Package 'BatchJobs'

May 15, 2019

Title Batch Computing with R

Description Provides Map, Reduce and Filter variants to generate jobs on batch computing systems like PBS/Torque, LSF, SLURM and Sun Grid Engine. Multicore and SSH systems are also supported. For further details see the project web page.

Author Bernd Bischl <bernd_bischl@gmx.net>, Michel Lang <michellang@gmail.com>, Henrik Bengtsson <henrikb@braju.com>

Maintainer Bernd Bischl <bernd_bischl@gmx.net>

URL https://github.com/tudo-r/BatchJobs

BugReports https://github.com/tudo-r/BatchJobs/issues

MailingList batchjobs@googlegroups.com

License BSD_2_clause + file LICENSE

Depends R (>= 3.0.0), BBmisc (>= 1.9), methods

Imports backports (>= 1.1.1), brew, checkmate (>= 1.8.0), data.table (>= 1.9.6), DBI, digest, parallel, RSQLite (>= 1.0.9011), sendmailR, stats, stringi (>= 0.4-1), utils

Suggests MASS, testthat

LazyData yes

ByteCompile yes

Version 1.8

RoxygenNote 6.1.1

Encoding UTF-8

NeedsCompilation no

Repository CRAN

Date/Publication 2019-05-14 22:00:16 UTC

R topics documented:

`addRegistryPackages` *Add packages to registry.*

Description

Mutator function for `packages` in `makeRegistry`.

Usage

```
addRegistryPackages(reg, packages)
```

Arguments

reg	[Registry] Registry.
packages	[character] Packages to add to registry.

Value

Registry . Changed registry.

See Also

Other exports: addRegistrySourceDirs, addRegistrySourceFiles, batchExport, batchUnexport, loadExports, removeRegistryPackages, removeRegistrySourceDirs, removeRegistrySourceFiles, setRegistryPackages

---

addRegistrySourceDirs *Add source dirs to registry.*

Description

Mutator function for src.dirs in makeRegistry.

Usage

```
addRegistrySourceDirs(reg, src.dirs, src.now = TRUE)
```

Arguments

reg	[Registry] Registry.
src.dirs	[character] Paths to add to registry. See makeRegistry.
src.now	[logical(1)] Source files now on master? Default is TRUE.

Value

Registry . Changed registry.

See Also

Other exports: addRegistryPackages, addRegistrySourceFiles, batchExport, batchUnexport, loadExports, removeRegistryPackages, removeRegistrySourceDirs, removeRegistrySourceFiles, setRegistryPackages

addRegistrySourceFiles *Add source files to registry.*

Description

Mutator function for `src.files` in `makeRegistry`.

Usage

```
addRegistrySourceFiles(reg, src.files, src.now = TRUE)
```

Arguments

`reg`	[`Registry`] Registry.
`src.files`	[`character`] Paths to add to registry. See `makeRegistry`.
`src.now`	[`logical(1)`] Source files now on master? Default is `TRUE`.

Value

`Registry` . Changed registry.

See Also

Other exports: `addRegistryPackages`, `addRegistrySourceDirs`, `batchExport`, `batchUnexport`, `loadExports`, `removeRegistryPackages`, `removeRegistrySourceDirs`, `removeRegistrySourceFiles`, `setRegistryPackages`

batchExpandGrid *Map function over all combinations.*

Description

Maps an n-ary-function over a list of all combinations which are given by some vectors. Internally `expand.grid` is used to compute the combinations, then `batchMap` is called.

Usage

```
batchExpandGrid(reg, fun, ..., more.args = list())
```

Arguments

reg	[Registry] Empty Registry that will store jobs for the mapping.
fun	[function] Function to map over the combinations.
...	[any] Vectors that are used to compute all combinations. If the arguments are named, these names are used to bind to arguments of fun.
more.args	[list] A list of other arguments passed to fun. Default is empty list.

Value

data.frame . Expanded grid of combinations produced by expand.grid.

Examples

```
reg = makeRegistry(id = "BatchJobsExample", file.dir = tempfile(), seed = 123)
f = function(x, y, z) x * y  + z
# lets store the param grid
grid = batchExpandGrid(reg, f, x = 1:2, y = 1:3, more.args = list(z = 10))
submitJobs(reg)
waitForJobs(reg)
y = reduceResultsVector(reg)
# later, we can always access the param grid like this
grid = getJobParamDf(reg)
cbind(grid, y = y)
```

---

batchExport *Export R object to be available on the slaves.*

Description

Saves objects as RData files in the "exports" subdirectory of your file.dir to be later loaded on the slaves.

Usage

```
batchExport(reg, ..., li = list(), overwrite = FALSE)
```

Arguments

reg	[Registry] Registry.
...	[any] Objects to export. You must provide a valid name.

li [list]
More objects to export provided as a named list.

overwrite [logical(1)]
If set to FALSE (default), exported objects are protected from being overwritten by multiple calls of this function. Setting this to TRUE disables this check.

Value

character . Invisibly returns a character vector of exported objects.

See Also

Other exports: addRegistryPackages, addRegistrySourceDirs, addRegistrySourceFiles, batchUnexport, loadExports, removeRegistryPackages, removeRegistrySourceDirs, removeRegistrySourceFiles, setRegistryPackages

---

BatchJobs *The BatchJobs package*

Description

The BatchJobs package

Additional information

Homepage: https://github.com/tudo-r/BatchJobs

Wiki: https://github.com/tudo-r/BatchJobs/wiki

FAQ: https://github.com/tudo-r/BatchJobs/wiki/FAQ

Configuration: https://github.com/tudo-r/BatchJobs/wiki/Configuration

The package currently support the following further R options, which you can set either in your R profile file or a script via options:

BatchJobs.verbose This boolean flag can be set to FALSE to reduce the console output of the package operations. Usually you want to see this output in interactive work, but when you use the package in e.g. knitr documents, it clutters the resulting document too much.

BatchJobs.check.posix If this boolean flag is enabled, the package checks your registry file dir (and related user-defined directories) quite strictly to be POSIX compliant. Usually this is a good idea, you do not want to have strange chars in your file paths, as this might results in problems when these paths get passed to the scheduler or other command-line tools that the package interoperates with. But on some OS this check might be too strict and cause problems. Setting the flag to FALSE allows to disable the check entirely. The default is FALSE on Windows systems and TRUE else.

batchMap — *Maps a function over lists or vectors, adding jobs to a registry.*

Description

You can then submit these jobs to the batch system.

Usage

```
batchMap(reg, fun, ..., more.args = list(), use.names = FALSE)
```

Arguments

reg	[Registry] Empty Registry that will store jobs for the mapping.
fun	[function] Function to map over ....
...	[any] Arguments to vectorize over (list or vector).
more.args	[list] A list of other arguments passed to fun. Default is empty list.
use.names	[logical(1)] Store parameter names to enable named results in loadResults and some other functions. Default is FALSE.

Value

Vector of type integer with job ids.

Examples

```
reg = makeRegistry(id = "BatchJobsExample", file.dir = tempfile(), seed = 123)
f = function(x) x^2
batchMap(reg, f, 1:10)
print(reg)
```

batchMapQuick — *Combination of makeRegistry, batchMap and submitJobs.*

Description

Combination of makeRegistry, batchMap and submitJobs for quick computations on the cluster. Should only be used by skilled users who know what they are doing. Creates the file.dir, maps function, potentially chunks jobs and submits them.

Usage

```
batchMapQuick(fun, ..., more.args = list(), file.dir = NULL,
  packages = character(0L), chunk.size, n.chunks,
  chunks.as.arrayjobs = FALSE, inds, resources = list())
```

Arguments

fun	[function] Function to map over ....
...	[any] Arguments to vectorize over (list or vector).
more.args	[list] A list of other arguments passed to fun. Default is empty list.
file.dir	[character] See makeRegistry. Default is NULL, which means that it is created in the current directory under the name "bmq_[random alphanumerics]".
packages	[character] See makeRegistry.
chunk.size	[integer(1)] Preferred number of jobs in each chunk. Can not be used in combination with n.chunks. Note that the ids will get shuffled to balance out possible run time differences. Default is not to use chunking.
n.chunks	[integer(1)] Preferred number chunks. Can not be used in combination with chunk.size. Note that the ids will get shuffled to balance out possible run time differences. Default is not to use chunking.
chunks.as.arrayjobs	[logical(1)] Submit chunks as array jobs? Default is FALSE.
inds	[integer] Indices of ids / chunks to submit. Default is all. If ids get chunked, this subsets the list of shuffled ids.
resources	[list] Required resources for all batch jobs. Default is empty list.

Value

Registry

batchMapResults *Maps a function over the results of a registry by using batchMap.*

Description

Maps a function over the results of a registry by using batchMap.

Usage

```
batchMapResults(reg, reg2, fun, ..., ids, part = NA_character_,
  more.args = list())
```

Arguments

reg	[Registry] Registry whose results should be mapped by fun.
reg2	[Registry] Empty registry that should store the job for the mapping.
fun	[function(job, res, ...)] Function to map over results of reg. Further arguments come from ... of batchMapResults and more.args.
...	[any] Furher arguments to vectorize over (list or vector). Must all be the same length as number of results in reg.
ids	[integer] Ids of jobs whose results should be mapped with fun. Default is all jobs.
part	[character] Only useful for multiple result files, then defines which result file part(s) should be loaded. NA means all parts are loaded, which is the default.
more.args	[list] A list of other arguments passed to fun. Default is empty list.

Value

Vector of type integer with job ids.

