Contents

1 Getting Started 3
2 Usage 7
3 About 21
4 Related Projects 23
5 Index 25
Python Module Index 27
abstar is a core component of the ab[x] toolkit for antibody sequence analysis. abstar performs V(D)J germline gene assignment and primary sequence annotation and can readily scale from a single sequence to billions of sequences.
1.1 Overview

With technical breakthroughs in the throughput and read-length of next-generation sequencing platforms, antibody repertoire sequencing is becoming an increasingly important tool for detailed characterization of the immune response to infection and immunization. Accordingly, there is a need for open, scalable software for the genetic analysis of repertoire-scale antibody sequence data.

We built abstar to be a modular component of these analyses. abstar is engineered to be highly flexible, capable of processing a single sequence or billions of sequences and scaling from individual laptops to large clusters of cloud computing instances.

1.1.1 Workflows

In addition to V(D)J germline gene assignment and primary antibody sequence annotation, abstar contains utilities for sequence acquisition, pre-processing, and database import. abstar also exposes a high-level public API to many of the core functions, which allows users to easily construct custom analysis workflows using multiple abstar utilities as well as other third-party tools. To ease integration of abstar into currently existing antibody analysis pipelines based on IMGT, abstar can optionally produce output that mimics the IMGT-HighV/Quest Summary output file.

1.1.2 File formats

AbTools accepts standard FASTA or FASTQ files and produces, by default, JSON-formatted output. This output format allows us to build the output using data structures that match the way we process data programatically. JSON is also easily importable into NoSQL databases like MongoDB. We have found NoSQL databases to be very well suited for performing downstream analyses of antibody repertoire data, as the flexible schema allows for easy updating of sequence records with additional annotation information. Although additional data can be added to relational databases, querying this data often involves joining tables, which can require significant optimization for very large datasets.
1.1.3 Scalability

Cloud computing has dramatically changed the landscape of high-performance computing (HPC), and has allowed small academic labs to ‘rent’ access to computational resources that would have been previously far outside their budget. abstar is tightly integrated with abcloud, which provides tools for launching, configuring and managing clusters of compute instances on Amazon’s Elastic Compute Cloud (EC2). Using the Celery distributed task queue, jobs are distributed to worker nodes and processed in parallel.

In order to maintain compatibility with alternate cloud computing platforms with minimal effort, an abstar Docker image is also provided.

1.2 Install

The easiest way to install abstar locally (on OSX or Linux) is to use pip:

```
$ pip install abstar
```

If you don’t have pip, the Anaconda Python distribution contains pip along with a ton of useful scientific Python packages and is a great way to get started with Python.

abstar does not run natively on Windows, but Windows users can run abstar with Docker:

```
$ docker pull briney/abstar
$ docker run -it briney/abstar
```

Stable and development versions of abstar can also be downloaded from Github. You can manually install the latest development version of abstar with:

```
$ git clone https://github.com/briney/abstar
$ cd abstar/
$ python setup.py install
```

Note: If installing manually via setup.py and you don’t already have scikit-bio installed, you may get an error when setuptools attempts to install scikit-bio. This can be fixed by first installing scikit-bio with pip:

```
$ pip install scikit-bio
```

and then retrying the manual install of abstar. Starting with version 0.5, scikit-bio dropped support for Python 2.7, so install scikit-bio on Python 2.7 with:

```
$ pip install scikit-bio<=0.4.2
```

1.2.1 Requirements

- Python 2.7 or 3.5+
- abutils
- biopython
- celery
- pymongo
1.2.2 Optional dependencies

Several optional abstar components have additional dependencies:

- **abstar.preprocessing** requires FASTQC, cutadapt and sickle
- sequence merging requires PANDAseq
- downloading data from BaseSpace requires the BaseSpace Python SDK
- **batch_mongoimport** requires MongoDB

If using Docker, all of the optional dependencies are included.
2.1 Commandline Use

Running abstar from the command line is reasonably simple, even for users with minimal experience with command-line applications. In the most basic case, with a single input file of human antibody sequences:

```
$ abstar -i /path/to/mydata.fasta -t /path/to/temp/ -o /path/to/output/
```

abstar will process all sequences contained in `mydata.fasta` and the results will be written to `/path/to/output/mydata.json`. If either (or both) of `/path/to/temp/` or `/path/to/output/` don’t exist, they will be created.

