Package 'systemPipeR'

October 12, 2016

Type Package

Title systemPipeR: NGS workflow and report generation environment

Version 1.6.4

Date 2016-08-06

Author Thomas Girke

Maintainer Thomas Girke <thomas.girke@ucr.edu>

biocViews Genetics, Infrastructure, DataImport, Sequencing, RNASeq, RiboSeq, ChIPSeq, MethylSeq, SNP, GeneExpression, Coverage, GeneSetEnrichment, Alignment, QualityControl

Description R package for building and running automated end-to-end analysis workflows for a wide range of next generation sequence (NGS) applications such as RNA-Seq, ChIP-Seq, VAR-Seq and Ribo-Seq. Important features include a uniform workflow interface across different NGS applications, automated report generation, and support for running both R and command-line software, such as NGS aligners or peak/variant callers, on local computers or compute clusters. Efficient handling of complex sample sets and experimental designs is facilitated by a consistently implemented sample annotation infrastructure. Instructions for using systemPipeR are given in the Overview Vignette (HTML). The remaining Vignettes, linked below, are workflow templates for common NGS use cases.

Depends Rsamtools, Biostrings, ShortRead, methods

Imports BiocGenerics, GenomicRanges, GenomicFeatures, SummarizedExperiment, VariantAnnotation, rjson, ggplot2, grid, limma, edgeR, DESeq2, GOstats, GO.db, annotate, pheatmap, BatchJobs

Suggests ape, RUnit, BiocStyle, knitr, rmarkdown, biomaRt, BiocParallel

VignetteBuilder knitr

SystemRequirements systemPipeR can be used to run external command-line software (e.g. short read aligners), but the corresponding tool needs to be installed on a system.

License Artistic-2.0

URL https://github.com/tgirke/systemPipeR

NeedsCompilation no

R topics documented:

alignStats *Alignment statistics*

Description

Generate data frame containing important read alignment statistics such as the total number of reads in the FASTQ files, the number of total alignments, as well as the number of primary alignments in the corresponding BAM files.

Usage

```
alignStats(args)
```

Arguments

args	Object of class SYSargs.

Value

data.frame with alignment statistics.

Author(s)

Thomas Girke

Examples

```
## Construct SYSargs object from param and targets files
param <- system.file("extdata", "tophat.param", package="systemPipeR")
targets <- system.file("extdata", "targets.txt", package="systemPipeR")
args <- systemArgs(sysma=param, mytargets=targets)
args
names(args); modules(args); cores(args); outpaths(args); sysargs(args)

## Not run:
## Execute SYSargs on single machine
runCommandline(args=args)

## Execute SYSargs on multiple machines
qsubargs <- getQsubargs(queue="batch", Nnodes="nodes=1", cores=cores(tophat), memory="mem=10gb", time="walltime=
qsubRun(args=args, qsubargs=qsubargs, Nqsubs=1, package="systemPipeR")
## Alignment stats
read_statsDF <- alignStats(args)
read_statsDF <- cbind(read_statsDF[targets$FileName,], targets)
write.table(read_statsDF, "results/alignStats.xls", row.names=FALSE, quote=FALSE, sep="\t")

## End(Not run)
```

catDB-class *Class* "catDB"

Description

Container for storing mappings of genes to annotation categories such as gene ontologies (GO), pathways or conserved sequence domains. The catmap slot stores a list of data.frames providing the direct assignments of genes to annotation categories (e.g. gene-to-GO mappings); catlist is a list of lists of all direct and indirect associations to the annotation categories (e.g. genes mapped to a pathway); and idconv allows to store a lookup-table for converting identifiers (e.g. array feature ids to gene ids).

Objects from the Class

Objects can be created by calls of the form new("catDB", ...).

Slots

catmap: Object of class "list" list of data.frames

catlist: Object of class "list" list of lists

idconv: Object of class "ANY" list of data.frames

Methods

catlist signature(x = "catDB"): extracts data from catlist slot

catmap signature(x = "catDB"): extracts data from catmap slot

coerce signature(from = "list", to = "catDB"): as(list, "catDB")

idconv signature(x = "catDB"): extracts data from idconv slot

names signature(x = "catDB"): extracts slot names

show signature(object = "catDB"): summary view of catDB objects

Author(s)

Thomas Girke

See Also

makeCATdb, GOHyperGAll, GOHyperGAll_Subset, GOHyperGAll_Simplify, GOCluster_Report, goBarplot

Examples

```
showClass("catDB")
## Not run:
## Obtain annotations from BioMart
listMarts() # To choose BioMart database
m <- useMart("ENSEMBL_MART_PLANT"); listDatasets(m)
m <- useMart("ENSEMBL_MART_PLANT", dataset="athaliana_eg_gene")
listAttributes(m) # Choose data types you want to download
go <- getBM(attributes=c("go_accession", "tair_locus", "go_namespace_1003"), mart=m)
go <- go[go[,3]!="",]; go[,3] <- as.character(go[,3])
write.table(go, "GOannotationsBiomart_mod.txt", quote=FALSE, row.names=FALSE, col.names=FALSE, sep="\t")

## Create catDB instance (takes a while but needs to be done only once)
catdb <- makeCATdb(myfile="GOannotationsBiomart_mod.txt", lib=NULL, org="", colno=c(1,2,3), idconv=NULL)
catdb

## End(Not run)
```

---

catmap *catDB accessor methods*

Description

Methods to access information from catDB object.

Usage

```
catmap(x)
```

Arguments

x	object of class catDB

Value

various outputs

Author(s)

Thomas Girke

Examples

```
## Not run:
## Obtain annotations from BioMart
m <- useMart("ENSEMBL_MART_PLANT"); listDatasets(m)
m <- useMart("ENSEMBL_MART_PLANT", dataset="athaliana_eg_gene")
listAttributes(m) # Choose data types you want to download
go <- getBM(attributes=c("go_accession", "tair_locus", "go_namespace_1003"), mart=m)
```

```
go <- go[go[,3]!="",]; go[,3] <- as.character(go[,3])
write.table(go, "GOannotationsBiomart_mod.txt", quote=FALSE, row.names=FALSE, col.names=FALSE, sep="\t")

## Create catDB instance (takes a while but needs to be done only once)
catdb <- makeCATdb(myfile="GOannotationsBiomart_mod.txt", lib=NULL, org="", colno=c(1,2,3), idconv=NULL)
catdb

## Access methods for catDB
catmap(catdb)$D_MF[1:4,]
catlist(catdb)$L_MF[1:4]
idconv(catdb)


## End(Not run)
```

---

clusterRun *Submit command-line tools to cluster*

Description

Submits non-R command-line software to queueing/scheduling systems of compute clusters using run specifications defined by functions similar to `runCommandline`. `runCluster` can be used with most queueing systems since it is based on utilities from the `BatchJobs` package which supports the use of template files (`*.tmpl`) for defining the run parameters of the different schedulers. The path to the `*.tmpl` file needs to be specified in a conf file provided under the `conffile` argument.

Usage

```
clusterRun(args, FUN=runCommandline, conffile = ".BatchJobs.R", template = "torque.tmpl", Njobs, runi
```

Arguments

`args`	Object of class `SYSargs`.
`FUN`	Accpets functions such as `runCommandline(args, ...)` where the `args` argument is mandatory and needs to be of class `SYSargs`.
`conffile`	Path to conf file (default location `./.BatchJobs.R`). This file contains in its simplest form just one command, such as this line for the Torque scheduler: `cluster.functions <- makeClusterFunctionsTorque("torque.tmpl")`. For more detailed information visit this page: https://code.google.com/p/batchjobs/wiki/DortmundUsage
`template`	The template files for a specific queueing/scheduling systems can be downloaded from here: https://github.com/tudo-r/BatchJobs/blob/master/examples/cfTorque/simple.tmpl
`Njobs`	Interger defining the number of cluster jobs. For instance, if `args` contains 18 command-line jobs and `Njobs=9`, then the function will distribute them accross 9 cluster jobs each running 2 command-line jobs. To increase the number of CPU cores used by each process, one can do this under the corresonding argument of the command-line tool, e.g. `-p` argument for Tophat.
`runid`	Run identifier used for log file to track system call commands. Default is `"01"`.

`resourceList` List for reserving for each cluster job sufficient computing resources including memory, number of nodes, CPU cores, walltime, etc. For more details, one can consult the template file for each queueing/scheduling system.

Value

Object of class `Registry`, as well as files and directories created by the executed command-line tools.

Author(s)

Thomas Girke

References

For more details on `BatchJobs`, please consult the following pages: http://sfb876.tu-dortmund.de/PublicPublicationFiles/bisc https://github.com/tudo-r/BatchJobs http://goo.gl/k3Tu5Y

See Also

`clusterRun` replaces the older functions `getQsubargs` and `qsubRun`.

Examples

```
## Construct SYSargs object from param and targets files
param <- system.file("extdata", "tophat.param", package="systemPipeR")
targets <- system.file("extdata", "targets.txt", package="systemPipeR")
args <- systemArgs(sysma=param, mytargets=targets)
args
names(args); modules(args); cores(args); outpaths(args); sysargs(args)

## Not run:
## Execute SYSargs on single machine
runCommandline(args=args)

## Execute SYSargs on multiple machines of a compute cluster. The following
## example uses the conf and template files for the Torque scheduler. Please
## read the instructions above how to obtain the corresponding files for other schedulers.
file.copy(system.file("extdata", ".BatchJobs.R", package="systemPipeR"), ".")
file.copy(system.file("extdata", "torque.tmpl", package="systemPipeR"), ".")
resources <- list(walltime="00:25:00", nodes=paste0("1:ppn=", cores(args)), memory="2gb")
reg <- clusterRun(args, conffile=".BatchJobs", template="torque.tmpl", Njobs=18, runid="01", resourceList=resour

## Monitor progress of submitted jobs
showStatus(reg)
file.exists(outpaths(args))
sapply(1:length(args), function(x) loadResult(reg, x)) # Works once all jobs have completed successfully.

## Alignment stats
read_statsDF <- alignStats(fqpaths=tophatargs$infile1, bampaths=bampaths, fqgz=TRUE)
read_statsDF <- cbind(read_statsDF[targets$FileName,], targets)
write.table(read_statsDF, "results/alignStats.xls", row.names=FALSE, quote=FALSE, sep="\t")
```

```
## End(Not run)
```

---

countRangeset *Read counting for several range sets*

Description

Convenience function to perform read counting iteratively for serveral range sets, e.g. peak range sets or feature types. Internally, the read counting is performed with the summarizeOverlaps function from the GenomicAlignments package. The resulting count tables are directly saved to files.

Usage

```
countRangeset(bfl, args, format="tabular", ...)
```

Arguments

bfl	BamFileList object containing paths to one or more BAM files.
args	Object of class SYSargs where infile1(args) specifies the paths to the tabular range data files (e.g. peak ranges) used for counting.
format	Format of input range files. Currently, supported are tabular or bed. If tabular is selected then the input range files need to contain the proper column titles to coerce with as(..., "GRanges") to GRanges objects after importing them with read.delim. The latter is the case for the peak files (*peaks.xls) generated by the MACS2 software.
...	Arguments to be passed on to internally used summarizeOverlaps function.

Value

Named character vector containing the paths from outpaths(args) to the resulting count table files.

Author(s)

Thomas Girke

See Also

summarizeOverlaps

Examples

```
## Paths to BAM files
param <- system.file("extdata", "bowtieSE.param", package="systemPipeR")
targets <- system.file("extdata", "targets.txt", package="systemPipeR")
args_bam <- systemArgs(sysma=param, mytargets=targets)
bfl <- BamFileList(outpaths(args_bam), yieldSize=50000, index=character())

## Not run:
## SYSargs with paths to range data and count files
args <- systemArgs(sysma="param/count_rangesets.param", mytargets="targets_macs.txt")

## Iterative read counting
countDFnames <- countRangeset(bfl, args, mode="Union", ignore.strand=TRUE)
writeTargetsout(x=args, file="targets_countDF.txt", overwrite=TRUE)


## End(Not run)
```

`featureCoverage` *Genome read coverage by transcript models*

Description

Computes read coverage along single and multi component features based on genomic alignments. The coverage segments of component features are spliced to continuous ranges, such as exons to transcripts or CDSs to ORFs. The results can be obtained with single nucleotide resolution (e.g. around start and stop codons) or as mean coverage of relative bin sizes, such as 100 bins for each feature. The latter allows comparisons of coverage trends among transcripts of variable length. The results can be obtained for single or many features (e.g. any number of transcritpts) at once. Visualization of the coverage results is facilitated by a downstream `plotfeatureCoverage` function.

Usage

```
featureCoverage(bfl, grl, resizereads = NULL, readlengthrange = NULL, Nbins = 20,
               method = mean, fixedmatrix, resizefeatures, upstream, downstream,
                outfile, overwrite = FALSE)
```

Arguments

`bfl`	Paths to BAM files provided as `BamFileList` object. The name slot of the BAM files will be used for naming samples in the results.
`grl`	Genomic ranges provided as `GRangesList` typically generated form `txdb` instances with operations like: `cdsBy(txdb, "tx")` or `exonsBy(txdb, "tx")`. Single component features will be processed the same way as multi component features.

resizereads — Positive integer defining the length alignments should be resized to prior to the coverage calculation. NULL will omit the resizing step.

readlengthrange — Positive integer of length 2 determining the read length range to use for the coverage calculation. Reads falling outside of the specified length range will be excluded from the coverage calculation. For instance, readlengthrange=c(30:40) will base the coverage calculation on reads between 30 to 40 bps. Assigning NULL will skip this filtering step.

Nbins — Single positive integer defining the number of segments the coverage of each feature should be binned into in order to obtain coverage summaries of constant length, e.g. for plotting purposes.

method — Defines the summary statistics to use for binning. The default is method=mean.

fixedmatrix — If set to TRUE, a coverage matrix with single nucleotide resolution will be returned for any number of transcripts centered around precise anchor points in a genome annotation, such a stop/start codons or transcription start sites. For instance, a matrix with coverage information 20bps upstream and downstream of the stop/start codons can be obtained with fixedmatrix=TRUE, upstream=20, downstream=20 along with a grl instance containing the CDS exon ranges required for this operation, e.g. generated with cdsBy(txdb, "tx").

resizefeatures — Needs to be set to TRUE when fixedmatrix=TRUE. Internally, this will use the systemPipeR:::.resizeFeature function to extend single and multi component features at their most left and most right end coordinates. The corresponding extension values are specified under the upstream and downstream arguments.

upstream — Single positive integer specifying the upstream extension length relative to the orientation of each feature in the genome. More details are given above.

downstream — Single positive integer specifying the downstream extension length relative to the orientation of each feature in the genome. More details are given above.

outfile — Default NULL omits writing of the results to a file. If a file name is specified then the results are written to a tabular file. If bfl contains the paths to several BAM files then the results will be appended to the same file where the first column specifies the sample labels. Redirecting the results to file is particularly useful when processing large files of many sample where computation times can be significant.

overwrite — If set to TRUE any existing file assigned to outfile will be overwritten.

