takes as the input the results of the `ngs_mapping` step (aligned germline reads) and builds a model that can be used by GATK4 gCNV for a particular library kit.

## 8.1 Step Input

The step uses Snakemake sub workflows for the result of the `ngs_mapping` (aligned reads BAM files).

## 8.2 Step Output

All donors will be used to generate the two parts of the required gCNV model, specifically: `ploidy-model` and `cnv_calls-model`. Both are required to execute gCNV in CASE mode.

For example, the relevant directories might look as follows:

```
work/
+-- bwa.gcnv_contig_ploidy.<library_kit_name>
    `-- out
        `-- bwa.gcnv_contig_ploidy.<library_kit_name>
            |-- SAMPLE_0
            |   |-- contig_ploidy.tsv
            |   |-- global_read_depth.tsv
            |   |-- mu_psi_s_log__.tsv
            |   |-- sample_name.txt
            |   `-- std_psi_s_log__.tsv
            |-- [...]
            `-- bwa.gcnv_contig_ploidy.<library_kit_name>
                `-- ploidy-model
                    |-- contig_ploidy_prior.tsv
                    |-- gcnvkernel_version.json
                    |-- interval_list.tsv
                    |-- mu_mean_bias_j_lowerbound__.tsv
                    |-- mu_psi_j_log__.tsv
                    |-- ploidy_config.json
                    |-- std_mean_bias_j_lowerbound__.tsv
                    `-- std_psi_j_log__.tsv
+-- bwa.gcnv_call_cnvs.<library_kit_name>.***_of_***
```

(continues on next page)

(continued from previous page)

```
    `-- out
        `-- bwa.gcnv_call_cnvs.<library_kit_name>.***_of_***
            |-- cnv_calls-calls
            |   |-- SAMPLE_0
            |   |   `-- [...]
            |   |-- [...]
            |-- cnv_calls-model
            |  |-- denoising_config.json
            |  |-- gcnvkernel_version.json
            |  |-- interval_list.tsv
            |  |-- log_q_tau_tk.tsv
            |  |-- mu_W_tu.tsv
            |  |-- mu_ard_u_log__.tsv
            |  |-- mu_log_mean_bias_t.tsv
            |  |-- mu_psi_t_log__.tsv
            |  |-- std_W_tu.tsv
            |  |-- std_ard_u_log__.tsv
            |  |-- std_log_mean_bias_t.tsv
            |  `-- std_psi_t_log__.tsv
            `-- cnv_calls-tracking
                `-- [...]
```

## 8.3 Global Configuration

- At the moment, no global configuration is used.

## 8.4 Default Configuration

The default configuration is as follows.

```
# Default configuration helper_gcnv_model_targeted
step_config:

  helper_gcnv_model_targeted:
    path_ngs_mapping: ../ngs_mapping  # REQUIRED

    gcnv:
      path_uniquely_mapable_bed: null  # REQUIRED - path to BED file with uniquely␣
↪mappable regions.
      path_target_interval_list_mapping: []  # REQUIRED - define one or more set of␣
↪target intervals.
      # The following will match both the stock IDT library kit and the ones
      # with spike-ins seen from Yale genomics.  The path above would be
      # mapped to the name "default".
      # - name: IDT_xGen_V1_0
      #   pattern: "xGen Exome Research Panel V1\\.0*"
      #   path: "path/to/targets.bed"
```

CHAPTER

NINE

# GERMLINE BUILD WGS GCNV MODEL

Implementation of the `helper_gcnv_model_wgs` step

The `helper_gcnv_model_wgs` step takes as the input the results of the `ngs_mapping` step (aligned germline reads) and builds a model that can be used by GATK4 gCNV. Important: the workflow assumes that all samples in the cohort use the same library kit and all are WGS.

## 9.1 Step Input

The step uses Snakemake sub workflows for the result of the `ngs_mapping` (aligned reads BAM files).

## 9.2 Step Output

All donors will be used to generate the two parts of the required gCNV model, specifically: `ploidy-model` and `cnv_calls-model`. Both are required to execute gCNV in CASE mode.

For example, the relevant directories might look as follows:

```
work/
+-- bwa.gcnv_contig_ploidy.default
    `-- out
        `-- bwa.gcnv_contig_ploidy.default
            |-- SAMPLE_0
            |   |-- contig_ploidy.tsv
            |   |-- global_read_depth.tsv
            |   |-- mu_psi_s_log__.tsv
            |   |-- sample_name.txt
            |   `-- std_psi_s_log__.tsv
            |-- [...]
            `-- bwa.gcnv_contig_ploidy.default
                `-- ploidy-model
                    |-- contig_ploidy_prior.tsv
                    |-- gcnvkernel_version.json
                    |-- interval_list.tsv
                    |-- mu_mean_bias_j_lowerbound__.tsv
                    |-- mu_psi_j_log__.tsv
                    |-- ploidy_config.json
                    |-- std_mean_bias_j_lowerbound__.tsv
                    `-- std_psi_j_log__.tsv
```

(continues on next page)

(continued from previous page)

```
+-- bwa.gcnv_call_cnvs.default.***_of_***
    `-- out
        `-- bwa.gcnv_call_cnvs.default.***_of_***
            |-- cnv_calls-calls
            |   |-- SAMPLE_0
            |       `-- [...]
            |   |-- [...]
            |-- cnv_calls-model
            |  |-- denoising_config.json
            |  |-- gcnvkernel_version.json
            |  |-- interval_list.tsv
            |  |-- log_q_tau_tk.tsv
            |  |-- mu_W_tu.tsv
            |  |-- mu_ard_u_log__.tsv
            |  |-- mu_log_mean_bias_t.tsv
            |  |-- mu_psi_t_log__.tsv
            |  |-- std_W_tu.tsv
            |  |-- std_ard_u_log__.tsv
            |  |-- std_log_mean_bias_t.tsv
            |  `-- std_psi_t_log__.tsv
            `-- cnv_calls-tracking
                `-- [...]
```

## 9.3 Global Configuration

- At the moment, no global configuration is used.

## 9.4 Default Configuration

The default configuration is as follows.

```
# Default configuration helper_gcnv_model_wgs
step_config:
  helper_gcnv_model_wgs:
    path_ngs_mapping: ../ngs_mapping  # REQUIRED

    gcnv:
      # Path to BED file with uniquely mappable regions.
      path_uniquely_mapable_bed: null  # REQUIRED
```

CHAPTER

# TEN

# HLA TYPING

Implementation of the `hla_typing` step

The hla_typing step allows for the HLA typing from NGS read data (WGS, targeted DNA sequencing, or RNA-seq).

## 10.1 Step Input

Gene fusion calling starts at the raw RNA-seq reads. Thus, the input is very similar to one of *ngs_mapping step*.

See *Step Input* for more information.

## 10.2 Step Output

HLA typing will be performed for all NGS libraries in all sample sheets. For each combination of HLA typer and library, a directory `{hla_typer}.{lib_name}-{lib_pk}/out` will be created. Therein, the following files will be created:

- `{hla_typer}.{lib_name}-{lib_pk}.txt`
- `{hla_typer}.{lib_name}-{lib_pk}.txt.md5`

For example, it might look as follows for the example from above:

```
output/
+-- optitype.P001-N1-DNA1-WES1-4
|   `-- out
|       |-- optitype.P001-N1-DNA1-WES1-4.txt
|       `-- optitype.P001-N1-DNA1-WES1-4.txt.md5
[...]
```

## 10.3 Default Configuration

The default configuration is as follows.

```
# Default configuration ngs_mapping
step_config:
  hla_typing:
    path_ngs_mapping: ../ngs_mapping
    path_link_in: ""   # OPTIONAL Override data set configuration search paths for FASTQ␣
↪files
```

(continues on next page)

(continued from previous page)

```
  tools: [optitype]  # REQUIRED - available: 'optitype' and 'arcashla'
  optitype:
    max_reads: 5000  # suggestion by OptiType author
    num_mapping_threads: 4
  arcashla:
    mapper: star
```

## 10.4 Available HLA Typing Tools

The following HLA typing tools are currently available

- "optitype"
- "arcashla"

CHAPTER

# ELEVEN

# IGV SESSION GENERATION

Implementation of the `igv_session_generation` step

This step takes as the input the output of the following steps and generates an IGV session XML file that displays the results as genome tracks:

- `ngs_mapping`
- `variant_annotation` or `variant_calling`

## 11.1 Step Input

The IGV session generation step takes as the input of the following CUBI pipeline steps:

- `ngs_mapping`
- `variant_annotation` or `variant_calling`

## 11.2 Step Output

## 11.3 Global Configuration

## 11.4 Default Configuration

The default configuration is as follows.

```
# Default configuration igv_session_generation
step_config:
  igv_session_generation:
    path_ngs_mapping: ../ngs_mapping
    # One of the following must be given!
    path_variant_phasing: ''
    path_variant_annotation: ''
    path_variant_calling: ''
    tools_ngs_mapping: [] # defaults to ngs_mapping tool
    tools_variant_calling: [] # defaults to variant_annotation tool
```

## 11.5 Reports

Currently, no reports are generated.

CHAPTER

TWELVE

# NGS DATA QC

Implementation of the `ngs_data_qc` step

## 12.1 Default Configuration

The default configuration is as follows.

```
# Default configuration ngs_mapping
step_config:
  ngs_data_qc:
    path_link_in: ""  # OPTIONAL Override data set configuration search paths for FASTQ␣
↪files
    tools: [fastqc, picard]  # REQUIRED - available: 'fastqc' & 'picard' (for QC on bam␣
↪files)
    picard:
      path_ngs_mapping: ../ngs_mapping  # REQUIRED
      path_to_baits: ""                 # Required when CollectHsMetrics is among the␣
↪programs
      path_to_targets: ""               # When missing, same as baits
      bait_name: ""                     # Exon enrichment kit name (optional)
      programs: []  # Available metrics:
                    # * Generic metrics [* grouped into CollectMultipleMetrics]
      #                 - CollectAlignmentSummaryMetrics      *
      #                 - CollectBaseDistributionByCycle      *
      #                 - CollectGcBiasMetrics                *
      #                 - CollectInsertSizeMetrics            *
      #                 - CollectJumpingLibraryMetrics
      #                 - CollectOxoGMetrics
      #                 - CollectQualityYieldMetrics          *
      #                 - CollectSequencingArtifactMetrics    *
      #                 - EstimateLibraryComplexity
      #                 - MeanQualityByCycle                  *
      #                 - QualityScoreDistribution            *
                    # * WGS-specific metrics
      #                 - CollectRawWgsMetrics
      #                 - CollectWgsMetrics
      #                 - CollectWgsMetricsWithNonZeroCoverage
                    # * Other assay-specific metrics
      #                 - CollectHsMetrics                    Whole Exome Sequencing
      #                 - CollectTargetedPcrMetrics           Panel sequencing
```

(continues on next page)

(continued from previous page)

```
      #                 - CollectRnaSeqMetrics                mRNA sequencing, not␣
↪implemented yet
      #                 - CollectRbsMetrics                   bi-sulfite sequencing, not␣
↪implemented yet
```

CHAPTER

# THIRTEEN

# NGS MAPPING

Implementation of the `ngs_mapping` step

The ngs_mapping step allows for the alignment of NGS data with standard read mappers, such as BWA for DNA data and STAR for RNA data. Also, it provides functionality to compute post-alignment statistics, such as the coverage of target (e.g., exome or panel) regions.

There is a distinction made between "normal" DNA reads (short reads from Illumina) and "long" DNA reads, such as PacBio/Oxford Nanopore. Again, the NGS mapping step will perform alignment of all NGS libraries.

The precise actions that are performed in the alignment are defined by the wrappers (e.g., the `bwa` or `star`) wrappers. Generally, this includes converting into BAM format, sorting by coordinate, an indexing using a BAI file. For short reads, this can include marking of duplicates using Samblaster and depends on the actual configuration (see below for the default configuration).

## 13.1 Properties

overall stability

**stable**

applicable to

germline and somatic read alignment

generally applicable to

short and long read DNA and RNA sequencing

## 13.2 Step Input

For each library defined in all sample sheets, the instances of this step will search for the input files according to the configuration. The found read files will be linked into `work/input_links/{library_name}` (status quo, not a output path, thus path not guaranteed to be stable between minor versions).

This is different to the other steps that use the output of previous steps for their input.

### 13.2.1 Data Set Configuration

Consider the following data set definition from the main configuration file.

```
data_sets:
  first_batch:
    file: 01_first_batch.json
    search_patterns:
      # Note that currently only "left" and "right" key known
      - {'left': '*/L???/*_R1.fastq.gz', 'right': '*/L???/*_R2.fastq.gz'}
    search_paths: ['../input/01_first_batch']
```

Here, the data set `first_batch` is defined. The sample sheet file is named `01_first_batch.json` and looked for in the relative path to the configuration file. The input search will be start in the (one, but could be more than one) path `../input/01_first_batch` (relative to the directory containing the configuration file). The sample sheet provides a `folderName` `extraInfo` entry for each NGS library. This folder name is searched for (e.g., `P001-N1-DNA1-WES`). Once such a folder is found, the patterns in the values of the dict `search_patterns` are used for locating the paths of the actual files.

Currently, the only supported keys in the `search_patterns` dict are `"left"` and `"right""` (the lattern can be omitted when only searching for single-end reads).

Consider the following example:

```
../input/
`-- 01_first_batch
    |-- P001-N1-DNA1-WES1
    |   `-- 42KF5AAXX
    |       `-- L001
    |           |-- P001-N1-DNA1-WES1_R1.fastq.gz
    |           |-- P001-N1-DNA1-WES1_R1.fastq.gz.md5
    |           |-- P001-N1-DNA1-WES1_R2.fastq.gz
    |           `-- P001-N1-DNA1-WES1_R2.fastq.gz.md5
    [...]
```

Here, the folder `01_first_batch` will be searched for a directory named `P001-N1-DNA1-WES`. After finding, the relative paths `42KF5AAXX/L001/P001-N1-DNA1-WES1_R1.fastq.gz` and `42KF5AAXX/L001/P001-N1-DNA1-WES1_R2.fastq.gz` will be found and used for the left/right parts of a paired read set.

### Mixing Single-End and Paired-End Reads

By default, it is checked that for each `search_pattern`, the same number of matching files has to be found, otherwise directories are ignored. The reason is to reduce the number of possible errors when linking in files. You can change this behaviour by specifying `mixed_se_pe: True` in the data set information. Then, it will be allowed to have the matches for the `right` entry to be empty. However, you will need to consistently have either SE or PE data for each library; it is allowed to mix SE and PE libraries within one project but not to have PE and SE data for one library.

## 13.3 Step Output

For each NGS library with name `library_name` and each read mapper `mapper` that the library has been aligned with, the pipeline step will create a directory `output/{mapper}.{library_name}/out` with symlinks of the following names to the resulting sorted BAM files with corresponding BAI and MD5 files.

- `{mapper}.{library_name}.bam`
- `{mapper}.{library_name}.bam.bai`
- `{mapper}.{library_name}.bam.md5`
- `{mapper}.{library_name}.bam.bai.md5`

In addition, several tools are used to automatically generate reports based on the BAM and BAI files. See the Reports section below for more details

The BAM files are only postprocessed if configured so.

---

**Note:** In contrast to other pipeline steps, the NGS mapping step will also generate the BAM files for the background data sets as there are currently problems with Snakemake sub workflows and input functions.

---

## 13.4 Global Configuration

- `static_data_config/reference/path` must be set appropriately

## 13.5 Default Configuration

The default configuration is as follows.

```
step_config:
  ngs_mapping:
    # Aligners to use for the different NGS library types
    tools:
      dna: []      # Required if DNA analysis; otherwise, leave empty. Example: 'bwa'.
      rna: []      # Required if RNA analysis; otherwise, leave empty. Example: 'star'.
      dna_long: [] # Required if long-read mapper used; otherwise, leave empty. Example:
↪'minimap2'.
    path_link_in: ""   # OPTIONAL Override data set configuration search paths for FASTQ␣
↪files
    # Thresholds for targeted sequencing coverage QC.  Enabled by specifying
    # the path_arget_regions setting above
```

(continues on next page)

(continued from previous page)

```
    target_coverage_report:
      # Mapping from enrichment kit to target region BED file, for either computing per--
↪target
      # region coverage or selecting targeted exons.
      #
      # The following will match both the stock IDT library kit and the ones
      # with spike-ins seen fromr Yale genomics.  The path above would be
      # mapped to the name "default".
      # - name: IDT_xGen_V1_0
      #   pattern: "xGen Exome Research Panel V1\\.0*"
      #   path: "path/to/targets.bed"
      path_target_interval_list_mapping: []
      # Maximal/minimal/warning coverage
      max_coverage: 200
      min_cov_warning: 20  # >= 20x for WARNING
      min_cov_ok: 50  # >= 50x for OK
      detailed_reporting: false  # per-exon details (cannot go into multiqc)
    # Depth of coverage collection, mainly useful for genomes.
    bam_collect_doc:
      enabled: false
      window_length: 1000
    # Compute fingerprints with ngs-chew
    ngs_chew_fingerprint:
      enabled: true
    # Configuration for BWA
    bwa:
      path_index: REQUIRED # Required if listed in ngs_mapping.tools.dna; otherwise, can␣
↪be removed.
      num_threads_align: 16
      num_threads_trimming: 8
      num_threads_bam_view: 4
      num_threads_bam_sort: 4
      memory_bam_sort: 4G
      trim_adapters: false
      mask_duplicates: true
      split_as_secondary: false  # -M flag
    # Configuration for BWA-MEM2
    bwa_mem2:
      path_index: REQUIRED # Required if listed in ngs_mapping.tools.dna; otherwise, can␣
↪be removed.
      bwa_mode: auto  # in ['auto', 'bwa-aln', 'bwa-mem']
      num_threads_align: 16
      num_threads_trimming: 8
      num_threads_bam_view: 4
      num_threads_bam_sort: 4
      memory_bam_sort: 4G
      trim_adapters: false
      mask_duplicates: true
      split_as_secondary: true  # -M flag
    # Configuration for STAR
    star:
      path_index: REQUIRED # Required if listed in ngs_mapping.tools.rna; otherwise, can␣
↪be removed.
```

(continues on next page)

(continued from previous page)

```
      path_features: ""    # Required for computing gene counts
      num_threads_align: 16
      num_threads_trimming: 8
      num_threads_bam_view: 4
      num_threads_bam_sort: 4
      memory_bam_sort: 4G
      genome_load: NoSharedMemory
      raw_star_options: ''
      align_intron_max: 1000000                # ENCODE option
      align_intron_min: 20                     # ENCODE option
      align_mates_gap_max: 1000000             # ENCODE option
      align_sjdb_overhang_min: 1               # ENCODE option
      align_sj_overhang_min: 8                 # ENCODE option
      out_filter_mismatch_n_max: 999           # ENCODE option
      out_filter_mismatch_n_over_l_max: 0.04   # ENCODE option
      out_filter_multimap_n_max: 20            # ENCODE option
      out_filter_type: BySJout                 # ENCODE option
      out_filter_intron_motifs: None     # or for cufflinks: RemoveNoncanonical
      out_sam_strand_field: None         # or for cufflinks: intronMotif
      transcriptome: false               # true to output transcript coordinate bam for␣
↪RSEM
      trim_adapters: false
      mask_duplicates: false
      include_unmapped: true
    strandedness:
      path_exon_bed: REQUIRED   # Location of usually highly expressed genes. Known␣
↪protein coding genes is a good choice
      strand: -1                # -1: unknown value, use infer_, 0: unstranded, 1:␣
↪forward, 2: reverse (from featurecounts)
      threshold: 0.85           # Minimum proportion of reads mapped to forward/reverse␣
↪direction to call the protocol
    # Configuration for Minimap2
    minimap2:
      mapping_threads: 16
```

## 13.6 Available Read Mappers

The following read mappers are available for the alignment of DNA-seq and RNA-seq reads.

