Package ‘Rsubread’

March 29, 2017

Version 1.24.2

Date 2017-03-17

Title Subread sequence alignment for R

Author Wei Shi and Yang Liao with contributions from Gordon Smyth, Jenny Dai and Timothy Triche, Jr.

Maintainer Wei Shi <shi@wehi.edu.au>

Description Provides powerful and easy-to-use tools for analyzing next-gen sequencing read data. Includes quality assessment of sequence reads, read alignment, read summarization, exon-exon junction detection, fusion detection, detection of short and long indels, absolute expression calling and SNP calling. Can be used with reads generated from any of the major sequencing platforms including Illumina GA/HiSeq/MiSeq, Roche GS-FLX, ABI SOLiD and LifeTech Ion PGM/Proton sequencers.

URL http://bioconductor.org/packages/release/bioc/html/Rsubread.html

License GPL-3

biocViews Sequencing, Alignment, SequenceMatching, RNASeq, ChIPSeq, GeneExpression, GeneRegulation, Genetics, SNP, GeneticVariability, Preprocessing, QualityControl, GenomeAnnotation, Software

NeedsCompilation yes

R topics documented:

align 2
atgcContent 8
buildindex 9
createAnnotationFile 10
detectionCall 11
detectionCallAnnotation 12
exactSNP 12
featureCounts 14
findCommonVariants 21
getInBuiltAnnotation 22
processExons 23

propmapped 24
qualityScores 25
removeDupReads 26
repair 27
RsubreadUsersGuide 28
sam2bed 28

Index **30**

`align` *Align sequence reads to a reference genome via seed-and-vote*

Description

`align` function can align both DNA and RNA sequencing reads. `Subjunc` is an RNA-seq aligner and it reports full alignment of each read (`align` reports partial alignment for exon spanning reads).

Usage

```
align(
# index for reference sequences
index,

# input reads and output
readfile1,
readfile2=NULL,
type="rna",
input_format="gzFASTQ",
output_format="BAM",
output_file=paste(as.character(readfile1),"subread",output_format,sep="."),

# offset value added to Phred quality scores of read bases
phredOffset=33,

# thresholds for mapping
nsubreads=10,
TH1=3,
TH2=1,
maxMismatches=3,

# unique mapping and multi-mapping
unique=TRUE,
nBestLocations=1,

# indel detection
indels=5,
complexIndels=FALSE,

# read trimming
nTrim5=0,
nTrim3=0,
```

```
# distance and orientation of paired end reads
minFragLength=50,
maxFragLength=600,
PE_orientation="fr",

# number of CPU threads
nthreads=1,

# read group
readGroupID=NULL,
readGroup=NULL,

# color space reads
color2base=FALSE,

# dynamic programming
DP_GapOpenPenalty=-1,
DP_GapExtPenalty=0,
DP_MismatchPenalty=0,
DP_MatchScore=2,

# detect structural variants
detectSV=FALSE,

# gene annotation
useAnnotation=FALSE,
annot.inbuilt="mm10",
annot.ext=NULL,
isGTF=FALSE,
GTF.featureType="exon",
GTF.attrType="gene_id",
chrAliases=NULL)

subjunc(
# index for reference sequences
index,

# input reads and output
readfile1,
readfile2=NULL,
input_format="gzFASTQ",
output_format="BAM",
output_file=paste(as.character(readfile1),"subjunc",output_format,sep="."),

# offset value added to Phred quality scores of read bases
phredOffset=33,

# thresholds for mapping
nsubreads=14,
TH1=1,
TH2=1,
maxMismatches=3,
```

```
# unique mapping and multi-mapping
unique=TRUE,
nBestLocations=1,

# indel detection
indels=5,
complexIndels=FALSE,

# read trimming
nTrim5=0,
nTrim3=0,

# distance and orientation of paired end reads
minFragLength=50,
maxFragLength=600,
PE_orientation="fr",

# number of CPU threads
nthreads=1,

# read group
readGroupID=NULL,
readGroup=NULL,

# color space reads
color2base=FALSE,

# dynamic programming
DP_GapOpenPenalty=-1,
DP_GapExtPenalty=0,
DP_MismatchPenalty=0,
DP_MatchScore=2,

# detect all junctions including gene fusions
reportAllJunctions=FALSE,

# gene annotation
useAnnotation=FALSE,
annot.inbuilt="mm10",
annot.ext=NULL,
isGTF=FALSE,
GTF.featureType="exon",
GTF.attrType="gene_id",
chrAliases=NULL)
```

Arguments

`index`	character string giving the basename of index file. Index files should be located in the current directory.
`readfile1`	a character vector including names of files that include sequence reads to be aligned. For paired-end reads, this gives the list of files including first reads

in each library. File format is FASTQ/FASTA by default. See `input_format` option for more supported formats.

`readfile2` a character vector giving names of files that include second reads in paired-end read data. Files included in `readfile2` should be in the same order as their mate files included in `readfile1` .`NULL` by default.

`type` a character string or an integer giving the type of sequencing data. Possible values include `rna` (or `0`; RNA-seq data) and `dna` (or `1`; genomic DNA-seq data such as WGS, WES, ChIP-seq data etc.). Character strings are case insensitive.

`input_format` character string specifying format of read input files. `gzFASTQ` by default (this also includes FASTQ, FASTA and gzipped FASTA formats). Other supported formats include `SAM` and `BAM`. Character values are case insensitive.

`output_format` character string specifying format of output file. `BAM` by default. Acceptable formats include `SAM` and `BAM`.

`output_file` a character vector specifying names of output files. By default, names of output files are set as the file names provided in `readfile1` added with an suffix string.

`phredOffset` numeric value added to base-calling Phred scores to make quality scores (represented as ASCII letters). Possible values include `33` and `64`. By default, `33` is used.

`nsubreads` numeric value giving the number of subreads extracted from each read.

`TH1` numeric value giving the consensus threshold for reporting a hit. This is the threshold for the first reads if paired-end read data are provided.

`TH2` numeric value giving the consensus threhold for the second reads in paired-end data.

`maxMismatches` numeric value giving the maximum number of mis-matched bases allowed in the alignment. `3` by default. Mis-matches found in soft-clipped bases are not counted.

`unique` logical indicating if uniquely mapped reads should be reported only. `TRUE` by default. Uniquely mapped reads must have one (1) mapping location that has less mis-matched bases than other candidate locations.

`nBestLocations` numeric value giving the maximal number of equally-best mapping locations allowed to be reported for the read. 1 by default. The allowed value is between 1 to 16 (inclusive). 'NH' tag is used to indicate how many alignments are reported for the read and 'HI' tag is used for numbering the alignments reported for the same read, in the output. Note that the `unique` argument takes precedence over `nBestLocations` argument.

`indels` numeric value giving the maximum number of insertions/deletions allowed during the mapping. `5` by default.

