screed Documentation

Release 1.0

Alex Nolley and Titus Brown

Apr 06, 2017

Contents

1 User Manual 3
- 1.1 Installation 3
- 1.2 Command-line Quick Start 3
- 1.3 Python Quick Start 4
- 1.4 File Formats As Understood By Screed 6
- 1.5 FASTA <-> FASTQ Conversion 7

2 screed examples 9
- 2.1 Basic Usage 9

3 The screed developer documentation 11
- 3.1 Writing Custom Sequence Parsers 11
- 3.2 Coding guidelines and code review checklist 13
- 3.3 Release Documentation 14

4 Release notes for past versions of screed 19
- 4.1 Release 1.0 19
- 4.2 Release v0.9 20
- 4.3 Release v0.8 21
- 4.4 Release v0.5 22

5 Known Issues 23
- 5.1 List of known issues 23

6 Contributor Code of Conduct 25

7 License 27

8 Indices and tables 29

Authors Alex Nolley, C. Titus Brown

Contact ctb@msu.edu

Contents:

CHAPTER 1

User Manual

Note: Some doctests are included in *screed examples*. The examples in this document are meant for human consumption only. They will not work in doctests!

screed parses FASTA and FASTQ files, generates databases, and lets you query these databases. Values such as sequence name, sequence description, sequence quality, and the sequence itself can be retrieved from these databases.

Installation

The following software packages are required to run screed:

- Python 2 (2.7) or Python 3 (3.3 or newer)
- pytest (only required to running tests)

Use pip to download, and install Screed and its dependencies:

```
pip install screed
```

To run the optional tests type:

```
python -m screed.tests
```

Command-line Quick Start

Creating a database

```
$ screed db <fasta/fastq file>
```

Dumping a database to a file

```
$ screed dump_fasta <screed database file> <fasta output>
$ screed dump_fastq <screed database file> <fastq output>
```

If no output file is provided, sequences are written to the terminal (stdout) by default.

Python Quick Start

Reading FASTA/FASTQ files

```
>>> import screed
>>> with screed.open(filename) as seqfile:
>>>     for read in seqfile:
...         print(read.name, read.sequence)
```

Here, `filename` can be a FASTA or FASTQ file, and can be uncompressed, gzip-compressed, or bzip2-compressed. screed natively supports FASTA and FASTQ databases creation. If your sequences are in a different format see the developer documentation on *Writing Custom Sequence Parsers*.

Creating a database

```
>>> import screed
>>> screed.make_db('screed/tests/test-data/test.fa')
```

This loads a FASTA file `screed/tests/test-data/test.fa` into a screed database named `screed/tests/test-data/test.fa_screed`. A couple of things to note:

- The screed database is independent of the text file from which it was derived, so moving, renaming or deleting `screed/tests/test-data/test.fa` will not affect the newly created database.
- The `make_db` function inferred the file type as FASTA automatically. The `read_fasta_sequences()` and `read_fastq_sequences()` functions are available if you'd prefer to be explicit.

  ```
  >>> screed.read_fasta_sequences('screed/tests/test-data/test.fasta')
  >>> screed.read_fastq_sequences('screed/tests/test-data/test.fastq')
  ```

Opening a database

The class `ScreedDB` is used to read screed databases, regardless of what file format they were derived from (FASTA/FASTQ/hava/etc.). One reader to rule them all!

From the Python prompt, import the ScreedDB class and load some databases:

```
>>> from screed import ScreedDB
>>> fadb = ScreedDB('screed/tests/test-data/test.fa')
>>> fqdb = ScreedDB('screed/tests/test-data/test.fastq')
```

Notice how you didn't need to write the '_screed' at the end of the file names? screed automatically adds that to the file name if you didn't.

Database dictionary interface

Since screed emulates a read-only dictionary interface, any methods that don't modify a dictionary are supported:

```
>>> fadb.keys()
>>> fqdb.keys()
```

Each record in the database contains 'fields' such as name and sequence information. If the database was derived from a FASTQ file, quality and optional annotation strings are included. Conversely, FASTA-derived databases have a description field.

