mirtop Documentation

Release 0.3.12a

Lorena Pantano, Thomas Desvignes, Karen Ellbeck, Ioannis Vlach

Feb 03, 2020

Contents:

1 Logo competition **1**

2 Installation **3**
- 2.1 bioconda . . . 3
- 2.2 pypi . . . 3
- 2.3 update to develop version from pip . . . 3
- 2.4 install develop version . . . 3

3 Quick Start **5**
- 3.1 Importer . . . 5
- 3.2 Operations . . . 6
- 3.3 Export . . . 7

4 Output **9**
- 4.1 GFF command . . . 9
- 4.2 Stats command . . . 9
- 4.3 Compare command . . . 9
- 4.4 Counts command . . . 10
- 4.5 Export command . . . 10

5 Structure of the code **11**

6 Examples of contributions **13**
- 6.1 How to add a new sub-command . . . 13
- 6.2 Add a unit test . . . 14
- 6.3 for the internal function . . . 14
- 6.4 for the sub-command . . . 14
- 6.5 test the unit . . . 15

7 Documentation for the Code **17**
- 7.1 bam . . . 17
- 7.2 exporter . . . 18
- 7.3 gff . . . 18
- 7.4 importer . . . 20
- 7.5 libs . . . 22
- 7.6 mirna . . . 23
- 7.7 classes . . . 28

Python Module Index **29**

Index **31**

CHAPTER 1

Logo competition

Looking for a logo, enter the competition here. Deadline 07/07/2018. Win a t-shirt and stickers if your logo is selected!

We got a logo: https://github.com/miRTop/mirtop/tree/master/artwork

CHAPTER 2

Installation

2.1 bioconda

```
conda install mirtop -c bioconda
```

2.2 pypi

```
pip install mirtop
```

2.3 update to develop version from pip

```
pip install --upgrade --no-deps git+https://github.com/miRTop/mirtop.git#egg=mirtop
```

2.4 install develop version

Thes best solution is to install conda to get an independent enviroment.

```
wget http://repo.continuum.io/miniconda/Miniconda-latest-Linux-x86_64.sh

bash Miniconda-latest-Linux-x86_64.sh -b -p ~/mirtop_env

export PATH=$PATH:~/mirtop_env

conda install -c bioconda bioconda bedtools samtools pip nose pysam pandas dateutil␣
↪pyyaml pybedtools biopython setuptools

git clone http://github.com/miRTop/mirtop
```

(continues on next page)

(continued from previous page)

```
cd mirtop
git fetch origin dev
git checkout dev

python setup.py develop
```

CHAPTER 3

Quick Start

3.1 Importer

3.1.1 From Bam files to GFF3

```
git clone mirtop
cd mirtop/data
```

You can use the example data. Here the reads have been mapped to the precursor sequences.

```
mirtop gff -sps hsa --hairpin examples/annotate/hairpin.fa --gtf examples/annotate/
↪hsa.gff3 -o test_out sim_isomir.bam
```

3.1.2 From `seqbuster`: :`miraligner` files to GFF3

miRNA annotation generated from miraligner tool:

```
mirtop gff --format seqbuster --sps hsa --hairpin examples/annotate/hairpin.fa --gtf␣
↪examples/annotate/hsa.gff3 -o test_out examples/seqbuster/reads.mirna
```

3.1.3 From `sRNAbench` files to GFF3

miRNA annotation generated from sRNAbench tool:

```
mirtop gff --format sranbench -sps hsa --hairpin examples/annotate/hairpin.fa --gtf␣
↪examples/annotate/hsa.gff3 -o test_out srnabench examples/srnabench
```

3.1.4 From PROST! files to GFF3

miRNA annotation generated from PROST! tool. Export isomiRs tab from excel file to a tabular text format file.

```
mirtop gff --format prost -sps hsa --hairpin  examples/annotate/hairpin.fa --gtf␣
↪examples/annotate/hsa.gff3 -o test_out examples/prost/prost.example.txt
```

3.1.5 From isomiR-SEA files to GFF3

miRNA annotation generated from isomiR-SEA tool.

```
mirtop validate examples/gff/correct_file.gff
```

3.2 Operations

3.2.1 Validator

To validate your mirGFF3 file and make sure if follows the current format:

```
mirtop gff --format isomirsea -sps hsa --hairpin  examples/annotate/hairpin.fa --gtf␣
↪examples/annotate/hsa.gff3 -o  test_out examples/isomir-sea/tagMir-all.gff
```

3.2.2 Get statistics from GFF

Get number of isomiRs and miRNAs annotated in the GFF file by isomiR category.

```
cd mirtop/data
mirtop stats -o test_out example/gff/correct_file.gff
```

3.2.3 Compare GFF file with reference

Compare the sequences from two or more GFF files. The first one will be used as the reference data.

```
cd mirtop/data
mirtop compare -o test_out example/gff/correct_file.gff example/gff/alternative.gff
```

3.2.4 Updates mirGFF3

Updates older versions with the most current one.

```
cd mirtop/data
mirtop update -o test_out_mirs examples/versions/version1.0.gff
```

3.3 Export

3.3.1 Export file to isomiRs format

To be compatible with isomiRs bioconductor package use:

```
cd mirtop/data
mirtop export -o test_out_mirs --hairpin examples/annotate/hairpin.fa --gtf examples/
↪annotate/hsa.gff3 examples/gff correct_file.gff
```

