Package 'flowr'

August 15, 2015

Type Package

Title Streamlining Design and Deployment of Complex Workflows

Description An interface to streamline design of complex workflows and their deployment to a High Performance Computing Cluster.

Version 0.9.7.5

Date 2015-08-15

Depends methods,
utils

Imports diagram,
whisker,
params (>= 0.2.4),
tools,
knitr

Suggests reshape2,
ggplot2,
openxlsx,
testthat

VignetteBuilder knitr

URL https://github.com/sahilseth/flowr

BugReports https://github.com/sahilseth/flowr/issues

License MIT + file LICENSE

R topics documented:

`as.flowdef` *flow definition*

Description

Reeading a flow definition file and checking it.

Usage

```
as.flowdef(x)

is.flowdef(x)
```

Arguments

`x`	can be a data.frame or a path for a flow definition file

as.flowmat *flow mat*

Description

as.flowmat(): reads a file and checks for required columns. If x is data.frame checks for required columns.

Usage

```
as.flowmat(x, grp_col, jobname_col, cmd_col, ...)

is.flowmat(x)
```

Arguments

x	a data.frame or path to file with flow details in it.
grp_col	column used for grouping, default samplename.
jobname_col	column specifying jobname, default jobname
cmd_col	column specifying commands to run, default cmd
...	not used

check *Check consistency of flowdef and flowmat*

Description

check consistency of objects Currently checks objects S3 flowdef, flowmat

Usage

```
check(x, ...)

## S3 method for class 'flowmat'
check(x, ...)

## S3 method for class 'flowdef'
check(x, ...)
```

Arguments

x	a flowdef or flowmat object
...	suppled to check.classname function

`check_args` *checks all the arguments in the parent frame. None of them should be null.*

Description

checks all the arguments in the parent frame. None of them should be null.

Usage

```
check_args()
```

`cmds_to_flow` *cmds_to_flow: DEPRECIATED*

Description

Create a flow object from a list of commands

Usage

```
cmds_to_flow(cmd.list, samplename = "", infomat, q_obj = queue(type = "lsf",
  verbose = FALSE), flowname = "stage2", execute = FALSE,
  flow_run_path = "/scratch/iacs/flow_pipe/tmp")
```

Arguments

`cmd.list`	list of commands
`samplename`	name of the sample
`infomat`	flowdef
`q_obj`	queue object
`flowname`	name of the flow
`execute`	TRUE/FALSE
`flow_run_path`	outpath

detect_dep_type *detect_dep_type*

Description

detect_dep_type

Usage

```
detect_dep_type(x, cmds, prev_job)
```

Arguments

x	job object
cmds	a string of commands
prev_job	previous job name

fetch *A generic functions to search for files*

Description

These functions help in searching for specific files in the user's space.

fetch_pipes(): Fetches pipelines in the following places,

- - available in 'pipelines' folders in flowr and ngsflows packages.
- - ~/flowr/pipelines
- - github repos (currently not supported)

fetch_conf(): Fetches configuration files in the following places,

- - available in 'conf' folders in flowr and ngsflows packages.
- - ~/flowr/conf folder

By default flowr loads, ~/flowr/conf/flowr.conf and ~/flowr/conf/ngsflows.conf

Usage

```
fetch(x, places, urls, verbose = FALSE)

fetch_pipes(x, places, last_only = FALSE,
  urls = get_opts("flowr_pipe_urls"), silent = FALSE, ask = TRUE)

fetch_conf(x = "flowr.conf", places, ...)
```

Arguments

x	name of the file to search for
places	places (paths) to look for it. Its best to use the defaults
urls	urls to look for, works well for pipelines.
verbose	be chatty?
last_only	[fetch_pipes only]. If multiple pipelines match the pattern, return the last one.
silent	[fetch_pipes() only]. logical, be silent even if no such pipeline is available.
ask	ask before downloading or copying, not used !
...	not used

Examples

```
{
fetch_conf("torque.sh")
}
```

---

flow *Flow constructor*

Description

Flow constructor

Usage

```
flow(jobs = list(new("job")), name = "newflow", desc = "my_super_flow",
  mode = c("scheduler", "trigger", "R"),
  flow_run_path = get_opts("flow_run_path"), trigger_path = "",
  flow_path = "", version = "0.0", status = "", execute = "")
```

