Parallel-SSH Documentation

Release 1.9.1+1.gcf2b65e

P Kittenis

Dec 10, 2018

Contents

It uses non-blocking asynchronous SSH sessions and is to date the only publicly available non-blocking SSH client library, as well as the only non-blocking *parallel* SSH client library available for Python.

CHAPTER 1

Design And Goals

`parallel-ssh`'s design goals and motivation are to provide a *library* for running *asynchronous* SSH commands in parallel with little to no load induced on the system by doing so with the intended usage being completely programmatic and non-interactive.

To meet these goals, API driven solutions are preferred first and foremost. This frees up the developer to drive the library via any method desired, be that environment variables, CI driven tasks, command line tools, existing OpenSSH or new configuration files, from within an application et al.

1.1 Design Principles

Taking a cue from PEP 20, heavy emphasis is in the following areas.

- Readability
- Explicit is better than implicit
- Simple is better than complex
- Beautiful is better than ugly

Contributions are asked to keep these in mind.

CHAPTER 2

Installation

Installation is handled by Python's standard `setuptools` library and `pip`.

2.1 Pip Install

`pip` may need to be updated to be able to install binary wheel packages.

```
pip install -U pip

pip install parallel-ssh
```

If `pip` is not available on your Python platform, see this installation guide.

2.2 Dependencies

When installing from source, dependencies must be satisfied by `pip install -r requirements.txt`. For pre-built binary wheel packages with dependencies included, see *Pip Install*.

From `2.0.0` onwards, paramiko will become an _optional_ extra, with `libssh2`, and for a limited set of functionality, `libssh` native library based clients via `ssh2-python` and `ssh-python` bindings respectively replacing it.

Dependency	Minimum Version
`ssh2-python`	`0.16.0`
`gevent`	`1.1`
`paramiko`	`1.15.3`

2.3 Building from Source

`parallel-ssh` is hosted on GitHub and the repository can be cloned with the following

```
git clone git@github.com:ParallelSSH/parallel-ssh.git
cd parallel-ssh
```

To install from source run:

```
python setup.py install
```

Or with `pip`'s development mode which will ensure local changes are made available:

```
pip install -e .
```

2.4 Building System Packages

For convenience, a script making use of Docker is provided at ci/docker/build-packages.sh that will build system packages for Centos/RedHat 6/7, Ubuntu 14.04/16.04, Debian 7/8 and Fedora 22/23/24.

Note that these packages make use of system libraries that may need to be updated to be compatible with `parallel-ssh` - see *Dependencies*.

```
git clone git@github.com:ParallelSSH/parallel-ssh.git
cd parallel-ssh
# Checkout a tag for tagged builds - git tag; git checkout <tag>
./ci/docker/build-packages.sh
ls -1tr
```

```
python-parallel-ssh-1.2.0+4.ga811e69.dirty-1.el6.x86_64.rpm
python-parallel-ssh-1.2.0+4.ga811e69.dirty-1.el7.x86_64.rpm
python-parallel-ssh-1.2.0+4.ga811e69.dirty-1.fc22.x86_64.rpm
python-parallel-ssh-1.2.0+4.ga811e69.dirty-1.fc23.x86_64.rpm
python-parallel-ssh-1.2.0+4.ga811e69.dirty-1.fc24.x86_64.rpm
python-parallel-ssh_1.2.0+4.ga811e69.dirty-debian7_amd64.deb
python-parallel-ssh_1.2.0+4.ga811e69.dirty-debian8_amd64.deb
```

2.4.1 Specific System Package Build

To build for only a specific system/distribution, run the two following commands, substituting distribution with the desired one from ci/docker. See existing Dockerfiles for examples on how to create system packages for other distributions.

Debian based

```
docker build --cache-from parallelssh/parallel-ssh-pkgs:debian7 ci/docker/debian7 -t␣
↪debian7
docker run -v "$(pwd):/src/" debian7 --iteration debian7 -s python -t deb setup.py
```

RPM based

```
docker build --cache-from parallelssh/parallel-ssh-pkgs:centos7 ci/docker/centos7 -t␣
↪centos7
docker run -v "$(pwd):/src/" centos7 --rpm-dist el7 -s python -t rpm setup.py
```

See fpm for making system packages of various types.

2.5 Deprecated Python Versions

`1.1.x` and above releases are not compatible with Python `2.6`.

If you are running a deprecated Python version such as `2.6` you may need to install an older version of `parallel-ssh` that is compatible with that Python platform.

For example, to install the `1.0.0` version, run the following.

```
pip install parallel-ssh==1.0.0
```

`1.0.0` is compatible with all Python versions over or equal to `2.6`, including all of the `3.x` series.

Older versions such as *0.70.x* are compatible with Python `2.5` and `2.x` but not the `3.x` series.

CHAPTER 3

Quickstart

First, make sure that `parallel-ssh` is installed.

Note: When using the paramiko based clients, `parallel-ssh` makes use of gevent's monkey patching to enable asynchronous use of the Python standard library's network I/O as paramiko does not and cannot natively support non-blocking mode.

Monkey patching is only done for the clients under `pssh.clients.miko` and the deprecated imports `pssh.pssh_client` and `pssh.ssh_client`.

Make sure that these imports come **before** any other imports in your code in this case. Otherwise, patching may not be done before the standard library is loaded which will then cause the (g)event loop to be blocked.

If you are seeing messages like `This operation would block forever`, this is the cause.

Native clients under `pssh.clients.native` **do not perform monkey patching** and are an option if monkey patching is not suitable. These clients will become the default, replacing the current `pssh.pssh_client`, in a future major release - `2.0.0`.

3.1 Run a command on hosts in parallel

The most basic usage of `parallel-ssh` is, unsurprisingly, to run a command on multiple hosts in parallel.

Examples in this documentation will be using `print` as a function, for which a future import is needed in Python `2.7` and below.

Make a list or other iterable of the hosts to run on:

```
from __future__ import print_function
from pssh.clients import ParallelSSHClient

hosts = ['host1', 'host2', 'host3', 'host4']
```

Where `host1` to `host4` are valid host names. IP addresses may also be used.

Create a client for these hosts:

```
client = ParallelSSHClient(hosts)
```

The client object can, and should, be reused. Existing connections to hosts will remain alive as long as the client object is kept alive. Subsequent commands to the same host(s) will reuse their existing connection and benefit from much faster response times.

Now one or more commands can be run via the client:

```
output = client.run_command('whoami')
```

When the call to `run_command` returns, the commands are already executing in parallel.

Output is keyed by host name and contains a host output object. From that, SSH output is available.

Note: Multiple identical hosts will have their output key de-duplicated so that their output is not lost. The real host name used is available as `host_output.host` where `host_output` is a *`pssh.output.HostOutput`* object.

3.2 Standard Output

Standard output, aka `stdout` for `host1`:

```
for line in output['host1'].stdout:
    print(line)
```

Output

```
<your username here>
```

There is nothing special needed to ensure output is available.

Please note that retrieving all of a command's standard output by definition requires that the command has completed.

Iterating over `stdout` for any host *to completion* will therefor *only complete* when that host's command has completed unless interrupted.

The `timeout` keyword argument to `run_command` may be used to cause output generators to timeout if no output is received after the given number of seconds - see join and output timeouts (native clients only).

`stdout` is a generator. Iterating over it will consume the remote standard output stream via the network as it becomes available. To retrieve all of stdout can wrap it with list, per below.

```
stdout = list(output['host1'].stdout)
```

Warning: This will store the entirety of stdout into memory and may exhaust available memory if command output is large enough.

3.2.1 All hosts iteration

Of course, iterating over all hosts can also be done the same way.

```
for host, host_output in output.items():
    for line in host_output.stdout:
        print("Host [%s] - %s" % (host, line))
```

3.3 Exit codes

Exit codes are available on the host output object.

First, ensure that all commands have finished and exit codes gathered by joining on the output object, then iterate over all host's output to print their exit codes.

```
client.join(output)
for host, host_output in output.items():
    print("Host %s exit code: %s" % (host, host_output.exit_code))
```

See also:

pssh.output.HostOutput Host output class documentation.

3.4 Authentication

By default `parallel-ssh` will use an available SSH agent's credentials to login to hosts via public key authentication.

3.4.1 User/Password authentication

User/password authentication can be used by providing user name and password credentials:

```
client = ParallelSSHClient(hosts, user='my_user', password='my_pass')
```

Note: On Posix platforms, user name defaults to the current user if not provided.

On Windows, user name is required.

3.4.2 Programmatic Private Key authentication

It is also possible to programmatically provide a private key for authentication.

Native Client

For the native client - `pssh.clients.ParallelSSHClient` - only private key filepath is needed. The corresponding public key *must* be available in the same directory as `my_pkey.pub` where private key file is `my_pkey`. Public key file name and path will be made configurable in a future version.

```
from pssh.clients import ParallelSSHClient

client = ParallelSSHClient(hosts, pkey='my_pkey')
```

Paramiko Client

For the paramiko based client **only**, the helper function *load_private_key* is provided to easily load all possible key types. It takes either a file path or a file-like object.

File path

```
from pssh.clients.miko import ParallelSSHClient
from pssh.utils import load_private_key

pkey = load_private_key('my_pkey.pem')
client = ParallelSSHClient(hosts, pkey=pkey)
```

Note: The two available clients support different key types and authentication mechanisms - see Paramiko and libssh2 documentation for details, as well as clients features comparison.

3.5 Output for Last Executed Commands

Output for last executed commands can be retrieved by `get_last_output`:

```
client.run_command('uname')
output = client.get_last_output()
for host, host_output in output.items():
    for line in host.stdout:
        print(line)
```

This function can also be used to retrieve output for previously executed commands in the case where output object was not stored or is no longer available.

New in 1.2.0

3.5.1 Retrieving Last Executed Commands

Commands last executed by `run_command` can also be retrieved from the `cmds` attribute of `ParallelSSHClient`:

```
client.run_command('uname')
output = {}
for i, host in enumerate(hosts):
    cmd = self.cmds[i]
    client.get_output(cmd, output)
    print("Got output for host %s from cmd %s" % (host, cmd))
```

New in 1.2.0

3.6 Host Logger

There is a built in host logger that can be enabled to automatically log output from remote hosts. This requires the `consume_output` flag to be enabled on *join*.