Examples

```
reg1 = makeRegistry(id = "BatchJobsExample1", file.dir = tempfile(), seed = 123)
# square some numbers
f = function(x) x^2
batchMap(reg1, f, 1:10)

# submit jobs and wait for the jobs to finish
submitJobs(reg1)
waitForJobs(reg1)
```

```
# look at results
reduceResults(reg1, fun = function(aggr,job,res) c(aggr, res))

reg2 = makeRegistry(id = "BatchJobsExample2", file.dir = tempfile(), seed = 123)

# define function to tranform results, we simply do the inverse of the squaring
g = function(job, res) sqrt(res)
batchMapResults(reg1, reg2, fun = g)

# submit jobs and wait for the jobs to finish
submitJobs(reg2)
waitForJobs(reg2)

# check results
reduceResults(reg2, fun = function(aggr,job,res) c(aggr, res))
```

---

batchQuery *Manually query the BatchJobs database*

Description

Manually query the BatchJobs database

Usage

```
batchQuery(reg, query, flags = "ro")
```

Arguments

reg	[Registry] Registry.
query	[character(1)] SQL query to send to the database.
flags	[character(1)] One of "ro", "rw" or "rwc" which is translated to SQLITE_RO, SQLITE_RW or SQLITE_RWC, respectively. See SQLITE_RO for more info.

Value

data.frame Result of the query.

Examples

```
reg = makeRegistry("test", file.dir = tempfile())
batchMap(reg, identity, i = 1:10)
batchQuery(reg, "SELECT * FROM test_job_status")
```

batchReduce *Reduces via a binary function over a list adding jobs to a registry.*

Description

Each jobs reduces a certain number of elements on one slave. You can then submit these jobs to the batch system.

Usage

```
batchReduce(reg, fun, xs, init, block.size, more.args = list())
```

Arguments

reg	[Registry] Empty Registry.
fun	[function(aggr, x, ...)] Function to reduce xs with.
xs	[vector] Vector to reduce.
init	[any] Initial object for reducing.
block.size	[integer(1)] Number of elements of xs reduced in one job.
more.args	[list] A list of other arguments passed to fun. Default is empty list.

Value

Vector of type integer with job ids.

Examples

```
# define function to reduce on slave, we want to sum a vector
f = function(aggr, x) aggr + x
reg = makeRegistry(id = "BatchJobsExample", file.dir = tempfile(), seed = 123)

# sum 20 numbers on each slave process, i.e. 5 jobs
batchReduce(reg, fun = f, 1:100, init = 0, block.size = 5)
submitJobs(reg)
waitForJobs(reg)

# now reduce one final time on master
reduceResults(reg, fun = function(aggr,job,res) f(aggr, res))
```

batchReduceResults *Reduces results via a binary function and adds jobs for this to a registry.*

Description

Each jobs reduces a certain number of results on one slave. You can then submit these jobs to the batch system. Later, you can do a final reduction with `reduceResults` on the master.

Usage

```
batchReduceResults(reg, reg2, fun, ids, part = NA_character_, init,
  block.size, more.args = list())
```

Arguments

`reg`	[`Registry`] Registry whose results should be reduced by `fun`.
`reg2`	[`Registry`] Empty registry that should store the job for the mapping.
`fun`	[`function(aggr, job, res, ...)`] Function to reduce results with.
`ids`	[`integer`] Ids of jobs whose results should be reduced with `fun`. Default is all jobs.
`part`	[`character`] Only useful for multiple result files, then defines which result file part(s) should be loaded. `NA` means all parts are loaded, which is the default.
`init`	[any] Initial object for reducing.
`block.size`	[`integer(1)`] Number of results reduced in one job.
`more.args`	[`list`] A list of other arguments passed to `fun`. Default is empty list.

Value

Vector of type `integer` with job ids.

Examples

```
# generating example results:
reg1 = makeRegistry(id = "BatchJobsExample1", file.dir = tempfile(), seed = 123)
f = function(x) x^2
batchMap(reg1, f, 1:20)
submitJobs(reg1)
waitForJobs(reg1)
```

```
# define function to reduce on slave, we want to sum the squares
myreduce = function(aggr, job, res) aggr + res

# sum 5 results on each slave process, i.e. 4 jobs
reg2 = makeRegistry(id = "BatchJobsExample2", file.dir = tempfile(), seed = 123)
batchReduceResults(reg1, reg2, fun = myreduce, init = 0, block.size = 5)
submitJobs(reg2)
waitForJobs(reg2)

# now reduce one final time on master
reduceResults(reg2, fun = myreduce)
```

---

batchUnexport *Unload exported R objects.*

Description

Removes RData files from the "exports" subdirectory of your file.dir and thereby prevents loading on the slave.

Usage

```
batchUnexport(reg, what)
```

Arguments

reg	[Registry] Registry.
what	[character] Names of objects to remove.

Value

character . Invisibly returns a character vector of unexported objects.

See Also

Other exports: addRegistryPackages, addRegistrySourceDirs, addRegistrySourceFiles, batchExport, loadExports, removeRegistryPackages, removeRegistrySourceDirs, removeRegistrySourceFiles, setRegistryPackages

`callFunctionOnSSHWorkers` *Call an arbitrary function on specified SSH workers.*

Description

Calls can be made in parallel or consecutively, the function waits until all calls have finished and returns call results. In consecutive mode the output on the workers can also be shown on the master during computation.

Please read and understand the comments for argument `dir`.

Note that this function should only be used for short administrative tasks or information gathering on the workers, the true work horse for real computation is `submitJobs`.

In `makeSSHWorker` various options for load management are possible. Note that these will be ignored for the current call to execute it immediatly.

Usage

```
callFunctionOnSSHWorkers(nodenames, fun, ..., consecutive = FALSE,
  show.output = consecutive, simplify = TRUE, use.names = TRUE,
  dir = getwd())
```

Arguments

`nodenames`	[`character`] Nodenames of workers to call function on. Only workers which were specified in your `makeClusterFunctionsSSH` configuration can be used.
`fun`	[`function`] Function to call on workers.
`...`	[any] Arguments for `fun`.
`consecutive`	[`logical(1)`] Do calls consecutively and always wait until each worker is done. Default is `FALSE`.
`show.output`	[`logical(1)`] Show output of workers on master during call. Can be useful to see what is happening. Can only be used in consecutive mode. Default is `consecutive`.
`simplify`	[`logical(1)`] Should the result be simplified? See `sapply`. Default is `TRUE`.
`use.names`	[`logical(1)`] Name results by `nodenames`. Default is `TRUE`.
`dir`	[`character(1)`] Directory under which a temporary registry will be created in a subdirectory for communication. This has to be somewhere on the shared filesystem. The created subdirectory will be cleaned up on exit. Default is current working directory.

Value

Results of function calls, either a list or simplified.

`cfBrewTemplate` *Cluster functions helper: Brew your template into a job description file.*

Description

This function is only intended for use in your own cluster functions implementation.

Calls brew silently on your template, any error will lead to an exception. If debug mode is turned on in the configuration, the file is stored at the same place as the corresponding R script in the "jobs"-subdir of your files directory, otherwise in the temp dir via `tempfile`.

Usage

```
cfBrewTemplate(conf, template, rscript, extension)
```

Arguments

`conf`	[environment] BatchJobs configuration.
`template`	[character(1)] Job desfription template as a char vecrtor, possibly read in via `cfReadBrewTemplate`.
`rscript`	[character(1)] File path to you your corresponding R script for the job.
`extension`	[character(1)] Extension for the job description file, e.g. "pbs".

Value

`character(1)` . File path of result.

`cfHandleUnknownSubmitError` *Cluster functions helper: Handle an unknown error during job submission.*

Description

This function is only intended for use in your own cluster functions implementation.

Simply constructs a `SubmitJobResult` object with status code 101, NA as batch job id and an informative error message containing the output of the OS command in `output`.

Usage

```
cfHandleUnknownSubmitError(cmd, exit.code, output)
```

Arguments

cmd	[character(1)] OS command used to submit the job, e.g. qsub.
exit.code	[integer(1)] Exit code of the OS command, should not be 0.
output	[character] Output of the OS command, hopefully an informative error message. If these are mutiple lines in a vector, they are automatically pasted together.

Value

SubmitJobResult .

---

cfKillBatchJob *Cluster functions helper: Kill a batch job via OS command*

Description

This function is only intended for use in your own cluster functions implementation.

Calls the OS command to kill a job via system like this: "cmd batch.job.id". If the command returns an exit code > 0, the command is repeated after a 1 second sleep max.tries-1 times. If the command failed in all tries, an exception is generated.

Usage

```
cfKillBatchJob(cmd, batch.job.id, max.tries = 3L)
```

Arguments

cmd	[character(1)] OS command, e.g. "qdel".
batch.job.id	[character(1)] Id of the batch job on the batch system.
max.tries	[integer(1)] Number of total times to try execute the OS command in cases of failures. Default is 3.

Value

Nothing.

cfReadBrewTemplate *Cluster functions helper: Read in your brew template file.*

Description

This function is only intended for use in your own cluster functions implementation.

Simply reads your template and returns it as a character vector. If you do this in the constructor of your cluster functions once, you can avoid this repeated file access later on.

Usage

```
cfReadBrewTemplate(template.file)
```

Arguments

`template.file`	[`character(1)`] File path.

Value

`character` .

checkIds *Check job ids.*

Description

Simply checks if probided vector of job ids is valid and throws an error if something is odd.

Usage

```
checkIds(reg, ids, check.present = TRUE, len = NULL)
```

Arguments

`reg`	[`Registry`] Registry.
`ids`	[`integer`] Vector of job ids.
`check.present`	[`logical(1)`] Check if the ids are present in the database? Default is `TRUE`.
`len`	[`integer(1)`] Expected length of id vector. Passed to `asInteger`.

Value

Invisibly returns the vector of ids, converted to integer.

configuration *BatchJobs configuration.*

Description

In order to understand how the package should be configured please read https://github.com/tudo-r/BatchJobs/wiki/Configuration.

