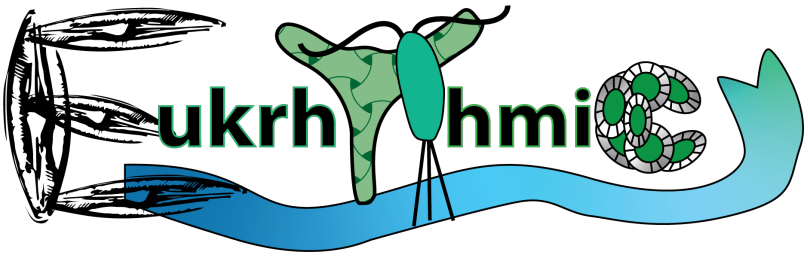
eukrhythmic

Feb 28, 2023

Contents:


eukrhythmic

CHAPTER 1

Installing eukrhythmic

eukrhythmic is designed as a modular pipeline that you can fully customize as the user of the resource. As such, we recommend setting up the base environment provided in the environment.yaml file prior to running anything.

1.1 Downloading eukrhythmic

You may download the pipeline by cloning it directly from GitHub.:

```
git clone https://github.com/AlexanderLabWHOI/eukrhythmic
cd eukrhythmic
```

All eukrhythmic commands are run from the base directory.

If you want to receive further information or plan on executing from the command line, go ahead and run:

```
alias eukrhythmic='./bin/eukrhythmic.sh'
```

Then, you can execute *eukrhythmic -h* to see that the software is present in the workspace.

Refer to "Running the pipeline from the command line" for more information on using command-line arguments with *eukrhythmic*.

We note that most of our documentation on the use of a scheduler to run eukrhythmic in parallel on multiple machines is written for the SLURM scheduling system. We provide some documentation on the use of the PBS system in the "Using eukrhythmic" section, but invite you to submit an issue on our GitHub page if you would like guidance on how to use PBS or an alternative system to run the pipeline.

1.2 Setting up a conda environment for running Snakemake

Initialize the pipeline by setting up a conda environment, such that all the requested packages are loaded.:

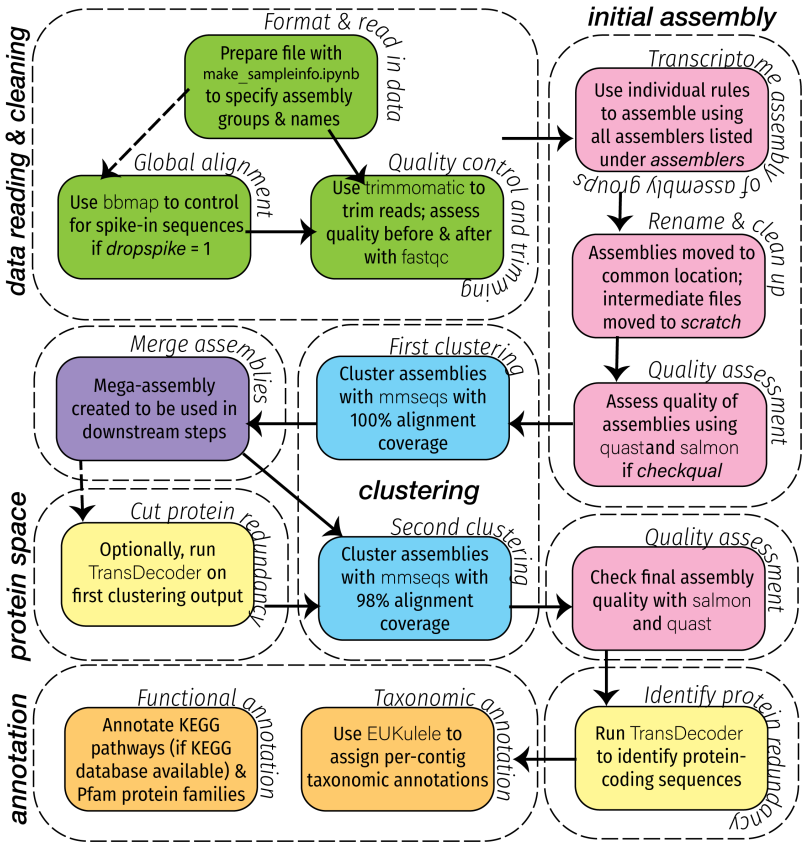
```
conda env create eukrhythmic --file environment.yaml
```

If this doesn't work for you, please try to set a `conda` environment manually that contains `Snakemake`, preferably newer than version 6, `Python`, preferably newer than version 3.8, `mamba`, some version of `pandas` and some version of `pyyaml`.

