mwc Documentation

Release 0.7.1-dev

Fredrik Boulund, Lisa Olsson

Jun 14, 2023

Contents:

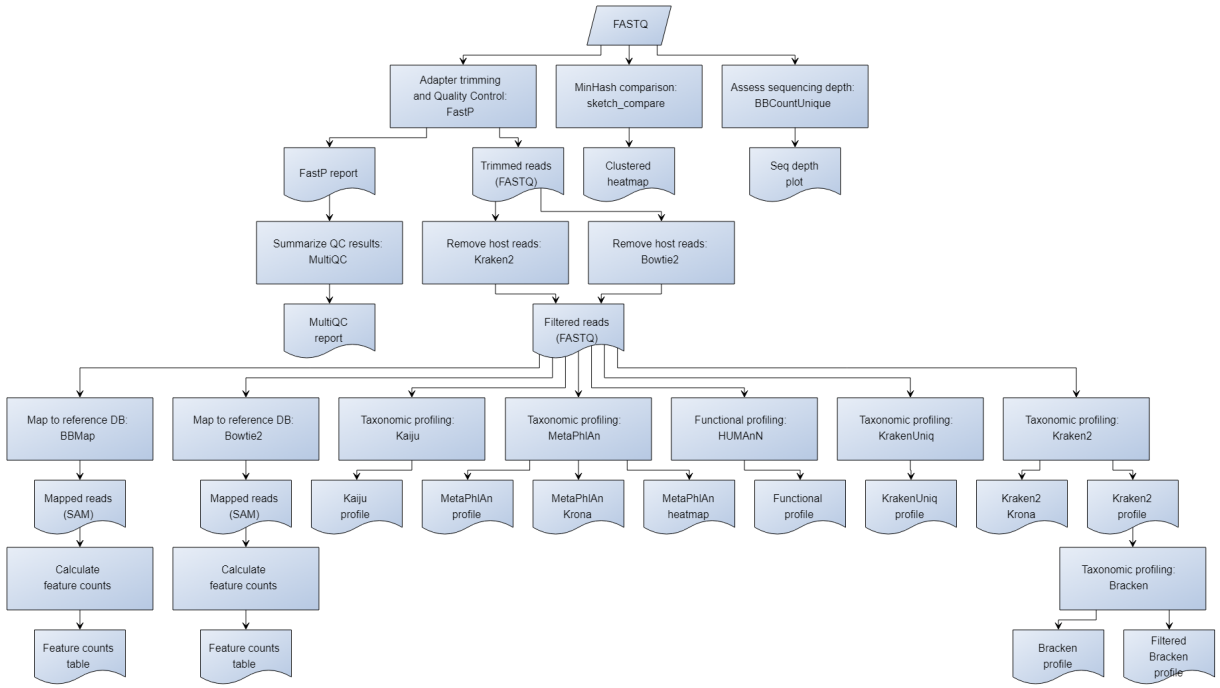
The StaG metagenomic workflow collaboration (mwc) is a Snakemake implementation of a general metagenomic analysis workflow intended for microbiome research. It started out as a joint project between The Center for Translational Microbiome Research (CTMR) at Karolinska Institutet in Stockholm and Fredrik Bäckhed's research group at Sahlgrenska in Gothenburg, and has since evolved into the default workflow for metagenomics analyses used at CTMR.

CHAPTER 1

Overview of StaG-mwc

StaG-mwc aims to be a universal starting point for analysis of shotgun metagenomics data. It started out as a small workflow intended to easily evaluate and compare the results from several different taxonomic profiling tools to each other, but has since grown in scope to become a workflow that runs several basic analyses on shotgun metagenomics data that can be used to produce a set of primary analysis results that are useful to have before starting further downstream analyses.

The following image shows a simplified graph of the workflow of StaG-mwc:



CHAPTER 2

Installing StaG-mwc

StaG-mwc depends on a number of individual tools, most prominently it requires Snakemake, which is used to manage the overall analysis workflow. In addition, another important component for the StaG-mwc framework is Conda, which is used to manage additional dependencies. This documentation assumes you already have git installed.

2.1 Install conda and Snakemake

The first two things you need to install are:

1. Conda
2. Snakemake

The recommended way to get started using StaG-mwc is to download and install Conda. A good starting point is a clean miniconda3 installation. Miniconda3 is quick to install and does not require administrator permissions. Please also consider setting up your environment for Bioconda.