See Also

Other conf: getConfig, loadConfig, setConfig

debugMulticore *Helper function to debug multicore mode.*

Description

Useful in case of severe errors. Tries different operations of increasing difficulty and provides debug output on the console

Usage

```
debugMulticore(r.options)
```

Arguments

r.options	[list] Options for R and Rscript, one option per element of the vector, a la "–vanilla". Default is c("--no-save", "--no-restore", "--no-init-file", "--no-site-file").

Value

Nothing.

See Also

Other debug: debugSSH, getErrorMessages, getJobInfo, getLogFiles, grepLogs, killJobs, resetJobs, setJobFunction, showLog, testJob

debugSSH *Helper function to debug SSH mode.*

Description

Useful in case of configuration problems. Tries different operations of increasing difficulty and provides debug output on the console.

Note that this function does not access nor use information specified for your cluster functions in your configuration.

Usage

```
debugSSH(nodename, ssh.cmd = "ssh", ssh.args = character(0L),
  rhome = "", r.options = c("--no-save", "--no-restore",
  "--no-init-file", "--no-site-file"), dir = getwd())
```

Arguments

nodename	[character(1)] Node on which worker should be constructed for the test.
ssh.cmd	[character(1))] CLI command to ssh into remote node. Default is "ssh".
ssh.args	[character] CLI args for ssh.cmd. Default is none.
rhome	[character(1)] Path to R installation on the worker. "" means R installation on the PATH is used, of course this implies that it must be on the PATH (also for non-interactive shells)! Default is "".
r.options	[list] Options for R and Rscript, one option per element of the vector, a la "–vanilla". Default is c("--no-save", "--no-restore", "--no-init-file", "--no-site-file").
dir	[character(1)] Path where internally used test registries can be created. Note that this must be shared for the worker. Default is current working directory.

Value

Nothing.

See Also

Other debug: debugMulticore, getErrorMessages, getJobInfo, getLogFiles, grepLogs, killJobs, resetJobs, setJobFunction, showLog, testJob

filterResults *Find all results where a specific condition is true.*

Description

Find all results where a specific condition is true.

Usage

```
filterResults(reg, ids, fun, ...)
```

Arguments

reg	[Registry] Registry.
ids	[integer] Ids of jobs whose results you want to test for the condition. Default is all jobs for which results are available.
fun	[fun(job, res)] Predicate function that returns TRUE or FALSE.
...	[any] Additional arguments to fun.

Value

integer . Ids of jobs where fun(job, result) returns TRUE.

Examples

```
reg = makeRegistry(id = "BatchJobsExample", file.dir = tempfile(), seed = 123)
f = function(x) x^2
batchMap(reg, f, 1:10)
submitJobs(reg)
waitForJobs(reg)

# which square numbers are even:
filterResults(reg, fun = function(job, res) res %% 2 == 0)
```

findDone *Find jobs depending on computional state.*

Description

findDone: Find jobs which succesfully terminated.

Usage

```
findDone(reg, ids, limit = NULL)

findNotDone(reg, ids, limit = NULL)

findMissingResults(reg, ids, limit = NULL)

findErrors(reg, ids, limit = NULL)

findNotErrors(reg, ids, limit = NULL)

findTerminated(reg, ids, limit = NULL)

findNotTerminated(reg, ids, limit = NULL)

findSubmitted(reg, ids, limit = NULL)

findNotSubmitted(reg, ids, limit = NULL)

findOnSystem(reg, ids, limit = NULL)

findNotOnSystem(reg, ids, limit = NULL)

findRunning(reg, ids, limit = NULL)

findNotRunning(reg, ids, limit = NULL)

findStarted(reg, ids, limit = NULL)

findNotStarted(reg, ids, limit = NULL)

findExpired(reg, ids, limit = NULL)

findDisappeared(reg, ids, limit = NULL)
```

Arguments

reg	[Registry] Registry.

ids	[integer] Subset of job ids to restrict the result to. Default is all jobs.
limit	[integer(1)] Limit the number of returned ids. Default is all ids.

Value

integer . Ids of jobs.

---

findJobs *Finds ids of jobs that match a query.*

Description

Finds ids of jobs that match a query.

Usage

```
findJobs(reg, ids, pars, jobnames)
```

Arguments

reg	[Registry] Registry.
ids	[integer] Subset of job ids to restrict the result to. Default is all jobs.
pars	[R expression] All jobs whose parameters match the given expression are selected. This implies that you have named the parameters when you passed the vectors. If you forgot to do this you can use .arg1, .arg2, etc., to refer to the the unnamed ones.
jobnames	[character] Restrict to jobs with stored names. Exact matching is used.

Value

integer . Ids for jobs which match the query.

Examples

```
reg = makeRegistry(id = "BatchJobsExample", file.dir = tempfile(), seed = 123)
f = function(x, y) x * y
batchExpandGrid(reg, f, x = 1:2, y = 1:3)
findJobs(reg, pars = (y > 2))
```

getConfig *Returns a list of BatchJobs configuration settings*

Description

Returns a list of BatchJobs configuration settings

Usage

```
getConfig()
```

Value

`list` of current configuration variables with classs "Config".

See Also

Other conf: `configuration`, `loadConfig`, `setConfig`

getErrorMessages *Get error messages of jobs.*

Description

Get error messages of jobs.

Usage

```
getErrorMessages(reg, ids)
```

Arguments

`reg`	[`Registry`] Registry.
`ids`	[`integer`] Ids of jobs. Default is all jobs with errors.

Value

`character` . Error messages for jobs as character vector
`NA` if job has terminated successfully.

See Also

Other debug: `debugMulticore`, `debugSSH`, `getJobInfo`, `getLogFiles`, `grepLogs`, `killJobs`, `resetJobs`, `setJobFunction`, `showLog`, `testJob`

getJob *Get job from registry by id.*

Description

Get job from registry by id.

Usage

```
getJob(reg, id, check.id = TRUE)
```

Arguments

reg	[Registry] Registry.
id	[integer(1)] Id of job.
check.id	[logical(1)] Check the job id? Default is TRUE.

Value

Job .

getJobIds *Get ids of jobs in registry.*

Description

Get ids of jobs in registry.

Usage

```
getJobIds(reg)
```

Arguments

reg	[Registry] Registry.

Value

character .

getJobInfo *Get computational information of jobs.*

Description

Returns time stamps (submitted, started, done, error), time running, approximate memory usage (in Mb), error messages (shortened, see showLog for detailed error messages), time in queue, hostname of the host the job was executed, assigned batch ID, the R PID and the seed of the job.

To estimate memory usage the sum of the last column of gc is used.

Column "time.running" displays the time until either the job was done, or an error occured; it will by NA in case of time outs or hard R crashes.

Usage

```
getJobInfo(reg, ids, pars = FALSE, prefix.pars = FALSE, select,
  unit = "seconds")
```

Arguments

reg	[Registry] Registry.
ids	[integer] Ids of jobs. Default is all jobs.
pars	[logical(1)] Include job parameters in the output? Default is FALSE.
prefix.pars	[logical(1)] Should a prefix be added to job parameter names (column names) to avoid name clashes? Default is FALSE.
select	[character] Select only a subset of columns. Usually this is not required and you can subset yourself, but in some rare cases it may be advantageous to not query all information. Note that the column "id" (job id) is always selected. If not provided, all columns are queried and returned.
unit	[character(1)] Unit to convert execution and queing times to. Possible values: "seconds", "minutes", "hours", "days" and "weeks". Default is "seconds".

Value

data.frame .

See Also

Other debug: debugMulticore, debugSSH, getErrorMessages, getLogFiles, grepLogs, killJobs, resetJobs, setJobFunction, showLog, testJob

getJobLocation *Get the physical location of job files on the hard disk.*

Description

Get the physical location of job files on the hard disk.

Usage

```
getJobLocation(reg, ids)
```

Arguments

reg	[Registry] Registry.
ids	[integer] Ids of jobs. Default is all jobs.

Value

character Vector of directories.

getJobNr *Get number of jobs in registry.*

Description

Get number of jobs in registry.

Usage

```
getJobNr(reg)
```

Arguments

reg	[Registry] Registry.

Value

integer(1) .

getJobParamDf *Retrieve Job Parameters.*

Description

Returns parameters for all jobs as the rows of a data.frame.

Usage

```
getJobParamDf(reg, ids)
```

Arguments

reg	[Registry] Registry.
ids	[integer] Ids of jobs. Default is all jobs.

Value

data.frame . Rows are named with job ids.

Examples

```
# see batchExpandGrid
```

getJobResources *Function to get the resources that were submitted for some jobs.*

Description

Throws an error if call it for unsubmitted jobs.

Usage

```
getJobResources(reg, ids, as.list = TRUE)
```

Arguments

reg	[Registry] Registry.
ids	[integer] Ids of jobs. Default is all submitted jobs.
as.list	[integer(1)] If FALSE a data.frame will be returned. Default is TRUE.

Value

list | data.frame . List (or data.frame) of resource lists as passed to submitJobs.

---

getJobs *Get jobs from registry by id.*

Description

Get jobs from registry by id.

Usage

```
getJobs(reg, ids, check.ids = TRUE)
```

Arguments

reg	[Registry] Registry.
ids	[integer] Ids of jobs. Default is all jobs.
check.ids	[logical(1)] Check the job ids? Default is TRUE.

Value

list of Job .

---

getLogFiles *Get log file paths for jobs.*

Description

Get log file paths for jobs.

Usage

```
getLogFiles(reg, ids)
```

Arguments

reg	[Registry] Registry.
ids	[integer] Ids of jobs. Default is all jobs.

Value

character . Vector of file paths to log files.

See Also

Other debug: debugMulticore, debugSSH, getErrorMessages, getJobInfo, grepLogs, killJobs, resetJobs, setJobFunction, showLog, testJob

---

getResources *Function to get job resources in job function.*

Description

Return the list passed to submitJobs, e.g. nodes, walltime, etc.

