

---

# MitoHiFi

**Marcela Uliano-Silva**

**May 23, 2023**



CONTENTS:

1	src	1
1.1	alignContigs module	1
1.2	circularizationCheck module	1
1.3	cleanUpCWD module	2
1.4	compareGenesLists module	2
1.5	createCoveragePlot module	2
1.6	fetch module	3
1.7	fetch_mitos module	4
1.8	filterfasta module	4
1.9	findFrameShifts module	5
1.10	findMitoReference module	5
1.11	find_frameshifts_mitos module	6
1.12	fixContigHeaders module	6
1.13	fix_MitoFinder_headers module	6
1.14	fix_improper_gff module	6
1.15	getGenesList module	6
1.16	getMitoLength module	7
1.17	getReprContig module	7
1.18	get_depth module	8
1.19	get_mitos_stats module	8
1.20	gfa2fa module	8
1.21	gff_to_gbk module	8
1.22	make_genome module	8
1.23	mitohifi module	9
1.24	parallel_annotation module	9
1.25	parallel_annotation_mitos module	10
1.26	parse_blast module	11
1.27	plot_annotation module	11
1.28	plot_annotation_GFF module	12
1.29	plot_coverage module	12
1.30	plot_coverage_final_mito module	13
1.31	reverse_complement module	13
1.32	rotate_genbank module	13
1.33	rotation module	13
1.34	rotation_mitos module	14
2	Indices and tables	17
	Python Module Index	19



## 1.1 alignContigs module

Align multiple sequence FASTA files with MAFFT.

This script allows the user to concatenate multiple sequence files in FASTA format using the `concatenate_contigs()` function and to align the multifasta file with the `mafft_align()` function.

```
alignContigs.concatenate_contigs(contigs_list, out_file='all_mitogenomes.rotated.fa')  
  
alignContigs.mafft_align(multifasta_file, threads='1', out_file='all_mitogenomes.rotated.aligned.aln',  
                          clustal_format=False)
```

## 1.2 circularizationCheck module

This script circularizes a contig containing repeats in the ends.

Copyright (c) 2014 Alex Schomaker Bastos - LAMPADA/UFRJ

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

```
circularizationCheck.circularizationCheck(resultFile, contig_id, circularSize=220, circularOffset=40)
```

Check, with blast, if there is a match between the start and the end of a sequence. Returns a tuple with (True, start, end) or False, accordingly.

```
circularizationCheck.get_circo_mito(contig_id, circular_size, circular_offset)
```

It takes a contig ID and circularizes it, returning the circularization points.

### Parameters

- **contig\_id** (*str*) – ID from contig to be circularized

- **circular\_size** (*int*) – size to consider when checking for circularization
- **circular\_offset** (*int*) – offset from start and finish to consider when looking for circularization

**Returns**

circularization history for the input contig

**Return type**

list

## 1.3 cleanUpCWD module

This script cleans up the working directory after MitoHiFi finishes running.

During the cleanup process, intermediate files are either deleted or moved to the proper subdirectories in order to better organize the final files.

```
cleanUpCWD.clean_up_work_dir(contigs_list)
```

```
cleanUpCWD.main()
```

## 1.4 compareGenesLists module

This script allows the user to compare two list of genes and return the set of genes that are either i) shared by both lists; ii) present in only one list; iii) present only in the other list.

The comparison of the lists is done with the `compare_genes_dicts()` function. The `get_genes_counts()` function works to count the number of occurrence of each gene in a list, and the `get_clean_gene()` function returns the “clean” string representation of a gene’s name (one final clean representation is needed to keep the `compare_genes_dicts()` from considering the same gene written in two different ways as different genes -> e.g. *cytb* and *cob*).

```
compareGenesLists.compare_genes_dicts(genes1, genes2, alphabetically_sorted=False)
```

Takes two lists of genes and return genes shared and specific.

```
compareGenesLists.get_clean_gene(in_gene)
```

Takes a raw gene name and returns its clean format.

```
compareGenesLists.get_genes_counts(genes_list)
```

Takes a list of genes and returns a dictionary containing the name of the gene and the number of occurrences.

