RNFtools Documentation

Release 0.3.1.2

Karel Břinda

Jun 27, 2018

Contents

This is the manual of RNFtools which is an associate software package for **Read Naming Format** (RNF), a generic format for naming simulated Next-Generation Sequencing reads.

RNFtools can be used for simulation of NGS reads, evaluation of mappers, and conversion of RNF-compliant data.

Publication

11. Břinda, V. Boeva, G. Kucherov. **RNF: a general framework to evaluate NGS read mappers**. Bioinformatics 32(1), pp. 136-139, 2016. [doi:10.1093/bioinformatics/btv524]

CHAPTER 1

Links

Project webpage - RNF specification - GitHub repository - Bug reporting - Mailing list - Contact

CHAPTER 2

Table of contents

2.1 Introduction

RNFtools is an associate software package for RNF, a generic format for assigning read names of simulated Next-Generation Sequencing reads. The format aims to remove dependency of evaluation tools of read mappers on the used read simulators.

RNFtools consist of three principal parts:

1. *MIShmash* (MIS => SIM => simulation) - a tool for simulating NGS reads in RNF format using existing simulators.
2. *LAVEnder* (LAVE => EVAL => evaluation) - a tool for evaluation of NGS read mappers using simulated reads in RNF format.
3. *RNF format library* - a Python library for handling the RNF format.

Technically, the entire RNFtools package is based on SnakeMake, a Make-like Python-based software developed for creating bioinformatics pipelines. The resulting pipelines are fully reproducible and they can be distributed as **single SnakeMake files**.

There exists also a console variant of RNFtools with a basic functionality (see *Command-line tool*).

2.1.1 How to start with RNFtools

First, we recommend to start with *Tutorials* which demonstrates how to install RNFtools and how to use it. All examples are also located in a dedicated directory. When you get familiar with basics, the main source of information will be *Reference*.

If anything is not clear, please look into *FAQ* and if this question has not been answered yet, send an e-mail to the Mailing list.

2.2 Tutorials

The tutorials will help you to install RNFtools and will show you how to use it. All the presented examples are also located in a dedicated Github directory.

2.2.1 Installation

RNFtools is distributed as a Python-based package, which is distributed through BioConda (a bioinformatics channel for the Conda package manager) and PIP. Since BioConda does not require a root account and installs also all the RNFtools dependencies, it is the recommended way of installation.

Contents

Requirements

Requirements for basic installation of RNFtools are:

- Unix-like operating system (Linux, OS X, etc.).
- Python 3.3+.

When RNFtools is installed using BioConda, all the additional dependencies are installed automatically. If not, installation of the following programs is up to user.

- Art
- CuReSim
- DWGsim
- Mason
- WGsim
- SamTools

Installation using Bioconda (recommended)

The easiest and safest approach of RNFtools installation is to create a separate Bioconda environment.

```
conda create -n rnftools rnftools
```

Once the environment is installed, you can activate it by

```
source activate rnftools
```

and deactivate by

```
source deactivate
```

Alternatively, RNFtools can be installed directly in the default Conda environment. However, this approach might not work in certain situations due to possible collisions with the dependencies of your already installed packages.

```
conda install rnftools
```

Installation using PIP from PyPI

RNFtools can be installed using PIP from PyPI by

```
pip install rnftools
```

If this command does not work, check if pip is installed in your system (the command may have a slightly different name, e.g., `pip`, `pip-3`, `pip.4`, `pip-3.4`). If not, install PIP by the official instructions (or try `easy_install3 pip`).

Upgrade to the newest version can be done also using PIP.

```
pip install --upgrade rnftools
```

Installation using PIP from GIT

To install RNFtools directly from GIT repository, run

```
git clone git://github.com/karel-brinda/rnftools
pip install rnftools
```

or

```
pip install git+http://github.com/karel-brinda/rnftools
```

Installation using PIP without a root account

First, we need to create a directory where RNFtools will be installed.

```
mkdir ~/rnftools
```

Then we have to add its path into the variable `PYTHONUSERBASE`

```
export PYTHONUSERBASE=~/rnftools
```

Now we can finally install RNFtools. The parameter `--user` implies installation into the predefined directory.

```
pip install --user rnftools
```

As the last step, we need to add the following lines to `~/.bashrc`

```
export PYTHONUSERBASE=~/rnftools
export PATH=$PATH:~/rnftools/bin
```

2.2.2 Hello world!

In this chapter, we show how to create and use RNFtools on a Hello world example. Little knowledge of Python can be helpful, but it is not required.

RNFtools is based on Snakemake, a Python-based Make-like build system. To simulate reads or evaluate alignments, you create simple configuration Python scripts and RNFtools subsequently creates a set of rules to be run by Snakemake. The rules can be then executed in a single thread, in parallel (`--cores` Snakemake parameter), or on a cluster. For more details about Snakemake, please see its documentation.

This approach allows to create big and reproducible pipelines, which are easy to share (it suffices to publish a single configuration script).

Every RNFtools script consists of three parts:

```
# 1) importing the rnftools Python package
import rnftools

# 2) all your Python code will go here
print("Hello world!")

# 3) including Snakemake rules created by RNFtools and defining the main
#    Snakemake rule (declaring which files are requested)
include: rnftools.include()
rule: input: rnftools.input()
```

As it is mentioned in the second comment, all your code should be inserted into part 2. Now save this file with name `Snakefile` and run

```
snakemake
```

in the same directory. A "Hello world" message will be printed, together with several Snakemake informative messages.

2.2.3 Read simulation

In this chapter, we show on several basic example how to simulate reads using a component of RNFtools called MISmash.

Basic example

First, let us show how to simulate reads from a single genome, stored in a FASTA file, using a single simulator. A corresponding RNFtools configuration script can look as follows:

```
1  import rnftools
2
3  rnftools.mishmash.sample("simple_example",reads_in_tuple=1)
4
5  # put your reference genome here
6  fa="../../../example1.fa"
7
8  rnftools.mishmash.ArtIllumina(
9          fasta=fa,
10         number_of_read_tuples=10000,
11         read_length_1=100,
12         read_length_2=0,
13 )
14
15 include: rnftools.include()
16 rule: input: rnftools.input()
```

Lines 1, 15, 16 have been already described in the previous chapter. Function `rnf.mishmash.sample` (line 3) initializes a new sample of simulated "single-end reads". When a new instance of `rnftools.mishmash.ArtIllumina` is created (line 8), it is automatically registered to the last created sample. This class is used for simulating reads by the Art Illumina read simulator. The parameters signalize parameters of the simulation: `fasta` is the reference file, `number_of_read_tuples` sets number of simulated read tuples, and `read_length_1` and `read_length_2` indicate lengths of simulated reads.

Within the RNF framework, a single simulated unit is a read tuple, which consists of one or more reads. For more details, see the RNFtools paper. `read_length_2=0` implies "single-end" simulation (in our notation: a single read in every read tuple).

When we run `snakemake`, reads are simulated and we obtain the final `simple_example.fq` file with all the simulated reads.

RNFtools supports several different read simulators. Their use is similar, though their interfaces are slightly different. A full documentation of all the supported simulators with all their parameters is available on the page *MIShmash*.

Simulation of 'paired-end' reads

To simulate "paired-end reads" (i.e., read tuples of two reads), two minor changes must be made in the original `Snakefile`. First, `rnftools.mishmash.sample` must be called with `reads_in_tuple=2`, then the length of second reads of a tuple must be set to a non-zero value.