3.3.2 Export file to FASTA format

```
cd mirtop/data
mirtop export -o test_out_mirs --format fasta -d -vd --hairpin examples/annotate/
↪hairpin.fa --gtf examples/annotate/hsa.gff3 examples/gff/correct_file.gff
```

3.3.3 Export file to VCF format

```
cd mirtop/data
mirtop export -o test_out_mirs --format vcf --hairpin examples/annotate/hairpin.fa --
↪gtf examples/a
nnotate/hsa.gff3 examples/gff/correct_file.gff
```

3.3.4 Get count file

This file it is useful to load into R as a matrix. It contains the minimal information about each sequence and the count data in columns for each samples.

```
cd mirtop/data
mirtop counts -o test_out_mirs --hairpin examples/annotate/hairpin.fa --gtf examples/
↪annotate/hsa.gff3 examples/synthetic/let7a-5p.gtf
```

CHAPTER 4

Output

4.1 GFF command

The `mirtop gff` generates the GFF3 adapter format to capture miRNA variations. The output is explained here.

4.2 Stats command

The `mirtop stats` generates a table with different statistics for each type of isomiRs:

- total counts
- average counts
- total sequences

It generates as well a JSON file with the same information to be integrated easily with QC tools like MultiQC.

4.3 Compare command

The `mirtop compare` generates a tabular file with information about the difference and similarities. The first file in the command line will be considered the reference and the following files will be compared to the reference. Each line of the output has the following information for each file:

- sample
- idu
- seq
- tag: `E` if not in reference, `D` detected in both, `M` missing in target file
- same_mirna: if the sequence map to the same miRNA in the reference and target file

- one column for each isomiR type with the following tags: `FP` (variation not in reference), `TP` (variation in both), `FN` (variation not in target file)

4.4 Counts command

The `mirtop counts` generates a tabular file with the following columns:

- unique identifier
- read sequence
- miRNA name
- Variant attribute from GFF3 column
- One column for each isomiR type showing the exact variation
- One column for each sample with the counts for that sequence

4.5 Export command

The `mirtop export` generates different files from a mirGFF3 file:

- isomiRs compatible files
- FASTA files
- VCF files

CHAPTER 5

Structure of the code

- mirtop/bam
  - **bam.py**
    - `read_bam`: reads BAM files with pysamtools and store in a key - value object
  - **filter.py**
    - `tune`: if option `--clean` is on, filter according generic rules
    - `clean_hits`: get the top hits
- mirtop/gff
  - **init.py** wraps the convertion process to GFF3
  - **body.py** `create` will create the line according GFF format established.
    - `read_gff_line`: Inside a for loop to read line of the file. It'll return and structure key:value dictionary for each column.
  - **header.py** generate header and read header section.
  - **check.py** checks header and single lines to be valid according GFF format (NOT IMPLEMENTED)
  - **stats.py** GFF stats counting number of isomiR, their total and average expression
  - **query.py** accept SQlite queries after option -q ""
  - **convert.py**
    - `create_counts` table of counts
    - allow filtering by attribute
    - allow collapse by miRNA/isomiR type
  - **filter.py**, parse from query (NOT IMPLEMENTED)
- mirtop/mirna
  - **fasta.py**:

        * `read_precursor` fasta file: key - value
    - **realign.py**:
        * `hits`: class that defines hits
        * `isomir`: class that defines each sequence
        * `cigar_correction`: function that use CIGAR to make sequence to miRNA alignemt
        * `read_id` and `make_id`: shorter ID for sequences
        * `make_cigar`: giving an alignment return the CIGAR of it
        * `reverse_complement`: return the reverse complement of a sequence
        * `align`: uses biopython to align two sequences of the same size
        * `expand_cigar`: from a 12M to MMMMMMMMMMMM
        * `cigar2snp`: from CIGAR code to list of changes with position and reference and target nts
    - **mapper.py**:
        * `read_gtf` file: map genomic miRNA position to precursos position, then it needs genomic position for the miRNA and the precursor. Return would be like {mirna: [start, end]}
    - **annotate.py**:
        * `annotate`: read isomiRs and populate all attributes related to isomiRs
- mirtop/importer:
    - seqbuster.py
    - prost.py
    - srnabench.py
    - isomirsea.py
- mirtop/exporter:
    - isomirs.py: export file to match isomiRs BioC package.
- data/examples/
    - check gff files: example of correct, invalid, warning GFF files
    - check BAM file
    - check mapping from genome position to precursor position, example of +/- strand. Using `mirtop/mirna/map.read_gtf`.
    - check clean option: sequence mapping to multiple precursors/mirna, get the best score. Using `mirtop/bam/filter.clean_hits`.

To add new sub-commands, modify the following:

- mirtop/lib/parse.py
    - query: TODO
    - transform: TODO
    - create: TODO
    - check: TODO

CHAPTER 6

Examples of contributions

6.1 How to add a new sub-command

You need first to clone and install the tool in develop mode

Let's say that you want to add a new operation to `mirtop`, for instance, similar to the `stats` command to work with sGFF3 files. Assume a `test` function for this exmaple to just read the file and print `Hello GFF3.`

- Create the folder inside `mirtop/test`. The create to empty files named:
- `test.py`
- `__init__.py`
- Modify the `test.py` file with this content:

```
from mirtop.gff.body import read_gff_line

import mirtop.libs.logger as mylog
logger = mylog.getLogger(__name__)

def test(args):
        for fn in args.files:
                _test(fn)
                logger.info("Hello GFF3: %s" % fn)


def _test(fn):
        logger.debug("I am going to read this file: %s" % fn)
        for line in fn:
                read_gff_line(line)
```