## 1.5 createCoveragePlot module

This script builds a plot (*coverage\_plot.png*) containing the mean coverage depth distribution for the final representative mitogenom and other assembled potential mito contigs.

```
createCoveragePlot.create_coverage_plot(mapped_contigs, winSize, repr_contig, is_final_mito=False)
```

```
createCoveragePlot.get_contigs_headers(in_fasta)
```

Takes a multifasta file and returns a dictionary with the `contigs_ids` as keys and the contigs headers as values

`createCoveragePlot.get_contigs_to_map()`

Iterates over *contigs\_stats.tsv* to list all potential contigs and returns a list with the filenames of their rotated FASTA files

`createCoveragePlot.map_final_mito(in_reads, threads=1, covMap=20)`

#### Parameters

- **in\_reads** (*list*) – reads file to be mapped against contigs
- **threads** (*int*) – number of threads to be used for computation
- **covMap** (*int*) – minimum mapping quality to filter reads when building final coverage plot

#### Returns

(str) Filename of the sorted mapping file in BAM format

`createCoveragePlot.map_potential_contigs(in_reads, contigs, threads=1, covMap=20)`

#### Parameters

- **in\_reads** (*list*) – reads file to be mapped against contigs
- **contigs** (*list*) – list of contigs FASTA files to concatenate and map the reads against
- **threads** (*int*) – number of threads to be used for computation
- **covMap** (*int*) – minimum mapping quality to filter reads when building final coverage plot

#### Returns

(str) Filename of the sorted mapping file in BAM format

`createCoveragePlot.merge_images(img_list, out_file)`

`createCoveragePlot.split_mapping_by_contig(all_contigs_mapping, contigs_headers, threads=1)`

Takes a mapping file with reads mapped to a multifasta file and creates individual mapping files for each contig from the *contigs\_headers* dictionary. *contigs\_headers* contains *contigs\_ids* as values and *contigs\_headers* as values

## 1.6 fetch module

This script allows the calculation of statistics over MitoHiFi execution.

The `get_num_seqs()` function is used to calculate the number of sequences in a FASTA file. The `get_ref_tRNA()` function allows the definition of the reference tRNA to be used to rotate the potential mito contigs.

`fetch.get_num_seqs(in_fasta)`

Gets the number of sequences in a FASTA file.

#### Parameters

**in\_fasta** (*str*) – input FASTA file

#### Returns

number of sequences in FASTA file

#### Return type

int

`fetch.get_ref_tRNA()`

Defines the reference tRNA to be used for rotating contigs.

**Returns**

reference tRNA

**Return type**

str

## 1.7 fetch\_mitos module

This script allows the calculation of statistics over MitoHiFi execution. It is analogous to the *fetch.py*, except the *fetch.py* is suitable to process files when *MitoFinder* was used for annotation (default) and *fetch\_mitos* is used when *MITOS* was chosen as the annotation tool.

The `get_num_seqs()` function is used to calculate the number of sequences in a FASTA file. The `get_ref_tRNA()` function allows the definition of the reference tRNA to be used to rotate the potential mito contigs.

`fetch_mitos.get_num_seqs(in_fasta)`

Gets the number of sequences in a FASTA file.

**Parameters**

**in\_fasta** (str) – input FASTA file

**Returns**

number of sequences in FASTA file

**Return type**

int

`fetch_mitos.get_ref_tRNA()`

Defines the reference tRNA to be used for rotating contigs.

**Returns**

reference tRNA

**Return type**

str

## 1.8 filterfasta module

This script allows filtering of FASTA files.

**License:**

Copyright 2014-2019 Felix Heeger This script is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version. This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details. You should have received a copy of the GNU General Public License along with this program. If not, see <<http://www.gnu.org/licenses/>>.

`filterfasta.filterFasta(inStream, outPath, minLength=None, idList=None, random=None, fastq=False, regex=False, neg=False, log=<_io.TextIOWrapper name='<stderr>' mode='w' encoding='utf-8'>)`



```
filterfasta.filterLengthIdList(inStream, outPath, format, minLength=None, idList=None, regex=False,
                               neg=False, log=<_io.TextIOWrapper name='<stderr>' mode='w'
                               encoding='utf-8'>)
```

```
filterfasta.sampleRandom(inStream, outPath, format, number, log)
```

## 1.9 findFrameShifts module

This script iterates over a multifasta sequence file and returns all proteins that contains stop codons in the middle of their sequences.

The `find_frameshifts()` function is used for the (default) annotation using *MitoFinder*, while `find_frameshifts_mitos()` is used when *MITOS* was selected as the annotation tool.

```
findFrameShifts.find_frameshifts(in_gb)
```

```
findFrameShifts.find_frameshifts_mitos(in_proteins_fasta)
```

Takes a multifasta protein file and returns information on proteins that contain frameshifts.

```
findFrameShifts.get_gb_stats(in_gb)
```

```
findFrameShifts.get_mitos_stats(in_gff, in_fasta)
```

Takes in a GFF file produced by MITOS and returns annotation statistics.

```
findFrameShifts.main()
```

## 1.10 findMitoReference module

This script finds mitogenomes from related species in public databases.