If you have a directory of FASTA/Q-formatted files for abstar to process, you can pass a directory via `-i` and all files in the directory will be processed:

```
$ abstar -i /path/to/input/ -t /path/to/temp/ -o /path/to/output/
```

For input directories that contain paired FASTQ files that need to be merged prior to processing, passing the `-m` flag instructs abstar to merge paired files with PANDAseq:

```
$ abstar -i /path/to/input/ -t /path/to/temp/ -o /path/to/output/ -m
```

The merged reads will be deposited into a merged directory located in the parent directory of the input directory. By default, abstar will use PANDAseq’s `simple_bayesian` merging algorithm, although alternate merging algorithms can be selected with `--pandaseq-algo`.

For data generated with Illumina sequencers, abstar can directly interface with BaseSpace to download raw sequencing data. In order for abstar to connect to BaseSpace, you need BaseSpace access token. The easiest way to do this is to set up a BaseSpace developer account following these instructions. Once you have your credentials, you can generate a BaseSpace credentials file by running:

```
$ make_basespace_credfile
```

and following the instructions.
When downloading data from BaseSpace, you obviously don’t have an input directory of data for abstar to process (since that data hasn’t been downloaded yet). Instead of providing input, output and temp directories, you can just pass abstar a project directory using -p and abstar will create all of the necessary subdirectories within the project directory. Running abstar with the -b option indicates that input data should be downloaded from BaseSpace:

$ abstar -p /path/to/project_dir/ -b

A list of available BaseSpace projects will be displayed and you can select the appropriate project. If downloading data from BaseSpace, -m is assumed and paired-end reads will be merged.

abstar uses a human germline database by default, but germline databases are also provided for macaque, mouse and rabbit. To process macaque antibody sequences (from BaseSpace):

$ abstar -p /path/to/project_dir/ -b -s macaque

### 2.2 API Examples

abstar and abutils both expose a public API containing many of the core functions. This makes it reasonably straightforward to build custom pipelines that include several abstar/abutils components or integrate these tools with third-party tools. A few simple examples are shown below.

#### 2.2.1 Case #1

Sequencing data consists of an Illumina MiSeq run on human samples, with the raw data stored in BaseSpace (project ID: 123456789). Samples are indexed, so each sample will be downloaded from BaseSpace as a separate pair of read files. We’d like to do several things:

- get a FASTQC report on the raw data
- remove adapters
- quality trim
- get another FASTQC report on the cleaned data
- merge paired reads
- annotate with abstar

```python
import os
import abstar
from abstar.utils import basespace, pandaseq

PROJECT_DIR = '/path/to/project'
PROJECT_ID = '123456789'

# download data from BaseSpace
bs_dir = os.path.join(PROJECT_DIR, 'raw_data')
basespace.download(bs_dir, project_id=PROJECT_ID)

# FASTQC on the raw data
fastq1_dir = os.path.join(PROJECT_DIR, 'fastqc-pre')
abstar.fastqc(bs_dir, output=fastq1_dir)

# adapter trimming
```

(continues on next page)
adapter_dir = os.path.join(PROJECT_DIR, 'adapter_trimmed')
adapters = '/path/to/adapters.fasta'
abstar.adapter_trim(bs_dir, output=adapter_dir, adapter_both=adapters)

# quality trimming
quality_dir = os.path.join(PROJECT_DIR, 'quality_trimmed')
abstar.quality_trim(adapter_dir, output=quality_dir)

# FASTQC on the cleaned data
fastqc2_dir = os.path.join(PROJECT_DIR, 'fastqc-post')
abstar.fastqc(quality_dir, output=fastqc2_dir)

# read merging
merged_dir = os.path.join(PROJECT_DIR, 'merged')
pandaseq.run(quality_dir, merged_dir)

# run abstar
temp_dir = os.path.join(PROJECT_DIR, 'temp')
json_dir = os.path.join(PROJECT_DIR, 'json')
abstar.run(input=merged_dir,
           temp=temp_dir,
           output=json_dir)

