PyTest local FTP Server Documentation

Release 1.0.1

Oz Tiram

Jan 27, 2021

Contents

Contents:

CHAPTER 1

PyTest FTP Server

A PyTest plugin which provides an FTP fixture for your tests

- Free software: MIT license
- Documentation: https://pytest-localftpserver.readthedocs.io/en/latest/index.html

1.1 Attention!

As of version `1.0.0` the support for python 2.7 and 3.4 was dropped. If you need to support those versions you should pin the version to `0.6.0`, i.e. add the following lines to your "requirements_dev.txt":

```
# pytest_localftpserver==0.6.0
https://github.com/oz123/pytest-localftpserver/archive/v0.6.0.zip
```

1.2 Usage Quickstart:

This Plugin provides the fixtures `ftpserver` and `ftpserver_TLS`, which are threaded instances of a FTP server, with which you can upload files and test FTP functionality. It can be configured using the following environment variables:

Environment variable	Usage
FTP_USER	Username of the registered user.
FTP_PASS	Password of the registered user.
FTP_PORT	Port for the normal ftp server to run on.
FTP_HOME	Home folder (host system) of the registered user.
FTP_FIXTURE_SCOPE	Scope/lifetime of the fixture.
FTP_PORT_TLS	Port for the TLS ftp server to run on.
FTP_HOME_TLS	Home folder (host system) of the registered user, used by the TLS ftp server.
FTP_CERTFILE	Certificate (host system) to be used by the TLS ftp server.

See the tests directory or the documentation for examples.

You can either set environment variables on a system level or use tools such as pytest-env or tox, to change the default settings of this plugin. Sample config for pytest-cov:

```
$ cat pytest.ini
[pytest]
env =
    FTP_USER=benz
    FTP_PASS=erni1
    FTP_HOME = /home/ftp_test
    FTP_PORT=31175
    FTP_FIXTURE_SCOPE=function
    # only affects ftpserver_TLS
    FTP_PORT_TLS = 31176
    FTP_HOME_TLS = /home/ftp_test_TLS
    FTP_CERTFILE = ./tests/test_keycert.pem
```

Sample config for Tox:

```
$ cat tox.ini
[tox]
envlist = py{35,36,37}

[testenv]
setenv =
    FTP_USER=benz
    FTP_PASS=erni1
    FTP_HOME = {envtmpdir}
    FTP_PORT=31175
    FTP_FIXTURE_SCOPE=function
    # only affects ftpserver_TLS
    FTP_PORT_TLS = 31176
    FTP_HOME_TLS = /home/ftp_test_TLS
    FTP_CERTFILE = {toxinidir}/tests/test_keycert.pem
commands =
    py.test tests
```

1.3 Credits

This package was inspired by, https://pypi.org/project/pytest-localserver/ made by Sebastian Rahlf, which lacks an FTP server.

This package was created with Cookiecutter and the audreyr/cookiecutter-pypackage project template.

CHAPTER 2

Installation

2.1 Stable release

To install PyTest FTP Server, run this command in your terminal:

```
$ pip install pytest-localftpserver
```

If you don't have pip installed, this Python installation guide can guide you through the process.

Or if you prefer to use conda:

```
$ conda install -c conda-forge pytest-localftpserver
```

This are the preferred methods to install PyTest FTP Server, as it will always install the most recent stable release.

2.2 From sources

The sources for PyTest FTP Server can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/oz123/pytest-localftpserver
```

Or download the tarball:

```
$ curl  -OL https://github.com/oz123/pytest-localftpserver/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

CHAPTER 3

Usage

After installing *pytest_localftpserver* the fixture `ftpserver` is available for your pytest test functions. Note that you can't use fixtures outside of functions and need to pass them as arguments.

3.1 Basic usage

A basic example of using *pytest_localftpserver* would be, if you wanted to test code, which uploads a file to a FTP-server.

```
import os


def test_your_code_to_upload_files(ftpserver):
    your_code_to_upload_files(host="localhost",
                              port=ftpserver.server_port,
                              username=ftpserver.username,
                              password=ftpserver.password,
                              files=["testfile.txt"])

    uploaded_file_path = os.path.join(ftpserver.server_home, "testfile.txt")
    with open("testfile.txt") as original, open(uploaded_file_path) as uploaded:
        assert original.read() == uploaded.read()
```

Note: Like most public FTP-servers *pytest_localftpserver* doesn't allow the anonymous user to upload files. The anonymous user is only allowed to browse the folder structure and download files. If you want to upload files you need to use the registered user, with its password.

An other common use case would be retrieving a file from a FTP-server.

```
import os
from shutil import copyfile
```

(continues on next page)

(continued from previous page)

```
def test_your_code_retrieving_files(ftpserver):
    dest_path = os.path.join(ftpserver.anon_root, "testfile.txt")
    copyfile("testfile.txt", dest_path)
    your_code_retrieving_files(host="localhost",
                               port=ftpserver.server_port
                               file_paths=[{"remote": "testfile.txt",
                                            "local": "testfile_downloaded.txt"
                                            }])
    with open("testfile.txt") as original, open("testfile_downloaded.txt") as␣
↪downloaded:
        assert original.read() == downloaded.read()
```

3.1.1 Login with the TLS server

Since Python 2 and 3, as well as different versions of python 3 differ in their implementation of the client `FTP_TLS` and/or ssl protocol/context, we provide you with an example which works on python 2.7, 3.4, 3.5 3.6 and 3.7 (note that, this example utilizes methods of the the high-level interface, which are explained in *Getting login credentials* and *Gaining information about the content of files on the server*).