To retrieve the names of records in the database:

```
>>> names = fadb.keys()
```

The size of the databases (number of sequence records) is easily found:

```
>>> len(fadb)
22
>>> len(fqdb)
125
```

Retrieving records from a database

A record is the standard container unit in screed. Each has *fields* that vary slightly depending on what kind of file the database was derived from. For instance, a FASTQ-derived screed database has an id, a name, a quality score and a sequence. A FASTA-derived screed database has an id, name, description and a sequence.

Retrieving entire records:

```
>>> records = [r for r in fadb.itervalues()]
```

Each record is a dictionary of fields. The names of fields are keys into this dictionary with the actual information as values. For example:

```
>>> record = fadb[fadb.keys()[0]]
>>> index = record['id']
>>> name = record['name']
>>> description = record['description']
>>> sequence = record['sequence']
```

What this does is retrieve the first record object in the screed database, then retrieve the index, name, description and sequence from the record object using standard dictionary key -> value pairs.

Retrieving partial sequences (slicing)

screed supports the concept of retrieving a *slice* or a subset of a sequence string. The motivation is speed: if you have a database entry with a very long sequence string but only want a small portion of the string, it is faster to retrieve only the portion than to retrieve the entire string and then perform standard Python string slicing.

By default, screed's FASTA database creator sets up the `sequence` column to support slicing. For example, if you have an entry with name `someSeq` which has a 10K long sequence, and you want a slice of the sequence spanning positions 4000 to 4080:

```
>>> seq = db['someSeq'].sequence
>>> slice = seq[4000:4080]
```

This is much faster than say:

```
>>> seq = str(db['someSeq'].sequence)
>>> slice = seq[4000:4080]
```

Because deep down, less information is being read off the disk. The :code‘str()‘ method above causes the entire sequence to be retrieved as a string. Then Python slicing is done on the string `seq` and the subset stored in `slice`.

Retrieving records *via* index

Sometimes you don’t care what the name of a sequence is; you’re only interested in its position in the database. In these cases, retrieval via index is the method you’ll want to use:

```
>>> record = fqdb.loadRecordByIndex(5)
```

An index is like an offset into the database. The order records were kept in the FASTA or FASTQ file determines the index in their resulting screed database. The first record in a sequence file will have an index of 0, the second, an index of 1 and so on.

File Formats As Understood By Screed

While the screed database remains non-specific to file formats, the included FASTA and FASTQ parsers expect specific formats. These parsers attempt to handle the most common attributes of sequence files, though they can not support all features.

FASTQ

The FASTQ parsing function is `read_fastq_sequences()` and is located in the screed module.

The first line in a record must begin with ‘@’ and is followed by a record identifier (a name). An optional annotations string may be included after a space on the same line.

The second line begins the sequence line(s) which may be line wrapped. screed defines no limit on the length of sequence lines and no length on how many sequence lines a record may contain.

After the sequence line(s) comes a ‘+’ character on a new line. Some FASTQ formats require the first line to be repeated after the ‘+’ character, but since this adds no new information to the record, `read_fastq_sequences()` will ignore this if it is included.

The quality line(s) is last. Like the sequence line(s) this may be line wrapped. `read_fastq_sequences()` will raise an exception if the quality and sequence strings are of unequal length. screed performs no checking for valid quality scores.

FASTA

The FASTA parsing function is read_fasta_sequences() and is also located in the screed module.

The first line in a record must begin with ‘>’ and is followed with the sequence’s name and an optional description. If the description is included, it is separated from the name with a space. Note that though the FASTA format doesn’t require named records, screed does. Without a unique name, screed can’t look up sequences by name.

The second line begins the line(s) of sequence. Like the FASTQ parser, `read_fasta_sequences()` allows any number of lines of any length.

FASTA <-> FASTQ Conversion

As an extra nicety, screed can convert FASTA files to FASTQ and back again.