The helper function `pssh.utils.enable_host_logger` will enable host logging to standard output, for example:

```
from pssh.utils import enable_host_logger
enable_host_logger()

output = client.run_command('uname')
client.join(output, consume_output=True)
```

Output

```
[localhost]       Linux
```

3.7 Using standard input

Along with standard output and error, input is also available on the host output object. It can be used to send input to the remote host where required, for example password prompts or any other prompt requiring user input.

The `stdin` attribute is a file-like object giving access to the remote stdin channel that can be written to:

```
output = client.run_command('read')
stdin = output['localhost'].stdin
stdin.write("writing to stdin\\n")
stdin.flush()
for line in output['localhost'].stdout:
    print(line)
```

Output

```
writing to stdin
```

3.8 Errors and Exceptions

By default, `parallel-ssh` will fail early on any errors connecting to hosts, whether that be connection errors such as DNS resolution failure or unreachable host, SSH authentication failures or any other errors.

Alternatively, the `stop_on_errors` flag is provided to tell the client to go ahead and attempt the command(s) anyway and return output for all hosts, including the exception on any hosts that failed:

```
output = client.run_command('whoami', stop_on_errors=False)
```

With this flag, the `exception` output attribute will contain the exception on any failed hosts, or `None`:

```
client.join(output)
for host, host_output in output.items():
    print("Host %s: exit code %s, exception %s" % (
          host, host_output.exit_code, host_output.exception))
```

Output

```
host1: 0, None
host2: None, AuthenticationException <..>
```

See also:

Exceptions raised by the library can be found in the *pssh.exceptions* module.

CHAPTER 4

Clients Feature Comparison

For the `ssh2-python` (`libssh2`) based clients, not all features supported by the paramiko based clients are currently supported by the underlying library or implemented in `parallel-ssh`.

Below is a comparison of feature support for the two client types.

If any of missing features are required for a use case, then the paramiko based clients should be used instead. Note there are several breaking bugs and low performance in some paramiko functionality, mileage may vary.

In all other cases the `ssh2-python` based clients offer significantly greater performance at less overhead and are preferred.

CHAPTER 5

Advanced Usage

There are several more advanced usage features of `parallel-ssh`, such as tunnelling (aka proxying) via an intermediate SSH server and per-host configuration and command substitution among others.

5.1 Agents and Private Keys

5.1.1 SSH Agent forwarding

SSH agent forwarding, what `ssh -A` does on the command line, is supported and enabled by default. Creating a client object as:

```
ParallelSSHClient(hosts, forward_ssh_agent=False)
```

will disable this behaviour.

5.1.2 Programmatic Private Keys

By default, `parallel-ssh` will use all keys in an available SSH agent and identity keys under the user's SSH directory - `id_rsa`, `id_dsa` and `identity` in `~/.ssh`.

A private key can also be provided programmatically.

```
from pssh.utils import load_private_key
from pssh.pssh_client import ParallelSSHClient

client = ParallelSSHClient(hosts, pkey=load_private_key('my_key'))
```

Where `my_key` is a private key file in current working directory.

The helper function *`load_private_key`* will attempt to load all available key types and raises *`SSHException`* if it cannot load the key file.

See also:

load_private_key

5.1.3 Disabling use of system SSH Agent

Use of an available SSH agent can also be disabled.

```
client = ParallelSSHClient(hosts, pkey=load_private_key('my_key'),
                           allow_agent=False)
```

Warning: For large number of hosts, it is recommended that private keys are provided programmatically and use of SSH agent is disabled via `allow_agent=False` as above.

If the number of hosts is large enough, available connections to the system SSH agent may be exhausted which will stop the client from working on a subset of hosts.

This is a limitation of the underlying SSH client used by `parallel-ssh`.

5.1.4 Programmatic SSH Agent

Paramiko client only.

It is also possible to programmatically provide an SSH agent for the client to use, instead of a system provided one. This is useful in cases where hosts need different private keys and a system SSH agent is not available.

```
from pssh.agent import SSHAgent
from pssh.utils import load_private_key
from pssh.clients.miko import ParallelSSHClient

agent = SSHAgent()
agent.add_key(load_private_key('my_private_key_filename'))
agent.add_key(load_private_key('my_other_private_key_filename'))
hosts = ['my_host', 'my_other_host']

client = ParallelSSHClient(hosts, agent=agent)
client.run_command(<..>)
```

Note: Supplying an agent programmatically implies that a system SSH agent will *not* be used even if available.

See also:

pssh.agent.SSHAgent

5.2 Native clients

Starting from version `1.2.0`, a new client is supported in `parallel-ssh` which offers much greater performance and reduced overhead than the current default client.

The new client is based on `libssh2` via the `ssh2-python` extension library and supports non-blocking mode natively. Binary wheel packages with `libssh2` included are provided for Linux, OSX and Windows platforms and all supported Python versions.

See this post for a performance comparison of the available clients.

To make use of this new client, `ParallelSSHClient` can be imported from `pssh.clients.native` instead. Their respective APIs are almost identical.

```
from pssh.clients.native import ParallelSSHClient

hosts = ['my_host', 'my_other_host']
client = ParallelSSHClient(hosts)
client.run_command(<..>)
```

See also:

Feature comparison for how the client features compare.

API documentation for parallel and single native clients.

5.3 Tunneling

This is used in cases where the client does not have direct access to the target host and has to authenticate via an intermediary, also called a bastion host, commonly used for additional security as only the bastion host needs to have access to the target host.

ParallelSSHClient ——> Proxy host ——> Target host

Proxy host can be configured as follows in the simplest case:

```
hosts = [<..>]
client = ParallelSSHClient(hosts, proxy_host='bastion')
```

Configuration for the proxy host's user name, port, password and private key can also be provided, separate from target host user name.

```
from pssh.utils import load_private_key

hosts = [<..>]
client = ParallelSSHClient(hosts, user='target_host_user',
                           proxy_host='bastion', proxy_user='my_proxy_user',
                           proxy_port=2222,
                           proxy_pkey=load_private_key('proxy.key'))
```

Where `proxy.key` is a filename containing private key to use for proxy host authentication.

In the above example, connections to the target hosts are made via SSH through `my_proxy_user@bastion:2222 -> target_host_user@<host>`.

Note: Proxy host connections are asynchronous and use the SSH protocol's native TCP tunneling - aka local port forward. No external commands or processes are used for the proxy connection, unlike the *ProxyCommand* directive in OpenSSH and other utilities.

While connections initiated by `parallel-ssh` are asynchronous, connections from proxy host -> target hosts may not be, depending on SSH server implementation. If only one proxy host is used to connect to a large number of target hosts and proxy SSH server connections are *not* asynchronous, this may adversely impact performance on the proxy host.

5.4 Join and Output Timeouts

New in 1.5.0

The native clients have timeout functionality on reading output and `client.join`.

```
from pssh.exceptions import Timeout

output = client.run_command(..)
try:
    client.join(output, timeout=5)
except Timeout:
    pass
```

```
output = client.run_command(.., timeout=5)
for host, host_out in output.items():
    try:
        for line in host_out.stdout:
            pass
        for line in host_out.stderr:
            pass
    except Timeout:
        pass
```

The client will raise a `Timeout` exception if remote commands have not finished within five seconds in the above examples.

5.4.1 Reading Partial Output of Commands That Do Not Terminate

In some cases, such as when the remote command never terminates unless interrupted, it is necessary to use PTY and to close the channel to force the process to be terminated before a `join` sans timeout can complete. For example:

```
output = client.run_command('tail -f /var/log/messages', use_pty=True, timeout=1)

# Read as many lines of output as server has sent before the timeout
stdout = []
for host, host_out in output.items():
    for host, host_out in output.items():
        try:
            for line in host_out.stdout:
                stdout.append(line)
        except Timeout:
            pass

# Closing channel which has PTY has the effect of terminating
# any running processes started on that channel.
for host, host_out in output.items():
    client.host_clients[host].close_channel(host_out.channel)
# Join is not strictly needed here as channel has already been closed and
# command has finished, but is safe to use regardless.
client.join(output)
```

Without a PTY, a `join` call with a timeout will complete with timeout exception raised but the remote process will be left running as per SSH protocol specifications.

Furthermore, once reading output has timed out, it is necessary to restart the output generators as by Python design they only iterate once. This can be done as follows:

```
output = client.run_command(<..>, timeout=1)
for host, host_out in output.items():
    try:
        stdout = list(host_out.stdout)
    except Timeout:
        client.reset_output_generators(host_out)
```

Generator reset shown above is also performed automatically by calls to `join` and does not need to be done manually when `join` is used after output reading.

Note: `join` with a timeout forces output to be consumed as otherwise the pending output will keep the channel open and make it appear as if command has not yet finished.

To capture output when using `join` with a timeout, gather output first before calling `join`, making use of output timeout as well, and/or make use of *Host Logger* functionality.

Warning: Beware of race conditions when using timeout functionality. For best results, only send one command per call to `run_command` when using timeout functionality.

As the timeouts are performed on `select` calls on the socket which is responsible for all client <-> server communication, whether or not a timeout will occur depends on what the socket is doing at that time.

Multiple commands like `run_command('echo blah; sleep 5')` where `sleep 5` is a placeholder for something taking five seconds to complete will result in a race condition as the second command may or may not have started by the time `join` is called or output is read which will cause timeout to *not* be raised even if the second command has not started or completed.

It is responsibility of developer to avoid these race conditions such as by only sending one command in such cases.

5.5 Per-Host Configuration

Sometimes, different hosts require different configuration like user names and passwords, ports and private keys. Capability is provided to supply per host configuration for such cases.

```
from pssh.utils import load_private_key

host_config = {'host1' : {'user': 'user1', 'password': 'pass',
                          'port': 2222,
                          'private_key': load_private_key(
                              'my_key.pem')},
               'host2' : {'user': 'user2', 'password': 'pass',
                          'port': 2223,
                          'private_key': load_private_key(
                              open('my_other_key.pem'))},
               }
hosts = host_config.keys()

client = ParallelSSHClient(hosts, host_config=host_config)
client.run_command('uname')
<..>
```

In the above example, `host1` will use user name `user1` and private key from `my_key.pem` and `host2` will use user name `user2` and private key from `my_other_key.pem`.

