Package 'systemPipeR'

May 15, 2025

Type Package

Title systemPipeR: Workflow Environment for Data Analysis and Report Generation

Version 2.14.0

Date 2024-08-31

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, ImmunoOncology, ReportWriting, WorkflowStep, WorkflowManagement

Description systemPipeR is a multipurpose data analysis workflow environment that unifies R with command-line tools. It enables scientists to analyze many types of large- or small-scale data on local or distributed computer systems with a high level of reproducibility, scalability and portability. At its core is a command-line interface (CLI) that adopts the Common Workflow Language (CWL). This design allows users to choose for each analysis step the optimal R or command-line software. It supports both end-to-end and partial execution of workflows with built-in restart functionalities. Efficient management of complex analysis tasks is accomplished by a flexible workflow control container class. Handling of large numbers of input samples and experimental designs is facilitated by consistent sample annotation mechanisms. As a multi-purpose workflow toolkit, systemPipeR enables users to run existing workflows, customize them or design entirely new ones while taking advantage of widely adopted data structures within the Bioconductor ecosystem. Another important core functionality is the generation of reproducible scientific analysis and technical reports. For result interpretation, systemPipeR offers a wide range of plotting functionality, while an associated Shiny App offers many useful functionalities for interactive result exploration. The vignettes linked from this page include (1) a general introduction, (2) a description of technical details, and (3) a collection of workflow templates.

Depends Rsamtools (>= 1.31.2), Biostrings, ShortRead (>= 1.37.1), methods

Imports GenomicRanges, SummarizedExperiment, ggplot2, yaml, stringr, magrittr, S4Vectors, crayon, BiocGenerics, htmlwidgets

Suggests BiocStyle, knitr, rmarkdown, systemPipeRdata, GenomicAlignments, grid, dplyr, testthat, rjson, annotate, AnnotationDbi, kableExtra, GO.db, GenomeInfoDb, DT, rtracklayer, limma, edgeR, DESeq2, IRanges, batchtools, GenomicFeatures, txdbmaker, VariantAnnotation (>= 1.25.11)

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://systempipe.org/, https://github.com/tgirke/systemPipeR

git_url https://git.bioconductor.org/packages/systemPipeR

git_branch RELEASE_3_21

git_last_commit b9aa126

git_last_commit_date 2025-04-15

Repository Bioconductor 3.21

Date/Publication 2025-05-14

Contents

`systemPipeR-package` *systemPipeR package for Workflow Environment*

Description

The systemPipeR package provides a suite of R/Bioconductor for designing, building and running end-to-end analysis workflows on local machines, HPC clusters and cloud systems, while generating at the same time publication quality analysis reports.

For detailed information on usage, see the package vignette, by typing vignette("systemPipeR"), or more information on the project here: https://systempipe.org/spr

All software-related questions should be posted to the Bioconductor Support Site: https://support.bioconductor.org

The code can be viewed at the GitHub repository: https://github.com/tgirke/systemPipeR

Author(s)

Daniela Cassol, Tyler Backman, Thomas Girke

References

Backman TWH, Girke T (2016) systemPipeR: NGS workflow and report generation environment. BMC Bioinformatics 17 (1). https://doi.org/10.1186/s12859-016-1241-0

`addAssay-methods` *Extension accessor methods for SummarizedExperiment object*

Description

Accessors for adding new data to the 'assay' and 'metadata' slot of a SummarizedExperiment object

Usage

```
addAssay(x, ...)
addMetadata(x, ...)
```

Arguments

`x`	Object of class `SummarizedExperiment`.
`...`	dots, name of the object.

Methods

addAssay `signature(x = "SummarizedExperiment")`: add new dataset to `assays` slot

addMetadata `signature(x = "SummarizedExperiment")`: add new dataset to `metadata` slot

Author(s)

Daniela Cassol

`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, fqpaths, pairEnd = TRUE, output_index = 1, subset="FileName1")
```

Arguments

`args`	Object of class `SYSargs` or `SYSargs2` or `named character vector` with BAM files PATH and the elements names should be the sampleID.
`fqpaths`	`named character vector` with raw FASTQ files PATH and the names should be the sampleID. Required when `class(args)` is "character".
`pairEnd`	logical. For pair-end libraries, select `TRUE`.
`output_index`	A numeric index positions of the file in `SYSargs2` object, slot `output`. Default is `output_index=1`.
`subset`	`subset` are the variables defined in the `param.yml` file, for example "FileName1".

Value

`data.frame` with alignment statistics.

Author(s)

Thomas Girke

See Also

`clusterRun` and `runCommandline` and `output_update`

Examples

```
########################################
## Examples with \code{SYSargs2} object ##
########################################
## Construct SYSargs2 object from CWl param, CWL input, and targets files
targetspath <- system.file("extdata", "targets.txt", package="systemPipeR")
dir_path <- system.file("extdata/cwl", package="systemPipeR")
WF <- loadWorkflow(targets=targetspath, wf_file="hisat2/hisat2-mapping-se.cwl",
                   input_file="hisat2/hisat2-mapping-se.yml", dir_path=dir_path)
WF <- renderWF(WF, inputvars=c(FileName="_FASTQ_PATH1_", SampleName="_SampleName_"))
WF

names(WF); modules(WF); targets(WF)[1]; cmdlist(WF)[1:2]; output(WF)

## Not run:
## Execute SYSargs2 on single machine
WF <- runCommandline(args=WF, make_bam=TRUE)

## Alignment stats
read_statsDF <- alignStats(WF, subset="FileName")
write.table(read_statsDF, "results/alignStats.xls", row.names=FALSE, quote=FALSE, sep="\t")

## End(Not run)

########################################
## Examples with \code{SYSargs} object ##
########################################
## Construct SYSargs object from param and targets files
param <- system.file("extdata", "hisat2.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)

## Alignment stats
read_statsDF <- alignStats(args)
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
library("biomaRt")
listMarts()  # To choose BioMart database
listMarts(host = "plants.ensembl.org")
m <- useMart("plants_mart", host = "plants.ensembl.org")
listDatasets(m)
m <- useMart("plants_mart", dataset = "athaliana_eg_gene", host = "plants.ensembl.org")
listAttributes(m)  # Choose data types you want to download
go <- getBM(attributes = c("go_id", "tair_locus", "namespace_1003"), mart = m)
go <- go[go[, 3] != "", ]
go[, 3] <- as.character(go[, 3])
go[go[, 3] == "molecular_function", 3] <- "F"
```

```
go[go[, 3] == "biological_process", 3] <- "P"
go[go[, 3] == "cellular_component", 3] <- "C"
go[1:4, ]

dir.create("./data/GO", recursive = TRUE)
write.table(go, "data/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 = "data/GO/GOannotationsBiomart_mod.txt", lib = NULL, org = "",
                   colno = c(1, 2, 3), idconv = NULL)
catdb
save(catdb, file = "data/GO/catdb.RData")
load("data/GO/catdb.RData")

## 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)
```

`check.output` *Checking if the outfiles files exist*

Description

This function returns a `data.frame` indicating the number of existing files and how many files are missing.

Usage

```
check.output(sysargs, type="data.frame")
check.outfiles(sysargs, type="data.frame")
```

Arguments

`sysargs`	object of class `SYSargs2` or `SYSargsList`.
`type`	return object option. It can be `data.frame` or `list`.

Value

`data.frame` or `list` containing all the `outfiles` file information.

Author(s)

Daniela Cassol and Thomas Girke

See Also

- SYSargs2-class
- SYSargsList-class

Examples

```
## Construct SYSargs2 object
targets <- system.file("extdata", "targets.txt", package="systemPipeR")
dir_path <- system.file("extdata/cwl", package="systemPipeR")
WF <- loadWorkflow(targets=targets, wf_file="hisat2/hisat2-mapping-se.cwl",
                   input_file="hisat2/hisat2-mapping-se.yml", dir_path=dir_path)
WF <- renderWF(WF, inputvars=c(FileName="_FASTQ_PATH1_", SampleName="_SampleName_"))
WF
## Check output
check.output(WF)
check.output(WF, "list")

## Construct SYSargsList object
sal <- SPRproject(overwrite=TRUE)
targetspath <- system.file("extdata/cwl/example/targets_example.txt", package="systemPipeR")
appendStep(sal) <- SYSargsList(step_name = "echo",
                       targets=targetspath, dir=TRUE,
                wf_file="example/workflow_example.cwl", input_file="example/example.yml",
                       dir_path = system.file("extdata/cwl", package="systemPipeR"),
                       inputvars = c(Message = "_STRING_", SampleName = "_SAMPLE_"))
## Check outfiles
check.outfiles(sal)
```

`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`. `clusterRun` can be used with most queueing systems since it is based on utilities from the `batchtools` 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,
            more.args = list(args = args, make_bam = TRUE),
            conffile = ".batchtools.conf.R",
            template = "batchtools.slurm.tmpl",
            Njobs,
            runid = "01",
            resourceList)
```

Arguments

`args`	Object of class `SYSargs` or `SYSargs2`.

FUN	Accepts functions such as runCommandline(args, ...) where the args argument is mandatory and needs to be of class SYSargs or SYSargs2.
more.args	Object of class list, which provides the arguments that control the FUN function.
conffile	Path to conf file (default location ./.batchtools.conf.R). This file contains in its simplest form just one command, such as this line for the Slurm scheduler: cluster.functions <- makeClusterFunctionsSlurm(template="batchtools.slurm.tmpl"). For more detailed information visit this page: https://mllg.github.io/batchtools/index.html
template	The template files for a specific queueing/scheduling systems can be downloaded from here: https://github.com/mllg/batchtools/tree/master/inst/templates. Slurm, PBS/Torque, and Sun Grid Engine (SGE) templates are provided.
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 (Megabyte), number of nodes, CPU cores, walltime (minutes), 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)

Daniela Cassol and Thomas Girke

References

For more details on batchtools, please consult the following page: https://github.com/mllg/batchtools/

See Also

clusterRun replaces the older functions getQsubargs and qsubRun.

Examples

```
########################################
## Examples with \code{SYSargs} object ##
########################################
## Construct SYSargs object from param and targets files
param <- system.file("extdata", "hisat2.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 multiple machines of a compute cluster. The following
## example uses the conf and template files for the Slurm scheduler. Please
## read the instructions on how to obtain the corresponding files for other schedulers.
file.copy(system.file("extdata", ".batchtools.conf.R", package="systemPipeR"), ".")
file.copy(system.file("extdata", "batchtools.slurm.tmpl", package="systemPipeR"), ".")
resources <- list(walltime=120, ntasks=1, ncpus=cores(args), memory=1024)
reg <- clusterRun(args, FUN = runCommandline,
                    more.args = list(args = args, make_bam = TRUE),
                    conffile=".batchtools.conf.R",
                    template="batchtools.slurm.tmpl",
                    Njobs=18, runid="01",
                    resourceList=resources)

## Monitor progress of submitted jobs
getStatus(reg=reg)
file.exists(outpaths(args))

## End(Not run)

########################################
## Examples with \code{SYSargs2} object ##
########################################
## Construct SYSargs2 object from CWl param, CWL input, and targets files
targets <- system.file("extdata", "targets.txt", package="systemPipeR")
dir_path <- system.file("extdata/cwl", package="systemPipeR")
WF <- loadWorkflow(targets=targets, wf_file="hisat2/hisat2-mapping-se.cwl",
                  input_file="hisat2/hisat2-mapping-se.yml", dir_path=dir_path)
WF <- renderWF(WF, inputvars=c(FileName="_FASTQ_PATH1_", SampleName="_SampleName_"))
WF
names(WF); modules(WF); targets(WF)[1]; cmdlist(WF)[1:2]; output(WF)

## Not run:
## Execute SYSargs2 on multiple machines of a compute cluster. The following
## example uses the conf and template files for the Slurm scheduler. Please
## read the instructions on how to obtain the corresponding files for other schedulers.
file.copy(system.file("extdata", ".batchtools.conf.R", package="systemPipeR"), ".")
file.copy(system.file("extdata", "batchtools.slurm.tmpl", package="systemPipeR"), ".")
resources <- list(walltime=120, ntasks=1, ncpus=4, memory=1024)
reg <- clusterRun(WF, FUN = runCommandline,
                    more.args = list(args = WF, make_bam = TRUE),
                    conffile=".batchtools.conf.R",
                    template="batchtools.slurm.tmpl",
                    Njobs=18, runid="01", resourceList=resources)

## Monitor progress of submitted jobs
getStatus(reg=reg)

## Updates the path in the object \code{output(WF)}
WF <- output_update(WF, dir=FALSE, replace=TRUE, extension=c(".sam", ".bam"))
```

```
## Alignment stats
read_statsDF <- alignStats(WF)
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)
```

---

config.param *Adding param file*

Description

Replace or adding a input configuration setting at "YML" param file

Usage

```
config.param(input_file = NULL, param, file = "default", silent = FALSE)
```

Arguments

input_file	a list of parameters, param file path, or SYSargs2 object.
param	object of class list, expressing the values and names to be added or replace at the command-line definition or the configuration files.
file	name and path of the new file. If set to default, the name of the write file will have the pattern: <Date-Time>_<OriginalName>.yml. If set to append, the param information it will be append on the same file.
silent	if set to TRUE, all messages returned by the function will be suppressed.

Author(s)

Daniela Cassol

Examples

```
## Not run:
input_file <- system.file("extdata", "cwl/hisat2/hisat2-mapping-se.yml", package="systemPipeR")
param <- list(thread=10, fq=list(class="File", path="./results2"))
input <- config.param(input_file=input_file, param, file="default")

## End(Not run)
```

configWF *Workflow Steps Selection*

Description

This function allows the user to control of which step of the workflow will be run and the generation of the new RMarkdown.

Usage

```
configWF(x, input_steps = "ALL", exclude_steps = NULL, silent = FALSE, ...)
```

Arguments

x	object of class SYSargsList.
input_steps	a character vector of all steps desires to preserve on the output file. Default is ALL. It can be used the symbol ":" to select many steps in sequence, for example, input_steps=1:5.2, from step 1 to step 5.2. The symbol "." represents the substeps and symbol "," is used to separate selections. Jump from a major step to sub-step is supported, but if a major step is selected/excluded, all sub-steps of this major step will be selected/excluded. Repeatedly selected steps will only result in a unique step.It is recommended to put major steps in input_steps, like 1:4, 6:8, 10; and unwanted sub-steps in exclude_step, like 1.1, 3.1.1-3.1.3, 6.5. Reverse selecting is supported e.g. 10:1.
exclude_steps	a character vector of all steps desires to remove on the output file.
silent	if set to TRUE, all messages returned by the function will be suppressed.
...	Additional arguments to pass on to configWF().

