khmer Documentation

Release 2.1.2

Jul 03, 2017

Contents

CHAPTER 1

What is khmer?

The khmer software is a set of command-line tools for working with DNA shotgun sequencing data from genomes, transcriptomes, metagenomes, and single cells. khmer can make *de novo* assemblies faster, and sometimes better. khmer can also identify (and fix) problems with shotgun data. You can read more about khmer in our software paper.

khmer is free and open source software.

To install khmer, you will need a Linux or Mac computer, together with Python 2.7 or Python 3.x. See *our installation docs* for detailed instructions.

To use khmer, you will generally need to work at the UNIX command line. See *our command line documentation*.

We have **additional documentation** in several places, including protocols for metagenome and mRNAseq assembly and recipes for several common research tasks. You might also be interested in *papers using or citing khmer*.

To get help, *please follow this guide*.

We welcome contributions to the khmer project! We are friendly and supportive of new contributors, and have a *code of conduct*. Please see our *docs on getting started on khmer development*.

CHAPTER 2

Details

Authors Daniel Standage, Hussien F. Alameldin, Ali Aliyari, Sherine Awad, Elmar Bucher, Adam Caldwell, Reed Cartwright, Amanda Charbonneau, Lisa Cohen, Bede Constantinides, Michael R. Crusoe, Greg Edvenson, Scott Fay, Jacob Fenton, Thomas Fenzl, Jordan Fish, Leonor Garcia- Gutierrez, Phillip Garland, Jonathan Gluck, Iván González, Sarah Guermond, Jiarong Guo, Aditi Gupta, Tim Head, Joshua R. Herr, Adina Howe, Alex Hyer, Andreas Härpfer, Luiz Irber, Shannon EK Joslin, Rhys Kidd, Nicole Kingsley, David Lin, Justin Lippi, Tamer Mansour, Pamela McA'Nulty, Eric McDonald, Jessica Mizzi, Kevin D. Murray, Joshua R. Nahum, Kaben Nanlohy, Russell Neches, Alexander Johan Nederbragt, Humberto Ortiz-Zuazaga, Jeramia Ory, Jason Pell, Charles Pepe-Ranney, Zachary N Russ, Erich Schwarz, Camille Scott, Josiah Seaman, Ryan Shean, Scott Sievert, Jared Simpson, Connor T. Skennerton, James Spencer, Ramakrishnan Srinivasan, James A. Stapleton, Joe Stein, Sascha Steinbiss, Susan R Steinman, Cait Sydney, Benjamin Taylor, Will Trimble, Heather L. Wiencko, Michael Wright, Brian Wyss, Qingpeng Zhang, en zyme, C. Titus Brown

Contact khmer-project@idyll.org

GitHub https://github.com/dib-lab/khmer

License BSD

Introduction to khmer

Introduction

khmer is a software library and toolkit for k-mer based analysis and transformation of nucleotide sequence data. The primary focus of development has been scaling assembly of metagenomes and transcriptomes.

khmer supports a number of transformations, both inexact transformations (abundance filtering; error trimming) and exact transformations (graph-size filtering, to throw away disconnected reads; partitioning, to split reads into disjoint sets). All of these transformations operate with constant memory consumption (with the exception of partitioning), and typically require less memory than is required to assemble the data.

Most of khmer is built around a single underlying probabilistic data structure known as a Bloom filter (also see Count-Min Sketch and These Are Not The k-mers You're Looking For). In khmer this data structure is implemented as a

set of hash tables, each of different size, with no collision detection. These hash tables can store either the presence (Bloom filter) or frequency (Count-Min Sketch) of specific k-mers. The lack of collision detection means that the data structure may report a k-mer as being *present* when it is not, in fact, in the data set. However, it will never incorrectly report a k-mer as being absent when it *truly is* present. This one-sided error makes the Bloom filter very useful for certain kinds of operations.

khmer supports arbitrarily large k-sizes, although certain graph-based operations are limited to k <= 32.

The khmer core library is implemented in C++, while all of the khmer scripts and tests access the core library via a Python wrapper.

Tutorials highlighting khmer are available at khmer-protocols and khmer-recipes. The former provides detailed protocols for using khmer to analyze either a transcriptome or a metagenome. The latter provides individual recipes for using khmer in a variety of sequence-oriented tasks such as extracting reads by coverage, estimating a genome or metagenome size from unassembled reads, and error-trimming reads via streaming k-mer abundance.

Using khmer

khmer comes "out of the box" with a number of scripts that make it immediately useful for a few different operations, including (but not limited to) the following.

- normalizing read coverage ("digital normalization")
- dividing reads into disjoint sets that do not connect ("partitioning")
- eliminating reads that will not be used by a de Bruijn graph assembler;
- removing reads with low- or high-abundance k-mers;
- trimming reads of certain kinds of sequencing errors;
- counting k-mers and estimating data set coverage based on k-mer counts;
- running Velvet and calculating assembly statistics;
- converting FASTQ to FASTA;
- converting between paired and interleaved formats for paired FASTQ data

Practical considerations

The most important thing to think about when using khmer is whether or not the transformation or filter you're applying is appropriate for the data you're trying to assemble. For example, two of the most powerful operations available in khmer, graph-size filtering and graph partitioning, only make sense for assembly datasets with many theoretically unconnected components. This is typical of metagenomic data sets. Also, while digital normalization can be helpful for transcriptome *assembly*, it is inappropriate for other RNA-seq applications, such as differential expression analysis, that rely on signal from variable coverage.

The second most important consideration is memory usage. The effectiveness of all of the Bloom filter-based functions (which is everything interesting in khmer!) depends critically on having enough memory to do a good job. See *Setting khmer memory usage* for more information.

Copyright and license

the BSD License. Distribution, modification and redistribution, incorporation into other software, and pretty much everything else is allowed.

Contributors and Acknowledgements

khmer is a product of the Lab for Data Intensive Biology at the University of California, Davis (the succesor to the GED lab at Michigan State University),

http://ivory.idyll.org/lab/

—

C. Titus Brown <titus@idyll.org> wrote the initial Bloom filter and Count-Min Sketch implementations, has contributed to their continued improvement and refactoring, and has contributed extensively to feature development and code review throughout the codebase more generally.

Jason Pell implemented many of the C++ k-mer filtering functions.

Qingpeng contributed code to do unique k-mer counting.

Adina Howe, Rosangela Canino-Koning, and Arend Hintze contributed significantly to discussions of approaches and algorithms; Adina wrote a number of scripts.

Jared T. Simpson (University of Cambridge, Sanger Institute) contributed paired-end support for digital normalization.

Eric McDonald thoroughly revised many aspects of the code base, made much of the codebase thread safe, and otherwise improved performance dramatically.

Michael R. Crusoe took over maintainership in June 2013, streamlining and improving many of khmer's development, deployment, and community processes.

Jacob Fenton...

Kevin Murray enabled use of the C++ code-base by external projects, fixed numerous bugs and documentation issues, implemented unit tests, enabled machine-readable statistics and miscellaneous code cleaning, refactoring, and reviewing.

Luiz Irber implemented an efficient HyperLogLog-based cardinality estimator, contributed substantially to screed/khmer integration (including spearheading the screed 1.0 release), and has contributed extensively to code review.

Camille Scott has contributed significantly to khmer's assembly and graph traversal functionality, and has contributed to feature development and code review throughout the codebase more generally.

Tim Head has contributed to refactoring the core data structures, performance benchmarking, and has contributed extensively to feature development and code review throughout the codebase more generally.

Daniel Standage took over maintainership in May 2016, has refined the documentation extensively, contributed Python and C++ code examples, refactored core data structures for a more extensible sequence loading functionality, and contributed more generally to the codebase.

Last updated by DSS on 2017-05-22

Citations

Software Citation

If you use the khmer software, you must cite:

Crusoe et al., The khmer software package: enabling efficient nucleotide sequence analysis. 2015. http://dx.doi.org/10.12688/f1000research.6924.1

```
@article{khmer2015,
   author = "Crusoe, Michael R. and Alameldin, Hussien F. and Awad, Sherine
and Bucher, Elmar and Caldwell, Adam and Cartwright, Reed and Charbonneau,
Amanda and Constantinides, Bede and Edvenson, Greg and Fay, Scott and Fenton,
Jacob and Fenzl, Thomas and Fish, Jordan and Garcia-Gutierrez, Leonor and
Garland, Phillip and Gluck, Jonathan and GonzÃ¡lez, IvÃ¡n and Guermond, Sarah
and Guo, Jiarong and Gupta, Aditi and Herr, Joshua R. and Howe, Adina and
Hyer, Alex and HÃŒrpfer, Andreas and Irber, Luiz and Kidd, Rhys and Lin, David
and Lippi, Justin and Mansour, Tamer and McA'Nulty, Pamela and McDonald, Eric
and Mizzi, Jessica and Murray, Kevin D. and Nahum, Joshua R. and Nanlohy,
Kaben and Nederbragt, Alexander Johan and Ortiz-Zuazaga, Humberto and Ory,
Jeramia and Pell, Jason and Pepe-Ranney, Charles and Russ, Zachary N and
Schwarz, Erich and Scott, Camille and Seaman, Josiah and Sievert, Scott and
Simpson, Jared and Skennerton, Connor T. and Spencer, James and Srinivasan,
Ramakrishnan and Standage, Daniel and Stapleton, James A. and Stein, Joe and
Steinman, Susan R and Taylor, Benjamin and Trimble, Will and Wiencko, Heather
L. and Wright, Michael and Wyss, Brian and Zhang, Qingpeng and zyme, en and
Brown, C. Titus"
   title = "The khmer software package: enabling efficient nucleotide
sequence analysis",
   year = "2015",
   month = "08",
   publisher = "F1000",
   url = "http://dx.doi.org/10.12688/f1000research.6924.1"
}
```

If you use any of our published scientific methods you should *also* cite the relevant paper(s) as directed below. Additionally some scripts use the SeqAn library for read parsing: the full citation for that library is also included below.

To see a quick summary of papers for a given script just run it without using any command line arguments.

Graph partitioning and/or compressible graph representation

The **`load-graph.py`**, **`partition-graph.py`**, and **`find-knots.py`** scripts are part of the compressible graph representation and partitioning algorithms described in:

Pell J, Hintze A, Canino-Koning R, Howe A, Tiedje JM, Brown CT. Scaling metagenome sequence assembly with probabilistic de Bruijn graphs Proc Natl Acad Sci U S A. 2012 Aug 14;109(33):13272-7. http://dx.doi.org/10.1073/pnas.1121464109. PMID: 22847406

```
@article{Pell2012,
    author = "Pell, Jason and Hintze, Arend and Canino-Koning, Rosangela and
Howe, Adina and Tiedje, James M. and Brown, C. Titus",
    title = "Scaling metagenome sequence assembly with probabilistic de Bruijn
graphs",
    volume = "109",
    number = "33",
    pages = "13272-13277",
    year = "2012",
    doi = "10.1073/pnas.1121464109",
    abstract ="Deep sequencing has enabled the investigation of a wide range of
environmental microbial ecosystems, but the high memory requirements for de
novo assembly of short-read shotgun sequencing data from these complex
populations are an increasingly large practical barrier. Here we introduce a
memory-efficient graph representation with which we can analyze the k-mer
```

```
connectivity of metagenomic samples. The graph representation is based on a
probabilistic data structure, a Bloom filter, that allows us to efficiently
store assembly graphs in as little as 4 bits per k-mer, albeit inexactly. We
show that this data structure accurately represents DNA assembly graphs in low
memory. We apply this data structure to the problem of partitioning assembly
graphs into components as a prelude to assembly, and show that this reduces the
overall memory requirements for de novo assembly of metagenomes. On one soil
metagenome assembly, this approach achieves a nearly 40-fold decrease in the
maximum memory requirements for assembly. This probabilistic graph
representation is a significant theoretical advance in storing assembly graphs
and also yields immediate leverage on metagenomic assembly.",
    URL = "http://www.pnas.org/content/109/33/13272.abstract",
    eprint = "http://www.pnas.org/content/109/33/13272.full.pdf+html",
    journal = "Proceedings of the National Academy of Sciences"
}
```

Digital normalization

The **normalize-by-median.py** and **count-median.py** scripts are part of the digital normalization algorithm, described in:

> A Reference-Free Algorithm for Computational Normalization of Shotgun Sequencing Data Brown CT, Howe AC, Zhang Q, Pyrkosz AB, Brom TH arXiv:1203.4802 [q-bio.GN] http://arxiv.org/abs/1203.4802

```
@unpublished{diginorm,
    author = "C. Titus Brown and Adina Howe and Qingpeng Zhang and Alexis B.
Pyrkosz and Timothy H. Brom",
    title = "A Reference-Free Algorithm for Computational Normalization of
Shotgun Sequencing Data",
    year = "2012",
    eprint = "arXiv:1203.4802",
    url = "http://arxiv.org/abs/1203.4802",
}
```

Efficient k-mer error trimming

The **script trim-low-abund.py** is described in:

> Crossing the streams: a framework for streaming analysis of short DNA sequencing reads Zhang Q, Awad S, Brown CT https://dx.doi.org/10.7287/peerj.preprints.890v1

```
@unpublished{semistream,
    author = "Qingpeng Zhang and Sherine Awad and C. Titus Brown",
    title = "Crossing the streams: a framework for streaming analysis of
        short DNA sequencing reads",
    year = "2015",
    eprint = "PeerJ Preprints 3:e1100",
    url = "https://dx.doi.org/10.7287/peerj.preprints.890v1"
}
```

K-mer counting

The **abundance-dist.py**, **filter-abund.py**, and **load-into-counting.py** scripts implement the probabilistic k-mer counting described in:

These Are Not the K-mers You Are Looking For: Efficient Online K-mer Counting Using a Probabilistic Data Structure Zhang Q, Pell J, Canino-Koning R, Howe AC, Brown CT. http://dx.doi.org/10.1371/journal.pone.0101271

```
@article{khmer-counting,
    author = "Zhang, Qingpeng AND Pell, Jason AND Canino-Koning, Rosangela
AND Howe, Adina Chuang AND Brown, C. Titus",
    journal = "PLoS ONE",
    publisher = "Public Library of Science",
    title = "These Are Not the K-mers You Are Looking For: Efficient Online
K-mer Counting Using a Probabilistic Data Structure",
    year = "2014",
    month = "07",
    volume = "9",
    url = "http://dx.doi.org/10.1371/journal.pone.0101271",
    pages = "e101271",
    abstract = "<p>K-mer abundance analysis is widely used for many purposes in
nucleotide sequence analysis, including data preprocessing for de novo
assembly, repeat detection, and sequencing coverage estimation. We present the
khmer software package for fast and memory efficient <italic>online</italic>
counting of k-mers in sequencing data sets. Unlike previous methods based on
data structures such as hash tables, suffix arrays, and trie structures, khmer
relies entirely on a simple probabilistic data structure, a Count-Min Sketch.
The Count-Min Sketch permits online updating and retrieval of k-mer counts in
memory which is necessary to support online k-mer analysis algorithms. On
sparse data sets this data structure is considerably more memory efficient than
any exact data structure. In exchange, the use of a Count-Min Sketch introduces
a systematic overcount for k-mers; moreover, only the counts, and not the
k-mers, are stored. Here we analyze the speed, the memory usage, and the
miscount rate of khmer for generating k-mer frequency distributions and
retrieving k-mer counts for individual k-mers. We also compare the performance
of khmer to several other k-mer counting packages, including Tallymer,
Jellyfish, BFCounter, DSK, KMC, Turtle and KAnalyze. Finally, we examine the
effectiveness of profiling sequencing error, k-mer abundance trimming, and
digital normalization of reads in the context of high khmer false positive
rates. khmer is implemented in C++ wrapped in a Python interface, offers a
tested and robust API, and is freely available under the BSD license at
github.com/dib-lab/khmer.</p>",
    number = "7",
    doi = "10.1371/journal.pone.0101271"
}
```

FASTA and FASTQ reading

Several scripts use the SeqAn library for FASTQ and FASTA reading as described in:

SeqAn An efficient, generic C++ library for sequence analysis Döring A, Weese D, Rausch T, Reinert K. http://dx.doi.org/10.1186/1471-2105-9-11

```
@Article{SeqAn,
  AUTHOR = {Doring, Andreas and Weese, David and Rausch, Tobias and Reinert,
    Knut},
  TITLE = {SeqAn An efficient, generic C++ library for sequence analysis},
  JOURNAL = {BMC Bioinformatics},
  VOLUME = {9},
  YEAR = {2008},
  NUMBER = {1},
```

```
  PAGES = {11},
  URL = {http://www.biomedcentral.com/1471-2105/9/11},
  DOI = {10.1186/1471-2105-9-11},
  PubMedID = {18184432},
  ISSN = {1471-2105},
  ABSTRACT = {BACKGROUND: The use of novel algorithmic techniques is pivotal
  to many important problems in life science. For example the sequencing of
  the human genome [1] would not have been possible without advanced assembly
  algorithms. However, owing to the high speed of technological progress and
  the urgent need for bioinformatics tools, there is a widening gap between
  state-of-the-art algorithmic techniques and the actual algorithmic
  components of tools that are in widespread use. RESULTS: To remedy this
  trend we propose the use of SeqAn, a library of efficient data types and
  algorithms for sequence analysis in computational biology. SeqAn comprises
  implementations of existing, practical state-of-the-art algorithmic
  components to provide a sound basis for algorithm testing and development.
  In this paper we describe the design and content of SeqAn and demonstrate
  its use by giving two examples. In the first example we show an application
  of SeqAn as an experimental platform by comparing different exact string
  matching algorithms. The second example is a simple version of the well-
  known MUMmer tool rewritten in SeqAn. Results indicate that our
  implementation is very efficient and versatile to use. CONCLUSION: We
  anticipate that SeqAn greatly simplifies the rapid development of new
  bioinformatics tools by providing a collection of readily usable, well-
  designed algorithmic components which are fundamental for the field of
  sequence analysis. This leverages not only the implementation of new
  algorithms, but also enables a sound analysis and comparison of existing
  algorithms.},
}
```

Release notes

Contents:

khmer v1.0 release notes

582 changed files with 40,527 additions and 31,772 deletions.

The team has been hard at work since v0.8 to refine the codebase into a stable product.

https://khmer.readthedocs.org/en/latest/

With the 1.0 release we are making a commitment to using Semantic Versioning[0]: the version number will reflect the impact of the changes between releases. New major versions will likely require you to change how you use the project. Minor versions indicate new functionality that doesn't impact the existing. Patch versions indicate backwards-compatible fixes. Right now we are limiting this promise to the command-line interface. A future release will introduce a stable and mature Python API to the khmer project and at that time we will extend the version system to include that API.

New items of note:

CITATION: Each script now outputs information on how to cite it. There is a new paper to describes the project overall: MR Crusoe et al., 2014. doi: 10.6084/m9.figshare.979190

The documentation for the scripts has undergone an overhaul. The scripts now output extensive notes and the formal documentation website is generated from the scripts themselves and will never be out of sync.

https://khmer.readthedocs.org/en/latest/scripts.html

Notable bugs fixed/issues closed:

git clone of the khmer repo reqs > 0.5 GiB #223 @mr-c new khmer/file module #357 @RamRS Floating point exception in count-overlap.py #282 @qingpeng add documentation for sample-reads-randomly #192 @mr-c only build zlib and bzip2 when needed #168 @mr-c

Minor updates

khmer tools should output intelligent error messages when fed empty files #135 @RamRS set IParser::ParserState::ParserState:fill_id to zero at initialization #356 @mr-c demote nose & sphinx to extra dependencies. #351 @mr-c CID 1054792 (Medium) Uninitialized scalar field (UNINIT_CTOR) #179 @mr-c CID 1077117 (Medium): Division or modulo by zero (DIVIDE_BY_ZERO) #182 @mr-c if –savehash is specified then don't continue if there is not enough free disk space #245 @RamRS finish fixing implicit downcasts #330 @mr-c Clean up compile warnings in subset.cc #172 @mr-c all scripts need to output their version #236 @mr-c environmental variables need documenting #303 @mr-c C++ code should be consistently formatted #261 @mr-c Clean up ancillary files #146 @mr-c squash option not implemented in abundance-dist-single.py #271 @RamRS Add documentation on how to tie into a particular tagged version #29 @mr-c pip install -e fails with compile error #352 @mr-c remove the unused KTable object #337 @luizirber zlib 1.2.3 -> zlib 1.2.8 #336 @mr-c CID 1173035: Uninitialized scalar field (UNINIT_CTOR) #311 @mr-c CID 1153101: Resource leak in object (CTOR_DTOR_LEAK) #309 @mr-c remove khmer::read_parsers::IParser::ParserState::thread_id #323 @mr-c several modifications about count-overlap.py script #324 @qingpeng fixed runscript to handle SystemExit #332 @ctb CID 1063852: Uninitialized scalar field (UNINIT_CTOR) #313 @mr-c [infrastructure] update to new Doxyfile format, make version number autoupdate #315 @mr-c Removed an extraneous using namespace khmer; in kmer.hh, #276 @fishjord Minimum and recommended python version #94 @mr-c KmerCount class appears to be unused #302 @mr-c If loadhash is specified in e.g. normalize-by-median, don't complain about default hashsize parameters #117 @RamRS

Known Issues

All of these are pre-existing.