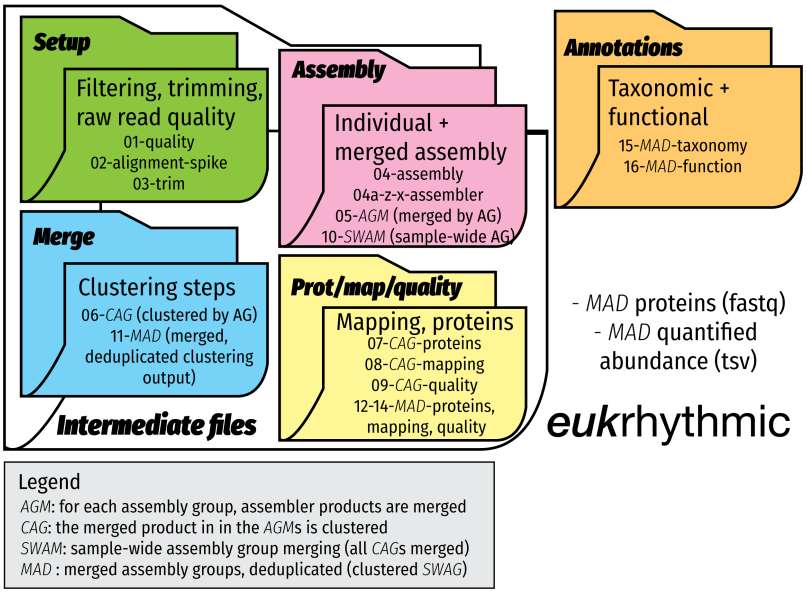
CHAPTER 2

Execution flowchart

You can reference this flowchart for an idea of the steps involved in the `eukrhythmic` pipeline:



And this diagram for the outputs returned from the pipeline:



Legend
AGM: for each assembly group, assembler products are merged
CAG: the merged product in in the *AGM*s is clustered
SWAM: sample-wide assembly group merging (all *CAG*s merged)
MAD : merged assembly groups, deduplicated (clustered *SWAG*)

CHAPTER 3

Running eukrhythmic

Eukrhythmic is built using `Snakemake` (documentation).

Users who prefer to use command-line arguments and/or are unfamiliar with `Snakemake` should read the below section "How to use the pipeline from the command line".

Users familiar with `Snakemake` that would like to execute the pipeline as a `Snakemake` workflow can skip to "How to use the pipeline directly through Snakemake".

If you have trouble getting `eukrhythmic` configured, we suggest navigating to the "Running eukrhythmic with the sample data" tab and running the software on our provided subsampled data.

3.1 How to use the pipeline from the command line

To execute `eukrhythmic` from the command line, alias the executable while in the `eukrhythmic` base directory after cloning from `GitHub`, like so:

```
alias eukrhythmic='./bin/eukrhythmic.sh'
```

And then invoke:

```
eukrhythmic <arguments>
```

```
(base) akrinos@pn033:eukrhythmic$ ./bin/eukrhythmic.sh
               __            __            _ __                _
              [  |  _       [  |          / |[  |              (_)
 .---. __   _  | | / ] _ .--.| |--.   _   _`| |-| |--.  _ .--..--.  __   .---.
/ /__\[  | | | | '' < [ `/'`\| .-. | [ \ [  | | | .-. |[ `.-. .-. |[  |/ /'`\]
| \__.,| \_/ |,| |`\ \ | |   | | | |  \ '/ /| |,| | | | | | | | | | || \__.
 '.__.''.__.'_[__|  \_[___] [___]|__[\_:  / \__[___]|__[___||__||__[___'.___.'
                                      \__.'
******************************************************************************
******************************************************************************

Welcome to eukrhythmic! Visit the readthedocs for more information
on how to use the command line functionality. Note that not all options
will be implemented for the command line.

******************************************************************************
******************************************************************************

No arguments were given to eukrhythmic. Please provide arguments or run
eukrhythmic with the -h flag for more information.
```

