Package 'flowr'

October 5, 2015

Type Package

Title Streamlining Design and Deployment of Complex Workflows

Description This framework allows you to design and implement complex pipelines, and deploy them on your institution's computing cluster. This has been built keeping in mind the needs of bioinformatics workflows. However, it is easily extendable to any field where a series of steps (shell commands) are to be executed in a (work)flow.

Version 0.9.8.2

Depends R (>= 3.0.2),
methods,
params (>= 0.2.8),
utils

Imports diagram,
whisker,
tools

Suggests reshape2,
knitr,
ggplot2,
openxlsx,
testthat,
funr

VignetteBuilder knitr

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

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

License MIT + file LICENSE

R topics documented:

---

check *Check consistency of flowdef and flowmat*

Description

Check consistency of flowdef and flowmat, using various rules.

Usage

```
check(x, ...)

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

## S3 method for class 'flowdef'
check(x, verbose = get_opts("verbose"), ...)
```

Arguments

x	a flowdef or flowmat object
...	Passed onto either `check.flowdef` OR `check.flowmat` functions
verbose	A numeric value indicating the amount of messages to produce. Values are integers varying from 0, 1, 2, 3, ... Please refer to the verbose page for more details. [`get_opts("verbose")`] [1]

Details

A typical output from flowdef with verbose level: 2

```
checking if required columns are present...
checking if resources columns are present...
checking if dependency column has valid names...
checking if submission column has valid names...
checking for missing rows in def...
checking for extra rows in def...
checking submission and dependency types...
jobname prev.sub_type --> dep_type --> sub_type: relationship
1: aln1_a none --> none --> scatter
2: aln2_a scatter --> none --> scatter
3: sampe_a scatter --> serial --> scatter rel: complex one:one
4: fixrg_a scatter --> serial --> scatter rel: complex one:one
5: merge_a scatter --> gather --> serial rel: many:one
6: markdup_a serial --> serial --> serial rel: simple one:one
7: target_a serial --> serial --> serial rel: simple one:one
8: realign_a serial --> burst --> scatter rel: one:many
9: baserecalib_a scatter --> serial --> scatter rel: complex one:one
10: printreads_a scatter --> serial --> scatter rel: complex one:one
```

`check_args` *Assert none of the arguemnts of a function are null.*

Description

Checks all the arguments in the parent function and makes sure that none of them are NULL

Usage

```
check_args(ignore, select)
```

Arguments

`ignore`	optionally ignore a few variables for checking.
`select`	optionally only check a few variables of the function.

Details

This function has now been moved to params package.

`fetch` *Two generic functions to search for pipelines and configuration files.*

Description

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

`fetch_pipes()`: Fetches pipelines in the following places, in this specific order:

- **user's folder**: `~/flowr/pipelines`
- **current wd**: `./`

NOTE: If same pipeline is availabe in multiple places; intitutively, one from the later folder would be selected. As such, giving priority to user's home, and current working directories.

`fetch_conf()`: Fetches configuration files in ALL of the following places:

- **package**: `conf` folders in flowr and ngsflows packages.
- **user's folder**: `~/flowr/conf` folder.
- **current wd**: `./`

NOTE: This function would greedily return all matching conf files. One would load all of them in the order returned by this functions. If the same variable is repeated in multiple files, value from the later files would replace those formerly defined. Thus (as explained above), giving priority to options defined in user's home and current working directories.

By default flowr loads, `flowr.conf` and `ngsflows.conf`. See the details sections, for more explanation on this.