The below example test logs into the TLS ftpserver, creates the file `testfile.txt`, with content 'test text' and checks if it was written properly.

```
import sys
from ftplib import FTP_TLS

if sys.version_info[0] == 3:
    PYTHON3 = True
else:
    PYTHON3 = False

if PYTHON3:
    from ssl import SSLContext
    try:
        from ssl import PROTOCOL_TLS
    except Exception:
        from ssl import PROTOCOL_SSLv23 as PROTOCOL_TLS


def test_TLS_login(ftpserver_TLS):
    if PYTHON3:
        ssl_context = SSLContext(PROTOCOL_TLS)
        ssl_context.load_cert_chain(certfile=DEFAULT_CERTFILE)
        ftp = FTP_TLS(context=ssl_context)
    else:
        ftp = FTP_TLS(certfile=DEFAULT_CERTFILE)

    login_dict = ftpserver_TLS.get_login_data()
    ftp.connect(login_dict["host"], login_dict["port"])
    ftp.login(login_dict["user"], login_dict["passwd"])
    ftp.prot_p()
    ftp.cwd("/")
    filename = "testfile.txt"
    file_path_local = tmpdir.join(filename)
```

(continues on next page)

(continued from previous page)

```
    file_path_local.write("test text")
    with open(str(file_path_local), "rb") as f:
        ftp.storbinary("STOR "+filename, f)
    ftp.quit()
    file_list = list(ftpserver_TLS.get_file_contents()
    assert file_list == [{"path": "testfile.txt", "content": "test text"}]
```

3.2 High-Level Interface

To allow you a faster and more comfortable handling of common ftp tasks a high-level interface was implemented. Most of the following methods have the keyword `anon`, which allows to switch between the registered (*anon=False*) and the anonymous (*anon=True*) user. For more information on how those methods work, take a look at the *API Documentation* .

Note: The following examples aren't working code, since the aren't called from within a function, which means that the `ftpserver` fixture isn't available. They are thought to be a quick overview of the available functionality and its output.

3.2.1 Getting login credentials

To quickly get all needed login data you can use `get_login_data`, which will either return a dict or an url to log into the ftp:

```
>>> ftpserver.get_login_data()
{"host": "localhost", "port": 8888, "user": "fakeusername", "passwd": "qweqwe"}

>>> ftpserver.get_login_data(style="url", anon=False)
ftp://fakeusername:qweqwe@localhost:8888

>>> ftpserver.get_login_data(style="url", anon=True)
ftp://localhost:8888
```

3.2.2 Populating the FTP server with files and folders

To test ftp download capabilities of your code, you might want to populate the files on the server. To "upload" files to the server you can use the method `put_files`:

```
>>> ftpserver.put_files("test_folder/test_file", style="rel_path", anon=False)
["test_file"]

>>> ftpserver.put_files("test_folder/test_file", style="url", anon=False)
["ftp://fakeusername:qweqwe@localhost:8888/test_file"]

>>> ftpserver.put_files("test_folder/test_file", style="url", anon=True)
["ftp://localhost:8888/test_file"]

>>> ftpserver.put_files({"src": "test_folder/test_file",
...                      "dest": "remote_folder/uploaded_file"},
...                     style="url", anon=True)
```

(continues on next page)

(continued from previous page)

```
["ftp://localhost:8888/remote_folder/uploaded_file"]

>>> ftpserver.put_files("test_folder/test_file", return_content=True)
[{"path": "test_file", "content": "some text in test_file"}]

>>> ftpserver.put_files("test_file.zip", return_content=True, read_mode="rb")
[{"path": "test_file.zip", "content": b'PK\\x03\\x04\\x14\\x00\\x00...'}]

>>> ftpserver.put_files("test_file", return_paths="new")
UserWarning: test_file does already exist and won't be overwritten.
    Set `overwrite` to True to overwrite it anyway.
[]

>>> ftpserver.put_files("test_file", return_paths="new", overwrite=True)
["test_file"]

>>> ftpserver.put_files("test_file3", return_paths="all")
["test_file", "remote_folder/uploaded_file", "test_file.zip"]
```

3.2.3 Resetting files on the server

Since `ftpserver` is a module scope fixture, you might want to make sure that uploaded files get deleted after/before a test. This can be done by using the method `reset_tmp_dirs`.

filesystem before:

```
+---server_home
|   +---test_file1
|   +---test_folder
|       +---test_file2
|
+---anon_root
    +---test_file3
    +---test_folder
        +---test_file4
```

```
>>> ftpserver.reset_tmp_dirs()
```

filesystem after:

```
+---server_home
|
+---anon_root
```

3.2.4 Gaining information on which files are on the server

If you want to know which files are on the server, i.e. if you want to know if your file upload functionality is working, you can use the `get_file_paths` method, which will yield the paths to all files on the server.

```
filesystem
+---server_home
|   +---test_file1
|   +---test_folder
```

(continues on next page)

(continued from previous page)

```
|       +---test_file2
|
+---anon_root
    +---test_file3
    +---test_folder
        +---test_file4
```

```
>>> list(ftpserver.get_file_paths(style="rel_path", anon=False))
["test_file1", "test_folder/test_file2"]

>>> list(ftpserver.get_file_paths(style="rel_path", anon=True))
["test_file3", "test_folder/test_file4"]
```

