BBMap/BBTools usage guide.
Last updated December 11, 2015

Table of Contents

System Requirements

BBTools is written in Java and requires Java 7 or higher for full functionality. It is tested on Oracle's JDK, not OpenJDK. Most tools will work with Java 6 (if not, they will throw a ClassNotFound exception), and most tools will work with OpenJDK, but if you experience a problem with Java 6 or OpenJDK it is recommended that you install Oracle's latest JDK. All operating systems that support Java are supported. Note that many of the tools require a substantial amount of memory, so the 64-bit JDK should be installed if you are using a 64-bit operating system. You can determine your current Java version like this:

```
java -Xmx90m -version
```

Installation (for non-NERSC users; if you don't know what NERSC is, you are not a NERSC user)

BBTools can be installed by downloading the gzipped tar file from Sourceforge (http://sourceforge.net/projects/bbmap/files/latest/download) and decompressing it. In Linux, the command would be:

```
tar xvzf BBMap_35.74.tar.gz
```

Then, optionally, you can export the path of the shellscripts to your environment to make it easier to run. The Java code is already compiled and does not need recompilation. Source code is also available from bitbucket (https://bitbucket.org/berkeleylab/jgi-bbtools), but this is currently a private Berkeley account.

BBTools includes bash wrapper scripts to make the command lines shorter. The package also contains C code that can accelerate certain programs, and experimental MPI code that can make it difficult to compile BBTools on systems without MPI support. None of these is required. But, you can accelerate BBMerge, Dedupe, and BBMap by following the instructions in /jni/README.txt if you have a C compiler.

Installation (for NERSC users)

BBTools is in Shifter, which can be used on Denovo and Cori with no installation. If you want to run the command "reformat.sh in=x.fq out=y.fq", the syntax would be:

```
shifter --image=bryce911/bbtools reformat.sh in=x.fq out=y.fq
```

Terminology Notes

"Read" in this file is used synonymously with "sequence", whether it is contig in a fasta file or a short read produced by a sequencing platform. "Paired reads" or "pair" refer to 2 reads that are generated by sequencing both ends of a single fragment of DNA. These are typically delivered in two fastq files, named something like "read1.fastq.gz" and "read2.fastq.gz". The alternative is single-ended reads, in which only one end of the molecule is sequenced. When paired reads are available, it is important to always process them together, rather than for example mapping the read 1 file and the read 2 file in two separate processes.

Usage

Most BBTools use the same syntax and operate with a set of standard flags. Individual tools also have specific flags - for example, kmer-based tools support the flag "k" to specify the kmer length, and non-kmer-based tools don't. This guide describes the standard syntax and most common flags. Custom syntax and flags for a given tool are described in that tool's shellscript.

Standard Syntax

Most BBTools (such as Reformat or BBNorm) process genomic sequences in some fashion, and are executed like this:

```
reformat.sh in=reads.fq out=processed.fq
```

The shellscript allows autodetection of memory (in some cases) and classpath.
The above command is equivalent to this:

```
java -ea -Xmx200m -cp /path/to/bbmap/current/ jgi.ReformatReads in=reads.fq
out=processed.fq
```

Note that �/path/to/bbmap/current/� needs to be replaced with an actual path. While the shellscript will only work in bash (or some other Linux/Unix/MacOS shells),
the full Java command will work on any environment with Java installed, such as Windows.

Tools that use a reference (such as BBMap, BBDuk, and Seal) will also need the additional flag "ref=":

```
bbmap.sh in=reads.fq out=mapped.sam ref=genome.fasta
```

In each of the above cases, the flags can be arranged in any order.

Paired Reads

Most BBTools support paired reads. These may be in two files, or interleaved in a single file, which BBTools will autodetect based on the read names. When the reads are in two files, you can use the in2 and out2 flags, like this:

```
reformat.sh in1=read1.fq in2=read2.fq out1=processed1.fq out2=processed2.fq
```

It is also possible to specify paired files like this:

```
reformat.sh in=read#.fq out=processed#.fq
```

...which is equivalent to the above command.

It is important to process paired files together in one command so that they are kept in the proper order. If you have dual input files and only 1 output file, the output will be written interleaved, and vice-versa. All tools that support paired reads will keep pairs together. For example, Reformat supports subsampling; if read 1 is discarded, read 2 will also be discarded. This prevents a loss of synchronization that corrupts the output.

Multiple Output and % Symbol

Some tools (such as Seal, BBSplit, BBMap, Dedupe) can use the % symbol as a wildcard, to be replaced by some other word when generating many files from a single input. It is recommended that the % symbol be avoided in filenames. As an example, assume you run Seal to bin some reads based on matching sequences in the fasta file "ref.fa", which contains the genomes of e.coli and salmonella:

```
seal.sh in=reads.fq pattern=out_%.fq ref=ref.fa
```

This would produce the output files "out_e.coli.fq" and "out_salmonella.fq".

File Formats and Extensions

BBTools support most standard sequence formats, including fastq, fasta, fasta+qual, scarf, sam, and (if samtools is installed) bam. They also support gzip and (if bzip2 or pbzip2 is installed) bzip2. The tools are sensitive to file extensions. For example:

```
reformat.sh in=reads.fq.gz out=processed.fa
```

In this case, reformat will try to read a gzip-compressed fastq file and output an uncompressed fasta file. For BBMap, this means that it will output a sam file if you name the output ".sam", bam if you name it ".bam", fastq if you name it ".fastq", and so forth. BBTools are usually capable of autodetecting input format (for example, if you feed it a fasta file called "stuff.txt" it will be able determine that it is in fasta format), but this is not recommended. Also, it is possible to specify an extensionless name for an output file, in which case the default format is used; the default varies by tool.

List of supported file extensions:

Fastq: fastq, fq
Fasta: fasta, fa, fas, fna, ffn, frn, seq, fsa, faa
Bread: bread (BBMap internal format; deprecated)
Sam: sam
Bam: bam
Qual: qual (should be accompanied with fasta)

Scarf: scarf (an old Illumina format; input only)
Phylip: phylip (only supported by phylip2fasta; input only)
Text: txt (used for logs, stats, and histograms)
Header: header (use this extension to write read names only)
One Line: oneline (name <tab> sequence)
Sketch: sketch (for Sketch tools).

List of supported compression extensions:

Gzip: gzip, gz
Bzip2: bz2
Zip: zip

Piping and Screen Output

Most tools can accept input from stdin and write output to stdout, with notable exceptions being BBNorm and Tadpole in some processing modes, which require reading the input file multiple times. Piping works like this:

```
cat reads.fq.gz | reformat.sh in=stdin.fq.gz out=stdout.fa int=f > x.fa
```

Note that the extensions are added to stdin and stdout so that Reformat knows how to interpret the data; when piping, it cannot first autodetect the file format. Similarly, it cannot autodetect whether the reads are interleaved or not. So, "int=f" (equivalent to "interleaved=false") was added to force it to treat the data as single-ended.

By default, all tools write status information to stderr, not stdout. To capture a program�s screen output, do this:

```
reformat.sh in=a.fq out=b.fq 1>out.txt 2>err.txt
```

Or, to direct both to a single file:

```
reformat.sh in=a.fq out=b.fq 1>out.txt 2>&1
```

Memory and Java Flags

There are two flags that are passed by the shellscripts directly to Java rather than to BBTools, "-Xmx" and "-da".
Java does not dynamically grow virtual memory as needed like C programs. The amount of virtual memory must be specified up front, and it will immediately be grabbed; the physical memory used will only increase as needed. The shellscripts will try to autodetect memory and set it to an appropriate value, but sometimes this will need to be overridden (for example, if you are using a shared node and don't really need all the memory, or not enough memory was

allocated and the program crashed with a memory exception). To force a program to use 3 gigs of RAM, use the flag "-Xmx3g". For example:

```
reformat.sh in=reads.fq out=processed.fq -Xmx3g
```

That's the equivalent of:

```
java -ea -Xmx3g -cp /path/to/bbmap/current/ jgi.ReformatReads in=reads.fq out=processed.fq
```

The "-ea" flag means "enable assertions", which will make BBTools crash if they detect a problem. If you want to ignore the problem and force it to run anyway, you can use the "-da" flag. The -da flag may also increase speed slightly.