Author(s)

Daniela Cassol

Examples

```
## Not run:
library(systemPipeRdata)
targets <- system.file("extdata", "targets.txt", package="systemPipeR")
script <- system.file("extdata/workflows/rnaseq", "systemPipeRNAseq.Rmd", package="systemPipeRdata")
SYSconfig <- initProject(projPath="./", targets=targets, script=script, overwrite=TRUE, silent=TRUE)

sysargslist <- configWF(x=sysargslist)

## 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, outfiles=NULL, format="tabular", ...)
```

Arguments

bfl	BamFileList object containing paths to one or more BAM files.
args	An instance of SYSargs or SYSargs2 constructed from a targets file where the first column (targetsin(args) or targets.as.df(targets(args))) contains the paths to the tabular range data files (e.g. peak ranges) used for counting. Another possibily is named character vector with the paths to the tabular range data files and the elements names should be the sampleID.
outfiles	Default is NULL. When args is an object of named character vector class, outfile argument is required. Named character vector with the paths to the resulting count tables and the elements names should be the sampleID.
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:
########################################
## Examples with \code{SYSargs2} object ##
########################################
## Construct SYSargs2 object
## SYSargs2 with paths to range data and count files
dir_path <- system.file("extdata/cwl/count_rangesets", package="systemPipeR")
args <- loadWF(targets = "targets_macs.txt", wf_file = "count_rangesets.cwl",
    input_file = "count_rangesets.yml", dir_path = dir_path)
args <- renderWF(args, inputvars = c(FileName = "_FASTQ_PATH1_", SampleName = "_SampleName_"))

## Iterative read counting
countDFnames <- countRangeset(bfl, args, mode="Union", ignore.strand=TRUE)

########################################
## Examples with \code{SYSargs} object ##
########################################
## Construct SYSargs object
## 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)
```

`createParam` *createParam*

Description

The constructor function creates an `SYSargs2` S4 class object from command-line string. Also, the function creates and saves the CWL param files. The latest storages all the parameters required for running command-line software, following the standard and specification defined on Common Workflow Language (CWL).

Usage

```
createParamFiles(commandline, cwlVersion = "v1.1", class = "CommandLineTool",
                  results_path = "./results", module_load = "baseCommand",
                  file = "default", syntaxVersion = "v1",
```

```
                    writeParamFiles = TRUE, confirm = FALSE,
                    overwrite = FALSE, silent = FALSE)

writeParamFiles(sysargs, file = "default", overwrite = TRUE, silent = FALSE,
                 syntaxVersion = "v1")
```

Arguments

commandline	string. Original command-line to create the CWL files from. Please see Details for more information.
cwlVersion	string. The version of the Common Workflow Language. More information here: https://www.commonwl.org/.
class	character. Name of Common Workflow Language Description class. The following is accepted: CommandLineTool.
results_path	Path to the results folder. Default is results.
module_load	string, Name of software to load by Environment Modules system. Default is "baseCommand", which creates a subfolder and two files: *.cwl and *.yml at ./param/cwl/.
file	character. Name and path of output files. If set to "default" then the name of the output files will have the pattern <software>.cwl and <software>.yml, where <software> will be what baseCommand(x) returns, when x is an object of class SYSargs2. Also, it creates a subfolder at ./param/cwl/ with name <software>.
syntaxVersion	character. One of "v1" or "v2", what CWL parsing syntax version to use. Default is to use the old version. The V2 version comes with more feature support, but has more syntax restrictions. See details.
writeParamFiles	logical. If set to TRUE, it will write to file the content of the CWL files:*.cwl and *.yml. Default is TRUE.
confirm	If set to FALSE and in an interactive section, it will prompt a question to proceed or not.
overwrite	logical. If set to TRUE, existing files of the same name will be overwritten. Default is FALSE.
silent	logical. If set to TRUE, all messages returned by the function will be suppressed. Default is FALSE.
sysargs	Object of class SYSargs2. Output from the createParamFiles function.

Details

Version 1 syntax:

- First line of the command-line object will be treated as the baseCommand;

- For argument lines (starting from the second line), any word before the first space with leading '-' or '–' in each will be treated as a prefix, like -S or --min. Any line without this first word will be treated as no prefix;

- All defaults are placed inside <...>;

- First argument is the input argument type. F for "File", int for integer, string for string;

- Optional: use the keyword `out` followed the type with a `,` comma separation to indicate if this argument is also a CWL output;

- Then, use `:` to separate keywords and default values, any non-space value after the `:` will be treated as the default value;

- If any argument has no default value, just a flag, like `--verbose`, there no need to add any `<...>`.

- The `\` is not required, however for consistency it is recommended to add.

Version 2 syntax:

- First line of the command-line object will be treated as the `baseCommand`;

- Each line specifies one argument and its default value.

- Each line is composed with exact 2 `;` to seprate 3 columns.

- Text before first `;` will be will used as prefix/names. If it starts with keyword "`p:`", anything after "`p:`" and before the first `;` will be used as prefix, and the name of this position will be the prefix but with leading dash(s) "`-`", "`-`" removed. If there is any duplication, a number index will be added to the end. If there is no keyword "`p:`" before first `;`, all text before first `;` will be the name.

- If there is keyword "`p:`" before first `;` but nothing before and after the second `;`, this position will be treated as CWL argument instead of input.

- Text between first and second `;` is type. Must be one of File, Directory, string, int, double, float, long, boolean.

- Text after second `;` and before `\` or end of the line is the default value. If it starts with keyword "`out`" or "`stdout`", this position will also be added to outputs or standard output.

- There is only 1 position with "`stdout`" allowed and usually it is the last position arguement.

- Ending with "`\`" is recommended but not required.

Value

`SYSargs2` object

Author(s)

Le Zhang and Daniela Cassol

References

For more details on `CWL`, please consult the following page: https://www.commonwl.org/

See Also

`writeParamFiles` `printParam` `subsetParam` `replaceParam` `renameParam` `appendParam` `loadWorkflow` `renderWF` `showClass("SYSargs2")`

Examples

```
## syntax version 1  example
command <- "
hisat2 \
    -S <F, out: ./results/M1A.sam> \
```

```
    -x <F: ./data/tair10.fasta> \
    -k <int: 1> \
    -min-intronlen <int: 30> \
    -max-intronlen <int: 3000> \
    -threads <int: 4> \
    -U <F: ./data/SRR446027_1.fastq.gz> \
    --verbose
"
cmd <- createParam(command, writeParamFiles=FALSE)
cmdlist(cmd)

## syntax version 2  example
command2 <- '
mycmd2 \
    p: -s; File; sample1.txt \
    p: -s; File; sample2.txt \
    p: --c; ; \
    p: -o; File; out: myout.txt \
    ref_genome; File; a.fasta \
    p: --nn; int; 12 \
    mystdout; File; stdout: abc.txt
'
cmd2 <- createParam(command2, syntaxVersion = "v2", writeParamFiles=FALSE)
cmdlist(cmd2)
```

`cwlFilesUpdate` *Update CWL description files*

Description

Function to sync and download the most updated CWL description files from the systemPipeR package.

Usage

```
cwlFilesUpdate(destdir, force = FALSE, verbose = TRUE)
```

Arguments

`destdir`	a character string with the directory path where the param are stored.
`force`	logical. Force the download of the CWL files.
`verbose`	logical. The setting `FALSE` suppresses all print messages.

Author(s)

Daniela Cassol

Examples

```
## Not run:
destdir <- "param/"
cwlFilesUpdate(destdir)

## End(Not run)
```

`EnvModules-class` *Class* `"EnvModules"`

Description

The function `module` enables use of the Environment Modules system (http://modules.sourceforge.net/) from within the R environment. By default the user's login shell environment (ie. `bash -l`) will be used to initialize the current session. The module function can also; load or unload specific software, list all the loaded software within the current session, and list all the applications available for loading from the module system. Lastly, the module function can remove all loaded software from the current session.

Objects from the Class

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

Slots

`available_modules`**:** Object of class `"list"` ~~

`loaded_modules`**:** Object of class `"list"` ~~

`default_modules`**:** Object of class `"list"` ~~

`modulecmd`**:** Object of class `"character"` ~~

Methods

[`signature(x = "EnvModules")`: ...

[[`signature(x = "EnvModules", i = "ANY", j = "missing")`: ...

[[<- `signature(x = "EnvModules")`: ...

$ `signature(x = "EnvModules")`: ...

available_modules `signature(x = "EnvModules")`: ...

coerce `signature(from = "EnvModules", to = "list")`: ...

coerce `signature(from = "list", to = "EnvModules")`: ...

default_modules `signature(x = "EnvModules")`: ...

EnvModules `signature(x = "EnvModules")`: ...

loaded_modules `signature(x = "EnvModules")`: ...

modulecmd `signature(x = "EnvModules")`: ...

names `signature(x = "EnvModules")`: ...

show `signature(object = "EnvModules")`: ...

Author(s)

Jordan Hayes and Daniela Cassol

Examples

```
showClass("EnvModules")
```

---

evalCode *Toggles option* eval *on the RMarkdown files*

Description

Function to evaluate (eval=TRUE) or not evaluate (eval=FALSE) R chunk codes in the Rmarkdown file. This function does not run the code, just toggles between TRUE or FALSE the option eval and writes out a new file with the chosen option.

Usage

```
evalCode(infile, eval = TRUE, output)
```

Arguments

infile	name and path of the infile file, format Rmd.
eval	whether to evaluate the code and include its results. The default is TRUE.
output	name and path of the output file. File format Rmd.

Value

Writes Rmarkdown file containing the exact copy of the infile file with the option choose on the eval argument. It will be easy to toggle between run all the R chunk codes or not.

Author(s)

Daniela Cassol

Examples

```
library(systemPipeRdata)
file <- system.file("extdata/workflows/rnaseq", "systemPipeRNAseq.Rmd", package="systemPipeRdata")
evalCode(infile=file, eval=FALSE, output=file.path(tempdir(), "test.Rmd"))
```

`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 SYSargs2 object from param and targets files
targetspath <- system.file("extdata", "targets.txt", package="systemPipeR")
dir_path <- system.file("extdata/cwl", package="systemPipeR")
args <- loadWorkflow(targets=targetspath, wf_file="hisat2/hisat2-mapping-se.cwl",
                 input_file="hisat2/hisat2-mapping-se.yml", dir_path=dir_path)
args <- renderWF(args, inputvars=c(FileName="_FASTQ_PATH1_", SampleName="_SampleName_"))
args

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

targetspath <- system.file("extdata", "targets.txt", package="systemPipeR")
dir_path <- system.file("extdata/cwl", package="systemPipeR")
args <- loadWorkflow(targets=targetspath, wf_file="hisat2/hisat2-mapping-se.cwl",
                 input_file="hisat2/hisat2-mapping-se.yml", dir_path=dir_path)
args <- renderWF(args, inputvars=c(FileName="_FASTQ_PATH1_", SampleName="_SampleName_"))
args <- runCommandline(args, make_bam = TRUE, dir = TRUE)
outpaths <- subsetWF(args , slot="output", subset=1, index=1)
file.exists(outpaths)

## (A) Generate binned coverage for two BAM files and 4 transcripts
grl <- cdsBy(txdb, "tx", use.names=TRUE)
fcov <- featureCoverage(bfl=BamFileList(outpaths[1:2]), grl=grl[1:4], resizereads=NULL,
                   readlengthrange=NULL, Nbins=20, method=mean, fixedmatrix=FALSE,
                   resizefeatures=TRUE, upstream=20, downstream=20,
                   outfile="results/featureCoverage.xls", overwrite=TRUE)
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[1:2]), grl=grl[1:4], resizereads=NULL,
                   readlengthrange=NULL, Nbins=NULL, method=mean, fixedmatrix=TRUE,
                   resizefeatures=TRUE, upstream=20, downstream=20,
                   outfile="results/featureCoverage_up_down.xls", overwrite=TRUE)
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[1:2]), grl=grl[1:4], resizereads=NULL,
                   readlengthrange=NULL, Nbins=20, method=mean, fixedmatrix=TRUE,
                   resizefeatures=TRUE, upstream=20, downstream=20,
                   outfile="results/featureCoverage_binned.xls", overwrite=TRUE)
```

```
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[1:2]), grl=grl[1:4], resizereads=NULL,
                    readlengthrange=NULL, Nbins=NULL, method=mean, fixedmatrix=FALSE,
                    resizefeatures=TRUE, upstream=20, downstream=20,
                    outfile="results/featureCoverage_query.xls", overwrite=TRUE)

## 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 SYSargs2 object from param and targets files
targets <- system.file("extdata", "targets.txt", package="systemPipeR")
dir_path <- system.file("extdata/cwl", package="systemPipeR")
args <- loadWorkflow(targets=targets, wf_file="hisat2/hisat2-mapping-se.cwl",
                 input_file="hisat2/hisat2-mapping-se.yml", dir_path=dir_path)
args <- renderWF(args, inputvars=c(FileName="_FASTQ_PATH1_", SampleName="_SampleName_"))
args

## Not run:
## Run alignments
args <- runCommandline(args, dir = FALSE, make_bam = TRUE)
outpaths <- subsetWF(args, slot = "output", subset = 1, index = 1)

## Features from sample data of systemPipeRdata package
library(txdbmaker)
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, yieldSize=50000), grl=feat, singleEnd=TRUE, readlength=c(74:76
p <- plotfeaturetypeCounts(x=fc, graphicsfile="featureCounts.pdf", graphicsformat="pdf", scales="fixed", anyread

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

## 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.char = "#")
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(files, filter, varcaller="gatk", organism,
           out_dir="results")
```

Arguments

files	named character vector with paths of the input VCF files.
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.
out_dir	Character vector of a results directory name. Default is results.

Value

Output files in VCF format. Their paths can be obtained with outpaths(args) or output(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:
library(VariantAnnotation)
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_type"))

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 txdbmaker package
library(txdbmaker)
gffFile <- system.file("extdata", "GFF3_files", "a.gff3", package="txdbmaker")
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=TRUE, upstream=1000, downstream=0)

## 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 = c("MF", "BP", "CC"), myslimv = "default",
            correct = TRUE, recordSpecGO = NULL, ...)