3.2.5 Gaining information about the content of files on the server

If you are interested in the content of a specific file, multiple files or all files, i.e. to verify that your file upload functionality did work properly, you can use the `get_file_contents` method.

```
filesystem
+---server_home
    +---test_file1.txt
    +---test_folder
        +---test_file2.zip
```

```
>>> list(ftpserver.get_file_contents())
[{"path": "test_file1.txt", "content": "test text"},
 {"path": "test_folder/test_file2.txt", "content": "test text2"}]

>>> list(ftpserver.get_file_contents("test_file1.txt"))
[{"path": "test_file1.txt", "content": "test text"}]

>>> list(ftpserver.get_file_contents("test_file1.txt", style="url"))
[{"path": "ftp://fakeusername:qweqwe@localhost:8888/test_file1.txt",
  "content": "test text"}]

>>> list(ftpserver.get_file_contents(["test_file1.txt", "test_folder/test_file2.zip"],
...                                  read_mode="rb"))
[{"path": "test_file1.txt", "content": b"test text"},
 {"path": "test_folder/test_file2.zip", "content": b'PK\\x03\\x04\\x14\\x00\\x00...'}]
```

3.3 Configuration

To configure custom values for for the username, the users password, the ftp port and/or the location of the users home folder on the local storage, you need to set the environment variables `FTP_USER`, `FTP_PASS`, `FTP_PORT`, `FTP_HOME`, `FTP_FIXTURE_SCOPE`, `FTP_PORT_TLS`, `FTP_HOME_TLS` and `FTP_CERTFILE`.

Environment variable	Usage
FTP_USER	Username of the registered user.
FTP_PASS	Password of the registered user.
FTP_PORT	Port for the normal ftp server to run on.
FTP_HOME	Home folder (host system) of the registered user.
FTP_FIXTURE_SCOPE	Scope/lifetime of the fixture.
FTP_PORT_TLS	Port for the TLS ftp server to run on.
FTP_HOME_TLS	Home folder (host system) of the registered user, used by the TLS ftp server.
FTP_CERTFILE	Certificate (host system) to be used by the TLS ftp server.

You can either set environment variables on a system level or use tools such as pytest-env or tox, which would be the recommended way.

Note: You might run into `OSError: [Errno 48] Address already in use` when setting a fixed port (`FTP_PORT`/ `FTP_PORT_TLS`). This is due to the server still listening on that port, which prevents it from adding another listener on that port. When using pythons buildin `ftplib`, you should use the quit method to terminate the connection, since it's the *'the "polite" way to close a connection'* and lets the server know that the client isn't just experiencing connection problems, but won't come back.

3.3.1 Configuration with pytest-env

The configuration of pytest-env is done in the `pytest.ini` file. The following example configuration will use the username `benz`, the password `erni1`, the ftp port `31175` and the home folder `/home/ftp_test`. For the encrypted version of the fixture it uses port `31176`, the home folder `/home/ftp_test` and the certificate `./tests/test_keycert.pem`:

```
$ cat pytest.ini
[pytest]
env =
    FTP_USER=benz
    FTP_PASS=erni1
    FTP_HOME = /home/ftp_test
    FTP_PORT=31175
    FTP_FIXTURE_SCOPE=function
    # only affects ftpserver_TLS
    FTP_PORT_TLS = 31176
    FTP_HOME_TLS = /home/ftp_test_TLS
    FTP_CERTFILE = ./tests/test_keycert.pem
```

3.3.2 Configuration with Tox

The configuration of tox is done in the `tox.ini` file. The following example configuration will run the tests in the folder `tests` on python 3.5, 3.6 and 3.7 and use the username `benz`, the password `erni1`, the tempfolder of each virtual environment the tests are run in (`{envtmpdir}`) and the ftp port `31175`. For the encrypted version of the fixture it uses port `31176` and the certificate `{toxinidir}/tests/test_keycert.pem`:

```
$ cat tox.ini
[tox]
envlist = py{35,36,37}
```

(continues on next page)

(continued from previous page)

```
[testenv]
setenv =
    FTP_USER=benz
    FTP_PASS=erni1
    FTP_HOME = {envtmpdir}
    FTP_PORT=31175
    FTP_FIXTURE_SCOPE=function
    # only affects ftpserver_TLS
    FTP_PORT_TLS = 31176
    FTP_HOME_TLS = /home/ftp_test_TLS
    FTP_CERTFILE = {toxinidir}/tests/test_keycert.pem
commands =
    py.test tests
```

CHAPTER 4

API Documentation

This is the detailed documentation of `FunctionalityWrapper`, which holds all the functionality you gain by `PyTest local FTP Server`.

FunctionalityWrapper	Baseclass which holds the functionality of ftpserver.

4.1 FunctionalityWrapper

class FunctionalityWrapper(*use_TLS=False*)

Baseclass which holds the functionality of ftpserver. The derived classes are ThreadFTPServer and ProcessFTPServer, which (depending on the OS) are the classes of the ftpserver instance.

Parameters **use_TLS** (*bool*) – Whether or not to use TLS/SSL encryption.

Notes

For custom configuration the following environment variables can be used:

General:

FTP_USER: str Name of the registered user.

FTP_PASS: str Password of the registered user.

FTP_HOME: str Local path to FTP home for the registered user.

FTP_PORT: int Desired port for the unencrypted server to listen to.

FTP_FIXTURE_SCOPE: {'function', 'module', 'session'}: default 'module' Scope the fixture will be in.

TLS only:

FTP_HOME_TLS = str Local path to FTP home for the registered user of the encrypted server.

FTP_PORT_TLS: int Desired port for the encrypted server to listen to.

FTP_CERTFILE: str Path to the certificate used by the encrypted server.

Attributes Summary

anon_root	Local path to FTP home for the anonymous user.
cert_path	Path to the used certificate File.
password	Password of the registered user.
server_home	Local path to FTP home for the registered user.
server_port	Port the server is running on.
username	Name of the registered user.
uses_TLS	Weather or not the server uses TLS/SSL encryption.