Value

The function allows to return the following four distinct outputs. The settings to return these instances are illustrated below in the example section.

(A) — data.frame containing binned coverage where rows are features and columns coverage bins. The first four columns contain (i) the sample names, (ii) the number of total aligned reads in the corresponding BAM files (useful for normalization), (iii) the feature IDs, (iv) strand of the coverage. All following columns are numeric and contain the actual coverage data for the sense and antisense strand of each feature.

(B) data.frame containing coverage with single nucleotide resolution around anchor points such as start and stop codons. The two matrix components are appended column-wise. To clearly distinguish the two data components, they are separated by a specialty column containing pipe characters. The first four columns are the same as described under (A). The column title for the anchor point is 0. For instance, if the features are CDSs then the first 0 corresponds to the first nucleotide of the start codon and the second 0 to the last nucleotide of the stop codon. Upstream and downstream positions are indicated by negative and positive column numbers, respectively.

(C) data.frame containing combined results of (A) and (B) where the first set of columns contains to the coverage around the start codons, the second one the binned coverage of the CDSs and the third one the coverage around the stop codons separated by the same pipe columns mentioned under (B).

(D) Rle list containing the nucleotide level coverage of each feature

Author(s)

Thomas Girke

See Also

plotfeatureCoverage

Examples

```
## Construct SYSargs object from param and targets files
param <- system.file("extdata", "tophat.param", package="systemPipeR")
targets <- system.file("extdata", "targets.txt", package="systemPipeR")
args <- systemArgs(sysma=param, mytargets=targets)

## Not run:
## Features from sample data of systemPipeRdata package
library(GenomicFeatures)
file <- system.file("extdata/annotation", "tair10.gff", package="systemPipeRdata")
txdb <- makeTxDbFromGFF(file=file, format="gff3", organism="Arabidopsis")

## (A) Generate binned coverage for two BAM files and 4 transcripts
grl <- cdsBy(txdb, "tx", use.names=TRUE)
fcov <- featureCoverage(bfl=BamFileList(outpaths(args)[1:2]), grl=grl[1:4], resizereads=NULL,
                    readlengthrange=NULL, Nbins=20, method=mean, fixedmatrix=FALSE,
                    resizefeatures=TRUE, upstream=20, downstream=20)
plotfeatureCoverage(covMA=fcov, method=mean, scales="fixed", scale_count_val=10^6)

## (B) Coverage matrix upstream and downstream of start/stop codons
fcov <- featureCoverage(bfl=BamFileList(outpaths(args)[1:2]), grl=grl[1:4], resizereads=NULL,
                    readlengthrange=NULL, Nbins=NULL, method=mean, fixedmatrix=TRUE,
                    resizefeatures=TRUE, upstream=20, downstream=20)
plotfeatureCoverage(covMA=fcov, method=mean, scales="fixed", scale_count_val=10^6)

## (C) Combined matrix for both binned and start/stop codon
fcov <- featureCoverage(bfl=BamFileList(outpaths(args)[1:2]), grl=grl[1:4], resizereads=NULL,
```

```
                    readlengthrange=NULL, Nbins=20, method=mean, fixedmatrix=TRUE,
                    resizefeatures=TRUE, upstream=20, downstream=20)
plotfeatureCoverage(covMA=fcov, method=mean, scales="fixed", scale_count_val=10^6)

## (D) Rle coverage objects one for each query feature
fcov <- featureCoverage(bfl=BamFileList(outpaths(args)[1:2]), grl=grl[1:4], resizereads=NULL,
                    readlengthrange=NULL, Nbins=NULL, method=mean, fixedmatrix=FALSE,
                    resizefeatures=TRUE, upstream=20, downstream=20)


## End(Not run)
```

`featuretypeCounts` *Plot read distribution across genomic features*

Description

Counts how many reads in short read alignment files (BAM format) overlap with entire annotation categories. This utility is useful for analyzing the distribution of the read mappings across feature types, e.g. coding versus non-coding genes. By default the read counts are reported for the sense and antisense strand of each feature type separately. To minimize memory consumption, the BAM files are processed in a stream using utilities from the `Rsamtools` and `GenomicAlignment` packages. The counts can be reported for each read length separately or as a single value for reads of any length. Subsequently, the counting results can be plotted with the associated `plotfeaturetypeCounts` function.

Usage

```
featuretypeCounts(bfl, grl, singleEnd = TRUE, readlength = NULL, type = "data.frame")
```

Arguments

`bfl`	`BamFileList` object containing the paths to the target BAM files stored on disk. Note, memory-efficient processing is achieved by streaming through BAM files rather than reading entire files into memory at once. The maximum number of alignments to process in each iteration is determined by the `yieldSize` value passed on to the `BamFileList` function. For details see `?BamFileList`.
`grl`	`GRangesList` object containing in each list component the range set of a feature type. Typically, this object is generated with the `genFeatures` function. For details see the example section below and `?genFeatures`.
`singleEnd`	Specifies whether the targets BAM files contain alignments for single-end (SE) or paired-end read data. `TRUE` is for SE and `FALSE` for PE data.
`readlength`	Integer vector specifying the read length values for which to report counts separately. If `readlength=NULL` the length of the reads will be ignored resulting in a single value for each feature type and strand. Note, for PE data the two reads in a pair may differ in length. In those cases the length of the two reads

is averaged and then assigned to the corresponding length category after rounding the mean length to the closest integer. This is not an ideal solution but a reasonable compromise for the purpose of the summary statistics generated by `featuretypeCounts`.

`type` Determines whether the results are returned as `data.frame` (`type="data.frame"`) or as `list` (`type="list"`). Each list component contains the counting results for one BAM file and is named after the corresponding sample. The `data.frame` result contains this sample assignment information in a separate column.

Value

The results are returned as `data.frame` or `list` of `data.frames`. For details see above under `types` argument. The result `data.frames` contain the following columns in the given order:

`SampleName` Sample names obtained from `BamFileList` object.

`Strand` Sense or antisense strand of read mappings.

`Featuretype` Name of feature type provided by `GRangesList` object. Note, the total number of aligned reads is reported under the special feature type 'N_total_aligned'. This value is useful for scaling/normalization purposes in plots, e.g. counts per million reads.

`Featuretypelength`
Total genomic length of each reduced feature type in bases. This value is useful to normalize the read counts by genomic length units, e.g. in plots.

`Subsequent columns`
Counts for reads of any length or for individual read lengths.

Author(s)

Thomas Girke

See Also

`plotfeaturetypeCounts`, `genFeatures`

Examples

```
## Construct SYSargs object from param and targets files
param <- system.file("extdata", "tophat.param", package="systemPipeR")
targets <- system.file("extdata", "targets.txt", package="systemPipeR")
args <- systemArgs(sysma=param, mytargets=targets)

## Not run:
## Features from sample data of systemPipeRdata package
library(GenomicFeatures)
file <- system.file("extdata/annotation", "tair10.gff", package="systemPipeRdata")
txdb <- makeTxDbFromGFF(file=file, format="gff3", organism="Arabidopsis")
feat <- genFeatures(txdb, featuretype="all", reduce_ranges=TRUE, upstream=1000, downstream=0, verbose=TRUE)

## Generate and plot feature counts for specific read lengths
fc <- featuretypeCounts(bfl=BamFileList(outpaths(args), yieldSize=50000), grl=feat, singleEnd=TRUE, readlength=
```

```
p <- plotfeaturetypeCounts(x=fc, graphicsfile="featureCounts.pdf", graphicsformat="pdf", scales="fixed", anyrea

## Generate and plot feature counts for any read length
fc2 <- featuretypeCounts(bfl=BamFileList(outpaths(args), yieldSize=50000), grl=feat, singleEnd=TRUE, readlength=
p2 <- plotfeaturetypeCounts(x=featureCounts2, graphicsfile="featureCounts2.pdf", graphicsformat="pdf", scales="

## End(Not run)
```

`filterDEGs` *Filter and plot DEG results*

Description

Filters and plots DEG results for a given set of sample comparisons. The gene idenifiers of all (i) Up_or_Down, (ii) Up and (iii) Down regulated genes are stored as separate list components, while the corresponding summary statistics, stored in a fourth list component, is plotted in form of a stacked bar plot.

Usage

```
filterDEGs(degDF, filter, plot = TRUE)
```

Arguments

`degDF`	`data.frame` generated by `run_edgeR`
`filter`	Named vector with filter cutoffs of format `c(Fold=2, FDR=1)` where `Fold` refers to the fold change cutoff (unlogged) and `FDR` to the p-value cutoff.
`plot`	Allows to turn plotting behavior on and off with default set to `TRUE`.

Details

Currently, there is no community standard available how to calculate fold changes (here logFC) of genomic ranges, such as gene or feature ranges, to unambiguously refer to them as features with increased or decreased read abundandce; or in case of gene expression experiments to up or down regulated genes, respectively. To be consistent within `systemPipeR`, the corresponding functions, such as `filterDEGs`, use here the following definition. Genomic ranges with positive logFC values are classified as `up` and those with negative logFC values as `down`. This means if a comparison among two samples `a` and `b` is specified in the corresponding targets file as `a-b` then the feature with a positive logFC has a higher _normalized_ read count value in sample `a` than in sample `b`, and vice versa. To inverse this assignment, users want to change the specification of their chosen sample comparison(s) in the targets file accordingly, e.g. change `a-b` to `b-a`. Alternatively, one can swap the column order of the matrix assigned to the `cmp` argument of the `run_edgeR` or `run_DESeq2` functions. Users should also be aware that for logFC values close to zero (noise range), the direction of the fold change (sign of logFC) can be very sensitive to minor differences in the normalization method, while this assignment is much more robust for more pronounced changes or higher absolute logFC values.

Value

Returns `list` with four components

`UporDown`	List of up or down regulated gene/transcript indentifiers meeting the chosen filter settings for all comparisons defined in data frames `pval` and `log2FC`.
`Up`	Same as above but only for up regulated genes/transcript.
`Down`	Same as above but only for down regulated genes/transcript.

Author(s)

Thomas Girke

See Also

`run_edgeR`

Examples

```
targetspath <- system.file("extdata", "targets.txt", package="systemPipeR")
targets <- read.delim(targetspath, comment="#")
cmp <- readComp(file=targetspath, format="matrix", delim="-")
countfile <- system.file("extdata", "countDFeByg.xls", package="systemPipeR")
countDF <- read.delim(countfile, row.names=1)
edgeDF <- run_edgeR(countDF=countDF, targets=targets, cmp=cmp[[1]], independent=FALSE, mdsplot="")
pval <- edgeDF[, grep("_FDR$", colnames(edgeDF)), drop=FALSE]
fold <- edgeDF[, grep("_logFC$", colnames(edgeDF)), drop=FALSE]
DEG_list <- filterDEGs(degDF=edgeDF, filter=c(Fold=2, FDR=10))
names(DEG_list)
DEG_list$Summary
```

`filterVars` *Filter VCF files*

Description

Convenience function for filtering VCF files based on user definable quality parameters. The function imports each VCF file into R, applies the filtering on an internally generated `VRanges` object and then writes the results to a new VCF file.

Usage

```
filterVars(args, filter, varcaller, organism)
```

Arguments

args	Object of class SYSargs. The paths of the input VCF files are specified under infile1(args) and the paths of the output files under outfile1(args).
filter	Character vector of length one specifying the filter syntax that will be applied to the internally created VRanges object.
varcaller	Character vector of length one specifying the variant caller used for generating the input VCFs. Currently, this argument can be assigned 'gatk', 'bcftools' or 'vartools'.
organism	Character vector specifying the organism name of the reference genome.

Value

Output files in VCF format. Their paths can be obtained with outpaths(args).

Author(s)