`complexIndels` logical indicating if complex indels will be detected. If `TRUE`, the program will try to detect multiple short indels that occurs concurrently in a small genomic region (indels could be as close as 1bp apart).

`nTrim5` numeric value giving the number of bases trimmed off from 5' end of each read. `0` by default.

`nTrim3` numeric value giving the number of bases trimmed off from 3' end of each read. `0` by default.

`minFragLength` numeric value giving the minimum fragment length. 50 by default.

`maxFragLength` numeric value giving the maximum fragment length. 600 by default.

PE_orientation
: character string giving the orientation of the two reads from the same pair. It has three possible values including `fr`, `ff` and `rf`. Letter `f` denotes the forward strand and letter `r` the reverse strand. `fr` by default (ie. the first read in the pair is on the forward strand and the second read on the reverse strand).

nthreads
: numeric value giving the number of threads used for mapping. `1` by default.

readGroupID
: a character string giving the read group ID. The specified string is added to the read group header field and also be added to each read in the mapping output. `NULL` by default.

readGroup
: a character string in the format of `tag:value`. This string will be added to the read group (RG) header in the mapping output. `NULL` by default.

color2base
: logical. If `TRUE`, color-space read bases will be converted to base-space bases in the mapping output. Note that the mapping itself will still be performed at color-space. `FALSE` by default.

DP_GapOpenPenalty
: a numeric value giving the penalty for opening a gap when using the Smith-Waterman dynamic programming algorithm to detect insertions and deletions. The Smith-Waterman algorithm is only applied for those reads which are found to contain insertions or deletions. `-1` by default.

DP_GapExtPenalty
: a numeric value giving the penalty for extending the gap, used by the Smith-Waterman algorithm. `0` by default.

DP_MismatchPenalty
: a numeric value giving the penalty for mismatches, used by the Smith-Waterman algorithm. `0` by default.

DP_MatchScore
: a numeric value giving the score for matches used by the Smith-Waterman algorithm. `2` by default.

detectSV
: logical indicating if structural variants (SVs) will be detected during read mapping. See below for more details.

reportAllJunctions
: logical indicating if all discovered junctions will be reported. This includes exon-exon junctions and also gene fusions. Presence of donor/receptor sites is not required when `reportAllJunctions` is `TRUE`. This option should only be used for RNA-seq data.

useAnnotation
: logical indicating if gene annotation information will be used in the mapping. `FALSE` by default.

annot.inbuilt
: a character string specifying an in-built annotation used for read summarization. It has four possible values including `"mm10"`, `"mm9"`, `"hg38"` and `"hg19"`, corresponding to the NCBI RefSeq annotations for genomes 'mm10', 'mm9', 'hg38' and 'hg19', respectively. `"mm10"` by default. See `featureCounts` function for more details on the in-built annotations.

annot.ext
: A character string giving name of a user-provided annotation file or a data frame including user-provided annotation data. If the annotation is in GTF format, it can only be provided as a file. If it is in SAF format, it can be provided as a file or a data frame. If an annotation is provided via `annot.ext`, the `annot.inbuilt` parameter will be ignored. See `featureCounts` function for more details about this parameter.

isGTF
: logical indicating if the annotation provided via the `annot.ext` argument is in GTF format or not. `FALSE` by default. This option is only applicable when `annot.ext` is not `NULL`.

`GTF.featureType`
: a character string denoting the type of features that will be extracted from a GTF annotation. "exon" by default. This argument is only applicable when isGTF is TRUE.

`GTF.attrType`
: a character string denoting the type of attributes in a GTF annotation that will be used to group features. "gene_id" by default. The grouped features are called meta-features. For instance, a feature can be an exon and several exons can be grouped into a gene (meta-feature). This argument is only applicable when isGTF is TRUE.

`chrAliases`
: a character string providing the name of a comma-delimited text file that includes aliases of chromosome names. This file should contain two columns. First column contains names of chromosomes included in the SAF or GTF annotation and second column contains corresponding names of chromosomes in the reference genome. No column headers should be provided. Also note that chromosome names are case sensitive. This file can be used to match chromosome names between the annotation and the reference genome.

Details

The align function implements the Subread aligner (Liao et al., 2013) that uses a new mapping paradigm called "seed-and-vote". Subread is general-purpose aligner that can be used to align both genomic DNA-seq reads and RNA-seq reads.

Subjunc is designed for mapping RNA-seq reads. The major difference between Subjunc and Subread is that Subjunc reports discovered exon-exon junctions and it also performs full alignments for every read including exon-spanning reads. Subread uses the largest mappable regions in the reads to find their mapping locations. The seed-and-vote paradigm has been found to be not only more accurate than the conventional seed-and-extend (adopted by most aligners) in read mapping, but it is a lot more efficient (Liao et al., 2013).

Both Subread and Subjunc can be used to align reads generated from major sequencing platforms including Illumina GA/HiSeq, ABI SOLiD, Roche 454 and Ion Torrent sequencers. Note that to map color-space reads (e.g. SOLiD reads), a color-space index should be built for the reference genome (see buildindex for details).

Subread and Subjunc have adjustable memory usage. See buildindex function for more details.

Mapping performance is largely determined by the number of subreads extracted from each read nsubreads and the consensus threshold TH1 (also TH2 for paired-end read data). Default settings are recommended for most of the read mapping tasks.

Subjunc requires donor/receptor sites to be present when detecting exon-exon junctions. It can detect up to four junction locations in each exon-spanning read.

detectSV option should be used for SV detection in genomic DNA sequencing data. For RNA-seq data, users may use subjunc with the reportAllJunctions option to detect SVs (and also junctions). For each library, breakpoints detected from SV events will be saved to a file with suffix name '.breakpoints.txt', which includes chromosomal coordinates of SV breakpoints and numbers of supporting reads. The BAM/SAM output includes extra fields to describe the complete alignments of breakpoint-containing reads. For a breakpoint-containing read, mapping of its major sequence segment is described in the main fields of BAM/SAM ouptut whereas mapping of its minor sequence segment, which does not map along with the major segment due to the presence of a breakpoint, is described in the extra fields including 'CC'(Chr), 'CP'(Position),'CG'(CIGAR) and 'CT'(strand). Note that each breakpoint-containing read occupies only one row in BAM/SAM output.

Value

No value is produced but SAM or BAM format files are written to the current working directory. For `Subjunc`, BED files including discovered exon-exon junctions are also written to the current working directory.

Author(s)

Wei Shi and Yang Liao

References

Yang Liao, Gordon K Smyth and Wei Shi. The Subread aligner: fast, accurate and scalable read mapping by seed-and-vote. Nucleic Acids Research, 41(10):e108, 2013.

