Package 'remoter'

August 29, 2016

Type Package

Title Remote R: Control a Remote R Session from a Local One

Version 0.3-2

Description A set of utilities for controlling a remote R session from a local one. Simply set up a server (see package vignette for more details) and connect to it from your local R session, including 'RStudio'. Network communication is handled by the 'ZeroMQ' library by way of the 'pbdZMQ' package. The client/server framework is a custom 'REPL'.

License BSD 2-clause License + file LICENSE

Depends R (>= 3.2.0), pbdZMQ (>= 0.2-2)

Imports stats, utils, tools, assertthat (>= 0.1), getPass (>= 0.1-0)

Suggests knitr, rmarkdown, sodium (>= 0.2)

VignetteBuilder knitr

NeedsCompilation no

ByteCompile yes

URL https://github.com/RBigData/remoter

BugReports https://github.com/RBigData/remoter/issues

Maintainer Drew Schmidt <wrathematics@gmail.com>

Author Drew Schmidt [aut, cre],
Wei-Chen Chen [aut]

RoxygenNote 5.0.1

Repository CRAN

Date/Publication 2016-04-29 07:31:34

R topics documented:

`remoter-package` *remoter*

Description

A package for running R commands on a remote R session. With remoter, you can use RStudio running on your laptop to execute commands on an R session running on, for example, Amazon's EC2.

Author(s)

Drew Schmidt and Wei-Chen Chen

References

Project URL: https://github.com/RBigData/remoter

`c2s` *Client-to-Server Object Transfer*

Description

This function allows you to pass an object from the local R session (the client) to server.

Usage

```
c2s(object, newname, env = .GlobalEnv)
```

Arguments

object	A local R object.
newname	The name the object should take when it is stored on the remote server. If left blank, the remote name will be the same as the original (local) object's name.
env	The environment into which the assignment will take place. The default is the remoter "working environment".

Details

Localize R objects.

Value

Returns TRUE invisibly on successful exit.

Examples

```
## Not run:
### Prompts are listed to clarify when something is eval'd locally vs remotely
> library(remoter)
> x <- "some data"
> remoter::connect("my.remote.server")
remoteR> x
### Error: object 'x' not found
remoteR> c2s(x)
remoteR> x
###  [1] "some data"

## End(Not run)
```

---

client *Client Launcher*

Description

Connect to a remote server (launch the client).

Usage

```
client(addr = "localhost", port = 55555, prompt = "remoter",
  timer = FALSE)
```

Arguments

addr	The remote host/address/endpoint.
port	The port (number) that will be used for communication between the client and server. The port value for the client and server must agree.
prompt	The prompt to use to delineate the client from the normal R REPL.
timer	Logical; should the "performance prompt", which shows timing statistics after every command, be used?

Details

The port values between the client and server must agree. If they do not, this can cause the client to hang. The client is a specialized REPL that intercepts commands sent through the R interpreter. These commands are then sent from the client to and evaluated on the server. The client communicates over ZeroMQ with the server using a REQ/REP pattern. Both commands (from client to server) and returns (from server to client) are handled in this way.

To shut down the server and the client, see exit().

Value

Returns TRUE invisibly on successful exit.

---

evalc *evalc*

Description

A function to evaluate expressions on the client's R session. To eval expressions on the server, just use eval(). Instead of using this function, you could also just kill the client, do your local operations, then re-run your client() command.

Usage

```
evalc(expr)
```

Arguments

expr	Expression to be evaluated on the client.

Details

Evaluate expressions on the client.

Value

Returns TRUE invisibly on successful exit.

`exit` *exit*

Description

This function cleanly shuts down the remoter server the client is currently connected to, as well as shutting down the client. One can also use `q()` (while the client is running), and this will not close the active R session on the client.

Usage

```
exit(client.only = TRUE, q.server = TRUE)
```

Arguments

`client.only`	Logical; if `TRUE`, then the client disconnects from the server. Otherwise, the server is shut down together with the client.
`q.server`	Logical; if `TRUE`, then the server calls `q("no")` after shuting down with the client. This is useful for cases where the server is running in an interactive R session, and you wish to shut the entire thing down.

Details

Exit the remoter client/server.

Value

Returns `TRUE` invisibly on successful exit.

`has.sodium` *has.sodium*

Description

Report if the sodium package is availabe for use.