Note: Proxy host cannot be provided via per-host configuration at this time.

5.6 Per-Host Command substitution

For cases where different commands should be run on each host, or the same command with different arguments, functionality exists to provide per-host command arguments for substitution.

The `host_args` keyword parameter to `run_command` can be used to provide arguments to use to format the command string.

Number of `host_args` items should be at least as many as number of hosts.

Any Python string format specification characters may be used in command string.

In the following example, first host in hosts list will use cmd `host1_cmd` second host `host2_cmd` and so on

```
output = client.run_command('%s', host_args=('host1_cmd',
                                              'host2_cmd',
                                              'host3_cmd',))
```

Command can also have multiple arguments to be substituted.

```
output = client.run_command('%s %s',
host_args = (('host1_cmd1', 'host1_cmd2'),
             ('host2_cmd1', 'host2_cmd2'),
             ('host3_cmd1', 'host3_cmd2'),))
```

A list of dictionaries can also be used as `host_args` for named argument substitution.

In the following example, first host in host list will use cmd `host-index-0`, second host `host-index-1` and so on.

```
host_args = [{'cmd': 'host-index-%s' % (i,)}
             for i in range(len(client.hosts))]
output = client.run_command('%(cmd)s', host_args=host_args)
```

5.7 Run command features and options

See `run_command API documentation` for a complete list of features and options.

Note: With a PTY, the default, stdout and stderr output is combined into stdout.

Without a PTY, separate output is given for stdout and stderr, although some programs and server configurations require a PTY.

5.7.1 Run with sudo

`parallel-ssh` can be instructed to run its commands under `sudo`:

```
client = <..>

output = client.run_command(<..>, sudo=True)
client.join(output)
```

While not best practice and password-less `sudo` is best configured for a limited set of commands, a sudo password may be provided via the stdin channel:

```
client = <..>

output = client.run_command(<..>, sudo=True)
for host in output:
    stdin = output[host].stdin
    stdin.write('my_password\n')
    stdin.flush()
client.join(output)
```

5.7.2 Output encoding

By default, output is encoded as `UTF-8`. This can be configured with the `encoding` keyword argument.

```
client = <..>

client.run_command(<..>, encoding='utf-16')
stdout = list(output[client.hosts[0]].stdout)
```

Contents of `stdout` will be *UTF-16* encoded.

Note: Encoding must be valid Python codec

5.7.3 Disabling use of pseudo terminal emulation

For cases where use of a *PTY* is not wanted, such as having separate stdout and stderr outputs, the remote command is a daemon that needs to fork and detach itself or when use of a shell is explicitly disabled, use of PTY can also be disabled.

The following example prints to stderr with PTY disabled.

```
from __future__ import print_function

client = <..>

client.run_command("echo 'asdf' >&2", use_pty=False)
for line in output[client.hosts[0]].stderr:
    print(line)
```

Output

```
asdf
```

5.7.4 Combined stdout/stderr

With a PTY on the paramiko client, stdout and stderr output is combined.

The same example as above with a PTY:

```
from __future__ import print_function

client = <..>

client.run_command("echo 'asdf' >&2")
for line in output[client.hosts[0]].stdout:
    print(line)
```

Note output is now from the `stdout` channel.

Output

```
asdf
```

Stderr is empty:

```
for line in output[client.hosts[0]].stderr:
    print(line)
```

No output from `stderr`.

5.8 SFTP

SFTP - *SCP version 2* - is supported by `parallel-ssh` and two functions are provided by the client for copying files with SFTP.

SFTP does not have a shell interface and no output is provided for any SFTP commands.

As such, SFTP functions in `ParallelSSHClient` return greenlets that will need to be joined to raise any exceptions from them. `gevent.joinall()` may be used for that.

5.8.1 Copying files to remote hosts in parallel

To copy the local file with relative path `../test` to the remote relative path `test_dir/test` - remote directory will be created if it does not exist, permissions allowing. `raise_error=True` instructs `joinall` to raise any exceptions thrown by the greenlets.

```
from pssh.pssh_client import ParallelSSHClient
from gevent import joinall

client = ParallelSSHClient(hosts)

greenlets = client.copy_file('../test', 'test_dir/test')
joinall(greenlets, raise_error=True)
```

To recursively copy directory structures, enable the `recurse` flag:

```
greenlets = client.copy_file('my_dir', 'my_dir', recurse=True)
joinall(greenlets, raise_error=True)
```

See also:

`copy_file` API documentation and exceptions raised.

`gevent.joinall()` Gevent's `joinall` API documentation.

5.8.2 Copying files from remote hosts in parallel

Copying remote files in parallel requires that file names are de-duplicated otherwise they will overwrite each other. `copy_remote_file` names local files as `<local_file><suffix_separator><host>`, suffixing each file with the host name it came from, separated by a configurable character or string.

```
from pssh.pssh_client import ParallelSSHClient
from gevent import joinall

client = ParallelSSHClient(hosts)

greenlets = client.copy_remote_file('remote.file', 'local.file')
joinall(greenlets, raise_error=True)
```

The above will create files `local.file_host1` where `host1` is the host name the file was copied from.

See also:

`copy_remote_file` API documentation and exceptions raised.

5.8.3 Single host copy

If wanting to copy a file from a single remote host and retain the original filename, can use the single host `SSHClient` and its `copy_file` directly.

```
from pssh.pssh_client import SSHClient

client = SSHClient('localhost')
client.copy_remote_file('remote_filename', 'local_filename')
```

See also:

`SSHClient.copy_remote_file` API documentation and exceptions raised.

5.9 Hosts filtering and overriding

5.9.1 Iterators and filtering

Any type of iterator may be used as hosts list, including generator and list comprehension expressions.

List comprehension

```
hosts = ['dc1.myhost1', 'dc2.myhost2']
client = ParallelSSHClient([h for h in hosts if h.find('dc1')])
```

Generator

```
hosts = ['dc1.myhost1', 'dc2.myhost2']
client = ParallelSSHClient((h for h in hosts if h.find('dc1')))
```

Filter

```
hosts = ['dc1.myhost1', 'dc2.myhost2']
client = ParallelSSHClient(filter(lambda h: h.find('dc1'), hosts))
client.run_command(<..>)
```

Note: Since generators by design only iterate over a sequence once then stop, `client.hosts` should be re-assigned after each call to `run_command` when using generators as target of `client.hosts`.

5.9.2 Overriding hosts list

Hosts list can be modified in place. A call to `run_command` will create new connections as necessary and output will only contain output for the hosts `run_command` executed on.

```
client = <..>

client.hosts = ['otherhost']
print(client.run_command('exit 0'))
{'otherhost': exit_code=None, <..>}
```

5.10 Additional options for underlying SSH libraries

Not all SSH library configuration options are used directly by `parallel-ssh`.

Additional options can be passed on to the underlying SSH libraries used via an optional keyword argument.

Please note that the underlying SSH libraries used are subject to change and not all features are present in all SSH libraries used. Future releases will have more than one option on which SSH library to use, depending on user requirements and preference.

New in version 1.1.

5.10.1 Paramiko (current default SSH library)

GSS-API Authentication - aka Kerberos

```
client = ParallelSSHClient(hosts)

client.run_command('id', gss_auth=True, gss_kex=True, gss_host='my_gss_host')
```

In this example, `gss_auth`, `gss_kex` and `gss_host` are keyword arguments passed on to paramiko.client.SSHClient.connect to instruct the client to enable GSS-API authentication and key exchange with the provided GSS host.

Note: The GSS-API features of Paramiko require that the `python-gssapi` package be installed manually - it is optional and not installed by any *extras* option of Paramiko.

```
pip install python-gssapi
```

Compression

Any other options not directly referenced by `run_command` can be passed on to paramiko.client.SSHClient.connect, for example the `compress` option.

```
client = ParallelSSHClient(hosts)

client.run_command('id', compress=True)
```

CHAPTER 6

API Documentation

6.1 Native Parallel Client

API documentation for the `ssh2-python` (`libssh2`) based parallel client.

class pssh.clients.native.parallel.**ParallelSSHClient**(*hosts*, *user=None*, *password=None*, *port=22*, *pkey=None*, *num_retries=3*, *timeout=None*, *pool_size=10*, *allow_agent=True*, *host_config=None*, *retry_delay=5*, *proxy_host=None*, *proxy_port=22*, *proxy_user=None*, *proxy_password=None*, *proxy_pkey=None*, *forward_ssh_agent=False*, *tunnel_timeout=None*, *keepalive_seconds=60*)

ssh2-python based parallel client.

Parameters

- **hosts** (*list(str)*) – Hosts to connect to
- **user** (*str*) – (Optional) User to login as. Defaults to logged in user
- **password** (*str*) – (Optional) Password to use for login. Defaults to no password
- **port** (*int*) – (Optional) Port number to use for SSH connection. Defaults to 22.
- **pkey** (*str*) – Private key file path to use. Path must be either absolute path or relative to user home directory like `~/<path>`.

- **num_retries** (*int*) – (Optional) Number of connection and authentication attempts before the client gives up. Defaults to 3.
- **retry_delay** (*int*) – Number of seconds to wait between retries. Defaults to `pssh.constants.RETRY_DELAY`
- **timeout** (*float*) – (Optional) SSH session timeout setting in seconds. This controls timeout setting of socket operations used for SSH sessions. Defaults to OS default - usually 60 seconds.
- **pool_size** (*int*) – (Optional) Greenlet pool size. Controls concurrency, on how many hosts to execute tasks in parallel. Defaults to 10. Overhead in event loop will determine how high this can be set to, see scaling guide lines in project's readme.
- **host_config** (*dict*) – (Optional) Per-host configuration for cases where not all hosts use the same configuration.
- **allow_agent** (*bool*) – (Optional) set to False to disable connecting to the system's SSH agent.
- **proxy_host** (*str*) – (Optional) SSH host to tunnel connection through so that SSH clients connect to host via client -> proxy_host -> host
- **proxy_port** (*int*) – (Optional) SSH port to use to login to proxy host if set. Defaults to 22.
- **proxy_user** (*str*) – (Optional) User to login to `proxy_host` as. Defaults to logged in user.
- **proxy_password** (*str*) – (Optional) Password to login to `proxy_host` with. Defaults to no password.
- **proxy_pkey** (*Private key file path to use.*) – (Optional) Private key file to be used for authentication with `proxy_host`. Defaults to available keys from SSHAgent and user's SSH identities.
- **forward_ssh_agent** (*bool*) – (Optional) Turn on SSH agent forwarding - equivalent to *ssh -A* from the *ssh* command line utility. Defaults to False if not set. Requires agent forwarding implementation in libssh2 version used.
- **tunnel_timeout** (*float*) – (Optional) Timeout setting for proxy tunnel connections.

