

---

# **rfstools**

***Release 1.8.1***

**Přemysl Šťastný**

**Apr 08, 2024**



**CONTENTS:**

<b>1</b>	<b>rfslib module</b>	<b>3</b>
<b>2</b>	<b>rfslib.abstract_pconnection module</b>	<b>5</b>
<b>3</b>	<b>rfslib.sftp_pconnection module</b>	<b>11</b>
<b>4</b>	<b>rfslib.ftp_pconnection module</b>	<b>17</b>
<b>5</b>	<b>rfslib.smb12_pconnection module</b>	<b>23</b>
<b>6</b>	<b>rfslib.smb23_pconnection module</b>	<b>29</b>
<b>7</b>	<b>rfslib.fs_pconnection module</b>	<b>35</b>
<b>8</b>	<b>rfslib.path_utils module</b>	<b>41</b>
<b>9</b>	<b>_rfstools.arg_parser module</b>	<b>43</b>
<b>10</b>	<b>_rfstools.arg_processor module</b>	<b>45</b>
	<b>Python Module Index</b>	<b>47</b>
	<b>Index</b>	<b>49</b>



This is developer documentation of rfstools. For user documentation, use please README.md in base repository.

To create a new development enviroment, it is recommended to create python virtual enviroment and install dependencies in requirements.txt

If you want to create a new pdf documentation, you are required to install also texlive on your system.



## RFSLIB MODULE

### **class rfslib.pconnection\_settings**

Bases: object

This object represents settings applicable for all PConnection instances (instances of class, which inherits from PConnection).

**default\_dmask: int = 18**

If mode (permissions) of a directory can't be fetched, this value will be used instead of it.

**default\_fmask: int = 91**

If mode (permissions) of a nondirectory file can't be fetched, this value will be used instead of it.

**direct\_write: bool = False**

NOT IMPLEMENTED YET. If True, push will write output directly to file. If False all push operations on regular files will create firstly tmp file in target folder and then move result to file.

**local\_crlf: bool = False**

Does local files use CRLF? If True, it is supposed, they do. If False, it is supposed, they use LF.

**local\_encoding: str = 'UTF8'**

The encoding of local text files. (eg. 'UTF8')

**remote\_crlf: bool = False**

Does remote files use CRLF? If True, it is supposed, they do. If False, it is supposed, they use LF.

**remote\_encoding: str = 'UTF8'**

The encoding of remote text files. (eg. 'cp1250')

**skip\_validation: bool = False**

NOT IMPLEMENTED YED. If True, all validations of input will be skipped. Undefined behavior may happen if input is wrong. Increas performance.

**text\_transmission: bool = False**

If true, all files, which will be transmitted, will be recoded from local\_encoding to remote\_encoding and from local\_crlf to remote\_crlf. If False, there will be no encoding done during transmission.





## RFSLIB.ABSTRACT\_PCONNECTION MODULE

**class** rfslib.abstract\_pconnection.PConnection(*settings*: pconnection\_settings)

Bases: ABC

**abstract** *\_exists*(*remote\_path*: str) → bool

Protected method which checks, whether a remote file exist. If the remote file is a broken symlink, it returns False.

**Parameters**

**remote\_path** – Path of a remote file.

**Returns**

True, if remote file is exist. False, if remote file doesn't exist

**abstract** *\_isdir*(*remote\_path*: str) → bool

Protected method which checks, whether a remote file is a directory. The function is DEPRECATED and will be substituted using stat or lstat.

**Parameters**

**remote\_path** – A path of a directory.

**Returns**

True, if remote file is folder. False, if it isn't a folder. Undefined if the file doesn't exist.

**abstract** *\_lexists*(*remote\_path*: str) → bool

Protected method which checks, whether a remote file exist. If the remote file is a broken symlink, it returns True.

KNOWN BUG: Behavior is undefined in case of broken symlinks.

**Parameters**

**remote\_path** – Path of a remote file.

**Returns**

True, if remote file is exist. False, if remote file doesn't exist

**abstract** *\_listdir*(*remote\_path*: str) → List[str]

Protected method which returns a list of files in the folder including hidden files. It might contain '.' and '..'. Undefined if the remote file doesn't exist or isn't a folder.

**Parameters**

**remote\_path** – The remote path of a remote folder.

**Returns**

A list of files in the remote folder.

**abstract** `_mkdir(remote_path: str)`

Protected method which creates a new directory. Behavior is undefined if remote folder already exist, or destination folder doesn't exist.

**Parameters**

**remote\_path** – A path of a new remote directory.

**abstract** `_pull(remote_path: str, local_path: str)`

Protected method which downloads/pulls a nondirectory file from a remote storage to a local storage in the binary form. Behavior is undefined if source file or destination folder doesn't exist.

**Parameters**

- **remote\_path** – Path of a remote file to download.
- **local\_path** – Path of a local file, where to download/pull a remote file or local file already exists.

**abstract** `_push(local_path: str, remote_path: str)`

Protected method which uploads/pushes a nondirectory file from a local storage to a remote storage in the binary form. Behavior is undefined if destination folder or source file doesn't exist, source is directory or remote file already exists.

**Parameters**

- **local\_path** – Path of a local file to upload.
- **remote\_path** – Path on the remote storage, where to upload/push a local file.

**abstract** `_rename(old_name: str, new_name: str)`

Protected method which renames/moves a file. Behavior is undefined, if *new\_name* file exists or *old\_name* file doesn't exist.

**Parameters**

- **old\_name** – Remote path a file to move.
- **new\_name** – Remote path to which move the file.

**abstract** `_rmdir(remote_path: str)`

Protected method which removes an empty remote directory. Behavior is undefined if remote directory doesn't exist or it isn't empty.

**Parameters**

**remote\_path** – Path of an empty remote directory to delete.

**abstract** `_stat(remote_path: str) → stat_result`

Protected method which returns statistics of a file (eg. size, last date modified,...) Follows symlinks to a destination file. Undefined behavior if remote file doesn't exist or it is a broken symlink.

**Parameters**

**remote\_path** – Path of a remote file.

**Returns**

The function returns `os.stat_result` like object, which is further parsed by `_stat_unpack` function. For more details please see source code.

**abstract** `_unlink(remote_path: str)`

Protected method which removes a nondirectory file. Behavior is undefined if remote file doesn't exist or is a directory.

**Parameters**

**remote\_path** – Path of a remote regular file to delete.

**abstract close()**

Method to close the opened connection.

**cp**(*old\_names: List[str], new\_name: str, recursive: bool = False*)

**dcp**(*old\_names: List[str], target\_dir: str, recursive: bool = False*)

**dmv**(*old\_names: List[str], target\_dir: str*)

**exists**(*remote\_path: str*) → bool

Method which checks, whether a remote file exist. Returns False for broken symlinks.

**Parameters**

**remote\_path** – Path of a remote file.

**Returns**

True, if remote file exists. False, if remote file doesn't exist.

**fcp**(*old\_name: str, new\_name: str*)

**find**(*remote\_path: str, child\_first: bool = False*) → List[str]

A public method which returns DFS (depth-first search) of remote\_path including hidden files. It never returns '.' or '..'.

**Parameters**

**child\_first** – If True, childs of a directory will be returned before the directory itself.

**Returns**

The result of DFS as a list of remote\_paths.

**fmv**(*old\_name: str, new\_name: str*)

**get\_default\_dmask**() → int

Returns default\_dmask settings. For more details see pconnection\_settings.

**get\_default\_fmask**() → int

Returns default\_fmask settings. For more details see pconnection\_settings.

**get\_settings**() → *pconnection\_settings*

The procedure sets all generic settings for PConnection.

**Returns**

A pconnection\_settings object with all generic settings of PConnection.

**isdir**(*remote\_path: str*)

A public method, which checks, whether there is an folder on remote\_path. If yes, true is returned. Otherwise false.

**Parameters**

**remote\_path** – A path, where to check, whether there is an folder.

**lexists**(*remote\_path*)

Method which checks, whether a remote file exist. Returns True for broken symlinks.

**Parameters**

**remote\_path** – Path of a remote file.

**Returns**

True, if remote file exists. False, if remote file doesn't exist.

**listdir**(*remote\_path: str*)

Public method which returns a list of files in the folder including hidden files. It never returns '.' or '..'.

**Parameters**

**remote\_path** – The remote path of a remote folder.

**Returns**

A list of files in the remote folder.

**ls**(*remote\_path: str*)**lstat**(*remote\_path: str*) → *p\_stat\_result*

Returns statistics of a file (eg. size, last date modified,...) Doesn't follow symlinks.

**Parameters**

**remote\_path** – Path of a remote file.

**Returns**

An object whose attributes correspond to the attributes of Python's stat structure as returned by os.stat, except that it contains fewer fields.

**mkdir**(*remote\_path: str*)

A public method, which creates a folder. If directory can't be created, because a file already exist, an exception is raised. No other directories on path will be created and if any of them is missing, an exception is raised.

**Parameters**

**remote\_path** – A path, where to create a new directory.

**mv**(*old\_names: List[str], new\_name: str*)**pmkdir**(*remote\_path: str*)**pull**(*remote\_path: str, local\_path: str*)**push**(*local\_path: str, remote\_path: str*)

Uploads/pushes a file from a local storage to a remote storage in the binary form.

**Parameters**

- **local\_path** – Path of a local file to upload.
- **remote\_path** – Path on the remote storage, where to upload/push a local file.

**rename**(*old\_name: str, new\_name: str*)**rm**(*remote\_path: str, recursive: bool = False*)**rmdir**(*remote\_path: str*)**rpull**(*remote\_path: str, local\_path: str*)**rpush**(*local\_path: str, remote\_path: str*)**set\_settings**(*settings: pconnection\_settings*)

The procedure sets all generic settings for PConnection.

**Parameters**

**settings** – A pconnection\_settings object with all generic settings for PConnection. If some attribute in object is missing, no operation will be done with it.

