pysftp Documentation

Release 0.2.9

Jeff Hinrichs

February 15, 2017

Contents

A simple interface to sftp. based on zeth's ssh.py

CHAPTER 1

Example

```
import pysftp

with pysftp.Connection('hostname', username='me', password='secret') as sftp:
    with sftp.cd('public'):             # temporarily chdir to public
        sftp.put('/my/local/filename')  # upload file to public/ on remote
        sftp.get('remote_file')         # get a remote file
```

CHAPTER 2

Supports

Tested on Python 2.7, 3.2, 3.3, 3.4

CHAPTER 3

Additional Information

- Project: https://bitbucket.org/dundeemt/pysftp
- Download: https://pypi.python.org/pypi/pysftp
- Documentation: https://pysftp.rtfd.org/
- License: BSD

CHAPTER 4

requirements

paramiko >= 1.15.2

Contents:

Cook Book

While in many ways, pysftp is just a thin wrapper over paramiko's SFTPClient, there are a number of ways that we make it more productive and easier to accomplish common, higher-level tasks. The following snippets show where we add value to this great module. See the API docs for a complete listing.

pysftp.Connection()

The Connection object is the base of pysftp. It supports connections via username and password.

```
import pysftp
sftp = pysftp.Connection('hostname', username='me', password='secret')
#
# ... do sftp operations
#
sftp.close()    # close your connection to hostname
```

The Connection object is also context aware so you can use it with a `with` statement.

```
import pysftp
with pysftp.Connection('hostname', username='me', password='secret') as sftp:
    #
    # ... do sftp operations
    #
# connection closed automatically at the end of the with-block
```

Want to use an RSA or DSA key pair, that is simple too.

```
import pysftp
with pysftp.Connection('hostname', username='me', private_key='/path/to/keyfile') as sftp:
    #
    # ... do sftp operations
    #
```

If you key is password protected, just add `private_key_pass` to the argument list.

How about a `paramiko.AgentKey` ? no problem, just set the private_key equal to it.

```
import pysftp
with pysftp.Connection('hostname', username='me', private_key=my_agentkey) as sftp:
    #
    # ... do sftp operations
    #
```

The connection object also allows you to use an IP Address for the `host` and you can set the `port` which defaults to 22, as well.

pysftp.CnOpts

You can also specify additional connection options using the pysftp.CnOpts object. These options are advanced and not applicable to most uses, because of this they have been segmented from the Connection parameter list and made available via CnOpts obj/parameter.

Host Key checking is enabled by default. It will use `~/.ssh/known_hosts` by default. If you wish to disable host key checking (NOT ADVISED) you will need to modify the default CnOpts and set the .hostkeys to None.

```
import pysftp
cnopts = pysftp.CnOpts()
cnopts.hostkeys = None
with pysftp.Connection('host', username='me', password='pass', cnopts=cnopts):
    # do stuff here
```

To use a completely different known_hosts file, you can override CnOpts looking for `~/.ssh/known_hosts` by specifying the file when instantiating.

```
import pysftp
cnopts = pysftp.CnOpts(knownhosts='path/to/your/knownhostsfile')
cnopts.hostkeys = None
with pysftp.Connection('host', username='me', password='pass', cnopts=cnopts):
    # do stuff here
```

If you wish to use `~/.ssh/known_hosts` but add additional known host keys you can merge with update additional known_host format files by using .load method.

```
import pysftp
cnopts = pysftp.CnOpts()
cnopts.hostkeys.load('path/to/your/extra_knownhosts')
with pysftp.Connection('host', username='me', password='pass', cnopts=cnopts):
    # do stuff here
```

For both the knownhost parameter and the load argument, pysftp expands user, so you can use tilde notation in your pathing.

OTHER AVAILABLE CONNECTION OPTIONS via CnOpts:

- .log - replaces the log parameter in the Connection method
- .compression - False (Default) no compression, True - enable compression
- .ciphers - replaces the ciphers parameter in the Connection method.
- log and ciphers in the Connection parameter list are deprecated and will be removed in version 0.3.0 Use the CnOpts to specify them.