Examples

```
# Build an index for the artificial sequence included in file 'reference.fa'.
library(Rsubread)
ref <- system.file("extdata","reference.fa",package="Rsubread")
buildindex(basename="./reference_index",reference=ref)

# align a sample read dataset ('reads.txt') to the sample reference
reads <- system.file("extdata","reads.txt.gz",package="Rsubread")
align(index="./reference_index",readfile1=reads,output_file="./Rsubread_alignment.BAM",phredOffset=64)
```

`atgcContent` *Calculate percentages of nucletodies A, T, G and C in a sequencing read datafile*

Description

Calculate percentages of nucletodies A, T, G and C

Usage

```
atgcContent(filename, basewise=FALSE)
```

Arguments

`filename`	character string giving the name of input FASTQ/FASTA file
`basewise`	logical. If `TRUE`, nucleotide percentages will be calculated for each base position in the read across all the reads. By default, percentages are calculated for the entire dataset.

Details

Sequencing reads could contain letter "N" besides "A", "T", "G" and "C". Percentage of "N" in the read dataset is calcuated as well.

The `basewise` calculation is useful for examining the GC bias towards the base position in the read. By default, the percentages of nucleotides in the entire dataset will be reported.

Value

A named vector containing percentages for each nucleotide type if `basewise` is `FALSE`. Otherwise, a data matrix containing nucleotide percentages for each base position of the reads.

Author(s)

Zhiyin Dai and Wei Shi

---

buildindex — *Build index for a reference genome*

Description

An index needs to be built before read mapping can be performed. This function creates a hash table for the reference genome, which can then be used by Subread and Subjunc aligners for read alignment.

Usage

```
buildindex(basename,reference,gappedIndex=TRUE,indexSplit=TRUE,memory=8000,
TH_subread=100,colorspace=FALSE)
```

Arguments

`basename`	character string giving the basename of created index files.
`reference`	charater string giving the name of a FASTA-format file that includes sequences of all chromosomes and contigs.
`gappedIndex`	logical. If `FALSE`, 16mers (subreads) will be extracted from every chromosomal location of a reference genome and then they will be used to build a hash table index. By default(`TRUE`), subreads are extracted in every three bases from the genome.
`indexSplit`	logical. If `TRUE`, the built index is allowed to be splitted into multiple segments. The number of such segments is determined by `memory` value, genome size and permitting of gaps between subreads(`gappedIndex`). If `indexSplit` is set to `FALSE`, a single-segment index (no splitting) will be generated regardless of what value is chosen for `memory`.
`memory`	numeric value specifying the amount of memory to be requested in megabytes. 8000 MB by default.
`TH_subread`	numeric value specifying the threshold for removing highly repetitive subreads (16bp mers). 100 by default. Subreads will be excluded from the index if they occur more than threshold number of times in the genome.
`colorspace`	logical. If `TRUE`, a color space index will be built. Otherwise, a base space index will be built.

Details

This function generates a hash table (an index) for a reference genome, in which keys are subreads (16mers) and values are their chromosomal locations in the reference genome. By default, subreads will be extracted in every three bases from a reference genome. However, if `gappedIndex` is set to `FALSE`, then subreads will be extracted from every chromosomal location of genome for index building. The built index can then be used by Subread (`align`) and `subjunc` aligners to map reads(Liao et al. 2013).

Highly repetitive subreads (or uninformative subreads) are excluded from the hash table so as to reduce mapping ambiguity. `TH_subread` specifies the maximal number of times a subread is allowed to occur in the reference genome to be included in hash table.

The built index might be splitted into multiple segments if its size is greater than `memory` value. The number of such segments is dependent on `memory` value, size of reference genome and whether gaps are allowed between subreads extracted from genome. Only one segment is loaded into memory at any time when read alignment is being carried out. The larger the `memory` value, the faster the read mapping will be. If `indexSplit` is set to `FALSE`, the index will not be splitted and this will enable maximum mapping speed to be achieved.

The index needs to be built only once and it can then be re-used in the subsequent alignments.

Value

No value is produced but index files are written to the current working directory.

Author(s)

Wei Shi and Yang Liao

References

Yang Liao, Gordon K Smyth and Wei Shi. The Subread aligner: fast, accurate and scalable read mapping by seed-and-vote. Nucleic Acids Research, 41(10):e108, 2013.

Examples

```
# Build an index for the artifical sequence included in file 'reference.fa'
library(Rsubread)
ref <- system.file("extdata","reference.fa",package="Rsubread")
buildindex(basename="./reference_index",reference=ref)
```

`createAnnotationFile` *Create an annotation file from a GRanges object, suitable for featureCounts()*

Description

Any of rtracklayer::import.bed('samplesubjunc.bed'), unlist(spliceGraph(TxDb)), transcripts(TxDb), exons(TxDb), or features(FDB) will produce a GRanges object containing usable features for read counting.

This function converts a suitably streamlined GRanges object into annotations which can then be used by featureCounts() to quickly count aligned reads.

The GRanges object must contain an elementMetadata column named 'id'.

Usage

```
createAnnotationFile(GR)
write.Rsubread(GR)
```

Arguments

GR	The GRanges object to convert to an Rsubread annotation file

Value

A data frame with five columns named GeneID, Chr, Start, End and Strand.

Author(s)

Tim Triche, Jr. and Wei Shi

Examples

```
## Not run:
  library(TxDb.Hsapiens.UCSC.hg19.lincRNAsTranscripts)
  hg19LincRNAs <- transcripts(TxDb.Hsapiens.UCSC.hg19.lincRNAsTranscripts)
  names(values(hg19LincRNAs)) <- gsub('tx_id','id',names(values(hg19LincRNAs)))
  annot_for_featureCounts <- createAnnotationFile(hg19LincRNAs)

## End(Not run)
```

---

detectionCall — *Determine detection p values for each gene in an RNA-seq dataset*

Description

Use GC content adjusted background read counts to determine the detection p values for each gene

Usage

```
detectionCall(dataset, species="hg", plot=FALSE)
```

Arguments

dataset	a character string giving the filename of a SAM format file, which is the output of read alignment.
species	a character string specifing the species. Options are hg and mm.
plot	logical, indicating whether a density plot of detection p values will be generated.

Value

A data frame which includes detection p values and annotation information for each genes.

Author(s)

Zhiyin Dai and Wei Shi

detectionCallAnnotation

Generate annotation data used for calculating detection p values

Description

This is for internal use only.

Usage

```
detectionCallAnnotation(species="hg", binsize=2000)
```

Arguments

species	character string specifying the species to analyase
binsize	binsize of integenic region

Details

This is an internal function and should not be called by users directly.

It takes as input the annotation files produced by `processExons` function, calculates GC percentages for each exon of genes and also for intergenic regions and add GC info into the annotations. The new annotation data are then saved to files which can be used by `detectionCall` function for calling absolutely expressed genes.