FASTA to FASTQ

The function used for this process is called ‘ToFastq’ and is located in the screed module. It takes the path to a screed database as the first argument and a path to the desired FASTQ file as the second argument. There is also a shell interface if the screed module is in your PYTHONPATH:

```
$ python -m screed dump_fastq <path to fasta db> [ <converted fastq file> ]
```

The FASTA name attribute is directly dumped from the file. The sequence attribute is also dumped pretty much directly, but is line wrapped to 80 characters if it is longer.

Any description line in the FASTA database is stored as a FASTQ annotation string with no other interpretation done.

Finally, as there is no quality or quality score in a FASTA file, a default one is generated. The generation of the quality follows the Sanger FASTQ conventions. The score is 1 (ASCII: “”) meaning a probability of about 75% that the read is incorrect (1 in 4 chance). This PHRED quality score is calculated from the Sanger format: Q = -10log(p) where p is the probability of an incorrect read. Obviously this is a very rough way of providing a quality score and it is only intended to fill in the requirements of a FASTQ file. Any application needing a true measurement of the quality should not rely on this automatic conversion.

FASTQ to FASTA

The function used for this process is called ‘toFasta’ and is located in the screed module. It takes the path to a screed database as the first argument and a path to the desired FASTA file as the second argument. Like the ToFastq function before, there is a shell interface to ToFasta if the screed module is in your PYTHONPATH:

```
$ python -m screed dump_fasta <path to fastq db> [ <converted fasta file> ]
```

As above, the name and sequence attributes are directly dumped from the FASTQ database to the FASTA file with the sequence line wrapping to 80 characters.

If it exists, the FASTQ annotation tag is stored as the FASTA description tag. As there is no equivalent in FASTA, the FASTQ quality score is ignored.

CHAPTER 2

screed examples

Basic Usage

Load screed, index the database, and return a dictionary-like object:

```
>>> import screed
>>> db = screed.read_fasta_sequences('../screed/tests/test.fa')
```

Get the list of sequence names, sort alphabetically, and look at the first one:

```
>>> names = db.keys()
>>> names.sort()
>>> names[0]
u'ENSMICT00000000730'
```

Retrieve that record:

```
>>> r = db[names[0]]
>>> print r.keys()
[u'description', u'id', u'name', u'sequence']
```

Print out the internal ID number and the name:

```
>>> print r.id
13
>>> print r.name
ENSMICT00000000730
```

CHAPTER 3

The screed developer documentation

This section of the documentation is for people who are contributing (or would like to contribute to) the screed project codebase, either by contributing code or by helping improve the documentation.

Please note that this project is released with a *Contributor Code of Conduct*. By participating in the development of this project you agree to abide by its terms.

Contents:

Writing Custom Sequence Parsers

screed is built to be adaptable to new kinds of file sequence formats. Included with screed are parsers for handling FASTA and FASTQ sequence file types, though if you need screed to work with a new format, all you need to do is write a new parser.

Field Roles

Each field in a screed database is assigned a role. These roles describe what kind of information is stored in their field. Right now there are only 4 different roles in a screed database: the text role, the sliceable role, the indexed key role and the primary key role. All roles are defined in the file: screed/DBConstants.py

The text role (DBConstants._STANDARD_TEXT) is the role most fields in a database will have. This role tells screed that the associated field is storing standard textual data. Nothing special.

The sliceable role (DBConstants._SLICEABLE_TEXT) is a role that can be assigned to long sequence fields. screed's default FASTA parser defines the 'sequence' field with the sliceable role. When screed retrieves a field that has the sliceable role, it builds a special data structure that supports slicing into the text.

The indexed key role (DBConstants._INDEXED_TEXT_KEY) is associated with exactly one of the fields in a screed database. In screed's FASTA and FASTQ parsers, this role is fulfilled by the 'name' field. This field is required because it is the field screed tells sqlite to index when creating the database and it is the field used for name look-ups when querying a screed database.

The primary key role (DBConstants._PRIMARY_KEY_ROLE) is a role automatically associated with the 'id' field in each database. This field is always created with each screed database and always holds this role. You as a user of screed won't need to worry about this one.

General Parsing Function Format

create_db is the function central to the creation of screed databases. This function accepts a file path, a tuple of field names and roles, and an iterator function. The file path describes where the screed database should go, the tuple contains the names of fields and their associated roles and the iterator function yields records in a dictionary format.

