Package 'bamUtils'

January 23, 2018

Title Utility functions for manipulating bams

Version 0.0.0.9000

Description Utility functions for manipulating BAMs

biocViews

Depends R (>= 3.1),
data.table (>= 1.9),
GenomicRanges,
GenomicAlignments,
Rsamtools (>= 1.18),
gUtils

Suggests testthat

License GPL-2

BugReports http://github.com/mskilab/bamUtils/issues

LazyData true

RoxygenNote 6.0.1.9000

R topics documented:

bam.cov.gr . . . 2
bam.cov.tile . . . 3
bamflag . . . 4
bamtag . . . 4
chunk . . . 5
count.clips . . . 5
countCigar . . . 6
get.mate.gr . . . 6
get.pairs.grl . . . 7
hets . . . 7
is.paired.end . . . 8
mafcount . . . 8
oneoffs . . . 9
read.bam . . . 10
splice.cigar . . . 11
varbase . . . 12
varcount . . . 12

Index **14**

`bam.cov.gr` *Get coverage as GRanges from BAM on custom set of GRanges*

Description

Gets coverage from BAM in supplied GRanges using 'countBam()', returning GRanges with coverage counts in each of the provided GRanges (different from 'bamUtils::bam.cov()') specified as the columns $file, $records, and $nucleotides in the values field

Basically a wrapper for 'Rsamtools::countBam()' with some standard settings for 'Rsamtools::ScanBamParams()'

Usage

```
bam.cov.gr(bam, bai = NULL, intervals = NULL, all = FALSE,
  count.all = FALSE, isPaired = TRUE, isProperPair = TRUE,
  isUnmappedQuery = FALSE, hasUnmappedMate = FALSE,
  isNotPassingQualityControls = FALSE, isDuplicate = FALSE, mc.cores = 1,
  chunksize = 10, verbose = FALSE, ...)
```

Arguments

`bam`	string Input BAM file. Advisable to make the input BAM a BamFile instance instead of a plain string, so that the index does not have to be reloaded.
`bai`	string Input BAM index file
`intervals`	GRanges of intervals to retrieve
`all`	boolean Flag to read in all of BAM as a GRanges via 'si2gr(seqinfo())' (default = FALSE)
`isPaired`	boolean Flag indicates whether unpaired (FALSE), paired (TRUE), or any (NA) read should be returned. See documentation for Rsamtools::scanBamFlag(). (default == NA)
`isProperPair`	boolean Flag indicates whether improperly paired (FALSE), properly paired (TRUE), or any (NA) read should be returned. A properly paired read is defined by the alignment algorithm and might, e.g., represent reads aligning to identical reference sequences and with a specified distance. See documentation for Rsamtools::scanBamFlag(). (default == NA)
`isUnmappedQuery`	boolean Flag indicates whether unmapped (TRUE), mapped (FALSE), or any (NA) read should be returned. See documentation for Rsamtools::scanBamFlag(). (default == NA)
`hasUnmappedMate`	boolean Flag indicates whether reads with mapped (FALSE), unmapped (TRUE), or any (NA) mate should be returned. See documentation for Rsamtools::scanBamFlag(). (default == NA)
`isNotPassingQualityControls`	boolean Flag indicates whether reads passing quality controls (FALSE), reads not passing quality controls (TRUE), or any (NA) read should be returned. See documentation for Rsamtools::scanBamFlag(). (default == NA)

isDuplicate	boolean Flag indicates that un-duplicated (FALSE), duplicated (TRUE), or any (NA) reads should be returned. 'Duplicated' reads may represent PCR or optical duplicates. See documentation for Rsamtools::scanBamFlag(). (default == FALSE)
mc.cores	integer Number of cores in `mclapply` call
chunksize	integer How many intervals to process per core (default == 10)
verbose	boolean "verbose" flag (default == FALSE)
...	futher arguments passed into Rsamtools::scanBamFlag()

Value

GRanges parallel to input GRanges, but with metadata filled in.

`bam.cov.tile` *Get coverage as GRanges from BAM on genome tiles across seqlengths of genome*

Description

Quick way to get tiled coverage via piping to samtools (e.g. ~10 CPU-hours for 100bp tiles, 5e8 read pairs)

Gets coverage for window size "window" [bp], pulling "chunksize" records at a time and incrementing bin corresponding to midpoint or overlaps of corresponding (proper pair) fragment (uses TLEN and POS for positive strand reads that are part of a proper pair)

Usage

```
bam.cov.tile(bam.file, window = 100, chunksize = 1e+05, min.mapq = 30,
  verbose = TRUE, max.tlen = 10000, st.flag = "-f 0x02 -F 0x10",
  fragments = TRUE, region = NULL, do.gc = FALSE, midpoint = TRUE)
```