- Choose a sub_command name, in this case: `test`.
- Add the arguments function at the end of this file: https://github.com/miRTop/mirtop/blob/dev/mirtop/libs/parse.py, using a naming following `add_subparser_test`.

```
def add_subparser_test(subparsers):
    parser = subparsers.add_parser("test", help="test function")
    parser.add_argument("files", nargs="*", help="GFF/GTF files.")
    parser = _add_debug_option(parser)
    return parser
```

- Add the function name to `parse_cl` function, at the end of the `sub_cmds` array.

```
    sub_cmds = {"gff": add_subparser_gff,
                "stats": add_subparser_stats,
                "compare": add_subparser_compare,
                "target": add_subparser_target,
                "simulator": add_subparser_simulator,
                "counts": add_subparser_counts,
                "export": add_subparser_export,
                "test": add_subparser_test
                }
```

- To get the function re-directed from the command line when you use the sub_cmd name, add a line to the `command_line.py` file, adding another `else` statement:

```
        elif "test" in kwargs:
        logger.info("Run test.")
        test(kwargs["args"])
```

- The function you use to link to the operation added need to be imported at the beginning. Let's say that the `test` function is at `mirtop/test/test.py`:

```
from mirtop.test import test
```

Try the new operation:

```
mirtop test data/examples/correct_file.gff
```

6.2 Add a unit test

6.3 for the internal function

Add to the end of `test/test_functions.py`, but inside `class FunctionsTest(unittest.TestCase):` this code:

```
    @attr(fn_test=True)
    def test_function_test(self):
        from mirtop import test
        test._test("data/examples/gff/correct_file.gff")
```

6.4 for the sub-command

Add to the end of `test/test_function.py`, but inside `class AutomatedAnalysisTest(unittest.TestCase):` this code:

```
    @attr(cmd_test=True)
    def test_srnaseq_annotation_bam(self):
        """Run test analysis
        """
        with make_workdir():
            clcode = ["mirtop",
                      "test",
                      "../../data/examples/gff/correct_file.gff"]
            print("")
            print(" ".join(clcode))
            subprocess.check_call(clcode)
```

6.5 test the unit

nose is needed: pip install nose

Run the function test from the top parent folder:

```
./run_test.sh fn_test
```

Run the command test from the top parent folder:

```
./run_test.sh cmd_test
```

CHAPTER 7

Documentation for the Code

7.1 bam

mirtop.bam.filter.**clean_hits**(*reads*)

Select only best matches from a list of hits from the same read.

Args: *reads*: dictionary as:

```
>>> {'read_id': mirtop.realign.hits, ...}
```

Returns:

reads: same than input but with best hits only.

mirtop.bam.filter.**tune**(*seq*, *precursor*, *start*, *cigar*)

The actual fn that will realign the sequence to find the nt changes at 5', 3' sequence and nt variations.

Args: *seq (str)*: sequence of the read.

precursor (str): sequence of the precursor.

start (int): start position of sequence on the precursor, +1.

cigar (str): similar to SAM CIGAR attribute.

Returns:

list with:

subs (list): substitutions

add (list): nt added to the end

cigar (str): updated cigar

7.2 exporter

Read GFF files and output isomiRs compatible format

mirtop.exporter.isomirs.**convert**(*args*)
Main function to convert from GFF3 to isomiRs Bioc Package.

Reads a GFF file to produces output file containing Expression counts

Args:

***args(namedtuple)*: arguments parsed from command line with** *mirtop.libs.parse.add_subparser_counts()*.

Returns:

***file (file)*: with columns like:** UID miRNA Variant Sample1 Sample2 ... Sample N

Read GFF files and output FASTA format

mirtop.exporter.fasta.**convert**(*args*)
Main function to convert from GFF3 to FASTA format.

Args:

***args*: supported options for this sub-command.** See *mirtop.libs.parse.add_subparser_export()*.

mirtop.exporter.vcf.**cigar_2_key**(*cigar*, *readseq*, *refseq*, *pos*, *var5p*, *var3p*, *parent_ini_pos*, *parent_end_pos*, *hairpin*)

Args: 'cigar(str)': CIGAR standard of a compressed alignment representation, this CIGAR omits the '1' integer. 'readseq(str)': the read sequence 'refseq(str)': the reference sequence 'pos(str)': the start current position 'var5p(int)': extra nucleotides not in the reference miRNA (5p strand) 'var3p(int)': extra nucleotides not in the reference miRNA (3p strand) 'parent_ini_pos(int)': the start position of the parent miRNA 'parent_end_pos(int)': the last position of the parent miRNA 'hairpin(str)': the string of the hairpin for all the miRNA

Returns: 'key_pos(str list)': a list with the positions of the variants. 'key_var(str list)': a list with the variant keys found. 'ref(str)': reference base(s). 'alt(str)': altered base(s).

mirtop.exporter.vcf.**convert**(*args*)
Main function to convert from GFF3 to VCF.

