

Package ‘ssh.utils’

July 21, 2014

Title Local and remote system commands with output and error capture.

OS_type unix

Version 1.0

Author Sergei Izrailev

Maintainer Sergei Izrailev <sizrailev@collective.com>

Description This package provides utility functions for system command execution, both locally and remotely using ssh/scp. The command output is captured and provided to the caller. This functionality is intended to streamline calling shell commands from R, retrieving and using their output, while instrumenting the calls with appropriate error handling.

URL <http://github.com/collectivemedia/ssh.utils>

Depends R (>= 3.0.3),stringr,fork

License Apache License (== 2.0)

Copyright Copyright (C) Collective, Inc. | file inst/COPYRIGHTS

LazyData true

R topics documented:

cp.remote	2
file.exists.remote	2
mem.usage	3
mkdir.remote	3
ps.grep.remote	4
run.remote	4
ssh.utils	6
Index	7

cp.remote	<i>scp wrapper</i>
-----------	--------------------

Description

A wrapper around the scp shell command that handles local/remote files and allows copying between remote hosts via the local machine.

Usage

```
cp.remote(remote.src, path.src, remote.dest, path.dest, verbose = FALSE,
  via.local = FALSE, local.temp.dir = tempdir())
```

Arguments

remote.src	Remote machine for the source file in the format <code>user@machine</code> or an empty string for local.
path.src	Path of the source file.
remote.dest	Remote machine for the destination file in the format <code>user@machine</code> or an empty string for local.
path.dest	Path for the source file; can be a directory.
verbose	Prints elapsed time if TRUE
via.local	Copies the file via the local machine. Useful when two remote machines can't talk to each other directly.
local.temp.dir	When copying via local machine, the directory to use as scratch space.

`file.exists.remote` *Checks if a local or remote file exists.*

Description

A wrapper around a bash script. Works with local files too if `remote=""`.

Usage

```
file.exists.remote(file, remote = "")
```

Arguments

file	File path.
remote	Remote machine specification for ssh, in format such as <code>user@server</code> that does not require interactive password entry. For local execution, pass an empty string "" (default).

Value

TRUE or FALSE indicating whether the file exists.

mem.usage	<i>Measure the resident memory usage of a process.</i>
-----------	--

Description

Returns the memory usage in KB of a process with the specified process id. By default, returns the memory usage of the current R process. This can be used to measure and log the memory usage of the R process during script execution.

Usage

```
mem.usage(pid = getpid())
```

Arguments

pid Process ID (default is the current process id).

Value

The resident memory usage in KB.

mkdir.remote	<i>Creates a remote directory with the specified group ownership and permissions.</i>
--------------	---

Description

If the directory already exists, attempts to set the group ownership to the `user.group`. The allowed group permissions are one of `c("g+rx", "g+rx", "go-w", "go-rwx")`, or `"-"`. The value `"-"` means "don't change permissions".

Usage

```
mkdir.remote(path, user.group = NULL, remote = "",
  permissions = c("g+rx", "g+rx", "go-w", "go-rwx", "-"))
```

Arguments

path Directory path. If using `remote`, this should be a full path or a path relative to the user's home directory.

user.group The user group. If `NULL`, the default group is used.

remote Remote machine specification for `ssh`, in format such as `user@server` that does not require interactive password entry. For local execution, pass an empty string `""` (default).

permissions The group permissions on the directory. Default is `'rwx'`.

ps.grep.remote	<i>Checks for processes running on a local or remote machine.</i>
----------------	---

Description

One of the use cases for this function is to ensure that an R process is already running and not start another one accidentally.

Usage

```
ps.grep.remote(grep.string, remote, stop.if.any = FALSE,
               stop.if.none = FALSE, count.self = FALSE, ps.options = "aux")
```

Arguments

grep.string	String(s) to check for in ps. If a vector, runs a chain of piped grep commands for each string.
remote	Remote machine specification for ssh, in format such as user@server that does not require interactive password entry. For local execution, pass an empty string "" (default).
stop.if.any	Stop if any of grep.string is running
stop.if.none	Stop if none of grep.string is running
count.self	When FALSE, excludes the calling process name from the count, if it gets matched.
ps.options	Gives the ability to run different options to ps.

See Also

run.remote

run.remote	<i>Functions to run commands remotely via ssh and capture output.</i>
------------	---

Description

run.withwarn - Evaluates the expression (e.g. a function call) and returns the result with additional attributes:

- num.warnings - number of warnings occurred during the evaluation
- last.message - the last warning message

Otherwise, run.withwarn is similar to base::suppressWarnings

run.remote - Runs the command locally or remotely using ssh.

