screed Documentation

Release 0.7.1

Alex Nolley and Titus Brown

April 05, 2015

Contents

Authors Alex Nolley, C. Titus Brown

Contact ctb@msu.edu

Contents:

CHAPTER 1

Basic Documentation

1.1 Notes on this document

This is the default documentation for screed. Some doctests are included in the file 'example.txt'. The examples in this file are meant for humans only: they will not work in doctests.

1.2 Introduction

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.

1.3 Getting Going

The following software packages are required to run screed:

- Python 2.4 or newer
- nose (for testing)

1.3.1 Downloading

You will need git to download a copy from the public git repository:

    git clone git://github.com/ged-lab/screed.git

1.3.2 Installing

Assuming you have already downloaded the package, this is how to install:

```
$ python setup.py install
```

To run the optional tests type:

```
$ python -m screed.tests.__main__
```

1.4 Quick-Start

1.4.1 Reading FASTA/FASTQ files

At the Python prompt, type:

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

Here, 'filename' can be a FASTA or FASTQ file, and can be uncompressed, gzipped, or bz2-zipped.

1.4.2 Creating a database from the API

From a Python prompt type:

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

That command just parsed the FASTA file 'screed/tests/test.fa' into a screed-database named 'screed/tests/test.fa_screed'. The screed database is independent from the text file it was derived from, so moving, renaming or deleting the 'screed/tests/test.fa' file will not affect screed's operation. To create a screed database from a FASTQ file the syntax is similar:

```
>>> screed.read_fastq_sequences('screed/tests/test.fastq')
```

1.4.3 Creating a database from a script

To create a screed db from a FASTQ file at the shell:

```
$ ./fqdbm screed/tests/test.fastq
```

Similarly, to create a screed db from a fasta file:

```
$ ./fadbm screed/tests/test.fa
```

Alternately, if the screed module is in your PATH:

```
$ python -m screed.fadbm <fasta file>
$ python -m screed.fqdbm <fastq file>
```

where <fast* file> is the path to a sequence file.

screed natively supports FASTA and FASTQ database creation. If you have a new sequence you want screed to work with, see the section below on Writing Custom Sequence Parsers.

1.5 Reading databases

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!

1.5.1 Opening

In the Python environment, import the ScreedDB class and load some databases:

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

1.5.2 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, accuracy 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()
```

Length of the databases are easily found:

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

1.5.3 Retrieving Records

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 whole records:

```
>>> records = []
>>> for record in fadb.itervalues():
...     records.append(record)
```

What is returned 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.

1.5.4 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 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'.

1.5.5 Retrieving 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.

1.6 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.

1.6.1 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.

1.6.2 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()
```

1.6.3 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.

1.6.4 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.

1.7 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.

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

1.7.2 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.

1.8 FASTA <-> FASTQ Conversion

@CTB this doesn't work?

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

1.8.1 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 called ToFastq.py:

```
$ ./ToFastq.py <path to fasta db> <converted fastq file>
```

or:

```
$ python -m screed.ToFastq <path to fasta db> <converted fastq file>
```

if the screed module is in your PATH.

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 accuracy or quality score in a FASTA file, a default one is generated. The generation of the accuracy 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 accuracy should not rely on this automatic conversion.

1.8.2 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:

```
$ ./ToFasta.py <path to fastq db> <converted fasta file>
```

or:

```
$ python -m screed.ToFasta <path to fastq db> <converted fasta file>
```

if the screed module is in your PATH.

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 accuracy score is ignored.

CHAPTER 2

screed examples

2.1 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

Indices and tables

- *genindex*
- *modindex*
- *search*