Args:

***args*: supported options for this sub-command.** See *mirtop.libs.parse.add_subparser_export()*.

mirtop.exporter.vcf.**create_vcf**(*mirgff3*, *precursor*, *gtf*, *vcffile*)

Args: 'mirgff3(str)': File with mirGFF3 format that will be converted 'precursor(str)': Fasta format sequences of all miRNA hairpins 'gtf(str)': Genome coordinates 'vcffile': name of the file to be saved

Returns: Nothing is returned, instead, a VCF file is generated

7.3 gff

GFF reader and creator helpers

mirtop.gff.body.**create**(*reads*, *database*, *sample*, *args*, *quiet=False*)
Read https://github.com/miRTop/mirtop/issues/9

mirtop.gff.body.**lift_to_genome**(*line*, *mapper*)

Function to get a class of type feature from classgff.py and map the precursors coordinates to the genomic coordinates

Args: *line(str)*: string GFF line. *mapper(dict)*: dict with mirna-precursor-genomic coordinas from

mirna.mapper.read_gtf_to_mirna function.

Returns: *(line)*: string with GFF line with updated chr, star, end, strand

mirtop.gff.body.**paste_columns**(*line*, *sep=' '*)
Create GFF/GTF line from read_gff_line

mirtop.gff.body.**read**(*fn*, *args*)
Read GTF/GFF file and load into annotate, chrom counts, sample, line

mirtop.gff.body.**read_gff_line**(*line*)
Read GFF/GTF line and return dictionary with fields

mirtop.gff.body.**read_variant**(*attrb*, *sep=' '*)
Read string in variants attribute.

Args: *attrb(str)*: string in Variant attribute.

Returns:

***(gff_dict)*: dictionary with:**

```
>>> {'iso_3p': -3, ...}
```

mirtop.gff.body.**variant_with_nt**(*line*, *precursors*, *matures*)
Return nucleotides changes for each variant type using Variant attribute, precursor sequences and mature position.

Compare multiple GFF files to a reference

mirtop.gff.compare.**compare**(*args*)
From a list of GFF files produce comparison with a reference set.

Args:

***args(namedtuple)*: arguments parsed from command line with** *mirtop.libs.parse.add_subparser_compare()*. First file will be considered the reference set.

Returns: *(out_file)*: comparison of the GFF files with the reference.

mirtop.gff.compare.**read_reference**(*fn*)
Read GFF into UID:Variant

Args: *fn (str)*: GFF file.

Returns: *srna (dict)*: dict with >>> {'UID': 'iso_snp:-2,...'}

Helpers to define the header fo the GFF file

mirtop.gff.header.**create**(*samples*, *database*, *custom*, *filter=None*)
Create header for GFF file.

Args: *samples (list)*: character list with names for samples

database (str): name of the database.

custom (str): extra lines.

filter (list): character list with filter definition.

Returns: *header (str)*: header string.

mirtop.gff.header.**read_samples**(*fn*)
Read samples from the header of a GFF file.

Args: *fn(str)*: GFF file to read.

Returns: *(list)*: character list with sample names.

mirtop.gff.header.**read_version**(*fn*)
Extract mirGFF3 version

mirtop.gff.merge.**merge**(*dts*, *samples*)
For dictionary with sample as keys and values as lines merge them into one GFF file.

Args: *dts(dict)*: dictionary as >>> {'file': {'mirna': {start: gff_list}}}. gff_list has the format as defined in *mirtop.gff.body.read()*.

samples(list): character list with sample names.

Returns: *merged_lines (nested dicts)*:gff_list has the format as defined in *mirtop.gff.body.read()*.

Produce stats from GFF3 format

mirtop.gff.stats.**stats**(*args*)
From a list of GFF files produce general isomiRs stats.

Args:

***args (namedtupled)*: arguments parsed from command line with** *mirtop.libs.parse.add_subparser_stats()*.

Returns: *(stdout) or (out_file)*: GFF general stats.

Update gff3 files to newest version

mirtop.gff.update.**convert**(*args*)
Update previous GFF3 versions.

Args:

***args (namedtupled)*: arguments parsed from command line with** *mirtop.libs.parse.add_subparser_update()*.

Returns: *(out_file)*: most updated GFF3 file.

mirtop.gff.update.**update_file**(*gff_file*, *new_gff_file*)
Update file from file version to current version

mirtop.gff.validator.**check_multiple**(*args*)
Check GFF3 format.

Args:

***args (namedtupled)*: arguments parsed from command line with** *mirtop.libs.parse.add_subparser_validator()*.

Returns: *(std_out)*: warnings or errors of the files showing issues with the format.

7.4 importer

Read isomiR GFF files

mirtop.importer.isomirsea.**cigar2variants**(*cigar*, *sequence*, *tag*)
From cigar to Variants in GFF format

mirtop.importer.isomirsea.**header**(*fn*)
Custom header for isomiR-SEA importer.

Args: *fn (str)*: file name with isomiR-SEA GFF output

Returns: *(str)*: isomiR-SEA header string.

mirtop.importer.isomirsea.**read_file**(*fn*, *args*)
Read isomiR-SEA file and convert to mirtop GFF format.

Args: *fn(str)*: file name with isomiR-SEA output information.

database(str): database name.

***args(namedtuple)*: arguments from command line.** See *mirtop.libs.parse.add_subparser_gff()*.

Returns:

***reads (nested dicts)*:gff_list has the format as** defined in *mirtop.gff.body.read()*.

Read prost! files

mirtop.importer.prost.**header**()
Custom header for PROST! importer.

Returns: *(str)*: PROST! header string.

mirtop.importer.prost.**read_file**(*fn*, *hairpins*, *database*, *mirna_gtf*)
Read PROST! file and convert to mirtop GFF format.

Args: *fn(str)*: file name with PROST output information.

database(str): database name.

