Parallel-SSH Documentation

Release 2.0.0b2

P Kittenis

Sep 18, 2020

CONTENTS

`parallel-ssh` is a non-blocking parallel SSH client library.

It provides clients based on C libraries with an easy to use Python API providing native code levels of performance and stability.

CHAPTER

ONE

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 pssh.clients import ParallelSSHClient

client = ParallelSSHClient(['localhost', 'localhost'])
output = client.run_command('uname')
for host_out in output:
    for line in host_out.stdout:
        print(line)
    exit_code = host_out.exit_code
```

Output

```
<Uname output>
<Uname output>
```

CHAPTER

TWO

SINGLE HOST CLIENT

Single host client is also available with similar API.

```
from pssh.clients import SSHClient

client = SSHClient('localhost')
host_out = client.run_command('uname')
for line in host_out.stdout:
    print(line)
exit_code = host_out.exit_code
```

Output

```
<Uname output>
```

2.1 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.

2.1.1 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.

2.2 Installation

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

2.2.1 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.

2.2.2 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*.

Dependency	Minimum Version
`ssh2-python`	`0.19.0`
`ssh-python`	`0.6.0`
`gevent`	`1.1`

2.2.3 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 for developing changes:

```
pip install -r requirements_dev.txt
```

2.2.4 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.

This script and docker files can be adapted for other distributions.

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
```

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.

2.3 Quickstart

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

2.3.1 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.

A complete example is shown below.

Examples all assume a valid key is available on a running SSH agent. See Programmatic Private Key Authentication for authenticating without an SSH agent.

Complete Example

Host list can contain identical hosts. Commands are executed concurrently on every host given to the client regardless.

```
from pssh.clients import ParallelSSHClient

hosts = ['localhost', 'localhost', 'localhost', 'localhost']
client = ParallelSSHClient(hosts)
cmd = 'uname'

output = client.run_command(cmd)
for host_out in output:
    for line in host_out.stdout:
        print(line)
```

Output:

```
Linux
Linux
Linux
Linux
```

2.3.2 Single Host Client

`parallel-ssh` has a fully featured, non-blocking single host client that it uses for all its parallel commands.

Users that do not need the parallel capabilities can use the single host client for a simpler way to run asynchronous non-blocking commands on a remote host.

```
from pssh.clients import SSHClient

host = 'localhost'
cmd = 'uname'
client = SSHClient(host)

host_out = client.run_command(cmd)
for line in host_out.stdout:
    print(line)
```

Output::

```
Linux
```

Step by Step

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

```
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('uname')
```

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

2.3.3 Run Command Output

Standard Output

Standard output, aka `stdout`, for a given *`HostOutput`* object.

```
for line in host_out.stdout:
    print(line)
```

Output

```
<line by line output>
<line by line output>
<..>
```

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.

`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(host_out.stdout)
```

Warning: This will store the entirety of stdout into memory.

All hosts iteration

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

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

Complete Example

```
from pssh.clients import ParallelSSHClient

client = ParallelSSHClient(['localhost', 'localhost'])
output = client.run_command('whoami')
client.join(output)

for host_output in output:
    hostname = host_output.host
    stdout = list(host_output.stdout)
    print("Host %s: exit code %s, output %s" % (
          hostname, host_output.exit_code, stdout))
```

Output

```
localhost: exit code 0, stdout ['<username>']
localhost: exit code 0, stdout ['<username>']
```

New in 1.10.0

2.3.4 Exit codes

Exit codes are available on the host output object as a dynamic property. Exit code will be `None` if not available, or the exit code as reported by channel.

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

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

As of `1.11.0`, `client.join` is not required as long as output has been gathered.

```
for host_out in output:
    for line in host_out.stdout:
        print(line)
    print(host_out.exit_code)
```

See also:

pssh.output.HostOutput Host output class documentation.

2.3.5 Authentication

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

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.

Programmatic Private Key authentication

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

```
from pssh.clients import ParallelSSHClient

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

Where `my_pkey` is a private key file in the current directory.