Threads

Most BBTools are multithreaded, and will automatically detect and use all available threads. This is usually desirable when you have exclusive use of a computer, but may not be on a shared node. The number of threads can be capped at X with the flag "t=X" (threads=X). The total CPU usage may still go higher, though, due to several factors:
1) Input and output are handled in separate threads; "t=X" only regulates the number of worker threads.
2) Java uses additional threads for garbage collection and other virtual machine tasks.
3) When subprocesses (such as pigz) are spawned, they also individually obey the thread limit, but if you set "t=4" and the process spawns 3 pigz instances, you could still potentially use over 16 threads - 4 worker threads, 4 threads for each pigz process, plus other threads for the JVM and I/O.
If you have exclusive use of a computer, you don't need to worry about spawning too many threads; this is only an issue with regards to fairness on shared nodes.

Subprocesses

If they are installed, BBTools will automatically use samtools for sam<->bam conversion, and bzip2 or pbzip2 for processing bz2 files. It may use pigz to accelerate processing of gzipped files, depending on the number of threads available. This is generally fine on a standalone computer, but in some circumstances, depending on the cluster configuration, the scheduler may kill a process that spawns a subprocess for violating virtual memory limits (Amazon instances may do this). In that case, you can disable pigz support with the flags "pigz=f unpigz=f". Alternatively, you can force pigz to be used with "pigz=t unpigz=t". The default for those flags depends on the tool. Pigz processes will never be spawned unless the number of threads allowed is at least 3. It is also possible to spawn gzip instances instead of pigz instances, but this only gives a small speed increase over using Java for gzip processing.

Additional Help

There are many forum threads on SeqAnswers describing the usage of different BBTools, linked from this thread:
http://seqanswers.com/forums/showthread.php?t=41057

Standard flags

The flags below work with many or all BBTools that process reads, but the list is not complete because it does not include flags specific to only one or a few tools - those are listed in the shellscript. They are listed with their default setting, but some of the defaults differ between tools; the specific default is also listed in the shellscript. Where the description starts with something in parentheses, like "(in1)", that is an acceptable alternative version of the flag.

Flag Syntax

With the exception of certain special flags like help flags (--help, --version) and Java flags (-Xmx, -da), all flags are in the same format: �a=b� where �a� is the name of the flag, and is not case-sensitive, and �b� is the value, which is case-sensitive (for filenames). Flags may be in any order, and never need leading hyphens, except for those special flags mentioned above. If a flag is set twice, the later value will override the former; for example, �reformat.sh in=x.fq in=y.fq� will use y.fq as input. The special value �null� means �blank�. For example, �reformat.sh in=a.fq out=null� and �reformat.sh in=a.fq� are equivalent - neither will output anything.
For boolean variables, �null� is equivalent to �true�, and values may be abbreviated �t and f. So, these are all equivalent:

Help Flags

--help Print the usage information from the shellscript (when run from a shellscript). Alternately you can just look at the shellscript with a text editor.
--version Print the version of BBTools.

Config Files

config=<file> A file or a comma-delimited list of files. If this flag is present, the contents of the config file will be added to the command line. Config files must contain one argument per line. Config files are never required, but may be useful when a command line would be too long or when arguments contain whitespace. See readme_config.txt for more information.

Input Flags

in=<file>   (in1) Main input.
in2=<file>   Input for 2nd read of pairs in a different file.
interleaved=auto   (int) t/f overrides interleaved autodetection.
samplerate=1   Set lower to only process a fraction of input reads.
qfin=<.qual file>   Read qualities from this qual file, for the reads coming from 'in'
qfin2=<.qual file>   Read qualities from this qual file, for the reads coming from 'in2'
extout=   Allows overriding of input file format. For example, "extin=.fq" would force the input to be read in fastq format regardless of the file name.
trimreaddescription=f   (trd) Trim the names of reads after the first whitespace.
touppercase=f   (tuc) Convert lowercase letters in reads to upper case.
lowercaseton=f   (lctn) Convert lowercase letters in reads to N.
utot=f   Convert U bases to T.

Output Flags

out=<file>   (out1) Main output.
out2=<file>   Output for 2nd read of pairs in a different file.
qfout=<.qual file>   Write qualities from this qual file, for the reads going to 'out'
qfout2=<.qual file>   Write qualities from this qual file, for the reads coming from 'out2'
extout=   Allows overriding of output file format. For example, "extout=.fq" would force the output to be written in fastq format regardless of the file name.
fastawrap=70   Length of lines in fasta output.
overwrite=f   Allow overwriting of existing files.
append=f   Append to existing files.

Sampling Flags

reads=-1   Set to a positive number to only process this many input reads (or pairs), then quit.
samplerate=1   Randomly output only this fraction of reads; 1 means sampling is disabled.
sampleseed=-1   Set to a positive number to use that prng seed for sampling (allowing deterministic sampling). A negative number will use a random seed.

Threading Flags

threads=auto   (t) Number of worker threads to spawn.

Compression Flags

ziplevel=2   (zl) Compression level for zip or gzip output; 1-9.
unpigz=   Spawn a pigz process for faster decompression. Requires pigz to be installed. Valid values are t or f; the default varies by program.

pigz= Spawn a pigz process for faster compression. Requires pigz to be installed. Valid values are t, f, or a number; the default varies by program. "pigz=X" will enable pigz, and also force all pigz processes to use exactly X threads.

Quality-Related Flags

qin=auto Input quality offset: 33 (Sanger), 64, or auto.
qout=auto ASCII offset for output quality. May be 33 (Sanger), 64 (Illumina), or auto (same as input).
qfake=30 Quality value used for fasta to fastq reformatting.
maxcalledquality=41 Cap quality values at this upper level.
mincalledquality=0 Cap quality values at this lower level.
ignorebadquality (ibq) Don't crash if quality values appear to be incorrect.
qtrim=f Enable or disable quality trimming. May be set to r, l, or rl to trim the right, left, or both sides.
trimq= Trim bases below this quality value.

Length-Related Flags

fastareadlen= Fasta sequences longer than this are broken into subsequences of at most this length, and given a suffix such as _part_1. Only works with fasta files; generally designed for mapping very long sequences with BBMap.
fastaminlen= Discard fasta sequences shorter than this.
maxlen= Has different meanings depending on the program. For BBMap, reads longer than this will be broken to pieces this length. For most other programs, it acts as a filter.
minlen= Has different meanings depending on the program. Typically, reads shorter than this will be discarded.

Histogram Flags

bhist=<file>	Base composition histogram by position.
qhist=<file>	Quality histogram by position.
qchist=<file>	Count of bases with each quality value.
aqhist=<file>	Histogram of average read quality.
bqhist=<file>	Quality histogram designed for box plots.
lhist=<file>	Read length histogram.
gchist=<file>	Read GC content histogram.
gcbins=100	Number gchist bins. Set to 'auto' to use read length.

Advanced Flags
Debugging and Benchmarking Flags

verbose=f Print status messages for debugging.

parsecustom=f Parse synthetic read names for custom data stored by RandomReads.

Buffering and I/O Flags

readbufferlength=200 Number of reads to store per ListNum. A ListNum is the smallest unit of work sent to a worker thread.
readbuffers= Number of ListNums to store in the queue waiting for worker threads. The default is 150% of the number of threads.
bf1= Set to true to force ByteFile1 to be used for reading files.
bf2= Set to true to force ByteFile2 to be used for reading files (faster).