Arguments

jobs	list A list of jobs to be included in this flow
name	character Name of the flow. Defaults to 'newname' Used in submit_flow to name the working directories.
desc	character Description of the flow This is used to name folders (when submitting jobs, see submit_flow). It is good practice to avoid spaces and other special characters. An underscore '_' seems like a good word separator. Defaults to 'my_super_flow'. We usually use this to put sample names of the data.
mode	character Mode of submission of the flow.
flow_run_path	The base path of all the flows you would submit. Defaults to ~/flows. Best practice to ignore it.
trigger_path	character Defaults to ~/flows/trigger. Best practice to ignore it.
flow_path	character
version	version of flowr used to create and execute this flow.
status	character Not used at this time
execute	execuction status of flow object.

Examples

```
cmds = rep("sleep 5", 10)
qobj <- queue(platform='torque')
## run the 10 commands in parallel
jobj1 <- job(q_obj=qobj, cmd = cmds, submission_type = "scatter", name = "job1")

## run the 10 commands sequentially, but WAIT for the previous job to complete
## Many-To-One
jobj2 <- job(q_obj=qobj, cmd = cmds, submission_type = "serial",
 dependency_type = "gather", previous_job = "job1", name = "job2")

## As soon as first job on 'job1' is complete
## One-To-One
jobj3 <- job(q_obj=qobj, cmd = cmds, submission_type = "scatter",
 dependency_type = "serial", previous_job = "job1", name = "job3")

fobj <- flow(jobs = list(jobj1, jobj2, jobj3))

## plot the flow
plot_flow(fobj)
## Not run:
## dry run, only create the structure without submitting jobs
submit_flow(fobj)

## execute the jobs: ONLY works on computing cluster, would fail otherwise
submit_flow(fobj, execute = TRUE)

## End(Not run)
```

`flow-class` *flow defines the class*

Description

flow defines the class

`flowopts` *Default options/params used in ngsflows and flowr*

Description

There are three helper functions which attempt to manage params used by flowr and ngsflows:

- get_opts OR `opts_flow$get`: show all default options
- set_opts OR `opts_flow$set`: set default options
- load_opts OR `opts_flow$load`: load options specified in a tab seperated text file

For more details regarding these funtions refer to params.

Usage

```
flowopts

opts_flow

get_opts(...)

set_opts(...)

load_opts(...)
```

Arguments

...	• get: names of options to fetch • set: a set of options in a name=value format seperated by commas

Format

```
opts_flow
```

Details

By default flowr loads, `~/flowr/conf/flowr.conf` and `~/flowr/conf/ngsflows.conf`

Below is a list of default flowr options, retrieved via

`opts_flow$get()`:

name	value
default_regex	(.*)
flow_base_path	~/flowr
flow_conf_path	~/flowr/conf
flow_parse_lsf	.*(\<[0-9]*\>).*
flow_parse_moab	(.*)
flow_parse_sge	(.*)
flow_parse_slurm	(.*)
flow_parse_torque	(.?)\..*
flow_pipe_paths	~/flowr/pipelines
flow_pipe_urls	~/flowr/pipelines
flow_platform	local
flow_run_path	~/flowr/runs
my_conf_path	~/flowr/conf
my_dir	path/to/a/folder
my_path	~/flowr
my_tool_exe	/usr/bin/ls
verbose	FALSE

Examples

```
## Set options: set_opts()
opts = set_opts(flow_run_path = "~/mypath")
## OR if you would like to supply a long list of options:
opts = set_opts(.dots = list(flow_run_path = "~/mypath"))
```

```
## load options from a configuration file: load_opts()
myconfile = fetch_conf("flowr.conf")
load_opts(myconfile)

## Fetch options: get_opts()
get_opts("flow_run_path")
get_opts()
```

`get_resources` *get_resources*

Description

get_resources currenty this only works on LSF

Usage

```
get_resources(x, odir, ...)
```

Arguments

`x`	A character vector of lenth 1. This may be a parent level folder with directories with multiple flow runs.
`odir`	Output directory to save the results
`...`	other arguments sent to get_resources_lsf

Details

If `x` is a parent level folder, then resources are summarized for all its child folders.