## 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/Affymetrix
                                                download=TRUE)
# catdb_conv <- makeCATdb(myfile="GOannotationsBiomart_mod.txt", lib=NULL, org="",
                           colno=c(1,2,3), idconv=list(affy=affy2locusDF))
# 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=0.001,
                                   gocats=c("MF", "BP", "CC"),
                                recordSpecGO=c("GO:0003674", "GO:0008150", "GO:0005575"))

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


## End(Not run)
```

`importWF` *Import R Markdown file as workflow*

Description

Import R Markdown file as workflow. Each R code chunk will be set as a step in the workflow. This operation requires a few extra settings on the R Markdown chunk options, to include a particular code chunk in the workflow analysis. Please check `Details`.

Usage

```
importWF(
```

```
    sysargs, file_path, ignore_eval = TRUE,
    update = FALSE, confirm = FALSE,
    check_tool = !update,
    check_module = check_tool,
    verbose = TRUE
)
```

Arguments

sysargs	SYSargsList empty object. More information on Details.
file_path	string, file path of the workflow file.
ignore_eval	logical, treat all R chunks' eval option as TRUE in workflow Rmd file even if some chunks have eval=FALSE.
update	logical, If you have ever changed the template and want to sync new changes, turn TRUE to update the workflow. This function will find the difference between old template and this new template, update line number records, update preprocessing code, and try to import new steps. The updated template is useful in renderReport() function.
confirm	logical, Only useful when you combine update = TRUE, some questions will be asked during update. Changing this to TRUE would skip these questions and directly say "Yes" to all answers. Useful in non-interactive mode. Default is to say "No" in non-interactive mode.
check_tool	logical, whether to check all required tools by this workflow are in PATH (callable). It uses the listCmdTools function to perform the check. The default is the reverse of update. It means if it is the first importing the workflow, tools and modules will be checked. If it is updating the existing workflow, tools will not be checked.
check_module	logical, whether to check all required modules are available. To do so, a modular system has be to installed. If no modular system, this check will be skipped, even if check_module = TRUE. It uses the listCmdModules function to perform the check. Check the help of module to know more about modular system.
verbose	logical, print out verbose message while function running.

Details

To include a particular code chunk from the R Markdown file in the workflow analysis, please use the following code chunk options:

- spr = 'r': for code chunks with R code lines; - spr = 'sysargs': for code chunks with an 'SYSargsList' object; - spr.dep = <StepName>: for specify the previous dependency. If this options is not found, it will automaticly add the previous step.

For spr = 'sysargs', the last object assigned needs to be the SYSargsList. If the spr flag is not found, the R chunk will not be included in the workflow.

It is required to start a project using SPRproject() function, and use the object to populate the steps from R Markdown file.

Value

`importWF` will return an `SYSargsList` object.

Author(s)

Le Zhang and Daniela Cassol

See Also

`SYSargsList` `renderReport`

Examples

```
file_path <- system.file("extdata/spr_simple_lw.Rmd", package="systemPipeR")
sal <- SPRproject(overwrite = TRUE)
sal <- importWF(sal, file_path)
```

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)
```

LineWise-class *Class* "LineWise"

Description

S4 class container for storing R-based code for a workflow step. `LineWise` `class` instances are constructed by the `LineWise` function, based on the R-based code, step name, and dependency tree. When the container is built from the R Markdown, using `importWF` function, two other slots are populated: `codeChunkStart` and `rmdPath`. `codeChunkStart` will store the first line of each R chunk, and `rmdPath` will store the R Markdown file path.

Usage

```
## Constructor

LineWise(code, step_name = "default", codeChunkStart = integer(),
            rmdPath = character(), dependency="",
            run_step = "mandatory",
            run_session = "management",
            run_remote_resources = NULL)
```

Arguments

code	R code separated either by a semi-colon (`;`), or by a newline, and enclosed by braces (`{ }`).
step_name	character. Step name needs to be unique and is required when appending this step to the workflow.
codeChunkStart	integer. R Markdown code chunk line start. This element will be populated when the object is built by `importWF`.
rmdPath	character. Path of R Markdown file used by `importWF`.
dependency	character. Dependency tree, required when appending this step to the workflow. Character name of a previous step in the workflow. Default is empty string `""`.
run_step	character. If the step has "mandatory" or "optional" flag for the execution.
run_session	character. If the step has "management" or "compute" flag for the execution.
run_remote_resources	`List` for reserving for each cluster job sufficient computing resources including memory (Megabyte), number of nodes, CPU cores, walltime (minutes), etc. It is necessary two aditional files: `conffile` and `template`. `conffile` is the path to conf file (default location `./.batchtools.conf.R`). This file contains in its simplest form just one command, such as this line for the Slurm scheduler: `cluster.functions <- makeClusterFunctionsSlurm(template="batchtools.slurm.tmpl")`. For more detailed information visit this page: https://mllg.github.io/batchtools/index.html `template` The template files for a specific queueing/scheduling systems can be downloaded from here: https://github.com/mllg/batchtools/tree/master/inst/templates. Slurm, PBS/Torque, and Sun Grid Engine (SGE) templates are provided

Objects from the Class

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

Slots

`codeLine`**:** Object of class `"expression"` storing R-based code.

`codeChunkStart`**:** Object of class `"interger"` storing start line from the `rmdPath` file, when the `"LineWise"` is built from R Markdown.

`stepName`**:** Object of class `"character"` storing step name.

`dependency`**:** Object of class `"list"` storing dependency tree.

`status`: Object of class `"list"` storing status steps.

`files`: Object of class `"list"` storing file for R Markdown file and the file containing stdout and stderr after running the R-based code.

`runInfo`: Object of class `"list"` storing all the `runInfo` information of the workflow

Functions and Methods

See 'Usage' for details on invocation.

Constructor:

LineWise: Returns a `LineWise` object.

Accessors:

codeLine Printing method for the `CodeLine` slot.

codeChunkStart Extract start line of the R Markdown R chunk.

rmdPath Extract Rmarkdown file path.

stepName Extract the step name.

dependency Extract the dependency tree.

status Extract status of the step.

files Extract log file path storing stdout and stderr after running step.

appendCodeLine<- Replacement method for append a R code line.

replaceCodeLine<- Replacement method for replace a R code line.

Methods:

[Return a new `LineWise` object made of the selected R code lines.

[[Extract the slot information from `LineWine` object.

[[<- Replacement method for `LineWine` slots.

$ Extract slots elements by name.

length Extract number of R-based code lines.

names Extract slot names.

show Summary view of `LineWise` elements.

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

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

linewise Coerce back to list `as(LineWise, "list")`

Author(s)

Daniela Cassol

See Also

`SYSargsList`

Examples

```
showClass("LineWise")
lw <- LineWise(code = {
              log_out <- log(10)
              },
              step_name = "R_log")

codeLine(lw)

## ImportWF option
file_path <- system.file("extdata/spr_simple_lw.Rmd", package="systemPipeR")
sal <- SPRproject(overwrite = TRUE)
# file_path <- "../inst/extdata/spr_simple_lw.Rmd"
sal <- importWF(sal, file_path)
sal <- runWF(sal)
lw2 <- sal$stepsWF[[2]]
lw2
names(lw2)
length(lw2)

## Accessors
codeLine(lw2)
codeChunkStart(lw2)
rmdPath(lw2)
stepName(lw2)
dependency(lw2)
status(lw2)
files(lw2)

## Replacement
appendCodeLine(lw2, after = 0) <- "log <- log(10)"
codeLine(lw2)
replaceCodeLine(lw2, 1) <- "plot(iris)"
codeLine(lw2)

## Coerce
lw2 <- linewise(lw2) ## OR lw2 <- as(lw2, "list")
lw2 <- as(lw2, "LineWise")
```

`listCmdTools` *List/check the existence of command-line tools of a workflow*

Description

These functions list/check whether required command-line tools/modules are installed in the PATH and are callable.

Usage

```
listCmdTools(sal, check_path = FALSE, check_module = FALSE)

listCmdModules(sal, check_module = FALSE)
```

Arguments

sal	SPR workflow object in SYSargsList class.
check_path	logical, whether to check if the required tools are in PATH.
check_module	logical, whether to check if the required modules are installed.

Details

Both functions by default will not check the existence of tools or modules. The default is to list the requirement.

Value

Both functions print out the list/check results as dataframe. The first column is workflow step names that require certain tools/modules. The second column is the tool/module names. The third column is logical, TRUE for the existence of the tool in PATH/modular system, if check_path = TRUE or check_module = TRUE. Otherwise, the third column will be NA.

In the case of both check_path = TRUE, check_module = TRUE for listCmdTools, the returned dataframe is still results for tool PATH checking but not module checking results. If one wish to obtain the module checking results, please use listCmdModules.

When the current workflow has no command-line (SYSargs) step, or there is no module required, or there is no modular system installed, the return will be NULL.

These two functions are automatically performed when importWF is called.

Author(s)

Le Zhang

See Also

importWF module

Examples

```
# See examples of `importWF`
```

loadWorkflow *Constructs SYSargs2 object from CWL param and targets files*

Description

The constructor functions create an SYSargs2 S4 class object from three input files: a CWL param and input files, and one simple tabular or yml file, a targets file. The latter is optional for workflow steps lacking input files. TheCWL param provides all the parameters required for running command-line software, following the standard and specification defined on Common Workflow Language (CWL). The input file provides additional information for the command-line, allowing each sample level input/outfile operation uses its own SYSargs2 instance. In the targets file users could provide the paths to the initial sample input files (e.g. FASTQ) along with sample labels, and if appropriate biological replicate and contrast information for controlling differential abundance analyses.

The renderWF function populates all the command-line for each sample in each step of the particular workflow. Each sample level input/outfile operation uses its own SYSargs2 instance. The output of SYSargs2 define all the expected output files for each step in the workflow, which usually it is the sample input for the next step in an SYSargs2 instance. By chaining several SYSargs2 steps together one can construct complex workflows involving many sample-level input/output file operations with any combination of command-line or R-based software. Between different instances, this connectivity is established by `appendStep<-` method. Please check more details from SYSargsList-class class.

Usage

```
loadWorkflow(targets = NULL, wf_file, input_file, dir_path = "param/cwl", id = "SampleName")

renderWF(WF, inputvars = NULL)

updateWF(WF, write.yaml=FALSE, name.yaml="default", new_targets=NULL,
                     new_targetsheader=NULL, inputvars=NULL, silent=FALSE)
```

Arguments

targets	either the path to targets file or an object of SummarizedExperiment class. The targets file can be either a simple tabular or yml file. Also, it is possible to assign NULL to run the pipeline without the 'targets' file. This can be useful for running specific workflows that do not require input files.
wf_file	name and path to CWL parameters file.
input_file	name and path to input parameters file.
dir_path	path to the parameters directory with the wf_file and input_file files. It is recommended to keep both files in the same directory.
id	A column from targets file, which will be used as an id for each one of the samples. It is required to be unique.
WF	Object of class SYSargs2, generated by loadWF .

inputvars — named character vector. Variables defined in the input file that matches the column names defined in the targets file.

write.yaml — logical. If set to TRUE, it will write to file the content of the CWL files: *.yml. Default is FALSE.

name.yaml — name and path to input parameters file, if write.yaml is set to TRUE. Default value will write a file at the same directory of dir_path appending to the file name the current date.

new_targets — new targets files as list. 'targets' data.frame can be converted by targets.as.list function. Default is NULL, and it will maintain the original.

new_targetsheader — character. New header/comment lines of targets file. Default is NULL, and it will maintain the original.

silent — If set to TRUE, all messages returned by the function will be suppressed.

Value

SYSargs2 object.

Author(s)

Daniela Cassol and Thomas Girke

See Also

showClass("SYSargs2")

Examples

```
## Construct SYSargs2 object from CWl param, CWL input, and targets files
targets <- system.file("extdata", "targets.txt", package="systemPipeR")
dir_path <- system.file("extdata/cwl", package="systemPipeR")
WF <- loadWorkflow(targets=targets, wf_file="hisat2/hisat2-mapping-se.cwl",
                   input_file="hisat2/hisat2-mapping-se.yml", dir_path=dir_path)
WF <- renderWF(WF, inputvars=c(FileName="_FASTQ_PATH1_", SampleName="_SampleName_"))
WF

## If required to update the object
yamlinput(WF, "thread")  <- 6L
WF <- updateWF(WF)
cmdlist(WF)[1]
yamlinput(WF)$thread
```

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 or SYSargs2 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. When a named character vector is provided as input, a data.frame with a target containing the paths to the merged BAM files as output.

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, targetsDF = NULL, mergefactor = "Factor",
                 out_dir = file.path("results", "merge_bam"),
                 overwrite = FALSE, silent = FALSE, ...)
```

Arguments

args	An instance of SYSargs or SYSargs2 constructed from a targets file where the first column (targetsin(args) or targets.as.df(targets(args))) contains the paths to the BAM files along with the column title FileName. Another possibily is named character vector with BAM files PATH and the elements names should be the sampleID.
targetsDF	This argument is required when named character vector is provided as input. Default is NULL. Object of class DFrame, and it can be obtained with targetsWF(<SYSargsList>).
mergefactor	factor containing the grouping information required for merging the BAM files referenced in the first column of targetsin(args) or targets.as.df(targets(args)). The default uses Factor column from the targets files as factor. The latter merges BAM files for which replicates are specified in the Factor column.
out_dir	The directory path to store merged bam files. Default uses "merge_bam" directory inside the results directory. directory not existing before running the function is alllowed. It will be created while running.
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 or SYSargs2 object where all output file paths contain the paths to the merged BAM files. When a named character vector is provided as input, a data.frame with a target containing the paths to the merged BAM files as output. 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 allow full use of the Environment Modules system for Unix*

Description

The function module enables use of the Environment Modules system (http://modules.sourceforge.net/) from within the R environment. The user's login shell environment (i.e. bash -l) will be used to initialize the current session. The module function can also; load or unload specific software, list all the loaded software within the current session, and list all the applications available for loading from the module system. Lastly, the module function can remove all loaded software from the current session.

Usage

```
module(action_type, module_name = NULL)
moduleload(module_name)
moduleUnload(module_name)
modulelist()
moduleAvail()
moduleClear()
moduleInit()
```

Arguments

action_type	Name of the action to be executed as character vector. The following switches are accepted: avail, list, init, load, unload, and clear.
module_name	Name of software to load as character vector. Examples: "hisat2", "hisat2/2.1.0", c("hisat2", "samtools").

Details

Partial failure would also result 'FALSE', e.g. "load" two modules, one successful and the other failed, then the return is 'FALSE'. For "unload" action will always return 'TRUE' even if the module is not loaded at all or not found.

Author(s)

Tyler Backman, Jordan Hayes and Thomas Girke

Examples

```
## Not run:
## List all available software from the module system
avail <- moduleAvail()

## List loaded software in the current session
modulelist()

## Example for loading a software into the shell environment
moduleload("hisat2")
moduleload("hisat2/2.2.1")

## Example for removing software from the shell environment
moduleUnload("hisat2")

## Clear all of the software from the shell's initialization files
moduleClear()

## List and load all the software loaded in users default login shell into the
current session (default)
moduleInit()

## 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 = "de
```

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"))
```

---

olRanges *Identify Range Overlaps for IRanges and GRanges Object*

Description

Function for identifying consensus peak among two peaks sets sharing a minimum relative overlap.