Here is a common scenario, you have your connection information stored in a persistence mechanism, like yamjam and when you access it, it is returned in dictionary form. `{'host':'myhost', username:'me', ...}` Just send the dict into the connection object like so:

```
import pysftp
cinfo = {'host':'hostname', 'username':'me', 'password':'secret', 'port':2222}
with pysftp.Connection(**cinfo) as sftp:
    #
    # ... do sftp operations
    #
```

pysftp.Connection.get()

In addition to the normal paramiko call, you can optionally set the `preserve_mtime` parameter to `True` and the operation will make sure that the modification times on the local copy match those on the server.

```
# ...
sftp.get('myfile', preserve_mtime=True)
```

pysftp.Connection.get_d()

This pysftp method is an abstraction above *get()* that allows you to copy all the files in a remote directory to a local path.

```
# copy all files under public to a local path, preserving modification time
sftp.get_d('public', 'local-backup', preserve_mtime=True)
```

pysftp.Connection.get_r()

This pysftp method is an abstraction that recursively copies files *and* directories from the remote to a local path.

```
# copy all files AND directories under public to a local path
sftp.get_r('public', 'local-backup', preserve_mtime=True)
```

pysftp.Connection.put()

In addition to the normal paramiko call, you can optionally set the `preserve_mtime` parameter to `True` and the operation will make sure that the modification times on the server copy match those on the local.

```
# copy myfile, to the current working directory on the server, preserving modification time
sftp.put('myfile', preserve_mtime=True)
```

pysftp.Connection.put_d()

The opposite of *get_d()*, put_d allows you to copy the contents of a local directory to a remote one via SFTP.

```
# copy files from images, to remote static/images directory, preserving modification time
sftp.put_d('images', 'static/images', preserve_mtime=True)
```

pysftp.Connection.put_r()

This method copies all files *and* directories from a local path to a remote path. It creates directories, and happily succeeds even if the target directories already exist.

```
# recursively copy files and directories from local static, to remote static,
# preserving modification times on the files
sftp.put_r('static', 'static', preserve_mtime=True)
```

pysftp.Connection.cd()

This method is a with-context capable version of *chdir()*. Restoring the original directory when the `with` statement goes out of scope. It can be called with a remote directory to temporarily change to

```
with sftp.cd('static'):     # now in ./static
    sftp.chdir('here')      # now in ./static/here
    sftp.chdir('there')     # now in ./static/here/there
# now back to the original current working directory
```

Or it can be called without a remote directory to just act as a bookmark you want to return to later.

```
with sftp.cd():             # still in .
    sftp.chdir('static')    # now in ./static
    sftp.chdir('here')      # now in ./static/here
# now back to the original current working directory
```

pysftp.Connection.chmod()

chmod() is a wrapper around paramiko's except for the fact it will takes an integer representation of the octal mode. No leading 0 or 0o wanted. We know it's suppose to be an octal, but who really remembers that?

This way it is just like a command line `chmod 644 readme.txt`

```
user group other
rwx  rwx   rwx
421  421   421

user  - read/write = 4+2 = 6
group - read       = 4   = 4
other - read       = 4   = 4
```

```
sftp.chmod('readme.txt', 644)
```

pysftp.st_mode_to_int()

converts an octal mode result back to an integer representation. The .st_mode information returned in SFTPAttribute object .stat(*fname*).st_mode contains extra things you probably don't care about, in a form that has been converted from octal to int so you won't recognize it at first. This function clips the extra bits and hands you the file mode bits in a way you'll recognize.

```
>>> attr = sftp.stat('readme.txt')
>>> attr.st_mode
33188
>>> pysftp.st_mode_to_int(attr.st_mode)
644
```

pysftp.Connection.chown()

pysftp's method allows you to specify just, gid or the uid or both. If either gid or uid is None *(default)*, then pysftp does a stat to get the current ids and uses that to fill in the missing parameter because the underlying paramiko method requires that you explicitly set both.

NOTE uid and gid are integers and relative to each system. Just because you are uid 102 on your local system, a uid of 102 on the remote system most likely won't be your login. You will need to do some homework to make sure that you are setting these values as you intended.

pysftp.Connection.cwd()

cwd() is a synonym for *chdir()*. Its purpose is to make transposing hand typed commands at an sftp command line into those used by pysftp, easier to do.

```
...
sftp.cwd('public')  # is equivalent to sftp.chdir('public')
```

pysftp.Connection.pwd

Returns the current working directory. It returns the result of *.normalize('.')* but makes your code and intention easier to read. Paramiko has a method, *getcwd()*, that we expose, but that method returns `None` if *chdir()* has not been called prior.

```
...
>>> print(sftp.getcwd())
None
>>> sftp.pwd
u'/home/test'
```

pysftp.Connection.listdir()

The difference here, is that pysftp's version returns a sorted list instead of paramiko's arbitrary order. Sorted by filename.

```
...
>>> sftp.listdir()
[u'pub', u'readme.sym', u'readme.txt']
```

pysftp.Connection.listdir_attr()

The difference here, is that pysftp's version returns a sorted list instead of paramiko's arbitrary order. Sorted by SFTPAttribute.filename.

```
...
>>> for attr in sftp.listdir_attr():
...     print attr.filename, attr
...
pub dr-xrwxr-x   1 501      502             5 19 May 23:22 pub
readme.sym lrwxr-xr-x   1 501      502            10 21 May 23:29 readme.sym
readme.txt -r--r--r--   1 501      502          8192 26 May 23:32 readme.txt
```

pysftp.Connection.makedirs()