The arguments that you can use from the command line include:

Table 1: Title

Flag	`config.yaml` entry	Meaning
`-n` / `--job-name`	jobname	A descriptive/unique name to be applied to SLURM jobs.
`-s` / `--sample-file-name`	metaT_sample	The listing of SampleIDs, SampleNames, FastqFiles, and assembly groups for each sample.
`-o` / `--out-dir`	outputDIR	The output directory to write to.
`-i` / `--in-dir`	inputDIR	The location of the sample FASTQ files (crucial unless running sample data).
`-q` / `--check-quality`	checkqual	Takes no arguments. If specified, quality assessment is run on assemblies.
`-b` / `--run-bbmap`	runbbmap; spikefile	If your data contains spiked reads, you can list this flag and specify a FASTA file containing the spike.
`-l` / `--slurm`	N/A	If provided, run `snakemake` on SLURM.
`-g` / `--generate-file`	N/A	If provided, create the sample file automatically from the files present in the input directory.
`--use-sample`	N/A	If provided, override all other parameters and run the sample file.
`-c` / `--scratch-dir`	scratch	Takes argument of the directory to be used as scratch space.

If you need to use more options than this, you'll need to create a configuration file. You can edit the provided `config.yaml` file in order to do this. In the "Advanced: Writing a configuration file" tab of the documentation, you can find a list of the supported entries that you can add to your `config.yaml` file.

To get started using the pipeline, the most important thing to do is to specify the paths to your particular input and output directories. Strictly speaking, there are four things you'll probably want to specify before running `eukrhythmic` on your own data:

- jobname - some random name that will determine how your SLURM sub-jobs are being called
- metaT_sample - the location of the file containing all relevant information needed to run the pipeline (e.g.,

created using Google Sheets in TSV form) ideally located in input/ (see below for more information on the metaT_sample file)

- inputDIR - where your raw fastq sequence files are
- outputDIR - where you want your output to go
- scratch - where you want non-essential intermediates to go

Which could be done on the command line using:

```
eukrhythmic -o <output_dir> -i <input_dir> -s <metaT_sample_file> -c <scratch_
↪directory>
```

All input `fastq` files must be contained in the same directory, the `inputDIR` location, specified by the `--in-dir` flag. Only these metatranscriptomic data will be included in the analysis. These files do *not*, however, need to be located inside the `eukrhythmic` directory (and it is recommended that they are located elsewhere).

The next thing that needs to be done is to produce the sample file, containing all of the relevant information to run the pipeline. You can create this yourself (:ref:`manual`), using only a list of Sample IDs (:ref:`fastqauto`), or completely automatically (:ref:`fullauto`), which can be done all in one with the `eukrhythmic` bash script, by specifying:

```
eukrhythmic --in-dir <name-of-your-input> --out-dir <name-of-your-output> -g
```

using the `-g` or `--generate-file` option, which runs the included script (:ref:`fullauto`). For further information on any of these options, please read the "Naming your samples and helping eukrhythmic find them" section of the documentation.

Using a scheduler

If you are running on an HPC system that uses the `SLURM` scheduler, invoke `eukrhythmic` with one argument: `-l` or `--slurm` to leverage your computing resources. In that case, you will also want to invoke `sbatch eukrhythmic --slurm`, to avoid running any steps on your current remote machine. You can also use the `-np` or `--dry_run` flag to run do a `Snakemake` dry run, to see whether the jobs to be run align with your expectation (you can do this whether or not you are using a scheduler). If you need to use a different scheduler than `SLURM`, please run the pipeline through Snakemake (instructions below).

If you use `PBS`, a flag already exists. Otherwise, please submit an issue on our GitHub repository so that we can work together to find a solution! You will want to edit the `cluster.yaml` file to reflect the memory and time requirements of your system. Just populating the `_default_` and `required` sections will do.