Thomas Girke

See Also

variantReport combineVarReports, varSummar

Examples

```
## Alignment with BWA (sequentially on single machine)
param <- system.file("extdata", "bwa.param", package="systemPipeR")
targets <- system.file("extdata", "targets.txt", package="systemPipeR")
args <- systemArgs(sysma=param, mytargets=targets)
sysargs(args)[1]

## Not run:
system("bwa index -a bwtsw ./data/tair10.fasta")
bampaths <- runCommandline(args=args)

## Alignment with BWA (parallelized on compute cluster)
resources <- list(walltime="20:00:00", nodes=paste0("1:ppn=", cores(args)), memory="10gb")
reg <- clusterRun(args, conffile=".BatchJobs.R", template="torque.tmpl", Njobs=18, runid="01",
                  resourceList=resources)

## Variant calling with GATK
## The following creates in the inital step a new targets file
## (targets_bam.txt). The first column of this file gives the paths to
## the BAM files created in the alignment step. The new targets file and the
## parameter file gatk.param are used to create a new SYSargs
## instance for running GATK. Since GATK involves many processing steps, it is
## executed by a bash script gatk_run.sh where the user can specify the
## detailed run parameters. All three files are expected to be located in the
## current working directory. Samples files for gatk.param and
## gatk_run.sh are available in the subdirectory ./inst/extdata/ of the
## source file of the systemPipeR package.
```

```
writeTargetsout(x=args, file="targets_bam.txt")
system("java -jar CreateSequenceDictionary.jar R=./data/tair10.fasta O=./data/tair10.dict")
# system("java -jar /opt/picard/1.81/CreateSequenceDictionary.jar R=./data/tair10.fasta O=./data/tair10.dict")
args <- systemArgs(sysma="gatk.param", mytargets="targets_bam.txt")
resources <- list(walltime="20:00:00", nodes=paste0("1:ppn=", 1), memory="10gb")
reg <- clusterRun(args, conffile=".BatchJobs.R", template="torque.tmpl", Njobs=18, runid="01",
                  resourceList=resources)
writeTargetsout(x=args, file="targets_gatk.txt")

## Variant calling with BCFtools
## The following runs the variant calling with BCFtools. This step requires in
## the current working directory the parameter file sambcf.param and the
## bash script sambcf_run.sh.
args <- systemArgs(sysma="sambcf.param", mytargets="targets_bam.txt")
resources <- list(walltime="20:00:00", nodes=paste0("1:ppn=", 1), memory="10gb")
reg <- clusterRun(args, conffile=".BatchJobs.R", template="torque.tmpl", Njobs=18, runid="01",
                  resourceList=resources)
writeTargetsout(x=args, file="targets_sambcf.txt")

## Filtering of VCF files generated by GATK
args <- systemArgs(sysma="filter_gatk.param", mytargets="targets_gatk.txt")
filter <- "totalDepth(vr) >= 2 & (altDepth(vr) / totalDepth(vr) >= 0.8) & rowSums(softFilterMatrix(vr))==4"
# filter <- "totalDepth(vr) >= 20 & (altDepth(vr) / totalDepth(vr) >= 0.8) & rowSums(softFilterMatrix(vr))==6"
filterVars(args, filter, varcaller="gatk", organism="A. thaliana")
writeTargetsout(x=args, file="targets_gatk_filtered.txt")

## Filtering of VCF files generated by BCFtools
args <- systemArgs(sysma="filter_sambcf.param", mytargets="targets_sambcf.txt")
filter <- "rowSums(vr) >= 2 & (rowSums(vr[,3:4])/rowSums(vr[,1:4]) >= 0.8)"
# filter <- "rowSums(vr) >= 20 & (rowSums(vr[,3:4])/rowSums(vr[,1:4]) >= 0.8)"
filterVars(args, filter, varcaller="bcftools", organism="A. thaliana")
writeTargetsout(x=args, file="targets_sambcf_filtered.txt")

## Annotate filtered variants from GATK
args <- systemArgs(sysma="annotate_vars.param", mytargets="targets_gatk_filtered.txt")
txdb <- loadDb("./data/tair10.sqlite")
fa <- FaFile(systemPipeR::reference(args))
variantReport(args=args, txdb=txdb, fa=fa, organism="A. thaliana")

## Annotate filtered variants from BCFtools
args <- systemArgs(sysma="annotate_vars.param", mytargets="targets_sambcf_filtered.txt")
txdb <- loadDb("./data/tair10.sqlite")
fa <- FaFile(systemPipeR::reference(args))
variantReport(args=args, txdb=txdb, fa=fa, organism="A. thaliana")

## Combine results from GATK
args <- systemArgs(sysma="annotate_vars.param", mytargets="targets_gatk_filtered.txt")
combineDF <- combineVarReports(args, filtercol=c(Consequence="nonsynonymous"))
write.table(combineDF, "./results/combineDF_nonsyn_gatk.xls", quote=FALSE, row.names=FALSE, sep="\t")

## Combine results from BCFtools
args <- systemArgs(sysma="annotate_vars.param", mytargets="targets_sambcf_filtered.txt")
combineDF <- combineVarReports(args, filtercol=c(Consequence="nonsynonymous"))
```

```
write.table(combineDF, "./results/combineDF_nonsyn_sambcf.xls", quote=FALSE, row.names=FALSE, sep="\t")

## Summary for GATK
args <- systemArgs(sysma="annotate_vars.param", mytargets="targets_gatk_filtered.txt")
write.table(varSummary(args), "./results/variantStats_gatk.xls", quote=FALSE, col.names = NA, sep="\t")

## Summary for BCFtools
args <- systemArgs(sysma="annotate_vars.param", mytargets="targets_sambcf_filtered.txt")
write.table(varSummary(args), "./results/variantStats_sambcf.xls", quote=FALSE, col.names = NA, sep="\t")

## Venn diagram of variants
args <- systemArgs(sysma="annotate_vars.param", mytargets="targets_gatk_filtered.txt")
varlist <- sapply(names(outpaths(args))[1:4], function(x) as.character(read.delim(outpaths(args)[x])$VARID))
vennset_gatk <- overLapper(varlist, type="vennsets")
args <- systemArgs(sysma="annotate_vars.param", mytargets="targets_sambcf_filtered.txt")
varlist <- sapply(names(outpaths(args))[1:4], function(x) as.character(read.delim(outpaths(args)[x])$VARID))
vennset_bcf <- overLapper(varlist, type="vennsets")
vennPlot(list(vennset_gatk, vennset_bcf), mymain="", mysub="GATK: red; BCFtools: blue", colmode=2, ccol=c("blue"

## End(Not run)
```

---

genFeatures *Generate feature ranges from TxDb*

Description

Function to generate a variety of feature types from TxDb objects using utilities provided by the GenomicFeatures package. The feature types are organized per gene and can be returned on that level in their non-reduced or reduced form.

Currently, supported features include intergenic, promoter, intron, exon, cds, 5'/3'UTR and different transcript types. The latter contains as many transcript types as available in the tx_type column when extracting transcripts from TxDb objects as follows: transcripts(txdb, c("tx_name", "gene_id", "tx_ty

Usage

```
genFeatures(txdb, featuretype = "all", reduce_ranges, upstream = 1000, downstream = 0, verbose = TRUE)
```

Arguments

txdb	TxDb object
featuretype	Feature types can be specified by assigning a character vector containing any of the following: c("tx_type", "promoter", "intron", "exon", "cds", "fiveUTR", "threeUTR", "intergenic"). The default all is a shorthand to select all supported features.
reduce_ranges	If set to TRUE the feature ranges will be reduced on the gene level. As a result overlapping feature components of the same type and from the same gene will be merged to a single range, e.g. two overlapping exons from the same gene are merged to one. Intergenic ranges are not affected by this setting. Note, all reduced feature types are labeled with the suffix '_red'.

upstream	Defines for promoter features the number of bases upstream from the transcription start site.
downstream	Defines for promoter features the number of bases downstream from the transcription start site.
verbose	verbose=FALSE turns off all print messages.

Value

The results are returned as a GRangesList where each component is a GRanges object containing the range set of each feature type. Intergenic ranges are assigned unique identifiers and recorded in the featuretype_id column of the metadata block. For this the ids of their adjacent genes are concatenated with two underscores as separator. If the adjacent genes overlap with other genes then their identifiers are included in the id string as well and separated by a single underscore.

Author(s)

Thomas Girke

See Also

transcripts and associated TxDb accessor functions from the GenomicFeatures package.

Examples

```
## Sample from GenomicFeatures package
library(GenomicFeatures)
gffFile <- system.file("extdata", "GFF3_files", "a.gff3", package="GenomicFeatures")
txdb <- makeTxDbFromGFF(file=gffFile, format="gff3", organism="Solanum lycopersicum")
feat <- genFeatures(txdb, featuretype="all", reduce_ranges=FALSE, upstream=1000, downstream=0)

## List extracted feature types
names(feat)

## Obtain feature lists by genes, here for promoter
split(feat$promoter, unlist(mcols(feat$promoter)$feature_by))

## Return all features in single GRanges object
unlist(feat)

## Not run:
## Sample from systemPipeRdata package
file <- system.file("extdata/annotation", "tair10.gff", package="systemPipeRdata")
txdb <- makeTxDbFromGFF(file=file, format="gff3", organism="Arabidopsis")
feat <- genFeatures(txdb, featuretype="all", reduce_ranges=FALSE, upstream=1000, downstream=0)

## End(Not run)
```

getQsubargs *Arguments for qsub*

Description

Note: This function as been deprecated. Please use clusterRun instead. getQsubargs defines arguments to submit runX job(s) to queuing system (e.g. Torque) via qsub.

Usage

```
getQsubargs(software = "qsub", queue = "batch", Nnodes = "nodes=1", cores = as.numeric(gsub("^.* ", ""
```

Arguments

software	Software to use for submission to queuing system. Default is qsub.
queue	Name of queue to to use. Default is batch.
Nnodes	Number of compute nodes to use for processing. Default is nodes=1.
cores	Number of CPU cores to use per compute node. Default will use what is provided by under -p in myargs of systemArgs() output.
memory	Amount of RAM to reserve per node.
time	Walltime limit each job is allowed to run per node.

Value

list

Author(s)

Thomas Girke

Examples

```
## Construct SYSargs object from param and targets files
param <- system.file("extdata", "tophat.param", package="systemPipeR")
targets <- system.file("extdata", "targets.txt", package="systemPipeR")
args <- systemArgs(sysma=param, mytargets=targets)
args
names(args); modules(args); cores(args); outpaths(args); sysargs(args)

## Not run:
## Execute SYSargs on single machine
runCommandline(args=args)

## Execute SYSargs on multiple machines
qsubargs <- getQsubargs(queue="batch", Nnodes="nodes=1", cores=cores(tophat), memory="mem=10gb", time="walltime=
qsubRun(args=args, qsubargs=qsubargs, Nqsubs=1, package="systemPipeR")
## Alignment stats
```

```
read_statsDF <- alignStats(fqpaths=tophatargs$infile1, bampaths=bampaths, fqgz=TRUE)
read_statsDF <- cbind(read_statsDF[targets$FileName,], targets)
write.table(read_statsDF, "results/alignStats.xls", row.names=FALSE, quote=FALSE, sep="\t")

## End(Not run)
```

GOHyperGAll *GO term enrichment analysis for large numbers of gene sets*

Description

To test a sample population of genes for over-representation of GO terms, the core function GOHyperGAll computes for all nodes in the three GO networks (BP, CC and MF) an enrichment test based on the hypergeometric distribution and returns the corresponding raw and Bonferroni corrected p-values. Subsequently, a filter function supports GO Slim analyses using default or custom GO Slim categories. Several convenience functions are provided to process large numbers of gene sets (e.g. clusters from partitioning results) and to visualize the results.

Note: GOHyperGAll provides similar utilities as the GOHyperG function in the GOstats package. The main difference is that GOHyperGAll simplifies processing of large numbers of gene sets, as well as the usage of custom array-to-gene and gene-to-GO mappings.

Usage

```
## Generate gene-to-GO mappings and store as catDB object
makeCATdb(myfile, lib = NULL, org = "", colno = c(1, 2, 3), idconv = NULL, rootUK=FALSE)

## Enrichment function
GOHyperGAll(catdb, gocat = "MF", sample, Nannot = 2)

## GO slim analysis
GOHyperGAll_Subset(catdb, GOHyperGAll_result, sample = test_sample, type = "goSlim", myslimv)

## Reduce GO term redundancy
GOHyperGAll_Simplify(GOHyperGAll_result, gocat = "MF", cutoff = 0.001, correct = TRUE)

## Batch analysis of many gene sets
GOCluster_Report(catdb, setlist, id_type = "affy", method = "all", CLSZ = 10, cutoff = 0.001, gocats =

## Bar plot of GOCluster_Report results
goBarplot(GOBatchResult, gocat)
```

Arguments

myfile	File with gene-to-GO mappings. Sample files can be downloaded from geneontology.org (http://geneontology.org/GO.downloads.annotations.shtml) or from BioMart as shown in example below.

colno — Column numbers referencing in myfile the three target columns containing GOID, GeneID and GOCAT, in that order.

org — Optional argument. Currently, the only valid option is org="Arabidopsis" to get rid of transcript duplications in this particular annotation.

lib — If the gene-to-GO mappings are obtained from a *.db package from Bioconductor then the package name can be specified under the lib argument of the sampleDFgene2GO function.

idconv — Optional id conversion data.frame

catdb — catdb object storing mappings of genes to annotation categories. For details, see ?"SYSargs-class".

rootUK — If the argument rootUK is set to TRUE then the root nodes are treated as terminal nodes to account for the new unknown terms.

sample — character vector containing the test set of gene identifiers

Nannot — Defines the minimum number of direct annotations per GO node from the sample set to determine the number of tested hypotheses for the p-value adjustment.

gocat — Specifies the GO type, can be assigned one of the following character values: "MF", "BP" and "CC".

GOHyperGAll_result — data.frame generated by GOHyperGAll

type — The function GOHyperGAll_Subset subsets the GOHyperGAll results by directly assigned GO nodes or custom goSlim categories. The argument type can be assigned the values goSlim or assigned.

myslimv — optional argument to provide custom goSlim vector

cutoff — p-value cutoff for GO terms to show in result data.frame

correct — If TRUE the function will favor the selection of terminal (informationich) GO terms that have at the same time a large number of sample matches.

setlist — list of character vectors containing gene IDs (or array feature IDs). The names of the list components correspond to the set labels, e.g. DEG comparisons or cluster IDs.

id_type — specifies type of IDs in input, can be assigned gene or affy

method — Specifies analysis type. Current options are all for GOHyperGAll, slim for GOHyperGAll_Subset or simplify for GOHyperGAll_Simplify.

CLSZ — minimum gene set (cluster) size to consider. Gene sets below this cutoff will be ignored.

gocats — Specifies GO type, can be assigned the values "MF", "BP" and "CC".

recordSpecGO — argument to report in the result data.frame specific GO IDs for any of the 3 ontologies disregarding whether they meet the specified p-value cutoff, e.g: recordSpecGO=c("GO:0003674", "GO:0008150", "GO:0005575")

GOBatchResult — data.frame generated by GOCluster_Report

... — additional arguments to pass on

Details

GOHyperGAll_Simplify: The result data frame from GOHyperGAll will often contain several connected GO terms with significant scores which can complicate the interpretation of large sample sets. To reduce this redundancy, the function GOHyperGAll_Simplify subsets the data frame by a user specified p-value cutoff and removes from it all GO nodes with overlapping children sets (OFFSPRING), while the best scoring nodes are retained in the result data.frame.

GOCluster_Report: performs the three types of GO term enrichment analyses in batch mode: GOHyperGAll, GOHyperGAll_Subset or GOHyperGAll_Simplify. It processes many gene sets (e.g. gene expression clusters) and returns the results conveniently organized in a single result data frame.

Value

makeCATdb generates catDB object from file.

Author(s)

Thomas Girke

References

This workflow has been published in Plant Physiol (2008) 147, 41-57.

