sshcon

Release 0.2.3

racoonx2p

Oct 29, 2020

CONTENTS:

1 Installation **3**

2 API Documentation **5**
- 2.1 sshcon.main module 5
- 2.2 sshcon.exceptions module 8

3 How to use **9**

4 Indices and tables **11**

5 Links **13**

Python Module Index **15**

Index **17**

Python SSH connector **sshcon** for linux systems based on super fast SSH2 protocol library -> ssh2-python.

CHAPTER

ONE

INSTALLATION

Use pip to install:

```
pip install sshcon
```

Only dependency is great package by Panos Kittenis called ssh2-python.

CHAPTER

TWO

API DOCUMENTATION

2.1 sshcon.main module

Python SSH connector/wrapper for linux systems based on super fast SSH2 protocol library -> ssh2-python

class `sshcon.main.`**`CompletedCommand`**(*rcode: int*, *stdout: Union[str, bytes]*, *stderr: str*)

Bases: `tuple`

Class to represent ssh connection.

Parameters **`NamedTuple`** (*`rcode, stdout, stderr`*) – Constructs all the necessary attributes for the CompletedCommand object.

property rcode

Alias for field number 0

property stderr

Alias for field number 2

property stdout

Alias for field number 1

class `sshcon.main.`**`SshCon`**(*host: str*, *user: str*, *key: Union[pathlib.Path, str]*, *port: int = 22*)

Bases: `object`

A class to represent ssh connection.

chmod(*path: Union[pathlib.Path, str]*, *mode: int*, *recursive: bool = False*) → None

Change file mode bits in remote machine.

Parameters

- **path** (*`Union[Path, str]`*) – Path to a target.
- **mode** (*`int`*) – File mode bits.
- **recursive** (*`bool, optional`*) – Change file mod recursively. Defaults to False.

chown(*path: Union[pathlib.Path, str]*, *owner: str*, *group: str*, *recursive: bool = False*) → None

Change file owner and group in remote machine.

Parameters

- **path** (*`Union[Path, str]`*) – Path to a target.
- **owner** (*`str`*) – Username of an owner.
- **group** (*`str`*) – Group to use.
- **recursive** (*`bool, optional`*) – Change owner/group mod recursively. Defaults to False.

get_file (*file: Union[pathlib.Path, str]*, *destination: Union[pathlib.Path, str]*, *force: bool = False*) → None

Get remote file from a remote location.

Parameters

- **file** (*Union[Path, str]*) – File to get from a remote.
- **destination** (*Union[Path, str]*) – Local destination.
- **force** (*bool*) – Rewrite the file, if exists. Defaults to False

Raises **IsADirectoryError** – Raises if file is a directory.

:raises FileNotFoundError:: Raises if remote file not found.:

get_filemode (*path: Union[pathlib.Path, str]*)

Get file status.

Parameters **path** (*Union[Path, str]*) – Target path.

isdir (*path: Union[str, pathlib.Path]*) → bool

Check if path is directory.

Parameters **path** (*Union[str, Path]*) – Target path.

Returns True if directory else False.

Return type bool

isfile (*path: Union[str, pathlib.Path]*) → bool

Check if path is file.

Parameters **path** (*Union[str, Path]*) – Target path.

Returns True if file else False.

Return type bool

ismounted (*mount*) → bool

Check if path is mountpoint.

Parameters **path** (*Union[str, Path]*) – Target path.

Returns True if mountpoint else False.

Return type bool

mkdir (*path: Union[pathlib.Path, str]*, *mode: int = 511*, *exist_ok: bool = True*, *parents: bool = False*) → None

Make dir in a remote machine.

Parameters

- **path** (*Union[Path, str]*) – Path of directory to create.
- **mode** (*int, optional*) – Permissions mode of new directory. Defaults to 511.
- **exist_ok** (*bool, optional*) – No error if existing. Defaults to True.
- **parents** (*bool, optional*) – Make parent directories as needed. Defaults to False.

mount (*source: str*, *target: Union[pathlib.Path, str]*, *force: bool = False*, *mkdir: bool = False*) → None

Mounts a filesystem in remote machine.

Parameters

- **source** (*str*) – Source filesystem.
- **target** (*Union[Path, str]*) – Target filesystem.

- **force** (*bool, optional*) – Umount current filesystem and mount new filesystem. Defaults to False.
- **mkdir** (*bool, optional*) – Make directory if not exist. Defaults to False.

Raises *SshConError* – Raises error if target is already a mountpoint and option force is not used.

read_text (*file: Union[pathlib.Path, str], encoding: str = 'utf-8'*) → Optional[str]

Gets text from the remote file.

Parameters

- **file** (*Union[Path, str]*) – Path to a file.
- **encoding** (*str, optional*) – Which encoding to use. Defaults to "utf-8".

Raises **FileNotFoundError** – Raises when file is not found.

Returns File content as string.

Return type Optional[str]

remove (*path: Union[pathlib.Path, str], force: bool = False, recursive: bool = False*) → None

Remove files or directories.