Then the final `Snakefile` can be:

```
1  import rnftools
2
3  rnftools.mishmash.sample("simple_example",reads_in_tuple=2)
4
5  fa="../../../example1.fa"
6
7  rnftools.mishmash.ArtIllumina(
8          fasta=fa,
9          number_of_read_tuples=10000,
10         read_length_1=100,
11         read_length_2=100,
12 )
13
14 include: rnftools.include()
15 rule: input: rnftools.input()
```

Different simulator

To change a simulator in our example, we just replace `rnftools.mishmash.ArtIllumina` by another simulator, e.g., `rnftools.mishmash.ArtIllumina`. Parameters as `fasta`, `read_length_1`, `read_length_2`, and `number_of_read_tuples` are the same for all the simulators, but with several limitations:

- CuReSim supports only "single-end reads".
- ART Illumina in "paired-end" mode can simulate only reads of equal lengths.

```
import rnftools

rnftools.mishmash.sample("simple_example",reads_in_tuple=1)

fa="../../../example1.fa"

rnftools.mishmash.DwgSim(
        fasta=fa,
        number_of_read_tuples=10000,
        read_length_1=100,
        read_length_2=0,
)

include: rnftools.include()
rule: input: rnftools.input()
```

More genomes

To simulate reads from multiples reference within a single sample (in order to simulate, e.g., a metagenome or a contamination), we create a new instance of class of a simulator for each reference.

In the example below, we are simulating 10.000 read tuples from two reference genomes.

```
import rnftools

rnftools.mishmash.sample("simple_example",reads_in_tuple=1)

fa1="../../../example1.fa"
fa2="../../../example2.fa"

rnftools.mishmash.ArtIllumina(
        fasta=fa1,
        number_of_read_tuples=10000,
        read_length_1=100,
        read_length_2=0,
)

rnftools.mishmash.ArtIllumina(
        fasta=fa2,
        number_of_read_tuples=10000,
        read_length_1=100,
        read_length_2=0,
)

include: rnftools.include()
rule: input: rnftools.input()
```

Once reads are simulated for each of the references, they are mixed and put into the resulting FASTQ file.

More samples

We can also simulated multiple samples using a single `Snakemake`.

```
import rnftools

rnftools.mishmash.sample("simple_end_simulation",reads_in_tuple=1)

fa="../../../example1.fa"

rnftools.mishmash.ArtIllumina(
        fasta=fa,
        number_of_read_tuples=10000,
        read_length_1=100,
        read_length_2=0,
)

rnftools.mishmash.sample("paired_end_simulation",reads_in_tuple=2)

rnftools.mishmash.ArtIllumina(
        fasta=fa,
        number_of_read_tuples=10000,
        read_length_1=100,
        read_length_2=100,
)

include: rnftools.include()
rule: input: rnftools.input()
```

Sequence extraction

It may be sometimes useful to extract certain sequences from the reference file before the simulation itself. For instance, reads from each sequence could be simulated with a different coverage. For this purpose, we can use the `sequences` parameter. Sequences for extraction can be specified either by their number in the original FASTA file (starting from 0), or using their name.

```
import rnftools

rnftools.mishmash.sample("reads_first_sequence",reads_in_tuple=1)

rnftools.mishmash.ArtIllumina(
        fasta="example.fa",
        sequences=[0],
        number_of_read_tuples=10000,
        read_length_1=100,
        read_length_2=0,
)

rnftools.mishmash.sample("reads_second_sequence",reads_in_tuple=1)

rnftools.mishmash.ArtIllumina(
        fasta="example.fa",
        sequences=['seq2'],
        number_of_read_tuples=10000,
        read_length_1=100,
        read_length_2=0,
```

(continues on next page)

(continued from previous page)

```
)


include: rnftools.include()
rule: input: rnftools.input()
```

Non-standard parameters

Not all command-line parameters of every simulator are directly supported by RNFtools. However, such parameters can still be passed through the parameter `other_params` like in this example:

```
import rnftools

rnftools.mishmash.sample("simple_example",reads_in_tuple=1)

fa="../../../example1.fa"

rnftools.mishmash.ArtIllumina(
        fasta=fa,
        number_of_read_tuples=10000,
        read_length_1=100,
        read_length_2=0,
        other_params="-qs 5",
)

include: rnftools.include()
rule: input: rnftools.input()
```

2.2.4 Mapper evalution

In this chapter, we show how to evaluate read mappers using RNFtools. For this task, we will use a component called LAVEnder.

Basic example

The basic approach of mapper evaluation consists of the following steps:

1. Simulation of reads (see the previous chapter of this tutorial).
2. Mapping reads to a reference genome.
3. Creating the report.

The first step was described in the previous chapter. Once your simulated reads are mapped (step 2), you can create the following Snakefile:

```
import rnftools

rnftools.lavender.Report(
        # insert a list of directories with BAM files here:
        bam_dirs=["../02_mapping"],
        name="report_SE",
        keep_intermediate_files=True,
```

(continues on next page)

(continued from previous page)

```
        allowed_delta=5,
        default_x_run=[0.01,0.5]
)

rule all: input: rnftools.input()

include: rnftools.include()
```

When you run *snakemake*, RNFtools detect all BAM files in the specified directories, and starts evaluation and creates an interactive HTML report containing one panel for each directory. The `name` argument defines the name of the report (the final HTML file will have name `{name}.html`). Parameter `keep_intermediate_files` sets if the intermediate ES (evaluated individual segments) and ET (evaluated read pairs) files created during evaluation should be kept. The argument `allowed_delta` is used for setting maximum allowed distance between reported position and original position for considering the segment still correctly mapped.

Auxiliary files

For every BAM file, the following files are created.

- **HTML** – detailed report for the BAM file
- **ES** (mapping information: segments) – file with information about mapping categories of each segment
- **ET** (mapping information: read tuples) – file with information about category of entire read tuples
- **ROC** – source file for plotting graphs, it contains information about how many reads are in which category in dependence on threshold on mapping qualities
- **GP** – GnuPlot file used for plotting detailed graphs for this BAM (**SVG**, **PDF**)

Adjusting plotted graphs

For details about adjusting graphs, please see *LAVEnder*.

First, you can change default values for the basic graphs:

- range of x and y axes (`default_x_run`, `default_y_run`),
- sizes of created PDF and SVG files (`default_pdf_size_cm`, `default_svg_size_px`)
- label of x-axis (`default_x_label`)
- values on x-axis (`default_x_axis`).

Then you can add your own graphs using `rnftools.lavender.Report.add_graph` method.

2.2.5 Extending RNFtools

This chapter describes how to develop an extension for a new read simulator.

Step 1: Wrapper of the simulator in RNFtools

All wrappers of read simulators are located in this directory: https://github.com/karel-brinda/rnftools/blob/master/rnftools/mishmash/. To create a new wrapper, copy some existing one and modify the code inside.

Method `__init__` saves the parameters for a simulation. Some of them are mandatory (read length, FASTA file of the genome, etc.) and these are passed to `__init__` of the mother abstract class for a simulator.

Method `get_input` returns list of input files for simulation and one of them is your program.

Method, `create_fq` simulates reads (by calling a shell command using `rnftools.utils.shell`) and converts the obtained files to RNF-FASTQ.

The previous step should be done using a dedicated static method (typically named `recode_*_reads`). All code for conversion should be located in this method so that it can be used externally without creating an instance of the class.

When all these functions are created/adjusted, the class should be imported in `__init__.py`. Get inspired by existing code, with a high probability only little changes will be needed.

Step 2: Support in the `rnftools` program

The last step is plugging your converting static function into `rnftools` console program, which is contained in this file: https://github.com/karel-brinda/rnftools/blob/master/rnftools/scripts.py. You will have to add a new subcommand (which will call the static function) and create a parser for it. Again, follow existing code.

Step 3: Tests

Add corresponding tests (see the test directory).

