An Introduction to *Rsamtools*

Martin Morgan

Modified: 18 March, 2010. Compiled: October 15, 2013

Contents


```
> library(Rsamtools)
```

1 Introduction

The *Rsamtools* package provides an interface to `BAM` files. `BAM` files are produced by *samtools* and other software, and represent a flexible format for storing 'short' reads aligned to reference genomes. `BAM` files typically contain sequence and base qualities, and alignment coordinates and quality measures. `BAM` files are appealing for several reasons. The format is flexible enough to represent reads generated and aligned using diverse technologies. The files are binary so that file access is relatively efficient. `BAM` files can be indexed, allowing ready access to localized chromosomal regions. `BAM` files can be accessed remotely, provided the remote hosting site supports such access and a local index is available. This means that specific regions of remote files can be accessed without retrieving the entire (large!) file. A full description is available in the `BAM` format specification (`http://samtools.sourceforge.net/SAM1.pdf`)

The main purpose of the *Rsamtools* package is to import `BAM` files into *R*. *Rsamtools* also provides some facility for file access such as record counting, index file creation, and filtering to create new files containing subsets of the original. An important use case for *Rsamtools* is as a starting point for creating *R* objects suitable for a diversity of work flows, e.g., *AlignedRead* objects in the *ShortRead* package (for quality assessment and read manipulation), or *GAlignments* objects in *GenomicRanges* package (for RNA-seq and other applications). Those desiring more functionality are encouraged to explore *samtools* and related software efforts.

2 Input

The essential capability provided by *Rsamtools* is `BAM` input. This is accomplished with the `scanBam` function. `scanBam` takes as input the name of the `BAM` file to be parsed. In addition, the `param` argument determines which genomic coordinates of the `BAM` file, and what components of each record, will be input. *R*param is an instance of the *ScanBamParam*

class. To create a param object, call ScanBamParam. Here we create a param object to extract reads aligned to three distinct ranges (one on seq1, two on seq2). From each of read in those ranges, we specify that we would like to extract the reference name (rname, e.g., seq1), strand, alignment position, query (i.e., read) width, and query sequence:

```
> which <- RangesList(seq1=IRanges(1000, 2000),
+                     seq2=IRanges(c(100, 1000), c(1000, 2000)))
> what <- c("rname", "strand", "pos", "qwidth", "seq")
> param <- ScanBamParam(which=which, what=what)
```

Additional information can be found on the help page for ScanBamParam. Reading the relevant records from the BAM file is accomplished with

```
> bamFile <-
+     system.file("extdata", "ex1.bam", package="Rsamtools")
> bam <- scanBam(bamFile, param=param)
```

Like scan, scanBam returns a list of values. Each element of the list corresponds to a range specified by the which argument to ScanBamParam.

```
> class(bam)
```

```
[1] "list"
```

```
> names(bam)
```

```
[1] "seq1:1000-2000" "seq2:100-1000"  "seq2:1000-2000"
```

Each element is itself a list, containing the elements specified by the what and tag arguments to ScanBamParam.

```
> class(bam[[1]])
```

```
[1] "list"
```

```
> names(bam[[1]])
```

```
[1] "rname"  "strand" "pos"    "qwidth" "seq"
```

The elements are either basic *R* or *IRanges* data types

```
> sapply(bam[[1]], class)
```

```
         rname         strand            pos         qwidth
      "factor"       "factor"      "integer"      "integer"
           seq
"DNAStringSet"
```

A paradigm for collapsing the list-of-lists into a single list is

```
> .unlist <- function (x)
+ {
+     ## do.call(c, ...) coerces factor to integer, which is undesired
+     x1 <- x[[1L]]
+     if (is.factor(x1)) {
+         structure(unlist(x), class = "factor", levels = levels(x1))
+     } else {
+         do.call(c, x)
+     }
+ }
> bam <- unname(bam) # names not useful in unlisted result
> elts <- setNames(bamWhat(param), bamWhat(param))
> lst <- lapply(elts, function(elt) .unlist(lapply(bam, "[[", elt)))
```