**stat**(*remote\_path: str*) → *p\_stat\_result*

Returns statistics of a file (eg. size, last date modified,...) Follows symlinks to a destination file.

**Parameters**

**remote\_path** – Path of a remote file.

**Returns**

An object whose attributes correspond to the attributes of Python's stat structure as returned by os.stat, except that it contains fewer fields.

**touch**(*remote\_path: str*)

**unlink**(*remote\_path: str*)

**xls**(*remote\_path: str*)

**class** rfslib.abstract\_pconnection.**p\_stat\_result**

Bases: object

Representation of the attributes of a file (or proxied file). It attempts to mirror the object returned by os.stat as closely as possible.

**st\_atime: int = None**

This is the time of the last access of file data.

**st\_gid: int = None**

This field contains the ID of the group owner of the file.

**st\_mode: int = None**

This field contains the file type and mode.

**st\_mtime: int = None**

This is the time of last modification of file data.

**st\_nlink: int = None**

This field contains the number of hard links to the file.

**st\_size: int = None**

This field gives the size of the file (if it is a regular file or a symbolic link) in bytes. The size of a symbolic link is the length of the pathname it contains, without a terminating null byte.

**st\_uid: int = None**

This field contains the user ID of the owner of the file.



## RFSLIB.SFTP\_PCONNECTION MODULE

```
class rfslib.sftp_pconnection.SftpPConnection(settings: pconnection_settings, host: str, username: str,  
                                             password: str = None, keyfile: str = '~/.ssh/id_rsa', port:  
                                             int = 22, no_host_key_checking: bool = False)
```

Bases: *PConnection*

Class for SFTP connection. Public interface with an exception of `__init__` and `close` is inherited from *PConnection*.

**`_exists(remote_path)`**

Protected method which checks, whether a remote file exist. If the remote file is a broken symlink, it returns `False`.

**Parameters**

**`remote_path`** – Path of a remote file.

**Returns**

True, if remote file is exist. False, if remote file doesn't exist

**`_isdir(remote_path)`**

Protected method which checks, whether a remote file is a directory. The function is DEPRECATED and will be substituted using `stat` or `lstat`.

**Parameters**

**`remote_path`** – A path of a directory.

**Returns**

True, if remote file is folder. False, if it isn't a folder. Undefined if the file doesn't exist.

**`_lexists(remote_path)`**

Protected method which checks, whether a remote file exist. If the remote file is a broken symlink, it returns `True`.

KNOWN BUG: Behavior is undefined in case of broken symlinks.

**Parameters**

**`remote_path`** – Path of a remote file.

**Returns**

True, if remote file is exist. False, if remote file doesn't exist

**`_listdir(remote_path)`**

Protected method which returns a list of files in the folder including hidden files. It might contain `'.'` and `'..'`. Undefined if the remote file doesn't exist or isn't a folder.

**Parameters**

**`remote_path`** – The remote path of a remote folder.

**Returns**

A list of files in the remote folder.

**`_mkdir(remote_path)`**

Protected method which creates a new directory. Behavior is undefined if remote folder already exist, or destination folder doesn't exist.

**Parameters**

**remote\_path** – A path of a new remote directory.

**`_pull(remote_path, local_path)`**

Protected method which downloads/pulls a nondirectory file from a remote storage to a local storage in the binary form. Behavior is undefined if source file or destination folder doesn't exist.

**Parameters**

- **remote\_path** – Path of a remote file to download.
- **local\_path** – Path of a local file, where to download/pull a remote file or local file already exists.

**`_push(local_path, remote_path)`**

Protected method which uploads/pushes a nondirectory file from a local storage to a remote storage in the binary form. Behavior is undefined if destination folder or source file doesn't exist, source is directory or remote file already exists.

**Parameters**

- **local\_path** – Path of a local file to upload.
- **remote\_path** – Path on the remote storage, where to upload/push a local file.

**`_rename(old_name, new_name)`**

Protected method which renames/moves a file. Behavior is undefined, if *new\_name* file exists or *old\_name* file doesn't exist.

**Parameters**

- **old\_name** – Remote path a file to move.
- **new\_name** – Remote path to which move the file.

**`_rmdir(remote_path)`**

Protected method which removes an empty remote directory. Behavior is undefined if remote directory doesn't exist or it isn't empty.

**Parameters**

**remote\_path** – Path of an empty remote directory to delete.

**`_stat(remote_path)`**

Protected method which returns statistics of a file (eg. size, last date modified,...) Follows symlinks to a destination file. Undefined behavior if remote file doesn't exist or it is a broken symlink.

**Parameters**

**remote\_path** – Path of a remote file.

**Returns**

The function returns `os.stat_result` like object, which is further parsed by `_stat_unpack` function. For more details please see source code.



**`_unlink(remote_path)`**

Protected method which removes a nondirectory file. Behavior is undefined if remote file doesn't exist or is a directory.

**Parameters**

**`remote_path`** – Path of a remote regular file to delete.

**`close()`**

Method to close the opened connection.

**`cp(old_names: List[str], new_name: str, recursive: bool = False)`**

**`dcp(old_names: List[str], target_dir: str, recursive: bool = False)`**

**`dmv(old_names: List[str], target_dir: str)`**

**`exists(remote_path: str) → bool`**

Method which checks, whether a remote file exist. Returns False for broken symlinks.

**Parameters**

**`remote_path`** – Path of a remote file.

**Returns**

True, if remote file exists. False, if remote file doesn't exist.

**`fcg(old_name: str, new_name: str)`**

**`find(remote_path: str, child_first: bool = False) → List[str]`**

A public method which returns DFS (depth-first search) of `remote_path` including hidden files. It never returns `'.'` or `'..'`.

**Parameters**

**`child_first`** – If True, childs of a directory will be returned before the directory itself.

**Returns**

The result of DFS as a list of `remote_paths`.

**`fmv(old_name: str, new_name: str)`**

**`get_default_dmask() → int`**

Returns default `dmask` settings. For more details see `pconnection_settings`.

**`get_default_fmask() → int`**

Returns default `fmask` settings. For more details see `pconnection_settings`.

**`get_settings() → pconnection_settings`**

The procedure sets all generic settings for `PConnection`.

**Returns**

A `pconnection_settings` object with all generic settings of `PConnection`.

**`isdir(remote_path: str)`**

A public method, which checks, whether there is an folder on `remote_path`. If yes, true is returned. Otherwise false.

**Parameters**

**`remote_path`** – A path, where to check, whether there is an folder.

**lexists**(*remote\_path*)

Method which checks, whether a remote file exist. Returns True for broken symlinks.

**Parameters**

**remote\_path** – Path of a remote file.

**Returns**

True, if remote file exists. False, if remote file doesn't exist.

**listdir**(*remote\_path: str*)

Public method which returns a list of files in the folder including hidden files. It never returns '.' or '..'.

**Parameters**

**remote\_path** – The remote path of a remote folder.

**Returns**

A list of files in the remote folder.

**ls**(*remote\_path: str*)**lstat**(*remote\_path: str*) → *p\_stat\_result*

Returns statistics of a file (eg. size, last date modified,...) Doesn't follow symlinks.

**Parameters**

**remote\_path** – Path of a remote file.

**Returns**

An object whose attributes correspond to the attributes of Python's stat structure as returned by os.stat, except that it contains fewer fields.

**mkdir**(*remote\_path: str*)

A public method, which creates a folder. If directory can't be created, because a file already exist, an exception is raised. No other directories on path will be created and if any of them is missing, an exception is raised.

**Parameters**

**remote\_path** – A path, where to create a new directory.

**mv**(*old\_names: List[str], new\_name: str*)**pmkdir**(*remote\_path: str*)**pull**(*remote\_path: str, local\_path: str*)**push**(*local\_path: str, remote\_path: str*)

Uploads/pushes a file from a local storage to a remote storage in the binary form.

**Parameters**

- **local\_path** – Path of a local file to upload.
- **remote\_path** – Path on the remote storage, where to upload/push a local file.

**rename**(*old\_name: str, new\_name: str*)**rm**(*remote\_path: str, recursive: bool = False*)**rmdir**(*remote\_path: str*)**rpull**(*remote\_path: str, local\_path: str*)

**rpush**(*local\_path: str, remote\_path: str*)

**set\_settings**(*settings: pconnection\_settings*)

The procedure sets all generic settings for PConnection.

**Parameters**

**settings** – A pconnection\_settings object with all generic settings for PConnection. If some attribute in object is missing, no operation will be done with it.

**stat**(*remote\_path: str*) → *p\_stat\_result*

Returns statistics of a file (eg. size, last date modified,...) Follows symlinks to a destination file.

**Parameters**

**remote\_path** – Path of a remote file.

**Returns**

An object whose attributes correspond to the attributes of Python's stat structure as returned by os.stat, except that it contains fewer fields.

**touch**(*remote\_path: str*)

**unlink**(*remote\_path: str*)

**xls**(*remote\_path: str*)



## RFSLIB.FTP\_PCONNECTION MODULE

```
class rfslib.ftp_pconnection.FtpPConnection(settings: pconnection_settings, host: str, username: str,  
                                             password: str, port: int = 21, tls: bool = False,  
                                             passive_mode: bool = False, debug_level: int = 1,  
                                             connection_encoding: str = 'UTF8', dont_use_list_a: bool  
                                             = False)
```

Bases: *PConnection*

Class for FTP connection. Public interface with an exception of `__init__` and `close` is inherited from *PConnection*.

**`_exists(remote_path)`**

Protected method which checks, whether a remote file exist. If the remote file is a broken symlink, it returns False.

**Parameters**

**`remote_path`** – Path of a remote file.

**Returns**

True, if remote file is exist. False, if remote file doesn't exist

**`_isdir(remote_path)`**

Protected method which checks, whether a remote file is a directory. The function is DEPRECATED and will be substituted using `stat` or `lstat`.

**Parameters**

**`remote_path`** – A path of a directory.

**Returns**

True, if remote file is folder. False, if it isn't a folder. Undefined if the file doesn't exist.