- **(short/Illumina) DNA**
  - "bwa"
  - "bwa_mem2"
  - "external"
- **(short/Illumina) RNA-seq**
  - "star"
  - "external"
- **(long/PacBio/Nanopore) DNA**

```
- "minimap2"
- "external"
```

## 13.7 Notes on *STAR* mapper configuration

Recent versions of *STAR* offer the possibility to output gene counts and alignments of reads on the transcritpome, rather than on the genome.

In both cases, this requires that *STAR* is aware of the genes, transcripts, exon & introns features. These can be provided either during the indexing stage, or with recent versions, during mapping.

The configuration provides the possibility to pass to *STAR* the location of a *gtf* file describing the features. This removes the need to include gene models into the generation of indices, so that the user can select the gene models (either from ENSEMBL or GENCODE, for example).

When the configuration option *path_features* is set, the step will output a table of expression counts for all genes, in *output/star.{library_name}/out/star.{library_name}.GeneCounts.tab*.

If the configuration option *transcriptome* is set to *true*, the step will output a bam file of reads mapped to the transcriptome (*output/stat.{library_name}/out/star.{library_name}.toTranscriptome.bam*). *STAR* will rely on the *path_features* configuration option, or on the gene models embedded in the indices to generate the mappings. If both are absent, the step will fail. Note that the mappings to the transcriptome will not be indexes using *samtools index*, because the absence of the positional mappings.

## 13.8 Reports

Currently, the following reports are generated based on the BAM and BAI file output by this step.

**General Alignment Statistics (.txt)** The tools `samtools bamstats`, `samtools flagstats` and `samtools idxstats` are always called by default, and are linked out into the `output/{mapper}.{library_name}/report/bam_qc` directory. The file names for these reports (and their MD5s) use the following naming convention:

- `{mapper}.{library_name}.bamstats.txt`
- `{mapper}.{library_name}.flagstats.txt`
- `{mapper}.{library_name}.idxstats.txt`
- `{mapper}.{library_name}.bamstats.txt.md5`
- `{mapper}.{library_name}.flagstats.txt.md5`
- `{mapper}.{library_name}.idxstats.txt.md5`

For example, it will look as follows for the example bam files shown above:

```
output/
+-- bwa.P001-N1-DNA1-WES1
|   |-- out
|   |   |-- bwa.P001-N1-DNA1-WES1.bam
|   |   |-- bwa.P001-N1-DNA1-WES1.bam.bai
|   |   |-- bwa.P001-N1-DNA1-WES1.bam.bai.md5
|   |   `-- bwa.P001-N1-DNA1-WES1.bam.md5
|   `-- report
```

(continues on next page)

(continued from previous page)

```
|       `-- bam_qc
|           |-- bwa.P001-N1-DNA1-WES1.bam.bamstats.txt
|           |-- bwa.P001-N1-DNA1-WES1.bam.bamstats.txt.md5
|           |-- bwa.P001-N1-DNA1-WES1.bam.flagstats.txt
|           |-- bwa.P001-N1-DNA1-WES1.bam.flagstats.txt.md5
|           |-- bwa.P001-N1-DNA1-WES1.bam.idxstats.txt
|           `-- bwa.P001-N1-DNA1-WES1.bam.idxstats.txt.md5
[...]
```

**Target Coverage Report (.txt)** If `ngs_mapping/path_target_regions` is set to a BED file with the target regions (either capture regions of capture kits in the case of targeted sequencing or exons for WES/WGS sequencing) a target coverage report is generated and linked out into the `output/{mapper}.{library_name}/report/cov_qc` directory. The file names for these reports (and their MD5s) use the following naming convention:

- `{mapper}.{library_name}.txt`
- `{mapper}.{library_name}.txt.md5`

For example, it will look as follows for the example bam files shown above:

```
output/
+-- bwa.P001-N1-DNA1-WES1
|   `-- report
|       |-- bam_qc
|       |   [...]
|       `-- cov_qc
|           |-- bwa.P001-N1-DNA1-WES1.txt
|           `-- bwa.P001-N1-DNA1-WES1.txt.md5
[...]
```

**Genome-wide Coverage Count (.bed.gz)** If `ngs_mapping/compute_coverage_bed` to be set to `true` a report is generated that gives the depth at each base of the genome. (note: currently this report only appears in `work/` and is not yet linked out into the `output/` directory).

(TODO: add file name rules and example)

```
work/
+-- bwa.P001-N1-DNA1-WES1
|   `-- report
|       `-- bam_qc
|           |-- bwa.P001-N1-DNA1-WES1.bam.bamstats.d
|           |   |-- acgt-cycles.gp
|           |   |-- acgt-cycles.png
|           |   |-- coverage.gp
|           |   |-- coverage.png
|           |   |-- gc-content.gp
|           |   |-- gc-content.png
|           |   |-- gc-depth.gp
|           |   |-- gc-depth.png
|           |   |-- indel-cycles.gp
|           |   |-- indel-cycles.png
|           |   |-- indel-dist.gp
|           |   |-- indel-dist.png
|           |   |-- index.html
```

(continues on next page)

(continued from previous page)

```
|           |   |-- insert-size.gp
|           |   |-- insert-size.png
|           |   |-- quals2.gp
|           |   |-- quals2.png
|           |   |-- quals3.gp
|           |   |-- quals3.png
|           |   |-- quals.gp
|           |   |-- quals-hm.gp
|           |   |-- quals-hm.png
|           |   `-- quals.png
|           [...]
[...]
```

CHAPTER

FOURTEEN

# NGS SANITY CHECKING

Implementation of the `ngs_sanity_checking` step

Perform sanity checking from mapped reads for germline sample sheets, optionally taking the result of `hla_typing` into consideration.

**Note:** Status: not implemented yet

## 14.1 Step Input

**Note:** TODO

## 14.2 Step Output

**Note:** TODO

## 14.3 Default Configuration

The default configuration is as follows.

```
# Default configuration ngs_sanity_checking
step_config:
  ngs_sanity_checking:
    path_ngs_mapping: ../path_ngs_mapping  # REQUIRED
    path_hla_typing: ../path_hla_typing    # OPTIONAl
    check_hla: true
```

CHAPTER

FIFTEEN

# SOMATIC GENE FUSION CALLING

Implementation of the `somatic_gene_fusion_calling` step

The somatic_gene_fusion calling step allows for the detection of gene fusions from RNA-seq data in cancer. The wrapped tools start at the raw RNA-seq reads and generate filtered lists of predicted gene fusions.

## 15.1 Step Input

Gene fusion calling starts at the raw RNA-seq reads. Thus, the input is very similar to one of *ngs_mapping step*.

See *Step Input* for more information.

## 15.2 Step Output

**Note:** TODO

## 15.3 Default Configuration

The default configuration is as follows.

```
step_config:
  somatic_gene_fusion_calling:
    path_link_in: ""  # OPTIONAL Override data set configuration search paths for FASTQ␣
↪files
    tools: ['fusioncatcher', 'jaffa', 'arriba', 'defuse', 'hera', 'pizzly', 'star_fusion
↪']  # REQUIRED, available: 'fusioncatcher', 'jaffa', 'arriba', 'defuse', 'hera', 'pizzly',
↪'star_fusion'.
    fusioncatcher:
      data_dir: REQUIRED   # REQUIRED
      configuration: null  # optional
      num_threads: 16
    pizzly:
      kallisto_index: REQUIRED    # REQUIRED
      transcripts_fasta: REQUIRED # REQUIRED
      annotations_gtf: REQUIRED       # REQUIRED
```

(continues on next page)

(continued from previous page)

```
      kmer_size: 31
    hera:
      path_index: REQUIRED    # REQUIRED
      path_genome: REQUIRED   # REQUIRED
    star_fusion:
      path_ctat_resource_lib: REQUIRED
    defuse:
      path_dataset_directory: REQUIRED
    arriba:
      path_index: REQUIRED        # REQUIRED  STAR path index (preferably 2.7.10 or later)
      features: REQUIRED          # REQUIRED  Gene features (for ex. ENCODE or ENSEMBL)␣
↪in gtf format
      blacklist: ""               # optional (provided in the arriba distribution, see /
↪fast/work/groups/cubi/projects/biotools/static_data/app_support/arriba/v2.3.0)
      known_fusions: ""           # optional
      tags: ""                    # optional (can be set to the same path as known_
↪fusions)
      structural_variants: ""     # optional
      protein_domains: ""         # optional
      num_threads: 8
      trim_adapters: false
      num_threads_trimming: 2
      star_parameters:
      - " --outFilterMultimapNmax 50"
      - " --peOverlapNbasesMin 10"
      - " --alignSplicedMateMapLminOverLmate 0.5"
      - " --alignSJstitchMismatchNmax 5 -1 5 5"
      - " --chimSegmentMin 10"
      - " --chimOutType WithinBAM HardClip"
      - " --chimJunctionOverhangMin 10"
      - " --chimScoreDropMax 30"
      - " --chimScoreJunctionNonGTAG 0"
      - " --chimScoreSeparation 1"
      - " --chimSegmentReadGapMax 3"
      - " --chimMultimapNmax 50"
```

## 15.4 Available Gene Fusion Callers

- fusioncatcher

CHAPTER

SIXTEEN

# SOMATIC NEOEPITOPE PREDICTION

Implementation of the `somatic_neoepitope_prediction` step

The somatic_neoepitope_prediction step allows for the prediction of neoepitopes from somatic (small) variant calling results and a transcript database such as ENSEMBL. Further, the step allows for the binding prediction to a given set of HLA alleles.

**Note:** Status: not implemented yet

## 16.1 Step Input

**Note:** TODO

## 16.2 Step Output

**Note:** TODO

## 16.3 Default Configuration

The default configuration is as follows.

```
step_config:
  somatic_neoepitope_prediction:
    path_somatic_variant_calling: REQUIRED  # REQUIRED
```

CHAPTER

SEVENTEEN

# SOMATIC NGS SANITY CHECKING

Implementation of the `somatic_ngs_sanity_checking` step

Perform sanity checking from mapped reads for cancer sample sheets, optionally taking the result of `hla_typing` into consideration.

**Note:** Status: not implemented yet

## 17.1 Step Input

**Note:** TODO

## 17.2 Step Output

**Note:** TODO

## 17.3 Default Configuration

The default configuration is as follows.

```
# Default configuration somatic_ngs_sanity_checking
step_config:
  somatic_ngs_sanity_checking:
    path_ngs_mapping: ../path_ngs_mapping  # REQUIRED
    path_hla_typing: ../path_hla_typing    # OPTIONAl
    check_hla: true
```

CHAPTER

EIGHTEEN

# SOMATIC PURITY & PLOIDY ESTIMATE

Implementation of purity and ploidy checking for somatic NGS samples

## 18.1 Default Configuration

The default configuration is as follows.

```
step_config:
  somatic_purity_ploidy_estimate:
    tools: ['ascat']  # REQUIRED - available: 'ascat'
    tool_cnv_calling: cnvetti
    # Configuration with read mapper and path to mapping output. Will use this
    # for generating a pileup using samtools for obtaining the b allele
    # fraction and computing coverage.
    tool_ngs_mapping: bwa
    path_ngs_mapping: ../ngs_mapping
    # Configuration of ASCAT method.
    ascat:
      # BED file with loci for B allele frequency.
      b_af_loci: REQUIRED  # REQUIRED
```

CHAPTER

# NINETEEN

# SOMATIC TARGETED SEQ. CNV CALLING

Implementation of the `somatic_target_seq_cnv_calling` step

This step allows for the detection of CNV events for cancer samples from targeted sequenced (e.g., exomes or large panels). The wrapped tools start from the aligned reads (thus off `ngs_mapping`) and generate CNV calls for somatic variants.

The wrapped tools implement different strategies. Some work "reference free" and just use the somatic BAM files for their input, some work in "matched cancer normal mode" and need the cancer and normal BAM files, others again need both normal and cancer BAM files, and additionally a set of non-cancer BAM files for their background.

## 19.1 Step Input

Gene somatic CNV calling for targeted sequencing starts off the aligned reads, i.e., `ngs_mapping`.

## 19.2 Step Output

Generally, the following links are generated to `output/`.

**Note:** Tool-Specific Output

As the only integrated tool is cnvkit at the moment, the output is very tailored to the result of this tool. In the future, this section will contain "common" output and tool-specific output sub sections.

- **{mapper}.cnvkit.{lib_name}-{lib_pk}/out/**
  - {mapper}.cnvkit.{lib_name}-{lib_pk}.bed
  - {mapper}.cnvkit.{lib_name}-{lib_pk}.seg
  - {mapper}.cnvkit.{lib_name}-{lib_pk}.vcf.gz
  - {mapper}.cnvkit.{lib_name}-{lib_pk}.vcf.gz.tbi
- **{mapper}.cnvkit.{lib_name}-{lib_pk}/report**
  - {mapper}.cnvkit.{lib_name}-{lib_pk}.diagram.pdf
  - {mapper}.cnvkit.{lib_name}-{lib_pk}.scatter.pdf
  - {mapper}.cnvkit.{lib_name}-{lib_pk}.heatmap.pdf
  - {mapper}.cnvkit.{lib_name}-{lib_pk}.heatmap.chr1.pdf

        - ...
        - {mapper}.cnvkit.{lib_name}-{lib_pk}.scatter.chrX.pdf

- **{mapper}.cnvkit.{lib_name}-{lib_pk}/report**
        - {mapper}.cnvkit.{lib_name}-{lib_pk}.breaks.txt
        - {mapper}.cnvkit.{lib_name}-{lib_pk}.genemetrics.txt
        - {mapper}.cnvkit.{lib_name}-{lib_pk}.gender.txt
        - {mapper}.cnvkit.{lib_name}-{lib_pk}.metrics.txt
        - {mapper}.cnvkit.{lib_name}-{lib_pk}.segmetrics.txt

For example:

```
output/
|-- bwa.cnvkit.P001-T1-DNA1-WES1-000007
|   `-- out
|       |-- bwa.cnvkit.P001-T1-DNA1-WES1-000007.bed
|       |-- bwa.cnvkit.P001-T1-DNA1-WES1-000007.seg
|       `-- bwa.cnvkit.P001-T1-DNA1-WES1-000007.vcf
|-- bwa.cnvkit.P002-T1-DNA1-WES1-000016
|   `-- report
|       |-- bwa.cnvkit.P002-T1-DNA1-WES1-000016.diagram.pdf
|       |-- bwa.cnvkit.P002-T1-DNA1-WES1-000016.heatmap.pdf
|       |-- bwa.cnvkit.P002-T1-DNA1-WES1-000016.scatter.pdf
|       |-- bwa.cnvkit.P002-T1-DNA1-WES1-000016.heatmap.chr1.pdf
|       |-- ...
|       `-- bwa.cnvkit.P002-T1-DNA1-WES1-000016.scatter.chrX.pdf
|-- bwa.cnvkit.P002-T1-DNA1-WES1-000016
|   `-- report
|       |-- bwa.cnvkit.P002-T1-DNA1-WES1-000016.breaks.txt
|       |-- bwa.cnvkit.P002-T1-DNA1-WES1-000016.genemetrics.txt
|       |-- bwa.cnvkit.P002-T1-DNA1-WES1-000016.gender.txt
|       |-- bwa.cnvkit.P002-T1-DNA1-WES1-000016.metrics.txt
|       `-- bwa.cnvkit.P002-T1-DNA1-WES1-000016.segmetrics.txt
[...]
```

Note that tool `cnvetti` doesn't follow the snappy convention above: the tool name is followed by an underscore & the action, where the action is one of `coverage`, `segment` and `postprocess`. For example, the output directory would contain a directory named `bwa.cnvetti_coverage.P002-T1-DNA1-WES1-000016`.

## 19.3 Default Configuration

The default configuration is as follows.

```
# Default configuration somatic_targeted_seq_cnv_calling
step_config:
  somatic_targeted_seq_cnv_calling:
    tools: ['cnvkit']  # REQUIRED - available: 'cnvkit', 'copywriter', 'cnvetti_on_target',␣
↪and 'cnvetti_off_target'
    path_ngs_mapping: ../ngs_mapping  # REQUIRED
    cnvkit:
```

(continues on next page)

(continued from previous page)

```
      path_target: REQUIRED             # Usually ../panel_of_normals/output/cnvkit.
↪target/out/cnvkit.target.bed
      path_antitarget: REQUIRED         # Usually ../panel_of_normals/output/cnvkit.
↪antitarget/out/cnvkit.antitarget.bed
      path_panel_of_normals: REQUIRED   # Usually ../panel_of_normals/output/{mapper}.
↪cnvkit.create_panel/out/{mapper}.cnvkit.panel_of_normals.cnn
      plot: True                        # Generate plots (very slow)
      min_mapq: 0                       # [coverage] Mininum mapping quality score to␣
↪count a read for coverage depth
      count: False                      # [coverage] Alternative couting algorithm
      gc_correction: True               # [fix] Use GC correction
      edge_correction: True             # [fix] Use edge correction
      rmask_correction: True            # [fix] Use rmask correction
      # BCBIO uses
      # seg_method: haar
      # seg_threshold: 0.0001
      # -- OR
      # seg_method: cbs
      # seg_threshold: 0.000001
      segmentation_method: cbs          # [segment] One of cbs, flasso, haar, hmm, hmm-
↪tumor, hmm-germline, none
      segmentation_threshold: 0.000001  # [segment] Significance threshold (hmm methods:␣
↪smoothing window size)
      drop_low_coverage: False          # [segment, call, genemetrics] Drop very low␣
↪coverage bins
      drop_outliers: 10                 # [segment] Drop outlier bins (0 for no outlier␣
↪filtering)
      smooth_cbs: True                  # [segment] Additional smoothing of CBS␣
↪segmentation (WARNING- not the default value)
      center: ""                        # [call] Either one of mean, median, mode,␣
↪biweight, or a constant log2 ratio value.
      filter: ampdel                    # [call] One of ampdel, cn, ci, sem (merging␣
↪segments flagged with the specified filter), "" for no filtering
      calling_method: threshold         # [call] One of threshold, clonal, none
      call_thresholds: "-1.1,-0.25,0.2,0.7" # [call] Thresholds for calling integer copy␣
↪number
      ploidy: 2                         # [call] Ploidy of sample cells
      purity: 0                         # [call] Estimated tumor cell fraction (0 for␣
↪discarding tumor cell purity)
      gender: ""                        # [call, diagram] Specify the chromosomal sex of␣
↪all given samples as male or female. Guess when missing
      male_reference: False             # [call, diagram] Create male reference
      diagram_threshold: 0.5            # [diagram] Copy number change threshold to␣
↪label genes
      diagram_min_probes: 3             # [diagram] Min number of covered probes to␣
↪label genes
      shift_xy: True                    # [diagram] Shift X & Y chromosomes according to␣
↪sample sex
      breaks_min_probes: 1              # [breaks] Min number of covered probes for a␣
↪break inside the gene
      genemetrics_min_probes: 3         # [genemetrics] Min number of covered probes to␣
↪consider a gene
```

(continues on next page)

(continued from previous page)

```
      genemetrics_threshold: 0.2        # [genemetrics] Min abs log2 change to consider␣
↪a gene
      genemetrics_alpha: 0.05           # [genemetrics] Significance cutoff
      genemetrics_bootstrap: 100        # [genemetrics] Number of bootstraps
      segmetrics_alpha: 0.05            # [segmetrics] Significance cutoff
      segmetrics_bootstrap: 100         # [segmetrics] Number of bootstraps
      smooth_bootstrap: False           # [segmetrics] Smooth bootstrap results
    copywriter:
      path_target_regions: REQUIRED # REQUIRED
      bin_size: 20000 # TODO: make actually configurable
      plot_genes: REQUIRED  # Path to civic annotation
      genome: hg19          # Could be hg38 (consider setting prefix to 'chr' when using␣
↪GRCh38.v1)
      features: EnsDb.Hsapiens.v75::EnsDb.Hsapiens.v75
      prefix: ''
      nThread: 8
    cnvetti_on_target:
      path_target_regions: REQUIRED # REQUIRED
    cnvetti_off_target:
      path_target_regions: REQUIRED # REQUIRED
      window_length: 20000
```

## 19.4 Available Somatic Targeted CNV Caller

- cnvkit

CHAPTER

TWENTY

# SOMATIC VARIANT ANNOTATION

Implementation of the `somatic_variant_annotation` step

The `somatic_variant_annotation` step takes as the input the results of the `somatic_variant_calling` step (bgzip-ed and indexed VCF files) and performs annotation of the somatic variants. The result are annotated versions of the somatic variant VCF files (again bgzip-ed and indexed VCF files).