MPI and JNI Flags

usejni=f Set to true to enable JNI-accelerated versions of BBMerge, BBMap, and Dedupe. Requires the C code to be compiled.
mpi=0 Inform the program of how many MPI processes will be used. Most programs are not currently MPI-capable.
crismpi=f Use an MPI version of ConcurrentReadInputStreams.
mpikeepall=t If using MPI, send all reads to all processes.

Note on threads:

Virtually all BBTools are multithreaded. If a description indicates that a tool is singlethreaded, that generally means there is only 1 worker thread. File input and output are usually in separate threads, so a "singlethreaded" program like ReformatReads may be observed using over 250% of the resources of a single core (in other words, 2.5 threads on average, with 1 input file and 1 output file). Programs listed as multithreaded, on the other hand, will automatically use all available threads (meaning the number of logical processors) unless restricted. Most multithreaded tools scale near-linearly with the number of cores up to at least 32.

Note on memory:

The memory usage classification of "low" or "high" is based on assumptions; with the exception of AssemblyStats (which uses a fixed amount of memory), the actual amount of memory needed varies based on the parameters and input files. While all programs can be forced to use a specific amount of memory with the -Xmx flag, the tools classified as low memory will try to grab only a small amount of memory by default when run via the shellscript, while the ones listed as high memory will try to grab all available memory.

Alignment and Coverage-Related

Name: align2.BBMap
Shellscript: bbmap.sh, removehuman.sh, removehuman2.sh, mapnt.sh
Description: Fast and accurate splice-aware read aligner for DNA and RNA. Finds optimal global alignments. Maximum read length is 600bp.
Notes: Multithreaded, high memory. Memory usage depends on the size of the reference; roughly 6 bytes per base, or 3 bytes per base with the flag "usemodulo".
Additional Shellscripts: removehuman.sh calls BBMap with a prebuilt index and parameters designed to remove human contamination with zero false-positives; removehuman2.sh is designed to minimize false-negatives at the expense of allowing some false-positives. mapnt.sh calls BBMap with a prebuilt index and parameters designed to allow mapping to nt while running on a 120GB node. All of these are designed exclusively for Genepool and will not function elsewhere, so should not be distributed outside LBL.

Name: align2.BBMapPacBio
Shellscript: mapPacBio.sh
Description: Version of BBMap for long reads up to 6kbp. Designed for PacBio and Nanopore reads; uses alignment penalties weighted for PacBio's error model.
Notes: Multithreaded, high memory. Memory usage depends on the size of the reference and number of threads.

Name: align2.BBMapPacBioSkimmer
Shellscript: bbmapskimmer.sh
Description: Version of BBMap for mapping reads to all sites above a certain score threshold, rather than finding the single best mapping location. Uses alignment penalties weighted for PacBio's error model, as it was originally created to map Illumina reads to PacBio reads for error-correction.
Notes: Multithreaded, high memory. Memory usage depends on the size of the reference and number of threads.

Name: align2.BBSplit
Shellscript: bbsplit.sh
Description: Uses BBMap to map to multiple references simultaneously, and output one file per reference, containing all the reads that match it better than the other references. Used for metagenomic binning, distinguishing between closely-related organisms, and contamination removal.
Notes: See BBMap.

Name: align2.BBWrap
Shellscript: bbwrap.sh
Description: Allows multiple runs of BBMap on different input files without reloading the reference. Useful when the reference is very large.
Notes: See BBMap.

Name: jgi.CoveragePileup

Shellscript: pileup.sh
Description: Calculates coverage information from an unsorted or sorted sam or bam file. Outputs per-scaffold coverage, per-base coverage, binned coverage, normalized coverage, per-ORF coverage (using PRODIGAL's format), coverage histograms, stranded coverage, physical coverage, FPKMs, and various others.
Notes: Singlethreaded, high memory. TODO: Would not be overly difficult to make a multithreaded version using A_SampleMT, but would require locks or queues.

Name: driver.SummarizeCoverage
Shellscript: summarizescafstats.sh
Description: Summarizes the scafstats output of BBMap for evaluation of cross-contamination. The intended use is to map multiple libraries or assemblies, of different multiplexed organisms, to a concatenated reference containing one fused scaffold per organism. This will convert all of the resulting stats files (one per library) to a single text file, with multiple columns, indicating how much of the input hit the primary versus nonprimary scaffolds. See also BBMap, Pileup, SummarizeSealStats.
Notes: Singlethreaded, low memory.

Name: jgi.FilterByCoverage
Shellscript: filterbycoverage.sh.
Description: Filters an assembly by contig coverage, to remove contigs below a coverage cutoff, or with fewer than some percent of their bases covered. Uses coverage stats produced by BBMap or Pileup.
Notes: Singlethreaded, low memory.

Name: driver.MergeCoverageOTU
Shellscript: mergeOTUs.sh
Description: Merges coverage stats lines (from Pileup) for the same OTU, according to some custom naming scheme. See also CoveragePileup.
Notes: Singlethreaded, low memory.

Name: jgi.SamToEst
Shellscript: bbest.sh
Description: Calculates EST (expressed sequence tags) capture by an assembly from a sam file. Designed to use BBMap output generated with these flags: k=13 maxindel=100000 customtag ordered
Notes: Singlethreaded, low memory.

Name: assemble.Postfilter
Shellscript: postfilter.sh
Description: Maps reads, then filters an assembly by contig coverage. Intended to reduce misassembly rate of SPAdes by removing suspicious contigs. See also BBMap and FilterByCoverage.
Notes: Multithreaded, high memory.

Kmer Matching

Name: jgi.BBDukF
Shellscript: bbduk.sh
Description: Multipurpose tool for read preprocessing, which does adapter-trimming, quality-trimming, contaminant filtering, entropy filtering, sequence masking, quality score recalibration, format conversion, histogram generation, barcode filtering, gc filtering, kmer cardinality estimation, and many similar tasks.
Notes: Multithreaded, high memory. Memory usage depends on the size of the reference (roughly 20 bytes per kmer) and whether hdist or edist are set (they multiply memory consumption by a large factor); if no reference is loaded, little memory is needed.

Name: jgi.BBDuk2
Shellscript: bbduk2.sh
Description: Version of BBDuk that can do multiple kmer-based operations at once - left-trim, right-trim, filter, and mask.
Notes: See BBDuk.

Name: jgi.Seal
Shellscript: seal.sh
Description: Performs high-speed alignment-free sequence quantification or binning, by counting the number of long kmers that match between a read and a set of reference sequences. Designed for RNA-seq versus a transcriptome, metagenomic binning and abundance analysis, quantifying contamination, and similar. Very similar to BBDuk except that Seal associates each kmer with multiple reference sequences instead of just one, so it is superior in situations where multiple reference sequences may share a kmer. Unlike BBSplit, this supports unlimited read length. Can generate per-scaffold coverage, FPKMs when mapping to a transcriptome, and so forth. Also supports taxonomic classification.
Notes: Multithreaded, high memory. Memory usage depends on the size of the reference (roughly 30 bytes per kmer) and whether hdist or edist are set (they multiply memory consumption by a large factor).

Name: driver.SummarizeSealStats
Shellscript: summarizeseal.sh
Description: Summarizes the stats output of Seal for evaluation of cross-contamination. The intended use is to map multiple libraries or assemblies, of different multiplexed organisms, to a concatenated reference containing one fused scaffold per organism. This will convert all of the resulting stats files (one per library) to a single text file, with multiple columns, indicating how much of the input hit the primary versus nonprimary scaffolds. Also allows filtering of certain libraries to mask some classes of contamination. Because Seal supports arbitrarily-long sequences, this is a better choice than BBMap for evaluating assemblies. See also Seal, SummarizeCoverage.

