RedExpect Documentation

Release 2.0.3-3 stable

Red_M

Oct 10, 2021

CONTENTS:

CHAPTER

ONE

REDEXPECT

class redexpect.**RedExpect**(*prompt='.+?[\\#\\$]\\s+'*, *encoding='utf8'*, *newline='\r'*, *expect_timeout=300.0*, ***kwargs*)

Bases: redssh.redssh.RedSSH

Instances the start of an SSH connection. Extra options are available after *redexpect.RedExpect.login()* is called. This only takes the arguements below and the rest are defered to redssh.RedSSH.

Please also note that this class inherits from redssh.RedSSH and tries to not override anything from that.

Parameters

- **prompt** (regex string) – The basic prompt to expect for the first command line.
- **encoding** (str) – Set the encoding to something other than the default of 'utf16' when your target SSH server doesn't return UTF-16.
- **newline** (str) – Set the newline for sending and recieving text to the remote server.
- **expect_timeout** (float) – Set the timeout in seconds for when expecting a certain string to appear, this means that the string or regex has to be matched within this time. Set to 0 to disable.

check_closed(*channel=None*)

Returns True or False when the main channel has recieved an EOF or an associated channel.

close_tunnels()

Closes all SSH tunnels if any are open.

command(*cmd*, *clean_output=True*, *remove_newline=False*, *timeout=None*)

Run a command in the remote terminal.

Parameters

- **cmd** (str) – Command to execute, this will send characters exactly as if they were typed. (crtl+c could be sent via this).
- **clean_output** (bool) – Set to False to remove the "smart" cleaning, useful for debugging or for when you want the prompt as well.
- **remove_newline** (bool) – Set to True to remove the last newline on a return, useful when a command adds a newline to its output.
- **timeout** (float) – Set the timeout for this command to complete within, if set to None this will use the value of **:var:`redexpect.RedExpect.expect_timeout`** which can be set at first instance.

Returns str

Raises *redexpect.exceptions.ExpectTimeout* if the timeout for expect has been reached.

connect(*hostname*, *port=22*, *username=''*, *password=None*, *allow_agent=False*, *host_based=None*, *key_filepath=None*, *passphrase=None*, *look_for_keys=False*, *sock=None*, *timeout=None*)

> **Warning:** Do not use this function, instead use *redexpect.RedExpect.login()*.

device_init(***kwargs*)
Override this function to intialize a device that does not simply drop to the terminal or a device will kick you out if you send any key/character other than an "acceptable" one. This default one will work on linux quite well but devices such as pfsense or mikrotik might require this function and *redexpect.RedExpect.get_unique_prompt()* to be overriden.

dynamic_tunnel(*local_port*, *bind_addr='127.0.0.1'*, *error_level=TunnelErrorLevel.warn*)
Opens a SOCKS proxy AKA gateway or dynamic port the same way the `-D` option does for the OpenSSH client.

Providing a `0` for the local port will mean the OS will assign an unbound port for you. This port number will be provided to you by this function.

Parameters

- **local_port** (`int`) – The local port on the local machine to bind to.
- **bind_addr** (`str`) – The bind address on this machine to bind to for the local port.
- **error_level** (`redssh.enums.TunnelErrorLevel`) – The level of verbosity that errors in tunnel threads will use.

Returns `int` The local port that has been bound.

eof()
Returns `True` or `False` when the main channel has recieved an `EOF`.

execute_command(*command*)
Run a command. This will block as the command executes.

Parameters **command** (`str`) – Command to execute.

Returns `tuple (int, str)` - of `(return_code, command_output)`

exit()
Kill the current session if connected.

expect(*re_strings=''*, *default_match_prefix=''*, *strip_ansi=True*, *timeout=None*)
This function takes in a regular expression (or regular expressions) that represent the last line of output from the server. The function waits for one or more of the terms to be matched. The regexes are matched using expression `r'\n<regex>$'` so you'll need to provide an easygoing regex such as `'.*server.*'` if you wish to have a fuzzy match.

This has been originally taken from paramiko_expect and modified to work with RedExpect. I've also made the style consistent with the rest of the library.

Parameters

- **re_strings** (`array` or `regex str`) – Either a regex string or list of regex strings that we should expect; if this is not specified, then `EOF` is expected (i.e. the shell is completely closed after the exit command is issued)
- **default_match_prefix** (`str`) – A prefix to all match regexes, defaults to `''`. Useful for making sure you have a prefix to match the start of a prompt.
- **strip_ansi** (`bool`) – If `True`, will strip ansi control chars befores regex matching.