See Also

GOHyperGAll_Subset, GOHyperGAll_Simplify, GOCluster_Report, goBarplot

Examples

```
## Not run:

## Obtain annotations from BioMart
listMarts() # To choose BioMart database
m <- useMart("ENSEMBL_MART_PLANT"); listDatasets(m)
m <- useMart("ENSEMBL_MART_PLANT", dataset="athaliana_eg_gene")
listAttributes(m) # Choose data types you want to download
go <- getBM(attributes=c("go_accession", "tair_locus", "go_namespace_1003"), mart=m)
go <- go[go[,3]!="",]; go[,3] <- as.character(go[,3])
write.table(go, "GOannotationsBiomart_mod.txt", quote=FALSE, row.names=FALSE, col.names=FALSE, sep="\t")

## Create catDB instance (takes a while but needs to be done only once)
catdb <- makeCATdb(myfile="GOannotationsBiomart_mod.txt", lib=NULL, org="", colno=c(1,2,3), idconv=NULL)
catdb

## Create catDB from Bioconductor annotation package
# catdb <- makeCATdb(myfile=NULL, lib="ath1121501.db", org="", colno=c(1,2,3), idconv=NULL)

## AffyID-to-GeneID mappings when working with AffyIDs
# affy2locusDF <- systemPipeR:::.AffyID2GeneID(map = "ftp://ftp.arabidopsis.org/home/tair/Microarrays/Affymetri
# catdb_conv <- makeCATdb(myfile="GOannotationsBiomart_mod.txt", lib=NULL, org="", colno=c(1,2,3), idconv=list(a
# systemPipeR:::.AffyID2GeneID(catdb=catdb_conv, affyIDs=c("244901_at", "244902_at"))
```

```
## Next time catDB can be loaded from file
save(catdb, file="catdb.RData")
load("catdb.RData")

## Perform enrichment test on single gene set
test_sample <- unique(as.character(catmap(catdb)$D_MF[1:100,"GeneID"]))
GOHyperGAll(catdb=catdb, gocat="MF", sample=test_sample, Nannot=2)[1:20,]

## GO Slim analysis by subsetting results accordingly
GOHyperGAll_result <- GOHyperGAll(catdb=catdb, gocat="MF", sample=test_sample, Nannot=2)
GOHyperGAll_Subset(catdb, GOHyperGAll_result, sample=test_sample, type="goSlim")

## Reduce GO term redundancy in 'GOHyperGAll_results'
simplifyDF <- GOHyperGAll_Simplify(GOHyperGAll_result, gocat="MF", cutoff=0.001, correct=T)
# Returns the redundancy reduced data set.
data.frame(GOHyperGAll_result[GOHyperGAll_result[,1]

## Batch Analysis of Gene Clusters
testlist <- list(Set1=test_sample)
GOBatchResult <- GOCluster_Report(catdb=catdb, setlist=testlist, method="all", id_type="gene", CLSZ=10, cutoff=0

## Plot 'GOBatchResult' as bar plot
goBarplot(GOBatchResult, gocat="MF")


## End(Not run)
```

INTERSECTset-class *Class* "INTERSECTset"

Description

Container for storing standard intersect results created by the overLapper function. The setlist slot stores the original label sets as vectors in a list; intersectmatrix organizes the label sets in a present-absent matrix; complexitylevels represents the number of comparisons considered for each comparison set as vector of integers; and intersectlist contains the standard intersect vectors.

Objects from the Class

Objects can be created by calls of the form new("INTERSECTset", ...).

Slots

setlist: Object of class "list": list of vectors

intersectmatrix: Object of class "matrix": binary matrix

complexitylevels: Object of class "integer": vector of integers

intersectlist: Object of class "list": list of vectors

Methods

as.list `signature(x = "INTERSECTset")`: coerces `INTERSECTset` to `list`

coerce `signature(from = "list", to = "INTERSECTset")`: `as(list, "INTERSECTset")`

complexitylevels `signature(x = "INTERSECTset")`: extracts data from `complexitylevels` slot

intersectlist `signature(x = "INTERSECTset")`: extracts data from `intersectlist` slot

intersectmatrix `signature(x = "INTERSECTset")`: extracts data from `intersectmatrix` slot

length `signature(x = "INTERSECTset")`: returns number of original label sets

names `signature(x = "INTERSECTset")`: extracts slot names

setlist `signature(x = "INTERSECTset")`: extracts data from `setlist` slot

show `signature(object = "INTERSECTset")`: summary view of `INTERSECTset` objects

Author(s)

Thomas Girke

See Also

`overLapper`, `vennPlot`, `olBarplot`, `VENNset-class`

Examples

```
showClass("INTERSECTset")

## Sample data
setlist <- list(A=sample(letters, 18), B=sample(letters, 16),
                C=sample(letters, 20), D=sample(letters, 22),
                E=sample(letters, 18), F=sample(letters, 22))

## Create VENNset
interset <- overLapper(setlist[1:5], type="intersects")
class(interset)

## Accessor methods for VENNset/INTERSECTset objects
names(interset)
setlist(interset)
intersectmatrix(interset)
complexitylevels(interset)
intersectlist(interset)

## Coerce VENNset/INTERSECTset object to list
as.list(interset)
```

mergeBamByFactor *Merge BAM files based on factor*

Description

Merges BAM files based on sample groupings provided by a factor using internally the mergeBam function from the Rsamtools package. The function also returns an updated SYSargs object containing the paths to the merged BAM files as well as to the unmerged BAM files if there are any. All rows of merged parent samples are removed.

The functionality provided by mergeBamByFactor is useful for experiments where pooling of replicates is advantageous to maximize the depth of read coverage, such as prior to peak calling in ChIP-Seq or miRNA gene prediction experiments.

Usage

```
mergeBamByFactor(args, mergefactor = targetsin(args)$Factor, overwrite = FALSE, silent = FALSE, ...)
```

Arguments

args	An instance of SYSargs constructed from a targets file where the first column (targetsin(args)) contains the paths to the BAM files along with the column title FileName.
mergefactor	factor containing the grouping information required for merging the BAM files referenced in the first column of targetsin(args). The default uses targetsin(args)$Factor as factor. The latter merges BAM files for which replicates are specified in the Factor column.
overwrite	If overwrite=FALSE existing BAM files of the same name will not be overwritten.
silent	If silent=TRUE print statements will be suppressed.
...	To pass on additional arguments to the internally used mergeBam function from Rsamtools.

Value

The merged BAM files will be written to output files with the following naming convention: <first_BAM_file_name>_<group
In addition, the function returns an updated SYSargs object where all output file paths contain the paths to the merged BAM files. The rows of the merged parent samples are removed and the rows of the unmerged samples remain unchanged.

Author(s)

Thomas Girke

See Also

writeTargetsout, writeTargetsRef

Examples

```
## Construct initial SYSargs object
targetspath <- system.file("extdata", "targets_chip.txt", package="systemPipeR")
parampath <- system.file("extdata", "bowtieSE.param", package="systemPipeR")
args <- systemArgs(sysma=parampath, mytargets=targetspath)

## Not run:
## After running alignmets (e.g. with Bowtie2) generate targets file
## for the corresponding BAM files. The alignment step is skipped here.
writeTargetsout(x=args, file="targets_bam.txt", overwrite=TRUE)
args <- systemArgs(sysma=NULL, mytargets="targets_bam.txt")

## Merge BAM files and return updated SYSargs object
args_merge <- mergeBamByFactor(args, overwrite=TRUE, silent=FALSE)

## Export modified targets file
writeTargetsout(x=args_merge, file="targets_mergeBamByFactor.txt", overwrite=TRUE)

## End(Not run)
```

---

moduleload *Interface to module system*

Description

Functions to list and load software from a module system in R. The functions are the equivalent of `module avail` and `module load` on the Linux command-line, respectively.

Usage

```
moduleload(module)

modulelist()
```

Arguments

`module`	Name of software to load character vector.

Author(s)

Tyler Backman and Thomas Girke

Examples

```
## Not run:
## List all software from module system
moduleload()
## Example for loading Bowtie 2
```

```
modulelist("bowtie2/2.0.6")

## End(Not run)
```

---

olBarplot *Bar plot for intersect sets*

Description

Generates bar plots of the intersect counts of VENNset and INTERSECTset objects generated by the overLapper function. It is an alternative to Venn diagrames (e.g. vennPlot) that scales to larger numbers of label sets. By default the bars in the plot are colored and grouped by complexity levels of the intersect sets.

Usage

```
olBarplot(x, mincount = 0, complexity="default", myxlabel = "default", myylabel="Counts", mytitle = "
```

Arguments

x	Object of class VENNset or INTERSECTset.
mincount	Sets minimum number of counts to consider in the bar plot. Default mincount=0 considers all counts.
complexity	Allows user to limit the bar plot to specific complexity levels of intersects by specifying the chosen ones with an integer vector. Default complexity="default" considers all complexity levels.
myxlabel	Defines label of x-axis.
myylabel	Defines label of y-axis.
mytitle	Defines main title of plot.
...	Allows to pass on additional arguments to geom_bar from ggplot2. For instance, fill=seq(along=vennlist(x)) or fill=seq(along=intersectlist(x)) will assign a different color to each bar, or fill="blue" will color all of them blue. The default bar coloring is by complexity levels of the intersect sets.

Value

Bar plot.

Note

The functions provided here are an extension of the Venn diagram resources on this site: http://manuals.bioinformatics.ucr.edu
Venn-Diagrams

Author(s)

Thomas Girke

See Also

overLapper, vennPlot

Examples

```
## Sample data: list of vectors with object labels
setlist <- list(A=sample(letters, 18), B=sample(letters, 16),
                C=sample(letters, 20), D=sample(letters, 22),
                E=sample(letters, 18), F=sample(letters, 22))

## 2-way Venn diagram
vennset <- overLapper(setlist[1:2], type="vennsets")
vennPlot(vennset)

## 3-way Venn diagram
vennset <- overLapper(setlist[1:3], type="vennsets")
vennPlot(vennset)

## 4-way Venn diagram
vennset <- overLapper(setlist[1:4], type="vennsets")
vennPlot(list(vennset, vennset))

## Pseudo 4-way Venn diagram with circles
vennPlot(vennset, type="circle")

## 5-way Venn diagram
vennset <- overLapper(setlist[1:5], type="vennsets")
vennPlot(vennset)

## Alternative Venn count input to vennPlot (not recommended!)
counts <- sapply(vennlist(vennset), length)
vennPlot(counts)

## 6-way Venn comparison as bar plot
vennset <- overLapper(setlist[1:6], type="vennsets")
olBarplot(vennset, mincount=1)

## Bar plot of standard intersect counts
interset <- overLapper(setlist, type="intersects")
olBarplot(interset, mincount=1)

## Accessor methods for VENNset/INTERSECTset objects
names(vennset)
names(interset)
setlist(vennset)
intersectmatrix(vennset)
complexitylevels(vennset)
vennlist(vennset)
intersectlist(interset)

## Coerce VENNset/INTERSECTset object to list
as.list(vennset)
```

```
as.list(interset)

## Pairwise intersect matrix and heatmap
olMA <- sapply(names(setlist),
function(x) sapply(names(setlist),
function(y) sum(setlist[[x]] %in% setlist[[y]])))
olMA
heatmap(olMA, Rowv=NA, Colv=NA)

## Presence-absence matrices for large numbers of sample sets
interset <- overLapper(setlist=setlist, type="intersects", complexity=2)
(paMA <- intersectmatrix(interset))
heatmap(paMA, Rowv=NA, Colv=NA, col=c("white", "gray"))
```

---

overLapper *Set Intersect and Venn Diagram Functions*

Description

Function for computing Venn intersects or standard intersects among large numbers of label sets provided as `list` of `vectors`. The resulting intersect objects can be used for plotting 2-5 way Venn diagrams or intersect bar plots using the functions `vennPlot` or `olBarplot`, respectively. The `overLapper` function scales to 2-20 or more label vectors for Venn intersect calculations and to much larger sample numbers for standard intersects. The different intersect types are explained below under the definition of the `type` argument. The upper Venn limit around 20 label sets is unavoidable because the complexity of Venn intersects increases exponentially with the label set number `n` according to this relationship: `2^n - 1`. The current implementation of the plotting function `vennPlot` supports Venn diagrams for 2-5 label sets. To visually analyze larger numbers of label sets, a variety of intersect methods are introduced in the `olBarplot` help file. These methods are much more scalable than Venn diagrams, but lack their restrictive intersect logic.

Usage

```
overLapper(setlist, complexity = "default", sep = "_", cleanup = FALSE, keepdups = FALSE, type)
```

Arguments

`setlist`	Object of class `list` where each list component stores a label set as `vector` and the name of each label set is stored in the name slot of each list component. The names are used for naming the label sets in all downstream analysis steps and plots.
`complexity`	Complexity level of intersects specified as integer vector. For Venn intersects it needs to be assigned `1:length(setlist)` (default). If `complexity=2` the function returns all pairwise intersects.
`sep`	Character used to separate set labels.
`cleanup`	If set to `TRUE` then all characters of the label sets are set to upper case, and leading and trailing spaces are removed. The default `cleanup=FALSE` omits this step.

keepdups — By default all duplicates are removed from the label sets. The setting keepdups=TRUE will retain duplicates by appending a counter to each entry.

type — With the default setting type="vennsets" the overLapper function computes the typical Venn intersects for the label sets provided under setlist. With the setting type="intersects" the function will compute pairwise intersects (not compatible with Venn diagrams). Venn intersects follow the typical 'only in' intersect logic of Venn comparisons, such as: labels present only in set A, labels present only in the intersect of A & B, etc. Due to this restrictive intersect logic, the combined Venn sets contain no duplicates. In contrast to this, regular intersects follow this logic: labels present in the intersect of A & B, labels present in the intersect of A & B & C, etc. This approach results usually in many duplications of labels among the intersect sets.

Details

Additional Venn diagram resources are provided by the packages limma, gplots, vennerable, eVenn and VennDiagram, or online resources such as shapes, Venn Diagram Generator and Venny.

Value

overLapper returns standard intersect and Venn intersect results as INTERSECTset or VENNset objects, respectively. These S4 objects contain the following components:

setlist — Original label sets accessible with setlist().

intersectmatrix — Present-absent matrix accessible with intersectmatrix(), where each overlap set in the vennlist data component is labeled according to the label set names provided under setlist. For instance, the composite name 'ABC' indicates that the entries are restricted to A, B and C. The seperator used for naming the intersect sets can be specified under the sep argument.

complexitylevels — Complexity levels accessible with complexitylevels().

vennlist — Venn intersects for VENNset objects accessible with vennlist().

intersectlist — Standard intersects for INTERSECTset objects accessible with intersectlist().