Some users have reported that normalize-by-median.py will utilize more memory than it was configured for. This is being investigated in https://github.com/dib-lab/khmer/issues/266

Some FASTQ files confuse our parser when running with more than one thread. For example, while using load-into-counting.py. If you experience this then add "–threads=1" to your command line. This issue is being tracked in https://github.com/dib-lab/khmer/issues/249

If your k-mer table (hashfile) gets truncated, perhaps from a full filesystem, then our tools currently will get stuck. This is being tracked in https://github.com/dib-lab/khmer/issues/247 and https://github.com/dib-lab/khmer/issues/96 and https://github.com/dib-lab/khmer/issues/246

Paired-end reads from Casava 1.8 currently require renaming for use in normalize-by-median and abund-filter when used in paired mode. The integration of a fix for this is being tracked in https://github.com/dib-lab/khmer/issues/23

annotate-partitions.py only outputs FASTA even if given a FASTQ file. This issue is being tracked in https://github.com/dib-lab/khmer/issues/46

A user reported that abundance-dist-single.py fails with small files and many threads. This issue is being tracked in https://github.com/dib-lab/khmer/issues/75

Contributors

@camillescott, @mr-c, @ctb, @luizirber, @RamRS, @qingpeng

[0] http://semver.org/

khmer v1.0.1 release notes

This is bugfix release. Note: the installation instructions have been slightly simplified.

https://khmer.readthedocs.org/en/v1.0.1/

New items of note:

This release successfully installs and passes its unit tests on Debian 6.0 "Squeeze", Debian 7.0 "Wheezy", Fedora 19, OS X 7 "Lion", OS X 8 "Mountain Lion", Red Hat Enterprise Linux 6, Scientific Linux 6, Ubuntu 10.04 LTS, and Ubuntu 12.04 LTS. Thanks to the UW-Madison Build and Test Lab for their testing infrastructure.

Notable bugs fixed/issues closed:

fixed thread hanging issue #406 @ctb Explicit python2 invocation #404 @mr-c MANIFEST.in,setup.py: fix to correct zlib packaging #365 @mr-c fixed check_space_for_hashtable to use args.n_tables #382 @ctb Bug fix: make-initial-stoptags.py error on missing .ht input file, actual input file is .pt #391 @mr-c

Minor updates

include calc-best-assembly.py in v1.0.1 #409 @ctb updated normalize-by-median documentation for loadtable #378 @ctb updated diginorm for new FP rate info; corrected spelling error #398 @ctb Add spellcheck to code review checklist. #397 @ctb

Known Issues

All of these are pre-existing.

Some users have reported that normalize-by-median.py will utilize more memory than it was configured for. This is being investigated in https://github.com/dib-lab/khmer/issues/266

Some FASTQ files confuse our parser when running with more than one thread. For example, while using load-into-counting.py. If you experience this then add "–threads=1" to your command line. This issue is being tracked in https://github.com/dib-lab/khmer/issues/249

If your k-mer table (hashfile) gets truncated, perhaps from a full filesystem, then our tools currently will get stuck. This is being tracked in https://github.com/dib-lab/khmer/issues/247 and https://github.com/dib-lab/khmer/issues/246

Paired-end reads from Casava 1.8 currently require renaming for use in normalize-by-median and abund-filter when used in paired mode. The integration of a fix for this is being tracked in https://github.com/dib-lab/khmer/issues/23

annotate-partitions.py only outputs FASTA even if given a FASTQ file. This issue is being tracked in https://github.com/dib-lab/khmer/issues/46

A user reported that abundance-dist-single.py fails with small files and many threads. This issue is being tracked in https://github.com/dib-lab/khmer/issues/75

Contributors

@mr-c, @ctb, @luizirber, @RamRS, @ctSkennerton

khmer v1.1 release notes

This is v1.1, a minor version release; this version adds several new scripts.

Docs at: https://khmer.readthedocs.org/en/v1.1/

Release notes w/links: https://github.com/dib-lab/khmer/releases/tag/v1.1

New items of note:

- removed unnecessary files from PyPI package; distribution is now under 2 MB (#419) @mr-c
- tests are now distributed with package and can be run after 'pip install' (#451) @mr-c
- complain properly on file read failures (#333) @ctb
- Sequence loading scripts will now report total numbers of k-mers if given –report_total_kmers (#491/#429) @mr-c
- added metagenome protocol to acceptance testing (#472) @SherineAwad @ctb

Notable bugs fixed/issues closed:

- removed sandbox/load-into-hashbits.py (superseded by scripts/load-graph.py –no-tagset) (#480, @wrightmhw)
- promoted extract-long-sequences.py to scripts (#461, @wrightmhw)
- promoted fastq-to-fasta.py to scripts (#436, @wrightmhw)
- remove incorrect filesystem space check from abundance-dist.py (#452, @chuckpr)
- when counting hash writes fail, produce error message (#411, @znruss)
- removed a number of memory leaks found by Coverity and valgrind (#451, @mr-c)
- updated reservoir sampling to produce multiple subsamples with -S (#197, @ctb)
- fixed pip2, python2 issues (#428 and #485, @accaldwell @mr-c)
- removed untested/unused code and scripts (#438, @mr-c)

Known issues:

All of these are pre-existing.

Some users have reported that normalize-by-median.py will utilize more memory than it was configured for. This is being investigated in https://github.com/dib-lab/khmer/issues/266

Some FASTQ files confuse our parser when running with more than one thread. For example, while using load-into-counting.py. If you experience this then add "–threads=1" to your command line. This issue is being tracked in https://github.com/dib-lab/khmer/issues/249

If your k-mer table is truncated on write, an error may not be reported; this is being tracked in https://github.com/dib-lab/khmer/issues/443. However, khmer will now (correctly) fail when trying to read a truncated file (See #333).

Paired-end reads from Casava 1.8 currently require renaming for use in normalize-by-median and abund-filter when used in paired mode. The integration of a fix for this is being tracked in https://github.com/dib-lab/khmer/issues/23

Some scripts only output FASTA even if given a FASTQ file. This issue is being tracked in https://github.com/dib-lab/khmer/issues/46

A user reported that abundance-dist-single.py fails with small files and many threads. This issue is being tracked in https://github.com/dib-lab/khmer/issues/75

Contributors

@mr-c, @ctb, @camillescott, @wrightmhw, @chuckpr, @luizirber, @accaldwell, @znruss

khmer v1.2 release notes

This is the v1.2 release of khmer: minor new features and bug fixes. The start of this release cycle coincided with the Mozilla Science Lab Global Sprint 2014. We honor and thank the 19 new contributors (including four Michigan State University undergraduates) who volunteered their time to contribute!

Docs at: https://khmer.readthedocs.org/en/v1.2/

New items of note:

@mr-c and @ctb are proud to announce khmer's code of conduct http://khmer.readthedocs.org/en/v1.2/dev/CODE_OF_CONDUCT.html #664 All scripts list which files have been created during their execution #477 @bocajnotnef All scripts now only output status messages to STDERR instead of STDOUT #626 @b-wyss docs/ a fairly major re-organization and brand new developer docs @ctb @mr-c load-into-counting.py: `--summary-info`: machine readable summary in JSON or TSV format #649 @kdmurray91 scripts/extract-partitions.py: added documentation for .dist columns #516 @chuckpr Makefile: a new target `make install-dependencies` is useful for developers #539 @mr-c Sandbox scripts have been cleaned up, or removed (see the sandbox/README.rst for details) #589 @ctb

Notable bugs fixed/issues closed:

do-partition.py's excessive spawning of threads fixed. #637 @camillescott Fixed unique k-mer count reporting in load-graph, load-into-counting, and normalize-by-median. #562 @mr-c Clarified and test the requirement for a 64-bit operating system #529 @Echelon9 Removed some of the broken multi-threading options #511 @majoras-masque Fix table.get("wrong_length_string") gives core dump #585 @Echelon9 filter-abund lists parameters that it doesn't use #524 @jstapleton Reduction of memory required to run the test suite #542 @leogargu BibTeX included in CITATIONS #541 @HLWiencko

Additional fixes/features

delete ScoringMatrix::assign as it is unused #502 @RodPic Root all of our C++ exceptions to a common base exception #508 @iglpdc deleted KhmerError #503 @drlabratory normalize-by-median reporting output after main loop exits, in case it hadn't been triggered #586 @ctb Many issues discovered by cppcheck cleaned up #506 @brtaylor92 Developers have a new Makefile target to autofix formatting: `make format` #612 @brtaylor92 normalize-by-median.py test coverage increased #361 @SherineAwad Several unused functions were removed #599 @brtaylor92 Developer docs now link to the stdc++ docs as appropriate #629 @mr-c Added tests for non-sequential access to input files #644 @bocajnotnef Removed khmer/theading_args.py #653 @bocajnotnef Improved test for maximum k value #658 @pgarland ReadParser no longer crashes if n_threads = 0 #86 @jiarong

Known issues:

All of these are pre-existing.

Multithreaded reading will drop reads. This major issue has been present for several khmer releases and was only found via a much larger test case that we had been previously using. Credit to @camillescott. Workaround: disable threading. The next release will fix this and the other FAST[AQ] parsing issues. https://github.com/dib-lab/khmer/issues/681

Some users have reported that normalize-by-median.py will utilize more memory than it was configured for. This is being investigated in https://github.com/dib-lab/khmer/issues/266

Some FASTQ files confuse our parser when running with more than one thread. For example, while using load-into-counting.py. If you experience this then add "–threads=1" to your command line. This issue is being tracked in https://github.com/dib-lab/khmer/issues/249

If your k-mer table is truncated on write, an error may not be reported; this is being tracked in https://github.com/dib-lab/khmer/issues/443. However, khmer will now (correctly) fail when trying to read a truncated file (See #333).

Paired-end reads from Casava 1.8 currently require renaming for use in normalize-by-median and abund-filter when used in paired mode. The integration of a fix for this is being tracked in https://github.com/dib-lab/khmer/issues/23

Some scripts only output FASTA even if given a FASTQ file. This issue is being tracked in https://github.com/dib-lab/khmer/issues/46

A user reported that abundance-dist-single.py fails with small files and many threads. This issue is being tracked in https://github.com/dib-lab/khmer/issues/75

Contributors

@mr-c, @ctb, ‡@bocajnotnef, ‡@Echelon9, ‡@jlippi, ‡@kdmurray91, @qingpeng, ‡@leogargu, ‡@jiarong, ‡@brtaylor92, ‡@iglpdc, @camillescott, ‡@HLWiencko, ‡@cowguru2000, ‡@drlabratory, ‡@jstapleton, ‡@b-wyss, ‡@jgluck, @fishjord, ‡@SherineAwad, ‡@pgarland, ‡@majoras-masque, @chuckpr, ‡@RodPic, @luizirber, ‡@jrherr

‡ Denotes new contributor

khmer v1.3 release notes

This is the v1.3 release of khmer featuring a new FAST[AQ] parser from the SeqAn project.

Docs at: https://khmer.readthedocs.org/en/v1.3/

New items of note:

Fixes the two multithreaded reading of sequence files issues: FASTQ parsing and the recently found read dropping issue. Several khmer scripts now support reading from non-seekable plain and gziped FAST[AQ] files (a.k.a pipe or streaming support). @mr-c #642

Notable bugs fixed/issues closed:

restore threading to load-graph.py #699 @mr-c

Additional fixes/features

increase filter_abund.py coverage #568 @wrightmhw Provide scripts/ testing coverage for check_space_for_hashtable #386 #678 #718 @b-wyss Use absolute URI in CODE_OF_CONDUCT #684 @jsspencer give SeqAn credit #712 @mr-c Added testing to make sure all sandbox scripts are import-able and execfile-able. #709 @ctb reduce memory requirements to run tests #701 @ctb Two minor bug fixes to sandbox scripts #706 @ctb Upgrade of trim-low-abund for better, more profitable streaming. #601 @ctb Add –force or –expert or –ignore flag to all khmer scripts that do sanity checking #399 #647 @jessicamizzi Add XDECREF for returned read tuple in ReadParser.read_pair_iterator() #693 @mr-c @camillescott

Known issues:

All of these are pre-existing.

Some users have reported that normalize-by-median.py will utilize more memory than it was configured for. This is being investigated in #266

If your k-mer table is truncated on write, an error may not be reported; this is being tracked in https://github.com/dib-lab/khmer/issues/443. However, khmer will now (correctly) fail when trying to read a truncated file (See #333).

Paired-end reads from Casava 1.8 currently require renaming for use in normalize-by-median and abund-filter when used in paired mode. The integration of a fix for this is being tracked in #23

Some scripts only output FASTA even if given a FASTQ file. This issue is being tracked in #46

A user reported that abundance-dist-single.py fails with small files and many threads. This issue is being tracked in #75

Contributors

@mr-c, @ctb, @camillescott, @b-wyss, @wrightmhw, @jsspencer

khmer v1.4 release notes

This is the v1.4 release of khmer featuring the results of our March and April (PyCon) coding sprints and the 16 new contributors; the use of the new v0.8 release of screed (the library we use for pure Python reading of nucleotide sequence files); and the addition of @luizirber's HyperLogLog counter for quick cardinality estimation.

Documentation is at https://khmer.readthedocs.org/en/v1.4/

New items of note:

Casava 1.8 read naming is now fully supported and in general the scripts no longer mangle read names. Side benefits: `split-paired-reads.py` will no longer drop reads with 'bad' names; `count-median.py` can generate output in CSV format. #759 #818 @ctb #873 @ahaerpfer

Most scripts now support a "broken" interleaved paired-read format for FASTA/ FASTQ nucleotide sequence files. `trim-low-abund.py <http://khmer.readthedocs.org/en/v1.4/user/scripts.html#trim-low-abund-py>'__ has been promoted from the sandbox as well (with streaming support). #759 @ctb #963 @sguermond #933 @standage

The script to transform an interleaved paired-read nucleotide sequence file into two files now allows one to name the output files which can be useful in combination with named pipes for streaming processing #762 @ctb

Streaming everywhere: thanks to screed v0.8 we now support streaming of almost all inputs and outputs. #830 @aditi9783 #812 @mr-c #917 @bocajnotnef #882 @standage

Need a quick way to count total number of unique k-mers in very low memory? the `unique-kmers.py` script in the sandbox uses a HyperLogLog counter to quickly (and with little memory) provide an estimate with a controllable error rate. #257 #738 #895 #902 @luizirber

`normalize-by-median.py` can now process both a paired interleaved sequence file and a file of unpaired reads in the same invocation thus removing the need to write the counting table to disk as required in the workaround. #957 @susinmotion

Notable bugs fixed/issues closed:

Paired-end reads from Casava 1.8 no longer require renaming for use in `normalize-by-median.py` and `abund-filter.py` when used in paired mode #818 @ctb

Python version support clarified. We do not (yet) support Python 3.x #741 @mr-c

If a single output file mode is chosen for normalize-by-median.py we now default to overwriting the output. Appending the output is available by using the append redirection operator from the shell. #843 @drtamermansour

Scripts that consume sequence data using C++ will now properly throw an error on truncated files. #897 @kdmurray91 And while writing to disk we properly check for errors #856 #962 @mr-c

`abundance-dist-single.py` no longer fails with small files and many threads. #900 @mr-c

Additional fixes/features

Of interest to users:

Many documentation updates #753 @PamelaM, #782 @bocajnotnef, #845 @alameldin, #804 @ctb, #870 @SchwarzEM, #953 #942 @safay, #929,@davelin1, #687 #912 #926 @mr-c

Installation instructions for Conda, Arch Linux, and Mac Ports have been added #723 @reedacartwright #952 @elmbeech #930 @ahaerpfer

The example script for the STAMPS database has been fixed to run correctly #781 @drtamermansour

`split-paired-reads.py`: added `-o` option to allow specification of an output directory #752 @bede

Fixed a string formatting and a boundary error in `sample-reads-randomly.py` #773 @qingpeng #995 @ctb

CSV output added to `abundance-dist.py`, `abundance-dist-single.py`, and `count-overlap.py`, and `readstats.py` #831 #854 #855 @drtamermansour #959 @anotherthomas

TSV/JSON output of `load-into-counting.py` enhanced with the total number of reads processed #996 @kdmurray91 Output files are now also checked to be writable *before* loading the input files #672 @pgarland @bocajnotnef

`interleave-reads.py` now prints the output filename nicely #827 @kdmurray91

Cleaned up error for input file not existing #772 @jessicamizzi #851 @ctb

Fixed error in `find-knots.py` #860 @TheOneHyer

The help text for `load-into-counting.py` for the `--no-bigcounts`/`-b` flag has been clarified #857 @kdmurray91

@lexnederbragt confirmed an old bug has been fixed with his test for whitespace in sequence identifiers interacting with the `extract-partitions.py` script #979

Now safe to copy-and-paste from the user documentation as the smart quotes have been turned off. #967 @ahaerpfer

The script `make-coverage.py` has been restored to the sandbox. #920 @SherineAwad

`normalize-by-median.py` will warn if two of the input files have the same name #932 @elmbeech

Of interest to developers:

Switched away from using `--user` install for developers #740 @mr-c @drtamermansour & #883 @standage

Developers can now see a summary of important Makefile targets via `make help` #783 @standage

The unused `khmer.load_pe` module has been removed #828 @kdmurray91

Versioneer bug due to new screed release was squashed #835 @mr-c

A Python 2.6 and 2.7.2 specific bug was worked around #869 @kdmurray91 @ctb

Added functions hash_find_all_tags_list and hash_get_tags_and_positions to CountingHash objects #749 #765 @ctb

The `make diff-cover` and ChangeLog formatting requirements have been added to checklist #766 @mr-c

A useful message is now presented if large tables fail to allocate enough memory #704 @mr-c

A checklist for developers adding new CPython types was added #727 @mr-c

The sandbox graduation checklist has been updated to include streaming support #951 @sguermond

Specific policies for sandbox/ and scripts/ content, and a process for adding new command line scripts into scripts/ have been added to the developer documentation #799 @ctb

Sandbox scripts update: corrected #! Python invocation #815 @Echelon9, executable bits, copyright headers, no underscores in filenames #823 #826 #850 @alameldin several scripts deleted, docs + requirements updated #852 @ctb

Avoid running big-memory tests on OS X #819 @ctb

Unused callback code was removed #698 @mr-c

The CPython code was updated to use the new checklist and follow additional best practices #785 #842 @luizirber

Added a read-only view of the raw counting tables #671 @camillescott #869 @kdmurray91

Added a Python method for quickly getting the number of underlying tables in a counting or presence table #879 #880 @kdmurray91

The C++ library can now be built separately for the brave and curious developer #788 @kdmurray91

The ReadParser object now keeps track of the number of reads processed #877 @kdmurray91

Documentation is now reproducible #886 @mr-c

Python future proofing: specify floor division #863 @mr-c

Miscellaneous spelling fixes; thanks codespell! #867 @mr-c

Debian package list update #984 @mr-c

`khmer.kfile.check_file_status()` has been renamed to `check_input_files()` #941 @proteasome `filter-abund.py` now uses it to check the input counting table #931 @safay

`normalize-by-median.py` was refactored to not pass the ArgParse object around #965 @susinmotion

Developer communication has been clarified #969 @sguermond

Tests using the 'fail_okay=true' parameter to `runscript` have been updated to confirm the correct error occurred. 3 faulty tests were fixed and the docs were clarified #968 #971 @susinmotion

FASTA test added for `extract-long-sequences.py` #901 @jessicamizzi

'added silly test for empty file warning' #557 @wltrimbl @bocajnotnef

A couple tests were made more resilient and some extra error checking added in CPython land #889 @mr-c

Copyright added to pull request checklist #940 @sguermond

`khmer_exceptions` are now based on `std::strings` which plugs a memory leak #938 @anotherthomas

Python docstrings were made PEP257 compliant #936 @ahaerpfer

Some C++ comments were converted to be Doxygen compliant #950 @josiahseaman

The counting and presence table warning logic was refactored and centralized #944 @susinmotion

The release checklist was updated to better run the post-install tests #911 @mr-c

The unused method `find_all_tags_truncate_on_abundance` was removed from the CPython API #924 @anotherthomas

OS X warnings quieted #887 @mr-c

Known issues:

All of these are pre-existing.