To use files under a user's `.ssh` directory:

```
import os

client = ParallelSSHClient(hosts, pkey='~/.ssh/my_pkey')
```

2.3.6 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_output in output:
    for line in host_output.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

2.3.7 Host Logger

There is a built in host logger that can be enabled to automatically log standard output from remote hosts. This requires the `consume_output=True` flag 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()
```

(continues on next page)

(continued from previous page)

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

Output

```
[localhost]       Linux
```

2.3.8 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 on *`HostOutput`* is a file-like object giving access to the remote stdin channel that can be written to:

```
output = client.run_command('read')
host_output = output[0]
stdin = host_output.stdin
stdin.write("writing to stdin\\n")
stdin.flush()
for line in host_output.stdout:
    print(line)
```

Output

```
writing to stdin
```

2.3.9 Errors and Exceptions

By default, `parallel-ssh` will raise exception 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_output in output:
    host = host_output.host
    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 and in API documentation.

2.4 Clients Feature Comparison

The two available client types, default native clients based on `ssh2-python` (`libssh2`) and clients under `pssh.clients.ssh` based on `ssh-python` (`libssh`) support different authentication mechanisms and features.

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

Feature	ssh2-python (libssh2)	ssh-python (libssh)
Agent forwarding	No	Not yet implemented
Proxying/tunnelling	Yes	No
Kerberos (GSS) authentication	Not supported	Yes
Private key file authentication	Yes	Yes
Agent authentication	Yes	Yes
Password authentication	Yes	Yes
SFTP copy to/from hosts	Yes	No
OpenSSH config parsing	Not yet implemented	Not yet implemented
ECDSA keys support	Yes	Yes
ED25519 keys support	Yes	Yes
Certificate authentication	Not supported	Not yet implemented
SCP functionality	Yes	No
Keep-alive functionality	Yes	No

The default client offers the most features, but lacks certain authentication mechanisms like GSS-API and certificate authentication. Both client types are based on C libraries and offer similar levels of performance.

Users that need the authentication mechanisms not supported by the default client can `from pssh.clients.ssh import ParallelSSHClient` instead of the default client.

2.5 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.

2.5.1 Agents and Private Keys

Programmatic Private Keys

By default, `parallel-ssh` will attempt to use loaded keys in an available SSH agent as well as default identity files under the user's home directory.

See *IDENTITIES* in *SSHClient* for the list of default identity files.

A private key can also be provided programmatically.

```
from pssh.clients import ParallelSSHClient

client = ParallelSSHClient(hosts, pkey="~/.ssh/my_key")
```

Where `my_key` is a private key file under *.ssh* in the user's home directory.

2.5.2 Native Clients

ssh2-python (libssh2)

Starting from version `1.2.0`, the default client in `parallel-ssh` is based on *ssh2-python* (*libssh2*). It is a native client, offering C level performance with an easy to use Python API.

See this post for a performance comparison of the available clients in the *1.x.x* series.

```
from pssh.clients import ParallelSSHClient

hosts = ['my_host', 'my_other_host']
client = ParallelSSHClient(hosts)

output = client.run_command('uname')
for host_out in output:
    for line in host_out.stdout:
        print(line)
```

See also:

Feature comparison for how the *2.x.x* client types compare.

API documentation for parallel and single native clients.

ssh-python (libssh) Client

From version *1.12.0* another client based on libssh via *ssh-python* is provided for testing purposes.

The API is similar to the default client, while `ssh-python` offers more supported authentication methods compared to the default client.

On the other hand, this client lacks SCP, SFTP and proxy functionality.

```
from pssh.clients.ssh import ParallelSSHClient

hosts = ['localhost', 'localhost']
client = ParallelSSHClient(hosts)

output = client.run_command('uname')
client.join(output)
for host_out in output:
    for line in host_out.stdout:
        print(line)
```

GSS-API Authentication - aka Kerberos

GSS authentication allows logins using Windows LDAP configured user accounts via Kerberos on Linux.

```
from pssh.clients.ssh import ParallelSSHClient

client = ParallelSSHClient(hosts, gssapi_auth=True, gssapi_server_identity='gss_
↪server_id')

output = client.run_command('id')
client.join(output)
```

(continues on next page)

(continued from previous page)

```
for host_out in output:
    for line in output.stdout:
        print(line)
```

ssh-python *ParallelSSHClient* only.

2.5.3 Tunnelling

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 proxy or 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 configuration.