Usage

```
getResources()
```

Details

Can only be called in job function during job execution on slave.

Value

list .

---

getSSHWorkersInfo *Print and return R installation and other information for SSH workers.*

Description

Workers are queried in parallel via callFunctionOnSSHWorkers.

The function will display a warning if the first lib path on the worker is not writable as this indicates potential problems in the configuration and installPackagesOnSSHWorkers will not work.

Usage

```
getSSHWorkersInfo(nodenames)
```

Arguments

nodenames	[character] Nodenames of workers.

Value

`list` . Displayed information as a list named by nodenames.

See Also

`callFunctionOnSSHWorkers`

`grepLogs` *Grep log files for a pattern.*

Description

Searches for occurence of `pattern` in log files.

Usage

```
grepLogs(reg, ids, pattern = "warn", ignore.case = TRUE,
  verbose = FALSE, range = 2L)
```

Arguments

`reg`	[`Registry`] Registry.
`ids`	[`integer`] Ids of jobs to grep. Default is all terminated jobs (done + errors).
`pattern`	[`character(1)`] Pattern to search for. See `grep`. Default is `"warn"`.
`ignore.case`	[`logical(1)`] Ignore case. See `grep`. Default is `TRUE`.
`verbose`	[`logical(1)`] Print matches. Default is `FALSE`.
`range`	[`integer(1)`] If `verbose` is set to `TRUE`, print `range` leading and trailing lines for contextual information about the warning. Default is `2`.

Value

`integer` . Ids of jobs where pattern was found in the log file.

See Also

Other debug: `debugMulticore`, `debugSSH`, `getErrorMessages`, `getJobInfo`, `getLogFiles`, `killJobs`, `resetJobs`, `setJobFunction`, `showLog`, `testJob`

`installPackagesOnSSHWorkers` *Install packages on SSH workers.*

Description

Installation is done via `callFunctionOnSSHWorkers` and `install.packages`.

Note that as usual the function tries to install the packages into the first path of `.libPaths()` of each each worker.

Usage

```
installPackagesOnSSHWorkers(nodenames, pkgs, repos = getOption("repos"),
  consecutive = TRUE, show.output = consecutive, ...)
```

Arguments

`nodenames`	[`character`] Nodenames of workers.
`pkgs`	[`character`] See `install.packages`.
`repos`	[`character`] See `install.packages`. If the user must be queried this is of course done on the master.
`consecutive`	[`logical(1)`] See `callFunctionOnSSHWorkers`. Default is `TRUE`.
`show.output`	[`logical(1)`] See `callFunctionOnSSHWorkers`. Default is `consecutive`.
`...`	[any] Passed to `install.packages`.

Value

Nothing.

See Also

`callFunctionOnSSHWorkers`

killJobs *Kill some jobs on the batch system.*

Description

Kill jobs which have already been submitted to the batch system. If a job is killed its internal state is reset as if it had not been submitted at all.

The function informs if (a) the job you want to kill has not been submitted, (b) the job has already terminated, (c) for some reason no batch job id is available. In all 3 cases above, nothing is changed for the state of this job and no call to the internal kill cluster function is generated.

In case of an error when killing, the function tries - after a short sleep - to kill the remaining batch jobs again. If this fails again for some jobs, the function gives up. Only jobs that could be killed are reset in the DB.

Usage

```
killJobs(reg, ids, progressbar = TRUE)
```

Arguments

reg	[Registry] Registry.
ids	[integer] Ids of jobs to kill. Default is none.
progressbar	[logical(1)] Set to FALSE to disable the progress bar. To disable all progress bars, see makeProgressBar.

Value

integer . Ids of killed jobs.

See Also

Other debug: debugMulticore, debugSSH, getErrorMessages, getJobInfo, getLogFiles, grepLogs, resetJobs, setJobFunction, showLog, testJob

Examples

```
## Not run:
reg = makeRegistry(id = "BatchJobsExample", file.dir = tempfile(), seed = 123)
f = function(x) Sys.sleep(x)
batchMap(reg, f, 1:10 + 5)
submitJobs(reg)
waitForJobs(reg)

# kill all jobs currently _running_
```

```
killJobs(reg, findRunning(reg))
# kill all jobs queued or running
killJobs(reg, findNotTerminated(reg))

## End(Not run)
```

`loadConfig` *Load a specific configuration file.*

Description

Load a specific configuration file.

Usage

```
loadConfig(conffile = ".BatchJobs.R")
```

Arguments

`conffile`	`[character(1)]` Location of the configuration file to load. Default is ".BatchJobs.conf" in the current working directory.

Value

Invisibly returns a list of configuration settings.

See Also

Other conf: `configuration`, `getConfig`, `setConfig`

`loadExports` *Load exported R data objects.*

Description

Loads exported `RData` object files in the "exports" subdirectory of your `file.dir` and assigns the objects to the global environment.

Usage

```
loadExports(reg, what = NULL)
```

Arguments

reg	[Registry] Registry.
what	[character] Names of objects to load. Defaults to all objects exported.

Value

character . Invisibly returns a character vector of loaded objects.

See Also

Other exports: addRegistryPackages, addRegistrySourceDirs, addRegistrySourceFiles, batchExport, batchUnexport, removeRegistryPackages, removeRegistrySourceDirs, removeRegistrySourceFiles, setRegistryPackages

---

loadRegistry *Load a previously saved registry.*

Description

Loads a previously created registry from the file system. The file.dir is automatically updated upon load if adjust.paths is set to TRUE, so be careful if you use the registry on multiple machines simultaneously, e.g. via sshfs or a samba share.

There is a heuristic included which tries to detect if the location of the registry has changed and returns a read-only registry if necessary.

Usage

```
loadRegistry(file.dir, work.dir, adjust.paths = FALSE)
```

Arguments

file.dir	[character(1)] Location of the file.dir to load the registry from.
work.dir	[character(1)] Location of the work. Unchanged if missing.
adjust.paths	[logical(1)] If set to FALSE (default), the paths for the file.dir and work.dir will not be updated. Set to TRUE if you moved the directoy to another system *after* all computations finished.

Value

Registry .

loadResult *Loads a specific result file.*

Description

Loads a specific result file.

Usage

```
loadResult(reg, id, part = NA_character_, missing.ok = FALSE,
  impute.val = NULL)
```

Arguments

reg	[Registry] Registry.
id	[integer(1)] Id of job.
part	[character] Only useful for multiple result files, then defines which result file part(s) should be loaded. NA means all parts are loaded, which is the default.
missing.ok	[logical(1)] If FALSE an error is thrown if no result file is found. Otherwise NULL is returned. Default is FALSE.
impute.val	[any] The value to return when no result is available. Defaults to NULL (the previous behavior)

Value

any . Result of job.

See Also

reduceResults

loadResults *Loads result files for id vector.*

Description

Loads result files for id vector.

Usage

```
loadResults(reg, ids, part = NA_character_, simplify = FALSE,
  use.names = "ids", missing.ok = FALSE, impute.val = NULL)
```

Arguments

reg	[Registry] Registry.
ids	[integer] Ids of jobs. Default is all done jobs.
part	[character] Only useful for multiple result files, then defines which result file part(s) should be loaded. NA means all parts are loaded, which is the default.
simplify	[logical(1)] Should the result be simplified to a vector, matrix or higher dimensional array if possible? Default is TRUE.
use.names	[character(1)] Name the results with job ids ("ids"), stored job names ("names") or return a unnamed result ("none"). Default is ids.
missing.ok	[logical(1)] If FALSE an error is thrown if the results are not found. Otherwise missing results are imputed to NULL. Default is FALSE.
impute.val	[any] The value to return when no result is available. Defaults to NULL (the previous behavior)

Value

list . Results of jobs as list, possibly named by ids.

See Also

reduceResults

---

makeClusterFunctions *Create a ClusterFuntions object.*

Description

Use this funtion when you implement a backend for a batch system. You must define the functions specified in the arguments.

Usage

```
makeClusterFunctions(name, submitJob, killJob, listJobs, getArrayEnvirName,
  class = NULL, ...)
```

Arguments

name	[character(1)] Name of cluster functions.
submitJob	[function(conf, reg, job.name, rscript, log.file, job.dir, resources, ...)] Function to submit a new job. Must return a SubmitJobResult object. The arguments are: conf [environment]: The user configuration. reg [Registry]: The registry. job.name [character(1)]: Name of job, used if the job is displayed on the batch system. This is just for display and not an id! rscript [character(1)]: File path to R script that is used to execute the job. log.file [character(1)]: File path where log file (.Rout) has to be placed. job.dir [character(1)]: Directory where all files relating to this job are placed. resources [list]: Freely definable list of required resources for this job, e.g. walltime or memory.
killJob	[function(conf, reg, batch.job.id)] Function to kill a job on the batch system. Make sure that you definately kill the job! Return value is currently ignored. The arguments are: conf [environment]: The user configuration. reg [Registry]: The registry. batch.job.id [character(1)]: Batch job id, as produced by submitJob. Set killJob to NULL if killing jobs cannot be supported.
listJobs	[function(conf, reg)] List all jobs on the batch system for the current user / registry. This includes queued, running, held, idle, etc. jobs. Must return an integer vector of batch job ids, same format as they are produced by submitJob. It does not matter if you return a few job ids too many (e.g. all for the current user instead of all for the current registry), but you have to include all relevant ones. The arguments are: conf [environment]: The user configuration. reg [Registry]: The registry. Set listJobs to NULL if listing jobs cannot be supported.
getArrayEnvirName	[function()] Returns the name of the environment variable specifying the array ID. Should return NA if not supported.
class	[character(1)] Optional class name for cluster functions object. Useful to provide a nice print method which might show additional information about the workers. Default is NULL.
...	[any] Currently ignored.