**`_lexists(remote_path)`**

Protected method which checks, whether a remote file exist. If the remote file is a broken symlink, it returns True.

KNOWN BUG: Behavior is undefined in case of broken symlinks.

**Parameters**

**`remote_path`** – Path of a remote file.

**Returns**

True, if remote file is exist. False, if remote file doesn't exist

**`_listdir(remote_path)`**

Protected method which returns a list of files in the folder including hidden files. It might contain `'.'` and `'..'`. Undefined if the remote file doesn't exist or isn't a folder.

**Parameters**

**remote\_path** – The remote path of a remote folder.

**Returns**

A list of files in the remote folder.

**\_mkdir**(*remote\_path*)

Protected method which creates a new directory. Behavior is undefined if remote folder already exist, or destination folder doesn't exist.

**Parameters**

**remote\_path** – A path of a new remote directory.

**\_pull**(*remote\_path*, *local\_path*)

Protected method which downloads/pulls a nondirectory file from a remote storage to a local storage in the binary form. Behavior is undefined if source file or destination folder doesn't exist.

**Parameters**

- **remote\_path** – Path of a remote file to download.
- **local\_path** – Path of a local file, where to download/pull a remote file or local file already exists.

**\_push**(*local\_path*, *remote\_path*)

Protected method which uploads/pushes a nondirectory file from a local storage to a remote storage in the binary form. Behavior is undefined if destination folder or source file doesn't exist, source is directory or remote file already exists.

**Parameters**

- **local\_path** – Path of a local file to upload.
- **remote\_path** – Path on the remote storage, where to upload/push a local file.

**\_rename**(*old\_name*, *new\_name*)

Protected method which renames/moves a file. Behavior is undefined, if *new\_name* file exists or *old\_name* file doesn't exist.

**Parameters**

- **old\_name** – Remote path a file to move.
- **new\_name** – Remote path to which move the file.

**\_rmdir**(*remote\_path*)

Protected method which removes an empty remote directory. Behavior is undefined if remote directory doesn't exist or it isn't empty.

**Parameters**

**remote\_path** – Path of an empty remote directory to delete.

**\_stat**(*remote\_path*)

Protected method which returns statistics of a file (eg. size, last date modified,...) Follows symlinks to a destination file. Undefined behavior if remote file doesn't exist or it is a broken symlink.

**Parameters**

**remote\_path** – Path of a remote file.

**Returns**

The function returns `os.stat_result` like object, which is further parsed by `_stat_unpack` function. For more details please see source code.

**`_unlink(remote_path)`**

Protected method which removes a nondirectory file. Behavior is undefined if remote file doesn't exist or is a directory.

**Parameters**

**`remote_path`** – Path of a remote regular file to delete.

**`close()`**

Method to close the opened connection.

**`cp(old_names: List[str], new_name: str, recursive: bool = False)`**

**`dcp(old_names: List[str], target_dir: str, recursive: bool = False)`**

**`dmv(old_names: List[str], target_dir: str)`**

**`exists(remote_path: str) → bool`**

Method which checks, whether a remote file exist. Returns False for broken symlinks.

**Parameters**

**`remote_path`** – Path of a remote file.

**Returns**

True, if remote file exists. False, if remote file doesn't exist.

**`fcg(old_name: str, new_name: str)`**

**`find(remote_path: str, child_first: bool = False) → List[str]`**

A public method which returns DFS (depth-first search) of `remote_path` including hidden files. It never returns `'.'` or `'..'`.

**Parameters**

**`child_first`** – If True, childs of a directory will be returned before the directory itself.

**Returns**

The result of DFS as a list of `remote_paths`.

**`fmv(old_name: str, new_name: str)`**

**`get_default_dmask() → int`**

Returns default `dmask` settings. For more details see `pconnection_settings`.

**`get_default_fmask() → int`**

Returns default `fmask` settings. For more details see `pconnection_settings`.

**`get_settings() → pconnection_settings`**

The procedure sets all generic settings for `PConnection`.

**Returns**

A `pconnection_settings` object with all generic settings of `PConnection`.

**`isdir(remote_path: str)`**

A public method, which checks, whether there is an folder on `remote_path`. If yes, true is returned. Otherwise false.

**Parameters**

**`remote_path`** – A path, where to check, whether there is an folder.

**lexists**(*remote\_path*)

Method which checks, whether a remote file exist. Returns True for broken symlinks.

**Parameters**

**remote\_path** – Path of a remote file.

**Returns**

True, if remote file exists. False, if remote file doesn't exist.

**listdir**(*remote\_path: str*)

Public method which returns a list of files in the folder including hidden files. It never returns '.' or '..'.

**Parameters**

**remote\_path** – The remote path of a remote folder.

**Returns**

A list of files in the remote folder.

**ls**(*remote\_path: str*)**lstat**(*remote\_path: str*) → *p\_stat\_result*

Returns statistics of a file (eg. size, last date modified,...) Doesn't follow symlinks.

**Parameters**

**remote\_path** – Path of a remote file.

**Returns**

An object whose attributes correspond to the attributes of Python's stat structure as returned by os.stat, except that it contains fewer fields.

**mkdir**(*remote\_path: str*)

A public method, which creates a folder. If directory can't be created, because a file already exist, an exception is raised. No other directories on path will be created and if any of them is missing, an exception is raised.

**Parameters**

**remote\_path** – A path, where to create a new directory.

**mv**(*old\_names: List[str], new\_name: str*)**pmkdir**(*remote\_path: str*)**pull**(*remote\_path: str, local\_path: str*)**push**(*local\_path: str, remote\_path: str*)

Uploads/pushes a file from a local storage to a remote storage in the binary form.

**Parameters**

- **local\_path** – Path of a local file to upload.
- **remote\_path** – Path on the remote storage, where to upload/push a local file.

**rename**(*old\_name: str, new\_name: str*)**rm**(*remote\_path: str, recursive: bool = False*)**rmdir**(*remote\_path: str*)**rpull**(*remote\_path: str, local\_path: str*)



**rpush**(*local\_path: str, remote\_path: str*)

**set\_settings**(*settings: pconnection\_settings*)

The procedure sets all generic settings for PConnection.

**Parameters**

**settings** – A pconnection\_settings object with all generic settings for PConnection. If some attribute in object is missing, no operation will be done with it.

**stat**(*remote\_path: str*) → *p\_stat\_result*

Returns statistics of a file (eg. size, last date modified,...) Follows symlinks to a destination file.

**Parameters**

**remote\_path** – Path of a remote file.

**Returns**

An object whose attributes correspond to the attributes of Python's stat structure as returned by os.stat, except that it contains fewer fields.

**touch**(*remote\_path: str*)

**unlink**(*remote\_path: str*)

**xls**(*remote\_path: str*)



## RFSLIB.SMB12\_PCONNECTION MODULE

```
class rfslib.smb12_pconnection.Smb12PConnection(settings: pconnection_settings, host: str,
                                              service_name: str, username: str, password: str,
                                              port: int = 139, use_direct_tcp: bool = False,
                                              client_name: str = 'RFS', use_ntlm_v1: bool = False)
```

Bases: *PConnection*

Class for SMB connection version 1 or 2. Public interface with an exception of `__init__` and `close` is inherited from `PConnection`.

**`_exists(remote_path)`**

Protected method which checks, whether a remote file exist. If the remote file is a broken symlink, it returns `False`.

**Parameters**

**`remote_path`** – Path of a remote file.

**Returns**

True, if remote file is exist. False, if remote file doesn't exist

**`_isdir(remote_path)`**

Protected method which checks, whether a remote file is a directory. The function is DEPRECATED and will be substituted using `stat` or `lstat`.

**Parameters**

**`remote_path`** – A path of a directory.

**Returns**

True, if remote file is folder. False, if it isn't a folder. Undefined if the file doesn't exist.

**`_lexists(remote_path)`**

Protected method which checks, whether a remote file exist. If the remote file is a broken symlink, it returns `True`.

KNOWN BUG: Behavior is undefined in case of broken symlinks.

**Parameters**

**`remote_path`** – Path of a remote file.

**Returns**

True, if remote file is exist. False, if remote file doesn't exist

**`_listdir(remote_path)`**

Protected method which returns a list of files in the folder including hidden files. It might contain `'.'` and `'..'`. Undefined if the remote file doesn't exist or isn't a folder.

**Parameters**

**remote\_path** – The remote path of a remote folder.

**Returns**

A list of files in the remote folder.

**\_mkdir**(*remote\_path*)

Protected method which creates a new directory. Behavior is undefined if remote folder already exist, or destination folder doesn't exist.

**Parameters**

**remote\_path** – A path of a new remote directory.

**\_pull**(*remote\_path*, *local\_path*)

Protected method which downloads/pulls a nondirectory file from a remote storage to a local storage in the binary form. Behavior is undefined if source file or destination folder doesn't exist.

**Parameters**

- **remote\_path** – Path of a remote file to download.
- **local\_path** – Path of a local file, where to download/pull a remote file or local file already exists.

**\_push**(*local\_path*, *remote\_path*)

Protected method which uploads/pushes a nondirectory file from a local storage to a remote storage in the binary form. Behavior is undefined if destination folder or source file doesn't exist, source is directory or remote file already exists.

**Parameters**

- **local\_path** – Path of a local file to upload.
- **remote\_path** – Path on the remote storage, where to upload/push a local file.

**\_rename**(*old\_name*, *new\_name*)

Protected method which renames/moves a file. Behavior is undefined, if *new\_name* file exists or *old\_name* file doesn't exist.

**Parameters**

- **old\_name** – Remote path a file to move.
- **new\_name** – Remote path to which move the file.

**\_rmdir**(*remote\_path*)

Protected method which removes an empty remote directory. Behavior is undefined if remote directory doesn't exist or it isn't empty.