Usage

```
run.withwarn(expr)
```

```
run.remote(cmd, remote = "", intern = T, stderr.redirect = T,
           verbose = F)
```

Arguments

<code>expr</code>	Expression to be evaluated.
<code>cmd</code>	Command to run. If run locally, quotes should be escaped once. If run remotely, quotes should be escaped twice.
<code>remote</code>	Remote machine specification for ssh, in format such as <code>user@server</code> that does not require interactive password entry. For local execution, pass an empty string <code>""</code> (default).
<code>intern</code>	Useful for debugging purposes: if there's an error in the command, the output of the remote command is lost. Re-running with <code>intern=FALSE</code> causes the output to be printed to the console. Normally, we want to capture output and return it.
<code>stderr.redirect</code>	When TRUE appends <code>2>&1</code> to the command. Generally, one should use that to capture STDERR output with <code>intern=TRUE</code> , but this should be set to <code>FALSE</code> if the command manages redirection on its own.
<code>verbose</code>	When TRUE prints the command.

Details

In `run.remote` the remote commands are enclosed in wrappers that allow to capture output. By default `stderr` is redirected to `stdout`. If there's a genuine error, e.g., the remote command does not exist, the output is not captured. In this case, one can see the output by setting `intern` to `FALSE`. However, when the command is run but exits with non-zero code, `run.remote` intercepts the generated warning and saves the output.

The remote command will be put inside double quotes twice, so all quotes in `cmd` must be escaped twice: `\\`". However, if the command is not remote, i.e., `remote` is `NULL` or empty string, quotes should be escaped only once.

If the command itself redirects output, the `stderr.redirect` flag should be set to `FALSE`.

Value

`run.remote` returns a list containing the results of the command execution, error codes and messages.

- `cmd.error` - flag indicating if a warning was issued because command exited with non-zero code
- `cmd.out` - the result of the command execution. If there was no error, this contains the output as a character array, one value per line, see [system](#). If there was an error (as indicated by `cmd.error`), this most likely contains the error message from the command itself. The `elapsed.time` attribute contains the elapsed time for the command in seconds.
- `warn.msg` - the warning message when `cmd.error` is `TRUE`.

Warnings are really errors here so the error flag is set if there are warnings.

Additionally, `cmd.out` has the `elapsed.time`, `num.warnings` and, if the number of warnings is greater than zero, `last.warning` attributes.

Examples

```
## Not run:
# Example of error handling:
res <- run.remote(cmd=command, remote=remote)
if (res$cmd.error)
{
  stop(paste(paste(res$cmd.out, collapse="\n"), res$warn.msg, sep="\n"))
}

## End(Not run)
```

ssh.utils

*Local and remote system commands with output and error capture.***Description**

Package `ssh.utils` provides utility functions for system command execution, both locally and remotely using `ssh/scp`. The command output is captured and provided to the caller. This functionality is intended to streamline calling shell commands from R, retrieving and using their output, while instrumenting the calls with appropriate error handling.

OS_type

unix

Maintainer

Sergei Izrailev

Copyright

Copyright (C) Collective, Inc.

LicenseApache License, Version 2.0, available at <http://www.apache.org/licenses/LICENSE-2.0>**URL**<http://github.com/collectivemedia/ssh.utils>**Installation from github**

```
devtools::install_github("collectivemedia/ssh.utils")
```

Author(s)

Sergei Izrailev

See Also

[run.remote](#), [cp.remote](#), [file.exists.remote](#), [mkdir.remote](#), [ps.grep.remote](#), [mem.usage](#)

Index

- *Topic **bash**
 - ssh.utils, 6
- *Topic **capture**
 - ssh.utils, 6
- *Topic **remote**
 - ssh.utils, 6
- *Topic **scp**
 - ssh.utils, 6
- *Topic **shell**
 - ssh.utils, 6
- *Topic **ssh**
 - ssh.utils, 6
- *Topic **system**
 - ssh.utils, 6

cp.remote, 2, 6

file.exists.remote, 2, 6

mem.usage, 3, 6

mkdir.remote, 3, 6

ps.grep.remote, 4, 6

run.remote, 4, 6

run.withwarn(*run.remote*), 4

ssh.utils, 6

ssh.utils-package(*ssh.utils*), 6

system, 5