Derek Barnett – 22 March 2011

BamTools Toolkit Tutorial
Version 1.0

This document serves as user's guide for the BamTools command-line utility suite.

Table of Contents

Introduction

BamTools provides a small, but powerful suite of command-line utility programs for manipulating and querying BAM files for data.

Common Options

Input/Output

All BamTools utilities handle I/O operations using a common set of arguments. These include:

-in <BAM file>

The input BAM files(s).

If a tool accepts multiple BAM files as input, each file gets its own "-in" option on the command line. If no "-in" is provided, the tool will attempt to read BAM data from stdin.

To read a single BAM file, use a single "-in" option:

```
> bamtools *tool* -in myData1.bam ...ARGS...
```

To read multiple BAM files, use multiple "-in" options:

> bamtools *tool* -in myData1.bam -in myData2.bam ...ARGS...

To read from stdin (if supported), omit the "-in" option:
> bamtools *tool* ...ARGS...

-out <BAM file>

The output BAM file.

If a tool outputs a result BAM file, specify the filename using this option. If none is provided, the tool will typically write to stdout.

Note: Not all tools output BAM data (e.g. count, header, etc.)

-region <REGION>

A region of interest. See below for accepted 'REGION string' formats.

Many of the tools accept this option, which allows a user to only consider alignments that overlap this region (whether counting, filtering, merging, etc.).

An alignment is considered to overlap a region if any part of the alignments intersects the left/right boundaries. Thus, a 50bp alignment at position 70 will overlap a region beginning at position 100.

REGION string format: A proper REGION string can be formatted like any of the following examples (where 'chr1' is the name of a reference (not its ID)and '' is any valid integer position within that reference.) :

-region chr1
only alignments on (entire) reference 'chr1'
-region chr1:500
only alignments overlapping the region starting at chr1:500 and continuing to the end of chr1
-region chr1:500..1000
only alignments overlapping the region starting at chr1:500 and continuing to chr1:1000
-region chr1:500..chr3:750
only alignments overlapping the region starting at chr1:500 and continuing to chr3:750. This 'spanning' region assumes that the reference specified as the right boundary will occur somewhere in the file after the left boundary. On a sorted BAM, a REGION of 'chr4:500..chr2:1500' will produce undefined (incorrect) results. So don't do it. :)

Note: Most of the tools that accept a REGION string will perform without an index file, but typically at great cost to performance (having to plow through the entire file until the region of interest is found). For optimum speed, be sure that index files are available for your data.

-forceCompression

Force compression of BAM output.

When tools are piped together (see details below), the default behavior is to turn off compression. This can greatly increase performance when the data does not have to be constantly decompressed and recompressed. This option is ignored any time an output BAM file is specified using "-out".

Piping

Many of the tools in BamTools can be chained together by piping. Any tool that accepts stdin can be piped into, and any that can output stdout can be piped from. For example:

> bamtools filter -in data1.bam -in data2.bam -mapQuality ">50" | bamtools count

will give a count of all alignments in your 2 BAM files with a mapQuality of greater than 50. And of course, any tool writing to stdout can be piped into other utilities.

The Tools

Fix formatting and add more details for each tool.

convert

Description: converts BAM to a number of other formats

Currently supported output formats (BAM -> X)

Format type	FORMAT (command-line argument)
BED	bed
FASTA	fasta
FASTQ	fastq
JSON	json
Pileup	pileup
SAM	sam
YAML	yaml

Usage example:
> bamtools convert -format json -in myData.bam -out myData.json

Derek Barnett – 22 March 2011

Pileup Options have no effect on formats other than "pileup"
SAM Options have no effect on formats other than "sam"

count
Description: prints number of alignments in BAM file(s).

coverage
Description: prints coverage data for a single BAM file.

filter
Description: filters BAM file(s).

In addition to the command line options, the BamTools filter tool allows you to use an external filter script to define complex filtering behavior. This script uses what I'm calling properties, filters, and a rule - all implemented in a JSON syntax.