**Parameters**

**remote\_path** – Path of an empty remote directory to delete.

**\_stat**(*remote\_path*)

Protected method which returns statistics of a file (eg. size, last date modified,...) Follows symlinks to a destination file. Undefined behavior if remote file doesn't exist or it is a broken symlink.

**Parameters**

**remote\_path** – Path of a remote file.

**Returns**

The function returns `os.stat_result` like object, which is further parsed by `_stat_unpack` function. For more details please see source code.

**`_unlink(remote_path)`**

Protected method which removes a nondirectory file. Behavior is undefined if remote file doesn't exist or is a directory.

**Parameters**

**`remote_path`** – Path of a remote regular file to delete.

**`close()`**

Method to close the opened connection.

**`cp(old_names: List[str], new_name: str, recursive: bool = False)`**

**`dcp(old_names: List[str], target_dir: str, recursive: bool = False)`**

**`dmv(old_names: List[str], target_dir: str)`**

**`exists(remote_path: str) → bool`**

Method which checks, whether a remote file exist. Returns False for broken symlinks.

**Parameters**

**`remote_path`** – Path of a remote file.

**Returns**

True, if remote file exists. False, if remote file doesn't exist.

**`fcg(old_name: str, new_name: str)`**

**`find(remote_path: str, child_first: bool = False) → List[str]`**

A public method which returns DFS (depth-first search) of `remote_path` including hidden files. It never returns `'.'` or `'..'`.

**Parameters**

**`child_first`** – If True, childs of a directory will be returned before the directory itself.

**Returns**

The result of DFS as a list of `remote_paths`.

**`fmv(old_name: str, new_name: str)`**

**`get_default_dmask() → int`**

Returns default `dmask` settings. For more details see `pconnection_settings`.

**`get_default_fmask() → int`**

Returns default `fmask` settings. For more details see `pconnection_settings`.

**`get_settings() → pconnection_settings`**

The procedure sets all generic settings for `PConnection`.

**Returns**

A `pconnection_settings` object with all generic settings of `PConnection`.

**`isdir(remote_path: str)`**

A public method, which checks, whether there is an folder on `remote_path`. If yes, true is returned. Otherwise false.

**Parameters**

**`remote_path`** – A path, where to check, whether there is an folder.

**lexists**(*remote\_path*)

Method which checks, whether a remote file exist. Returns True for broken symlinks.

**Parameters**

**remote\_path** – Path of a remote file.

**Returns**

True, if remote file exists. False, if remote file doesn't exist.

**listdir**(*remote\_path: str*)

Public method which returns a list of files in the folder including hidden files. It never returns '.' or '..'.

**Parameters**

**remote\_path** – The remote path of a remote folder.

**Returns**

A list of files in the remote folder.

**ls**(*remote\_path: str*)**lstat**(*remote\_path: str*) → *p\_stat\_result*

Returns statistics of a file (eg. size, last date modified,...) Doesn't follow symlinks.

**Parameters**

**remote\_path** – Path of a remote file.

**Returns**

An object whose attributes correspond to the attributes of Python's stat structure as returned by os.stat, except that it contains fewer fields.

**mkdir**(*remote\_path: str*)

A public method, which creates a folder. If directory can't be created, because a file already exist, an exception is raised. No other directories on path will be created and if any of them is missing, an exception is raised.

**Parameters**

**remote\_path** – A path, where to create a new directory.

**mv**(*old\_names: List[str], new\_name: str*)**pmkdir**(*remote\_path: str*)**pull**(*remote\_path: str, local\_path: str*)**push**(*local\_path: str, remote\_path: str*)

Uploads/pushes a file from a local storage to a remote storage in the binary form.

**Parameters**

- **local\_path** – Path of a local file to upload.
- **remote\_path** – Path on the remote storage, where to upload/push a local file.

**rename**(*old\_name: str, new\_name: str*)**rm**(*remote\_path: str, recursive: bool = False*)**rmdir**(*remote\_path: str*)**rpull**(*remote\_path: str, local\_path: str*)

**rpush**(*local\_path: str, remote\_path: str*)

**set\_settings**(*settings: pconnection\_settings*)

The procedure sets all generic settings for PConnection.

**Parameters**

**settings** – A pconnection\_settings object with all generic settings for PConnection. If some attribute in object is missing, no operation will be done with it.

**stat**(*remote\_path: str*) → *p\_stat\_result*

Returns statistics of a file (eg. size, last date modified,...) Follows symlinks to a destination file.

**Parameters**

**remote\_path** – Path of a remote file.

**Returns**

An object whose attributes correspond to the attributes of Python's stat structure as returned by os.stat, except that it contains fewer fields.

**touch**(*remote\_path: str*)

**unlink**(*remote\_path: str*)

**xls**(*remote\_path: str*)





## RFSLIB.SMB23\_PCONNECTION MODULE

```
class rfslib.smb23_pconnection.Smb23PConnection(settings: pconnection_settings, host: str,
                                              service_name: str, username: str, password: str,
                                              port: int = 445, enable_encryption: bool = False,
                                              dont_require_signing: bool = False)
```

Bases: *PConnection*

Class for SMB connection version 2 or 3. Public interface with an exception of `__init__` and `close` is inherited from *PConnection*.

**`_exists(remote_path)`**

Protected method which checks, whether a remote file exist. If the remote file is a broken symlink, it returns False.

**Parameters**

**`remote_path`** – Path of a remote file.

**Returns**

True, if remote file is exist. False, if remote file doesn't exist

**`_isdir(remote_path)`**

Protected method which checks, whether a remote file is a directory. The function is DEPRECATED and will be substituted using `stat` or `lstat`.

**Parameters**

**`remote_path`** – A path of a directory.

**Returns**

True, if remote file is folder. False, if it isn't a folder. Undefined if the file doesn't exist.

**`_lexists(remote_path)`**

Protected method which checks, whether a remote file exist. If the remote file is a broken symlink, it returns True.

KNOWN BUG: Behavior is undefined in case of broken symlinks.

**Parameters**

**`remote_path`** – Path of a remote file.

**Returns**

True, if remote file is exist. False, if remote file doesn't exist

**`_listdir(remote_path)`**

Protected method which returns a list of files in the folder including hidden files. It might contain `'.'` and `'..'`. Undefined if the remote file doesn't exist or isn't a folder.

**Parameters**

**remote\_path** – The remote path of a remote folder.

**Returns**

A list of files in the remote folder.

**\_mkdir**(*remote\_path*)

Protected method which creates a new directory. Behavior is undefined if remote folder already exist, or destination folder doesn't exist.

**Parameters**

**remote\_path** – A path of a new remote directory.

**\_pull**(*remote\_path*, *local\_path*)

Protected method which downloads/pulls a nondirectory file from a remote storage to a local storage in the binary form. Behavior is undefined if source file or destination folder doesn't exist.

**Parameters**

- **remote\_path** – Path of a remote file to download.
- **local\_path** – Path of a local file, where to download/pull a remote file or local file already exists.

**\_push**(*local\_path*, *remote\_path*)

Protected method which uploads/pushes a nondirectory file from a local storage to a remote storage in the binary form. Behavior is undefined if destination folder or source file doesn't exist, source is directory or remote file already exists.

**Parameters**

- **local\_path** – Path of a local file to upload.
- **remote\_path** – Path on the remote storage, where to upload/push a local file.

**\_rename**(*old\_name*, *new\_name*)

Protected method which renames/moves a file. Behavior is undefined, if *new\_name* file exists or *old\_name* file doesn't exist.

**Parameters**

- **old\_name** – Remote path a file to move.
- **new\_name** – Remote path to which move the file.

**\_rmdir**(*remote\_path*)

Protected method which removes an empty remote directory. Behavior is undefined if remote directory doesn't exist or it isn't empty.

**Parameters**

**remote\_path** – Path of an empty remote directory to delete.

**\_stat**(*remote\_path*)

Protected method which returns statistics of a file (eg. size, last date modified,...) Follows symlinks to a destination file. Undefined behavior if remote file doesn't exist or it is a broken symlink.

**Parameters**

**remote\_path** – Path of a remote file.

**Returns**

The function returns `os.stat_result` like object, which is further parsed by `_stat_unpack` function. For more details please see source code.

**`_unlink(remote_path)`**

Protected method which removes a nondirectory file. Behavior is undefined if remote file doesn't exist or is a directory.

**Parameters**

**`remote_path`** – Path of a remote regular file to delete.

**`close()`**

Method to close the opened connection.

**`cp(old_names: List[str], new_name: str, recursive: bool = False)`**

**`dcp(old_names: List[str], target_dir: str, recursive: bool = False)`**

**`dmv(old_names: List[str], target_dir: str)`**

**`exists(remote_path: str) → bool`**

Method which checks, whether a remote file exist. Returns False for broken symlinks.

**Parameters**

**`remote_path`** – Path of a remote file.

**Returns**

True, if remote file exists. False, if remote file doesn't exist.

**`fcg(old_name: str, new_name: str)`**

**`find(remote_path: str, child_first: bool = False) → List[str]`**

A public method which returns DFS (depth-first search) of `remote_path` including hidden files. It never returns `'.'` or `'..'`.

**Parameters**

**`child_first`** – If True, childs of a directory will be returned before the directory itself.

**Returns**

The result of DFS as a list of `remote_paths`.

**`fmv(old_name: str, new_name: str)`**

**`get_default_dmask() → int`**

Returns default `dmask` settings. For more details see `pconnection_settings`.

**`get_default_fmask() → int`**

Returns default `fmask` settings. For more details see `pconnection_settings`.

**`get_settings() → pconnection_settings`**

The procedure sets all generic settings for `PConnection`.

**Returns**

A `pconnection_settings` object with all generic settings of `PConnection`.

**`isdir(remote_path: str)`**

A public method, which checks, whether there is an folder on `remote_path`. If yes, true is returned. Otherwise false.

**Parameters**

**`remote_path`** – A path, where to check, whether there is an folder.

**lexists**(*remote\_path*)

Method which checks, whether a remote file exist. Returns True for broken symlinks.