Usage

```
fetch(x, places, urls, verbose = get_opts("verbose"))

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

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

Arguments

`x`	name of the file to search for (without extension). By default fetch_pipes and fetch_conf search for files ending with `.R` and `.conf` respectively.
`places`	places (paths) to look for files matching the name. Defaults are already defined in the function.
`urls`	urls to look for, works well for pipelines [not implemented yet]
`verbose`	A numeric value indicating the amount of messages to produce. Values are integers varying from 0, 1, 2, 3, ... Please refer to the verbose page for more details. [`get_opts("verbose")`] [1]
`last_only`	fetch_pipes():. If multiple pipelines match the pattern, return the last one. [TRUE]
`silent`	fetch_pipes(): logical, be silent even if no such pipeline is available. [FALSE]
`ask`	ask before downloading or copying. [not implemented]
`...`	[not implemented]

Details

For example flowr has a variable `flow_run_path` where it puts all the execution logs etc. The default value is picked up from packages's internal `flowr.conf` file. To redefine this value, one could create a new file called `~/flowr/conf/flowr.conf` and add a line:

`flow_run_path<TAB>my_awesome_path`, where <TAB> is a tab character, since these are tab seperated files.

Also, at any time you can run, `load_conf('super_specific_opts.conf');` to load custom options.

See Also

flowopts

Examples

```
## let us find a default conf file
conf = fetch_conf("flowr.conf");conf
## load this
load_opts(conf)

## this returns a list, which prints pretty
pip = fetch_pipes("sleep_pipe")
pip$name
pip$pipe
pip$def
```

---

flow-class *Describing the flow class*

Description

Internal function (used by to_flow), which aids in creating a flow object.

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 = "created", execute = "")
```

Arguments

jobs	list: A list of jobs to be included in this flow
name	character: Name of the flow. ['newflow']
desc	character Description of the flow, used to uniquely identify a flow instance. ['my_super_flow']
mode	character Mode of submission of the flow (depreciated). ['scheduler']
flow_run_path	The base path of all the flows you would submit. [~/flows]
trigger_path	character [~/flows/trigger].

flow_path	character: A unique path identifying a flow instance, populated by submit_flow.
version	version of flowr used to create and execute this flow.
status	character: Status of the flow.
execute	execuction status of flow object. [FALSE]

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)
```

---

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

Description

There are three helper functions which attempt to manage parameters 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 package.

Usage

```
flowopts

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
time_format	%a %b %e %H:%M:%S CDT %Y
verbose	FALSE

See Also

fetch params read_sheet

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()
conffile = fetch_conf("flowr.conf")
load_opts(conffile)

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

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 *Describing details of the job object*

Description

Internal function (used by to_flow), which aids in creating a job object.

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 all jobs submitted to the computing platform, for one or multiple flows*

Description

NOTE:

This requires files which are created at the end of the submit_flow command.

Even if you want to kill the flow, its best to let submit_flow do its job, when done simply use `kill(flow_wd)`. If submit_flow is interrupted, files like flow_details.rds etc are not created, thus flowr looses the association of jobs with flow instance and cannot monitor, kill or re-run the flow.

Usage

```
kill(x, ...)

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

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

Arguments

x	either path to flow wd or object of class flow
...	not used
force	You need to set force=TRUE, to kill multiple flows. This makes sure multiple flows are NOT killed by accident.
kill_cmd	The command used to kill. flowr tries to guess this commands, as defined in the detect_kill_cmd(). Supplying it here; fot custom platoforms.
verbose	A numeric value indicating the amount of messages to produce. Values are integers varying from 0, 1, 2, 3, ... Please refer to the verbose page for more details. [get_opts("verbose")] [1]
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
## In case path matches multiple folders, flowr asks before killing
kill(x='fastq_haplotyper*')
 Flowr: streamlining workflows
 found multiple wds:
 /fastq_haplotyper-MS132-20150825-16-24-04-0Lv1PbpI
 /fastq_haplotyper-MS132-20150825-17-47-52-5vFIkrMD
 Really kill all of them ? kill again with force=TRUE

## submitting again with force=TRUE will kill them:
kill(x='fastq_haplotyper*', force = TRUE)

## End(Not run)
```