Usage

```
olRanges(query, subject, output = "gr")
```

Arguments

query	Object of class GRanges, which is a vector of genomic locations and associated annotations.
subject	Object of class GRanges.
output	By default "gr" returns any overlap with OL length information in an object of class GRanges. Also, can returns an object of class data.frame with "df".

Author(s)

Thomas Girke

Examples

```
## Sample Data Sets
grq <- GRanges(seqnames = Rle(c("chr1", "chr2", "chr1", "chr3"), c(1, 3, 2, 4)), ranges = IRanges::IRanges(seq(1, 1
grs <- shift(grq[c(2,5,6)], 5)
## Run olRanges function
olRanges(query=grq, subject=grs, output="df")
olRanges(query=grq, subject=grs, output="gr")
```

output_update *Updates the output files paths in the* SYSargs2 *object*

Description

After executing all the command-lines by the runCommadline function, the output files can be created in specific directories rather then results in a particular directory. Also, the runCommadline function allows converting the SAM file outputs to sorted and indexed BAM files. Thus, the output_update function allows updating the location of these files in the output of the SYSargs2 object.

Usage

```
output_update(args, dir = FALSE, dir.name = NULL, replace = FALSE, extension = NULL, make_bam=FALSE, del
```

Arguments

args	object of class SYSargs2.
dir	assign TRUE to update the location of the output files in the args object accordingly with the workflow name directory. Default is dir=FALSE.
dir.name	if the results directory name is not specified in the input file, it is possible to specify here the name. This argument is required if the path name return NULL from the input file. Default is dir.name=NULL.
replace	replace the extension for selected output files in the args object. Default is replace=FALSE.
extension	object of class "character" storing the current extension of the files and the respective replacement. For example, runCommandline function by default autodetects SAM file outputs in the args object and create the BAM files. In order to update the output of args object, the extension argument should be set: extension = c(".sam", ".bam").
make_bam	Auto detects SAM file outputs and update them on the SYSargs2 object for sorted and indexed BAM files. Default is make_bam=FALSE. This argument should be used in integration with runCommandline function.
del_sam	This option allows deleting the SAM files created when the make_BAM converts the SAM files to sorted and indexed BAM files. Default is del_sam=TRUE.

Value

SYSargs2 object with output location files updated.

Author(s)

Daniela Cassol and Thomas Girke

See Also

To check directory name in the input file: yamlinput(WF)$results_path$path.

Examples

```
## Construct SYSargs2 object from CWl param, CWL input, and targets files
targets <- system.file("extdata", "targets.txt", package="systemPipeR")
dir_path <- system.file("extdata/cwl", package="systemPipeR")
WF <- loadWorkflow(targets=targets, wf_file="hisat2/hisat2-mapping-se.cwl",
                   input_file="hisat2/hisat2-mapping-se.yml", dir_path=dir_path)
WF <- renderWF(WF, inputvars=c(FileName="_FASTQ_PATH1_", SampleName="_SampleName_"))
WF
output(WF)

## Not run:
runCommandline(args=WF, make_bam=TRUE)
## Output paths update
WF <- output_update(WF, dir=FALSE, replace=TRUE, extension=c(".sam", ".bam"))

runCommandline(args=WF, make_bam=TRUE, dir=TRUE)
## Output paths update
WF <- output_update(WF, dir=TRUE, replace=TRUE, extension=c(".sam", ".bam"))

## End(Not run)
```

---

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 SYSargs2 object from param and targets files
targets <- system.file("extdata", "targets.txt", package="systemPipeR")
dir_path <- system.file("extdata/cwl", package="systemPipeR")
args <- loadWorkflow(targets=targets, wf_file="hisat2/hisat2-mapping-se.cwl",
                  input_file="hisat2/hisat2-mapping-se.yml", dir_path=dir_path)
args <- renderWF(args, inputvars=c(FileName="_FASTQ_PATH1_", SampleName="_SampleName_"))
args

## Not run:
## Run alignments
args <- runCommandline(args, dir = FALSE, make_bam = TRUE)
outpaths <- subsetWF(args, slot = "output", subset = 1, index = 1)

## Features from sample data of systemPipeRdata package
library(txdbmaker)
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[1:2]), grl=grl[1:4], resizereads=NULL,
```

```
                    readlengthrange=NULL, Nbins=20, method=mean, fixedmatrix=FALSE,
                    resizefeatures=TRUE, upstream=20, downstream=20,
                    outfile="results/featureCoverage.xls")
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[1:2]), grl=grl[1:4], resizereads=NULL,
                    readlengthrange=NULL, Nbins=NULL, method=mean, fixedmatrix=TRUE,
                    resizefeatures=TRUE, upstream=20, downstream=20,
                    outfile="results/featureCoverage_UpDown.xls")
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[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[1:2]), grl=grl[1:4], resizereads=NULL,
                    readlengthrange=NULL, Nbins=NULL, method=mean, fixedmatrix=FALSE,
                    resizefeatures=TRUE, upstream=20, downstream=20,
                    outfile="results/RleCoverage.xls")

## 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 = FALSE,
                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 SYSargs2 object from param and targets files
targets <- system.file("extdata", "targets.txt", package="systemPipeR")
dir_path <- system.file("extdata/cwl", package="systemPipeR")
args <- loadWorkflow(targets=targets, wf_file="hisat2/hisat2-mapping-se.cwl",
                  input_file="hisat2/hisat2-mapping-se.yml", dir_path=dir_path)
args <- renderWF(args, inputvars=c(FileName="_FASTQ_PATH1_", SampleName="_SampleName_"))
args

## Not run:
## Run alignments
args <- runCommandline(args, dir = FALSE, make_bam = TRUE)
outpaths <- subsetWF(args, slot = "output", subset = 1, index = 1)

## Features from sample data of systemPipeRdata package
library(txdbmaker)
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, yieldSize=50000), grl=feat, singleEnd=TRUE, readlength=c(74:76
p <- plotfeaturetypeCounts(x=fc, graphicsfile="featureCounts.pdf", graphicsformat="pdf", scales="fixed", anyread

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

## End(Not run)
```

---

plotWF *Visualize SPR workflow and status*

Description

Visualize SPR workflow and status. `plotWF` is the general function that creates the plot. `plotwfOutput` and `renderPlotwf` are used in Shiny UI and server respectively, similar to `plotOutput` and `renderPlot`.

Usage

```
plotWF(
  sysargs,
  width = NULL, height = NULL,
  elementId = NULL,
  responsive = TRUE,
  branch_method = "auto",
  branch_no = NULL,
  layout = "compact",
  no_plot = FALSE,
```

```
  plot_method = "svg",
  out_format = "plot",
  out_path = NULL,
  show_legend = TRUE,
  mark_main_branch = FALSE,
  rstudio = FALSE,
  in_log = FALSE,
  rmarkdown = "detect",
  verbose = FALSE,
  show_warns = FALSE,
  plot_ctr = TRUE,
  pan_zoom = FALSE,
  exit_point = 0
)

plotwfOutput(
  outputId,
  width = '100%',
  height = '400px'
)

renderPlotwf(
  expr,
  env = parent.frame(),
  quoted = FALSE
)
```

Arguments

sysargs	object of class SYSargsList.
width	string, a valid CSS string for width, like "500px", "100%".
height	string, a valid CSS string for height, like "500px", "100%".
elementId	string, optional ID value for the plot.
responsive	bool, should the plot be responsive? useful in Rstudio built-in viewer, Rmarkdown, Shiny or embed it into other web pages.
branch_method	string, one of "auto", "choose". How to determine the main branch of the workflow. "auto" will be determined by internal alrgothrim: Branches connecting the frist and last step and/or the longest will be favored. "choose" will list all possible branches and you can make a choice.
branch_no	numeric, only works if branch_method == "choose". Specify a branch number to be the main branch instead of choosing from the prompt. This option can be good if you are in a non-interactive mode, e.g. rendering Rmd.
layout	string, one of "compact", "vertical", "horizontal", "execution".
no_plot	bool, if you want to assgin the plot to a variable and do not want to see it interactively, change this to FALSE.
plot_method	string, one of "svg", "png", how to make plot, use svg or png to embed the plot.

`out_format` string, one of "plot", "html", "dot", "dot_print"

- plot: directly open your viewer or browser of the plot
- html: save the plot to a html file
- dot: save the plot in DOT language, need a dot engine to remake the plot
- dot_print: directly cat the DOT code on console

See `details` section if one wish to generate other output format, such as `jpg` or `png`.

`out_path` string, if the `out_format` is not "plot" or "dot_print", provide a path of where to save the plot.

`show_legend` bool, show plot legend?

`mark_main_branch` bool, color the main branch on the plot?

`rstudio` bool, if you are using Rstudio, open the built-in viewer to see the plot? Default is no, open the browser tab to see it plot. The default viewer is too small to see the full plot clearly, so we recommend to use the browser tab. However, if you are using this plot in Shiny apps, always turn `rstudio = TRUE`.

`in_log` bool, is this plot been made in a SPR log file? If `TRUE` will add links of steps to the corresponding log sections.

`rmarkdown` are you rendering this plot in a Rmarkdown document? default value is "detect", this function will determine based on current R environment, or you can force it to be `TRUE` or `FALSE`.

`verbose` bool, turn on verbose mode will give you more information.

`show_warns` bool, print the warning messages on the plot?.

`plot_ctr` bool, add the plot control panel to the plot? This requires you to have internet connection. It will download some additional javascript libraries, and allow you to save the plot as png, jpg, svg, pdf or graphviz directly from the browser.

`pan_zoom` bool, allow panning and zooming of the plot? Use mouse wheel or touch pad to zoom in and out of the plot. You need to have internet connection, additional javascript libraries will be loaded automatically online. Cannot be used with `responsive = TRUE` together. If both `TRUE`, `responsive` will be automatically set to `FALSE`.

`exit_point` numeric, for advanced debugging only, see details

`outputId` string, shiny output ID

`expr` An expression that generates a plotwf, like `plotWF(sal)`

`env` The environment in which to evaluate `expr`.

`quoted` Is `expr` a quoted expression (with `quote()`)? This is useful if you want to save an expression in a variable.

Details

layout:

- compact: try to plot steps as close as possible.

- vertical: main branch will be placed vertically and side branches will be placed on the same horizontal level and sub steps of side branches will be placed vertically.
- horizontal: main branch is placed horizontally and side branches and sub steps will be placed vertically.
- execution: a linear plot to show the workflow execution order of all steps.

exit_point:

return intermediate results at different points and exit the function

- 0: no early exit
- 1: after all branches are found, return tree
- 2: after the new tree has been built, return new nodes
- 3: after dot translation, return graph string

Rmarkdown:

Rmarkdown will change some of the format and cause conflicts. If the plot can be rendered outside Rmd but cannot within Rmd, try to turn this option on. Some additional javascript processing will be performed to avoid the conflict but may cause unknown issues.

Other output formats:

The plot rendering uses `htmlwidgets`, which generates an interactive HTML page. Saving these plots directly to standard image files, such as `png`, is not possible. However, a few workarounds exist to save to these image formats:

- 1: use `webshot2::webshot` function.
- 2: use the interactive panel located on the top-left corner to download as an image after the plot is rendered.
- 3: use `plotWF(sal, plot_method = "png")` to embed the plot as `png`, and then right-click to save the image.

Please see our website for examples: https://systempipe.org/sp/spr/sp_run/step_vis/

Shiny:

When the plot is rendered in a Shiny app, the `rstudio` option must be turned on, `plotWF(sal, rstudio = TRUE, ...)`.

Value

see `out_format` and `exit_point`

`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 = "ATG
```

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(txdbmaker); 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 = NULL,
                FileName1 = NULL, FileName2 = NULL,
                outfile1 = NULL, outfile2 = NULL,
                Fct, batchsize = 100000, overwrite = TRUE, ...)
```

Arguments

args	Object of class SYSargs or SYSargs2.
FileName1	Path to input forward fastq file. Defaul is NULL.
FileName2	Path to input reverse fastq file. Defaul is NULL
outfile1	Path to output forward fastq file. Defaul is NULL
outfile2	Path to output reverse fastq file. Defaul is NULL
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
dir_path <- system.file("extdata/cwl/preprocessReads/trim-se", package="systemPipeR")
targetspath <- system.file("extdata", "targets.txt", package="systemPipeR")
trim <- loadWorkflow(targets=targetspath, wf_file="trim-se.cwl", input_file="trim-se.yml", dir_path=dir_path)
trim <- renderWF(trim, inputvars=c(FileName="_FASTQ_PATH1_", SampleName="_SampleName_"))
## Not run:
preprocessReads(args=trim[1], Fct="trimLRPatterns(Rpattern='GCCCGGGTAA', subject=fq)", batchsize=100000, overwri

## End(Not run)

## Preprocessing of paired-end reads
dir_path <- system.file("extdata/cwl/preprocessReads/trim-pe", package="systemPipeR")
targetspath <- system.file("extdata", "targetsPE.txt", package="systemPipeR")
trim <- loadWorkflow(targets=targetspath, wf_file="trim-pe.cwl", input_file="trim-pe.yml", dir_path=dir_path)
trim <- renderWF(trim, inputvars=c(FileName1="_FASTQ_PATH1_", FileName2="_FASTQ_PATH2_", SampleName="_SampleName
trim
## Not run:
preprocessReads(args=trim[1], Fct="trimLRPatterns(Rpattern='GCCCGGGTAA', subject=fq)", batchsize=100000, overwri

## End(Not run)
```

printParam *Accessories function to modify the Command-line Version 1*

Description

Accessories function to modify the Command-line Version 1

Usage

```
printParam(sysargs, position, index = NULL)
subsetParam(sysargs, position, index = NULL, trim = TRUE, mute = FALSE)
replaceParam(sysargs, position, index = NULL, replace, mute = FALSE)
renameParam(sysargs, position, index = FALSE, rename, mute = FALSE)
appendParam(sysargs, position, index = NULL, append, after, mute = FALSE)
```

Arguments

sysargs	Object of class SYSargs2. Output from the createParamFiles function.
position	string, one of baseCommand, inputs, outputs to view or apply a modification.
index	numeric or character vector, index to view or change a single item in baseCommand, inputs, outputs.
trim	logical, only keep arguments specified by index. Default is "TRUE".
replace	named list, replace arguments in different positions. Replace list length must be the same as index. Different positions will have different requirements.
rename	character vector, rename items in different positions. rename vector length must be the same as index.
append	named list, same requirements as replace, however it cannot append baseCommand.
after	a subscript, after which the values are to be appended. If NULL will be after the last argument or specify a numeric integer.
mute	logical, print the raw command-line string and output after replacing or rename.

