Package 'pipeFrame'

October 16, 2019

Type Package

Title Pipeline framework for bioinformatics in R

Version 1.0.0

Author Zheng Wei, Shining Ma

Maintainer Zheng Wei <wzweizheng@qq.com>

Description pipeFrame is an R package for building
a componentized bioinformatics pipeline.
Each step in this pipeline is wrapped in the framework,
so the connection among steps is created seamlessly and automatically.
Users could focus more on fine-tuning arguments rather than spending a
lot of time on transforming file format, passing task outputs to task
inputs or installing the dependencies.
Componentized step elements can be customized into other
new pipelines flexibly as well.
This pipeline can be split into several important functional steps,
so it is much easier for users to understand the complex
arguments from each step
rather than parameter combination from the whole pipeline.
At the same time, componentized pipeline can restart at the
breakpoint and avoid rerunning the whole pipeline,
which may save a lot of time for users on pipeline tuning or
such issues as power off or process other interrupts.

License GPL-3

Encoding UTF-8

LazyData FALSE

Imports BSgenome, digest, visNetwork, magrittr, methods, Biostrings,
GenomeInfoDb, parallel, stats, utils

Suggests BiocManager, knitr, rtracklayer, testthat

RoxygenNote 6.1.1

VignetteBuilder knitr

biocViews Software, Infrastructure, WorkflowStep

URL https://github.com/wzthu/pipeFrame

BugReports https://github.com/wzthu/pipeFrame/issues

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

git_branch RELEASE_3_9

git_last_commit 33c02a1

git_last_commit_date 2019-05-02

Date/Publication 2019-10-15

R topics documented:

`graphMng` *Step graph management*

Description

The step relations are managed and restricted to directed acyclic graph. The direction of data flow is from upstream to downstream. So when users create a new step object, restricting its relation with existing steps is necessary.

Usage

```
addEdges(edges, argOrder)

getPrevSteps(stepName, argOrder)

getNextSteps(stepName, argOrder)

printMap(stepName = NULL, display = TRUE, ...)
```

Arguments

`edges`	`Character` vector. Contain the names of start and end points for all edges. It needs to follow the format like c("startpt1","endpt1","startpt2", "endpt2","startpt3","endpt3").
`argOrder`	`Numeric` scalar. The argument order of the input Step object.
`stepName`	`Character` scalar. Step class name of each step.
`display`	`Logical` scalar. Whether show the picture on device or not.
`...`	Additional arguments, currently used.

Value

addEdges	No value will be returned.
getPrevSteps	Names of previous steps
getNextSteps	Names of next steps
printMap	Print the flow map for the pipeline.

Examples

```
addEdges(edges = c("RandomRegionOnGenome",
                   "OverlappedRandomRegion"),argOrder = 1)
printMap()

getPrevSteps("OverlappedRandomRegion",1)
```

---

initPipeFrame *initialize the pipeFrame package*

Description

This function should be called first in R terminal for general users. And it should be used in .onLoad() function for package developers. In this function, several parameters need to be defined and configured, including genome, job name, reference directory, temporary directory, check and install function, threads number, reference list, etc.

Usage

```
initPipeFrame(defaultJobName, availableGenome = c("hg19", "hg38", "mm9",
  "mm10", "danRer10", "galGal5", "galGal4", "rheMac3", "rheMac8",
  "panTro4", "rn5", "rn6", "sacCer2", "sacCer3", "susScr3", "testgenome"),
  defaultCheckAndInstallFunc = NULL, defaultThreads = 2,
  defaultTmpDir = getwd(), defaultRefDir = file.path(getwd(),
  "refdir"), defaultReference = list(test = list(file = "fileName", rc =
  "obj")))
```

Arguments

defaultJobName	Character scalar. The default job name for the package. When users use pipeFrame package, defaultJobName is "pipeFrame-pipeline".
availableGenome	Character scalar or vector. Configure the available valid genome such as "hg19", "mm10", etc.
defaultCheckAndInstallFunc	Function scalar. The function needs to call runWithFinishCheck
defaultThreads	Numeric scalar. The maximum thread limit for each step. Default:2
defaultTmpDir	Character scalar. The directory of intermediate results for all steps. Default: Current working directory.
defaultRefDir	Character scalar. The directory of reference data. Default: file.path(getwd(),"refdir")
defaultReference	List scalar. List of reference files.