### License:

Copyright 2022 Ksenia Krasheninnikova This script is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version. This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details. You should have received a copy of the GNU General Public License along with this program. If not, see <<http://www.gnu.org/licenses/>>.

```
findMitoReference.find_full_mito(group, outfolder, length_threshold, considered,
                                  org_type='mitochondrion', n=1)
```

```
findMitoReference.get_lineage(species)
```

## 1.11 find\_frameshifts\_mitos module

This script iterates over a multifasta sequence file and returns all proteins that contains stop codons in the middle of their sequences.

`find_frameshifts_mitos.find_frameshifts_mitos(in_proteins_fasta)`

Takes a multifasta protein file and returns information on proteins that contain frameshifts.

## 1.12 fixContigHeaders module

This script fixes headers from contigs given as input to MitoHiFi in contigs mode (-c).

The fix removes/changes suffixes from the contigs IDs in case they conflict with suffixes that are given by MitoHiFi while the pipeline runs.

`fixContigHeaders.fix_headers(fasta_in, fasta_out)`

## 1.13 fix\_MitoFinder\_headers module

`fix_MitoFinder_headers.fix_header(in_gb, out_gb)`

## 1.14 fix\_improper\_gff module

This script checks and removes features that have start position greater than end position.

This is necessary because we have noticed that MITOS sometimes create features that follow this pattern, although it is not compatible with the GFF standard format.

`fix_improper_gff.fix_improper_gff(in_gff)`

Takes a GFF file and fixes all features that have start position greater than end position.

## 1.15 getGenesList module

This script returns a list of annotated genes.

It recognizes annotations in either genbank or GFF format.

`getGenesList.get_genes_list(in_annotation, format='genbank')`

Takes a GFF/Genbank file and returns a list of annotated genes.

## 1.16 getMitoLength module

This script calculates statistics for the related mitogenome given as input for MitoHiFi.

The `get_mito_length()` function calculates the total length of the input related mitogenome, while the `get_mito_genes()` function returns the number of genes in the related mitogenome.

`getMitoLength.get_mito_genes(mito_file)`

Returns the number of genes from a mitogenome

Keyword arguments: `mito_file` – the input mitogenome GENBANK file

`getMitoLength.get_mito_length(mito_file)`

Returns the length of a mitogenome

Keyword arguments: `mito_file` – the input mitogenome FASTA file

`getMitoLength.main()`

## 1.17 getReprContig module

This script chooses between all potential mito contigs the representative one that is going to be considered the final mitogenome.

The `get_repr_contig()` is the function originally referred to by the main `mitohifi.py` script. Over the process of choosing the final mitogenome, the `get_circularization_info()` function is called to check if the contig was circularized and the `get_repr_contig_info()` function calculates some other metrics for all potential contigs and does the actual choice of the final mitogenome, retrieving its ID and its related stats.

The

`getReprContig.get_circularization_info(seq_id)`

Retrieves information if contig was circularized

### Parameters

**seq\_id** (*str*) – identifier of the target sequence (contig)

### Returns

returns True if contig was circularized and False otherwise

### Return type

bool

`getReprContig.get_repr_contig(contigs_fasta, rel_mito_len, rel_mito_num_genes, threads='1', debug=False)`

Gets representative contig from a multifasta file

### Parameters

- **contigs\_fasta** (*str*) – file containing all sequences
- **threads** (*str*) – number of threads to be used when running CDHIT

### Returns

Representative contig ID, CDHIT cluster where the representative contig came from

### Return type

tuple

`getReprContig.get_repr_contig_info(cdhit_clstr_file, rel_mito_len, rel_mito_num_genes, rel_mito_perc=0.35, debug=False)`

## 1.18 get\_depth module

This script calculates the per-base sequencing coverage of a contig.

```
get_depth.get_depth(bam_file, genome_file, target_seq)
```

```
get_depth.get_windows_depth(windows_file, bam_file, target_seq)
```

## 1.19 get\_mitos\_stats module

```
get_mitos_stats.get_mitos_stats(in_gff, in_fasta)
```

Takes in a GFF file produced by MITOS and returns annotation statistics.

## 1.20 gfa2fa module

Converts genome from GFA to FASTA format.

```
gfa2fa.gfa2fa(gfa_input, fasta_output="")
```

## 1.21 gff\_to\_gbk module

Convert a GFF and associated FASTA file into GenBank format.