Value

Two annotation files, which contain GC content for exons of genes and for intergenic regions respectively, are written to the current working directory. This function returns a `NULL` object.

Author(s)

Zhiyin Dai and Wei Shi

exactSNP

exactSNP - an accurate and efficient SNP caller

Description

Measure background noises and perform Fisher's Exact tests to detect SNPs.

Usage

```
exactSNP(readFile,isBAM=FALSE,refGenomeFile,SNPAnnotationFile=NULL,
outputFile=paste(readFile,".exactSNP.VCF",sep=""),qvalueCutoff=12,minAllelicFraction=0,
minAllelicBases=1,minReads=1,maxReads=3000,minBaseQuality=13,nTrimmedBases=3,nthreads=1)
```

Arguments

- `readFile` — a character string giving the name of a file including read mapping results. This function takes as input a SAM file by default. If a BAM file is provided, the `isBAM` argument should be set to `TRUE`.
- `isBAM` — logical indicating if the file provided via `readFile` is a BAM file. `FALSE` by default.
- `refGenomeFile` — a character string giving the name of a file that includes reference sequences (FASTA format).
- `SNPAnnotationFile` — a character string giving the name of a VCF-format file that includes annotated SNPs. Such annotation files can be downloaded from public databases such as the dbSNP database. Incorporating known SNPs into SNP calling has been found to be helpful. However note that the annotated SNPs may or may not be called for the sample being analyzed.
- `outputFile` — a character string giving the name of the output file to be generated by this function. The output file includes all the reported SNPs (in VCF format). It includes discovered indels as well.
- `qvalueCutoff` — a numeric value giving the q-value cutoff for SNP calling at sequencing depth of 50X. `12` by default. The q-value is calcuated as -log10(p), where p is the p-value yielded from the Fisher's Exact test. Note that this function automatically adjusts the q-value cutoff for each chromosomal location according to its sequencing depth, based on this cutoff.
- `minAllelicFraction` — a numeric value giving the minimum fraction of allelic bases out of all read bases included at a chromosomal location required for SNP calling. Its value must be within `0` and `1`. `0` by default.
- `minAllelicBases` — a numeric value giving the minimum number of allelic (mis-matched) bases a SNP must have at a chromosomal location. `1` by default.
- `minReads` — a numeric value giving the minimum number of mapped reads a SNP-containing location must have (ie. the minimum coverage). `1` by default.
- `maxReads` — Specify the maximum number of mapped reads a SNP-containing location can have. `3000` by default. Any location having more than this threshold number of reads will not be considered for SNP calling. This option is useful for removing PCR artefacts.
- `minBaseQuality` — a numeric value giving the minimum base quality score (Phred score) read bases should satisfy before being used for SNP calling. `13` by default(corresponding to base calling p value of 0.05). Read bases with quality scores less than 13 will be excluded from analysis.
- `nTrimmedBases` — a numeric value giving the number of bases trimmed off from each end of the read. `3` by default.
- `nthreads` — a numeric value giving the number of threads/CPUs used. `1` by default.

Details

This function takes as input a SAM/BAM format file, measures local background noise for each chromosomal location and then performs Fisher's exact tests to find statistically significant SNPs .

This function implements a novel algorithm for discovering SNPs. This algorithm is comparable with or better than existing SNP callers, but it is fast more efficient. It can be used to call SNPs for

individual samples (ie. no control samples are required). Detail of the algorithm is described in a manuscript which is currently under preparation.

Value

No value is produced but but a VCF format file is written to the current working directory. This file contains detailed information for discovered SNPs including chromosomal locations, reference bases, alternative bases, read coverages, allele frequencies and p values.

Author(s)

Yang Liao and Wei Shi

---

featureCounts *featureCounts: a general-purpose read summarization function*

Description

This function assigns mapped sequencing reads to genomic features

Usage

```
featureCounts(files,

# annotation
annot.inbuilt="mm10",
annot.ext=NULL,
isGTFAnnotationFile=FALSE,
GTF.featureType="exon",
GTF.attrType="gene_id",
chrAliases=NULL,

# level of summarization
useMetaFeatures=TRUE,

# overlap between reads and features
allowMultiOverlap=FALSE,
minOverlap=1,
fracOverlap=0,
largestOverlap=FALSE,
readExtension5=0,
readExtension3=0,
read2pos=NULL,

# multi-mapping reads
countMultiMappingReads=FALSE,

# fractional counting
fraction=FALSE,

# read filtering
```

```
minMQS=0,
splitOnly=FALSE,
nonSplitOnly=FALSE,
primaryOnly=FALSE,
ignoreDup=FALSE,

# strandness
strandSpecific=0,

# exon-exon junctions
juncCounts=FALSE,
genome=NULL,

# parameters specific to paired end reads
isPairedEnd=FALSE,
requireBothEndsMapped=FALSE,
checkFragLength=FALSE,
minFragLength=50,
maxFragLength=600,
countChimericFragments=TRUE,
autosort=TRUE,

# number of CPU threads
nthreads=1,

# miscellaneous
maxMOp=10,
reportReads=FALSE,
tmpDir=".")
```

Arguments

`files` a character vector giving names of input files containing read mapping results. The files can be in either SAM format or BAM format. The file format is automatically detected by the function.

`annot.inbuilt` a character string specifying an in-built annotation used for read summarization. It has four possible values including `"mm10"`, `"mm9"`, `"hg38"` and `"hg19"`, corresponding to the NCBI RefSeq annotations for genomes 'mm10', 'mm9', 'hg38' and 'hg19', respectively. `"mm10"` by default. The in-built annotation has a SAF format (see below).

`annot.ext` A character string giving name of a user-provided annotation file or a data frame including user-provided annotation data. If the annotation is in GTF format, it can only be provided as a file. If it is in SAF format, it can be provided as a file or a data frame. See below for more details about SAF format annotation. `annot.ext` will override `annot.inbuilt` if they are both provided.

`isGTFAnnotationFile` logical indicating whether the annotation provided via the `annot.ext` argument is in GTF format or not. `FALSE` by default. This option is only applicable when `annot.ext` is not `NULL`.

`GTF.featureType` a character string giving the feature type used to select rows in the GTF annotation which will be used for read summarization. `"exon"` by default. This

argument is only applicable when `isGTFAnnotationFile` is `TRUE`.

`GTF.attrType` a character string giving the attribute type in the GTF annotation which will be used to group features (eg. exons) into meta-features (eg. genes). `"gene_id"` by default. This argument is only applicable when `isGTFAnnotationFile` is `TRUE`.

`chrAliases` a character string giving the name of a chromosome name alias file. This should be a two-column comma-delimited text file. Chromosome name aliases included in this file are used to match chr names in annotation with those in the reads. First column in the file should include chr names in the annotation and second column should include chr names in the reads. Chr names are case sensitive. No column header should be included in the file.