3.2 How to use the pipeline directly through Snakemake

To use the pipeline as a `Snakemake` workflow, the most important thing to do is to populate `config.yaml` with the paths to your particular input and output directories, and to generate the sample file (either manually (:ref:`manual`) or using a semi (:ref:`fastqauto`) or completely automatic procedure (:ref:`fullauto`). Personalizing this will allow the pipeline to pull the relevant files when computing the results of individual rules, so this step is crucial. You should also edit any other important parts of the configuration file, as described in the separate section of the documentation.

You'll find further information about the configuration file in "Advanced: Writing a configuration file", and more information about the sample file in "Naming your samples and helping eukrhythmic find them".

Once the pieces are in place, and you have either activated an environment using `environment.yaml` or otherwise installed `snakemake`, you can run the pipeline using:

```
sbatch submit/snake_submit.sh <snakefile> <number of jobs> <optionally, --rerun-
↪incomplete>
```

Or, wrapping this command with the ability to specify `eukrhythmic` subroutines rather than selecting a configuration file, you can invoke `eukrhythmic` with:

```
python submit/eukrhythmic <subroutine>
```

Where "<subroutine>" is the subset of eukrhythmic functionality that you wish to use for this run. In most cases, you'll write "all" here, to indicate that you wish to run all of the steps of the pipeline sequentially.

If you are using the `SLURM` scheduler, you can run the pipeline by executing the `submit/snake_submit.sh` file in the `eukrhythmic` directory, after configuring options for your particular system (:ref:`slurm`), or by setting the `rewritecluster` configuration flag to 1, and specifying the options for all jobs in the `required` section of the `cluster.yaml` file. If you are not using a scheduler, or are logged into a computer with sufficient computational resources (e.g., a `SLURM` job run in interactive mode), you can execute `Snakemake` directly.

You can also do this using the `submit/eukrhythmic` script <Arianna needs to explain this script and the subroutines. She also needs to add ability to specify subroutines in the `bin/eukrhythmic` file.>

Running the pipeline with ''SLURM''

In order to run the pipeline with `SLURM` or any other similar scheduling platform on an HPC system, the file `cluster.yaml` in the base directory needs to be populated. Specifications for individual rules can be added or removed as needed; the default configuration is what must absolutely be specified for the pipeline to run properly. Make sure that you include the following:

- Your account name
- Any flags that you typically use when running commands on the system, in the `__default__`->''slurm''->''command'' string
- The partition of your system that you plan to use, as `queue`. By default, this might be `compute` or `normal`.

If you set your account name at the top of the `cluster.yaml` file, as well as setting the default partition just once, and you do not change the parameter `rewritecluster` to 0 in `config.yaml`, you can use the command line interface or the provided submission file to circumvent filling out the rest of `cluster.yaml`. You can also do this by invoking `python scripts/importworkspace.py` once before running the pipeline, if you already have a valid `config.yaml`. If you do this, you won't need to change these values for the specifications for all of the individual rules, unless you have specific computational needs or usage requirements, in which case you should set `rewritecluster` to 0. If defaults are not specified at the beginning of the `cluster.yaml` file for the user, maximum memory usage, maximum number of cores, and maximum number of threads, `eukrhythmic` will not execute successfully and an error will be thrown.

Running the pipeline with ''PBS''

There are four flags you can use with the `python submit/eukrhythmic` command for the use of an alternative scheduling system. These presently include the `pbs` and `slurm` systems; `slurm` is accessible with `--system slurm` or `--system sbatch`, and `pbs` is accessible with `--system pbs` or `--system qsub`. PBS is presently in beta mode for testing purposes, but should function more or less identically to the use of the `SLURM` system.

Setting CPUs and memory requirements

As a general rule for memory-intensive assemblers, the memory available to the process should be about ten times the number of cores/CPUs that you have available to you on the machine. For example, if using a machine with 30 cores available and 300 GB of available memory, you may want to configure your jobs to use 15 cores and 150 GB of memory, to allow two jobs to run concurrently on one node, and optimize memory relative to number of cores.