After installing Conda and activating the base environment, install snakemake into your base environment:

```
(base)$ conda install -c bioconda -c conda-forge snakemake
```

These are the only external dependencies you need to install manually. The correct versions of any remaining dependencies will be automatically downloaded and installed when you run the workflow the first time.

2.2 Download the workflow code

Clone the StaG-mwc repository using git to get a copy of the workflow:

```
(base)$ git clone https://www.github.com/ctmrbio/stag-mwc
```

This will clone the repo into a folder called `stag-mwc` inside your current working directory. You will manage all workflow-related business from inside this folder (i.e. configuring and running the workflow).

The intended way of working with StaG-mwc is that you download/clone a complete copy of the repository for each analysis you intend to make. That way, the local copy you have will remain after the analysis has been run, so you can go back and see exactly what was run, and how. This forms the basis of how Snakemake enables traceability and reproducibility.

2.3 Congratulations

You have now installed StaG-mwc.

CHAPTER 3

Running StaG-mwc

You need to configure a workflow before you can run StaG-mwc. The code you downloaded in the previous `git clone` step includes a file called `config.yaml`, which is used to configure the workflow.

3.1 Selecting input files

There are two ways to define which files StaG-mwc should run on: either by specifying an input directory and a filename pattern, or by providing a sample sheet. The two ways are exclusive and cannot be combined, so you have to pick the one that suits you best.

3.1.1 Input directory

If your input FASTQ files are all in the same folder and they all follow the same filename pattern, the input directory option is often the most convenient.

Open `config.yaml` in your favorite editor and change input file settings under the `Run configuration` heading: the input directory, the input filename pattern. They can be declared using absolute or relative filenames (relative to the StaG-mwc repository directory). Input and output directories can technically be located anywhere, i.e. their locations are not restricted to the repository folder, but it is recommended to keep them in the repository directory. A common practice is to put symlinks to the files you want to analyze in a folder called `input` in the repository folder.

3.1.2 Samplesheet

If your input FASTQ files are spread across several filesystem locations or potentially exist in remote locations (e.g. S3), or your input FASTQ filenames do not follow a common filename pattern, the samplesheet option is the most convenient. The samplesheet input option also allows you to specify custom sample names that are not derived from a substring of the input filenames.

The format of the samplesheet is tab-separated text and it must contain a header line with at least the following three columns: `sample_id`, `fastq_1`, and `fastq_2`. An example file could look like this (columns are separated by TAB characters):

sample_id	fastq_1	fastq_2
ABC123	/path/to/sample1_1.fq.gz	/path/to/sample1_2.fq.gz
DEF456	s3://bucketname/sample_R1.fq.gz	s3://bucketname/sample_R2.fq.gz
GHI789	http://domain.com/sample_R1.fq.gz	http://domain.com/sample_R2.fq.gz

Open `config.yaml` in your favorite editor and enter the path to a samplesheet TSV file that you have prepared in advance in the `samplesheet` field under the `Run configuration` heading. The FASTQ paths can be declared using absolute or relative filenames (relative to the StaG-mwc repository directory). Input files can be located anywhere, i.e. their locations are not restricted to the repository folder and they can even be located in remote storage systems like S3.

Note: When the path to a samplesheet TSV file has been specified in the config file StaG-mwc will ignore the `inputdir` and `input_fn_pattern` settings.

When using remote input files on S3 the access and secret keys must be available in environment variables `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY`.

Using remote files is also possible from http:// and https:// sources.

It is possible to keep a local copy of remote input files in the repository folder after the run by setting `keep_local: True` in the config file.

The samplesheet can be specified on the command line by utilizing Snakemake's built-in functionality for modifying configuration settings via the command line directive `--config samplesheet=samplesheet.tsv`.

3.2 Configuring which tools to run

Next, configure the settings under the `Pipeline steps included` heading. This is where you define what steps should be included in your workflow. Simply assign `True` or `False` to the steps you want to include. Note that the default configuration file already includes `qc_reads` and `host_removal`. These two steps are the primary read processing steps and most other steps depends on host filtered reads (i.e. the output of the `host_removal` step). Note that these two steps will pretty much always run, regardless of their setting in the config file, because they produce output files that almost all other workflow steps depend on.