2.2.2 Case #2

Sequencing data is a directory of single-read FASTQ files that have already been quality/adapter trimmed. We’d like to do the following:

- get a FASTQC report
- annotate with abstar
- import the JSONs into a MongoDB database named MyDatabase

Our FASTQ file names are formatted as: SampleNumber-SampleName.fastq, which means the abstar output file name would be SampleNumber-SampleName.json. We’d like the corresponding MongoDB collection to just be named SampleName.

import os
import abstar
from abstar.utils import mongoimport

PROJECT_DIR = '/path/to/project'
FASTQ_DIR = '/path/to/fastqs'

MONGO_IP = '123.45.67.89'
MONGO_PORT = 27017
MONGO_USER = 'MyUsername'
MONGO_PASS = 'Secr3t'

# FASTQC on the input data
fastqc_dir = os.path.join(PROJECT_DIR, 'fastqc')
abstar.fastqc(FASTQ_DIR, output=fastqc_dir)

# run abstar
temp_dir = os.path.join(PROJECT_DIR, 'temp')
json_dir = os.path.join(PROJECT_DIR, 'json')
abstar.run(input=FASTQ_DIR,
    temp=temp_dir,
    output=json_dir)

# import into MongoDB
mongoimport.run(ip=MONGO_IP,
    port=MONGO_PORT
    user=MONGO_USER,
    password=MONGO_PASS,
    input=json_dir,
    db='MyDatabase'
    delim1='-',
    delim2='.'
2.3 API Reference

2.3.1 abstar.abstar

abstar.core.abstar

abstar.core.abstar.run(*args, **kwargs)

Runs AbStar.

Input sequences can be provided in several different formats:

1. individual sequences as positional arguments: run(seq1, seq2, temp=temp, output=output)
2. a list of sequences, as an argument: run([seq1, seq2], temp=temp, output=output)
3. a single FASTA/Q-formatted input file, passed via input
4. a directory of FASTA/Q-formatted files, passed via input

When passing sequences (not FASTA/Q files), the sequences can be in any format recognized by abtools.sequence.Sequence, including:

- a raw nucleotide sequence, as a string (a random sequence ID will be assigned)
- a list/tuple of the format [sequence_id, sequence]
- a BioPython SeqRecord object
- an AbTools Sequence object

Either sequences, project_dir, or all of input, output and temp are required.

Examples

If processing a single sequence, you can pass the raw sequence, as a string:

```python
import abstar

result = abstar.run('ATGC')
```

or a list/tuple of the format [sequence_id, sequence]:

```python
result = abstar.run(['seq1', 'ATGC'])
```

If you pass just the raw sequence, a random sequence ID will be generated with uuid.uuid4(). In either case, when given a single sequence, abstar.run() will return a single AbTools Sequence object. If running multiple sequences, you can either pass each sequence as a positional argument:

```python
result_list = run(['seq1', 'ATGC'], ['seq2', 'CGTA'])
```
or you can pass a list of sequences as the first argument, in this case using sequences parsed from a FASTA file using Biopython:

```python
from Bio import SeqIO

fasta = open('my_sequences.fasta', 'r')
seqs = [s for s in SeqIO.parse(fasta, 'fasta')]
result_list = abstar.run(seqs)
```

When given multiple sequences, `abstar.run()` will return a list of AbTools Sequence objects, one per input sequence.

If you’d prefer not to parse the FASTQ/A file into a list (for example, if the input file is extremely large), you can pass the input file path directly, along with a temp directory and output directory:

```python
result_files = abstar.run(input='/path/to/my_sequences.fasta',
                          temp='/path/to/temp',
                          output='/path/to/output')
```

Given a file path, `abstar.run()` returns a list of output file paths. In the above case, `result_files` will be a list containing a single output file path: `/path/to/output/my_sequences.json`.

If you have a directory containing multiple FASTQ/A files, you can pass the directory path using `input`:

```python
result_files = abstar.run(input='/path/to/input',
                          temp='/path/to/temp',
                          output='/path/to/output')
```

As before, `result_files` will contain a list of output file paths.