Value

No value will be returned.

Examples

```
initPipeFrame(availableGenome = c("hg19", "hg38", "mm9", "mm10"),
              defaultJobName = paste0("pkgname","-pipeline")
)
```

`runWithFinishCheck` *Install dependent data or software with finishing check*

Description

Install dependent data or software with finishing check

Usage

```
runWithFinishCheck(func, refName, refFilePath = NULL)

checkAndInstallBSgenome(refFilePath, genome = getGenome())

checkAndInstallOrgDb(refFilePath, genome = getGenome())

checkAndInstallTxDb(refFilePath, genome = getGenome())

checkAndInstallGenomeFa(refFilePath)
```

Arguments

`func`	`Function` scalar. The function with refFilePath argument (reference file directory). The returned value will be set as the reference object.
`refName`	`Character` scalar. Reference name for `getRef`, `getRefFiles` and `getRefRc`.
`refFilePath`	`Character` scalar. The reference file relative directory under the "refdir/genome/"
`genome`	`Character` scalar. The genome like "hg19". Default: getGenome()

Value

`runWithFinishCheck`
: No value will be returned

`checkAndInstallBSgenome`
: check if there is the BSgenome package installed for curent genome and install it if not. No value will be returned.

`checkAndInstallOrgDb`
: check if there is the OrgDb package installed for curent genome and install it if not. No value will be returned.

`checkAndInstallTxDb`
: check if there is the TxDb package installed for curent genome and install it if not. Nothing will be returned.

`checkAndInstallGenomeFa`
: check if genome FASTA file exist and install if not. No value will be returned

Examples

```
checkAndInstall <- function(){
   runWithFinishCheck(func = checkAndInstallBSgenome,refName = "bsgenome")
   runWithFinishCheck(func = checkAndInstallGenomeFa,refName = "fasta",
   refFilePath = paste0(getGenome(),".fa"))
}
initPipeFrame(availableGenome = c("hg19", "hg38","mm9","mm10","testgenome"),
              defaultJobName = paste0("pkgname","-pipeline")
)

setGenome("hg19")
```

`setGenome` *Configure genome for all steps*

Description

Configure the reference genome assembly for all steps.

Usage

```
getValidGenome()

setGenome(genome)

getGenome()
```

Arguments

`genome`	`Character` scalar. Valid genome to be configured.

Value

`getValidGenome`	`Character` scalar. All valid genome assemblies for this package.
`setGenome`	All packages and dependencies are configured and installed. No value will be returned.
`getGenome`	`Character` scalar. Display the configured genome.

Examples

```
getValidGenome()
setGenome("hg19")
getGenome()
```

setJobName *Configure the job name for following steps.*

Description

Configure the job name for following steps.

Usage

```
setJobName(jobName)

getJobName()

getJobDir()
```

Arguments

jobName	Character scalar. Job name for following steps.

Value

setJobName	No value will be returned
setJobName	Set a job name for following steps.
setJobName	Set the job directory

Examples

```
setJobName("testJobName")
getJobName()
getJobDir()
```

setRefDir *Set the reference directory*

Description

Set the reference directory

Usage

```
setRefDir(refdir, createDir = TRUE)

getRefDir()

getRef(refName)

getRefFiles(refName)

getRefRc(refName)
```

Arguments

refdir	Character scalar. The directory to store the reference data.
createDir	Logica scalar. Create the directory if the directory does not exist. Default: TRUE
refName	Character scalar. The name of reference data.