Note: You can create several copies of `config.yaml`, named whatever you want, in order to manage several analyses from the same StaG-mwc directory. If you create a copy called e.g. `microbime_analysis.yaml`, you can easily run the workflow with this configuration file by using the `--configfile` commandline argument when running the workflow.

A reference database is required in order to run the `host_removal` step. If you already have it downloaded somewhere, point StaG-mwc to the location using the `db_path` parameter under the `remove_host` section of `config.yaml`.

The config file contains a parameter called `email`. This can be used to have the workflow send an email after a successful or failed run. Note that this requires that the Linux system your workflow is running on has a working email configuration. It is also quite common that most email clients will mark email sent from unknown random computers as spam, so don't forget to check your spam folder.

3.3 Running

It is recommended to run Snakemake with the `-n`/`--dryrun` argument before starting an analysis for real. Executing a dryrun will let Snakemake check that all the requirements are available and it will then print a summary of what it intends to do, without actually doing anything. After finishing the configuration by editing `config.yaml`, test your configuration with:

```
snakemake --dryrun
```

If you are satisfied with the workflow plan output by the dryrun, you can run the workflow. The typical command to run StaG-mwc on your local computer is:

```
snakemake --use-conda --cores N
```

where `N` is the maximum number of cores you want to allow for the workflow. Snakemake will automatically reduce the number of cores available to individual steps to this limit. Another variant of `--cores` is called `--jobs`, which you might encounter occassionally. The two commands are equivalent.

Note: If several people are running StaG-mwc on a shared server or on a shared file system, it can be useful to use the `--singularity-prefix`/`--conda-prefix` parameter to use a common folder to store the conda environments created by StaG-mwc, so they can be re-used between different people or analyses. This reduces the risk of producing several copies of the same conda environment in different folders. This is can also be necessary when running on cluster systems where paths are usually very deep. Then for example create a folder in your home directory and use that with the `--singularity-prefix`/`--conda-prefix` option.

If you want to keep your customized `config.yaml` in a separate file, let's say `my_config.yaml`, then you can run snakemake using that custom configuration file with the `--configfile my_config.yaml` command line argument.

Another useful command line argument to snakemake is `--keep-going`. This will instruct snakemake to keep going even if a job should fail, e.g. maybe the taxonomic profiling step will fail for a sample if the sample contains no assignable reads after quality filtering.

If you are having trouble running StaG-mwc with conda, try with Singularity (assuming you have Singularity installed on your system). There are pre-built Singularity images that are ready to use with StaG-mwc. Consider using `--singularity-prefix` to specify a folder where Snakemake can download and re-use the downloaded Singularity images for future invocations. The command to run StaG-mwc with Singularity instead of conda is:

```
snakemake --use-singularity --singularity-prefix /path/to/prefix/folder --dryrun
```

There are some additional details that need to be considered when using Singularity instead of conda, most notably that you will have to specify bind paths (specifying-bind-paths) so that your reference databases are accessible from within the containers when running StaG-mwc. It might look something like this:

```
snakemake --use-singularity --singularity-prefix /path/to/prefix/folder --singularity-
↪args "-B /home/username/databases"
```

The above example assumes you have entered paths to your databases in `config.yaml` with a base path like the one shown in the above command (e.g. `/home/username/databases/kraken2/kraken2_human/`).

3.4 Running on cluster resources

In order to run StaG-mwc on a cluster, you need a cluster profile. StaG-mwc ships with a pre-made cluster profile for use on CTMR's Gandalf Slurm cluster. The profile can be adapter to use on other Slurm systems if needed. The profile is distributed together with the StaG-mwc workflow code and is available in the `profiles` directory in the repository. The cluster profile specifies which cluster account to use (i.e. Slurm project account and partition), as well as the number of CPUs, time, and memory requirements for each individual step. Snakemake uses this information when submitting jobs to the cluster scheduler.

When running on a cluster it will likely work best if you run StaG using Singularity. The workflow comes preconfigured to automatically download and use containers from various sources for the different workflow steps. The CTMR Gandalf Slurm profile is preconfigured to use Singularity by default.

Note: Do not combine `--use-conda` with `--use-singularity`.

To prevent StaG-mwc from unnecessarily downloading the Singularity container images again between several projects you can use the `--singularity-prefix` to specify a directory where Snakemake can store the downloaded images for reuse between projects.

