BBtools

A suite of command line utilities for sequence quality control, mapping, assembly, and kmer based analysis

Written by Brian Bushnell.

These examples are maintained by Adam Rivers.

As the BBtools website describes it:

> BBTools is a suite of fast, multithreaded bioinformatics tools designed for analysis of DNA and RNA sequence data. BBTools can handle common sequencing file formats such as fastq, fasta, sam, scarf, fasta+qual, compressed or raw, with autodetection of quality encoding and interleaving. It is written in Java and works on any platform supporting Java, including Linux, MacOS, and Microsoft Windows and Linux; there are no dependencies other than Java (version 7 or higher). Program descriptions and options are shown when running the shell scripts with no parameters.

> BBTools is open source and free for unlimited use, and is used regularly by DOE JGI and other institutions around the world.

> The BBTools suite includes programs such as:
>
> - bbduk – filters or trims reads for adapters and contaminants using k-mers
> - bbmap – short-read aligner for DNA and RNA-seq data
> - bbmerge – merges overlapping or nonoverlapping pairs into a single reads
> - reformat – converts sequence files between different formats such as fastq and fasta

For more information see the BBtools website and the BBtools user guide. Support is available from a number of threads at the SeqAnswers site. A listing of those threads is provided here.

The package contains 125 scripts and programs but this examples folder highlights some of the most common use cases.

Directory contents

1. `bbmap_examples` - A directory with bbmap examples

- `basic_mapping` - A directory with a shell script that generates an index then maps reads to the index

    - `bbmap_basic_mapping.sh` - example shell script
    - `results` - Data files generated by this analysis will be placed in this directory. This format is used in other examples.

2. `bbduk_examples` - A directory with examples of BBDuk

- `adaptor_trimming` - A directory with a shell script that trims sequencing adaptors from fastq formatted reads
    - `bbduk_adaptor_trimming.sh` - example shell script
- `quality_trimming` - A directory with a shell script that trims low quality bases
    - `bbduk_quality_trimming.sh` - example shell script
- `kmer filtering` - A directory with a shell script that removes reads containing kmers from a reference. Often used to remove PhiX spike-in sequences that have made it past Illumina barcode based removal.
    - `bbduk_kmer_filtering.sh` - example shell script
- `histogram_generation` - A directory with a shell script that generates quality histograms
    - `bbduk_histogram_generation.sh` - example shell script

3. `bbmerge_examples` - A directory containing examples for the multithreaded read-merging tool bbmerge

- `basic_merging` - A directory with a shell script that merges overlapping reads and creates an insert size histogram
    - `bbmerge_basic_merging.sh` - example shell script
- `error_correction` - A directory with a shell script that corrects sequencing errors by mate-pair overlap then returns the reads unmerged
    - `bbmerge_error_correction.sh` - example shell scripts

Running notes

- For simplicity I have created the script `env.sh` that creates the top level path directory ($bbtoolsExamplesDir). By sourcing this file `source env.sh` you can run the example scripts from different directories.
- `setup.sh` is a maintenance script that can be used with the `build` or `clean` option to run all examples and place the data into the results directories or delete all data from the results directories.
- If an example script is run individually, progress and statistics will be written to the screen through the standard error stream. That output can be seen in the `terminaloutput.txt` file of the `results` directory.

Detailed tutorial instructions

Since we are going to be running computations lets request an interactive node from the job scheduling system. ```bash
`srun --pty -p short -t 01:00:00 -n 16 -N 1 /bin/bash -I`

```
You can begin exploring the functionality of BBtools by copying the example
directory into your own home directory:
```

```bash
cp -r /scinet01/gov/usda/ars/scinet/project/training_files/bbtools "$HOME"
```

From there you can examine the output of the results directories for each example. These results are also available at the original location for reference.

First, start by "sourcing" the `env.sh` script. Doing this this sets up a variable called `$bbtoolsExamplesDir` that tells the example scripts where the bbtools examples directory is located.

```
source env.sh
```

Since we are going to be running these example scripts one-by-one lets remove all the data that is currently in the results directories. Running a helper script can accomplish this easily for you.

```
./setup.sh clean
```

Trimming adaptors

A typical workflow for processing Illumina sequencing data takes the demultiplexed data from the sequencer and perform a series of trimming and contaminant removal steps. The first step is usually to remove adaptor sequences. These adaptors sometimes occur in sequences because the insert size is less than read length and the sequencer reads through to the other side.

This can be accomplished by running this script.

```
bbduk_examples/adaptor_trimming/bbduk_adaptor_trimming.sh
```

Lets look at the contents of that script.