```
hosts = [<..>]
client = ParallelSSHClient(hosts, user='target_host_user',
                           proxy_host='bastion', proxy_user='my_proxy_user',
                           proxy_port=2222,
                           proxy_pkey='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: The current implementation of tunnelling suffers from poor performance when first establishing connections to many hosts - this is due to be resolved in a future release.

Proxy host connections are asynchronous and use the SSH protocol's native TCP tunnelling - 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.

2.5.4 Join and Output Timeouts

Clients have timeout functionality on reading output and `client.join`. Join timeout is a timeout on all parallel commands in total and is separate from `ParallelSSHClient(timeout=<..>)` which is applied to SSH session operations individually.

Timeout exceptions contain attributes for which commands have finished and which have not so client code can get output from any finished commands when handling timeouts.

```
from pssh.exceptions import Timeout

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

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

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

In the case of reading from output such as in the example above, timeout value is per output stream - meaning separate timeouts for stdout and stderr as well as separate timeout per host remote command.

New in 1.5.0

Reading Output from Partially Finished Commands

Timeout exception when calling `join` has finished and unfinished commands as arguments.

This can be used to handle sets of commands that have finished and those that have not separately, for example to only gather output on finished commands to avoid blocking.

```
output = client.run_command(..)
try:
    client.join(output, timeout=5)
except Timeout as ex:
    # Some commands timed out
    finished_output = ex.args[2]
    unfinished_output = ex.args[3]
else:
    # No timeout, all commands finished within five seconds
    finished_output = output
    unfinished_output = None
for host_out in finished_output:
    for line in host_out.stdout:
        print(line)
if unfinished_output is not None:
    <handle unfinished output>
```

In the above example, output is printed only for those commands which have completed within the five second timeout.

Client code may choose to then join again only on the unfinished output if some commands have failed in order to gather remaining output.

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_out in output:
    try:
        for line in host_out.stdout:
            stdout.append(line)
    except Timeout:
        # This allows client code to continue to read output after timeout
        client.reset_output_generators(host_out, timeout=1)

# Closing channel which has PTY has the effect of terminating
# any running processes started on that channel.
for host_out in output:
    host_out.client.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 is done by `client.reset_output_generators` in the above example.

Generator reset 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.

2.5.5 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.config import HostConfig

hosts = ['localhost', 'localhost']
host_config = [
    HostConfig(port=2222, user='user1',
               password='pass', private_key='my_pkey.pem'),
    HostConfig(port=2223, user='user2',
               password='pass', private_key='my_other_key.pem'),
]

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

In the above example, the client is configured to connect to hostname `localhost`, port `2222` with username `user1`, password `pass` and private key file `my_pkey.pem` and hostname `localhost`, port `2222` with username `user1`, password `pass` and private key file `my_other_pkey.pem`.

When using `host_config`, the number of `HostConfig` entries must match the number of hosts in `client.hosts`. An exception is raised on client initialisation if not.

Note: Currently only `port`, `user`, `password` and `private_key` `HostConfig` values are used.

Note: Proxy host configuration is currently per `ParallelSSHClient` and cannot yet be provided via per-host configuration. Multiple clients can be used to make use of multiple proxy hosts.

This feature will be provided in future releases.

2.5.6 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'),))
```

This expands to the following per host commands:

```
host1: 'host1_cmd1 host1_cmd2'
host2: 'host2_cmd1 host2_cmd2'
host3: '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 `echo command-1`, second host `echo command-2` and so on.

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

This expands to the following per host commands:

```
host1: 'echo command-0'
host2: 'echo command-1'
host3: 'echo command-2'
```

2.5.7 Run command features and options

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

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)
```

Note: Note the inclusion of the new line `\n` when using sudo with a password.

Run with configurable shell

By default the client will use the login user's shell to execute commands per the SSH protocol.

Shell to use is configurable:

```
client = <..>

output = client.run_command(<..>, shell='zsh -c')
for host_out in output;
    for line in host_out.stdout:
        print(line)
```

Commands will be run under the `zsh` shell in the above example. The command string syntax of the shell must be used, typically `<shell> -c`.