A common scenario where you need to create all directories in a path as needed, setting their mode, if created. Takes a mode argument, just like *chmod()*, that is an integer representation of the mode you want.

```
...
sftp.makedirs('pub/show/off')  # will happily make all non-existing directories
```

pysftp.Connection.mkdir()

Just like *chmod()*, the mode is an integer representation of the octal number to use. Just like the unix cmd, *chmod* you use 744 not 0744 or 0o744.

```
...
sftp.mkdir('show', mode=644)  # user r/w, group and other read-only
```

pysftp.Connection.isdir()

Does all the busy work of stat'ing and dealing with the stat module returning a simple True/False.

```
...
>>> sftp.isdir('pub')
True
```

pysftp.Connection.isfile()

Does all the busy work of stat'ing and dealing with the stat module returning a simple True/False.

```
...
>>> sftp.isfile('pub')
False
```

pysftp.Connection.readlink()

The underlying paramiko method can return either an absolute or a relative path. pysftp forces this to always be an absolute path by laundering the result with a *.normalize* before returning.

```
...
>>> sftp.readlink('readme.sym')
u'/home/test/readme.txt'
```

pysftp.Connection.exists()

Returns True if a remote entity exists

```
...
>>> sftp.exists('readme.txt')    # a file
True
>>> sftp.exists('pub')           # a dir
True
```

pysftp.Connection.lexists()

Like *exists()*, but returns True for a broken symbolic link

pysftp.Connection.truncate()

Like the underlying .truncate method, by pysftp returns the file's new size after the operation.

```
>>> sftp.truncate('readme.txt', 4096)
4096
```

pysftp.Connection.walktree()

Is a powerful method that can recursively (*default*) walk a **remote** directory structure and calls a user-supplied callback functions for each file, directory or unknown entity it encounters. It is used in the get_x methods of pysftp and can be used with great effect to do your own bidding. Each callback is supplied the pathname of the entity. (form: `func(str)`)

pysftp.Connection.sftp_client

Don't like how we have over-ridden or modified a paramiko method? Use this attribute to get at paramiko's original version. Remember, our goal is to augment not supplant paramiko.

pysftp.path_advance

generator to iterate over a file path

```
...
>>> list(pysftp.path_advance('./pub/example/example01'))
['.', './pub', './pub/example', './pub/example/example01']
```

pysftp.path_retreat

generator to iterate over a file path in reverse

```
...
>>> list(pysftp.path_retreat('./pub/example/example01'))
['./pub/example/example01', './pub/example', './pub', '.']
```

pysftp.reparent

Pythons `os.path.join('backup', '/home/test/pub')` returns '/home/test/pub', but if you want to copy a directory structure to a new path this won't do what you want. But, reparent will.

```
...
 >>> pysftp.reparent('backup', '/home/test/pub')
'backup/./home/test/pub'
```

pysftp.walktree

Is similar to *`pysftp.Connection.walktree()`* except that it walks a **local** directory structure. It has the same callback mechanism.

pysftp.cd

A with-context aware version of `os.chdir` for use on the **local** file system. The yin to *pysftp.Connection.cd()* yang.

```
...
>>> import os
>>> os.getcwd()
'/home/jlh/Projects/pysftp/src'
>>> with pysftp.cd('docs'):
...     print os.getcwd()
...
/home/jlh/Projects/pysftp/src/docs
>>> os.getcwd()
'/home/jlh/Projects/pysftp/src'
```

Remarks

We think paramiko is a great python library and it is the backbone of pysftp. The methods pysftp has created are abstractions that serve a programmer's productivity by encapsulating many of the higher function use cases of interacting with SFTP. Instead of writing your own code to walk directories and call get and put, dealing with not only paramiko but Python's own `os` and `stat` modules and writing tests *(many code snippets on the net are incomplete and don't account for edge cases)* pysftp supplies a complete library for dealing with all three. Leaving you to focus on your primary task.

Paramiko also tries very hard to stay true to Python's `os` module, which means sometimes, things are weird or a bit too low level. We think paramiko's goals are good and don't believe they should change. Those changes are for an abstraction library like pysftp.

API

A friendly Python SFTP interface.

class `pysftp.`**`CnOpts`**(*knownhosts=None*)

additional connection options beyond authentication

Variables

- **`log`** (*`bool`* / *`str`*) – initial value: False - log connection/handshake details? If set to True, pysftp creates a temporary file and logs to that. If set to a valid path and filename, pysftp logs to that. The name of the logfile can be found at `.logfile`
- **`compression`** (*`bool`*) – initial value: False - Enables compression on the transport, if set to True.
- **`ciphers`** (*`list`* / *`None`*) – initial value: None - List of ciphers to use in order.
- **`hostkeys`** (*`paramiko.hostkeys.HostKeys`* / *`None`*) – HostKeys object to use for host key checking.