Notes: Singlethreaded, low memory.

Kmer Counting

Name: jgi.LogLog
Shellscript: loglog.sh
Description: Estimates the number of unique kmers within a dataset to within ~10%.
Notes: Multithreaded, low memory. This can also be done with other programs such as BBDuk by adding the loglog flag.

Name: jgi.KmerCountExact
Shellscript: kmercountexact.sh
Description: Counts kmers in sequence data. Capable of outputting the kmers and their counts as fasta or 2-column tsv, as well as a frequency histogram. No kmer length limits.
Notes: Multithreaded, high memory.

Name: jgi.KmerNormalize (generally referred to as BBNorm)
Shellscript: bbnorm.sh, ecc.sh, khist.sh
Description: Uses a lossy data structure (count-min sketch) to perform kmer-based normalization, error-correction, and/or depth-binning on reads.
Notes: Multithreaded, high memory. BBNorm will never run out of memory; rather, as the amount of data increases, the accuracy decreases. Therefore you should always use all available memory for best accuracy. The error correction by Tadpole is superior, but Tadpole can run out of memory with large datasets.
Additional Shellscripts: KmerNormalize is called by 3 different shellscripts, which differ only in their default parameters (which can be overridden). bbnorm.sh does 2-pass normalization only; ecc.sh does error-correction only; and khist.sh only makes a kmer histogram, without ignoring the low-quality kmers (as is done by ecc and bbnorm). But, if add the flag "ecc" to bbnorm.sh and it will do error-correction also, and so forth - with the same parameters they are all identical.

Name: jgi.CalcUniqueness
Shellscript: bbcountunique.sh
Description: Generates a kmer uniqueness histogram, binned by file position. Designed to analyze library complexity, and determine how much sequencing is needed before reaching saturation. Outputs both single-read uniqueness and pair uniqueness.
Notes: Singlethreaded, high memory (around 100 bytes per read pair).

Name: jgi.SmallKmerFrequency
Shellscript: commonkmers.sh
Description: Prints the most common kmers in a sequence, their counts, and the sequence header. K is limited to 15.
Notes: Singlethreaded, low memory. Memory is proportional to 4^k, and is trivial for short kmers under 10.

Name: jgi.KmerCoverage
Shellscript: kmercoverage.sh
Description: Annotates reads with their kmer depth.
Notes: Deprecated. Multithreaded, high memory.

Name: jgi.CallPeaks
Shellscript: callpeaks.sh
Description: Calls peaks from a kmer frequency histogram, such as that from BBNorm or KmerCountExact. Also estimates genome size and other statistics.
Notes: Singlethreaded, low memory. Normally called automatically by programs that make the histogram. The peak-calling logic is not very sophisticated and could be improved.

Assembly

Name: assemble.Tadpole
Shellscript: tadpole.sh
Description: Very fast kmer-based assembler, designed for haploid organisms. Performs well on single cells, viruses, organelles, and in other situations with small genomes and potentially uneven or very high coverage. Also has modes for read error-correction and extension, instead of assembly; Tadpole's error-correction is superior to BBNorm's. No upper limit on kmer length. See also KmerCountExact, KmerCompressor, LogLog, BBMerge, KmerNormalize.
Notes: Multithreaded, high memory. Memory consumption is a strict function of the number of unique input kmers.

Name: assemble.TadpoleWrapper
Shellscript: tadwrapper.sh
Description: Generates multiple assemblies with Tadpole to estimate the optimal kmer length.
Notes: Multithreaded, high memory.

Name: assemble.KmerCompressor
Shellscript: kcompress.sh
Description: Generates a minimal fasta file containing each kmer from the input sequence exactly once. Optionally allows the inclusion only of kmers within a certain depth range. Arbitrary kmer set operations are possible via multiple passes. Very similar to an assembler.
Notes: Multithreaded, high memory. Contains a singlethreaded phase.

Name: jgi.AssemblyStats2
Shellscript: stats.sh
Description: Generates basic assembly statistics such as scaffold count, N50, L50, GC content, gap percent, etc. Also generates per-scaffold length and base content statistics, and can estimate BBMap's memory requirements for an assembly. See also StatsWrapper.
Notes: Singlethreaded, low memory.

Name: jgi.AssemblyStatsWrapper
Shellscript: statswrapper.sh
Description: Generates stats on multiple assemblies, allowing tab-delimited columns with one assembly per row, and only one header.
Notes: Singlethreaded, low memory.

Name: jgi.CountGC
Shellscript: countgc.sh
Description: Counts GC content of reads or scaffolds.
Notes: Deprecated; superceded by AssemblyStats.

Name: jgi.FungalRelease
Shellscript: fungalrelease.sh
Description: Reformats a fungal assembly for release. Also creates contig and agp files.
Notes: Singlethreaded, low memory.

Taxonomy

Name: tax.FilterByTaxa
Shellscript: filterbytaxa.sh
Description: Filters sequences according to their taxonomy, as determined by the sequence name. Sequences should be labeled with a gi number, NCBI taxID, or species name. Relies on NCBI taxdump processed using taxtree.sh and gitable.sh.
Notes: Singlethreaded, low memory.

Name: tax.RenameGiToNcbi
Shellscript: gi2taxid.sh
Description: Renames sequences with gi numbers to NCBI taxa IDs. This allows taxonomy processing without a gi number lookup.
Notes: Singlethreaded, high memory. TODO: Can be made low memory if slightly altered to accept gitable.int1d files.

Name: tax.GiToNcbi
Shellscript: gitable.sh
Description: Condenses gi_taxid_nucl.dmp from NCBI taxdmp to gitable.int1d, a more efficient representation, used by other tools for translating gi numbers to taxID's. See also TaxTree.
Notes: Singlethreaded, high memory.

Name: tax.SortByTaxa
Shellscript: sortbytaxa.sh
Description: Sorts sequences into taxonomic order by some depth-first traversal of the Tree of Life as defined by NCBI taxdump. Sequences must be labelled with taxonomic identifiers.

Notes: Singlethreaded, high memory.

Name: tax.SplitByTaxa
Shellscript: splitbytaxa.sh
Description: Splits sequences according to their taxonomy, as determined by the sequence name. Sequences should be labeled with a gi number, NCBI taxID, or species name.
Notes: Multithreaded, high memory. If the number of threads is restricted and the sequences are fairly short, regardless of the total number, this may be run using low memory.

Name: tax.PrintTaxonomy
Shellscript: taxonomy.sh
Description: Prints the full taxonomy of a given taxonomic identifier (such as homo_sapiens).
Notes: Singlethreaded, low memory.

Name: tax.TaxTree
Shellscript: taxtree.sh
Description: Creates tree.taxtree from names.dmp and nodes.dmp, which are in NCBI tax dump. The taxtree file is needed for programs that can deal with taxonomy, like Seal and SortByTaxa.
Notes: Singlethreaded, high memory.

Name: driver.ReduceSilva
Shellscript: reducesilva.sh
Description: Reduces Silva entries down to one entry per specified taxonomic level. Designed to increase the efficiency of operations like mapping, in which having thousands of substrains represented are not helpful.
Notes: Singlethreaded, low memory.

Cross-Contamination

Name: jgi.SynthMDA
Shellscript: synthmda.sh
Description: Generates synthetic reads following an MDA-amplified single cell's coverage distribution. Designed for single-cell assembly and analysis optimization. See also CrossContaminate, RandomReads.
Notes: Singlethreaded, medium memory (needs around 4GB).

