Package ‘remoter’

October 14, 2022

Type Package

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

Version 0.4-0

Description A set of utilities for client/server computing with R, controlling a remote R session (the server) from a local one (the client). Simply set up a server (see package vignette for more details) and connect to it from your local R session ('RStudio', terminal, etc). The client/server framework is a custom 'REPL' and runs entirely in your R session without the need for installing a custom environment on your system. Network communication is handled by the 'ZeroMQ' library by way of the 'pbdZMQ' package.

License BSD 2-clause License + file LICENSE

Depends R (>= 3.2.0)

Imports pbdZMQ (>= 0.3-0), getPass (>= 0.1-0), argon2 (>= 0.2-0), stats, utils, tools, grDevices, graphics, png (>= 0.1-7)

Suggests sodium (>= 0.2), pbdRPC (>= 0.1-0)

NeedsCompilation no

ByteCompile yes

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

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

Maintainer Drew Schmidt <`wrathematics@gmail.com`>

RoxygenNote 6.0.1

Author Drew Schmidt [aut, cre],
Wei-Chen Chen [aut],
R Core team [ctb] (some functions are modified from the R source code)

Repository CRAN

Date/Publication 2018-01-05 05:04:32 UTC

R topics documented:

`remoter-package` *remoter*

Description

A set of utilities for client/server computing with R, controlling a remote R session (the server) from a local one (the client). Simply set up a server (see package vignette for more details) and connect to it from your local R session ('RStudio', terminal, etc). The client/server framework is a custom 'REPL' and runs entirely in your R session without the need for installing a custom environment on your system. Network communication is handled by the 'ZeroMQ' library by way of the 'pbdZMQ' package.

Author(s)

Drew Schmidt and Wei-Chen Chen

References

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

batch — *Batch Execution*

Description

Run a local script on a remote server in batch. Similar to R's own `source()` function.

Usage

```
batch(addr = "localhost", port = 55555, password = NULL, file, script,
  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.
`password`	An initial password to pass to the server. If the server is not accepting passwords, then this argument is ignored. If the initial pasword is incorrect, then assuming the server's `maxretry>1`, then you will be interactively asked to enter the password.
`file`	A character string pointing to the file you wish to execute/source. Either this or `script` (but not both) should be procided.
`script`	A character string containing the commands you wish to execute/source. Either this or `script` (but not both) should be procided.
`timer`	Logical; should the "performance prompt", which shows timing statistics after every command, be used?

Details

Note that `batch()` can not be run from inside an active connection. Its purpose is to bypass the need to start a connection via `client()`

Value

Returns `TRUE` invisibly on successful exit.

Examples

```
## Not run:
library(remoter)
### NOTE first run a server via remoter::server() )in a separate R session.
### For simplicity, assume they are on the same machine.

# Run a script in an R file on the local/client machine
file <- "/path/to/an/R/script.r"
batch(file=file)
```

```
# Run a script stored in a character vector
script <- "1+1"
batch(script="1+1")

## End(Not run)
```

---

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, password = NULL,
  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.
`password`	An initial password to pass to the server. If the server is not accepting passwords, then this argument is ignored. If the initial pasword is incorrect, then assuming the server's `maxretry>1`, then you will be interactively asked to enter the password.
`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)

shutdown()

kill(addr = "localhost", port = 55555)
```

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.
`addr, port`	The server address and port, as in `server()`.

Details

Exit the remoter client/server.

The `shutdown()` function is shorthand for `exit(FALSE, TRUE)`. The `kill()` function is shorthand for running `batch()` with `script="shutdown()"`.

Value

Returns `TRUE` invisibly on successful exit.

See Also

`server` and `batch`

`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.

`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.

rDevices *Local Graphic Devices*

Description

Functions for controlling graphic device locally when the client of remote R is on. All these functions are evaluated in local R from within the remote R prompt.

`dev.curc()` locally evals `grDevices::dev.cur()`.

`dev.listc()` locally evals `grDevices::dev.list()`.

`dev.nextc()` locally evals `grDevices::dev.next()`.

`dev.prevc()` locally evals `grDevices::dev.prev()`.

`dev.offc()` locally evals `grDevices::dev.off()`.

`dev.setc()` locally evals `grDevices::dev.set()`.

`dev.newc()` locally eval `grDevices::dev.new()`.

`dev.sizec()` locally evals `grDevices::dev.size()`.

Usage

```
dev.curc()

dev.listc()

dev.nextc(which = grDevices::dev.cur())

dev.prevc(which = grDevices::dev.cur())

dev.offc(which = grDevices::dev.cur())

dev.setc(which = grDevices::dev.cur())

dev.newc(..., noRstudioGD = FALSE)

dev.sizec(units = c("in", "cm", "px"))
```

Arguments

`which`	An integer specifying a device number as in `grDevices::dev.off()`
`...`	arguments to be passed to the device selected as in `grDevices::dev.new()`
`noRstudioGD`	as in `grDevices::dev.new()`
`units`	as in `grDevices::dev.size()`

Details

Local Graphic Device Controlling Functions

See Also

`rpng()`

Examples

```
## Not run:
### Prompts are listed to clarify when something is eval'd locally vs
### remotely
> library(remoter, quietly = TRUE)
> client()

remoter> rpng.new(plot(1:5))
remoter> dev.newc(width = 6, height = 4)
remoter> a <- function() plot(iris$Sepal.Length, iris$Petal.Length)
remoter> rpng.new(a, width = 6 * 72, height = 4 * 72)

remoter> dev.curc()
remoter> dev.listc()
remoter> dev.offc()

remoter> q()
>

## End(Not run)
```

`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.