Parameters **`knownhosts`** (*`filepath`* / *`None`*) – initial value: None - file to load hostkeys. If not specified, uses ~/.ssh/known_hosts

Returns (obj) CnOpts - A connection options object, used for passing extended options to the Connection

Raises **`HostKeysException`** –

get_hostkey(*host*)
return the matching hostkey to use for verification for the host indicated or raise an SSHException

class pysftp.**Connection**(*host*, *username=None*, *private_key=None*, *password=None*, *port=22*, *private_key_pass=None*, *ciphers=None*, *log=False*, *cnopts=None*, *default_path=None*)
Connects and logs into the specified hostname. Arguments that are not given are guessed from the environment.

Parameters

- **host** (*str*) – The Hostname or IP of the remote machine.
- **username** (*str|None*) – *Default: None* - Your username at the remote machine.
- **private_key** (*str|obj|None*) – *Default: None* - path to private key file(str) or paramiko.AgentKey
- **password** (*str|None*) – *Default: None* - Your password at the remote machine.
- **port** (*int*) – *Default: 22* - The SSH port of the remote machine.
- **private_key_pass** (*str|None*) – *Default: None* - password to use, if private_key is encrypted.
- **ciphers** (*list|None*) – *Deprecated* - see pysftp.CnOpts and cnopts parameter
- **log** (*bool|str*) – *Deprecated* - see pysftp.CnOpts and cnopts parameter
- **cnopts** (*None|CnOpts*) – *Default: None* - extra connection options set in a CnOpts object.
- **default_path** (*str|None*) – *Default: None* - set a default path upon connection.

Returns (obj) connection to the requested host

Raises

- **ConnectionException** –
- **CredentialException** –
- **SSHException** –
- **AuthenticationException** –
- **PasswordRequiredException** –
- **HostKeysException** –

active_ciphers
Get tuple of currently used local and remote ciphers.

Returns (tuple of str) currently used ciphers (local_cipher, remote_cipher)

active_compression
Get tuple of currently used local and remote compression.

Returns (tuple of str) currently used compression (local_compression, remote_compression)

cd(*remotepath=None*)
context manager that can change to a optionally specified remote directory and restores the old pwd on exit.

Parameters **remotepath** (*str|None*) – *Default: None* - remotepath to temporarily make the current directory

Returns None

Raises IOError, if remote path doesn't exist

chdir(*remotepath*)
change the current working directory on the remote

Parameters **remotepath** (*str*) – the remote path to change to

Returns None

Raises IOError, if path does not exist

chmod(*remotepath*, *mode=777*)
set the mode of a remotepath to mode, where mode is an integer representation of the octal mode to use.

Parameters

- **remotepath** (*str*) – the remote path/file to modify
- **mode** (*int*) – *Default: 777* - int representation of octal mode for directory

Returns None

Raises IOError, if the file doesn't exist

chown(*remotepath*, *uid=None*, *gid=None*)
set uid and/or gid on a remotepath, you may specify either or both. Unless you have **permission** to do this on the remote server, you will raise an IOError: 13 - permission denied

Parameters

- **remotepath** (*str*) – the remote path/file to modify
- **uid** (*int*) – the user id to set on the remotepath
- **gid** (*int*) – the group id to set on the remotepath

Returns None

Raises IOError, if you don't have permission or the file doesn't exist

close()
Closes the connection and cleans up.

cwd(*remotepath*)
change the current working directory on the remote

Parameters **remotepath** (*str*) – the remote path to change to

Returns None

Raises IOError, if path does not exist

execute(*command*)
Execute the given commands on a remote machine. The command is executed without regard to the remote *pwd*.

Parameters **command** (*str*) – the command to execute.

Returns (list of str) representing the results of the command

Raises Any exception raised by command will be passed through.

exists(*remotepath*)
Test whether a remotepath exists.

Parameters **remotepath** (*str*) – the remote path to verify

Returns (bool) True, if remotepath exists, else False

get(*remotepath*, *localpath=None*, *callback=None*, *preserve_mtime=False*)
Copies a file between the remote host and the local host.