Paths to databases need to be located so that they are accessible from inside the Singularity containers. It's easiest if they are all available from the same folder, so you can bind the main database folder into the Singularity container with e.g. `--singularity-args "-B /path/to/db"`. Note that database paths need to specified in the config file so that the paths are correct from inside the Singularity container. Read more about specifying bind paths in the official Singularity docs: specifying-bind-paths.

To run StaG-mwc on CTMR's Gandalf cluster, run the following command from inside the workflow repository directory:

```
snakemake --profile profiles/ctmr_gandalf
```

This will make Snakemake submit each workflow step as a separate cluster job using the CPU and time requirements specified in the profile. The above command assumes you are using the default `config.yaml` configuration file. If you are using a custom configuration file, just add `--configfile <name_of_your_config_file>` to the command line.

Note: Have a look in the `profiles/ctmr_gandalf/config.yaml` to see how to modify the resource configurations used for the Slurm job submissions.

Some very lightweight rules will run on the submitting node (typically directly on the login node), but the number of concurrent local jobs is limited to 2 in the default profiles.

3.5 Execution report

Snakemake provides facilites to produce an HTML report of the execution of the workflow. A zipped HTML report is automatically created when the workflow finishes.

CHAPTER 4

Modules

StaG-mwc is a workflow framework that connects several other tools. The basic assumption is that all analyses start with a quality control of the sequencing reads (using FastP), followed by host sequence removal (using Kraken2 or Bowtie2). This section of the documentation aims to describe useful details about the separate tools that are used in StaG-mwc.

The following subsections describe the function of each module included in StaG-mwc. Each tool produces output in a separate subfolder inside the output folder specified in the configuration file. The output folders mentioned underneath each module heading refers to the subfolder inside the main output folder specified in the configuration file.

4.1 Pre-processing

4.1.1 qc_reads

Tools FastP

Output folder `fastp`

The quality control module uses FastP to produce HTML reports of the quality of the input and output reads. The quality control module also trims adapter sequences and performs quality trimming of the input reads. The quality assured reads are output into `fastp`. Output filenames are:

```
<sample>_{1,2}.fq.gz
```

Note: It is possible to skip fastp processing. StaG then replaces the output files with symlinks to the input files.

4.1.2 remove_host

The `remove_host` module can use either Kraken2 or Bowtie2 to classify reads against a database of host sequences to remove reads matching to non-desired host genomes.

Note: It is possible to skip host removal. StaG then replaces the output files with symlinks to the fastp output files.

Tool Kraken2

Output folder `host_removal`

The output from Kraken2 are two sets of pairs of paired-end FASTQ files, and optionally one Kraken2 classification file and one Kraken2 summary report. In addition, two PDF files with 1) a basic histogram plot of the proportion of host reads detected in each sample, and 2) a barplot of the same. A TSV table with the raw proportion data is also provided:

```
<sample>_{1,2}.fq.gz
<sample>.host_{1,2}.fq.gz
host_barplot.pdf
host_histogram.pdf
host_proportions.txt
```

Tool Bowtie2

Output folder `host_removal`

The output from Bowtie2 is a set of paired-end FASTQ files:

```
<sample>_{1,2}.fq.gz
```

4.1.3 preprocessing_summary

This module summarize the number of reads passing through each preprocessing step and produces a summary table showing the number of reads after each step. For more detailed information about read QC please refer to the MulitQC report.

4.1.4 multiqc

Tool MultiQC

Output folder `multiqc`

MultiQC summarizes information about several steps in StaG in an easy-to-use HTML report. Refer to this report for details about e.g. read QC.

4.2 Naive sample analysis

4.2.1 sketch_compare

Tools `sketch.sh`, `comparesketch.sh` from BBMap

Output folder `sketch_compare`

The `sketch_compare` module uses the very fast MinHash implementation in BBMap to compute MinHash sketches of all samples to do an all-vs-all comparison of all samples based on their kmer content. The module outputs gzip-compressed sketches for each sample, as well as two heatmap plots showing the overall similarity of all samples (one with hierarchical clustering).

4.2.2 assess_depth

Tool BBCountUnique

Output folder `bbcountunique`

The `assess_depth` module uses BBMap's very fast kmer counting algorithms to produce saturation curves. The saturation curve shows a histogram of the proportion of unique kmers observed per reads processed, and can be used to assess how deep a sample has been sequenced. The module outputs one plot per sample.