Some users have reported that normalize-by-median.py will utilize more memory than it was configured for. This is being investigated in https://github.com/dib-lab/khmer/issues/266

Some scripts only output FASTA even if given a FASTQ file. This issue is being tracked in https://github.com/dib-lab/khmer/issues/46

Contributors

@ctb, @kdmurray91, @mr-c, @drtamermansour, @luizirber, @standage, @bocajnotnef, ‡@susinmotion, @jessicamizzi, ‡@elmbeech, ‡@anotherthomas, ‡@sguermond, ‡@ahaerpfer, ‡@alameldin, ‡@TheOneHyer, ‡@aditi9783, ‡@proteasome, ‡@bede, ‡@davelin1, @Echelon9, ‡@reedacartwright, @qingpeng, ‡@SchwarzEM, ‡@scottsievert, @PamelaM, @SherineAwad, ‡@josiahseaman, ‡@lexnederbragt,

‡ Indicates new contributors

Issue reporters

@moorepants, @teshomem, @macmanes, @lexnederbragt, @r-gaia-cs, @magentashades

khmer v2.0 release notes

This is the v2.0 release of khmer and the first from our new lab at the University of California, Davis. It features Python 3 compatibility, streaming I/O from Unix Pipes, mixed-pair sequence file format support, and a new parameter to simplify memory usage. We also have a software paper in-press describing the project and the citation reminders have been updated to reflect that.

Overall there are an additional 2,380 lines of Python code (mostly tests) and 283 less lines of C++ (despite adding features). This release is the product of over 1,000 commits to the codebase since v1.4.

Documentation is at https://khmer.readthedocs.org/en/v2.0/

New items of note:

New behavior

Streaming I/O from Unix Pipes

All scripts now accept input from named (like `/dev/stdin`, or that created using `<( list )` process substituion) and unnamed pipes (like output piped in from another program with `|`). The STDIN stream can also be specified using a single dash: `-`. #1186 @mr-c #1042 #763 @SherineAwad #1085 @ctb

New parameter for memory usage, and/or tablesize/number of table parameters.

There is now a `-M`/`--max-memory-usage` parameter that sets the number of tables (`-N`/`--n\_tables`) and tablesize (`-x`/`--max-tablesize`) parameters automatically to match the desired memory usage. #1106 #621 #1126 #390 #1117 #1055 #1050 #1214 #1179 #1133 #1145 @ctb @qingpeng @bocajnotnef

Digital normalization script now supports mixed paired and unpaired read input

`normalize-by-median.py` now supports mixed paired and unpaired (or "broken-paired") input. Behavior can be forced to either treat all reads as singletons or to require all reads be properly paired using `--force\_single` or `--paired`, respectively. If `--paired` is set, `--unpaired-reads` can be used to include a file of unpaired reads. The unpaired reads will be examined after all of the other sequence files. `normalize-by-median.py` now has a `--quiet` option to reduce the amount of output. #1200 @bocajnotnef

Mixed-pair sequence file format support

`split-paired-reads.py --output-orphaned/-0` has been added to allow for orphaned reads and give them a file to be sorted into. #847 #1164 @ctb

Scripts now output columnar data in CSV format by default

All scripts that output any kind of columnar data now do so in CSV format, with headers. Previously this had to be enabled with `--csv`. (Affects `abundance-dist-single.py`, `abundance-dist.py`, `count-median.py`, and `count-overlap.py`.) `normalize-by-median.py --report` also now outputs in CSV format. #1011 #1180 @ctb

Reservoir sampling script extracts paired reads by default

`sample-reads-randomly.py` now retains pairs in the output, by default. This can be overridden to match previous behavior with `--force\_single`.

Most input and output files can be compressed

We support gzip and bzip2 input and output file compression everywhere that it makes sense #505 #747 @bocajnotnef

New scripts

Estimate number of unique kmers

`unique-kmers.py` estimates the k-mer cardinality of a dataset using the HyperLogLog probabilistic data structure. This allows very low memory consumption, which can be configured through an expected error rate. Even with

low error rate (and higher memory consumption), it is still much more efficient than exact counting and alternative methods. It supports multicore processing (using OpenMP) and streaming, and so can be used in conjunction with other scripts (like `normalize-by-median.py` and `filter-abund.py`). This script is the work of @luizirber and the subject of a paper in draft. #390 #1239 #1252 #1053 #1072 #1145 #1176 #1207 #1204 #1245

Incompatible changes

New datastructure and script names

For clarity the Count-Min Sketch based data structure previously known as "counting_hash" or "counting_table" and variations of these is now known as `countgraph`. Likewise with the Bloom Filter based data structure previously known at "hashbits", "presence_table" and variations of these is now known as `nodegraph`. Many options relating to `table` have been changed to `graph`. #1112 #1209 @mr-c

Binary file formats have changed

All binary khmer formats (presence tables, counting tables, tag sets, stop tags, and partition subsets) have changed. Files are now pre-pended with the string `OXLI` to indicate that they are from this project. #519 #1031 @mr-c #1159 @luizirber

Files of the above types made in previous versions of khmer are not compatible with v2.0; the reverse is also true.

In addition to the `OXLI` string, the Nodegraph and Countgraph file format now includes the number of occupied bins. See http://khmer.readthedocs.org/en/v2.0/dev/binary-file-formats for details. #1093 @ctb @mr-c #1101 #1103 @kdmurray91

load-graph.py no longer appends .pt to the specified filename

Previously, `load-graph.py` appended a `.pt` extension to the specified output filename and partition-graph.py appended a `.pt` to the given input filename. Now, `load-graph.py` writes to the specified output filename and `partition-graph.py` does not append a `.pt` to the given input filename. #1189 #747 @bocajnotnef

Some reporting options have been turned always on

The total number of unique k-mers will always be reported every time a new countgraph is made. The `--report-total-kmers` option has been removed from `abundance-dist-single.py`, `filter-abund-single.py`, and `normalize-by-median.py` to reflect this. Likewise with `--write-fp-rate` for `load-into-counting.py` and `load-graph.py`; the false positive rate will always be written to the `.info` files. #1097 #1180 @ctb

An uncommon error recovery routine was removed

To simplify the codebase `--save-on-failure` and its helper option `--dump-frequency` have been removed from `normalize-by-median.py`.

Single file output option names have been normalized

`--out` is now `--output` for both `normalize-by-median.py` and `trim-low-abund.py`. #1188 #1164 @ctb

Miscellaneous changes

The common option `--min-tablesize` was renamed to `--max-tablesize` to reflect this more desirable behavior.

In conjuction with the new `split-paired-reads.py --output-orphaned` option, the option `--force-paired`/`-p` has been eliminated.

As CSV format is now the default, the `--csv` option has been removed.

Removed script

count-overlap.py has been removed.

Notable bugs fixed/issues closed:

When `normalize-by-median.py` decides to keep both parts of a pair of reads it was only adding the k-mers & counts from one to the countgraph. #1000 #1010 @drtamermansour @bocajnotnef

The partition map file format was not robust to truncation and would hang waiting for more data. #437 #1037 #1048 @ctb

`extract-paired-reads.py` and `split-paired-reads.py` no longer create default files when the user supplies filename(s). #1005 #1132 @kdmurray91

Additional fixes/features

`find-knots.py` was missing a `--force` option and unit tests. #358 #1078 @ctb The check for excessively high false-positive rate has also received a `--force` option #1168 @bocajnotnef

A bug leading to an infinite loop with large gzipped countgraphs was found #1038 #1043 @kdmurray91

All scripts that create nodegraphs or countgraphs report the total number of unique k-mers. #491 #609 #429 @mr-c

Read pairs from SRA are fully supported. Reported by @macmanes in #1027, fixed by @kdmurray91 @SherineAwad in #1173 #1088

Of interest to users:

Added `Hashtable::get_kmers()`, `get_kmer_hashes()`, and `get_kmer_counts()` with corresponding CPython functions. #1047 #1049 @ctb

The `DEFAULT_DESIRED_COVERAGE` for `normalize-by-median.py` is now 20. #1073 #1081 @ctb

FIFOs are no longer seen as empty. #1147 #1163 @bocajnotnef

When the k-size is requested to be larger than 32 (which is unsupported) a helpful error message is reported. #1094 #1050 @ctb

We try to report more helpfully during errors, such as suggesting the `--force` option when outputs files already exist. #1162 #1170 @bocajnotnef

There is a paper related to `trim-low-abund.py`: "Crossing the streams: a framework for streaming analysis of short DNA sequencing reads" and it has been added to the CITATION file and program output. #1180 #1130 @ctb

We have dropped support for Python 2.6 #1009 #1180 @ctb

Our user documentation got a bit out of date and has been updated. #1156 #1247 @bocajnotnef @mr-c #1104 @kdmurray91 #1267 @ctb Links to lists of publications that use khmer have been added #1063 #1222 @mr-c The help text from the scripts has also had a thorough cleanup for formatting. #1268 @mr-c

`fastq-to-fasta.py`'s `--n_keep` option has incorrect help text. We now point out that all reads with Ns will be dropped by default unless this option is supplied. #657 #814 #1208 @ACharbonneau @bocajnotnef

We've updated the URL to the '88m-reads.fa.gz' file. #1242 #1269 @mr-c

@camillescott designed and implemented an optimization for `normalize-by-median.py` #862

`abundance-dist.py` can now be used without counts over 255 with `--no-bigcount`. #1067 #909 @drtamermansour @bocajnotnef Its input file requirement can no longer be overridden #1201 #1202 @bocajnotnef

khmer v2.0 will be released as a package for the Debian GNU/Linux operating system. Big thanks to @kdmurray91 for his assistance. #1148 #1240 The C++ library, now named liboxli, will have its own package as well.

`sandbox/multi-rename.py` now wraps long FASTA sequences at 80 columns. #450 #1136 @SherineAwad

Of interest to developers:

The khmer project is now a Python 3 codebase with backwards compatibility to Python 2.7. Huge credit to @luizirber #978 #922 #1045 #1066 #1089 #1157 #1191 #1108 Many developer impacting changes including the file `khmer/\_khmermodule.cc` is now `khmer/\_khmer.cc`. #169 #904

@camillescott did an extensive refactor of the C++ graph traversal code which removed a considerable amount of redundant code and will be very useful for future work. #1231 #1080

We now use some and allow all C++11 features in the codebase. #598 #1122 @mr-c

`normalize-by-median.py` was extensively refactored. #1006 #1010 #1057 #1039 #1135 #1182 @bocajnotnef @ctb @camillescott

The CPython glue was refactored so that CountingHash and Hashbits inherit from Hashtable. #1044 @ctb

The tests no longer stop on the first failed test. #1124 #1134 @ctb and some noisy tests were silenced #1125 #1137 @bocajnotnef

The `check_space()` calls were cleaned up. #1167 #1166 #1170 #993

Developer docs have been expanded #737 #1184 @bocajnotnef #1083 #1282 @ctb @mr-c #1269

A lot of code was deleted: TRACE related code in #274 #1180 @ctb `hashtable_collect_high_abundance_kmers` in #1142 #1044 @ctb `lib/ht-diff.cc`, `lib/test-HashTables.cc`, `lib/test-Parser.cc` #1144, @mr-c `bink.ipynb`, `lib/graphtest.cc`, `lib/primes.hh` #1289 @mr-c

@bocajnotnef deleted more unused code and added new tests elsewhere to increase testing coverage in #1236. @mr-c had his own go in #1279

cppcheck installation for OSX has been documented #777 #952 #945 @elmbeech

ccache and git-merge-changelog has been documented for Linux users #610 #1122 #614 @mr-c

The graphalign parameters can be saved/loaded from disk. In addition the `align_forward` method has been introduced. #755 #750 @mr-c @ctb

`labelhash` is now known as `graphlabels` #1032 #1209 @mr-c It is also now a 'friend' of Hashtable and one can make either a nodegraph or countgraph version. These graphlabels can now be saved & loaded from disk. #1021 @ctb

Spelling is hard; we've added instructions on how to run codespell to the developer docs. #890 #1203 @bocajnotnef

A redundant and contradictory named test has been removed. Reported by @jgluck in #662 fixed by @bocajnotnef in #1220 @SherineAwad contributed some additional tests #809 #615.

The new oxli command, while disabled in the v2.0 release, has been added to all the QA makefile targets as we continue to refactor the codebase. #1199 #1218 @bocajnotnef

The CPython code was audited to ensure that all possible C++ exceptions were caught and dealt with. The exception hierarchy was also simplified #1016 #1015 #1017 #1151 @kdmurray91 @mr-c

`get_kadian_count` has been removed. #1034 #1194 @ctb

We use argparse's `metavars` to aid with autogenerated documentation for the scripts. This has been documented in the dev docs. #620 #1222 @mr-c

Sometimes one makes a lot of commits while refining a feature or pull request. We've documented a field-tested way to turn a pile of commits into a single commit without the pain of `git rebase`. #1013 #660 #1222 @mr-c

We use Coverity to test for various issues with our C++ code. The Makefile target has been updated for changes on their side. #1007 #1222 @mr-c

There is a new `update()` function to merge two nodegraphs of the same size and ksize. #1051 @ctb

Despite the checklist, formatting errors still occur. We must be vigilant! #1075 @luizirber

There is a new `filter_on_median` function. #862 #1077 @camillescott

There are new scripts in the `sandbox/` which output k-mer counts: sandbox/{count-kmers.py,count-kmers-single.py}. #983 @ctb

A large effort to make the codebase 'pylint clean' has begun with #1175 @bocajnotnef Likewise the cpychecker tool was re-run on the CPython code and issues found there were addressed #1196 @mr-c

As repeatedly promised, we've updated our list of contributors to include everyone with a commit in git. #1023 @mr-c

`thread_utils.is_pair()` has been dropped in favor of `utils.check_is_pair()` #1284 @mr-c

The Doxygen produced documentation is improving. The location of included headers is now autodetected for Doxygen and cppcheck.

Known issues:

`load-graph.py` in multithreaded mode will find slightly different number of unique kmers. This is being investigated in #1248

Contributors

@ctb, @bocajnotnef, @mr-c, @luizirber, @kdmurray91, @SherineAwad, @camillescott, ‡@ACharbonneau

‡ Indicates new contributors

Issue reporters

@jgluck, @ACharbonneau, @macmanes

khmer v2.1 release notes

We are pleased to announce release version 2.1 of khmer! This release includes several new features, bug fixes, and internal changes. CHANGELOG.md includes a complete description of changes made since the previous release. A concise summary of the most relevant changes is provided below.

The latest version of the khmer documentation is available at https://khmer.readthedocs.org/en/v2.1/.

Items of note

New features

- New storage class using half a byte per entry. Exposed as SmallCounttable and SmallCountgraph.
- New Counttable, SmallCounttable, and Nodetable classes to support non-reversible hashing functionality and k > 32.
- Support for human-friendly memory requests (2G) in addition to the previous style of requests (2000000000 or 2e9).
- Support for variable-coverage trimming in the `filter-abund-single.py` script.
- khmer package version now included in `.info` files.
- Several simple examples of the Python API and the C++ API in `examples/python-api` and `examples/c++-api`, respectively.
- Support for assembling directly from k-mer graphs, and a new JunctionCountAssembler class.
- New sandbox script `extract-compact-dbg.py` for computing a compact de Bruijn graph from sequence data.

Bug fixes

- Streaming gzip-compressed output from scripts now works correctly.
- The `load-graph.py` script now calculates required graph space correctly.
- The `broken_paired_reader` no longer drops short reads when `require_paired` is set.

Other changes

- Command-line options `-x` and `-N` now suppressed by default in script help messages.
- Renamed core data structures: CountingHash –> Countgraph, Hashbits –> Nodegraph.
- Replaced the `IParser` and `FastxParser` classes with a single `ReadParser` class. Different input formats are supported by templating `ReadParser` with a reader class.
- Renamed `consume_fasta` and related functions to `consume_seqfile`, with support for reading sequences from additional formats pending.

Known issues:

- `load-graph.py` in multithreaded mode will find slightly different number of unique kmers. This is being investigated in https://github.com/dib-lab/khmer/issues/1248

- Any scripts that consume FASTA or FASTQ data are unresponsive to attempts to terminate the program with `ctrl-c`. Eventually khmer should handle this properly, but for now it should be possible to halt a script using `ctrl-\`. This is being tracked in https://github.com/dib-lab/khmer/issues/1618.

Contributors

‡@aaliyari, @ctb, ‡@caitsydney, @camillescott, @standage, @kdmurray91, @ljcohen, @luizirber, @mr-c, ‡@NBKingsley, ‡@ryneches, ‡@rcs333, ‡@satta, ‡@shannonekj, ‡@betatim

‡ Indicates new contributors

The khmer user documentation

Contents:

Installing and running khmer

You'll need a 64-bit operating system, internet access, and Python 2.7.x OR Python 3.3 or greater.

If you are running khmer in a HPC environment or for other reasons do not have root access try creating a virtual environment as described in the OS X instructions below (even if you are on linux) and then installing khmer in that newly created virtual environment.

Build requirements

OS X

1. From a terminal download the virtualenv package and create a virtual environment with it:

```
curl -O https://pypi.python.org/packages/source/v/virtualenv/virtualenv-1.11.6.
↪tar.gz
tar xzf virtualenv*
cd virtualenv-*; python2.7 virtualenv.py ../khmerEnv; cd ..
source khmerEnv/bin/activate
```

2. Go to *Installing khmer inside the virtualenv* to install khmer itself.

Linux

1. Install the python development environment, virtualenv, pip, gcc, and g++.
   - On recent Debian and Ubuntu this can be done with:

```
sudo apt-get install python2.7-dev python-virtualenv python-pip gcc \
      g++
```

   - For RHEL6:

```
sudo yum install -y python-devel python-pip git gcc gcc-c++ make
sudo pip install virtualenv
```

2. Create a virtualenv and activate it:

```
cd a/writable/directory/
python2.7 -m virtualenv khmerEnv
source khmerEnv/bin/activate
```

Linux users without root access can try the OS X instructions above.

Installing khmer inside the virtualenv

1. Use pip to download, build, and install khmer and its dependencies:

```
pip2 install khmer
```

2. The scripts are now in the `env/bin` directory and ready for your use. You can directly use them by name, see *khmer's command-line interface*.
3. When returning to khmer after installing it you will need to reactivate the virtualenv first:

```
source khmerEnv/bin/activate
```

A few examples

See the examples directory for complete examples.

STAMPS data set

The 'stamps' data set is a fake metagenome-like data set containing two species, mixed at a 10:1 ratio. The source genomes are in data/stamps-genomes.fa. The reads file is in data/stamps-reads.fa.gz, and consists of 100-base reads with a 1% error rate.

The example shows how to construct k-mer abundance histograms, as well as the effect of digital normalization and partitioning on the k-mer abundance distribution.

See the script for running everything and the IPython Notebook.

For an overall discussion and some slides to explain what's going on, visit the Web site for a 2013 HMP metagenome assembly webinar that Titus Brown gave.

An assembly handbook for khmer - rough draft

date 2012-11-2

An increasing number of people are asking about using our assembly approaches for things that we haven't yet written (or posted) papers about. Moreover, our assembly strategies themselves are also under constant evolution as we do more research and find ever-wider applicability of our approaches.

Note, this is modified from Titus' blog post, here – go check the bottom of that for comments.

Authors

This handbook distills the cumulative expertise of Adina Howe, Titus Brown, Erich Schwarz, Jason Pell, Camille Scott, Elijah Lowe, Kanchan Pavangadkar, Likit Preeyanon, and others.

Introduction

khmer is really focused on short read data, and, more specifically, Illumina, because that's where we have a too-much-data problem. However, a lot of the prescriptions below can be adapted to longer read technologies such as 454 and Ion Torrent without much effort.

Don't try to use our k-mer approaches with PacBio – the error rate is too high.

There are many blog posts about this stuff on Titus Brown's blog. We will try to link them in where appropriate.

Preparing your sequences

Do all the quality filtering, trimming, etc. that you think you should do.

The khmer tools work "out of the box" on interleaved paired-end data.

All of our scripts will take in .fq or .fastq files as FASTQ, and all other files as FASTA. gzip files are always accepted. Let us know if not; that's a bug!

Genome assembly, including MDA samples and highly polymorphic genomes

1. Apply digital normalization as follows.

Broadly, normalize each insert library separately, in the following way:

For high-coverage libraries (> ~50x), do three-pass digital normalization: run **normalize-by-median.py** with *--cutoff=20* and then run **filter-abund.py** with *--cutoff=2*. Now split out the remaining paired-end/interleaved and single-end reads using **extract-paired-reads.py**, and run **normalize-by-median.py** on the paired-end and single-end files (using *--unpaired-reads*) with *--cutoff=5*.