## 20.1 Step Input

The somatic variant annotation step uses Snakemake sub workflows for using the result of the `somatic_variant_calling` step.

The main assumption is that each VCF file contains the two matched normal and tumor samples.

## 20.2 Step Input

The variant annotation step uses Snakemake sub workflows for using the result of the `variant_calling` step.

## 20.3 Step Output

Users can annotate all genes & transcripts overlapping with the variant locus, or they can select one representative gene and transcript for annotation. In the latter case, the output vcf file will only contain one annotation per variant, while in the former case, there might be over 100 annotations for each variant.

The ordering of features driving the representative annotation choice is under user control. The default order is:

1. `biotype`: protein coding genes come first, it is unclear what is the order for other types of genes
2. `mane`: the MANE transcript is selected before other transcripts
3. `appris`: the APPRIS principal isoform is selected before alternates
4. `tsl`: Transcript Support Level values in increasing order
5. `ccds`: Transcripts with CCDS ids are selected before those without
6. `canonical`: ENSEMBL canonical transcripts are selected before the others
7. `rank`: VEP internal ranking is used
8. `length`: longer transcripts are preferred to shorter ones

This order is (hopefully) suitable for cBioPortal export, as well defined transcripts from protein-coding genes are selected when possible. However, it is recommended to check the full annotation for variants in or nearby disease-relevant genes.

All annotators generate a vcf with one annotation per transcript, and some annotators (only ENSEMBL's Variant Effect Predictor in the current implementation) can also produce another output containing all annotations. The single annotation vcf is named `<mapper>.<caller>.<annotator>.vcf.gz` and the full annotation output is named `<mapper>.<caller>.<annotator>.full.vcf.gz`

## 20.4 Global Configuration

TODO

## 20.5 Default Configuration

The default configuration is as follows.

```
# Default configuration variant_annotation
step_config:
  variant_annotation:
    path_variant_calling: ../variant_calling
    tools:
      - vep
    vep:
      # We will always run VEP in cache mode.  You have to provide the directory to the
      # cache to use (VEP would be ``~/.vep``).
      cache_dir: null # OPTIONAL
      # The cache version to use.  gnomAD v2 used 85, gnomAD v3.1 uses 101.
      cache_version: "85"
      # The assembly to use.  gnomAD v2 used "GRCh37", gnomAD v3.1 uses "GRCh38".
      assembly: "GRCh37"
      # The flag selecting the transcripts.  One of "gencode_basic", "refseq", and
↪"merged".
      tx_flag: "gencode_basic"
      # Number of threads to use with forking, set to 0 to disable forking.
      num_threads: 16
      # Additional flags.
      more_flags: "--af_gnomade --af_gnomadg"
      # The --buffer_size parameter
      buffer_size: 100000
```

## 20.6 Reports

Currently, no reports are generated.

CHAPTER

# TWENTYONE

# SOMATIC VARIANT CALLING

Implementation of the `somatic_variant_calling` step

The `somatic_variant_calling` step takes as the input the results of the `ngs_mapping` step (aligned reads in BAM format) and performs somatic variant calling. The result are variant files with somatic variants (bgzip-ed and indexed VCF files).

Usually, the somatic variant calling step is followed by the `somatic_variant_annotation` step.

## 21.1 Step Input

The somatic variant calling step uses Snakemake sub workflows for using the result of the `ngs_mapping` step.

## 21.2 Step Output

For each tumor DNA NGS library with name `lib_name`/key `lib_pk` and each read mapper `mapper` that the library has been aligned with, and the variant caller `var_caller`, the pipeline step will create a directory `output/{mapper}.{var_caller}.{lib_name}-{lib_pk}/out` with symlinks of the following names to the resulting VCF, TBI, and MD5 files.

- `{mapper}.{var_caller}.{lib_name}-{lib_pk}.vcf.gz`
- `{mapper}.{var_caller}.{lib_name}-{lib_pk}.vcf.gz.tbi`
- `{mapper}.{var_caller}.{lib_name}-{lib_pk}.vcf.gz.md5`
- `{mapper}.{var_caller}.{lib_name}-{lib_pk}.vcf.gz.tbi.md5`

For example, it might look as follows for the example from above:

```
output/
+-- bwa.mutect.P001-N1-DNA1-WES1-4
|   `-- out
|       |-- bwa.mutect.P001-N1-DNA1-WES1-4.vcf.gz
|       |-- bwa.mutect.P001-N1-DNA1-WES1-4.vcf.gz.tbi
|       |-- bwa.mutect.P001-N1-DNA1-WES1-4.vcf.gz.md5
|       `-- bwa.mutect.P001-N1-DNA1-WES1-4.vcf.gz.tbi.md5
[...]
```

Generally, these files will be unfiltered, i.e., contain low-quality variants and also variants flagged as being non-somatic.

## 21.3 Global Configuration

- If the somatic variant caller MuTect is used, then the global settings `static_data_config/dbsnp` and `static_data_config/cosmic` must be given as MuTect uses this in its algorithm.
- `static_data_config/reference/path` must be set appropriately

## 21.4 Default Configuration

The default configuration is as follows.

```
# Default configuration somatic_variant_calling
step_config:
  somatic_variant_calling:
    tools: ['mutect', 'scalpel']  # REQUIRED, examples: 'mutect' and 'scalpel'.
    path_ngs_mapping: ../ngs_mapping  # REQUIRED
    ignore_chroms:            # patterns of chromosome names to ignore
    - NC_007605    # herpes virus
    - hs37d5       # GRCh37 decoy
    - chrEBV       # Eppstein-Barr Virus
    - '*_decoy'    # decoy contig
    - 'HLA-*'      # HLA genes
    - 'GL000220.*' # Contig with problematic, repetitive DNA in GRCh37
    # Configuration for joint calling with samtools+bcftools.
    bcftools_joint:
      max_depth: 4000
      max_indel_depth: 4000
      window_length: 10000000
      num_threads: 16
    # Configuration for joint calling with Platypus.
    platypus_joint:
      split_complex_mnvs: true  # whether or not to split complex and MNV variants
      num_threads: 16
    # VCF annotation databases are given as mapping from name to
    #   {'file': '/path.vcf.gz',
    #    'info_tag': 'VCF_TAG',
    #    'description': 'VCF header description'}
    # Configuration for MuTect
    mutect:
      # Parallelization configuration
      num_cores: 2               # number of cores to use locally
      window_length: 3500000     # split input into windows of this size, each triggers␣
↪a job
      num_jobs: 500              # number of windows to process in parallel
      use_profile: true          # use Snakemake profile for parallel processing
      restart_times: 5           # number of times to re-launch jobs in case of failure
      max_jobs_per_second: 2     # throttling of job creation
      max_status_checks_per_second: 10   # throttling of status checks
      debug_trunc_tokens: 0      # truncation to first N tokens (0 for none)
      keep_tmpdir: never         # keep temporary directory, {always, never, onerror}
      job_mult_memory: 1         # memory multiplier
      job_mult_time: 1           # running time multiplier
```

(continues on next page)

(continued from previous page)

```
      merge_mult_memory: 1       # memory multiplier for merging
      merge_mult_time: 1         # running time multiplier for merging
    # Configuration for MuTect 2
    mutect2:
      panel_of_normals: ''       # Set path to panel of normals vcf if required
      germline_resource: ''      # Germline variants resource (same as panel of normals)
      common_variants: ''        # Common germline variants for contamination estimation
      extra_arguments: []        # List additional Mutect2 arguments
                                 # Each additional argument xust be in the form:
                                 # "--<argument name> <argument value>"
                                 # For example, to filter reads prior to calling & to
                                 # add annotations to the output vcf:
                                 # - "--read-filter CigarContainsNoNOperator"
                                 # - "--annotation AssemblyComplexity BaseQuality"
      # Parallelization configuration
      num_cores: 2               # number of cores to use locally
      window_length: 50000000    # split input into windows of this size, each triggers a␣
↪job
      num_jobs: 500              # number of windows to process in parallel
      use_profile: true          # use Snakemake profile for parallel processing
      restart_times: 5           # number of times to re-launch jobs in case of failure
      max_jobs_per_second: 2     # throttling of job creation
      max_status_checks_per_second: 10   # throttling of status checks
      debug_trunc_tokens: 0      # truncation to first N tokens (0 for none)
      keep_tmpdir: never         # keep temporary directory, {always, never, onerror}
      job_mult_memory: 1         # memory multiplier
      job_mult_time: 1           # running time multiplier
      merge_mult_memory: 1       # memory multiplier for merging
      merge_mult_time: 1         # running time multiplier for merging
    # Configuration for Scalpel
    scalpel:
      path_target_regions: REQUIRED  # REQUIRED
    # Configuration for strelka2
    strelka2:
      path_target_regions: ""    # For exomes: include a bgzipped bed file with tabix␣
↪index. That also triggers the --exome flag
    gatk_hc_joint:
      # Parallelization configuration
      num_cores: 2               # number of cores to use locally
      window_length: 50000000    # split input into windows of this size, each triggers a␣
↪job
      num_jobs: 500              # number of windows to process in parallel
      use_profile: true          # use Snakemake profile for parallel processing
      restart_times: 5           # number of times to re-launch jobs in case of failure
      max_jobs_per_second: 10    # throttling of job creation
      max_status_checks_per_second: 10   # throttling of status checks
      debug_trunc_tokens: 0      # truncation to first N tokens (0 for none)
      keep_tmpdir: never         # keep temporary directory, {always, never, onerror}
      job_mult_memory: 1         # memory multiplier
      job_mult_time: 1           # running time multiplier
      merge_mult_memory: 1       # memory multiplier for merging
      merge_mult_time: 1         # running time multiplier for merging
```

(continues on next page)

(continued from previous page)

```
      # GATK HC--specific configuration
      allow_seq_dict_incompatibility: false
      annotations:
      - BaseQualityRankSumTest
      - FisherStrand
      - GCContent
      - HaplotypeScore
      - HomopolymerRun
      - MappingQualityRankSumTest
      - MappingQualityZero
      - QualByDepth
      - ReadPosRankSumTest
      - RMSMappingQuality
      - DepthPerAlleleBySample
      - Coverage
      - ClippingRankSumTest
      - DepthPerSampleHC
    gatk_ug_joint:
      # Parallelization configuration
      num_cores: 2              # number of cores to use locally
      window_length: 50000000   # split input into windows of this size, each triggers a␣
↪job
      num_jobs: 500             # number of windows to process in parallel
      use_profile: true         # use Snakemake profile for parallel processing
      restart_times: 5          # number of times to re-launch jobs in case of failure
      max_jobs_per_second: 10   # throttling of job creation
      max_status_checks_per_second: 10   # throttling of status checks
      debug_trunc_tokens: 0     # truncation to first N tokens (0 for none)
      keep_tmpdir: never        # keep temporary directory, {always, never, onerror}
      job_mult_memory: 1        # memory multiplier
      job_mult_time: 1          # running time multiplier
      merge_mult_memory: 1      # memory multiplier for merging
      merge_mult_time: 1        # running time multiplier for merging
      # GATK UG--specific configuration
      downsample_to_coverage: 250
      allow_seq_dict_incompatibility: false
      annotations:
      - BaseQualityRankSumTest
      - FisherStrand
      - GCContent
      - HaplotypeScore
      - HomopolymerRun
      - MappingQualityRankSumTest
      - MappingQualityZero
      - QualByDepth
      - ReadPosRankSumTest
      - RMSMappingQuality
      - DepthPerAlleleBySample
      - Coverage
      - ClippingRankSumTest
      - DepthPerSampleHC
    varscan_joint:
```

(continues on next page)

(continued from previous page)

```
      # Parallelization configuration
      num_cores: 2              # number of cores to use locally
      window_length: 5000000    # split input into windows of this size, each triggers a␣
↪job
      num_jobs: 500             # number of windows to process in parallel
      use_profile: true         # use Snakemake profile for parallel processing
      restart_times: 5          # number of times to re-launch jobs in case of failure
      max_jobs_per_second: 2    # throttling of job creation
      max_status_checks_per_second: 10   # throttling of status checks
      # Configuration for samtools mpileup
      max_depth: 4000
      max_indel_depth: 4000
      min_bq: 13
      no_baq: True
      # Configuration for Varscan
      min_coverage: 8
      min_reads2: 2
      min_avg_qual: 15
      min_var_freq: 0.01
      min_freq_for_hom: 0.75
      p_value: 99e-02
```

## 21.5 Available Somatic Variant Callers

The following somatic variant callers are currently available

- "mutect"
- "scalpel"

## 21.6 Reports

Currently, no reports are generated.

CHAPTER

# TWENTYTWO

# SOMATIC VARIANT CHECKING

Implementation of the germline `somatic_variant_checking` step

The `somatic_variant_checking` step takes as the input the results of the `somatic_variant_annotation` step. It then executes various tools computing statistics on the result files and consistency checks with the pedigrees.

**Note:** Status: not implemented yet

## 22.1 Step Input

The variant calling step uses Snakemake sub workflows for using the result of the `somatic_variant_annotation` step.

## 22.2 Step Output

**Note:** TODO

## 22.3 Global Configuration

**Note:** TODO

## 22.4 Default Configuration

The default configuration is as follows.

```
step_config:
  somatic_variant_checking:
    path_somatic_variant_calling: ../somatic_variant_calling  # REQUIRED
```

## 22.5 Reports

Currently, no reports are generated.

CHAPTER

# TWENTYTHREE

# SOMATIC VARIANT EXPRESSION

Implementation of the `somatic_variant_expression` step

This step allows the combination of somatic variant calling results with their expression from RNA-seq data. This allows for (1) extending a somatic VCF file with columns for the corresponding RNA-seq data giving depth of coverage and minor allele fraction in the tumor RNA-eq and (2) for computing a p value for likelihood of observation by chance.

**Note:** Status: not implemented yet

## 23.1 Step Input

**Note:** TODO

## 23.2 Step Output

**Note:** TODO

## 23.3 Default Configuration

The default configuration is as follows.

```
step_config:
  somatic_variant_expression:
    path_ngs_mapping: ../ngs_mapping                          # REQUIRED
    path_somatic_variant_calling: ../somatic_variant_calling  # REQUIRED
```

CHAPTER

# TWENTYFOUR

# SOMATIC VARIANT FILTRATION

Implementation of the `somatic_variant_filtration` step

## 24.1 Default Configuration

The default configuration is as follows.

```
# Default configuration variant_annotation
step_config:
  somatic_variant_filtration:
    path_somatic_variant_annotation: ../somatic_variant_annotation
    path_ngs_mapping: ../ngs_mapping
    tools_ngs_mapping: null
    tools_somatic_variant_calling: null
    filter_sets:
    # no_filter: no_filters    # implicit, always defined
      dkfz_only: ''  # empty
      dkfz_and_ebfilter:
        ebfilter_threshold: 2.4
      dkfz_and_ebfilter_and_oxog:
        vaf_threshold: 0.08
        coverage_threshold: 5
      dkfz_and_oxog:
        vaf_threshold: 0.08
        coverage_threshold: 5
    exon_lists: {}
    # genome_wide: null         # implicit, always defined
    # ensembl74: path/to/ensembl47.bed
    ignore_chroms:            # patterns of chromosome names to ignore
    - NC_007605    # herpes virus
    - hs37d5       # GRCh37 decoy
    - chrEBV       # Eppstein-Barr Virus
    - '*_decoy'    # decoy contig
    - 'HLA-*'      # HLA genes
    - 'GL000220.*' # Contig with problematic, repetitive DNA in GRCh37
    eb_filter:
      shuffle_seed: 1
      panel_of_normals_size: 25
      min_mapq: 20
      min_baseq: 15
```

(continues on next page)

(continued from previous page)

```
    # Parallelization configuration
    window_length: 10000000   # split input into windows of this size, each triggers a␣
↪job
    num_jobs: 500             # number of windows to process in parallel
    use_profile: true         # use Snakemake profile for parallel processing
    restart_times: 5          # number of times to re-launch jobs in case of failure
    max_jobs_per_second: 2    # throttling of job creation
    max_status_checks_per_second: 10   # throttling of status checks
    debug_trunc_tokens: 0     # truncation to first N tokens (0 for none)
    keep_tmpdir: never        # keep temporary directory, {always, never, onerror}
    job_mult_memory: 1        # memory multiplier
    job_mult_time: 1          # running time multiplier
    merge_mult_memory: 1      # memory multiplier for merging
    merge_mult_time: 1        # running time multiplier for merging
```

## 24.2 Important

Because the EB Filter step is so time consuming, the data going can be heavily prefiltered! (e.g. using Jannovar with the offExome flag).

TODO: document filter, for now see the eb_filter wrapper!

## 24.3 Concept

All variants are annotated with the dkfz-bias-filter to remove sequencing and PCR artifacts. The variants annotatated with EBFilter are variable, i.e. only variants that have the PASS flag set because we assume only those will be kept.

We borrowed the general workflow from variant_filtration, i.e. working with pre-defined filter sets and exon/region lists.