Parameters

- **path** (*Union[Path, str]*) – File or folder to remove.
- **force** (*bool, optional*) – Ignore nonexistent files, never prompt. Defaults to False.
- **recursive** (*bool, optional*) – Remove directories and their contents recursively. Defaults to False.

rmdir (*path: Union[pathlib.Path, str]*) → None

Remove empty directories.

Parameters **path** (*Union[Path, str]*) – Empty directory to remove.

run (*cmd: Union[List, str], capture_output: bool = False, check: bool = True, user: Optional[str] = None, encoding: Optional[str] = 'utf-8'*) → Optional[*sshcon.main.CompletedCommand*]

Run command on the remote machine.

Raises **OSError** – Raises error if command returns non-zero error code.

Returns Object CompletedCommand with rcode, stdout and stderr.

Return type *CompletedCommand*

send_file (*file: Union[pathlib.Path, str], destination: Union[pathlib.Path, str], force: bool = False*) → None

Send local file to a remote location.

Parameters

- **file** (*Union[Path, str]*) – File to send.
- **destination** (*Union[Path, str]*) – Target filename in remote machine.
- **force** (*bool, optional*) – Replace file if already exists. Defaults to False.

Raises

- **IsADirectoryError** – Raises if target destination is a directory.
- **FileExistsError** – Raises if target destination exists.

umount (*target: Union[pathlib.Path, str]*, *rmdir: bool = False*) → None

Unmount filesystems.

Parameters

- **target** (*Union[Path, str]*) – Path to a filesystem to unmount.
- **rmdir** (*bool, optional*) – Remove directory after unmount. Defaults to False.

write_text (*data: str*, *file: Union[pathlib.Path, str]*, *append: bool = False*, *encoding: str = 'utf-8'*, *force: bool = False*) → None

Write text to a remote file.

Parameters

- **data** (*str*) – Content to write to a file.
- **file** (*Union[Path, str]*) – Path to a file.
- **append** (*bool, optional*) – If append text to a file instead of rewrite a content. Defaults to False.
- **encoding** (*str, optional*) – Which encoding to use. Defaults to "utf-8".

Raises

- **IsADirectoryError** – If target file is a directory.
- **FileExistsError** – If target file exists already.

2.2 sshcon.exceptions module

exception sshcon.exceptions.**SshConError** (*function*, *msg*)

Bases: Exception

exception sshcon.exceptions.**SshConSftpError** (*cmd*, *rcode*)

Bases: Exception

CHAPTER

THREE

HOW TO USE

Examples how to use:

```
from sshcon.main import SshCon

hostname = "myserver"
ssh_user = "myuser"
ssh_key = "/home/user/.ssh/mykey"
ssh = SshConn(hostname, ssh_user, ssh_key)

# Run command and save output to the variable
ls = ssh.run(["ls", "-latr", path("/mnt"], capture_output=True, check=True).stdout

# Mount directory
ssh.mount(source="storage:/data", "/mnt", force=True, mkdir=True)

# Remove files
ssh.remove("/my/folder/*.tar", force=True)

# Read text
text = ssh.read_text("/folder/text.txt")

# Write text
ssh.write_text("Hey!", "/folder/text.txt")

# Check if file
if ssh.isfile("/my/file):
    print("It's a file!")
```

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- search

CHAPTER

FIVE

LINKS

- Github Repo
- Pypi

PYTHON MODULE INDEX

S

sshcon.exceptions, 8
sshcon.main, 5

INDEX

C

`chmod()` (*sshcon.main.SshCon method*), 5
`chown()` (*sshcon.main.SshCon method*), 5
`CompletedCommand` (*class in sshcon.main*), 5

G

`get_file()` (*sshcon.main.SshCon method*), 5
`get_filemode()` (*sshcon.main.SshCon method*), 6

I

`isdir()` (*sshcon.main.SshCon method*), 6
`isfile()` (*sshcon.main.SshCon method*), 6
`ismounted()` (*sshcon.main.SshCon method*), 6

M

`mkdir()` (*sshcon.main.SshCon method*), 6
`module`
    `sshcon.exceptions`, 8
    `sshcon.main`, 5
`mount()` (*sshcon.main.SshCon method*), 6

R

`rcode()` (*sshcon.main.CompletedCommand property*), 5
`read_text()` (*sshcon.main.SshCon method*), 7
`remove()` (*sshcon.main.SshCon method*), 7
`rmdir()` (*sshcon.main.SshCon method*), 7
`run()` (*sshcon.main.SshCon method*), 7

S

`send_file()` (*sshcon.main.SshCon method*), 7
`SshCon` (*class in sshcon.main*), 5
`sshcon.exceptions`
    `module`, 8
`sshcon.main`
    `module`, 5
`SshConError`, 8
`SshConSftpError`, 8
`stderr()` (*sshcon.main.CompletedCommand property*), 5
`stdout()` (*sshcon.main.CompletedCommand property*), 5

U

`umount()` (*sshcon.main.SshCon method*), 7

W

`write_text()` (*sshcon.main.SshCon method*), 8