For low-coverage libraries (< 50x) do single-pass digital normalization: run **normalize-by-median.py** to *--cutoff=10*.

2. Extract any remaining paired-end reads and lump remaining orphan reads into singletons using **extract-paired-reads.py**
3. Then assemble as normal, with appropriate insert size specs etc. for the paired end reads.

You can read about this process in the digital normalization paper.

mRNAseq assembly

1. Apply single-pass digital normalization. Run **normalize-by-median.py** with *--cutoff=20*.
2. Extract any remaining paired-end reads and lump remaining orphan reads into singletons using **extract-paired-reads.py**
3. Then assemble as normal, with appropriate insert size specs etc. for the paired end reads.

You can read about this process in the digital normalization paper.

Metagenome assembly

1. Apply single-pass digital normalization. Run **normalize-by-median.py** with *--cutoff=20* (we've also found *--cutoff=10* works fine).

2. Run `sandbox/filter-below-abund.py` with `--cutoff=50` (if you ran **`normalize-by-median.py`** with *`--cutoff=10`*) or wiht `--cutoff=100` if you ran **`normalize-by-median.py`** with *`--cutoff=20`*)
3. Partition reads with **`load-graph.py`**, etc. etc.
4. Assemble groups as normal, extracting paired-end reads and lumping remaining orphan reads into singletons using **`extract-paired-reads.py`**.

(We actually use Velvet at this point, but there should be no harm in using a metagenome assembler such as MetaVelvet or MetaIDBA or SOAPdenovo.)

Read more about this in the partitioning paper. We have some upcoming papers on partitioning and metagenome assembly, too; we'll link those in when we can.

Metatranscriptome assembly

(Not tested by us!)

1. Apply single-pass digital normalization by running **`normalize-by-median.py`** with *`--cutoff=20`*.
2. Extract any remaining paired-end reads and lump remaining orphan reads into singletons using **`extract-paired-reads.py`**.
3. Then assemble with a genome or metagenome assembler, *not* an mRNAseq assembler. Use appropriate insert size specs etc. for the paired end reads.

Preprocessing Illumina for other applications

(Not tested by us!)

Others have told us that you can apply digital normalization to Illumina data prior to using Illumina for RNA scaffolding or error correcting PacBio reads.

Our suggestion for this, based on no evidence whatsoever, is to run **`normalize-by-median.py`** with *`--cutoff=20`* on the Illumina data.

Quantifying mRNAseq or metagenomes assembled with digital normalization

For now, khmer only deals with assembly! So: assemble. Then, go back to your original, unnormalized reads, and map those to your assembly with e.g. bowtie. Then count as you normally would).

Philosophy of digital normalization

The basic philosophy of digital normalization is "load your most valuable reads first." Diginorm gets rid of redundancy iteratively, so you are more likely to retain the first reads fed in; this means you should load in paired end reads, or longer reads, first.

Iterative and independent normalization

You can use *`--loadgraph`* and *`--savegraph`* to do iterative normalizations on multiple files in multiple steps. For example, break

```
normalize-by-median.py [ ... ] file1.fa file2.fa file3.fa
```

into multiple steps like so:

```
normalize-by-median.py [ ... ] --savegraph file1.ct file1.fa
normalize-by-median.py [ ... ] --loadgraph file1.ct --savegraph file2.ct file2.fa
normalize-by-median.py [ ... ] --loadgraph file2.ct --savegraph file3.ct file3.fa
```

The results should be identical!

If you want to independently normalize multiple files for speed reasons, go ahead. Just remember to do a combined normalization at the end. For example, instead of

```
normalize-by-median.py [ ... ] file1.fa file2.fa file3.fa
```

you could do

```
normalize-by-median.py [ ... ] file1.fa
normalize-by-median.py [ ... ] file2.fa
normalize-by-median.py [ ... ] file3.fa
```

and then do a final

```
normalize-by-median.py [ ... ] file1.fa.keep file2.fa.keep file3.fa.keep
```

The results will not be identical, but should not differ significantly. The multipass approach will take more total time but may end up being faster walltime because you can execute the independent normalizations on multiple computers.

For a cleverer approach that we will someday implement, read the Beachcomber's Dilemma.

khmer's command-line interface

The simplest way to use khmer's functionality is through the command line scripts, located in the scripts/ directory of the khmer distribution. Below is our documentation for these scripts. Note that all scripts can be given `-h`/`--help` which will print out a list of arguments taken by that script.

Scripts that use k-mer counting tables or k-mer graphs take an *-M* parameter, which sets the maximum memory usage in bytes. This should generally be set as high as possible; see *Setting khmer memory usage* for more information.

1. *k-mer counting and abundance filtering*
2. *Partitioning*
3. *Digital normalization*
4. *Read handling: interleaving, splitting, etc.*

Note: Almost all scripts take in either FASTA and FASTQ format, and output the same.

Gzip and bzip2 compressed files are detected automatically.

---

k-mer counting and abundance filtering

load-into-counting.py

Build a k-mer countgraph from the given sequences.

usage: load-into-counting.py [–version] [–info] [-h] [-k KSIZE] [-U UNIQUE_KMERS] [–fp-rate FP_RATE] [-M MAX_MEMORY_USAGE] [–small-count] [-T THREADS] [-b] [-s FORMAT] [-f] [-q] output_countgraph_filename input_sequence_filename [input_sequence_filename ...]

output_countgraph_filename
The name of the file to write the k-mer countgraph to.

input_sequence_filename
The names of one or more FAST[AQ] input sequence files.

--version
show program's version number and exit

--info
print citation information

-h, --help
show this help message and exit

-k <ksize>, **--ksize** <ksize>
k-mer size to use

--n_tables <n_tables>, **-N** <n_tables>
==SUPPRESS==

-U <unique_kmers>, **--unique-kmers** <unique_kmers>
approximate number of unique kmers in the input set

--fp-rate <fp_rate>
Override the automatic FP rate setting for the current script

--max-tablesize <max_tablesize>, **-x** <max_tablesize>
==SUPPRESS==

-M <max_memory_usage>, **--max-memory-usage** <max_memory_usage>
maximum amount of memory to use for data structure

--small-count
Reduce memory usage by using a smaller counter for individual kmers.

-T <threads>, **--threads** <threads>
Number of simultaneous threads to execute

-b, --no-bigcount
The default behaviour is to count past 255 using bigcount. This flag turns bigcount off, limiting counts to 255.

-s {json,tsv}, **--summary-info** {json,tsv}
What format should the machine readable run summary be in? (*json* or *tsv*, disabled by default)

-f, --force
Overwrite output file if it exists

-q, --quiet

Note: with *-b*/*--no-bigcount* the output will be the exact size of the k-mer countgraph and this script will use a constant amount of memory. In exchange k-mer counts will stop at 255. The memory usage of this script with *-b* will be about 1.15x the product of the *-x* and *-N* numbers.

Example:

```
load-into-counting.py -k 20 -x 5e7 out data/100k-filtered.fa
```

Multiple threads can be used to accelerate the process, if you have extra cores to spare.

Example:

```
load-into-counting.py -k 20 -x 5e7 -T 4 out data/100k-filtered.fa
```

abundance-dist.py

Calculate abundance distribution of the k-mers in the sequence file using a pre-made k-mer countgraph.

usage: abundance-dist.py [–version] [–info] [-h] [-z] [-s] [-b] [-f] [-q] input_count_graph_filename input_sequence_filename output_histogram_filename

input_count_graph_filename
The name of the input k-mer countgraph file.

input_sequence_filename
The name of the input FAST[AQ] sequence file.

output_histogram_filename
The columns are: (1) k-mer abundance, (2) k-mer count, (3) cumulative count, (4) fraction of total distinct k-mers.

--version
show program's version number and exit

--info
print citation information

-h, --help
show this help message and exit

-z, --no-zero
Do not output zero-count bins

-s, --squash
Overwrite existing output_histogram_filename

-b, --no-bigcount
Do not count k-mers past 255

-f, --force
Continue even if specified input files do not exist or are empty.

-q, --quiet

Example:

```
load-into-counting.py -x 1e7 -N 2 -k 17 counts \
        tests/test-data/test-abund-read-2.fa
abundance-dist.py counts tests/test-data/test-abund-read-2.fa test-dist
```

abundance-dist-single.py

Calculate the abundance distribution of k-mers from a single sequence file.

usage: abundance-dist-single.py [–version] [–info] [-h] [-k KSIZE] [-U UNIQUE_KMERS] [–fp-rate FP_RATE] [-M MAX_MEMORY_USAGE] [–small-count] [-T THREADS] [-z] [-b] [-s] [–savegraph filename] [-f] [-q] input_sequence_filename output_histogram_filename

input_sequence_filename
The name of the input FAST[AQ] sequence file.

output_histogram_filename
The name of the output histogram file. The columns are: (1) k-mer abundance, (2) k-mer count, (3) cumulative count, (4) fraction of total distinct k-mers.

--version
show program's version number and exit

--info
print citation information

-h, --help
show this help message and exit

-k <ksize>, **--ksize** <ksize>
k-mer size to use

--n_tables <n_tables>, **-N** <n_tables>
==SUPPRESS==

-U <unique_kmers>, **--unique-kmers** <unique_kmers>
approximate number of unique kmers in the input set

--fp-rate <fp_rate>
Override the automatic FP rate setting for the current script

--max-tablesize <max_tablesize>, **-x** <max_tablesize>
==SUPPRESS==

-M <max_memory_usage>, **--max-memory-usage** <max_memory_usage>
maximum amount of memory to use for data structure

--small-count
Reduce memory usage by using a smaller counter for individual kmers.

-T <threads>, **--threads** <threads>
Number of simultaneous threads to execute

-z, --no-zero
Do not output zero-count bins

-b, --no-bigcount
Do not count k-mers past 255

-s, --squash
Overwrite output file if it exists

--savegraph <filename>
Save the k-mer countgraph to the specified filename.

-f, --force
Override sanity checks

-q, --quiet

Note that with *-b*/*--no-bigcount* this script is constant memory; in exchange, k-mer counts will stop at 255. The memory usage of this script with *-b* will be about 1.15x the product of the *-x* and *-N* numbers.

To count k-mers in multiple files use **load_into_counting.py** and **abundance_dist.py**.

Example:

```
abundance-dist-single.py -x 1e7 -N 2 -k 17 \
        tests/test-data/test-abund-read-2.fa test-dist
```

filter-abund.py

Trim sequences at a minimum k-mer abundance.

usage: filter-abund.py [–version] [–info] [-h] [-T THREADS] [-C CUTOFF] [-V] [-Z NORMALIZE_TO] [-o optional_output_filename] [-f] [-q] [–gzip | –bzip] input_count_graph_filename input_sequence_filename [input_sequence_filename ...]

input_count_graph_filename
The input k-mer countgraph filename

input_sequence_filename
Input FAST[AQ] sequence filename

--version
show program's version number and exit

--info
print citation information

-h, --help
show this help message and exit

-T <threads>, **--threads** <threads>
Number of simultaneous threads to execute

-C <cutoff>, **--cutoff** <cutoff>
Trim at k-mers below this abundance.

-V, --variable-coverage
Only trim low-abundance k-mers from sequences that have high coverage.

-Z <normalize_to>, **--normalize-to** <normalize_to>
Base the variable-coverage cutoff on this median k-mer abundance.

-o <optional_output_filename>, **--output** <optional_output_filename>
Output the trimmed sequences into a single file with the given filename instead of creating a new file for each input file.

-f, --force
Overwrite output file if it exists

-q, --quiet

--gzip
Compress output using gzip

--bzip
Compress output using bzip2

Trimmed sequences will be placed in `${input_sequence_filename}.abundfilt` for each input sequence file. If the input sequences are from RNAseq or metagenome sequencing then *--variable-coverage* should be used.

Example:

```
load-into-counting.py -k 20 -x 5e7 countgraph data/100k-filtered.fa
filter-abund.py -C 2 countgraph data/100k-filtered.fa
```

filter-abund-single.py

Trims sequences at a minimum k-mer abundance (in memory version).

usage: filter-abund-single.py [–version] [–info] [-h] [-k KSIZE] [-U UNIQUE_KMERS] [–fp-rate FP_RATE] [-M MAX_MEMORY_USAGE] [–small-count] [-T THREADS] [-C CUTOFF] [-V] [-Z NORMALIZE_TO] [–savegraph filename] [-o optional_output_filename] [-f] [-q] [–gzip | –bzip] input_sequence_filename

input_sequence_filename
FAST[AQ] sequence file to trim

--version
show program's version number and exit

--info
print citation information

-h, --help
show this help message and exit

-k <ksize>, **--ksize** <ksize>
k-mer size to use

--n_tables <n_tables>, **-N** <n_tables>
==SUPPRESS==

-U <unique_kmers>, **--unique-kmers** <unique_kmers>
approximate number of unique kmers in the input set

--fp-rate <fp_rate>
Override the automatic FP rate setting for the current script

--max-tablesize <max_tablesize>, **-x** <max_tablesize>
==SUPPRESS==

-M <max_memory_usage>, **--max-memory-usage** <max_memory_usage>
maximum amount of memory to use for data structure

--small-count
Reduce memory usage by using a smaller counter for individual kmers.

-T <threads>, **--threads** <threads>
Number of simultaneous threads to execute

-C <cutoff>, **--cutoff** <cutoff>
Trim at k-mers below this abundance.

-V, --variable-coverage
Only trim low-abundance k-mers from sequences that have high coverage.

-Z <normalize_to>, **--normalize-to** <normalize_to>
Base the variable-coverage cutoff on this median k-mer abundance.

--savegraph <filename>
If present, the name of the file to save the k-mer countgraph to

-o <optional_output_filename>, **--outfile** <optional_output_filename>
Override default output filename and output trimmed sequences into a file with the given filename.

-f, --force
Overwrite output file if it exists

-q, --quiet

--gzip
Compress output using gzip

--bzip
Compress output using bzip2

Trimmed sequences will be placed in `${input_sequence_filename}.abundfilt`.

This script is constant memory.

To trim reads based on k-mer abundance across multiple files, use **load-into-counting.py** and **filter-abund.py**.

Example:

```
filter-abund-single.py -k 20 -x 5e7 -C 2 data/100k-filtered.fa
```

trim-low-abund.py

Trim low-abundance k-mers using a streaming algorithm.

usage: trim-low-abund.py [–version] [–info] [-h] [-k KSIZE] [-U UNIQUE_KMERS] [–fp-rate FP_RATE] [-M MAX_MEMORY_USAGE] [–small-count] [-C CUTOFF] [-Z TRIM_AT_COVERAGE] [-o output_filename] [-V] [-l filename] [-s filename] [-q] [–summary-info FORMAT] [–force] [–ignore-pairs] [-T TEMPDIR] [–gzip | –bzip] [–diginorm] [–diginorm-coverage DIGINORM_COVERAGE] [–single-pass] input_filenames [input_filenames ...]

input_filenames

--version
show program's version number and exit

--info
print citation information

-h, --help
show this help message and exit

-k <ksize>, **--ksize** <ksize>
k-mer size to use

--n_tables <n_tables>, **-N** <n_tables>
==SUPPRESS==

-U <unique_kmers>, **--unique-kmers** <unique_kmers>
approximate number of unique kmers in the input set

--fp-rate <fp_rate>
Override the automatic FP rate setting for the current script

--max-tablesize <max_tablesize>, **-x** <max_tablesize>
==SUPPRESS==

-M <max_memory_usage>, **--max-memory-usage** <max_memory_usage>
maximum amount of memory to use for data structure

--small-count
Reduce memory usage by using a smaller counter for individual kmers.

-C <cutoff>, **--cutoff** <cutoff>
remove k-mers below this abundance

-Z <trim_at_coverage>, **--trim-at-coverage** <trim_at_coverage>, **--normalize-to** <trim_at_cover
trim reads when entire read above this coverage

-o <output_filename>, **--output** <output_filename>
only output a single file with the specified filename; use a single dash "-" to specify that output should go to STDOUT (the terminal)

-V, **--variable-coverage**
Only trim low-abundance k-mers from sequences that have high coverage.

-l <filename>, **--loadgraph** <filename>
load a precomputed k-mer graph from disk

-s <filename>, **--savegraph** <filename>
save the k-mer countgraph to disk after allreads are loaded.

-q, **--quiet**

--summary-info {json,tsv}
What format should the machine readable run summary be in? (*json* or *tsv*, disabled by default)

--force

--ignore-pairs
treat all reads as if they were singletons

-T <tempdir>, **--tempdir** <tempdir>
Set location of temporary directory for second pass

--gzip
Compress output using gzip

--bzip
Compress output using bzip2

--diginorm
Eliminate high-coverage reads altogether (digital normalization).

--diginorm-coverage <diginorm_coverage>
Coverage threshold for –diginorm

--single-pass
Do not do a second pass across the low coverage data

The output is one file for each input file, <input file>.abundtrim, placed in the current directory. This output contains the input sequences trimmed at low-abundance k-mers.

The *-V*/*--variable-coverage* parameter will, if specified, prevent elimination of low-abundance reads by only trimming low-abundance k-mers from high-abundance reads; use this for non-genomic data sets that may have variable coverage.

Note that the output reads will not necessarily be in the same order as the reads in the input files; if this is an important consideration, use **load-into-counting.py** and **filter-abund.py**. However, read pairs will be kept together, in "broken-paired" format; you can use **extract-paired-reads.py** to extract read pairs and orphans.

Example:

```
trim-low-abund.py -x 5e7 -k 20 -C 2 data/100k-filtered.fa
```

count-median.py

Count k-mers summary stats for sequences

usage: count-median.py [–version] [–info] [-h] [-f] input_count_graph_filename input_sequence_filename output_summary_filename

input_count_graph_filename
input k-mer countgraph filename

input_sequence_filename
input FAST[AQ] sequence filename

output_summary_filename
output summary filename

--version
show program's version number and exit

--info
print citation information

-h, --help
show this help message and exit

-f, --force
Overwrite output file if it exists

Count the median/avg k-mer abundance for each sequence in the input file, based on the k-mer counts in the given k-mer countgraph. Can be used to estimate expression levels (mRNAseq) or coverage (genomic/metagenomic).

The output file contains sequence id, median, average, stddev, and seq length, in comma-separated value (CSV) format.

Example:

```
load-into-counting.py counts tests/test-data/test-reads.fq.gz
count-median.py counts tests/test-data/test-reads.fq.gz medians.txt
```

NOTE: All 'N's in the input sequences are converted to 'A's.

unique-kmers.py

Estimate number of unique k-mers, with precision <= ERROR_RATE.

usage: unique-kmers.py [–version] [–info] [-h] [-q] [-k KSIZE] [-e ERROR_RATE] [-R filename] [-S] [–diagnostics] input_sequence_filename [input_sequence_filename ...]

input_sequence_filename
Input FAST[AQ] sequence filename(s).

--version
show program's version number and exit

--info
print citation information

-h, --help
show this help message and exit

-q, --quiet

-k <ksize>, **--ksize** <ksize>
k-mer size to use

-e <error_rate>, **--error-rate** <error_rate>
Acceptable error rate

-R <filename>, **--report** <filename>
generate informational report and write to filename

-S, **--stream-records**
write input sequences to STDOUT

--diagnostics
print out recommended tablesize arguments and restrictions

A HyperLogLog counter is used to do cardinality estimation. Since this counter is based on a tradeoff between precision and memory consumption, the *-e*/*--error-rate* can be used to control how much memory will be used. In practice the memory footprint is small even at low error rates (< 0.01).

-k/*--ksize* should be set to the desired k-mer size.

Informational output is sent to STDERR, but a report file can be generated with *-R*/*--report*.

--stream-records will write the sequences taken in to STDOUT. This is useful for workflows: count unique kmers in a stream, then do digital normalization.

--diagnostics will provide detailed options for tablesize and memory limitations for various false positive rates. This is useful for configuring other khmer scripts. This will be written to STDERR.