Name: jgi.CrossContaminate
Shellscript: crosscontaminate.sh
Description: Generates synthetic cross-contaminated files from clean files. Intended for use with synthetic reads generated by SynthMDA or RandomReads. Designed to evaluate the effects of cross-contamination on assembly, and the efficacy of decontamination methods.
Notes: Singlethreaded, high memory.

Name: jgi.DecontaminateByNormalization
Shellscript: decontaminate.sh, crossblock.sh
Description: Removes contaminant contigs from assemblies of multiplexed libraries via normalization and mapping.
Notes: Multithreaded, high memory. Mostly a wrapper for other programs like BBMap, BBNorm, and FilterByCoverage.

Deduplication and Clustering

Name: jgi.Dedupe
Shellscript: dedupe.sh
Description: Accepts one or more files containing sets of sequences (reads or scaffolds). Removes duplicate sequences, which may be specified to be exact matches, fully contained subsequences, or subsequence within some edit distance. Can also find overlapping sequences and group them into clusters based on transitive reachability; for example, clustering full-length 16S PacBio reads by species.
Notes: Multithreaded, high memory. This program has a jni mode which increases speed dramatically if an edit distance is used.

Name: jgi.Dedupe2
Shellscript: dedupe2.sh
Description: Allows more kmer seeds than Dedupe. This will be automatically called by Dedupe if needed.
Notes: See Dedupe.

Name: jgi.DedupeByMapping
Shellscript: dedupebymapping.sh
Description: Removes duplicate reads or read pairs from a sam/bam file based on mapping coordinates. The sam file does not need to be sorted.
Notes: Singlethreaded, high memory.

Name: clump.Clumpify
Shellscript: clumpify.sh
Description: Rearranges unsorted reads into small clumps of reads, such that each clump shares a kmer, and thus probably overlaps. Can also create consensus sequence from these clumps.
Notes: Multithreaded, low or high memory. Memory consumption may be made arbitrarily small by using a user-specified number of temp files for bucket-sorting. By default, it will try to grab all available memory.

Read Merging

Name: jgi.BBMerge
Shellscript: bbmerge.sh, bbmerge-auto.sh
Description: Merges paired reads into single reads by overlap detection. With sufficient coverage, can also merge nonoverlapping reads by kmer extension.
Notes: Multithreaded, low memory. If kmers are used (for extension or error-correction), it will need much more memory, and the shellscript bbmerge-auto.sh should be used, which tries to acquire all available RAM. This program has a jni mode which increases speed by around 20%.

Name: jgi.MateReadsMT
Shellscript: bbmergegapped.sh
Description: Uses gapped kmers to merge nonoverlapping reads.
Notes: Deprecated; superceded by BBMerge.

Synthetic Read Generation and Benchmarking

Name: align2.RandomReads3
Shellscript: randomreads.sh
Description: Generates random synthetic reads from a reference genome, annotated with their genomic origin. Allows precise customization of things like insert size and synthetic mutation type, sizes, and rates. Read names are parsed by various other BBTools to grade accuracy.
Notes: Singlethreaded, high memory.

Name: jgi.FakeReads
Shellscript: bbfakereads.sh
Description: Generates fake read pairs from ends of contigs or single reads. Intended for use in generating a fake LMP library for scaffolding, using additional information like another assembly, or very long reads (like PacBio). This can also be accomplished with RandomReads.
Notes: Singlethreaded, low memory.

Name: align2.GradeSamFile
Shellscript: gradesam.sh
Description: Grades the accuracy of an aligner (such as BBMap) by parsing the output. The reads must be single-ended and annotated as though generated by RandomReads.
Notes: Singlethreaded, low memory.

Name: align2.MakeRocCurve
Shellscript: samtoroc.sh
Description: Creates an ROC plot (technically, true-positive versus false-positive) from a sam or bam file of mapped reads. The reads should be single-ended with headers generated by RandomReads.
Notes: Singlethreaded, low memory.

Name: jgi.AddAdapters
Shellscript: addadapters.sh
Description: Randomly adds adapters to a file, or grades a trimmed file. The input is a set of reads, paired or unpaired. The output is those same reads with adapter sequence replacing some of the bases in some reads. For paired reads, adapters are located in the same position in read1 and read2. This is designed for benchmarking adapter-trimming software (such as BBDuk), and evaluating methodology. Adapters can alternately be added by RandomReads, in which case insert size is used to determine where the adapters go.
Notes: Singlethreaded, low memory.

Name: jgi.GradeMergedReads
Shellscript: grademerge.sh
Description: Grades the accuracy of a read-merging program (such as BBMerge) by parsing the output. The reads must be annotated by their insert size. This can be done by generating them with RandomReads and renaming with RenameReads
Notes: Singlethreaded, low memory.

Name: align2.PrintTime
Shellscript: printtime.sh
Description: Prints time elapsed since last called on the same file.
Notes: Singlethreaded, low memory.

16S, Primers, and Amplicons

Name: jgi.FindPrimers
Shellscript: msa.sh
Description: Aligns a query sequence to reference sequences. Outputs the best matching position per reference sequence. If there are multiple queries, only the best-matching query will be used. Designed to find primer binding sites in a sequence that may contain indels, such as a PacBio read, using a MultiStateAligner.
Notes: Singlethreaded, high memory. TODO: Could easily be made multithreaded using A_SampleMT.

Name: jgi.CutPrimers
Shellscript: cutprimers.sh
Description: Cuts out sequences corresponding to primers identified in sam files. Used in conjunction with FindPrimers (msa.sh).
Notes: Singlethreaded, low memory.

Name: jgi.IdentityMatrix
Shellscript: idmatrix.sh
Description: Generates an identity matrix via all-to-all alignment of sequences in a file. Intended for 16S or other amplicon analysis. See also CorrelateIdentity.

Notes: Multithreaded, high-memory. Time complexity is O(N^2) with the number of reads.

Name: driver.CorrelateIdentity
Shellscript: matrixtocolumns.sh
Description: Transforms two matched identity matrices into 2-column format, one row per entry, one column per matrix. Designed for comparing different 16S subregions. See also IdentityMatrix, FindPrimers.
Notes: Singlethreaded, high memory. The actual amount of memory just depends on the matrix sizes.

Barcodes

Name: jgi.CountBarcodes
Shellscript: countbarcodes.sh
Description: Counts the number of reads with each barcode. Assumes read names have the barcode at the end.
Notes: Singlethreaded, low memory.

Name: jgi.CorrelateBarcodes
Shellscript: filterbarcodes.sh
Description: Filters barcodes by quality, and generates quality histograms. See also MergeBarcodes.
Notes: Singlethreaded, low memory.

Name: jgi.MergeBarcodes
Shellscript: mergebarcodes.sh
Description: Concatenates barcodes and barcode quality onto read names. Designed to analyze the effects of barcode quality on library misassignment. See also CorrelateBarcodes.
Notes: Singlethreaded, low memory.

Name: jgi.RemoveBadBarcodes
Shellscript: removebadbarcodes.sh
Description: Removes reads with improper barcodes - either with no barcode, or a barcode containing a degenerate base.
Notes: Singlethreaded, low memory. Mostly a test case for extending BBTool_ST.

Filtering and Demultiplexing

Name: jgi.DemuxByName
Shellscript: demuxbyname.sh
Description: Demultiplexes reads into multiple files based on their name, by matching a suffix or prefix.

Notes: Singlethreaded, low memory.

Name: jgi.FilterBySequence
Shellscript: filterbysequence.sh
Description: Filters reads by exact sequence match. Allows case-sensitive or insensitive matches, and reverse-complement matches or only forward matches.
Notes: Multithreaded, high memory.

Name: driver.FilterReadsByName
Shellscript: filterbyname.sh
Description: Filters reads by name. Allows substring matching, though that is much slower.
Notes: Singlethreaded, low memory.