2.3 Reference

Here you can find an automatically generated reference for all components of RNFtools.

2.3.1 MIShmash

Here you can find documentation for **all supported NGS read simulators**. Please note that you can use simplified names of classes like `rnftools.mishmash.ArtIllumina` (instead of longer `rnftools.mishmash.artIllumina.ArtIllumina`). General information about read simulators can be found on a special page called *Exhaustive list of read simulators*.

Contents

Classes for individual simulators

Art-Illumina: `rnftools.mishmash.ArtIllumina`

class rnftools.mishmash.ArtIllumina.**ArtIllumina**(*fasta*, *sequences=None*, *coverage=0*, *number_of_read_tuples=0*, *read_length_1=100*, *read_length_2=0*, *distance=500*, *distance_deviation=50.0*, *rng_seed=1*, *other_params=''*)

Bases: *rnftools.mishmash.Source.Source*

Class for the ART Illumina (http://www.niehs.nih.gov/research/resources/software/biostatistics/art/).

Single-end reads and pair-end reads simulations are supported. For pair-end simulations, lengths of both ends must be equal.

Parameters

- **fasta** (*str*) – File name of the genome from which read tuples are created (FASTA file). Corresponding ART parameter: `-i --in`.
- **sequences** (*set of int or str*) – FASTA sequences to extract. Sequences can be specified either by their ids, or by their names.
- **coverage** (*float*) – Average coverage of the genome (if number_of_read_tuples specified, then at least number_of_read_tuples will be simulated). Corresponding ART parameter: `-f --fcov`.
- **number_of_read_tuples** (*int*) – Number of read tuples (if coverage specified, then it sets the minimum number of reads to simulate). Corresponding ART parameter: `-c --rcount`.
- **read_length_1** (*int*) – Length of the first read. Corresponding ART parameter: `-l --len`.
- **read_length_2** (*int*) – Length of the second read (if zero, then single-end reads are simulated). Corresponding ART parameter: `-l --len`.
- **distance** (*int*) – Mean inner distance between reads. Corresponding ART parameter: `-m --mflen`.
- **distance_deviation** (*int*) – Standard devation of inner distances between reads. Corresponding ART parameter: `-s --sdev`.
- **rng_seed** (*int*) – Seed for simulator's random number generator. Corresponding ART parameter: `-rs --rndSeed`.
- **other_params** (*str*) – Other parameters which are used on commandline.

Raises `ValueError`

create_fq()
Simulate reads.

get_input()
Get list of input files (required to do simulation).

Returns List of input files

Return type list

get_output()
Get list of output files (created during simulation).

Returns List of input files

Return type list

CuReSim: rnftools.mishmash.CuReSim

class rnftools.mishmash.CuReSim.**CuReSim**(*fasta*, *sequences=None*, *coverage=0*, *number_of_read_tuples=0*, *read_length_1=100*, *read_length_2=0*, *random_reads=False*, *rng_seed=1*, *other_params=''*)
Bases: *rnftools.mishmash.Source.Source*

Class for CuReSim (http://www.pegase-biosciences.com/curesim-a-customized-read-simulator).

Only single-end reads simulations are supported.

Parameters

- **fasta** (*str*) – File name of the genome from which reads are created (FASTA file). Corresponding CuReSim parameter: `-f`.
- **sequences** (*set of int or str*) – FASTA sequences to extract. Sequences can be specified either by their ids, or by their names.
- **coverage** (*float*) – Average coverage of the genome (if number_of_read_tuples specified, then at least number_of_read_tuples will be simulated).
- **number_of_read_tuples** (*int*) – Number of read tuples (if coverage specified, then it sets the minimum number of reads to simulate). Corresponding CuReSim parameter: `-n`.
- **read_length_1** (*int*) – Length of the first read. Corresponding CuReSim parameter: `-m`.
- **read_length_2** (*int*) – Length of the second read. Fake parameter (unsupported by CuReSim).
- **random_reads** (*bool*) – Simulate random reads (see CuReSim documentation for more details).
- **rng_seed** (*int*) – Seed for simulator's random number generator. Fake parameter (unsupported by CuReSim).
- **other_params** (*str*) – Other parameters which are used on command-line.

Raises `ValueError`

create_fq()
Simulate reads.

get_input()
Get list of input files (required to do simulation).

Returns List of input files

Return type list

get_output()
Get list of output files (created during simulation).

Returns List of input files

Return type list

static recode_curesim_reads(*curesim_fastq_fo*, *rnf_fastq_fo*, *fai_fo*, *genome_id*, *number_of_read_tuples=1000000000*, *recode_random=False*)

Recode CuReSim output FASTQ file to the RNF-compatible output FASTQ file.

Parameters

- **curesim_fastq_fo** (*file object*) – File object of CuReSim FASTQ file.
- **fastq_rnf_fo** (*file object*) – File object of RNF FASTQ.
- **fai_fo** (*file object*) – File object for FAI file of the reference genome.
- **genome_id** (*int*) – RNF genome ID to be used.
- **number_of_read_tuples** (*int*) – Expected number of read tuples (to estimate number of digits in RNF).
- **recode_random** (*bool*) – Recode random reads.

Raises ValueError

DwgSim: rnftools.mishmash.DwgSim

class rnftools.mishmash.DwgSim.**DwgSim**(*fasta*, *sequences=None*, *coverage=0*, *number_of_read_tuples=0*, *read_length_1=100*, *read_length_2=0*, *distance=500*, *distance_deviation=50.0*, *rng_seed=1*, *haploid_mode=False*, *error_rate_1=0.02*, *error_rate_2=0.02*, *mutation_rate=0.001*, *indels=0.15*, *prob_indel_ext=0.3*, *estimate_unknown_values=False*, *other_params=""*)

Bases: *rnftools.mishmash.Source.Source*

Class for DWGsim (https://github.com/nh13/DWGSIM/wiki).

Both single-end and paired-end simulations are supported. In paired-end simulations, reads can have different lengths. Note that there is a bug in DWGsim documentation: coordinates are 1-based.

Parameters

- **fasta** (*str*) – File name of the genome from which reads are created (FASTA file).
- **sequences** (*set of int or str*) – FASTA sequences to extract. Sequences can be specified either by their ids, or by their names.
- **coverage** (*float*) – Average coverage of the genome (if number_of_read_tuples specified, then at least number_of_read_tuples will be simulated). Corresponding DWGsim parameter: -C.
- **number_of_read_tuples** (*int*) – Number of read tuples (if coverage specified, then it sets the minimum number of reads to simulate). Corresponding DWGsim parameter: -N.
- **read_length_1** (*int*) – Length of the first read. Corresponding DWGsim parameter: -1.
- **read_length_2** (*int*) – Length of the second read (if zero, then single-end simulation performed). Corresponding DWGsim parameter: -2.
- **distance** (*int*) – Mean inner distance between reads. Corresponding DWGsim parameter: -d.

- **distance_deviation** (*int*) – Standard deviation of inner distances between both reads. Corresponding DWGsim parameter: `-s`.
- **rng_seed** (*int*) – Seed for simulator's random number generator. Corresponding DWGsim parameter: `-z`.
- **haploid_mode** (*bools*) – Simulate reads in haploid mode. Corresponding DWGsim parameter: `-H`.
- **error_rate_1** (*float*) – Sequencing error rate in the first read. Corresponding DWGsim parameter: `-e`.
- **error_rate_2** (*float*) – Sequencing error rate in the second read. Corresponding DWGsim parameter: `-E`.
- **mutation_rate** (*float*) – Mutation rate. Corresponding DWGsim parameter: `-e`.
- **indels** (*float*) – Rate of indels in mutations. Corresponding DWGsim parameter: `-R`.
- **prob_indel_ext** (*float*) – Probability that an indel is extended. Corresponding DWGsim parameter: `-X`.
- **estimate_unknown_values** (*bool*) – Estimate unknown values (coordinates missing in DWGsim output).
- **other_params** (*str*) – Other parameters which are used on command-line.