Example:

```
unique-kmers.py -k 17 tests/test-data/test-abund-read{,-2,-3}.fa
```

Example:

```
unique-kmers.py -k 17 --diagnostics tests/test-data/test-abund-read.fa
```

Example:

```
unique-kmers.py --stream-records -k 17 tests/test-data/test-reads.fa | \
normalize-by-median.py -k 17 -o normalized /dev/stdin
```

Example:

```
unique-kmers.py -R unique_count -k 30 \
tests/test-data/test-abund-read-paired.fa
```

Partitioning

do-partition.py

Load, partition, and annotate FAST[AQ] sequences

usage: do-partition.py [–version] [–info] [-h] [-k KSIZE] [-U UNIQUE_KMERS] [–fp-rate FP_RATE] [-M MAX_MEMORY_USAGE] [-T THREADS] [-s SUBSET_SIZE] [–no-big-traverse] [–keep-subsets] [-f] graphbase input_sequence_filename [input_sequence_filename ...]

graphbase
base name for output files

input_sequence_filename
input FAST[AQ] sequence filenames

--version
show program's version number and exit

--info
print citation information

-h, --help
show this help message and exit

-k <ksize>, **--ksize** <ksize>
k-mer size to use

--n_tables <n_tables>, **-N** <n_tables>
==SUPPRESS==

-U <unique_kmers>, **--unique-kmers** <unique_kmers>
approximate number of unique kmers in the input set

--fp-rate <fp_rate>
Override the automatic FP rate setting for the current script

--max-tablesize <max_tablesize>, **-x** <max_tablesize>
==SUPPRESS==

-M <max_memory_usage>, **--max-memory-usage** <max_memory_usage>
maximum amount of memory to use for data structure

-T <threads>, **--threads** <threads>
Number of simultaneous threads to execute

-s <subset_size>, **--subset-size** <subset_size>
Set subset size (usually 1e5-1e6 is good)

--no-big-traverse
Truncate graph joins at big traversals

--keep-subsets
Keep individual subsets

-f, --force
Overwrite output file if it exists

Load in a set of sequences, partition them, merge the partitions, and annotate the original sequences files with the partition information.

This script combines the functionality of **load-graph.py**, **partition-graph.py**, **merge-partitions.py**, and **annotate-partitions.py** into one script. This is convenient but should probably not be used for large data sets, because **do-partition.py** doesn't provide save/resume functionality.

Example:

```
do-partition.py -k 20 example tests/test-data/random-20-a.fa
```

load-graph.py

Load sequences into the compressible graph format plus optional tagset.

usage: load-graph.py [–version] [–info] [-h] [-k KSIZE] [-U UNIQUE_KMERS] [–fp-rate FP_RATE] [-M MAX_MEMORY_USAGE] [-T THREADS] [–no-build-tagset] [-f] output_nodegraph_filename input_sequence_filename [input_sequence_filename ...]

output_nodegraph_filename
output k-mer nodegraph filename.

input_sequence_filename
input FAST[AQ] sequence filename

--version
show program's version number and exit

--info
print citation information

-h, --help
show this help message and exit

-k <ksize>, **--ksize** <ksize>
k-mer size to use

--n_tables <n_tables>, **-N** <n_tables>
==SUPPRESS==

-U <unique_kmers>, **--unique-kmers** <unique_kmers>
approximate number of unique kmers in the input set

--fp-rate <fp_rate>
Override the automatic FP rate setting for the current script

--max-tablesize <max_tablesize>, **-x** <max_tablesize>
==SUPPRESS==

-M <max_memory_usage>, **--max-memory-usage** <max_memory_usage>
maximum amount of memory to use for data structure

-T <threads>, **--threads** <threads>
Number of simultaneous threads to execute

--no-build-tagset, -n
Do NOT construct tagset while loading sequences

-f, --force
Overwrite output file if it exists

See **extract-partitions.py** for a complete workflow.

partition-graph.py

Partition a sequence graph based upon waypoint connectivity

usage: partition-graph.py [–version] [–info] [-h] [-S filename] [-s SUBSET_SIZE] [–no-big-traverse] [-f] [-T THREADS] basename

basename
basename of the input k-mer nodegraph + tagset files

--version
show program's version number and exit

--info
print citation information

-h, --help
show this help message and exit

-S <filename>, **--stoptags** <filename>
Use stoptags in this file during partitioning

-s <subset_size>, **--subset-size** <subset_size>
Set subset size (usually 1e5-1e6 is good)

--no-big-traverse
Truncate graph joins at big traversals

-f, --force
Overwrite output file if it exists

-T <threads>, **--threads** <threads>
Number of simultaneous threads to execute

The resulting partition maps are saved as `${basename}.subset.#.pmap` files.

See 'Artifact removal' to understand the stoptags argument.

merge-partition.py

Merge partition map '.pmap' files.

usage: merge-partition.py [–version] [–info] [-h] [-k KSIZE] [–keep-subsets] [-f] graphbase

graphbase
basename for input and output files

--version
show program's version number and exit

--info
print citation information

-h, --help
show this help message and exit

-k <ksize>, **--ksize** <ksize>
k-mer size (default: 32)

--keep-subsets
Keep individual subsets (default: False)

-f, --force
Overwrite output file if it exists

Take the `${graphbase}.subset.#.pmap` files and merge them all into a single `${graphbase}.pmap.merged` file for **annotate-partitions.py** to use.

annotate-partitions.py

Annotate sequences with partition IDs.

usage: annotate-partitions.py [–version] [–info] [-h] [-k KSIZE] [-f] graphbase input_sequence_filename [input_sequence_filename ...]

graphbase
basename for input and output files

input_sequence_filename
input FAST[AQ] sequences to annotate.

--version
show program's version number and exit

--info
print citation information

-h, --help
show this help message and exit

-k <ksize>, **--ksize** <ksize>
k-mer size (default: 32)

-f, --force
Overwrite output file if it exists

Load in a partitionmap (generally produced by **partition-graph.py** or **merge-partitions.py**) and annotate the sequences in the given files with their partition IDs. Use **extract-partitions.py** to extract sequences into separate group files.

Example (results will be in `random-20-a.fa.part`):

```
load-graph.py -k 20 example tests/test-data/random-20-a.fa
partition-graph.py example
merge-partitions.py -k 20 example
annotate-partitions.py -k 20 example tests/test-data/random-20-a.fa
```

extract-partitions.py

Separate sequences that are annotated with partitions into grouped files.

usage: extract-partitions.py [–version] [–info] [-h] [-X MAX_SIZE] [-m MIN_PART_SIZE] [-n] [-U] [-f] [–gzip | –bzip] output_filename_prefix input_partition_filename [input_partition_filename ...]

output_filename_prefix

input_partition_filename

--version
show program's version number and exit

--info
print citation information

-h, --help
show this help message and exit

-X <max_size>, **--max-size** <max_size>
Max group size (n sequences)

-m <min_part_size>, **--min-partition-size** <min_part_size>
Minimum partition size worth keeping

-n, --no-output-groups
Do not actually output groups files.

-U, --output-unassigned
Output unassigned sequences, too

-f, --force
Overwrite output file if it exists

--gzip
Compress output using gzip

--bzip
Compress output using bzip2

Example (results will be in `example.group0000.fa`):

```
load-graph.py -k 20 example tests/test-data/random-20-a.fa
partition-graph.py example
merge-partitions.py -k 20 example
annotate-partitions.py -k 20 example tests/test-data/random-20-a.fa
extract-partitions.py example random-20-a.fa.part
```

(**extract-partitions.py** will produce a partition size distribution in <base>.dist. The columns are: (1) number of reads, (2) count of partitions with n reads, (3) cumulative sum of partitions, (4) cumulative sum of reads.)

Artifact removal

The following scripts are specialized scripts for finding and removing highly-connected k-mers (HCKs). See *Partitioning large data sets (50m+ reads)*.

make-initial-stoptags.py

Find an initial set of highly connected k-mers.

usage: make-initial-stoptags.py [–version] [–info] [-h] [-k KSIZE] [-U UNIQUE_KMERS] [–fp-rate FP_RATE] [-M MAX_MEMORY_USAGE] [–small-count] [-s SUBSET_SIZE] [-S filename] [-f] graphbase

graphbase
basename for input and output filenames

--version
show program's version number and exit

--info
print citation information

-h, --help
show this help message and exit

-k <ksize>, **--ksize** <ksize>
k-mer size to use

--n_tables <n_tables>, **-N** <n_tables>
==SUPPRESS==

-U <unique_kmers>, **--unique-kmers** <unique_kmers>
approximate number of unique kmers in the input set

--fp-rate <fp_rate>
Override the automatic FP rate setting for the current script

--max-tablesize <max_tablesize>, **-x** <max_tablesize>
==SUPPRESS==

-M <max_memory_usage>, **--max-memory-usage** <max_memory_usage>
maximum amount of memory to use for data structure

--small-count
Reduce memory usage by using a smaller counter for individual kmers.

-s <subset_size>, **--subset-size** <subset_size>
Set subset size (default 1e4 is prob ok)

-S <filename>, **--stoptags** <filename>
Use stoptags in this file during partitioning

-f, --force
Overwrite output file if it exists

Loads a k-mer nodegraph/tagset pair created by **load-graph.py**, and does a small set of traversals from graph waypoints; on these traversals, looks for k-mers that are repeatedly traversed in high-density regions of the graph, i.e. are highly connected. Outputs those k-mers as an initial set of stoptags, which can be fed into **partition-graph.py**, **find-knots.py**, and **filter-stoptags.py**.

The k-mer countgraph size options parameters are for a k-mer countgraph to keep track of repeatedly-traversed k-mers. The subset size option specifies the number of waypoints from which to traverse; for highly connected data sets, the default (1000) is probably ok.

find-knots.py

Find all highly connected k-mers.

usage: find-knots.py [–version] [–info] [-h] [-k KSIZE] [-U UNIQUE_KMERS] [–fp-rate FP_RATE] [-M MAX_MEMORY_USAGE] [–small-count] [-f] graphbase

graphbase
Basename for the input and output files.

--version
show program's version number and exit

--info
print citation information

-h, --help
show this help message and exit

-k <ksize>, **--ksize** <ksize>
k-mer size to use

--n_tables <n_tables>, **-N** <n_tables>
==SUPPRESS==

-U <unique_kmers>, **--unique-kmers** <unique_kmers>
approximate number of unique kmers in the input set

--fp-rate <fp_rate>
Override the automatic FP rate setting for the current script

--max-tablesize <max_tablesize>, **-x** <max_tablesize>
==SUPPRESS==

-M <max_memory_usage>, **--max-memory-usage** <max_memory_usage>
maximum amount of memory to use for data structure

--small-count
Reduce memory usage by using a smaller counter for individual kmers.

-f, --force
Continue past warnings

Load an k-mer nodegraph/tagset pair created by **load-graph.py**, and a set of pmap files created by **partition-graph.py**. Go through each pmap file, select the largest partition in each, and do the same kind of traversal as in **make-initial-stoptags.py** from each of the waypoints in that partition; this should identify all of the Highly Connected Kmers in that partition. These HCKs are output to <graphbase>.stoptags after each pmap file.

Parameter choice is reasonably important. See the pipeline in *Partitioning large data sets (50m+ reads)* for an example run.

This script is not very scalable and may blow up memory and die horribly. You should be able to use the intermediate stoptags to restart the process, and if you eliminate the already-processed pmap files, you can continue where you left off.

filter-stoptags.py

Trim sequences at stoptags.

usage: filter-stoptags.py [–version] [–info] [-h] [-k KSIZE] [-f] input_stoptags_filename input_sequence_filename [input_sequence_filename ...]

input_stoptags_filename

input_sequence_filename

--version
show program's version number and exit

--info
print citation information

-h, --help
show this help message and exit

-k <ksize>, **--ksize** <ksize>
k-mer size

-f, --force
Overwrite output file if it exists

Load stoptags in from the given *.stoptags* file and use them to trim or remove the sequences in <*file1-N*>. Trimmed sequences will be placed in <*fileN*>*.stopfilt*.

Digital normalization

normalize-by-median.py

Do digital normalization (remove mostly redundant sequences)

usage: normalize-by-median.py [–version] [–info] [-h] [-k KSIZE] [-U UNIQUE_KMERS] [–fp-rate FP_RATE] [-M MAX_MEMORY_USAGE] [–small-count] [-q] [-C CUTOFF] [-p] [–force_single] [-u unpaired_reads_filename] [-s

filename] [-R report_filename] [–report-frequency report_frequency] [-f] [-o filename] [-l filename] [–gzip | –bzip] input_sequence_filename [input_sequence_filename ...]

input_sequence_filename
Input FAST[AQ] sequence filename.

--version
show program's version number and exit

--info
print citation information

-h, --help
show this help message and exit

-k <ksize>, **--ksize** <ksize>
k-mer size to use

--n_tables <n_tables>, **-N** <n_tables>
==SUPPRESS==

-U <unique_kmers>, **--unique-kmers** <unique_kmers>
approximate number of unique kmers in the input set

--fp-rate <fp_rate>
Override the automatic FP rate setting for the current script

--max-tablesize <max_tablesize>, **-x** <max_tablesize>
==SUPPRESS==

-M <max_memory_usage>, **--max-memory-usage** <max_memory_usage>
maximum amount of memory to use for data structure

--small-count
Reduce memory usage by using a smaller counter for individual kmers.

-q, --quiet

-C <cutoff>, **--cutoff** <cutoff>
when the median k-mer coverage level is above this number the read is not kept.

-p, --paired
require that all sequences be properly paired

--force_single
treat all sequences as single-ended/unpaired

-u <unpaired_reads_filename>, **--unpaired-reads** <unpaired_reads_filename>
include a file of unpaired reads to which -p/–paired does not apply.

-s <filename>, **--savegraph** <filename>
save the k-mer countgraph to disk after all reads are loaded.

-R <report_filename>, **--report** <report_filename>
write progress report to report_filename

--report-frequency <report_frequency>
report progress every report_frequency reads

-f, --force
continue past file reading errors

-o <filename>, **--output** <filename>
only output a single file with the specified filename; use a single dash "-" to specify that output should go to STDOUT (the terminal)

-l <filename>, **--loadgraph** <filename>
load a precomputed k-mer graph from disk

--gzip
Compress output using gzip

--bzip
Compress output using bzip2

Discard sequences based on whether or not their median k-mer abundance lies above a specified cutoff. Kept sequences will be placed in <fileN>.keep.

By default, paired end reads will be considered together; if either read should be kept, both will be kept. (This keeps both reads from a fragment, and helps with retention of repeats.) Unpaired reads are treated individually.

If *-p*/*--paired* is set, then proper pairing is required and the script will exit on unpaired reads, although *--unpaired-reads* can be used to supply a file of orphan reads to be read after the paired reads.

--force_single will ignore all pairing information and treat reads individually.

With *-s*/*--savegraph*, the k-mer countgraph will be saved to the specified file after all sequences have been processed. *-l*/*--loadgraph* will load the specified k-mer countgraph before processing the specified files. Note that these graphs are are in the same format as those produced by **load-into-counting.py** and consumed by **abundance-dist.py**.

To append reads to an output file (rather than overwriting it), send output to STDOUT with *–output* - and use UNIX file redirection syntax (>>) to append to the file.

Example:

```
normalize-by-median.py -k 17 tests/test-data/test-abund-read-2.fa
```

Example:

```
normalize-by-median.py -p -k 17 \
tests/test-data/test-abund-read-paired.fa
```

Example:

```
normalize-by-median.py -p -k 17 -o - tests/test-data/paired.fq \
>> appended-output.fq
```

Example:

```
normalize-by-median.py -k 17 -f tests/test-data/test-error-reads.fq \
tests/test-data/test-fastq-reads.fq
```

Example:

```
normalize-by-median.py -k 17 -s test.ct \
tests/test-data/test-abund-read-2.fa \
tests/test-data/test-fastq-reads.fq
```

Read handling: interleaving, splitting, etc.

extract-long-sequences.py

Extract FASTQ or FASTA sequences longer than specified length (default: 200 bp).

usage: extract-long-sequences.py [–version] [–info] [-h] [-o output] [-l LENGTH] [–gzip | –bzip] input_filenames [input_filenames ...]

input_filenames
Input FAST[AQ] sequence filename.

--version
show program's version number and exit

--info
print citation information

-h, --help
show this help message and exit

-o <output>, **--output** <output>
The name of the output sequence file.

-l <length>, **--length** <length>
The minimum length of the sequence file.

--gzip
Compress output using gzip

--bzip
Compress output using bzip2

Example:

```
extract-long-sequences.py --length 10 tests/test-data/paired-mixed.fa
```

extract-paired-reads.py

Take a mixture of reads and split into pairs and orphans.

usage: extract-paired-reads.py [–version] [–info] [-h] [-d OUTPUT_DIR] [-p filename] [-s filename] [-f] [–gzip | –bzip] [infile]

infile

--version
show program's version number and exit

--info
print citation information

-h, --help
show this help message and exit

-d <output_dir>, **--output-dir** <output_dir>
Output split reads to specified directory. Creates directory if necessary

-p <filename>, **--output-paired** <filename>
Output paired reads to this file

-s <filename>, **--output-single** <filename>
Output orphaned reads to this file

-f, --force
Overwrite output file if it exists

--gzip
Compress output using gzip

--bzip
Compress output using bzip2

Many read-handling programs (assemblers, mappers, etc.) require that you give them either perfectly interleaved files, or files containing only single reads. This script takes files that were originally interleaved but where reads may have been orphaned (via error filtering, application of abundance filtering, digital normalization in non-paired mode, or partitioning) and separates the interleaved reads from the orphaned reads.

The default output is two files, <*input file*>*.pe* and <*input file*>*.se*, placed in the current directory. The .pe file contains interleaved and properly paired sequences, while the .se file contains orphan sequences.

The directory into which the interleaved and orphaned reads are output may be specified using *-d*/*--output-dir*. This directory will be created if it does not already exist.

Alternatively, you can specify the filenames directly with *-p*/*--output-paired* and *-s*/*--output-single*, which will override the *-d*/*--output-dir* option.

Example:

```
extract-paired-reads.py tests/test-data/paired.fq
```

fastq-to-fasta.py

Converts FASTQ format (.fq) files to FASTA format (.fa).

usage: fastq-to-fasta.py [–version] [–info] [-h] [-o filename] [-n] [–gzip | –bzip] input_sequence

input_sequence
The name of the input FASTQ sequence file.

--version
show program's version number and exit

--info
print citation information

-h, --help
show this help message and exit

-o <filename>**, --output** <filename>
The name of the output FASTA sequence file.

-n, --n_keep
Option to keep reads containing 'N's in input_sequence file. Default is to drop reads

--gzip
Compress output using gzip

--bzip
Compress output using bzip2

interleave-reads.py

Produce interleaved files from R1/R2 paired files

usage: interleave-reads.py [–version] [–info] [-h] [-o filename] [–no-reformat] [-f] [–gzip | –bzip] left right

left

right

--version
show program's version number and exit

--info
print citation information

-h, --help
show this help message and exit

-o <filename>, **--output** <filename>

--no-reformat
Do not reformat read names or enforce consistency

-f, --force
Overwrite output file if it exists

--gzip
Compress output using gzip

--bzip
Compress output using bzip2

The output is an interleaved set of reads, with each read in <R1> paired with a read in <R2>. By default, the output goes to stdout unless *-o*/*--output* is specified.

As a "bonus", this file ensures that if read names are not already formatted properly, they are reformatted consistently, such that they look like the pre-1.8 Casava format (*@name/1*, *@name/2*). This reformatting can be switched off with the *--no-reformat* flag.

Example:

```
interleave-reads.py tests/test-data/paired.fq.1 \
        tests/test-data/paired.fq.2 -o paired.fq
```

readstats.py

Display summary statistics for one or more FASTA/FASTQ files.

usage: readstats.py [–version] [–info] [-h] [-o filename] [–csv] filenames [filenames ...]

filenames

--version
show program's version number and exit

--info
print citation information

-h, --help
show this help message and exit

-o <filename>, **--output** <filename>
output file for statistics; defaults to stdout.

--csv
Use the CSV format for the statistics, including column headers.

Report number of bases, number of sequences, and average sequence length for one or more FASTA/FASTQ files; and report aggregate statistics at end.

With *-o*/*--output*, the output will be saved to the specified file.

Example:

```
readstats.py tests/test-data/test-abund-read-2.fa
```

sample-reads-randomly.py

Uniformly subsample sequences from a collection of files

usage: sample-reads-randomly.py [–version] [–info] [-h] [-N NUM_READS] [-M MAX_READS] [-S NUM_SAMPLES] [-R RANDOM_SEED] [–force_single] [-o filename] [-f] [–gzip | –bzip] filenames [filenames ...]

filenames

--version
show program's version number and exit

--info
print citation information

-h, --help
show this help message and exit

-N <num_reads>, **--num_reads** <num_reads>

-M <max_reads>, **--max_reads** <max_reads>

-S <num_samples>, **--samples** <num_samples>

-R <random_seed>, **--random-seed** <random_seed>
Provide a random seed for the generator

--force_single
Ignore read pair information if present

-o <filename>, **--output** <filename>

-f, --force
Overwrite output file if it exits

--gzip
Compress output using gzip

--bzip
Compress output using bzip2

Take a list of files containing sequences, and subsample 100,000 sequences (*-N*/*--num_reads*) uniformly, using reservoir sampling. Stop after first 100m sequences (*-M*/*--max_reads*). By default take one subsample, but take *-S*/*--samples* samples if specified.

The output is placed in *-o*/*--output* <file> (for a single sample) or in <file>.subset.0 to <file>.subset.S-1 (for more than one sample).