`useMetaFeatures`
logical indicating whether the read summarization should be performed at the feature level (eg. exons) or meta-feature level (eg. genes). If `TRUE`, features in the annotation (each row is a feature) will be grouped into meta-features, using their values in the `GeneID` column in the SAF-format annotation file or using the `GTF.attrType` attribute in the GTF-format annotation file, and then reads will be assiged to the meta-features instead of the features. See below for more details.

`allowMultiOverlap`
logical indicating if a read is allowed to be assigned to more than one feature (or meta-feature) if it is found to overlap with more than one feature (or meta-feature). `FALSE` by default.

`minOverlap` `integer` giving the minimum number of overlapped bases required for assigning a read to a feature (or a meta-feature). For assignment of read pairs (fragments), number of overlapping bases from each read in the same pair will be summed. If a negative value is provided, then a gap of up to specified size will be allowed between read and the feature that the read is assigned to. `1` by default.

`fracOverlap` `numeric` giving minimum fraction of overlapping bases in a read that is required for read assignment. Value should be within range [0,1]. `0` by default. Number of overlapping bases is counted from both reads if paired end. Both this option and `minOverlap` option need to be satisfied before a read can be assigned.

`largestOverlap` If `TRUE`, a read (or read pair) will be assigned to the feature (or meta-feature) that has the largest number of overlapping bases, if the read (or read pair) overlaps with multiple features (or meta-features).

`readExtension5` `integer` giving the number of bases extended upstream from 5' end of each read. `0` by default.

`readExtension3` `integer` giving the number of bases extended downstream from 3' end of each read. `0` by default.

`read2pos` Specifying whether each read should be reduced to its 5' most base or 3' most base. It has three possible values: `NULL`, `5` (denoting 5' most base) and `3` (denoting 3' most base). The default value is `NULL`. With the default value, the whole read is used for summarization. When `read2pos` is set to `5` (or `3`), read summarization will be performed based on the 5' (or 3') most base position. `read2pos` can be used together with `readExtension5` and `readExtension3` parameters to set any desired length for reads.

`countMultiMappingReads`
logical indicating if multi-mapping reads/fragments should be counted, `FALSE` by default. If `TRUE`, a multi-mapping read will be counted up to N times if it has N reported mapping locations. This function uses the 'NH' tag to find multi-mapping reads.

`fraction` logical indicating if fractional counts are produced for multi-mapping reads and/or multi-overlapping reads. `FALSE` by default. See below for more details.

`minMQS` `integer` giving the minimum mapping quality score a read must satisfy in order to be counted. For paired-end reads, at least one end should satisfy this criteria. `0` by default.

`splitOnly` logical indicating whether only split alignments (their CIGAR strings contain letter 'N') should be included for summarization. `FALSE` by default. Example split alignments are exon-spanning reads from RNA-seq data. `useMetaFeatures` should be set to `FALSE` and `allowMultiOverlap` should be set to `TRUE`, if the purpose of summarization is to assign exon-spanning reads to all their overlapping exons.

`nonSplitOnly` logical indicating whether only non-split alignments (their CIGAR strings do not contain letter 'N') should be included for summarization. `FALSE` by default.

`primaryOnly` logical indicating if only primary alignments should be counted. Primary and secondary alignments are identified using bit 0x100 in the Flag field of SAM/BAM files. If `TRUE`, all primary alignments in a dataset will be counted no matter they are from multi-mapping reads or not (ie. `countMultiMappingReads` is ignored).

`ignoreDup` logical indicating whether reads marked as duplicates should be ignored. `FALSE` by default. Read duplicates are identified using bit Ox400 in the FLAG field in SAM/BAM files. The whole fragment (read pair) will be ignored if paired end.

`strandSpecific` `integer` indicating if strand-specific read counting should be performed. It has three possible values: `0` (unstranded), `1` (stranded) and `2` (reversely stranded). `0` by default.

`juncCounts` logical indicating if number of reads supporting each exon-exon junction will be reported. Junctions are identified from those exon-spanning reads in input data. `FALSE` by default.

`genome` a character string giving the name of a FASTA-format file that includes the reference sequences used in read mapping that produced the provided SAM/BAM files. `NULL` by default. This argument should only be used when `juncCounts` is `TRUE`. Note that providing reference sequences is optional when `juncCounts` is set to `TRUE`.

`isPairedEnd` logical indicating if paired-end reads are used. If `TRUE`, fragments (templates or read pairs) will be counted instead of individual reads. `FALSE` by default.

`requireBothEndsMapped`
logical indicating if both ends from the same fragment are required to be successfully aligned before the fragment can be assigned to a feature or meta-feature. This parameter is only appliable when `isPairedEnd` is `TRUE`.

`checkFragLength`
logical indicating if the two ends from the same fragment are required to satisify the fragment length criteria before the fragment can be assigned to a feature or meta-feature. This parameter is only appliable when `isPairedEnd` is `TRUE`. The fragment length criteria are specified via `minFragLength` and `maxFragLength`.

`minFragLength` `integer` giving the minimum fragment length for paired-end reads. `50` by default.

`maxFragLength` `integer` giving the maximum fragment length for paired-end reads. `600` by default. `minFragLength` and `maxFragLength` are only applicable when `isPairedEnd` is `TRUE`. Note that when a fragment spans two or more exons, the observed fragment length might be much bigger than the nominal fragment length.

countChimericFragments
: logical indicating whether a chimeric fragment, which has its two reads mapped to different chromosomes, should be counted or not. TRUE by default.

autosort
: logical specifying if the automatic read sorting is enabled. This option is only applicable for paired-end reads. If TRUE, reads will be automatically sorted by their names if reads from the same pair are found not to be located next to each other in the input. No read sorting will be performed if there are no such reads found.

nthreads
: integer giving the number of threads used for running this function. 1 by default.

maxMOp
: integer giving the maximum number of 'M' operations (matches or mis-matches) allowed in a CIGAR string. 10 by default. Both 'X' and '=' operations are treated as 'M' and adjacent 'M' operations are merged in the CIGAR string.

reportReads
: logical indicating if read counting result for each read/fragment is saved to a file. If TRUE, read counting results for reads/fragments will be saved to a tab-delimited file that contains four columns including name of read/fragment, status(assigned or the reason if not assigned), name of target feature/meta-feature and number of hits if the read/fragment is counted multiple times. Name of the file is the same as name of the input read file except a suffix '.featureCounts' is added. Multiple files will be generated if there is more than one input read file.

tmpDir
: a character string specifying the directory under which intermediate files are saved (later removed). By default, current working directory is used.

Details

featureCounts is a general-purpose read summarization function, which assigns to the genomic features (or meta-features) the mapped reads that were generated from genomic DNA and RNA sequencing.