Raises `ValueError`

create_fq()
Simulate reads.

get_input()
Get list of input files (required to do simulation).

Returns List of input files

Return type list

get_output()
Get list of output files (created during simulation).

Returns List of input files

Return type list

static recode_dwgsim_reads(*dwgsim_prefix*, *fastq_rnf_fo*, *fai_fo*, *genome_id*, *estimate_unknown_values*, *number_of_read_tuples=1000000000*)
Convert DwgSim FASTQ file to RNF FASTQ file.

Parameters

- **dwgsim_prefix** (*str*) – DwgSim prefix of the simulation (see its commandline parameters).
- **fastq_rnf_fo** (*file*) – File object of RNF FASTQ.
- **fai_fo** (*file*) – File object for FAI file of the reference genome.
- **genome_id** (*int*) – RNF genome ID to be used.
- **estimate_unknown_values** (*bool*) – Estimate unknown values (right coordinate of each end).
- **number_of_read_tuples** (*int*) – Estimate of number of simulated read tuples (to set width).

Mason (Illumina mode): `rnftools.mishmash.MasonIllumina`

class rnftools.mishmash.MasonIllumina.**MasonIllumina**(*fasta*, *sequences=None*, *coverage=0*, *number_of_read_tuples=0*, *read_length_1=100*, *read_length_2=0*, *distance=500*, *distance_deviation=50*, *rng_seed=1*, *other_params=''*)

Bases: *rnftools.mishmash.Source.Source*

Class for the Mason - Illumina mode (https://www.seqan.de/projects/mason/).

Single-end reads and pair-end reads simulations are supported. For pair-end simulations, lengths of both ends must be equal.

Parameters

- **fasta** (*str*) – File name of the genome from which read tuples are created (FASTA file). Corresponding Mason parameter: `-ir, --input-reference`.
- **sequences** (*set of int or str*) – FASTA sequences to extract. Sequences can be specified either by their ids, or by their names.
- **coverage** (*float*) – Average coverage of the genome (if number_of_read_tuples specified, then at least number_of_read_tuples will be simulated).
- **number_of_read_tuples** (*int*) – Number of read tuples (if coverage specified, then it sets the minimum number of reads to simulate). Corresponding Mason parameter: `-n, --num-fragments`.
- **read_length_1** (*int*) – Length of the first read. Corresponding Mason parameter: `--illumina-read-length`.
- **read_length_2** (*int*) – Length of the read read (if zero, then single-end reads are simulated). Corresponding Mason parameter: `--illumina-read-length`.
- **distance** (*int*) – Mean inner distance between reads. Corresponding Mason parameter: `--fragment-mean-size`.
- **distance_deviation** (*int*) – Standard devation of inner distances between reads. Corresponding Mason parameter: `--fragment-size-std-dev`.
- **rng_seed** (*int*) – Seed for simulator's random number generator. Corresponding Mason parameter: `--seed`.
- **other_params** (*str*) – Other parameters which are used on command-line.

Raises `ValueError`

create_fq()
Simulate reads.

get_input()
Get list of input files (required to do simulation).

Returns List of input files

Return type list

get_output()
Get list of output files (created during simulation).

Returns List of input files

Return type list

WgSim: rnftools.mishmash.WgSim

class rnftools.mishmash.WgSim.**WgSim**(*fasta*, *sequences=None*, *coverage=0*, *number_of_read_tuples=0*, *read_length_1=100*, *read_length_2=0*, *distance=500*, *distance_deviation=50.0*, *rng_seed=1*, *haploid_mode=False*, *error_rate=0.02*, *mutation_rate=0.001*, *indels=0.15*, *prob_indel_ext=0.3*, *other_params=''*)

Bases: *rnftools.mishmash.Source.Source*

Class for the WGsim (https://github.com/lh3/wgsim).

Single-end and pair-end simulations are supported. For pair-end simulations, reads can have different lengths.

Parameters

- **fasta** (*str*) – File name of the genome from which reads are created (FASTA file).
- **sequences** (*set of int or str*) – FASTA sequences to extract. Sequences can be specified either by their ids, or by their names.
- **coverage** (*float*) – Average coverage of the genome (if number_of_read_tuples specified, then at least number_of_read_tuples will be simulated).
- **number_of_read_tuples** (*int*) – Number of read tuples (if coverage specified, then this sets the minimum number of reads to simulate). Corresponding WGsim parameter: `-N`.
- **read_length_1** (*int*) – Length of the first read. Corresponding WGsim parameter: `-1`.
- **read_length_2** (*int*) – Length of the second read (if zero, then single-end reads are simulated). Corresponding WGsim parameter: `-2`.
- **distance** (*int*) – Mean outer distance of reads. Corresponding WGsim parameter: `-d`.
- **distance_deviation** (*int*) – Standard deviation of outer distances of reads. Corresponding WGsim parameter: `-s`.
- **rng_seed** (*int*) – Seed for simulator's random number generator. Corresponding WGsim parameter: `-S`.
- **haploid_mode** (*bools*) – Simulate reads in haploid mode. Corresponding WGsim parameter: `-h`.
- **error_rate** (*float*) – Sequencing error rate (sequencing errors). Corresponding WGsim parameter: `-e`.
- **mutation_rate** (*float*) – Mutation rate. Corresponding WGsim parameter: `-r`.
- **indels** (*float*) – Rate of indels in mutations. Corresponding WGsim parameter: `-R`.
- **prob_indel_ext** (*float*) – Probability that an indel is extended. Corresponding WGsim parameter: `-X`.
- **other_params** (*str*) – Other parameters on commandline.

Raises `ValueError`

create_fq()

Simulate reads.

get_input()
Get list of input files (required to do simulation).

Returns List of input files

Return type list

get_output()
Get list of output files (created during simulation).

Returns List of input files

Return type list

static recode_wgsim_reads(*rnf_fastq_fo*, *fai_fo*, *genome_id*, *wgsim_fastq_1_fn*, *wgsim_fastq_2_fn=None*, *number_of_read_tuples=1000000000*)
Convert WgSim FASTQ files to RNF FASTQ files.

Parameters

- **rnf_fastq_fo** (*file*) – File object of the target RNF file.
- **fai_fo** (*file*) – File object of FAI index of the reference genome.
- **genome_id** (*int*) – RNF genome ID.
- **wgsim_fastq_1_fn** (*str*) – File name of the first WgSim FASTQ file.
- **wgsim_fastq_2_fn** (*str*) – File name of the second WgSim FASTQ file.
- **number_of_read_tuples** (*int*) – Expected number of read tuples (to estimate widths).

Abstract class for a simulator: rnftools.mishmash.Source

class rnftools.mishmash.Source.**Source**(*fasta*, *reads_in_tuple*, *rng_seed*, *sequences*, *number_of_required_cores=1*)
Bases: object

Abstract class for a genome from which read tuples are simulated.

Parameters

- **fasta** (*str*) – File name of the genome from which reads are created (FASTA file).
- **reads_in_tuple** (*int*) – Number of reads in each read tuple.
- **rng_seed** (*int*) – Seed for simulator's random number generator.
- **sequences** (*set of int or str*) – FASTA sequences to extract. Sequences can be specified either by their ids, or by their names.
- **number_of_required_cores** (*int*) – Number of cores used by the simulator. This parameter is used to prevent running other threads or programs at the same time.

clean()
Clean working directory.

create_fa()
Create a FASTA file with extracted sequences.

create_fq()
Simulate reads.

fa0_fn()
Get input FASTA file.