Value

setRefDir	No value will be returned
getRefDir	Character scalar. Display the directory of reference data.
getRef	List scalar. A list object which contains "files" (reference file paths) and "rc" (reference R object)
getRefFiles	Character scalar or vector. Display the reference file directory.
getRefRc	Uncertain scalar or vector. Display any reference R object.

Examples

```
setRefDir("./refdir")
getRefDir()
getRef("test")

getRefFiles("test")
getRefRc("test")
```

---

setThreads *Configure the maximum number of threads*

Description

Configure the maximum number of threads for all steps

Usage

```
setThreads(threads = detectCores())

getThreads()
```

Arguments

threads	Numeric scalar. The maximum number of threads that can be allocated to each step.

Value

setThreads	No value will be returned
getThreads	Numeric scalar. Display the maximum number of threads that can be allocated to each step.

Examples

```
setThreads()
getThreads()
```

`setTmpDir` *Configure the directory for intermediate results of all steps*

Description

Configure the directory for intermediate results of all steps

Usage

```
setTmpDir(tmpDir = getwd())

getTmpDir()
```

Arguments

`tmpDir`	`Character` scalar. The directory to store intermediate results of all steps. Default: Current directory.

Value

`setTmpDir`	No value will be returned
`getTmpDir`	`Character` scalar. Display the directory for intermediate results of all steps.

Examples

```
setTmpDir()
getTmpDir()
```

`Step-class` *Methods for Step objects*

Description

Users can call Step object operation methods below to obtain information in objects.

Usage

```
## S4 method for signature 'Step'
init(.Object, prevSteps = list(), ...)

## S4 method for signature 'Step'
getStepName(.Object, ...)

## S4 method for signature 'Step'
getDefName(.Object, ...)
```

```
## S4 method for signature 'Step'
input(.Object)

## S4 replacement method for signature 'Step'
input(.Object) <- value

## S4 method for signature 'Step'
output(.Object)

## S4 replacement method for signature 'Step'
output(.Object) <- value

## S4 method for signature 'Step'
param(.Object)

## S4 replacement method for signature 'Step'
param(.Object) <- value

## S4 method for signature 'Step'
property(.Object)

## S4 replacement method for signature 'Step'
property(.Object) <- value

## S4 method for signature 'Step'
report(.Object)

## S4 replacement method for signature 'Step'
report(.Object) <- value

## S4 method for signature 'Step'
argv(.Object)

## S4 replacement method for signature 'Step'
argv(.Object) <- value

## S4 method for signature 'Step'
x$name

## S4 method for signature 'Step'
getParam(.Object, item, type = c("input", "output",
  "other"), ...)

## S4 method for signature 'Step'
getParamItems(.Object, type = c("input", "output",
  "other"), ...)

## S4 method for signature 'Step'
isReady(.Object, ...)

## S4 method for signature 'Step'
clearStepCache(.Object, ...)
```

```
## S4 method for signature 'Step'
getReportVal(.Object, item, ...)

## S4 method for signature 'Step'
getReportItems(.Object, ...)

## S4 method for signature 'Step'
getAutoPath(.Object, originPath, regexSuffixName, suffix,
  ...)

## S4 method for signature 'Step'
checkRequireParam(.Object, ...)

## S4 method for signature 'Step'
checkAllPath(.Object, ...)

## S4 method for signature 'Step'
getParamMD5Path(.Object, ...)

## S4 method for signature 'Step'
getStepWorkDir(.Object, filename = NULL, ...)

## S4 method for signature 'Step'
getStepId(.Object, ...)

## S4 method for signature 'Step'
writeLog(.Object, msg, ..., isWarnning = FALSE,
  appendLog = TRUE, showMsg = TRUE)

processing(.Object, ...)

## S4 method for signature 'Step'
getReportValImp(.Object, item, ...)

## S4 method for signature 'Step'
getReportItemsImp(.Object, item, ...)
```