This sub-section describes general steps for preparing and using screed with a custom sequence parser. Though they don't have to be, future sequence parsers should be located in the seqparse.py file for convenience. These steps will be described in the context of working from the Python shell.

First import the create_db function:

```
>>> from screed import create_db
```

The create_db class handles the formatting of screed databases and provides a simple interface for storing sequence data.

Next the database fields and roles must be specified. The fields tell screed the names and order of the data fields inside each record. For instance, lets say our new sequence has types 'name', 'bar', and 'baz', all text. The tuple will be:

```
>>> fields = (('name', DBConstants._INDEXED_TEXT_KEY),
              ('bar', DBConstants._STANDARD_TEXT),
              ('baz', DBConstants._STANDARD_TEXT))
```

Notice how 'name' is given the indexed key role and bar and baz are given text roles? If, for instance, you know 'baz' fields can be very long and you want to be able to retrieve slices of them, you could specify fields as:

```
>>> fields = (('name', DBConstants._INDEXED_TEXT_KEY),
              ('bar', DBConstants._STANDARD_TEXT),
              ('baz', DBConstants._SLICEABLE_TEXT))
```

All screed databases come with an 'id' field, which is a sequential numbering order starting at 0 for the first record, 1 for the second, and so on. The names and number of the other fields are arbitrary with one restriction: one and only one of the fields must fulfill the indexed key role.

Next, you need to setup an iterator function that will return records in a dictionary format. Have a look at the 'fastq_iter', 'fasta_iter', or 'hava_iter' functions in the screed/fastq.py, screed/fasta.py, and screed/hava.py files, respectively for examples on how to write one of these. If you don't know what an iterator function is, the documentation on the Python website gives a good description: http://docs.python.org/library/stdtypes.html#iterator-types.

Once the iterator function is written, it needs to be instantiated. In the context of the built-in parsing functions, this means opening a file and passing the file handle to the iterator function:

```
>>> seqfile = open('path_to_seq_file', 'rb')
>>> iter_instance = myiter(seqfile)
```

Assuming that your iterator function is called 'myiter', this sets up an instance of it ready to use with create_db.

Now the screed database is created with one command:

```
>>> create_db('path_to_screed_db', fields, iter_instance)
```

If you want the screed database saved at 'path_to_screed_db'. If instead you want the screed database created in the same directory and with a similar file name as the sequence file, its OK to do this:

```
>>> create_db('path_to_seq_file', fields, iter_instance)
```

create_db will just append '_screed' to the end of the file name and make a screed database at that file path so the original file won't be overwritten.

When you're done the sequence file should be closed:

```
>>> seqfile.close()
```

Using the Built-in Sequence Iterator Functions

This section shows how to use the 'fastq_iter' and 'fasta_iter' functions for returning records from a sequence file.

These functions both take a file handle as the only argument and then return a dictionary for each record in the file containing names of fields and associated data. These functions are primarily used in conjunction with the db_create() function, but they can be useful by themselves.

First, import the necessary module and open a text file containing sequences. For this example, the 'fastq_iter' function will be used:

```
>>> import screed.fastq
>>> seqfile = open('path_to_seqfile', 'rb')
```

Now, the 'fastq_iter' can be instantiated and iterated over:

```
>>> fq_instance = screed.fastq(seqfile)
>>> for record in fq_instance:
...     print record.name
```

That will print the name of every sequence in the file. If instead you want to accumulate the sequences:

```
>>> sequences = []
>>> for record in fq_instance:
...     sequences.append(record.sequence)
```

These iterators are the core of screed's sequence modularity. If there is a new sequence format you want screed to work with, all it needs is its own iterator.

Error checking in parsing methods

The existing FASTA/FASTQ parsing functions contain some error checking, such as making sure the file can be opened and checking correct data is being read. Though screed doesn't enforce this, it is strongly recommended to include error checking code in your parser. To remain non-specific to one file sequence type or another, the underlying screed library can't contain error checking code of this kind. If errors are not detected by the parsing function, they will be silently included into the database being built and could cause problems much later when trying to read from the database.