Examples

```
## Not run:
get_resources(x = x, odir = ~/tmp)

## End(Not run)
```

`get_wds` *Get all the (sub)directories in a folder*

Description

Get all the (sub)directories in a folder

Usage

```
get_wds(x)
```

Arguments

`x`	path to a folder

job *job class*

Description

job class

Usage

```
job(cmds = "", name = "myjob", q_obj = new("queue"), previous_job = "",
  cpu = 1, memory, walltime, submission_type = c("scatter", "serial"),
  dependency_type = c("none", "gather", "serial", "burst"), ...)
```

Arguments

cmds	the commands to run
name	name of the job
q_obj	queue object
previous_job	character vector of previous job. If this is the first job, one can leave this empty, NA, NULL, '.', or ''. In future this could specify multiple previous jobs.
cpu	no of cpu's reserved
memory	The amount of memory reserved. Units depend on the platform used to process jobs
walltime	The amount of time reserved for this job. Format is unique to a platform. Typically it looks like 12:00 (12 hours reserved, say in LSF), in Torque etc. we often see measuring in seconds: 12:00:00
submission_type	submission type: A character with values: scatter, serial. Scatter means all the 'cmds' would be run in parallel as seperate jobs. Serial, they would combined into a single job and run one-by-one.
dependency_type	depedency type. One of none, gather, serial, burst. If previous_job is specified, then this would not be 'none'. [Required]
...	other passed onto object creation. Example: memory, walltime, cpu

Examples

```
qobj <- queue(platform="torque")

## torque job with 1 CPU running command 'sleep 2'
jobj <- job(q_obj=qobj, cmd = "sleep 2", cpu=1)

## multiple commands
cmds = rep("sleep 5", 10)

## run the 10 commands in parallel
jobj1 <- job(q_obj=qobj, cmd = cmds, submission_type = "scatter", name = "job1")

## run the 10 commands sequentially, but WAIT for the previous job to complete
jobj2 <- job(q_obj=qobj, cmd = cmds, submission_type = "serial",
```

```
   dependency_type = "gather", previous_job = "job1")

fobj <- flow(jobs = list(jobj1, jobj2))

## plot the flow
plot_flow(fobj)
## Not run:
## dry run, only create the structure without submitting jobs
submit_flow(fobj)

## execute the jobs: ONLY works on computing cluster, would fail otherwise
submit_flow(fobj, execute = TRUE)


## End(Not run)
```

`kill` *kill*

Description

kill

works on flow_path. Reads flow object and calls kill.flow()

works on flow object

Usage

```
kill(x, ...)

## S3 method for class 'character'
kill(x, ...)

## S3 method for class 'flow'
kill(x, kill_cmd, jobid_col = "job_sub_id", ...)
```

Arguments

`x`	either path to flow [character] or fobj object of class flow
`...`	not used
`kill_cmd`	The command used to kill. Default is 'bkill' (LSF). One can used qdel for 'torque', 'sge' etc.
`jobid_col`	Advanced use. The column name in 'flow_details.txt' file used to fetch jobids to kill

Examples

```
## Not run:
## example for terminal
## flowr kill_flow x=path_to_flow_directory

## End(Not run)
```

parse_jobids *parse_jobids*

Description

parse_jobids

Usage

```
parse_jobids(jobids, platform)
```

Arguments

jobids	output from HPCC upon job submission, as a character vector
platform	string specifying the platform. This determines how the jobids are parsed

parse_lsf_out *parse LSF output files*

Description

parse LSF output files

Usage

```
parse_lsf_out(x, scale_time = 1/3600, n = 100,
  time_format = get_opts("time_format"))
```

Arguments

x	file
scale_time	time is usually in seconds, scale of 1/60 shows minutes, 1/3600 shows in hours
n	how many lines to read; usually resources details are on top. 100 works well. .Depreciated
time_format	format of time in the execution logs. This should match the format in lsf/torque etc. shell script templates.

plot_flow *plot_flow*

Description

plot the flow object

plot_flow.character: works on a flowdef file.