**Parameters**

**remote\_path** – Path of a remote file.

**Returns**

True, if remote file exists. False, if remote file doesn't exist.

**listdir**(*remote\_path: str*)

Public method which returns a list of files in the folder including hidden files. It never returns '.' or '..'.

**Parameters**

**remote\_path** – The remote path of a remote folder.

**Returns**

A list of files in the remote folder.

**ls**(*remote\_path: str*)**lstat**(*remote\_path: str*) → *p\_stat\_result*

Returns statistics of a file (eg. size, last date modified,...) Doesn't follow symlinks.

**Parameters**

**remote\_path** – Path of a remote file.

**Returns**

An object whose attributes correspond to the attributes of Python's stat structure as returned by os.stat, except that it contains fewer fields.

**mkdir**(*remote\_path: str*)

A public method, which creates a folder. If directory can't be created, because a file already exist, an exception is raised. No other directories on path will be created and if any of them is missing, an exception is raised.

**Parameters**

**remote\_path** – A path, where to create a new directory.

**mv**(*old\_names: List[str], new\_name: str*)**pmkdir**(*remote\_path: str*)**pull**(*remote\_path: str, local\_path: str*)**push**(*local\_path: str, remote\_path: str*)

Uploads/pushes a file from a local storage to a remote storage in the binary form.

**Parameters**

- **local\_path** – Path of a local file to upload.
- **remote\_path** – Path on the remote storage, where to upload/push a local file.

**rename**(*old\_name: str, new\_name: str*)**rm**(*remote\_path: str, recursive: bool = False*)**rmdir**(*remote\_path: str*)**rpull**(*remote\_path: str, local\_path: str*)

**rpush**(*local\_path: str, remote\_path: str*)

**set\_settings**(*settings: pconnection\_settings*)

The procedure sets all generic settings for PConnection.

**Parameters**

**settings** – A pconnection\_settings object with all generic settings for PConnection. If some attribute in object is missing, no operation will be done with it.

**stat**(*remote\_path: str*) → *p\_stat\_result*

Returns statistics of a file (eg. size, last date modified,...) Follows symlinks to a destination file.

**Parameters**

**remote\_path** – Path of a remote file.

**Returns**

An object whose attributes correspond to the attributes of Python's stat structure as returned by os.stat, except that it contains fewer fields.

**touch**(*remote\_path: str*)

**unlink**(*remote\_path: str*)

**xls**(*remote\_path: str*)

**rfslib.smb23\_pconnection.config\_smb23**(*username: str = None, password: str = None, no\_dfs: bool = False, disable\_secure\_negotiate: bool = False, dfs\_domain\_controller: str = None, auth\_protocol: str = 'negotiate'*)

The procedure changes global setting for SMB version 2 or 3 across all connection. Don't change value, if any SMB connection version 2 or 3 is active.

**Parameters**

- **username** – Optional default username used when creating a new SMB session.
- **password** – Optional default password used when creating a new SMB session.
- **no\_dfs** – Disables DFS support - useful as a bug fix.
- **disable\_secure\_negotiate** – Disables secure negotiate requirement for a SMB connection.
- **dfs\_domain\_controller** – The DFS domain controller address. Useful in case, when rfstools fails to find it itself.
- **auth\_protocol** – The protocol to use for authentication. Possible values are 'negotiate', 'ntlm' or 'kerberos'. Defaults to 'negotiate'.



## RFSLIB.FS\_PCONNECTION MODULE

**class** rfslib.fs\_pconnection.FsPConnection(settings: pconnection\_settings)

Bases: *PConnection*

Class for operating with local filesystem. Public interface with an exception of `__init__` and `close` is inherited from *PConnection*.

**`_exists(remote_path)`**

Protected method which checks, whether a remote file exist. If the remote file is a broken symlink, it returns False.

**Parameters**

**remote\_path** – Path of a remote file.

**Returns**

True, if remote file is exist. False, if remote file doesn't exist

**`_isdir(remote_path)`**

Protected method which checks, whether a remote file is a directory. The function is DEPRECATED and will be substituted using `stat` or `lstat`.

**Parameters**

**remote\_path** – A path of a directory.

**Returns**

True, if remote file is folder. False, if it isn't a folder. Undefined if the file doesn't exist.

**`_lexists(remote_path)`**

Protected method which checks, whether a remote file exist. If the remote file is a broken symlink, it returns True.

KNOWN BUG: Behavior is undefined in case of broken symlinks.

**Parameters**

**remote\_path** – Path of a remote file.

**Returns**

True, if remote file is exist. False, if remote file doesn't exist

**`_listdir(remote_path)`**

Protected method which returns a list of files in the folder including hidden files. It might contain `'.'` and `'..'`. Undefined if the remote file doesn't exist or isn't a folder.

**Parameters**

**remote\_path** – The remote path of a remote folder.

**Returns**

A list of files in the remote folder.

**\_mkdir**(*remote\_path*)

Protected method which creates a new directory. Behavior is undefined if remote folder already exist, or destination folder doesn't exist.

**Parameters**

**remote\_path** – A path of a new remote directory.

**\_pull**(*remote\_path*, *local\_path*)

Protected method which downloads/pulls a nondirectory file from a remote storage to a local storage in the binary form. Behavior is undefined if source file or destination folder doesn't exist.

**Parameters**

- **remote\_path** – Path of a remote file to download.
- **local\_path** – Path of a local file, where to download/pull a remote file or local file already exists.

**\_push**(*local\_path*, *remote\_path*)

Protected method which uploads/pushes a nondirectory file from a local storage to a remote storage in the binary form. Behavior is undefined if destination folder or source file doesn't exist, source is directory or remote file already exists.

**Parameters**

- **local\_path** – Path of a local file to upload.
- **remote\_path** – Path on the remote storage, where to upload/push a local file.

**\_rename**(*old\_name*, *new\_name*)

Protected method which renames/moves a file. Behavior is undefined, if *new\_name* file exists or *old\_name* file doesn't exist.

**Parameters**

- **old\_name** – Remote path a file to move.
- **new\_name** – Remote path to which move the file.

**\_rmdir**(*remote\_path*)

Protected method which removes an empty remote directory. Behavior is undefined if remote directory doesn't exist or it isn't empty.

**Parameters**

**remote\_path** – Path of an empty remote directory to delete.

**\_stat**(*remote\_path*)

Protected method which returns statistics of a file (eg. size, last date modified,...) Follows symlinks to a destination file. Undefined behavior if remote file doesn't exist or it is a broken symlink.

**Parameters**

**remote\_path** – Path of a remote file.

**Returns**

The function returns `os.stat_result` like object, which is further parsed by `_stat_unpack` function. For more details please see source code.

**\_unlink**(*remote\_path*)

Protected method which removes a nondirectory file. Behavior is undefined if remote file doesn't exist or is a directory.



**Parameters****remote\_path** – Path of a remote regular file to delete.**close()**

Method to close the opened connection.

**cp**(*old\_names: List[str], new\_name: str, recursive: bool = False*)**dcp**(*old\_names: List[str], target\_dir: str, recursive: bool = False*)**dmv**(*old\_names: List[str], target\_dir: str*)**exists**(*remote\_path: str*) → bool

Method which checks, whether a remote file exist. Returns False for broken symlinks.

**Parameters****remote\_path** – Path of a remote file.**Returns**

True, if remote file exists. False, if remote file doesn't exist.

**fcp**(*old\_name: str, new\_name: str*)**find**(*remote\_path: str, child\_first: bool = False*) → List[str]

A public method which returns DFS (depth-first search) of remote\_path including hidden files. It never returns '.' or '..'.

**Parameters****child\_first** – If True, childs of a directory will be returned before the directory itself.**Returns**

The result of DFS as a list of remote\_paths.

**fmv**(*old\_name: str, new\_name: str*)**get\_default\_dmask**() → int

Returns default\_dmask settings. For more details see pconnection\_settings.

**get\_default\_fmask**() → int

Returns default\_fmask settings. For more details see pconnection\_settings.

**get\_settings**() → *pconnection\_settings*

The procedure sets all generic settings for PConnection.

**Returns**

A pconnection\_settings object with all generic settings of PConnection.

**isdir**(*remote\_path: str*)

A public method, which checks, whether there is an folder on remote\_path. If yes, true is returned. Otherwise false.

**Parameters****remote\_path** – A path, where to check, whether there is an folder.**lexists**(*remote\_path*)

Method which checks, whether a remote file exist. Returns True for broken symlinks.

**Parameters****remote\_path** – Path of a remote file.**Returns**

True, if remote file exists. False, if remote file doesn't exist.

**listdir**(*remote\_path: str*)

Public method which returns a list of files in the folder including hidden files. It never returns '.' or '..'.

**Parameters**

**remote\_path** – The remote path of a remote folder.

**Returns**

A list of files in the remote folder.

**ls**(*remote\_path: str*)

**lstat**(*remote\_path: str*) → *p\_stat\_result*

Returns statistics of a file (eg. size, last date modified,...) Doesn't follow symlinks.

**Parameters**

**remote\_path** – Path of a remote file.

**Returns**

An object whose attributes correspond to the attributes of Python's stat structure as returned by os.stat, except that it contains fewer fields.

**mkdir**(*remote\_path: str*)

A public method, which creates a folder. If directory can't be created, because a file already exist, an exception is raised. No other directories on path will be created and if any of them is missing, an exception is raised.

**Parameters**

**remote\_path** – A path, where to create a new directory.

**mv**(*old\_names: List[str], new\_name: str*)

**pmkdir**(*remote\_path: str*)

**pull**(*remote\_path: str, local\_path: str*)

**push**(*local\_path: str, remote\_path: str*)

Uploads/pushes a file from a local storage to a remote storage in the binary form.

**Parameters**

- **local\_path** – Path of a local file to upload.
- **remote\_path** – Path on the remote storage, where to upload/push a local file.