Returns Input FASTA file.

Return type str

fa_fn()
Get output FASTA file (with selected chromosomes).

Returns Output FASTA file.

Return type str

fq_fn()
Get file name of the output FASTQ file.

Returns Output FASTQ file

Return type str

get_dir()
Get working directory.

Returns Working directory.

Return type str

get_genome_id()
Get genome ID.

Returns Genome ID.

Return type int

get_input()
Get list of input files (required to do simulation).

Returns List of input files

Return type list

get_number_of_required_cores()
Get number of required cores.

Returns Number of required cores.

Return type int

get_output()
Get list of output files (created during simulation).

Returns List of input files

Return type list

get_reads_in_tuple()
Get number of entries in a read tuple.

Returns Number of reads in a read tuple.

Return type int

static recode_sam_reads(*sam_fn*, *fastq_rnf_fo*, *fai_fo*, *genome_id*, *number_of_read_tuples=1000000000*, *simulator_name=None*, *allow_unmapped=False*)
Transform a SAM file to RNF-compatible FASTQ.

Parameters

- **sam_fn** (*str*) – SAM/BAM file - file name.

- **fastq_rnf_fo** (*str*) – Output FASTQ file - file object.
- **fai_fo** (*str*) – FAI index of the reference genome - file object.
- **genome_id** (*int*) – Genome ID for RNF.
- **number_of_read_tuples** (*int*) – Expected number of read tuples (to set width of read tuple id).
- **simulator_name** (*str*) – Name of the simulator. Used for comment in read tuple name.
- **allow_unmapped** (*bool*) – Allow unmapped reads.

Raises `NotImplementedError`

2.3.2 LAVEnder

Contents

Report class: `rnftools.lavender.Report`

class rnftools.lavender.Report.**Report**(*name*, *title=None*, *description=''*, *allowed_delta=5*, *bam_dirs=None*, *panels=None*, *default_x_run=(1e-05, 1.0)*, *default_y_run=(60, 100)*, *default_pdf_size_cm=(10, 10)*, *default_svg_size_px=(640, 640)*, *render_pdf_method='any'*, *keep_intermediate_files=False*, *compress_intermediate_files=True*, *default_x_axis='({m}+{w})/({M}+{m}+{w})'*, *default_x_label='FDR in mapping {{/:Italic(#wrongly mapped reads / #mapped reads)}} '*, *gp_style_func=<function _default_gp_style_func>*)

Bases: `object`

Class for an entire report.

Parameters

- **name** (*str*) – Name of the report (name of output file and dir).
- **title** (*str*) – Title of the report (if None, then name is used).
- **description** (*str*) – Description of the report.
- **bam_dirs** (*list of str*) – Directories with BAM files (this option is mutually exclusive with the 'panels' option).
- **panels** (*list of dicts*) – Advanced configuration for panels (list of dictionaries with the following keys: 'bam_dir' (mandatory); 'panel_dir', 'name', 'title' (optional)).

- **allowed_delta** (*int*) – Tolerance of difference of coordinates between true (i.e., expected) alignment and real alignment (very important parameter!).
- **default_x_run** (*(float, float)*) – Range for x-axis in GnuPlot plots.
- **default_y_run** (*(float, float)*) – Range for y-axis in GnuPlot plots.
- **default_pdf_size_cm** (*(float, float)*) – Legacy parameter (does not have any effect).
- **default_svg_size_px** (*(int, int)*) – Size of SVG picture.
- **render_pdf_method** (*str*) – Method for svg42pdf to render PDF (None / 'any' (default) / 'cairo' / 'reportlab' / 'inkscape' / 'imagemagick' / 'wkhtmltopdf').
- **keep_intermediate_files** (*bool*) – Keep files created in intermediate steps during evaluation.
- **compress_intermediate_files** (*bool*) – Compress files created in intermediate steps during evaluation.
- **default_x_axis** (*str*) – Values on x-axis, e.g., "({m}+{w})/({M}+{m}+{w})".
- **default_x_label** (*str*) – Label on x-axis.
- **gp_style_func** (*function(i, nb)*) – Function assigning GnuPlot styles for overall graphs. Arguments: i: 0-based id of curve, nb: number of curves.

add_graph(*y*, *x_label=None*, *y_label=''*, *title=''*, *x_run=None*, *y_run=None*, *svg_size_px=None*, *key_position='bottom right'*)

Add a new graph to the overlap report.

Parameters

- **y** (*str*) – Value plotted on y-axis.
- **x_label** (*str*) – Label on x-axis.
- **y_label** (*str*) – Label on y-axis.
- **title** (*str*) – Title of the plot.
- **x_run** (*(float, float)*) – x-range.
- **y_run** (*(int, int)*) – y-rang.
- **svg_size_px** (*(int, int*) – Size of SVG image in pixels.
- **key_position** (*str*) – GnuPlot position of the legend.

clean()

Remove all temporary files.

create_html()

Create HTML report.

get_panels()

Get all contained panels.

get_report_dir()

Get directory report's auxiliary files.

html_fn()

Get name of the HTML file of the report.

Panel class: `rnftools.lavender.Panel`

class rnftools.lavender.Panel.**Panel**(*report*, *bam_dir*, *panel_dir*, *name*, *title*, *render_pdf_method*, *keep_intermediate_files*, *compress_intermediate_files*, *default_x_axis*, *default_x_label*, *gp_style_func=<function _default_gp_style_func>*)

Bases: `object`

Class for a single panel in a HTML report.

Parameters

- **report** (*rnftools.lavender.Report*) – The owner report.
- **bam_dir** (*str*) – Directory to the BAM files for this panel.
- **panel_dir** (*str*) – Directory with auxiliary files for this panel.
- **name** (*str*) – Name of the panel (used for CSS, etc.).
- **title** (*str*) – Title of the panel (to be displayed).
- **render_pdf_method** (*str*) – Method for svg42pdf to render PDF (None / 'any' / 'cairo' / 'reportlab' / 'inkscape' / 'imagemagick' / 'wkhtmltopdf').
- **keep_intermediate_files** (*bool*) – Keep files created in intermediate steps during evaluation.
- **compress_intermediate_files** (*bool*) – Compress files created in intermediate steps during evaluation.
- **default_x_axis** (*str*) – Values on x-axis, e.g., "({m}+{w})/({M}+{m}+{w})".
- **default_x_label** (*str*) – Label on x-axis.
- **gp_style_func** (*function(i, nb)*) – Function assigning GnuPlot styles for overall graphs. Arguments: i: 0-based id of curve, nb: number of curves.

Raises `ValueError`

create_gp()
Create GnuPlot file.

create_graphics()
Create images related to this panel.

create_tar()
Create a tar file with all the files.

get_bams()
Get BAMs for this panel.

get_html_column()
Get a HTML column for this panel.

get_panel_dir()
Get the directory with panel's auxiliary files.

get_report()
Get the report.

get_required_files()
Get all required files.

gp_fn()
Get the GnuPlot file name for the overall graphs.

svg_fns()
Get the PDF file names for the overall graphs (empty list if they are not rendered).

tar_fn()
Get the TAR file name for the archive with data.

BAM class: rnftools.lavender.Bam

class rnftools.lavender.Bam.**Bam**(*panel*, *bam_fn*, *name*, *render_pdf_method*, *keep_intermediate_files*, *compress_intermediate_files*, *default_x_axis*, *default_x_label*)

Bases: object

Class for a BAM file.