See Also

Other clusterFunctions: makeClusterFunctionsInteractive, makeClusterFunctionsLSF, makeClusterFunctionsLoca
makeClusterFunctionsMulticore, makeClusterFunctionsOpenLava, makeClusterFunctionsSGE,

makeClusterFunctionsSLURM, makeClusterFunctionsSSH, makeClusterFunctionsTorque

---

makeClusterFunctionsInteractive

Create cluster functions for sequential execution in same session.

Description

All jobs executed under these cluster functions are executed sequentially, in the same interactive R process that you currently are. That is, submitJob does not return until the job has finished. The main use of this ClusterFunctions implementation is to test and debug programs on a local computer.

Listing jobs returns an empty vector (as no jobs can be running when you call this) and killJob returns at once (for the same reason).

Usage

```
makeClusterFunctionsInteractive(write.logs = TRUE)
```

Arguments

write.logs	[logical(1)] Sink the output to log files. Turning logging off can increase the speed of calculations but makes it next to impossible to debug. Default is TRUE.

Value

ClusterFunctions .

See Also

Other clusterFunctions: makeClusterFunctionsLSF, makeClusterFunctionsLocal, makeClusterFunctionsMulticore, makeClusterFunctionsOpenLava, makeClusterFunctionsSGE, makeClusterFunctionsSLURM, makeClusterFunctionsSSH, makeClusterFunctionsTorque, makeClusterFunctions

---

makeClusterFunctionsLocal

Create cluster functions for sequential execution on local host.

Description

All jobs executed under these cluster functions are executed sequentially, but in an independent, new R session. That is, submitJob does not return until the job has finished. The main use of this ClusterFunctions implementation is to test and debug programs on a local computer.

Listing jobs returns an empty vector (as no jobs can be running when you call this) and killJob returns at once (for the same reason).

Usage

```
makeClusterFunctionsLocal()
```

Value

ClusterFunctions .

See Also

Other clusterFunctions: makeClusterFunctionsInteractive, makeClusterFunctionsLSF, makeClusterFunctionsMult
makeClusterFunctionsOpenLava, makeClusterFunctionsSGE, makeClusterFunctionsSLURM,
makeClusterFunctionsSSH, makeClusterFunctionsTorque, makeClusterFunctions

---

makeClusterFunctionsLSF

Create cluster functions for LSF systems.

Description

Job files are created based on the brew template `template.file`. This file is processed with brew and then submitted to the queue using the `bsub` command. Jobs are killed using the `bkill` command and the list of running jobs is retrieved using `bjobs -u $USER -w`. The user must have the appropriate privileges to submit, delete and list jobs on the cluster (this is usually the case).

The template file can access all arguments passed to the `submitJob` function, see here ClusterFunctions. It is the template file's job to choose a queue for the job and handle the desired resource allocations. Examples can be found on https://github.com/tudo-r/BatchJobs/tree/master/examples/cfLSF.

Usage

```
makeClusterFunctionsLSF(template.file, list.jobs.cmd = c("bjobs",
  "-u $USER", "-w"))
```

Arguments

`template.file`	[character(1)] Path to a brew template file that is used for the PBS job file.
`list.jobs.cmd`	[character] Change default system command / options to list jobs. The first entry is the command, the following the options. See system3.

Value

ClusterFunctions .

See Also

Other clusterFunctions: makeClusterFunctionsInteractive, makeClusterFunctionsLocal, makeClusterFunctionsMu
makeClusterFunctionsOpenLava, makeClusterFunctionsSGE, makeClusterFunctionsSLURM,
makeClusterFunctionsSSH, makeClusterFunctionsTorque, makeClusterFunctions

---

`makeClusterFunctionsMulticore` *Use multiple cores on local Linux machine to spawn parallel jobs.*

Description

Jobs are spawned by starting multiple R sessions on the commandline (similar like on true batch systems). Packages `parallel` or `multicore` are not used in any way.

Usage

```
makeClusterFunctionsMulticore(ncpus = max(getOption("mc.cores",
  parallel::detectCores()) - 1L, 1L), max.jobs, max.load, nice,
  r.options = c("--no-save", "--no-restore", "--no-init-file",
  "--no-site-file"), script)
```

Arguments

`ncpus`	[integer(1)] Number of VPUs of worker. Default is to use all cores but one, where total number of cores "available" is given by option `mc.cores` and if that is not set it is inferred by `detectCores`.
`max.jobs`	[integer(1)] Maximal number of jobs that can run concurrently for the current registry. Default is `ncpus`.
`max.load`	[numeric(1)] Load average (of the last 5 min) at which the worker is considered occupied, so that no job can be submitted. Default is inferred by `detectCores`, cf. argument `ncpus`.
`nice`	[integer(1)] Process priority to run R with set via nice. Integers between -20 and 19 are allowed. If missing, processes are not nice'd and the system default applies (usually 0).
`r.options`	[character] Options for R and Rscript, one option per element of the vector, a la "–vanilla". Default is `c("--no-save", "--no-restore", "--no-init-file", "--no-site-file")`.
`script`	[character(1)] Path to helper bash script which interacts with the worker. You really should not have to touch this, as this would imply that we have screwed up and published an incompatible version for your system. This option is only provided as a last resort for very experienced hackers. Note that the path has to be absolute. This

is what is done in the package: https://github.com/tudo-r/BatchJobs/blob/master/inst/bin/linux-helper Default means to take it from package directory.

Value

ClusterFunctions .

See Also

Other clusterFunctions: makeClusterFunctionsInteractive, makeClusterFunctionsLSF, makeClusterFunctionsLoca makeClusterFunctionsOpenLava, makeClusterFunctionsSGE, makeClusterFunctionsSLURM, makeClusterFunctionsSSH, makeClusterFunctionsTorque, makeClusterFunctions

---

makeClusterFunctionsOpenLava

Create cluster functions for OpenLava systems.

Description

Job files are created based on the brew template `template.file`. This file is processed with brew and then submitted to the queue using the `bsub` command. Jobs are killed using the `bkill` command and the list of running jobs is retrieved using `bjobs -u $USER -w`. The user must have the appropriate privileges to submit, delete and list jobs on the cluster (this is usually the case).

The template file can access all arguments passed to the `submitJob` function, see here ClusterFunctions. It is the template file's job to choose a queue for the job and handle the desired resource allocations. Examples can be found on https://github.com/tudo-r/BatchJobs/tree/master/examples/cfOpenLava.

Usage

```
makeClusterFunctionsOpenLava(template.file, list.jobs.cmd = c("bjobs",
  "-u $USER", "-w"))
```

Arguments

`template.file`	[character(1)] Path to a brew template file that is used for the PBS job file.
`list.jobs.cmd`	[character] Change default system command / options to list jobs. The first entry is the command, the following the options. See system3.

Value

ClusterFunctions .

See Also

Other clusterFunctions: makeClusterFunctionsInteractive, makeClusterFunctionsLSF, makeClusterFunctionsLoca
makeClusterFunctionsMulticore, makeClusterFunctionsSGE, makeClusterFunctionsSLURM,
makeClusterFunctionsSSH, makeClusterFunctionsTorque, makeClusterFunctions

---

`makeClusterFunctionsSGE`

Create cluster functions for Sun Grid Engine systems.

Description

Job files are created based on the brew template `template.file`. This file is processed with brew and then submitted to the queue using the `qsub` command. Jobs are killed using the `qdel` command and the list of running jobs is retrieved using `qselect`. The user must have the appropriate privileges to submit, delete and list jobs on the cluster (this is usually the case).

The template file can access all arguments passed to the `submitJob` function, see here `ClusterFunctions`. It is the template file's job to choose a queue for the job and handle the desired resource allocations. Examples can be found on https://github.com/tudo-r/BatchJobs/tree/master/examples/cfSGE.

Usage

```
makeClusterFunctionsSGE(template.file, list.jobs.cmd = c("qstat",
  "-u $USER"))
```

Arguments

`template.file`	[character(1)] Path to a brew template file that is used for the PBS job file.
`list.jobs.cmd`	[character] Change default system command / options to list jobs. The first entry is the command, the following the options. See `system3`.

Value

`ClusterFunctions` .

See Also

Other clusterFunctions: makeClusterFunctionsInteractive, makeClusterFunctionsLSF, makeClusterFunctionsLoca
makeClusterFunctionsMulticore, makeClusterFunctionsOpenLava, makeClusterFunctionsSLURM,
makeClusterFunctionsSSH, makeClusterFunctionsTorque, makeClusterFunctions

makeClusterFunctionsSLURM *Create cluster functions for SLURM-based systems.*

Description

Job files are created based on the brew template `template.file`. This file is processed with brew and then submitted to the queue using the `sbatch` command. Jobs are killed using the `scancel` command and the list of running jobs is retrieved using `squeue`. The user must have the appropriate privileges to submit, delete and list jobs on the cluster (this is usually the case).

The template file can access all arguments passed to the `submitJob` function, see here `ClusterFunctions`. It is the template file's job to choose a queue for the job and handle the desired resource allocations. Examples can be found on https://github.com/tudo-r/BatchJobs/tree/master/examples/cfSLURM.

Usage

```
makeClusterFunctionsSLURM(template.file, list.jobs.cmd = c("squeue",
  "-h", "-o %i", "-u $USER"))
```

Arguments

`template.file`	[`character(1)`] Path to a brew template file that is used for the PBS job file.
`list.jobs.cmd`	[`character`] Change default system command / options to list jobs. The first entry is the command, the following the options. See `system3`.

Value

`ClusterFunctions` .

