Have you ever needed to connect to a remote server over SSH to transfer files via SCP or to setup a secure tunnel, and wished you could do so from R itself? The new rOpenSci ssh package provides a native ssh client in R allows you to do that and even more, like running a command or script on the host while streaming stdout and stderr directly to the client. The package is based on libssh, a powerful C library implementing the SSH protocol.

install.packages("ssh")

Because the ssh package is based on libssh it does not need to shell out. Therefore it works natively on all platforms without any runtime dependencies. Even on Windows.

The package is still work in progress, but the core functionality should work.

## **Installation**

On Windows or MacOS you can install the binary package directly from CRAN:

install.packages("ssh")

Installation from source requires libssh (which is **not** libssh2). On **Debian** or **Ubuntu** use libssh-dev:

sudo apt-get install -y libssh-dev

On **Fedora** we need libssh-devel:

sudo yum install libssh-devel

On **CentOS / RHEL** we install libssh-devel via EPEL:

sudo yum install epel-release

sudo yum install libssh-devel

On **OS-X** use libssh from Homebrew:

brew install libssh

## **Connecting to an SSH server**

First create an ssh session by connecting to an SSH server.

session <- ssh\_connect("jeroen@dev.opencpu.org")

print(session)

<ssh session>

connected: jeroen@dev.opencpu.org:22

server: 1c:86:dc:b9:77:2c:74:ee:2d:00:49:6f:2e:c8:b2:61:5e:14:53:73

Once established, a session is closed automatically by the garbage collector when the object goes out of scope or when R quits. You can also manually close it using ssh\_disconnect() but this is not strictly needed.

## **Authentication**

The client attempts to use the following authentication methods (in this order) until one succeeds:

1. try key from privkey argument in ssh\_connect() if specified
2. if ssh-agent is available, try private key from ssh-agent
3. try user key specified in ~/.ssh/config or any of the default locations: ~/.ssh/id\_ed25519, ~/.ssh/id\_ecdsa, ~/.ssh/id\_rsa, or .ssh/id\_dsa.
4. Try challenge-response password authentication (if permitted by the server)
5. Try plain password authentication (if permitted by the server)

To debug authentication set verbosity to at least level 2 or 3:

session <- ssh\_connect("jeroen@dev.opencpu.org", verbose = 2)

ssh\_socket\_connect: Nonblocking connection socket: 7

ssh\_connect: Socket connecting, now waiting for the callbacks to work

socket\_callback\_connected: Socket connection callback: 1 (0)

ssh\_client\_connection\_callback: SSH server banner: SSH-2.0-OpenSSH\_7.2p2 Ubuntu-4ubuntu2.4

ssh\_analyze\_banner: Analyzing banner: SSH-2.0-OpenSSH\_7.2p2 Ubuntu-4ubuntu2.4

ssh\_analyze\_banner: We are talking to an OpenSSH client version: 7.2 (70200)

ssh\_packet\_dh\_reply: Received SSH\_KEXDH\_REPLY

ssh\_client\_curve25519\_reply: SSH\_MSG\_NEWKEYS sent

ssh\_packet\_newkeys: Received SSH\_MSG\_NEWKEYS

ssh\_packet\_newkeys: Signature verified and valid

ssh\_packet\_userauth\_failure: Access denied. Authentication that can continue: publickey

ssh\_packet\_userauth\_failure: Access denied. Authentication that can continue: publickey

ssh\_agent\_get\_ident\_count: Answer type: 12, expected answer: 12

ssh\_userauth\_publickey\_auto: Successfully authenticated using /Users/jeroen/.ssh/id\_rsa

Tools for setting up and debugging your ssh keys are provided via the credentials package.

ssh\_key\_info()

$key

[1] "/Users/jeroen/.ssh/id\_rsa"

$pubkey

[1] "ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQDwI/G2v42n/fVFbuyOL/7eh06MOK71djbe7wr8op0LcMbCiMOHOFdMRpb+UcwlfnXDYWwNZsfr3l4Hvfbdi5CesDwZmx4FQ4BC4e+2IRqCbwknVwdIMfBwLWd9HNs8hJ8M7YW4lt0TucMtVkzGN4wFR+onvzyU0VSiSiT1JQUY+7g4YMJlmhKN3lES+as1RFMExLosFlzydD7o35nMuKp1VzdkUaWm3xjFCTl7MVl6cbCWh7m3IDl95tTT0jH+pxvK2Vgb3u+r+tH7dWbbjm9tkwH7NS5HwEjy08cKUv+8BUnDLC27EuK2/k+Ro6BsEDHmq0z0nf6FgpTXQ0iJxnUZ"

## **Execute Script or Command**

Run a command or script on the host and block while it runs. By default stdout and stderr are steamed directly back to the client. This function returns the exit status of the remote command (hence it does not automatically error for an unsuccessful exit status).

out <- ssh\_exec\_wait(session, command = 'whoami')

jeroen

print(out)

[1] 0

You can also run a script that consists of multiple commands.