Coding guidelines and code review checklist

This document is for anyone who want to contribute code to the screed project, and describes our coding standards and code review checklist.

Coding standards

All plain-text files should have line widths of 80 characters or less unless that is not supported for the particular file format.

Vim user can set the indentation with:

```
set expandtab
set shiftwidth=4
set softtabstop=4
```

We are a pure Python project and PEP 8 is our standard. The `` `pep8` `` and `` `autopep8` `` Makefile targets are helpful.

Code and documentation must have its spelling checked. Vim users can run:

```
:setlocal spell spelllang=en_us
```

Use *]s* and *[s* to navigate between misspellings and *z=* to suggest a correctly spelled word. *zg* will add a word as a good word.

GNU *aspell* can also be used to check the spelling in a single file:

```
aspell check --mode $filename
```

Code Review

Please read 11 Best Practices for Peer Code Review.

See also Code reviews: the lab meeting for code and the PyCogent coding guidelines.

Checklist

Copy and paste the following into a pull request comment when it is ready for review:

```
- [ ] Is it mergeable?
- [ ] `make test` Did it pass the tests?
- [ ] `make clean diff-cover` If it introduces new functionality, is it tested?
- [ ] `make format diff_pylint_report doc` Is it well formatted?
- [ ] Is it documented in the `ChangeLog`?
  http://en.wikipedia.org/wiki/Changelog#Format
- [ ] Was a spellchecker run on the source code and documentation after
  changes were made?
```

Note that after you submit the comment you can check and uncheck the individual boxes on the formatted comment; no need to put x or y in the middle.

Release Documentation

Introduction

This is the release documentation for releasing a new version of screed. This document is meant for screed release managers. Michael R. Crusoe and C. Titus Brown have released screed in the past. Jake Fenton is the first to release screed using this checklist.

Getting Started

1. Start with a clean checkout:

```
cd `mktemp -d`
git clone git@github.com:dib-lab/screed.git
cd screed
```

2. Install/update versioneer:

```
pip install versioneer
versioneer install
```

If there is a new version of versioneer, follow the instruction to update it and fix the configuration if needed:

```
git diff
versioneer install
git commit -p -m "new version of versioneer.py"
# or abandon the changes
git checkout -- versioneer.py screed/_version.py screed/__init.py \
        MANIFEST.in
```

3. Review the git logs since the previous release and that ChangeLog reflects the major changes:

```
git log --minimal --patch \
        `git describe --tags --always --abbrev=0`..HEAD
```

4. Review the issue list for any existing bugs that won't be fixed in the release and ensure they're documented in `doc/known-issues.txt`

5. Verify that the build is clean: https://travis-ci.org/dib-lab/screed

6. Set the new version number and release candidate:

```
new_version=1.1
rc=rc3
```

Tag the release candidate with the new version prefixed by the letter 'v':

```
git tag v${new_version}-${rc}
git push --tags git@github.com:dib-lab/screed.git
```

7. Test the release candidate:

```
cd ..
virtualenv testenv1
virtualenv testenv2
virtualenv testenv3
virtualenv testenv4

# first we test the tag
cd testenv1
source bin/activate
git clone --depth 1 --branch v${new_version}-${rc} \
        https://github.com/dib-lab/screed.git
cd screed
make install-dependencies
make install
make test
```

```
python -c 'import screed; print(screed.__version__)' # double-check version number


# Test via pip
cd ../../testenv2
source bin/activate
pip install -e \
        git+https://github.com/dib-lab/screed.git@v${new_version}-${rc}#egg=screed
cd src/screed
make dist
make install
pip install pytest
pytest --pyargs screed -m 'not known_failing'
python -c 'import screed; print(screed.__version__)'
cp dist/screed*tar.gz ../../../testenv3

# test if the dist made in testenv2 is complete enough to build another
# functional dist

cd ../../../testenv3
source bin/activate
pip install pytest
pip install screed*tar.gz
pytest --pyargs screed -m 'not known_failing'
python -c 'import screed; print(screed.__version__)'
tar xzf screed*tar.gz
cd screed*
make dist
make test
```