CHAPTER 4

Running eukrhythmic with the sample data

Note that running eukrhythmic on the sample data will take a little while. When we tested this functionality on our system, we used the following amounts of time and memory:

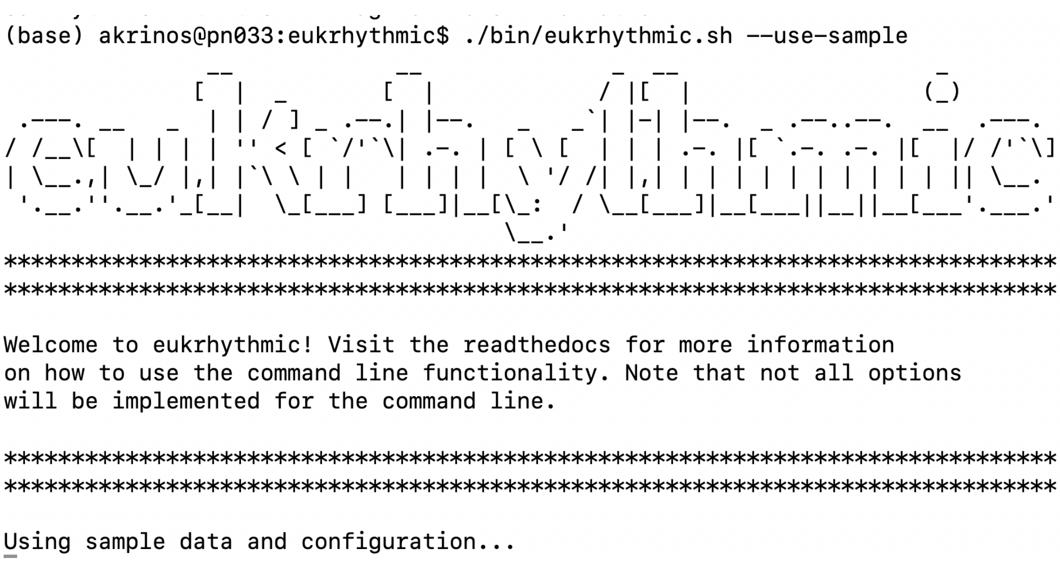
The sample data being run can be found in `input/testdata/`, and contains five small sample raw read files. The sample metaT file for this dataset is in `input/sampledata.txt`:

SampleName	SampleID	AssemblyGroup	FastqFile
HN008_subsampled	HN008	samplegroup1	HN008/HN008_subsampled
HN009_subsampled	HN009	samplegroup2	HN009_subsampled
HN016_subsampled	HN016	samplegroup1	HN016_subsampled
HN036_subsampled	HN036	samplegroup2	HN036_subsampled
HN043_subsampled	HN043	samplegroup2	HN043_subsampled

In the `SampleName` column, we put our longer descriptive name that matches our filename, whereas `SampleID` contains the smallest unique token we can make out of our sample names. In `AssemblyGroups`, we list the files we want the assembly software to have to assemble simultaneously. Even though all of our samples will eventually get combined, some of them will be treated independently to begin with by the assembly tools, and others won't. In this example, HN008 and HN016 are likely from the same site, for example.

In `FastqFile`, we use the full path of the file relative to the sample directory (which in our case is `input/testdata`). `HN008` has a subdirectory, so this syntax lets us leave our file organization as it is as we're running eukrhythmic.

To run `eukrhythmic` on the provided sample data, invoke `eukrhythmic` on a clean install of the program without arguments. You can also run the sample data by using the argument `--use-sample`, which will copy the relevant configuration entries.



