sshutil Documentation

Christian E. Hopps

Sep 29, 2018

Contents

sshutil supports caching authenticated ssh connections and provides ease of use methods to open sessions and run commands on remote hosts.

All connections are by default cached for a short time so no extra work is required to take advantage of the caching.

Contents:

CHAPTER 1

Installation

At the command line:

```
$ pip install sshutil
```

Or, if you have virtualenvwrapper installed:

```
$ mkvirtualenv sshutil
$ pip install sshutil
```

CHAPTER 2

Usage

To use sshutil in a project:

```
import sshutil
```

To run a command over SSH:

```
from sshutil.cmd import SSHCommand

cmd = SSHCommand("hostname", "red.example.com")
assert "red" == output.cmd.run()
```

To read and write to a command over SSH:

```
from sshutil.conn import SSHCommandSession

session = SSHCommandSession("cat", "red.example.com")

s = "testing\n"
session.sendall(s)

rs = session.recv(len(s))
assert rs == s
```

To run many commands on a host:

```
from sshutil.host import Host

host = Host("red.example.com")
assert "red" == host.run("hostname")
assert "red.example.com" == host.run("hostname -f")
```

To globally disable ssh connection caching:

```
import sshutil

sshutil.DisableGlobalCaching()
```

CHAPTER 3

Reference

3.1 The `sshutil.cache` Module

class sshutil.cache.**SSHConnectionCache**(*desc=u”*, *close_timeout=1*, *max_channels=8*)
Bases: sshutil.cache._SSHConnectionCache

An ssh connection cache.

Authenticated connections to a given host are cached for a specified amount of time before being closed. This allows for connection reuse as well as avoiding re-authentication.

Parameters

- **close_timeout** – Amount of time to wait before closing an opened unused ssh socket.
- **max_channels** – Maximum number of channels to open on a given ssh socket.

flush(*debug=False*)
Flush (close) any un-referenced open entries currently waiting for timeout.

get_ssh_socket(*host*, *port*, *username*, *password*, *debug*, *proxycmd=None*)
Returns a socket to the given host using the given credentials.

If a socket has already been opened with the supplied arguments, then it will be reference counted and returned. Otherwise a new socket will be opened. If *usernamee* is *None* getpass will be used to obtain the current user name.

Parameters

- **host** – The hostname to connect to.
- **port** – The TCP port number to connect to.
- **username** – The username to use for authentication or *None*. If *None* then *getpass.get_user()* will be used.
- **password** – The password/key for authentication or None.
- **debug** – Boolean indicating if debug messages should be enabled.

- **proxycmd** – A proxy command to use when making the ssh connection.

Raises ssh.AuthenticationException

release_ssh_socket(*ssh_socket*, *debug*)
Release a refrence on an open socket.

release_ssh_socket must be paired with each call to *get_ssh_socket*.

class sshutil.cache.**SSHNoConnectionCache**(*desc=u''*)
Bases: sshutil.cache._SSHConnectionCache

Simple non-caching cache class

flush(*debug=False*)

get_ssh_socket(*host*, *port*, *username*, *password*, *debug*, *proxycmd=None*)

release_ssh_socket(*ssh_socket*, *debug=False*)

3.2 The sshutil.cmd Module

exception sshutil.cmd.**CalledProcessError**(*code*, *command*, *output=None*, *error=None*)
Bases: subprocess.CalledProcessError

class sshutil.cmd.**SSHCommand**(*command*, *host*, *port=22*, *username=None*, *password=None*, *debug=False*, *cache=None*, *proxycmd=None*)
Bases: *sshutil.conn.SSHConnection*

run()
Run a command, return stdout.

Returns stdout

Raises CalledProcessError

```
>>> cmd = SSHCommand("ls -d /etc", "localhost")
>>> print(cmd.run(), end="")
/etc
>>> cmd = SSHCommand("grep foobar doesnt-exist", "localhost")
>>> cmd.run()
Traceback (most recent call last):
    ...
CalledProcessError: Command 'grep foobar doesnt-exist' returned non-zero exit␣
↪status 2
```

run_status()
Run a command, return exitcode and stdout.