Details

- printParam: prints its arguments defined by position and index.

- subsetParam: returns subsets of command-line, keeping the arguments defined by position and index.

- replaceParam: replaces the values in command-line with indices given in list by those given in values

- renameParam: rename the names of the arguments.

- appendParam: Add arguments to the original command line.

Value

SYSargs2 object

Author(s)

Le Zhang and Daniela Cassol

References

For more details on CWL, please consult the following page: https://www.commonwl.org/

See Also

writeParamFiles createParamFiles loadWorkflow renderWF showClass("SYSargs2")

Examples

```
command <- "
hisat2 \
    -S <F, out: ./results/M1A.sam> \
    -x <F: ./data/tair10.fasta> \
    -k <int: 1> \
    -min-intronlen <int: 30> \
    -max-intronlen <int: 3000> \
    -threads <int: 4> \
    -U <F: ./data/SRR446027_1.fastq.gz> \
    --verbose
"
cmd <- createParamFiles(command)
cmdlist(cmd)
```

---

printParam2 *Accessories function to modify the Command-line Version 2*

Description

Accessories function to modify the Command-line Version 2

Usage

```
printParam2(sysargs, base = FALSE, args = FALSE, inputs = FALSE,
        outputs = FALSE, stdout = FALSE, raw_cmd = FALSE, all = TRUE)
appendParam2(sysargs, x, position = c("inputs", "args", "outputs"),
        after = NULL, verbose = FALSE)
replaceParam2(sysargs, x, index=NULL,
             position = c("inputs", "baseCommand", "args", "outputs", "stdout"),
              verbose = FALSE)
removeParam2(sysargs, index=NULL, position = c("inputs", "args", "outputs", "stdout"),
             verbose = FALSE)
renameParam2(sysargs, index=NULL, new_names,
           position = c("inputs", "args", "outputs", "stdout"), verbose = FALSE)
```

Arguments

sysargs	Object of class SYSargs2. Output from the createParamFiles function.
base	logical, print out base command information?
args	logical, print out arguments information?
inputs	logical, print out inputs information?
outputs	logical, print out outputs information?
stdout	logical, print out stdout information?
raw_cmd	logical, print out parsed raw command information?
all	logical, print out all base command, arguments, inputs, outputs, and raw command information? Turn this to TRUE will overwrite all above to TRUE. If you need to print out selected positions, turn this to FALSE and turn other positions to TRUE.
position	string, one of the positions to apply a modification. For appendParam2: "inputs", "args", "outputs", for replaceParam2: "inputs", "baseCommand", "args", "outputs", "stdout", for removeParam2, renameParam2: "inputs", "args", "outputs", "stdout".
index	numeric or character vector, index of items to remove or rename item(s) inside the position you choose, in removeParam2, renameParam2, replaceParam2.
after	integer, in appendParam2, after which current item you want to append the new items in the position you have selected? For example, if you want the new item to be the first, then use 0, if last, then use 999 to make sure it goes to the last one.
x	named list or string, new items to replace or append in different positions. Replace list length must be the same as index. Different positions will have different requirements. See details of the x format requirements.
new_names	character vector, new names that you wish to replace the old names. new_names vector length must be the same as index in renameParam2.
verbose	logical, show addtional information during/after operation? for example, print the new changes.

Details

- printParam2: prints its arguments defined by position and index.

- removeParam2: removes items in certain positions you select.

- replaceParam2: replaces the values in command-line with indices given in list by those given in values

- renameParam2: rename the names of items in certain position.

- appendParam2: Add arguments to the original command line. Adding new basecommand or standard out is not allowed.

x format: - If x is a character, it requires exact 3 semi-colons ; to separate the string in to 4 columns. Values before the third column is the same as createParam inputs, first column:

prefix/argument name, second column: type, third column: default value. The fourth column (new): numeric, index of the new item, this will be translated into `position` entries in CWL.

- If `x` is a list, it must be named. Following items must be included in list: `preF`, `type`, `value`, `index`. They refer to prefix, param type, default value, and position index correspondingly.

Value

`SYSargs2` object

Author(s)

Le Zhang and Daniela Cassol

References

For more details on `CWL`, please consult the following page: https://www.commonwl.org/

See Also

`writeParamFiles` `createParamFiles` `loadWorkflow` `renderWF` `showClass("SYSargs2")`

Examples

```
command2 <- '
mycmd2 \
    p: -s; File; sample1.txt \
    p: -s; File; sample2.txt \
    p: --c; ; \
    p: -o; File; out: myout.txt \
    ref_genome; File; a.fasta \
    p: --nn; int; 12 \
    mystdout; File; stdout: abc.txt
'
cmd2 <- createParam(command2, syntaxVersion = "v2", writeParamFiles=FALSE)
# string format
new_cmd <- 'p: -abc; string; abc; 7'
cmd2 <- appendParam2(cmd2, x = new_cmd, position = "inputs")
printParam2(cmd2, all = FALSE, inputs = TRUE, raw_cmd = TRUE)
# list format
new_cmd <- list(name = "new_arg", preF = "--foo", index = "8")
cmd2 <- appendParam2(cmd2, x = new_cmd, position = "args")
printParam2(cmd2, all = FALSE, args = TRUE, raw_cmd = TRUE)
# rename
cmd2 <- renameParam2(cmd2, "new_name_arg", index = "new_arg", position = "args")
printParam2(cmd2, all = FALSE, args = TRUE, raw_cmd = TRUE)
# remove
cmd2 <- removeParam2(cmd2, index = "new_name_arg", position = "args")
printParam2(cmd2, all = FALSE, args = TRUE, raw_cmd = TRUE)
```

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 or SYSargs2 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 SYSargs2 object
targets <- system.file("extdata", "targets.txt", package="systemPipeR")
dir_path <- system.file("extdata/cwl", package="systemPipeR")
args <- loadWorkflow(targets=targets, wf_file="hisat2/hisat2-mapping-se.cwl",
                  input_file="hisat2/hisat2-mapping-se.yml", dir_path=dir_path)
args <- renderWF(args, inputvars=c(FileName="_FASTQ_PATH1_", SampleName="_SampleName_"))
args
readComp(args, format = "vector", delim = "-")
```

renderLogs *Render RMarkdown Logs Report*

Description

Render the logs report file to the specified output format using pandoc.

Usage

```
renderLogs(sysargs, type = c("html_document", "pdf_document"),
            fileName = "default", quiet = FALSE,
            open_file = TRUE)
```

Arguments

sysargs	object of class SYSargsList.
type	The R Markdown output format to convert to. The option can be the name of a format (e.g. "pdf_document" or "html_document").
fileName	character string naming a file output. Default is "logs_<date>.Rmd".
quiet	If set to TRUE, all messages returned by the function will be suppressed.
open_file	Default is TRUE.

Value

It will return an SYSargsList updated.

Author(s)

Daniela Cassol

See Also

See also as SYSargsList-class.

Examples

```
## Construct SYSargsList object from Rmd file
sal <- SPRproject(overwrite=TRUE)
targetspath <- system.file("extdata/cwl/example/targets_example.txt", package="systemPipeR")

## Constructor and `appendStep<-`
appendStep(sal) <- SYSargsList(step_name = "echo",
                               targets=targetspath, dir=TRUE,
                     wf_file="example/workflow_example.cwl", input_file="example/example.yml",
                            dir_path = system.file("extdata/cwl", package="systemPipeR"),
                            inputvars = c(Message = "_STRING_", SampleName = "_SAMPLE_"))
appendStep(sal) <- LineWise(code = {
```

```
                    hello <- lapply(getColumn(sal, step=1, 'outfiles'), function(x) yaml::read_yaml(x))
                             },
                             step_name = "R_read",
                             dependency = "echo")
sal <- runWF(sal)
sal <- renderLogs(sal, open_file = FALSE)
```

---

renderReport *Render RMarkdown Report*

Description

Render the technical report file to the specified output format using pandoc.

Usage

```
renderReport(sysargs, fileName ="SPR_Report",
             rmd_title = "SPR workflow Template - Report",
             rmd_author = "Author",
             rmd_date= "Last update: `r format(Sys.time(), '%d %B, %Y')`",
             type = c("html_document"),
             desc = "This is a workflow template.",
             quiet = FALSE, open_file = TRUE)
```

Arguments

sysargs	object of class SYSargsList.
fileName	character string naming a file output. Default is "spr_report.Rmd".
rmd_title	string, title of the Rmd.
rmd_author	string, author(s) of the Rmd, put all authors in a single character string.
rmd_date	string, date header of Rmd.
type	The R Markdown output format to convert to. The option can be the name of a format (e.g. "pdf_document" or "html_document").
desc	string, or character vector of strings, some description text in format Rmarkdown that will be added to the document before the workflow steps start. It can be a single line or multiple lines by providing a character vector, each item is one line.
quiet	If set to TRUE, all messages returned by the function will be suppressed.
open_file	Default is TRUE.

Value

It will return an SYSargsList updated, with the file path location.

Author(s)

Daniela Cassol

See Also

See also as SYSargsList-class.

Examples

```
sal <- SPRproject(overwrite = TRUE)
file_path <- system.file("extdata", "spr_simple_wf.Rmd", package = "systemPipeR")
sal <- importWF(sal, file_path = file_path, verbose = FALSE)
targetspath <- system.file("extdata/cwl/example/targets_example.txt", package = "systemPipeR")
appendStep(sal) <- SYSargsList(step_name = "echo",
                               targets = targetspath, dir = TRUE,
                     wf_file = "example/workflow_example.cwl", input_file = "example/example.yml",
                          dir_path = system.file("extdata/cwl", package = "systemPipeR"),
                            inputvars = c(Message = "_STRING_", SampleName = "_SAMPLE_"))
sal <- renderReport(sal, open_file = FALSE)
```

---

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 and SYSargs2*

Description

Function to execute system parameters specified in SYSargs and SYSargs2 object.

Usage

```
runCommandline(args, runid = "01", make_bam = FALSE, del_sam=TRUE, dir = TRUE,
               dir.name = NULL, force=FALSE, input_targets = NULL, ...)
```

Arguments

args	object of class SYSargs or SYSargs2.
runid	Run identifier used for log file to track system call commands. Default is "01".
make_bam	Auto-detects SAM file outputs and converts them to sorted and indexed BAM files. Default is make_bam=FALSE.
del_sam	This option allows deleting the SAM files created when the make_BAM converts the SAM files to sorted and indexed BAM files. Default is del_sam=TRUE.
dir	This option allows creating an exclusive results folder for each step in the workflow and a sub-folder for each sample defined in the targets file. All the outputs and log files for the particular step will be created in the respective folders. Default is dir=TRUE. Option available only for an object of class SYSargs2.
dir.name	Name of the workflow directory. Default is dir.name=FALSE. Note: This argument is required when the dir=TRUE.
force	Internally, the function checks if the expected output files exist, and it skips the command lines when the respective files exist. If the argument force is set to TRUE, the command line will be executed and the files overwrite. Default is force=FALSE.
input_targets	This option allows selecting which targets file and, by consequence which command line will be executed. Default is NULL, in which all command lines will be executed.
...	Additional arguments to pass on to runCommandline().

Value

Output files, their paths can be obtained with outpaths() from SYSargs container or output() from SYSargs2. In addition, a character vector is returned containing the same paths.

Author(s)

Daniela Cassol and Thomas Girke

Examples

```
########################################
## Examples with \code{SYSargs2} object ##
########################################
## Construct SYSargs2 object from CWl param, CWL input, and targets files
targets <- system.file("extdata", "targets.txt", package="systemPipeR")
dir_path <- system.file("extdata/cwl", package="systemPipeR")
WF <- loadWorkflow(targets=targets, wf_file="hisat2/hisat2-mapping-se.cwl",
                 input_file="hisat2/hisat2-mapping-se.yml", dir_path=dir_path)
WF <- renderWF(WF, inputvars=c(FileName="_FASTQ_PATH1_", SampleName="_SampleName_"))
WF
names(WF); modules(WF); targets(WF)[1]; cmdlist(WF)[1:2]; output(WF)

## Not run:
## Execute SYSargs2 on single machine
WF <- runCommandline(args=WF)

## Execute SYSargs on multiple machines of a compute cluster.
file.copy(system.file("extdata", ".batchtools.conf.R",
            package="systemPipeR"), ".")
file.copy(system.file("extdata", "batchtools.slurm.tmpl",
            package="systemPipeR"), ".")
resources <- list(walltime=120, ntasks=1, ncpus=4, memory=1024)
reg <- clusterRun(WF, FUN = runCommandline,
        more.args = list(args = WF, make_bam = TRUE),
        conffile=".batchtools.conf.R", template="batchtools.slurm.tmpl",
        Njobs=18, runid="01", resourceList=resources)

## Monitor progress of submitted jobs
getStatus(reg=reg)

## Updates the path in the object \code{output(WF)}
WF <- output_update(WF, dir=FALSE, replace=TRUE, extension=c(".sam", ".bam"))

## Alignment stats
read_statsDF <- alignStats(WF)
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)

########################################
## Examples with \code{SYSargs} object ##
########################################
## Construct SYSargs object from param and targets files
param <- system.file("extdata", "hisat2.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.
file.copy(system.file("extdata", ".batchtools.conf.R", package="systemPipeR"), ".")
file.copy(system.file("extdata", "batchtools.slurm.tmpl", package="systemPipeR"), ".")
resources <- list(walltime=120, ntasks=1, ncpus=cores(args), memory=1024)
reg <- clusterRun(args, FUN = runCommandline, conffile=".batchtools.conf.R",
                  template="batchtools.slurm.tmpl", Njobs=18,
                  runid="01", resourceList=resources)

## Monitor progress of submitted jobs
getStatus(reg=reg)
file.exists(outpaths(args))

## 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)
```

---

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, outfiles=NULL, diffFct, targets, cmp, dbrfilter, ...)
```

Arguments

args	An instance of SYSargs or SYSargs2 constructed from a targets file where the first column (targetsin(args) or targets.as.df(targets(args))) contains the paths to the tabular read count data files. Another possibily is named character vector with the paths to the tabular range data files and the elements names should be the sampleID.
outfiles	Default is NULL. When args is an object of named character vector class, outfile argument is required. Named character vector with the paths to the resulting count tables and the elements names should be the sampleID.