---

plot_flow *Plot a clean and scalable flowchart describing the (work)flow*

Description

Plot a flowchart using a flow object or flowdef

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, ...)
```

Arguments

x	Object of class flow, or a list of flow objects or a flowdef
...	experimental and only for advanced use.
detailed	include submission and dependency types in the plot [TRUE]
type	1 is original, and 2 is a elipse with less details [1]
pdf	create a pdf instead of plotting interactively [FALSE]
pdffile	output file name for the pdf file. [flow_path/flow_details.pdf]

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-class *A* queue *object defines details regarding how a job is submitted*

Description

Internal function (used by to_flow), to define the format used to submit a job.

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')
```

`rerun` *Re-run a pipeline in case of hardware or software failures.*

Description

- **hardware** no change required, simple rerun: `rerun(x=flow_wd)`
- **software** either a change to flowmat or flowdef has been made: `rerun(x=flow_wd, mat = new_flowmat, def = r`

NOTE:

flow_wd: flow working directory, same input as used for status

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,
  select, ignore, verbose = get_opts("verbose"), ...)
```

Arguments

`x`	flow working directory
`...`	passed onto to_flow
`mat`	(optional) flowmat fetched from previous submission if missing. For more information regarding the format refer to to_flowmat
`def`	(optional) flowdef fetched from previous submission if missing. For more information regarding the format refer to to_flowdef
`start_from`	which job to start from, this is a job name.
`execute`	[logical] whether to execute or not
`kill`	(optional) logical indicating whether to kill the jobs from the previous execution of flow.
`select`	select a subset of jobs to rerun [character vector]
`ignore`	ignore a subset of jobs to rerun [character vector]
`verbose`	A numeric value indicating the amount of messages to produce. Values are integers varying from 0, 1, 2, 3, ... Please refer to the verbose page for more details. [`get_opts("verbose")`] [1]

Details

This function fetches details regarding the previous execution from the flow working directory (flow_wd).

It reads the flow object from the flow_details.rds file, and extracts flowdef and flowmat from it using to_flowmat and to_flowdef functions.

Using new flowmat OR flowdef for re-run:

Optionally, if either of flowmat or flowdef are supplied; supplied ones are used instead of those extracted from previous submission.

This functions efficiently updates job details of the latest submission into the previous file; thus information regarding previous job ids and their status is not lost.

Examples

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

## End(Not run)
```

`run` *Run automated Pipelines*

Description

Run complete pipelines, by wrapping several steps into one convinient function:

Taking `sleep_pipe` as a example.

- Use fetch_pipes to get paths to a Rscript, flowdef file and optionally a configuration file with various default options used.
- Create a flowmat (using the function defined in the Rscript)
- Create a 'flow' object, using flowmat created and flowdef (as fetched using fetch_pipes)
- Submit the flow to the cluster (using submit_flow)

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 as specified in x

Examples

```
## Not run:

## Run a short pipeline (dry run)
run("sleep_pipe")

## Run a short pipeline on the local machine
run("sleep_pipe", platform = "local", execute = TRUE)

## Run a short pipeline on the a torque cluster (qsub)
run("sleep_pipe", platform = "torque", execute = TRUE)

## Run a short pipeline on the a MOAB cluster (msub)
run("sleep_pipe", platform = "moab", execute = TRUE)

## Run a short pipeline on the a IBM (LSF) cluster (bsub)
run("sleep_pipe", platform = "lsf", execute = TRUE)

## Run a short pipeline on the a MOAB cluster (msub)
run("sleep_pipe", platform = "moab", execute = TRUE)

## change parameters of the pipeline
## All extra parameters are passed on to the function function.
run("sleep_pipe", platform = "lsf", execute = TRUE, x = 5)


## End(Not run)
```

`setup` *Setup and initialize flowr*

Description

This functions creates a directory structure in user's home directory. Additionally it creates a shortcut to the `flowr` helper script in `~/bin`.