4.3 Taxonomic profiling

4.3.1 Kaiju

Tool Kaiju

Output folder `kaiju`

Run Kaiju on the trimmed and filtered reads to produce a taxonomic profile. Outputs several files per sample (one per taxonomic level specified in the config), plus two files that combine all samples in the run: an HTML Krona report with the profiles of all samples and a TSV table per taxonomic level. The output files are:

```
<sample>.kaiju
<sample>.kaiju.<level>.txt
<sample>.krona
all_samples.kaiju.krona.html
all_samples.kaiju.<level>.txt
```

4.3.2 Kraken2

Tool Kraken2

Output folder `kraken2`

Run Kraken2 on the trimmed and filtered reads to produce a taxonomic profile. Optionally Bracken can be run to produce abundance profiles for each sample at a user-specified taxonomic level. The Kraken2 module produces the following files:

```
<sample>.kraken
<sample>.kreport
all_samples.kraken2.txt
all_samples.mpa_style.txt
```

This modules outputs two tables containing the same information in two formats: one is the default Kraken2 output format, the other is a MetaPhlAn2-like format (`mpa_style`). The optional Bracken further adds additional output files for each sample:

```
<sample>.<taxonomic_level>.bracken
<sample>.<taxonomic_level>.filtered.bracken
<sample>_bracken.kreport
<sample>.bracken.mpa_style.txt
all_samples.<taxonomic_level>.bracken.txt
all_samples.<taxonomic_level>.filtered.bracken.txt
all_samples.bracken.mpa_style.txt
```

4.3.3 KrakenUniq

Tool KrakenUniq

Output folder `krakenuniq`

Run KrakenUniq on the trimmed and filtered reads to produce a taxonomic profile. The KrakenUniq module produces the following output files:

```
<sample>.kraken.gz
<sample>.kreport
all_samples.krakenuniq.txt
```

4.3.4 MetaPhlAn

Tool MetaPhlAn

Output folder `metaphlan`

Please refer to the official MetaPhlAn installation instructions on how to install the bowtie2 database in a separate directory outside the conda environment.

Run MetaPhlAn on the trimmed and filtered reads to produce a taxonomic profile. Outputs four files per sample, plus four summaries for all samples:

```
<sample>.bowtie2.bz2
<sample>.sam.bz2
<sample>.metaphlan.krona
<sample>.metaphlan.txt

all_samples.metaphlan.krona.html
all_samples.Species_top50.pdf
all_samples.metaphlan.txt
area_plot.metaphlan.pdf
```

The file called `all_samples.Species_top50.pdf` contains a clustered heatmap plot showing abundances of the top 50 species across all samples. The taxonomic level and the top `N` can be adjusted in the config. The file called `area_plot.metaphlan.pdf` is an areaplot summarizing the samples.

4.3.5 StrainPhlAn

Tool *StrainPhlAn*

Output folder `strainphlan`

Run *StrainPhlAn* on the `<sample>.sam.bz2` output from MetaPhlAn. Generates these output files of primary interest:

```
available_clades.txt
RAxML_bestTree.{clade_of_interest}.StrainPhlAn3.tre
{clade_of_interest}.StrainPhlAn3_concatenated.aln
```

`{clade_of_interest}` is specified in `config.yaml`. The outputs are a tree and an alignment file. Note that StrainPhlAn uses output generated from MetaPhlan and will thus also need to run MetaPhlAn-associated steps (even if it is not set to `True` in `config.yaml`).

If the pipeline fails the most common issue will be that `{clade_of_interest}` cannot be detected in sufficient quantities in a sufficient number of samples. Review the file `available_clades.txt` to reveal which clades can be investigated in your set of samples.

4.4 Functional profiling

4.4.1 HUMAnN

Tool HUMAnN

Output folder `humann`

Run HUMAnN on the trimmed and filtered reads to produce a functional profile. Outputs five files per sample, plus three summaries for all samples:

```
<sample>.genefamilies_{unit}.txt
<sample>.genefamilies.txt
<sample>.pathabundance_{unit}.txt
<sample>.pathabundance.txt
<sample>.pathcoverage.txt

all_samples.humann_genefamilies.txt
all_samples.humann_pathcoverage.txt
all_samples.humann_pathabundances.txt
```

`{unit}` refers to the normalization method specified in `config.yaml`, the default unit is counts per million (cpm).

Note: HUMAnN uses the taxonomic profiles produced by MetaPhlAn as input, so all MetaPhlAn-associated steps are run regardless of whether it is actually enabled in `config.yaml` or not.