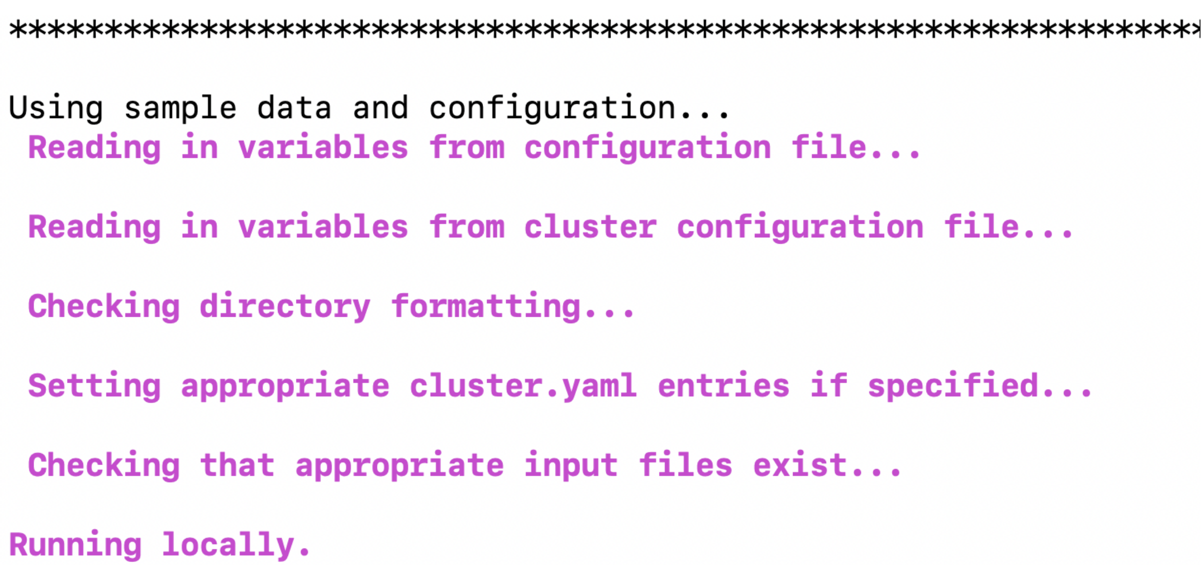
```
(base) akrinos@pn033:eukrhythmic$ ./bin/eukrhythmic.sh --use-sample
              __            __              _  __                  _
             [  |  _       [  |            / |[  |                (_)
 .---. __   _  | | / ] _ .--.| |--.   _   _`| |-| |--.  _ .--..--.  __  .---.
/ /__\[  | | | | | '' < [ `/'`\| .-. | [ \ [  | | | .-. |[ `.-. .-. |[  |/ /'`\]
| \__.,| \_/ |,| |`\ \ | |    | | | |  \ '/ /| |,| | | | | | | | | | | || \__.
 '.__.''.__.'_[__|  \_[___] [___]|__[\_:  / \__[___]|__[___||__||__[___'.___.'
                                      \__.'
********************************************************************************
********************************************************************************

Welcome to eukrhythmic! Visit the readthedocs for more information
on how to use the command line functionality. Note that not all options
will be implemented for the command line.

********************************************************************************
********************************************************************************

Using sample data and configuration...
```

Without any additional flags, `eukrhythmic` will be run against the provided sample data on your local machine (or your current node on the cluster, if you're logged into one). You should see a dialogue like this one:



```
*******************************************************************

Using sample data and configuration...
 Reading in variables from configuration file...

 Reading in variables from cluster configuration file...

 Checking directory formatting...

 Setting appropriate cluster.yaml entries if specified...

 Checking that appropriate input files exist...

Running locally.
```

CHAPTER 5

Naming your samples and helping eukrhythmic find them

5.1 File naming

Your file names should not subset one another. So if one file is called "supercool_sample", another should not be called "supercool_sample_2". However, if the two were called "supercool_sample_1" and "supercool_sample_2", this would be fine, because neither name is *entirely* found within the other.

5.2 The `metaT_sample` file

In the *config.yaml* file, there is a listing for a file called *metaT_sample* in the configuration file. This is essentially the data input source as far as what sample names you are expecting to include in your analysis, as well as any other information about the samples that you would like to be used. This is essential if you would like to apply groupings and co-assemble several samples together, and in general it is essential for the pipeline to work as intended.

Depending on the application, some columns of this file must be added or may not be necessary. For example, for repeated samples in the same location, the latitude and longitude may not be necessary, because geographic variation in the metatranscriptomic assembly will not be evaluated. Any data that are not included in the default steps in the provided script can be excluded.

As detailed in *scripts/make_sampleinfo.ipynb*, the minimum required to run the general (default) pipeline without any comparative analysis between samples are the "SampleName", "SampleID" and the "FastqFileNames". SampleName and SampleID can be identical if no such distinction exists in your sample. However, strictly, "SampleName" is a descriptive name for your samples, whereas "SampleID" is how the samples will be named throughout the pipeline, thus is is preferable to minimize special characters and length in the "SampleID" column, whereas "SampleName" may be more verbose.