***args(namedtuple)*: arguments from command line.** See *mirtop.libs.parse.add_subparser_gff()*.

Returns: *reads*: dictionary where keys are read_id and values are *mirtop.realign.hits*

Read seqbuster files

mirtop.importer.seqbuster.**header**()
Custom header for seqbuster importer.

Returns: *(str)*: seqbuster header string.

mirtop.importer.seqbuster.**read_file**(*fn*, *args*)
Read seqbuster file and convert to mirtop GFF format.

Args: *fn(str)*: file name with seqbuster output information.

database(str): database name.

***args(namedtuple)*: arguments from command line.** See *mirtop.libs.parse.add_subparser_gff()*.

Returns: *reads*: dictionary where keys are read_id and values are *mirtop.realign.hits*

Read sRNAbench files

mirtop.importer.srnabench.**read_file**(*folder*, *args*)
Read sRNAbench file and convert to mirtop GFF format.

Args: *fn(str)*: file name with sRNAbench output information.

database(str): database name.

***args(namedtuple)*: arguments from command line.** See *mirtop.libs.parse.add_subparser_gff()*.

Returns:

***reads (nested dicts)*:gff_list has the format as** defined in *mirtop.gff.body.read()*.

Read isomiR GFF files from optimir tool

mirtop.importer.optimir.**read_file**(*fn*, *args*)
Read OptimiR file and convert to mirtop GFF format.

Args: *fn(str)*: file name with isomiR-SEA output information.

database(str): database name.

***args(namedtuple)*: arguments from command line.** See *mirtop.libs.parse.add_subparser_gff()*.

Returns:

***reads (nested dicts)*:gff_list has the format as** defined in *mirtop.gff.body.read()*.

Read Manatee files

mirtop.importer.manatee.**read_file**(*fn*, *database*, *args*)
Read Manatee file and convert to mirtop GFF format.

Args: *fn(str)*: file name with Manatee output information.

database(str): database name.

***args(namedtuple)*: arguments from command line.** See *mirtop.libs.parse.add_subparser_gff()*.

Returns:

***reads (nested dicts)*:gff_list has the format as** defined in *mirtop.gff.body.read()*.

7.5 libs

Centralize running of external commands, providing logging and tracking. Integrated from bcbio package with some changes.

mirtop.libs.do.**find_bash**()
Find bash full path

mirtop.libs.do.**find_cmd**(*cmd*)
Find comand in session

mirtop.libs.do.**run**(*cmd*, *data=None*, *checks=None*, *region=None*, *log_error=True*, *log_stdout=False*)
Run the provided command, logging details and checking for errors.

Helpers to work with fastq files

mirtop.libs.fastq.**is_fastq**(*in_file*)

Check whether file is fastq accepting txt, fq and fastq extensions understanding compression with gzip: .gzip and .gz (copy from bcbio)

Args: *in_file(str)*: file name.

Returns: *(boolean)*: Yes or Not.

mirtop.libs.fastq.**open_fastq**(*in_file*)

open a fastq file, using gzip if it is gzipped (from bcbio package)

Args: *in_file(str)*: file name.

Returns: *(File)*: file handler.

mirtop.libs.fastq.**splitext_plus**(*fn*)

Split on file extensions, allowing for zipped extensions. (copy from bcbio)

Args: *fn(str)*: file name.

Returns: *base, ext(str, str)*: basename and extesion.

mirtop.libs.parse.**parse_cl**(*in_args*)
Function to parse the subcommands arguments.

utils from http://www.github.com/chapmanb/bcbio-nextgen.git

mirtop.libs.utils.**chdir**(**args*, ***kwds*)
Change dir temporaly using *with*:

```
>>> with chdir(temporal):
        do_something()
```

mirtop.libs.utils.**file_exists**(*fname*)
Check if a file exists and is non-empty.

mirtop.libs.utils.**safe_dirs**(*dirs*)
Create folder if not exitsts

mirtop.libs.utils.**safe_remove**(*fn*)
Remove file skipping

7.6 mirna

Read bam files

mirtop.mirna.annotate.**annotate**(*reads*, *mature_ref*, *precursors*, *quiet=False*)
Using coordinates, mismatches and realign to annotate isomiRs

Args:

***reads(dicts of hits)*:** dict object that comes from *mirotp.bam.bam.read_bam()*

***mirbase_ref (dict of mirna positions)*:** dict object that comers from *mirtop.mirna.read_mature()*

***precursors dict object (key : fasta)*:** that comes from *mirtop.mirna.fasta.read_precursor()*

***quiet(boolean)*:** verbosity state

Return:

***reads (dict)*:** dictionary where keys are read_id and values are *mirtop.realign.hits*

Read precursor fasta file

mirtop.mirna.fasta.**read_precursor**(*precursor*, *sps=None*)
Load precursor file for that species

Args: *precursor(str)*: file name with fasta sequences

***sps(str)*: if any, select species to keep.** It'll do a *header_sequence.find(sps)*.

Returns:

***hairpin(dict)*: keys are precursor names and** values are precursor sequences.

Read database information

mirtop.mirna.mapper.**get_primary_transcript**(*database*)

Get the ID to identify the primary transcript in the GTF file with the miRNA and precursor coordinates to be able to parse BAM files with genomic coordinates.

mirtop.mirna.mapper.**guess_database**(*args*)
Guess database name from GFF file.

Args:

***gtf(str)*: file name with GFF miRNA genomic positions and** header lines.

Returns: *database(str)*: name of the database

TODO: this needs to be generic to other databases.

mirtop.mirna.mapper.**read_gtf_chr2mirna**(*gtf*)
Load GTF file with precursor positions on genome.