Raises *pssh.exceptions.PKeyFileError* on errors finding provided private key.

copy_file(*local_file*, *remote_file*, *recurse=False*, *copy_args=None*)

Copy local file to remote file in parallel via SFTP.

This function returns a list of greenlets which can be *join*-ed on to wait for completion.

`gevent.joinall()` function may be used to join on all greenlets and will also raise exceptions from them if called with `raise_error=True` - default is *False*.

Alternatively call *.get()* on each greenlet to raise any exceptions from it.

Exceptions listed here are raised when either `gevent.joinall(<greenlets>, raise_error=True)` or `.get()` on each greenlet are called, not this function itself.

Parameters

- **local_file** (*str*) – Local filepath to copy to remote host
- **remote_file** (*str*) – Remote filepath on remote host to copy file to
- **recurse** (*bool*) – Whether or not to descend into directories recursively.

- **copy_args** (*tuple or list*) – (Optional) format local_file and remote_file strings with per-host arguments in `copy_args`. `copy_args` length must equal length of host list - *pssh.exceptions.HostArgumentException* is raised otherwise

Return type list(`gevent.Greenlet`) of greenlets for remote copy commands

Raises `ValueError` when a directory is supplied to local_file and recurse is not set

Raises *pssh.exceptions.HostArgumentException* on number of per-host copy arguments not equal to number of hosts

Raises `pss.exceptions.SFTPError` on SFTP initialisation errors

Raises *pssh.exceptions.SFTPIOError* on I/O errors writing via SFTP

Raises `OSError` on local OS errors like permission denied

Note: Remote directories in `remote_file` that do not exist will be created as long as permissions allow.

---

copy_remote_file(*remote_file*, *local_file*, *recurse=False*, *suffix_separator='_'*, *copy_args=None*, *encoding='utf-8'*)

Copy remote file(s) in parallel via SFTP as <local_file><suffix_separator><host>

With a `local_file` value of `myfile` and default separator `_` the resulting filename will be `myfile_myhost` for the file from host `myhost`.

This function, like *ParallelSSHClient.copy_file()*, returns a list of greenlets which can be *join*-ed on to wait for completion.

`gevent.joinall()` function may be used to join on all greenlets and will also raise exceptions if called with `raise_error=True` - default is *False*.

Alternatively call *.get* on each greenlet to raise any exceptions from it.

Exceptions listed here are raised when either `gevent.joinall(<greenlets>, raise_error=True)` is called or `.get` is called on each greenlet, not this function itself.

Parameters

- **remote_file** (*str*) – remote filepath to copy to local host
- **local_file** (*str*) – local filepath on local host to copy file to
- **recurse** (*bool*) – whether or not to recurse
- **suffix_separator** (*str*) – (Optional) Separator string between filename and host, defaults to _. For example, for a `local_file` value of `myfile` and default separator the resulting filename will be `myfile_myhost` for the file from host `myhost`. `suffix_separator` has no meaning if `copy_args` is provided
- **copy_args** (*tuple or list*) – (Optional) format remote_file and local_file strings with per-host arguments in `copy_args`. `copy_args` length must equal length of host list - *pssh.exceptions.HostArgumentException* is raised otherwise
- **encoding** (*str*) – Encoding to use for file paths.

Return type list(`gevent.Greenlet`) of greenlets for remote copy commands

Raises `ValueError` when a directory is supplied to local_file and recurse is not set

Raises *pssh.exceptions.HostArgumentException* on number of per-host copy arguments not equal to number of hosts

Raises `pss.exceptions.SFTPError` on SFTP initialisation errors

Raises *pssh.exceptions.SFTPIOError* on I/O errors reading from SFTP

Raises `OSError` on local OS errors like permission denied

Note: Local directories in *local_file* that do not exist will be created as long as permissions allow.

Note: File names will be de-duplicated by appending the hostname to the filepath separated by `suffix_separator`.

---

join(*output*, *consume_output=False*, *timeout=None*)

Wait until all remote commands in output have finished and retrieve exit codes. Does *not* block other commands from running in parallel.

Parameters

- **output** (dict as returned by `pssh.pssh_client.ParallelSSHClient.get_output()`) – Output of commands to join on
- **consume_output** (*bool*) – Whether or not join should consume output buffers. Output buffers will be empty after `join` if set to `True`. Must be set to `True` to allow host logger to log output on call to `join` when host logger has been enabled.
- **timeout** (*int*) – Timeout in seconds if remote command is not yet finished. Note that use of timeout forces `consume_output=True` otherwise the channel output pending to be consumed always results in the channel not being finished.

Raises *pssh.exceptions.Timeout* on timeout requested and reached with commands still running.

Return type `None`

reset_output_generators(*host_out*, *timeout=None*, *client=None*, *channel=None*, *encoding='utf-8'*)

Reset output generators for host output.

Parameters

- **host_out** (*pssh.output.HostOutput*) – Host output
- **client** (`pssh.ssh2_client.SSHClient`) – (Optional) SSH client
- **channel** (`ssh2.channel.Channel`) – (Optional) SSH channel
- **timeout** (*int*) – (Optional) Timeout setting
- **encoding** (*str*) – (Optional) Encoding to use for output. Must be valid Python codec

Return type tuple(stdout, stderr)

run_command(*command*, *sudo=False*, *user=None*, *stop_on_errors=True*, *use_pty=False*, *host_args=None*, *shell=None*, *encoding='utf-8'*, *timeout=None*, *greenlet_timeout=None*)

Run command on all hosts in parallel, honoring self.pool_size, and return output dictionary.

This function will block until all commands have been received by remote servers and then return immediately.

More explicitly, function will return after connection and authentication establishment and after commands have been accepted by successfully established SSH channels.

Any connection and/or authentication exceptions will be raised here and need catching *unless* `run_command` is called with `stop_on_errors=False` in which case exceptions are added to individual host output instead.

Parameters

- **command** (*str*) – Command to run
- **sudo** (*bool*) – (Optional) Run with sudo. Defaults to False
- **user** (*str*) – (Optional) User to run command as. Requires sudo access for that user from the logged in user account.
- **stop_on_errors** (*bool*) – (Optional) Raise exception on errors running command. Defaults to True. With stop_on_errors set to False, exceptions are instead added to output of *run_command*. See example usage below.
- **shell** (*str*) – (Optional) Override shell to use to run command with. Defaults to login user's defined shell. Use the shell's command syntax, eg *shell='bash -c'* or *shell='zsh -c'*.
- **use_pty** (*bool*) – (Optional) Enable/Disable use of pseudo terminal emulation. Disabling it will prohibit capturing standard input/output. This is required in majority of cases, exceptions being where a shell is not used and/or input/output is not required. In particular when running a command which deliberately closes input/output pipes, such as a daemon process, you may want to disable `use_pty`. Defaults to `True`
- **host_args** (*tuple or list*) – (Optional) Format command string with per-host arguments in `host_args`. `host_args` length must equal length of host list - *pssh.exceptions.HostArgumentException* is raised otherwise
- **encoding** (*str*) – Encoding to use for output. Must be valid Python codec
- **timeout** (*int*) – (Optional) Timeout in seconds for reading from stdout or stderr. Defaults to no timeout. Reading from stdout/stderr will raise *pssh.exceptions.Timeout* after `timeout` number seconds if remote output is not ready.
- **greenlet_timeout** (*float*) – (Optional) Greenlet timeout setting. Defaults to no timeout. If set, this function will raise `gevent.Timeout` after `greenlet_timeout` seconds if no result is available from greenlets. In some cases, such as when using proxy hosts, connection timeout is controlled by proxy server and getting result from greenlets may hang indefinitely if remote server is unavailable. Use this setting to avoid blocking in such circumstances. Note that `gevent.Timeout` is a special class that inherits from `BaseException` and thus **can not be caught** by `stop_on_errors=False`.

Return type Dictionary with host as key and *pssh.output.HostOutput* as value as per `pssh.pssh_client.ParallelSSHClient.get_output()`

Raises *pssh.exceptions.AuthenticationException* on authentication error

Raises *pssh.exceptions.UnknownHostException* on DNS resolution error

Raises *pssh.exceptions.ConnectionErrorException* on error connecting

Raises *pssh.exceptions.HostArgumentException* on number of host arguments not equal to number of hosts

Raises `TypeError` on not enough host arguments for cmd string format

Raises `KeyError` on no host argument key in arguments dict for cmd string format

Raises *pssh.exceptions.ProxyError* on errors connecting to proxy if a proxy host has been set.

Raises gevent.Timeout on greenlet timeout. Gevent timeout can not be caught by stop_on_errors=False.

Raises Exceptions from ssh2.exceptions for all other specific errors such as ssh2.exceptions.SocketDisconnectError et al.

scp_recv(*remote_file*, *local_file*, *recurse=False*, *copy_args=None*, *suffix_separator='_'*)

Copy remote file(s) in parallel via SCP as <local_file><suffix_separator><host> or as per copy_args argument.

With a local_file value of myfile and default separator _ the resulting filename will be myfile_myhost for the file from host myhost.

De-duplication behaviour is configurable by providing copy_args argument, see below.

This function, like *ParallelSSHClient.scp_send()*, returns a list of greenlets which can be *join*-ed on to wait for completion.

gevent.joinall() function may be used to join on all greenlets and will also raise exceptions if called with raise_error=True - default is *False*.