Parameters

- **panel** (*rnftools.lavender.Panel*) – Panel containing this BAM file.
- **bam_fn** (*str*) – BAM filename.
- **name** (*str*) – Name for this report.
- **keep_intermediate_files** (*bool*) – Keep files created in intermediate steps during evaluation.
- **render_pdf_method** (*str*) – Method for svg42pdf to render PDF (None / 'any' / 'cairo' / 'reportlab' / 'inkscape' / 'imagemagick' / 'wkhtmltopdf').
- **compress_intermediate_files** (*bool*) – Compress files created in intermediate steps during evaluation.
- **default_x_axis** (*str*) – Values on x-axis, e.g., "({m}+{w})/({M}+{m}+{w})".
- **default_x_label** (*str*) – Label on x-axis.

static bam2es(*bam_fn*, *es_fo*, *allowed_delta*)
Convert BAM file to ES file.

Parameters

- **bam_fn** (*str*) – File name of the BAM file.
- **bam_fo** (*file*) – File object of the ES file.
- **allowed_delta** (*int*) – Maximal allowed coordinates difference for correct reads.

bam_fn()
Get name of the BAM file.

create_es()
Create an ES (intermediate) file for this BAM file. This is the function which asses if an alignment is correct

create_et()
Create a et file for this BAM file (mapping information about read tuples).

raises: ValueError

create_gp()
Create a GnuPlot file for this BAM file.

create_graphics()
Create images related to this BAM file using GnuPlot.

create_html()
Create a HTML page for this BAM file.

create_roc()
Create a ROC file for this BAM file.

raises: ValueError

static es2et(*es_fo*, *et_fo*)
Convert ES to ET.

Parameters

- **es_fo** (*file*) – File object for the ES file.
- **et_fo** (*file*) – File object for the ET file.

es_fn()
Get name of the es file.

static et2roc(*et_fo*, *roc_fo*)
ET to ROC conversion.

Parameters

- **et_fo** (*file*) – File object for the ET file.
- **roc_fo** (*file*) – File object for the ROC file.

raises: ValueError

et_fn()
Get name of the et file.

get_name()
Get name associated with the BAM.

get_required_files()
Get names of all files required to complete the report.

gp_fn()
Get name of the GP file.

html_fn()
Get name of the HTML report.

pdf_fn()
Get name of the SVG file.

roc_fn()
Get name of the ROC file.

svg_fn()
Get name of the SVG file.

2.3.3 RNF format library

Read tuple: rnftools.rnfformat.ReadTuple

class rnftools.rnfformat.ReadTuple.**ReadTuple**(*segments=[]*, *read_tuple_id=0*, *prefix=''*, *suffix=''*)

Bases: object

Class for a RNF read tuple.

Parameters

- **segments** (*list of rnftools.rnfformat.Segment*) – Segments of the read.
- **read_tuple_id** (*int*) – Read tuple ID.
- **prefix** (*str*) – Prefix for the read name.
- **suffix** (*str*) – Suffix for the read name.

destringize(*string*)

Get RNF values for this read from its textual representation and save them into this object.

Parameters **string** (*str*) – Textual representation of a read.

Raises ValueError

stringize(*rnf_profile=<rnftools.rnfformat.RnfProfile.RnfProfile object>*)

Create RNF representation of this read.

Parameters

- **read_tuple_id_width** (*int*) – Maximal expected string length of read tuple ID.
- **genome_id_width** (*int*) – Maximal expected string length of genome ID.
- **chr_id_width** (*int*) – Maximal expected string length of chromosome ID.
- **coor_width** (*int*) – Maximal expected string length of a coordinate.

Segment: rnftools.rnfformat.Segment

class rnftools.rnfformat.Segment.**Segment**(*genome_id=0*, *chr_id=0*, *direction='N'*, *left=0*, *right=0*)

Bases: object

Class for a single segment in a RNF read name.

destringize(*string*)

Get RNF values for this segment from its textual representation and save them into this object.

Parameters **string** (*str*) – Textual representation of a segment.

stringize(*rnf_profile*)

Create RNF representation of this segment.

Parameters **rnf_profile** (*rnftools.rnfformat.RnfProfile*) – RNF profile (with widths).

RNF profile: `rnftools.rnfformat.RnfProfile`

class rnftools.rnfformat.RnfProfile.**RnfProfile**(*prefix_width=0*, *read_tuple_id_width=8*, *genome_id_width=1*, *chr_id_width=2*, *coor_width=9*, *read_tuple_name=None*)

Bases: `object`

Class for profile of RNF reads (widths).

Parameters

- **prefix_width** (*int*) – Length of prefix.
- **read_tuple_id_width** (*int*) – Width of read tuple ID
- **genome_id_width** (*int*) – Width of genome ID.
- **chr_id_width** (*int*) – Width of chromosome ID.
- **coor_width** (*int*) – Width of coordinate width.
- **read_tuple_name** (*str*) – Read tuple name to initialize all the values.

prefix_width
int – Length of prefix.

read_tuple_id_width
int – Width of read tuple ID

genome_id_width
int – Width of genome ID.

chr_id_width
int – Width of chromosome ID.

coor_width
int – Width of coordinate width.

apply(*read_tuple_name*, *read_tuple_id=None*, *synchronize_widths=True*)
Apply profile on a read tuple name and update read tuple ID.

Parameters

- **read_tuple_name** (*str*) – Read tuple name to be updated.
- **read_tuple_id** (*id*) – New read tuple ID.
- **synchronize_widths** (*bool*) – Update widths (in accordance to this profile).

check(*read_tuple_name*)
Check if the given read tuple name satisfies this profile.

Parameters **read_tuple_name** (*str*) – Read tuple name.

combine()
Combine more profiles and set their maximal values.

Parameters ***rnf_profiles** (*rnftools.rnfformat.RnfProfile*) – RNF profile.

get_rnf_name(*read_tuple*)
Get well-formatted RNF representation of a read tuple.

read_tuple (rnftools.rnfformat.ReadTuple): Read tuple.

load(*read_tuple_name*)
Load RNF values from a read tuple name.

Parameters **read_tuple_name** (*str*) – Read tuple name which the values are taken from.

FASTQ creator: rnftools.rnfformat.FqCreator

class rnftools.rnfformat.FqCreator.**FqCreator**(*fastq_fo*, *read_tuple_id_width=16*, *genome_id_width=2*, *chr_id_width=2*, *coor_width=8*, *info_reads_in_tuple=True*, *info_simulator=None*)

Bases: object

Class for writing RNF reads to FASTQ files.

Every new read is added to the internal buffer. If read tuple ID is different, buffer is flushed. Hence, reads from the same tuple must be added in a series. It does not matter in which order are blocks reported and with which exact reads, they will be sorted during flushing.

Parameters

- **fastq_fo** (*str*) – Output FASTQ file - file object.
- **read_tuple_id_width** (*int*) – Maximal expected string length of read tuple ID.
- **genome_id_width** (*int*) – Maximal expected string length of genome ID.
- **chr_id_width** (*int*) – Maximal expected string length of chromosome ID.
- **coor_width** (*int*) – Maximal expected string length of a coordinate.
- **info_reads_in_tuple** (*bool*) – Include information about reads as a RNF comment.
- **info_simulator** (*str*) – Name of used simulator (to be included as a RNF comment).

add_read(*read_tuple_id*, *bases*, *qualities*, *segments*)
Add a new read to the current buffer. If it is a new read tuple (detected from ID), the buffer will be flushed.

Parameters

- **read_tuple_id** (*int*) – ID of the read tuple.
- **bases** (*str*) – Sequence of bases.
- **qualities** (*str*) – Sequence of FASTQ qualities.
- **segments** (*list of rnftools.rnfformat.segment*) – List of segments constituting the read.

empty()
Empty all internal buffers.

flush_read_tuple()
Flush the internal buffer of reads.

is_empty()
All internal buffer empty?