Arguments

bam.file	string Input BAM file
window	integer Window size (in bp) (default == 1e2)
chunksize	integer Size of window (default == 1e5)
min.mapq	integer Minimim map quality reads to consider for counts (default == 30)
verbose	boolean "verbose" flag (default == TRUE)
max.tlen	max paired-read insert size to consider
st.flag	boolean Samtools flag to filter reads on (default == '-f 0x02 -F 0x10')
fragments	boolean Flag (default == FALSE)
region	um? (default == NULL)
do.gc	boolean Flag to execute garbage collection via 'gc()' (default == FALSE)
midpoint	boolean Flag if TRUE will only use the fragment midpoint, if FALSE will count all bins that overlap the fragment

Value

GRanges of "window" bp tiles across seqlengths of bam.file with meta data field $counts specifying fragment counts centered in the given bin.

bamflag *Returns matrix of bits from BAM flags*

Description

Shortcut function: assumes reads are GappedAlignments with flag variable or actual integers representing BAM flag

Usage

```
bamflag(reads)
```

Arguments

reads	GenomicRanges or 'GappedAlignments' or data.table holding the reads

Value

matrix of bits from BAM flags

bamtag *Outputs a tag to identify duplicate reads in GRanges input*

Description

Outputs a tag that cats 'qname', first vs first second mate +/- secondary alignment +/- gr.string to give an identifier for determine duplicates in a read pile

Usage

```
bamtag(reads, secondary = FALSE, gr.string = FALSE)
```

Arguments

reads	GenomicRanges or GappedAlignments or data.frame holding the reads
secondary	boolean including secondary alignment(s) (default == FALSE)
gr.string	boolean input reads into gr.string() (default == FALSE)

chunk *chunk*

Description

Internal function takes same input as seq (from, to, by, length.out) and outputs a 2 column matrix of indices corresponding to "chunks"

Usage

```
chunk(from, to = NULL, by = 1, length.out = NULL)
```

Arguments

from	integer where to begin sequence
to	integer to end sequence
by	interval to space sequence
length.out	number of desired chunks, i.e. nrows of output matrix

Value

2 column matrix of indices, each row representing chunk

Author(s)

Marcin Imielinski

count.clips *Return data.frame with fields of "right" soft clips and "left" soft clips*

Description

Takes GRanges or GappedAlignments object and uses cigar field (or takes character vector of cigar strings) and returns data.frame with fields (for character input) $right.clips number of "right" soft clips (e.g. cigar 89M12S) $left.clips number of "left" soft clips (e.g. cigar 12S89M), or appends these fields to the reads object

Usage

```
count.clips(reads)
```

Arguments

reads	GenomicRanges or GappedAlignments or data.frame or data.table holding the reads

Value

GRanges with 'right.clips' and 'left.clips' columns added

countCigar *Count bases in cigar string*

Description

Counts the total number of bases, per cigar, that fall into D, I, M, S categories. countCigar makes no distinction between, for instance 1S2M2S, 2S2M1S, or 3S2M

Usage

```
countCigar(cigar)
```

Arguments

cigar	character vector of cigar strings

Value

matrix of dimensions (4-column, length(cigar)) with the total counts for each type

get.mate.gr *returns GRanges corresponding to mates of reads*

Description

Inputs GRanges or data.frame/data.table of reads. Outputs GRanges corresponding to mates of reads.

Usage

```
get.mate.gr(reads)
```

Arguments

reads	GRanges or data.table/data.frame Input reads

Value

GRanges corresponding to mates of reads

get.pairs.grl — *Get coverage as GRanges from BAM on custom set of GRanges*

Description

Takes reads object and returns GRangesList with each read and its mate (if exists)

Usage

```
get.pairs.grl(reads, pairs.grl.split = TRUE, verbose = FALSE)
```

Arguments

reads	GRanges holding reads
pairs.grl.split	boolean returns as GRangesList if TRUE (default == TRUE)
verbose	boolean verbose flag (default == FALSE)

hets — *Simple het "caller" meant to be used at validated het SNP sites for tumor / normal pairs*

Description

hets dumps a tsv file of alt ($alt.count.t, $alt.count.n) and ref ($ref.count.t, $ref.count.n) read counts to out.file for a tumor / normal pair across a set of sites specified by an input VCF

Usage

```
hets(tum.bam, norm.bam = NULL, out.file, vcf.file, chunk.size1 = 1000,
  chunk.size2 = 100, mc.cores = 14, verbose = T, na.rm = TRUE,
  filt.norm = T)
```