Alternatively call *.get* on each greenlet to raise any exceptions from it.

Exceptions listed here are raised when either gevent.joinall(<greenlets>, raise_error=True) is called or .get is called on each greenlet, not this function itself.

Parameters

- **remote_file** (*str*) – remote filepath to copy to local host
- **local_file** (*str*) – local filepath on local host to copy file to
- **recurse** (*bool*) – whether or not to recurse
- **suffix_separator** (*str*) – (Optional) Separator string between filename and host, defaults to _. For example, for a local_file value of myfile and default separator the resulting filename will be myfile_myhost for the file from host myhost. suffix_separator has no meaning if copy_args is provided
- **copy_args** (*tuple or list*) – (Optional) format remote_file and local_file strings with per-host arguments in copy_args. copy_args length *must* equal length of host list - *pssh.exceptions.HostArgumentException* is raised otherwise

Return type list(gevent.Greenlet) of greenlets for remote copy commands.

Raises ValueError when a directory is supplied to local_file and recurse is not set.

Raises *pssh.exceptions.HostArgumentException* on number of per-host copy arguments not equal to number of hosts.

Raises pss.exceptions.SCPError on errors copying file.

Raises OSError on local OS errors like permission denied.

Note: Local directories in local_file that do not exist will be created as long as permissions allow.

Note: File names will be de-duplicated by appending the hostname to the filepath separated by suffix_separator or as per copy_args argument if provided.

---

scp_send(*local_file*, *remote_file*, *recurse=False*)

Copy local file to remote file in parallel via SCP.

This function returns a list of greenlets which can be *join*-ed on to wait for completion.

`gevent.joinall()` function may be used to join on all greenlets and will also raise exceptions from them if called with `raise_error=True` - default is *False*.

Alternatively call *.get()* on each greenlet to raise any exceptions from it.

Note: Creating remote directories when either `remote_file` contains directory paths or `recurse` is enabled requires SFTP support on the server as libssh2 SCP implementation lacks directory creation support.

Parameters

- **local_file** (*str*) – Local filepath to copy to remote host
- **remote_file** (*str*) – Remote filepath on remote host to copy file to
- **recurse** (*bool*) – Whether or not to descend into directories recursively.

Return type list(`gevent.Greenlet`) of greenlets for remote copy commands.

Raises `pss.exceptions.SCPError` on errors copying file.

Raises `OSError` on local OS errors like permission denied.

6.2 Native Single Host Client

Native single host non-blocking client. Suitable for running asynchronous commands on a single host.

class pssh.clients.native.single.**SSHClient**(*host*, *user=None*, *password=None*, *port=None*, *pkey=None*, *num_retries=3*, *retry_delay=5*, *allow_agent=True*, *timeout=None*, *forward_ssh_agent=False*, *proxy_host=None*, *_auth_thread_pool=True*, *keepalive_seconds=60*)

ssh2-python (libssh2) based non-blocking SSH client.

Parameters

- **host** (*str*) – Host name or IP to connect to.
- **user** (*str*) – User to connect as. Defaults to logged in user.
- **password** (*str*) – Password to use for password authentication.
- **port** (*int*) – SSH port to connect to. Defaults to SSH default (22)
- **pkey** (*str*) – Private key file path to use for authentication. Path must be either absolute path or relative to user home directory like `~/<path>`.
- **num_retries** (*int*) – (Optional) Number of connection and authentication attempts before the client gives up. Defaults to 3.
- **retry_delay** (*int*) – Number of seconds to wait between retries. Defaults to `pssh.constants.RETRY_DELAY`
- **timeout** (*int*) – SSH session timeout setting in seconds. This controls timeout setting of authenticated SSH sessions.

- **allow_agent** (*bool*) – (Optional) set to False to disable connecting to the system's SSH agent
- **forward_ssh_agent** (*bool*) – (Optional) Turn on SSH agent forwarding - equivalent to *ssh -A* from the *ssh* command line utility. Defaults to True if not set.
- **proxy_host** (*str*) – Connection to host is via provided proxy host and client should use self.proxy_host for connection attempts.
- **keepalive_seconds** – Interval of keep alive messages being sent to server. Set to `0` or `False` to disable.

Raises *pssh.exceptions.PKeyFileError* on errors finding provided private key.

copy_file(*local_file*, *remote_file*, *recurse=False*, *sftp=None*, *_dir=None*)
Copy local file to host via SFTP.

Parameters

- **local_file** (*str*) – Local filepath to copy to remote host
- **remote_file** (*str*) – Remote filepath on remote host to copy file to
- **recurse** (*bool*) – Whether or not to descend into directories recursively.

Raises `ValueError` when a directory is supplied to `local_file` and `recurse` is not set

Raises `pss.exceptions.SFTPError` on SFTP initialisation errors

Raises *pssh.exceptions.SFTPIOError* on I/O errors writing via SFTP

Raises `IOError` on local file IO errors

Raises `OSError` on local OS errors like permission denied

copy_remote_file(*remote_file*, *local_file*, *recurse=False*, *sftp=None*, *encoding='utf-8'*)
Copy remote file to local host via SFTP.

Parameters

- **remote_file** (*str*) – Remote filepath to copy from
- **local_file** (*str*) – Local filepath where file(s) will be copied to
- **recurse** (*bool*) – Whether or not to recursively copy directories
- **encoding** (*str*) – Encoding to use for file paths.

Raises `ValueError` when a directory is supplied to `local_file` and `recurse` is not set

Raises `pss.exceptions.SFTPError` on SFTP initialisation errors

Raises *pssh.exceptions.SFTPIOError* on I/O errors reading from SFTP

Raises `IOError` on local file IO errors

Raises `OSError` on local OS errors like permission denied

disconnect()
Disconnect session, close socket if needed.

execute(*cmd*, *use_pty=False*, *channel=None*)
Execute command on remote server.

Parameters

- **cmd** (*str*) – Command to execute.
- **use_pty** (*bool*) – Whether or not to obtain a PTY on the channel.

- **channel** (ssh2.channel.Channel) – Use provided channel for execute rather than creating a new one.

mkdir(*sftp*, *directory*, *_parent_path=None*)
Make directory via SFTP channel.

Parent paths in the directory are created if they do not exist.

Parameters

- **sftp** (paramiko.sftp_client.SFTPClient) – SFTP client object
- **directory** (*str*) – Remote directory to create

Catches and logs at error level remote IOErrors on creating directory.

open_session()
Open new channel from session

read_output(*channel*, *timeout=None*)
Read standard output buffer from channel.

Parameters **channel** (ssh2.channel.Channel) – Channel to read output from.

read_output_buffer(*output_buffer*, *prefix=None*, *callback=None*, *callback_args=None*, *encoding='utf-8'*)
Read from output buffers and log to host_logger.

Parameters

- **output_buffer** (*iterator*) – Iterator containing buffer
- **prefix** (*str*) – String to prefix log output to host_logger with
- **callback** (*function*) – Function to call back once buffer is depleted:
- **callback_args** (*tuple*) – Arguments for call back function

read_stderr(*channel*, *timeout=None*)
Read standard error buffer from channel.

Parameters **channel** (ssh2.channel.Channel) – Channel to read output from.

run_command(*command*, *sudo=False*, *user=None*, *use_pty=False*, *shell=None*, *encoding='utf-8'*, *timeout=None*)
Run remote command.

Parameters

- **command** (*str*) – Command to run.
- **sudo** (*bool*) – Run command via sudo as super-user.
- **user** (*str*) – Run command as user via sudo
- **use_pty** (*bool*) – Whether or not to obtain a PTY on the channel.
- **shell** (*str*) – (Optional) Override shell to use to run command with. Defaults to login user's defined shell. Use the shell's command syntax, eg *shell='bash -c'* or *shell='zsh -c'*.
- **encoding** (*str*) – Encoding to use for output. Must be valid Python codec

Return type (channel, host, stdout, stderr, stdin) tuple.

scp_recv(*remote_file*, *local_file*, *recurse=False*, *sftp=None*, *encoding='utf-8'*)
Copy remote file to local host via SCP.

Note - Remote directory listings are gather via SFTP when `recurse` is enabled - SCP lacks directory list support. Enabling recursion therefore involves creating an extra SFTP channel and requires SFTP support on the server.

Parameters

- **remote_file** (*str*) – Remote filepath to copy from
- **local_file** (*str*) – Local filepath where file(s) will be copied to
- **recurse** (*bool*) – Whether or not to recursively copy directories
- **encoding** (*str*) – Encoding to use for file paths when recursion is enabled.

Raises *pssh.exceptions.SCPError* when a directory is supplied to `local_file` and `recurse` is not set.

Raises *pssh.exceptions.SCPError* on errors copying file.

Raises `IOError` on local file IO errors.

Raises `OSError` on local OS errors like permission denied.

scp_send(*local_file*, *remote_file*, *recurse=False*, *sftp=None*)

Copy local file to host via SCP.

Note - Directories are created via SFTP when `recurse` is enabled - SCP lacks directory create support. Enabling recursion therefore involves creating an extra SFTP channel and requires SFTP support on the server.

Parameters

- **local_file** (*str*) – Local filepath to copy to remote host
- **remote_file** (*str*) – Remote filepath on remote host to copy file to
- **recurse** (*bool*) – Whether or not to descend into directories recursively.

Raises `ValueError` when a directory is supplied to `local_file` and `recurse` is not set

Raises `pss.exceptions.SFTPError` on SFTP initialisation errors

Raises *pssh.exceptions.SFTPIOError* on I/O errors writing via SFTP

Raises `IOError` on local file IO errors

Raises `OSError` on local OS errors like permission denied

spawn_send_keepalive()

Spawns a new greenlet that sends keep alive messages every self.keepalive_seconds

wait_finished(*channel*, *timeout=None*)

Wait for EOF from channel and close channel.

Used to wait for remote command completion and be able to gather exit code.

Parameters **channel** (`ssh2.channel.Channel`) – The channel to use.