Usage:

```
gff_to_genbank.py <GFF annotation file> [<FASTA sequence file> <molecule type>]
```

<FASTA sequence file>: input sequences matching records in GFF. Optional if sequences are in the GFF

<molecule type>: type of molecule in the GFF file. Defaults to DNA, the most common case.

```
gff_to_gbk.main(gff_file, fasta_file=None, molecule_type='DNA')
```

## 1.22 make\_genome module

This script creates a genome file compatible with bedtools

The `make_genome_file()` function creates the genome file itself, while the `make_genome_windows()` function creates a file containing windows of a given size (`win_size`) for each sequence from the genome.

```
make_genome.make_genome_file(in_bam, target_seq)
```

```
make_genome.make_genome_windows(genome_file, win_size)
```

## 1.23 mitohifi module

This is the main MitoHiFi script.

`mitohifi.main()`

## 1.24 parallel\_annotation module

This script circularizes, annotates and rotates mito contigs using MitoFinder for annotation.

The `process_contig()` function does the circularization, i.e. removal of artifactual repeated sequences at both ends of the contig and the annotation, i.e. gene prediction. The `process_contig_02()` function rotates the contig, given a reference gene that will be set as the beginning of the sequence. This function also calculates statistics for the contig, which are saved to a file named `{contig_id}.individual.stats`.

In the context of the `mitohifi.py` script, both functions are usually run in parallel for each potential mito contig. The `process_contig_02()` function will only be called after `process_contig()` is run for all potential contigs.

`parallel_annotation.process_contig(threads_per_contig, circular_size, circular_offset, contigs, max_contig_size, rel_gbk, gen_code, contig_id)`

Circularize and annotate a contig.

### Parameters

- **threads\_per\_contig** (*int*) – number of threads to be used
- **circular\_size** (*int*) – size to consider when checking for circularization
- **circular\_offset** (*int*) – offset from start and finish to consider when looking for circularization
- **contigs** (*str*) – filename of contigs file (containing all contigs)
- **max\_contig\_size** (*int*) – maximum contig size allowed
- **rel\_gbk** (*str*) – filename of related mito genbank file
- **gen\_code** (*str*) – species genetic code
- **contig\_id** (*str*) – target contig ID

### Returns

None

`parallel_annotation.process_contig_02(ref_tRNA, threads_per_contig, circular_size, circular_offset, contigs, max_contig_size, rel_gbk, gen_code, contig_id)`

Rotate a contig related to a reference tRNA gene and calculate contig statistics.

### Parameters

- **ref\_tRNA** (*str*) – tRNA gene to be used as reference for rotation (contig starts at reference tRNA)
- **threads\_per\_contig** (*int*) – number of threads to be used
- **circular\_size** (*int*) – size to consider when checking for circularization
- **circular\_offset** (*int*) – offset from start and finish to consider when looking for circularization
- **contigs** (*str*) – filename of contigs file (containing all contigs)

- **max\_contig\_size** (*int*) – maximum contig size allowed
- **rel\_gbk** (*str*) – filename of related mito genbank file
- **gen\_code** (*str*) – species genetic code
- **contig\_id** (*str*) – target contig ID

**Returns**

None

## 1.25 parallel\_annotation\_mitos module

This script circularizes, annotates and rotates mito contigs using MITOS for annotation.

The `process_contig_mitos()` function does the circularization, i.e. removal of artifactual repeated sequences at both ends of the contig and the annotation, i.e. gene prediction. The `process_contig_02_mitos()` function rotates the contig, given a reference gene that will be set as the beginning of the sequence. This function also calculates statistics for the contig, which are saved to a file named `{contig_id}.individual.stats`.

In the context of the `mitohifi.py` script, both functions are usually run in parallel for each potential mito contig. The `process_contig_02_mitos()` function will only be called after `process_contig_mitos()` is run for all potential contigs.

```
parallel_annotation_mitos.process_contig_02_mitos(ref_tRNA, threads_per_contig, circular_size,  
                                                  circular_offset, contigs, max_contig_size, rel_gbk,  
                                                  gen_code, contig_id)
```

Rotate a contig related to a reference tRNA gene and calculate contig statistics.

**Parameters**

- **ref\_tRNA** (*str*) – tRNA gene to be used as reference for rotation (contig starts at reference tRNA)
- **threads\_per\_contig** (*int*) – number of threads to be used
- **circular\_size** (*int*) – size to consider when checking for circularization
- **circular\_offset** (*int*) – offset from start and finish to consider when looking for circularization
- **contigs** (*str*) – filename of contigs file (containing all contigs)
- **max\_contig\_size** (*int*) – maximum contig size allowed
- **rel\_gbk** (*str*) – filename of related mito genbank file
- **gen\_code** (*str*) – species genetic code
- **contig\_id** (*str*) – target contig ID

**Returns**

None

```
parallel_annotation_mitos.process_contig_mitos(threads_per_contig, circular_size, circular_offset,  
                                              contigs, max_contig_size, rel_gbk, gen_code,  
                                              refseq_db, contig_id)
```

Circularize and annotate a contig.