Usage

```
setup(bin = "~/bin", flow_base_path = get_opts("flow_base_path"),
  flow_run_path = get_opts("flow_run_path"),
  flow_conf_path = get_opts("flow_conf_path"),
  flow_pipe_path = get_opts("flow_pipe_paths"))
```

Arguments

`bin`	path to bin folder
`flow_base_path`	The base of flowr configuration and execution folders.
`flow_run_path`	Path to a folder. Main operating folder for this flow. [`get_opts("flow_run_path")`] [`~/flowr/runs`].
`flow_conf_path`	Flowr configuration folder, used by fetch_conf.
`flow_pipe_path`	Folder with all pipelines, used by fetch_pipes.

Details

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

`status` *Monitor status of flow(s)*

Description

Summarize status of a flow OR multiple flows OR a high-level summary of all flows in a folder.

Usage

```
status(x, use_cache = FALSE, verbose = get_opts("verbose"),
  out_format = "markdown")

get_status(x, ...)

## S3 method for class 'flow'
get_status(x, verbose, use_cache, out_format, ...)

## S3 method for class 'character'
get_status(x, verbose, use_cache, out_format, ...)

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

Arguments

`x`	path to the flow root folder or a parent folder to summarize several flows.
`use_cache`	This skips checking status of jobs which have already been completed a and assumes no new jobs were submitted in the flow(s) being monitored. [FALSE]
`verbose`	A numeric value indicating the amount of messages to produce. Values are integers varying from 0, 1, 2, 3, ... Please refer to the verbose page for more details. [`get_opts("verbose")`] [1]
`out_format`	passed onto knitr:::kable. supports: markdown, rst, html... [markdown]
`...`	not used

Details

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

- Get status of all the flows: (all flows with 'sleep_pipe' in their name are checked and their status is shown)
 `flowr status x=~/flowr/runs/sleep_pipe*`
- Provide a high level summary of ALL flows in a folder:
 `flowr status x=~/flowr/runs`

Use **use_cache**=TRUE to speed up checking the status. This assumes that no new jobs have been submitted and skips (re-)checking status of completed jobs.

Once all the jobs have been submitted to the cluster you may always use `use_cache=TRUE`.

Examples

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

## End(Not run)
```

`submit_flow` *Submit a flow to the cluster*

Description

Submit a flow to the cluster or perform a dry-run to check and debug issues.

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.

Details

NOTE: Even if you want to kill the flow, its best to let submit_flow do its job, when done simply use `kill(flow_wd)`. If submit_flow is interrupted, files like flow_details.rds etc are not created, thus flowr looses the association of jobs with flow instance and cannot monitor, kill or re-run the flow.

Examples

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

submit_job *Submit a step of a flow*

Description

Internal function (used by submit_flow), which submit a single step of a flow.

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)
```

submit_run *Submit several flow objects, limit the max running concurrently*

Description

Submit several flow objects, limit the max running concurrently

Usage

```
submit_run(x, wd, max_processing = 7)
```

Arguments

x	a list of flow objects
wd	a folder to monitor (flow_run_path)
max_processing	max number of flow which may be processed concurrently

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'.

[Depreciated]: This function has been superseded by `run("sleep_pipe", platform = "lsf", execute=TRUE)`

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.

vector: a file with flowmat table

a named list of commands for a sample. Its best to supply a flowmat instead.

Usage

```
to_flow(x, ...)

is.flow(x)

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

## S3 method for class 'flowmat'
to_flow(x, def, grp_col, jobname_col, cmd_col, flowname,
  flow_run_path, platform, submit = FALSE, execute = FALSE, qobj,
  verbose = get_opts("verbose"), ...)
```

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

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

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). [samplename]
jobname_col	column name with job names. [jobname]
cmd_col	column name with commands. [cmd]
flowname	name of the flow [flowname]
flow_run_path	Path to a folder. Main operating folder for this flow. [get_opts("flow_run_path")] [~/flowr/runs].
platform	character vector, specifying the platform to use. local, lsf, torque, moab, sge, slurm, ... This over-rides the platform column in flowdef. (optional)
submit	Use submit_flow on flow object this function returns. TRUE/FALSE. [FALSE]
execute	Use submit_flow on flow object this function returns. TRUE/FALSE, an paramter to submit_flow(). [FALSE]
qobj	Depreciated, modify cluster templates instead. A object of class queue.
verbose	A numeric value indicating the amount of messages to produce. Values are integers varying from 0, 1, 2, 3, ... Please refer to the verbose page for more details. [get_opts("verbose")] [1]
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

See Also

to_flowmat, to_flowdef, to_flowdet, flowopts and submit_flow

Examples