ssh\_exec\_wait(session, command = c(

'curl -O https://cran.r-project.org/src/contrib/Archive/jsonlite/jsonlite\_1.4.tar.gz',

'R CMD check jsonlite\_1.4.tar.gz',

'rm -f jsonlite\_1.4.tar.gz'

))

% Total % Received % Xferd Average Speed Time Time Time Current

Dload Upload Total Spent Left Speed

100 1071k 100 1071k 0 0 654k 0 0:00:01 0:00:01 --:--:-- 654k

\* using log directory '/home/jeroen/jsonlite.Rcheck'

\* using R version 3.4.3 (2017-11-30)

\* using platform: x86\_64-pc-linux-gnu (64-bit)

\* using session charset: ASCII

\* checking for file 'jsonlite/DESCRIPTION' ... OK

\* this is package 'jsonlite' version '1.4'

\* checking package namespace information ... OK

\* checking package dependencies ...

...

### **Capturing output**

The ssh\_exec\_internal() is a convenient wrapper for ssh\_exec\_wait() which buffers the output steams and returns them as a raw vector. Also it raises an error by default when the remote command was not successful.

out <- ssh\_exec\_internal(session, "R -e 'rnorm(10)'")

print(out$status)

[1] 0

cat(rawToChar(out$stdout))

R version 3.5.3 (2019-03-11) -- "Great Truth"

Copyright (C) 2019 The R Foundation for Statistical Computing

Platform: x86\_64-pc-linux-gnu (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.

You are welcome to redistribute it under certain conditions.

Type 'license()' or 'licence()' for distribution details.

Natural language support but running in an English locale

R is a collaborative project with many contributors.

Type 'contributors()' for more information and

'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or

'help.start()' for an HTML browser interface to help.

Type 'q()' to quit R.

> rnorm(10)

[1] -0.4657579 0.1628843 -1.8308908 -1.4333284 -1.6689843 -0.3603318

[7] -0.3299621 -0.2162147 1.0788035 -0.9536821

>

>

This function is very useful if you are running a remote command and want to use it’s output as if you had executed it locally.

### **Using sudo**

Note that the exec functions are non interactive so they cannot prompt for a sudo password. A trick is to use -S which reads the password from stdin:

command <- 'echo "mypassword!" | sudo -s -S apt-get update -y'

out <- ssh\_exec\_wait(session, command)

Be very careful with hardcoding passwords!

## **Transfer Files via SCP**

Upload and download files via SCP. Directories are automatically traversed as in scp -r.

*# Upload a file to the server*

file\_path <- R.home("COPYING")

scp\_upload(session, file\_path)

*# Download the file back and verify it is the same*

scp\_download(session, "COPYING", to = tempdir())

tools::md5sum(file\_path)

tools::md5sum(file.path(tempdir(), "COPYING"))

## **Hosting a Tunnel**

Opens a port on your machine and tunnel all traffic to a custom target host via the SSH server.

ssh\_tunnel(session, port = 5555, target = "ds043942.mongolab.com:43942")

This function blocks while the tunnel is active. Use the tunnel by connecting to localhost:5555 from a separate process. The tunnel can only be used once and will automatically be closed when the client disconnects.

## **Disconnecting**

When you are done with the session you should disconnect:

ssh\_disconnect(session)

If you forgot to disconnect, the garbage collector will do so for you (with a warning).

**Connecting to an SSH server**

First create an ssh session by connecting to an SSH server.

session <- ssh\_connect("ABC@dev.opencpu.org")

print(session)

##

## connected: jeroen@dev.opencpu.org:22

## server: 1e:28:44:af:84:91:e5:88:fe:82:ca:34:d7:c8:cf:a8:0d:2f:ec:af

Once established, a session is closed automatically by the garbage collector when the object goes out of scope or when R quits. You can also manually close it using ssh\_disconnect() but this is not strictly needed.

**Authentication**

The client attempts to use the following authentication methods (in this order) until one succeeds:

1. try key from privkey argument in ssh\_connect() if specified
2. if ssh-agent is available, try private key from ssh-agent
3. try user key specified in ~/.ssh/config or any of the default locations: ~/.ssh/id\_ed25519, ~/.ssh/id\_ecdsa, ~/.ssh/id\_rsa, or .ssh/id\_dsa.
4. Try challenge-response password authentication (if permitted by the server)
5. Try plain password authentication (if permitted by the server)

To debug authentication set verbosity to at least level 2 or 3:

session <- ssh\_connect("jeroen@dev.opencpu.org", verbose = 2)