6.3 Paramiko based Single Host Client

class pssh.clients.miko.single.**SSHClient**(*host*, *user=None*, *password=None*, *port=None*, *pkey=None*, *forward_ssh_agent=True*, *num_retries=3*, *agent=None*, *allow_agent=True*, *timeout=10*, *proxy_host=None*, *proxy_port=22*, *proxy_user=None*, *proxy_password=None*, *proxy_pkey=None*, *channel_timeout=None*, *_openssh_config_file=None*, ***paramiko_kwargs*)

SSH client based on Paramiko with sane defaults.

Honours `~/.ssh/config` and `/etc/ssh/ssh_config` host entries for host, user name, port and key overrides.

Parameters

- **host** (*str*) – Hostname to connect to
- **user** (*str*) – (Optional) User to login as. Defaults to logged in user or user from ~/.ssh/config if set
- **password** (*str*) – (Optional) Password to use for login. Defaults to no password
- **port** (*int*) – (Optional) Port number to use for SSH connection. Defaults to `None` which uses SSH default
- **pkey** (`paramiko.pkey.PKey`) – (Optional) Client's private key to be used to connect with
- **num_retries** (*int*) – (Optional) Number of retries for connection attempts before the client gives up. Defaults to 3.
- **timeout** (*int*) – (Optional) Number of seconds to timeout connection attempts before the client gives up
- **forward_ssh_agent** (*bool*) – (Optional) Turn on SSH agent forwarding - equivalent to *ssh -A* from the *ssh* command line utility. Defaults to True if not set.
- **agent** (`paramiko.agent.Agent`) – (Optional) Override SSH agent object with the provided. This allows for overriding of the default paramiko behaviour of connecting to local SSH agent to lookup keys with our own SSH agent object.
- **forward_ssh_agent** – (Optional) Turn on SSH agent forwarding - equivalent to *ssh -A* from the *ssh* command line utility. Defaults to True if not set.
- **proxy_host** (*str*) – (Optional) SSH host to tunnel connection through so that SSH clients connects to self.host via client -> proxy_host -> host
- **proxy_port** (*int*) – (Optional) SSH port to use to login to proxy host if set. Defaults to 22.
- **channel_timeout** (*int*) – (Optional) Time in seconds before an SSH operation times out.
- **allow_agent** (*bool*) – (Optional) set to False to disable connecting to the SSH agent
- **paramiko_kwargs** (*dict*) – (Optional) Extra keyword arguments to be passed on to `paramiko.client.SSHClient.connect()`

copy_file(*local_file*, *remote_file*, *recurse=False*, *sftp=None*)

Copy local file to host via SFTP/SCP

Copy is done natively using SFTP/SCP version 2 protocol, no scp command is used or required.

Parameters

- **local_file** (*str*) – Local filepath to copy to remote host
- **remote_file** (*str*) – Remote filepath on remote host to copy file to
- **recurse** (*bool*) – Whether or not to descend into directories recursively.

Raises `ValueError` when a directory is supplied to `local_file` and `recurse` is not set

Raises `IOError` on I/O errors writing files

Raises `OSError` on OS errors like permission denied

copy_remote_file(*remote_file*, *local_file*, *recurse=False*, *sftp=None*)
Copy remote file to local host via SFTP/SCP

Copy is done natively using SFTP/SCP version 2, no scp command is used or required.

Parameters

- **remote_file** (*str*) – Remote filepath to copy from
- **local_file** (*str*) – Local filepath where file(s) will be copied to
- **recurse** (*bool*) – Whether or not to recursively copy directories

Raises `ValueError` when a directory is supplied to `local_file` and `recurse` is not set

Raises `IOError` on I/O errors creating directories or file

Raises `OSError` on OS errors like permission denied

exec_command(*command*, *sudo=False*, *user=None*, *shell=None*, *use_shell=True*, *use_pty=True*)
Wrapper to `paramiko.SSHClient.exec_command()`

Opens a new SSH session with a new pty and runs command before yielding the main gevent loop to allow other greenlets to execute.

Parameters

- **command** (*str*) – Command to execute
- **sudo** (*bool*) – (Optional) Run with sudo. Defaults to False
- **user** (*str*) – (Optional) User to switch to via sudo to run command as. Defaults to user running the python process
- **shell** – (Optional) Shell override to use instead of user login configured shell. For example `shell='bash -c'`
- **use_shell** (*bool*) – (Optional) Force use of shell on/off. Defaults to *True* for on
- **use_pty** (*bool*) – (Optional) Enable/Disable use of pseudo terminal emulation. This is required in vast majority of cases, exception being where a shell is not used and/or stdout/stderr/stdin buffers are not required. Defaults to `True`

Return type Tuple of *(channel, hostname, stdout, stderr, stdin)*. Channel is the remote SSH channel, needed to ensure all of stdout has been got, hostname is remote hostname the copy is to, stdout and stderr are buffers containing command output and stdin is standard input channel

mkdir(*sftp*, *directory*)
Make directory via SFTP channel.

Parent paths in the directory are created if they do not exist.

Parameters

- **sftp** (paramiko.sftp_client.SFTPClient) – SFTP client object
- **directory** (*str*) – Remote directory to create

Catches and logs at error level remote IOErrors on creating directory.

read_output_buffer(*output_buffer*, *prefix=None*, *callback=None*, *callback_args=None*, *encoding='utf-8'*)

Read from output buffers and log to host_logger

Parameters

- **output_buffer** (*iterator*) – Iterator containing buffer
- **prefix** (*str*) – String to prefix log output to `host_logger` with
- **callback** (*function*) – Function to call back once buffer is depleted:
- **callback_args** (*tuple*) – Arguments for call back function

6.4 Paramiko based Parallel Client

class pssh.clients.miko.parallel.**ParallelSSHClient**(*hosts*, *user=None*, *password=None*, *port=None*, *pkey=None*, *forward_ssh_agent=True*, *num_retries=3*, *timeout=120*, *pool_size=10*, *proxy_host=None*, *proxy_port=22*, *proxy_user=None*, *proxy_password=None*, *proxy_pkey=None*, *agent=None*, *allow_agent=True*, *host_config=None*, *channel_timeout=None*, *retry_delay=5*)

Parallel SSH client using paramiko based SSH client

Parameters

- **hosts** (*list(str)*) – Hosts to connect to
- **user** (*str*) – (Optional) User to login as. Defaults to logged in user or user from ~/.ssh/config or /etc/ssh/ssh_config if set
- **password** (*str*) – (Optional) Password to use for login. Defaults to no password
- **port** (*int*) – (Optional) Port number to use for SSH connection. Defaults to `None` which uses SSH default
- **pkey** (paramiko.pkey.PKey) – (Optional) Client's private key to be used to connect with
- **num_retries** (*int*) – (Optional) Number of retries for connection attempts before the client gives up. Defaults to 3.
- **retry_delay** (*int*) – Number of seconds to wait between retries. Defaults to `pssh.constants.RETRY_DELAY`

- **timeout** (*int*) – (Optional) Number of seconds to wait before connection and authentication attempt times out. Note that total time before timeout will be `timeout * num_retries + (5 * (num_retries-1))` number of seconds, where (5 * (num_retries-1)) refers to a five (5) second delay between retries.
- **forward_ssh_agent** (*bool*) – (Optional) Turn on/off SSH agent forwarding - equivalent to *ssh -A* from the *ssh* command line utility. Defaults to `True` if not set.
- **pool_size** (*int*) – (Optional) Greenlet pool size. Controls on how many hosts to execute tasks in parallel. Defaults to 10. Overhead in event loop will determine how high this can be set to, see scaling guide lines in project's readme.
- **proxy_host** (*str*) – (Optional) SSH host to tunnel connection through so that SSH clients connect to host via client -> proxy_host -> host
- **proxy_port** (*int*) – (Optional) SSH port to use to login to proxy host if set. Defaults to 22.
- **proxy_user** (*str*) – (Optional) User to login to `proxy_host` as. Defaults to logged in user.
- **proxy_password** (*str*) – (Optional) Password to login to `proxy_host` with. Defaults to no password
- **proxy_pkey** (`paramiko.pkey.PKey`) – (Optional) Private key to be used for authentication with `proxy_host`. Defaults to available keys from SSHAgent and user's home directory keys
- **agent** (*pssh.agent.SSHAgent*) – (Optional) SSH agent object to programmatically supply an agent to override system SSH agent with
- **host_config** (*dict*) – (Optional) Per-host configuration for cases where not all hosts use the same configuration values.
- **channel_timeout** (*int*) – (Optional) Time in seconds before reading from an SSH channel times out. For example with channel timeout set to one, trying to immediately gather output from a command producing no output for more than one second will timeout.
- **allow_agent** (*bool*) – (Optional) set to False to disable connecting to the system's SSH agent

finished(*output*)

Check if commands have finished without blocking

Parameters **output** – As returned by `pssh.pssh_client.ParallelSSHClient.get_output()`

Return type bool

get_output(*cmd*, *output*, *encoding='utf-8'*)

Get output from command greenlet.

output parameter is modified in-place.

Parameters

- **cmd** (`gevent.Greenlet`) – Command to get output from
- **output** (*dict*) – Dictionary containing *pssh.output.HostOutput* values to be updated with output from cmd

Return type None

join(*output*, *consume_output=False*)

Block until all remote commands in output have finished and retrieve exit codes

Parameters

- **output** (dict as returned by `pssh.pssh_client.ParallelSSHClient.get_output()`) – Output of commands to join on
- **consume_output** (*bool*) – Whether or not join should consume output buffers. Output buffers will be empty after `join` if set to `True`. Must be set to `True` to allow host logger to log output on call to `join`.

run_command(*command*, *sudo=False*, *user=None*, *stop_on_errors=True*, *shell=None*, *use_shell=True*, *use_pty=True*, *host_args=None*, *encoding='utf-8'*, ***paramiko_kwargs*)

Run command on all hosts in parallel, honoring self.pool_size, and return output buffers.

This function will block until all commands have been received by remote servers and then return immediately.

More explicitly, function will return after connection and authentication establishment and after commands have been received by successfully established SSH channels.

Any connection and/or authentication exceptions will be raised here and need catching *unless* `run_command` is called with `stop_on_errors=False` in which case exceptions are added to host output instead.