## 24.4 Workflow

- 1. Do the filtering genome wide (this file needs to be there, always)
  - dkfz-ebfilter-filterset1-genomewide
- 2. optionally, subset to regions defined in bed file, which return
  - dkfz-ebfilter-filterset1-regions1

and so on for filterset1 to n

filterset1: filter bPcr, bSeq flags from dkfz-bias-filter

filterset2: additionally filter variants with EBscore < x, x is configurable

CHAPTER

# TWENTYFIVE

# SOMATIC WGS CNV CALLING

Implementation of the `somatic_wgs_cnv_calling` step

The `somatic_wgs_cnv_calling` step takes as the input the results of the `ngs_mapping` step (aligned NGS reads) and performs somatic CNV calling on them. The result are called CNVs in VCF format.

## 25.1 Step Input

The variant annotation step uses Snakemake sub workflows for using the result of the `ngs_mapping` and `somatic_variant_calling` steps. Somatic (small) variant calling is required for b-allele based filtration. For the somatic variant calling, one somatic (small) variant caller must be configured of which to use the results.

## 25.2 Step Output

For each tumor DNA NGS library with name `lib_name`/key `lib_pk` and each read mapper `mapper` that the library has been aligned with, and the variant caller `var_caller`, the pipeline step will create a directory `output/{mapper}.{var_caller}.{lib_name}-{lib_pk}/out` with symlinks of the following names to the resulting VCF, TBI, and MD5 files.

- `{mapper}.{var_caller}.{lib_name}-{lib_pk}.vcf.gz`
- `{mapper}.{var_caller}.{lib_name}-{lib_pk}.vcf.gz.tbi`
- `{mapper}.{var_caller}.{lib_name}-{lib_pk}.vcf.gz.md5`
- `{mapper}.{var_caller}.{lib_name}-{lib_pk}.vcf.gz.tbi.md5`

For example, it might look as follows for the example from above:

```
output/
+-- bwa.canvas.P001-T1-DNA1-WGS1-4
|   `-- out
|       |-- bwa.canvas.P001-T1-DNA1-WGS1-4.vcf.gz
|       |-- bwa.canvas.P001-T1-DNA1-WGS1-4.vcf.gz.tbi
|       |-- bwa.canvas.P001-T1-DNA1-WGS1-4.vcf.gz.md5
|       `-- bwa.canvas.P001-T1-DNA1-WGS1-4.vcf.gz.tbi.md5
[...]
```

Generally, these files will be unfiltered, i.e., contain low-quality variants and also variants flagged as being non-somatic.

## 25.3 Global Configuration

None so far

## 25.4 Default Configuration

The default configuration is as follows.

## 25.5 Available Somatic CNV Callers

The following somatic CNV callers are currently available

- `"canvas"`

## 25.6 Reports

Currently, no reports are generated.

CHAPTER

# TWENTYSIX

# SOMATIC WGS SV CALLING

Implementation of the somatic_wgs_sv_calling step

The somatic_wgs_sv_calling step takes as the input the results of the ngs_mapping step (aligned NGS reads) and performs somatic SV calling on them. The result are called SVs in VCF format.

## 26.1 Step Input

The variant annotation step uses Snakemake sub workflows for using the result of the ngs_mapping step.

## 26.2 Step Output

For each tumor DNA NGS library with name lib_name/key lib_pk and each read mapper mapper that the library has been aligned with, and the variant caller var_caller, the pipeline step will create a directory output/{mapper}.{var_caller}.{lib_name}-{lib_pk}/out with symlinks of the following names to the resulting VCF, TBI, and MD5 files.

- {mapper}.{var_caller}.{lib_name}-{lib_pk}.vcf.gz
- {mapper}.{var_caller}.{lib_name}-{lib_pk}.vcf.gz.tbi
- {mapper}.{var_caller}.{lib_name}-{lib_pk}.vcf.gz.md5
- {mapper}.{var_caller}.{lib_name}-{lib_pk}.vcf.gz.tbi.md5

For example, it might look as follows for the example from above:

```
output/
+-- bwa.manta.P001-T1-DNA1-WGS1-4
|   `-- out
|       |-- bwa.manta.P001-T1-DNA1-WGS1-4.vcf.gz
|       |-- bwa.manta.P001-T1-DNA1-WGS1-4.vcf.gz.tbi
|       |-- bwa.manta.P001-T1-DNA1-WGS1-4.vcf.gz.md5
|       `-- bwa.manta.P001-T1-DNA1-WGS1-4.vcf.gz.tbi.md5
[...]
```

Generally, these files will be unfiltered, i.e., contain low-quality variants and also variants flagged as being non-somatic.

## 26.3 Global Configuration

- The `static_data_config/reference/path` has to be configured with the path to the reference FASTA file.

## 26.4 Default Configuration

The default configuration is as follows.

```
# Default configuration somatic_wgs_sv_calling
step_config:
  somatic_wgs_sv_calling:
    path_ngs_mapping: ../ngs_mapping  # REQUIRED
    tools: [manta]  # REQUIRED - available: 'delly2' and 'manta'
    delly2:
      path_exclude_tsv: null  # optional
      max_threads: 16
```

## 26.5 Available Somatic CNV Callers

The following somatic SV callers are currently available

- `"manta"`
- `"delly2"`

## 26.6 Reports

Currently, no reports are generated.

CHAPTER

TWENTYSEVEN

# GERMLINE TARGETED SEQ. CNV CALLING

SV calling for targeted sequencing

Based on the output of `ngs_mapping`, call structural variants from depth of coverage, read pair, and split read signal.

CHAPTER

TWENTYEIGHT

# GERMLINE TARGETED SEQ. MEI CALLING

Implementation of the `targeted_seq_mei_calling` step

The `targeted_seq_mei_calling` step takes as the input the results of the `ngs_mapping` step (aligned reads in BAM format) and performs germline mobile element insertion (MEI) identification. The result are VCF files with mobile insertions.

## 28.1 Stability

This step is considered experimental, use it at your own discretion.

## 28.2 Step Input

MEI identification step uses Snakemake sub workflows for using the result of the `ngs_mapping` step.

## 28.3 Step Output

For all samples, MEI identification will be performed on the primary DNA NGS libraries separately for each configured read mapper and mobile element identification tool. The name of the primary DNA NGS library will be used as an identification token in the output file.

For each read mapper, MEI tool, and sample the following files will be generated:

- `{mapper}.{mei_tool}.{lib_name}.vcf.gz`
- `{mapper}.{mei_tool}.{lib_name}.vcf.gz.md5`

For example, it might look as follows for the example from above:

```
output/
+-- bwa.scramble.P001-N1-DNA1-WES1
|   `-- out
|       |-- bwa.scramble.P001-N1-DNA1-WES1.vcf.gz
|       |-- bwa.scramble.P001-N1-DNA1-WES1.vcf.gz.md5
[...]
```

## 28.4 Global Configuration

Not applicable.

## 28.5 Default Configuration

The default configuration is as follows.

```
# Default configuration
step_config:
  targeted_seq_mei_calling:
    # Path to the ngs_mapping step
    path_ngs_mapping: ../ngs_mapping

    tools: [scramble]  # REQUIRED - available: 'scramble'

    scramble:
      blast_ref: null  # REQUIRED: path to FASTA reference with BLAST DB (`makeblastdb`)
      mei_refs: null  # OPTIONAL: MEI reference file (FASTA), if none provided will use␣
↪default.
      n_cluster: 5  # OPTIONAL: minimum cluster size, depth of soft-clipped reads.
      mei_score: 50  # OPTIONAL: minimum MEI alignment score.
      indel_score: 80  # OPTIONAL: minimum INDEL alignment score.
      mei_polya_frac: 0.75  # OPTIONAL: minimum fraction of clipped length for calling␣
↪polyA tail.
```

## 28.6 Available MEI Identification Tools

The following germline MEI identification tool is currently available:

- "Scramble"

## 28.7 Reports

Not applicable.

## 28.8 Parallel Execution

Not available.

CHAPTER

# TWENTYNINE

# GERMLINE REPEAT EXPANSION ANALYSIS

Implementation of the `repeat_analysis` step

The `repeat_analysis` step takes as the input the results of the `ngs_mapping` step (aligned reads in BAM format) and performs repeat expansion analysis. The result are variant files (VCF) with the repeat expansions definitions, and associated annotations (JSON).

## 29.1 Stability

This step is considered experimental, use it at your own discretion.

## 29.2 Step Input

The repeat analysis step uses Snakemake sub workflows for using the result of the `ngs_mapping` step.

## 29.3 Step Output

For all samples, repeat analysis will be performed on the primary DNA NGS libraries separately for each configured read mapper and repeat analysis tool. The name of the primary DNA NGS library will be used as an identification token in the output file.

For each read mapper, repeat analysis tool, and sample, the following files will be generated:

- `{mapper}.{repeat_tool}.{lib_name}.vcf`
- `{mapper}.{repeat_tool}.{lib_name}.vcf.md5`
- `{mapper}.{repeat_tool}_annotated.{lib_name}.json`
- `{mapper}.{repeat_tool}_annotated.{lib_name}.json.md5`

For example, it might look as follows for the example from above:

```
output/
+-- bwa.expansionhunter.P001-N1-DNA1-WES1
|   `-- out
|       |-- bwa.expansionhunter.P001-N1-DNA1-WES1.vcf
|       |-- bwa.expansionhunter.P001-N1-DNA1-WES1.vcf.md5
+-- bwa.expansionhunter_annotated.P001-N1-DNA1-WES1
|   `-- out
```

(continues on next page)

(continued from previous page)

```
|       |-- bwa.expansionhunter_annotated.P001-N1-DNA1-WES1.json
|       |-- bwa.expansionhunter_annotated.P001-N1-DNA1-WES1.json.md5
[...]
```

## 29.4 Global Configuration

Not applicable.

## 29.5 Default Configuration

The default configuration is as follows:

```
# Default configuration repeat_expansion
step_config:
  repeat_expansion:
    # Repeat expansions definitions - used in ExpansionHunter call
    repeat_catalog: REQUIRED
    # Repeat expansions annotations, e.g., normality range - custom file
    repeat_annotation: REQUIRED
    # Path to the ngs_mapping step
    path_ngs_mapping: ../ngs_mapping
```

## 29.6 Available Repeat Analysis Tools

The following germline repeat analysis tool is currently available:

- "ExpansionHunter"

## 29.7 Parallel Execution

Not available.

CHAPTER

THIRTY

# T CELL CRG REPORT

Implementation of the `tcell_crg_report` step

This step collects all of the calls and information gathered for the BIH T cell CRG and generates an Excel report for each patient.

**Note:** Status: not implemented yet

## 30.1 Step Input

The BIH T cell CRG report generator uses the following as input:

- `somatic_variant_annotation`
- `somatic_epitope_prediction`
- `somatic_variant_checking`
- `somatic_ngs_sanity_checks`

## 30.2 Step Output

**Note:** TODO

## 30.3 Default Configuration

The default configuration is as follows.

```
# Default configuration tcell_crg_report
step_config:
  tcell_crg_report:
    path_somatic_variant_annotation: ../somatic_variant_annotation  # REQUIRED
```

## 30.4 Available Gene Fusion Callers

- `cnvkit`

CHAPTER

# THIRTYONE

# GERMLINE VARIANT ANNOTATION

Implementation of the `variant_annotation` step

The `variant_annotation` step takes as the input the results of the `variant_calling` step (called germline variants in vcf.gz format) and annotates the variants, e.g., using VEP.

## 31.1 Stability

TBD

## 31.2 Step Input

The variant annotation step uses Snakemake sub workflows for using the result of the `variant_calling` step.

## 31.3 Step Output

TBD

## 31.4 Global Configuration

TBD

## 31.5 Default Configuration

The default configuration is as follows.

```
# Default configuration variant_annotation
step_config:
  variant_annotation:
    path_variant_calling: ../variant_calling
    tools:
      - vep
    vep:
      # We will always run VEP in cache mode.  You have to provide the directory to the
```

(continues on next page)

(continued from previous page)

```
      # cache to use (VEP would be ``~/.vep``).
      cache_dir: null # OPTIONAL
      # The cache version to use.  gnomAD v2 used 85, gnomAD v3.1 uses 101.
      cache_version: "85"
      # The assembly to use.  gnomAD v2 used "GRCh37", gnomAD v3.1 uses "GRCh38".
      assembly: "GRCh37"
      # The flag selecting the transcripts.  One of "gencode_basic", "refseq", and
↪"merged".
      tx_flag: "gencode_basic"
      # Number of threads to use with forking, set to 0 to disable forking.
      num_threads: 16
      # Additional flags.
      more_flags: "--af_gnomade --af_gnomadg"
      # The --buffer_size parameter
      buffer_size: 100000
```

## 31.6 Available Variant Annotators

The following variant annotator is currently available:

- "vep" : See the software documentation for more details

## 31.7 Reports

N/A

CHAPTER

THIRTYTWO

# GERMLINE VARIANT CALLING

Implementation of the `variant_calling` step

The `variant_calling` step takes the output of the `ngs_mapping` step and performs small variant calling on the read alignments. The output are variant calls in VCF (and optionally gVCF) files and quality control statistics on these data.

## 32.1 Properties

overall stability

**stable**

applicable to

germline variant calling

generally applicable to

short read variant calling

## 32.2 Step Input

BAM files from the `ngs_mapping` step.

## 32.3 Step Output

Creates one output directory for each read mapper (from `ngs_mapping`), each variant caller, and each pedigree from the germline sample sheet.

**Primary Output**

- `output/{mapper}.{caller}.{index_library}/out/{mapper}.{caller}.{index_library}.vcf.gz`

**Additional Output**

The callers implementing a gVCF workflow (currently only `gatk4_hc_gvcf`) also create one output gVCF file for the pedigree.

- `output/{mapper}.{caller}.{index_library}/out/{mapper}.{caller}.{index_library}.g.vcf.gz`

Further, each VCF and gVCF file gets an appropriate TBI index file `{vcf_file}.tbi` and each output is gets an appropriate MD5 checksum file `{file}.md5`.

## 32.4 Global Configuration

- If GATK HaplotypeCaller or GATK UnifiedGenotyper are activated then `static_data_config/dbsnp/path` must be properly configured
- `static_data_config/reference/path` must be set appropriately

## 32.5 Default Configuration

The default configuration is as follows.

```
# Default configuration variant_calling
step_config:
  variant_calling:
    # Common configuration
    path_ngs_mapping: ../ngs_mapping  # REQUIRED

    # Report generation
    baf_file_generation:
      enabled: true
      min_dp: 10  # minimal DP of variant, must be >=1
    bcftools_stats:
      enabled: true
    jannovar_stats:
      enabled: true
      path_ser: REQUIRED  # REQUIRED
    bcftools_roh:
      enabled: true
      path_targets: null  # REQUIRED; optional
      path_af_file: null  # REQUIRED
      ignore_homref: false
      skip_indels: false
      rec_rate: 1e-8

    # Variant calling tools and their configuration
    #
    # Common configuration
    tools: ['gatk4_hc_gvcf']  # REQUIRED
    ignore_chroms:
    - '^NC_007605$' # herpes virus
    - '^hs37d5$'    # GRCh37 decoy
    - '^chrEBV$'    # Eppstein-Barr Virus
    - '_decoy$'     # decoy contig
    - '^HLA-'       # HLA genes

    # Variant caller specific configuration
    bcftools_call:
```

(continues on next page)

(continued from previous page)

```
      max_depth: 250
      max_indel_depth: 250
      window_length: 10000000
      num_threads: 16
    gatk3_hc:
      num_threads: 16
      window_length: 10000000
      allow_seq_dict_incompatibility: false
    gatk3_ug:
      num_threads: 16
      window_length: 10000000
      allow_seq_dict_incompatibility: false
      downsample_to_coverage: 250
    gatk4_hc_joint:
      window_length: 10000000
      num_threads: 16
      allow_seq_dict_incompatibility: false
    gatk4_hc_gvcf:
      window_length: 10000000
      num_threads: 16
      allow_seq_dict_incompatibility: false
```

## 32.6 Variant Callers

The following germline variant callers are currently available.

`gatk4_hc_gvcf`

> Variant calling with GATK v4 HaplotypeCaller using the gVCF workflow consisting of variant discovery with `HaplotypeCaller`, merging of the gVCF files withing each pedigree with `CombineGVCFs` and genotyping with `GenotypeGVCFs`.
>
> This is the mainly used variant caller and the only one enabled by default.
>
> The reason is this being the main advertised run mode by the GATK team and this workflow enables physical phasing information in the output VCF files.

`gatk4_hc_joint`

> Variant calling with the GATK v4 HaplotypeCaller using joint calling with direct VCF generation.
>
> This variant caller is provided as a fallback to explore problems with *de novo* variant calls that may have been introduced by the gVCF workflow.
>
> Disabled by default.

`gatk3_hc`

> Joint calling with GATK v3 HaplotypeCaller.
>
> This caller is provided for historical reasons as earlier versions of SNAPPY pipeline were based on this workflow.
>
> Disabled by default.

`gatk3_ug`

Joint calling with GATK v3 UnifiedGenotyper.

This caller is provided for historical reasons and to provide a vote in creating consensus sets of variant calls.

`bcftools_call`

Variant calling with `bcftools mpileup | bcftools call`.

This caller is provided for establishing baseline variant calls in benchmark situations. BCFtools allows for fast and efficient variant calling at the cost of some sensitivity and specificity.

Disabled by default.

## 32.7 Reports

`jannovar_stats`

Create statistics on variants using `jannovar statsistics` for each pedigree.

```
report/jannovar_stats/{mapper}.{caller}.{index_library}.{donor_library}.txt
```

`bcftools_stats`

Create statistics on variants using `bcftools stats` for each donor in each pedigree for each mapper and caller.

```
report/bcftools_stats/{mapper}.{caller}.{index_library}.{donor_library}.txt
```

`baf_file_generation`

Create one UCSC BigWig file for each individual in each pedigree for each mapper and caller with B-allele fraction. These files can be used for to visually confirm structural variants or runs of homozygosity.

```
report/baf/{mapper}.{caller}.{index_library}.{donor_library}.bw
```

`roh_calling`

Perform run-of-homozygosity calling with `bcftools roh`.

## 32.8 Log Files

For each variant caller and report generator, the following log files are created into the `log` directory.

`{file}.conda_info.txt`

Output of `conda info` of the executing conda environment.

`{file}.conda_list.txt`

Output of `conda list` of the executing conda environment with list of the full package list and exact versions.

`{file}.log`

Log output of the execution.

`{file}.wrapper.py`

The actual Snakemake wrapper file with all input / output / parameter values.

## 32.9 Implementation Notes

- All variant callers are parallelized using GNU parallel on genome-wide windows generated by GATK v4 `PreprocessIntervals`.
- Each output file has an accompanying MD5 sum.