FASTQ merger: rnftools.rnfformat.FqMerger

class rnftools.rnfformat.FqMerger.**FqMerger**(*mode*, *input_files_fn*, *output_prefix*)
Bases: object

Class for merging several RNF FASTQ files.

Parameters

- **mode** (*str*) – Output mode (single-end / paired-end-bwa / paired-end-bfast).
- **input_files_fn** (*list*) – List of file names of input FASTQ files.
- **output_prefix** (*str*) – Prefix for output FASTQ files.

run()
Run merging.

RNF validator: rnftools.rnfformat.Validator

class rnftools.rnfformat.Validator.**Validator**(*initial_read_tuple_name*, *report_only_first=True*, *warnings_as_errors=False*)

Bases: object

Class for validation of RNF.

Parameters

- **initial_read_tuple_name** (*str*) – Initial read tuple name to detect profile (widths).
- **report_only_first** (*bool*) – Report only first occurrence of every error.
- **warnings_as_errors** (*bool*) – Treat warnings as errors (error code).

get_return_code()
Get final return code (0 = ok, 1=error appeared).

report_error(*read_tuple_name*, *error_name*, *wrong=''*, *message=''*, *warning=False*)
Report an error.

Parameters

- **()** (*error_name*) – Name of the read tuple.
- **()** – Name of the error.
- **wrong** (*str*) – What is wrong.
- **message** (*str*) – Additional msessage to be printed.
- **warning** (*bool*) – Warning (not an error).

validate(*read_tuple_name*)
Check RNF validity of a read tuple.

Parameters **read_tuple_name** (*str*) – Read tuple name to be checked.s

2.3.4 Command-line tool

Even though SnakeMake-based approach is the prefered way to use RNFtools, we provide also a command-line tool `rnftools` with most of functionality. Here you can find help messages for its subcommands.

Contents

General

`rnftools` (list of subcommands)

```
$ rnftools  -h

usage: rnftools [-h]
                {,check,publication,validate,liftover,sam2rnf,art2rnf,
↪curesim2rnf,dwgsim2rnf,mason2rnf,wgsim2rnf,merge,sam2es,es2et,et2roc,
↪sam2roc}
                ...

==================================================
RNFtools -  http://karel-brinda.github.io/rnftools
--------------------------------------------------
version: 0.3.1.1
contact: Karel Brinda (karel.brinda@univ-mlv.fr)
==================================================

positional arguments:
  {,check,publication,validate,liftover,sam2rnf,art2rnf,curesim2rnf,
↪dwgsim2rnf,mason2rnf,wgsim2rnf,merge,sam2es,es2et,et2roc,sam2roc}

    check               Check for the latest version.
```

(continues on next page)

(continued from previous page)

```
    publication          Print information about the associated publication.
    validate             Validate RNF names in a FASTQ file.
    liftover             Liftover genomic coordinates in RNF names.

                         ---------------------[MIShmash]----------------------
    sam2rnf              Convert a SAM/BAM file to RNF-FASTQ.
    art2rnf              Convert output of Art to RNF-FASTQ.
    curesim2rnf          Convert output of CuReSim to RNF-FASTQ.
    dwgsim2rnf           Convert output of DwgSim to RNF-FASTQ.
    mason2rnf            Convert output of Mason to RNF-FASTQ.
    wgsim2rnf            Convert output of WgSim to RNF-FASTQ.
    merge                Merge RNF-FASTQ files and convert the output to
                         'traditional' FASTQ.

                         ---------------------[LAVEnder]----------------------
    sam2es               Convert SAM/BAM with reads in RNF to ES (evaluation␣
↪of
                         segments).
    es2et                Convert ES to ET (evaluation of read tuples).
    et2roc               Convert ET to ROC (final statistics).
    sam2roc              Previous three steps in a single command.

optional arguments:
  -h, --help             show this help message and exit
```

rnftools check

```
$ rnftools check -h

usage: rnftools check [-h]

Check if RNFtools are up-to-date.

optional arguments:
  -h, --help  show this help message and exit
```

rnftools publication

```
$ rnftools publication -h

usage: rnftools publication [-h]

Print information about the associated publication.

optional arguments:
  -h, --help  show this help message and exit
```

rnftools validate

```
$ rnftools validate -h

usage: rnftools validate [-h] -i file [-w] [-a]

Validate RNF names in a FASTQ file.

optional arguments:
  -h, --help            show this help message and exit
  -i file, --fastq file
                        FASTQ file to be validated.
  -w, --warnings-as-errors
                        Treat warnings as errors.
  -a, --all-occurrences
                        Report all occurrences of warnings and errors.
```

rnftools liftover

```
$ rnftools liftover -h

usage: rnftools liftover [-h] [-c file] -g int [-x file] [--invert]
                         [--input-format str] [--output-format str]
                         input output

Liftover genomic coordinates in RNF names in a SAM/BAM files or in a FASTQ
file.

positional arguments:
  input                 Input file to be transformed (- for standard input).
  output                Output file to be transformed (- for standard␣
↪output).

optional arguments:
  -h, --help            show this help message and exit
  -c file, --chain file
                        Chain liftover file for coordinates transformation.
                        [no transformation]
  -g int, --genome-id int
                        ID of genome to be transformed.
  -x file, --faidx file
                        Fasta index of the reference sequence. [extract from
                        chain file]
  --invert              Invert chain file (transformation in the other
                        direction).
  --input-format str    Input format (SAM/BAM/FASTQ). [autodetect]
  --output-format str   Output format (SAM/BAM/FASTQ). [autodetect]
```

MIShmash

rnftools sam2rnf

```
$ rnftools sam2rnf -h

usage: rnftools sam2rnf [-h] -s file -o file -x file [-g int] [-u]
```

(continues on next page)

(continued from previous page)

```
Convert a SAM/BAM file to RNF-FASTQ.

optional arguments:
  -h, --help            show this help message and exit
  -s file, --sam file   Input SAM/BAM with true (expected) alignments of the
                        reads (- for standard input).
  -o file, --rnf-fastq file
                        Output FASTQ file (- for standard output).
  -x file, --faidx file
                        FAI index of the reference FASTA file (- for standard
                        input). It can be created using 'samtools faidx'.
  -g int, --genome-id int
                        Genome ID in RNF (default: 1).
  -u, --allow-unmapped  Allow unmapped reads.
```

rnftools art2rnf

```
$ rnftools art2rnf -h

usage: rnftools art2rnf [-h] -s file -o file -x file [-g int] [-u] [-n str]

Convert an Art SAM file to RNF-FASTQ. Note that Art produces non-standard SAM
files and manual editation might be necessary. In particular, when a FASTA
file contains comments, Art left them in the sequence name. Comments must be
removed in their corresponding @SQ headers in the SAM file, otherwise all
reads are considered to be unmapped by this script.

optional arguments:
  -h, --help            show this help message and exit
  -s file, --sam file   Input SAM/BAM with true (expected) alignments of the
                        reads (- for standard input).
  -o file, --rnf-fastq file
                        Output FASTQ file (- for standard output).
  -x file, --faidx file
                        FAI index of the reference FASTA file (- for standard
                        input). It can be created using 'samtools faidx'.
  -g int, --genome-id int
                        Genome ID in RNF (default: 1).
  -u, --allow-unmapped  Allow unmapped reads.
  -n str, --simulator-name str
                        Name of the simulator (for RNF).
```

rnftools curesim2rnf

```
$ rnftools curesim2rnf -h

usage: rnftools curesim2rnf [-h] -c file -o file -x file [-g int]

Convert a CuReSim FASTQ file to RNF-FASTQ.

optional arguments:
```