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[0].stdout)
```

Contents of `stdout` are *UTF-16* encoded.

Note: Encoding must be valid Python codec

Enabling use of pseudo terminal emulation

Pseudo Terminal Emulation (PTY) can be enabled when running commands, defaults to off.

Enabling it has some side effects on the output and behaviour of commands such as combining stdout and stderr output - see *bash* man page for more information.

All output, including stderr, is sent to the `stdout` channel with PTY enabled.

```
client = <..>

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

Note output is from the `stdout` channel while it was writeen to `stderr`.

Output

```
asdf
```

Stderr is empty:

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

No output from `stderr`.

2.5.8 SFTP and SCP

SFTP and SCP are both supported by `parallel-ssh` and functions are provided by the client for copying files with SFTP to and from remote servers - default native client only.

Neither SFTP nor SCP have a shell interface and no output is provided for any SFTP/SCP 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.

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.clients 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.

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.

Configurable per host Filenames

File name arguments, for both local and remote files and for copying to and from remote hosts, can be configured on a per-host basis similarly to *host arguments* in `run_command`.

For example, to copy the local files `['local_file_1', 'local_file_2']` as remote files `['remote_file_1', 'remote_file_2']` on the two hosts `['host1', 'host2']`

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

client = ParallelSSHClient(hosts)

copy_args = [{'local_file': 'local_file_1',
              'remote_file': 'remote_file_1',
              },
             {'local_file': 'local_file_2',
              'remote_file': 'remote_file_2',
              }]
cmds = client.copy_file('%(local_file)s', '%(remote_file)s',
                        copy_args=copy_args)
joinall(cmds)
```

The client will copy `local_file_1` to `host1` as `remote_file_1` and `local_file_2` to `host2` as `remote_file_2`.

Items in `copy_args` list may be tuples or dictionaries as shown above. Number of `copy_args` must match length of `client.hosts` if provided.

`copy_remote_file` may be used in the same manner to configure remote and local file names per host.

See also:

`copy_remote_file` API documentation and exceptions raised.

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_remote_file`* directly.

```
from pssh.clients import SSHClient

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

See also:

`SSHClient.copy_remote_file` API documentation and exceptions raised.

2.5.9 Hosts filtering and overriding

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`.