## ssh\_socket\_connect: Nonblocking connection socket: 7

## ssh\_connect: Socket connecting, now waiting for the callbacks to work

## socket\_callback\_connected: Socket connection callback: 1 (0)

## ssh\_client\_connection\_callback: SSH server banner: SSH-2.0-OpenSSH\_7.2p2 Ubuntu-4ubuntu2.4

## ssh\_analyze\_banner: Analyzing banner: SSH-2.0-OpenSSH\_7.2p2 Ubuntu-4ubuntu2.4

## ssh\_analyze\_banner: We are talking to an OpenSSH client version: 7.2 (70200)

## ssh\_packet\_dh\_reply: Received SSH\_KEXDH\_REPLY

## ssh\_client\_curve25519\_reply: SSH\_MSG\_NEWKEYS sent

## ssh\_packet\_newkeys: Received SSH\_MSG\_NEWKEYS

## ssh\_packet\_newkeys: Signature verified and valid

## ssh\_packet\_userauth\_failure: Access denied. Authentication that can continue: publickey

## ssh\_packet\_userauth\_failure: Access denied. Authentication that can continue: publickey

## ssh\_agent\_get\_ident\_count: Answer type: 12, expected answer: 12

## ssh\_userauth\_publickey\_auto: Successfully authenticated using /Users/jeroen/.ssh/id\_rsa

**Execute Script or Command**

Run a command or script on the host and block while it runs. By default stdout and stderr are steamed directly back to the client. This function returns the exit status of the remote command (hence it does not automatically error for an unsuccessful exit status).

out <- ssh\_exec\_wait(session, command = 'whoami')

## jeroen

print(out)

## [1] 0

You can also run a script that consists of multiple commands.

ssh\_exec\_wait(session, command = c(

'curl -O https://cran.r-project.org/src/contrib/Archive/jsonlite/jsonlite\_1.4.tar.gz',

'R CMD check jsonlite\_1.4.tar.gz',

'rm -f jsonlite\_1.4.tar.gz'

))

## % Total % Received % Xferd Average Speed Time Time Time Current

## Dload Upload Total Spent Left Speed

## 100 1071k 100 1071k 0 0 654k 0 0:00:01 0:00:01 --:--:-- 654k

## \* using log directory '/home/jeroen/jsonlite.Rcheck'

## \* using R version 3.4.3 (2017-11-30)

## \* using platform: x86\_64-pc-linux-gnu (64-bit)

## \* using session charset: ASCII

## \* checking for file 'jsonlite/DESCRIPTION' ... OK

## \* this is package 'jsonlite' version '1.4'

## \* checking package namespace information ... OK

## \* checking package dependencies ...

...

**Capturing output**

The ssh\_exec\_internal() is a convenient wrapper for ssh\_exec\_wait() which buffers the output steams and returns them as a raw vector. Also it raises an error by default when the remote command was not successful.

out <- ssh\_exec\_internal(session, "R -e 'rnorm(10)'")

print(out$status)

## [1] 0

cat(rawToChar(out$stdout))

## R version 3.4.4 (2018-03-15) -- "Someone to Lean On"

## Copyright (C) 2018 The R Foundation for Statistical Computing

## Platform: x86\_64-pc-linux-gnu (64-bit)

##

## R is free software and comes with ABSOLUTELY NO WARRANTY.

## You are welcome to redistribute it under certain conditions.

## Type 'license()' or 'licence()' for distribution details.

##

## R is a collaborative project with many contributors.

## Type 'contributors()' for more information and

## 'citation()' on how to cite R or R packages in publications.

##

## Type 'demo()' for some demos, 'help()' for on-line help, or

## 'help.start()' for an HTML browser interface to help.

## Type 'q()' to quit R.

##

## > rnorm(10)

## [1] 0.14301778 -0.26873489 0.83931307 0.22034917 0.87214122 -0.13655736

## [7] -0.08793867 -0.68616146 0.23469591 0.93871035

This function is very useful if you are running a remote command and want to use it’s output as if you had executed it locally.

**Using sudo**

Note that the exec functions are non interactive so they cannot prompt for a sudo password. A trick is to use -S which reads the password from stdin:

command <- 'echo "mypassword!" | sudo -s -S apt-get update -y'

out <- ssh\_exec\_wait(session, command)

Be very careful with hardcoding passwords!

**Transfer Files via SCP**

Upload and download files via SCP. Directories are automatically traversed as in scp -r.

# Upload a file to the server

file\_path <- R.home("COPYING")

scp\_upload(session, file\_path)

## [100%] /Library/Frameworks/R.framework/Versions/3.5/Resources/COPYING

This will upload the file to the home directory on your server. Let’s download it back:

# Download the file back and verify it is the same

scp\_download(session, "COPYING", to = tempdir())

## 18011 /var/folders/l8/bhmtp25n2lx0q0dgv1x4gf1w0000gn/T//Rtmpldz4eO/COPYING

We can compare the checksums to verify that the files are identical:

tools::md5sum(file\_path)

## "eb723b61539feef013de476e68b5c50a"

tools::md5sum(file.path(tempdir(), "COPYING"))

## "eb723b61539feef013de476e68b5c50a"

**Hosting a Tunnel**

Opens a port on your machine and tunnel all traffic to a custom target host via the SSH server.

ssh\_tunnel(session, port = 5555, target = "ds043942.mongolab.com:43942")

This function blocks while the tunnel is active. Use the tunnel by connecting to localhost:5555 from a separate process. The tunnel can only be used once and will automatically be closed when the client disconnects.