**Parameters**

- **threads\_per\_contig** (*int*) – number of threads to be used
- **circular\_size** (*int*) – size to consider when checking for circularization

- **circular\_offset** (*int*) – offset from start and finish to consider when looking for circularization
- **contigs** (*str*) – filename of contigs file (containing all contigs)
- **max\_contig\_size** (*int*) – maximum contig size allowed
- **rel\_gbk** (*str*) – filename of related mito genbank file
- **gen\_code** (*str*) – species genetic code
- **contig\_id** (*str*) – target contig ID
- **refseq\_db** (*str*) – refseq database to be used

**Returns**

None

## 1.26 parse\_blast module

This script parses the output of BLAST to find potential mito contigs.

**License:**

Copyright 2020 Marcela Uliano-Silva This script is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version. This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details. You should have received a copy of the GNU General Public License along with this program. If not, see <<http://www.gnu.org/licenses/>>.

`parse_blast.get_contigs_ids(blast_output)`

args: blast\_line is a blast output, that can be either parsed\_blast.txt or parsed\_blast\_all.txt

returns: The ID from each contig from blast\_output, i.e., the BLAST queries

`parse_blast.main()`

`parse_blast.parse_blast(query_perc=50, min_query_perc=80, max_query_len=5)`

## 1.27 plot\_annotation module

This script creates a plot with all the genes annotated in the mito contigs

The `plot_annotation()` function is called when the input contig was annotated with MitoFinder and the `plot_annotation_mitos()` function when the annotation was done with MITOS.

The `merge_images()` function concatenates the plots generated for all potential mito contigs into a single image.

**class** `plot_annotation.MyCustomTranslator(*args: Any, **kwargs: Any)`

Bases: `BiopythonTranslator`

`compute_filtered_features(features)`

**class** `plot_annotation.MyCustomTranslatorMitos(*args: Any, **kwargs: Any)`

Bases: `BiopythonTranslator`

`compute_feature_label(feature)`

```
compute_filtered_features(features)
plot_annotation.merge_images(img_list, out_file)
plot_annotation.plot_annotation(in_gb, out_gb=None)
plot_annotation.plot_annotation_mitos(in_gff, out_gff=None)
```

## 1.28 plot\_annotation\_GFF module

This script creates a PNG image containing all genes from an annotation file in GFF format.

```
class plot_annotation_GFF.MyCustomTranslator(*args: Any, **kwargs: Any)
    Bases: BiopythonTranslator
    compute_feature_label(feature)
    compute_filtered_features(features)
plot_annotation_GFF.merge_images(img_list, out_file)
plot_annotation_GFF.plot_annotation(in_gff, out_gff=None)
```

## 1.29 plot\_coverage module

This script supports the creation of an image showing the distribution of the sequencing depth over the final mito contigs.

The `make_genome_file()` function creates a genome file compatible with bedtools.

The `make_genome_windows()` function creates a file in bed format containing the coordinates of all sequence windows (size of window is given as input).

The `get_windows_depth()` function does the actual calculation of the mean depth for each sequence window, which are the values that will be used to create the depth image.

The `final_mitogenome_coverage()` function moves intermediate files created while building the depth image to a directory named `final_mitogenome_coverage/`. This is for better organization of output files.

The `plot_coverage()` function creates the actual depth image based on the mean depths calculated for each sequence window.

```
plot_coverage.get_windows_depth(windows_file, bam_file)
plot_coverage.main()
plot_coverage.make_genome_file(contig_id)
plot_coverage.make_genome_windows(genome_file, winSize)
plot_coverage.move_intermediate_files(files_list)
plot_coverage.plot_coverage(contig_id, depth_file, winSize, isFinalMito=False)
```



## 1.30 plot\_coverage\_final\_mito module

This script supports the creation of an image containing the depth distribution over the final mitogenome.

```
plot_coverage_final_mito.get_windows_depth(windows_file, bam_file)

plot_coverage_final_mito.main()

plot_coverage_final_mito.make_genome_file(contig_id)

plot_coverage_final_mito.make_genome_windows(genome_file, winSize)

plot_coverage_final_mito.move_intermediate_files(files_list)

plot_coverage_final_mito.plot_coverage(contig_id, depth_file, winSize, isFinalMito=False)
```

## 1.31 reverse\_complement module

This script creates the reverse complement of a sequence.