Usage

```
plot_flow(x, ...)

## S3 method for class 'flow'
plot_flow(x, ...)

## S3 method for class 'list'
plot_flow(x, ...)

## S3 method for class 'character'
plot_flow(x, ...)

## S3 method for class 'flowdef'
plot_flow(x, detailed = TRUE, type = c("1", "2"),
  pdf = FALSE, pdffile = sprintf("%s.pdf", x@name), ...)
```

Arguments

x	Object of class flow, or a list of flow objects or a flowdef
...	experimental
detailed	include some details
type	1 is original, and 2 is a elipse with less details
pdf	create a pdf instead of plotting interactively
pdffile	output file name for the pdf file

Examples

```
qobj = queue(type="lsf")
cmds = rep("sleep 5", 10)
jobj1 <- job(q_obj=qobj, cmd = cmds, submission_type = "scatter", name = "job1")
jobj2 <- job(q_obj=qobj, name = "job2", cmd = cmds, submission_type = "scatter",
             dependency_type = "serial", previous_job = "job1")
fobj <- flow(jobs = list(jobj1, jobj2))
plot_flow(fobj)

### Gather: many to one relationship
jobj1 <- job(q_obj=qobj, cmd = cmds, submission_type = "scatter", name = "job1")
jobj2 <- job(q_obj=qobj, name = "job2", cmd = cmds, submission_type = "scatter",
             dependency_type = "gather", previous_job = "job1")
fobj <- flow(jobs = list(jobj1, jobj2))
plot_flow(fobj)
```

```
### Burst: one to many relationship
jobj1 <- job(q_obj=qobj, cmd = cmds, submission_type = "serial", name = "job1")
jobj2 <- job(q_obj=qobj, name = "job2", cmd = cmds, submission_type = "scatter",
             dependency_type = "burst", previous_job = "job1")
fobj <- flow(jobs = list(jobj1, jobj2))
plot_flow(fobj)
```

---

queue *Create a* queue *object which containg details about how a job is submitted.*

Description

This function defines the queue used to submit jobs to the cluster. In essence details about the computing cluster in use.

Usage

```
queue(object, platform = c("local", "lsf", "torque", "sge", "moab"),
  format = "", queue = "long", walltime, memory, cpu = 1,
  extra_opts = "", submit_exe, nodes = "1", jobname = "name",
  email = Sys.getenv("USER"), dependency = list(), server = "localhost",
  verbose = FALSE, cwd = "", stderr = "", stdout = "", ...)
```

Arguments

object	this is not used currenlty, ignore.
platform	Required and important. Currently supported values are 'lsf' and 'torque'. [Used by class job]
format	[advanced use] We have a default format for the final command line string generated for 'lsf' and 'torque'.
queue	the type of queue your group usually uses 'bsub' etc.
walltime	max walltime of a job.
memory	The amount of memory reserved. Units depend on the platform used to process jobs
cpu	number of cpus you would like to reserve [Used by class job]
extra_opts	[advanced use] Extra options to be supplied while create the job submission string.
submit_exe	[advanced use] Already defined by 'platform'. The exact command used to submit jobs to the cluster example 'qsub'
nodes	[advanced use] number of nodes you would like to request. Or in case of torque name of the nodes.*optional* [Used by class job]
jobname	[debug use] name of this job in the computing cluster
email	[advanced use] Defaults to system user, you may put you own email though may get tons of them.
dependency	[debug use] a list of jobs to complete before starting this one

server	[not used] This is not implemented currently. This would specify the head node of the computing cluster. At this time submission needs to be done on the head node of the cluster where flow is to be submitted
verbose	[logical] TRUE/FALSE
cwd	[debug use] Ignore
stderr	[debug use] Ignore
stdout	[debug use] Ignore
...	other passed onto object creation. Example: memory, walltime, cpu

Details

Resources: Can be defined **once** using a queue object and recylced to all the jobs in a flow. If resources (like memory, cpu, walltime, queue) are supplied at the job level they overwrite the one supplied in queue Nodes: can be supplied ot extend a job across multiple nodes. This is purely experimental and not supported. ## Server: This a hook which may be implemented in future. ## Submission script: The 'platform' variable defines the format, and submit_exe; however these two are avaible for someone to create a custom submission command.