Properties
A 'property' is a typical JSON entry of the form:

```
"propertyName" : "value"
```

Here are the property names that BamTools will recognize:

- alignmentFlag
- cigar
- insertSize
- isDuplicate
- isFailedQC
- isFirstMate
- isMapped
- isMateMapped
- isMateReverseStrand
- isPaired
- isPrimaryAlignment
- isProperPair
- isReverseStrand
- isSecondMate
- mapQuality
- matePosition
- mateReference
- name
- position
- queryBases

reference
tag

For properties with boolean values, use the words "true" or "false".
For example,

"isMapped" : "true"

will keep only alignments that are flagged as 'mapped'.

For properties with numeric values, use the desired number with optional comparison operators (>, >=, <, <=, !). For example,

"mapQuality" : ">=75"

will keep only alignments with mapQuality greater than or equal to 75.

If you're familiar with JSON, you know that integers can be bare (without quotes). However, if you use a comparison operator, be sure to enclose in quotes.

For string-based properties, the above operators are available. In addition, you can also use some basic pattern-matching operators, using the '*' character as a wildcard. For example,

"reference" : "ALU*" // reference **starts-with** 'ALU'
"name" : "*foo" // name **ends-with** 'foo'
"cigar" : "*D*" // cigar **contains** a 'D' anywhere

Additional notes:
The reference property refers to the reference name, not the BAM reference numeric ID.

The tag property has an extra layer, so that the syntax will look like this:

"tag" : "XX:value"

where XX is the 2-letter SAM/BAM tag and value is, well, the value. Comparison operators can still apply to values, so tag properties of:

"tag" : "AS:>60"
"tag" : "RG:foo*"

are perfectly valid.

Derek Barnett – 22 March 2011

Filters

A 'filter' is a JSON container of properties that will be AND-ed together. For example,

```
{
  "reference" : "chr1",
  "mapQuality" : ">50",
  "tag" : "NM:<4"
}
```

would result in an output BAM file containing only alignments from chr1 with a mapQuality >50 and edit distance of less than 4.

A single, unnamed filter like this is the minimum necessary for a complete filter script. Save this file and use as the -script parameter and you should be all set.

Moving on to more potent filtering...

You can also define multiple filters. To do so, you just need to use the "filters" keyword along with JSON array syntax, like this:

```
{
  "filters" :
    [
      {
        "reference" : "chr1",
        "mapQuality" : ">50"
      },
      {
        "reference" : "chr1",
        "isReverseStrand" : "true"
      }
    ]
}
```

These filters will be (inclusive) OR-ed together by default. So you'd get a resulting BAM with only alignments from chr1 that had either mapQuality >50 or on the reverse strand (or both).

Rule

Alternatively to anonymous OR-ed filters, you can also provide what I've called a "rule". By giving each filter an "id", using this "rule" keyword you can describe boolean relationships between your filter sets.

Available rule operators:

```
&   // and
|   // or
!   // not
```

This might sound a little fuzzy at this point, so let's get back to an example:

```
{
   "filters" : [
      { "id" : "inAnyErrorReadGroup",
        "tag" : "RG:ERR*"
      },
      { "id" : "highMapQuality",
        "mapQuality" : ">=75"
      },
      { "id" : "bothMatesMapped",
        "isMapped" : "true",
        "isMateMapped" : "true"
      }
    ],
  "rule" : " !inAnyErrorReadGroup & (highMapQuality | bothMatesMapped) "
}
```

In this case, we would only retain aligments that had high map quality OR both mates mapped (inclusive), while excluding all alignments from read groups that start with "ERR".

header
Description: prints header from BAM file(s).

index
Description: creates index for BAM file.

merge
Description: merges multiple BAM files into one.

random
Description: grab a random subset of alignments.

revert
Description: removes duplicate marks and restores original (non-recalibrated) base qualities.

sort
Description: sorts a BAM file.

split
Description: splits a BAM file on user-specified property, creating a new BAM output file for each value found.

stats
Description: prints general alignment statistics.

Conclusion

This document is a work-in-progress, more details will be added for most of the tools. Feel free to contact me if there is anything specific you would like to see added to this tutorial.

Thanks.

Derek Barnett
Marth Lab, Boston College
derekwbarnett@gmail.com