See Also

Other clusterFunctions: `makeClusterFunctionsInteractive`, `makeClusterFunctionsLSF`, `makeClusterFunctionsLoca`
`makeClusterFunctionsMulticore`, `makeClusterFunctionsOpenLava`, `makeClusterFunctionsSGE`,
`makeClusterFunctionsSSH`, `makeClusterFunctionsTorque`, `makeClusterFunctions`

makeClusterFunctionsSSH *Create an SSH cluster to execute jobs.*

Description

Worker nodes must share the same file system and be accessible by ssh without manually entering passwords (e.g. by ssh-agent or passwordless pubkey). Note that you can also use this function to parallelize on multiple cores on your local machine. But you still have to run an ssh server and provide passwordless access to localhost.

Usage

```
makeClusterFunctionsSSH(..., workers)
```

Arguments

...	[SSHWorker] Worker objects, all created with makeSSHWorker.
workers	[list of SSHWorker] Alternative way to pass workers.

Value

ClusterFunctions .

See Also

makeSSHWorker

Other clusterFunctions: makeClusterFunctionsInteractive, makeClusterFunctionsLSF, makeClusterFunctionsLoca
makeClusterFunctionsMulticore, makeClusterFunctionsOpenLava, makeClusterFunctionsSGE,
makeClusterFunctionsSLURM, makeClusterFunctionsTorque, makeClusterFunctions

Examples

```
## Not run:

# Assume you have three nodes larry, curley and moe. All have 6
# cpu cores. On curley and moe R is installed under
# "/opt/R/R-current" and on larry R is installed under
# "/usr/local/R/". larry should not be used extensively because
# somebody else wants to compute there as well.
# Then a call to 'makeClusterFunctionsSSH'
# might look like this:

cluster.functions = makeClusterFunctionsSSH(
  makeSSHWorker(nodename = "larry", rhome = "/usr/local/R", max.jobs = 2),
  makeSSHWorker(nodename = "curley", rhome = "/opt/R/R-current"),
```

```
  makeSSHWorker(nodename = "moe", rhome = "/opt/R/R-current"))

## End(Not run)
```

`makeClusterFunctionsTorque` *Create cluster functions for torque-based systems.*

Description

Job files are created based on the brew template `template.file`. This file is processed with brew and then submitted to the queue using the qsub command. Jobs are killed using the `qdel` command and the list of running jobs is retrieved using `qselect`. The user must have the appropriate privileges to submit, delete and list jobs on the cluster (this is usually the case).

The template file can access all arguments passed to the `submitJob` function, see here `ClusterFunctions`. It is the template file's job to choose a queue for the job and handle the desired resource allocations. Examples can be found on https://github.com/tudo-r/BatchJobs/tree/master/examples/cfTorque.

Usage

```
makeClusterFunctionsTorque(template.file, list.jobs.cmd = c("qselect",
  "-u $USER", "-s EHQRTW"))
```

Arguments

`template.file`	[character(1)] Path to a brew template file that is used for the PBS job file.
`list.jobs.cmd`	[character] Change default system command / options to list jobs. The first entry is the command, the following the options. See `system3`.

Value

`ClusterFunctions` .

See Also

Other clusterFunctions: `makeClusterFunctionsInteractive`, `makeClusterFunctionsLSF`, `makeClusterFunctionsLoca`
`makeClusterFunctionsMulticore`, `makeClusterFunctionsOpenLava`, `makeClusterFunctionsSGE`,
`makeClusterFunctionsSLURM`, `makeClusterFunctionsSSH`, `makeClusterFunctions`

makeJob *Creates a job description.*

Description

Usually you will not do this manually. Every object is a list that contains the passed arguments of the constructor.

Usage

```
makeJob(id = NA_integer_, fun, fun.id = digest(fun), pars, name, seed)
```

Arguments

id	[integer(1)] Job id, determined by DB autoincrement. Default is NA.
fun	[function] Job function to apply on parameters.
fun.id	[character(1)] Id used to store function on disk. Default is digest(fun).
pars	[list] Parameter list for job function.
name	[character(1)] Alias name for this job.
seed	[integer(1)] Random seed for job.

makeRegistry *Construct a registry object.*

Description

Note that if you don't want links in your paths (file.dir, work.dir) to get resolved and have complete control over the way the path is used internally, pass an absolute path which begins with "/".

Usage

```
makeRegistry(id, file.dir, sharding = TRUE, work.dir,
  multiple.result.files = FALSE, seed, packages = character(0L),
  src.dirs = character(0L), src.files = character(0L), skip = TRUE)
```

Arguments

id	[character(1)] Name of registry. Displayed e.g. in mails or in cluster queue.
file.dir	[character(1)] Path where files regarding the registry / jobs should be saved. Default is "<id>-files" in current working directory if id is set.
sharding	[logical(1)] Enable sharding to distribute result files into different subdirectories? Important if you have many experiments. Default is TRUE.
work.dir	[character(1)] Working directory for R process when experiment is executed. Default is the current working directory when registry is created.
multiple.result.files	[logical(1)] Should a result file be generated for every list element of the returned list of the job function? Note that the function provided to batchMap or batchReduce must return a named list if this is set to TRUE. The result file will be named "<id>-result-<element name>.RData" instead of "<id>-result.RData". Default is FALSE.
seed	[integer(1)] Start seed for experiments. The first experiment in the registry will use this seed, for the subsequent ones the seed is incremented by 1. Default is a random number from 1 to .Machine$integer.max/2.
packages	[character] Packages that will always be loaded on each node. Default is character(0).
src.dirs	[character] Directories containing R scripts to be sourced on registry load (both on slave and master). Files not matching the pattern "\.[Rr]$" are ignored. Useful if you have many helper functions that are needed during the execution of your jobs. These files should only contain function definitions and no executable code. Default is character(0).
src.files	[character] R scripts files to be sourced on registry load (both on slave and master). Useful if you have many helper functions that are needed during the execution of your jobs. These files should only contain function and constant definitions and no long running, executable code. These paths are considered to be relative to your work.dir. As a last remedy in problematic cases you can use absolute paths, by passing paths that start with "/", see the comment about file.dir and work.dir above, where we allow the same thing. Note that this is a less portable approach and therefore usually a less good idea. Default is character(0).
skip	[logical(1)] Skip creation of a new registry if a registry is found in file.dir. Defaults to TRUE.

Details

Every object is a list that contains the passed arguments of the constructor.

Value

Registry

Examples

```
reg = makeRegistry(id = "BatchJobsExample", file.dir = tempfile(), seed = 123)
print(reg)
```

---

makeSSHWorker *Create SSH worker for SSH cluster functions.*

Description

Create SSH worker for SSH cluster functions.

Usage

```
makeSSHWorker(nodename, ssh.cmd = "ssh", ssh.args = character(0L),
  rhome = "", ncpus, max.jobs, max.load, nice,
  r.options = c("--no-save", "--no-restore", "--no-init-file",
  "--no-site-file"), script)
```

Arguments

nodename	[character(1)] Host name of node.
ssh.cmd	[character(1))] CLI command to ssh into remote node. Default is "ssh".
ssh.args	[character] CLI args for ssh.cmd. Default is none.
rhome	[character(1)] Path to R installation on worker. "" means R installation on the PATH is used, of course this implies that it must be on the PATH (also for non-interactive shells)! Default is "".
ncpus	[integers(1)] Number of VPUs of worker. Default means to query the worker via "/proc/cpuinfo".
max.jobs	[integer(1)] Maximal number of jobs that can run concurrently for the current registry. Default is ncpus.
max.load	[numeric(1)] Load average (of the last 5 min) at which the worker is considered occupied, so that no job can be submitted. Default is ncpus-1.

nice [integer(1)]
Process priority to run R with set via nice. Integers between -20 and 19 are allowed. If missing, processes are not nice'd and the system default applies (usually 0). Default is no niceing.

r.options [character]
Options for R and Rscript, one option per element of the vector, a la "–vanilla". Default is `c("--no-save", "--no-restore", "--no-init-file", "--no-site-file")`.

script [character(1)]
Path to helper bash script which interacts with the worker. You really should not have to touch this, as this would imply that we have screwed up and published an incompatible version for your system. This option is only provided as a last resort for very experienced hackers. Note that the path has to be absolute. This is what is done in the package: https://github.com/tudo-r/BatchJobs/blob/master/inst/bin/linux-helper Default means to take it from package directory.

Value

SSHWorker .

---

makeSubmitJobResult *Create a SubmitJobResult object.*

Description

Use this function in your implementation of makeClusterFunctions to create a return value for the submitJob function.

Usage

```
makeSubmitJobResult(status, batch.job.id, msg, ...)
```

Arguments

status [integer(1)]
Launch status of job. 0 means success, codes bewteen 1 and 100 are temporary errors and any error greater than 100 is a permanent failure.

batch.job.id [character(1)]
Unique id of this job on batch system. Note that this is not the usual job id used in BatchJobs! Must be globally unique so that the job can be terminated using just this information.

msg [character(1)]
Optional error message in case status is not equal to 0. Default is "OK", "TEMPERR", "ERROR", depending on status.

... [any]
Currently unused.

Value

SubmitJobResult . A list, containing status, batch.job.id and msg.

---

reduceResults *Reduce results from result directory.*

Description

The following functions provide ways to reduce result files into either specific R objects (like vectors, lists, matrices or data.frames) or to arbitrarily aggregate them, which is a more general operation.