This might be further transformed, e.g., to a *DataFrame*, with

```
> head(do.call("DataFrame", lst))
```

```
DataFrame with 6 rows and 5 columns
     rname   strand       pos    qwidth
  <factor> <factor> <integer> <integer>
1     seq1        +       970        35
2     seq1        +       971        35
3     seq1        +       972        35
4     seq1        +       973        35
5     seq1        +       974        35
6     seq1        -       975        35
                              seq
                     <DNAStringSet>
1 TATTAGGAAATGCTTTACTGTCATAACTATGAAGA
2 ATTAGGAAATGCTTTACTGTCATAACTATGAAGAG
3 TTAGGAAATGCTTTACTGTCATAACTATGAAGAGA
4 TAGGAAATGCTTTACTGTCATAACTATGAAGAGAC
5 AGGAAATGCTTTACTGTCATAACTATGAAGAGACT
6 GGAAATGCTTTACTGTCATAACTATGAAGAGACTA
```

Often, an alternative is to use a *ScanBamParam* object with desired fields specified in `what` as an argument to `GenomicRanges::readGAlignments`; the specified fields are added as columns to the returned *GAlignments*.

The `BAM` file in the previous example includes an index, represented by a separate file with extension `.bai`:

```
> list.files(dirname(bamFile), pattern="ex1.bam(.bai)?")
```

```
[1] "ex1.bam"     "ex1.bam.bai"
```

Indexing provides two significant benefits. First, an index allows a `BAM` file to be efficiently accessed by range. A corollary is that providing a `which` argument to `ScanBamParam` requires an index. Second, coordinates for extracting information from a `BAM` file can be derived from the index, so a portion of a remote `BAM` file can be retrieved with local access only to the index. For instance, provided an index file exists on the local computer, it is possible to retrieve a small portion of a `BAM` file residing on the 1000 genomes HTTP server. The url `ftp://ftp-trace.ncbi.nih.gov/1000genomes/ftp/pilot_data/data/NA19240/alignment/NA19240.chrom6.SLX.maq.SRP000032.2009_07.bam` points to the `BAM` file corresponding to individual NA19240 chromosome 6 Solexa (Illumina) sequences aligned using MAQ. The remote file is very large (about 10 GB), but the corresponding index file is small (about 500 KB). With `na19240url` set to the above address, the following retrieves just those reads in the specified range

```
> which <- RangesList("6"=IRanges(100000L, 110000L))
> param <- ScanBamParam(which=which, what=scanBamWhat())
> na19240bam <- scanBam(na19240url, param=param)
```

Invoking `scanBam` without an index file, as above, first retrieves the index file from the remote location, and then queries the remote file using the index; for repeated queries, it is more efficient to retrieve the index file first (e.g., with `download.file`) and then use the local index as an argument to `scanBam`. Many `BAM` files were created in a way that causes `scanBam` to report that the "EOF marker is absent"; this message can safely be ignored.

`BAM` files may be read by functions in packages other than *Rsamtools*. In *ShortRead*, `readAligned` invoked with `type="bam"` will read `BAM` files in to an *AlignedRead* object. This function takes a `param` argument, just as `scanBam`, so the user can control which portions of the file are input. Similar input facilities are available through the `readGAlignments` function in *GenomicRanges*.

Additional ways of interacting with `BAM` files include `scanBamHeader` (to extract header information) and `countBam` (to count records matching `param`). `filterBam` filters reads from the source file according to the criteria of the *ScanBamParam* parameter, writing reads passing the filter to a new file. The function `sortBam` sorts a previously unsorted `BAM`, while The function `indexBam` creates an index file from a sorted `BAM` file.

`readPileup` reads a `pileup` file created by *samtools*, importing SNP, indel, or all variants into a *GRanges* object.

2.1 Large bam files

BAM files can be large, containing more information on more genomic regions than are of immediate interest or than can fit in memory. The first strategy for dealing with this is to select, using the `what` and `which` arguments to `ScanBamParam`, just those portions of the BAM file that are essential to the current analysis, e.g., specifying `what=c('rname', 'qname', 'pos')` when wishing to calculate coverage of ungapped reads.