HUMAnN requires large amounts of temporary disk space when processing a sample and will automatically use a suitable temporary directory from system environment variable `$TMPDIR`, using Snakemake's resources feature to evaluate the variable at runtime (which means it can utilize node-local temporary disk if executing on a compute cluster).

4.5 Mappers

StaG-mwc allows the use of regular read mapping tools to map the quality controlled reads to any reference database. All mappers can be used to map reads against several different databases (see Mapping to multiple databases below). In addition, all mappers can optionally summarize read counts per annotated feature via one of two options: 1) supplying a two-column tab-separated annotation file with one annotation per reference sequence, or 2) supplying a GTF or SAF format annotation file for features on the reference sequences. Option number 1 uses a custom Python script (`scripts/make_count_table.py`) to merge read counts per annotation, which works well for annotations as large as your memory allows, and option number 2 uses featureCounts to summarize read counts per annotated feature. Option number 2 is more flexible and fairly fast for typical annotation scenarios, but might not work when the number of unique features is much lower than the number of reference sequences. Read more about these alternatives in Summarizing read counts below.

Note: The mapper modules are great for mapping reads to databases with e.g. antibiotic resistance genes (like megares) or other functionally annotated genes of interest.

4.5.1 BBMap

Tool BBMap

Output folder `bbmap/<database_name>`

This module maps read using BBMap. The output is in sorted and indexed BAM format (with an option to keep the intermediary SAM file used to create the BAM). It is possible to configure the mapping settings almost entirely according to preference, with the exception of changing the output format. Use the configuration parameter `bbmap:extra` to add any standard BBMap commandline parameter you want.

4.5.2 Bowtie2

Tool Bowtie2

Output folder `bowtie2/<database_name>`

This module maps read using Bowtie2. The output is in BAM format. It is possible to configure the mapping settings almost entirely according to preference, with the exception of changing the output format from BAM. Use the configuration parameter `bowtie2:extra` to add any standard Bowtie2 commandline parameter you want.

4.5.3 Mapping to multiple databases

Note that the configuration settings of all mapper modules are slightly different from the configuration settings from most other modules. They are defined as lists in `config.yaml`, e.g. (note the leading – that signifies a list):

```
bbmap:
    - db_name: ""
      db_path: ""
      min_id: 0.76
      keep_sam: False
      keep_bam: True
      extra: ""
      counts_table:
          annotations: ""
      featureCounts:
          annotations: ""
          feature_type: ""
          attribute_type: ""
          extra: ""
```

This makes it possible to map the reads against several databases, each with their own mapping options and/or custom annotations. To map against more than one database, just create another list item underneath, containing all the same configuration options, but with different settings. For example, to map against `db1` and `db2` with different annotation files for each:

```
bbmap:
    - db_name: "db1"
      db_path: "/path/to/db1"
      min_id: 0.76
      keep_sam: False
      keep_bam: True
      extra: ""
      counts_table:
          annotations: ""
          columns: ""
```

(continues on next page)

(continued from previous page)

```
      featureCounts:
          annotations: ""
          feature_type: ""
          attribute_type: ""
          extra: ""
    - db_name: "db2"
      db_path: "/path/to/db2"
      min_id: 0.76
      keep_sam: False
      keep_bam: True
      extra: ""
      counts_table:
          annotations: "/path/to/db2/annotations.txt"
          columns: "Genus,Phylum"
      featureCounts:
          annotations: ""
          feature_type: ""
          attribute_type: ""
          extra: ""
```

4.5.4 Summarizing read counts

make_count_table.py

Tool `make_count_table.py`

Output folder `<mapper>/<database_name>`

A custom Python script that produces tab-separated count tables with one row per annotation, and one column per sample. The input is an annotation file that consists of at least two tab-separated columns. The first line is a header line with column names (must not contain spaces and avoid strange characters). Here is an example of column names:

```
Reference
Annotation1
Annotation2
...
AnnotationN
```

The column names doesn't matter, but the names defined in the annotation file can be used to select a subset of columns to summarize read counts for (see more below). The first column should contain the FASTA header for each reference sequence in the reference database used in the mapping. The count table script truncates the header string at the first space (because Bowtie2 does this automatically it's easier to just always do it). In practice, since the script performs truncation of headers, it doesn't matter which mapper was used or if the annotation file contains entire headers or only the truncated headers, as long as the bit up until the first space in each reference header is unique. The script sums counts for each annotation for each sample.