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, dbrfilte
writeTargetsout(x=args_diff, file="targets_rundiff.txt", overwrite=TRUE)
```

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

---

runWF *Execute SYSargsList*

Description

Function to execute all the code list specified in SYSargsList object.

Usage

```
runWF(sysargs, steps = NULL, targets = NULL,
                  force = FALSE, saveEnv = TRUE,
                  run_step = "ALL", ignore.dep = FALSE,
                  warning.stop = FALSE, error.stop = TRUE, silent = FALSE, ...)
```

Arguments

sysargs	object of class SYSargsList.
steps	character or numeric. Step name or index. If NULL, all the steps will be executed.
targets	This option allows selecting which targets file and, by consequence which command line will be executed for each SYSargs2 class step. Default is NULL, in which all command lines will be executed.
force	Internally, the option checks if the expected output files exist, and it skips the command lines when the respective files exist. If the argument force is set to TRUE, the command line will be executed and the files overwrite. Default is force=FALSE.
saveEnv	If set to TRUE, the environment will be saved to an RDS file. To check the RDS file location, please use projectInfo(sysargs)[["envir"]].
run_step	character. If the step has "mandatory" or "optional" flag for the execution. When ALL, all the steps will be executed.
ignore.dep	logical. This option allow to igonore the dependency tree, when TRUE.
warning.stop	If set to TRUE, the process will be interrupted when a warning is detected.
error.stop	If set to TRUE, the process will be interrupted when a error is detected.
silent	If set to TRUE, all messages returned by the function will be suppressed.
...	Additional arguments to pass on from runCommandline().

Value

It will return an SYSargsList updated.

Author(s)

Daniela Cassol and Thomas Girke

See Also

See also as SYSargsList-class.

Examples

```
## Construct SYSargsList object from Rmd file
sal <- SPRproject(overwrite=TRUE)
targetspath <- system.file("extdata/cwl/example/targets_example.txt", package="systemPipeR")

## Constructor and `appendStep<-`
appendStep(sal) <- SYSargsList(step_name = "echo",
                               targets=targetspath, dir=TRUE,
                     wf_file="example/workflow_example.cwl", input_file="example/example.yml",
                            dir_path = system.file("extdata/cwl", package="systemPipeR"),
                            inputvars = c(Message = "_STRING_", SampleName = "_SAMPLE_"))
appendStep(sal) <- LineWise(code = {
                   hello <- lapply(getColumn(sal, step=1, 'outfiles'), function(x) yaml::read_yaml(x))
                            },
                            step_name = "R_read",
                            dependency = "echo")
## Not run:
sal <- runWF(sal)

## 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, lfcShrink=FALSE, type="normal")
```

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.
lfcShrink	logiacal. If TRUE adds shrunken log2 fold changes (LFC) to the object.
type	please check DESeq2::lfcShrink() documentation. Available character alternatives: "apeglm"; "ashr"; "normal".

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.char = "#")
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.char = "#")
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
```

sal2bash *Translate SYSargsList back to a bash workflow*

Description

This function takes a SYSargsList object and translate it to an executable bash script, so one can run the workflow without loading SPR or using an R console.

Usage

```
sal2bash(sal, out_dir = ".", bash_path = "/bin/bash", stop_on_error = TRUE)
```

Arguments

sal	SYSargsList object.
out_dir	string, a relative or absolute path to a directory. If the directory does not exist, this function will first try to create it.
bash_path	string, the path to the bash executable program
stop_on_error	bool, should the bash script stop if any error happens in running

Details

out files

1. The main executable bash file will be created to the root of ‘out_dir‘

2. All R steps will be stored as R scripts and along with other supporting files inside a folder called ‘spr_bash‘ under ‘out_dir‘

3. Not all R steps will have an individual file. This function will "collapse" adjacent R steps into one file as much as possible. Namely, if there is no sysArgs steps in between, R steps will be merged into one file, otherwise they will be in different files.

R steps

Similarly as running the workflow in R console, all R steps will share the same environment variables and loaded packages. This is done by loading and saving the R environment into a file ‘spr_wf.RData‘ before and after the R script execution. Therefore, it will be good to keep all R steps bundle together as much as possible to avoid the package and environment loading/saving overhead time.

Initially, this environment only contains the SYSargsList object that was used to create the bash script. Note: the SYSargsList object name will be the same as what you pass to ‘sal2bash‘. If you have ‘sal2bash(my_sal)‘, then in the ‘spr_wf.RData‘ there will be an object called ‘my_sal‘ saved there. It is important to keep using the same name for the SYSargsList object managing the workflow and in workflow running. For example, if you have an R step that requires to query a column from the outfiles of the SAL, ‘getColumn(sal, "FileName1")‘, but you pass ‘my_sal‘ to ‘sal2bash(my_sal)‘, this will cause this R step cannot find the SAL object when run the workflow from bash.

Execution

This way of execution is not able to handle complex dependency graphs. The original step dependencies from SAL object will be ignored, so all steps will be executed in a linear manner. It is recommended to adjust the workflow order before using this function.

Value

no return

Author(s)

Le Zhang and Daniela Cassol

Examples

```
file_path <- system.file("extdata/spr_simple_wf.Rmd", package="systemPipeR")
sal <- SPRproject(overwrite = TRUE)
sal <- importWF(sal, file_path)
sal2bash(sal)
```

---

sal2rmd *Translate SYSargsList back to a workflow template Rmarkdown file*

Description

This function takes a SYSargsList object and translate it to SPR workflow template Rmarkdown format.

Usage

```
sal2rmd(sal, out_path = "spr_template.Rmd", rmd_title = "SPR workflow template",
        rmd_author = "my name",
        rmd_date = "Last update: `r format(Sys.time(), '%d %B, %Y')`",
        rmd_output = "html_document",
        desc = "This is a workflow template.", verbose = TRUE)
```

Arguments

sal	SYSargsList object.
out_path	string, output file name.
rmd_title	string, title of the Rmd.
rmd_author	string, author(s) of the Rmd, put all authors in a single character string.
rmd_date	string, date header of Rmd.
rmd_output	string, output format of Rmd, used in header.
desc	string, or character vector of strings, some description text in format Rmarkdown that will be added to the document before the workflow steps start. It can be a single line or multiple lines by providing a character vector, each item is one line.
verbose	logical. If TRUE will show you more information as the function runs.

Value

no return

Author(s)

Le Zhang and Daniela Cassol

Examples

```
file_path <- system.file("extdata/spr_simple_wf.Rmd", package="systemPipeR")
sal <- SPRproject(overwrite = TRUE)
sal <- importWF(sal, file_path)
sal2rmd(sal)
```

---

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

```
library(IRanges)
## 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(txdbmaker); 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:
targets <- system.file("extdata", "targets.txt", package="systemPipeR")
dir_path <- system.file("extdata/cwl", package="systemPipeR")
args <- loadWorkflow(targets=targets, wf_file="hisat2/hisat2-mapping-se.cwl",
                  input_file="hisat2/hisat2-mapping-se.yml", dir_path=dir_path)
args <- renderWF(args, inputvars=c(FileName="_FASTQ_PATH1_", SampleName="_SampleName_"))
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)
```

---

showDF *Create an HTML table using DT package with fixed columns*

Description

Create an HTML table using DT package with fixed columns

Usage

```
showDF(data, ...)
```

Arguments

data	data object (either a matrix or a data frame).
...	Additional arguments used by dDT::atatable() function.

Value

returns an object of datatables and htmlwidget.

Examples

```
showDF(iris)
```

SPRproject *Workflow Project Initiation*

Description

Function to construct SYSargsList workflow control environment (S4 object). This function creates and checks the directory structure. If the expected directories are not available, it is possible to create those. The project directory default structure expected is:

- SPRproject/
  - data/
  - param/
  - results/

The project working directory names can be modified, but users need to edit the code accordingly.

Usage

```
SPRproject(projPath = getwd(), data = "data", param = "param", results = "results",
                      logs.dir= ".SPRproject", sys.file="SYSargsList.yml",
                      envir = new.env(),
                      restart = FALSE, resume=FALSE,
                      load.envir = FALSE,
                      overwrite = FALSE, silent = FALSE)
```

Arguments

projPath	a character vector of a full project path name. Default is the current path.
data	a character vector of a data directory name. Default is data. This subdirectory in the project stores all the raw data, reference, and annotation files.
param	a character vector of a param directory name. Default is param. This subdirectory in the project stores all the parameter and configuration files.
results	a character vector of a results directory name. Default is results. This subdirectory in the project stores all the analysis results, including but not limited: alignment, variant, and peak files (BAM, VCF, BED); tabular result files; and image/plot files.
logs.dir	a character vector of a logs directory name. Default is .SPRproject.
sys.file	a character vector of SYSargsList file name which will store all the project configuration information. Default is SYSargsList.yml and it will be save in the logs.dir folder.
envir	the environment in which workflow will be evaluated. Default will create a new.env().
restart	if set to TRUE, existing SYSargsList object saved in the logs.dir directory will be used to restart the workflow. This option will delete all the log files.
resume	if set to TRUE, existing SYSargsList object saved in the logs.dir directory will be used to resume the workflow.
load.envir	This argument allows to load the environment and recover all the objects saved during the workflow execution (please check runWF function for more details). This argument can be set as TRUE when restart = TRUE or resume = TRUE are used.
overwrite	if set to TRUE, existing logs.dir directory and all the content, as logs files and SYSargsList file will be removed, and a new and empty SYSargsList object will be created. This option should be used with caution!
silent	if set to TRUE, all messages returned by the function will be suppressed.

Details

If an SYSargsList instance was created before or independent of the project initialization, it is possible to append this instance after the project is created. Please see check appendStep<- function.

Value

SPRproject will return a SYSargsList object.

Author(s)

Daniela Cassol

See Also

See also as SYSargsList-class.

Examples

```
sal <- SPRproject(projPath = tempdir())
sal
```

---

subsetWF *Subsetting SYSargs2 class slots*

Description

Return subsets of character for the input, output or the list of command-line for each workflow step.

Usage

```
subsetWF(args, slot, subset=NULL, index=NULL, delete=FALSE)
```

Arguments

args	object of class SYSargs2.
slot	three options available: type="input" returns input slot from SYSargs2 object; type="output" returns output slot from SYSargs2 object; and type="step" returns all the command-line for each workflow step from SYSargs2 object.
subset	name or numeric position of the values to be subsetting in the slot. If slot="input", the subset are the variables defined in the param.yml file. If slot="step", the subset is the command line defined on the SYSargs2 object for all the steps of the workflow. If slot="output", the subset is the path for the expected output files for all the steps in the workflow. Default is subset=NULL
index	A numeric index positions of the file in SYSargs2 object, slot output. It requires a subset to be defined. Default is index=NULL.
delete	allows to delete a subset of files in the case of slot="output". Default is delete=NULL.

Author(s)

Daniela Cassol and Thomas Girke

See Also

loadWorkflow renderWF

Examples

```
## Construct SYSargs2 object
targets <- system.file("extdata", "targets.txt", package="systemPipeR")
dir_path <- system.file("extdata/cwl", package="systemPipeR")
WF <- loadWorkflow(targets=targets, wf_file="hisat2/hisat2-mapping-se.cwl",
                   input_file="hisat2/hisat2-mapping-se.yml", dir_path=dir_path)
WF <- renderWF(WF, inputvars=c(FileName="_FASTQ_PATH1_", SampleName="_SampleName_"))
WF

## Testing subset_wf function
input <- subsetWF(WF, slot="input", subset='FileName')
output <- subsetWF(WF, slot="output", subset=1, index=1)
step.cmd <- subsetWF(WF, slot="step", subset=1) ## subset all the HISAT2 commandline
# subsetWF(WF, slot="output", subset=1, index=1, delete=TRUE) ## in order to delete the subset files list
```

`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` or `SYSargs2` or `named character vector` with BAM files PATH and the elements names should be the sampleID.
`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 SYSargs2 object from param and targets files
targets <- system.file("extdata", "targets.txt", package="systemPipeR")
dir_path <- system.file("extdata/cwl", package="systemPipeR")
args <- loadWorkflow(targets=targets, wf_file="hisat2/hisat2-mapping-se.cwl",
                  input_file="hisat2/hisat2-mapping-se.yml", dir_path=dir_path)
args <- renderWF(args, inputvars=c(FileName="_FASTQ_PATH1_", SampleName="_SampleName_"))
args

## Not run:
## Run alignments
args <- runCommandline(args, dir = FALSE, make_bam = TRUE)

## Create sym links and URL file for IGV
symLink2bam(sysargs=args, command="ln -s", htmldir=c("~/.html/", "somedir/"), ext=c(".bam", ".bai"), urlbase="htt

## 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", "hisat2.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 tabular 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 combination 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=20:00:00")
qsubRun(appfct="runCommandline(args=args)", appargs=args, qsubargs=qsubargs,
        Nqsubs=1, submitdir="results", package="systemPipeR")