5.3 Autogeneration of full `metaT_sample` file

Using the script `scripts/autogenerate_metaT_sample.py`, you can autogenerate a working sample file for whatever files are present in the directory specified in your configuration file as `inputDIR`. This file should be

run from the base *eukrhythmic* directory. At minimum, it requires a name for the output `metaT_sample` file as a parameter, which will be saved in the *input* directory, and then should be specified in the configuration file as the *metaT_sample* file. (*Note: in the future, we aim to autopopulate the config file with this entry and allow users to run the pipeline without even running this separately, with a default name for the ''metaT_sample'' file, as long as the ''inputDIR'' is specified*).

So within the base `eukrhythmic` directory, the following command may be run:

```
python scripts/autogenerate_metaT_sample.py testsampledata.txt
```

Optionally, additional parameters may be provided. The second optional parameter is a file extension, which defaults to "fastq.gz". The third and fourth optional parameters are labels for forward and reverse reads (defaults to "_1" and "_2", respectively), and the fifth optional parameter is an additional file suffix used to split the filename (e.g. 001; defaults to the file extension; specifically important for single-end reads).

5.4 Autogeneration of "FastqFileNames" column with "SampleID" column

In /scripts/, there is a Python script called `make_sample_file.py` that will generate the `fastq` file names column, given your data input folder and an existing `metaT_sample` file that contains sample IDs. This is a good option if you only wish to run a subset of the files in your input data folder, and want to make sure the `fastq` file names are properly formatted.

If you use this script, all of the files with the fastq extension listed in your `INPUTDIR` that have a match to entries in your SampleID column will be included. Optionally, the script accepts up to two input arguments that specify how forward/reverse reads are labeled. By default, "_R", "_1", and "_2" are searched for.

5.5 Notes about manually creating `metaT_sample`

If you specify "FastqFileNames" manually, **ensure that the files are named uniquely and that the entire unique choice of name is specified in this column**. Filenames that match to more than two `fastq` files in your input directory will raise an exception.

In the event that you want more control over how your samples are named, use `scripts/make_sampleinfo.ipynb`. The AssemblyGroup column may be omitted if you do not mind if your samples are assigned assembly groups according to numbers 1-*n*, where *n* is your number of samples, but in this case `separategroups` must be set to 0 in your `config.yaml` file. (*In the future, this may be updated to be done automatically if the column is absent, as well*).

Once you have generated the `metaT_sample` file containing the information about your samples, and have populated `config.yaml` with the relevant directories, including, importantly, `outputDIR`, which will be the location of your results, you are ready to run the pipeline.

CHAPTER 6

Advanced: Writing a configuration file

To write a configuration file for `eukrhythmic`, you need to edit the `config.yaml` file included in the `eukrhythmic` base directory. This YAML-formatted file can be modified by changing the entries to the right of each of the colons in each line of the file.

6.1 Configuration file entries

Below is a listing of each supported entry in the configuration file (`config.yaml` in the base directory) and how to specify each flag when using the pipeline.