This function takes as input a set of files containing read mapping results output from a read aligner (e.g. align or subjunc), and then assigns mapped reads to genomic features. Both SAM and BAM format input files are accepted.

featureCounts accepts two annotation formats: SAF (Simplified Annotation Format) and GTF/GFF formats. Specification of GTF/GFF format can be found at https://genome.ucsc.edu/FAQ/FAQformat.html. SAF is a simplified annotation format and below shows an example of this format:

```
GeneID Chr Start End Strand
497097 chr1 3204563 3207049 -
497097 chr1 3411783 3411982 -
497097 chr1 3660633 3661579 -
100503874 chr1 3637390 3640590 -
100503874 chr1 3648928 3648985 -
100038431 chr1 3670236 3671869 -
...
```

SAF annotation includes the following five required columns: GeneID, Chr, Start, End and Strand. The GeneID column includes identifiers of features. These identifiers can be integer or character string. The Chr column includes chromosome names of features and these names should match the chromosome names includes in the provided SAM/BAM files. The Start and End columns include start and end coordinates of features, respectively. Both start and end coordinates are inclusive. The Strand column indicates the strand of features ("+" or "-"). Column names in a SAF annotation

should be the same as those shown in the above example (case insensitive). Columns can be in any order. Extra columns are allowed to be added into the annotation.

In-built annotations, which were generated based on NCBI RefSeq gene annotations, are provided to faciliate convenient read summarization. We provide in-built annotations for the following genomes: "hg38", "hg19", "mm10" and "mm9". The content of in-built annotations can be accessed via the getInBuiltAnnotation function. These annotations have a SAF format.

The in-built annotations are a modified version of NCBI RefSeq gene annotations. We downloaded the RefSeq gene annotations from NCBI ftp server (eg. RefSeq annotation for mm10 was downloaded from ftp://ftp.ncbi.nlm.nih.gov/genomes/M_musculus/ARCHIVE/BUILD.38.1/mapview/seq_gene.md.gz). We then used these annotations to create our in-built annotations. For each gene, we used its CDS (coding DNA sequence) and UTR (untranslated) regions provided in the original annotation to construct a list of exons, by merging and concatenating all CDSs and UTRs belonging to the same gene. Exons within each gene include all chromosomal bases included in the original CDS and UTR regions, but they include each base only once (they might be included multiple times in the original CDSs and UTRs). Also, exons within the same gene do not overlap with each other.

Users may provide an external annotation for read summarization via the annot.ext argument. If the external annotation is in SAF format, it can be provdied as either a data.frame or a tab-delimited text file with proper column names included. If it is in GTF/GFF format, it should be provided as a file only (and isGTFAnnotationFile should be set to TRUE).

featureCounts function uses the GTF.attrType attribute in a GTF/GFF annotation to group features to form meta-features when performing read summarization at meta-feature level.

The argument useMetaFeatures specifies whether read summarization should be performed at feature level or at meta-feature level. A feature represents a continuous genomic region and a meta-feature is a group of features. For instance, an exon is a feature and a gene comprising one or more exons is a meta-feature. To assign reads to meta-features, featureCounts firstly groups into meta-features the features that have the same gene identifiers. featureCounts looks for gene identifiers in GeneID column of a SAF annotation or by using GTF.attrType attribute in a GTF/GFF annotation. Then for each read featureCounts searches for meta-features that have at least one feature that overlaps with the read. A read might be found to overlap with more than one feature within the same meta-feature (eg. an exon-spanning read overlaps with more than one exon from the same gene), however this read will still be counted only once for the meta-feature.

RNA-seq reads are often summarized to meta-features to produce read counts for genes. Further downstream analysis can then be carried out to discover differentially expressed genes. Feature-level summarization of RNA-seq data often yields exon-level read counts, which is useful for investigating alternative splicing of genes.

featureCounts provides multiple options to count multi-mapping reads (reads mapping to more than one location in the reference genome). Users can choose to ignore such reads by setting countMultiMappingReads to FALSE, or fully count every alignment reported for a multi-mapping read by setting countMultiMappingReads to TRUE (each alignment carries 1 count), or count each alignment fractionally by setting both countMultiMappingReads and fraction to TRUE (each alignment carries 1/x count where x is the total number of alignments reported for the read).

featureCounts also provides multiple options to count multi-overlapping reads (reads overlapping with more than one meta-feature when conducting meta-feature-level summarization or overlapping with more than one feature when conducting feature-level summarization). Users can choose to ignore such reads by setting allowMultiOverlap to FALSE, or fully count them for each overlapping meta-feature/feature by setting allowMultiOverlap to TRUE (each overlapping meta-feature/feature receives a count of 1 from a read), or assign a fractional count to each overlapping meta-feature/feature by setting both allowMultiOverlap and fraction to TRUE (each overlapping meta-feature/feature receives a count of 1/y from a read where y is the total number of meta-features/features overlapping with the read).

If a read is both multi-mapping and multi-overlapping, then each overlapping meta-feature/feature will receive a fractional count of `1/(x*y)` if `countMultiMappingReads`, `allowMultiOverlap` and `fraction` are all set to `TRUE`.

When `isPairedEnd` is `TRUE`, fragments (pairs of reads) instead of reads will be counted. `featureCounts` function checks if reads from the same pair are adjacent to each other (this could happen when reads were for example sorted by their mapping locations), and it automatically reorders those reads that belong to the same pair but are not adjacent to each other in the input read file.

Value

A list with the following components:

`counts` a data matrix containing read counts for each feature or meta-feature for each library.

`counts_junction (optional)` a data frame including primary and secondary genes overlapping an exon junction, position information for the left splice site ('Site1') and the right splice site ('Site2') of a junction (including chromosome name, coordinate and strand) and number of supporting reads for each junction in each library. Both primary and secondary genes overlap at least one of the two splice sites of a junction. Secondary genes do not overlap more splice sites than the primary gene. When the primary and secondary genes overlap same number of splice sites, the gene with the smallest leftmost base position is selected as the primary gene. For each junction, no more than one primary gene is reported but there might be more than one secondary genes reported. If a junction does not overlap any genes, it has a missing value in the fields of primary gene and secondary gene. Note that this data frame is only generated when `juncCounts` is `TRUE`.

`annotation` a data frame with six columns including `GeneID`, `Chr`, `Start`, `End` and `Length`. When read summarization was performed at feature level, each row in the data frame is a feature and columns in the data frame give the annotation information for the features. When read summarization was performed at meta-feature level, each row in the data frame is a meta-feature and columns in the data frame give the annotation information for the features included in each meta feature except the `Length` column. For each meta-feature, the `Length` column gives the total length of genomic regions covered by features included in that meta-feature. Note that this length will be less than the sum of lengths of features included in the meta-feature when there are features overlapping with each other. Also note the `GeneID` column gives Entrez gene identifiers when the in-built annotations are used.

`targets` a character vector giving sample information.

