Package: PEcAn.remote (via r-universe)

September 18, 2024

Type Package

Title PEcAn Model Execution Utilities

Version 1.8.0.9000

Description This package contains utilities for communicating with and executing code on local and remote hosts. In particular, it has PEcAn-specific utilities for starting ecosystem model runs.

Imports dplyr, foreach, furrr, PEcAn.logger, httr, jsonlite, urltools

Suggests doSNOW, getPass, mockery, testthat, tools, withr

License BSD_3_clause + file LICENSE

Encoding UTF-8

LazyData true

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

Repository https://pecanproject.r-universe.dev

RemoteUrl https://github.com/PecanProject/pecan

RemoteRef HEAD

RemoteSha f22a7c4bbc532e4551f7bc9624cef649da317ac1

Contents

`check_model_run` *Check if model run was successful*

Description

Check if model run was successful

Usage

```
check_model_run(out, stop.on.error = TRUE)
```

Arguments

`out`	Output from model execution, as a character.
`stop.on.error`	Throw error if *any* of the runs fails. Default TRUE.

Value

`TRUE` if model run succeeded. If model run failed, throw an error if `stop.on.error`, or return FALSE.

`fqdn` *Returns the fully qualified hostname.*

Description

Returns the fully qualified hostname. This is potentially different from `Sys.info()['nodename']` which can return just the hostname part and not the domain as well. For example the machine pecan.ncsa.illinois.edu will return just that as fqdn but only pecan for hostname.

Usage

```
fqdn()
```

Value

fully qualified hostname

Author(s)

Rob Kooper

Examples

```
fqdn()
```

`is.localhost` *Check if local host*

Description

Check if host is local

Usage

```
is.localhost(host)
```

Arguments

`host`	the hostname to be checked

Details

Given the hostname is this the localhost. This returns true if either the value is localhost, or the value is the same as the fqdn.

Value

true if the host is the local host name

Author(s)

Rob Kooper

Examples

```
is.localhost(fqdn())
```

`kill.tunnel` *Kill tunnel to remote machine*

Description

Kill tunnel to remote machine

Usage

```
kill.tunnel(settings, exe = TRUE, data = TRUE)
```

Arguments

`settings`	PEcAn settings list
`exe`	Kill tunnel to executable?
`data`	Kill tunnel to data?

Author(s)

Rob Kooper

`merge_job_files` *Merge multiple job.sh files into one larger file.*

Description

Merge multiple job.sh files into one larger file.

Usage

```
merge_job_files(settings, jobs_per_file = 10, outdir = NULL)
```

Arguments

`settings`	PEcAn.settings object with host section.
`jobs_per_file`	the number of files you want to merge.
`outdir`	output directory of merged job files.

Value

vector of the newly created filenames

Author(s)

Dongchen Zhang

`open_tunnel` *Open an SSH tunnel, prompting for passwords as needed*

Description

Open an SSH tunnel, prompting for passwords as needed

Usage

```
open_tunnel(
  remote_host,
  user = NULL,
  password = NULL,
  tunnel_dir = "~/.pecan/tunnel/",
  wait.time = 15,
  tunnel_script = "~/pecan/web/sshtunnel.sh"
)
```

Arguments

`remote_host`	name of remote server to connect to (e.g. geo.bu.edu)
`user`	username on remote_host
`password`	password on remote_host
`tunnel_dir`	directory to store tunnel file in, typically from settings$host
`wait.time`	how long to give system to connect before deleting password (seconds)
`tunnel_script`	Path to sshtunnel.sh script file for opening tunnel

Value

numeric giving ssh PID if configured, otherwise logical with TRUE = success

`qsub_get_jobid` *Get Job ID from qsub output*

Description

Get Job ID from qsub output

Usage

```
qsub_get_jobid(out, qsub.jobid, stop.on.error)
```

Arguments

`out`	Output from model execution, as a character.
`qsub.jobid`	(character) Regular expression string for extracting job ID from qsub output. Usually from `settings$host$qsub.jobid`
`stop.on.error`	Throw error if *any* of the runs fails. Default TRUE.