If your input directory contains paired FASTQ files (gzip compressed or uncompressed) that need to be merged prior to processing with AbStar:

```python
result_files = abstar.run(input='/path/to/input',
                          temp='/path/to/temp',
                          output='/path/to/output',
                          merge=True)
```

The paired read files in `input` will be merged with PANDAseq prior to processing with AbStar. By default, PANDAseq’s ‘simple bayesian’ read merging algorithm is used, although alternate algorithms can be selected with `pandaseq_algo`.

AbStar also provides an alternate CSV-formatted output type that mimics the IMGT Summary file. This option is provided to minimize the effort needed to convert existing IMGT-based pipelines to AbStar. Alternate output is only available when passing an input file or directory; passing individual sequences or a list of sequences will always return Sequence objects. To produce IMGT-formatted output:

```python
result_files = abstar.run(input='/path/to/input',
                          temp='/path/to/temp',
                          output='/path/to/output',
                          output_type='imgt')
```

### Parameters

- **project_dir** *(str)* – Path to the project directory. Most useful when directly downloading files from BaseSpace, and all subdirectories will be created by AbStar.

- **input** *(str)* – Path to input directory, containing FASTA/Q files. If performing read merging with PANDAseq, paired FASTQ files may be gzip compressed.
• **output** *(str)* – Path to output directory.
• **temp** *(str)* – Path to temp directory, where intermediate job files will be stored.
• **log** *(str)* – Path to log file. If not provided and `project_dir` is provided, the log will be written to `/path/to/project_dir/abstar.log`. If output is provided, log will be written to `/path/to/output/abstar.log`.
• **isotype** *(bool)* – If True, the isotype will inferred by aligning the sequence region downstream of the J-gene. If False, the isotype will not be determined. Default is True.
• **uid** *(int)* – Length (in nucleotides) of the Unique Molecular ID used to barcode input RNA. A positive integer results in the UMID being parsed from the start of the read (or merged read), a negative integer results in parsing from the end of the read. Default is 0, which results in no UMID parsing.
• **gzip** *(bool)* – If True, compresses output files with gzip. Default is False.
• **pretty** *(bool)* – If True, formats JSON output files to be more human-readable. If False, JSON output files contain one record per line. Default is False.
• **output_type** *(str)* – Options are ‘json’ or ‘imgt’. IMGT output mimics the Summary table produced by IMGT High-V/Quest, to maintain a level of compatibility with existing IMGT-based pipelines. JSON output is much more detailed. Default is ‘json’.
• **merge** *(bool)* – If True, input must be paired-read FASTA files (gzip compressed or un-compressed) which will be merged with PANDAseq prior to processing with AbStar. If `basespace` is True, `merge` is automatically set to True. Default is False.
• **pandaseq_algo** *(str)* – Define merging algorithm to be used by PANDAseq. Options are ‘simple_bayesian’, ‘ea_util’, ‘flash’, ‘pear’, ‘rdp_mle’, ‘stitch’, or ‘uparse’. Default is ‘simple_bayesian’, which is the default PANDAseq algorithm.
• **debug** *(bool)* – If True, `abstar.run()` runs in single-threaded mode, the log is much more verbose, and temporary files are not removed. Default is False.

### Returns

If the input is a single sequence, `run` returns a single AbTools `Sequence` object.

If the input is a list of sequences, `run` returns a list of AbTools `Sequence` objects.

If the input is a file or a directory of files, `run` returns a list of output files.

### 2.3.2 abstar.preprocess

**abstar.preprocess**

**abstar.preprocess.quality_trim**(input_directory=None, output_directory=None, quality_cutoff=20, length_cutoff=50, quality_type=u'sanger', compress_output=True, file_pairs=None, singles_directory=None, nextseq=False, paired_reads=True, allow_Sprime_trimming=False, print_debug=False)

Performs quality trimming with sickle.

**Parameters**

...
• **input_directory** (*str*) – Path to a directory of files to be quality trimmed. If the directory contains paired reads, they should follow the Illumina MiSeq naming scheme. If you have paired reads that do not follow the MiSeq naming scheme, you can group the paired read files yourself and pass them to --file-pairs.