Args:

***gtf(str)*: file name with GFF miRNA genomic positions and** header lines.

Returns:

***db_mir(dict)*: dictionary with keys being chr and values** mirna and genomic positions.

mirtop.mirna.mapper.**read_gtf_to_mirna**(*gtf*)
Load GTF file with precursor positions on genome.

Args:

***gtf(str)*: file name with GFF miRNA genomic positions and** header lines.

Returns:

***db_mir(dict)*: dictionary with keys being mirnas and values** genomic positions.

mirtop.mirna.mapper.**read_gtf_to_precursor**(*gtf*)
Load GTF file with precursor positions on genome Return dict with key being precursor name and value a dict of mature miRNA with relative position to precursor.

Args:

***gtf(str)*: file name with GFF miRNA genomic positions and** header lines.

Returns: *map_dict(dict)*:

```
>>> {'parent': {mirna: [start, end]}}
```

mirtop.mirna.mapper.**read_gtf_to_precursor_mirbase**(*gtf*, *format='precursor'*)
Load GTF file with precursor positions on genome Return dict with key being precursor name and value a dict of mature miRNA with relative position to precursor. For miRBase and similar GFF3 files.

Args:

***gtf(str)*: file name with GFF miRNA genomic positions and** header lines.

Returns: *map_dict(dict)*:

```
>>> {'parent': {mirna: [start, end]}}
```

mirtop.mirna.mapper.**read_gtf_to_precursor_mirgenedb**(*gtf*, *format='precursor'*)
Load GTF file with precursor positions on genome Return dict with key being precursor name and value a dict of mature miRNA with relative position to precursor. For MirGeneDB and similar GFF3 files.

Args:

***gtf(str)*: file name with GFF miRNA genomic positions and** header lines.

Returns: *map_dict(dict)*:

```
>>> {'parent': {mirna: [start, end]}}
```

mirtop.mirna.realign.**align**(*x*, *y*, *local=False*)

Pairwise alignments between two sequenes. https://medium.com/towards-data-science/pairwise-sequence-alignment-using-biopython-d1a9d0ba861f

Args: *x(str)*: short sequence.

y(str): long sequence.

local(boolean): local or global alignment.

Returns: *aligned_x(hit)*: alignment information, socre and positions.

mirtop.mirna.realign.**align_from_variants**(*sequence*, *mature*, *variants*)

Giving the sequence read, the mature from get_mature_sequence, and the variant GFF annotation: get a list of substitutions

Args: *sequence(str)*: read sequence.

***mature(str)*: mature sequence from** *mirtop.mirna.realing.get_mature_sequence()*.

variants(str): string from Variant attribute in GFF file.

Returns: *snp(list)*: [[pos, target, reference]]

mirtop.mirna.realign.**cigar2snp**(*cigar*, *reference*)

From a CIGAR string and reference sequence detect mistmatches positions and reference and target nucleotides.

Args: *cigar(str)*: CIGAR string.

reference(str): reference sequence.

Returns: *snp(list)*: position of mismatches (indels included) as:

```
>>> [pos, seq_nt, ref_nt]
```

mirtop.mirna.realign.**cigar_correction**(*cigarLine*, *query*, *target*)

Read from CIGAR in BAM file to define mismatches.

Args: *cirgarLine(str)*: CIGAR string from BAM file.

query(str): read sequence.

target(str): target sequence.

Returns: *(list)*: [query_nts, target_nts]

mirtop.mirna.realign.**expand_cigar**(*cigar*)

From short CIGAR version to long CIGAR version where each character is each nts in the sequence.

Args: *cigar(str)*: CIGAR string.

```
>>> 10MA3M
```

Returns: *cigar_long(str)*: CIGAR long.

```
>>> MMMMMMMMMMAMMM
```

mirtop.mirna.realign.**get_mature_sequence**(*precursor*, *mature*, *exact=False*, *nt=5*)

From precursor and mature positions get mature sequence with +/- 4 flanking nts.

Args: *precursor(str)*: long sequence.

mature(list): [start, end].

exact(boolean): not add 4+/- flanking nts.

nt(int): number of nts to get.

Returns: *(str)*: mature sequence.

class mirtop.mirna.realign.**hits**
"Class with alignment information.

mirtop.mirna.realign.**is_sequence**(*seq*)
This function check whether the sequence is valid or not.

Args: *seq(str)*: string acting as a sequence.

Returns: *boolean*: whether is or not a valid nucleotide sequence.

class mirtop.mirna.realign.**isomir**
Class to represent isomiRs information.

format(*sep='\t'*)
Create tabular line from variant fields.

formatGFF()
Create Variant attribute.

format_id(*sep='\t'*)
Create simple identifier from variant fields.

get_score(*sc*)
Get score from variant fields.

is_iso()
Define whether element is isomiR or not.

set_pos(*start*, *l*, *strand='+'*)
Set end position

mirtop.mirna.realign.**make_cigar**(*seq*, *mature*)
Function that will create CIGAR string from aligment between read and reference sequence.

Args: *seq(str)*: read sequence.

mature(str): short sequence.

Return: *short(str)*: CIGAR string.

mirtop.mirna.realign.**make_id**(*seq*)
Create a unique identifier for the sequence from the nucleotides, replacing 5 nts for a unique sequence.

It uses the code from *mirtop.mirna.keys()*.

Inspired by MINTplate: https://cm.jefferson.edu/MINTbase https://github.com/TJU-CMC-Org/MINTmap/tree/master/MINTplates

Args: *seq(str)*: nucleotides sequences.