Value

Job ID, as a string

`qsub_parallel` *qsub_parallel*

Description

qsub_parallel

Usage

```
qsub_parallel(
  settings,
  files = NULL,
  prefix = "sipnet.out",
  sleep = 10,
  hybrid = TRUE
)
```

Arguments

`settings`	pecan settings object
`files`	allow submit jobs based on job.sh file paths.
`prefix`	used for detecting if jobs are completed or not.
`sleep`	time (in second) that we wait each time for the jobs to be completed.
`hybrid`	A Boolean argument decide the way of detecting job completion. If it's TRUE then we will detect both the outputted files and job ids on the server. If it's FALSE then we will only detect the job ids on the server.

Author(s)

Dongchen Zhang

Examples

```
## Not run:
  qsub_parallel(settings)

## End(Not run)
```

`qsub_run_finished` *Check if qsub run finished*

Description

Check if qsub run finished

Usage

```
qsub_run_finished(run, host, qstat)
```

Arguments

`run`	run ID, as an integer
`host`	host structure to execute command on
`qstat`	(string) qstat command for checking job status

Value

`TRUE` if run is marked as DONE, otherwise FALSE.

`rabbitmq_create_queue` *Create a queue in RabbitMQ.*

Description

This will first check to see if the queue already exists in RabbitMQ, if not it will create the queue. If the queue exists, or is created it will return TRUE, it will return FALSE otherwise.

Usage

```
rabbitmq_create_queue(
  url,
  auth,
  vhost,
  queue,
  auto_delete = FALSE,
  durable = TRUE
)
```

Arguments

`url`	parsed RabbitMQ URL.
`auth`	the httr authentication object to use.
`vhost`	the vhost where to create the queue.
`queue`	the queue that should be checked/created.
`auto_delete`	should the queue be deleted afterwards (FALSE is default)
`durable`	should the messages exists after a server restart (TRUE is default)

Value

TRUE if the queue now exists, FALSE otherwise.

Author(s)

Rob Kooper

`rabbitmq_get_message` *Get message from RabbitMQ.*

Description

This will get a message from RabbitMQ, if the queue does not exist it will be created. The message will be converted to a json message that is returned.

Usage

```
rabbitmq_get_message(uri, queue, count = 1, prefix = "", port = 15672)
```

Arguments

`uri`	RabbitMQ URI or URL to rest endpoint
`queue`	the queue the message is received from.
`count`	the number of messages to retrieve from the queue.
`prefix`	prefix for the rabbitmq api endpoint, default is for no prefix.
`port`	port for the management interface, the default is 15672.

Value

NA if no message was retrieved, or a list of the messages payload.

Author(s)

Alexey Shiklomanov, Rob Kooper

`rabbitmq_parse_uri` *parse the RabbiMQ URI.*

Description

This will parse the uri into smaller pieces that can be used to talk to the rest endpoint for RabbitMQ.

Usage

```
rabbitmq_parse_uri(uri, prefix = "", port = 15672)
```

Arguments

`uri`	the amqp URI
`prefix`	the prefix that the RabbitMQ managmenet interface uses
`port`	the port for rabbitmq managment interface

Value

a list that contains the url to the mangement interface, username password and vhost.

`rabbitmq_post_message` *Post message to RabbitMQ.*

Description

This will submit a message to RabbitMQ, if the queue does not exist it will be created. The message will be converted to a json message that is submitted.

Usage

```
rabbitmq_post_message(uri, queue, message, prefix = "", port = 15672)
```

Arguments

`uri`	RabbitMQ URI or URL to rest endpoint
`queue`	the queue the message is submitted to
`message`	the message to submit, will beconverted to json.
`prefix`	prefix for the rabbitmq api endpoint, default is for no prefix.
`port`	port for the management interface, the default is 15672.

Value

the result of the post if message was send, or NA if it failed.

Author(s)

Alexey Shiklomanov, Rob Kooper

`rabbitmq_send_message` *Send a message to RabbitMQ rest API.*

Description

It will check the resulting status code and print a message in case something goes wrong.