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

## End(Not run)
```

SYSargs2-class *Class* "SYSargs2"

Description

SYSargs2 class stores all the information and instructions needed for processing a set of input files with a specific command-line or a series of command-line within a workflow. The SYSargs2 S4 class object is created from the loadWF and renderWF function, which populates all the command-line for each sample in each step of the particular workflow. Each sample level input/outfile operation uses its own SYSargs2 instance. The output of SYSargs2 define all the expected output files for each step in the workflow, which usually it is the sample input for the next step in an SYSargs2 instance. By chaining several SYSargs2 steps together one can construct complex workflows involving many sample-level input/output file operations with any combination of command-line or R-based software.

Objects from the Class

Objects can be created by calls of the form new("SYSargs2", ...).

Slots

targets: Object of class "list" storing data from each sample from targets file

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

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

wf: Object of class "list" storing data from Workflow CWL parameters file

clt: Object of class "list" storing data from each CommandLineTool substep in the Workflow or the single CommandLineTool CWL parameters file

yamlinput: Object of class "list" storing data from input (*.yml) file

`cmdlist`: Object of class "`list`" storing all command-line used to execute external software

`input`: Object of class "`list`" storing data from each target defined in `inputvars`

`output`: Object of class "`list`" paths to final outputs files

`files`: Object of class "`list`" paths to `input` and `CWL parameters` files

`inputvars`: Object of class "`list`" storing data from each `inputvars`

`cmdToCwl`: Object of class "`list`" storing data from each `cmdToCwl`

`status`: Object of class "`list`" storing data from each `status`

`internal_outfiles`: Object of class "`list`" storing raw data from each `output`

Methods

[Subsetting of class with bracket operator.

[[Subsetting of class with bracket operator.

[[<- Replacement method for "`SYSargs2`" class.

$ Extracting slots elements by name.

length Extracts number of samples.

names Extracts slot names.

show Summary view of `SYSargs2` objects.

coerce `signature(from = "list", to = "SYSargs2")`: `as(list, "SYSargs2")`

coerce `signature(from = "SYSargs2", to = "list")as(SYSargs2, "list")`

coerce `signature(from = "SYSargs2", to = "DataFrame")`: `as(x, "DataFrame")`; for targets slot.

sysargs2 Coerce back to list `as(SYSargs2, "list")`

targets Extract data from `targets` slot.

targetsheader Extracts data from `targetsheader` slot.

modules Extracts data from `modules` slot.

wf Extracts data from `wf` slot.

clt Extracts data from `clt` slot.

yamlinput Extracts data from `yamlinput` slot.

cmdlist Extracts data from `cmdlist` slot.

input Extracts data from `input` slot.

output Extracts data from `cmdlist` slot.

files Extracts data from `files` slot.

inputvars Extracts data from `inputvars` slot.

cmdToCwl Extracts data from `cmdToCwl` slot.

status Extracts data from `status` slot.

infile1 extracting paths to first FASTQ file.

infile2 extracting paths to second FASTQ file if data is PE.

baseCommand Extracts baseCommand from command-line used to execute external software.

SampleName Extracts all samples names.

yamlinput<- Replacement method for `yamlinput` slot input.

Author(s)

Daniela Cassol and Thomas Girke

See Also

loadWF and renderWF and runCommandline and clusterRun

Examples

```
showClass("SYSargs2")

## Construct SYSargs2 object from CWl param, CWL input, and targets files
targetspath <- system.file("extdata/cwl/example/targets_example.txt", package="systemPipeR")
dir_path <- system.file("extdata/cwl", package="systemPipeR")
WF <- loadWorkflow(targets=targetspath,
                   wf_file="example/workflow_example.cwl",
                   input_file="example/example.yml",
                   dir_path=dir_path)
WF <- renderWF(WF, inputvars=c(Message = "_STRING_", SampleName = "_SAMPLE_"))
WF

## Methods
names(WF)
length(WF)
baseCommand(WF)
SampleName(WF)

## Accessors
targets(WF)
targetsheader(WF)
modules(WF)
yamlinput(WF)
cmdlist(WF)
input(WF)
output(WF)
files(WF)
inputvars(WF)
cmdToCwl(WF)
status(WF)

## The subsetting operator '[' allows to select specific command-line/sample
WF2 <- WF[1:2]

## Not run:
## Execute SYSargs2 on single machine
WF2 <- runCommandline(WF2)

## End(Not run)
## Not run:
## Execute SYSargs2 on multiple machines of a compute cluster. The following
## example uses the conf and template files for the Slurm scheduler. Please
## read the instructions on how to obtain the corresponding files for other schedulers.
```

```
file.copy(system.file("extdata", ".batchtools.conf.R", package="systemPipeR"), ".")
file.copy(system.file("extdata", "batchtools.slurm.tmpl", package="systemPipeR"), ".")
resources <- list(walltime=120, ntasks=1, ncpus=4, memory=1024)
reg <- clusterRun(WF, FUN = runCommandline, conffile=".batchtools.conf.R",
            template="batchtools.slurm.tmpl", Njobs=2, runid="01", resourceList=resources)

## Monitor progress of submitted jobs
getStatus(reg=reg)


## End(Not run)
```

SYSargsList *SYSargsList Constructor*

Description

SYSargsList instances are constructed by the SYSargsList function.

Usage

```
SYSargsList(sysargs = NULL, step_name = "default",
            targets = NULL, wf_file = NULL, input_file = NULL, dir_path = ".",
            id = "SampleName",
            inputvars = NULL, rm_targets_col = NULL,
            dir = TRUE, dependency = NA,
            run_step = "mandatory",
            run_session = "management",
            run_remote_resources = NULL,
            silent = FALSE,
            projPath = getOption("projPath", getwd()))
```

Arguments

sysargs	SYSargs2 object. If the object already exists, it can be used to construct the SYSargsList object.
step_name	character with the step index name.
targets	the path to targets file. The targets file can be either a simple tabular or yml file. Also, it is possible to assign NULL to run the pipeline without the 'targets' file. This can be useful for running specific workflows that do not require input files.
wf_file	name and path to CWL parameters file.
input_file	name and path to input parameters file.
dir_path	full path to the directory with the CWL parameters and input files.
id	A column from targets file, which will be used as an id for each one of the samples. It is required to be unique.

inputvars	Each vector element is required to be defined in the input file, and the names of the elements are needed to match the column names defined in the targets file.
rm_targets_col	targets file colunms to be removed.
dir	This option allows creating an exclusive results folder for each step in the workflow. All the outfiles and log files for the particular step will be created in the respective folders. Default is dir=TRUE.
dependency	character. Dependency tree, required when appending this step to the workflow. Character name of a previous step in the workflow. Default is NA.
run_step	character. If the step has "mandatory" or "optional" flag for the execution.
run_session	character. If the step has "management" or "compute" flag for the execution.
run_remote_resources	List for reserving for each cluster job sufficient computing resources including memory (Megabyte), number of nodes (Int), CPU cores, walltime (Minutes), etc. It is necessary two additional files: conffile and template. conffile is the path to configuration file (default location ./.batchtools.conf.R). This file contains in its simplest form just one command, such as this line for the scheduler: cluster.functions <- makeClusterFunctionsSlurm(template="batchtools.slurm.tm For more detailed information visit this page: https://mllg.github.io/batchtools/index.html template The template files for a specific queueing/scheduling systems can be downloaded from here: https://github.com/mllg/batchtools/tree/master/inst/templates. Slurm, PBS/Torque, and Sun Grid Engine (SGE) templates are provided within the package demo data.
silent	If set to TRUE, all messages returned by the function will be suppressed.
projPath	a character vector of a full project path name. Default is the current path.

Author(s)

Daniela Cassol

See Also

SYSargs2, LineWise, and SPRproject

Examples

```
sal <- SPRproject(overwrite=TRUE)
targetspath <- system.file("extdata/cwl/example/targets_example.txt",
                           package="systemPipeR")

## Constructor and `appendStep<-`
appendStep(sal) <- SYSargsList(step_name = "echo",
                               targets=targetspath, dir=TRUE,
                               wf_file="example/workflow_example.cwl",
                               input_file="example/example.yml",
                               dir_path = system.file("extdata/cwl",
                               package="systemPipeR"),
                               inputvars = c(Message = "_STRING_",
```

```
                                SampleName = "_SAMPLE_"))

appendStep(sal) <- LineWise(code = {
                   hello <- lapply(getColumn(sal, step=1, 'outfiles'), function(x) yaml::read_yaml(x))
                            },
                            step_name = "R_read",
                            dependency = "echo")

sal
```

SYSargsList-class *Class* "SYSargsList"

Description

SYSargsList S4 class is a list-like container where each instance stores all the input/output paths and parameter components required for a particular data analysis step based on command-line- or R-based software.

SYSargsList instances are constructed by the SYSargsList function.

Usage

```
## Accessors

stepsWF(x)
statusWF(x)
targetsWF(x)
outfiles(x)
SE(x, ...)
dependency(x)
projectInfo(x)
runInfo(x)

## Methods

cmdlist(x, ...)
codeLine(x, ...)
SampleName(x, ...)
stepName(x)
baseCommand(x, ...)
targetsheader(x, ...)
yamlinput(x, ...)
viewEnvir(x, silent = FALSE)
copyEnvir(x, list = character(), new.env = globalenv(), silent = FALSE)
addResources(x, step, resources)

## Subset Methods
```

```
subset(x, ...)
getColumn(x, step, position = c("outfiles", "targetsWF"), column = 1,
          names = SampleName(x, step))

## Replacement

appendStep(x, after = length(x), ...) <- value
yamlinput(x, paramName, ...) <- value
replaceStep(x, step, step_name = "default") <- value
renameStep(x, step, ...) <- value
dependency(x, step, ...) <- value
appendCodeLine(x, after = length(x), ...) <- value
replaceCodeLine(x, line, ...) <- value
updateColumn(x, step, position = c("outfiles", "targetsWF")) <- value
```

Arguments

x	An instance of class SYSargsList.
step	character or numeric. Workflow step name or position index.
silent	If set to TRUE, all messages returned by the function will be suppressed.
list	a character vector naming objects to be copyied from the enviroment.
new.env	An environment to copy to. Default is globalenv().
resources	List for reserving for each cluster job sufficient computing resources including memory (Megabyte), number of nodes (Int), CPU cores, walltime (Minutes), etc. It is necessary two additional files: conffile and template. conffile is the path to configuration file (default location ./.batchtools.conf.R). This file contains in its simplest form just one command, such as this line for the scheduler: cluster.functions <- makeClusterFunctionsSlurm(template="batchtools.slurm.tm For more detailed information visit this page: https://mllg.github.io/batchtools/index.html. template The template files for a specific queueing/scheduling systems can be downloaded from here: https://github.com/mllg/batchtools/tree/master/inst/templates. Slurm, PBS/Torque, and Sun Grid Engine (SGE) templates are provided within the package demo data.
position	character. Options are "outfiles" or "targetsWF" slots.
column	character or numeric. Which column will be subset from the position argument.
names	character vector. Names of the workflow step.
after	A subscript, after which the values are to be appended.
paramName	character. Input name from yamlinput(x), which value should be replaced.
step_name	character with the new step name. Default value will automatically give a name: step_<step index position>.
line	numeric. Index position of the code line to be added or replaced.

value — object containing the values to be replaced to SYSargsList. Values may be of the same class as the orginal values. For updateColumn<- a DataFrame must have the same rows as the original DataFrame. However, if there is no column/rows in the original DataFrame, the new DataFrame will replace the empty one. If there is a non-empty DataFrame, any existing columns with the same name as this new DataFrame will be replaced. Any columns that do not exist in the original DataFrame will be added to the original DataFrame.

... — Further arguments to be passed to or from other methods.

Objects from the Class

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

Slots

stepsWF: Object of class "list" storing all the steps objects of the workflow. Each step can either be SYSargs2 or LineWise.

statusWF: Object of class "list" storing all the success and failure of each step in the workflow.

targetsWF: Object of class "list" storing all the targets DataFrame for each step in the workflow. For the LineWise steps, a DataFrame with 0 rows and 0 columns will be displayed.

outfiles: Object of class "list" storing all the output DataFrame for each step in the workflow. For the LineWise steps, a DataFrame with 0 rows and 0 columns will be displayed.

SE: Object of class "list" storing all the SummarizedExperiment objects in the workflow.

dependency: Object of class "list" storing all the dependency graphs in the workflow.

projectInfo: Object of class "list" storing all the projectInfo information of the workflow.

runInfo: Object of class "list" storing all the runInfo information of each step in the workflow.

targets_connection: Object of class "list" storing all targets files connection in the workflow.

Methods

[signature(x = "SYSargsList", i = "ANY", j = "ANY", drop = "ANY"): subsetting of class with bracket operator

[[signature(x = "SYSargsList", i = "ANY", j = "ANY"): subsetting of class with bracket operator

$ signature(x = "SYSargsList"): extracting slots elements by name

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

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

sysargslist signature(x = "SYSargsList"): Coerce back to list as(SYSargsList, "list")

length signature(x = "SYSargsList"): extracts number of SYSargsList steps

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

show signature(object = "SYSargsList"): summary view of SYSargsList steps

stepsWF signature(x = "SYSargsList"): extract data from stepsWF slot

statusWF signature(x = "SYSargsList"): extract data from statusWF slot

targetsWF `signature(x = "SYSargsList")`: extract data from `targetsWF` slot

outfiles `signature(x = "SYSargsList")`: extract data from `outfiles` slot

SE `signature(x = "SYSargsList")`: extract data from `SE` slot

dependency `signature(x = "SYSargsList")`: extract data from dependency slot

projectInfo `signature(x = "SYSargsList")`: extract data from `projectInfo` slot

runInfo `signature(x = "SYSargsList")`: extract data from `runInfo` slot

cmdlist `signature(x = "SYSargsList", ...)`: extracts data from `cmdlist` slot for each `SYSargs2` step

codeLine `signature(x = "SYSargsList", step)`: extracts data from `codeLine` slot for `LineWise` step

SampleName `signature(x = "SYSargsList", step)`: extracts Sample ID from `SYSargs2` instance step

stepName `signature(x = "SYSargsList")`: extracts steps names from workflow instance

baseCommand `signature(x = "SYSargsList", step)`: extracts `baseCommand` from `SYSargs2` instance step

targetsheader `signature(x = "SYSargsList", step)`: extracts `targetsheader` from `SYSargs2` instance step

yamlinput `signature(x = "SYSargsList", step)`: extracts data from `yamlinput` slot for each `SYSargs2` step

viewEnvir `signature(x = "SYSargsList", silent = FALSE)`: return a vector of character strings giving the names of the objects in the `SYSargsList` environment

copyEnvir `signature(x = "SYSargsList", list = character(), new.env = globalenv(), silent = FALSE)`: copy of the contents or select objects from `SYSargsList` environment and place them into new.env

addResources `signature(x = "SYSargsList", step, resources)`: Adds the computing resources for one or multiple steps in the workflow. If the particular step(s) is set to be executed "management section," when the resources is added, the step(s) will be executed on the "compute section."

getColumn `signature(x = "SYSargsList", step, position = c("outfiles", "targetsWF"), column = 1, names = SampleName(x, step))`: extracts the information for `targetsWF` or `outfiles` slots. The information can be used in an R code downstream

[[<- `signature(x = "SYSargsList", i = "ANY", j = "ANY", value = "ANY")`: replacement method for `SYSargsList` class

appendStep<- `signature(x = "SYSargsList", after = length(x))`: insert the `SYSargsList` or `LineWise` object onto x at the position given by `after`

yamlinput<- `signature(x = "SYSargsList", step, paramName )`: replace a value in the `yamlinput` slot for a specific step instance

replaceStep<- `signature(x = "SYSargsList", step, step_name = "default")`: replace a specific step in the workflow instance

renameStep<- `signature(x = "SYSargsList")`: rename a `stepName` in the workflow instance

dependency<- `signature(x = "SYSargsList", step)`: replace dependency graph for a specific step instance

appendCodeLine<- `signature(x = "SYSargsList", step, after = length(x))`: insert the R code in a specific step at the position given by `after`

replaceCodeLine<- `signature(x = "SYSargsList", step, line)`: replace the R code in a specific step at the position given by `line`

updateColumn<- `signature(x = "SYSargsList", step, position = c("outfiles", "targetsWF"))`: update or add a new column in `targetsWF` or `outfiles` slots

Author(s)

Daniela Cassol and Thomas Girke

See Also

`SYSargs2`, `LineWise`, and `SPRproject`

Examples