Note

The functions provided here are an extension of the Venn diagram resources on this site: http://manuals.bioinformatics.ucr.edu
Venn-Diagrams

Author(s)

Thomas Girke

References

See examples in 'The Electronic Journal of Combinatorics': http://www.combinatorics.org/files/Surveys/ds5/VennSymmExa

See Also

vennPlot, olBarplot

Examples

```
## Sample data
setlist <- list(A=sample(letters, 18), B=sample(letters, 16),
                C=sample(letters, 20), D=sample(letters, 22),
                E=sample(letters, 18), F=sample(letters, 22))

## 2-way Venn diagram
vennset <- overLapper(setlist[1:2], type="vennsets")
vennPlot(vennset)

## 3-way Venn diagram
vennset <- overLapper(setlist[1:3], type="vennsets")
vennPlot(vennset)

## 4-way Venn diagram
vennset <- overLapper(setlist[1:4], type="vennsets")
vennPlot(list(vennset, vennset))

## Pseudo 4-way Venn diagram with circles
vennPlot(vennset, type="circle")

## 5-way Venn diagram
vennset <- overLapper(setlist[1:5], type="vennsets")
vennPlot(vennset)

## Alternative Venn count input to vennPlot (not recommended!)
counts <- sapply(vennlist(vennset), length)
vennPlot(counts)

## 6-way Venn comparison as bar plot
vennset <- overLapper(setlist[1:6], type="vennsets")
olBarplot(vennset, mincount=1)

## Bar plot of standard intersect counts
interset <- overLapper(setlist, type="intersects")
olBarplot(interset, mincount=1)

## Accessor methods for VENNset/INTERSECTset objects
names(vennset)
names(interset)
setlist(vennset)
intersectmatrix(vennset)
complexitylevels(vennset)
vennlist(vennset)
intersectlist(interset)

## Coerce VENNset/INTERSECTset object to list
as.list(vennset)
```

```
as.list(interset)

## Pairwise intersect matrix and heatmap
olMA <- sapply(names(setlist),
function(x) sapply(names(setlist),
function(y) sum(setlist[[x]] %in% setlist[[y]])))
olMA
heatmap(olMA, Rowv=NA, Colv=NA)

## Presence-absence matrices for large numbers of sample sets
interset <- overLapper(setlist=setlist, type="intersects", complexity=2)
(paMA <- intersectmatrix(interset))
heatmap(paMA, Rowv=NA, Colv=NA, col=c("white", "gray"))
```

`plotfeatureCoverage` *Plot feature coverage results*

Description

Plots the 3 tabular data types (A-C) generated by the `featureCoverage` function. It accepts data from single or many features (e.g. CDSs) and samples (BAM files). The coverage from multiple features will be summarized using methods such as `mean`, while the data from multiple samples will be plotted in separate panels.

Usage

```
plotfeatureCoverage(covMA, method = mean, scales = "fixed", extendylim=2, scale_count_val = 10^6)
```

Arguments

`covMA`	Object of class `data.frame` generated by `featureCoverage` function.
`method`	Defines the summary statistics to use when `covMA` contains coverage data from multiple features (e.g. transcripts). The default calculates the mean coverage for each positon and/or bin of the corresponding coverage vectors.
`scales`	Scales setting passed on to the `facet_wrap` function of `ggplot2`. For details see `ggplot2::facet_wrap`. The default `fixed` assures a constant scale across all bar plot panels, while `free` uses the optimum scale within each bar plot panel. To evaluate plots in all their details, it may be necessary to generate two graphics files one for each scaling option.
`extendylim`	Allows to extend the upper limit of the y axis when `scales=fixed`. Internally, the function identifies the maximum value in the data and then multiplies this maximum value by the value provided under `extendylim`. The default is set to `extendylim=2`.
`scale_count_val`	Scales (normalizes) the read counts to a fixed value of aligned reads in each sample such as counts per million aligned reads (default is 10^6). For this calculation the `N_total_aligned` values are used that are reported in the input `data.frame` generated by the upstream `featureCoverage` function. Assign `NULL` to turn off scaling.

Value

Currently, the function returns ggplot2 bar plot graphics.

Author(s)

Thomas Girke

See Also

featureCoverage

Examples

```
## Construct SYSargs object from param and targets files
param <- system.file("extdata", "tophat.param", package="systemPipeR")
targets <- system.file("extdata", "targets.txt", package="systemPipeR")
args <- systemArgs(sysma=param, mytargets=targets)

## Not run:
## Features from sample data of systemPipeRdata package
library(GenomicFeatures)
file <- system.file("extdata/annotation", "tair10.gff", package="systemPipeRdata")
txdb <- makeTxDbFromGFF(file=file, format="gff3", organism="Arabidopsis")

## (A) Generate binned coverage for two BAM files and 4 transcripts
grl <- cdsBy(txdb, "tx", use.names=TRUE)
fcov <- featureCoverage(bfl=BamFileList(outpaths(args)[1:2]), grl=grl[1:4], resizereads=NULL,
                    readlengthrange=NULL, Nbins=20, method=mean, fixedmatrix=FALSE,
                    resizefeatures=TRUE, upstream=20, downstream=20)
fcov <- featureCoverage(bfl=BamFileList(outpaths(args)[1:2]), grl=grl[1:4], resizereads=NULL,
                    readlengthrange=NULL, Nbins=20, method=mean, fixedmatrix=TRUE,
                    resizefeatures=TRUE, upstream=20, downstream=20)
plotfeatureCoverage(covMA=fcov, method=mean, scales="fixed", scale_count_val=10^6)

## (B) Coverage matrix upstream and downstream of start/stop codons
fcov <- featureCoverage(bfl=BamFileList(outpaths(args)[1:2]), grl=grl[1:4], resizereads=NULL,
                    readlengthrange=NULL, Nbins=NULL, method=mean, fixedmatrix=TRUE,
                    resizefeatures=TRUE, upstream=20, downstream=20)
plotfeatureCoverage(covMA=fcov, method=mean, scales="fixed", scale_count_val=10^6)

## (C) Combined matrix for both binned and start/stop codon
fcov <- featureCoverage(bfl=BamFileList(outpaths(args)[1:2]), grl=grl[1:4], resizereads=NULL,
                    readlengthrange=NULL, Nbins=20, method=mean, fixedmatrix=TRUE,
                 resizefeatures=TRUE, upstream=20, downstream=20, outfile="results/test.xls")
plotfeatureCoverage(covMA=fcov, method=mean, scales="fixed", scale_count_val=10^6)

## (D) Rle coverage objects one for each query feature
fcov <- featureCoverage(bfl=BamFileList(outpaths(args)[1:2]), grl=grl[1:4], resizereads=NULL,
                    readlengthrange=NULL, Nbins=NULL, method=mean, fixedmatrix=FALSE,
                    resizefeatures=TRUE, upstream=20, downstream=20)
```

```
## End(Not run)
```

---

plotfeaturetypeCounts *Plot read distribution across genomic features*

Description

Function to visualize the distribution of reads across different feature types for many alignment files in parallel. The plots are stacked bar plots representing the raw or normalized read counts for the sense and antisense strand of each feature. The graphics results are generated with ggplot2. Typically, the expected input is generated with the affiliated featuretypeCounts function.

Usage

```
plotfeaturetypeCounts(x, graphicsfile, graphicsformat = "pdf", scales = "fixed", anyreadlength = FALS
                  drop_N_total_aligned = TRUE, scale_count_val = 10^6, scale_length_val = NULL)
```

Arguments

x	data.frame with feature counts generated by the featuretypeCounts function.
graphicsfile	Path to file where to write the output graphics. Note, the function returns the graphics instructions from ggplot2 for interactive plotting in R. However, due to the complexity of the graphics generated here, the finished results are written to a file directly.
graphicsformat	Graphics file format. Currently, supported formats are: pdf, png or jpeg. Argument accepts one of them as character string.
scales	Scales setting passed on to the facet_wrap function of ggplot2. For details see ggplot2::facet_wrap. The default fixed assures a constant scale across all bar plot panels, while free uses the optimum scale within each bar plot panel. To evaluate plots in all their details, it may be necessary to generate two graphics files one for each scaling option.
anyreadlength	If set to TRUE read length specific read counts will be summed up to a single count value to plot read counts for any read length. Otherwise the bar plots will show the counts for each read length value.
drop_N_total_aligned	If set to TRUE the special feature count N_total_aligned will not be included as a separate feature in the plots. However, the information will still be used internally for scaling the read counts to a fixed value if this option is requested under the scale_count_val argument.
scale_count_val	Scales (normalizes) the read counts to a fixed value of aligned reads in each sample such as counts per million aligned reads (default is 10^6). For this calculation the N_total_aligned values are used that are reported in the input data.frame generated by the upstream featuretypeCounds function. Assign NULL to turn off scaling by aligned reads.

scale_length_val

Allows to adjust the raw or scaled read counts to a constant length interval (e.g. scale_length_val=10^3 in bps) considering the total genomic length of the corresponding feature type. The required genomic length information for each feature type is obtained from the Featuretypelength column of the input data.frame generated by the featuretypeCount function. To turn off feature length adjustment, assign NULL (default).

Value

The function returns bar plot graphics for aligned read counts with read length resolution if the input contains this information and argument anyreadlength is set to FALSE. If the input contains counts for any read length and/or anyreadlength=TRUE then there will be only one bar per feature and sample. Due to the complexity of the plots, the results are directly written to file in the chosen graphics format. However, the function also returns the plotting instructions returned by ggplot2 to display the result components using R's plotting device.

Author(s)

Thomas Girke

See Also

featuretypeCounts, genFeatures

Examples

```
## Construct SYSargs object from param and targets files
param <- system.file("extdata", "tophat.param", package="systemPipeR")
targets <- system.file("extdata", "targets.txt", package="systemPipeR")
args <- systemArgs(sysma=param, mytargets=targets)

## Not run:
## Features from sample data of systemPipeRdata package
library(GenomicFeatures)
file <- system.file("extdata/annotation", "tair10.gff", package="systemPipeRdata")
txdb <- makeTxDbFromGFF(file=file, format="gff3", organism="Arabidopsis")
feat <- genFeatures(txdb, featuretype="all", reduce_ranges=TRUE, upstream=1000, downstream=0, verbose=TRUE)

## Generate and plot feature counts for specific read lengths
fc <- featuretypeCounts(bfl=BamFileList(outpaths(args), yieldSize=50000), grl=feat, singleEnd=TRUE, readlength=c
p <- plotfeaturetypeCounts(x=fc, graphicsfile="featureCounts.pdf", graphicsformat="pdf", scales="fixed", anyreac

## Generate and plot feature counts for any read length
fc2 <- featuretypeCounts(bfl=BamFileList(outpaths(args), yieldSize=50000), grl=feat, singleEnd=TRUE, readlength=
p2 <- plotfeaturetypeCounts(x=featureCounts2, graphicsfile="featureCounts2.pdf", graphicsformat="pdf", scales="

## End(Not run)
```

predORF *Predict ORFs*

Description

Predicts open reading frames (ORFs) and coding sequences (CDSs) in DNA sequences provided as DNAString or DNAStringSet objects.

Usage

```
predORF(x, n = 1, type = "grl", mode = "orf", strand = "sense", longest_disjoint=FALSE, startcodon = "
```

Arguments

x	DNA query sequence(s) provided as DNAString or DNAStringSet object.
n	Defines the maximum number of ORFs to return for each input sequence. The ORFs identified are sorted decreasingly by their length. For instance, n=1 (default) returns the longest ORF, n=2 the two longest ones, and so on.
type	One of three options provided as character values: 'df' returns results as data.frame, while 'gr' and 'grl' (default) return them as GRanges or GRangesList objects, respectively.
mode	The setting mode='ORF' returns a continuous reading frame that begins with a start codon and ends with a stop codon. The setting mode='CDS' return continuous reading frames that do not need to begin or end with start or stop codons, respectively.
strand	One of three options passed on as character vector of length one: 'sense' performs the predictions only for the sense strand of the query sequence(s), 'antisense' does it only for the antisense strand and 'both' does it for both strands.
longest_disjoint	If set to TRUE and n='all', the results will be subsetted to non-overlapping ORF set containing longest ORF.
startcodon	Defines the start codon(s) for ORF predictions. The default is set to the standard start codon 'ATG'. Any custom set of triplet DNA sequences can be assigned here.
stopcodon	Defines the stop codon(s) for ORF predictions. The default is set to the three standard stop codons 'TAA', 'TAG' and 'TGA'. Any custom set of triplet DNA sequences can be assigned here.

Value

Returns ORF/CDS ranges identified in query sequences as GRanges or data.frame object. The type argument defines which one of them will be returned. The objects contain the following columns:

- seqnames: names of query sequences

- subject_id: identified ORF/CDS ranges numbered by query
- start/end: start and end positions of ORF/CDS ranges
- strand: strand of query sequence used for prediction
- width: length of subject range in bases
- inframe2end: frame of identified ORF/CDS relative to 3' end of query sequence. This can be important if the query sequence was extracted directly upstream of an ORF (e.g. 5' UTR upstream of main ORF). The value 1 stands for in-frame with downstream ORF, while 2 or 3 indicates a shift of one or two bases, respectively.

Author(s)

Thomas Girke

See Also

scaleRanges

Examples

```
## Load DNA sample data set from Biostrings package
file <- system.file("extdata", "someORF.fa", package="Biostrings")
dna <- readDNAStringSet(file)

## Predict longest ORF for sense strand in each query sequence
(orf <- predORF(dna[1:4], n=1, type="gr", mode="orf", strand="sense"))

## Not run:
## Usage for more complex example
library(GenomicFeatures); library(systemPipeRdata)
gff <- system.file("extdata/annotation", "tair10.gff", package="systemPipeRdata")
txdb <- makeTxDbFromGFF(file=gff, format="gff3", organism="Arabidopsis")
futr <- fiveUTRsByTranscript(txdb, use.names=TRUE)
genome <- system.file("extdata/annotation", "tair10.fasta", package="systemPipeRdata")
dna <- extractTranscriptSeqs(FaFile(genome), futr)
uorf <- predORF(dna, n="all", mode="orf", longest_disjoint=TRUE, strand="sense")
grl_scaled <- scaleRanges(subject=futr, query=uorf, type="uORF", verbose=TRUE)
export.gff3(unlist(grl_scaled), "uorf.gff")

## End(Not run)
```

---

preprocessReads *Run custom read preprocessing functions*

Description

Applies custom read preprocessing functions to single-end or paired-end FASTQ files. The function uses the FastqStreamer function from the ShortRead package to stream through large files in a memory-efficient manner.

Usage

```
preprocessReads(args, Fct, batchsize = 1e+05, overwrite = TRUE, ...)
```

Arguments

args	Object of class SYSargs
Fct	character string of custom read preprocessing function call where both the input and output needs to be an object of class ShortReadQ. The name of the input ShortReadQ object needs to be fq.
batchsize	Number of reads to process in each iteration by the internally used FastqStreamer function.
overwrite	If TRUE existing file will be overwritten.
...	To pass on additional arguments to the internally used writeFastq function.

Value

Writes to files in FASTQ format. Their names are specified by outpaths(args).