- **timeout** (float) – Set the timeout for this finish blocking within, setting to None takes the value from **:var:`redexpect.RedExpect.expect_timeout`** which can be set at first instance, set to 0 to disable.

Returns int - An EOF returns -1, a regex metch returns 0 and a match in a list of regexes returns the index of the matched string in the list.

Raises *redexpect.exceptions.ExpectTimeout* if the timeout for expect has been reached.

flush()
Flush all data on the primary channel's stdin to the remote connection. Only works if connected, otherwise returns 0.

Returns int - Amount of bytes sent to remote machine.

get_unique_prompt()
Return a unique prompt from the existing SSH session. Override this function to generate the compiled regex however you'd like, eg, from a database or from a hostname.

Returns compiled regex str

last_error()
Get the last error from the current session.

Returns str

local_tunnel(*local_port*, *remote_host*, *remote_port*, *bind_addr='127.0.0.1'*, *error_level=TunnelErrorLevel.warn*)
Forwards a port on the remote machine the same way the -L option does for the OpenSSH client.

Providing a 0 for the local port will mean the OS will assign an unbound port for you. This port number will be provided to you by this function.

Parameters

- **local_port** (int) – The local port on the local machine to bind to.
- **remote_host** (str) – The remote host to connect to via the remote machine.
- **remote_port** (int) – The remote host's port to connect to via the remote machine.
- **bind_addr** (str) – The bind address on this machine to bind to for the local port.
- **error_level** (redssh.enums.TunnelErrorLevel) – The level of verbosity that errors in tunnel threads will use.

Returns int The local port that has been bound.

login(**args*, *auto_unique_prompt=True*, ***kwargs*)
This uses redssh.RedSSH.connect to connect and then login to a remote host. This only takes a single optional arguement and the rest are defered to redssh.RedSSH.connect.

Parameters **auto_unique_prompt** (float) – Automatically set a unique prompt to search for once logged into the remote server.

methods(*method*)
Returns what value was settled on during session negotiation.

out_feed(*raw_data*)
Override to get the raw data from the remote machine into a function.

Useful as a way to get data from the expect() to another library without impacting the expect side.

prompt(*additional_matches=[]*, *timeout=None*)
Get a command line prompt in the terminal. Useful for using *redexpect.RedExpect.sendline()* to send commands then using this for when you want to get back to a prompt to enter further commands.

Parameters

- **additional_matches** (arr) – Additional matches to count as a prompt.
- **timeout** (float or int) – Timeout for the prompt to be reached.

Raises *redexpect.exceptions.ExpectTimeout* if the timeout for expect has been reached.

Returns int - Match number, 0 is always the prompt defined for the session.

read(*block=False*)
Recieve data from the remote session. Only works if the current session has made it past the login process.

Parameters **block** (bool) – Block until data is received from the remote server. True will block until data is recieved and False may return b'' if no data is available from the remote server.

Returns generator - A generator of byte strings that has been recieved in the time given.

remote_tunnel(*local_port*, *remote_host*, *remote_port*, *bind_addr='127.0.0.1'*, *error_level=TunnelErrorLevel.warn*)
Forwards a port to the remote machine via the local machine the same way the -R option does for the OpenSSH client.

Parameters

- **local_port** (int) – The local port on the remote side for clients to connect to.
- **remote_host** (str) – The remote host to connect to via the local machine.
- **remote_port** (int) – The remote host's port to connect to via the local machine.
- **error_level** (redssh.enums.TunnelErrorLevel) – The level of verbosity that errors in tunnel threads will use.

Returns None

send(*string*)
Send data to the remote session. Only works if the current session has made it past the login process.

Parameters **string** (str) – String to send to the remote session.

Returns int - Amount of bytes sent to remote machine.

sendline(*send_string*, *newline=None*)
Saves and sends the send string provided to the remote session with a newline added.

Parameters

- **send_string** (str) – String to send to the remote session.
- **newline** (str) – Override the newline character sent to the remote session.

sendline_raw(*string*)
Use this when you want to directly interact with the remote session.

Parameters **string** (str) – String to send to the remote session.

set_unique_prompt(*use_basic_prompt=True*, *set_prompt=False*)
Set a unique prompt in the existing SSH session.