Examples

```
qobj <- queue(platform='lsf')
```

---

queue-class *queue defines the class*

Description

queue defines the class

---

read_fobj *read flow object given a flow execution folder*

Description

read flow object given a flow execution folder

Usage

```
read_fobj(x)
```

Arguments

x	path to a flow execution folder

Value

if it finds a fobj, returns that. If not return back the path x

`render_dependency` *render dependency Advanced use, for debugging. Or adding a currenltly un-supported platform.*

Description

render dependency Advanced use, for debugging. Or adding a currenltly un-supported platform.

Usage

```
render_dependency(x, ...)
```

Arguments

`x`	is a 'job' object
`...`	not used

`render_queue_cmd` *render_queue_cmd*

Description

render_queue_cmd

Usage

```
render_queue_cmd(jobj, file, index, fobj)
```

Arguments

`jobj`	job object
`file`	path to the output file
`index`	If more than one, which command to focus on. Can be from `1:length(cmds)`
`fobj`	flow object

rerun *rerun*

Description

rerun

Usage

```
rerun(x, ...)

## S3 method for class 'character'
rerun(x, ...)

## S3 method for class 'flow'
rerun(x, mat, def, start_from, execute = TRUE, kill = TRUE,
  ...)
```

Arguments

x	Either path to flow folder or the flow object which has been 'returned' from submit_flow.
...	not used
mat	path to flow_mat. should fetch on the fly
def	path to should fetch on the fly
start_from	which job to start from
execute	[logical] whether to execute or not
kill	logical indicating whether to kill the jobs from old flow

Details

We need path to the flow folder (wd). The flow object needs to have upate 'base_path' slow with wd (the path to the flow folder). Also its important to know that we need details regarding the previous submission from flow_details.txt file. Which should typically be in wd

Examples

```
## Not run:
rerun_flow(wd = wd, fobj = fobj, execute = TRUE, kill = TRUE)

## End(Not run)
```

run *run pipelines*

Description

Running examples flows This wraps a few steps: Get all the commands to run (flow_mat) Create a ‘flow‘ object, using flow_mat and a default flowdef (picked from the same folder). Use ‘submit_flow()‘ to submit this to the cluster.

Usage

```
run(x, platform, def, flow_run_path = get_opts("flow_run_path"),
  execute = FALSE, ...)

run_pipe(x, platform, def, flow_run_path = get_opts("flow_run_path"),
  execute = FALSE, ...)
```

Arguments

x	name of the pipeline to run. This is a function called to create a flow_mat.
platform	what platform to use, overrides flowdef
def	flow definition
flow_run_path	passed onto to_flow. Default it picked up from flowr.conf. Typically this is ~/flowr/runs
execute	TRUE/FALSE
...	passed onto the pipeline function specified in x

setup *Setup and initialize some scripts.*

Description

Setup and initialize some scripts.

Usage

```
setup(bin = "~/bin")
```

Arguments

bin	path to bin folder

Details

Will add more to this to identify cluster and aid in other things

split_multi_dep *split_multi_dep Split rows with multiple dependencies*

Description

split_multi_dep Split rows with multiple dependencies

Usage

```
split_multi_dep(x)
```

Arguments

x	this is a flow def

status *status*

Description

Summarize status of executed flow(x)

Usage

```
status(x, out_format = "markdown")

get_status(x, ...)

## S3 method for class 'character'
get_status(x, out_format = "markdown", ...)

## S3 method for class 'data.frame'
get_status(x, ...)

## S3 method for class 'flow'
get_status(x, out_format = "markdown", ...)
```

Arguments

x	path to the flow root folder or a parent folder to summarize several flows.
out_format	passed onto knitr:::kable. supports: markdown, rst, html...
...	not used

Details

basename(x) is used in a wild card search.

- If x is a path with a single flow, it outputs the status of one flow.
- If the path has more than one flow then this could give a summary of **all** of them.
- Instead if x is supplied with paths to more than one flow, then this individually prints status of each.