• **output_directory** (*str*) – Path to the output directory, into which quality trimmed read files will be deposited. If not provided, a directory will be created in the parent directory of input_directory. Required if using file_pairs instead of input_directory.

• **quality_cutoff** (*int*) – Quality score at which to truncate reads. Default is 20.

• **length_cutoff** (*int*) – Reads will be discarded if, after quality trimming, the length is shorter than this cutoff. Default is 50.

• **quality_type** (*str*) – Quality score type. Options are solexa, illumina, and sanger. illumina is equivalent to Casava 1.3-1.7 and sanger is Casava >= 1.8. Default is sanger.

• **compress_output** (*bool*) – If True, output files will be gzip compressed. Default is True.

• **file_pairs** (*list*) – If input files are paired-end reads that don’t follow Illumina’s MiSeq naming scheme, you can pass a list of lists/tuples, with each list/tuple containing a pair of read file paths.

• **singles_directory** (*str*) – Path to singles output directory. If processing paired reads and one read of the pair passes quality/length filters and the other doesn’t, the single passing read will be written to this file. Default is None, which results in the single sequences being discarded and not written to file.

• **nextseq** (*bool*) – Set to True if the sequencing data comes from a NextSeq run. The file naming scheme for NextSeq runs is different that MiSeq runs, and setting this option will allow NextSeq paired read files to be processed appropriately. Default is False.

• **paired_reads** (*bool*) – If True, reads will be processed as paired reads. If False, each read will be processed separately. It is not advisable to process paired reads with paired_reads set to False because if paired read files are processed separately and one read passes filters while the paired read doesn’t, this may cause problems with downstream processes (like read merging).

• **allow_5prime_trimming** (*bool*) – If True, quality trimming will be performed on the 5’ end of the reads as well as the 3’ end. Default is False.

**Returns** Path to the output directory

**Return type** str

```python
abstar.preprocess.adapter_trim(input_directory, output_directory=None, adapter_5prime=None, adapter_3prime=None, adapter_5prime_anchored=None, adapter_3prime_anchored=None, adapter_both=None, compress_output=True)
```

Trims adapters with cutadapt.

**Parameters**

• **input_directory** (*str*) – Path to a directory of FASTQ files to be adapter trimmed. Required.

• **output_directory** (*str*) – Path to the output directory. If not provided, a directory will be created in the parent directory of input_directory.
• `adapter_5prime (str)` – Path to a FASTA-formatted file of adapters to be trimmed from the 5’ end of reads.

• `adapter_3prime (str)` – Path to a FASTA-formatted file of adapters to be trimmed from the 3’ end of reads.

• `adapter_5prime_anchored (str)` – Path to a FASTA-formatted file of adapters to be trimmed from the 5’ end of reads. More strictly requires the read to be anchored to the 5’ end of the read than when using `adapter_5prime`.

• `adapter_3prime_anchored (str)` – Path to a FASTA-formatted file of adapters to be trimmed from the 3’ end of reads. More strictly requires the read to be anchored to the 3’ end of the read than when using `adapter_3prime`.

• `adapter_both (str)` – Path to a FASTA-formatted file of adapters that will be trimmed from either end of the reads.

• `compress_output (bool)` – If True, output files will be gzip compressed. Default is True.

Returns Path to the output directory

Return type str

```
abstar.preprocess.fastqc(input_directory, output_directory=None, threads=-1)
```

Performs FASTQC analysis on raw NGS data.

Parameters

• `input_directory (str)` – Path to the input directory, containing one or more FASTQ files (either gzip compressed or uncompressed).

• `output_directory (str)` – Path to the output directory, where the FASTQC results will be deposited. If not provided, a directory named ‘fastqc_reports’ will be created in the parent directory of `input_directory`

• `threads (int)` – Number of threads to be used (passed to the `-t` flag when running `fastqc`). Default is -1, which uses all cores.