8. Publish the new release on the testing PyPI server. You will need to change your PyPI credentials as documented here: https://wiki.python.org/moin/TestPyPI. You may need to re-register:

   ```
   python setup.py register --repository test
   ```

   Now, upload the new release:

   ```
   python setup.py sdist upload -r test
   ```

   Test the PyPI release in a new virtualenv:

   ```
   cd ../../testenv4
   source bin/activate
   pip install -U setuptools pip
   pip install pytest
   pip install -i https://testpypi.python.org/pypi --pre --no-clean screed
   pytest --pyargs screed -m 'not known_failing'
   python -c 'import screed; print(screed.__version__)'
   cd build/screed
   ./setup.py test
   ```

9. Do any final testing (acceptance tests, etc.) Note that the acceptance tests for screed are to run the khmer automated tests with the new version of screed installed and then to run the khmer acceptance tests.

10. Make sure any release notes are merged into doc/release-notes/. Release notes should be written in the *.md* format to satisfy the requirements for GitHub release notes. The *convert-release-notes* make target can be used to generate *.rst* files from the *.md* notes.

How to make a final release

When you have a thoroughly tested release candidate, cut a release like so:

1. Create the final tag and publish the new release on PyPI (requires an authorized account)

   ```
   cd ../../../screed
   git tag v${new_version}
   python setup.py register sdist upload
   ```

2. Delete the release candidate tag and push the tag updates to GitHub:

   ```
   git tag -d v${new_version}-${rc}
   git push git@github.com:dib-lab/screed.git
   git push --tags git@github.com:dib-lab/screed.git
   ```

3. Add the release on GitHub, using the tag you just pushed. Name it "Version X.Y.Z" and copy/paste in the release notes.

4. Update the Read the Docs to point to the new version. Visit https://readthedocs.org/builds/screed/ and 'Build Version: master' to pick up the new tag. Once that build has finished check the "Activate" box next to the new version at https://readthedocs.org/dashboard/screed/versions/ under "Choose Active Versions". Finally change the default version at https://readthedocs.org/dashboard/screed/advanced/ to the new version.

5. Delete any RC tags created:

   ```
   git tag -d ${new_version}-${rc}
   git push origin :refs/tags/${new_version}-${rc}
   ```

6. Tweet about the new release

7. Send email including the release notes to khmer@lists.idyll.org and khmer-announce@lists.idyll.org

Notes on this document

This is the procedure for cutting a new release of screed. It has been adapted from the release documentation for the khmer project, found at http://khmer.readthedocs.org/en/v1.1/release.html.

CHAPTER 4

Release notes for past versions of screed

Contents:

Release 1.0

We are pleased to announce the release of screed 1.0. Screed is a biological sequence parsing and storage/retrieval library for DNA and protein sequences. It's designed to be lightweight and easy to use from Python.

This version is the first with API compatibility guarantees, following the semantic versioning guidelines. Most changes are internal or API clarifications, but there is a new shell command for screed functions and an unified function for writing FAST{A,Q} records.

Documentation is available at http://screed.readthedocs.org/en/v1.0

New items of note:

- New shell commands for common screed operations:
  - `db` for database creation (`screed db <filename>`)
  - dumping FAST{A,Q} records from a db (`screed dump_fasta <db> <output>` and `screed dump_fastq <db> <output>`). #55 @luizirber
- Remove `\*_Writer` classes and unify record writing in the `write_fastx` function. #53 @standage
- We now use pytest as a test runner, codecov for code coverage, and a simplified changelog format. #50 #49 #59 @luizirber @standage

Other bugs fixed/issues closed:

- Fix reverse complement problems for Python 2.7. #47 @ctb
- Fix operator comparison. #48 @luizirber

- Update tests & constrain behavior for screed Records. #54 @ctb
- Allow sqlite3 import to fail. #56 @ctb
- Cleanup user docs and code. #62 #57 @standage
- Simplify use of 'open' internally. #65 @ctb

Known Issues

These are all pre-existing

- Screed does not support gzip file streaming. This is an issue with Python 2.x and will likely *not* be fixed in future releases. This is being tracked in dib-lab/khmer#700
- Screed is overly tolerant of spaces in fast{a,q} which is against spec. This is being tracked in dib-lab/khmer#108

Contributors

@luizirber *@standage @ctb *@betatim

* Indicates new contributors

Release v0.9

We are pleased to announce the release of Screed v0.9. Screed is a database engine capable of storing and retrieving short-read sequence data and is designed to be fast and adaptable to different sequence file formats.