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'))
    host='otherhost'
    exit_code=None
    <..>
```

2.6 API Documentation

2.6.1 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=100*, *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*, *identity_auth=True*)

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.
- **timeout** – (Optional) Individual SSH client timeout setting in seconds passed on to each SSH client spawned by *ParallelSSHClient*.

  This controls timeout setting of socket operations used for SSH sessions *on a per session basis* meaning for each individual SSH session.

  Defaults to OS default - usually 60 seconds.

  Parallel functions like *run_command* and *join* have a cummulative timeout setting that is separate to and not affected by *self.timeout*.
- **pool_size** (*int*) – (Optional) Greenlet pool size. Controls concurrency, on how many hosts to execute tasks in parallel. Defaults to 100. 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.

- **identity_auth** (*bool*) – (Optional) set to False to disable attempting to authenticate with default identity files from *pssh.clients.base_ssh_client.BaseSSHClient.IDENTITIES*
- **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`.

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*, *return_list=False*)

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

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 in the case of on new connections and after execute 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. Defaults to `False`
- **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_list** (*bool*) – No-op - list of `HostOutput` always returned. Parameter kept for backwards compatibility - to be removed in future releases.

Return type Dictionary with host as key and *pssh.output.HostOutput* as value *or* list(*pssh.output.HostOutput*) when `return_list=True`

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.

2.6.2 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*, *identity_auth=True*)
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
- **identity_auth** (*bool*) – (Optional) set to False to disable attempting to authenticate with default identity files from *pssh.clients.base_ssh_client.BaseSSHClient.IDENTITIES*
- **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.

auth()

close_channel(*channel*)

configure_keepalive()

copy_file(*local_file*, *remote_file*, *recurse=False*, *sftp=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.

finished(*channel*)
Checks if remote command has finished - has server sent client EOF.

Return type bool

get_exit_status(*channel*)

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

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

Parameters

- **sftp** (`ssh2.sftp.SFTP`) – 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. Returns a generator of line by line output.

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

Return type generator

read_stderr(*channel*, *timeout=None*)
Read standard error buffer from channel. Returns a generator of line by line output.

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

Return type generator

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 gathered 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

sftp_get(*sftp*, *remote_file*, *local_file*)

sftp_put(*sftp*, *local_file*, *remote_file*)

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.
- **timeout** (*float*) – Timeout value in seconds - defaults to no timeout.

Raises *pssh.exceptions.Timeout* after <timeout> seconds if timeout given.

2.6.3 ssh-python based Parallel Client

API documentation for the `ssh-python` (`libssh`) based parallel client.

class pssh.clients.ssh.parallel.**ParallelSSHClient**(*hosts*, *user=None*, *password=None*, *port=22*, *pkey=None*, *num_retries=3*, *timeout=None*, *pool_size=100*, *allow_agent=True*, *host_config=None*, *retry_delay=5*, *forward_ssh_agent=False*, *gssapi_auth=False*, *gssapi_server_identity=None*, *gssapi_client_identity=None*, *gssapi_delegate_credentials=False*, *identity_auth=True*)

ssh-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) Individual SSH client timeout setting in seconds passed on to each SSH client spawned by *ParallelSSHClient*.

  This controls timeout setting of socket operations used for SSH sessions *on a per session basis* meaning for each individual SSH session.

  Defaults to OS default - usually 60 seconds.

  Parallel functions like *run_command* and *join* have a cummulative timeout setting that is separate to and not affected by *self.timeout*.
- **pool_size** (*int*) – (Optional) Greenlet pool size. Controls concurrency, on how many hosts to execute tasks in parallel. Defaults to 100. 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. Currently unused - always off.
- **identity_auth** (*bool*) – (Optional) set to False to disable attempting to authenticate with default identity files from *pssh.clients.base_ssh_client.BaseSSHClient.IDENTITIES*
- **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. Currently unused meaning always off.
- **gssapi_server_identity** (*str*) – Set GSSAPI server identity.
- **gssapi_server_identity** – Set GSSAPI client identity.
- **gssapi_delegate_credentials** (*bool*) – Enable/disable server credentials delegation.

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

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*, *return_list=False*)

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

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 in the case of on new connections and after execute 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. Defaults to `False`
- **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.