Usage

```
reduceResults(reg, ids, part = NA_character_, fun, init, impute.val,
  progressbar = TRUE, ...)

reduceResultsList(reg, ids, part = NA_character_, fun, ...,
  use.names = "ids", impute.val, progressbar = TRUE)

reduceResultsVector(reg, ids, part = NA_character_, fun, ...,
  use.names = "ids", impute.val)

reduceResultsMatrix(reg, ids, part = NA_character_, fun, ...,
  rows = TRUE, use.names = "ids", impute.val)

reduceResultsDataFrame(reg, ids, part = NA_character_, fun, ...,
  use.names = "ids", impute.val,
  strings.as.factors = default.stringsAsFactors())

reduceResultsDataTable(reg, ids, part = NA_character_, fun, ...,
  use.names = "ids", impute.val)
```

Arguments

reg	[Registry] Registry.
ids	[integer] Ids of selected jobs. Default is all jobs for which results are available.
part	[character] Only useful for multiple result files, then defines which result file part(s) should be loaded. NA means all parts are loaded, which is the default.
fun	[function] For reduceResults, a function function(aggr, job, res, ...) to reduce things, for all others, a function function(job, res, ...) to select stuff. Here, job is the current job descriptor (see Job), result is the current result object and aggr are the so far aggregated results. When using reduceResults,

your function should add the stuff you want to have from job and result to aggr and return that. When using the other reductions, you should select the stuff you want to have from job and result and return something that can be coerced to an element of the selected return data structure (reasonable conversion is tried internally). Default behavior for this argument is to return res, except for reduceResults where no default is available.

init
: [ANY]
Initial element, as used in Reduce. Default is first result.

impute.val
: [any]
For reduceResults: If not missing, the value of impute.val is passed to function fun as argument res for jobs with missing results.
For the specialized reduction functions reduceResults[Type]: If not missing, impute.val is used as a replacement for the return value of fun on missing results.

progressbar
: [logical(1)]
Set to FALSE to disable the progress bar. To disable all progress bars, see makeProgressBar.

...
: [any]
Additional arguments to fun.

use.names
: [character(1)]
Name the results with job ids ("ids"), stored job names ("names") or return a unnamed result ("none"). Default is ids.

rows
: [logical(1)]
Should the selected vectors be used as rows (or columns) in the result matrix? Default is TRUE.

strings.as.factors
: [logical(1)] Should all character columns in result be converted to factors? Default is default.stringsAsFactors().

Value

Aggregated results, return type depends on function. If ids is empty: reduceResults returns init (if available) or NULL, reduceResultsVector returns c(), reduceResultsList returns list(), reduceResultsMatrix returns matrix(0,0,0), reduceResultsDataFrame returns data.frame().

Examples

```
# generate results:
reg = makeRegistry(id = "BatchJobsExample", file.dir = tempfile(), seed = 123)
f = function(x) x^2
batchMap(reg, f, 1:5)
submitJobs(reg)
waitForJobs(reg)

# reduce results to a vector
reduceResultsVector(reg)
# reduce results to sum
reduceResults(reg, fun = function(aggr, job, res) aggr+res)
```

```
reg = makeRegistry(id = "BatchJobsExample", file.dir = tempfile(), seed = 123)
f = function(x) list(a = x, b = as.character(2*x), c = x^2)
batchMap(reg, f, 1:5)
submitJobs(reg)
waitForJobs(reg)

# reduce results to a vector
reduceResultsVector(reg, fun = function(job, res) res$a)
reduceResultsVector(reg, fun = function(job, res) res$b)
# reduce results to a list
reduceResultsList(reg)
# reduce results to a matrix
reduceResultsMatrix(reg, fun = function(job, res) res[c(1,3)])
reduceResultsMatrix(reg, fun = function(job, res) c(foo = res$a, bar = res$c), rows = TRUE)
reduceResultsMatrix(reg, fun = function(job, res) c(foo = res$a, bar = res$c), rows = FALSE)
# reduce results to a data.frame
print(str(reduceResultsDataFrame(reg)))
# reduce results to a sum
reduceResults(reg, fun = function(aggr, job, res) aggr+res$a, init = 0)
```

`removeRegistry` *Remove a registry object.*

Description

If there are no live/running jobs, the registry will be closed and all of its files will be removed from the file system. If there are live/running jobs, an informative error is generated. The default is to prompt the user for confirmation.

Usage

```
removeRegistry(reg, ask = c("yes", "no"))
```

Arguments

`reg`	[Registry] Registry.
`ask`	[character(1)] If `"yes"` the user is prompted to confirm the action. If trying to prompt the user this way in a non-interactive session, then an informative error is generated. If `"no"`, the registry will be removed without further confirmation.

Value

`logical[1]`

`removeRegistryPackages` *Remove packages from registry.*

Description

Mutator function for packages in `makeRegistry`.

Usage

```
removeRegistryPackages(reg, packages)
```

Arguments

`reg`	[`Registry`] Registry.
`packages`	[`character`] Packages to remove from registry.

Value

`Registry` . Changed registry.

See Also

Other exports: `addRegistryPackages`, `addRegistrySourceDirs`, `addRegistrySourceFiles`, `batchExport`, `batchUnexport`, `loadExports`, `removeRegistrySourceDirs`, `removeRegistrySourceFiles`, `setRegistryPackages`

`removeRegistrySourceDirs` *Remove packages from registry.*

Description

Mutator function for `src.dirs` in `makeRegistry`.

Usage

```
removeRegistrySourceDirs(reg, src.dirs)
```

Arguments

`reg`	[`Registry`] Registry.
`src.dirs`	[`character`] Paths to remove from registry.

Value

Registry . Changed registry.

See Also

Other exports: addRegistryPackages, addRegistrySourceDirs, addRegistrySourceFiles, batchExport, batchUnexport, loadExports, removeRegistryPackages, removeRegistrySourceFiles, setRegistryPackages

---

removeRegistrySourceFiles

Remove source files from registry.

Description

Mutator function for src.files in makeRegistry.

Usage

```
removeRegistrySourceFiles(reg, src.files)
```

Arguments

reg	[Registry] Registry.
src.files	[character] Paths to remove from registry.

Value

Registry . Changed registry.

See Also

Other exports: addRegistryPackages, addRegistrySourceDirs, addRegistrySourceFiles, batchExport, batchUnexport, loadExports, removeRegistryPackages, removeRegistrySourceDirs, setRegistryPackages

`resetJobs` *Reset computational state of jobs.*

Description

Reset state of jobs in the database. Useful under two circumstances: Either to re-submit them because of changes in e.g. external data or to resolve rare issues when jobs are killed in an unfortunate state and therefore blocking your registry.

The function internally lists all jobs on the batch system and if those include some of the jobs you want to reset, it informs you to kill them first by raising an exception. If you really know what you are doing, you may set `force` to `TRUE` to omit this sanity check. Note that this is a dangerous operation to perform which may harm the database integrity. In this case you HAVE to make externally sure that none of the jobs you want to reset are still running.

Usage

```
resetJobs(reg, ids, force = FALSE)
```

Arguments

`reg`	[Registry] Registry.
`ids`	[integer] Ids of jobs to kill. Default is none.
`force`	[logical(1)] Reset jobs without checking whether they are currently running. READ THE DETAILS SECTION! Default is `FALSE`.

Value

Vector of reseted job ids.

See Also

Other debug: `debugMulticore`, `debugSSH`, `getErrorMessages`, `getJobInfo`, `getLogFiles`, `grepLogs`, `killJobs`, `setJobFunction`, `showLog`, `testJob`

sanitizePath *Sanitize a path*

Description

Replaces backward slashes with forward slashes and optionally normalizes the path.

Usage

```
sanitizePath(path, make.absolute = TRUE, normalize.absolute = FALSE)
```

Arguments

path	[character] Vector of paths to sanitize.
make.absolute	[logical] If TRUE convert to an absolute path.
normalize.absolute	[logical] Also call normalizePath on absolute paths? This will immediately resolve symlinks.

Value

character with sanitized paths.

setConfig *Set and overwrite configuration settings*

Description

Set and overwrite configuration settings

Usage

```
setConfig(conf = list(), ...)
```

Arguments

conf	[Config or list] List of configuration parameters as returned by loadConfig or getConfig.
...	[ANY] Named configuration parameters. Overwrites parameters in conf, if provided.

Value

Invisibly returns a list of configuration settings.

See Also

Other conf: configuration, getConfig, loadConfig

`setJobFunction` *Sets the job function for already existing jobs.*

Description

Use this function only as last measure when there is a bug in a part of your job function and you have already computed a large number of (unaffected) results. This function allows you to fix the error and to associate the jobs with the corrected function.

Note that by default the computational state of the affected jobs is also reset.

Usage

```
setJobFunction(reg, ids, fun, more.args = list(), reset = TRUE,
  force = FALSE)
```

Arguments

`reg`	[Registry] Registry.
`ids`	[integer] Ids of jobs. Default is all jobs.
`fun`	[function] Replacement function.
`more.args`	[list] A list of other arguments passed to `fun`. Default is empty list.
`reset`	[logical(1)] Reset job status via `resetJobs`. Default is `TRUE`.
`force`	[logical(1)] See `resetJobs`. Default is `FALSE`.

Value

Nothing.

See Also

Other debug: debugMulticore, debugSSH, getErrorMessages, getJobInfo, getLogFiles, grepLogs, killJobs, resetJobs, showLog, testJob

setJobNames *Set job names.*

Description

Set job names.

Usage

```
setJobNames(reg, ids, jobnames)
```

Arguments

reg	[Registry] Registry.
ids	[integer] Ids of jobs. Default is all jobs.
jobnames	[character] Character vector with length equal to `length(ids)`. `NA` removes the names stored in the registry. A single `NA` is replicated to match the length of ids provided.

Value

Named vector of job ids.

setRegistryPackages *Set packages for a registry.*

Description

Mutator function for packages in `makeRegistry`.

Usage

```
setRegistryPackages(reg, packages)
```

Arguments

reg	[Registry] Registry.
packages	[character] Character vector of package names to load.

Value

Registry . Changed registry.

