Package: ssh (via r-universe)

August 30, 2026

Type Package

Title Secure Shell (SSH) Client for R

Version 0.9.4

Description Connect to a remote server over SSH to transfer files via SCP, setup a secure tunnel, or run a command or script on the host while streaming stdout and stderr directly to the client.

License MIT + file LICENSE

Encoding UTF-8

SystemRequirements libssh >= 0.6.0 (the original, not libssh2)

RoxygenNote 7.1.1

Roxygen list(markdown = TRUE)

Imports credentials, askpass

Suggests knitr, rmarkdown, spelling, sys, testthat, mongolite

Language en-GB

URL https://docs.ropensci.org/ssh/ https://ropensci.r-universe.dev/ssh

BugReports https://github.com/ropensci/ssh/issues

VignetteBuilder knitr

Config/pak/sysreqs libssh-dev libssh2-1-dev libssl-dev

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

Date/Publication 2025-03-26 22:23:33 UTC

RemoteUrl https://github.com/ropensci/ssh

RemoteRef master

RemoteSha 01ec91838850159f78dbc87fca7336224a72c1ec

Contents

scp 2
ssh_connect 3
ssh_exec 4
ssh_tunnel 5

Index **6**

---

scp *SCP (Secure Copy)*

Description

Upload and download files to/from the SSH server via the scp protocol. Directories in the `files` argument are automatically traversed and uploaded / downloaded recursively.

Usage

```
scp_download(session, files, to = ".", verbose = TRUE)

scp_upload(session, files, to = ".", verbose = TRUE)
```

Arguments

`session`	ssh connection created with `ssh_connect()`
`files`	path to files or directory to transfer
`to`	existing directory on the destination where `files` will be copied into
`verbose`	print progress while copying files

Details

Note that the syntax is slightly different from the `scp` command line tool because the `to` parameter is always a target *directory* where all `files` will be copied **into**. If `to` does not exist, it will be created.

The `files` parameter in `scp_upload()` is vectorised hence all files and directories will be recursively uploaded **into** the `to` directory. For `scp_download()` the `files` parameter must be a single string which may contain wildcards.

The default path `to = "."` means that files get downloaded to the current working directory and uploaded to the user home directory on the server.

See Also

Other ssh: `ssh_connect()`, `ssh_credentials`, `ssh_exec`, `ssh_tunnel()`

Examples

```
## Not run:
# recursively upload files and directories
session <- ssh_connect("dev.opencpu.org")
files <- c(R.home("doc"), R.home("COPYING"))
scp_upload(session, files, to = "~/target")

# download it back
scp_download(session, "~/target/*", to = tempdir())
```

```
# delete it from the server
ssh_exec_wait(session, command = "rm -Rf ~/target")
ssh_disconnect(session)

## End(Not run)
```

---

ssh_connect *SSH Client*

Description

Create an ssh session using `ssh_connect()`. The session can be used to execute commands, scp files or setup a tunnel.

Usage

```
ssh_connect(host, keyfile = NULL, passwd = askpass, verbose = FALSE)

ssh_session_info(session)

ssh_disconnect(session)

libssh_version()
```

Arguments

`host`	an ssh server string of the form `[user@]hostname[:@port]`. An ipv6 hostname should be wrapped in brackets like this: `[2001:db8::1]:80`.
`keyfile`	path to private key file. Must be in OpenSSH format (see details)
`passwd`	either a string or a callback function for password prompt
`verbose`	either TRUE/FALSE or a value between 0 and 4 indicating log level: 0: no logging, 1: only warnings, 2: protocol, 3: packets or 4: full stack trace.
`session`	ssh connection created with `ssh_connect()`

Details

The client first tries to authenticate using a private key, either from ssh-agent or `/.ssh/id_rsa` in the user home directory. If this fails it falls back on challenge-response (interactive) and password auth if allowed by the server. The `passwd` parameter can be used to provide a passphrase or a callback function to ask prompt the user for the passphrase when needed.

The session will automatically be disconnected when the session object is removed or when R exits but you can also use `ssh_disconnect()`.