Name: jgi.FilterReadsWithSubs
Shellscript: filtersubs.sh
Description: Filters a sam file to select only reads with substitution errors for bases with quality scores in a certain interval. Used for manually examining specific reads that may contain incorrectly calibrated quality scores.
Notes: Singlethreaded, low memory.

Name: jgi.GetReads
Shellscript: getreads.sh
Description: Fetches the reads with specified numeric IDs (unrelated to their names). The first read (or pair) in a file has ID 0, the second read (or pair) has ID 1, etc.
Notes: Singlethreaded, low memory.

Name: driver.EstherFilter
Shellscript: estherfilter.sh
Description: BLASTs queries against reference, and filters out hits with scores less than 'cutoff'.
Notes: All the work is done by blastall, which dictates the performance characteristics.

JGI-Exclusive Preprocessing Wrappers

Name: jgi.BBQC
Shellscript: bbqc.sh
Description: Wrapper for various read preprocessing operations.
Notes: Deprecated; superceded by RQCFilter. Designed exclusively for Genepool and will not function elsewhere, so should not be distributed outside LBL.

Name: jgi.RQCFilter
Shellscript: rqcfilter.sh
Description: Acts as a wrapper/pipeline for read preprocessing. Performs quality-trimming, artifact removal, linker-trimming, adapter trimming, spike-in removal, vertebrate contaminant

removal, microbial contaminant removal, and generates various histogram and statistics files used by RQC.
Notes: Multithreaded, high memory. Currently requires 39500m RAM and thus can run on a 40G node, but it's recommended to submit it exclusive, as all stages are fully multithreaded. Designed exclusively for Genepool and will not function elsewhere, so should not be distributed outside LBL.

Shredding and Sorting

Name: jgi.Shred
Shellscript: shred.sh
Description: Shreds long sequences into shorter sequences, with overlap length and variable-length options. See also Fuse.
Notes: Singlethreaded, low memory.

Name: jgi.FuseSequence
Shellscript: fuse.sh
Description: Fuses sequences together, padding junctions with Ns. Does not support total length greater than 2 billion. Designed for use with Seal or BBDuk to make kmer tracking for a given genome more efficient. See also Shred.
Notes: Singlethreaded, high memory.

Name: jgi.Shuffle
Shellscript: shuffle.sh
Description: Reorders reads randomly, keeping pairs together. Also supports some sorting operations, like alphabetically by name or by sequence.
Notes: Singlethreaded, high memory. All operations are in-memory.

Non-Sequence-Related

Name: Calcmem - Shellscript Only
Shellscript: calcmem.sh
Description: Calculates available memory for other shellscripts. Designed for Genepool but works fine on many Linux configurations.
Notes: If java is being killed for allocating too much memory, this is the script to fix.

Name: fileIO.TextFile
Shellscript: textfile.sh
Description: Displays contents of a text file, optionally between a start and stop line. Useful mainly in Windows where there are few command-line utilities.
Notes: Singlethreaded, low memory.

Name: driver.CountSharedLines
Shellscript: countsharedlines.sh
Description: Counts the number of lines shared between sets of files. One output file will be printed for each input file. For example, an output file for a file in the 'in1' set will contain one line per file in the 'in2' set, indicating how many lines are shared. This is not designed for sequence data, but more for things like sequence names or organism names. See filterlines.sh for actually filtering shared lines in a more normal fashion.
Notes: Singlethreaded, low memory.

Name: driver.FilterLines
Shellscript: filterlines.sh
Description: Filters lines by exact match or substring. This is not designed for sequence data, but for things like sequence names or organism names.
Notes: Singlethreaded, low memory.

Other Tools

Name: jgi.A_SampleMT
Shellscript: a_sample_mt.sh
Description: Does nothing. Serves as a template for easily making new BBTools by dropping in code.
Notes: Multithreaded, high memory. Be sure to modify the shellscript line "   freeRam 4000m 84" as needed. The first is the amount of memory used if available memory cannot be calculated, the second is the percentage of free memory to use if it can be calculated.

Name: jgi.BBMask
Shellscript: bbmask.sh
Description: Masks sequences of low-complexity, or containing repeat kmers, or covered by mapped reads. Used to make masked versions of human, cat, dog, and mouse genomes; these are used for filtering vertebrate contamination from fungal/plant/microbial data without risk of false-positive removals.
Notes: Multithreaded, high memory. Uses around 2 bytes per reference base.

Name: jgi.CalcTrueQuality
Shellscript: calctruequality.sh
Description: Generates matrices used for quality-score recalibration. Requires one or more mapped sam files as input. The actual recalibration is done with another program such as BBDuk.
Notes: Multithreaded, low memory.

Name: jgi.MakeChimeras
Shellscript: makechimeras.sh

Description: Makes chimeric sequences by randomly fusing together nonchimeric sequences. Designed for analyzing chimera removal effectiveness.
Notes: Singlethreaded, low memory.

Name: jgi.PhylipToFasta
Shellscript: phylip2fasta.sh
Description: Transforms interleaved phylip to fasta.
Notes: Singlethreaded, high memory.

Name: jgi.MakeLengthHistogram
Shellscript: readlength.sh
Description: Makes a length histogram of sequences.
Notes: Singlethreaded, low memory. Can also be accomplished with Reformat or BBDuk, but with less flexibility.

Name: jgi.ReformatReads
Shellscript: reformat.sh
Description: Reformats sequence data into another format, such as interleaved ASCII-33 fastq to twin-file ASCII-64. Also supports a huge collection of simple optional operations, like trimming, filtering, reverse-complementing, modifying read names, and modifying read sequence.
Notes: Singlethreaded, low memory.

Name: pacbio.RemoveAdapters2
Shellscript: removesmartbell.sh
Description: Detects or removes SmartBell adapters from PacBio reads, by aligning the adapter using a customized version of the MultiStateAligner.
Notes: Multithreaded, low memory.

Name: jgi.RenameReads
Shellscript: rename.sh
Description: Renames reads according to some specified prefix. Can also rename by insert size or mapping location.
Notes: Singlethreaded, low memory.

Name: jgi.SplitPairsAndSingles
Shellscript: repair.sh, bbsplitpairs.sh
Description: Separates paired reads into files of pairs and singletons by removing reads that are shorter than a min length, or have no mate. Can also reorder arbitrarily-ordered reads in files where the pairing order was desynchronized. See also Reformat's vint flag.
Notes: Singlethreaded, low or high memory. All operations are low-memory except reordering arbitrarily disordered files, which is optional.

Name: jgi.SplitNexteraLMP

Shellscript: splitnextera.sh
Description: Trims and splits Nextera LMP libraries into subsets based on linker orientation: LMP, fragment, unknown, and singleton.
Notes: Singlethreaded, low memory. TODO: Should be reimplemented using A_SampleMT.

Name: jgi.SplitSamFile
Shellscript: splitsam.sh
Description: Splits a sam file into three files: Plus-mapped reads, Minus-mapped reads, and Unmapped.
Notes: Singlethreaded, low memory.

Name: fileIO.FileFormat
Shellscript: testformat.sh
Description: Tests the format of a sequence-containing file. Determines format (fasta, fastq, etc), quality encoding, compression type, interleaving, and read length. All BBTools use this to determine how to process a file.
Notes: Singlethreaded, low memory.

Name: jgi.TranslateSixFrames
Shellscript: translate6frames.sh
Description: Translates nucleotide sequences to all 6 amino acid frames, or amino acids to a canonical nucleotide representation.
Notes: Singlethreaded, low memory.

Template

Name:
Shellscript:
Description:
Notes:

BBTools Table Of Contents
==================

DESCRIPTION
===========

BBTools is a robust suite of fast tools for bioinformatics analysis. It is written in Java and usable on any operating system.