## 32.10 Example Output

Given a pedigree with index `index` and two more donors `mother` and `father`, the following files would be created into `output/` (each VCF file has a `.tbi` file and overall each file has a `.md5` file). In this case, the read mapper is `bwa` and the variant caller is `gatk4_hc_gvcf`.

```
` bwa.gatk4_hc_gvcf.index-N1-DNA1-WES1/log/bwa.gatk4_hc_gvcf.index-N1-DNA1-WES1.
index-N1-DNA1-WES1.baf_file_generation_run.conda_info.txt bwa.gatk4_hc_gvcf.
index-N1-DNA1-WES1/log/bwa.gatk4_hc_gvcf.index-N1-DNA1-WES1.index-N1-DNA1-WES1.
baf_file_generation_run.conda_list.txt bwa.gatk4_hc_gvcf.index-N1-DNA1-WES1/log/bwa.
gatk4_hc_gvcf.index-N1-DNA1-WES1.index-N1-DNA1-WES1.baf_file_generation_run.environment.
yaml bwa.gatk4_hc_gvcf.index-N1-DNA1-WES1/log/bwa.gatk4_hc_gvcf.index-N1-DNA1-WES1.
index-N1-DNA1-WES1.baf_file_generation_run.log bwa.gatk4_hc_gvcf.index-N1-DNA1-WES1/
log/bwa.gatk4_hc_gvcf.index-N1-DNA1-WES1.index-N1-DNA1-WES1.baf_file_generation_run.
wrapper.py bwa.gatk4_hc_gvcf.index-N1-DNA1-WES1/log/bwa.gatk4_hc_gvcf.index-N1-DNA1-WES1.
index-N1-DNA1-WES1.bcftools_stats_run.conda_info.txt bwa.gatk4_hc_gvcf.
index-N1-DNA1-WES1/log/bwa.gatk4_hc_gvcf.index-N1-DNA1-WES1.index-N1-DNA1-WES1.
bcftools_stats_run.conda_list.txt bwa.gatk4_hc_gvcf.index-N1-DNA1-WES1/log/bwa.
gatk4_hc_gvcf.index-N1-DNA1-WES1.index-N1-DNA1-WES1.bcftools_stats_run.environment.
yaml bwa.gatk4_hc_gvcf.index-N1-DNA1-WES1/log/bwa.gatk4_hc_gvcf.index-N1-DNA1-WES1.
index-N1-DNA1-WES1.bcftools_stats_run.log bwa.gatk4_hc_gvcf.index-N1-DNA1-WES1/log/
bwa.gatk4_hc_gvcf.index-N1-DNA1-WES1.index-N1-DNA1-WES1.bcftools_stats_run.wrapper.
py bwa.gatk4_hc_gvcf.index-N1-DNA1-WES1/log/bwa.gatk4_hc_gvcf.index-N1-DNA1-WES1.
father-N1-DNA1-WES1.baf_file_generation_run.conda_info.txt bwa.gatk4_hc_gvcf.
index-N1-DNA1-WES1/log/bwa.gatk4_hc_gvcf.index-N1-DNA1-WES1.father-N1-DNA1-WES1.
baf_file_generation_run.conda_list.txt bwa.gatk4_hc_gvcf.index-N1-DNA1-WES1/log/bwa.
gatk4_hc_gvcf.index-N1-DNA1-WES1.father-N1-DNA1-WES1.baf_file_generation_run.environment.
yaml bwa.gatk4_hc_gvcf.index-N1-DNA1-WES1/log/bwa.gatk4_hc_gvcf.index-N1-DNA1-WES1.
father-N1-DNA1-WES1.baf_file_generation_run.log bwa.gatk4_hc_gvcf.index-N1-DNA1-WES1/
log/bwa.gatk4_hc_gvcf.index-N1-DNA1-WES1.father-N1-DNA1-WES1.baf_file_generation_run.
wrapper.py bwa.gatk4_hc_gvcf.index-N1-DNA1-WES1/log/bwa.gatk4_hc_gvcf.index-N1-DNA1-WES1.
father-N1-DNA1-WES1.bcftools_stats_run.conda_info.txt bwa.gatk4_hc_gvcf.
index-N1-DNA1-WES1/log/bwa.gatk4_hc_gvcf.index-N1-DNA1-WES1.father-N1-DNA1-WES1.
bcftools_stats_run.conda_list.txt bwa.gatk4_hc_gvcf.index-N1-DNA1-WES1/log/bwa.
gatk4_hc_gvcf.index-N1-DNA1-WES1.father-N1-DNA1-WES1.bcftools_stats_run.environment.
yaml bwa.gatk4_hc_gvcf.index-N1-DNA1-WES1/log/bwa.gatk4_hc_gvcf.index-N1-DNA1-WES1.
father-N1-DNA1-WES1.bcftools_stats_run.log bwa.gatk4_hc_gvcf.index-N1-DNA1-WES1/log/
bwa.gatk4_hc_gvcf.index-N1-DNA1-WES1.father-N1-DNA1-WES1.bcftools_stats_run.wrapper.
py bwa.gatk4_hc_gvcf.index-N1-DNA1-WES1/log/bwa.gatk4_hc_gvcf.index-N1-DNA1-WES1.
mother-N1-DNA1-WES1.baf_file_generation_run.conda_info.txt bwa.gatk4_hc_gvcf.
index-N1-DNA1-WES1/log/bwa.gatk4_hc_gvcf.index-N1-DNA1-WES1.mother-N1-DNA1-WES1.
baf_file_generation_run.conda_list.txt bwa.gatk4_hc_gvcf.index-N1-DNA1-WES1/log/bwa.
gatk4_hc_gvcf.index-N1-DNA1-WES1.mother-N1-DNA1-WES1.baf_file_generation_run.environment.
yaml bwa.gatk4_hc_gvcf.index-N1-DNA1-WES1/log/bwa.gatk4_hc_gvcf.index-N1-DNA1-WES1.
mother-N1-DNA1-WES1.baf_file_generation_run.log bwa.gatk4_hc_gvcf.index-N1-DNA1-WES1/
log/bwa.gatk4_hc_gvcf.index-N1-DNA1-WES1.mother-N1-DNA1-WES1.baf_file_generation_run.
```

```
wrapper.py bwa.gatk4_hc_gvcf.index-N1-DNA1-WES1/log/bwa.gatk4_hc_gvcf.index-N1-DNA1-WES1.
mother-N1-DNA1-WES1.bcftools_stats_run.conda_info.txt bwa.gatk4_hc_gvcf.
index-N1-DNA1-WES1/log/bwa.gatk4_hc_gvcf.index-N1-DNA1-WES1.mother-N1-DNA1-WES1.
bcftools_stats_run.conda_list.txt bwa.gatk4_hc_gvcf.index-N1-DNA1-WES1/log/bwa.
gatk4_hc_gvcf.index-N1-DNA1-WES1.mother-N1-DNA1-WES1.bcftools_stats_run.environment.
yaml bwa.gatk4_hc_gvcf.index-N1-DNA1-WES1/log/bwa.gatk4_hc_gvcf.index-N1-DNA1-WES1.
mother-N1-DNA1-WES1.bcftools_stats_run.log bwa.gatk4_hc_gvcf.index-N1-DNA1-WES1/log/
bwa.gatk4_hc_gvcf.index-N1-DNA1-WES1.mother-N1-DNA1-WES1.bcftools_stats_run.wrapper.
py bwa.gatk4_hc_gvcf.index-N1-DNA1-WES1/log/bwa.gatk4_hc_gvcf.index-N1-DNA1-WES1.
gatk4_hc_gvcf_genotype.conda_info.txt bwa.gatk4_hc_gvcf.index-N1-DNA1-WES1/
log/bwa.gatk4_hc_gvcf.index-N1-DNA1-WES1.gatk4_hc_gvcf_genotype.conda_list.txt
bwa.gatk4_hc_gvcf.index-N1-DNA1-WES1/log/bwa.gatk4_hc_gvcf.index-N1-DNA1-WES1.
gatk4_hc_gvcf_genotype.environment.yaml bwa.gatk4_hc_gvcf.index-N1-DNA1-WES1/log/
bwa.gatk4_hc_gvcf.index-N1-DNA1-WES1.gatk4_hc_gvcf_genotype.log bwa.gatk4_hc_gvcf.
index-N1-DNA1-WES1/log/bwa.gatk4_hc_gvcf.index-N1-DNA1-WES1.gatk4_hc_gvcf_genotype.
wrapper.py bwa.gatk4_hc_gvcf.index-N1-DNA1-WES1/log/bwa.gatk4_hc_gvcf.index-N1-DNA1-WES1.
jannovar_stats_run.conda_info.txt bwa.gatk4_hc_gvcf.index-N1-DNA1-WES1/log/bwa.
gatk4_hc_gvcf.index-N1-DNA1-WES1.jannovar_stats_run.conda_list.txt bwa.gatk4_hc_gvcf.
index-N1-DNA1-WES1/log/bwa.gatk4_hc_gvcf.index-N1-DNA1-WES1.jannovar_stats_run.
environment.yaml bwa.gatk4_hc_gvcf.index-N1-DNA1-WES1/log/bwa.gatk4_hc_gvcf.
index-N1-DNA1-WES1.jannovar_stats_run.log bwa.gatk4_hc_gvcf.index-N1-DNA1-WES1/
log/bwa.gatk4_hc_gvcf.index-N1-DNA1-WES1.jannovar_stats_run.wrapper.py bwa.
gatk4_hc_gvcf.index-N1-DNA1-WES1/out/bwa.gatk4_hc_gvcf.index-N1-DNA1-WES1.g.vcf.gz
bwa.gatk4_hc_gvcf.index-N1-DNA1-WES1/out/bwa.gatk4_hc_gvcf.index-N1-DNA1-WES1.vcf.gz
bwa.gatk4_hc_gvcf.index-N1-DNA1-WES1/report/baf/bwa.gatk4_hc_gvcf.index-N1-DNA1-WES1.
index-N1-DNA1-WES1.baf.bw bwa.gatk4_hc_gvcf.index-N1-DNA1-WES1/report/baf/bwa.
gatk4_hc_gvcf.index-N1-DNA1-WES1.father-N1-DNA1-WES1.baf.bw bwa.gatk4_hc_gvcf.
index-N1-DNA1-WES1/report/baf/bwa.gatk4_hc_gvcf.index-N1-DNA1-WES1.mother-N1-DNA1-WES1.
baf.bw bwa.gatk4_hc_gvcf.index-N1-DNA1-WES1/report/bcftools_stats/bwa.gatk4_hc_gvcf.
index-N1-DNA1-WES1.index-N1-DNA1-WES1.txt bwa.gatk4_hc_gvcf.index-N1-DNA1-WES1/
report/bcftools_stats/bwa.gatk4_hc_gvcf.index-N1-DNA1-WES1.father-N1-DNA1-WES1.
txt bwa.gatk4_hc_gvcf.index-N1-DNA1-WES1/report/bcftools_stats/bwa.gatk4_hc_gvcf.
index-N1-DNA1-WES1.mother-N1-DNA1-WES1.txt bwa.gatk4_hc_gvcf.index-N1-DNA1-WES1/report/
jannovar_stats/bwa.gatk4_hc_gvcf.index-N1-DNA1-WES1.txt `
```

CHAPTER

# THIRTYTHREE

# GERMLINE VARIANT SANITY CHECKING

Implementation of the germline `variant_checking` step

The `variant_checking` step takes as the input the results of the `variant_annotation` step. It then executes various tools computing statistics on the result files and consistency checks with the pedigrees.

## 33.1 Step Input

The variant calling step uses Snakemake sub workflows for using the result of the `variant_annotation` step.

## 33.2 Step Output

**Note:** TODO

## 33.3 Global Configuration

**Note:** TODO

## 33.4 Default Configuration

The default configuration is as follows.

```
step_config:
  variant_checking:
    tools_ngs_mapping: []  # optional, copied from ngs mapping config
    tools_variant_calling: []  # optional, copied from variant calling config
    path_variant_calling: ../variant_calling  # REQUIRED
    tools: ['peddy']  # REQUIRED - available: 'peddy'
```

## 33.5 Available Variant Checkers

The following variant checkers integrated:

- `"bcftools_stats"` – call `bcftools stats` for various statistics
- `"peddy"` – check variants against a PED file

## 33.6 Reports

Currently, no reports are generated.

CHAPTER

# THIRTYFOUR

# GERMLINE VARIANT *DE NOVO* FILTRATION

Implementation of the `variant_denovo_filtration` step.

This step implements filtration of variants to *de novo* variants. This step was introduced for the "Ionizing Radiation" study in ca. 2016 and the aim here is to get a set of high-confidence *de novo* sequence variants (both SNVs and indels, although the latter turned out to be less reliable). Further, if the variants are phased, assigning to paternal or maternal allele can be attempted. This allows to study paternal age effects.

Note that in contrast to `variant_calling` and `variant_annotation` but in consistency with `variant_phasing`, the central individual here are children and not the index of pedigrees.

## 34.1 Step Input

The step reads in the variant call files from one of the following steps:

- `variant_calling`
- `variant_annotation`
- `variant_phasing`

Of course, assignment to parental allele can only be performed on phased variants. Further, only filtering annotated variants is really useful as one wants to excludes variants in problematic genomic regions.

## 34.2 Step Output

For all children with both parents present, variant *de novo* annotation will be attempted on the primary DNA NGS library of that child. The name of this library will be used as the identification token in the output file and file name. For each read mapper, variant caller, and pedigree, the following files will be generated:

- `{mapper}.{var_caller}.{annotation}.{phasing}.de_novos.{lib_name}.vcf.gz.tbi`
- `{mapper}.{var_caller}.{annotation}.{phasing}.de_novos.{lib_name}.vcf.gz`
- `{mapper}.{var_caller}.{annotation}.{phasing}.de_novos.{lib_name}.vcf.gz.md5`
- `{mapper}.{var_caller}.{annotation}.{phasing}.de_novos.{lib_name}.vcf.gz.tbi.md5`
- `{mapper}.{var_caller}.{annotation}.{phasing}.de_novos_hard.{lib_name}.vcf.gz`
- `{mapper}.{var_caller}.{annotation}.{phasing}.de_novos_hard.{lib_name}.vcf.gz.tbi`
- `{mapper}.{var_caller}.{annotation}.{phasing}.de_novos_hard.{lib_name}.vcf.gz.md5`
- `{mapper}.{var_caller}.{annotation}.{phasing}.de_novos_hard.{lib_name}.vcf.gz.tbi.md5`

- `{mapper}.{var_caller}.{annotation}.{phasing}.de_novos_hard.{lib_name}.summary.txt`
- `{mapper}.{var_caller}.{annotation}.{phasing}.de_novos_hard.{lib_name}.summary.txt.md5`

The the `annotation` and `phasing` will only be persent when the input is read from the `variant_annotation` or `variant_phasing` steps, respectively.

For example, it might look as follows for the example from above:

```
output/
+-- bwa.gatk3_hc.de_novos.P001-N1-DNA1-WES1
|   `-- out
|       |-- bwa.gatk3_hc.de_novos.P001-N1-DNA1-WES1.vcf.gz
|       |-- bwa.gatk3_hc.de_novos.P001-N1-DNA1-WES1.vcf.gz.md5
|       |-- bwa.gatk3_hc.de_novos.P001-N1-DNA1-WES1.vcf.gz.tbi
|       |-- bwa.gatk3_hc.de_novos.P001-N1-DNA1-WES1.vcf.gz.tbi.md5
|       |-- bwa.gatk3_hc.de_novos.P001-N1-DNA1-WES1.vcf.gz
|       |-- bwa.gatk3_hc.de_novos_hard.P001-N1-DNA1-WES1.vcf.gz.md5
|       |-- bwa.gatk3_hc.de_novos_hard.P001-N1-DNA1-WES1.vcf.gz.tbi
|       |-- bwa.gatk3_hc.de_novos_hard.P001-N1-DNA1-WES1.vcf.gz.tbi.md5
|       |-- bwa.gatk3_hc.de_novos_hard.P001-N1-DNA1-WES1.vcf.gz
|       |-- bwa.gatk3_hc.de_novos_hard.P001-N1-DNA1-WES1.summary.txt
|       `-- bwa.gatk3_hc.de_novos_hard.P001-N1-DNA1-WES1.summary.txt.md5
[...]
```

## 34.3 Global Configuration

No global configuration is in use.

## 34.4 Default Configuration

The default configuration is as follows.

```
step_config:
  variant_denovo_filtration:
    # One of the following must be given!
    path_variant_phasing: ''
    path_variant_annotation: ''
    path_variant_calling: ''
    path_ngs_mapping: ../ngs_mapping
    tools_ngs_mapping: null          # defaults to ngs_mapping tool
    tools_variant_calling: null      # defaults to variant_annotation tool
    info_key_reliable_regions: []    # optional INFO keys with reliable regions
    info_key_unreliable_regions: []  # optional INFO keys with unreliable regions
    params_besenbacher:              # parameters for Besenbacher quality filter
      min_gq: 50
      min_dp: 10
      max_dp: 120
      min_ab: 0.20
      max_ab: 0.9
```

(continues on next page)

(continued from previous page)

```
      max_ad2: 1
    bad_region_expressions: []
    # e.g.,
    # - 'UCSC_CRG_MAPABILITY36 == 1'
    # - 'UCSC_SIMPLE_REPEAT == 1'
    collect_msdn: True                # whether or not to collect MSDN (requires GATK␣
↪HC+UG)
```

## 34.5 Reports

Currently, no reports are generated.

CHAPTER

THIRTYFIVE

# GERMLINE VARIANT PHASING

Implementation of the germline `variant_phasing` step

This step takes the result of the `variant_annotation` step and performs phasing of the variants using the GATK tools. Note that there are some issues with the GATK tools implementing this step:

- The result of the PhaseByTransmission tool changes the genotype of some variants which is problematic when trying to phase *de novo* variants.
- The read backed phasing is also not 100% reliable at the moment.

Thus, the functionality of the tools is made available by this pipeline step but it is not as fully integrated as it could because it is unclear how useful this is for clinical studies. Also, so far only the GATK variant caller results can be phased.

Also note that this step generates one output file for each child in a pedigree where both parents have been sequenced.

## 35.1 Step Input

The variant annotation step uses the output of the following CUBI pipeline steps:

- `ngs_mapping`
- `variant_annotation`

## 35.2 Step Output

For each input VCF file (i.e., for each mapper and pedigree), a directory `output/{mapper}.{caller}.{phaser}.{index_ngs_library}/out` will be created with the following output files.

The `{phaser}` placeholder can take the values gatk_phase_by_transmission, gatk_read_backed_phasing, and gatk_phased_both (for the latter, first phasing by transmission and then read backed phasing is performed).