Methods Summary

format_file_path	Formats the relative path to as relative path or url.
get_cert	Returns the path to the used certificate or its content as string or bytes.
get_file_contents	Yields dicts containing the *path* and *content* of files on the FTP server.
get_file_paths	Yields the paths of all files server_home/anon_root, in the given *style*.
get_local_base_path	Returns the basepath on the local file system.
get_login_data	Returns the login data as dict or url.
put_files	Copies the files defined in *files_on_local* to the sever.
reset_tmp_dirs	Clears all temp files generated on the FTP server.
stop	Stops the server, closes all the open ports and deletes all temp files.

4.1.1 format_file_path

FunctionalityWrapper.**format_file_path**(*rel_file_path*, *style='rel_path'*, *anon=False*)

Formats the relative path to as relative path or url. Relative paths are relative to the server_home/anon_root, which can be used by a FTP client. Urls can be used by a browser/downloader. This method works, weather the file exists or not.

Notes

Even so taking a relative path and returning a relative path may seam pointless, this is needed to prevent errors with Windows path formatting (\ instead of /).

Parameters

- **rel_file_path** (*str*) – Relative filepath to server_home/anon_root depending on the value of anon.
- **style** (*{'rel_path', 'url'}, default 'rel_path'*) –

  'rel_path': path relative to server_home/anon_root is returned.

  'url': url to the file is returned.

- **anon** (*bool*) –

  True: return the filepaths/url of file in anon_root

  False: return the filepaths/url of file in server_home

Returns **file_path** – Relative path or url depending on the value of style

Return type str

Raises

- `TypeError` – If *style* is not a `str`
- `TypeError` – If *anon* is not a `bool`
- `ValueError` – If the value of *style* is not 'rel_path' or 'url'

Examples

```
>>> ftpserver.format_file_path("test_folder\test_file", style="rel_path",␣
↪anon=False))
test_folder/test_file
```

```
>>> ftpserver.format_file_path("test_folder/test_file", style="rel_path",␣
↪anon=False))
test_folder/test_file
```

```
>>> ftpserver.format_file_path("test_folder/test_file", style="url",␣
↪anon=False))
ftp://fakeusername:qweqwe@localhost:8888/test_folder/test_file
```

```
>>> ftpserver.format_file_path("test_folder/test_file", style="url",␣
↪anon=True))
ftp://localhost:8888/test_folder/test_file
```

See also:

get_local_base_path(), *get_file_paths()*

4.1.2 get_cert

`FunctionalityWrapper.`**`get_cert`**(*style='path'*, *read_mode='r'*)

Returns the path to the used certificate or its content as string or bytes.

Parameters

- **style** (*{'path', 'content'}, default 'path'*) – List of filepaths/content dicts in server_home/anon_root
- **read_mode** (*{'r', 'rb'}, default 'r'*) – This only applies if *style* is 'content'. Mode in which files should be read (see `open("filepath", read_mode)`)

Returns **cert** – Path to or content of the used certificate

Return type str

Raises

- `TypeError` – If *style* is not a `str`
- `TypeError` – If *read_mode* is not a `str`
- `ValueError` – If the value of *style* is not 'path' or 'content'
- `ValueError` – If the value of *read_mode* is not 'r' or 'rb'
- `WrongFixtureError` – If used on `ftpserver` fixture, instead of `ftpserver_TLS` fixture.

Examples

```
>>> ftpserver_TLS.get_cert()
"/home/certs/TLS_cert.pem"
```

```
>>> ftpserver_TLS.get_cert(style="content")
"-----BEGIN RSA PRIVATE KEY-----\nMIICXw..."
```

```
>>> ftpserver_TLS.get_cert(style="content", read_mode="rb")
b"-----BEGIN RSA PRIVATE KEY-----\nMIICXw..."
```

4.1.3 get_file_contents

`FunctionalityWrapper.`**`get_file_contents`**(*rel_file_paths=None*, *style='rel_path'*, *anon=False*, *read_mode='r'*)

Yields dicts containing the *path* and *content* of files on the FTP server.

Parameters

- **`rel_file_paths`** (*`str, list of str, None, default None`*) –

  None: The content of all files on the server will be retrieved.

  str or list of str: Only the content of those files will be retrieved.

- **`style`** (*`{'rel_path', 'url'}, default 'rel_path'`*) –

  'rel_path': Path relative to server_home/anon_root is returned.

  'url': A url to the file is returned.

- **`anon`** (*`bool`*) –

  True: return the filepaths/url of files in anon_root

  False: return the filepaths/url of files in in server_home

- **`read_mode`** (*`{'r', 'rb'}, default 'r'`*) – Mode in which files should be read (see `open("filepath", read_mode)`)

Yields **content_dict** (*dict*) – Dict containing the file *path* as relpath or url (see *style*) and the *content* of the file as string or bytes (see *read_mode*)

Raises

- `TypeError` – If *rel_file_paths* is not `None`, a `str` or an `iterable`
- `TypeError` – If *style* is not a `str`
- `TypeError` – If *anon* is not a `bool`

- `TypeError` – If *read_mode* is not a `str`
- `ValueError` – If the value of *rel_file_paths* or its items are not valid filepaths
- `ValueError` – If the value of *style* is not 'rel_path' or 'url'
- `ValueError` – If the value of *read_mode* is not 'r' or 'rb'