One parameter for the count summarization is which columns in the annotation file to summarize on. The column names need to be assigned as a string of comma-separated column names. They must match exactly to the column names defined in the annotation file. This is configured in `config.yaml`. The script outputs one file per column, with output filename matching `counts.<column_name>.txt`. The count table feature is activated by entering an annotation filename in the relevant section of the configuration file, e.g.:

```
bbmap:
    counts_table:
```

(continues on next page)

(continued from previous page)

```
        annotations: "path/to/annotations.tab"
        columns: "Species,Genus,taxid"
```

featureCounts

Tool featureCounts

Output folder `<mapper>/<database_name>`

This uses featureCounts to summarize read counts per annotation and sample. The input is a file in GTF format (or SAF format, read more below). featureCounts can summarize read counts on any feature (or meta-feature) that is defined in your GTF file. Use the featureCounts `attribute_type` to summarize read counts for any attribute defined in your GTF file. To use featureCounts to summarize read counts, enter an annotation filename in the configuration file, e.g.:

```
bowtie2:
    featureCounts:
        annotations: "path/to/annotations.gtf"
```

The featureCounts module outputs several files:

```
all_samples.featureCounts
all_samples.featureCounts.summary
all_samples.featureCounts.table.txt
```

The first two files are the default output files from featureCounts, and the third file is a simplified tab-separated table with count per annotation, in a format similar to the one described for `make_count_table.py` above.

It is also possible to use the simplified annotation format instead of GTF. To tell featureCounts you are using a SAF file, add `-F SAF` to the featureCounts `extra` configuration setting, e.g.:

```
bowtie2:
    featureCounts:
        extra: "-F SAF"
```

4.6 Assembly

StaG does not offer an assembly workflow at this time.

CHAPTER 5

Frequently asked questions (FAQ)

StaG-mwc is designed to be fairly flexible. This section answers common questions on how to best utilize StaG-mwc's flexibility. Please help expand this section by making a Pull Request with suggestions.

5.1 Skip read QC

In some cases your data has already been subjected to adapter and quality trimming so you want to skip that step. In StaG-mwc it is possible to bypass both the read QC step and the host removal steps if you need to. In order to do so, we must "trick" Snakemake into thinking that those rules have already been performed. The rules for read QC and host removal are configured with bypasses so that if the user sets `host_removal: False` or `qc_reads: False`, those steps will be bypassed by creating symlinks directly to the input files in the respective output directories.

5.2 Skip host removal

It is possible to skip host removal. This might be appropriate for example when processing environmental samples that do not need host removal. It is implemented in StaG-mwc in a special way: if the user sets `host_removal: False` in the configuration file then StaG-mwc will put symlinks to the `qc_reads` output files in the `host_removal` output directory. This "tricks" Snakemake into thinking that host removal actually occured, enabling it to complete the dependency graph to process the data in downstream steps.

5.3 Pipeline stopped unexpectedly

If pipeline ends with error or if the session is locked after being unexpectedly disconnected and the pipeline needs to be restarted, you can try to remove slurm metadadata files before restarting pipeline using:

```
(base)$ rm -rfv .snakemake/metadata
```

CHAPTER 6

About

Authors Fredrik Boulund

Contact fredrik.boulund@ki.se

This is the documentation for StaG-mwc, version 0.7.1-dev, last updated Jun 14, 2023. The documentation is available online at https://stag-mwc.readthedocs.io.

StaG-mwc is published as open source under the MIT license and you are encouraged to download, use, and help improve the project by contributing code modifications as pull requests back to the project's Github repository.

CHAPTER 7

Contributing

If you want to contribute to the development or submit bug fixes, have a look at the contributing guidelines in the `CONTRIBUTING.md` file in the Github repository.

CHAPTER 8

Citing StaG-mwc

If you find StaG-mwc useful and publish something using StaG-mwc, please cite:

Fredrik Boulund, et al. (2018).
StaG-mwc: StaG metagenomics workflow collaboration.
https://github.com/ctmrbio/stag-mwc
https://doi.org/10.5281/zenodo.1483891

Also consider citing all of the tools that were run as components in specific the StaG-mwc workflow used in your analysis.

CHAPTER 9

Indices and tables

- genindex
- modindex
- search