This version of Screed features Python 3 syntax with compatibility with Python 2. Additional changes have broken backwards compatibility in several small ways in preparation for our 1.0 release and adoption of strict semantic versioning from there on out.

It is also the first release since our move to the University of Davis, California and also under our new name, the Lab for Data Intensive Biology.

Documentation is available at http://screed.readthedocs.org/en/v0.9/

New items of note:

- Now a primarily Python 3 codebase with Python 2 compatibility. https://github.com/dib-lab/screed/pull/41 @luizirber & @mr-c
- Tests now correctly run using temporary directories and the test data is now shipped allowing the tests to be run after installation. https://github.com/dib-lab/screed/pull/30 @bocajnotnef https://github.com/dib-lab/screed/pull/40 @mr-c
- The private method `screed/screedRecord._screed_record_dict()` has been renamed to `screed.screedRecord.Record()`. This is **not** a backwards compatible change. https://github.com/dib-lab/screed/pull/35 @sguermond
- `screed.open()` now accepts `-` as a synonym for STDIN and is now an (optional) context manager. It no longer defaults to parsing out a separate description from the name. The description field will br removed altogether from the next release. This is **not** a backwards compatible change. https://github.com/dib-lab/screed/pull/36 @anotherthomas https://github.com/dib-lab/screed/pull/39 https://github.com/dib-lab/screed/pull/41 @luizirber https://github.com/dib-lab/screed/pull/43 @ctb

- The FASTQ parser was improved and it no longer hangs in the presence of empty lines. https://github.com/dib-lab/screed/pull/38 @proteasome
- Screed records now slice correctly https://github.com/dib-lab/screed/pull/41 @wrightmhw @luizirber

Other bugs fixed/issues closed:

- Release notes are now a part of the documentation. https://github.com/dib-lab/screed/pull/33 @bocajnotnef
- A test was made more robust to prevent hangs. https://github.com/dib-lab/screed/pull/37 @anotherthomas

Known Issues

These are all pre-existing

- Screed does not support gzip file streaming. This is an issue with Python 2.x and will likely *not* be fixed in future releases. This is being tracked in ged-lab/khmer#700
- Screed is overly tolerant of spaces in fast{a,q} which is against spec. This is being tracked in ged-lab/khmer#108

Contributors

@luizirber @mr-c @bocajnotnef @ctb *@proteasome *@anotherthomas *@sguermond

* Indicates new contributors

Release v0.8

We are pleased to announce the release of Screed v0.8. Screed is a database engine capable of storing and retrieving short-read sequence data and is designed to be fast and adaptable to different sequence file formats.

This version of Screed contains developer documentation for contributing to the Screed project and a code of conduct for interacting with other contributors and project maintainers. Documentation is available at http://screed.readthedocs.org/en/v0.8/

New items of note:

This release successfully installs and passes its unit tests on Ubuntu 14.04 and the latest release of Mac OS X 10 “Yosemite”. It also passes the khmer acceptance tests as per the eelpond testing protocol.

This release of screed has renamed the ‘accuracy’ attribute of read records to ‘quality;’ this API change will need to be adopted by all users wanting to upgrade to this version. Unlike the khmer project, Screed is not currently under semantic versioning. It will be with the 1.0 release.