When selective input of BAM files is still too memory-intensive, the file can be processed in chunks, with each chunk distilled to the derived information of interest. Chromosomes will often be the natural chunk to process. For instance, here we write a summary function that takes a single sequence name (chromosome) as input, reads in specific information from the BAM file, and calculates coverage over that sequence.

```
> summaryFunction <-
+     function(seqname, bamFile, ...)
+ {
+     param <- ScanBamParam(what=c('pos', 'qwidth'),
+                           which=GRanges(seqname, IRanges(1, 1e7)),
+                           flag=scanBamFlag(isUnmappedQuery=FALSE))
+     x <- scanBam(bamFile, ..., param=param)[[1]]
+     coverage(IRanges(x[["pos"]], width=x[["qwidth"]]))
+ }
```

This might be used as follows; it is an ideal candidate for evaluation in parallel, e.g., using the *parallel* package and `srapply` function in *ShortRead*.

```
> seqnames <- paste("seq", 1:2, sep="")
> cvg <- lapply(seqnames, summaryFunction, bamFile)
> names(cvg) <- seqnames
> cvg
```

```
$seq1
integer-Rle of length 1569 with 1054 runs
  Lengths:  2  2  1  3  4  2  3  4 ...  1  1  1  1  1  1  1
  Values :  1  2  3  4  5  7  8  9 ...  9  7  6  5  3  2  1

$seq2
integer-Rle of length 1567 with 1092 runs
  Lengths:  1  3  1  1  1  3  1  4 ...  1  1  2  1  4  4  1
  Values :  3  4  5  8 12 14 15 16 ... 10  8  7  6  3  2  1
```

The result of the function (a coverage vector, in this case) will often be much smaller than the input.

3 Views

The functions described in the previous section import data in to *R*. However, sequence data can be very large, and it does not always make sense to read the data in immediately. Instead, it can be useful to marshal *references* to the data into a container and then act on components of the container. The *BamViews* class provides a mechanism for creating 'views' into a set of BAM files. The view itself is light-weight, containing references to the relevant BAM files and metadata about the view (e.g., the phenotypic samples corresponding to each BAM file).

One way of understanding a *BamViews* instance is as a rectangular data structure. The columns represent BAM files (e.g., distinct samples). The rows represent ranges (i.e., genomic coordinates). For instance, a ChIP-seq experiment might identify a number of peaks of high read counts.

3.1 Assembling a BamViews instance

To illustrate, suppose we have an interest in caffeine metabolism in humans. The 'rows' contain coordinates of genomic regions associated with genes in a KEGG caffeine metabolism pathway. The 'columns' represent individuals in the 1000 genomes project.

To create the 'rows', we identify possible genes that KEGG associates with caffeine metabolism:

```
> library(KEGG.db)
> kid <- revmap(KEGGPATHID2NAME)[["Caffeine metabolism"]]
> egid <- KEGGPATHID2EXTID[[sprintf("hsa%s", kid)]]
```

Then we use the appropriate *TranscriptDb* package to translate Entrez identifiers to obtain ranges of interest (one could also use *biomaRt* to retrieve coordinates for non-model organisms, perhaps making a `TranscriptDb` object as outlined in the *GenomicFeatures* vignette). We'll find that the names used for chromosomes in the alignments differ from those used at Ensembl, so `seqlevels<-` is used to map between naming schemes and to drop unused levels.

```
> library(TxDb.Hsapiens.UCSC.hg18.knownGene)
> bamRanges <- transcripts(TxDb.Hsapiens.UCSC.hg18.knownGene,
+                          vals=list(gene_id=egid))
> seqlevels(bamRanges) <-                 # translate seqlevels
+     sub("chr", "", seqlevels(bamRanges))
> lvls <- seqlevels(bamRanges)            # drop unused levels
> seqlevels(bamRanges) <- lvls[lvls %in% as.character(seqnames(bamRanges))]
```

Note that `bamRanges` 'knows' the genome for which the ranges are defined

```
> unique(genome(bamRanges))
```

```
[1] "hg18"
```

The details of creating the 'columns' of BAM files. Here we retrieve a vector of `BAM` file URLs (`slxMaq09`) from the package.

```
> slxMaq09 <- local({
+     fl <- system.file("extdata", "slxMaq09_urls.txt",
+                       package="Rsamtools")
+     readLines(fl)
+ })
```