The `reverse_complement()` function is the default function to create the reverse complement. The `reverse_complement_mitos()` function is the one that should be used when dealing with MITOS annotations. The `reverse_complement_annotation()` function adjusts the annotation coordinates of features to match the ones from the reverse complemented sequence.

```
reverse_complement.reverse_complement(in_gb, out_gb)

reverse_complement.reverse_complement_annotation(mitogenome_annotation, mitogenome_fasta,
                                                    mitogenome_annotation_rc)

reverse_complement.reverse_complement_mitos(in_fasta, out_fasta)
```

## 1.32 rotate\_genbank module

This script updates the annotation coords of features to match the rotated version of the contig (i.e. where the reference gene is set as the start)

```
rotate_genbank.get_feat_info(feat)

rotate_genbank.rotate_genbank(in_gbk, ref_gene, out_gbk)
```

## 1.33 rotation module

This script rotates a contig setting a reference gene as the start.

```
rotation.annotate(workdir, path, ref_gb, contig_id, o_code, max_contig_size, threads)

    Annotate reverse complemented genome.
```

`rotation.get_phe_pos(path)`

Gets the position of the tRNA-Phe gene

**Parameters**

**path** (*str*) – path of input genbank file

**Returns**

Start position of tRNA-Phe, Strand

**Return type**

tuple

`rotation.get_trna_pos(path)`

Gets the position for each tRNA in an input file

**Parameters**

**path** (*str*) – input file to be processed

**Returns**

Positions for each tRNA gene found

**Return type**

dict

`rotation.make_rc(path, rc_path)`

Creates a reverse complement.

**Parameters**

- **path** (*str*) – input FASTA file
- **rc\_path** (*str*) – new FASTA file created (reverse complement of input)

**Returns**

None

`rotation.rotate(genome, start, contig_id)`

Creates new genome file (suffix `.mitogenome.rotated.fa`) after rotating it.

**Parameters**

- **genome** (*str*) – input genome file
- **start** (*int*) – position at which rotate the input genome
- **contig\_id** (*str*) – identifier representing the original contig

**Returns**

None

## 1.34 rotation\_mitos module

This script rotates a contig setting a reference gene as the start.

It is equivalent to the `rotation.py` script, but adjusted to deal with MITOS annotations.

`rotation_mitos.annotate(workdir, path, ref_gb, contig_id, o_code, max_contig_size, threads)`

Annotate reverse complemented genome.

`rotation_mitos.get_phe_pos(path)`

Gets the position of the tRNA-Phe gene

**Parameters**

**path** (*str*) – path of input genbank file

**Returns**

Start position of tRNA-Phe, Strand

**Return type**

tuple

`rotation_mitos.get_trna_pos(path)`

Gets the position for each tRNA in an input file

**Parameters**

**path** (*str*) – input file to be processed

**Returns**

Positions for each tRNA gene found

**Return type**

dict

`rotation_mitos.make_rc(path, rc_path)`

Creates a reverse complement.

**Parameters**

- **path** (*str*) – input FASTA file
- **rc\_path** (*str*) – new FASTA file created (reverse complement of input)

**Returns**

None

`rotation_mitos.rotate(genome, start, contig_id)`

Creates new genome file (suffix .mitogenome.rotated.fa) after rotating it.

**Parameters**

- **genome** (*str*) – input genome file
- **start** (*int*) – position at which rotate the input genome
- **contig\_id** (*str*) – identifier representing the original contig

**Returns**

None

`rotation_mitos.rotate_annotation(mitogenome_annotation, mitogenome_fasta, start, contig_id)`

Takes a GFF file and the reference gene position and rotates the annotation to have the reference gene at position 1.



## INDICES AND TABLES

- `genindex`
- `modindex`
- `search`



## PYTHON MODULE INDEX

### a

`alignContigs`, 1

### c

`circularizationCheck`, 1

`cleanUpCWD`, 2

`compareGenesLists`, 2

`createCoveragePlot`, 2

### f

`fetch`, 3

`fetch_mitos`, 4

`filterfasta`, 4

`find_frameshifts_mitos`, 6

`findFrameShifts`, 5

`findMitoReference`, 5

`fix_improper_gff`, 6

`fix_MitoFinder_headers`, 6

`fixContigHeaders`, 6

### g

`get_depth`, 8

`get_mitos_stats`, 8

`getGenesList`, 6

`getMitoLength`, 7

`getReprContig`, 7

`gfa2fa`, 8

`gff_to_gbk`, 8

### m

`make_genome`, 8

`mitohifi`, 9

### p

`parallel_annotation`, 9

`parallel_annotation_mitos`, 10

`parse_blast`, 11

`plot_annotation`, 11

`plot_annotation_GFF`, 12

`plot_coverage`, 12

`plot_coverage_final_mito`, 13

### r

`reverse_complement`, 13

`rotate_genbank`, 13

`rotation`, 13

`rotation_mitos`, 14





## A

alignContigs  
    module, 1  
annotate() (in module rotation), 13  
annotate() (in module rotation\_mitos), 14