First it checks for the environment variable `$bbtoolsExamplesDir`

```
# Check that the bbtools example directory variable has been set for users
[[ -e "$bbtoolsExamplesDir" ]] || { echo >&2 "Please source the env.sh file \
(source env.sh) in the bbtools example directory before running this script"; exit 1;}
```

Then it loads bbtools from the module management system of CERES

```
# Load the bbtools module
module load bbtools
```

Finally it runs `bbduk.sh`, a program in the bbtools suite:

```
# Trim the adapters using the reference file adaptors.fa (provided by bbduk)
bbduk.sh in="$bbtoolsExamplesDir"/data/reads.fq.gz \
out="$bbtoolsExamplesDir"/bbduk_examples/adaptor_trimming/results/clean.fq.gz \
ref="$bbtoolsExamplesDir"/data/adapters.fa \
ktrim=r k=23 mink=11 hdist=1 tbo=t
```

`bbduk.sh` can read and write compressed or uncompressed formats and infers the format from the file extension you provide it.

- `in=` the fastq file to trim
- `out=` the fastq file to output
- `ref=` points to a file with the reference sequences to be removes.
- `ktrim=r` lets bbduk know to trim off the 3' end of each read.
- `k=23` sets the length of the kmer to find
- `mink=11` specifies that kmers as short as 11 bases on the ends can be trimmed too
- `hdist=1` sets the Hamming distance i.e. the number of mismatches allowed in the kmer
- `tbo=t` trims adaptors based on where they are paired

This will write out a fastq file with adaptors trimmed off.

Kmer filtering for contaminant sequence removal

The next step in quality control is often to remove reads with other contaminants. Typically the file generated from adaptor trimming will be run through bbduk in contaminant filtering mode to remove common lab contaminants like phiX174. For simplicity, this example will use the original input file though, not the output of adaptor trimming.

Run the kmer filtering script.

```
bbduk_examples/kmer_filtering/bbduk_kmer_filtering.sh
```

This calls bbduk.sh with these parameters.

```
# Filter out PhiX containing reads optionally placing them in their own file
bbduk.sh in="$bbtoolsExamplesDir"/data/reads.fq.gz \
out="$bbtoolsExamplesDir"/bbduk_examples/kmer_filtering/results/unmatched.fq.gz \
outm="$bbtoolsExamplesDir"/bbduk_examples/kmer_filtering/results/matched.fq.gz \
ref="$bbtoolsExamplesDir"/data/phix174_ill.ref.fa.gz \
k=31 hdist=1 stats="$bbtoolsExamplesDir"/bbduk_examples/kmer_filtering/results/stats.txt
```

The options are:

- `out=` the fastq file containing reads that matched the contaminant database (sometimes used to estimate error rates in the data).
- `outu=` the primary fastq file with all contaminant reads removed
- `ref=` the fasta file(s) with all the contaminants to be removed. In this case it is just PhiX174, but other contaminant references can be added. Long contaminant genomes (plants and animals) are better removed with bbmap.
- `k=31` sets the length of the kmer to find
- `hdist=1` sets the Hamming distance i.e. the number of mismatches allowed in the kmer
- `stats=` the file to write statistics about what contaminants were found in the data

Quality trimming

BBduk can perform hard quality trimming. Quality trimming was a pretty standard preprocessing step a few years ago but now the decsion to use it depends on the downstream workflow. Some merging, error correction mapping and assembly methods can use, correct or ignore low quality bases.

Run the adaptor trimming script.

```
bbduk_examples/quality_trimming/bbduk_quality_trimming.sh
```

This calls bbduk.sh with these parameters.

```
#Trim reads with a quality score of less than 10 from the right side of reads
bbduk.sh in="$bbtoolsExamplesDir"/data/reads.fq.gz \
out="$bbtoolsExamplesDir"/bbduk_examples/quality_trimming/results/clean.fq.gz \
qtrim=r trimq=10
```

The parameters are pretty simple for this process.

- `qtrim=r` trim the reads from the right (3') end(s)
- `trimq=10` trim bases with a quality score below 10

Histogram generation

BBduk will generate histogram data for a number of quality metrics. Most of these reports can be generated during another step like adaptor trimming. This example generates many of these reports in one run.