```
## Use this link for a few elaborate examples:
## http://docs.flowr.space/flowr/tutorial.html#define_modules

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")


## create a vector of shell commands
cmds = c("sleep 1", "sleep 2")
## create a named list
lst = list("sleep" = cmds)
## create a flowmat
flowmat = to_flowmat(lst, samplename = "samp")

## Use flowmat to create a skeleton flowdef
flowdef = to_flowdef(flowmat)

## use both (flowmat and flowdef) to create a flow
fobj = to_flow(flowmat, flowdef)

## submit the flow to the cluster (execute=TRUE) or do a dry-run (execute=FALSE)
## Not run:
fobj2 = submit_flow(fobj, execute=FALSE)
fobj3 = submit_flow(fobj, execute=TRUE)

## Get the status or kill all the jobs
status(fobj3)
kill(fobj3)

## End(Not run)
```

---

to_flowdef *Flow Definition defines how to stich pieces of the (work)flow into a flow.*

Description

This function enables creation of a skeleton flow definition with several default values, using a flowmat. To customize the flowdef, one may supply parameters such as sub_type and dep_type upfront. As such, these params must be of the same length as number of unique jobs using in the flowmat.

Each row in this table refers to one step of the pipeline. It describes the resources used by the step and also its relationship with other steps, especially, the step immediately prior to it.

Submission types: *This refers to the sub_type column in flow definition.*

Consider an example with three steps A, B and C. A has 10 commands from A1 to A10, similarly B has 10 commands B1 through B10 and C has a single command, C1. Consider another step D (with D1-D3), which comes after C.

step (number of sub-processes) A (10) —-> B (10) —-> C (1) —-> D (3)

- `scatter`: submit all commands as parallel, independent jobs.
  Submit A1 through A10 as independent jobs
- `serial`: run these commands sequentially one after the other.
  - Wrap A1 through A10, into a single job.

Dependency types

This refers to the dep_type column in flow definition.

- `none`: independent job.
  - *Initial step A has no dependency*
- `serial`: *one to one* relationship with previous job.
  - *B1 can start as soon as A1 completes, and B2 starts just after A2 and so on.*
- `gather`: *many to one*, wait for **all** commands in the previous job to finish then start the current step.
  - *All jobs of B (1-10), need to complete before C1 starts*
- `burst`: *one to many* wait for the previous step which has one job and start processing all cmds in the current step.
  - D1 to D3 are started as soon as C1 finishes.

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", verbose = get_opts("verbose"),
  ...)

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

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

as.flowdef(x, ...)

is.flowdef(x)
```

Arguments

`x`	can a path to a flowmat, flowmat 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
verbose	A numeric value indicating the amount of messages to produce. Values are integers varying from 0, 1, 2, 3, ... Please refer to the verbose page for more details. [get_opts("verbose")] [1]

Format

This is a tab separated file, with a minimum of 4 columns:

required columns:

- jobname: Name of the step
- sub_type: Short for submission type, refers to, how should multiple commands of this step be submitted. Possible values are 'serial' or 'scatter'.
- prev_jobs: Short for previous job, this would be the jobname of the previous job. This can be NA/./none if this is a independent/initial step, and no previous step is required for this to start. Additionally, one may use comma(s) to define multiple previous jobs (A,B).
- dep_type: Short for dependency type, refers to the relationship of this job with the one defined in 'prev_jobs'. This can take values 'none', 'gather', 'serial' or 'burst'.

resource columns (recommended):

Additionally, one may customize resource requirements used by each step. The format used varies and depends to the computing platform. Thus its best to refer to your institutions guide to specify these.