Table 1: Title

Flag in file	Meaning & how to specify
`metaT_sample`	The name of the sample file containing sample ids to be used as unique identifiers in the pipeline, descriptive sample names, and input FASTA file names.
`inputDIR`	The file directory where the input data is found. Currently, should be specified with "/" separators, but no trailing "/". Should begin with "/" only if you are going to the root of your file system (not a relative path).
`checkqual`	Boolean flag for whether to run quality checking with `salmon`, `QUAST`, `BUSCO`, etc. on assemblies. If 1, these quality checks are performed.
`spikefile`	A path to a FASTA file containing the sequence of any spiking that might affect reads. This will depend on experimental setup. If the file is not valid (e.g., if this flag is set to 0), nothing is done.
`runbbmap`	A boolean flag to specify whether to use a spike file to drop spiked reads, according to what was done in your experiment. If 1, the spikefile is used; otherwise, this filtering is either not performed or is not used downstream in the pipeline (depending on whether a spike file exists).
`kmers`	A list of *k*-mer sizes to use, where applicable, in assembly. These should all be integer values (default: 20, 50, 110). The median *k*-mer value in this list will be used when just 1 *k*-mer value is required.
`assemblers`	The assemblers to be used to assemble the metatranscriptomes (which will later be merged). All of the specified assemblers in this list should have matching Snakemake rules in the `modules` folder of the main pipeline directory (named identically), as well as "clean" rules (explained below).
`jobname`	A descriptive name to be used to name jobs on your high-performance computing system, such that you can track the progress of your workflow.
`adapter`	Path to a FASTA file containing the adapter used during sequencing. Defaults to a static adapter file in the `static` directory.
`separategroups`	A boolean flag. If 1, specified assembly groups in the `metaT_sample` file are used to co-assemble raw files. Otherwise, each raw file is assembled separately regardless of what is specified in the "AssemblyGroup" column of the input file.
`outputDIR`	The path to a directory where all program output will be stored.
`assembledDIR`	The directory to move assembled files to, relative to the output directory. Defaults to "assembled"; not necessary to specify.
`renamedDIR`	The directory to move "renamed" files to (which are files with the name of the assembler added to each FASTA header), relative to the output directory. Defaults to "assembled"; not necessary to specify.
`scratch`	The location to move unnecessary intermediate files to after computation.

CHAPTER 7

Advanced: Adding unsupported assemblers

7.1 Adding new assemblers

In order to add a new assembler to the pipeline that is not presently included, three things need to be included:

1. A rule for the assembler, including a command that can be run using software in the specified `conda` environment, called `<assembler>`.
2. A rule to "clean" the assembly output, named `<assembler>_clean`, including moving the completed assemblies to the shared assembly folder, specified as `assembledDIR`, which is a subdirectory of the `outputDIR`, also specified in the configuration file. Intermediate files should also be moved to a scratch directory or deleted, based on user preferences and space requirements. Any other files needed by other tools or desired by the user should be moved to a subdirectory of the output directory. If they are specified as output files, `snakemake` will generate them automatically. Otherwise, the user will need to manually create directories that do not already exist (specifying them as output files is more extensible).
3. A list entry for the assembler in the configuration file that matches the name of `<assembler>` in each `snakemake` rule.

CHAPTER 8

Intermediate files and cleanup

A "scratch" directory, specified in the configuration file, is used to store intermediate files after execution of rules such as assembly, which produce many files which are not needed downstream in the pipeline. To override this behavior, specify the output directory and the scratch directory to be the same location.

After the pipeline has been run, simply enter:

```
snakemake hardclean --cores 1
```

To safely remove the scratch directory, if you don't need the intermediate files generated in individual pipeline steps.

CHAPTER 9

Common errors you may encounter

For errors that you don't find on this page or in the Snakemake (documentation), we encourage you to submit a ticket through GitHub issues. Please read the past issues first to see if we've addressed it before. If an open issue already exists, but it hasn't been answered, please feel free to add additional details to the thread. We will get back to you as soon as we can!

If you receive this error:

```
Error: Directory cannot be locked. Please make sure that no other Snakemake process␣
↪is trying to create the same files in the following directory:

path/to/eukrhythmic/dir/eukrhythmic

If you are sure that no other instances of snakemake are running on this directory,␣
↪the remaining lock was likely caused by a kill signal or a power loss. It can be␣
↪removed with the --unlock argument.
```

This means that `eukrhythmic` was run at one point and was not able to exit gracefully. Just run `snakemake -s eukrhythmic --unlock` to remove the lock on your directory.

CHAPTER 10

Acknowledgments

The `eukrhythmic` pipeline was written by:

- Arianna Krinos
- Harriet Alexander
- Natalie Cohen

With invaluable guidance and assistance from coauthor:

- Mick Follows

And contributing reviewers:

- Maggi Mars Brisbin
- Sarah Hu

We thank our many contributors and testers for their help with development and use of the pipeline!

CHAPTER 11

Source Code

You can find the source for eukrhythmic on GitHub.

CHAPTER 12

Indices and tables

- genindex
- modindex
- search