Parameters

- **remotepath** (*str*) – the remote path and filename, source
- **localpath** (*str*) – the local path and filename to copy, destination. If not specified, file is copied to local current working directory
- **callback** (*callable*) – optional callback function (form: `func(int, int)`) that accepts the bytes transferred so far and the total bytes to be transferred.
- **preserve_mtime** (*bool*) – *Default: False* - make the modification time(st_mtime) on the local file match the time on the remote. (st_atime can differ because stat'ing the localfile can/does update it's st_atime)

Returns None

Raises IOError

get_d(*remotedir*, *localdir*, *preserve_mtime=False*)
get the contents of remotedir and write to locadir. (non-recursive)

Parameters

- **remotedir** (*str*) – the remote directory to copy from (source)
- **localdir** (*str*) – the local directory to copy to (target)
- **preserve_mtime** (*bool*) – *Default: False* - preserve modification time on files

Returns None

Raises

get_r(*remotedir*, *localdir*, *preserve_mtime=False*)
recursively copy remotedir structure to localdir

Parameters

- **remotedir** (*str*) – the remote directory to copy from
- **localdir** (*str*) – the local directory to copy to
- **preserve_mtime** (*bool*) – *Default: False* - preserve modification time on files

Returns None

Raises

getcwd()
return the current working directory on the remote. This is a wrapper for paramiko's method and not to be confused with the SFTP command, cwd.

Returns (str) the current remote path. None, if not set.

getfo(*remotepath*, *flo*, *callback=None*)
Copy a remote file (remotepath) to a file-like object, flo.

Parameters

- **remotepath** (*str*) – the remote path and filename, source
- **flo** – open file like object to write, destination.
- **callback** (*callable*) – optional callback function (form: `func(int, int)`) that accepts the bytes transferred so far and the total bytes to be transferred.

Returns (int) the number of bytes written to the opened file object

Raises Any exception raised by operations will be passed through.

isdir(*remotepath*)
return true, if remotepath is a directory

Parameters **remotepath** (*str*) – the path to test

Returns (bool)

isfile(*remotepath*)
return true if remotepath is a file

Parameters **remotepath** (*str*) – the path to test

Returns (bool)

lexists(*remotepath*)
Test whether a remotepath exists. Returns True for broken symbolic links

Parameters **remotepath** (*str*) – the remote path to verify

Returns (bool), True, if lexists, else False

listdir(*remotepath='.'*)
return a list of files/directories for the given remote path. Unlike, paramiko, the directory listing is sorted.

Parameters **remotepath** (*str*) – path to list on the server

Returns (list of str) directory entries, sorted

listdir_attr(*remotepath='.'*)
return a list of SFTPAttribute objects of the files/directories for the given remote path. The list is in arbitrary order. It does not include the special entries '.' and '..'.

The returned SFTPAttributes objects will each have an additional field: longname, which may contain a formatted string of the file's attributes, in unix format. The content of this string will depend on the SFTP server.

Parameters **remotepath** (*str*) – path to list on the server

Returns (list of SFTPAttributes), sorted

logfile
return the name of the file used for logging or False it not logging

Returns (str)logfile or (bool) False

lstat(*remotepath*)
return information about file/directory for the given remote path, without following symbolic links. Otherwise, the same as .stat()

Parameters **remotepath** (*str*) – path to stat

Returns (obj) SFTPAttributes object

makedirs(*remotedir*, *mode=777*)
create all directories in remotedir as needed, setting their mode to mode, if created.

If remotedir already exists, silently complete. If a regular file is in the way, raise an exception.

Parameters

- **remotedir** (*str*) – the directory structure to create
- **mode** (*int*) – *Default: 777* - int representation of octal mode for directory

Returns None

Raises OSError

mkdir(*remotepath*, *mode=777*)
Create a directory named remotepath with mode. On some systems, mode is ignored. Where it is used, the current umask value is first masked out.

Parameters

- **remotepath** (*str*) – directory to create‘
- **mode** (*int*) – *Default: 777* - int representation of octal mode for directory

Returns None

normalize(*remotepath*)
Return the expanded path, w.r.t the server, of a given path. This can be used to resolve symlinks or determine what the server believes to be the *pwd*, by passing ‘.’ as remotepath.

Parameters **remotepath** (*str*) – path to be normalized

Returns (str) normalized form of the given path

Raises IOError, if remotepath can’t be resolved

open(*remote_file*, *mode='r'*, *bufsize=-1*)
Open a file on the remote server.

See http://paramiko-docs.readthedocs.org/en/latest/api/sftp.html for details.

Parameters

- **remote_file** (*str*) – name of the file to open.
- **mode** (*str*) – mode (Python-style) to open file (always assumed binary)
- **bufsize** (*int*) – *Default: -1* - desired buffering

Returns (obj) SFTPFile, a handle the remote open file

Raises IOError, if the file could not be opened.

put(*localpath*, *remotepath=None*, *callback=None*, *confirm=True*, *preserve_mtime=False*)
Copies a file between the local host and the remote host.