2.6.4 ssh-python based Single Host Client

Single host non-blocking client based on `ssh-python` (`libssh`). Suitable for running asynchronous commands on a single host.

class pssh.clients.ssh.single.**SSHClient**(*host*, *user=None*, *password=None*, *port=None*, *pkey=None*, *num_retries=3*, *retry_delay=5*, *allow_agent=True*, *timeout=None*, *identity_auth=True*, *gssapi_auth=False*, *gssapi_server_identity=None*, *gssapi_client_identity=None*, *gssapi_delegate_credentials=False*, *_auth_thread_pool=True*)

ssh-python based non-blocking 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*) – (Optional) If provided, all commands will timeout after <timeout> number of seconds.
- **allow_agent** (*bool*) – (Optional) set to False to disable connecting to the system's SSH agent. Currently unused.
- **identity_auth** (*bool*) – (Optional) set to False to disable attempting to authenticate with default identity files from *pssh.clients.base_ssh_client.BaseSSHClient.IDENTITIES*
- **gssapi_server_identity** (*str*) – Enable GSS-API authentication. Uses GSS-MIC key exchange. Enabled if either gssapi_server_identity or gssapi_client_identity are provided.
- **gssapi_server_identity** – Set GSSAPI server identity.
- **gssapi_client_identity** (*str*) – Set GSSAPI client identity.
- **gssapi_delegate_credentials** (*bool*) – Enable/disable server credentials delegation.

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

auth()

close_channel(*channel*)
Close channel.

Parameters **channel** (ssh.channel.Channel) – The channel to close.

disconnect()
Close socket if needed.

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

Parameters

- **cmd** (*str*) – The command string to execute.
- **use_pty** (*bool*) – Whether or not to request a PTY on the channel executing command.
- **channel** (ssh.channel.Channel) – Channel to use. New channel is created if not provided.

finished(*channel*)
Checks if remote command has finished - has server sent client EOF.

Return type bool

get_exit_status(*channel*)
Get exit status from channel if ready else return *None*.

Return type int or *None*

open_session()
Open new channel from session.

read_output(*channel*, *timeout=None*, *is_stderr=False*)
Read standard output buffer from channel. Returns a generator of line by line output.

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

Return type generator

read_stderr(*channel*, *timeout=None*)
Read standard error buffer from channel. Returns a generator of line by line output.

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

Return type generator

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** (ssh.channel.Channel) – The channel to use.
- **timeout** (*float*) – Timeout value in seconds - defaults to no timeout.

Raises *pssh.exceptions.Timeout* after <timeout> seconds if timeout given.

2.6.5 BaseParallelSSHClient

API documentation for common parallel client functionality.

This class is abstract and contains functions that need to be implemented for each underlying SSH library.

Abstract parallel SSH client package

class pssh.clients.base.parallel.**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*, *identity_auth=True*)
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_last_output(*cmds=None*, *greenlet_timeout=None*, *return_list=True*)

Get output for last commands executed by `run_command`

Parameters

- **cmds** (list(`gevent.Greenlet`)) – Commands to get output for. Defaults to `client.cmds`
- **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_list** (*bool*) – No-op - list of `HostOutput` always returned. Parameter kept for backwards compatibility - to be removed in future releases.

Return type dict or list

join(*output*, *consume_output=False*, *timeout=None*, *encoding='utf-8'*)

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

Parameters

- **output** (*HostOutput* objects) – 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 **all** remote commands are 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. This function's timeout is for all commands in total and will therefor be affected by pool size and total number of concurrent commands in self.pool. Since self.timeout is passed onto each individual SSH session it is **not** used for any parallel functions like *run_command* or *join*.

- **encoding** (*str*) – Encoding to use for output. Must be valid Python codec

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*, *user=None*, *stop_on_errors=True*, *host_args=None*, *use_pty=False*, *shell=None*, *encoding='utf-8'*, *return_list=True*, **args*, ***kwargs*)

2.6.6 BaseSSHClient

API documentation for common single host client functionality.

This class is abstract and contains functions that need to be implemented for each underlying SSH library.

class pssh.clients.base.single.**BaseSSHClient**(*host*, *user=None*, *password=None*, *port=None*, *pkey=None*, *num_retries=3*, *retry_delay=5*, *allow_agent=True*, *timeout=None*, *proxy_host=None*, *_auth_thread_pool=True*, *identity_auth=True*)

auth()

close_channel(*channel*)

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

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

disconnect()

execute(*cmd*, *use_pty=False*, *channel=None*)

get_exit_status(*channel*)

mkdir(*sftp*, *directory*, *_parent_path=None*)

open_session()

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

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*)

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'*)

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

sftp_get(*sftp*, *remote_file*, *local_file*)

sftp_put(*sftp*, *local_file*, *remote_file*)

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

IDENTITIES = ('/home/docs/.ssh/id_rsa', '/home/docs/.ssh/id_dsa', '/home/docs/.ssh/ide