Alternatively, x can also be a flow object

Examples

```
## Not run:
status(x = "~/flowr/runs/sleep_pipe*")
## an example for running from terminal
flowr status x=path_to_flow_directory cores=6

## End(Not run)
```

---

submit_flow *submit_flow*

Description

submit_flow

Usage

```
submit_flow(x, verbose = get_opts("verbose"), ...)

## S3 method for class 'list'
submit_flow(x, verbose = get_opts("verbose"), ...)

## S3 method for class 'flow'
submit_flow(x, verbose = get_opts("verbose"),
  execute = FALSE, uuid, plot = TRUE, dump = TRUE, .start_jid = 1, ...)
```

Arguments

x	a object of class flow.
verbose	logical.
...	Advanced use. Any additional parameters are passed on to submit_job function.
execute	logical whether or not to submit the jobs
uuid	character Advanced use. This is the final path used for flow execution. Especially useful in case of re-running a flow.
plot	logical whether to make a pdf flow plot (saves it in the flow working directory).
dump	dump all the flow details to the flow path
.start_jid	Job to start this submission from. Advanced use, should be 1 by default.

Examples

```
## Not run:
submit_flow(fobj = fobj, ... = ...)
## End(Not run)
```

`submit_job` *submit_job*

Description

submit_job

Usage

```
submit_job(jobj, fobj, job_id, execute = FALSE, verbose = FALSE, ...)
```

Arguments

`jobj`	Object of calls job
`fobj`	Object of calls flow
`job_id`	job id
`execute`	A `logical` vector suggesting whether to submit this job
`verbose`	logical
`...`	not used

Examples

```
## Not run:
submit_job(jobj = jobj, fobj = fobj, execute = FALSE,
verbose = TRUE, wd = wd, job_id = job_id)

## End(Not run)
```

`subset_fdef` *subset_fdef*

Description

subset_fdef

Usage

```
subset_fdef(fobj, def, start_from)
```

Arguments

`fobj`	flow object
`def`	flowdef
`start_from`	where to start from

subset_fmat *subset_fmat*

Description

subset_fmat

Usage

```
subset_fmat(fobj, mat, start_from)
```

Arguments

fobj	flow object
mat	a part of flowdef
start_from,	where to start from

test_queue *test_queue*

Description

This function attempts to test the submission of a job to the queue. We would first submit one single job, then submit another with a dependency to see if configuration works. This would create a folder in home called 'flows'.

Usage

```
test_queue(q_obj, verbose = TRUE, ...)
```

Arguments

q_obj	queue object
verbose	toggle
...	These params are passed onto queue. ?queue, for more information

Examples

```
## Not run:
test_queue(q_obj = q_obj, ... = ...)
## End(Not run)
```

`to_flow` *Create flow objects*

Description

Use a set of shell commands and flow definiton to create flow object.

Usage

```
to_flow(x, ...)

## S3 method for class 'vector'
to_flow(x, def, grp_col, jobname_col, cmd_col, ...)

## S3 method for class 'data.frame'
to_flow(x, def, grp_col, jobname_col, cmd_col, flowname,
  flow_run_path, platform, submit = FALSE, execute = FALSE, qobj, ...)

## S3 method for class 'list'
to_flow(x, def, flowname, flow_run_path, desc, qobj, ...)
```

Arguments

`x`	path (char. vector) to flow_mat, a data.frame or a list.
`...`	Supplied to specific functions like to_flow.data.frame
`def`	A flow definition table. Basically a table with resource requirements and mapping of the jobs in this flow
`grp_col`	column name used to split x (flow_mat). Default: ‘samplename‘
`jobname_col`	column name with job names. Default: ‘jobname‘
`cmd_col`	column name with commands. Default: ‘cmd‘
`flowname`	name of the flow
`flow_run_path`	Path to a folder. Main operating folder for this flow. Default it ‘get_opts("flow_run_path")‘.
`platform`	character vector, specifying the platform to use. local, lsf, torque, moab, sge, slurm, ... This over-rides the platform column in flowdef.
`submit`	Depreciated. Use submit_flow on flow object this function returns. TRUE/FALSE
`execute`	Depreciated. Use submit_flow on flow object this function returns. TRUE/FALSE, an paramter to submit_flow()
`qobj`	Depreciated, modify cluster templates instead. A object of class queue.
`desc`	Advanced Use. final flow name, please don’t change.