Returns (status, stdout)

```
>>> status, output = SSHCommand("ls -d /etc", "localhost").run_status()
>>> status
0
>>> print(output, end="")
/etc
>>> status, output = SSHCommand("grep foobar doesnt-exist", "localhost").run_
↪status()
>>> status
2
>>> print(output, end="")
```

run_status_stderr()
Run the command returning exit code, stdout and stderr.

Returns (returncode, stdout, stderr)

```
>>> status, output, error = SSHCommand("ls -d /etc", "localhost").run_status_
↪stderr()
>>> status
0
>>> print(output, end="")
/etc
>>> print(error, end="")
>>> status, output, error = SSHCommand("grep foobar doesnt-exist", "localhost
↪").run_status_stderr()
>>> status
2
>>> print(output, end="")
>>>
>>> print(error, end="")
grep: doesnt-exist: No such file or directory
```

run_stderr()
Run a command, return stdout and stderr,

Returns (stdout, stderr)

Raises CalledProcessError

```
>>> cmd = SSHCommand("ls -d /etc", "localhost")
>>> output, error = cmd.run_stderr()
>>> print(output, end="")
/etc
>>> print(error, end="")
>>> cmd = SSHCommand("grep foobar doesnt-exist", "localhost")
>>> cmd.run_stderr()
Traceback (most recent call last):
    ...
CalledProcessError: Command 'grep foobar doesnt-exist' returned non-zero exit␣
↪status 2
```

class sshutil.cmd.**SSHPTYCommand**(*command*, *host*, *port=22*, *username=None*, *password=None*, *debug=False*, *cache=None*, *proxycmd=None*)
Bases: *sshutil.cmd.SSHCommand*

Instances of this class also obtain a PTY prior to executing the command

class sshutil.cmd.**ShellCommand**(*command*, *debug=False*)
Bases: object

run()
Run a command over an ssh channel, return stdout. Raise CalledProcessError on failure.

```
>>> cmd = ShellCommand("ls -d /etc", False)
>>> print(cmd.run(), end="")
/etc
>>> cmd = ShellCommand("grep foobar doesnt-exist", False)
>>> cmd.run()
Traceback (most recent call last):
    ...
CalledProcessError: Command 'grep foobar doesnt-exist' returned non-zero exit␣
↪status 2
```

(continues on next page)

(continued from previous page)

run_status()
Run a command over an ssh channel, return exitcode and stdout.

```
>>> status, output = ShellCommand("ls -d /etc").run_status()
>>> status
0
>>> print(output, end="")
/etc
>>> status, output = ShellCommand("grep foobar doesnt-exist").run_status()
>>> status
2
>>> print(output, end="")
```

run_status_stderr()
Run a command over an ssh channel, return exit code, stdout and stderr.

```
>>> cmd = ShellCommand("ls -d /etc")
>>> status, output, error = cmd.run_status_stderr()
>>> status
0
>>> print(output, end="")
/etc
>>> print(error, end="")
```

run_stderr()
Run a command over an ssh channel, return stdout and stderr, Raise CalledProcessError on failure

```
>>> cmd = ShellCommand("ls -d /etc")
>>> output, error = cmd.run_stderr()
>>> print(output, end="")
/etc
>>> print(error, end="")
>>> cmd = ShellCommand("grep foobar doesnt-exist")
>>> cmd.run_stderr()
Traceback (most recent call last):
    ...
CalledProcessError: Command 'grep foobar doesnt-exist' returned non-zero exit␣
↪status 2
```

sshutil.cmd.**read_to_eof**(*recvmethod*)

sshutil.cmd.**setup_module**(_)

sshutil.cmd.**shell_escape_single_quote**(*command*)
Escape single quotes for use in a shell single quoted string Explanation:

1. End first quotation which uses single quotes.
2. Start second quotation, using double-quotes.
3. Quoted character.
4. End second quotation, using double-quotes.
5. Start third quotation, using single quotes.