## 35.3 Global Configuration

- `static_data_config/reference/path` must be set appropriately

## 35.4 Default Configuration

The default configuration is as follows.

```
# Default configuration wgs_sv_filtration
step_config:
  variant_phasing:
    path_ngs_mapping: ../ngs_mapping
    path_variant_annotation: ../variant_annotation
    tools_ngs_mapping: []       # expected tools for ngs mapping
    tools_variant_calling: []   # expected tools for variant calling
    phasings:
    - gatk_phasing_both
    ignore_chroms:            # patterns of chromosome names to ignore
    - NC_007605               # herpes virus
    - hs37d5                  # GRCh37 decoy
    - chrEBV                  # Eppstein-Barr Virus
    - '*_decoy'               # decoy contig
    - 'HLA-*'                 # HLA genes
    gatk_read_backed_phasing:
      phase_quality_threshold: 20.0  # quality threshold for phasing
      window_length: 5000000    # split input into windows of this size, each triggers a␣
↪job
      num_jobs: 1000            # number of windows to process in parallel
      use_profil: true          # use Snakemake profile for parallel processing
      restart_times: 0          # number of times to re-launch jobs in case of failure
      max_jobs_per_second: 10   # throttling of job creation
      max_status_checks_per_second: 10   # throttling of status checks
      debug_trunc_tokens: 0     # truncation to first N tokens (0 for none)
      keep_tmpdir: never        # keep temporary directory, {always, never, onerror}
      job_mult_memory: 1        # memory multiplier
      job_mult_time: 1          # running time multiplier
      merge_mult_memory: 1      # memory multiplier for merging
      merge_mult_time: 1        # running time multiplier for merging
    gatk_phase_by_transmission:
      de_novo_prior: 1e-8       # default, use 1e-6 when interested in phasing de novos
```

## 35.5 Reports

Currently, no reports are generated.

CHAPTER

# THIRTYSIX

# GERMLINE VARIANT FILTRATION

Implementation of the `variant_filtration` step

This step takes annotated variants as the input from `variant_annotation` and performs various filtration and post-processing operations:

1. **filter to high-confidence variants**
    1. apply quality filter sets
    2. filter for consistency between different callers
2. filter to compatible mode of inheritance
3. filter by population/cohort frequency, remove polymorphisms
4. filter by region
5. filter by scores (e.g., conservation)
6. filter for het. comp. inheritance or keep all

#

```
#     1
#     stringent
#     loose
```

# 2 # $qual.denovo # $qual.dom # $qual.rec_hom

# 3 # $qual.denovo.denov_freq # $qual.dom.dom_freq # $qual.dom.rec_freq # $qual.rec_hom.rec_freq

# 4 # $qual.denovo.denov_freq.$region # $qual.dom.dom_freq.$region # $qual.dom.rec_freq.$region # $qual.rec_hom.rec_freq.$region

# 5 # $qual.denovo.denov_freq.$region.$scores # $qual.dom.dom_freq.$region.$scores # $qual.dom.rec_freq.$region.$scores # $qual.rec_hom.rec_freq.$region.$scores

# 6 # $qual.denovo.denov_freq.$region.keep_all # $qual.dom.dom_freq.$region.keep_all # $qual.dom.rec_freq.$region.$scores.same_gene # $qual.dom.rec_freq.$region.$scores.same_tad # $qual.dom.rec_freq.$region.$scores.itv_500bp # $qual.rec_hom.rec_freq.$region.keep_all

## 36.1 Filtration Steps

The combinations of the filters is given in the configuration setting `filter_combinations` as dot-separated values, e.g., `AA.BB.CC`.

## 36.2 Step Input

TODO

## 36.3 Step Output

TODO

## 36.4 Global Configuration

TODO

## 36.5 Default Configuration

The default configuration is as follows.

```
step_config:
  variant_filtration:
    path_variant_annotation: ../variant_annotation
    tools_ngs_mapping: null      # defaults to ngs_mapping tool
    tools_variant_calling: null  # defaults to variant_annotation tool
    thresholds:                  # quality filter sets, "keep_all" implicitely defined
      conservative:
        min_gq: 40
        min_dp_het: 10
        min_dp_hom: 5
        include_expressions:
        - 'MEDGEN_COHORT_INCONSISTENT_AC=0'
      relaxed:
        min_gq: 20
        min_dp_het: 6
        min_dp_hom: 3
        include_expressions:
        - 'MEDGEN_COHORT_INCONSISTENT_AC=0'
    frequencies:             # values to use for frequency filtration
      af_dominant: 0.001     # AF (allele frequency) values
      af_recessive: 0.01
      ac_dominant: 3         # AC (allele count in gnomAD) values
    region_beds:             # regions to filter to, "whole_genome" implicitely defined
      all_tads: /fast/projects/medgen_genomes/static_data/GRCh37/hESC_hg19_allTads.bed
      all_genes: /fast/projects/medgen_genomes/static_data/GRCh37/gene_bed/ENSEMBL_v75.
↪bed.gz
```

(continues on next page)

(continued from previous page)

```
      limb_tads: /fast/projects/medgen_genomes/static_data/GRCh37/newlimb_tads.bed
      lifted_enhancers: /fast/projects/medgen_genomes/static_data/GRCh37/all_but_onlyMB.
↪bed
      vista_enhancers: /fast/projects/medgen_genomes/static_data/GRCh37/vista_limb_
↪enhancers.bed
    score_thresholds:        # thresholds on scores to filter to, "all_scores"␣
↪implictely defined
      coding:
        require_coding: true
        require_gerpp_gt2: false
        min_cadd: null
      conservative:  # unused; TODO: rename?
        require_coding: false
        require_gerpp_gt2: false
        min_cadd: 0
      conserved:  # TODO: rename?
        require_coding: false
        require_gerpp_gt2: true
        min_cadd: null
    filter_combinations: # dot-separated {thresholds}.{inherit}.{freq}.{region}.{score}.
↪{het_comp}
    - conservative.de_novo.dominant_freq.lifted_enhancers.all_scores.passthrough
    - conservative.de_novo.dominant_freq.lifted_enhancers.conserved.passthrough
    - conservative.de_novo.dominant_freq.limb_tads.all_scores.passthrough
    - conservative.de_novo.dominant_freq.limb_tads.coding.passthrough
    - conservative.de_novo.dominant_freq.limb_tads.conserved.passthrough
    - conservative.de_novo.dominant_freq.vista_enhancers.all_scores.passthrough
    - conservative.de_novo.dominant_freq.vista_enhancers.conserved.passthrough
    - conservative.de_novo.dominant_freq.whole_genome.all_scores.passthrough
    - conservative.de_novo.dominant_freq.whole_genome.coding.passthrough
    - conservative.de_novo.dominant_freq.whole_genome.conserved.passthrough
    - conservative.dominant.dominant_freq.lifted_enhancers.all_scores.passthrough
    - conservative.dominant.dominant_freq.lifted_enhancers.conserved.passthrough
    - conservative.dominant.dominant_freq.limb_tads.all_scores.passthrough
    - conservative.dominant.dominant_freq.limb_tads.coding.passthrough
    - conservative.dominant.dominant_freq.limb_tads.conserved.passthrough
    - conservative.dominant.dominant_freq.vista_enhancers.all_scores.passthrough
    - conservative.dominant.dominant_freq.vista_enhancers.conserved.passthrough
    - conservative.dominant.dominant_freq.whole_genome.all_scores.passthrough
    - conservative.dominant.dominant_freq.whole_genome.coding.passthrough
    - conservative.dominant.dominant_freq.whole_genome.conserved.passthrough
    - conservative.dominant.recessive_freq.lifted_enhancers.all_scores.intervals500
    - conservative.dominant.recessive_freq.lifted_enhancers.conserved.intervals500
    - conservative.dominant.recessive_freq.lifted_enhancers.conserved.tads
    - conservative.dominant.recessive_freq.limb_tads.all_scores.intervals500
    - conservative.dominant.recessive_freq.limb_tads.coding.gene
    - conservative.dominant.recessive_freq.limb_tads.conserved.intervals500
    - conservative.dominant.recessive_freq.limb_tads.conserved.tads
    - conservative.dominant.recessive_freq.vista_enhancers.all_scores.intervals500
    - conservative.dominant.recessive_freq.vista_enhancers.conserved.intervals500
    - conservative.dominant.recessive_freq.vista_enhancers.conserved.tads
    - conservative.dominant.recessive_freq.whole_genome.all_scores.intervals500
```

(continues on next page)

(continued from previous page)

```
    - conservative.dominant.recessive_freq.whole_genome.coding.gene
    - conservative.dominant.recessive_freq.whole_genome.conserved.intervals500
    - conservative.dominant.recessive_freq.whole_genome.conserved.tads
    - conservative.recessive_hom.recessive_freq.lifted_enhancers.all_scores.passthrough
    - conservative.recessive_hom.recessive_freq.lifted_enhancers.conserved.passthrough
    - conservative.recessive_hom.recessive_freq.limb_tads.all_scores.passthrough
    - conservative.recessive_hom.recessive_freq.limb_tads.coding.passthrough
    - conservative.recessive_hom.recessive_freq.limb_tads.conserved.passthrough
    - conservative.recessive_hom.recessive_freq.vista_enhancers.all_scores.passthrough
    - conservative.recessive_hom.recessive_freq.vista_enhancers.conserved.passthrough
    - conservative.recessive_hom.recessive_freq.whole_genome.all_scores.passthrough
    - conservative.recessive_hom.recessive_freq.whole_genome.coding.passthrough
    - conservative.recessive_hom.recessive_freq.whole_genome.conserved.passthrough
    # The following are for input to variant_combination.
    - conservative.dominant.recessive_freq.whole_genome.coding.passthrough
    - conservative.dominant.recessive_freq.whole_genome.conserved.passthrough
```

## 36.6 Reports

Currently, no reports are generated.

CHAPTER

# THIRTYSEVEN

# GERMLINE SV CALLING

Implementation of the `sv_calling_wgs` step

CHAPTER

# THIRTYEIGHT

# GERMLINE WGS SV FILTRATION

CHAPTER

# THIRTYNINE

# DEVELOPER'S INTRODUCTION

**Note:** Before reading this chapter, you should

- have knowledge from the user's perspective of CUBI pipeline (start a *Usage*).

After reading this chapter, you should

- know about the Python programming techniques required from a CUBI pipeline developer
- have an overview of the components of a pipeline step
- know that `cubi-snake` only serves as a shortcut to the `snakemake` executable.

The target audience of this part of the documentation is developers who want to change or extend the pipeline. The aim is to give a good overview of the architecture of the pipeline system and dissect some typical existing pipeline steps for educational purposes. Most parts of the system follow a consistent programming and architecture style that should be followed to ease the understanding of the system.

If you are a proficient Python programmer then you should not have a too hard time to get started. If your Python karate is less strong (e.g., if you are a Bioinformatician coming from the "bio" and not the "informtician" side), take a deep breath and brace yourself, you will learn something here. Before we start, here is the Zen of Python as a reminder:

```
>>> import this
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
```

(continues on next page)

(continued from previous page)

```
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

## 39.1 Prerequisites – Your Tool Belt

The CUBI pipeline system is implemented using Python 3 (>=3.4 at the moment) and built upon the wonderful Snakemake (>=3.10 at the moment). For distributed, parallel execution, the pipeline is tailored towards execution with SGE Grid Engine. In order to follow this developer's documentation comfortably, you should be familiar with all three systems:

- Python 3
- Snakemake
- Grid Engine (or similar cluster job queueing system).

You should be familiar with the CUBI pipeline from the user perspective already.

Also, understanding in the following techniques will come in handy:

- Snakemake as a rule-based language for describing workflows,
- object oriented programming,
- Python generators (via `yield`, use of `yield from`),
- Python decorators,
- Python `itertools` package and built-ins such as `zip`/`map`
- Exception handlng
- JSON, JSON schema,
- scope, lambdas, and closures in Python,
- realize that classes are objects themselves and callable (their constructor),
- the `snakemake.io.expand()` helper function
- text manipulation using `str.format`, `textwrap.dedent`, `str.lstrip`,
- understanding in common Python standard library code such as `os[.path]`, `collections.OrderedDict`,
- the recently added Snakemake `unpack()` keyword,
- the concept of mixin classes.

The following are handy references about using Python effectively:

- The Hitchhiker's Guide to Python.
- Slatkin, Brett: Effective Python: 59 Specific Ways to Write Better Python

## 39.2 Anatomy of a Typical Pipeline Step

Each pipeline step is implemented as a Snakemake workflow. For each step, there is a module sub directory below `snappy_pipeline.workflows` containing:

- `__init__.py` with classes that actually implement the workflow
- `Snakefile` that contains the Snakemake rule definitions but usually just hooks in calls to the actual implementation code from `__init__.py`.

Usually, you define a *BaseStep* sub class in your Python code (`__init__.py`) that is then instantiated in your `Snakefile`. The current configuration is passed into the constructor of this class and it then "takes over" and applies default setting, generating cluster resource settings, etc. Then, you pass the result of method calls to your *BaseStep* instance as the values for the `input:`, `output:`, etc. sections of your `Snakefile`.

**Warning:** By convention your new Workflow step should be instantiated as `wf = StepClass(...)` in the `Snakefile` during object setup. Otherwise tools including cubi-tk might not be able to detect and parse your step. See existing workflow `Snakefile` for reference.

The *BaseStep* sub class itself uses *BaseStepPart* sub classes for the implementation of the individual parts. One part might be linking in FASTQ files from the raw input directory or linking from the `work/` to the `output/` directory. Another part might be the somatic variant calling using mutect or WGS SV calling using Delly2.

Each of the parts might be split into different actions if the implementing tools need their own more or less complex "workflow" themselves. An example for such a tool is Delly2 where first variant calling is performed for each sample, then the resulting site list is merged and used for genotpying is all samples individually. Finally, the wohle cohort's genotypes are merged and for each sample, only the variants that have been observed in it will be executed. If the tools can just be executed in one action, this action should be called `"run"`.

This approach has the advantage that most complex things happen in Python code for which tools for testing, (some) static code analysis, documentation, and style checking exist. In the Python files, we can use the whole Python tooling ecosystem whereas in the Snakemake files, tools would choke on the first `rule` keyword. In short, the `Snakefile` only serves as the entry point for your Python code.

## 39.3 Anatomy of the `cubi-snake` Executable

CUBI pipeline runs are invoked with the `cubi-snake` executable that internally calls Snakemake with sensible defaults for either local execution or execution on via SGE on an HPC cluster. It serves as a convenience wrapper that reads the current pipeline step from the current working directories `config.yaml` file (where available, otherwise you have to use the `--step` argument).

Some parameters are handed through directly to Snakemake, others are serve as macros that add more complex parameters with best pratice values or print the configuration setting.

This sounds like an aweful amount of "magic" but is quite simple and transparent, really. The generally useful `snakemake` parameters are also available to `cubi-snake` (or should be added, please create a ticket). Also, snakemake is invoked through the command line interface and a command line to copy and paste is printed at the beginning of every `cubi-snake` invocation.

CHAPTER

FORTY

# SOMATIC VARIANT CALLING DISSECTION

**Note:** Before reading this chapter, you should have

- have knowledge from the user's perspective of CUBI pipeline (start a *Usage*)
- read chapter *Developer's Introduction*.

After reading this chapter, you should

- understand the `BaseStep` and `BaseStepPart` classes and how to subclass them and override the different functions
- understand how the objects of these classes are tied into the `Snakefile` of each pipeline step
- understand how to create and use Snakemake wrappers for tools

## 40.1 Pipeline Step File Structure

Generally, all pipeline steps go into a sub module of `snappy_pipeline.workflows` (thus, a sub directory). In this chapter, we look at the `somatic_variant_calling` pipeline step. This step has the following structure on the file system:

```
somatic_variant_calling/
|-- __init__.py
`-- Snakefile
```

As you can see, it is a Python module (as it contains an `__init__.py` file) that also contains non-Python files (here `Snakefile`). The directory could also contain more files. This could be any small static data file that the module could require. Further, we could decide to factorize out the rules for a tool that requires many small rules (such as the tool cnvkit in `somatic_targeted_seq_cnv_calling`).

The code for generating input and output file lists etc. is in the `__init__.py` file and the module is available as `snappy_pipeline.workflow.somatic_variant_calling`. The `Snakefile` is used for creating the Snakemake workflow. When executing `cubi-snake` for a `somatic_variant_calling` step instance, you will note that the Snakemake command line displayed at the top will use the `--snakefile` argument and put the value to the `Snakefile` inside the `somatic_variant_calling` directory at the argument's value. Thus, `cubi-snake` is no real "magic" but simply a shortcut to the `snakemake` executable.

## 40.2 The Snakefile

We will first consider the `Snakefile`.

### 40.2.1 Necessary Imports

At the top, it starts with a line specifying UTF-8 coding and a Python docstring giving a short synopsis. It does some Python `imports` for making the *expand_ref()* function and the `SomaticVariantCallingWorkflow` class available.

```
# -*- coding: utf-8 -*-
"""CUBI Pipeline somatic_variant_calling step Snakefile"""

import os

from snappy_pipeline import expand_ref
from snappy_pipeline.workflows.somatic_variant_calling import␣
↪SomaticVariantCallingWorkflow

__author__ = "Manuel Holtgrewe <manuel.holtgrewe@bihealth.de>"
```

### 40.2.2 Configuration

Then follows the loading of the configuration. The Snakemake `configfile:` statement loads the file `config.yaml` from the current working directory. When executing the pipeline step with `cubi-snake`, this is either the directory the command is called in or the value of the `--directory` argument if given.

In the last line of this chunk, the JSON pointers in the configuration are expanded, i.e., the `"$ref"` values are interpreted. This is used for implementing "overriding behaviour", i.e., including and extending (OOP-like) the project's main configuration file with the per–step instance one. This way, you can set the `pipeline_step/name` to `somatic_variant_calling` in the somatic variant calling pipeline instance directory and to `ngs_mapping` in the ngs mapping pipeline step instance directory, for example.

```
configfile: "config.yaml"


# Expand "$ref" JSON pointers in configuration (also works for YAML)
config, lookup_paths, config_paths = expand_ref("config.yaml", config)
```

### 40.2.3 Local Rules / Rule `all`

In the next chunk, the rules that are to be executed locally and not generate any cluster jobs are defined. Then, the `all` rule is defined to obtain the list of files to generate by default using the `get_result_files()` method of the `SomaticVariantCallingWorkflow()` class.

```
localrules:
```

(continues on next page)

(continued from previous page)

```
    # Linking files from work/ to output/ should be done locally
    somatic_variant_calling_link_out_run,


rule all:
```

### 40.2.4 House-Keeping Rules

Next follow the "house-keeping" rules that do not perform any real work. In this case, the `somatic_variant_calling_link_out_run` rule performs the linking from the `work/` directory into the `output/` directory.

Note that the rule names are generated by concatenating the step name (here `somatic_variant_calling`), the part of the pipeline step (here `link_out`), followed by the action to be performed (here `run` as there is no other action for `link_out`).

```
        wf.get_result_files(),


# House-Keeping ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

# Generic linking out ------------------------------------------------------


rule somatic_variant_calling_link_out_run:
```

### 40.2.5 Rule for MuTect

Next comes the rule for running mutect. Note that both for `input:` and `output:`, `dict` values will be passed. These should be unpacked (similar to Python `**kwargs` unpacking).

As we are using an **input function** (i.e., a function object that accepts a wildcards argument), we have to use the recently introduced Snakemake `unpack` directive. This allows for lazy unpacking after the input function has been called with the wildcards argument. For the output files, no wildcards are required as only strings with placeholders are returned. Thus, the dict with key/value pairs of the named output files is to be unpacked directly using two asterisks (`**`).