Details

The parameter x can be a path to a flow_mat, or a data.frame (as read by read_sheet). This is a minimum three column matrix with three columns: samplename, jobname and cmd

Value

Returns a flow object. If execute=TRUE, fobj is rich with information about where and how the flow was executed. It would include details like jobids, path to exact scripts run etc. To use kill_flow, to kill all the jobs one would need a rich flow object, with job ids present.

Behaviour:: What goes in, and what to expect in return?

- submit=FALSE & execute=FALSE: Create and return a flow object
- submit=TRUE & execute=FALSE: dry-run, Create a flow object then, create a structured execution folder with all the commands
- submit=TRUE, execute=TRUE: Do all of the above and then, submit to cluster

Examples

```
ex = file.path(system.file(package = "flowr"), "pipelines")
flowmat = as.flowmat(file.path(ex, "sleep_pipe.tsv"))
flowdef = as.flowdef(file.path(ex, "sleep_pipe.def"))
fobj = to_flow(x = flowmat, def = flowdef, flowname = "sleep_pipe", platform = "lsf")
```

---

to_flowdef *Create a skeleton flow definition using a flowmat.*

Description

Creation of a skeleton flow definition with several default values.

All params may be of length one, or same as the number of jobnames

to_flowdef.character: x is a flowmat file.

Usage

```
to_flowdef(x, ...)

## S3 method for class 'flowmat'
to_flowdef(x, sub_type, dep_type, prev_jobs,
  queue = "short", platform = "torque", memory_reserved = "2000",
  cpu_reserved = "1", walltime = "1:00", ...)

## S3 method for class 'flow'
to_flowdef(x, ...)

## S3 method for class 'character'
to_flowdef(x, ...)
```

Arguments

x	can a path to a flowmat, flomat or flow object.
...	not used
sub_type	submission type, one of: scatter, serial. Character, of length one or same as the number of jobnames
dep_type	dependency type, one of: gather, serial or burst. Character, of length one or same as the number of jobnames

prev_jobs	previous job name
queue	Cluster queue to be used
platform	platform of the cluster: lsf, sge, moab, torque, slurm etc.
memory_reserved	amount of memory required.
cpu_reserved	number of cpu's required
walltime	amount of walltime required

`to_flowdet` *to_flowdet*

Description

to_flowdet

get a flow_details file from the directory structure. This has less information than the one generated using a flow object. Lacks jobids etc...

Usage

```
to_flowdet(x, ...)

## S3 method for class 'rootdir'
to_flowdet(x, ...)

## S3 method for class 'character'
to_flowdet(x, ...)

## S3 method for class 'flow'
to_flowdet(x, ...)
```

Arguments

x	this is a wd
...	not used

Details

if x is char. assumed a path, check if flow object exists in it and read it. If there is no flow object, try using a simpler function

to_flowmat *Taking in a named list and returns a two columns data.frame*

Description

Taking in a named list and returns a two columns data.frame

Usage

```
to_flowmat(x, ...)

## S3 method for class 'list'
to_flowmat(x, samplename, ...)

## S3 method for class 'data.frame'
to_flowmat(x, ...)

## S3 method for class 'flow'
to_flowmat(x, ...)
```

Arguments

x	a named list OR vector. Where name corresponds to the jobname and value is a vector of commands to run
...	not used
samplename	character of length 1 or that of nrow(x)

update_flow_det *update_flow_det*

Description

update_flow_det

Usage

```
update_flow_det(wd, mat_cmd)
```

Arguments

wd	flow working directory
mat_cmd	a table with details about cmd files

Details

Get the flow_det files from wd, and update it with new statuses.

whisker_render *Wrapper around whisker.render with some sugar on it...*

Description

This is a wrapper around whisker.render

Usage

```
whisker_render(template, data)
```

Arguments

template	template used
data	a list with variables to be used to fill in the template.

write_flow_details *write files desribing this flow*

Description

write files desribing this flow

Usage

```
write_flow_details(x, fobj, summ, flow_det, plot = FALSE)
```

Arguments

x	path to write to
fobj	flow object
summ	a status summary.
flow_det	a flow details data.frame
plot	logical, plot or not

Index