Arguments

tum.bam	character scalar or BamFile of tumor bam
norm.bam	character scalar or BamFile of normal bam (optional)
out.file	path to TSV output file to be generated
vcf.file	VCF file of sites (eg hapmap or 1000G) at which to compute read counts
chunk.size1	number of variants to process from VCF file at a time
chunk.size2	number of variants to access from BAM file in a single iteration
mc.cores	how many cores to parallelize
verbose	verbose logical flag
na.rm	logical flag to remove rows with NA counts
filt.norm	logical flag remove any sites that have allele fraction of 0 or 1 or NA in MAF

Value

nil

Author(s)

Marcin Imielinski

`is.paired.end` *Check if BAM file is paired end by using 0x1 flag*

Description

Check if BAM file is paired-end by using 0x1 flag, pipes to 'samtools' via command line

Create GRanges of read mates from reads

Usage

```
is.paired.end(bams)

get.mate.gr(reads)
```

Arguments

`bams`	vector of input BAMs

Value

boolean returns TRUE if BAM file is paired-end, returns FALSE if BAM not paired-end

`GRanges` corresponding to mates of reads

`mafcount` *Wrapper around varcount adapted to tumor and normal "paired" bams*

Description

mafcount

Returns base counts for reference and alternative allele for an input tum and norm bam and maf data frame or GRAnges specifying substitutions

maf is a single width GRanges describing variants and field 'ref' (or 'Reference_Allele'), 'alt' (or 'Tum_Seq_Allele1') specifying reference and alt allele. maf is assumed to have width 1 and strand is ignored.

Usage

```
mafcount(tum.bam, norm.bam = NULL, maf, chunk.size = 100, verbose = T,
  mc.cores = 1, ...)
```

Arguments

tum.bam	character scalar or BamFile specifying path to tumor sample
norm.bam	optional character scalar or BamFile specifying path to normal sample
maf	GRanges or data.frame or data.table of imported maf (i.e. output of read.delim or fread)
chunk.size	integer number of variants to extract from bam file at each iteration
verbose	logical flag whether to print verbose output
mc.cores	number of cores to parallelize
...	additional params to pass to varcount

Value

GRanges of maf annotated with fields $alt.count.t, $ref.count.t, $alt.count.n, $ref.count.n

Author(s)

Marcin Imielinski

---

oneoffs — *Calls samtools mpileup to dump tsv of "one off" variants / sites (i.e. that are present in exactly one read per site)*

Description

Calls samtools mpileup to dump tsv of "one off" variants / sites (i.e. that are present in exactly one read per site)

Usage

```
oneoffs(out.file, bam, ref, min.bq = 30, min.mq = 60, indel = FALSE,
  chunksize = 10000, verbose = TRUE)
```

Arguments

out.file	file to dump tsv to
bam	bam file path
ref	fasta path
min.bq	integer minimum base quality
min.mq	integer minimum mapping quality
indel	logical flag whether to collect one off indels (default is substitution)
chunksize	number of mpileup lines to put into memory
verbose	logical flag

Note

The denominator (ie total reads) is just the sum of counts$records

`read.bam` *Read BAM file into GRanges or data.table*

Description

Wrapper around Rsamtools bam scanning functions, by default, returns GRangesList of read pairs for which <at least one> read lies in the supplied interval

Usage

```
read.bam(bam, intervals = NULL, gr = intervals, all = FALSE, bai = NULL,
  pairs.grl = TRUE, stripstrand = TRUE, what = scanBamWhat(),
  unpack.flag = FALSE, verbose = FALSE, tag = NULL, isPaired = NA,
  isProperPair = NA, isUnmappedQuery = NA, hasUnmappedMate = NA,
  isNotPassingQualityControls = NA, isDuplicate = F,
  isValidVendorRead = TRUE, pairs.grl.split = TRUE, as.data.table = FALSE,
  ignore.indels = FALSE, ...)
```

Arguments

`bam`	Input bam file. Advisable to make "bam" a BamFile instance instead of a plain string, so that the index does not have to be reloaded.
`intervals`	GRanges of intervals to retrieve
`gr`	GRanges of intervals to retrieve
`stripstrand`	Flag to ignore strand information on the query intervals. Default TRUE
`what`	What fields to pull down from BAM. Default `scanBamWhat()`
`unpack.flag`	Add features corresponding to read flags. Default FALSE
`verbose`	Increase verbosity
`tag`	Additional tags to pull down from the BAM (e.g. 'R2')
`isPaired`	See documentation for `scanBamFlag`. Default NA
`isProperPair`	See documentation for `scanBamFlag`. Default NA
`isUnmappedQuery`	See documentation for `scanBamFlag`. Default NA
`hasUnmappedMate`	See documentation for `scanBamFlag`. Default NA
`isNotPassingQualityControls`	See documentation for `scanBamFlag`. Default NA
`isDuplicate`	See documentation for `scanBamFlag`. Default FALSE
`isValidVendorRead`	See documentation for `scanBamFlag`. Default TRUE
`as.data.table`	Return reads in the form of a data.table rather than GRanges/GRangesList
`ignore.indels`	messes with cigar to read BAM with indels removed. Useful for breakpoint mapping on contigs
`...`	passed to `scanBamFlag`
`bami`	Input bam index file.
`as.grl`	Return reads as GRangesList. Controls whether `get.pairs.grl` does split. Default TRUE