Parameters

- **localpath** (*str*) – the local path and filename
- **remotepath** (*str*) – the remote path, else the remote *pwd* and filename is used.
- **callback** (*callable*) – optional callback function (form: `func(int, int)`) that accepts the bytes transferred so far and the total bytes to be transferred.
- **confirm** (*bool*) – whether to do a stat() on the file afterwards to confirm the file size
- **preserve_mtime** (*bool*) – *Default: False* - make the modification time(st_mtime) on the remote file match the time on the local. (st_atime can differ because stat’ing the localfile can/does update it’s st_atime)

Returns (obj) SFTPAttributes containing attributes about the given file

Raises

- **IOError** – if remotepath doesn’t exist
- **OSError** – if localpath doesn’t exist

put_d (*localpath*, *remotepath*, *confirm=True*, *preserve_mtime=False*)
Copies a local directory's contents to a remotepath

Parameters

- **localpath** (*str*) – the local path to copy (source)
- **remotepath** (*str*) – the remote path to copy to (target)
- **confirm** (*bool*) – whether to do a stat() on the file afterwards to confirm the file size
- **preserve_mtime** (*bool*) – *Default: False* - make the modification time(st_mtime) on the remote file match the time on the local. (st_atime can differ because stat'ing the localfile can/does update it's st_atime)

Returns None

Raises

- **IOError** – if remotepath doesn't exist
- **OSError** – if localpath doesn't exist

put_r (*localpath*, *remotepath*, *confirm=True*, *preserve_mtime=False*)
Recursively copies a local directory's contents to a remotepath

Parameters

- **localpath** (*str*) – the local path to copy (source)
- **remotepath** (*str*) – the remote path to copy to (target)
- **confirm** (*bool*) – whether to do a stat() on the file afterwards to confirm the file size
- **preserve_mtime** (*bool*) – *Default: False* - make the modification time(st_mtime) on the remote file match the time on the local. (st_atime can differ because stat'ing the localfile can/does update it's st_atime)

Returns None

Raises

- **IOError** – if remotepath doesn't exist
- **OSError** – if localpath doesn't exist

putfo (*flo*, *remotepath=None*, *file_size=0*, *callback=None*, *confirm=True*)
Copies the contents of a file like object to remotepath.

Parameters

- **flo** – a file-like object that supports .read()
- **remotepath** (*str*) – the remote path.
- **file_size** (*int*) – the size of flo, if not given the second param passed to the callback function will always be 0.
- **callback** (*callable*) – optional callback function (form: `func(int, int)`) that accepts the bytes transferred so far and the total bytes to be transferred.
- **confirm** (*bool*) – whether to do a stat() on the file afterwards to confirm the file size

Returns (obj) SFTPAttributes containing attributes about the given file

Raises TypeError, if remotepath not specified, any underlying error

pwd
return the current working directory

Returns (str) current working directory

readlink (*remotelink*)
Return the target of a symlink (shortcut). The result will be an absolute pathname.

Parameters **remotelink** (*str*) – remote path of the symlink

Returns (str) absolute path to target

remote_server_key
return the remote server's key

remove (*remotefile*)
remove the file @ remotefile, remotefile may include a path, if no path, then *pwd* is used. This method only works on files

Parameters **remotefile** (*str*) – the remote file to delete

Returns None

Raises IOError

rename (*remote_src*, *remote_dest*)
rename a file or directory on the remote host.

Parameters

- **remote_src** (*str*) – the remote file/directory to rename
- **remote_dest** (*str*) – the remote file/directory to put it

Returns None

Raises IOError

rmdir (*remotepath*)
remove remote directory

Parameters **remotepath** (*str*) – the remote directory to remove

Returns None

security_options
return the available security options recognized by paramiko.

Returns (obj) security preferences of the ssh transport. These are tuples of acceptable *.ciphers*, *.digests*, *.key_types*, and key exchange algorithms *.kex*, listed in order of preference.

sftp_client
give access to the underlying, connected paramiko SFTPClient object

see http://paramiko-docs.readthedocs.org/en/latest/api/sftp.html

Params None

Returns (obj) the active SFTPClient object

stat (*remotepath*)
return information about file/directory for the given remote path

Parameters **remotepath** (*str*) – path to stat

Returns (obj) SFTPAttributes

symlink (*remote_src*, *remote_dest*)
create a symlink for a remote file on the server

Parameters

- **remote_src** (*str*) – path of original file
- **remote_dest** (*str*) – path of the created symlink

Returns None

Raises any underlying error, IOError if something already exists at remote_dest

timeout

(float|None) ***Default: None*** **-** get or set the underlying socket timeout for pending read/write ops.

Returns (float|None) seconds to wait for a pending read/write operation before raising socket.timeout, or None for no timeout

truncate (*remotepath*, *size*)
Change the size of the file specified by path. Used to modify the size of the file, just like the truncate method on Python file objects. The new file size is confirmed and returned.