Parameters

- **use_basic_prompt** (bool) – Use the dumb prompt from first login to the remote terminal.

- **set_prompt** (bool) – Set to True to set the prompt via :var:`redexpect.RedExpect.PROMPT_SET_SH`

setenv(*varname*, *value*)
Set an environment variable on the channel.

Parameters

- **varname** (str) – Name of environment variable to set on the remote channel.
- **value** (str) – Value to set varname to.

Returns None

shutdown_tunnel(*tunnel_type*, *sport*, *rhost=None*, *rport=None*, *bind_addr='127.0.0.1'*)
Closes an open tunnel. Provide the same arguments to this that was given for openning the tunnel.

Examples:

local_tunnel(9999,'localhost',8888) would be *shutdown_tunnel(redssh.enums.TunnelType.local,9999,'localhost',8888)*

remote_tunnel(7777,'localhost',8888) would be *shutdown_tunnel(redssh.enums.TunnelType.remote,7777,'localhost',8888)*

dynamic_tunnel(9999) would be *shutdown_tunnel(redssh.enums.TunnelType.dynamic,9999)*

dynamic_tunnel(9999,'10.0.0.1') would be *shutdown_tunnel(redssh.enums.TunnelType.dynamic,9999,bind_addr='10.0.0.1'*

Parameters

- **tunnel_type** (redssh.enums.TunnelType) – The tunnel type to shutdown.
- **sport** (str) – The bound port for local and dynamic tunnels or the local port on the remote side for remote tunnels.
- **rhost** (str) – The remote host for local and remote tunnels.
- **rport** (int) – The remote port for local and remote tunnels.
- **bind_addr** (str) – The bind address used for local and dynamic tunnels.

Returns None

start_scp()
Start the SCP client.

Returns None

start_sftp()
Start the SFTP client. The client will be available at *self.sftp* and will be an instance of *redssh.sftp.RedSFTP*

Returns None

sudo(*password*, *sudo=True*, *su_cmd='su -'*)
Sudo up or SU up or whatever up, into higher privileges.

Parameters

- **password** (str) – Password for gaining privileges
- **sudo** (bool) – Set to False to allow su_cmd to be executed instead.
- **su_cmd** (str) – Command to be executed when sudo is False, allows overriding of the 'sudo' default.

Returns None

Raises *redexpect.exceptions.BadSudoPassword* if the password provided does not allow for privilege escalation.

tunnel_is_alive(*tunnel_type*, *sport*, *rhost=None*, *rport=None*, *bind_addr='127.0.0.1'*)

Checks if a tunnel is alive. Provide the same arguments to this that was given for openning the tunnel.

Examples:

local_tunnel(9999,'localhost',8888) would be *tunnel_is_alive(redssh.enums.TunnelType.local,9999,'localhost',8888)*

remote_tunnel(7777,'localhost',8888) would be *tunnel_is_alive(redssh.enums.TunnelType.remote,7777,'localhost',8888)*

dynamic_tunnel(9999) would be *tunnel_is_alive(redssh.enums.TunnelType.dynamic,9999)*

dynamic_tunnel(9999,'10.0.0.1') would be *tunnel_is_alive(redssh.enums.TunnelType.dynamic,9999,bind_addr='10.0.0.1')*

Parameters

- **tunnel_type** (redssh.enums.TunnelType) – The tunnel type to shutdown.
- **sport** (str) – The bound port for local and dynamic tunnels or the local port on the remote side for remote tunnels.
- **rhost** (str) – The remote host for local and remote tunnels.
- **rport** (int) – The remote port for local and remote tunnels.
- **bind_addr** (str) – The bind address used for local and dynamic tunnels.

Returns bool, if bad tunnel type provided returns None

CHAPTER

TWO

REDEXPECT.EXCEPTIONS

exception redexpect.exceptions.**BadSudoPassword**
Bases: *redexpect.exceptions.RedExpectException*

This will be raised when a password does not acquire root via sudo/su.

with_traceback()
Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception redexpect.exceptions.**ExpectTimeout**(*wait_object*)
Bases: *redexpect.exceptions.RedExpectException*

This will be raised when searching for a certain string takes too long.

with_traceback()
Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception redexpect.exceptions.**RedExpectException**
Bases: Exception

Base error class for sub classing.

with_traceback()
Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

r

INDEX