Value

Reads in one of GRanges, GRangesList or data.table

`splice.cigar` *Get coverage as GRanges from BAM on custom set of GRanges*

Description

Takes GRanges or GappedAlignments object "reads" and parses cigar fields to return GRanges or GRangesList corresponding to spliced alignments on the genome, which correspond to portions of the cigar

i.e. each outputted GRanges/GRangesList element contains the granges corresponding to all non-N portions of cigar string

If GRangesList provided as input (e.g. paired reads) then all of the spliced ranges resulting from each input GRangesList element will be put into the corresponding output GRangesList element

NOTE: does not update MD tag

If use.D = TRUE, then will treat "D" flags (deletion) in addition to "N" flags as indicative of deletion event.

Usage

```
splice.cigar(reads, verbose = TRUE, fast = TRUE, use.D = TRUE,
  rem.soft = TRUE, get.seq = FALSE, return.grl = TRUE)
```

Arguments

`reads`	GenomicRanges or GappedAlignments or data.frame input reads
`verbose`	boolean verbose flag (default == TRUE)
`fast`	boolean Flag to use 'GenomicAlignments::cigarRangesAlongReferenceSpace()' to translate CIGAR to GRanges (default == TRUE)
`use.D`	boolean Treats "D" tags as deletions, along with "N" tags (default == TRUE)
`rem.soft`	boolean Pick up splice 'S', soft-clipped (default == TRUE)
`get.seq`	boolean Get InDels (default == TRUE)
`return.grl`	boolean Return as GRangesList (default == TRUE)

varbase — *Returns variant bases and ranges from GRanges or GappedAlignments input*

Description

Takes GRanges or GappedAlignments object "reads" and uses cigar, MD, seq fields to return variant bases and ranges

Teturns GRangesList (of same length as input) of variant base positions with character vector $varbase field populated with variant bases for each GRanges item in grl[[k]], with the following handling for insertions, deletions, and substitution GRange's:

Substitutions: nchar(gr$varbase) = width(gr) of the corresponding var Insertions: nchar(gr$varbase)>=1, width(gr) ==0 Deletions: gr$varbase = '', width(gr)>=1

Each GRanges also has $type flag which shows the cigar string code for the event i.e. S = soft clip –> varbase represents clipped bases I = insertion –> varbase represents inserted bases D = deletion –> varbase is empty X = mismatch –> varbase represents mismatched bases

Usage

```
varbase(reads, soft = TRUE, verbose = TRUE)
```

Arguments

reads	GenomicRanges or GRangesList or GappedAlignments or data.frame/data.table reads to extract variants from
soft	boolean Flag to include soft-clipped matches (default == TRUE)
verbose	boolean verbose flag (default == TRUE)

varcount — *Wrapper around applyPileups*

Description

takes in vector of bam paths, GRanges corresponding to sites / territories to query, and outputs a list with fields $counts = 3D matrix of base counts (A, C, G, T, N) x sites x bams subject to mapq and baseq thresholds #'

(uses varbase)

... = other args go to read.bam

Usage

```
varcount(bams, gr, min.mapq = 0, min.baseq = 20, max.depth = 500,
  indel = F, ...)
```

Arguments

bams	character vector of paths to bam files
gr	granges of (width 1) sites ie intervals at which to compute base coujnts
min.mapq	minimal mapping quality at which to compute bases
max.depth	max read depth to consider
indel	logical flag whether to consider indels (default FALSE)
max.baseq	minimal base qualitya t which to compute bases

Value

input GRanges gr annotated with fields $alt.count.t, $ref.count.t, $alt.count.n, $ref.count.n

Author(s)

Marcin Imielinski

Index

bam.cov.gr, 2
bam.cov.tile, 3
bamflag, 4
bamtag, 4

chunk, 5
count.clips, 5
countCigar, 6

get.mate.gr, 6
get.mate.gr*(is.paired.end)*, 8
get.pairs.grl, 7

hets, 7

is.paired.end, 8

mafcount, 8

oneoffs, 9

read.bam, 10

splice.cigar, 11

varbase, 12
varcount, 12