BBTools is developed at the Department of Energy's Joint Genome Institute and is distributed open-source, free for unlimited use by anyone. For more details see license.txt.

Additional documentation is in the bbmap/docs folder.

INSTALLATION

Extract the contents of BBMap_(version).tar.gz with

```
$ tar -xzvf BBMap_(version).tar.gz
```

This extracts everything to a new directory named bbmap in the current directory.
For ease of use, add the shellscripts to your path, e.g.

```
$ export PATH=$PATH:/path/to/bbmap/
```

Requires Java 7 Runtime Environment or later (8+ is preferred). BBTools is developed and tested with Oracle's JDK 8, and in some cases memory allocation is slightly different under OpenJDK. BBTools can also benefit from samtools, sambamba, pigz, pbzip2, lbzip2, bzip2, and bgzip, though none are required. Specifically, .bam files can only be read or written if samtools or sambamba is in the path; .bz2 files require pbzip2, lbzip2, or bzip2; block-gzipping (often used for vcf files) requires bgzip in the path; and compression and decompression of .gz files is much faster with pigz in the path.

Check that BBTools is working with:

```
$ bbversion.sh
(returns version)
```

Test stats.sh on the PhiX reference included in the bbmap/resources folder:

```
$ stats.sh in=bbmap/resources/phix174_ill.ref.fa.gz


A       C       G       T       N       IUPAC   Other   GC      GC_stdev
0.2399  0.2144  0.2326  0.3130  0.0000  0.0000  0.0000  0.4471  0.0000


Main genome scaffold total:             1
Main genome contig total:               1
Main genome scaffold sequence total:    0.005 MB
Main genome contig sequence total:      0.005 MB        0.000% gap
Main genome scaffold N/L50:             1/5.386 KB
Main genome contig N/L50:               1/5.386 KB
Main genome scaffold N/L90:             1/5.386 KB
Main genome contig N/L90:               1/5.386 KB
Max scaffold length:                    5.386 KB
Max contig length:                      5.386 KB
```

```
Number of scaffolds > 50 KB:           0
% main genome in scaffolds > 50 KB:     0.00%


Minimum        Number        Number        Total         Total         Scaffold
Scaffold       of            of            Scaffold      Contig        Contig
Length         Scaffolds     Contigs       Length        Length        Coverage
+--------      -------------- -------------- -------------- -------------- --------
   All                1            1         5,386         5,386   100.00%
   5 KB               1            1         5,386         5,386   100.00%
```

BBTOOLS SCRIPTS

Bash shell script wrappers are provided to make tools easier to run in Linux, though they are not strictly necessary. For example, the command:
reformat.sh in=x.fastq out=x.fasta
...could be executed on any operating system without using the script, like this:
java -Xmx1g -ea -cp /path/to/bbmap/current/ jgi.ReformatReads in=x.fastq out=y.fasta

This is a list of the shell scripts contained in BBTools with a brief description. Scripts not on this list are not intended to be used for analysis.

A detailed description and parameters for each tool are available by running the shell script with no parameters.

(e.g. $ bbcms.sh)

Script	Purpose	Comment
bbcms.sh	Performs error correction using a Count-Min Sketch	Intended for metagenome assembly assembly
bbcountunique.sh	Counts unique kmers in reads	
bbduk.sh	Trims, filters or masks reads using kmers	
bbmap.sh	Splice-aware aligner for short reads	

bbmapskimmer.sh	BBMap version designed for high levels of multimapping	
bbmask.sh	Masks references based on various things, such as sequence complexity	
bbmerge.sh	Merges overlapping paired reads	
bbmerge-auto.sh	Same as bbmerge, but tries to allocate all memory on the node	Use this version for kmer operations like extend
bbnorm.sh	Normalizes reads based on coverage	Mainly for use prior to single-cell assembly
bbsplit.sh	BBMap version that maps to multiple references simultaneously	Intended for decontamination; similar to Seal
bbversion.sh	Prints the version of BBTools	
bbwrap.sh	Wraps BBMap to process many files using same reference	Saves time by loading the index only once
calctruequality.sh	Allows recalibration of quality scores from mapped reads	This generates the correction matrix; BBDuk does the recalibration
callgenes.sh	Fast prokaryotic gene caller	Integrated into BBSketch
callvariants.sh	Fast variant caller	
callvariants2.sh	Same as callvariants.sh with the "multisample" flag	
clumpify.sh	Shrinks compressed fastq files, and can remove duplicate reads	Also supports error correction
comparesketch.sh	Compares sketches locally, without using a sketch server	
crossblock.sh	Alias for decontaminate.sh	
cutgff.sh	Cuts out features defined by gff file	E.g, generates one fasta entry per gene from a gff and an assembly
cutprimers.sh	Cuts out subregions of ribosomes	Mainly for 16S analysis
decontaminate.sh	Pool-level decontamination for single-cell MDA-amplified genomes	
dedupe.sh	Removes duplicate and fully-contained sequences	Can also be used to cluster 16S sequences
dedupe2.sh	Version of dedupe that supports more hash keys for greater sensitivity	
dedupebymapping.sh	Deduplicates reads based on mapping coordinates	
demuxbyname.sh	Demultiplexes based on sequences headers	

filterbyname.sh	Filters based on sequence headers	
filterbytaxa.sh	Filters sequences based on taxonomic classification	Used with NCBI datasets
filterbytile.sh	Removes reads that are in low quality areas on flowcell	
filterqc.sh	Part of JGI's fastq filtering pipeline	
filtersam.sh	Filters sam files to remove reads with multiple unsupported mismatches	Designed for NovaSeq
gitable.sh	Used to process NCBI taxonomy data	
khist.sh	Alias for bbnorm.sh with flags for making a kmer frequency histogram	
kmercountexact.sh	Counts kmers and produces a histogram	Uses more memory than BBNorm but allows exact counts
kmercountmulti.sh	Cardinality estimation over multiple kmer lengths	Uses LogLog; does not produce a histogram
mapPacBio.sh	BBMap version designed for PacBio or Nanopore reads	Reads longer than 5kbp get broken into 5kbp shreds
mergesketch.sh	Allows multiple sketches to be combined	
msa.sh	Alignment tool	Used with cutprimers.sh to cut subsections out of 16s
mutate.sh	Generates synthetic genomes by randomly mutating the input	
muxbyname.sh	Multiplex multiple files, renaming sequences based on input file name	Opposite of demuxbyname.sh
partition.sh	Splits a sequence file into multiple files	
pileup.sh	Calculates coverage from sam files	
plotflowcell.sh	Produces statistics about flowcell positions	
processhi-c.sh	Custom trimming for hi-C reads	In development
randomreads.sh	Generates synthetic data from real genome reference	Highly customizable
readqc.sh	Short read quality report	Alternative to fastqc
reformat.sh	Converts sequence files to another format	Has many additional options, includes subsampling
rename.sh	Renames sequences in various ways, such as adding a prefix	

repair.sh	Fixes broken pairing in fastq files	
representative.sh	Makes a smaller subset of a reference dataset by eliminating redundancy	Designed for use with BBSketch output
rqcfilter2.sh	Filtering pipeline used at JGI	portal.nersc.gov/dna/microbial/assembly/bushnell/RQCFilterData.tar
seal.sh	Counts kmer matches between query and reference sequences	
sendsketch.sh	Fast taxonomic classifier using webservers at JGI	
shred.sh	Breaks sequences into shorter, fixed-length pieces	
shuffle.sh	Randomly reorders input file	Crashes if input doesn't fit in memory
shuffle2.sh	Randomly reorders input file	Supports larger files, but output might be less random
sketch.sh	Makes reference sketches on a per-TaxID basis	
sketchblacklist.sh	Makes sketch blacklists of common kmers	
sortbyname.sh	Sorts sequences by name, length, quality, taxa, and other things	
summarizequast.sh	Generates box plots for multiple quast reports	
tadpipe.sh	Preprocessing and assembly pipeline using tadpole	
tadpole.sh	Fast short read assembler	
tadwrapper.sh	Runs Tadpole with multiple kmer lengths to select the best assembly	
taxserver.sh	Starts taxonomy and sketch servers	
testformat.sh	Determines if file is fasta, fastq, interleaved, etc. by reading first few lines	
testformat2.sh	Generates extensive statistics by reading the full file	
translate6frames.sh	Translates nucleotide sequence into amino acid sequence in all frames	
vcf2gff.sh	Converts vcf format to gff format	

AUTHORS
=======

 * Brian Bushnell (bbushnell@lbl.gov)
 * Bryce Foster (brycefoster@lbl.gov)
 * Jon Rood
 * Shijie Yao (syao@lbl.gov)

Citation
========

Please see bbmap/docs/citation.txt for information about citation.