Returns: *idName(str)*: unique identifier for the sequence.

mirtop.mirna.realign.**read_id**(*idu*)
Read a unique identifier for the sequence and convert it to the nucleotides, replacing an unique code for 5 nts.

It uses the code from *mirtop.mirna.keys()*.

Inspired by MINTplate: https://cm.jefferson.edu/MINTbase https://github.com/TJU-CMC-Org/MINTmap/tree/master/MINTplates

Args: *idu(str)*: unique identifier for the sequence.

Returns: *seq(str)*: nucleotides sequences.

mirtop.mirna.realign.**reverse_complement**(*seq*)

Get reverse complement of a sequences

Args: *seq(str)*: sequence.

```
>>> GCAT
```

Returns: *(str)*: reverse complemente sequence:

```
>>> ATGC
```

mirtop.mirna.realign.**variant_to_3p**(*hairpin*, *pos*, *variant*)

From a sequence and a start position get the nts +/- indicated by iso_3p. Pos option is 0-base-index

Args:

***hairpin(str)*: long sequence:**

```
>>> AAATTTT
```

position(int): >>> 3

***variant(int)*: number of nts involved in the variant:**

```
>>> -1
```

Returns:

***(str)*: nucleotide involved in the variant:**

```
>>> A
```

mirtop.mirna.realign.**variant_to_5p**(*hairpin*, *pos*, *variant*)

From a sequence and a start position get the nts +/- indicated by iso_5p. Pos option is 0-base-index

Args:

***hairpin(str)*: long sequence:**

```
>>> AAATTTT
```

position(int): >>> 3

***variant(int)*: number of nts involved in the variant:**

```
>>> -1
```

Returns:

***(str)*: nucleotide involved in the variant:**

```
>>> T
```

mirtop.mirna.realign.**variant_to_add**(*read*, *variant*)

From a sequence and a start position get the nts +/- indicated by iso_3p. Pos option is 0-base-index

Args:

***hairpin(str)*: long sequence:**

```
>>> AAATTTT
```

position(int): >>> 3

***variant(int)*: number of nts involved in the variant:**

```
>>> 2
```

Returns:

***(str)*: nucleotide involved in the variant:**

```
>>> TT
```

mirtop.mirna.snps.**create_vcf**(*isomirs*, *matures*, *gtf*, *vcf_file=None*)
Create vcf file of changes for all samples. PASS will be ones with > 3 isomiRs supporting the position and > 30% of reads, otherwise LOW

mirtop.mirna.snps.**liftover**(*pass_pos*, *matures*)
Make position at precursor scale

mirtop.mirna.snps.**liftover_to_genome**(*pass_pos*, *gtf*)
Liftover from precursor to genome

mirtop.mirna.snps.**print_vcf**(*data*)
Print vcf line following rules.

7.7 classes

class mirtop.mirna.realign.**hits**
"Class with alignment information.

class mirtop.mirna.realign.**isomir**
Class to represent isomiRs information.

format(*sep='\t'*)
Create tabular line from variant fields.

formatGFF()
Create Variant attribute.

format_id(*sep='\t'*)
Create simple identifier from variant fields.

get_score(*sc*)
Get score from variant fields.

is_iso()
Define whether element is isomiR or not.

set_pos(*start*, *l*, *strand='+'*)
Set end position

Python Module Index

m

mirtop, 17
mirtop.bam.filter, 17
mirtop.exporter.fasta, 18
mirtop.exporter.isomirs, 18
mirtop.exporter.vcf, 18
mirtop.gff.body, 18
mirtop.gff.compare, 19
mirtop.gff.header, 19
mirtop.gff.merge, 20
mirtop.gff.stats, 20
mirtop.gff.update, 20
mirtop.gff.validator, 20
mirtop.importer.isomirsea, 20
mirtop.importer.manatee, 22
mirtop.importer.optimir, 21
mirtop.importer.prost, 21
mirtop.importer.seqbuster, 21
mirtop.importer.srnabench, 21
mirtop.libs.do, 22
mirtop.libs.fastq, 22
mirtop.libs.parse, 23
mirtop.libs.utils, 23
mirtop.mirna.annotate, 23
mirtop.mirna.fasta, 23
mirtop.mirna.keys, 23
mirtop.mirna.mapper, 23
mirtop.mirna.realign, 25
mirtop.mirna.snps, 28

Index

A

align() (*in module mirtop.mirna.realign*), 25
align_from_variants() (*in module mirtop.mirna.realign*), 25
annotate() (*in module mirtop.mirna.annotate*), 23

C

chdir() (*in module mirtop.libs.utils*), 23
check_multiple() (*in module mirtop.gff.validator*), 20
cigar2snp() (*in module mirtop.mirna.realign*), 25
cigar2variants() (*in module mirtop.importer.isomirsea*), 20
cigar_2_key() (*in module mirtop.exporter.vcf*), 18
cigar_correction() (*in module mirtop.mirna.realign*), 25
clean_hits() (*in module mirtop.bam.filter*), 17
compare() (*in module mirtop.gff.compare*), 19
convert() (*in module mirtop.exporter.fasta*), 18
convert() (*in module mirtop.exporter.isomirs*), 18
convert() (*in module mirtop.exporter.vcf*), 18
convert() (*in module mirtop.gff.update*), 20
create() (*in module mirtop.gff.body*), 18
create() (*in module mirtop.gff.header*), 19
create_vcf() (*in module mirtop.exporter.vcf*), 18
create_vcf() (*in module mirtop.mirna.snps*), 28