Returns path to the output directory

Return type str

### 2.3.3 abstar.utils

**abstar.utils**

**abstar.utils.basespace**

```
abstar.utils.basespace.download(download_directory, project_id=None, project_name=None)
```

Downloads sequencing data from BaseSpace (Illumina’s cloud storage platform).

Before accessing BaseSpace through the AbStar API, you need to set up a credentials file:

1. You need a BaseSpace access token. The easiest way to do this is to set up a BaseSpace developer account following these instructions

2. Make a BaseSpace credentials file using your developer credentials:

```
$ make_basespace_credential
```

---

2.3. API Reference 15
and follow the instructions.

**Examples**

If you know the name of the project you’d like to download:

```python
from abstar.utils import basespace
basespace.download('/path/to/download_directory', project_name='MyProject')
```

If you know the ID of the project you’d like to download:

```python
basespace.download('/path/to/download_directory', project_id='ABC123')
```

If neither `project_id` nor `project_name` is provided, a list of your available BaseSpace projects will be provided and you can select a project from that list:

```python
basespace.download('/path/to/download_directory')
```

**Parameters**

- `download_directory (str)` – Directory into which the raw sequences files should be downloaded. If the directory does not exist, it will be created.
- `project_id (str)` – ID of the project to be downloaded.
- `project_name (str)` – Name of the project to be downloaded.

**Returns**
The number of sequence files downloaded.

**Return type**
`int`

### abstar.utils.mongoimport

**abstar.utils.mongoimport.run(**`**kwargs**`)

Imports one or more JSON files into a MongoDB database.

**Examples**

To import a single JSON file into MyDatabase on a local MongoDB database:

```python
from abstar.utils import mongoimport
mongoimport.run(input='/path/to/MySequences.json', db='MyDatabase')
```

This will result in a collection named ‘MySequences.json’ being created in MyDatabase on your local MongoDB instance (if it doesn’t already exist) and the data from MySequences.json being imported into that collection.

Doing the same thing, but with a remote MongoDB server running on port 27017:

```python
mongoimport.run(ip='123.45.67.89',
                user='my_username',
                password='Secr3t',
                input='/path/to/MySequences.json',
                db='MyDatabase')
```
But what if we want the collection name to be different than the file name? We can truncate the filename at the first occurrence of any given pattern with `delim1`:

```python
mongoimport.run(input='/path/to/MySequences.json, 
    db='MyDatabase',
    delim1='.')
```

In this case, the collection name is created by truncating the input file name at the first occurrence of `.`, so the collection name would be `MySequences`. We can also truncate the filename at the Nth occurrence of any given pattern by using `delim1` with `split1_pos`:

```python
mongoimport.run(input='/path/to/my_sequences_2016-01-01.json, 
    db='MyDatabase',
    delim1='-', 
    split1_pos=2)
```

which results in a collection name of `my_sequences`.

If we have more complex filenames, we can use `delim1` in combination with `delim2`. When `delim1` and `delim2` are used together, `delim1` becomes the pattern used to cut the filename on the left and `delim2` is used to cut the filename on the right. For example, if our filename is `plate-2_SampleName-01_redo.json` and we want the collection to be named `SampleName`, we would set `delim1` to `_` and `delim2` to `-`. We also need to specify that we want to cut at the second occurrence of `delim2`, which we can do with `split2_pos`:

```python
mongoimport.run(input='/path/to/plate-2_SampleName-01_redo.json, 
    db='MyDatabase',
    delim1='_', 
    delim2='-', 
    split2_pos=2)
```

Trimming filenames this way is nice, but it becomes much more useful if you’re importing more than one file at a time. `mongoimport.run()` will accept a list of file names, and will generate separate collection names for each input file:

```python
files = ['/path/to/A01-Sample01_2016-01-01', 
          '/path/to/A02-Sample02_2016-01-01', 
          '/path/to/A03-Sample03_2016-01-01']

mongoimport.run(input=files, 
    db='MyDatabase',
    delim1='-', 
    delim2='_')
```

The three input files will be imported into collections `Sample01`, `Sample02` and `Sample03`, respectively. Finally, you can pass the path to a directory containing one or more JSON files, and all the JSON files will be imported:

```python
mongoimport.run(input='/path/to/output/directory', 
    db='MyDatabase',
    delim1='-', 
    delim2='_')
```