Windows users: the private key must be in OpenSSH PEM format. If you open it in a text editor the first line must be: `-----BEGIN RSA PRIVATE KEY-----`. To convert a Putty PKK key, open it in the *PuttyGen* utility and go to *Conversions* -> *Export OpenSSH*.

See Also

Other ssh: scp, ssh_credentials, ssh_exec, ssh_tunnel()

Examples

```
## Not run:
session <- ssh_connect("dev.opencpu.org")
ssh_exec_wait(session, command = "whoami")
ssh_disconnect(session)

## End(Not run)
```

---

ssh_exec *Execute Remote Command*

Description

Run a command or script on the host while streaming stdout and stderr directly to the client.

Usage

```
ssh_exec_wait(
  session,
  command = "whoami",
  std_out = stdout(),
  std_err = stderr()
)

ssh_exec_internal(session, command = "whoami", error = TRUE)
```

Arguments

session	ssh connection created with ssh_connect()
command	The command or script to execute
std_out	callback function, filename, or connection object to handle stdout stream
std_err	callback function, filename, or connection object to handle stderr stream
error	automatically raise an error if the exit status is non-zero

Details

The ssh_exec_wait() function is the remote equivalent of the local sys::exec_wait(). It runs a command or script on the ssh server and streams stdout and stderr to the client to a file or connection. When done it returns the exit status for the remotely executed command.

Similarly ssh_exec_internal() is a small wrapper analogous to sys::exec_internal(). It buffers all stdout and stderr output into a raw vector and returns it in a list along with the exit status. By default this function raises an error if the remote command was unsuccessful.

See Also

Other ssh: scp, ssh_connect(), ssh_credentials, ssh_tunnel()

Examples

```
## Not run:
session <- ssh_connect("dev.opencpu.org")
ssh_exec_wait(session, command = c(
  'curl -O https://cran.r-project.org/src/contrib/jsonlite_1.5.tar.gz',
  'R CMD check jsonlite_1.5.tar.gz',
  'rm -f jsonlite_1.5.tar.gz'
))
ssh_disconnect(session)
## End(Not run)
```

---

ssh_tunnel *Create SSH tunnel*

Description

Opens a port on your machine and tunnel all traffic to a custom target host via the SSH server, for example to connect with a database server behind a firewall.

Usage

```
ssh_tunnel(session, port = 5555, target = "rainmaker.wunderground.com:23")
```

Arguments

session	ssh connection created with ssh_connect()
port	integer of local port on which to listen for incoming connections
target	string with target host and port to connect to via ssh tunnel

Details

This function blocks while the tunnel is active. Use the tunnel by connecting to `localhost:5555` from a separate process. Each tunnel can only be used once and will automatically be closed when the client disconnects. It is intended to tunnel a single connection, not as a long running proxy server.

See Also

Other ssh: scp, ssh_connect(), ssh_credentials, ssh_exec

Index

∗ **ssh**
  scp, 2
  ssh_connect, 3
  ssh_exec, 4
  ssh_tunnel, 5

libssh_version *(*ssh_connect*)*, 3

scp, 2, *4, 5*
scp_download *(*scp*)*, 2
scp_download(), *2*
scp_upload *(*scp*)*, 2
scp_upload(), *2*
ssh *(*ssh_connect*)*, 3
ssh_connect, *2*, 3, *5*
ssh_connect(), *2–5*
ssh_credentials, *2*, *4, 5*
ssh_disconnect *(*ssh_connect*)*, 3
ssh_disconnect(), *3*
ssh_exec, *2*, 4, *4, 5*
ssh_exec_internal *(*ssh_exec*)*, 4
ssh_exec_internal(), *4*
ssh_exec_wait *(*ssh_exec*)*, 4
ssh_exec_wait(), *4*
ssh_info *(*ssh_connect*)*, 3
ssh_session_info *(*ssh_connect*)*, 3
ssh_tunnel, *2*, *4*, 5, *5*
sys::exec_internal(), *4*
sys::exec_wait(), *4*