Run the histogram generation script.

```
bbduk_examples/histogram_generation/bbduk_histogram_generation.sh
```

This calls bbduk.sh with these parameters.

```
bbduk.sh in="$bbtoolsExamplesDir"/data/reads.fq.gz \
bhist="$bbtoolsExamplesDir"/bbduk_examples/histogram_generation/results/bhist.txt \
qhist="$bbtoolsExamplesDir"/bbduk_examples/histogram_generation/results/qhist.txt \
gchist="$bbtoolsExamplesDir"/bbduk_examples/histogram_generation/results/gchist.txt \
aqhist="$bbtoolsExamplesDir"/bbduk_examples/histogram_generation/results/aqhist.txt \
bqhist="$bbtoolsExamplesDir"/bbduk_examples/histogram_generation/results/bqhist.txt \
lhist="$bbtoolsExamplesDir"/bbduk_examples/histogram_generation/results/lhist.txt \
gchist="$bbtoolsExamplesDir"/bbduk_examples/histogram_generation/results/gchist.txt \
gcbins=auto
```

Each option generates the following histogram data files:

- `bhist=` base composition histogram by position
- `qhist=` quality histogram by position
- `qchist=` count of bases with each quality value
- `aqhist=` histogram of average read quality
- `bqhist=` quality histogram designed for box plots

- `lhist=` read length histogram
- `gchist=` read GC content histogram

Merging reads

For some applications like assembly, merging reads can reduce the memory required and reduce assembly errors. Running merging also produces an estimate of the read-pair insert size which can help the lab optimize their fragmentation and library creation methods.

Run the basic read merging script.

```
bbmerge_examples/basic_merging/bbduk_basic_merging.sh
```

This calls bbmerge.sh with these parameters.

```
bbmerge.sh in="$bbtoolsExamplesDir"/data/reads.fq.gz \
out="$bbtoolsExamplesDir"/bbmerge_examples/basic_merging/results/merged.fq.gz \
outu="$bbtoolsExamplesDir"/bbmerge_examples/basic_merging/results/unmerged.fq.gz \
ihist="$bbtoolsExamplesDir"/bbmerge_examples/basic_merging/results/ihist.txt \
usejni
```

The parameters are:

- `out=` A fastq file with the reads the were successfully merged
- `outu=` A fastq file with the read pairs that could not be merged
- `ihist=` An insert size histogram file
- `usejni` An option to use the "Java Native Interface (jni)" to call compiled routines in C++. This speeds up merging. The option is available in CERES.

Merging for error correction

Overlapping reads can be used to correct sequencing errors on the ends of the read pairs.

Run the histogram generation script.

```
bbmerge_examples/basic_merging/bbduk_error_correction.sh
```

This calls bbmerge.sh with these parameters.

```
bbmerge.sh in="$bbtoolsExamplesDir"/data/reads.fq.gz \
out="$bbtoolsExamplesDir"/bbmerge_examples/error_correction/results/corrected.fq.gz \
ecco mix usejni
```

The parameters are:

- `out=` The error-corrected merged reads
- `ecco` The flag to perform error correction but not merge the reads

- `mix` The flag to output the corrected and uncorrected reads into the same file
- `usejni` An option to use the "Java Native Interface (jni)" to call compiled routines in C++. This speeds up merging. The option is available in CERES.

Mapping Reads to a reference with bbmap

The reads in this tutorial come from a mock viral metagenome with isolate virus DNA mixed together. The reference database `Ref_database.fq.gz` used for mapping has those reference viral genomes placed into one file.

Run the basic read merging script. This will build an index and map reads to it.

```
bbmerge_examples/basic_merging/bbduk_basic_merging.sh
```

This calls bbmerge.sh with these parameters. First it builds an index:

```
# Create a mapping index for the phage phiX174 and save it to disk \
# (you only need to do this once)
bbmap.sh ref="$bbtoolsExamplesDir"/data/Ref_database.fa.gz \
path="$bbtoolsExamplesDir"/bbmap_examples/basic_mapping/results \
usejni
```

Next, it maps the reads to the reference.

```
bbmap.sh in="$bbtoolsExamplesDir"/data/reads.fq.gz \
path="$bbtoolsExamplesDir"/bbmap_examples/basic_mapping/results \
out="$bbtoolsExamplesDir"/bbmap_examples/basic_mapping/results/mapped.sam \
usejni pigz unpigz maxindel=90
```

The parameters are:

- `ref=` the reference sequence to map against
- `usejni` An option to use the "Java Native Interface (jni)" to call compiled routines in C++. This speeds up merging. The option is available in CERES.
- `pigz` and `unpigz` perform parallel compression and uncompression
- `-Xmx4g` limits the amount of memory for this very small job (note the dash)
- `threads=` limits the threads for the example. You should match this to the number of threads requested my your SLURM job. by default bbtools grabs all available threads.
- `maxindel=90` sets the maximum gap between the read pairs. The default is 16k (for eukaryotic RNA) but this is not needed for DNA