We now assemble the *BamViews* instance from these objects; we also provide information to annotated the `BAM` files (with the `bamSamples` function argument, which is a *DataFrame* instance with each row corresponding to a `BAM` file) and the instance as a whole (with `bamExperiment`, a simple named *list* containing information structured as the user sees fit).

```
> bamExperiment <-
+     list(description="Caffeine metabolism views on 1000 genomes samples",
+          created=date())
> bv <- BamViews(slxMaq09, bamRanges=bamRanges,
+                bamExperiment=bamExperiment)
> metadata(bamSamples(bv)) <-
+     list(description="Solexa/MAQ samples, August 2009",
+          created="Thu Mar 25 14:08:42 2010")
```

3.2 Using BamViews instances

The *BamViews* object can be queried for its component parts, e.g.,

```
> bamExperiment(bv)
```

```
$description
[1] "Caffeine metabolism views on 1000 genomes samples"

$created
[1] "Tue Oct 15 21:37:01 2013"
```

More usefully, Methods in *Rsamtools* are designed to work with *BamViews* objects, retrieving data from all files in the view. These operations can take substantial time and require reliable network access.

To illustrate, the following code (not evaluated when this vignette was created) downloads the index files associated with the bv object

```
> bamIndexDir <- tempfile()
> indexFiles <- paste(bamPaths(bv), "bai", sep=".")
> dir.create(bamIndexDir)
> idxFiles <- mapply(download.file, indexFiles,
+                    file.path(bamIndexDir, basename(indexFiles)) ,
+                    MoreArgs=list(method="curl"))
```

and then queries the 1000 genomes project for reads overlapping our transcripts; by loading the *parallel* package, we tell *Rsamtools* to use as many cores as are available on our machine (the *parallel* package does not provide parallel functionality on Windows computers)

```
> library(parallel)
> options(srapply_fapply="parallel", mc.cores=detectCores())
> olaps <- readGAlignmentsFromBam(bv)
```

The resulting object is about 11 MB in size. To avoid having to download this data each time the vignette is run, we instead load it from disk

```
> load(system.file("extdata", "olaps.Rda", package="Rsamtools"))
> olaps

List of length 24
names(24): NA06986.SLX.maq.SRP000031.2009_08.bam ...

> head(olaps[[1]])

GAlignments with 6 alignments and 0 metadata columns:
      seqnames strand       cigar    qwidth     start
         <Rle>  <Rle> <character> <integer> <integer>
  [1]        2      +         51M        51  31410650
  [2]        2      +         51M        51  31410658
  [3]        2      -         51M        51  31410663
  [4]        2      +         51M        51  31410666
  [5]        2      -         51M        51  31410676
  [6]        2      +         51M        51  31410676
             end     width      ngap
       <integer> <integer> <integer>
  [1]  31410700        51         0
  [2]  31410708        51         0
  [3]  31410713        51         0
  [4]  31410716        51         0
  [5]  31410726        51         0
  [6]  31410726        51         0
  ---
  seqlengths:
           1         2         3 ... NT_113898 NC_007605
   247249719 242951149 199501827 ...   1305230    171823
```

There are 34616 reads in NA06986.SLX.maq.SRP000031.2009_08.bam overlapping at least one of our transcripts. It is easy to explore this object, for instance discovering the range of read widths in each individual.

```
> head(t(sapply(olaps, function(elt) range(qwidth(elt)))))
```

```
                                       [,1] [,2]
NA06986.SLX.maq.SRP000031.2009_08.bam   51   51
NA06994.SLX.maq.SRP000031.2009_08.bam   36   51
NA07051.SLX.maq.SRP000031.2009_08.bam   51   51
NA07346.SLX.maq.SRP000031.2009_08.bam   48   76
NA07347.SLX.maq.SRP000031.2009_08.bam   51   51
NA10847.SLX.maq.SRP000031.2009_08.bam   36   51
```