`rhelp` *rhelp*

Description

Provide the primary interface to the help systems as `utils::help()`

Usage

```
rhelp(topic, package = NULL, lib.loc = NULL,
  verbose = getOption("verbose"),
  try.all.packages = getOption("help.try.all.packages"))

help(topic, package = NULL, lib.loc = NULL,
  verbose = getOption("verbose"),
  try.all.packages = getOption("help.try.all.packages"))

"?"(e1, e2)
```

Arguments

`topic, e1, e2`	A topic as in `utils::help()`
`package`	A package as in `utils::help()`
`lib.loc`	A lib location as in `utils::help()`
`verbose`	if verbose on/off as in `utils::help()`
`try.all.packages`	if try all packages as in `utils::help()`

Details

Remote R Help System

Examples

```
## Not run:
### Prompts are listed to clarify when something is eval'd locally vs
### remotely
> # suppressMessages(library(remoter, quietly = TRUE))
> # client()
> remoter::client("192.168.56.101")

remoter> rhelp("plot")
remoter> rhelp(package = "remoter")
remoter> rhelp("plot", package = "remoter")
remoter> rhelp("dev.off")
remoter> rhelp("dev.off", package = "remoter")
remoter> rhelp("dev.off", package = "grDevices")

remoter> help("par")

remoter> ?`+`
remoter> ?`?`
remoter> ?"??"
remoter> package?base
remoter> `?`(package, remoter)


remoter> q()
>

## End(Not run)
```

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.

rpng *rpng*

Description

Provide a graphic device locally for plots generated on server of Remote R

`rpng()` generates locally a device/window.

`rpng.new()` generates locally a device/window.

`rpng.off()` turns off locally a device/window.

`dev.off()` is an alias of `rpng.off()` in order to consisten with th original device function `grDevices::dev.off()`.

Usage

```
rpng.new(expr, filename = NULL, width = 587, height = 586, units = "px",
  pointsize = 12, bg = "white", res = 96, ...)

rpng.off(which = grDevices::dev.cur())

dev.off(which = grDevices::dev.cur())
```

Arguments

`expr`	An expression or a function generating a plot. This checks in the following orders: expression or ggplot. The ggplot are eval'd within the `rpng.new()`, while the expression is eval'd at `parent.frame()`.
`filename`	A temporary file to save the plot on server
`width`	width of the plot as in `grDevices::png()`
`height`	height of the plot as in `grDevices::png()`
`units`	units of the width and height as in `grDevices::png()`
`pointsize`	pointsze of the plotted text as in `grDevices::png()`
`bg`	background colour as in `grDevices::png()`
`res`	resolution as in `grDevices::png()`
`...`	additional arguments as in `grDevices::png()`
`which`	An integer specifying a device number as in `grDevices::dev.off()`

Details

Remote R PNG Device

See Also

`rDevices`

Examples

```
## Not run:
### Prompts are listed to clarify when something is eval'd locally vs
### remotely
> # suppressMessages(library(remoter, quietly = TRUE))
> # client()
> remoter::client("192.168.56.101")

remoter> plot(1:5)
remoter> rpng.off()

remoter> rpng()
remoter> plot(iris$Sepal.Length, iris$Petal.Length)
remoter> rpng.off()

remoter> library(ggplot2)
remoter> g1 <- ggplot(iris, aes(x = Sepal.Length, y = Petal.Length,
remoter+              color = Species)) +
remoter+       geom_point(aes(shape = Species))
remoter> rpng()
remoter> print(g1)
remoter> rpng.off()

remoter> g1 + geom_smooth(method = "lm")

remoter> rpng.new(plot(1:5))

remoter> rpng.new(g1)

remoter> b <- function() plot(iris$Sepal.Length, iris$Petal.Length)
remoter> rpng.new(b)

remoter> da <- data.frame(x = rnorm(100), y = rnorm(100))
remoter> g2 <- ggplot(da, aes(x, y)) + geom_point()
remoter> g2

remoter> pdf()
remoter> g2
remoter> print(g2 + geom_line())
remoter> dev.off()

remoter> q()
>

## End(Not run)
```

---

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. Must be the form of a character string. 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.

A `newname`, if specified, must be passed as a string (not a literal; i.e., `"mynewname"`, not `mynewname`). The name must also be syntactically valid (see `?make.names`).

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,
  userpng = TRUE, sync = TRUE)
```

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. If the value is 0, then a random open port will be selected.
`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.
`userpng`	Logical; if TRUE, rpng is set as the default device for displaying.
`sync`	Logical; if TRUE, the client will have `str()`'d versions of server objects recreated in the global environment. This is useful in IDE's like RStudio, but it carries a performance penalty. For terminal users, this is not recommended.

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