**rename**(*old\_name: str, new\_name: str*)

**rm**(*remote\_path: str, recursive: bool = False*)

**rmdir**(*remote\_path: str*)

**rpull**(*remote\_path: str, local\_path: str*)

**rpush**(*local\_path: str, remote\_path: str*)

**set\_settings**(*settings: pconnection\_settings*)

The procedure sets all generic settings for PConnection.

**Parameters**

**settings** – A pconnection\_settings object with all generic settings for PConnection. If some attribute in object is missing, no operation will be done with it.

**stat**(*remote\_path: str*) → *p\_stat\_result*

Returns statistics of a file (eg. size, last date modified,...) Follows symlinks to a destination file.

**Parameters**

**remote\_path** – Path of a remote file.

**Returns**

An object whose attributes correspond to the attributes of Python's stat structure as returned by os.stat, except that it contains fewer fields.

**touch**(*remote\_path: str*)

**unlink**(*remote\_path: str*)

**xls**(*remote\_path: str*)



## RFSLIB.PATH\_UTILS MODULE

```
class rfslib.path_utils.GenericPath(path)
    Bases: object
rfslib.path_utils.add_r_prefix(path)
rfslib.path_utils.generic_cp(conn, sources, dest, recursive=False)
rfslib.path_utils.generic_mv(conn, sources, dest)
rfslib.path_utils.generic_path_normalize(path)
rfslib.path_utils.is_remote(path)
rfslib.path_utils.path_normalize(path)
rfslib.path_utils.remove_r_prefix(path)
```



## **`_RFSTOOLS.ARG_PARSER` MODULE**

`_rfstools.arg_parser.default_arg_parser`(*description: str = "", wildcard\_skipper=False*) →  
ArgumentParser

A function, which is called in the beginning of all scripts in *bin*. It generates a base parser sceleton, which is later altered by script itself.

### **Parameters**

- **description** – The script description. Basically what the script does.
- **wildcard\_skipper** – Adds `–ignore-failed-wildcards` option.

### **Returns**

A base parser sceleton.

`_rfstools.arg_parser.many_to_one_arg_parser`(*description: str = ""*) → ArgumentParser

`_rfstools.arg_parser.one_arg_parser`(*description: str = ""*) → ArgumentParser

`_rfstools.arg_parser.oneplus_arg_parser`(*description: str = "", wildcard\_skipper=False*) →  
ArgumentParser





## **`_RFSTOOLS.ARG_PROCESSOR` MODULE**

`_rfstools.arg_processor.init(arg_parser, name, vars_to_pass)`



## PYTHON MODULE INDEX

—  
\_rfstools.arg\_parser, 43  
\_rfstools.arg\_processor, 45

### r

rfslib, 3  
rfslib.abstract\_pconnection, 5  
rfslib.fs\_pconnection, 35  
rfslib.ftp\_pconnection, 17  
rfslib.path\_utils, 41  
rfslib.sftp\_pconnection, 11  
rfslib.smb12\_pconnection, 23  
rfslib.smb23\_pconnection, 29



## Symbols

<code>_exists()</code> ( <i>rfslib.abstract_pconnection.PConnection</i> method), 5	<code>_mkdir()</code> ( <i>rfslib.abstract_pconnection.PConnection</i> method), 5
<code>_exists()</code> ( <i>rfslib.fs_pconnection.FsPConnection</i> method), 35	<code>_mkdir()</code> ( <i>rfslib.fs_pconnection.FsPConnection</i> method), 35
<code>_exists()</code> ( <i>rfslib.ftp_pconnection.FtpPConnection</i> method), 17	<code>_mkdir()</code> ( <i>rfslib.ftp_pconnection.FtpPConnection</i> method), 18
<code>_exists()</code> ( <i>rfslib.sftp_pconnection.SftpPConnection</i> method), 11	<code>_mkdir()</code> ( <i>rfslib.sftp_pconnection.SftpPConnection</i> method), 12
<code>_exists()</code> ( <i>rfslib.smb12_pconnection.Smb12PConnection</i> method), 23	<code>_mkdir()</code> ( <i>rfslib.smb12_pconnection.Smb12PConnection</i> method), 24
<code>_exists()</code> ( <i>rfslib.smb23_pconnection.Smb23PConnection</i> method), 29	<code>_mkdir()</code> ( <i>rfslib.smb23_pconnection.Smb23PConnection</i> method), 30
<code>_isdir()</code> ( <i>rfslib.abstract_pconnection.PConnection</i> method), 5	<code>_pull()</code> ( <i>rfslib.abstract_pconnection.PConnection</i> method), 6
<code>_isdir()</code> ( <i>rfslib.fs_pconnection.FsPConnection</i> method), 35	<code>_pull()</code> ( <i>rfslib.fs_pconnection.FsPConnection</i> method), 36
<code>_isdir()</code> ( <i>rfslib.ftp_pconnection.FtpPConnection</i> method), 17	<code>_pull()</code> ( <i>rfslib.ftp_pconnection.FtpPConnection</i> method), 18
<code>_isdir()</code> ( <i>rfslib.sftp_pconnection.SftpPConnection</i> method), 11	<code>_pull()</code> ( <i>rfslib.sftp_pconnection.SftpPConnection</i> method), 12
<code>_isdir()</code> ( <i>rfslib.smb12_pconnection.Smb12PConnection</i> method), 23	<code>_pull()</code> ( <i>rfslib.smb12_pconnection.Smb12PConnection</i> method), 24
<code>_isdir()</code> ( <i>rfslib.smb23_pconnection.Smb23PConnection</i> method), 29	<code>_pull()</code> ( <i>rfslib.smb23_pconnection.Smb23PConnection</i> method), 30
<code>_lexists()</code> ( <i>rfslib.abstract_pconnection.PConnection</i> method), 5	<code>_push()</code> ( <i>rfslib.abstract_pconnection.PConnection</i> method), 6
<code>_lexists()</code> ( <i>rfslib.fs_pconnection.FsPConnection</i> method), 35	<code>_push()</code> ( <i>rfslib.fs_pconnection.FsPConnection</i> method), 36
<code>_lexists()</code> ( <i>rfslib.ftp_pconnection.FtpPConnection</i> method), 17	<code>_push()</code> ( <i>rfslib.ftp_pconnection.FtpPConnection</i> method), 18
<code>_lexists()</code> ( <i>rfslib.sftp_pconnection.SftpPConnection</i> method), 11	<code>_push()</code> ( <i>rfslib.sftp_pconnection.SftpPConnection</i> method), 12
<code>_lexists()</code> ( <i>rfslib.smb12_pconnection.Smb12PConnection</i> method), 23	<code>_push()</code> ( <i>rfslib.smb12_pconnection.Smb12PConnection</i> method), 24
<code>_lexists()</code> ( <i>rfslib.smb23_pconnection.Smb23PConnection</i> method), 29	<code>_push()</code> ( <i>rfslib.smb23_pconnection.Smb23PConnection</i> method), 30
<code>_listdir()</code> ( <i>rfslib.abstract_pconnection.PConnection</i> method), 5	
<code>_listdir()</code> ( <i>rfslib.fs_pconnection.FsPConnection</i> method), 35	
<code>_listdir()</code> ( <i>rfslib.ftp_pconnection.FtpPConnection</i> method), 17	
<code>_listdir()</code> ( <i>rfslib.sftp_pconnection.SftpPConnection</i> method), 11	
<code>_listdir()</code> ( <i>rfslib.smb12_pconnection.Smb12PConnection</i> method), 23	
<code>_listdir()</code> ( <i>rfslib.smb23_pconnection.Smb23PConnection</i> method), 29	