Parameters

- **remotepath** (*str*) – remote file path to modify
- **size** (*int | long*) – the new file size

Returns (int) new size of file

Raises IOError, if file does not exist

unlink (*remotefile*)
remove the file @ remotefile, remotefile may include a path, if no path, then *pwd* is used. This method only works on files

Parameters **remotefile** (*str*) – the remote file to delete

Returns None

Raises IOError

walktree (*remotepath*, *fcallback*, *dcallback*, *ucallback*, *recurse=True*)
recursively descend, depth first, the directory tree rooted at remotepath, calling discreet callback functions for each regular file, directory and unknown file type.

Parameters

- **remotepath** (*str*) – root of remote directory to descend, use '.' to start at *pwd*
- **fcallback** (*callable*) – callback function to invoke for a regular file. (form: `func(str)`)
- **dcallback** (*callable*) – callback function to invoke for a directory. (form: `func(str)`)
- **ucallback** (*callable*) – callback function to invoke for an unknown file type. (form: `func(str)`)
- **recurse** (*bool*) – *Default: True* - should it recurse

Returns None

Raises

SFTPAttributes

see http://paramiko-docs.readthedocs.org/en/latest/api/sftp.html?highlight=sftpattributes#paramiko.sftp_attr.SFTPAttributes for details

SFTPFile

see http://paramiko-docs.readthedocs.org/en/latest/api/sftp.html?highlight=paramiko.sftp_file.sftpfile#paramiko.sftp_file.SFTPFile for details

SecurityOptions

a simple object returned with available Security Options

see http://paramiko-docs.readthedocs.org/en/latest/api/transport.html?highlight=ciphers#paramiko.transport.SecurityOptions for details

Change Log

- 0.2.9 (current, released 2016-07-04)
  - bugfix: correctly implement hostcheck. Now, be default pysftp will verify the host. See `pysftp.CnOpts.hostkeys`
  - added *`pysftp.Connection.remote_server_key()`* - used to retrieve the remote hosts server key.
  - added support for enabling compression, `compression` (J. Kruth)
  - added *`active_compression`*, to return the active local and remote compression settings as a tuple
  - fixed an unwanted logging side-effect, after you set logging, it would remain, even if you closed the .Connection and couldn't be changed to something else. Now when Connection closes, any logging handlers are closed and can be changed to something else upon the next .Connection
  - moved `log` parameter of Connection to the new CnOpts connection options object, deprecated the existing `log` parameter, will be removed in 0.3.0
  - modified `pysftp.Conection.walktree()` to always use posixpath conventions when walking a remote directory per the latest draft-ietf-secsh-filexfer-13.txt. Issue encountered with windows clients (#60)
  - modified `pysftp.reparent()` to handle mis-matched pathing, i.e. windows -> posix, better (#61)
- 0.2.8 (released 2014-05-28)
  - created `pysftp.walktree()` for walking local directories
  - added param recurse to *`pysftp.Connection.walktree()`* to allow it to do another trick
  - created *`put_d()`* to put the contents of a local directory to a remote one
  - created a context manager chdir method, *`pysftp.Connection.cd()`*
  - created *`put_r()`* to recursively put the contents of a local directory to a remote one
  - fixed a bug with `st_mode_to_int()` on py3 (#52)
  - *`listdir_attr()`* now returns a sorted list, sorted on filename

  - created `pysftp.cd()` with-context version of `os.chdir` for local directories
  - created docs, cookbook to show off some of the notable features of pysftp
- 0.2.7 (released 2014-05-24)
  - created *`pysftp.Connection.walktree()`*, recursively walk, depth first, a remote directory structure. Used as the base of *`get_r()`*. See tests/test_walktree.py for examples.
  - added *`unlink()`* as synonym for *`remove()`*
  - added *`normalize()`*
  - created *`get_r()`* to recursively copy remote directories to a local path
  - created *`pwd`* to return the current working directory
  - created *`cwd()`* as synonym for *`chdir()`*
  - modified *`listdir()`* to return a sorted list instead of an arbitrary one
  - added *`readlink()`*, always returns an absolute path
  - created *`get_d()`* to copy the remote directory to a local path (non-recursive)
  - added *`timeout`* to set the read/write timeout of the underlying channel for pending read/write ops
  - added *`listdir_attr()`*, wrapper for paramiko method
  - added *`truncate()`*, method returns the new file size
  - improved DRY'ness of test suite
- 0.2.6 (released 2014-05-17)
  - added `preserve_mtime` parameter to *`put()`*, optionally updates the remote file's st_mtime to match the local file.
  - added `preserve_mtime` parameter to *`get()`*, optionally updates the local file's st_mtime to match the remote file
  - added *`exists()`* and *`lexists()`*, use *`stat()`* and *`lstat()`* respectively
  - added *`symlink()`*
  - created *`isdir()`*, *`isfile()`*, *`makedirs()`*
  - added *`chmod()`*
  - added *`chown()`*
  - added *`sftp_client`* which exposes underlying, active `SFTPClient` object for advance use
- 0.2.5 (released 2014-05-15)
  - added `ciphers` parameter to *`Connection`* object (D. Reilly)
  - added *`active_ciphers`* to return local and remote cipher in use
  - added *`security_options`*, where you can get available ciphers, among other information
  - enhanced logging, and added documentation and tests
- 0.2.4 (released 2014-05-13)
  - *`Connection`* can be used in a `with` statement
  - add *`remove()`*
  - added support for callback and confirm params to *`put()`*