2.6.7 Host Output

Output module of ParallelSSH

class pssh.output.**HostOutput**(*host*, *channel*, *stdout*, *stderr*, *stdin*, *client*, *exception=None*)

Class to hold host output

Parameters

- **host** (*str*) – Host name output is for
- **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
- **client** (`pssh.clients.base_ssh_client.SSHClient`) – *SSHClient* output is coming from.
- **exception** (`Exception` or `None`) – Exception from host if any

channel

client

exception

property exit_code

host

stderr

stdin

stdout

2.6.8 Host Config

Host specific configuration.

class pssh.config.**HostConfig**(*user=None*, *port=None*, *password=None*, *private_key=None*, *allow_agent=None*, *num_retries=None*, *retry_delay=None*, *timeout=None*, *identity_auth=None*, *proxy_host=None*, *keepalive_seconds=None*)

Host configuration for ParallelSSHClient.

Used to hold individual configuration for each host in ParallelSSHClient host list.

Currently only user, port, password and private_key attributes can be configured for backwards compatibility with dictionary host_config implementation in the 1.x.x series.

Parameters

- **user** (*str*) – Username to login as.
- **port** (*int*) – Port number.
- **allow_agent** (*bool*) – Enable/disable SSH agent authentication.
- **num_retries** (*int*) – Number of retry attempts before giving up on connection and SSH operations.
- **retry_delay** (*int*) – Delay in seconds between retry attempts.
- **timeout** (*int*) – Timeout value for connection and SSH sessions in seconds.
- **identity_auth** (*bool*) – Enable/disable identity file authentication under user's home directory (~/.ssh).
- **proxy_host** (*str*) – Proxy SSH host to use for connecting to target SSH host. client -> proxy_host -> SSH host
- **keepalive_seconds** (*int*) – Seconds between keepalive packets being sent. 0 to disable.

Password Password to login with.

Private key Private key file to use for authentication.

allow_agent

identity_auth

keepalive_seconds

num_retries

password

port

private_key

proxy_host

retry_delay

timeout

user

2.6.9 Native Tunnel

Note this module is only intended for use as a proxy host for *ParallelSSHClient*. It will very likely need sub-classing and further enhancing to be used for other purposes.

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.

cleanup()

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

2.6.10 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

2.6.11 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)

2.7 Change Log

2.7.1 2.0.0

Changes

See Upgrading to API 2.0 for examples of code that will need updating.

- Removed paramiko clients and dependency.
- `ParallelSSHClient.run_command` now always returns a list of `HostOutput` - `return_list` argument is a no-op and may be removed.
- `ParallelSSHClient.get_last_output` now always returns a list of `HostOutput`.
- `SSHClient.run_command` now returns `HostOutput`.
- Removed deprecated since *1.0.0* `HostOutput` dictionary attributes.
- Removed deprecated since *1.0.0* imports and modules.
- Removed paramiko based `load_private_key` and `read_openssh_config` functions from `pssh.utils`.
- Removed paramiko based `pssh.tunnel`.
- Removed paramiko based `pssh.agent`.
- Removed deprecated `ParallelSSHClient.get_output` function.
- Removed deprecated `ParallelSSHClient.get_exit_code` and `get_exit_codes` functions.
- Removed deprecated `ParallelSSHClient host_config` dictionary implementation - now list of `HostConfig`.
- Removed `HostOutput.cmd` attribute.
- Removed `ParallelSSHClient.host_clients` attribute.

Fixes

- Removed now unnecessary locking around SSHClient initialisation so it can be parallelised - #219.
- `ParallelSSHClient.join` with encoding would not pass on encoding when reading from output buffers - #214.

2.7.2 1.13.0

Changes

- Added `pssh.config.HostConfig` for providing per-host configuration. Replaces dictionary `host_config` which is now deprecated. See per-host configuration documentation.
- `ParallelSSHClient.scp_send` and `scp_recv` with directory target path will now copy source file to directory keeping existing name instead of failing when recurse is off - #183.
- `pssh.clients.ssh.SSHClient wait_finished` timeout is now separate from `SSHClient(timeout=<timeout>)` session timeout.

- `ParallelSSHClient.join` with timeout now has finished and unfinished commands as `Timeout` exception arguments for use by client code.

Fixes

- `ParallelSSHClient.copy_file` with recurse enabled and absolute destination path would create empty directory in home directory of user - #197.
- `ParallelSSHClient.copy_file` and `scp_recv` with recurse enabled would not create remote directories when copying empty local directories.
- `ParallelSSHClient.scp_send` would require SFTP when recurse is off and remote destination path contains directory - #157.
- `ParallelSSHClient.scp_recv` could block infinitely on large - 200-300MB or more - files.
- `SSHClient.wait_finished` would not apply timeout value given.

2.7.3 1.12.1

Fixes

- Reading from output streams with timeout via *run_command(<..>, timeout=<timeout>)* would raise timeout early when trying to read from a stream with no data written to it while other streams have pending data - #180.

2.7.4 1.12.0

Changes

- Added *ssh-python* (libssh) based native client with *run_command* implementation.
- `ParallelSSHClient.join` with timeout no longer consumes output by default to allow reading of output after timeout.

Fixes

- `ParallelSSHClient.join` with timeout would raise `Timeout` before value given when client was busy with other commands.

Note: `ssh-python` client at *pssh.clients.ssh.ParallelSSHClient* is available for testing. Please report any issues.

To use:

```
from pssh.clients.ssh import ParallelSSHClient
```

This release adds (yet another) client, this one based on ssh-python (libssh). Key features of this client are more supported authentication methods compared to *ssh2-python*.

Future releases will also enable certificate authentication for the ssh-python client.

Please migrate to one of the two native clients if have not already as paramiko is very quickly accumulating yet more bugs and the *2.0.0* release which removes it is imminent.

Users that require paramiko for any reason can pin their parallel-ssh versions to *parallel-ssh<2.0.0*.

2.7.5 1.11.2

Fixes

- *ParallelSSHClient* going out of scope would cause new client sessions to fail if *client.join* was not called prior - #200

2.7.6 1.11.0

Changes

- Moved polling to gevent.select.poll to increase performance and better handle high number of sockets - #189
- `HostOutput.exit_code` is now a dynamic property returning either `None` when exit code not ready or the exit code as reported by channel. `ParallelSSHClient.get_exit_codes` is now a no-op and scheduled to be removed.
- Native client exit codes are now more explicit and return `None` if no exit code is ready. Would previously return `0` by default.

Packaging

- Removed OSX Python 3.6 and 3.7 wheels. OSX wheels for brew python, currently 3.8, on OSX 10.14 and 10.15 are provided.

Fixes

- Native client would fail on opening sockets with large file descriptor values - #189

2.7.7 1.10.0

Changes

- Added `return_list` optional argument to `run_command` to return list of `HostOutput` objects as output rather than dictionary - defaults to `False`. List output will become default starting from `2.0.0`.
- Updated native clients for new version of `ssh2-python`.
- Manylinux 2010 wheels.

Fixes

- Sockets would not be closed on client going out of scope - #175
- Calling `join()` would reset encoding set on `run_command` - #159

2.7.8 1.9.1

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.

2.7.9 1.9.0

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.

2.7.10 1.8.2

Fixes

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

2.7.11 1.8.1

Fixes

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

Packaging

- Added Windows 3.7 wheels

2.7.12 1.8.0

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.

2.7.13 1.7.0

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`.

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.

2.7.14 1.6.3

Changes

- Re-generated C code with latest Cython release.

Fixes

- `ssh2-python` >= 0.14.0 support.

2.7.15 1.6.2

Fixes

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

2.7.16 1.6.1

Fixes

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

2.7.17 1.6.0

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.

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.

2.7.18 1.5.5

Fixes

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

2.7.19 1.5.4

Changes

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

2.7.20 1.5.2

Changes

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

2.7.21 1.5.1

Fixes

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

2.7.22 1.5.0

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.

Fixes

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

2.7.23 1.4.0

Changes

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

Fixes

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

2.7.24 1.3.2

Fixes

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

2.7.25 1.3.1

Changes

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

Fixes

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

2.7.26 1.3.0

Changes

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

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

2.7.27 1.2.1

Fixes

- PyPy builds

2.7.28 1.2.0

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

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

2.7.29 1.1.1

Changes

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

2.7.30 1.1.0

Changes

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

2.7.31 1.0.0

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

2.8 Upgrading to API 2.0

Here can be found code examples for the `1.x` API and how they can be migrated to the new `2.x` API.

Code that was already making use of `run_command(<..>, return_list=True` is compatible with the `2.x` API - `return_list=True` parameter may now be removed.

2.8.1 Parallel Client Run Command

1.x code

```
client = ParallelSSHClient(..)

output = client.run_command(<cmd>)
for host, host_out in output.values():
    <..>
```

2.x code

```
client = ParallelSSHClient(..)

output = client.run_command(<cmd>)
for host_out in output:
    host = host_out.host
    <..>
```

2.8.2 Parallel Client Get last output

1.x code

```
client = ParallelSSHClient(..)

output = client.get_last_output()
for host, host_out in output.values():
    <..>
```

2.x code

```
client = ParallelSSHClient(..)

output = client.get_last_output()
for host_out in output:
    host = host_out.host
    <..>
```

2.8.3 Single Client Run Command

1.x code

```
client = SSHClient(..)

channel, host, stdout, stderr, stdin = client.run_command(<cmd>)
for line in stdout:
    <..>
exit_code = client.get_exit_status(channel)
```

2.x code

```
client = SSHClient(..)

host_out = client.run_command(<cmd>)
for line in host_out.stdout:
    <..>
exit_code = host_out.exit_code
```

2.9 Indices and tables

- genindex

PYTHON MODULE INDEX

p

INDEX

S

T

U

W