### Parameters

- **input (str, list)** – Input is required and may be one of three things:
  1. A list/tuple of JSON file paths
2. A path to a single JSON file
3. A path to a directory containing one or more JSON files.
   • **ip** *(str)* – The IP address of the MongoDB server. Default is ‘localhost’.
   • **port** *(int)* – MongoDB port. Default is 27017.
   • **user** *(str)* – Username with which to connect to the MongoDB database. If either of user or password is not provided, mongoimport.run() will attempt to connect to the MongoDB database without authentication.
   • **password** *(str)* – Password with which to connect to the MongoDB database. If either of user or password is not provided, mongoimport.run() will attempt to connect to the MongoDB database without authentication.
   • **db** *(str)* – Name of the MongoDB database for import. Required.
   • **log** *(str)* – Path to a logfile. If not provided log information will be written to stdout.
   • **delim1** *(str)* – Pattern on which to split the input file to generate the collection name. Default is None, which results in the file name being used as the collection name.
   • **split1_pos** *(int)* – Occurance of delim1 on which to split the input file name. Default is 1.
   • **delim2** *(str)* – Second pattern on which to split the input file name to generate the collection name. Default is None, which results in only delim1 being used.
   • **split2_pos** *(int)* – Occurance of delim2 on which to split the input file name. Default is 1.

**abstar.utils.pandaseq**

**abstar.utils.pandaseq.run** *(input, output, algorithm='simple_bayesian', nextseq=False)*
Merge paired-end FASTQ files with PANDAseq.

**Examples**

To merge a directory of raw (gzip compressed) files from a MiSeq run:

```python
merged_files = run('/path/to/input', '/path/to/output')
```

Same as above, but using the Pear read merging algorithm:

```python
merged_files = run('/path/to/input', '/path/to/output', algorithm='pear')
```

To merge a list of file pairs:

```python
file_pairs = [(sample1_R1.fastq, sample1_R2.fastq),
             (sample2_R1.fastq.gz, sample2_R2.fastq.gz),
             (sample3_R1.fastq, sample3_R2.fastq)]
merged_files = run(file_pairs, '/path/to/output')
```

**Parameters**

- **input** *(str, list)* – Input can be one of three things:
  1. path to a directory of paired FASTQ files
2. a list of paired FASTQ files

3. a list of read pairs, with each read pair being a list/tuple containing paths to two paired read files

Regardless of what input type is provided, paired FASTQ files can be either gzip compressed or uncompressed.

When providing a list of files or a directory of files, it is assumed that all files follow Illumina naming conventions. If your file names aren’t Illumina-like, submit your files as a list of read pairs to ensure that the proper pairs of files are merged.

• **output (str)** – Path to an output directory, into which merged FASTQ files will be deposited. To determine the filename for the merged file, the R1 file (or the first file in the read pair) is split at the first occurrence of the ‘_’ character. Therefore, the read pair ['my-sequences_R1.fastq', 'my-sequences_R2.fastq'] would be merged into my-sequences.fasta.


• **nextseq (bool)** – Set to True if the sequencing data was generated on a NextSeq. Needed because the naming conventions for NextSeq output files differs from MiSeq output.

**Returns** a list of merged file paths

**Return type** list
3.1 License

The MIT License (MIT)

Copyright (c) 2016 Bryan Briney

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.

3.2 News
Related Projects
CHAPTER 5

Index

- modindex
- search
Index

A
abstar.core.abstar (module), 11
abstar.preprocess (module), 13
abstar.utils.basespace (module), 15
abstar.utils.mongoinport (module), 16
abstar.utils.pandaseq (module), 18
adapter_trim() (in module abstar.preprocess), 14

D
download() (in module abstar.utils.basespace), 15

F
fastqc() (in module abstar.preprocess), 15

Q
quality_trim() (in module abstar.preprocess), 13

R
run() (in module abstar.core.abstar), 11
run() (in module abstar.utils.mongoinport), 16
run() (in module abstar.utils.pandaseq), 18