Usage

```
has.sodium()
```

Value

Returns `TRUE` if the sodium package is available, and `FALSE` otherwise.

internals *Remoter Internals*

Description

Internal remoter utilities needed to create a custom server backend for use with the remoter client. See pbdCS for an example.

is.secure *is.secure*

Description

Report if communications with the connected server are encrypted.

Usage

```
is.secure()
```

Value

Returns TRUE if messages between client and server are currently encrypted, and FALSE if not. If the client is not currently running (i.e., if executed from just a regular R prompt), then NA is returned.

lsc *ls on Client*

Description

A function to view environments on the client's R session. To view objects on the server, just use `ls()`. Instead of using this function, you could also just kill the client, do your local operations, then re-run your `client()` command.

Usage

```
lsc(envir, all.names = FALSE, pattern)
```

Arguments

envir	Environment (as in `ls()`).
all.names	Logical that determines if all names are returned or those beginning with a '.' are omitted (as in `ls()`).
pattern	Optional regular expression (as in `ls()`).

Details

View objects on the client.

Value

Returns `TRUE` invisibly on successful exit.

`relay` *Relay Launcher*

Description

Launcher for the remoter relay.

Usage

```
relay(addr, recvport = 55556, sendport = 55555, verbose = FALSE)
```

Arguments

`addr`	The address of the server.
`recvport`	The port for receiving commands from the client.
`sendport`	The port for sending commands to the server.
`verbose`	Show verbose messaging.

Details

The relay is an intermediary or "middleman" between the client and server meant for machines with split login/compute nodes.

Value

Returns `TRUE` invisibly on successful exit.

rmc *rmc*

Description

A function to remove objects from the client's R session. To remove objects on the server, just use `rm()`. Instead of using this function, you could also just kill the client, do your local operations, then re-run your `client()` command.

Usage

```
rmc(..., list = character(), envir)
```

Arguments

`...`	Objects to be removed from the client's R session.
`list`	Character vector naming objects to be removed (as in `rm()`).
`envir`	Environment (as in `rm()`).

Details

Remove objects on the client.

Value

Returns `TRUE` invisibly on successful exit.

s2c *Server-to-Client Object Transfer*

Description

This function allows you to pass an object from the server to the local R session behind the client.

Usage

```
s2c(object, newname, env = .GlobalEnv)
```

Arguments

`object`	A remote R object.
`newname`	The name the object should take when it is stored on the local client's R session. If left blank, the local name will be the same as the original (remote) object's name.
`env`	The environment into which the assignment will take place. The default is the global environment.

Details

Localize R objects.

Value

Returns TRUE invisibly on successful exit.

Examples

```
## Not run:
### Prompts are listed to clarify when something is eval'd locally vs remotely
> library(remoter)
> y
###  Error: object 'y' not found
> remoter::connect("my.remote.server")
remoteR> x
### Error: object 'x' not found
remoteR> x <- "some data"
remoteR> x
###  [1] "some data"
remoteR> s2c(x, "y")
remoteR> q()
> y
###  [1] "some data"

## End(Not run)
```

---

server *Server Launcher*

Description

Launcher for the remoter server.

Usage

```
server(port = 55555, password = NULL, maxretry = 5,
  secure = has.sodium(), log = TRUE, verbose = FALSE, showmsg = FALSE)
```

Arguments

port	The port (number) that will be used for communication between the client and server. The port value for the client and server must agree.
password	A password the client must enter before the user can process commands on the server. If the value is NULL, then no password checking takes place.
maxretry	The maximum number of retries for passwords before shutting everything down.

secure	Logical; enables encryption via public key cryptography of the 'sodium' package is available.
log	Logical; enables some basic logging in the server.
verbose	Logical; enables the verbose logger.
showmsg	Logical; if TRUE, messages from the client are logged

Details

By a 'secure' server, we mean one that encrypts messages it sends and only accepts encrypted messages. Encryption uses public key cryptography, using the 'sodium' package.

If the 'sodium' package is available to the server, then by default the server will be secure. If the package is not available, then you will not be able to start a secure server. If the server is secure, then a client can only connect if the client has the 'sodium' package available.

Value

Returns TRUE invisibly on successful exit.

---

showlog *showlog*

Description

Show the server log on the client.

Usage

```
showlog()
```

Index