`stat` a data frame giving numbers of unassigned reads and the reasons why they are not assigned (eg. ambiguity, multi-mapping, secondary alignment, mapping quality, fragment length, chimera, read duplicate, non-junction and so on), in addition to the number of successfully assigned reads for each library.

Author(s)

Wei Shi and Yang Liao

References

Yang Liao, Gordon K Smyth and Wei Shi. featureCounts: an efficient general-purpose program for assigning sequence reads to genomic features. Bioinformatics, 30(7):923-30, 2014.

See Also

getInBuiltAnnotation

Examples

```
## Not run:
library(Rsubread)

# Summarize SAM format single-end reads using built-in RefSeq annotation for mouse genome mm9:
featureCounts(files="mapping_results_SE.sam",annot.inbuilt="mm9")

# Summarize single-end reads using a user-provided GTF annotation file:
featureCounts(files="mapping_results_SE.sam",annot.ext="annotation.gtf",
isGTFAnnotationFile=TRUE,GTF.featureType="exon",GTF.attrType="gene_id")

# Summarize single-end reads using 5 threads:
featureCounts(files="mapping_results_SE.sam",nthreads=5)

# Summarize BAM format single-end read data:
featureCounts(files="mapping_results_SE.bam")

# Perform strand-specific read counting (strandSpecific=2 if reversely stranded):
featureCounts(files="mapping_results_SE.bam",strandSpecific=1)

# Summarize paired-end reads and counting fragments (instead of reads):
featureCounts(files="mapping_results_PE.bam",isPairedEnd=TRUE)

# Count fragments satisfying the fragment length criteria, eg. [50bp, 600bp]:
featureCounts(files="mapping_results_PE.bam",isPairedEnd=TRUE,
checkFragLength=TRUE,minFragLength=50,maxFragLength=600)

# Count fragments that have both ends successfully aligned without checking the fragment length:
featureCounts(files="mapping_results_PE.bam",isPairedEnd=TRUE,requireBothEndsMapped=TRUE)

# Exclude chimeric fragments from fragment counting:
featureCounts(files="mapping_results_PE.bam",isPairedEnd=TRUE,countChimericFragments=FALSE)

## End(Not run)
```

---

findCommonVariants *Finding the common variants among all input VCF files*

Description

The common variants (inc. SNPs and indels) among all the input files are found. A data frame containing these common variants is returned. The data frame has a similar format as VCF files.

Usage

```
findCommonVariants(VCF_files)
```

Arguments

VCF_files	a character vector giving the names of VCF format files.

Details

This function loads all variants (SNPs and indels) from the input VCF files, and find the common variants that are reported in all the VCF files. If a variant record in a input VCF file has multiple alternative sequences (split by ','), each alternative sequence is treated as a single variant. Two variants in two VCF files are the same only if their genomic locations, their reference sequences, their alternative sequences and their variant types are identical.

This function currently does not support other types of variants other than SNPs and indels.

There are eight columns in the returned data frame: chromosome name, position, identity, reference sequence, alternative sequence, quality, filter and extra information. The input may have more columns; these columns are not included in the data frame. If the identity, the quality, the filter and the extra information for the same variant are different among the input VCF files, those information associated with the lowest quality value of this variant among the VCF files is reported in the resulted data frame. For example, if an SNP on chrX:12345 (A=>G) is reported in all the three input VCF files, and the quality scores in the three VCF files are 100, 10, 50 respectively, the identity, the quality, the filter and the extra information in the second VCF file are reported in the resulted data frame for this SNP.

Value

A data frame containing the common variants among all the input VCF files is returned. The first eight columns are: chromosome name, position, identity, reference sequence, alternative sequence, quality, filter and extra information.

If there are not any common variants, this function returns an NA value.

Author(s)

Yang Liao and Wei Shi

Examples

```
## Not run:
# finding the common variants between to input VCF files: a.vcf and b.vcf
library(Rsubread)
findCommonVariants(c('a.vcf','b.vcf'))

## End(Not run)
```

---

getInBuiltAnnotation *Retrieve in-built annotations provided by featureCounts function*

Description

Retrieve an in-built annotation and save it to a data frame

Usage

```
getInBuiltAnnotation(annotation="mm10")
```

Arguments

annotation	a character string specifying the in-built annotation to be retrieved. It has four possible values including mm10, mm9, hg38 and hg19, corresponding to the NCBI RefSeq annotations for genomes 'mm10', 'mm9', 'hg38' and 'hg19', respectively. mm10 by default.

Details

The featureCounts read summarization function provides in-built annotations for conveniently summarizing reads to genes or exons, and this function allows users to have access to those in-built annotations.

For more information about these annotations, please refer to the help page for featureCounts function.

Value

A data frame with five columns including GeneID, Chr, Start, End and Strand.

Author(s)

Wei Shi

See Also

featureCounts

Examples

```
library(Rsubread)
x <- getInBuiltAnnotation("hg38")
x[1:5,]
```

---

processExons — *Obtain chromosomal coordiates of each exon using NCBI annotation*

Description

This is for internal use.

Usage

```
processExons(filename="human_seq_gene.md", species="hg")
```

Arguments

filename	a character string giving the name of input .md file (NCBI annotation file)
species	a character string specifying the species

Details

This is an internal function and should not be called by users directly.

It processes the NCBI mapview annotation data downloaded from the following links: (these annotations include chromosomal coordinates of UTR and CDS regions of genes).

ftp://ftp.ncbi.nlm.nih.gov/genomes/H_sapiens/mapview/seq_gene.md.gz

ftp://ftp.ncbi.nlm.nih.gov/genomes/M_musculus/mapview/seq_gene.md.gz

This function finds the chromosomal coordinates of intergenic regions (regions between neighbouring genes) and then outputs them to a file. It also outputs to a file chromosomal coordinates of exons of genes by concatenating UTRs with CDSs and merging overlapping CDSs within each gene. The generated annotation files will then be used by `detectionCallAnnotation` function to produce annotation data required by `detectionCall` function.

Value

Two annotation files are written to the current working directory. This function returns a `NULL` object.

Author(s)

Zhiyin Dai and Wei Shi

---

propmapped — *Calculate the proportion of mapped reads/fragments in SAM/BAM files*

Description

Number of mapped reads/fragments will be counted and fraction of such reads/fragments will be calculated.

Usage

```
propmapped(files,countFragments=TRUE,properlyPaired=FALSE)
```

Arguments

`files`	a character vector giving the names of SAM/BAM format files. Format of input files is automatically determined by the function.
`countFragments`	logical, indicating whether reads or fragments (read pairs) should be counted. If `TRUE`, fragments will be counted when paired-end read data are provided. This function automatically detects if the data are single end or paired end. For single end data, each read is treated as a fragment and therefore the value of this parameter should be set to `TRUE`.
`properlyPaired`	logical, indicating if only properly paired reads will be counted. This is only applicable for paired end data. `FALSE` by default.