Arguments

.Object	Step object scalar. Step object is returned by functions in each step.
prevSteps	List scalar of Step object
...	Additional arguments, currently unused.
value	any type scalar. The value to be set for corresponding item in a list.
x	Step object scalar. Step object is returned by functions in each step.
name	Character scalar. Name can be one of inputList, outputList, paramList, allList, propList or the item names of inputList, outputList or paramList
item	Character scalar. The items in parameter list (input, output and other) or report list.
type	Character scalar. Valid types of parameters including "input", "output" and "other"

originPath	Character scalar. The file name for output file is based on this original path name.
regexSuffixName	Character scalar. The suffix for replacement.
suffix	Character scalar. The new suffix for the file.
filename	Character scaler. The name of file under step working directiory
msg	Character scalar. The message to write into log file.
isWarnning	Logical scalar. Set this message as warning message. Default: FALSE
appendLog	Logical scalar. Append to the log file. Default: TRUE
showMsg	Logical scalar. Show the message on screen. Default: TRUE

Details

Step is a S4 class for generating Step S4 objects. All Step objects generated by child classes inherit from Step. To generate new Step objects, a function wrapper with fixed arguments needs to be implemented. Users use this function to generate new Step functions rather than Step S4 class to generate objects.

Value

the function and result of functions:

init	(For package developer only) A Step child class object with initialized input, output and other parameters
getStepName	get Step object Character name
getDefName	get Step object Character name
input	get input list
input<-	set input list
output	get output list
output<-	set output list
param	get other parameters list
param<-	set other parameters list
property	get property list
property<-	set property list
report	get report list
report<-	set report list
argv	get arguments list
argv<-	set argumnets list
$	get inputList, outputList, paramList, allList, propList or any item value in inputList, outputList or paramList
getParam	Get parameter value set by process function. See getParamItems to obtain valid items for query.
getParamItems	Get parameter name list
isReady	Is the process ready for downstream process
clearStepCache	Clear cache of Step object

`getReportVal`	Get report value of item. See `getReportItems` to obtain valid items for query.
`getReportItems`	Get all items that can be reported
`getAutoPath`	(For package developer) Developer can use this method to generate new file name based on exist input file name
`checkRequireParam`	(For package developer) Check required inputs or parameters are filled.
`checkRequireParam`	(For package developer) Check required inputs are filled.
`getParamMD5Path`	The Step object storage directory
`getStepWorkDir`	Get the step work directory of this object
`getStepId`	Get the step ID
`writeLog`	(For package developer) write log.
`processing`	(For package developer) Run pipeline step
`getReportValImp`	(For package developer) get Report Value
`getReportItemsImp`	(For package developer) getReportItemsImp

Author(s)