```
        wf.get_input_files("link_out", "run"),
    output:
        wf.get_output_files("link_out", "run"),
    run:
        shell(wf.get_shell_cmd("link_out", "run", wildcards))


# Somatic Variant Calling ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

# Run MuTect ---------------------------------------------------------------
```

### 40.2.6 Rule for Scalpel

The rule for Scalpel looks similar. However, here the name of the normal library is required (as it is not part of the wildcards in contrast to the somatic library).

The way this is implemented here is to introduce a parameter called `normal_library_name`. The attribute `get_normal_lib_name` of the scalpel *BaseStepPart* sub class object is passed in here (which is actually a function). On execution of the rule, the function will be called with the wildcards object as the parameter. It will then lookup the name of the normal library for the matched tumor NGS library and return it. This value is then available as `params.normal_library_name`.

Also note that for scalpel, the call is not generated directly by the *BaseStepPart* sub class, but a Snakemake wrapper is used instead. This wrapper is located in the directory `../wrappers/scalpel/run`, relative to the `somatic_variant_calling` Snakefile. The method `wrapper_path()` builds the correct path relative to the `wrappers` directory in the `snappy_pipeline` directory and its return value is passed to the `wrapper:` section.

```
rule somatic_variant_calling_mutect_run:
    input:
        unpack(wf.get_input_files("mutect", "run")),
    output:
        **wf.get_output_files("mutect", "run"),
    threads: wf.get_resource("mutect", "run", "threads")
    resources:
        time=wf.get_resource("mutect", "run", "time"),
```

## 40.3 The Module

### 40.3.1 Module Documentation

The file starts with the module-level documentation. Only the first four lines are shown below. This module-level documentation is also included into the user documentation, e.g., the one for the somatic variant calling module is included at *Somatic Variant Calling*.

```
# -*- coding: utf-8 -*-
"""Implementation of the ``somatic_variant_calling`` step

The ``somatic_variant_calling`` step takes as the input the results of the ``ngs_
↪mapping`` step
```

### 40.3.2 Imports

Then follow the necessary imports for the module. Note that the classes of the steps that are used for the input are also imported (this will be important below).

```
from itertools import chain
import os
import sys

from biomedsheets.shortcuts import CancerCaseSheet, CancerCaseSheetOptions, is_not_
↪background
```

(continues on next page)

(continued from previous page)

```
from snakemake.io import expand

from snappy_pipeline.utils import dictify, listify
from snappy_pipeline.workflows.abstract import (
    BaseStep,
    BaseStepPart,
    LinkOutStepPart,
    ResourceUsage,
)
```

### 40.3.3 Constants

The imports are followed by constant definitions.

In the case of the somatic variant calling methods, the different tools generate a common set of core files, here VCF files with TBI indices and MD5 files for both. Thus, it makes sense to store the extensions (and names for named input and output file lists) in module-level constants. Note the use of tuples over lists for marking this datas explicitely as immutable.

Further, the `DEFAULT_CONFIG` constant is defined with default configuration in YAML format. This is also displayed in the user configuration so the users know where configuration settings are available. Required configuration without any defaults should be set to `null` (or `[]`/`{}` for empty lists/dicts) and marked with a `# REQUIRED` comment. The different values are to be documented with YAML comments.

This default configuration will be loaded when initializing the *BaseStep* sub class object and then overridden with the project- and pipeline step instance–wide configuration.

```
__author__ = "Manuel Holtgrewe <manuel.holtgrewe@bih-charite.de>"

#: Extensions of files to create as main payload
EXT_VALUES = (".vcf.gz", ".vcf.gz.tbi", ".vcf.gz.md5", ".vcf.gz.tbi.md5")

#: Names of the files to create for the extension
EXT_NAMES = ("vcf", "vcf_tbi", "vcf_md5", "vcf_tbi_md5")

EXT_MATCHED = {
    "mutect": {
        "vcf": ".vcf.gz",
        "vcf_md5": ".vcf.gz.md5",
        "vcf_tbi": ".vcf.gz.tbi",
        "vcf_tbi_md5": ".vcf.gz.tbi.md5",
        "full_vcf": ".full.vcf.gz",
        "full_vcf_md5": ".full.vcf.gz.md5",
        "full_vcf_tbi": ".full.vcf.gz.tbi",
        "full_vcf_tbi_md5": ".full.vcf.gz.tbi.md5",
        "txt": ".txt",
        "txt_md5": ".txt.md5",
        "wig": ".wig",
        "wig_md5": ".wig.md5",
    },
    "scalpel": {
```

(continues on next page)

(continued from previous page)

```
        "vcf": ".vcf.gz",
        "vcf_md5": ".vcf.gz.md5",
        "vcf_tbi": ".vcf.gz.tbi",
```

### 40.3.4 The BaseStep Sub Class

Let's jump towards the end of the file. Here is the *BaseStep* sub class `SomaticVariantCallingWorkflow`.

Each sub class has to configure the `name` and `sheet_shortcut_class` class members. They will be used for identifying the step name and the BioMed Sheet sheet shortcut class. The somatic variant calling step sets these to `"somatic_variant_calling"` and the `CancerCaseSheet` class.

The static method `default_config_yaml()` must be overridden in each *BaseStep* sub class. Each of these functions will have the same content but it is important for the scope of accessing `DEFAULT_CONFIG` in the current module.

```
      allow_seq_dict_incompatibility: false
      annotations:
      - BaseQualityRankSumTest
      - FisherStrand
      - GCContent
      - HaplotypeScore
      - HomopolymerRun
      - MappingQualityRankSumTest
      - MappingQualityZero
      - QualByDepth
      - ReadPosRankSumTest
      - RMSMappingQuality
```

The constructor of the class calls the super class' constructor with the arguments from the `Snakefile`. It is very important to note that it also gets an iterable (here a one-element tuple) of the *BaseStep* sub classes that provide input for this step (here only `NGSMappingWorkflow` imported at the top). This information is required for making the default configuration of these steps available.

The constructor then proceeds to register the sub step classes that are used to implement the actual behaviour of the pipeline step. Here, it is for running MuTect, Scalpel, and linking out the somatic variant call VCF files from `work/` into `output/`.

```
      - Coverage
      - ClippingRankSumTest
      - DepthPerSampleHC
    gatk_ug_joint:
      # Parallelization configuration
      num_cores: 2              # number of cores to use locally
      window_length: 50000000   # split input into windows of this size, each triggers a␣
↪job
      num_jobs: 500             # number of windows to process in parallel
      use_profile: true         # use Snakemake profile for parallel processing
      restart_times: 5          # number of times to re-launch jobs in case of failure
```

The method `get_result_files()` returns a list of result files of this pipeline step. For this, it uses the `_yield_result_files_()` helper method and generates path in the `output/` folder. Snakemake knows that the link out rule will create these files but need corresponding files in `work/` for this. Through this mechanism, the individual tools' rules will be triggered.

Note the use of `cubi.utils.listify()` decorator that converts a generator (as created by using `yield` in the function) to a function returning a list with the yielded objects in the ordere that they are yielded.

```
      debug_trunc_tokens: 0     # truncation to first N tokens (0 for none)
      keep_tmpdir: never        # keep temporary directory, {always, never, onerror}
      job_mult_memory: 1        # memory multiplier
      job_mult_time: 1          # running time multiplier
      merge_mult_memory: 1      # memory multiplier for merging
      merge_mult_time: 1        # running time multiplier for merging
      # GATK UG--specific configuration
      downsample_to_coverage: 250
      allow_seq_dict_incompatibility: false
      annotations:
      - BaseQualityRankSumTest
      - FisherStrand
      - GCContent
      - HaplotypeScore
      - HomopolymerRun
      - MappingQualityRankSumTest
      - MappingQualityZero
      - QualByDepth
      - ReadPosRankSumTest
      - RMSMappingQuality
      - DepthPerAlleleBySample
      - Coverage
      - ClippingRankSumTest
      - DepthPerSampleHC
```

Finally, the `check_config()` implementation ensures that the path to the NGS mapping step is configured for the somatic variant calling step.

```
      window_length: 5000000    # split input into windows of this size, each triggers a␣
↪job
      num_jobs: 500             # number of windows to process in parallel
      use_profile: true         # use Snakemake profile for parallel processing
      restart_times: 5          # number of times to re-launch jobs in case of failure
      max_jobs_per_second: 2    # throttling of job creation
```

### 40.3.5 Module-Level BaseStepPart Sub Class

Now, we move up towards the top of the file again.

The `SomaticVariantCallingStepPart` class is the base class for the somatic variant calling implementations. The constructor builds a template string for generating result/output paths. It then builds the member `cancer_ngs_library_to_sample_pair` with a mapping from tumor DNA NGS library name to the BioMed Sheets `CancerSamplePair` object that contains information about both the tumor and normal sample. Note the use of `OrderedDict` to keep the order from the sample sheet definition.

```
        "full_vcf_tbi": ".full.vcf.gz.tbi",
        "full_vcf_tbi_md5": ".full.vcf.gz.tbi.md5",
        "tar": ".tar.gz",
        "tar_md5": ".tar.gz.md5",
    },
```

(continues on next page)

(continued from previous page)

```
    "mutect2": {
        "vcf": ".vcf.gz",
        "vcf_md5": ".vcf.gz.md5",
        "vcf_tbi": ".vcf.gz.tbi",
        "vcf_tbi_md5": ".vcf.gz.tbi.md5",
        "full_vcf": ".full.vcf.gz",
        "full_vcf_md5": ".full.vcf.gz.md5",
        "full_vcf_tbi": ".full.vcf.gz.tbi",
        "full_vcf_tbi_md5": ".full.vcf.gz.tbi.md5",
    },
}
```

The implementation of `get_input_files()` returns an input function that given the wildcards returns a dict with paths to the normal and tumor libraries' aligned BAM file from the sub workflow `ngs_mapping`. Note that the input function returns the actual path without any wildcards.

```
SOMATIC_VARIANT_CALLERS_MATCHED = ("mutect", "mutect2", "scalpel")

#: Available somatic variant callers that just call all samples from one donor together.
SOMATIC_VARIANT_CALLERS_JOINT = (
    "bcftools_joint",
    "platypus_joint",
    "gatk_hc_joint",
    "gatk_ug_joint",
    "varscan_joint",
)

#: Available somatic variant callers
SOMATIC_VARIANT_CALLERS = tuple(
    chain(SOMATIC_VARIANT_CALLERS_MATCHED, SOMATIC_VARIANT_CALLERS_JOINT)
)

#: Available somatic variant callers assuming matched samples.
SOMATIC_VARIANT_CALLERS_MATCHED = ("mutect", "mutect2", "scalpel", "strelka2")

#: Available somatic variant callers that just call all samples from one donor together.
```

The method `get_normal_lib_name()` returns the name of the matched normal NGS library for the given tumor NGS library name.

```
    "bcftools_joint",
    "platypus_joint",
    "gatk_hc_joint",
    "gatk_ug_joint",
```

The implementation of `get_output_files()` returns a dict with named output files for Snakemake. Note that this function returns named output files with wildcard placeholders.

```
)

#: Default configuration for the somatic_variant_calling schema
DEFAULT_CONFIG = r"""
```

(continues on next page)

(continued from previous page)

```
# Default configuration somatic_variant_calling
step_config:
  somatic_variant_calling:
```

Finally, the method `get_log_file()` returns the path to the log file to create by Snakemake. This is available as `{log}` in shell commands and as `{snakemake.log}` in Snakemake wrappers.

```
    path_ngs_mapping: ../ngs_mapping  # REQUIRED
    ignore_chroms:            # patterns of chromosome names to ignore
    - NC_007605    # herpes virus
```

### 40.3.6 MuTect BaseStepPart Sub Class

First, the class defines the class attribute `name` and sets it to `'mutect'`. This is used by the super class and also by `BaseStep` in the places where the name of the implementation is needed.

The `check_config()` implementation ensures that the necessary MuTect-specific configuration has been set.

```
    - '*_decoy'    # decoy contig
    - 'HLA-*'      # HLA genes
    - 'GL000220.*' # Contig with problematic, repetitive DNA in GRCh37
    # Configuration for joint calling with samtools+bcftools.
    bcftools_joint:
      max_depth: 4000
      max_indel_depth: 4000
      window_length: 10000000
      num_threads: 16
    # Configuration for joint calling with Platypus.
    platypus_joint:
      split_complex_mnvs: true  # whether or not to split complex and MNV variants
      num_threads: 16
    # VCF annotation databases are given as mapping from name to
    #   {'file': '/path.vcf.gz',
    #    'info_tag': 'VCF_TAG',
    #    'description': 'VCF header description'}
```

The function `get_shell_cmd()` generates the shell command to the CUBI wrapper (not Snakemake wrapper ;) to the MuTect tool. Here, the parallel CUBI wrapper for MuTect is used with appropriate configuration. Note the direct use of configuration and that no complex string operations are required for building the call to MuTect.

```
    mutect:
      # Parallelization configuration
      num_cores: 2               # number of cores to use locally
      window_length: 3500000     # split input into windows of this size, each triggers␣
↪a job
      num_jobs: 500              # number of windows to process in parallel
      use_profile: true          # use Snakemake profile for parallel processing
      restart_times: 5           # number of times to re-launch jobs in case of failure
      max_jobs_per_second: 2     # throttling of job creation
      max_status_checks_per_second: 10   # throttling of status checks
      debug_trunc_tokens: 0      # truncation to first N tokens (0 for none)
      keep_tmpdir: never         # keep temporary directory, {always, never, onerror}
```

(continues on next page)

(continued from previous page)

```
    job_mult_memory: 1          # memory multiplier
    job_mult_time: 1            # running time multiplier
    merge_mult_memory: 1        # memory multiplier for merging
    merge_mult_time: 1          # running time multiplier for merging
  # Configuration for MuTect 2
  mutect2:
    panel_of_normals: ''      # Set path to panel of normals vcf if required
    germline_resource: ''     # Germline variants resource (same as panel of normals)
    common_variants: ''       # Common germline variants for contamination estimation
    extra_arguments: []       # List additional Mutect2 arguments
                              # Each additional argument xust be in the form:
                              # "--<argument name> <argument value>"
                              # For example, to filter reads prior to calling & to
                              # add annotations to the output vcf:
                              # - "--read-filter CigarContainsNoNOperator"
```

Finally, the method `update_cluster_config()` takes the Snakemake `cluster` configuration and updates it appropriately. The three settings `cluster.h_vmem`, `cluster.h_rt`, and `cluster.pe` are used in the generated Snakemake command line generated by the `cubi-snake` tool appropriately for SGE.

```
    # Parallelization configuration
    num_cores: 2              # number of cores to use locally
    window_length: 50000000   # split input into windows of this size, each triggers a␣
↪job
    num_jobs: 500             # number of windows to process in parallel
    use_profile: true         # use Snakemake profile for parallel processing
    restart_times: 5          # number of times to re-launch jobs in case of failure
```

### 40.3.7 Scalpel BaseStepPart Sub Class

The integration for Scalpel is even simpler as Snakemake wrappers can be used and there is no need for the `get_shell_cmd()` function. The class sets the class attribute `name` to `'scalpel'`. Then, the `check_config()` implementation ensure the presence of the required configuration.

```
    debug_trunc_tokens: 0     # truncation to first N tokens (0 for none)
    keep_tmpdir: never        # keep temporary directory, {always, never, onerror}
    job_mult_memory: 1        # memory multiplier
    job_mult_time: 1          # running time multiplier
    merge_mult_memory: 1      # memory multiplier for merging
    merge_mult_time: 1        # running time multiplier for merging
  # Configuration for Scalpel
  scalpel:
    path_target_regions: REQUIRED  # REQUIRED
  # Configuration for strelka2
  strelka2:
```

Output file generation is similarly easy as for the MuTect part. However, the Scalpel working directory is tar-gzed in case the users wants to have access to the intermediate results and query the built MuTect database for variants with different coverages.

```
    gatk_hc_joint:
      # Parallelization configuration
      num_cores: 2              # number of cores to use locally
      window_length: 50000000   # split input into windows of this size, each triggers a␣
↪job
      num_jobs: 500             # number of windows to process in parallel
      use_profile: true         # use Snakemake profile for parallel processing
      restart_times: 5          # number of times to re-launch jobs in case of failure
      max_jobs_per_second: 10   # throttling of job creation
      max_status_checks_per_second: 10   # throttling of status checks
```

The `update_cluster_config()` method's implementation is also very simple.

```
      keep_tmpdir: never        # keep temporary directory, {always, never, onerror}
      job_mult_memory: 1        # memory multiplier
      job_mult_time: 1          # running time multiplier
      merge_mult_memory: 1      # memory multiplier for merging
      merge_mult_time: 1        # running time multiplier for merging
      # GATK HC--specific configuration
```

CHAPTER

# FORTYONE

# NGS MAPPING DISSECTION

This chapter gives a dissection of the NGS Mapping step (`ngs_mapping`) for CUBI pipeline developers. The NGS mapping step is an example for a pipeline step that works on the raw FASTQ NGS read files. This chapter assumes that you have read *Somatic Variant Calling Dissection* before.

The minority of pipeline steps will work directly with the raw read data. Most steps work on the results of the NGS read mapping step or even further downstream.

---

**Note:** Before reading this chapter, you should

- have knowledge from the user's perspective of CUBI pipeline (start a *Usage*)
- have read chapter *Developer's Introduction*
- have read chapter *Somatic Variant Calling Dissection*.

After reading this chapter, you should

- know how to work with raw FASTQ file input
- know how to use the using the `LinkInStep`

---

**This is still TODO, just look at the code for now ;)**

CHAPTER

# FORTYTWO

# API DOCUMENTATION

## 42.1 snappy_pipeline.base

Basic utility code for snappy_pipeline

**exception** snappy_pipeline.base.**InvalidConfiguration**
Raised on invalid configuration

**exception** snappy_pipeline.base.**MissingConfiguration**
Raised on missing configuration

**exception** snappy_pipeline.base.**SkipLibraryWarning**
Raised when libraries are skipped.

**exception** snappy_pipeline.base.**UnknownFiltrationSourceException**
Raised when user try to request an unknown filtration source.

**exception** snappy_pipeline.base.**UnsupportedActionException**
Raised when user try to call action that isn't supported.

snappy_pipeline.base.**expand_ref**(*config_path*, *dict_data*, *lookup_paths=None*, *dict_class=<class 'collections.OrderedDict'>*)
Expand "$ref" in JSON-like data `dict_data`

Returns triple:

- path to resolved file
- paths containing included config files
- config files included

snappy_pipeline.base.**merge_dicts**(*dict1*, *dict2*, *dict_class=<class 'collections.OrderedDict'>*)
Merge dictionary `dict2` into `dict1`

snappy_pipeline.base.**merge_kwargs**(*first_kwargs*, *second_kwargs*)
Merge two keyword arguments.

**Parameters**

- **first_kwargs** (*dict*) – First keyword arguments dictionary.
- **second_kwargs** (*dict*) – Second keyword arguments dictionary.

**Returns** Returns merged dictionary with inputted keyword arguments.

snappy_pipeline.base.**print_config**(*config*, *file=<_io.TextIOWrapper name='<stderr>' mode='w' encoding='UTF-8'>*)
Print human-readable version of configuration to `file`

snappy_pipeline.base.**print_sample_sheets**(*step*, *file=<_io.TextIOWrapper name='<stderr>' mode='w' encoding='UTF-8'>*)
Print loaded sample sheets from `BaseStep` in human-readable format

snappy_pipeline.base.**snakefile_path**(*step_name*)
Return absolute path to Snakefile for the given step name

## 42.2 snappy_pipeline.find_file

Code for crawling the file system and caching the results

**exception** snappy_pipeline.find_file.**FileNamesTooDifferent**
Raised when two file names are too different to be PE reads

**class** snappy_pipeline.find_file.**FileSystemCrawler**(*cache_path*, *invalidation_paths*, *lock_timeout=60*)
Crawl the file system

- start crawling the file system from a given directory
- look for files matching a given `PatternSet`
- that are below a directory with a given name

**cache**
The actual dict with the cache, loaded from path to `cache_path` if the cache file exists.

**cache_dirty**
Flag whether cache has been modified and needs saving

**cache_invalidated**
Flag whether cache has been invalidated already.

**cache_path**
Path to cache (will be stored in JSON format)

**invalidation_paths**
Path to files to use for checking invalidation.

**lock_timeout**
Timeout for obtaining file system lock on the file system

**logger**
The logger to use.

**run**(*root_dir*, *dir_name*, *pattern_sets*, *allow_empty_right*)
Perform the file system crawling from a root directory given a query pattern set

**allow_empty_right – for mixed PE/SE read data sets (must be either SE or PE** for one library!)

**save_cache**(*cache_path=None*)
Save cache, `cache_path` overriding `self.cache_path`

**class** snappy_pipeline.find_file.**FileSystemCrawlerResult**(*base_folder*, *files*, *names=None*)
n-tuple of optionally named files

**base_folder**
Folder to start crawling in

**files**
Patterns to search for

**named_files**
Dict with name-to-pattern mapping, `None` if `names` is not given

**names**
Names for the file patterns, optional; if given has to have the same length as files

**to_dict**()
Convert to dict, can only work if self.names and self.files is given

class snappy_pipeline.find_file.**PatternSet**(*patterns*, *names=None*)
Store named or unnamed list of patterns

**named_patterns**
Named patterns, if any, else `None`

**names**
Optional names

**patterns**
Patterns to search for with names

## 42.3 snappy_pipeline.utils

Utility code

class snappy_pipeline.utils.**DictQuery**
Helper class for comfortable access to nested dicts with `str` keys.

Source:

- https://www.haykranen.nl/2016/02/13/handling-complex-nested-dicts-in-python/

**get**(*path*, *default=None*)
Return the value for key if key is in the dictionary, else default.

snappy_pipeline.utils.**dictify**(*gen*)
Decorator that converts a generator into a function which returns a dict

Use it in the case where a generator is easier to write but you want to enforce returning a dict:

```
@listify
def counter(max_no):
    i = 0
    while i <= max_no:
        yield 'key{}'.format(i), i
```

snappy_pipeline.utils.**flatten**(*coll: List[Union[str, List[str]]]*) → List[str]
Flatten collection of strings or list of strings.

Source: https://stackoverflow.com/a/17865033

snappy_pipeline.utils.**is_none**(*value*)
Helper function returning whether `value is None`

snappy_pipeline.utils.**is_not_none**(*value*)
Helper function returning whether `value is not None`

snappy_pipeline.utils.**listify**(*gen*)
Decorator that converts a generator into a function which returns a list

Use it in the case where a generator is easier to write but you want to enforce returning a list:

```
@listify
def counter(max_no):
    i = 0
    while i <= max_no:
        yield i
```

snappy_pipeline.utils.**try_or_none**(*func*, *exceptions*)
Helper that tries to execute the function

If one of the `exceptions` is raised then return None

## 42.4 snappy_pipeline.workflows.abstract

Base classes for the actual pipeline steps

**class** snappy_pipeline.workflows.abstract.**BaseStep**(*workflow*, *config*, *config_lookup_paths*, *config_paths*, *work_dir*, *previous_steps=None*)
Base class for the pipeline steps

Each pipeline step is a Snakemake workflow

**check_config**()
Check `self.w_config`, raise `ConfigurationMissing` on problems

Override in sub classes.

**Raises:MissingConfiguration** on missing configuration

**config_lookup_paths**
Paths with configuration paths, important for later retrieving sample sheet files

**config_paths**
Tuple with absolute paths to configuration files read

**classmethod default_config_yaml**()
Override this function for providing default configuration

The configuration should be a YAML fragment. Your configuration should define a top-level key starting with '_' and then consist of the name of the schema, e.g., '_ngs_mapping_schema'. Your default configuration is then merged into the main configuration where the main configuration takes precedence.

Example:

```
def default_config_yaml(self):
    return textwrap.dedent("""
        schema_config:
          ngs_mapping:
            max_threads: 16
    """).lstrip())
```

Return `None` for no default configuration.

You can also return an iterable of configurations, these will be merged in the order given (earlier ones will be overwritten by later ones). This is useful if your schema needs configuration for a later one.

**ensure_w_config**(*config_keys*, *msg*, *e_class=<class 'snappy_pipeline.base.MissingConfiguration'>*)
Check parameters in configuration.

Method ensures required configuration setting are present in the provided configuration; if not, it raises exception.

**Parameters** **config_keys** – List of strings with all keys that must be present in the configuration

for a given step of the analysis to be performed. :type config_keys: tuple

**Parameters**

- **msg** (*str*) – Message to be used in case of exception.
- **e_class** – Preferred exception class to be raised in case of error.

Default: MissingConfiguration. :type e_class: class

**get_args**(*sub_step*, *action*)
Return arguments for action of substep with given wildcards

Delegates to the sub step object's get_input_files function

**get_input_files**(*sub_step*, *action*)
Return input files for action of substep with given wildcards

Delegates to the sub step object's get_input_files function

**get_log_file**(*sub_step*, *action*)
Return path to the log file

Delegates to the sub step object's get_log_file function

**get_output_files**(*sub_step*, *action*)
Return list of strings with output files/patterns

Delegates to the sub step object's get_output_files function

**get_params**(*sub_step*, *action*)
Return parameters

Delegates to the sub step object's get_params function

**get_resource**(*sub_step*, *action*, *resource_name*)
Get resource

Delegates to the sub step object's get_resource function

**get_result_files**()
Return actual list of file names to build

**get_shell_cmd**(*sub_step*, *action*, *wildcards*)
Return shell command for the pipeline sub step

Delegates to the sub step object's get_shell_cmd function

**get_tmpdir**()
Return temporary directory.

To be used directly or via get_resource("step", "action", "tmpdir")

1. Try to evaluate global_config/tmpdir. Interpret $-variables from environment. Provides the current date as $TODAY.
2. If this fails, try to use environment variable TMPDIR.
3. If this fails, use tempfile.gettempdir(), same as Snakemake default.

**name = None**
Override with step name

**previous_steps**
Classes of previously executed steps, used for merging their default configuration as well.

**register_sub_step_classes**(*classes*)
Register an iterable of sub step classes

Initializes objects in `self.sub_steps` dict

**register_sub_workflow**(*step_name*, *workdir*, *sub_workflow_name=None*)
Register workflow with given pipeline `step_name` and in the given `workdir`.

Optionally, the sub workflow name can be given separate from `step_name` (the default) value for it.

**run**(*sub_step*, *action*, *wildcards*)
Run command for the given action of the given sub step with the given wildcards

Delegates to the sub step object's run function

**sheet_shortcut_args = None**
Override with arguments to pass into sheet shortcut class constructor

**sheet_shortcut_class = None**
Override with the sheet shortcut class to use

**sheet_shortcut_kwargs = None**
Override with keyword arguments to pass into sheet shortcut class constructor

**sheets**
Shortcut to the BioMed SampleSheet objects

**shortcut_sheets**
Shortcut sheets

**sub_workflows**
Functions from sub workflows, can be used to generate output paths into these workflows

**substep_dispatch**(*step*, *function*, **args*, ***kwargs*)
Dispatch call to function of sub step implementation

**substep_getattr**(*step*, *name*)
Return attribute from substep

**w_config**
Merge default configuration with true configuration

**work_dir**
Absolute path to directory of where to perform work

**workflow**
Snakefile "workflow" object

**classmethod wrapper_path**(*path*)
Generate path to wrapper

**class** snappy_pipeline.workflows.abstract.**BaseStepPart**(*parent*)
Base class for a part of a pipeline step

**actions: Tuple[str] = None**
The actions available in the class.

**check_config**()
Check configuration, raise `ConfigurationMissing` on problems

Override in sub classes.

**Raises:MissingConfiguration** on missing configuration

`default_resource_usage: snappy_wrappers.resource_usage.ResourceUsage = ResourceUsage(threads=1, time='01:00:00', memory='2G', partition=None, tmpdir=None)`
Default resource usage for actions that are not given in `resource_usage`.

`get_args`(*action*)
Return args for the given action of the sub step

`static get_default_partition`() → str
Helper that returns the default partition.

`get_input_files`(*action*)
Return input files for the given action of the sub step

`get_log_file`(*action*)
Return path to log file

The default implementation tries to call `self._get_log_files()` and in the case of this function returning a `dict`, augments it with paths to MD5 files.

`get_output_files`(*action*)
Return output files for the given action of the sub step and

`get_resource`(*action: str*, *resource_name: str*)
Return the amount of resources to be allocated for the given action.

**Parameters**

- **action** – The action to return the resource requirement for.
- **resource_name** – The name to return the resource for.

`get_resource_usage`(*action: str*) → snappy_wrappers.resource_usage.ResourceUsage
Return the resource usage for the given action.

`get_shell_cmd`(*action*, *wildcards*)
Return shell command for the given action of the sub step and the given wildcards

`resource_usage: Dict[str, snappy_wrappers.resource_usage.ResourceUsage] = {}`
Configure resource usage here that should not use the default resource usage from `default_resource_usage`.

`run`(*action*, *wildcards*)
Run the sub steps action action's code with the given wildcards

`class snappy_pipeline.workflows.abstract.DataSearchInfo`(*sheet_path: str*, *base_paths: list*, *search_paths: list*, *search_patterns: list*, *mixed_se_pe: bool*)
Data search information - simplified version of `DataSetInfo`.

`class snappy_pipeline.workflows.abstract.DataSetInfo`(*name*, *sheet_path*, *base_paths*, *search_paths*, *search_patterns*, *sheet_type*, *is_background*, *naming_scheme*, *mixed_se_pe*, *sodar_uuid*, *sodar_title*, *pedigree_field=None*)
Information on a DataSet

`base_paths`
All base paths of all configuration, to look for `sheet_path`

`is_background`
Whether the data set info is to be used only for background

**mixed_se_pe**
Whether mixing SE and PE data sets is allowed.

**name**
Name of the data set

**pedigree_field_kwargs**
The (optional) custom field used to define pedigree

**search_paths**
Search paths for the files in the sample sheet

**search_patterns**
Search patterns

**sheet**
The BioMed SampleSheet

**sheet_path**
Path to the sheet file, for loading

**sodar_title**
The (optional) title of the project in SODAR.

**sodar_uuid**
The UUID of the corresponding SODAR project.

**exception** snappy_pipeline.workflows.abstract.**ImplementationUnavailableError**
Raised when a function that is to be overridden optionally is called

This is provided as an alternative to `NotImplementedError` as the Python linters warn if a class does not override functions throwing `NotImplementedError`.

**class** snappy_pipeline.workflows.abstract.**InputFilesStepPartMixin**
Mixin with predefined "get_input_files" function.

**ext_names = None**
Names of the files to create for the extension

**ext_values = None**
Extensions of files to create as main payload

**include_ped_file = None**
Whether to include path to PED file or not

**prev_class = None**
Class with input VCF file name

**class** snappy_pipeline.workflows.abstract.**LinkInBaiExternalStepPart**(*parent*)
Link in the external BAI files.

**actions: Tuple[str] = ('run',)**
Class available actions

**name = 'link_in_bai_external'**
Step name

**pattern_set_keys = ('bai', 'bai_md5')**
Patterns set keys

**class** snappy_pipeline.workflows.abstract.**LinkInBamExternalStepPart**(*parent*)
Link in the external BAM files.

**actions: Tuple[str] = ('run',)**
Class available actions

**name = 'link_in_bam_external'**
Step name

**pattern_set_keys = ('bam', 'bam_md5')**
Patterns set keys

class snappy_pipeline.workflows.abstract.**LinkInPathGenerator**(*work_dir*, *data_set_infos*, *config_paths*, *cache_file_name='.snappy_path_cache'*, *preprocessed_path=''*)

Helper class for generating paths to link in

**cache_file_name**
Name of cache file to create

**config_paths**
Path to configuration files, used for invalidating cache

**run**(*folder_name*, *pattern_set_keys=('left', 'right', 'left_md5', 'right_md5', 'bam')*)
Yield (src_path, path_infix, filename) one-by-one

Cache is saved after the last iteration

**work_dir**
Working directory

class snappy_pipeline.workflows.abstract.**LinkInStep**(*parent*)
Link in the raw files, e.g. FASTQ files

Depending on the configuration, the files are linked out after postprocessing

**get_input_files**(*action*)
Return required input files

**get_output_files**(*action*)
Return output files for the given action of the sub step and

**get_shell_cmd**(*action*, *wildcards*)
Return call for linking in the files

The files are linked, keeping their relative paths to the item matching the "folderName" intact.

**run**(*action*, *wildcards*)
Run the sub steps action action's code with the given wildcards

class snappy_pipeline.workflows.abstract.**LinkInVcfExternalStepPart**(*parent*)
Link in the external VCF files.

**actions: Tuple[str] = ('run',)**
Class available actions

**get_shell_cmd**(*action*, *wildcards*)
Return call for linking in the files

The files are linked, keeping their relative paths to the item matching the "folderName" intact.

**name = 'link_in_vcf_external'**
Step name

**pattern_set_keys = ('vcf', 'vcf_md5')**
Patterns set keys

class snappy_pipeline.workflows.abstract.**LinkOutStepPart**(*parent*, *disable_patterns=None*)
Generically link out

This is for output files that are created unconditionally, i.e., for output files where the output name is the same as for the work file.

**disable_patterns**
Patterns for disabling linking out to. This is useful/required when there is a specialized link out step part, e.g., for the case of alignment where realignment is performed or not, depending on the configuration.

**get_input_files**(*action*)
Return input file pattern

**get_output_files**(*action*)
Return output file pattern

**get_shell_cmd**(*action*, *wildcards*)
Return call for linking out

snappy_pipeline.workflows.abstract.**STDERR_TO_LOG_FILE = '#
----------\n# Redirect stderr to log file and enable printing executed commands\nexec 2> >(tee -a "{log}")\nset -x\n# ----------\n\n'**
String constant with bash command for redirecting stderr to `{log}` file

class snappy_pipeline.workflows.abstract.**WritePedigreeSampleNameStepPart**(**args*, ***kwargs*)
Class contains method to write pedigree file for primary DNA sample given the index NGS library name.It will create pedigree information based sole on sample name, example 'P001' instead of 'P001-N1-DNA1-WGS1'.

**name = 'write_pedigree_with_sample_name'**
Step name

**run**(*wildcards*, *output*)
Write out the pedigree information

**Parameters**

- **wildcards** (*snakemake.io.Wildcards*) – Snakemake wildcards associated with rule (unused).
- **output** (*snakemake.io.Namedlist*) – Snakemake output associated with rule.

class snappy_pipeline.workflows.abstract.**WritePedigreeStepPart**(*parent*, *require_dna_ngs_library=False*, *only_trios=False*)
Write out pedigree file for primary DNA sample given the index NGS library name

**actions: Tuple[str] = ('run',)**
Class available actions

**get_input_files**(*action*)
Returns function returning input files.

Returns a dict with entry `"bam"` mapping to list of input BAM files. This list will be empty if the parent step does not define an `"ngs_mapping"` workflow.

**get_output_files**(*action*)
Return output files for the given action of the sub step and

**name = 'write_pedigree'**
Step name

**require_dna_ngs_library**
Whether to prevent writing out of samples with out NGS library.

**run**(*wildcards*, *output*)
Write out the pedigree information

**Parameters**

- **wildcards** (*snakemake.io.Wildcards*) – Snakemake wildcards associated with rule (unused).
- **output** (*snakemake.io.Namedlist*) – Snakemake output associated with rule.

snappy_pipeline.workflows.abstract.**get_ngs_library_folder_name**(*sheets*, *library_name*)
Return library's folder name

The library is searched for based on the `library_name`. In the case of multiple NGS library matches, the first one is returned.

snappy_pipeline.workflows.abstract.**modified_environ**(**remove*, ***update*)
Temporarily updates the `os.environ` dictionary in-place.

The `os.environ` dictionary is updated in-place so that the modification is sure to work in all situations.

**Parameters**

- **remove** – Environment variables to remove.
- **update** – Dictionary of environment variables and values to add/update.

Source: https://stackoverflow.com/a/34333710/84349

CHAPTER

FORTYTHREE

# CONTRIBUTING

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

## 43.1 Types of Contributions

### 43.1.1 Report Bugs

Report bugs at https://github.com/bihealth/snappy-pipeline/issues

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

### 43.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

### 43.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

### 43.1.4 Write Documentation

CUBI Pipeline could always use more documentation, whether as part of the official CUBI Pipeline docs, in docstrings, or even on the web in blog posts, articles, and such.

### 43.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/bihealth/snappy-pipeline/issues

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

## 43.2 Get Started!

Ready to contribute? Here's how to set up *snappy-pipeline* for local development.

1. Fork the *snappy_pipeline* repo on BIH GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:bihealth/snappy-pipeline.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv snappy-pipeline
$ cd snappy-pipeline/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 snappy-pipeline tests
$ python setup.py test or py.test
$ tox
```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

## 43.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 3.7, 3.8 and 3.9, and for PyPy. Check https://travis-ci.org/holtgrewe/cubi_piepline/pull_requests and make sure that the tests pass for all supported Python versions.

## 43.4 Tips

To run a subset of tests:

```
$ py.test tests.test_snappy_pipeline
```

CHAPTER

FORTYFOUR

# HOW TO: RELEASE

1. Update the version in the following files:
   - `installation.rst` (look for `snappy_pipeline.git@VERSION`)
   - **TODO** more?
2. Create a tag and push it

```
$ git tag v0.1.0
$ git push --tags origin
```

That's it, so far we don't create packages or deploy the documentation.

CHAPTER

FORTYFIVE

# CREDITS

## 45.1 Active Contributors

- Eric Blanc <eric.blanc@bihealth.de>
- Manuel Holtgrewe <manuel.holtgrewe@bihealth.de>
- Clemens Messerschmidt <clemens.messerschmidt@bihealth.de>
- Nina Thiessen <nina.thiessen@bihealth.de>
- Oliver Stolpe <oliver.stolpe@bihealth.de>

## 45.2 Former Contributors

- Oliver Drechsel <oliver.drechsel@bihealth.de>

CHAPTER

FORTYSIX

# CHANGELOG

CHAPTER

FORTYSEVEN

# LICENSE

# PYTHON MODULE INDEX

## S

# INDEX

## I



## L



## M

## N



## P



## R

## T



## U



## W