If you do not place any whitespaces between (1) and (2), or between (4) and (5), the shell will interpret that string as a one long word

sshutil.cmd.**terminal_size**()

3.3 The sshutil.conn Module

class sshutil.conn.**SSHClientSession**(*host*, *port*, *subsystem*, *username=None*, *password=None*, *debug=False*, *cache=None*, *proxycmd=None*)

Bases: *sshutil.conn.SSHSession*

A client session to a host using a subsystem.

class sshutil.conn.**SSHCommandSession**(*host*, *port*, *command*, *username=None*, *password=None*, *debug=False*, *cache=None*, *proxycmd=None*)

Bases: *sshutil.conn.SSHSession*

A client session to a host using a command i.e., like a remote pipe

Objects of this class are useful for long running commands. For run-to-completion commands SSHCommand should be used.

recv_exit_status()

class sshutil.conn.**SSHConnection**(*host*, *port=22*, *username=None*, *password=None*, *debug=False*, *cache=None*, *proxycmd=None*)

Bases: object

A connection to an SSH server

close()

is_active()

class sshutil.conn.**SSHSession**(*host*, *port=22*, *username=None*, *password=None*, *debug=False*, *cache=None*, *proxycmd=None*)

Bases: *sshutil.conn.SSHConnection*

recv(*size=16384*)

recv_ready()

recv_stderr(*size=16384*)

recv_stderr_ready()

send(*chunk*)

sendall(*chunk*)

sshutil.conn.**shell_escape_single_quote**(*command*)

Escape single quotes for use in a shell single quoted string Explanation:

1. End first quotation which uses single quotes.
2. Start second quotation, using double-quotes.
3. Quoted character.
4. End second quotation, using double-quotes.
5. Start third quotation, using single quotes.

If you do not place any whitespaces between (1) and (2), or between (4) and (5), the shell will interpret that string as a one long word

3.4 The sshutil.host Module

class sshutil.host.**Host**(*server=None*, *port=22*, *cwd=None*, *username=None*, *password=None*, *debug=False*, *cache=None*, *proxycmd=None*)

Bases: object

A Host object is either local (shell) or remote host (ssh) and provides easy access to the given host for running commands etc.

copy_to(*localfile*, *remotefile*)

run(*command*)

Run a command, return stdout.

Returns stdout

Raises CalledProcessError

run_status(*command*)

Run a command, return exitcode and stdout.

Returns (status, stdout)

run_status_stderr(*command*)

Run the command returning exit code, stdout and stderr.

Returns (returncode, stdout, stderr)

```
>>> host = Host()
>>> status, output, error = host.run_status_stderr("ls -d /etc")
>>> status
0
>>> print(output, end="")
/etc
>>> print(error, end="")
>>> status, output, error = host.run_status_stderr("grep foobar doesnt-exist")
>>> status
2
>>> print(output, end="")
>>>
>>> print(error, end="")
grep: doesnt-exist: No such file or directory
```

run_stderr(*command*)

Run a command, return stdout and stderr,

Returns (stdout, stderr)

Raises CalledProcessError

3.5 The sshutil.server Module

class sshutil.server.**SSHServer**(*server_ctl=None*, *server_socket_class=None*, *server_session_class=None*, *extra_args=None*, *port=None*, *host_key=None*, *debug=False*)

Bases: object

An ssh server

close()

join()
Wait on server to terminate

remove_socket(*serversocket*)

class sshutil.server.**SSHServerSession**(*stream*, *server*, *extra_args*, *debug*)
Bases: object

close()

is_active()

reader_exits()

reader_handle_data(*data*)

reader_read_data()
Called by reader thread if a evaluate false value is returned thread exits

recv(*rlen*)

send(*data*)

class sshutil.server.**SSHServerSocket**(*server_ctl*, *session_class*, *extra_args*, *server*, *newsocket*, *addr*, *debug*)
Bases: object

An SSH socket connection from a client

close()

class sshutil.server.**SSHUserPassController**(*username=None*, *password=None*)
Bases: paramiko.server.ServerInterface

check_auth_none(*username*)
Determine if a client may open channels with no (further) authentication.

Return AUTH_FAILED if the client must authenticate, or AUTH_SUCCESSFUL if it's okay for the client to not authenticate.

