pystassh Documentation

Release 1.2.1

Julien Chaumont

Apr 05, 2022

Contents

An easy to use libssh wrapper to execute commands on a remote server via SSH with Python.

You can go straight to the API documentation here: *Reference/API*.

CHAPTER 1

Description

`pystassh` is a Python 3 library that allows you to easily run commands on a remote server via SSH.

CHAPTER 2

Compatibility

Currently supported versions and implementations of Python are:

- python 3.6+
- pypy3

Others might be compatible but are not officially supported.

CHAPTER 3

Requirements

Required libraries are automatically installed when installing `pystassh` (see *Installation* to learn more). You can install them by running `pip install -r requirements.txt`.

`libssh` and `libffi-dev` may have to be installed separately.

CHAPTER 4

Documentation

Refer to other sections of this documentation to learn more about `pystassh`.

4.1 Installation

4.1.1 Using pip

The `pystassh` library is hosted on pypi and can be installed with `pip`:

```
$ pip install pystassh
```

4.1.2 From the Sources

First download the code from the GitHub repository:

```
$ cd /tmp/
$ git clone https://github.com/julienc91/pystassh
$ cd pystassh/
$ python setup.py install
```

4.2 Examples

Establishing a connection:

```
>>> from pystassh import Session
>>> # With default private key
>>> session = Session('remote_host.org', username='user')
>>> # With username and password
```

(continues on next page)

(continued from previous page)

```
>>> session = Session('remote_host.org', username='foo', password='bar')
>>> # With specific private key and a passphrase
>>> session = Session('remote_host.org', privkey_file='/home/user/.ssh/my_key',␣
↪passphrase='baz')
```

Running simple commands:

```
>>> from pystassh import Session
>>> with Session('remote_host.org', username='user') as ssh_session:
...     res = ssh_session.execute('whoami')
>>> res.stdout
'foo'
```

Handling errors:

```
>>> from pystassh import Session
>>> with Session('remote_host.org', username='user') as ssh_session:
...     res = ssh_session.execute('whoam')
>>> res.stderr
'bash: whoam : command not found'
```

Running multiple commands:

```
>>> from pystassh import Session
>>> with Session('remote_host.org', username='user') as ssh_session:
...     ssh_session.execute('echo "bar" > /tmp/foo')
...     res = ssh_session.execute('cat /tmp/foo')
>>> res.stdout
'bar'
```

Using a session without a `with` block:

```
>>> from pystassh import Session
>>> ssh_session = Session('remote_host.org', username='user')
>>> ssh_session.connect()
>>> res = ssh_session.execute('whoami')
>>> res.stdout
'foo'
>>> ssh_session.disconnect()
```

Using a shell:

```
>>> from pystassh import Session
>>> with Session('remote_host.org', username='user') as ssh_session:
...     channel = ssh_session.channel
...     with channel:
...         channel.request_shell(request_pty=False)
...         # non blocking read to flush the motd, if there is one
...         channel.read_nonblocking(1024)
...         channel.write("export foo=42;\n")
...         channel.write("echo $foo;\n")
...         channel.read(2048)
b'42\n'
```

4.3 Tests

Each release of `pystassh` is garantied to be delivered with a complete set of unit tests that cover the entirety of the code. Of course, it cannot be the assurance of a completely devoid of bug source code; but that's something at least, right?

Tu run the tests, clone the GitHub repository (see *From the Sources*) and install the dependencies:

```
$ pip install requirements.txt
```

The run the test suite with:

```
$ pytest .
```

Code coverage can also be tested with the *pytest-cov <https://pypi.python.org/pypi/pytest-cov>* package this way:

```
$ pytest --cov pystassh
```

4.4 Reference/API

4.4.1 pystassh.session module

The Session object is the entry point of any command execution via SSH.

Examples

Open a SSH connection, run the "ls" command and print the output.

```
>>> with Session('localhost', 'foo', 'bar') as ssh_session:
...     result = ssh_session.execute('ls')
...     print(result.stdout)
```

class pystassh.session.**Session**(*hostname='localhost'*, *username=''*, *password=''*, *passphrase=''*, *port=22*, *privkey_file=''*)

Bases: `object`

channel

connect()

Initiate the connection and authentication process on the remote server.

Raises

- `ConnectionException` – if an error occurred during the connection process
- `AuthenticationException` – if an error occurred during the authentication process

disconnect()

Close the current connection.

execute(*command*)

Execute a command on the remote server.

Parameters **command** (*str*) – the command to run

Returns the Result object for this command

Return type *Result*

get_error_message(*session=None*)
Tries to retrieve an error message in case of error.

Parameters **session** – a session object that will be used instead of self._session

Returns An error message

Return type str

is_connected()
Check if the connexion is currently active.

Returns A boolean indicating whether or not the connexion is currently active.

Return type bool

4.4.2 pystassh.channel module

class pystassh.channel.**Channel**(*session*)
Bases: object

close()
Close the current channel.

execute(*command*)
Execute a command.

Parameters **command** (*str*) – the command to run

Returns the Result object for this command

Return type *Result*

get_error_message()
Tries to retrieve an error message in case of error.

Returns An error message

Return type str

is_eof()
Check if remote has sent an EOF.

open()
Open a new channel.

Raises ChannelException – if the channel could not be correctly initialized

read(*size=2048*, *from_stderr=False*)
Reads data from a channel. The read will block.

Parameters

- **size** (*int*) – bytes to read.
- **from_stderr** (*bool*) – read from standard error instead from stdout.

Returns the string read. Returns an empty string on EOF.

Return type string (str)

read_nonblocking(*size=2048*, *from_stderr=False*)
Do a nonblocking read on the channel.

Parameters

- **size** (*int*) – bytes to try to read atomically and without blocking.
- **from_stderr** (*bool*) – read from standard error instead from stdout.

Returns

the string read. It may be shorter than the expected size. An empty string does not imply an EOF: you still have to check it.

Return type string (str)

request_shell(*request_pty=False*)
Request a shell and optionally a PTY.

write(*data*)
Blocking write on a channel.

Parameters **data** (*str*) – data to encode (to bytes) and write (not binary safe).

Results: The number of bytes written.

4.4.3 pystassh.result module

class pystassh.result.**Result**(*channel*, *command*)
Bases: object

command
The command from wich the current results came from.

raw_stderr
The raw content of the standard error output, as a list of bytes.

raw_stdout
The raw content of the standard output, as a list of bytes.

return_code
The return code of the last command as an int.

stderr
The content of the standard error output, as a string. Decoding errors are not caught at this level.

stdout
The content of the standard output, as a string. Decoding errors are not caught at this level.

4.5 About

pystassh is open-source and published under the MIT License. It is a permissive license and therefore let people do whatever they want with the library as long as the attribution is made cleared. The author of the library cannot be held responsible for any use which may be made of pystassh.

Author Julien Chaumont - https://julienc.io

License MIT - https://opensource.org/licenses/MIT

Repository GitHub - https://github.com/julienc91/pystassh

4.5.1 Report a Bug

Please use the GitHub bugtracker to report any problem you might have with `pystassh`:

https://github.com/julienc91/pystassh/issues

4.5.2 Contribute

Contributions are welcome. Learn how to help on GitHub:

https://guides.github.com/activities/contributing-to-open-source/

4.5.3 Contact

Feel free to contact me by email for any question related to `pystassh` or to my work in general at `pystassh[at]julienc.io`.

Python Module Index

p

Index