This script uses the reservoir sampling algorithm.

split-paired-reads.py

Split interleaved reads into two files, left and right.

usage: split-paired-reads.py [–version] [–info] [-h] [-d output_directory] [-0 output_orphaned] [-1 output_first] [-2 output_second] [-f] [–gzip | –bzip] [infile]

infile

--version
show program's version number and exit

--info
print citation information

-h, --help
show this help message and exit

-d <output_directory>, **--output-dir** <output_directory>
Output split reads to specified directory. Creates directory if necessary

-0 <output_orphaned>, **--output-orphaned** <output_orphaned>
Allow "orphaned" reads and extract them to this file

-1 <output_first>, **--output-first** <output_first>
Output "left" reads to this file

-2 <output_second>, **--output-second** <output_second>
Output "right" reads to this file

-f, --force
Overwrite output file if it exists

--gzip
Compress output using gzip

--bzip
Compress output using bzip2

Some programs want paired-end read input in the One True Format, which is interleaved; other programs want input in the Insanely Bad Format, with left- and right- reads separated. This reformats the former to the latter.

The directory into which the left- and right- reads are output may be specified using *-d*/*--output-dir*. This directory will be created if it does not already exist.

Alternatively, you can specify the filenames directly with *-1*/*--output-first* and *-2*/*--output-second*, which will override the *-d*/*--output-dir* setting on a file-specific basis.

-0/:option:'–output-orphans' will allow broken-paired format, and orphaned reads will be saved separately, to the specified file.

Example:

```
split-paired-reads.py tests/test-data/paired.fq
```

Example:

```
split-paired-reads.py -0 reads-output-file tests/test-data/paired.fq
```

Example:

```
split-paired-reads.py -1 reads.1 -2 reads.2 tests/test-data/paired.fq
```

Blog posts and additional documentation

Hashtable and filtering

The basic inexact-matching approach used by the hashtable code is described in this blog post:

http://ivory.idyll.org/blog/jul-10/kmer-filtering

A test data set (soil metagenomics, 88m reads, 10gb) is here:

http://ci.oxli.org/userContent/88m-reads.fa.gz

Illumina read abundance profiles

khmer can be used to look at systematic variations in k-mer statistics across Illumina reads; see, for example, this blog post:

http://ivory.idyll.org/blog/jul-10/illumina-read-phenomenology

The fasta-to-abundance-hist and abundance-hist-by-position scripts can be used to generate the k-mer abundance profile data, after loading all the k-mer counts into a .ct file:

```
# first, load all the k-mer counts:
load-into-counting.py -k 20 -x 1e7 25k.ct data/25k.fq.gz

# then, build the '.freq' file that contains all of the counts by position
python sandbox/fasta-to-abundance-hist.py 25k.ct data/25k.fq.gz

# sum across positions.
python sandbox/abundance-hist-by-position.py data/25k.fq.gz.freq > out.dist
```

The hashtable method 'dump_kmers_by_abundance' can be used to dump high abundance k-mers, but we don't have a script handy to do that yet.

You can assess high/low abundance k-mer distributions with the hi-lo-abundance-by-position script:

```
load-into-counting.py -k 20 25k.ct data/25k.fq.gz
python sandbox/hi-lo-abundance-by-position.py 25k.ct data/25k.fq.gz
```

This will produce two output files, <filename>.pos.abund=1 and <filename>.pos.abund=255.

Finding valleys/minima in *k*-mer abundance profiles

Using *k*-mer abundance profiles to dynamically calculate the abundance threshold separating erroneous *k*-mers from real *k*-mers is described in this blog post:

https://bitsandbugs.org/2016/07/29/mash-and-khmer-abundance/

Setting khmer memory usage

If you look at the documentation for the scripts (*khmer's command-line interface*) you'll see a `-M` parameter that sets the maximum memory usage for any script that uses k-mer counting tables or k-mer graphs. What is this?

khmer uses a special data structure that lets it store counting tables and k-mer graphs in very low memory; the trick is that you must fix the amount of memory khmer can use before running it. (See Pell et al., 2012 and Zhang et al., 2014 for the details.) This is what the *-M* parameter does.

If you set it too low, khmer will warn you to set it higher at the end. See below for some good choices for various kinds of data.

Note for khmer 1.x users: As of khmer 2.0, the *-M* parameter sets the *-N*/*--n_tables* and *-x*/*--max-tablesize* parameters automatically. You can still set these parameters directly if you wish.

The really short version

There is no way (except for experience, rules of thumb, and intuition) to know what this parameter should be up front. So, use the maximum available memory:

```
-M 16G
```

for a machine with 16 GB of free memory, for example. The supported suffixes for setting memory usage are K, M, G, and T for kilobyte, megabyte, gigabyte, and terabyte, respectively.

The short version

This parameter specifies the maximum memory usage of the primary data structure in khmer, which is basically N big hash tables of size x. The **product** of the number of hash tables and the size of the hash tables specifies the total amount of memory used, which is what the *-M* parameter sets.

These tables are used to track k-mers. If they are too small, khmer will fail in various ways (and will complain), but there is no harm in making it too large. So, **the absolute safest thing to do is to specify as much memory as is available**. Most scripts will inform you of the total memory usage, and (at the end) will complain if it's too small.

Life is a bit more complicated than this, however, because some scripts – **load-into-counting.py** and **load-graph.py** – keep ancillary information that will consume memory beyond this table data structure. So if you run out of memory, decrease the table size.

Also see the rules of thumb, below.

The long version

khmer's scripts, at their heart, represents k-mers in a very memory efficient way by taking advantage of two data structures, Bloom filters and Count-Min Sketches, that are both *probabilistic* and *constant memory*. The "probabilistic" part means that there are false positives: the less memory you use, the more likely it is that khmer will think that k-mers are present when they are not, in fact, present.

Digital normalization (**normalize-by-median.py** and **filter-abund.py**) uses the Count-Min Sketch data structure.

Graph partitioning (**load-graph.py** etc.) uses the Bloom filter data structure.

The practical ramifications of this are pretty cool. For example, your digital normalization is guaranteed not to increase in memory utilization, and graph partitioning is estimated to be 10-20x more memory efficient than any other de Bruijn graph representation. And hash tables (which is what Bloom filters and Count-Min Sketches use) are really fast and efficient. Moreover, the optimal memory size for these primary data structures is dependent on the number of k-mers, but not explicitly on the size of k itself, which is very unusual.

In exchange for this memory efficiency, however, you gain a certain type of parameter complexity. Unlike your more typical k-mer package (like the Velvet assembler, or Jellyfish or Meryl or Tallymer), you are either guaranteed not to run out of memory (for digital normalization) or much less likely to do so (for partitioning).

The biggest problem with khmer is that there is a minimum hash number and size that you need to specify for a given number of k-mers, and you cannot confidently predict what it is before actually loading in the data. This, by the way, is also true for de Bruijn graph assemblers and all the other k-mer-based software – the final memory usage depends on the total number of k-mers, which in turn depends on the true size of your underlying genomic variation (e.g. genome or transcriptome size), the number of errors, and the k-mer size you choose (the k parameter) [see Conway & Bromage, 2011]. **The number of reads or the size of your data set is only somewhat correlated with the total number of k-mers.** Trimming protocols, sequencing depth, and polymorphism rates are all important factors that affect k-mer count.

The bad news is that we don't have good ways to estimate total k-mer count a priori, although we can give you some rules of thumb, below. In fact, counting the total number of distinct k-mers is a somewhat annoying challenge. Frankly, we recommend *just guessing* instead of trying to be all scientific about it.

The good news is that you can never give khmer too much memory! k-mer counting and set membership simply gets more and more accurate as you feed it more memory. (Although there may be performance hits from memory I/O, e.g. see the NUMA architecture.) The other good news is that khmer can measure the false positive rate (FPR) and detect dangerously low memory conditions. For partitioning, we actually *know* what a too-high FPR is – our k-mer percolation paper lays out the math. For digital normalization, we assume that a FPR of 20% is bad. In both cases the data-loading scripts will exit with an error-code.

If you insist on optimizing memory usage, the **`unique-kmers.py`** script will compute the approximate number of k-mers in a data set fairly quickly. This number can be provided to several scripts via the `-U` option, which will use it to calculate the FPR before processing any input data. If the amount of requested memory yields an unacceptable FPR, the script will complain loudly, giving you the chance to cancel the program before any time is wasted. It will also report the minimum amount of memory required for an acceptable FPR, so that you can immediately re-start the script with the desired settings.

Rules of thumb

For digital normalization, we recommend:

- `-M 8G` for any amount of sequencing for a single microbial genome, MDA-amplified or single colony.
- `-M 16G` for up to a billion mRNAseq reads from any organism. Past that, increase it.
- `-M 32G` for most eukaryotic genome samples.
- `-M 32G` will also handle most "simple" metagenomic samples (HMP on down)
- For metagenomic samples that are more complex, such as soil or marine, start as high as possible. For example, we are using `-M 256G` for ~300 Gbp of soil reads.

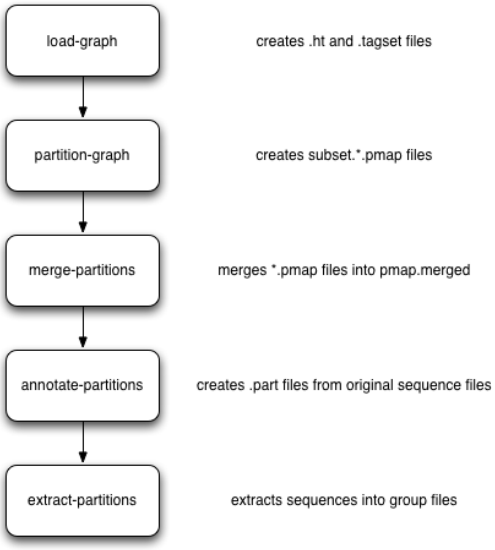
For partitioning of complex metagenome samples, we recommend starting as high as you can – something like half your system memory. So if you have 256 GB of RAM, use `-M 128G` which will use 128 GB of RAM for the basic graph storage, leaving other memory for the ancillary data structures.

Partitioning large data sets (50m+ reads)

"Partitioning" is what khmer calls the process of separating reads that do not connect to each other into different logical bins. The goal of partitioning is to apply divide & conquer to the process of metagenomic assembly.

Basic partitioning

The basic workflow for partitioning is in the figure below:

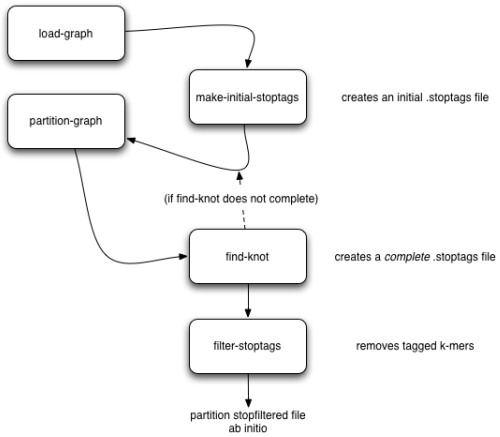


Briefly, you load everything into khmer's probabilistic graph representation; exhaustively explore the graph to find all disconnected sequences; merge the results of the (parallelized) graph exploration; annotate sequences with their partition; and then extract the different partitions into files grouped by partition size. These groups can then be assembled individually.

Artifact removal

As part of our partitioning research, we discovered that large Illumina data sets tend to contain a single large, connected component. This connected component seems to stem from sequencing artifacts that causes knots in the assembly graph. We have developed tools to forcibly remove the knot at the heart of the graph.

Here's the workflow:



Running on an example data set

Here is a set of commands for running both basic partitioning and artifact removal on a small soil metagenomics data set that we've made available for this purpose.

The data set is about 1.1G and you can download it from here:

https://s3.amazonaws.com/public.ged.msu.edu/khmer/iowa-corn-50m.fa.gz

```
cd /path/to/data

# the next command will create a '50m.ct' and a '50m.tagset',
# representing the de Bruijn graph
load-graph.py -k 32 -N 4 -x 16e9 50m iowa-corn-50m.fa.gz

# this will then partition that graph. should take a while.
# update threads to something higher if you have more cores.
# this creates a bunch of files, 50m.subset.*.pmap
partition-graph.py --threads 4 -s 1e5 50m

# now, merge the pmap files into one big pmap file, 50m.pmap.merged
merge-partitions.py 50m

# next, annotate the original sequences with their partition numbers.
# this will create iowa-corn-50m.fa.gz.part
annotate-partitions.py 50m iowa-corn-50m.fa.gz

# now, extract the partitions in groups into 'iowa-corn-50m.groupNNNN.fa'
extract-partitions.py iowa-corn-50m iowa-corn-50m.fa.gz.part

# at this point, you can assemble the group files individually.  Note,
# however, that the last one them is quite big?  this is because it's
# the lump! yay!

# if you want to break up the lump, go through the partitioning bit
# on the group file, but this time with a twist:
mv iowa-corn-50m.group0005.fa corn-50m.lump.fa

# create graph,
load-graph.py -x 8e9 lump corn-50m.lump.fa

# create an initial set of stoptags to help in knot-traversal; otherwise,
# partitioning and knot-traversal (which is systematic) is really expensive.
make-initial-stoptags.py lump

# now partition the graph, using the stoptags file
partition-graph.py --stoptags lump.stoptags lump

# use the partitioned subsets to find the k-mers that nucleate the lump
find-knots.py -x 2e8 -N 4 lump

# remove those k-mers from the fasta files
filter-stoptags.py *.stoptags corn-50m.lump.fa

# now, reload the filtered data set in and partition again.
# NOTE: 'load-graph.py' uses the file extension to determine
# if the file is formatted as FASTA or FASTQ. The default is
# fasta, therefore if your files are fastq formatted you need
# to append 'fastq' to the name so that 'load-graph.py'
# will parse the file correctly
load-graph.py -x 8e9 lumpfilt corn-50m.lump.fa.stopfilt
partition-graph.py -T 4 lumpfilt
merge-partitions.py lumpfilt
annotate-partitions.py lumpfilt corn-50m.lump.fa.stopfilt
extract-partitions.py corn-50m-lump corn-50m.lump.fa.stopfilt.part

# and voila, after all that, you should now have your de-knotted lump in
```

```
# corn-50m-lump.group*.fa.  The *.group????.fa files can now be
# assembled individually by your favorite assembler.
```

Post-partitioning assembly

The 'extract-partitions' script takes reads belonging to each partition and aggregates them into 'group' files; each group file contains at least one entire partition (and generally a lot more). Note, you can control the number of reads in each file (equiv, the size of these files) with some of the arguments that 'extract-partitions' takes.

Now that you have these files... what do you do with them? The short answer is: assemble them! Each of these group files contains reads that do not connect to reads in other files, so the files can be assembled individually (which is the whole point of partitioning).

If you're using Velvet, checkout the `sandbox/velvet-assemble.sh` script, which you can run like this:

```
bash /path/to/khmer/sandbox/velvet-assemble.sh <groupfile> <k>
```

This script does three things:

- first, it breaks the reads up into paired reads and single reads, and puts them in separate files (.pe and .se);
- second, it strips off the partition information from the reads, which confuses Velvet;
- and third, it runs velveth and velvetg to actually assemble.

You can implement your own approach, of course, but this is an example of what we do ourselves.

Example API Usage

Examples of how to run khmer command-line scripts are included in the main khmer documentation, as well as the khmer protocols collection. However, the intrepid user may be interested in having more direct access to khmer via its Python or C++ API.

The `examples/python-api` and `examples/c++-api` directories contain several small programs that demonstrate how one would write a program that uses the khmer API. These programs are run frequently as part of our continuous integration build, so they are quite stable. Note, however, that unlike the khmer command-line scripts, the Python and C++ API **ARE NOT** under semantic versioning. Consequently, internal changes to the khmer codebase may require changes to the API examples at any time. We expect to have the Python API under semantic versioning *by version 5.0*.

Known Issues

- `load-graph.py` in multithreaded mode will find slightly different number of unique kmers. This is being investigated in https://github.com/dib-lab/khmer/issues/1248
- Any scripts that consume FASTA or FASTQ data are unresponsive to attempts to terminate the program with `ctrl-c`. Eventually khmer should handle this properly, but for now it should be possible to halt a script using `ctrl-\`. This is being tracked in https://github.com/dib-lab/khmer/issues/1618.

Deploying the khmer project tools on Galaxy

This document is for people interested in deploying the khmer tools on the Galaxy platform.

We are developing the support for running all the khmer scripts in Galaxy.

Install the tools & tool description

In the administrative interface select "Search and browse tool sheds" under the heading "Tool sheds". Click on "Galaxy test tool shed" and search for khmer. Click on the "khmer" button and choose "Preview and install". Click the "Install to Galaxy" button at the top. At the bottom of the next page click the "Install" button.

Single Output Usage

For one or more files into a single file:

1. Choose 'Normalize By Median' from the 'khmer protocols' section of the 'Tools' menu.
2. Compatible files already uploaded to your Galaxy instance should be listed. If not then you may need to set their datatype manually.
3. After selecting the input files specify if they are paired-interleaved or not.
4. Specify the sample type or show the advanced parameters to set the tablesize yourself. Consult *Setting khmer memory usage* for assistance.

Papers using khmer

khmer was first published in Scaling metagenome sequence assembly with probabilistic de Bruijn graphs, Pell et al., 2012.

Since then, we have published several additional papers.

Please also see the instructions for citing our papers and software.

Biological uses outside of the group

See this list of papers for publications using khmer to analyze their sequencing data.

Tools building on khmer concepts

Several research papers have built off of concepts introduced in or elaborated on khmer.

How to get help

First, be sure that you:

1. Read the documentation (this site)
2. Google search for the error output and/or keywords related to your problem. Here you can search results from the mailing list, where others may have discussed solutions to the same issue.

Mailing List

The primary way to get help is through the khmer discussion list: http://lists.idyll.org/listinfo/khmer, though we are also available for closer-to-realtime support via Gitter.

Asking a question

1. Include your:
   - OS version (Mac OS X or Linux): `uname -mrs`
   - Python version: `python --version`
   - and khmer version: `pip freeze | grep khmer`
2. Precisely describe what you are trying to do. Reread it from the perspective of someone else trying to reproduce your task.
3. Copy-and-paste the exact command that is causing the problem. Include the steps you performed leading up to the issue.
4. Include the complete error message; if it is large, include a link to a file.

GitHub

You are also welcome to report an issue you are having using GitHub: https://github.com/dib-lab/khmer/issues/new

The khmer developer documentation

This section of the documentation is for people who are contributing (or would like to contribute to) the khmer 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:

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/

Getting started with khmer development

Contents

This document is for people who would like to contribute to khmer. It walks first-time contributors through making their own copy of khmer, building it, and submitting changes for review and merge into the master copy of khmer.

Start by making your own copy of khmer and setting yourself up for development; then, build khmer and run the tests; and finally, claim an issue and start developing! If you're unfamiliar with git and branching in particular, check out the git-scm book. We've also provided a quick guide to the khmer code base here: *A quick guide to the khmer codebase*.

One-time Preparation

1. Install the dependencies.

   (a) Mac users

      i. Install Xcode with:

```
xcode-select --install
```

      ii. Install Homebrew with:

```
/usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/
↪install/master/install)"
```

      iii. Install the python development environment and some additional development packages:

```
brew install python astyle gcovr cppcheck enchant
sudo pip install --upgrade pip virtualenv
```

   (b) Linux users

   Install the python development environment and some additional development packages. On recent versions of Debian or Ubuntu this can be done with:

```
sudo apt-get install python2.7-dev python-virtualenv python-pip \
gcc g++ git astyle gcovr cppcheck enchant
```

   For Red Hat, Fedora, and CentOS:

```
sudo yum install -y python-devel python-pip git gcc gcc-c++ make enchant
sudo pip install virtualenv
```

2. Create a GitHub account.

   We use GitHub to manage khmer contributions.

3. Fork github.com/dib-lab/khmer.

   Visit that page, and then click on the 'fork' button (upper right). This makes a copy of the khmer source code in your own GitHub account.

4. Clone your copy of khmer to your local development environment.

   Your shell command should look something like:

   ```
   git clone https://github.com/your-github-username/khmer.git
   ```

   This makes a local copy of khmer on your development machine.

5. Add a git reference to the khmer dib-lab repository:

   ```
   cd khmer
   git remote add dib https://github.com/dib-lab/khmer.git
   ```

   This makes it easy for you to pull down the latest changes in the main repository.

6. Optional: create a virtual Python environment for khmer development

   See the **Virtual Environments** heading in *Guidelines for continued development* for more details on creating and using a virtual environment for development. This is not strictly required but highly recommended, especially if you plan to make continued contributions.