These instructions are for JGI internal use.

If a taxonomy server or Sketch server dies:

Look at /global/projectb/sandbox/gaag/bbtools/server/start*Server.sh, which includes:

startTaxServer.sh
startRefseqServer.sh
startProteinServer.sh
startNtServer.sh
startSilvaServer.sh

Each file indicates where to run the script. So, for example, startTaxServer.sh says: #Run this on gpweb25
ssh to the indicated machine, change directory to /global/projectb/sandbox/gaag/bbtools/server/, and run startTaxServer.sh.
It does not matter which user you are logged in as. But if you run into permission problems, make a copy of the script somewhere else and run it there.

To update taxonomy, go to /global/projectb/sandbox/gaag/bbtools/tax
1) Create a new directory
2) Copy the old shellscripts from "latest"
3) Execute the shellscripts
4) Point the "latest" symlink to the new directory after execution finishes
5) Restart the taxonomy server

For example:
cd /global/projectb/sandbox/gaag/bbtools/tax/

```
mkdir feb13_2019
cp latest/*.sh feb13_2019
cd feb13_2019
sh fetchOuter.sh 1>fetch.o 2>&1 &
#(wait until it finishes; it will create a file called "finished" when complete)
cd ..
ln -sfn feb13_2019 latest
#(restart the taxonomy server)
```

BBTools are sensitive to filename extensions. For example, this command:

```
reformat.sh in=reads.fq out=reads.fa.gz
```

...will convert reads from fastq format to gzipped fasta. The recognized sequence file extensions are as follows:

fastq (fq)
fasta (fa, fna, fas, ffn, frn, seq, fsa, faa)
sam
bam [requires samtools]
qual
scarf [input only]
phylip [input only; only supported by phylip2fasta.sh]
header [output only]
oneline [tab delimited 2-column: name and bases]
embl [input only]
gbk [input only]

The recognized compression extensions:

gzip (gz) [can be accelerated by pigz]
zip
bz2 [requires bzip2 or pbzip2 or lbzip2]
fqz [requires fqz_comp]

In order to stream using standard in or standard out, it is recommended to include the format. For example:

```
cat data.fq.gz | reformat.sh in=stdin.fq.gz out=stdout.fa > file.fa
```

This allows the tool to determine the format. Otherwise it will revert to the default.

BBTools can usually determine the type of sequence data by examining the contents. To test this, run:

```
fileformat.sh in=file
```

...which will print the way the data is detected, e.g. Sanger (ASCII-33) quality, interleaved, etc. These can normally be overridden with the "qin" and "interleaved" flags.

When BBTools are processing gzipped files, they may, if possible, attempt to spawn a pigz process to accelerate it. This behavior can be forced with the "pigz=t unpigz=t" flags, or prevented with "pigz=f unpigz=f"; otherwise, the default behavior depends on the tool. In some cluster configurations, and some Amazon nodes, spawning a process may cause the program to killed with an indication that it used too much virtual memory. I recommend pigz be enabled unless that scenario occurs.

The most recent extension added is "header". You can use it like this:

```
reformat.sh in=reads.fq out=reads.header minlen=100
```

That will create a file containing headers of reads that pass the "minlen" filter.

BBTools Config File Readme
Written by Brian Bushnell
Last updated May 12, 2015

A config file is a text file with a set of parameters that will be added to the command line.
The format is one parameter per line, with the # symbol indicating comments.
To use a config file, use the config=file flag. For example, take BBDuk:

```
bbduk.sh in=reads.fq out=trimmed.fq ref=ref.fa k=23 mink=11 hdist=1 tbo tpe
```

That is equivalent to:

```
bbduk.sh in=reads.fq out=trimmed.fq ref=ref.fa config=trimadapters.txt
```

...if trimadapters.txt contained these lines:

```
k=23
mink=11
hdist=1
tbo
tpe
```

Any parameter placed AFTER the config file will override the same parameter if it is in the config file.
For example, in this case k=20 will be used:

```
bbduk.sh in=reads.fq out=trimmed.fq ref=ref.fa config=trimadapters.txt k=20
```

But in this case, k=23 will be used, from the config file:

```
bbduk.sh in=reads.fq out=trimmed.fq ref=ref.fa k=20 config=trimadapters.txt
```

What are config files for? Well, mainly, to overcome difficulties like whitespace in file paths, or command lines that are too long.

There are some example config files in bbmap/config/. They are not used unless you specifically tell a program to use them.

BBMap/BBTools readme
Written by Brian Bushnell
Last updated January 2, 2018

The BBTools package is primarily devloped by Brian Bushnell, with some optional JNI and MPI components written by Jonathan Rood.
Some parts have also been written or modified by Shijie Yao, Alex Copeland, and Bryce Foster.

Citation:

Please see citation.txt

License:

The BBTools package is open source and free to use with no restrictions. For more information, please read Legal.txt and license.txt.

Documentation:

Documentation is in the /bbmap/docs/ directory, and in each tool's shellscript in /bbmap/.
readme.txt: This file.
UsageGuide.txt: Contains basic installation and usage information. Please read this first!
ToolDescriptions.txt: Contains a list of many BBTools, a description of what they do, and their hardware requirements.
compiling.txt: Information on compiling JNI code.
readme_config.txt: Usage information about config files.
readme_filetypes.txt: More detailed information on file formats supported by BBTools.
changelog.txt: List of changes by version, and current known issues.

Tool-specific Guides:

Some tools have specific guides, like BBDukGuide.txt. They are in /bbmap/docs/guides/. For complete documentation of a tool, I recommend that you read UsageGuide.txt first (which covers the shared functionality of all tools), then the tool's specific guide if it has one (such as ReformatGuide.txt), then the tool's shellscript (such as reformat.sh) which lists all of the flags.

Pipelines:

/bbmap/pipelines/ contains shellscripts. These are different than the ones in /bbmap/, which are wrappers for specific tools. The pipelines do not print a help message and do not accept any arguments. They are given to provide examples of the command lines and order of tools used to accomplish specific tasks.

Resources:

/bbmap/resources/ contains various data files. Most are fasta contaminant sequences. For more information see /bbmap/resources/contents.txt.

If you have any questions not answered in the documentation, please look at the relevant SeqAnswers thread (linked from here: http://seqanswers.com/forums/showthread.php?t=41057) and post a question there if it is not already answered. You can also contact JGI's BBTools team at bbtools@lbl.gov, or me at bbushnell@lbl.gov. But please read the documentation first.

Special thanks for help with shellscripts goes to:
Alex Copeland (JGI), Douglas Jacobsen (JGI/NERSC), Bill Andreopoulos (JGI), sdriscoll (SeqAnswers), Jon Rood (JGI/NERSC), and Elmar Pruesse (UC Denver).

Special thanks for helping to support BBTools goes to Genomax (SeqAnswers).