Author(s)

Thomas Girke

See Also

FastqStreamer

Examples

```
## Preprocessing of single-end reads
param <- system.file("extdata", "trim.param", package="systemPipeR")
targets <- system.file("extdata", "targets.txt", package="systemPipeR")
args <- systemArgs(sysma=param, mytargets=targets)
## Not run:
preprocessReads(args=args, Fct="trimLRPatterns(Rpattern='GCCCGGGTAA', subject=fq)", batchsize=100000, overwrite

## End(Not run)

## Preprocessing of paired-end reads
param <- system.file("extdata", "trimPE.param", package="systemPipeR")
targets <- system.file("extdata", "targetsPE.txt", package="systemPipeR")
args <- systemArgs(sysma=param, mytargets=targets)
## Not run:
preprocessReads(args=args, Fct="trimLRPatterns(Rpattern='GCCCGGGTAA', subject=fq)", batchsize=100000, overwrite

## End(Not run)
```

qsubRun *Submit command-line tools to cluster*

Description

Note: This function as been deprecated. Please use `clusterRun` instead. `qsubRun` submits command-line tools to queue (e.g. Torque) of compute cluster using run specifications defined by `runX` and `getQsubargs` functions.

Usage

```
qsubRun(appfct="runCommandline(args=args, runid='01')", args, qsubargs, Nqsubs = 1, package = "system
```

Arguments

`appfct`	Accpets runX functions, such as `appfct="runCommandline(args, runid)"`
`args`	Argument list returned by `systemArgs()`.
`qsubargs`	Argument list returned by `getQsubargs()`.
`Nqsubs`	Interger defining the number of qsub processes. Note: the function will not assign more qsub processes than there are FASTQ files. E.g. if there are 10 FASTQ files and `Nqsubs=20` then the function will generate only 10 qsub processes. To increase the number of CPU cores used by each process, one can increase the `p` value under `systemArgs()`.
`package`	Package to load. Name provided as character vector of length one. Default is `sytemPipeR`.
`shebang`	defines `shebang` (fist line) used in submission shell script; default is set to `#!/bin/bash`.

Value

Returns list where list components contain FASTQ file names and their names are the qsub process IDs assiged by the queuing system. In addition, three files will be generated for each qsub submission process: `submitargs0X` (R object containing appargs), `submitargs0X.R` (R script using appargs) and `submitargs0X.sh` (shell submission script). In addition, the chosen `runX` function will output a `submitargs0X_log` file for each qsub process containing the executable commands processed by each qsub instance.

Author(s)

Thomas Girke

Examples

```
## Construct SYSargs object from param and targets files
param <- system.file("extdata", "tophat.param", package="systemPipeR")
targets <- system.file("extdata", "targets.txt", package="systemPipeR")
args <- systemArgs(sysma=param, mytargets=targets)
args
names(args); modules(args); cores(args); outpaths(args); sysargs(args)

## Not run:
## Execute SYSargs on single machine
runCommandline(args=args)

## Execute SYSargs on multiple machines
qsubargs <- getQsubargs(queue="batch", Nnodes="nodes=1", cores=cores(tophat), memory="mem=10gb", time="walltime=
qsubRun(args=args, qsubargs=qsubargs, Nqsubs=1, package="systemPipeR")
## Alignment stats
read_statsDF <- alignStats(fqpaths=tophatargs$infile1, bampaths=bampaths, fqgz=TRUE)
read_statsDF <- cbind(read_statsDF[targets$FileName,], targets)
write.table(read_statsDF, "results/alignStats.xls", row.names=FALSE, quote=FALSE, sep="\t")

## End(Not run)
```

---

readComp *Import sample comparisons from targets file*

Description

Parses sample comparisons specified in <CMP> line(s) of targets file or in `targetsheader` slot of `SYSargs` object. All possible comparisons can be specified with 'CMPset: ALL'.

Usage

```
readComp(file, format = "vector", delim = "-")
```

Arguments

`file`	Path to targets file. Alternatively, a `SYSargs` object can be assigned.
`format`	Object type to return: `vector` or `matrix`.
`delim`	Delimiter to use when sample comparisons are returned as `vector`.

Value

`list` where each component is named according to the name(s) used in the <CMP> line(s) of the targets file. The list will contain as many sample comparisons sets (list components) as there are sample comparisons lines in the corresponding targets file.

Author(s)

Thomas Girke

Examples

```
## Return comparisons from targets file
targetspath <- system.file("extdata", "targets.txt", package="systemPipeR")
read.delim(targetspath, comment.char = "#")
readComp(file=targetspath, format="vector", delim="-")

## Return comparisons from SYSargs object
param <- system.file("extdata", "tophat.param", package="systemPipeR")
targets <- system.file("extdata", "targets.txt", package="systemPipeR")
args <- systemArgs(sysma=param, mytargets=targets)
readComp(args, format = "vector", delim = "-")
```

---

returnRPKM *RPKM Normalization*

Description

Converts read counts to RPKM normalized values.

Usage

```
returnRPKM(counts, ranges)
```

Arguments

counts	Count data frame, e.g. from an RNA-Seq experiment.
ranges	GRangesList object, e.g. generated by exonsBy(txdb, by="gene").

Value

data.frame

Author(s)

Thomas Girke

Examples

```
## Not run:
countDFrpkm <- apply(countDF, 2, function(x) returnRPKM(counts=x, gffsub=eByg))

## End(Not run)
```

runCommandline *Execute SYSargs*

Description

Function to execute system parameters specified in SYSargs object

Usage

```
runCommandline(args, runid = "01", ...)
```

Arguments

args	object of class SYSargs
runid	Run identifier used for log file to track system call commands. Default is "01".
...	Additional plotting arguments to pass on to runCommandline().

Value

Output files, their paths can be obtained with outpaths() from SYSargs container. In addition, a character vector is returned containing the same paths.

Author(s)

Thomas Girke

Examples

```
## Construct SYSargs object from param and targets files
param <- system.file("extdata", "tophat.param", package="systemPipeR")
targets <- system.file("extdata", "targets.txt", package="systemPipeR")
args <- systemArgs(sysma=param, mytargets=targets)
args
names(args); modules(args); cores(args); outpaths(args); sysargs(args)

## Not run:
## Execute SYSargs on single machine
runCommandline(args=args)

## Execute SYSargs on multiple machines of a compute cluster
resources <- list(walltime="00:25:00", nodes=paste0("1:ppn=", cores(args)), memory="2gb")
reg <- clusterRun(args, conffile=".BatchJobs.R", template="torque.tmpl", Njobs=18, runid="01", resourceList=reso

## Monitor progress of submitted jobs
showStatus(reg)
file.exists(outpaths(args))
sapply(1:length(args), function(x) loadResult(reg, x)) # Works once all jobs have completed successfully.

## Alignment stats
```

```
read_statsDF <- alignStats(fqpaths=tophatargs$infile1, bampaths=bampaths, fqgz=TRUE)
read_statsDF <- cbind(read_statsDF[targets$FileName,], targets)
write.table(read_statsDF, "results/alignStats.xls", row.names=FALSE, quote=FALSE, sep="\t")

## End(Not run)
```

---

runDiff *Differential abundance analysis for many range sets*

Description

Convenience wrapper function for run_edgeR and run_DESeq2 to perform differential expression or abundance analysis iteratively for several count tables. The latter can be peak calling results for several samples or counts generated for different genomic feature types. The function also returns the filtering results and plots from filterDEGs.

Usage

```
runDiff(args, diffFct, targets, cmp, dbrfilter, ...)
```

Arguments

args	Object of class SYSargs where infile1(args) specifies the paths to the tabular read count data files and outpaths(args) to the result files.
diffFct	Defines which function should be used for the differential abundance analysis. Can be diffFct=run_edgeR or diffFct=run_DESeq2.
targets	targets data.frame
cmp	character matrix where comparisons are defined in two columns. This matrix should be generated with readComp() from the targets file. Values used for comparisons need to match those in the Factor column of the targets file.
dbrfilter	Named vector with filter cutoffs of format c(Fold=2, FDR=1) where Fold refers to the fold change cutoff (unlogged) and FDR to the p-value cutoff. Those values are passed on to the filterDEGs function.
...	Arguments to be passed on to the internally used run_edgeR or run_DESeq2 function.

Value

Returns list containing the filterDEGs results for each count table. Each result set is a list with four components which are described under ?filterDEGs. The result files contain the edgeR or DESeq2 results from the comparisons specified under cmp. The base names of the result files are the same as the corresponding input files specified under countfiles and the value of extension appended.

Author(s)

Thomas Girke

See Also

run_edgeR, run_DESeq2, filterDEGs

Examples

```
## Paths to BAM files
param <- system.file("extdata", "bowtieSE.param", package="systemPipeR")
targets <- system.file("extdata", "targets.txt", package="systemPipeR")
args_bam <- systemArgs(sysma=param, mytargets=targets)
bfl <- BamFileList(outpaths(args_bam), yieldSize=50000, index=character())

## Not run:
## SYSargs with paths to range data and count files
args <- systemArgs(sysma="param/count_rangesets.param", mytargets="targets_macs.txt")

## Iterative read counting
countDFnames <- countRangeset(bfl, args, mode="Union", ignore.strand=TRUE)
writeTargetsout(x=args, file="targets_countDF.txt", overwrite=TRUE)

## Run differential abundance analysis
cmp <- readComp(file=args_bam, format="matrix")
args_diff <- systemArgs(sysma="param/rundiff.param", mytargets="targets_countDF.txt")
dbrlist <- runDiff(args, diffFct=run_edgeR, targets=targetsin(args_bam), cmp=cmp[[1]], independent=TRUE, dbrfilt
writeTargetsout(x=args_diff, file="targets_rundiff.txt", overwrite=TRUE)

## End(Not run)
```

---

run_DESeq2 *Runs DESeq2*

Description

Convenience wrapper function to identify differentially expressed genes (DEGs) in batch mode with DESeq2 for any number of pairwise sample comparisons specified under the cmp argument. Users are strongly encouraged to consult the DESeq2 vignette for more detailed information on this topic and how to properly run DESeq2 on data sets with more complex experimental designs.

Usage

```
run_DESeq2(countDF, targets, cmp, independent = FALSE)
```

Arguments

countDF	date.frame containing raw read counts
targets	targets data.frame
cmp	character matrix where comparisons are defined in two columns. This matrix should be generated with the readComp() function from the targets file. Values used for comparisons need to match those in the Factor column of the targets file.

independent — If independent=TRUE then the countDF will be subsetted for each comparison. This behavior can be useful when working with samples from unrelated studies. For samples from the same or comparable studies, the setting independent=FALSE is usually preferred.

Value

data.frame containing DESeq2 results from all comparisons. Comparison labels are appended to column titles for tracking.

Author(s)

Thomas Girke

References

Please properly cite the DESeq2 papers when using this function: http://www.bioconductor.org/packages/devel/bioc/html/DES

See Also

run_edgeR, readComp and DESeq2 vignette

Examples

```
targetspath <- system.file("extdata", "targets.txt", package="systemPipeR")
targets <- read.delim(targetspath, comment="#")
cmp <- readComp(file=targetspath, format="matrix", delim="-")
countfile <- system.file("extdata", "countDFeByg.xls", package="systemPipeR")
countDF <- read.delim(countfile, row.names=1)
degseqDF <- run_DESeq2(countDF=countDF, targets=targets, cmp=cmp[[1]], independent=FALSE)
pval <- degseqDF[, grep("_FDR$", colnames(degseqDF)), drop=FALSE]
fold <- degseqDF[, grep("_logFC$", colnames(degseqDF)), drop=FALSE]
DEG_list <- filterDEGs(degDF=degseqDF, filter=c(Fold=2, FDR=10))
names(DEG_list)
DEG_list$Summary
```

---

run_edgeR *Runs edgeR*

Description

Convenience wrapper function to identify differentially expressed genes (DEGs) in batch mode with the edgeR GML method for any number of pairwise sample comparisons specified under the cmp argument. Users are strongly encouraged to consult the edgeR vignette for more detailed information on this topic and how to properly run edgeR on data sets with more complex experimental designs.

Usage

```
run_edgeR(countDF, targets, cmp, independent = TRUE, paired = NULL, mdsplot = "")
```

Arguments

countDF	date.frame containing raw read counts
targets	targets data.frame
cmp	character matrix where comparisons are defined in two columns. This matrix should be generated with readComp() from the targets file. Values used for comparisons need to match those in the Factor column of the targets file.
independent	If independent=TRUE then the countDF will be subsetted for each comparison. This behavior can be useful when working with samples from unrelated studies. For samples from the same or comparable studies, the setting independent=FALSE is usually preferred.
paired	Defines pairs (character vector) for paired analysis. Default is unpaired (paired=NULL).
mdsplot	Directory where plotMDS should be written to. Default setting mdsplot="" will omit the plotting step.

Value

data.frame containing edgeR results from all comparisons. Comparison labels are appended to column titles for tracking.

Author(s)

Thomas Girke

References

Please properly cite the edgeR papers when using this function: http://www.bioconductor.org/packages/devel/bioc/html/edgeR

See Also

run_DESeq2, readComp and edgeR vignette

Examples

```
targetspath <- system.file("extdata", "targets.txt", package="systemPipeR")
targets <- read.delim(targetspath, comment="#")
cmp <- readComp(file=targetspath, format="matrix", delim="-")
countfile <- system.file("extdata", "countDFeByg.xls", package="systemPipeR")
countDF <- read.delim(countfile, row.names=1)
edgeDF <- run_edgeR(countDF=countDF, targets=targets, cmp=cmp[[1]], independent=FALSE, mdsplot="")
pval <- edgeDF[, grep("_FDR$", colnames(edgeDF)), drop=FALSE]
fold <- edgeDF[, grep("_logFC$", colnames(edgeDF)), drop=FALSE]
DEG_list <- filterDEGs(degDF=edgeDF, filter=c(Fold=2, FDR=10))
names(DEG_list)
DEG_list$Summary
```

scaleRanges *Scale spliced ranges to genome coordinates*

Description

Function to scale mappings of spliced features (query ranges) to their corresponding genome coordinates (subject ranges). The method accounts for introns in the subject ranges that are absent in the query ranges. A use case example are uORFs predicted in the 5' UTRs sequences using predORF. These query ranges are given relative to the 5' UTR sequence. The scaleRanges function will scale them to the corresponding genome coordinates. This way they can be used in RNA-Seq expression experiments like other gene ranges.

Usage

```
scaleRanges(subject, query, type = "custom", verbose = TRUE)
```

Arguments

subject	Genomic ranges provided as GRangesList object. Their name and length requirements are described under query.
query	Feature level ranges provided as GRangesList object. The names of the query ranges need to match the names of the GRangesList object assigened to the subject argument. In addition, the length of each query range cannot exceed the total length of the corresponding subject range set.
type	Feature name to use in type column of GRangesList result.
verbose	The setting verbose=FALSE suppresses all print messages.