E

expand_cigar() (*in module mirtop.mirna.realign*), 25

F

file_exists() (*in module mirtop.libs.utils*), 23
find_bash() (*in module mirtop.libs.do*), 22
find_cmd() (*in module mirtop.libs.do*), 22
format() (*mirtop.mirna.realign.isomir method*), 26, 28
format_id() (*mirtop.mirna.realign.isomir method*), 26, 28
formatGFF() (*mirtop.mirna.realign.isomir method*), 26, 28

G

get_mature_sequence() (*in module mirtop.mirna.realign*), 25
get_primary_transcript() (*in module mirtop.mirna.mapper*), 23
get_score() (*mirtop.mirna.realign.isomir method*), 26, 28
guess_database() (*in module mirtop.mirna.mapper*), 23

H

header() (*in module mirtop.importer.isomirsea*), 20
header() (*in module mirtop.importer.prost*), 21
header() (*in module mirtop.importer.seqbuster*), 21
hits (*class in mirtop.mirna.realign*), 26, 28

I

is_fastq() (*in module mirtop.libs.fastq*), 22
is_iso() (*mirtop.mirna.realign.isomir method*), 26, 28
is_sequence() (*in module mirtop.mirna.realign*), 26
isomir (*class in mirtop.mirna.realign*), 26, 28

L

lift_to_genome() (*in module mirtop.gff.body*), 18
liftover() (*in module mirtop.mirna.snps*), 28
liftover_to_genome() (*in module mirtop.mirna.snps*), 28

M

make_cigar() (*in module mirtop.mirna.realign*), 26
make_id() (*in module mirtop.mirna.realign*), 26
merge() (*in module mirtop.gff.merge*), 20
mirtop (*module*), 17
mirtop.bam.filter (*module*), 17
mirtop.exporter.fasta (*module*), 18
mirtop.exporter.isomirs (*module*), 18

mirtop.exporter.vcf (*module*), 18
mirtop.gff.body (*module*), 18
mirtop.gff.compare (*module*), 19
mirtop.gff.header (*module*), 19
mirtop.gff.merge (*module*), 20
mirtop.gff.stats (*module*), 20
mirtop.gff.update (*module*), 20
mirtop.gff.validator (*module*), 20
mirtop.importer.isomirsea (*module*), 20
mirtop.importer.manatee (*module*), 22
mirtop.importer.optimir (*module*), 21
mirtop.importer.prost (*module*), 21
mirtop.importer.seqbuster (*module*), 21
mirtop.importer.srnabench (*module*), 21
mirtop.libs.do (*module*), 22
mirtop.libs.fastq (*module*), 22
mirtop.libs.parse (*module*), 23
mirtop.libs.utils (*module*), 23
mirtop.mirna.annotate (*module*), 23
mirtop.mirna.fasta (*module*), 23
mirtop.mirna.keys (*module*), 23
mirtop.mirna.mapper (*module*), 23
mirtop.mirna.realign (*module*), 25
mirtop.mirna.snps (*module*), 28

O

open_fastq() (*in module mirtop.libs.fastq*), 22

P

parse_cl() (*in module mirtop.libs.parse*), 23
paste_columns() (*in module mirtop.gff.body*), 19
print_vcf() (*in module mirtop.mirna.snps*), 28

R

read() (*in module mirtop.gff.body*), 19
read_file() (*in module mirtop.importer.isomirsea*), 21
read_file() (*in module mirtop.importer.manatee*), 22
read_file() (*in module mirtop.importer.optimir*), 21
read_file() (*in module mirtop.importer.prost*), 21
read_file() (*in module mirtop.importer.seqbuster*), 21
read_file() (*in module mirtop.importer.srnabench*), 21
read_gff_line() (*in module mirtop.gff.body*), 19
read_gtf_chr2mirna() (*in module mirtop.mirna.mapper*), 24
read_gtf_to_mirna() (*in module mirtop.mirna.mapper*), 24
read_gtf_to_precursor() (*in module mirtop.mirna.mapper*), 24
read_gtf_to_precursor_mirbase() (*in module mirtop.mirna.mapper*), 24
read_gtf_to_precursor_mirgenedb() (*in module mirtop.mirna.mapper*), 24
read_id() (*in module mirtop.mirna.realign*), 26
read_precursor() (*in module mirtop.mirna.fasta*), 23
read_reference() (*in module mirtop.gff.compare*), 19
read_samples() (*in module mirtop.gff.header*), 19
read_variant() (*in module mirtop.gff.body*), 19
read_version() (*in module mirtop.gff.header*), 20
reverse_complement() (*in module mirtop.mirna.realign*), 27
run() (*in module mirtop.libs.do*), 22

S

safe_dirs() (*in module mirtop.libs.utils*), 23
safe_remove() (*in module mirtop.libs.utils*), 23
set_pos() (*mirtop.mirna.realign.isomir method*), 26, 28
splitext_plus() (*in module mirtop.libs.fastq*), 22
stats() (*in module mirtop.gff.stats*), 20

T

tune() (*in module mirtop.bam.filter*), 17

U

update_file() (*in module mirtop.gff.update*), 20

V

variant_to_3p() (*in module mirtop.mirna.realign*), 27
variant_to_5p() (*in module mirtop.mirna.realign*), 27
variant_to_add() (*in module mirtop.mirna.realign*), 27
variant_with_nt() (*in module mirtop.gff.body*), 19