Examples

Assuming a file structure as follows.

```
filesystem
+---server_home
    +---test_file1.txt
    +---test_folder
        +---test_file2.zip
```

```
>>> list(ftpserver.get_file_contents())
[{"path": "test_file1.txt", "content": "test text"},
 {"path": "test_folder/test_file2.txt", "content": "test text2"}]
```

```
>>> list(ftpserver.get_file_contents("test_file1.txt"))
[{"path": "test_file1.txt", "content": "test text"}]
```

```
>>> list(ftpserver.get_file_contents("test_file1.txt", style="url"))
[{"path": "ftp://fakeusername:qweqwe@localhost:8888/test_file1.txt",
  "content": "test text"}]
```

```
>>> list(ftpserver.get_file_contents(["test_file1.txt", "test_folder/test_
↪file2.zip"],
...                                   read_mode="rb"))
[{"path": "test_file1.txt", "content": b"test text"},
 {"path": "test_folder/test_file2.zip", "content": b'PK\x03\x04\x14\x00\x00...
↪'}]
```

See also:

get_file_paths(), *put_files()*

4.1.4 get_file_paths

`FunctionalityWrapper.`**`get_file_paths`**(*style='rel_path'*, *anon=False*)

Yields the paths of all files server_home/anon_root, in the given *style*.

Parameters

- **`style`** (*{'rel_path', 'url'}, default 'rel_path'*) –

  'rel_path': path relative to server_home/anon_root is returned.

  'url': url to the file is returned.

- **`anon`** (*bool*) –

  True: filepaths/urls of all files in anon_root is returned.

  False: filepaths/urls of all files in server_home is returned.

Yields **file_path** (*str*) – Generator of all filepaths in the server_home/anon_root

Raises

- `TypeError` – If *style* is not a `str`
- `TypeError` – If *anon* is not a `bool`
- `ValueError` – If the value of *style* is not 'rel_path' or 'url'

Examples

Assuming a file structure as follows.

```
filesystem
+---server_home
|   +---test_file1
|   +---test_folder
|       +---test_file2
|
+---anon_root
    +---test_file3
    +---test_folder
        +---test_file4
```

```
>>> list(ftpserver.get_file_paths(style="rel_path", anon=False))
["test_file1", "test_folder/test_file2"]
```

```
>>> list(ftpserver.get_file_paths(style="rel_path", anon=True))
["test_file3", "test_folder/test_file4"]
```

```
>>> list(ftpserver.get_file_paths(style="url", anon=False))
["ftp://fakeusername:qweqwe@localhost:8888/test_file1",
"ftp://fakeusername:qweqwe@localhost:8888/test_folder/test_file2"]
```

```
>>> list(ftpserver.get_file_paths(style="url", anon=True))
["ftp://localhost:8888/test_file3", "ftp://localhost:8888/test_folder/test_
↪file4"]
```

4.1.5 get_local_base_path

`FunctionalityWrapper.`**`get_local_base_path`**(*anon=False*)

Returns the basepath on the local file system. Depending on anon the basepath is for the registered or anonymous user.

Parameters

- **anon** (*bool*) –
- **anon** –

  True: returns the local path to anon_root

  False: returns the local path to server_home

Returns **base_path** – Basepath on the local file system.

Return type str

Raises `TypeError` – If *anon* is not a `bool`

Examples

```
>>> ftpserver.get_local_base_path(anon=False))
/tmp/ftp_home_1rg7_i
```

```
>>> ftpserver.get_local_base_path(anon=True))
/tmp/anon_root_m6fknmyx
```

4.1.6 get_login_data

`FunctionalityWrapper.`**`get_login_data`**(*style='dict'*, *anon=False*)

Returns the login data as dict or url. What the returned value looks like is depending on *style* and the anonymous user or registered user depending *anon*.

Parameters

- **`style`** (*{'dict', 'url'}, default 'dict'*) –

  'dict': returns a dict with keys *host*, *port*, *user* and *passwd* or only *host* and *port*

  'url': returns a url containing the the login data

- **`anon`** (*bool*) –

  True: returns the login data for the anonymous user

  False: returns the login data for the registered user

Returns **login_data** – Login data as dict or url, depending on the value of *style*.

Return type dict, str

Raises

- `TypeError` – If *style* is not a `str`
- `TypeError` – If *anon* is not a `bool`
- `ValueError` – If the value of *style* is not 'dict' or 'url'

Examples

```
>>> ftpserver.get_login_data()
{"host": "localhost", "port": 8888, "user": "fakeusername",
"passwd": "qweqwe"}
```

```
>>> ftpserver.get_login_data(anon=True)
{"host": "localhost", "port": 8888}
```

```
>>> ftpserver.get_login_data(style="url")
ftp://fakeusername:qweqwe@localhost:8888
```

```
>>> ftpserver.get_login_data(style="url", anon=True)
ftp://localhost:8888
```

4.1.7 put_files

FunctionalityWrapper.**put_files**(*files_on_local*, *style='rel_path'*, *anon=False*, *overwrite=False*, *return_paths='input'*, *return_content=False*, *read_mode='r'*)

Copies the files defined in *files_on_local* to the sever. After 'uploading' the files it returns a list of paths or content_dicts depending on *return_content*

Parameters

- **files_on_local** (*str, dict, list of str/dict, iterable of str/dict*) – Path/-s to the local file/-s which should be copied to the server.

  str/list of str: all files will be copied to the chosen root.

  dict/list of dict:

  files_on_local["src"]: gives the local file path and

  files_on_local["dest"]: gives the relative path the file on the server.

- **style** (*{'rel_path', 'url'}, default 'rel_path'*) –

  'rel_path': path relative to server_home/anon_root is returned.

  'url': url to the file is returned.

- **anon** (*bool*) –

  True: Use anon_root as basepath

  False: Use server_home as basepath

- **overwrite** (*bool, default False*) –

  True: overwrites file without warning

  False: warns the user if a file exists and doesn't overwrite it

- **return_paths** (*{'all', 'input', 'new'}, default 'input'*) –

  'all': Return all files in the server_home/anon_root.