See Also

Other exports: addRegistryPackages, addRegistrySourceDirs, addRegistrySourceFiles, batchExport, batchUnexport, loadExports, removeRegistryPackages, removeRegistrySourceDirs, removeRegistrySourceFiles

---

showClusterStatus *Show information about available computational resources on cluster.*

Description

Currently only supported for multicore and SSH mode. Displays: Name of node, current load, number of running R processes, number of R processes with more than 50 The latter counts either jobs belonging to reg or all BatchJobs jobs if reg was not passed.

Usage

```
showClusterStatus(reg)
```

Arguments

reg	[Registry] Registry. Must not be passed and this is the default.

Value

data.frame .

---

showLog *Display the contents of a log file.*

Description

Display the contents of a log file, useful in case of errors.

Note this rare special case: When you use chunking, submit some jobs, some jobs fail, then you resubmit these jobs again in different chunks, the log files will contain the log of the old, failed job as well. showLog tries to jump to the correct part of the new log file with a supported pager.

Usage

```
showLog(reg, id, pager = getOption("pager"))
```

Arguments

reg	[Registry] Registry.
id	[integer(1)] Id of selected job. Default is first id in registry.
pager	[any] Pager to use to display the log. Defaults to getOption("pager"). This option is passed to file.show and is highly OS dependant and GUI dependant. If either R's pager or the environment variable "PAGER" is set to "less" or "vim", the correct part of the log file will be shown. Otherwise you find information about the correct part in the beginning of the displayed file.

Value

character(1) . Invisibly returns path to log file.

See Also

Other debug: debugMulticore, debugSSH, getErrorMessages, getJobInfo, getLogFiles, grepLogs, killJobs, resetJobs, setJobFunction, testJob

---

showStatus *Retrieve or show status information about jobs.*

Description

E.g.: How many there are, how many are done, any errors, etc. showStatus displays on the console, getStatus returns an informative result without console output.

Usage

```
showStatus(reg, ids, run.and.exp = TRUE, errors = 10L)

getStatus(reg, ids, run.and.exp = TRUE)
```

Arguments

reg	[Registry] Registry.
ids	[integer] Ids of selected jobs. Default is all jobs.
run.and.exp	[logical(1)] Show running and expired jobs? Requires to list the job on the batch system. If not possible, because that cluster function is not avilable, this option is ignored anyway. Default is TRUE.

errors [integer(1)]
How many of the error messages should be displayed if any errors occured in the jobs? Default is 10.

Value

list . List of absolute job numbers. showStatus returns them invisibly.

Examples

```
reg = makeRegistry(id = "BatchJobsExample", file.dir = tempfile(), seed = 123)
f = function(x) x^2
batchMap(reg, f, 1:10)
submitJobs(reg)
waitForJobs(reg)

# should show 10 submitted jobs, which are all done.
showStatus(reg)
```

---

sourceRegistryFiles *Source registry files*

Description

Sources all files found in src.dirs and specified via src.files.

Usage

```
sourceRegistryFiles(reg, envir = .GlobalEnv)
```

Arguments

reg [Registry]
Registry.

envir [environment]
Environment to source the files into. Default is the global environment.

Value

Nothing.

submitJobs *Submit jobs or chunks of jobs to batch system via cluster function.*

Description

If the internal submit cluster function completes successfully, the `retries` counter is set back to 0 and the next job or chunk is submitted. If the internal submit cluster function returns a fatal error, the submit process is completely stopped and an exception is thrown. If the internal submit cluster function returns a temporary error, the submit process waits for a certain time, which is determined by calling the user-defined `wait`-function with the current `retries` counter, the counter is increased by 1 and the same job is submitted again. If `max.retries` is reached the function simply terminates.

Potential temporary submit warnings and errors are logged inside your file directory in the file "submit.log". To keep track you can use `tail -f [file.dir]/submit.log` in another terminal.

Usage

```
submitJobs(reg, ids, resources = list(), wait, max.retries = 10L,
  chunks.as.arrayjobs = FALSE, job.delay = FALSE, progressbar = TRUE)
```

Arguments

`reg`	[Registry] Registry.
`ids`	[integer] Vector for job id or list of vectors of chunked job ids. Only corresponding jobs are submitted. Chunked jobs will get executed sequentially as a single job for the scheduler. Default is all jobs which were not yet submitted to the batch system.
`resources`	[list] Required resources for all batch jobs. The elements of this list (e.g. something like "walltime" or "nodes" are defined by your template job file. Defaults can be specified in your config file. Default is empty list.
`wait`	[function(retries)] Function that defines how many seconds should be waited in case of a temporary error. Default is exponential back-off with `10*2^retries`.
`max.retries`	[integer(1)] Number of times to submit one job again in case of a temporary error (like filled queues). Each time `wait` is called to wait a certain number of seconds. Default is 10 times.
`chunks.as.arrayjobs`	[logical(1)] If ids are passed as a list of chunked job ids, execute jobs in a chunk as array jobs. Note that your scheduler and your template must be adjusted to use this option. Default is `FALSE`.

job.delay [function(n, i) or logical(1)]
Function that defines how many seconds a job should be delayed before it starts. This is an expert option and only necessary to change when you want submit extremely many jobs. We then delay the jobs a bit to write the submit messages as early as possible to avoid writer starvation. n is the number of jobs and i the number of the ith job. The default function used with job.delay set to TRUE is no delay for 100 jobs or less and otherwise runif(1, 0.1*n, 0.2*n). If set to FALSE (the default) delaying jobs is disabled.

progressbar [logical(1)]
Set to FALSE to disable the progress bar. To disable all progress bars, see makeProgressBar.

Value

integer . Vector of submitted job ids.

Examples

```
reg = makeRegistry(id = "BatchJobsExample", file.dir = tempfile(), seed = 123)
f = function(x) x^2
batchMap(reg, f, 1:10)
submitJobs(reg)
waitForJobs(reg)

# Submit the 10 jobs again, now randomized into 2 chunks:
chunked = chunk(getJobIds(reg), n.chunks = 2, shuffle = TRUE)
submitJobs(reg, chunked)
```

---

sweepRegistry *Sweep obsolete files from the file system.*

Description

Removes R scripts, log files, resource informations and temporarily stored configuration files from the registry's file directory. Assuming all your jobs completed successfully, none of these are needed for further work. This operation potentially releases quite a lot of disk space, depending on the number of your jobs. BUT A HUGE WORD OF WARNING: IF you later notice something strange and need to determine the reason for it, you are at a huge disadvantage. Only do this at your own risk and when you are sure that you have successfully completed a project and only want to archive your produced experiments and results.

Usage

```
sweepRegistry(reg, sweep = c("scripts", "conf"))
```

Arguments

reg	[Registry] Registry.
sweep	[character] Possible choices: Temporary R scripts of jobs, really not needed for anything else then execution ("scripts"), log file of jobs, think about whether you later want to inspect them ("logs"), BatchJobs configuration files which are temporarily stored on submit, really not needed for anything else then execution ("conf"), resource lists of submitJobs which are temporarily stored on submit, think about whether you later want to inspect them ("resources"), Default is c("scripts", "conf").

Value

logical . Invisibly returns TRUE on success and FALSE if some files could not be removed.

---

syncRegistry *Syncronize staged queries into the registry.*

Description

If the option "staged.queries" is enabled, all communication from the nodes to the master is done via files in the subdirectory "pending" of the file.dir. This function checks for such files and merges the information into the database. Usually you do not have to call this function yourself.

Usage

```
syncRegistry(reg)
```

Arguments

reg	[Registry] Registry.

Value

Invisibly returns TRUE on success.

testJob *Tests a job by running it with Rscript in a new process.*

Description

Useful for debugging. Note that neither the registry, database or file directory are changed.

Usage

```
testJob(reg, id, resources = list(), external = TRUE)
```

Arguments

reg	[Registry] Registry.
id	[integer(1)] Id of job to test. Default is first job id of registry.
resources	[list] Usually not needed, unless you call the function getResources in your job. See submitJobs. Default is empty list.
external	[logical(1)] Run test in an independent external R session instead of current. The former allows to uncover missing variable definitions (which may accidentially be defined in the current global environment) and the latter is useful to get traceable execeptions. Default is TRUE.

Value

any . Result of job. If the job did not complete because of an error, NULL is returned.

See Also

Other debug: debugMulticore, debugSSH, getErrorMessages, getJobInfo, getLogFiles, grepLogs, killJobs, resetJobs, setJobFunction, showLog

Examples

```
reg = makeRegistry(id = "BatchJobsExample", file.dir = tempfile(), seed = 123)
f = function(x) if (x==1) stop("oops") else x
batchMap(reg, f, 1:2)
testJob(reg, 2)
```

`waitForJobs` *Wait for termination of jobs on the batch system.*

Description

Waits for termination of jobs while displaying a progress bar containing summarizing informations of the jobs. The following abbreviations are used in the progress bar: "S" for number of jobs on system, "D" for number of jobs successfully terminated, "E" for number ofjobs terminated with an R exception and "R" for number of jobs currently running on the system.

Usage

```
waitForJobs(reg, ids, sleep = 10, timeout = 604800,
  stop.on.error = FALSE, progressbar = TRUE)
```

Arguments

`reg`	[Registry] Registry.
`ids`	[integer] Vector of job ids. Default is all submitted jobs not yet terminated.
`sleep`	[numeric(1)] Seconds to sleep between status updates. Default is 10.
`timeout`	[numeric(1)] After waiting `timeout` seconds, show a message and return FALSE. This argument may be required on some systems where, e.g., expired jobs or jobs on hold are problematic to detect. If you don't want a timeout, set this to `Inf`. Default is 604800 (one week).
`stop.on.error`	[logical(1)] Immediately return if a job terminates with an error? Default is FALSE.
`progressbar`	[logical(1)] Set to FALSE to disable the progress bar. To disable all progress bars, see makeProgressBar.

Value

`logical(1)` . Returns TRUE if all jobs terminated successfully and FALSE if either an error occurred or the timeout is reached.

Index