Parameters

- **command** (*str*) – Command to run
- **sudo** (*bool*) – (Optional) Run with sudo. Defaults to False
- **user** (*str*) – (Optional) User to run command as. Requires sudo access for that user from the logged in user account.
- **stop_on_errors** (*bool*) – (Optional) Raise exception on errors running command. Defaults to True. With stop_on_errors set to False, exceptions are instead added to output of *run_command*. See example usage below.
- **shell** (*str*) – (Optional) Override shell to use to run command with. Defaults to login user's defined shell. Use the shell's command syntax, eg *shell='bash -c'* or *shell='zsh -c'*.
- **use_shell** (*bool*) – (Optional) Run command with or without shell. Defaults to True - use shell defined in user login to run command string
- **use_pty** (*bool*) – (Optional) Enable/Disable use of pseudo terminal emulation. Disabling it will prohibit capturing standard input/output. This is required in majority of cases, exceptions being where a shell is not used and/or input/output is not required. In particular when running a command which deliberately closes input/output pipes, such as a daemon process, you may want to disable `use_pty`. Defaults to `True`
- **host_args** (*tuple or list*) – (Optional) Format command string with per-host arguments in `host_args`. `host_args` length must equal length of host list - *pssh.exceptions.HostArgumentException* is raised otherwise
- **encoding** (*str*) – Encoding to use for output. Must be valid Python codec
- **paramiko_kwargs** (*dict*) – (Optional) Extra keyword arguments to be passed on to `paramiko.client.SSHClient.connect()`

Return type Dictionary with host as key and *pssh.output.HostOutput* as value as per `pssh.pssh_client.ParallelSSHClient.get_output()`

Raises *pssh.exceptions.AuthenticationException* on authentication error

Raises *pssh.exceptions.UnknownHostException* on DNS resolution error

Raises *pssh.exceptions.ConnectionErrorException* on error connecting

Raises *pssh.exceptions.SSHException* on other undefined SSH errors

Raises *pssh.exceptions.HostArgumentException* on number of host arguments not equal to number of hosts

Raises `TypeError` on not enough host arguments for cmd string format

Raises `KeyError` on no host argument key in arguments dict for cmd string format

6.5 BaseParallelSSHClient

API documentation for common parallel client functionality.

Abstract parallel SSH client package

class pssh.clients.base_pssh.**BaseParallelSSHClient**(*hosts*, *user=None*, *password=None*, *port=None*, *pkey=None*, *allow_agent=True*, *num_retries=3*, *timeout=120*, *pool_size=10*, *host_config=None*, *retry_delay=5*)

Parallel client base class.

copy_file(*local_file*, *remote_file*, *recurse=False*, *copy_args=None*)

Copy local file to remote file in parallel

This function returns a list of greenlets which can be *join*-ed on to wait for completion.

`gevent.joinall()` function may be used to join on all greenlets and will also raise exceptions from them if called with `raise_error=True` - default is *False*.

Alternatively call *.get* on each greenlet to raise any exceptions from it.

Exceptions listed here are raised when either `gevent.joinall(<greenlets>, raise_error=True)` is called or `.get` is called on each greenlet, not this function itself.

Parameters

- **local_file** (*str*) – Local filepath to copy to remote host
- **remote_file** (*str*) – Remote filepath on remote host to copy file to
- **recurse** (*bool*) – Whether or not to descend into directories recursively.
- **copy_args** (*tuple or list*) – (Optional) format local_file and remote_file strings with per-host arguments in `copy_args`. `copy_args` length must equal length of host list - *pssh.exceptions.HostArgumentException* is raised otherwise

Return type List(`gevent.Greenlet`) of greenlets for remote copy commands

Raises `ValueError` when a directory is supplied to local_file and recurse is not set

Raises *pssh.exceptions.HostArgumentException* on number of per-host copy arguments not equal to number of hosts

Raises `IOError` on I/O errors writing files

Raises `OSError` on OS errors like permission denied

Note: Remote directories in *remote_file* that do not exist will be created as long as permissions allow.

copy_remote_file(*remote_file*, *local_file*, *recurse=False*, *suffix_separator='_'*, *copy_args=None*, ***kwargs*)

Copy remote file(s) in parallel as <local_file><suffix_separator><host>

With a `local_file` value of `myfile` and default separator `_` the resulting filename will be `myfile_myhost` for the file from host `myhost`.

This function, like `ParallelSSHClient.copy_file()`, returns a list of greenlets which can be *join*-ed on to wait for completion.

`gevent.joinall()` function may be used to join on all greenlets and will also raise exceptions if called with `raise_error=True` - default is *False*.

Alternatively call *.get* on each greenlet to raise any exceptions from it.

Exceptions listed here are raised when either `gevent.joinall(<greenlets>, raise_error=True)` is called or `.get` is called on each greenlet, not this function itself.

Parameters

- **remote_file** (*str*) – remote filepath to copy to local host
- **local_file** (*str*) – local filepath on local host to copy file to
- **recurse** (*bool*) – whether or not to recurse
- **suffix_separator** (*str*) – (Optional) Separator string between filename and host, defaults to _. For example, for a `local_file` value of `myfile` and default separator the resulting filename will be `myfile_myhost` for the file from host `myhost`. `suffix_separator` has no meaning if `copy_args` is provided
- **copy_args** (*tuple or list*) – (Optional) Format remote_file and local_file strings with per-host arguments in `copy_args`. `copy_args` length must equal length of host list - *pssh.exceptions.HostArgumentException* is raised otherwise

Return type list(`gevent.Greenlet`) of greenlets for remote copy commands

Raises `ValueError` when a directory is supplied to local_file and recurse is not set

Raises *pssh.exceptions.HostArgumentException* on number of per-host copy arguments not equal to number of hosts

Raises `IOError` on I/O errors writing files

Raises `OSError` on OS errors like permission denied

Note: Local directories in `local_file` that do not exist will be created as long as permissions allow.

Note: File names will be de-duplicated by appending the hostname to the filepath separated by `suffix_separator`.

finished(*output*)

Check if commands have finished without blocking

Parameters **output** – As returned by `pssh.pssh_client.ParallelSSHClient.get_output()`

Return type bool

get_exit_code(*host_output*)
Get exit code from host output *if available*.

Parameters **host_output** – Per host output as returned by `pssh.pssh_client.ParallelSSHClient.get_output()`

Return type int or None if exit code not ready

get_exit_codes(*output*)
Get exit code for all hosts in output *if available*. Output parameter is modified in-place.

Parameters **output** – As returned by `pssh.pssh_client.ParallelSSHClient.get_output()`

Return type None

get_last_output(*cmds=None*)
Get output for last commands executed by `run_command`

Parameters **cmds** (list(`gevent.Greenlet`)) – Commands to get output for. Defaults to `client.cmds`

Return type dict

get_output(*cmd*, *output*, *timeout=None*)
Get output from command.

Parameters

- **cmd** (`gevent.Greenlet`) – Command to get output from
- **output** (*dict*) – Dictionary containing *pssh.output.HostOutput* values to be updated with output from cmd

Return type None

6.6 Host Output

Output module of ParallelSSH

class pssh.output.**HostOutput**(*host*, *cmd*, *channel*, *stdout*, *stderr*, *stdin*, *exit_code=None*, *exception=None*)
Class to hold host output

Parameters

- **host** (*str*) – Host name output is for
- **cmd** (`gevent.Greenlet`) – Command execution object
- **channel** (`socket.socket` compatible object) – SSH channel used for command execution
- **stdout** (*generator*) – Standard output buffer
- **stderr** (*generator*) – Standard error buffer
- **stdin** (`file()`-like object) – Standard input buffer
- **exit_code** (*int or None*) – Exit code of command
- **exception** (`Exception` or `None`) – Exception from host if any

update(*update_dict*)
Override of dict update function for backwards compatibility

6.7 SSH Agent

SSH agent module of parallel-ssh

class pssh.agent.**SSHAgent**
paramiko.agent.Agent compatible class for programmatically supplying an SSH agent.

add_key(*key*)
Add key to agent.

Parameters **key** (paramiko.pkey.PKey) – Key to add

get_keys()
Return the list of keys available through the SSH agent, if any. If no SSH agent was running (or it couldn't be contacted), an empty list will be returned.

Returns a tuple of *.AgentKey* objects representing keys available on the SSH agent

6.8 Native Tunnel

class pssh.clients.native.tunnel.**Tunnel**(*host*, *in_q*, *out_q*, *user=None*, *password=None*, *port=None*, *pkey=None*, *num_retries=3*, *retry_delay=5*, *allow_agent=True*, *timeout=None*, *channel_retries=5*)
SSH proxy implementation with direct TCP/IP tunnels.

Each tunnel object runs in its own thread and can open any number of direct tunnels to remote host:port destinations on local ports over the same SSH connection.

To use, append `(host, port)` tuples into `Tunnel.in_q` and read listen port for tunnel connection from `Tunnel.out_q`.

`Tunnel.tunnel_open` is a *thread* event that will be set once tunnel is ready.

Parameters

- **host** (*str*) – Remote SSH host to open tunnels with.
- **in_q** (collections.deque) – Deque for requesting new tunnel to given `((host, port))`
- **out_q** (collections.deque) – Deque for feeding back tunnel listening ports.
- **user** (*str*) – (Optional) User to login as. Defaults to logged in user
- **password** (*str*) – (Optional) Password to use for login. Defaults to no password
- **port** (*int*) – (Optional) Port number to use for SSH connection. Defaults to `None` which uses SSH default (22)
- **pkey** (*str*) – Private key file path to use. Note that the public key file pair *must* also exist in the same location with name `<pkey>.pub`
- **num_retries** (*int*) – (Optional) Number of connection and authentication attempts before the client gives up. Defaults to 3.

- **retry_delay** (*int*) – Number of seconds to wait between retries. Defaults to `pssh.constants.RETRY_DELAY`
- **timeout** (*int*) – SSH session timeout setting in seconds. This controls timeout setting of authenticated SSH sessions.
- **allow_agent** (*bool*) – (Optional) set to False to disable connecting to the system's SSH agent.

run()
Thread run target. Starts tunnel client and waits for incoming tunnel connection requests from `Tunnel.in_q`.