(continues on next page)

(continued from previous page)

```
  -h, --help            show this help message and exit
  -c file, --curesim-fastq file
                        CuReSim FASTQ file (- for standard input).
  -o file, --rnf-fastq file
                        Output FASTQ file (- for standard output).
  -x file, --faidx file
                        FAI index of the reference FASTA file (- for standard
                        input). It can be created using 'samtools faidx'.
  -g int, --genome-id int
                        Genome ID in RNF (default: 1).
```

rnftools dwgsim2rnf

```
$ rnftools dwgsim2rnf -h

usage: rnftools dwgsim2rnf [-h] -p str [-e] -o file -x file [-g int]

Convert a DwgSim FASTQ file (dwgsim_prefix.bfast.fastq) to RNF-FASTQ.

optional arguments:
  -h, --help            show this help message and exit
  -p str, --dwgsim-prefix str
                        Prefix for DwgSim.
  -e, --estimate-unknown
                        Estimate unknown values.
  -o file, --rnf-fastq file
                        Output FASTQ file (- for standard output).
  -x file, --faidx file
                        FAI index of the reference FASTA file (- for standard
                        input). It can be created using 'samtools faidx'.
  -g int, --genome-id int
                        Genome ID in RNF (default: 1).
```

rnftools mason2rnf

```
$ rnftools mason2rnf -h

usage: rnftools mason2rnf [-h] -s file -o file -x file [-g int] [-u] [-n str]

Convert a Mason SAM file to RNF-FASTQ.

optional arguments:
  -h, --help            show this help message and exit
  -s file, --sam file   Input SAM/BAM with true (expected) alignments of the
                        reads (- for standard input).
  -o file, --rnf-fastq file
                        Output FASTQ file (- for standard output).
  -x file, --faidx file
                        FAI index of the reference FASTA file (- for standard
                        input). It can be created using 'samtools faidx'.
  -g int, --genome-id int
                        Genome ID in RNF (default: 1).
```

(continues on next page)

(continued from previous page)

```
  -u, --allow-unmapped  Allow unmapped reads.
  -n str, --simulator-name str
                        Name of the simulator (for RNF).
```

rnftools wgsim2rnf

```
$ rnftools wgsim2rnf -h

usage: rnftools wgsim2rnf [-h] -1 file [-2 file] -o file -x file [-g int] [-
↪u]

Convert WgSim FASTQ files to RNF-FASTQ.

optional arguments:
  -h, --help            show this help message and exit
  -1 file, --wgsim-fastq-1 file
                        First WgSim FASTQ file (- for standard input)
  -2 file, --wgsim-fastq-2 file
                        Second WgSim FASTQ file (in case of paired-end reads,
                        default: none).
  -o file, --rnf-fastq file
                        Output FASTQ file (- for standard output).
  -x file, --faidx file
                        FAI index of the reference FASTA file (- for standard
                        input). It can be created using 'samtools faidx'.
  -g int, --genome-id int
                        Genome ID in RNF (default: 1).
  -u, --allow-unmapped  Allow unmapped reads.
```

rnftools merge

```
$ rnftools merge -h

usage: rnftools merge [-h] -i inp [inp ...] -m mode -o out

todo

optional arguments:
  -h, --help        show this help message and exit
  -i inp [inp ...]  input FASTQ files
  -m mode           mode for mergeing files (single-end / paired-end-bwa /␣
↪paired-end-bfast)
  -o out            output prefix

Source RNF-FASTQ files should satisfy the following conditions:
        1) Each file contains only reads corresponding to one genome (with␣
↪the
           same genome id).
        2) All files contain reads of the same type (single-end / paired-
↪end).
        3) Reads with more reads per tuple (e.g., paired-end) have '/1', etc.
           in suffix (for identification of nb of read).
```

LAVEnder

rnftools sam2es

```
$ rnftools sam2es -h

usage: rnftools sam2es [-h] -i file -o file [-d int]

todo

optional arguments:
  -h, --help            show this help message and exit
  -i file, --sam file   SAM/BAM with aligned RNF reads(- for standard input).
  -o file, --es file    Output ES file (evaluated segments, - for standard
                        output).
  -d int, --allowed-delta int
                        Tolerance of difference of coordinates between true
                        (i.e., expected) alignment and real alignment (very
                        important parameter!) (default: 5).
```

rnftools es2et

```
$ rnftools es2et -h

usage: rnftools es2et [-h] -i file -o file

todo

optional arguments:
  -h, --help          show this help message and exit
  -i file, --es file  Input ES file (evaluated segments, - for standard
                      input).
  -o file, --et file  Output ET file (evaluated read tuples, - for standard
                      output).
```

rnftools et2roc

```
$ rnftools et2roc -h

usage: rnftools et2roc [-h] -i file -o file

todo

optional arguments:
  -h, --help           show this help message and exit
  -i file, --et file   Input ET file (evaluated read tuples, - for standard
                       input).
  -o file, --roc file  Output ROC file (evaluated reads, - for standard
                       output).
```

rnftools sam2roc

```
$ rnftools sam2roc -h

usage: rnftools sam2roc [-h] -i file -o file [-d int]

todo

optional arguments:
  -h, --help            show this help message and exit
  -i file, --sam file   SAM/BAM with aligned RNF reads(- for standard input).
  -o file, --roc file   Output ROC file (- for standard output).
  -d int, --allowed-delta int
                        Tolerance of difference of coordinates between true
                        (i.e., expected) alignment and real alignment (very
                        important parameter!) (default: 5).
```

2.4 Additional information

2.4.1 FAQ

Frequently asked questions.

Contents

A script which used to work does not work any more

Try to upgrade RNFtools to the latest version

```
pip install --upgrade rnftools
```

If the problem still appears, please send us an e-mail to karel.brinda@gmail.com.

2.4.2 Information for developers

Do you develop bioinformatics software? Here you will find how RNF and RNFtools can be useful for you .

Contents

Do you develop a read mapper?

RNFtools can help you to debug your mapper. You can:

- Find reads which were not aligned correctly (when tuning your algorithm).
- Test how successful your mapper is in dependence on parameters of simulation (error rate, etc.).
- Test if contamination reads are well detected and staying really unaligned.
- Test which impact have pre- and post-processing tools (such as read clustering tools or re-alignment tools) in combination with you mapper.

First you can start with some simple simulator (e.g., WgSim) for basic tests and later easily switch to more realistic simulations with ART of Mason.

Do you develop a read simulator?

Even though MIShmash currently supports several simulators, implicit support of RNF in simulators is preferable. Usually, simulators do not save as much information as RNF does, hence MIShmash must sometimes estimate some of these unknown values which brings noise into data. Direct support of RNF support in simulators would imply higher precision in the forthcoming analysis as well as better usability of your software.

Adding support for RNF into your simulator is a simple step since the format is easy to adopt. For existing software, we recommend to add an extra parameter switching internal naming procedure to RNF.

RNFtools can also help you to debug your simulator. Switching between your simulator and another one, you can check if obtained results are similar as they should be. If not, you might have bugs in your code (such as coordinates of simulation are incorrectly shifted by few positions, etc.).

Do you develop an evaluation tool for read mappers?

RNF enables you writing a universal evaluation tool, compatible with all RNF-compatible read simulators and all simulators supported by MIShmash.

2.4.3 Exhaustive list of read simulators

This webpage used to contain a list of existing read simulators. This list has been completed is now available in Section 2.4 (pp. 13-16) of the following thesis:

Publication

K. Břinda. **Novel computational techniques for mapping and classifying Next-Generation Sequencing data.** Université Paris-Est, France, 2016. [pdf]

Index

G

H

I

L

M

P

R

S

T

V

W