Suppose we were particularly interested in the first transcript, which has a transcript id uc002ayr.1. Here we extract reads overlapping this transcript from each of our samples. As a consequence of the protocol used, reads aligning to either strand could be derived from this transcript. For this reason, we set the strand of our range of interest to *. We use the endoapply function, which is like lapply but returns an object of the same class (in this case, SimpleList) as its first argument.

```
> rng <- bamRanges(bv)[1]
> strand(rng) <- "*"
> olap1 <- endoapply(olaps, subsetByOverlaps, rng)
> olap1 <- lapply(olap1, "seqlevels<-", value=as.character(seqnames(rng)))
> head(olap1[[24]])

GAlignments with 6 alignments and 0 metadata columns:
      seqnames strand       cigar    qwidth     start
         <Rle>  <Rle> <character> <integer> <integer>
  [1]       15      -         36M        36  72828212
  [2]       15      +         36M        36  72828226
  [3]       15      -         36M        36  72828237
  [4]       15      -         36M        36  72828272
  [5]       15      -         36M        36  72828291
  [6]       15      -         36M        36  72828297
             end     width      ngap
       <integer> <integer> <integer>
  [1]  72828247        36         0
  [2]  72828261        36         0
  [3]  72828272        36         0
  [4]  72828307        36         0
  [5]  72828326        36         0
  [6]  72828332        36         0
  ---
  seqlengths:
          15
   100338915
```

To carry the example a little further, we calculate coverage of each sample:

```
> minw <- min(sapply(olap1, function(elt) min(start(elt))))
> maxw <- max(sapply(olap1, function(elt) max(end(elt))))
> cvg <- endoapply(olap1, coverage,
+                  shift=-start(ranges(bamRanges[1])),
+                  width=width(ranges(bamRanges[1])))
> cvg[[1]]

RleList of length 1
$`15`
integer-Rle of length 7758 with 1115 runs
  Lengths:  8  1 10 31  9 11  3  5 ...  1  2 13  1  5  2  4
  Values :  2  3  2  3  4  3  2  1 ...  3  2  3  4  5  6  5
```

Since the example includes a single region of uniform width across all samples, we can easily create a coverage matrix with rows representing nucleotide and columns sample and, e.g., document variability between samples and nucleotides

```
> m <- matrix(unlist(lapply(cvg, lapply, as.vector)),
+             ncol=length(cvg))
> summary(rowSums(m))
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  38.00   69.00   78.00   77.86   86.00  122.00
```

```
> summary(colSums(m))
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  12070   15930   22560   25170   33080   51200
```

4 Directions

This vignette has summarized facilities in the *Rsamtools* package. Important additional packages include *GenomicRanges* (for representing and manipulating gapped alignments), *ShortRead* for I/O and quality assessment of ungapped short read alignments, *Biostrings* and *BSgenome* for DNA sequence and whole-genome manipulation, *IRanges* for range-based manipulation, and *rtracklayer* for I/O related to the UCSC genome browser. Users might also find the interface to the integrative genome browser (IGV) in *SRAdb* useful for visualizing `BAM` files.

```
> packageDescription("Rsamtools")
```

```
Package: Rsamtools
Type: Package
Title: Binary alignment (BAM), variant call (BCF), or
      tabix file import
Version: 1.14.1
Author: Martin Morgan, Herv\'e Pag\`es, Valerie
      Obenchain
Maintainer: Bioconductor Package Maintainer
      <maintainer@bioconductor.org>
Description: This package provides an interface to
      the 'samtools', 'bcftools', and 'tabix'
      utilities (see 'LICENCE') for manipulating SAM
      (Sequence Alignment / Map), binary variant call
      (BCF) and compressed indexed tab-delimited
      (tabix) files.
URL:
      http://bioconductor.org/packages/release/bioc/html/Rsamtools.html
License: Artistic-2.0 | file LICENSE
LazyLoad: yes
Depends: methods, IRanges (>= 1.19.11), GenomicRanges
      (>= 1.13.35), XVector, Biostrings (>= 2.29.7)
Imports: utils, BiocGenerics (>= 0.1.3), zlibbioc,
      bitops
Suggests: ShortRead (>= 1.19.10), GenomicFeatures,
      TxDb.Dmelanogaster.UCSC.dm3.ensGene, KEGG.db,
      TxDb.Hsapiens.UCSC.hg18.knownGene,
      RNAseqData.HNRNPC.bam.chr14,
      BSgenome.Hsapiens.UCSC.hg19, pasillaBamSubset,
      RUnit, BiocStyle
LinkingTo: IRanges, XVector, Biostrings
```

```
biocViews: DataImport, Sequencing,
      HighThroughputSequencing
Built: R 3.0.2; x86_64-unknown-linux-gnu; 2013-10-16
      04:36:25 UTC; unix

-- File: /tmp/Rtmpuo1pxL/Rinst116a7877f8a6/Rsamtools/Meta/package.rds

> sessionInfo()

R version 3.0.2 (2013-09-25)
Platform: x86_64-unknown-linux-gnu (64-bit)

locale:
 [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
 [3] LC_TIME=en_US.UTF-8        LC_COLLATE=C
 [5] LC_MONETARY=en_US.UTF-8    LC_MESSAGES=en_US.UTF-8
 [7] LC_PAPER=en_US.UTF-8       LC_NAME=C
 [9] LC_ADDRESS=C               LC_TELEPHONE=C
[11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C

attached base packages:
[1] parallel  stats     graphics  grDevices utils
[6] datasets  methods   base

other attached packages:
 [1] TxDb.Hsapiens.UCSC.hg18.knownGene_2.10.1
 [2] GenomicFeatures_1.14.0
 [3] KEGG.db_2.10.1
 [4] RSQLite_0.11.4
 [5] DBI_0.2-7
 [6] AnnotationDbi_1.24.0
 [7] Biobase_2.22.0
 [8] Rsamtools_1.14.1
 [9] Biostrings_2.30.0
[10] GenomicRanges_1.14.1
[11] XVector_0.2.0
[12] IRanges_1.20.0
[13] BiocGenerics_0.8.0

loaded via a namespace (and not attached):
 [1] BSgenome_1.30.0    BiocStyle_1.0.0
 [3] RCurl_1.95-4.1     XML_3.98-1.1
 [5] biomaRt_2.18.0     bitops_1.0-6
 [7] rtracklayer_1.22.0 stats4_3.0.2
 [9] tools_3.0.2        zlibbioc_1.8.0
```