The default implementation always returns AUTH_FAILED.

Parameters **username** (*str*) – the username of the client.

Returns AUTH_FAILED if the authentication fails; AUTH_SUCCESSFUL if it succeeds.

Return type int

check_auth_password(*username*, *password*)
Determine if a given username and password supplied by the client is acceptable for use in authentication.

Return AUTH_FAILED if the password is not accepted, AUTH_SUCCESSFUL if the password is accepted and completes the authentication, or AUTH_PARTIALLY_SUCCESSFUL if your authentication is stateful, and this key is accepted for authentication, but more authentication is required. (In this latter case, *get_allowed_auths* will be called to report to the client what options it has for continuing the authentication.)

The default implementation always returns AUTH_FAILED.

Parameters

- **username** (*str*) – the username of the authenticating client.
- **password** (*str*) – the password given by the client.

Returns AUTH_FAILED if the authentication fails; AUTH_SUCCESSFUL if it succeeds; AUTH_PARTIALLY_SUCCESSFUL if the password auth is successful, but authentication must continue.

Return type int

check_channel_request(*kind*, *chanid*)

Determine if a channel request of a given type will be granted, and return OPEN_SUCCEEDED or an error code. This method is called in server mode when the client requests a channel, after authentication is complete.

If you allow channel requests (and an ssh server that didn't would be useless), you should also override some of the channel request methods below, which are used to determine which services will be allowed on a given channel:

- *check_channel_pty_request*
- *check_channel_shell_request*
- *check_channel_subsystem_request*
- *check_channel_window_change_request*
- *check_channel_x11_request*
- *check_channel_forward_agent_request*

The chanid parameter is a small number that uniquely identifies the channel within a *.Transport*. A *.Channel* object is not created unless this method returns OPEN_SUCCEEDED – once a *.Channel* object is created, you can call *.Channel.get_id* to retrieve the channel ID.

The return value should either be OPEN_SUCCEEDED (or 0) to allow the channel request, or one of the following error codes to reject it:

- OPEN_FAILED_ADMINISTRATIVELY_PROHIBITED
- OPEN_FAILED_CONNECT_FAILED
- OPEN_FAILED_UNKNOWN_CHANNEL_TYPE
- OPEN_FAILED_RESOURCE_SHORTAGE

The default implementation always returns OPEN_FAILED_ADMINISTRATIVELY_PROHIBITED.

Parameters

- **kind** (*str*) – the kind of channel the client would like to open (usually "session").
- **chanid** (*int*) – ID of the channel

Returns an *int* success or failure code (listed above)

check_channel_subsystem_request(*channel*, *name*)

Determine if a requested subsystem will be provided to the client on the given channel. If this method returns True, all future I/O through this channel will be assumed to be connected to the requested subsystem. An example of a subsystem is sftp.

The default implementation checks for a subsystem handler assigned via *.Transport.set_subsystem_handler*. If one has been set, the handler is invoked and this method returns True. Otherwise it returns False.

Note: Because the default implementation uses the *.Transport* to identify valid subsystems, you probably won't need to override this method.

Parameters

- **channel** (*Channel*) – the .*Channel* the pty request arrived on.
- **name** (*str*) – name of the requested subsystem.

Returns `True` if this channel is now hooked up to the requested subsystem; `False` if that subsystem can't or won't be provided.

get_allowed_auths(*username*)

Return a list of authentication methods supported by the server. This list is sent to clients attempting to authenticate, to inform them of authentication methods that might be successful.

The "list" is actually a string of comma-separated names of types of authentication. Possible values are `"password"`, `"publickey"`, and `"none"`.

The default implementation always returns `"password"`.

Parameters **username** (*str*) – the username requesting authentication.

Returns a comma-separated *str* of authentication types

sshutil.server.**from_private_key_file**(*keyfile*, *password=None*)

Return a private key from a file, try all the types.

sshutil.server.**is_sock_closed**(*sock*)

Check to see if the socket is ready for reading but nothing is there, IOW it's closed

Python Module Index

S

Index

S

T