Value

Object of class GRangesList

Author(s)

Thomas Girke

See Also

predORF

Examples

```
## Usage for simple example
subject <- GRanges(seqnames="Chr1", IRanges(c(5,15,30),c(10,25,40)), strand="+")
query <- GRanges(seqnames="myseq", IRanges(1, 9), strand="+")
scaleRanges(GRangesList(myid1=subject), GRangesList(myid1=query), type="test")

## Not run:
```

```
## Usage for more complex example
library(GenomicFeatures); library(systemPipeRdata)
gff <- system.file("extdata/annotation", "tair10.gff", package="systemPipeRdata")
txdb <- makeTxDbFromGFF(file=gff, format="gff3", organism="Arabidopsis")
futr <- fiveUTRsByTranscript(txdb, use.names=TRUE)
genome <- system.file("extdata/annotation", "tair10.fasta", package="systemPipeRdata")
dna <- extractTranscriptSeqs(FaFile(genome), futr)
uorf <- predORF(dna, n="all", mode="orf", longest_disjoint=TRUE, strand="sense")
grl_scaled <- scaleRanges(subject=futr, query=uorf, type="uORF", verbose=TRUE)
export.gff3(unlist(grl_scaled), "uorf.gff")

## End(Not run)
```

`seeFastq` *Quality reports for FASTQ files*

Description

The following `seeFastq` and `seeFastqPlot` functions generate and plot a series of useful quality statistics for a set of FASTQ files including per cycle quality box plots, base proportions, base-level quality trends, relative k-mer diversity, length and occurrence distribution of reads, number of reads above quality cutoffs and mean quality distribution. The functions allow processing of reads with variable length, but most plots are only meaningful if the read positions in the FASTQ file are aligned with the sequencing cycles. For instance, constant length clipping of the reads on either end or variable length clipping on the 3' end maintains this relationship, while variable length clipping on the 5' end without reversing the reads erases it.

The function seeFastq computes the summary stats and stores them in a relatively small list object that can be saved to disk with `save()` and reloaded with `load()` for later plotting. The argument 'klength' specifies the k-mer length and 'batchsize' the number of reads to random sample from each fastq file.

Usage

```
seeFastq(fastq, batchsize, klength = 8)

seeFastqPlot(fqlist, arrange = c(1, 2, 3, 4, 5, 8, 6, 7), ...)
```

Arguments

`fastq`	Named character vector containing paths to FASTQ file in the data fields and sample labels in the name slots.
`batchsize`	Number of reads to random sample from each FASTQ file that will be considered in the QC analysis. Smaller numbers reduce the memory footprint and compute time.
`klength`	Specifies the k-mer length in the plot for the relative k-mer diversity.
`fqlist`	`list` object returned by `seeFastq()`.

arrange	Integer vector from 1 to 7 specifying the row order of the QC plot. Dropping numbers eliminates the corresponding plots.
...	Additional plotting arguments to pass on to seeFastqPlot().

Value

The function seeFastq returns the summary stats in a list containing all information required for the quality plots. The function seeFastqPlot plots the information generated by seeFastq using ggplot2.

Author(s)

Thomas Girke

Examples

```
## Not run:
args <- systemArgs(sysma="tophat.param", mytargets="targets.txt")
fqlist <- seeFastq(fastq=infile1(args), batchsize=10000, klength=8)
pdf("fastqReport.pdf", height=18, width=4*length(fastq))
seeFastqPlot(fqlist)
dev.off()


## End(Not run)
```

---

symLink2bam *Symbolic links for IGV*

Description

Function for creating symbolic links to view BAM files in a genome browser such as IGV.

Usage

```
symLink2bam(sysargs, command="ln -s", htmldir, ext = c(".bam", ".bai"), urlbase, urlfile)
```

Arguments

sysargs	Object of class SYSargs
command	Shell command, defaults to "ln -s"
htmldir	Path to HTML directory with http access.
ext	File name extensions to use for BAM and index files.
urlbase	The base URL structure to use in URL file.
urlfile	Name and path of URL file.

Value

symbolic links and url file

Author(s)

Thomas Girke

Examples

```
## Construct SYSargs object from param and targets files
param <- system.file("extdata", "tophat.param", package="systemPipeR")
targets <- system.file("extdata", "targets.txt", package="systemPipeR")
args <- systemArgs(sysma=param, mytargets=targets)

## Not run:
## Create sym links and URL file for IGV
symLink2bam(sysargs=args, command="ln -s", htmldir=c("~/.html/", "somedir/"), ext=c(".bam", ".bai"), urlbase="ht

## End(Not run)
```

---

sysargs *SYSargs accessor methods*

Description

Methods to access information from SYSargs object.

Usage

```
sysargs(x)
```

Arguments

x	object of class SYSargs

Value

various outputs

Author(s)

Thomas Girke

Examples

```
## Construct SYSargs object from param and targets files
param <- system.file("extdata", "tophat.param", package="systemPipeR")
targets <- system.file("extdata", "targets.txt", package="systemPipeR")
args <- systemArgs(sysma=param, mytargets=targets)
args
names(args); modules(args); cores(args); outpaths(args); sysargs(args)
```

SYSargs-class *Class* "SYSargs"

Description

S4 class container for storing parameters of command-line- or R-based software. SYSargs instances are constructed by the systemArgs function from two simple tablular files: a targets file and a param file. The latter is optional for workflow steps lacking command-line software. Typically, a SYSargs instance stores all sample-level inputs as well as the paths to the corresponding outputs generated by command-line- or R-based software generating sample-level output files. Each sample level input/outfile operation uses its own SYSargs instance. The outpaths of SYSargs usually define the sample inputs for the next SYSargs instance. This connectivity is achieved by writing the outpaths with the writeTargetsout function to a new targets file that serves as input to the next systemArgs call. By chaining several SYSargs steps together one can construct complex workflows involving many sample-level input/output file operations with any combinaton of command-line or R-based software.

Objects from the Class

Objects can be created by calls of the form new("SYSargs", ...).

Slots

targetsin: Object of class "data.frame" storing tabular data from targets input file

targetsout: Object of class "data.frame" storing tabular data from targets output file

targetsheader: Object of class "character" storing header/comment lines of targets file

modules: Object of class "character" storing software versions from module system

software: Object of class "character" name of executable of command-line software

cores: Object of class "numeric" number of CPU cores to use

other: Object of class "character" additional arguments

reference: Object of class "character" path to reference genome file

results: Object of class "character" path to results directory

infile1: Object of class "character" paths to first FASTQ file

infile2: Object of class "character" paths to second FASTQ file if data is PE

outfile1: Object of class "character" paths to output files generated by command-line software

sysargs: Object of class "character" full commands used to execute external software

outpaths: Object of class "character" paths to final outputs including postprocessing by Rsamtools

Methods

SampleName `signature(x = "SYSargs")`: extracts sample names

[`signature(x = "SYSargs")`: subsetting of class with bracket operator

coerce `signature(from = "list", to = "SYSargs")`: `as(list, "SYSargs")`

cores `signature(x = "SYSargs")`: extracts data from `cores` slot

infile1 `signature(x = "SYSargs")`: extracts data from `infile1` slot

infile2 `signature(x = "SYSargs")`: extracts data from `infile2` slot

modules `signature(x = "SYSargs")`: extracts data from `modules` slot

names `signature(x = "SYSargs")`: extracts slot names

length `signature(x = "SYSargs")`: extracts number of samples

other `signature(x = "SYSargs")`: extracts data from `other` slot

outfile1 `signature(x = "SYSargs")`: extracts data from `outfile1` slot

outpaths `signature(x = "SYSargs")`: extracts data from `outpath` slot

reference `signature(x = "SYSargs")`: extracts data from `reference` slot

results `signature(x = "SYSargs")`: extracts data from `results` slot

show `signature(object = "SYSargs")`: summary view of `SYSargs` objects

software `signature(x = "SYSargs")`: extracts data from `software` slot

targetsheader `signature(x = "SYSargs")`: extracts data from `targetsheader` slot

targetsin `signature(x = "SYSargs")`: extracts data from `targetsin` slot

targetsout `signature(x = "SYSargs")`: extracts data from `targetsout` slot

Author(s)

Thomas Girke

See Also

`systemArgs` and `runCommandline`

Examples

```
showClass("SYSargs")
## Construct SYSargs object from param and targets files
param <- system.file("extdata", "tophat.param", package="systemPipeR")
targets <- system.file("extdata", "targets.txt", package="systemPipeR")
args <- systemArgs(sysma=param, mytargets=targets)
args
names(args); targetsin(args); targetsout(args); targetsheader(args);
software(args); modules(args); cores(args); outpaths(args)
sysargs(args); other(args); reference(args); results(args); infile1(args)
infile2(args); outfile1(args); SampleName(args)

## Return sample comparisons
readComp(args, format = "vector", delim = "-")
```

```
## The subsetting operator '[' allows to select specific samples
args[1:4]

## Not run:
## Execute SYSargs on single machine
runCommandline(args=args)

## Execute SYSargs on multiple machines
qsubargs <- getQsubargs(queue="batch", Nnodes="nodes=1", cores=cores(args), memory="mem=10gb", time="walltime=2(
qsubRun(appfct="runCommandline(args=args)", appargs=args, qsubargs=qsubargs, Nqsubs=1, submitdir="results", pac

## Write outpaths to new targets file for next SYSargs step
writeTargetsout(x=args, file="default")

## End(Not run)
```

`systemArgs` *Constructs SYSargs object from param and targets files*

Description

Constructs `SYSargs` S4 class objects from two simple tablular files: a `targets` file and a `param` file. The latter is optional for workflow steps lacking command-line software. Typically, a `SYSargs` instance stores all sample-level inputs as well as the paths to the corresponding outputs generated by command-line- or R-based software generating sample-level output files. Each sample level input/outfile operation uses its own `SYSargs` instance. The outpaths of `SYSargs` usually define the sample inputs for the next `SYSargs` instance. This connectivity is established by writing the outpaths with the `writeTargetsout` function to a new targets file that serves as input to the next `systemArgs` call. By chaining several `SYSargs` steps together one can construct complex workflows involving many sample-level input/output file operations with any combinaton of command-line or R-based software.

Usage

```
systemArgs(sysma, mytargets, type = "SYSargs")
```

Arguments

`sysma`	path to 'param' file; file structure follows a simple name/value syntax that converted into JSON format; for details about the file structure see sample files provided by package. Assign `NULL` to run the pipeline without 'param' file. This can be useful for running partial workflows, e.g. with pregenerated BAM files.
`mytargets`	path to targets file
`type`	`type="SYSargs"` returns `SYSargs`, `type="json"` returns param file content in JSON format (requires `rjson` library)

Value

SYSargs object or character string in JSON format

Author(s)

Thomas Girke

See Also

showClass("SYSargs")

Examples

```
## Construct SYSargs object from param and targets files
param <- system.file("extdata", "tophat.param", package="systemPipeR")
targets <- system.file("extdata", "targets.txt", package="systemPipeR")
args <- systemArgs(sysma=param, mytargets=targets)
args
names(args); modules(args); cores(args); outpaths(args); sysargs(args)

## Not run:
## Execute SYSargs on single machine
runCommandline(args=args)

## Execute SYSargs on multiple machines of a compute cluster
resources <- list(walltime="00:25:00", nodes=paste0("1:ppn=", cores(args)), memory="2gb")
reg <- clusterRun(args, conffile=".BatchJobs.R", template="torque.tmpl", Njobs=18, runid="01", resourceList=reso

## Monitor progress of submitted jobs
showStatus(reg)
file.exists(outpaths(args))
sapply(1:length(args), function(x) loadResult(reg, x)) # Works once all jobs have completed successfully.

## Alignment stats
read_statsDF <- alignStats(fqpaths=infile1(args), bampaths=outpaths(args), fqgz=TRUE)
read_statsDF <- cbind(read_statsDF[targets$FileName,], targets)
write.table(read_statsDF, "results/alignStats.xls", row.names=FALSE, quote=FALSE, sep="\t")

## Write outpaths to new targets file for next SYSargs step
writeTargetsout(x=args, file="default")

## End(Not run)
```

---

variantReport *Generate Variant Report*

Description

Functions for generating tabular variant reports including genomic context annotations and confidence statistics of variants. The annotations are obtained with utilities provided by the `VariantAnnotation` package and the variant statistics are retrieved from the input VCF files.

Usage

```
## Variant report
variantReport(args, txdb, fa, organism)

## Combine variant reports
combineVarReports(args, filtercol, ncol = 15)

## Create summary statistics of variants
varSummary(args)
```

Arguments

`args`	Object of class `SYSargs`. The paths of the input VCF files are specified under `infile1(args)` and the paths of the output files under `outfile1(args)`.
`txdb`	Annotation data stored as `TranscriptDb` object, which can be obtained from GFF/GTF files, BioMart, Bioc Annotation packages, UCSC, etc. For details see the vignette of the `GenomicFeatures` package. It is important to use here matching versions of the `txdb` and `fa` objects. The latter is the genome sequence used for read mapping and variant calling.
`fa`	`FaFile` object pointing to the sequence file of the corresponding reference genome stored in FASTA format or a `BSgenome` instance.
`organism`	Character vector specifying the organism name of the reference genome.
`filtercol`	Named character vector containing in the name field the column titles to filter on, and in the data field the corresponding values to include in the report. For instance, the setting `filtercol=c(Consequence="nonsynonymous")` will include only nonsysynonymous variances listed in the `Consequence` column. To omit the filtering step, one can use the setting `filtercol="All"`.
`ncol`	Integer specifying the number of columns in the tabular input files. Default is set to 15.

Value

Tabular output files.