  'input': Return files in the server_home/anon_root, which were added by put_files.

  'new': Return only changed files in the server_home/anon_root, which were added by put_files.

- **return_content** (*bool, default False*) –

  False: Elements of the iterable to be returned will consist of only the paths (str).

  True: Elements of the iterable to be returned will consist of content_dicts.

- **read_mode** (*{'r', 'rb'}, default 'r'*) – This only applies if *return_content* is True. Mode in which files should be read (see `open("filepath", read_mode)`)

Returns **file_list** – List of filepaths/content dicts in server_home/anon_root

Return type list

Raises

- `TypeError` – If *files_on_local* is not a `str`, `dict` or `iterable of str/dict`
- `TypeError` – If *style* is not a `str`
- `TypeError` – If *anon* is not a `bool`

- `TypeError` – If *overwrite* is not a `bool`
- `TypeError` – If *return_paths* is not a `str`
- `TypeError` – If *return_content* is not a `bool`
- `TypeError` – If *read_mode* is not a `str`
- `ValueError` – If *files_on_local* is/contains an invalid filepath.
- `ValueError` – If the value of *style* is not 'rel_path' or 'url'
- `ValueError` – If the value of *return_paths* is not 'all', 'input' or 'new'
- `ValueError` – If the value of *read_mode* is not 'r' or 'rb'
- `KeyError` – If dict or list of dicts is used for *files_on_local* and the dict is missing the keys 'src' and 'dest'.

Examples

```
>>> ftpserver.put_files("test_folder/test_file", style="rel_path", anon=False)
["test_file"]
```

```
>>> ftpserver.put_files("test_folder/test_file", style="url", anon=False)
["ftp://fakeusername:qweqwe@localhost:8888/test_file"]
```

```
>>> ftpserver.put_files("test_folder/test_file", style="url", anon=True)
["ftp://localhost:8888/test_file"]
```

```
>>> ftpserver.put_files({"src": "test_folder/test_file",
...                      "dest": "remote_folder/uploaded_file"},
...                     style="url", anon=True)
["ftp://localhost:8888/remote_folder/uploaded_file"]
```

```
>>> ftpserver.put_files("test_folder/test_file", return_content=True)
[{"path": "test_file", "content": "some text in test_file"}]
```

```
>>> ftpserver.put_files("test_file.zip", return_content=True, read_mode="rb")
[{"path": "test_file.zip", "content": b'PK\x03\x04\x14\x00\x00...'}]
```

```
>>> ftpserver.put_files("test_file", return_paths="new")
UserWarning: test_file does already exist and won't be overwritten.
    Set `overwrite` to True to overwrite it anyway.
[]
```

```
>>> ftpserver.put_files("test_file", return_paths="new", overwrite=True)
["test_file"]
```

```
>>> ftpserver.put_files("test_file3", return_paths="all")
["test_file", "remote_folder/uploaded_file", "test_file.zip"]
```

See also:

get_file_contents(), *get_file_paths()*

4.1.8 reset_tmp_dirs

FunctionalityWrapper.**reset_tmp_dirs**()
Clears all temp files generated on the FTP server. This method is implemented to have more control over the module scoped ftp server.

Examples

filesystem before:

```
filesystem
+---server_home
|    +---test_file1
|    +---test_folder
|        +---test_file2
|
+---anon_root
    +---test_file3
    +---test_folder
        +---test_file4
```

```
>>> ftpserver.reset_tmp_dirs()
```

filesystem after:

```
filesystem
+---server_home
|
+---anon_root
```

4.1.9 stop

FunctionalityWrapper.**stop**()
Stops the server, closes all the open ports and deletes all temp files. This is especially useful if you want to test if your code behaves gracefully, when the ftpserver isn't reachable.

> **Warning:** If pytest-localftpserver is run in 'module' (default) or 'session' scope, this should be the last test run using this fixture (in the given test module or suite), Since the server can't be restarted.

Examples

```
>>> ftpserver.stop()
>>> your_code_connecting_to_the_ftp()
RuntimeError: Server is offline/ not reachable.
```

Methods Documentation

format_file_path(*rel_file_path*, *style='rel_path'*, *anon=False*)
Formats the relative path to as relative path or url. Relative paths are relative to the

server_home/anon_root, which can be used by a FTP client. Urls can be used by a browser/downloader. This method works, weather the file exists or not.

Notes

Even so taking a relative path and returning a relative path may seam pointless, this is needed to prevent errors with Windows path formatting (\ instead of /).

Parameters

- **rel_file_path** (*str*) – Relative filepath to server_home/anon_root depending on the value of anon.
- **style** (*{'rel_path', 'url'}, default 'rel_path'*) –

  'rel_path': path relative to server_home/anon_root is returned.

  'url': url to the file is returned.
- **anon** (*bool*) –

  True: return the filepaths/url of file in anon_root

  False: return the filepaths/url of file in server_home

Returns **file_path** – Relative path or url depending on the value of style

Return type str

Raises

- `TypeError` – If *style* is not a `str`
- `TypeError` – If *anon* is not a `bool`
- `ValueError` – If the value of *style* is not 'rel_path' or 'url'

Examples

```
>>> ftpserver.format_file_path("test_folder\test_file", style="rel_path",␣
↪anon=False))
test_folder/test_file
```

```
>>> ftpserver.format_file_path("test_folder/test_file", style="rel_path",␣
↪anon=False))
test_folder/test_file
```

```
>>> ftpserver.format_file_path("test_folder/test_file", style="url",␣
↪anon=False))
ftp://fakeusername:qweqwe@localhost:8888/test_folder/test_file
```

```
>>> ftpserver.format_file_path("test_folder/test_file", style="url",␣
↪anon=True))
ftp://localhost:8888/test_folder/test_file
```

See also:

get_local_base_path(), *get_file_paths()*

get_cert(*style='path'*, *read_mode='r'*)
Returns the path to the used certificate or its content as string or bytes.