method), 30

`_rename()` (rfslib.abstract\_pconnection.PConnection method), 6

`_rename()` (rfslib.fs\_pconnection.FsPConnection method), 36

`_rename()` (rfslib.ftp\_pconnection.FtpPConnection method), 18

`_rename()` (rfslib.sftp\_pconnection.SftpPConnection method), 12

`_rename()` (rfslib.smb12\_pconnection.Smb12PConnection method), 24

`_rename()` (rfslib.smb23\_pconnection.Smb23PConnection method), 30

`_rfstools.arg_parser`  
module, 43

`_rfstools.arg_processor`  
module, 45

`_rmdir()` (rfslib.abstract\_pconnection.PConnection method), 6

`_rmdir()` (rfslib.fs\_pconnection.FsPConnection method), 36

`_rmdir()` (rfslib.ftp\_pconnection.FtpPConnection method), 18

`_rmdir()` (rfslib.sftp\_pconnection.SftpPConnection method), 12

`_rmdir()` (rfslib.smb12\_pconnection.Smb12PConnection method), 24

`_rmdir()` (rfslib.smb23\_pconnection.Smb23PConnection method), 30

`_stat()` (rfslib.abstract\_pconnection.PConnection method), 6

`_stat()` (rfslib.fs\_pconnection.FsPConnection method), 36

`_stat()` (rfslib.ftp\_pconnection.FtpPConnection method), 18

`_stat()` (rfslib.sftp\_pconnection.SftpPConnection method), 12

`_stat()` (rfslib.smb12\_pconnection.Smb12PConnection method), 24

`_stat()` (rfslib.smb23\_pconnection.Smb23PConnection method), 30

`_unlink()` (rfslib.abstract\_pconnection.PConnection method), 6

`_unlink()` (rfslib.fs\_pconnection.FsPConnection method), 36

`_unlink()` (rfslib.ftp\_pconnection.FtpPConnection method), 18

`_unlink()` (rfslib.sftp\_pconnection.SftpPConnection method), 12

`_unlink()` (rfslib.smb12\_pconnection.Smb12PConnection method), 24

`_unlink()` (rfslib.smb23\_pconnection.Smb23PConnection method), 30

## A

`add_r_prefix()` (in module `rfslib.path_utils`), 41

## C

`close()` (rfslib.abstract\_pconnection.PConnection method), 7

`close()` (rfslib.fs\_pconnection.FsPConnection method), 37

`close()` (rfslib.ftp\_pconnection.FtpPConnection method), 19

`close()` (rfslib.sftp\_pconnection.SftpPConnection method), 13

`close()` (rfslib.smb12\_pconnection.Smb12PConnection method), 25

`close()` (rfslib.smb23\_pconnection.Smb23PConnection method), 31

`config_smb23()` (in module `rfslib.smb23_pconnection`), 33

`cp()` (rfslib.abstract\_pconnection.PConnection method), 7

`cp()` (rfslib.fs\_pconnection.FsPConnection method), 37

`cp()` (rfslib.ftp\_pconnection.FtpPConnection method), 19

`cp()` (rfslib.sftp\_pconnection.SftpPConnection method), 13

`cp()` (rfslib.smb12\_pconnection.Smb12PConnection method), 25

`cp()` (rfslib.smb23\_pconnection.Smb23PConnection method), 31

## D

`dcp()` (rfslib.abstract\_pconnection.PConnection method), 7

`dcp()` (rfslib.fs\_pconnection.FsPConnection method), 37

`dcp()` (rfslib.ftp\_pconnection.FtpPConnection method), 19

`dcp()` (rfslib.sftp\_pconnection.SftpPConnection method), 13

`dcp()` (rfslib.smb12\_pconnection.Smb12PConnection method), 25

`dcp()` (rfslib.smb23\_pconnection.Smb23PConnection method), 31

`default_arg_parser()` (in module `_rfstools.arg_parser`), 43

`default_dmask` (rfslib.pconnection\_settings attribute), 3

`default_fmask` (rfslib.pconnection\_settings attribute), 3

`direct_write` (rfslib.pconnection\_settings attribute), 3

`dmv()` (rfslib.abstract\_pconnection.PConnection method), 7

`dmv()` (rfslib.fs\_pconnection.FsPConnection method), 37

`dmv()` (rfslib.ftp\_pconnection.FtpPConnection method), 19

`dmv()` (rfslib.sftp\_pconnection.SftpPConnection method), 13

`dmv()` (*rfslib.smb12\_pconnection.Smb12PConnection method*), 25

`dmv()` (*rfslib.smb23\_pconnection.Smb23PConnection method*), 31

## E

`exists()` (*rfslib.abstract\_pconnection.PConnection method*), 7

`exists()` (*rfslib.fs\_pconnection.FsPConnection method*), 37

`exists()` (*rfslib.ftp\_pconnection.FtpPConnection method*), 19

`exists()` (*rfslib.sftp\_pconnection.SftpPConnection method*), 13

`exists()` (*rfslib.smb12\_pconnection.Smb12PConnection method*), 25

`exists()` (*rfslib.smb23\_pconnection.Smb23PConnection method*), 31

## F

`fcv()` (*rfslib.abstract\_pconnection.PConnection method*), 7

`fcv()` (*rfslib.fs\_pconnection.FsPConnection method*), 37

`fcv()` (*rfslib.ftp\_pconnection.FtpPConnection method*), 19

`fcv()` (*rfslib.sftp\_pconnection.SftpPConnection method*), 13

`fcv()` (*rfslib.smb12\_pconnection.Smb12PConnection method*), 25

`fcv()` (*rfslib.smb23\_pconnection.Smb23PConnection method*), 31

`find()` (*rfslib.abstract\_pconnection.PConnection method*), 7

`find()` (*rfslib.fs\_pconnection.FsPConnection method*), 37

`find()` (*rfslib.ftp\_pconnection.FtpPConnection method*), 19

`find()` (*rfslib.sftp\_pconnection.SftpPConnection method*), 13

`find()` (*rfslib.smb12\_pconnection.Smb12PConnection method*), 25

`find()` (*rfslib.smb23\_pconnection.Smb23PConnection method*), 31

`fmv()` (*rfslib.abstract\_pconnection.PConnection method*), 7

`fmv()` (*rfslib.fs\_pconnection.FsPConnection method*), 37

`fmv()` (*rfslib.ftp\_pconnection.FtpPConnection method*), 19

`fmv()` (*rfslib.sftp\_pconnection.SftpPConnection method*), 13

`fmv()` (*rfslib.smb12\_pconnection.Smb12PConnection method*), 25

`fmv()` (*rfslib.smb23\_pconnection.Smb23PConnection method*), 31

`FsPConnection` (*class in rfslib.fs\_pconnection*), 35

`FtpPConnection` (*class in rfslib.ftp\_pconnection*), 17

## G

`generic_cp()` (*in module rfslib.path\_utils*), 41

`generic_mv()` (*in module rfslib.path\_utils*), 41

`generic_path_normalize()` (*in module rfslib.path\_utils*), 41

`GenericPath` (*class in rfslib.path\_utils*), 41

`get_default_dmask()` (*rfslib.abstract\_pconnection.PConnection method*), 7

`get_default_dmask()` (*rfslib.fs\_pconnection.FsPConnection method*), 37

`get_default_dmask()` (*rfslib.ftp\_pconnection.FtpPConnection method*), 19

`get_default_dmask()` (*rfslib.sftp\_pconnection.SftpPConnection method*), 13

`get_default_dmask()` (*rfslib.smb12\_pconnection.Smb12PConnection method*), 25

`get_default_dmask()` (*rfslib.smb23\_pconnection.Smb23PConnection method*), 31

`get_default_fmask()` (*rfslib.abstract\_pconnection.PConnection method*), 7

`get_default_fmask()` (*rfslib.fs\_pconnection.FsPConnection method*), 37

`get_default_fmask()` (*rfslib.ftp\_pconnection.FtpPConnection method*), 19

`get_default_fmask()` (*rfslib.sftp\_pconnection.SftpPConnection method*), 13

`get_default_fmask()` (*rfslib.smb12\_pconnection.Smb12PConnection method*), 25

`get_default_fmask()` (*rfslib.smb23\_pconnection.Smb23PConnection method*), 31

`get_settings()` (*rfslib.abstract\_pconnection.PConnection method*), 7

`get_settings()` (*rfslib.fs\_pconnection.FsPConnection method*), 37

`get_settings()` (*rfslib.ftp\_pconnection.FtpPConnection method*), 19

`get_settings()` (*rfslib.sftp\_pconnection.SftpPConnection method*), 13

[get\\_settings\(\)](#) (*rfslib.smb12\_pconnection.Smb12PConnection method*), 25  
[get\\_settings\(\)](#) (*rfslib.smb23\_pconnection.Smb23PConnection method*), 31  
**I**  
[init\(\)](#) (in module *\_rfstools.arg\_processor*), 45  
[is\\_remote\(\)](#) (in module *rfslib.path\_utils*), 41  
[isdir\(\)](#) (*rfslib.abstract\_pconnection.PConnection method*), 7  
[isdir\(\)](#) (*rfslib.fs\_pconnection.FsPConnection method*), 37  
[isdir\(\)](#) (*rfslib.ftp\_pconnection.FtpPConnection method*), 19  
[isdir\(\)](#) (*rfslib.sftp\_pconnection.SftpPConnection method*), 13  
[isdir\(\)](#) (*rfslib.smb12\_pconnection.Smb12PConnection method*), 25  
[isdir\(\)](#) (*rfslib.smb23\_pconnection.Smb23PConnection method*), 31  
**L**  
[lexists\(\)](#) (*rfslib.abstract\_pconnection.PConnection method*), 7  
[lexists\(\)](#) (*rfslib.fs\_pconnection.FsPConnection method*), 37  
[lexists\(\)](#) (*rfslib.ftp\_pconnection.FtpPConnection method*), 19  
[lexists\(\)](#) (*rfslib.sftp\_pconnection.SftpPConnection method*), 13  
[lexists\(\)](#) (*rfslib.smb12\_pconnection.Smb12PConnection method*), 25  
[lexists\(\)](#) (*rfslib.smb23\_pconnection.Smb23PConnection method*), 31  
[listdir\(\)](#) (*rfslib.abstract\_pconnection.PConnection method*), 8  
[listdir\(\)](#) (*rfslib.fs\_pconnection.FsPConnection method*), 38  
[listdir\(\)](#) (*rfslib.ftp\_pconnection.FtpPConnection method*), 20  
[listdir\(\)](#) (*rfslib.sftp\_pconnection.SftpPConnection method*), 14  
[listdir\(\)](#) (*rfslib.smb12\_pconnection.Smb12PConnection method*), 26  
[listdir\(\)](#) (*rfslib.smb23\_pconnection.Smb23PConnection method*), 32  
[local\\_crlf](#) (*rfslib.pconnection\_settings* attribute), 3  
[local\\_encoding](#) (*rfslib.pconnection\_settings* attribute), 3  
[ls\(\)](#) (*rfslib.abstract\_pconnection.PConnection method*), 8  
[ls\(\)](#) (*rfslib.fs\_pconnection.FsPConnection method*), 38  
[ls\(\)](#) (*rfslib.ftp\_pconnection.FtpPConnection method*), 20  
[ls\(\)](#) (*rfslib.sftp\_pconnection.SftpPConnection method*), 14  
[ls\(\)](#) (*rfslib.smb12\_pconnection.Smb12PConnection method*), 26  
[ls\(\)](#) (*rfslib.smb23\_pconnection.Smb23PConnection method*), 32  
**M**  
[many\\_to\\_one\\_arg\\_parser\(\)](#) (in module *\_rfstools.arg\_parser*), 43  
[mkdir\(\)](#) (*rfslib.abstract\_pconnection.PConnection method*), 8  
[mkdir\(\)](#) (*rfslib.fs\_pconnection.FsPConnection method*), 38  
[mkdir\(\)](#) (*rfslib.ftp\_pconnection.FtpPConnection method*), 20  
[mkdir\(\)](#) (*rfslib.sftp\_pconnection.SftpPConnection method*), 14  
[mkdir\(\)](#) (*rfslib.smb12\_pconnection.Smb12PConnection method*), 26  
[mkdir\(\)](#) (*rfslib.smb23\_pconnection.Smb23PConnection method*), 32  
**module**  
[\\_rfstools.arg\\_parser](#), 43  
[\\_rfstools.arg\\_processor](#), 45  
[rfslib](#), 3  
[rfslib.abstract\\_pconnection](#), 5  
[rfslib.fs\\_pconnection](#), 35  
[rfslib.ftp\\_pconnection](#), 17  
[rfslib.path\\_utils](#), 41  
[rfslib.sftp\\_pconnection](#), 11  
[rfslib.smb12\\_pconnection](#), 23  
[rfslib.smb23\\_pconnection](#), 29  
[mv\(\)](#) (*rfslib.abstract\_pconnection.PConnection method*), 8  
[mv\(\)](#) (*rfslib.fs\_pconnection.FsPConnection method*), 38  
[mv\(\)](#) (*rfslib.ftp\_pconnection.FtpPConnection method*), 20  
[mv\(\)](#) (*rfslib.sftp\_pconnection.SftpPConnection method*), 14  
[mv\(\)](#) (*rfslib.smb12\_pconnection.Smb12PConnection method*), 26