7. Install Python dependencies

   From within the khmer directory, invoke the `make install-dep` command. If you are not using a virtual environment, you may need to invoke `sudo make install-dep` instead. This will install several packages used in khmer testing and development.

Building khmer and running the tests

1. Build khmer:

   ```
   make
   ```

   This compiles the C++ source code into something that Python can run. If the command fails, we apologize—please go create a new issue, paste in the failure message, and we'll try to help you work through it!

2. Run the tests:

   ```
   make test
   ```

   This will run all of the Python tests in the `tests/` directory. You should see lots of output, with something like:

   ```
   ====== 1289 passed, 1 skipped, 25 deselected, 1 xpassed in 50.98 seconds =======
   ```

   at the end.

Congratulations! You're ready to develop!

Claiming an issue and starting to develop

1. Find an open issue and claim it.

   Go to the list of open khmer issues and find one you like; we suggest starting with the low-hanging fruit issues).

Once you've found an issue you like, make sure that no one has been assigned to it (see "assignee", bottom right near "notifications"). Then, add a comment "I am working on this issue." You've staked your claim!

(We're trying to avoid having multiple people working on the same issue.)

2. In your local copy of the source code, update your master branch from the main khmer master branch:

```
git checkout master
git pull dib master
```

(This pulls in all of the latest changes from whatever we've been doing on dib-lab.)

If git complains about a "merge conflict" when you execute `git pull`, refer to the **Resolving merge conflicts** section of *Guidelines for continued development*.

3. Create a new branch and link it to your fork on GitHub:

```
git checkout -b fix/brief_issue_description
git push -u origin fix/brief_issue_description
```

where you replace "fix/brief_issue_description" with 2-3 words, separated by underscores, describing the issue.

(This is the set of changes you're going to ask to be merged into khmer.)

4. Make some changes and commit them.

Though this will largely be issue-dependent the basics of committing are simple. After you've made a cohesive set of changes, run the command *git status*. This will display a list of all the files git has noticed you changed. A file in the 'untracked' section are files that haven't existed previously in the repository but git has noticed.

To commit changes you have to 'stage' them—this is done by issuing the following command:

```
git add path/to/file
```

Once you have staged your changes, it's time to make a commit:

```
git commit -m 'Here you provide a brief description of your changes'
```

Please make your commit message informative but concise - these messages become part of the 'official' history of the project.

Once your changes have been committed, push them up to the remote branch:

```
git push origin
```

again.

5. Periodically update your branch from the main khmer master branch:

```
git pull dib master
```

(This pulls in all of the latest changes from whatever we've been doing on dib-lab - important especially during periods of fast change or for long-running pull requests.)

6. Run the tests and/or build the docs *before* pushing to GitHub:

```
make doc test pep8 diff-cover
```

Make sure they all pass!

7. Push your branch to your own GitHub fork:

```
git push origin
```

(This pushes all of your changes to your own fork.)

8. Repeat until you're ready to merge your changes into "official" khmer.
9. Set up a Pull Request asking to merge your changes into the main khmer repository.

   In a Web browser, go to your GitHub fork of khmer, e.g.:

   ```
   https://github.com/your-github-username/khmer
   ```

   and you will see a list of "recently pushed branches" just above the source code listing. On the right side of that should be a "Compare & pull request" green button. Click on it. This will open up a submission form with a pull request checklist. In this form:

   - add a descriptive title (e.g. "updated tests for XXX")
   - include any relevant comments about your submission in the main body of the pull request text, above the checklist
   - make sure to include any relevant issue numbers in the comments (e.g. "fixes issue #532")

   then click "Create pull request."

   (This creates a new issue where we can all discuss your proposed changes; the khmer team will be automatically notified and you will receive e-mail notifications as we add comments. See GitHub flow for more info.)
10. Review the pull request checklist and make any necessary additional changes.

    Check off as many of the boxes as you can from the checklist that is automatically added to the first comment of the Pull Request discussion. If you have an *ORCID ID<https://orcid.org/>* post that as well. This will make it much easier for the khmer team to include you in khmer publications.

    As you add new commits to address bugs or formatting issues, you can keep pushing your changes to the pull request by doing:

    ```
    git push origin
    ```
11. When you are ready to have the pull request reviewed, please mention @luizirber, @camillescott, @standage, @betatim, and/or @ctb with the comment 'Ready for review!'
12. The khmer team will now review your pull request and communicate with you through the pull request page. Please feel free to add 'ping!' and an @ in the comments if you are looking for feedback—this will alert us that you are still on the line.

    If this is your first issue, please *don't* take another issue until we've merged your first one. Thanks!
13. If we request changes, return to the step "Make some changes and commit them" and go from there. Any additional commits you make and push to your branch will automatically be added to the pull request.

After your submission passes peer review and the test suite (`make test` is run on continuous integration server automatically for each pull request), your contribution will be merged into the main codebase. Congratulations on making your first contribution to the khmer library! You're now an experienced GitHub user and an official khmer contributor!

After your first issue is successfully merged...

Before getting started with your second (or third, or fourth, or nth) contribution, there are a couple of steps you need to take to clean up your local copy of the code:

```
git checkout master
git pull dib master
git branch -d fix/brief_issue_description     # delete the branch locally
git push origin :fix/brief_issue_description  # delete the branch on your GitHub fork
```

This will syncronize your local main (master) branch with the central khmer repository—including your newly integrated contribution—and delete the branch you used to make your submission.

Now your local copy of the code is queued up for another contribution. If you find another issue that interests you, go back to the beginning of these instructions and repeat! You will also want to take a look at *Guidelines for continued development*.

A quick guide to testing (for khmer)

This document is for contributors new to automated testing, and explains some of the motivation and logic behind the khmer project's testing approach.

One of our most important "secret sauces" for khmer development is that we do a fair bit of testing to make sure our code works and keeps working!

- We maintain fairly complete test coverage of our code. What this means is that we have automated tests that, when run, execute most of the lines of Python and C++ code in our src/, khmer/ and scripts/ directories. This doesn't *guarantee* things are correct, but it does mean that at least most of the code works at some basic level.
- we have other tests that we run periodically (for example, before each release) – see *Releasing a new version of khmer* for details. These tests check that our code works on multiple systems and with other people's software.

CTB and others have written a great deal about testing, and testing in Python in particular. Here's an introductory guide CTB wrote a long time ago. You might also be interested in reading this description of the different kinds of tests.

For the more general motivation, see the Lack of Testing Death Spiral.

But... how do you do testing??

First, let's talk about specific goals for testing. What should you be aiming for tests to do? You can always add more testing code, but that might not be useful if they are redundant or over-complicated.

An overall rule is to "keep it simple" – keep things as simple as possible, testing as few things as possible in each test.

We suggest the following approach to writing tests for **new code**:

1. Write a test that just *runs* the new code, generally by copying existing test code to a new test and changing it. Don't do anything clever for the first test – just run something straightforward, and try to use existing data.
2. Decide which use cases should be tested. This is necessarily code specific but our main advice is "don't be clever" – write some tests to make sure that the code basically works.
3. Add in tests for edge cases. By this we mean look for special cases in your code – if statements, fence-post bound errors, etc. – and write tests that exercise those bits of code specifically.
4. Make sure tests that expect a function call to fail (esp. with fail_ok=True) are failing for the expected reason. Run the code from the command line and see what the behavior is. For troubleshooting tests, catch the error with try: ... except: or print err.

For adding tests to **old code**, we recommend a mix of two approaches:

1. use "stupidity driven testing" and write tests that recapitulate bugs before we fix those bugs.

2. look at test coverage (see khmer's cobertura test coverage, here) and identify lines of C++ or Python code that are not being executed by the current tests. Then write new tests targeting the new code.

Next, to add a test, you have two options: either write a new one from scratch, or copy an existing one. (We recommend the latter.)

To write a new one, you'll need to know how to write tests. For getting an idea of the syntax, read this introductory guide and the writing tests documentation from Astropy. Then find the right file in `tests/*.py` and add your test!

A better approach is, frankly, to go into the existing test code, find a test that does something similar to what you want to do, copy it, rename it, and then modify it to do the new test.

Finally, *where* do you add new tests and how do you run just *your* test?

Put new tests somewhere in `tests/*.py`. If you have trouble figuring out what file to add them to, just put them in *some* file and we'll help you figure out where to move them when we do code review.

To run one specific test rather than all of them, you can do:

```
py.test tests/test_scripts.py::test_load_into_counting
```

Here, you're running just one test – the test function named `test_load_into_counting` in the file `test_scripts.py`.

You can also invoke the test via setup.py, which is a bit more verbose:

```
./setup.py test --addopts "tests/test_scripts.py::test_load_into_counting"
```

Let's consider a simple test as an example. The following code ensures that a k-mer and its reverse complement hash to the same value, since they represent the same molecule (just observed from a different orientation).

```
def test_kmer_rc_same_hash():
    kmer = 'GATTACAGATTACAGATTACA'
    kmer_rc = 'TGTAATCTGTAATCTGTAATC'

    ct = Counttable(21, 1e5, 2)
    assert ct.hash(kmer) == ct.hash(kmer_rc)
```

This example tests only a single function. Tests that execute entire scripts and tests involving file I/O can require a bit more code. Fortunately, we've created some helper functions that make this quite a bit easier in khmer. See `tests/test_scripts.py` for some examples of code for executing scripts, capturing their output, and tidying up afterwards. Also, see `tests/khmer_tst_utils.py` for helper functions that assist with loading test data and creating temporary output files.

A quick guide to the khmer codebase

The `CHANGELOG.md` file lists major changes to the codebase with each version, with the most recent version first.

The `include/` directory contains all C++ and C headers. The core C++ library is under `include/oxli`, and the CPython wrapper is under `include/khmer`.

The `src` directory contains all of the C++ and CPython code. This structure corresponds to the one in `include`. The main CPython module is built in `src/khmer/_cpy_khmer.hh`.

The `khmer/` directory contains the *khmer* package (`khmer/__init__.py`, etc) and experimental Cython bindings under `khmer/_oxli`.

The `scripts/` and `sandbox/` directory contain Python command-line scripts.

The `tests/` directory contains all of the tests. Each test is a function in one of the `tests/test*.py` files.

Guidelines for continued development

The khmer *Getting Started* documentation is intended for first-time contributors, and is designed to make that first contribution as easy and as painless as possible. For those with an interest in making continued contributions (or those with an obligation to maintain and contribute to the codebase), this document describes the coding guidelines we follow, as well as some tips that will improve the development process for everyone involved.

Beyond your first contribution

If your *first contribution to khmer* has been accepted and merged into the codebase, you may be wondering what to do next. You can poke around at additional "low-hanging fruit" issues, but if you'd like to sink your teeth into something more substantial, here are a few suggestions.

- If you're knowledgeable in C++ and/or Python and/or documentation and/or biology, we'd love to attract further contributions to khmer. Please visit the issues list and browse about and find something interesting looking.
- One general thing we'd like to do is increase our test coverage. You can go find test coverage information at Codecov.io by scrolling to the bottom and clicking on individual files; or, ask us on khmer-project@idyll.org for suggestions.
- Ask us! Ask khmer-project@idyll.org for suggestions on what to do next. We can suggest particularly ripe low-hanging fruit, or find some other issues that suit your interests and background.
- You can also help other people out by watching for new issues or looking at pull requests. Remember to be nice and polite!

Programming languages

All Python code in khmer must run correctly in both Python version 2 and 3.

For C++ code, any feature in C++11 is fine to use. Specifically we support features found in GCC 4.8.2. Our automated tests use gcc 4.8.4 on linux. See https://github.com/dib-lab/khmer/issues/598 for an in-depth discussion. Please do not use features from C++14 or newer.

Code style standards

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

For C++, we use Todd Hoff's coding standard, and astyle -A10 / "One True Brace Style" indentation and bracing. Note: @CTB needs Emacs settings that work for this.

Vim users may want to set the ARTISTIC_STYLE_OPTIONS shell variable to "-A10 –max-code-length=80" and run `` `:%!astyle` `` to reformat. The four space indentation can be set with:

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

For Python, PEP 8 is our standard. The `pep8` and `autopep8` Makefile targets are helpful.

Code, scripts, and documentation must have their spelling checked.

Python-based *codespell* can be applied to multiple files easily. *codespell* can be installed via the following:

```
pip install codespell
```

To run codespell over only what has been changed on the branch *my-branch*:

```
git diff master..my-branch > diff_file
codespell diff_file
```

To run codespell over a single file:

```
codespell path/to/file
```

To make codespell fix the issues it finds automatically:

```
codespell -w path/to/file
```

Please note that as *codespell* works off of a listing of possible misspellings it may not catch all errors. If you find a spelling error that is not caught by *codespell* feel free to open a pull request at the project page to add it to the dictionary.

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 ccpp $filename
```

Cython Style

Cython code can become messy very quickly, and as such, we have guidelines for style and structure.

When wrapping code from liboxli:

- *extern* definition should begin with *Cp*; for example, *CpHashtable* wraps *oxli::Hashtable*.
- If the extension class wrapping the liboxli class stores it as a pointer, it should be named *_this*. If it wraps a stack object directly, it should be named *_obj*.

For imports,

- *__future__* imports at the top, as usual.
- *libc* cimports next,
- then *libcpp* imports and cimports.
- followed by cimports
- and finally, regular imports.

Generally,

- Pure C methods should be underscore prefixed.

- *cpdef* methods do not need to be underscore prefixed.

Resolving merge conflicts

It is possible that when you do a *git pull* you will get a "merge conflict" – This is what happens when something changed in the branch you're pulling in in the same place you made a change in your local copy. This frequently happens in the *ChangeLog* file.

Git will complain loudly about merges and tell you specifically in which files they occurred. If you open the file, you'll see something vaguely like this in the place where the merge occurred:

```
<<<<<<< HEAD
Changes made on the branch that is being merged into. In most cases,
this is the branch that you have currently checked out
=======
Changes made on the branch that is being merged in, almost certainly
master.
>>>>>>> abcde1234
```

Though there are a variety of tools to assist with resolving merge conflicts they can be quite complicated at first glance and it is usually easy enough to manually resolve the conflict.

To resolve the conflict you simply have to manually 'meld' the changes together and remove the merge markers.

After this you'll have to add and commit the merge just like any other set of changes. It's also recommended that you run tests.

Virtual environments

The khmer package, like many software packages, relies on other third-party software. Some of this software has been bundled together with khmer and is compiled when you invoke `make` on the command line. But some of the software khmer depends on is distributed as Python packages separately from khmer.

Python virtual environments were designed to isolate a stable development environment for a particular project. This makes it possible to maintain different versions of a Python package for different projects on your computer.

The installation instructions in the *Getting Started* docs install the `virtualenv` command on your computer. After completing those instructions, you can create a virtual environment with the command:

```
virtualenv -p python2 env/
```

(You can substitute *python3* for *python2* if Python version 3 is installed on your system.) This command will create a new directory *env/* containing your new virtual environment. The command:

```
source env/bin/activate
```

will activate the virtual environment. Now any Python packages that you install with `pip` or `make install-dep` will be installed into your isolated virtual environment.

Note that any time you create a new terminal session, using the virtual environment requires that you re-activate it.

Pull request cleanup (commit squashing)

Submitters are invited to reduce the numbers of commits in their pull requests either via *git rebase -i dib/master* or this recipe:

```
git pull # make sure the local is up to date
git pull dib master # get up to date
# fix any merge conflicts
git status # sanity check
git diff dib/master # does the diff look correct? (no merge markers)
git reset --soft dib/master # un-commit the differences from dib/master
git status # sanity check
git commit --all # package all differences in one commit
git status # sanity check
git push # should fail
git push --force # override what's in GitHub's copy of the branch/pull request
```

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.

CPython Checklist

Here's a checklist for new CPython types with future-proofing for Python 3:

```
- [ ] the CPython object name is of the form `khmer_${OBJECTNAME}_Object`
- [ ] Named struct with `PyObject_HEAD` macro
- [ ] `static PyTypeObject khmer_${OBJECTNAME}_Type` with the following
  entries
   - [ ] `PyVarObject_HEAD_INIT(NULL, 0)` as the object init (this includes
     the `ob_size` field).
   - [ ] all fields should have their name in a comment for readability
   - [ ] The `tp_name` filed is a dotted name with both the module name and
     the name of the type within the module. Example: `khmer.ReadAligner`
   - [ ] Deallocator defined and cast to `(destructor)` in tp_dealloc
     - [ ] The object's deallocator must be
       `Py_TYPE(obj)->tp_free((PyObject*)obj);`
   - [ ] Do _not_ define a `tp_getattr`
   - [ ] BONUS: write methods to present the state of the object via
     `tp_str` & `tp_repr`
   - [ ] _Do_ pass in the array of methods in `tp_methods`
   - [ ] _Do_ define a new method in `tp_new`
- [ ] PyMethodDef arrays contain doc strings
   - [ ] Methods are cast to `PyCFunctions`s
- [ ] Type methods use their type Object in the method signature.
- [ ] Type creation method decrements the reference to self
  (`Py_DECREF(self);`) before each error-path exit (`return NULL;`)
- [ ] No factory methods. Example: `khmer_new_readaligner`
- [ ] Type object is passed to `PyType_Ready` and its return code is checked
  in `MOD_INIT()`
- [ ] The reference count for the type object is incremented before adding
  it to the module: `Py_INCREF(&khmer_${OBJECTNAME}_Type);`.
```

Command line scripts, `scripts/`, and `sandbox/`

Note: This document applies through khmer/oxli 2.0/3.0 (see *Roadmap to v3.0, v4.0, v5.0*) - we will revisit when the Python API falls under semantic versioning for oxli 4.0.

khmer has two conflicting goals: first, we want to provide a reliable piece of software to our users; and second, we want to be flexible and enable exploration of new algorithms and programs. To this end, we've split our command line scripts across two directories, `scripts/` and `sandbox/`. The former is the staid, boring, reliable code; the latter is a place for exploration.

As a result, we are committed to high test coverage, stringent code review, and Semantic Versioning for files in `scripts/`, but explicitly *not* committed to this for files and functionality implemented in `sandbox/`. So, putting a file into `scripts/` is a big deal, especially since it increases our maintenance burden for the indefinite future.

We've roughed out the following process for moving scripts into `scripts/`:

- Command line scripts start in `sandbox/`;
- Once their utility is proven (in a paper, for example), we can propose to move them into `scripts/`;
- There's a procedure for moving scripts from `sandbox/` into `scripts/`.

Read on!

Sandbox script requirements and suggestions

All scripts in `sandbox/` must:

- be importable (enforced by `test_import_all` in `test_sandbox_scripts.py`)
- be mentioned in `sandbox/README.rst`
- have a hash-bang line (`#! /usr/bin/env python`) at the top
- be command-line executable (`chmod a+x`)
- have a Copyright message (see below)
- have lowercase names
- use '-' as a word separator, rather than '_' or CamelCase

All *new* scripts being added to `sandbox/` should:

- have decent automated tests
- be used in a protocol (see khmer-protocols) or a recipe (see khmer-recipes)
- be pep8 clean and pylint clean-ish (see `make pep8` and `make_diff_pylint`).

Standard and reserved command line options for `scripts/`

The following options are reserved, and the short option flag cannot be redefined in any official script.

- `-h|--help` - this must always print out a descriptive usage statement
- `-v|--version` - this must always print out the khmer version number
- `-x|--max-tablesize` - this must always specify the approximate table size for storing sketches of k-mer hashes
- `-N|--n_tables` - this must always specify the number of tables for storing sketches of k-mer hashes

- `-M|--max-memory-usage` - this must always specify the maximum amount of memory to be consumed for storing sketches of k-mer hashes
- `-U|--unique-kmers` - this must always specify the approximate number of unique k-mers in the data set
- `-k|--ksize` - this must always specify the k-mer size to use
- `-q|--quiet` - this must always indicate that the script's diagnostic output should be minimized or altogether eliminated

Additionally, all scripts in `scripts/` should have the following options.

- `-h|--help`
- `-v|--version`
- `-f|--force` - if applicable, override any sanity checks that may prevent the script from running

If an option is of type `type=argparse.FileType('w')` then you need to also specify a `metavar` for the documentation and help formatting. Example:

```
parser.add_argument('-R', '--report', metavar='report_filename',
    type=argparse.FileType('w'))
```

Copyright message

The copyright message should be of the form:

```
#
# This file is part of khmer, https://github.com/dib-lab/khmer/, and is
# Copyright (C) ___WHO___, ___YEAR(s)___. It is licensed under
# the three-clause BSD license; see doc/LICENSE.txt.
# Contact: khmer-project@idyll.org
#
```

Where ___WHO___ is replaced with one or more of "Michigan State University" or "The Regents of the University of California" and ___YEAR(s)___ is replaced with the year or years the file was created or modified. The copyright statement for new files should only refer to "The Regents of the University of California".

Upgrading a script from 'sandbox' to 'scripts'

First, everything needed (all library support code) should be already committed to khmer master after the usual review process; the relevant script(s) should be in `sandbox/`.

Second, an issue should be started explicitly to discuss whether the script(s) should be moved from `sandbox/` into `scripts/`. This issue should discuss the general need for this script, outside of a particular paper pipeline. (Note that there is no imperative to move a script out of `sandbox/`; if we think it's useful code to have around and want to keep it functioning, we should just add in automated tests and otherwise level it up.)

Third, assuming we reach general agreement about moving the script(s) into `scripts/`, start a pull request to do so, referencing the issue and containing the following checklist. The PR should start by moving the script from `sandbox/` into `scripts/`, and moving the tests out of the `test_sandbox_scripts.py` file.

Last but not least, intensive code review may raise more general issues that could apply to the entire code base; if contentious or needing discussion, these issues may be punted to general issues so as to not block a merge.

A checklist for moving a script into the scripts/ directory from sandbox/

Copy or paste this checklist into the PR, in addition to the normal development/PR checklist:

```
- [ ] most or all lines of code are covered by automated tests (see output of ``make␣
↪diff-cover``)
- [ ] ``make diff_pylint`` is clean
- [ ] the script has been updated with a ``get_parser()`` and added to doc/user/
↪scripts.txt
- [ ] argparse help text exists, with an epilog docstring, with examples and options
- [ ] standard command line options are implemented
- [ ] version and citation information is output to STDERR (`khmer_args.info(...)`)
- [ ] support '-' (STDIN) as an input file, if appropriate
- [ ] support designation of an output file (including STDOUT), if appropriate
- [ ] script reads and writes sequences in compressed format
- [ ] runtime diagnostic information (progress, etc.) is output to STDERR
- [ ] script has been removed from sandbox/README.rst
```

A guide for khmer committers

This document is for people with commit rights to github.com/dib-lab/khmer.

If you have commit privileges to the dib-lab/khmer repository, here are a few useful tips.

First, never merge something unless it's been through a review! This rule can be broken under specific conditions when doing a release; see *Releasing a new version of khmer*.

Second, we ask that all contributors set up standing Pull Requests (PR) while they are working something. (This is a **requirement** if you're in the DIB lab.) This lets us track what's going on. On the flip side, please do not review PRs until they are indicated as "ready for review".

Third, if a PR goes stale, that is the original author stopped working on it, wait a while to see if they come back. If not post a message saying that you would like to pick up the PR and if that would be Ok with the original author. Wait a few working days to give them a chance to respond. Start a new branch and new PR referencing the old one, where appropriate cherry pick existing commits from the old PR so that we can give appropriate credit where credit is due.

Releasing a new version of khmer

This document is for khmer release managers, and details the process for making a new release of the khmer project.

How to make a khmer release candidate

Michael R. Crusoe, Luiz Irber, and C. Titus Brown have all been release makers, following this checklist by MRC.

1. Announce a few days ahead of time that you will cut a release. This will slow down the rate at which PRs are being merged. When all outstanding PRs scheduled for this release have been merged, announce it. From now on no more merges to master. PRs to fix oversights or bugs follow the usual rule of "author can't merge". When you reach the end of this checklist announce it. Back to normal. If completing this checklist takes longer than a few hours consider allowing merges to master, and starting from the top with cutting a release.
2. The below should be done in a clean checkout:

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

3. (Optional) Check for updates to versioneer:

```
pip install --upgrade versioneer
versioneer install
git diff --staged

git commit -m -a "new version of versioneer.py"
# or
git checkout -- versioneer.py khmer/_version.py khmer/__init__.py MANIFEST.in
```

4. Review the git logs since the last release and diffs (if needed) and ensure that `CHANGELOG.md` is up to date (presumably peer code review has ensured that it is):

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

5. Review the issue list for any new bugs that will not be fixed in this release. Add them to `doc/user/known-issues.rst`

6. Check for new authors (`git log --format='%aN' v2.0... | sort -u` lists all committers since the v2.0 tag). Update `.mailmap` to normalize their email address and name spelling. If they want to opt out update the `list-*` Makefile targets to exclude them. Run `make list-citation` and adapt the output to the relevant parts of `CITATION`, `setup.py`, `doc/index.rst`.

7. Verify that the build is clean: https://api.travis-ci.org/dib-lab/khmer.svg?branch=master

8. Set your new version number and release candidate:

```
new_version=2.2
rc=rc1
```

and then tag the release candidate with the new version number prefixed by the letter 'v':

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

9. Test the release candidate. Bonus: repeat on Mac OS X:

```
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/
↪khmer.git
cd khmer
make install-dependencies
make test
normalize-by-median.py --version 2>&1 \
    | grep khmer\ ${new_version}-${rc} \
```

```
    && echo 1st manual version check passed
pip uninstall -y khmer; pip uninstall -y khmer; make install
mkdir ../not-khmer # make sure py.test executes tests
                   # from the installed khmer module
# you might want to add 'and not huge' to the test selection
pushd ../not-khmer; pytest --pyargs khmer.tests -m 'not known_failing'; popd


# Secondly we test via pip
cd ../../testenv2
source bin/activate
pip install -U setuptools==3.4.1
pip install -e git+https://github.com/dib-lab/khmer.git@v${new_version}-${rc}
↪#egg=khmer
cd src/khmer
make install-dependencies
make dist
make test
cp dist/khmer*tar.gz ../../../testenv3/
pip uninstall -y khmer; pip uninstall -y khmer; make install
cd ../.. # no subdir named khmer here, safe for testing installed khmer module
normalize-by-median.py --version 2>&1 \
    | grep khmer\ ${new_version}-${rc} \
    && echo 2nd manual version check passed
pytest --pyargs khmer.tests -m 'not known_failing'


# Is the distribution in testenv2 complete enough to build another
# functional distribution?
cd ../testenv3/
source bin/activate
pip install -U setuptools==3.4.1
pip install khmer*tar.gz
pip install pytest
tar xzf khmer*tar.gz
cd khmer*
make dist
make test
pip uninstall -y khmer; pip uninstall -y khmer; make install
mkdir ../not-khmer
pushd ../not-khmer ; pytest --pyargs khmer.tests -m 'not known_failing' ; popd
```

10. 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==3.4.1
pip install screed pytest
pip install -i https://testpypi.python.org/pypi --pre --no-clean khmer
```

```
pytest --pyargs khmer.tests -m 'not known_failing'
normalize-by-median.py --version 2>&1 \
    | grep khmer\ ${new_version}-${rc} \
    && echo 3rd manual version check passed
cd build/khmer
make test
```

11. Do any final acceptance tests.
12. Make sure any release notes are merged into `doc/release-notes/`.

How to make a final release

When you've got 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 ../../../khmer
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/khmer.git
git push --tags git@github.com:dib-lab/khmer.git
```

3. Add the release on GitHub, using the tag you just pushed. Name it 'version X.Y.Z', and copy and paste in the release notes.
4. Make a binary wheel on OS X.:

```
virtualenv build
cd build
source bin/activate
pip install -U setuptools==3.4.1 wheel
pip install --no-clean khmer==${new_version}
cd build/khmer
./setup.py bdist_wheel upload
```

5. Update Read the Docs to point to the new version. Visit https://readthedocs.io/builds/khmer/ 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.io/dashboard/khmer/versions/ under "Choose Active Versions". Finally change the default version at https://readthedocs.io/dashboard/khmer/advanced/ to the new version.
6. Delete any RC tags created:

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

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

Setuptools Bootstrap

ez_setup.py is from https://bitbucket.org/pypa/setuptools/raw/bootstrap/

Before major releases it should be examined to see if there are new versions available and if the change would be useful

Versioning Explanation

Versioneer, from https://github.com/warner/python-versioneer, is used to determine the version number and is called by Setuptools and Sphinx. See the files `versioneer.py`, the top of `khmer/__init__.py`, `khmer/_version.py`, `setup.py`, and `doc/conf.py` for the implementation.

The version number is determined through several methods: see https://github.com/warner/python-versioneer#version-identifiers

If the source tree is from a git checkout then the version number is derived by `git describe --tags --dirty --always`. This will be in the format `${tagVersion}-${commits_ahead}-${revision_id}-${isDirty}`. Example: `v0.6.1-18-g8a9e430-dirty`

If from an unpacked tarball then the name of the directory is queried.

Lacking either of the two git-archive will record the version number at the top of `khmer/_version.py` via the `$Format:%d$` and `$Format:%H$` placeholders enabled by the "export-subst" entry in `.gitattributes`.

Non source distributions will have a customized `khmer/_version.py` that contains hard-coded version strings. (see `build/*/khmer/_version.py` after a `python setup.py build` for an example)

`ez_setup.py` bootstraps Setuptools (if needed) by downloading and installing an appropriate version

Development Nuts and Bolts

Third-party use

We ask that third parties who build upon the codebase to do so from a versioned release. This will help them determine when bug fixes apply and generally make it easier to collaborate. If more intensive modifications happen then we request that the repository is forked, again preferably from a version tag.

Build framework

make should build everything, including tests and "development" code.

git and GitHub strategies

Still in the works, but read this.

Make a branch on dib-lab (preferred so others can contribute) or fork the repository and make a branch there.

Each piece or fix you are working on should have its own branch; make a pull request to dib-lab/master to aid in code review, testing, and feedback.

If you want your code integrated then it needs to be mergeable.

Code coverage

Travis and CodeCov calculate code coverage for every build, and post changes in code coverage to every pull request thread after a successful build.

Code coverage should never go down and new functionality needs to be tested.

Pipelines

All khmer scripts used by a published recommended analysis pipeline must be included in `scripts/` and meet the standards therein implied.

Command line scripts

Python command-line scripts should use '-' instead of '_' in the name. (Only filenames containing code for import should use _.)

Please follow the command-line conventions used in `scripts/`, as described in the *scripts and sandbox documentation*.

Command line thoughts:

> If a input filename is required, typically UNIX commands don't use a flag to specify it.
>
> Also, positional arguments typically aren't used with multiple files.
>
> CTB's overall philosophy is that new files, with new names, should be created as the result of filtering etc.; this allows easy chaining of commands. We're thinking about how best to allow override of this, e.g.

```
filter-abund.py <ct file> <filename> [ -o <filename.keep> ]
```

All code in `scripts/` must have automated tests; see `tests/test_scripts.py`. Otherwise it belongs in `sandbox/`.

When files are overwritten, they should only be opened to be overwritten after the input files have been shown to exist. That prevents stupid command line mistakes from trashing important files.

A general error should be signaled by exit code *1* and success by *0*. Linux supports exit codes from *0* to *255* where the value *1* means a general error. An exit code of *-1* will get converted to *255*.

CLI reading:

> http://stackoverflow.com/questions/1183876/what-are-the-best-practices-for-implementing-a-cli-tool-in-perl
>
> http://catb.org/esr/writings/taoup/html/ch11s06.html
>
> http://figshare.com/articles/tutorial_pdf/643388

Python / C integration

The Python extension that wraps the C++ core of khmer lives in `src/khmer/_cpy_khmer.cc`

This wrapper code is tedious and annoying so we use a static analysis tool to check for correctness.

https://gcc-python-plugin.readthedocs.io/en/latest/cpychecker.html

Developers using Ubuntu Precise will want to install the gcc-4.6-plugin-dev package

Example usage:

```
CC="/home/mcrusoe/src/gcc-plugin-python/gcc-python-plugin/gcc-with-cpychecker
--maxtrans=512" python setup.py build_ext 2>&1 | less
```

False positives abound: ignore errors about the C++ standard library. This tool is primarily useful for reference count checking, error-handling checking, and format string checking.

Errors to ignore: "Unhandled Python exception raised calling 'execute' method", "AttributeError: 'NoneType' object has no attribute 'file'"

Warnings to address:

```
src/khmer/_cpy_khmer.cc:3109:1: note: this function is too complicated
for the reference-count checker to fully analyze: not all paths were
analyzed
```

Adjust –maxtrans and re-run.

```
src/khmer/_cpy_khmer.cc:2191:61: warning: Mismatching type in call to
Py_BuildValue with format code "i" [enabled by default]
  argument 2 ("D.68937") had type
    "long long unsigned int"
  but was expecting
    "int"
  for format code "i"
```

See below for a format string cheat sheet One also benefits by matching C type with the function signature used later.

"I" for unsigned int "K" for unsigned long long a.k.a oxli::HashIntoType.

Linking Against liboxli

The C++ library can be installed as a shared library and linked against from external projects. To build and install it, run:

```
make install-liboxli
```

This command can be given an optional `PREFIX` variable to control where the library and headers are installed (by default, in `/usr/local`. Code can then include the headers by prefixing their paths with `oxli/`. For example, to use `Hashgraph`, use `#include "oxli/hashgraph.hh"`. To compile, add `-Ioxli` to your compiler invocation.

Experimental Cython Bindings

khmer includes experimental Cython bindings in `khmer/_oxli`. `wrapper.pxd` contains all the C++ library declarations. To use extension classes in regular Python code, simply `import` them: for example, to get the wrapped `ReadParser`, use `from khmer._oxli.parsing import FastxParser`. Extension classes can all be used in external Cython code by using *cimport*; the declarations in `wrapper.pxd` can also be used, meaning you have access to liboxli. Note that for any `cimport`'ed code to work, you'll need to install liboxli and include `oxli` in your Cython project's `Extension` class. This is done by adding `oxli` to the `libraries` argument of your `Extension` object in `setup.py`, which instructs setuptools to add `-Ioxli` to its compiler invocation.

An example:

```
cy_ext = Extension('mypackage.example',
                   sources = 'mypackage/example.pyx',
                   extra_compile_args = ['-arch', 'x86_64', '-stdlib=libc++
↪'],
```

```
                    libraries = ['oxli'],
                    include_dirs = [],
                    language = 'c++')
```

Read handling

Several bugs have gone unnoticed due to inconsistencies in read handling. On the C++ side, there are an abundance of `consume` functions for loading Fasta/Fastq sequences. On the Python side, read handling is sometimes delegated to the C++ library, and sometimes handled in Python using screed.

In an attempt to normalize read handling in Python, the functions in `khmer/utils.py` should be used whenever possible. Here, `broken_paired_reader` in `khmer/utils.py` should be used to do all paired-end sequence handling, and sequence loading should go through `khmer.utils.clean_input_reads(iter)`; this is a generator that wraps the iterator produced by `screed.open`, and it adds a `cleaned_seq` attribute to screed `Record` objects. This attribute should be used for any k-mer or graph operations, while the normal `sequence` attribute is what should be written out. `write_record` and `write_record_pair` should be used to output records. All of these functions are aware of FASTA and FASTQ records, too.

For applying operations to collections of reads, the `ReadBundle` class is available. This is used to wrap a collection of reads for examination and processing in situations where (for example) something should be done to either both reads in a pair, or neither.

Some basic rules of sequence handling in khmer are:

- consume and produce "broken paired" format, such that pairs of sequences always stay together; see `khmer.utils.broken_paired_reader`.
- when looking at the coverage of reads (for trimming or digital normalization) always consider pairs; see `khmer.utils.ReadBundle(...)`.
- only apply graph or k-mer operations to sequences consisting only of ATCG; typically this will be `record.cleaned_seq`. See `khmer.utils.clean_input_read(...)`.

khmer/Oxli Binary File Formats

- C++ macro definitions are given in parenthesis.
- C++ types are given in square brackets.
- `Len` is the field's size, in bytes, and `Off` is the field's zero-based byte offset in the file/section.

khmer v1.4 and previous

CountingHash

The header is in the format below, in file offset order. There is no magic string.

Field	Length	Value
Version	1	`0x04` (`SAVED_FORMAT_VERSION`)
File Type	1	`0x01` (`SAVED_COUNTING_HT`)
Use Bigcount	1	`1` if bigcounts is used, else `0`
K-size	1	k-mer length, `1 <= k <= 32`
Number of Tables	1	Number of Count-min Sketch tables

khmer v2.0 formats

Magic string

All formats shall have the "magic string" `OXLI` as their first bytes, after any external compression/encoding (e.g. gzip encapsulation) is removed. Note that this makes them incompatible with older versions of khmer.

Countgraph

(a.k.a `CountingHash`, a Count-min Sketch)

The header is in the format below, again in the order of file offset.

Field	Len	Off	Value
Magic string	4	0	`OXLI` (`SAVED_SIGNATURE`)
Version	1	4	`0x04` (`SAVED_FORMAT_VERSION`)
File Type	1	5	`0x01` (`SAVED_COUNTING_HT`)
Use Bigcount	1	6	`0x01` if bigcounts is used, else `0x00`
K-size	4	7	k-mer length, `ht._ksize`. [`uint32_t`]
Number of Tables	1	11	Number of Count-min Sketch tables, `ht._n_tables`. [`uint8_t`]
Occupied Bins	8	12	Number of occupied bins

Then follows the Countgraph's tables. For each table:

Field	Len	Off	Value
Table size	8	0	Length of this table, `ht._tablesizes[i]`. [`uint64_t`]
Bins	N	8	This table's bins, length given by previous field. [`uint8_t`]

Then follows a single value, the [`uint64_t`] number of `kmer: count` pairs. Then follows the Bigcount map, if this number is greater than zero. For each kmer:

Field	Len	Off	Value
Kmer	8	0	Kmer's hash [`HashIntoType/uint64_t`].
Count	2	8	Kmer's count [`uint16_t`].

Nodegraph

(a.k.a `HashBits`, a Bloom Filter)

The header is in the format below, again in the order of file offset. Value macro definitions are given in parenthesis

Field	Len	Off	Value
Magic string	4	0	`OXLI` (`SAVED_SIGNATURE`)
Version	1	4	`0x04` (`SAVED_FORMAT_VERSION`)
File Type	1	5	`0x02` (`SAVED_HASHBITS`)
K-size	4	6	k-mer length, `ht._ksize`. [`unsigned int`]
Number of Tables	1	10	Number of Nodegraph tables. `ht._n_tables`. [`uint8_t`]
Occupied Bins	8	11	Number of occupied bins

Then follows the Nodegraph's tables. For each table:

Field	Len	Off	Value
Table size	8	0	Length of table, **in bits** (`uint64_t`).
Bins	N/8+1	8	This table's bytes, length given by previous field, divided by 8, plus 1 (`uint8_t`).

Roadmap to v3.0, v4.0, v5.0

Background

To make the khmer project easier to use and easier to build upon several fundamental changes need to happen. This document outlines our plan to do so while minimizing the impact of these changes on our existing users. The version numbers are approximate; there may be additional major number releases to support needed changes to the API along the way. This document has been updated for v2.0 onwards from the original roadmap.

The discussion that lead to this document can be read at https://github.com/dib-lab/khmer/issues/389

Remainder of v2.x series

Continue transition to a single entrypoint named oxli . This will be exempt from the project's semantic versioning and will not be advertised as it is experimental and unstable. (For the 2.0 version we removed the `oxli` script just prior to release and restored it afterwards to the development tree.)

The Python script functionality will continue to migrate to a Python module named oxli. As the code moves over there will be no change to external script functionality or their command line interfaces (except for new features).

v3.x series

The `oxli` command is now under semantic versioning. Scripts are still the advertised and preferred entry point for users. Developers and workflow systems can start to trial `oxli` but need not switch until 4.0. New functionality is added to both the scripts and the `oxli` command.

v4.0 and project renaming

Project renamed to 'oxli'; all references to 'khmer' removed from the code and documentation except for a single note in the docs. All scripts dropped as their functionality has been moved to the `oxli` command. Websites that we maintain that have 'khmer' in the URL will have redirects installed.

Refinement of the Python API continues, however it is not part of the semantic versioning of the project.

v5.0

The semantic versioning now extends to the Python API.

Python API wishlist

API for multiple container types and implementation of the same.

Cleanup of Python/C++ class hierarchy to cut down on boilerplate glue code.

Switch to new-style Python objects (see LabelHash & Hashbits)

License

There are two mailing lists dedicated to khmer, an announcements-only list and a discussion list. To search their archives and sign-up for them, please visit the following URLs:

- Discussion: http://lists.idyll.org/listinfo/khmer
- Announcements: http://lists.idyll.org/listinfo/khmer-announce

The archives for the khmer mailing list are available at: http://lists.idyll.org/pipermail/khmer/

khmer development was initially supported by AFRI Competitive Grant no. 2010-65205-20361 from the USDA NIFA, and is now funded by the National Human Genome Research Institute of the National Institutes of Health under Award Number R01HG007513 through May 2016, both to C. Titus Brown. More recently, we have received support from the Gordon and Betty Moore Foundation under Award number GBMF4551.

Index

Symbols

A

B

C

D

E

F

G

I

L

M

N

O

P