6.9 Utility functions

Module containing static utility functions for parallel-ssh.

pssh.utils.**enable_host_logger**()
Enable host logger for logging stdout from remote commands as it becomes available.

pssh.utils.**enable_logger**(*_logger*, *level=20*)
Enables logging to stdout for given logger

pssh.utils.**load_private_key**(*_pkey*)
Load private key from pkey file object or filename.

For Paramiko based clients only.

Parameters **pkey** (*file/str*) – File object or file name containing private key

pssh.utils.**read_openssh_config**(*host*, *config_file=None*)
Parses user's OpenSSH config for per hostname configuration for hostname, user, port and private key values

Parameters **host** – Hostname to lookup in config

6.10 Exceptions

Exceptions raised by parallel-ssh classes.

exception pssh.exceptions.**AuthenticationException**
Raised on authentication error (user/password/ssh key error)

exception pssh.exceptions.**ConnectionErrorException**
Raised on error connecting (connection refused/timed out)

exception pssh.exceptions.**HostArgumentException**
Raised on errors with per-host arguments to parallel functions

exception pssh.exceptions.**PKeyFileError**
Raised on errors finding private key file

exception pssh.exceptions.**ProxyError**
Raised on proxy errors

exception pssh.exceptions.**SCPError**
Raised on errors copying file via SCP

exception pssh.exceptions.**SFTPError**
Raised on SFTP errors

exception pssh.exceptions.**SFTPIOError**
Raised on SFTP IO errors

exception pssh.exceptions.**SSHException**
Raised on SSHException error - error authenticating with SSH server

exception pssh.exceptions.**SessionError**
Raised on errors establishing SSH session

exception pssh.exceptions.**Timeout**
Raised on timeout requested and reached

exception pssh.exceptions.**UnknownHostException**
Raised when a host is unknown (dns failure)

CHAPTER 7

Change Log

7.1 1.9.1

7.1.1 Fixes

- Native client SCP and SFTP uploads would not handle partial writes from waiting on socket correctly.
- Native client `copy_file` SFTP upload would get stuck repeating same writes until killed when copying multi-MB files from Windows clients - #148
- Native client `scp_send` would not correctly preserve file mask of local file on the remote.
- Native client tunnel, used for proxy implementation, would not handle partial writes from waiting on socket correctly.

7.2 1.9.0

7.2.1 Changes

- Removed libssh2 native library dependency in favour of bundled `ssh2-python` libssh2 library.
- Changed native client forward agent default behaviour to off due to incompatibility with certain SSH server implementations.
- Added keep-alive functionality to native client - defaults to `60` seconds. `ParallelSSHClient(<..>, keepalive_seconds=<interval>)` to configure interval. Set to `0` to disable.
- Added `~/.ssh/id_ecdsa` default identity location to native client.

7.3 1.8.2

7.3.1 Fixes

- Native parallel client `forward_ssh_agent` flag would not be applied correctly.

7.4 1.8.1

7.4.1 Fixes

- Native client socket timeout setting would be longer than expected - #133

7.4.2 Packaging

- Added Windows 3.7 wheels

7.5 1.8.0

7.5.1 Changes

- Native client no longer requires public key file for authentication.
- Native clients raise `pssh.exceptions.PKeyFileError` on object initialisation if provided private key file paths cannot be found.
- Native clients expand user directory (`~/<path>`) on provided private key paths.
- Parallel clients raise `TypeError` when provided `hosts` is a string instead of list or other iterable.

7.6 1.7.0

7.6.1 Changes

- Better tunneling implementation for native clients that supports multiple tunnels over single SSH connection for connecting multiple hosts through single proxy.
- Added `greenlet_timeout` setting to native client `run_command` to pass on to getting greenlet result to allow for greenlets to timeout.
- Native client raises specific exceptions on non-authentication errors connecting to host instead of generic `SessionError`.

7.6.2 Fixes

- Native client tunneling would not work correctly - #123.
- `timeout` setting was not applied to native client sockets.

- Native client would have `SessionError` instead of `Timeout` exceptions on timeout errors connecting to hosts.

7.7 1.6.3

7.7.1 Changes

- Re-generated C code with latest Cython release.

7.7.2 Fixes

- `ssh2-python` >= 0.14.0 support.

7.8 1.6.2

7.8.1 Fixes

- Native client proxy initialisation failures were not caught by `stop_on_errors=False` - #121.

7.9 1.6.1

7.9.1 Fixes

- Host would always be *127.0.0.1* when using `proxy_host` on native client - #120.

7.10 1.6.0

7.10.1 Changes

- Added `scp_send` and `scp_recv` functions to native clients for sending and receiving files via SCP respectively.
- Refactoring - clients moved to their own sub-package - `pssh.clients` - with backwards compatibility for imports from `pssh.pssh_client` and `pssh.pssh2_client`.
- Show underlying exception from native client library when raising `parallel-ssh` exceptions.
- `host` parameter added to all exceptions raised by parallel clients - #116
- Deprecation warning for client imports.
- Deprecation warning for default client changing from paramiko to native client as of `2.0.0`.
- Upgrade embedded `libssh2` in binary wheels to latest version plus enhancements.
- Adds support for ECDSA host keys for native client.
- Adds support for SHA-256 host key fingerprints for native client.

- Added SSH agent forwarding to native client, defaults to on as per paramiko client - `forward_ssh_agent` keyword parameter.
- Windows wheels switched to OpenSSL back end for native client.
- Windows wheels include zlib and have compression enabled for native client.
- Added OSX 10.13 wheel build.

7.10.2 Fixes

- Windows native client could not connect to newer SSH servers - thanks Pavel.

Note - libssh2 changes apply to binary wheels only. For building from source, see documentation.

7.11 1.5.5

7.11.1 Fixes

- Use of `sudo` in native client incorrectly required escaping of command.

7.12 1.5.4

7.12.1 Changes

- Compatibility with `ssh2-python` >= `0.11.0`.

7.13 1.5.2

7.13.1 Changes

- Output generators automatically restarted on call to `join` so output can resume on any timeouts.

7.14 1.5.1

7.14.1 Fixes

- Output `pssh.exceptions.Timeout` exception raising was not enabled.

7.15 1.5.0

7.15.1 Changes

- `ParallelSSH2Client.join` with timeout now consumes output to ensure command completion status is accurate.

- Output reading now raises `pssh.exceptions.Timeout` exception when timeout is requested and reached with command still running.

7.15.2 Fixes

- `ParallelSSH2Client.join` would always raise `Timeout` when output has not been consumed even if command has finished - #104.

7.16 1.4.0

7.16.1 Changes

- `ParallelSSH2Client.join` now raises `pssh.exceptions.Timeout` exception when timeout is requested and reached with command still running.

7.16.2 Fixes

- `ParallelSSH2Client.join` timeout duration was incorrectly for per-host rather than total.
- SFTP read flags were not fully portable.

7.17 1.3.2

7.17.1 Fixes

- Binary wheels would have bad version info and require *git* for installation.

7.18 1.3.1

7.18.1 Changes

- Added `timeout` optional parameter to `join` and `run_command`, for reading output, on native clients.

7.18.2 Fixes

- From source builds when Cython is installed with recent versions of `ssh2-python`.

7.19 1.3.0

7.19.1 Changes

- Native clients proxy implementation
- Native clients connection and authentication retry mechanism

Proxy/tunnelling implementation is experimental - please report any issues.

7.20 1.2.1

7.20.1 Fixes

- PyPy builds

7.21 1.2.0

7.21.1 Changes

- New `ssh2-python` (`libssh2`) native library based clients
- Added `retry_delay` keyword parameter to parallel clients
- Added `get_last_output` function for retrieving output of last executed commands
- Added `cmds` attribute to parallel clients for last executed commands

7.21.2 Fixes

- Remote path for SFTP operations was created incorrectly on Windows - #88 - thanks @moscoquera
- Parallel client key error when openssh config with a host name override was used - #93
- Clean up after paramiko clients

7.22 1.1.1

7.22.1 Changes

- Accept Paramiko version `2` but < `2.2` (it's buggy).

7.23 1.1.0

7.23.1 Changes

- Allow passing on of additional keyword arguments to underlying SSH library via `run_command` - #85

7.24 1.0.0

7.24.1 Changes from *0.9x* series API

- *ParallelSSHClient.join* no longer consumes output buffers

- Command output is now a dictionary of host name -> host output object with *stdout* and et al attributes. Host output supports dictionary-like item lookup for backwards compatibility. No code changes are needed to output use though documentation will from now on refer to the new attribute style output. Dictionary-like item access is deprecated and will be removed in future major release, like *2.x*.
- Made output encoding configurable via keyword argument on *run_command* and *get_output*
- *pssh.output.HostOutput* class added to hold host output
- Added *copy_remote_file* function for copying remote files to local ones in parallel
- Deprecated since *0.70.0 ParallelSSHClient* API endpoints removed
- Removed setuptools >= 28.0.0 dependency for better compatibility with existing installations. Pip version dependency remains for Py 2.6 compatibility with gevent - documented on project's readme
- Documented *use_pty* parameter of run_command
- *SSHClient read_output_buffer* is now public function and has gained callback capability
- If using the single *SSHClient* directly, *read_output_buffer* should now be used to read output buffers - this is not needed for *ParallelSSHClient*
- *run_command* now uses named positional and keyword arguments

CHAPTER 8

In a nutshell

Client will attempt to use all available keys under `~/.ssh` as well as any keys in an SSH agent, if one is available.

```
from __future__ import print_function

from pssh.clients import ParallelSSHClient

client = ParallelSSHClient(['localhost'])
output = client.run_command('whoami')
for line in output['localhost'].stdout:
    print(line)
```

Output

```
<your username here>
```

Note: There is also a now deprecated paramiko based client available under `pssh.clients.miko` that has much the same API. It supports some features not currently supported by the native client - see feature comparison.

From version `2.x.x` onwards, the clients under `pssh.clients.miko` will be an optional `extras` install.

8.1 Indices and tables

- genindex

Python Module Index

p

Index

M

O

P

R

S

T

U

W