- `mv()` (*rfslib.smb23\_pconnection.Smb23PConnection* method), 32
- ## O
- `one_arg_parser()` (in module *rfstools.arg\_parser*), 43
- `oneplus_arg_parser()` (in module *\_rfstools.arg\_parser*), 43
- ## P
- `p_stat_result` (class in *rfslib.abstract\_pconnection*), 9
- `path_normalize()` (in module *rfslib.path\_utils*), 41
- `PConnection` (class in *rfslib.abstract\_pconnection*), 5
- `pconnection_settings` (class in *rfslib*), 3
- `pmkdir()` (*rfslib.abstract\_pconnection.PConnection* method), 8
- `pmkdir()` (*rfslib.fs\_pconnection.FsPConnection* method), 38
- `pmkdir()` (*rfslib.ftp\_pconnection.FtpPConnection* method), 20
- `pmkdir()` (*rfslib.sftp\_pconnection.SftpPConnection* method), 14
- `pmkdir()` (*rfslib.smb12\_pconnection.Smb12PConnection* method), 26
- `pmkdir()` (*rfslib.smb23\_pconnection.Smb23PConnection* method), 32
- `pull()` (*rfslib.abstract\_pconnection.PConnection* method), 8
- `pull()` (*rfslib.fs\_pconnection.FsPConnection* method), 38
- `pull()` (*rfslib.ftp\_pconnection.FtpPConnection* method), 20
- `pull()` (*rfslib.sftp\_pconnection.SftpPConnection* method), 14
- `pull()` (*rfslib.smb12\_pconnection.Smb12PConnection* method), 26
- `pull()` (*rfslib.smb23\_pconnection.Smb23PConnection* method), 32
- `push()` (*rfslib.abstract\_pconnection.PConnection* method), 8
- `push()` (*rfslib.fs\_pconnection.FsPConnection* method), 38
- `push()` (*rfslib.ftp\_pconnection.FtpPConnection* method), 20
- `push()` (*rfslib.sftp\_pconnection.SftpPConnection* method), 14
- `push()` (*rfslib.smb12\_pconnection.Smb12PConnection* method), 26
- `push()` (*rfslib.smb23\_pconnection.Smb23PConnection* method), 32
- ## R
- `remote_crlf` (*rfslib.pconnection\_settings* attribute), 3
- `remote_encoding` (*rfslib.pconnection\_settings* attribute), 3
- `remove_r_prefix()` (in module *rfslib.path\_utils*), 41
- `rename()` (*rfslib.abstract\_pconnection.PConnection* method), 8
- `rename()` (*rfslib.fs\_pconnection.FsPConnection* method), 38
- `rename()` (*rfslib.ftp\_pconnection.FtpPConnection* method), 20
- `rename()` (*rfslib.sftp\_pconnection.SftpPConnection* method), 14
- `rename()` (*rfslib.smb12\_pconnection.Smb12PConnection* method), 26
- `rename()` (*rfslib.smb23\_pconnection.Smb23PConnection* method), 32
- rfslib**  
module, 3
- rfslib.abstract\_pconnection**  
module, 5
- rfslib.fs\_pconnection**  
module, 35
- rfslib.ftp\_pconnection**  
module, 17
- rfslib.path\_utils**  
module, 41
- rfslib.sftp\_pconnection**  
module, 11
- rfslib.smb12\_pconnection**  
module, 23
- rfslib.smb23\_pconnection**  
module, 29
- `rm()` (*rfslib.abstract\_pconnection.PConnection* method), 8
- `rm()` (*rfslib.fs\_pconnection.FsPConnection* method), 38
- `rm()` (*rfslib.ftp\_pconnection.FtpPConnection* method), 20
- `rm()` (*rfslib.sftp\_pconnection.SftpPConnection* method), 14
- `rm()` (*rfslib.smb12\_pconnection.Smb12PConnection* method), 26
- `rm()` (*rfslib.smb23\_pconnection.Smb23PConnection* method), 32
- `rmdir()` (*rfslib.abstract\_pconnection.PConnection* method), 8
- `rmdir()` (*rfslib.fs\_pconnection.FsPConnection* method), 38
- `rmdir()` (*rfslib.ftp\_pconnection.FtpPConnection* method), 20
- `rmdir()` (*rfslib.sftp\_pconnection.SftpPConnection* method), 14
- `rmdir()` (*rfslib.smb12\_pconnection.Smb12PConnection* method), 26
- `rmdir()` (*rfslib.smb23\_pconnection.Smb23PConnection* method), 32
- `rpull()` (*rfslib.abstract\_pconnection.PConnection* method), 8

rpull() (rflib.fs\_pconnection.FsPConnection method), 38  
 rpull() (rflib.ftp\_pconnection.FtpPConnection method), 20  
 rpull() (rflib.sftp\_pconnection.SftpPConnection method), 14  
 rpull() (rflib.smb12\_pconnection.Smb12PConnection method), 26  
 rpull() (rflib.smb23\_pconnection.Smb23PConnection method), 32  
 rpush() (rflib.abstract\_pconnection.PConnection method), 8  
 rpush() (rflib.fs\_pconnection.FsPConnection method), 38  
 rpush() (rflib.ftp\_pconnection.FtpPConnection method), 20  
 rpush() (rflib.sftp\_pconnection.SftpPConnection method), 14  
 rpush() (rflib.smb12\_pconnection.Smb12PConnection method), 26  
 rpush() (rflib.smb23\_pconnection.Smb23PConnection method), 32

## S

set\_settings() (rflib.abstract\_pconnection.PConnection method), 8  
 set\_settings() (rflib.fs\_pconnection.FsPConnection method), 38  
 set\_settings() (rflib.ftp\_pconnection.FtpPConnection method), 21  
 set\_settings() (rflib.sftp\_pconnection.SftpPConnection method), 15  
 set\_settings() (rflib.smb12\_pconnection.Smb12PConnection method), 27  
 set\_settings() (rflib.smb23\_pconnection.Smb23PConnection method), 33  
 SftpPConnection (class in rflib.sftp\_pconnection), 11  
 skip\_validation (rflib.pconnection\_settings attribute), 3  
 Smb12PConnection (class in rflib.smb12\_pconnection), 23  
 Smb23PConnection (class in rflib.smb23\_pconnection), 29  
 st\_atime (rflib.abstract\_pconnection.p\_stat\_result attribute), 9  
 st\_gid (rflib.abstract\_pconnection.p\_stat\_result attribute), 9  
 st\_mode (rflib.abstract\_pconnection.p\_stat\_result attribute), 9  
 st\_mtime (rflib.abstract\_pconnection.p\_stat\_result attribute), 9  
 st\_nlink (rflib.abstract\_pconnection.p\_stat\_result attribute), 9  
 st\_size (rflib.abstract\_pconnection.p\_stat\_result attribute), 9  
 st\_uid (rflib.abstract\_pconnection.p\_stat\_result attribute), 9  
 stat() (rflib.abstract\_pconnection.PConnection method), 8  
 stat() (rflib.fs\_pconnection.FsPConnection method), 38  
 stat() (rflib.ftp\_pconnection.FtpPConnection method), 21  
 stat() (rflib.sftp\_pconnection.SftpPConnection method), 15  
 stat() (rflib.smb12\_pconnection.Smb12PConnection method), 27  
 stat() (rflib.smb23\_pconnection.Smb23PConnection method), 33

## T

text\_transmission (rflib.pconnection\_settings attribute), 3  
 touch() (rflib.abstract\_pconnection.PConnection method), 9  
 touch() (rflib.fs\_pconnection.FsPConnection method), 39  
 touch() (rflib.ftp\_pconnection.FtpPConnection method), 21  
 touch() (rflib.sftp\_pconnection.SftpPConnection method), 15  
 touch() (rflib.smb12\_pconnection.Smb12PConnection method), 27  
 touch() (rflib.smb23\_pconnection.Smb23PConnection method), 33

## U

unlink() (rflib.abstract\_pconnection.PConnection method), 9  
 unlink() (rflib.fs\_pconnection.FsPConnection method), 39  
 unlink() (rflib.ftp\_pconnection.FtpPConnection method), 21  
 unlink() (rflib.sftp\_pconnection.SftpPConnection method), 15  
 unlink() (rflib.smb12\_pconnection.Smb12PConnection method), 27  
 unlink() (rflib.smb23\_pconnection.Smb23PConnection method), 33

## X

xls() (rflib.abstract\_pconnection.PConnection method), 9  
 xls() (rflib.fs\_pconnection.FsPConnection method), 39  
 xls() (rflib.ftp\_pconnection.FtpPConnection method), 21

`xls()` (*rfslib.sftp\_pconnection.SftpPConnection*  
*method*), [15](#)

`xls()` (*rfslib.smb12\_pconnection.Smb12PConnection*  
*method*), [27](#)

`xls()` (*rfslib.smb23\_pconnection.Smb23PConnection*  
*method*), [33](#)