## C

circularizationCheck  
    module, 1  
circularizationCheck() (in module circularizationCheck), 1  
clean\_up\_work\_dir() (in module cleanUpCWD), 2  
cleanUpCWD  
    module, 2  
compare\_genes\_dicts() (in module compareGenesLists), 2  
compareGenesLists  
    module, 2  
compute\_feature\_label()  
    (plot\_annotation.MyCustomTranslatorMitos  
    method), 11  
compute\_feature\_label()  
    (plot\_annotation\_GFF.MyCustomTranslator  
    method), 12  
compute\_filtered\_features()  
    (plot\_annotation.MyCustomTranslator  
    method), 11  
compute\_filtered\_features()  
    (plot\_annotation.MyCustomTranslatorMitos  
    method), 11  
compute\_filtered\_features()  
    (plot\_annotation\_GFF.MyCustomTranslator  
    method), 12  
concatenate\_contigs() (in module alignContigs), 1  
create\_coverage\_plot() (in module createCoveragePlot), 2  
createCoveragePlot  
    module, 2

## F

fetch  
    module, 3

fetch\_mitos  
    module, 4  
filterfasta  
    module, 4  
filterFasta() (in module filterfasta), 4  
filterLengthIdList() (in module filterfasta), 4  
find\_frameshifts() (in module findFrameShifts), 5  
find\_frameshifts\_mitos  
    module, 6  
find\_frameshifts\_mitos() (in module  
    find\_frameshifts\_mitos), 6  
find\_frameshifts\_mitos() (in module find-  
    FrameShifts), 5  
find\_full\_mito() (in module findMitoReference), 5  
findFrameShifts  
    module, 5  
findMitoReference  
    module, 5  
fix\_header() (in module fix\_MitoFinder\_headers), 6  
fix\_headers() (in module fixContigHeaders), 6  
fix\_improper\_gff  
    module, 6  
fix\_improper\_gff() (in module fix\_improper\_gff), 6  
fix\_MitoFinder\_headers  
    module, 6  
fixContigHeaders  
    module, 6

## G

get\_circo\_mito() (in module circularizationCheck), 1  
get\_circularization\_info() (in module ge-  
    tReprContig), 7  
get\_clean\_gene() (in module compareGenesLists), 2  
get\_contigs\_headers() (in module createCoverage-  
    Plot), 2  
get\_contigs\_ids() (in module parse\_blast), 11  
get\_contigs\_to\_map() (in module createCoverage-  
    Plot), 2  
get\_depth  
    module, 8  
get\_depth() (in module get\_depth), 8  
get\_feat\_info() (in module rotate\_genbank), 13

get\_gb\_stats() (in module findFrameShifts), 5  
 get\_genes\_counts() (in module compareGenesLists), 2  
 get\_genes\_list() (in module getGenesList), 6  
 get\_lineage() (in module findMitoReference), 5  
 get\_mito\_genes() (in module getMitoLength), 7  
 get\_mito\_length() (in module getMitoLength), 7  
 get\_mitos\_stats  
     module, 8  
 get\_mitos\_stats() (in module findFrameShifts), 5  
 get\_mitos\_stats() (in module get\_mitos\_stats), 8  
 get\_num\_seqs() (in module fetch), 3  
 get\_num\_seqs() (in module fetch\_mitos), 4  
 get\_phe\_pos() (in module rotation), 13  
 get\_phe\_pos() (in module rotation\_mitos), 14  
 get\_ref\_tRNA() (in module fetch), 3  
 get\_ref\_tRNA() (in module fetch\_mitos), 4  
 get\_repr\_contig() (in module getReprContig), 7  
 get\_repr\_contig\_info() (in module getReprContig), 7  
 get\_trna\_pos() (in module rotation), 14  
 get\_trna\_pos() (in module rotation\_mitos), 15  
 get\_windows\_depth() (in module get\_depth), 8  
 get\_windows\_depth() (in module plot\_coverage), 12  
 get\_windows\_depth() (in module plot\_coverage\_final\_mito), 13  
 getGenesList  
     module, 6  
 getMitoLength  
     module, 7  
 getReprContig  
     module, 7  
 gfa2fa  
     module, 8  
 gfa2fa() (in module gfa2fa), 8  
 gff\_to\_gbk  
     module, 8