  - added support for callback on *get()*
  - added support for *open()*
  - fixed password bug and now differentiates between an empty string and None
  - added support for `paramiko.AgentKey` to be passed in as the `private_key` for Connection
  - added support for *mkdir()*
  - added support for *rmdir()*
  - added support for *stat()* and *lstat()*
  - added helper function, `st_mode_to_int()`,to convert the `st_mode` value back into a common integer representation
  - added *getfo()*
  - added *putfo()*
- 0.2.3 (released 2014-05-10)
  - host code on pypi to keep pip happy
  - move code to bitbucket
  - enhance testing
  - README.rst and LICENSE named properly
  - cleaner error handling
- 0.2.2
  - additions
    * chdir(self, path) - change the current working directory on the remote
    * getcwd(self) - return the current working directory on the remote
    * listdir(self, path='.')return a list of files for the given path

Contributing

You can contribute to the project in a number of ways. Code is always good, bugs are interesting but tests make your famous!

Bug reports or feature enhancements that include a test are given preferential treatment. So instead of voting for an issue, write a test.

Code

1. Fork the repository on Bitbucket .
2. Install supporting software packages and pysftp in –editable mode
   (a) Make a virtualenv, clone the repos, install the deps from *pip install -r requirements-dev.txt*
   (b) Install pysftp in editable mode, *pip install -e .*
3. Write any new tests needed and ensure existing tests continue to pass without modification.

1. Setup CI testing on drone.io for your Fork. See current script .

2. As of verion 0.2.9, no tests are run against a public SFTP server. Keeping a server available was/is problematic and made for brittle testing. Becuase of this, we now use the pytest-sftpserver plugin for many tests. Testing features the concern authentication or authorization, i.e. different login types, chmod, chown have to be run against a local sshd and not the plugin as it does NOT support these types of tests.
3. You will need to setup an ssh daemon on your local machine and create a user: test with password of test1357 – Tests that can only be run locally are skipped using the `@skip_if_ci decorator` so they don't fail when the test suite is run on the CI server.
4. Ensure that your name is added to the end of the Authors file using the format Name <email@domain.com> (url), where the (url) portion is optional.
5. Submit a Pull Request to the project on Bitbucket.

Docs

We use sphinx to build the docs. `make html` is your friend, see docstrings for details on params, etc.

Bug Reports

If you encounter a bug or some surprising behavior, please file an issue on our tracker

Issue Priorities

This section lists the priority that will be assigned to an issue.

1. Developer Issues
2. Issues that have a pull request with a test(s) displaying the issue and code change(s) that satisfies the test suite
3. Issues that have a pull request with a test(s) displaying the issue
4. Naked pull requests - a code change request with no accompaning test
5. An issue without a pull request with a test displaying the issue
6. badly documented issue with no code or test - pysftp is not an end-user tool, it is a developer tool and it is expected that issues will be submitted like a developer and not an end-user. Issues in the realm of "the internet is broke" will be marked as invalid with a comment pointing the submitter to this section.

Testing

Tests specific to an issue should be put in the tests/ directory and the module should be named test_issue_xx.py The tests within that module should be named test_issue_xx or test_issue_xx_YYYYYY if more than one test. Pull requests should not modify existing tests. See tests/test_issue_xx.py for a template and further explanation.

Authors

Contributors of code, tests and documentation to the project who have agreed to have their work enjoined into the project and project license (BSD).

- Jeff Hinrichs <dundeemt@gmail.com>, http://inre.dundeemt.com/
- Don Reilly <dreilly1982@gmail.com>, http://github.com/dreilly1982

- James Kruth <james@kruth.org>, http://james.kruth.org/
- Nick Robinson-Wall

Acknowledgment

- pysftp is a fork, with permission, of ssh.py, originally authored by Zeth @ http://zeth.net/archive/2008/05/29/sftp-python-really-simple-ssh/
- paramiko - http://paramiko-docs.readthedocs.org/
- Justin Reiners - for support of the project by the donation of a SFTP server for testing

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

p