Usage

```
rabbitmq_send_message(url, auth, body, action = "POST", silent = FALSE)
```

Arguments

`url`	the full endpoint rest url
`auth`	authentication for rabbitmq in httr:auth
`body`	the actual body to send, this is a rabbitmq message.
`action`	the rest action to perform
`silent`	boolean to indicate if logging should be performed.

Value

will return NA if message failed, otherwise it will either return the resulting message, or if not availble an empty string "".

`remote.copy.from` *Copy file from remote to local*

Description

Copy file/dir from remote server to local server

Usage

```
remote.copy.from(
  host,
  src,
  dst,
  options = NULL,
  delete = FALSE,
  stderr = FALSE
)
```

Arguments

`host`	list with server, user and optionally tunnel to use.
`src`	remote file/dir to copy
`dst`	local file/dir to copy to
`options`	to be passed to rsync command, if nothing is specified everything will be rsynced
`delete`	in case of local dir should all non-existent files be removed
`stderr`	should stderr be returned

Details

Copies the file/dir from the remote server to the local server. If the dst is a folder it will copy the file into that folder.

Value

output of command executed

Author(s)

Rob Kooper

Examples

```
## Not run:
  host <- list(name='geo.bu.edu', user='kooper', tunnel='/tmp/geo.tunnel')
  remote.copy.from(host, '/tmp/kooper', '/tmp/geo.tmp', delete=TRUE)

## End(Not run)
```

---

remote.copy.to *Copy file/dir to remote server from local server*

Description

Copies the file/dir to the remote server from the local server. If the dst is a folder it will copy the file into that folder.

Usage

```
remote.copy.to(host, src, dst, options = NULL, delete = FALSE, stderr = FALSE)
```

Arguments

`host`	host structure to execute command on
`src`	local file/dir to copy
`dst`	remote file/dir to copy to
`options`	additional arguments to be passed to rsync command
`delete`	in case of local dir should all non-existent files be removed
`stderr`	should stderr be returned as well.

Value

output of command executed

Author(s)

Rob Kooper

Examples

```
## Not run:
  host <- list(name='geo.bu.edu', user='kooper', tunnel='/tmp/geo.tunnel')
  remote.copy.to(host, '/tmp/kooper', '/tmp/kooper', delete=TRUE)

## End(Not run)
```

`remote.execute.cmd` *Execute command remotely*

Description

Execute command remotely

Usage

```
remote.execute.cmd(host, cmd, args = character(), stderr = FALSE)
```

Arguments

`host`	host structure to execute command on
`cmd`	the system command to be invoked, as a character string.
`args`	a character vector of arguments to command.
`stderr`	should stderr be returned as well.

Details

Executes the given command on the remote host using ssh. If the user is set the system will login as the given user. If the host given is the local machine it will execute the command locally without ssh.

Value

the captured output of the command (both stdout and stderr)

Author(s)

Rob Kooper

Examples

```
## Not run:
  host <- list(name='geo.bu.edu', user='kooper', tunnel='/tmp/geo.tunnel')
  print(remote.execute.cmd(host, 'ls', c('-l', '/'), stderr=TRUE))

## End(Not run)
```

`remote.execute.R` *Execute command remotely*

Description

Execute command remotely

Usage

```
remote.execute.R(
  script,
  host = "localhost",
  user = NA,
  verbose = FALSE,
  R = "R",
  scratchdir = tempdir()
)
```

Arguments

`script`	the script to be invoked, as a list of commands.
`host`	settings host list
`user`	the username to use for remote login
`verbose`	should the output be printed to the console
`R`	Path to the R executable or binary file.
`scratchdir`	Path to the scratch directory for temporary files during remote execution.

Details

Executes the given command on the remote host using ssh. If the user is set the system will login as the given user. If the host given is the local machine it will execute the command locally without ssh.