Author(s)

Thomas Girke

See Also

`filterVars`

Examples

```
## Alignment with BWA (sequentially on single machine)
param <- system.file("extdata", "bwa.param", package="systemPipeR")
targets <- system.file("extdata", "targets.txt", package="systemPipeR")
args <- systemArgs(sysma=param, mytargets=targets)
sysargs(args)[1]

## Not run:
system("bwa index -a bwtsw ./data/tair10.fasta")
bampaths <- runCommandline(args=args)

## Alignment with BWA (parallelized on compute cluster)
resources <- list(walltime="20:00:00", nodes=paste0("1:ppn=", cores(args)), memory="10gb")
reg <- clusterRun(args, conffile=".BatchJobs.R", template="torque.tmpl", Njobs=18, runid="01",
                  resourceList=resources)

## Variant calling with GATK
## The following creates in the inital step a new targets file
## (targets_bam.txt). The first column of this file gives the paths to
## the BAM files created in the alignment step. The new targets file and the
## parameter file gatk.param are used to create a new SYSargs
## instance for running GATK. Since GATK involves many processing steps, it is
## executed by a bash script gatk_run.sh where the user can specify the
## detailed run parameters. All three files are expected to be located in the
## current working directory. Samples files for gatk.param and
## gatk_run.sh are available in the subdirectory ./inst/extdata/ of the
## source file of the systemPipeR package.
writeTargetsout(x=args, file="targets_bam.txt")
system("java -jar CreateSequenceDictionary.jar R=./data/tair10.fasta O=./data/tair10.dict")
# system("java -jar /opt/picard/1.81/CreateSequenceDictionary.jar R=./data/tair10.fasta O=./data/tair10.dict")
args <- systemArgs(sysma="gatk.param", mytargets="targets_bam.txt")
resources <- list(walltime="20:00:00", nodes=paste0("1:ppn=", 1), memory="10gb")
reg <- clusterRun(args, conffile=".BatchJobs.R", template="torque.tmpl", Njobs=18, runid="01",
                  resourceList=resources)
writeTargetsout(x=args, file="targets_gatk.txt")

## Variant calling with BCFtools
## The following runs the variant calling with BCFtools. This step requires in
## the current working directory the parameter file sambcf.param and the
## bash script sambcf_run.sh.
args <- systemArgs(sysma="sambcf.param", mytargets="targets_bam.txt")
resources <- list(walltime="20:00:00", nodes=paste0("1:ppn=", 1), memory="10gb")
reg <- clusterRun(args, conffile=".BatchJobs.R", template="torque.tmpl", Njobs=18, runid="01",
                  resourceList=resources)
writeTargetsout(x=args, file="targets_sambcf.txt")

## Filtering of VCF files generated by GATK
args <- systemArgs(sysma="filter_gatk.param", mytargets="targets_gatk.txt")
filter <- "totalDepth(vr) >= 2 & (altDepth(vr) / totalDepth(vr) >= 0.8) & rowSums(softFilterMatrix(vr))==4"
# filter <- "totalDepth(vr) >= 20 & (altDepth(vr) / totalDepth(vr) >= 0.8) & rowSums(softFilterMatrix(vr))==6"
filterVars(args, filter, varcaller="gatk", organism="A. thaliana")
writeTargetsout(x=args, file="targets_gatk_filtered.txt")
```

```
## Filtering of VCF files generated by BCFtools
args <- systemArgs(sysma="filter_sambcf.param", mytargets="targets_sambcf.txt")
filter <- "rowSums(vr) >= 2 & (rowSums(vr[,3:4])/rowSums(vr[,1:4]) >= 0.8)"
# filter <- "rowSums(vr) >= 20 & (rowSums(vr[,3:4])/rowSums(vr[,1:4]) >= 0.8)"
filterVars(args, filter, varcaller="bcftools", organism="A. thaliana")
writeTargetsout(x=args, file="targets_sambcf_filtered.txt")

## Annotate filtered variants from GATK
args <- systemArgs(sysma="annotate_vars.param", mytargets="targets_gatk_filtered.txt")
txdb <- loadDb("./data/tair10.sqlite")
fa <- FaFile(systemPipeR::reference(args))
variantReport(args=args, txdb=txdb, fa=fa, organism="A. thaliana")

## Annotate filtered variants from BCFtools
args <- systemArgs(sysma="annotate_vars.param", mytargets="targets_sambcf_filtered.txt")
txdb <- loadDb("./data/tair10.sqlite")
fa <- FaFile(systemPipeR::reference(args))
variantReport(args=args, txdb=txdb, fa=fa, organism="A. thaliana")

## Combine results from GATK
args <- systemArgs(sysma="annotate_vars.param", mytargets="targets_gatk_filtered.txt")
combineDF <- combineVarReports(args, filtercol=c(Consequence="nonsynonymous"))
write.table(combineDF, "./results/combineDF_nonsyn_gatk.xls", quote=FALSE, row.names=FALSE, sep="\t")

## Combine results from BCFtools
args <- systemArgs(sysma="annotate_vars.param", mytargets="targets_sambcf_filtered.txt")
combineDF <- combineVarReports(args, filtercol=c(Consequence="nonsynonymous"))
write.table(combineDF, "./results/combineDF_nonsyn_sambcf.xls", quote=FALSE, row.names=FALSE, sep="\t")

## Summary for GATK
args <- systemArgs(sysma="annotate_vars.param", mytargets="targets_gatk_filtered.txt")
write.table(varSummary(args), "./results/variantStats_gatk.xls", quote=FALSE, col.names = NA, sep="\t")

## Summary for BCFtools
args <- systemArgs(sysma="annotate_vars.param", mytargets="targets_sambcf_filtered.txt")
write.table(varSummary(args), "./results/variantStats_sambcf.xls", quote=FALSE, col.names = NA, sep="\t")

## Venn diagram of variants
args <- systemArgs(sysma="annotate_vars.param", mytargets="targets_gatk_filtered.txt")
varlist <- sapply(names(outpaths(args))[1:4], function(x) as.character(read.delim(outpaths(args)[x])$VARID))
vennset_gatk <- overLapper(varlist, type="vennsets")
args <- systemArgs(sysma="annotate_vars.param", mytargets="targets_sambcf_filtered.txt")
varlist <- sapply(names(outpaths(args))[1:4], function(x) as.character(read.delim(outpaths(args)[x])$VARID))
vennset_bcf <- overLapper(varlist, type="vennsets")
vennPlot(list(vennset_gatk, vennset_bcf), mymain="", mysub="GATK: red; BCFtools: blue", colmode=2, ccol=c("blue"

## End(Not run)
```

---

vennPlot *Plot 2-5 way Venn diagrams*

Description

Ploting function of 2-5 way Venn diagrams from 'VENNset' objects or count set vectors. A useful feature is the possiblity to combine the counts from several Venn comparisons with the same number of label sets in a single Venn diagram.

Usage

```
vennPlot(x, mymain = "Venn Diagram", mysub = "default", setlabels = "default", yoffset = seq(0, 10, by
```

Arguments

x	VENNset or list of VENNset objects. Alternatively, a vector of Venn counts or a list of vectors of Venn counts can be provided as input. If several Venn comparisons are provided in a list then their results are combined in a single Venn diagram, where the count sets are organized above each other.
mymain	Main title of plot.
mysub	Subtitle of plot. Default mysub="default" reports the number of unique items in all sets, as well as the number of unique items in each individual set, respectively.
setlabels	The argument setlabels allows to provide a vector of custom sample labels. However, assigning the proper names in the name slots of the initial setlist is preferred for tracking purposes.
yoffset	The results from several Venn comparisons can be combined in a single Venn diagram by assigning to x a list with several VENNsets or count vectors. The positonal offset of the count sets in the plot can be controlled with the yoffset argument. The argument setting colmode allows to assign different colors to each count set. For instance, with colmode=2 one can assign to ccol a color vector or a list, such as ccol=c("blue", "red") or ccol=list(1:8, 8:1).
ccol	Character or numeric vector to define colors of count values, e.g. ccol=c("black","black","red").
colmode	See argument yoffset.
lcol	Character or numeric vector to define colors of set labels, e.g. lcol=c("red", "green")
lines	Character or numeric vector to define colors of lines in plot.
mylwd	Defines line width of shapes used in plot.
diacol	See argument type.
type	Defines shapes used to plot 4-way Venn diagram. Default type="ellipse" uses ellipses. The setting type="circle" returns an incomplete 4-way Venn diagram as circles. This representation misses two overlap sectors, but is sometimes easier to navigate than the default ellipse version. The missing Venn intersects are reported below the Venn diagram. Their font color can be controled with the argument diacol.
ccex	Controls font size for count values.
lcex	Controls font size for set labels.
sepsplit	Character used to separate sample labels in Venn counts.
...	Additional arguments to pass on.

Value

Venn diagram plot.

Note

The functions provided here are an extension of the Venn diagram resources on this site: http://manuals.bioinformatics.ucr.edu
Venn-Diagrams

Author(s)

Thomas Girke

References

See examples in 'The Electronic Journal of Combinatorics': http://www.combinatorics.org/files/Surveys/ds5/VennSymmExa

See Also

`overLapper`, `olBarplot`

Examples

```
## Sample data
setlist <- list(A=sample(letters, 18), B=sample(letters, 16),
                C=sample(letters, 20), D=sample(letters, 22),
                E=sample(letters, 18), F=sample(letters, 22))

## 2-way Venn diagram
vennset <- overLapper(setlist[1:2], type="vennsets")
vennPlot(vennset)

## 3-way Venn diagram
vennset <- overLapper(setlist[1:3], type="vennsets")
vennPlot(vennset)

## 4-way Venn diagram
vennset <- overLapper(setlist[1:4], type="vennsets")
vennPlot(list(vennset, vennset))

## Pseudo 4-way Venn diagram with circles
vennPlot(vennset, type="circle")

## 5-way Venn diagram
vennset <- overLapper(setlist[1:5], type="vennsets")
vennPlot(vennset)

## Alternative Venn count input to vennPlot (not recommended!)
counts <- sapply(vennlist(vennset), length)
vennPlot(counts)

## 6-way Venn comparison as bar plot
```

```
vennset <- overLapper(setlist[1:6], type="vennsets")
olBarplot(vennset, mincount=1)

## Bar plot of standard intersect counts
interset <- overLapper(setlist, type="intersects")
olBarplot(interset, mincount=1)

## Accessor methods for VENNset/INTERSECTset objects
names(vennset)
names(interset)
setlist(vennset)
intersectmatrix(vennset)
complexitylevels(vennset)
vennlist(vennset)
intersectlist(interset)

## Coerce VENNset/INTERSECTset object to list
as.list(vennset)
as.list(interset)

## Pairwise intersect matrix and heatmap
olMA <- sapply(names(setlist),
function(x) sapply(names(setlist),
function(y) sum(setlist[[x]] %in% setlist[[y]])))
olMA
heatmap(olMA, Rowv=NA, Colv=NA)

## Presence-absence matrices for large numbers of sample sets
interset <- overLapper(setlist=setlist, type="intersects", complexity=2)
(paMA <- intersectmatrix(interset))
heatmap(paMA, Rowv=NA, Colv=NA, col=c("white", "gray"))
```

VENNset-class *Class* "VENNset"

Description

Container for storing Venn intersect results created by the overLapper function. The setlist slot stores the original label sets as vectors in a list; intersectmatrix organizes the label sets in a present-absent matrix; complexitylevels represents the number of comparisons considered for each comparison set as vector of integers; and vennlist contains the Venn intersect vectors.

Objects from the Class

Objects can be created by calls of the form new("VENNset", ...).

Slots

setlist: Object of class "list": list of vectors

intersectmatrix: Object of class "matrix": binary matrix

`complexitylevels`: Object of class `"integer"`: `vector` of `integers`

`vennlist`: Object of class `"list"`: `list` of `vectors`

Methods

as.list `signature(x = "VENNset")`: coerces `VENNset` to `list`

coerce `signature(from = "list", to = "VENNset")`: `as(list, "VENNset")`

complexitylevels `signature(x = "VENNset")`: extracts data from `complexitylevels` slot

intersectmatrix `signature(x = "VENNset")`: extracts data from `intersectmatrix` slot

length `signature(x = "VENNset")`: returns number of original label sets

names `signature(x = "VENNset")`: extracts slot names

setlist `signature(x = "VENNset")`: extracts data from `setlist` slot

show `signature(object = "VENNset")`: summary view of `VENNset` objects

vennlist `signature(x = "VENNset")`: extracts data from `vennset` slot

Author(s)

Thomas Girke

See Also

`overLapper`, `vennPlot`, `olBarplot`, `INTERSECTset-class`

Examples

```
showClass("VENNset")

## Sample data
setlist <- list(A=sample(letters, 18), B=sample(letters, 16),
                C=sample(letters, 20), D=sample(letters, 22),
                E=sample(letters, 18), F=sample(letters, 22))

## Create VENNset
vennset <- overLapper(setlist[1:5], type="vennsets")
class(vennset)

## Accessor methods for VENNset/INTERSECTset objects
names(vennset)
setlist(vennset)
intersectmatrix(vennset)
complexitylevels(vennset)
vennlist(vennset)

## Coerce VENNset/INTERSECTset object to list
as.list(vennset)
```

writeTargetsout *Write updated targets out to file*

Description

Convenience write function for generating targets files with updated FileName columns containing the paths to files generated by input/output processes. These processes can be commandline- or R-based software. Typically, the paths to the inputs are stored in the targets infile (targetsin(args)) and the outputs are stored in the targets outfile (targetsout(args)). Note: by default the function cannot overwrite any existing files. If a file exists then the user has to explicitly remove it or set overwrite=TRUE.

Usage

```
writeTargetsout(x, file = "default", silent = FALSE, overwrite=FALSE, ...)
```

Arguments

x	Object of class SYSargs.
file	Name and path of output file. If set to "default" then the name of the output file will have the pattern 'targets_<software>.txt', where <software> will be what software(x) returns.
silent	If set to TRUE, all messages returned by the function will be suppressed.
overwrite	If set to TRUE, existing files of same name will be overwritten.
...	To pass on additional arguments.

Value

Writes tabular targes files containing the header/comment lines from targetsheader(x) and the columns from targetsout(x).

Author(s)

Thomas Girke

See Also

writeTargetsRef

Examples

```
## Create SYSargs object
param <- system.file("extdata", "tophat.param", package="systemPipeR")
targets <- system.file("extdata", "targets.txt", package="systemPipeR")
args <- systemArgs(sysma=param, mytargets=targets)

## Not run:
```

```
## Write targets out file
writeTargetsout(x=args, file="default")

## End(Not run)
```

---

writeTargetsRef *Generate targets file with reference*

Description

Generates targets file with sample-wise reference as required for some NGS applications, such as ChIP-Seq containing input samples. The reference sample information needs to be provided in the input file in a column called `SampleReference` where the values reference the labels used in the `SampleName` column. Sample rows without reference assignments will be removed automatically.

Usage

```
writeTargetsRef(infile, outfile, silent = FALSE, overwrite = FALSE, ...)
```

Arguments

`infile`	Path to input targets file.
`outfile`	Path to output targets file.
`silent`	If set to `TRUE`, all messages returned by the function will be suppressed.
`overwrite`	If set to `TRUE`, existing files of same name will be overwritten.
`...`	To pass on additional arguments.

Value

Generates modified targets file with the paths to the reference samples in the second column named `FileName2`. Note, sample rows not assigned reference samples are removed automatically.

Author(s)

Thomas Girke

See Also

`writeTargetsout`, `mergeBamByFactor`

Examples

```
## Path to input targets file
targets <- system.file("extdata", "targets_chip.txt", package="systemPipeR")

## Not run:
## Write modified targets file with reference (e.g. input) sample
writeTargetsRef(infile=targets, outfile="~/targets_refsample.txt", silent=FALSE, overwrite=FALSE)

## End(Not run)
```

Index