- cpu_reserved integer, specifying number of cores to reserve [1]
- memory_reserved Usually in KB [2000]
- nodes number of server nodes to reserve, most tools can only use multiple cores on a **single** node [1]
- walltime maximum time allowed for a step, usually in a HH:MM or HH:MM:SS format. [1:00]
- queue the queue to use for job submission [short]

to_flowdet *Create a flow's submission detail file*

Description

Create a file describing details regarding jobs ids, submission scripts 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

The path provided should contain a flow_detaits.rds file (which is used to extract all the information).

Incase a parent folder with multiple flows is provided information regarding jobids is omitted.

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 *Create a flowmat using a list a commands.*

Description

Create a flowmat (data.frame) using a **named** list a commands.

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

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, ...)

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

is.flowmat(x)
```

Arguments

x	a named list, 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) ['samplename']
grp_col	column used for grouping, default samplename.
jobname_col	column specifying jobname, default jobname
cmd_col	column specifying commands to run, default cmd

Examples

```
## Use this link for a few examples:
## http://docs.flowr.space/flowr/tutorial.html#define_modules

## create a vector of shell commands
cmds = c("sleep 1", "sleep 2")
## create a named list
lst = list("sleep" = cmds)
## create a flowmat
flowmat = to_flowmat(lst, samplename = "samp")

## Use flowmat to create a skeleton flowdef
flowdef = to_flowdef(flowmat)

## use both (flowmat and flowdef) to create a flow
fobj = to_flow(flowmat, flowdef)

## submit the flow to the cluster (execute=TRUE) or do a dry-run (execute=FALSE)
## Not run:
fobj2 = submit_flow(fobj, execute=FALSE)
fobj3 = submit_flow(fobj, execute=TRUE)

## Get the status or kill all the jobs
status(fobj3)
kill(fobj3)

## End(Not run)
```

verbose *Verbose levels, defining verboseness of messages*

Description

There are several levels of verboseness one can choose from.

levels:

- level 0 is almost silent, producing only necessary messages
- level 1 is good for most purposes, where as,
- level 2 is good when developing a new pipeline.
- level 3 is good for debugging, especially when getting un-expected results.

One can set the level of verboseness using `opts_flow$set(verbose=2)`, which will be used across flowr and ngsflows packages. Additionally one may set this value in the configurations files: `~/flowr/conf/flowr.conf` OR `~/flowr/conf/ngsflows.conf`.

Usage

```
verbose
```

Format

```
NULL
```

Examples

```
fl = system.file("pipelines/abcd.def", package = "flowr")
def = as.flowdef(fl, verbose = 0)
# def seems to be a file, reading it...
def = as.flowdef(fl, verbose = 1)
# def seems to be a file, reading it...
# checking if required columns are present...
# checking if resources columns are present...
# checking if dependency column has valid names...
# checking if submission column has valid names...
# checking for missing rows in def...
# checking for extra rows in def...
# checking submission and dependency types...
def = as.flowdef(fl, verbose = 2)
# def seems to be a file, reading it...
# checking if required columns are present...
# checking if resources columns are present...
# checking if dependency column has valid names...
# checking if submission column has valid names...
# checking for missing rows in def...
# checking for extra rows in def...
# checking submission and dependency types...
# jobname prev.sub_type --> dep_type --> sub_type: relationship
#  1: A none --> none --> scatter
#  2: B scatter --> serial --> scatter rel: complex one:one
#  3: C scatter --> gather --> serial rel: many:one
#  4: D serial --> burst --> scatter rel: one:many
```

whisker_render *Wrapper around whisker.render with some additional checks*

Description

Internal function (used by submit_job), which creates a submission script using platform specific templates.

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