Zheng Wei

See Also

`setGenome` `setThreads`

Examples

```
library(BSgenome)
library(rtracklayer)
library(magrittr)

# generate new Step : RandomRegionOnGenome
setClass(Class = "RandomRegionOnGenome",
         contains = "Step"
)

setMethod(
    f = "init",
    signature = "RandomRegionOnGenome",
    definition = function(.Object,prevSteps = list(),...){
        # All arguments in function randomRegionOnGenome
        # will be passed from "..."
        # so get the arguments from "..." first.
        allparam <- list(...)
        sampleNumb <- allparam[["sampleNumb"]]
        regionLen <- allparam[["regionLen"]]
        genome <- allparam[["genome"]]
        outputBed <- allparam[["outputBed"]]
```

```
        # no previous steps for this step so ingnore the "prevSteps"
        # begin to set input parameters
        # no input for this step
        # begin to set output parameters
        if(is.null(outputBed)){
            output(.Object)$outputBed <-
                getStepWorkDir(.Object,"random.bed")
        }else{
            output(.Object)$outputBed <- outputBed
        }
        # begin to set other parameters
        param(.Object)$regionLen <-  regionLen
        param(.Object)$sampleNumb <- sampleNumb
        if(is.null(genome)){
            param(.Object)$bsgenome <-  getBSgenome(getGenome())
        }else{
            param(.Object)$bsgenome <-  getBSgenome(genome)
        }
        # don't forget to return .Object
        .Object
    }
)

setMethod(
    f = "processing",
    signature = "RandomRegionOnGenome",
    definition = function(.Object,...){
        # All arguments are set in .Object
        # so we can get them from .Object
        allparam <- list(...)
        sampleNumb <- getParam(.Object,"sampleNumb")
        regionLen <- getParam(.Object,"regionLen")
        bsgenome <- getParam(.Object,"bsgenome")
        outputBed <- getParam(.Object,"outputBed")
        # begin the calculation
        chrlens <-seqlengths(bsgenome)
        selchr <- grep("_|M",names(chrlens),invert=TRUE)
        chrlens <- chrlens[selchr]
        startchrlens <- chrlens - regionLen
        spchrs <- sample(x = names(startchrlens),
        size =  sampleNumb, replace = TRUE,
        prob = startchrlens / sum(startchrlens))
        gr <- GRanges()
        for(chr in names(startchrlens)){
            startpt <- sample(x = 1:startchrlens[chr],
            size = sum(spchrs == chr),replace = FALSE)
            gr <- c(gr,GRanges(seqnames = chr,
            ranges = IRanges(start = startpt, width = 1000)))
        }
        result <- sort(gr,ignore.strand=TRUE)
        rtracklayer::export.bed(object = result, con =  outputBed)
        # don't forget to return .Object
        .Object
    }
)

# This function is exported in NAMESPACE for user to use
```

```
randomRegionOnGenome <- function(sampleNumb, regionLen = 1000,
                                 genome = NULL, outputBed = NULL, ...){
    allpara <- c(list(Class = "RandomRegionOnGenome", prevSteps = list()),
                 as.list(environment()),list(...))
    step <- do.call(new,allpara)
    invisible(step)
}


# generate another new Step : OverlappedRandomRegion
setClass(Class = "OverlappedRandomRegion",
         contains = "Step"
)

setMethod(
    f = "init",
    signature = "OverlappedRandomRegion",
    definition = function(.Object,prevSteps = list(),...){
        # All arguments in function overlappedRandomRegion and
        # runOerlappedRandomRegion will be passed from "..."
        # so get the arguments from "..." first.
        allparam <- list(...)
        inputBed <- allparam[["inputBed"]]
        randomBed <- allparam[["randomBed"]]
        outputBed <- allparam[["outputBed"]]
        # inputBed can obtain from previous step object when running
        # runOerlappedRandomRegion
        if(length(prevSteps)>0){
            prevStep <- prevSteps[[1]]
            input(.Object)$randomBed <-  getParam(prevStep,"outputBed")
        }
        # begin to set input parameters
        if(!is.null(inputBed)){
            input(.Object)$inputBed <- inputBed
        }
        if(!is.null(randomBed)){
            input(.Object)$randomBed <- randomBed
        }
        # begin to set output parameters
        # the output is recemended to set under the step work directory
        if(!is.null(outputBed)){
            output(.Object)$outputBed <-  outputBed
        }else{
            output(.Object)$outputBed <-
                getAutoPath(.Object, getParam(.Object, "inputBed"),
                            "bed", suffix = "bed")
            # the path can also be generate in this way
            # ib <- getParam(.Object,"inputBed")
            # output(.Object)$outputBed <-
            #    file.path(getStepWorkDir(.Object),
            #    paste0(substring(ib,1,nchar(ib)-3), "bed"))
        }
        # begin to set other parameters
        # no other parameters
        # don't forget to return .Object
```

```
        .Object
    }
)
setMethod(
    f = "processing",
    signature = "OverlappedRandomRegion",
    definition = function(.Object,...){
        # All arguments are set in .Object
        # so we can get them from .Object
        allparam <- list(...)
        inputBed <- getParam(.Object,"inputBed")
        randomBed <- getParam(.Object,"randomBed")
        outputBed <- getParam(.Object,"outputBed")

        # begin the calculation
        gr1 <- import.bed(con = inputBed)
        gr2 <- import.bed(con = randomBed)
        gr <- second(findOverlapPairs(gr1,gr2))
        export.bed(gr,con = outputBed)
        # don't forget to return .Object
        .Object
    }
)


# This function is exported in NAMESPACE for user to use
overlappedRandomRegion <- function(inputBed, randomBed,
                                   outputBed = NULL, ...){
    allpara <- c(list(Class = "OverlappedRandomRegion",
        prevSteps = list()),as.list(environment()),list(...))
    step <- do.call(new,allpara)
    invisible(step)
}

setGeneric("runOverlappedRandomRegion",
           function(prevStep,
                    inputBed,
                    randomBed = NULL,
                    outputBed = NULL,
                    ...) standardGeneric("runOverlappedRandomRegion"))



setMethod(
    f = "runOverlappedRandomRegion",
    signature = "Step",
    definition = function(prevStep,
                          inputBed,
                          randomBed = NULL,
                          outputBed = NULL,
                          ...){
        allpara <- c(list(Class = "OverlappedRandomRegion",
            prevSteps = list(prevStep)),as.list(environment()),list(...))
        step <- do.call(new,allpara)
        invisible(step)
    }
)
```

```
# add to graph
addEdges(edges = c("RandomRegionOnGenome","OverlappedRandomRegion"),
         argOrder = 1)
# begin to test pipeline
setGenome("hg19")
# generate test BED file
test_bed <- file.path(tempdir(),"test.bed")
library(rtracklayer)
export.bed(GRanges("chr7:1-127473000"),test_bed)


rd <- randomRegionOnGenome(10000)
overlap <- runOverlappedRandomRegion(rd, inputBed = test_bed)

randombed <- getParam(rd,"outputBed")

randombed

overlap1 <-
    overlappedRandomRegion(inputBed = test_bed, randomBed = randombed)

clearStepCache(overlap1)
overlap1 <-
    overlappedRandomRegion(inputBed = test_bed, randomBed = randombed)
clearStepCache(rd)
clearStepCache(overlap1)
rd <- randomRegionOnGenome(10000) %>%
runOverlappedRandomRegion(inputBed = test_bed)

getStepName(rd)
getStepId(rd)
getDefName(rd)

isReady(rd)
```