Parameters

- **style** (*{'path', 'content'}, default 'path'*) – List of filepaths/content dicts in server_home/anon_root
- **read_mode** (*{'r', 'rb'}, default 'r'*) – This only applies if *style* is 'content'. Mode in which files should be read (see `open("filepath", read_mode)`)

Returns **cert** – Path to or content of the used certificate

Return type str

Raises

- `TypeError` – If *style* is not a `str`
- `TypeError` – If *read_mode* is not a `str`
- `ValueError` – If the value of *style* is not 'path' or 'content'
- `ValueError` – If the value of *read_mode* is not 'r' or 'rb'
- `WrongFixtureError` – If used on `ftpserver` fixture, instead of `ftpserver_TLS` fixture.

Examples

```
>>> ftpserver_TLS.get_cert()
"/home/certs/TLS_cert.pem"
```

```
>>> ftpserver_TLS.get_cert(style="content")
"-----BEGIN RSA PRIVATE KEY-----\nMIICXw..."
```

```
>>> ftpserver_TLS.get_cert(style="content", read_mode="rb")
b"-----BEGIN RSA PRIVATE KEY-----\nMIICXw..."
```

get_file_contents(*rel_file_paths=None*, *style='rel_path'*, *anon=False*, *read_mode='r'*)
Yields dicts containing the *path* and *content* of files on the FTP server.

Parameters

- **rel_file_paths** (*str, list of str, None, default None*) –

  None: The content of all files on the server will be retrieved.

  str or list of str: Only the content of those files will be retrieved.

- **style** (*{'rel_path', 'url'}, default 'rel_path'*) –

  'rel_path': Path relative to server_home/anon_root is returned.

  'url': A url to the file is returned.

- **anon** (*bool*) –

  True: return the filepaths/url of files in anon_root

  False: return the filepaths/url of files in in server_home

- **read_mode** (*{'r', 'rb'}, default 'r'*) – Mode in which files should be read (see `open("filepath", read_mode)`)

Yields **content_dict** (*dict*) – Dict containing the file *path* as relpath or url (see *style*) and the *content* of the file as string or bytes (see *read_mode*)

Raises

- `TypeError` – If *rel_file_paths* is not `None`, a `str` or an `iterable`
- `TypeError` – If *style* is not a `str`
- `TypeError` – If *anon* is not a `bool`
- `TypeError` – If *read_mode* is not a `str`
- `ValueError` – If the value of *rel_file_paths* or its items are not valid filepaths
- `ValueError` – If the value of *style* is not 'rel_path' or 'url'
- `ValueError` – If the value of *read_mode* is not 'r' or 'rb'

Examples

Assuming a file structure as follows.

```
filesystem
+---server_home
    +---test_file1.txt
    +---test_folder
        +---test_file2.zip
```

```
>>> list(ftpserver.get_file_contents())
[{"path": "test_file1.txt", "content": "test text"},
 {"path": "test_folder/test_file2.txt", "content": "test text2"}]
```

```
>>> list(ftpserver.get_file_contents("test_file1.txt"))
[{"path": "test_file1.txt", "content": "test text"}]
```

```
>>> list(ftpserver.get_file_contents("test_file1.txt", style="url"))
[{"path": "ftp://fakeusername:qweqwe@localhost:8888/test_file1.txt",
  "content": "test text"}]
```

```
>>> list(ftpserver.get_file_contents(["test_file1.txt", "test_folder/test_
↪file2.zip"],
...                                   read_mode="rb"))
[{"path": "test_file1.txt", "content": b"test text"},
 {"path": "test_folder/test_file2.zip", "content": b'PK\x03\x04\x14\x00\x00..
↪.'}]
```

See also:

get_file_paths(), *put_files()*

get_file_paths (*style='rel_path', anon=False*)

Yields the paths of all files server_home/anon_root, in the given *style*.

Parameters

- **style** (*{'rel_path', 'url'}, default 'rel_path'*) –

'rel_path': path relative to server_home/anon_root is returned.

'url': url to the file is returned.

- **anon** (*bool*) –

  True: filepaths/urls of all files in anon_root is returned.

  False: filepaths/urls of all files in server_home is returned.

Yields **file_path** (*str*) – Generator of all filepaths in the server_home/anon_root

Raises

- `TypeError` – If *style* is not a `str`
- `TypeError` – If *anon* is not a `bool`
- `ValueError` – If the value of *style* is not 'rel_path' or 'url'

Examples

Assuming a file structure as follows.

```
filesystem
+---server_home
|    +---test_file1
|    +---test_folder
|        +---test_file2
|
+---anon_root
    +---test_file3
    +---test_folder
        +---test_file4
```

```
>>> list(ftpserver.get_file_paths(style="rel_path", anon=False))
["test_file1", "test_folder/test_file2"]
```

```
>>> list(ftpserver.get_file_paths(style="rel_path", anon=True))
["test_file3", "test_folder/test_file4"]
```

```
>>> list(ftpserver.get_file_paths(style="url", anon=False))
["ftp://fakeusername:qweqwe@localhost:8888/test_file1",
"ftp://fakeusername:qweqwe@localhost:8888/test_folder/test_file2"]
```

```
>>> list(ftpserver.get_file_paths(style="url", anon=True))
["ftp://localhost:8888/test_file3", "ftp://localhost:8888/test_folder/test_
↪file4"]
```

get_local_base_path(*anon=False*)

Returns the basepath on the local file system. Depending on anon the basepath is for the registered or anonymous user.