Value

the captured output of the command (both stdout and stderr)

Author(s)

Rob Kooper

Examples

```
## Not run:
  remote.execute.R('list.files()', host='localhost', verbose=FALSE)

## End(Not run)
```

`setup_modellauncher` *Setup model launcher script and job list*

Description

Setup model launcher script and job list

Usage

```
setup_modellauncher(run, rundir, host_rundir, mpirun, binary)
```

Arguments

`run`	(numeric) run ID, as an integer
`rundir`	Local run directory. Usually from `settings$rundir`
`host_rundir`	Remote host run directory. Usually from `settings$host$rundir`
`mpirun`	MPI info, usually from `settings$host$modellauncher$mpirun`
`binary`	Binary info, usually from `settings$host$modellauncher$binary`

`start.model.runs` *Start selected ecosystem model runs within PEcAn workflow*

Description

DEFUNCT: This function has been moved to PEcAn.workflow::start_model_runs; please use that instead.

Usage

```
## S3 method for class 'model.runs'
start(settings, write = TRUE, stop.on.error = TRUE)

runModule.start.model.runs(settings, stop.on.error = TRUE)
```

Arguments

`settings`	pecan settings object
`write`	(logical) Whether or not to write to the database. Default TRUE.
`stop.on.error`	Throw error if *any* of the runs fails. Default TRUE.

Author(s)

Shawn Serbin, Rob Kooper, David LeBauer, Alexey Shiklomanov

Examples

```
## Not run:
  start.model.runs(settings)

## End(Not run)
```

`start_qsub` *Start qsub runs*

Description

Start qsub runs

Usage

```
start_qsub(
  run,
  qsub_string,
  rundir,
  host,
  host_rundir,
  host_outdir,
  stdout_log,
  stderr_log,
  job_script,
  qsub_extra = NULL
)
```

Arguments

`run`	(numeric) run ID, as an integer
`qsub_string`	qsub command string, with arguments. Usually from `settings$host$qsub`
`rundir`	Local run directory. Usually from `settings$rundir`
`host`	Remote host, as a list or character. Usually from `settings$host`.
`host_rundir`	Remote host run directory. Usually from `settings$host$rundir`
`host_outdir`	Remote host output directory. Usually from `settings$host$outdir`
`stdout_log`	Logfile for redirecting `stdout`.
`stderr_log`	Logfile for redirecting `stderr`
`job_script`	Base name (no path) of script to run. Usually either `job.sh` or `launcher.sh`.
`qsub_extra`	Extra `qsub` arguments. Usually from `settings$host$modellauncher$qsub.extra`

Value

Output of qsub command, as a character. This output can be parsed for ascertaining submission success.

`start_rabbitmq` *Start model execution using rabbitmq*

Description

Start model execution using rabbitmq

Usage

```
start_rabbitmq(folder, rabbitmq_uri, rabbitmq_queue)
```

Arguments

`folder`	Directory containing jobs to be started
`rabbitmq_uri`	RabbitMQ uri where messages should be posted
`rabbitmq_queue`	Queue to which messages are submitted

Value

Output of execution command, as a character (see `rabbitmq_post_message()`).

`start_serial` *Start model execution in serial mode*

Description

Start model execution in serial mode

Usage

```
start_serial(run, host, rundir, host_rundir, job_script)
```

Arguments

`run`	(numeric) run ID, as an integer
`host`	Remote host, as a list or character. Usually from `settings$host`.
`rundir`	Local run directory. Usually from `settings$rundir`
`host_rundir`	Remote host run directory. Usually from `settings$host$rundir`
`job_script`	Base name (no path) of script to run. Usually either `job.sh` or `launcher.sh`.

Value

Output of execution command, as a character (see `remote.execute.cmd()`).

test_remote *Test remote execution*

Description

Test remote execution

Usage

```
test_remote(host, stderr = TRUE, ...)
```

Arguments

host	host structure to execute command on
stderr	should stderr be returned as well.
...	additional arguments.

Value

TRUE is remote execution is successful. If unsuccessful, depends on the value of stderr. If stderr = TRUE (default), this function will throw an error. If stderr = FALSE, this function will print a logger error and return FALSE.

Examples

```
# Localhost execution should always work
good_host <- list(name = "localhost")
test_remote(good_host)

bad_host <- list(name = "bigbadwolf")
if (!test_remote(bad_host, stderr = FALSE)) {
  print("Big Bad Wolf is a bad host.")
}
```

Index