`Utils` *Functions for directory operations*

Description

Functions for directory operations

Usage

```
getBasenamePrefix(filepath, words, ...)

getPathPrefix(filepath, words, ...)

checkFileExist(filePath, ...)

checkPathExist(filePath, ...)

checkFileCreatable(filePath, ...)
```

Arguments

filepath	character scalar or vector.
words	character scalar. Remove substring of the path characters starting to match the word
...	Additional arguments, currently unused
filePath	Character scalar.

Value

getBasenamePrefix	Get the filepath basename with removed suffix
getPathPrefix	Get the filepath with removed suffix
checkFileExist	(For package developer) Check file is exist.
checkPathExist	(For package developer) Check directory is exist.
checkFileCreatable	(For package developer) Check file creatable.

Examples

```
getBasenamePrefix("aaa/bbb.ccc.ddd","cCc")

getBasenamePrefix("aaa/bbb.ccc.ddd","ddd")

getPathPrefix("aaa/bbb.ccc.ddd","dDd")

getPathPrefix("aaa/bbb.ccc.ddd","ccc")

file.create("test.bed")

checkFileExist("test.bed")

tryCatch({checkFileExist("test.bed1")},error = function(e) e)

dir.create("testdir")

checkPathExist(file.path(getwd(),"testdir"))

tryCatch({checkPathExist(file.path(dirname(getwd()),
"notexistfolder","testdir"))},error = function(e) e)

checkFileCreatable("aaa.bed")

tryCatch({checkFileCreatable("testdir1/aaa.bed")},error = function(e) e)
```

Index