- Screed now has automatic compression detection via magic bit sniffing for gzip and bzip2 compressed files (from @mr-c in dib-lab/khmer#432)
- Screed now supports streaming of uncompressed FASTA and FASTQ formatted nucleotide sequence data. bzip2 compressed FASTA and FASTQ formatted nucleotide sequence data can also be streamed but not gzip compressed FASTA and FASTQ formatted nucleotide sequence data. (from @mr-c, see dib-lab/khmer#633)
- Screed now has a Changelog, developer documentation and a code of conduct (from @ctb, @mr-c, @bocajnotnef in dib-lab/khmer#625)

- Versions are now autogenerated using git tags via Versioneer (from @bocajnotnef in cadceb5)
- Documentation is now autogenerated using Doxygen (from @mr-c in d8ed05b)

Notable bugs fixed/issues closed:

- A khmer script was not accepting reads on the stdin dib-lab/khmer#633 by @mr-c
- screed returning the wrong version and breaking dev installs dib-lab/khmer#803 by @mr-c

Known Issues

These are all pre-existing

- Screed records cannot be sliced requiring un-Pythonic techniques to achieve the same behavior. This will be included in a future release. This is being tracked in dib-lab/khmer#768
- Screed self-tests do not use a temporary directory which causes tests run from package-based installs to fail. This is being tracked in dib-lab/khmer#748
- Screed does not support gzip file streaming. This is an issue with Python 2.x and will likely *not* be fixed in future releases. This is being tracked in dib-lab/khmer#700
- Screed is overly tolerant of spaces in fast{a,q} which is against spec. This is being tracked in dib-lab/khmer#108

Contributors

@bocajnotnef @mr-c @brtaylor92 @wrightmhw @kdmurray91 @luizirber @ctb

Release v0.5

We are proud to announce the release of screed v0.5. screed is a database engine capable of storing and retriving short-read sequence data. screed is designed to be fast and adaptable to different sequence file formats. This marks the first release of screed which we consider stable and complete.

Features:

- Read sequence data from FASTA/FASTQ files into screed databases
- Save screed databases back to FASTA/FASTQ files
- Lookup sequence data by index (offset) or name
- Native support for sequence substring slicing
- Convert between FASTA <-> FASTQ file formats

screed is written entirely in Python and uses the Sqlite database for backend storage. screed can be downloaded from the public git repository: http://github.com/acr/screed.git

screed is licensed under the BSD license which can be viewed in the doc/LICENSE.txt file.

CHAPTER 5

Known Issues

This document details the known issues in the current release of screed. All issues for screed are tracked at https://github.com/dib-lab/khmer/labels/screed

List of known issues

Screed does not support gzip file streaming. This is an issue with Python 2.x and will likely *not* be fixed in future releases. https://github.com/dib-lab/khmer/issues/700

Screed is overly tolerant of spaces in fast{q,a} which is against spec. https://github.com/dib-lab/khmer/issues/108

CHAPTER 6

Contributor Code of Conduct

As contributors and maintainers of this project, we pledge to respect all people who contribute through reporting issues, posting feature requests, updating documentation, submitting pull requests or patches, and other activities.

We are committed to making participation in this project a harassment-free experience for everyone, regardless of level of experience, gender, gender identity and expression, sexual orientation, disability, personal appearance, body size, race, age, or religion.

Examples of unacceptable behavior by participants include the use of sexual language or imagery, derogatory comments or personal attacks, trolling, public or private harassment, insults, or other unprofessional conduct.

Project maintainers have the right and responsibility to remove, edit, or reject comments, commits, code, wiki edits, issues, and other contributions that are not aligned to this Code of Conduct. Project maintainers or contributors who do not follow the Code of Conduct may be removed from the project team.

Instances of abusive, harassing, or otherwise unacceptable behavior may be reported by emailing khmer-project@idyll.org which only goes to C. Titus Brown and Michael R. Crusoe. To report an issue involving either of them please email Judi Brown Clarke, Ph.D. the Diversity Director at the BEACON Center for the Study of Evolution in Action, an NSF Center for Science and Technology.

This Code of Conduct is adapted from the Contributor Covenant, version 1.0.0, available at http://contributor-covenant.org/version/1/0/0/

CHAPTER 7

License

CHAPTER 8

Indices and tables

- genindex
- modindex
- search