A Assembling a BamViews instance

A.1 Genomic ranges of interest

A.2 BAM files

Note: The following operations were performed at the time the vignette was written; location of on-line resources, in particular the organization of the 1000 genomes BAM files, may have changed.

We are interested in collecting the URLs of a number of BAM files from the 1000 genomes project. Our first goal is to identify files that might make for an interesting comparison. First, let's visit the 1000 genomes FTP site and discover available files. We'll use the *RCurl* package to retrieve the names of all files in an appropriate directory

```
> library(RCurl)
> ftpBase <-
+     "ftp://ftp-trace.ncbi.nih.gov/1000genomes/ftp/pilot_data/data/"
> indivDirs <-
+     strsplit(getURL(ftpBase, ftplistonly=TRUE), "\n")[[1]]
> alnDirs <-
+     paste(ftpBase, indivDirs, "/alignment/", sep="")
> urls0 <-
+     strsplit(getURL(alnDirs, dirlistonly=TRUE), "\n")
```

From these, we exclude directories without any files in them, select only the BAM index (extension .bai) files, and choose those files that exactly six '.' characters in their name.

```
> urls <- urls[sapply(urls0, length) != 0]
> fls0 <- unlist(unname(urls0))
> fls1 <- fls0[grepl("bai$", fls0)]
> fls <- fls1[sapply(strsplit(fls1, "\\."), length)==7]
```

After a little exploration, we focus on those files obtained form Solexa sequencing, aligned using MAQ, and archived in August, 2009; we remove the .bai extension, so that the URL refers to the underlying file rather than index. There are 24 files.

```
> urls1 <-
+     Filter(function(x) length(x) != 0,
+            lapply(urls,
+                   function(x) x[grepl("SLX.maq.*2009_08.*bai$", x)]))
> slxMaq09.bai <-
+    mapply(paste, names(urls1), urls1, sep="", USE.NAMES=FALSE)
> slxMaq09 <- sub(".bai$", "", slxMaq09.bai) #$
```

As a final step to prepare for using a *BamViews* file, we create local copies of the *index* files. The index files are relatively small (about 190 Mb total).

```
> bamIndexDir <- tempfile()
> dir.create(bamIndexDir)
> idxFiles <- mapply(download.file, slxMaq09.bai,
+                    file.path(bamIndexDir, basename(slxMaq09.bai)) ,
+                    MoreArgs=list(method="curl"))
```