Details

This function uses the FLAG field in the SAM/BAM to look for mapped reads and count them. Reads/fragments, which have more than one reported location, will be reported only once.

When counting single end reads, counting reads has the same meaning with counting fragments (the results are identical).

Value

A data frame containing the total number of reads, number of mapped reads and proportion of mapped reads for each library.

Author(s)

Wei Shi and Yang Liao

Examples

```
# build an index using the sample reference sequence provided in the package
# and save it to the current directory
library(Rsubread)
ref <- system.file("extdata","reference.fa",package="Rsubread")
buildindex(basename="./reference_index",reference=ref)

# align the sample read data provided in this packge to the sample reference
# and save the mapping results to the current directory
reads <- system.file("extdata","reads.txt.gz",package="Rsubread")
align(index="./reference_index",readfile1=reads,output_file="./Rsubread_alignment.BAM")

# get the percentage of successfully mapped reads
propmapped("./Rsubread_alignment.BAM")
```

---

qualityScores — *Extract quality score data in a sequencing read dataset*

Description

Extract quality strings and convert them to Phred scores

Usage

```
qualityScores(filename, input_format="gzFASTQ", offset=33, nreads=10000)
```

Arguments

filename	character string giving the name of an input file containing sequence reads.
input_format	character string specifying format of the input file. gzFASTQ (gzipped FASTQ) by default. Acceptable formats include gzFASTQ, FASTQ, SAM and BAM. Character string is case insensitive.
offset	numeric value giving the offset added to the base-calling Phred scores. Possible values include 33 and 64. By default, 33 is used.
nreads	numeric value giving the number of reads from which quality scores are extracted. 10000 by default.

Details

Quality scores of read bases are represented by ASCII characters in next-gen sequencing data. This function extracts the quality characters from each base in each read and then converts them to Phred scores using the provided offset value (`offset`).

If the total number of reads in a dataset is `n`, then every `n/nreads` read is extracted from the input data.

Value

A data matrix containing Phred scores for read bases. Rows in the matrix are reads and columns are base positions in each read.

Author(s)

Wei Shi, Yang Liao and Zhiyin Dai

Examples

```
library(Rsubread)
reads <- system.file("extdata","reads.txt.gz",package="Rsubread")
x <- qualityScores(filename=reads,offset=64,nreads=1000)
x[1:10,1:10]
```

`removeDupReads` *Remove sequencing reads which are mapped to identical locations*

Description

Remove reads which are mapped to identical locations, using mapping location of the first base of each read.

Usage

```
removeDupReads(SAMfile,threshold=50,outputFile)
```

Arguments

`SAMfile`	a character string giving the name of a SAM format input file.
`threshold`	a numeric value giving the threshold for removing duplicated reads, 50 by default. Reads will be removed if they are found to be duplicated equal to or more than `threshold` times.
`outputFile`	a character string giving the base name of output files.

Details

This function uses the mapping location of first base of each read to find duplicated reads. Reads are removed if they are duplicated more than `threshold` number of times.

Value

A SAM file including the remaining reads after duplicate removal.

Author(s)

Yang Liao and Wei Shi

`repair` *Re-order paired-end reads to place reads from the same pair next to each other*

Description

Fast re-odering of paired-end reads using read names and mapping locations.

Usage

```
repair(inFiles,inFormat="BAM",outFiles=paste(inFiles,"repair",sep="."),
addDummy=TRUE,fullData=TRUE,compress=FALSE,nthreads=8)
```

Arguments

`inFiles`	a character vector giving names of input files. These files are typically location-sorted BAM files.
`inFormat`	a character string specifying format of input files. Supported formats include `BAM` and `SAM`.
`outFiles`	a character string giving names of output files. Re-ordered reads are saved to BAM-format files.
`addDummy`	logical indicating if a dummy read will be added to each singleton read which has a missing pair in the input. `TRUE` by default.
`fullData`	logical indicating if sequences and base-calling quality scores of reads will be included in the output. `TRUE` by default.
`compress`	logical indicating if compression should be turned on when generating BAM output. `FALSE` by default.
`nthreads`	a numeric value giving number of CPU threads. 8 by default.

Details

This function takes as input paired-end BAM or SAM files, re-orders reads to make reads from the same pair be adajacent to each other and then outputs the re-ordered reads into BAM files.

The function makes use of both read names and mapping information of reads (eg. mapping coordinates) to identify reads belonging to the same pair. This makes sure that all paired-end reads are correctly re-ordered, especially those multi-mapping read pairs that include more than one reported alignment in the input.

The BAM files produced by this function are comptible with `featureCounts`, meaning that no read re-ordering will be performed when providing these files to `featureCounts`.

Value

No value is produced but BAM files with re-ordered reads are written to the current working directory.

Author(s)

Wei Shi and Yang Liao

RsubreadUsersGuide *View Rsubread Users Guide*

Description

Users Guide for Rsubread and Subread

Usage

```
RsubreadUsersGuide()
```

Details

The Subread/Rsubread Users Guide provides detailed description to the functions and programs included in the Subread and Rsubread software packages. It also includes case studies for analyzing next-gen sequencing data.

The Subread package is written in C and it can be downloaded from http://subread.sourceforge.net. The Rsubread package provides R wrappers functions for many of the programs included in Subread package.

Value

Character string giving the file location.

Author(s)

Wei Shi

See Also

vignette

sam2bed *Convert a SAM format file to a BED format file*

Description

SAM to BED conversion

Usage

```
sam2bed(samfile,bedfile,readlen)
```

Arguments

samfile	character string giving the name of input file. Input format should be in SAM format.
bedfile	character string giving the name of output file. Output file is in BED format.
readlen	numeric value giving the length of reads included in the input file.

Details

This function converts a SAM format file to a BED format file, which can then be displayed in a genome browser like UCSC genome browser, IGB, IGV.

Value

No value is produced but a BED format file is written to the current working directory. This file contains six columns including chromosomal name, start position, end position, name('.'), mapping quality score and strandness.

Author(s)

Wei Shi

Index

∗Topic **documentation**
RsubreadUsersGuide, 28

align, 2, *10*, *18*
atgcContent, 8

buildindex, *7*, 9

createAnnotationFile, 10

detectionCall, 11, *12*, *24*
detectionCallAnnotation, 12, *24*

exactSNP, 12

featureCounts, *6*, 14, *23*, *27*
findCommonVariants, 21

getInBuiltAnnotation, *19*, *21*, 22

processExons, *12*, 23
propmapped, 24

qualityScores, 25

removeDupReads, 26
repair, 27
RsubreadUsersGuide, 28

sam2bed, 28
subjunc, *10*, *18*
subjunc *(*align*)*, 2

vignette, *28*

write.Rsubread *(*createAnnotationFile*)*, 10