```
sal <- SPRproject(overwrite=TRUE)
targetspath <- system.file("extdata/cwl/example/targets_example.txt",
                           package="systemPipeR")

## Constructor and `appendStep<-`
appendStep(sal) <- SYSargsList(step_name = "echo",
                               targets=targetspath, dir=TRUE,
                               wf_file="example/workflow_example.cwl",
                               input_file="example/example.yml",
                               dir_path = system.file("extdata/cwl",
                               package="systemPipeR"),
                               inputvars = c(Message = "_STRING_",
                               SampleName = "_SAMPLE_"))

appendStep(sal) <- LineWise(code = {
                  hello <- lapply(getColumn(sal, step=1, 'outfiles'), function(x) yaml::read_yaml(x))
                           },
                           step_name = "R_read",
                           dependency = "echo")
sal

length(sal)
names(sal)

## Accessors

stepsWF(sal)
statusWF(sal)
targetsWF(sal)
outfiles(sal)
SE(sal)
dependency(sal)
projectInfo(sal)
runInfo(sal)
```

```
## Methods

cmdlist(sal, step=1, targets=1:2) ## SYSargs2 step
codeLine(sal, step=2) ## LineWise step
SampleName(sal, step="echo")
stepName(sal)
baseCommand(sal, 1) ## SYSargs2 step
targetsheader(sal, step=1) ## SYSargs2 step
yamlinput(sal, step=1) ## SYSargs2 step
viewEnvir(sal)
copyEnvir(sal, list = character(), new.env = globalenv())

resources <- list(conffile= system.file("extdata/.batchtools.conf.R",
                            package="systemPipeR"),
                  template= system.file("extdata/batchtools.slurm.tmpl",
                            package="systemPipeR"),
                  Njobs=3, ## Usually, the samples number
                  walltime=60, ## minutes
                  ntasks=1,
                  ncpus=4,
                  memory=1024 ## Mb
)
addResources(sal, 1, resources= resources)

## Subset Methods

sal_sub <- subset(sal, subset_steps=1, input_targets=1:2, keep_steps = TRUE)
sal_sub
targetsIn <- getColumn(sal, step=1, position = c("outfiles"))
targetsIn

## Replacement
renameStep(sal, step=1) <- "new_echo"
dependency(sal, step=2) <- "new_echo"
updateColumn(sal, step=2, position = c("targetsWF")) <- data.frame(targetsIn)
targetsWF(sal)

replaceStep(sal, step=2) <- LineWise(code = {
                            hello <- "Printing a new message"
                            },
                            step_name = "R_hello",
                            dependency = "new_echo")
codeLine(sal)
yamlinput(sal, step=1, paramName="results_path") <- list(results_path=list(
                                        class="Directory", path="./data"))
cmdlist(sal, step = 1, targets = 1)
appendCodeLine(sal, step=2, after = 0) <- "log <- log(10)"
codeLine(sal, 2)
replaceCodeLine(sal, step=2, line=1) <- LineWise(code = {
                            log <- log(50)
                            })
codeLine(sal, 2)
```

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=120, ntasks=1, ncpus=cores(args), memory=1024)
reg <- clusterRun(args, conffile=".batchtools.conf.R", template="batchtools.slurm.tmpl", Njobs=18, runid="01", re

## Monitor progress of submitted jobs
getStatus(reg=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(args)
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)
```

---

targets.as.df *Convert targets to list or data.frame*

Description

Convert targets files to `list` or `data.frame` object.

Usage

```
targets.as.df(x)
targets.as.list(x, id="SampleName")
```

Arguments

`x`	An object of the class `"list"` or `"data.frame"` that stores data from each target file, as `targets(WF)`.
`id`	A column from `targets` file, which will be used as an id for each one of the samples. It is required to be unique.

Value

`data.frame` or `list` containing all the `targets` file information.

Author(s)

Daniela Cassol

See Also

`showClass("SYSargs2")`

Examples

```
targetspath <- system.file("extdata", "targets.txt", package="systemPipeR")
targets <- read.delim(targetspath, comment.char = "#")
targetslist <- targets.as.list(x=targets)
targets.as.df(x=targetslist)
```

---

trimbatch *Genome read coverage by transcript models*

Description

Trims adaptors hierarchically from longest to shortest match from right end of read. ## If 'internalmatch=TRUE' then internal matches will trigger the same behavior. The ## argument minpatternlength defines shortest adaptor match to consider for reads ## containing only partial adaptors at the right end.

Usage

```
trimbatch(fq, pattern, internalmatch=FALSE, minpatternlength=8,
                            Nnumber=1, polyhomo=100, minreadlength=18,
                            maxreadlength)
```

Arguments

`fq`	`character` path to fastq file that contains the target sequences.
`pattern`	`character` pattern used to trim the sequence.
`internalmatch`	The default is `FALSE`. Trims adaptors hierarchically from longest to shortest match from right end of read. If 'internalmatch=TRUE' then internal matches will trigger the same behavior.
`minpatternlength`	It defines shortest adaptor match to consider for reads containing only partial adaptors at the right end.
`Nnumber`	A numeric value representing a minimum criterion for the filter. It selects elements with fewer than `Nnumber` 'N' symbols in each element.

polyhomo	A numeric value representing a maximum criterion for the filter. It selects elements with fewer than threshold copies of any nucleotide.
minreadlength	numeric value representing minimum read length.
maxreadlength	numeric value representing maximun read length.

Author(s)

Thomas Girke

Examples

```
## Preprocessing of paired-end reads
dir_path <- system.file("extdata/cwl/preprocessReads/trim-pe", package="systemPipeR")
targetspath <- system.file("extdata", "targetsPE.txt", package="systemPipeR")
trim <- loadWorkflow(targets=targetspath, wf_file="trim-pe.cwl", input_file="trim-pe.yml", dir_path=dir_path)
trim <- renderWF(trim, inputvars=c(FileName1="_FASTQ_PATH1_", FileName2="_FASTQ_PATH2_", SampleName="_SampleName
trim
## Not run:
iterTrim <- "trimbatch(fq, pattern='ACACGTCT', internalmatch=FALSE, minpatternlength=6, Nnumber=1, polyhomo=50, m
preprocessReads(args=trim[1], Fct=iterTrim, batchsize=100000, overwrite=TRUE, compress=TRUE)

## End(Not run)
```

---

tryCMD *Collect information about the third-party software*

Description

Function to check if third-party software or utility is installed and set in the PATH.

Usage

```
tryCMD(command, silent = FALSE)
```

Arguments

command	a character vector containing the command line name to be tested.
silent	If set to TRUE, all messages returned by the function will be suppressed.

Value

It will return a positive message if the software is set on the PATH or an error message if the software is not set it.

Note

Please note that not necessary the software is not installed if the message indicates an error, but it has not been exported on the current PATH.

Author(s)

Danela Cassol

Examples

```
## Not run:
tryCMD(command="R")
tryCMD(command="blastp")
tryCMD(command="hisat2")

## End(Not run)
```

---

tryPath *Validation of the files or directories*

Description

Function to check if the full path (file or directory) exists.

Usage

```
tryPath(path)
```

Arguments

path	a character vector of full path name.

Details

This function produces a character vector of the file or directory name defined on the path argument.

Value

A character vector containing the name of the file or directory. If the path does not exist, it will return an error message.

Author(s)

Daniela Cassol

Examples

```
file <- system.file("extdata/", "targets.txt", package="systemPipeR")
tryPath(path=file)
```

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(files, txdb, fa, organism, out_dir = "results")

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

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

Arguments

files	named character vector with paths of the input VCF files.
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 r eference 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.
out_dir	Character vector of a results directory name. Default is results.

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:
library(VariantAnnotation)
system("bwa index -a bwtsw ./data/tair10.fasta")
bampaths <- runCommandline(args=args)

## Alignment with BWA (parallelized on compute cluster)
resources <- list(walltime=120, ntasks=1, ncpus=cores(args), memory=1024)
reg <- clusterRun(args, conffile=".batchtools.conf.R", template="batchtools.slurm.tmpl", Njobs=18, runid="01", re

## 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=120, ntasks=1, ncpus=cores(args), memory=1024)
reg <- clusterRun(args, conffile=".batchtools.conf.R", template="batchtools.slurm.tmpl", Njobs=18, runid="01", re
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=120, ntasks=1, ncpus=cores(args), memory=1024)
reg <- clusterRun(args, conffile=".batchtools.conf.R", template="batchtools.slurm.tmpl", Njobs=18, runid="01", re
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)
```

---

writeTargets *Write updated targets out to file from* SYSargsList

---

Description

Convenience write function for generating targets files containing the paths to files generated by input processes. These processes can be commandline- or R-based software. Typically, the paths to the inputs are stored in the targets infile `targetsWF(sysargs)` for `SYSargsList` objects. 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

```
writeTargets(sysargs, step, file = "default", silent = FALSE, overwrite = FALSE)
```

Arguments

sysargs	Object of class SYSargsList.
step	character with the step name. To check all the names, please use `stepName(sysargs)`.
file	Name and path of the output file. If set to "default" then the name of the output file will have the pattern 'targets_<stepName>.txt'.
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.

Value

Writes tabular targes files containing the header/comment lines from `stepsWF(sysargs)[[step]][["targetsheader"]]` and the columns from `targetsWF(sysargs)[[step]]`.

Author(s)

Daniela Cassol

Examples

```
## Construct SYSargsList object from Rmd file
sal <- SPRproject(overwrite=TRUE)
targetspath <- system.file("extdata/cwl/example/targets_example.txt", package="systemPipeR")

## Constructor and `appendStep<-`
appendStep(sal) <- SYSargsList(step_name = "echo",
                               targets=targetspath, dir=TRUE,
                     wf_file="example/workflow_example.cwl", input_file="example/example.yml",
                           dir_path = system.file("extdata/cwl", package="systemPipeR"),
                           inputvars = c(Message = "_STRING_", SampleName = "_SAMPLE_"))
appendStep(sal) <- LineWise(code = {
                    hello <- lapply(getColumn(sal, step=1, 'outfiles'), function(x) yaml::read_yaml(x))
                           },
                           step_name = "R_read",
                           dependency = "echo")
writeTargets(sal, "echo")
```

`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)` for `SYSargs` objects or `targets.as.df(targets(WF))` for `SYSargs2` objects) and the outputs are stored in the targets outfile (`targetsout(args)` for `SYSargs` objects or `output(WF)`) for `SYSargs2` objects. 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, step = NULL, new_col=NULL, new_c
```

Arguments

`x`	Object of class `SYSargs` or `SYSargs2`.
`file`	Name and path of the 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, when `x` is an object of class `SYSargs`. For an object of class `SYSargs2`, the output file will have the pattern 'targets_<software>_<step>.txt', where <software> will be the workflow name (`files(x)$cwl`) and <step> will be the step chosen in the argument `step`.
`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.

step	Name or numeric position of the step in the workflow to write the output files. The names can be check running names(x$clt).
new_col	A vector of character strings of the new column name to add to target file.
new_col_output_index	A vector of numeric index positions of the file in SYSargs2 class output. It requires new_col definition.
remove	To remove the original columns. Default is FALSE.
...	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)

Daniela Cassol and Thomas Girke

See Also

writeTargetsRef

Examples

```
########################################
## Examples with \code{SYSargs2} object ##
########################################
## Construct SYSargs2 object
targets <- system.file("extdata", "targets.txt", package="systemPipeR")
dir_path <- system.file("extdata/cwl", package="systemPipeR")
WF <- loadWorkflow(targets=targets, wf_file="hisat2/hisat2-mapping-se.cwl",
                   input_file="hisat2/hisat2-mapping-se.yml", dir_path=dir_path)
WF <- renderWF(WF, inputvars=c(FileName="_FASTQ_PATH1_", SampleName="_SampleName_"))
WF
## Write targets out file
names(WF$clt)
writeTargetsout(x=WF, file="default", step=1, new_col=c("sam_file"), new_col_output_index=c(1))
```

---

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)
```

---

write_SYSargsList *Writeout SYSargsList object*

Description

Function to writeout SYSargsList workflow control environment (S4 object) object.

Usage

```
write_SYSargsList(sysargs, sys.file=".SPRproject/SYSargsList.yml", silent=TRUE)
```

Arguments

sysargs	object of class SYSargsList.
sys.file	name and path of the SYSargsList file which will store all the project configuration information. Default is .SPRproject/SYSargsList.yml.
silent	if set to TRUE, all messages returned by the function will be suppressed.

Value

write_SYSargsList will return a sys.file path.

Author(s)

Daniela Cassol

See Also

See also as SYSargsList-class.

Examples

```
## Construct SYSargsList object from Rmd file
sal <- SPRproject(overwrite=TRUE)
targetspath <- system.file("extdata/cwl/example/targets_example.txt", package="systemPipeR")

## Constructor and `appendStep<-`
appendStep(sal) <- SYSargsList(step_name = "echo",
                               targets=targetspath, dir=TRUE,
                     wf_file="example/workflow_example.cwl", input_file="example/example.yml",
                            dir_path = system.file("extdata/cwl", package="systemPipeR"),
                            inputvars = c(Message = "_STRING_", SampleName = "_SAMPLE_"))
appendStep(sal) <- LineWise(code = {
                   hello <- lapply(getColumn(sal, step=1, 'outfiles'), function(x) yaml::read_yaml(x))
                            },
                            step_name = "R_read",
                            dependency = "echo")

sal <- write_SYSargsList(sal)
sal
```

Index