## M

mafft\_align() (in module alignContigs), 1  
 main() (in module cleanUpCWD), 2  
 main() (in module findFrameShifts), 5  
 main() (in module getMitoLength), 7  
 main() (in module gff\_to\_gbk), 8  
 main() (in module mitohifi), 9  
 main() (in module parse\_blast), 11  
 main() (in module plot\_coverage), 12  
 main() (in module plot\_coverage\_final\_mito), 13  
 make\_genome  
     module, 8  
 make\_genome\_file() (in module make\_genome), 8  
 make\_genome\_file() (in module plot\_coverage), 12  
 make\_genome\_file() (in module plot\_coverage\_final\_mito), 13

make\_genome\_windows() (in module make\_genome), 8  
 make\_genome\_windows() (in module plot\_coverage), 12  
 make\_genome\_windows() (in module plot\_coverage\_final\_mito), 13  
 make\_rc() (in module rotation), 14  
 make\_rc() (in module rotation\_mitos), 15  
 map\_final\_mito() (in module createCoveragePlot), 3  
 map\_potential\_contigs() (in module createCoveragePlot), 3  
 merge\_images() (in module createCoveragePlot), 3  
 merge\_images() (in module plot\_annotation), 12  
 merge\_images() (in module plot\_annotation\_GFF), 12  
 mitohifi  
     module, 9  
 module  
     alignContigs, 1  
     circularizationCheck, 1  
     cleanUpCWD, 2  
     compareGenesLists, 2  
     createCoveragePlot, 2  
     fetch, 3  
     fetch\_mitos, 4  
     filterfasta, 4  
     find\_frameshifts\_mitos, 6  
     findFrameShifts, 5  
     findMitoReference, 5  
     fix\_improper\_gff, 6  
     fix\_MitoFinder\_headers, 6  
     fixContigHeaders, 6  
     get\_depth, 8  
     get\_mitos\_stats, 8  
     getGenesList, 6  
     getMitoLength, 7  
     getReprContig, 7  
     gfa2fa, 8  
     gff\_to\_gbk, 8  
     make\_genome, 8  
     mitohifi, 9  
     parallel\_annotation, 9  
     parallel\_annotation\_mitos, 10  
     parse\_blast, 11  
     plot\_annotation, 11  
     plot\_annotation\_GFF, 12  
     plot\_coverage, 12  
     plot\_coverage\_final\_mito, 13  
     reverse\_complement, 13  
     rotate\_genbank, 13  
     rotation, 13  
     rotation\_mitos, 14  
 move\_intermediate\_files() (in module plot\_coverage), 12  
 move\_intermediate\_files() (in module plot\_coverage\_final\_mito), 13  
 MyCustomTranslator (class in plot\_annotation), 11

MyCustomTranslator (class in *plot\_annotation\_GFF*),  
12

MyCustomTranslatorMitos (class in *plot\_annotation*),  
11

## P

parallel\_annotation  
module, 9

parallel\_annotation\_mitos  
module, 10

parse\_blast  
module, 11

parse\_blast() (in module *parse\_blast*), 11

plot\_annotation  
module, 11

plot\_annotation() (in module *plot\_annotation*), 12

plot\_annotation() (in module *plot\_annotation\_GFF*),  
12

plot\_annotation\_GFF  
module, 12

plot\_annotation\_mitos() (in module  
*plot\_annotation*), 12

plot\_coverage  
module, 12

plot\_coverage() (in module *plot\_coverage*), 12

plot\_coverage() (in module  
*plot\_coverage\_final\_mito*), 13

plot\_coverage\_final\_mito  
module, 13

process\_contig() (in module *parallel\_annotation*), 9

process\_contig\_02() (in module *parallel\_annotation*), 9

process\_contig\_02\_mitos() (in module *parallel\_annotation\_mitos*), 10

process\_contig\_mitos() (in module *parallel\_annotation\_mitos*), 10

## R

reverse\_complement  
module, 13

reverse\_complement() (in module  
*reverse\_complement*), 13

reverse\_complement\_annotation() (in module  
*reverse\_complement*), 13

reverse\_complement\_mitos() (in module  
*reverse\_complement*), 13

rotate() (in module *rotation*), 14

rotate() (in module *rotation\_mitos*), 15

rotate\_annotation() (in module *rotation\_mitos*), 15

rotate\_genbank  
module, 13

rotate\_genbank() (in module *rotate\_genbank*), 13

rotation  
module, 13

rotation\_mitos  
module, 14

## S

sampleRandom() (in module *filterfasta*), 5

split\_mapping\_by\_contig() (in module *createCoveragePlot*), 3