Parameters

- **anon** (*bool*) –
- **anon** –

  True: returns the local path to anon_root

False: returns the local path to server_home

Returns **base_path** – Basepath on the local file system.

Return type str

Raises `TypeError` – If *anon* is not a `bool`

Examples

```
>>> ftpserver.get_local_base_path(anon=False))
/tmp/ftp_home_lrg7_i
```

```
>>> ftpserver.get_local_base_path(anon=True))
/tmp/anon_root_m6fknmyx
```

get_login_data(*style='dict'*, *anon=False*)

Returns the login data as dict or url. What the returned value looks like is depending on *style* and the anonymous user or registered user depending *anon*.

Parameters

- **style** (*{'dict', 'url'}, default 'dict'*) –

  'dict': returns a dict with keys *host*, *port*, *user* and *passwd* or only *host* and *port*

  'url': returns a url containing the the login data

- **anon** (*bool*) –

  True: returns the login data for the anonymous user

  False: returns the login data for the registered user

Returns **login_data** – Login data as dict or url, depending on the value of *style*.

Return type dict, str

Raises

- `TypeError` – If *style* is not a `str`
- `TypeError` – If *anon* is not a `bool`
- `ValueError` – If the value of *style* is not 'dict' or 'url'

Examples

```
>>> ftpserver.get_login_data()
{"host": "localhost", "port": 8888, "user": "fakeusername",
"passwd": "qweqwe"}
```

```
>>> ftpserver.get_login_data(anon=True)
{"host": "localhost", "port": 8888}
```

```
>>> ftpserver.get_login_data(style="url")
ftp://fakeusername:qweqwe@localhost:8888
```

```
>>> ftpserver.get_login_data(style="url", anon=True)
ftp://localhost:8888
```

put_files (*files_on_local*, *style='rel_path'*, *anon=False*, *overwrite=False*, *return_paths='input'*, *return_content=False*, *read_mode='r'*)

Copies the files defined in *files_on_local* to the sever. After 'uploading' the files it returns a list of paths or content_dicts depending on *return_content*

Parameters

- **files_on_local** (*str, dict, list of str/dict, iterable of str/dict*) – Path/-s to the local file/-s which should be copied to the server.

  str/list of str: all files will be copied to the chosen root.

  dict/list of dict:

  files_on_local["src"]: gives the local file path and

  files_on_local["dest"]: gives the relative path the file on the server.

- **style** (*{'rel_path', 'url'}, default 'rel_path'*) –

  'rel_path': path relative to server_home/anon_root is returned.

  'url': url to the file is returned.

- **anon** (*bool*) –

  True: Use anon_root as basepath

  False: Use server_home as basepath

- **overwrite** (*bool, default False*) –

  True: overwrites file without warning

  False: warns the user if a file exists and doesn't overwrite it

- **return_paths** (*{'all', 'input', 'new'}, default 'input'*) –

  'all': Return all files in the server_home/anon_root.

  'input': Return files in the server_home/anon_root, which were added by put_files.

  'new': Return only changed files in the server_home/anon_root, which were added by put_files.

- **return_content** (*bool, default False*) –

  False: Elements of the iterable to be returned will consist of only the paths (str).

  True: Elements of the iterable to be returned will consist of content_dicts.

- **read_mode** (*{'r', 'rb'}, default 'r'*) – This only applies if *return_content* is True. Mode in which files should be read (see `open("filepath", read_mode)`)

Returns **file_list** – List of filepaths/content dicts in server_home/anon_root

Return type list

Raises

- `TypeError` – If *files_on_local* is not a `str`, `dict` or `iterable of str/dict`
- `TypeError` – If *style* is not a `str`
- `TypeError` – If *anon* is not a `bool`
- `TypeError` – If *overwrite* is not a `bool`
- `TypeError` – If *return_paths* is not a `str`

- `TypeError` – If *return_content* is not a `bool`
- `TypeError` – If *read_mode* is not a `str`
- `ValueError` – If *files_on_local* is/contains an invalid filepath.
- `ValueError` – If the value of *style* is not 'rel_path' or 'url'
- `ValueError` – If the value of *return_paths* is not 'all', 'input' or 'new'
- `ValueError` – If the value of *read_mode* is not 'r' or 'rb'
- `KeyError` – If dict or list of dicts is used for *files_on_local* and the dict is missing the keys 'src' and 'dest'.

Examples

```
>>> ftpserver.put_files("test_folder/test_file", style="rel_path",␣
↪anon=False)
["test_file"]
```

```
>>> ftpserver.put_files("test_folder/test_file", style="url", anon=False)
["ftp://fakeusername:qweqwe@localhost:8888/test_file"]
```

```
>>> ftpserver.put_files("test_folder/test_file", style="url", anon=True)
["ftp://localhost:8888/test_file"]
```

```
>>> ftpserver.put_files({"src": "test_folder/test_file",
...                      "dest": "remote_folder/uploaded_file"},
...                     style="url", anon=True)
["ftp://localhost:8888/remote_folder/uploaded_file"]
```

```
>>> ftpserver.put_files("test_folder/test_file", return_content=True)
[{"path": "test_file", "content": "some text in test_file"}]
```

```
>>> ftpserver.put_files("test_file.zip", return_content=True, read_mode="rb")
[{"path": "test_file.zip", "content": b'PK\x03\x04\x14\x00\x00...'}]
```

```
>>> ftpserver.put_files("test_file", return_paths="new")
UserWarning: test_file does already exist and won't be overwritten.
    Set `overwrite` to True to overwrite it anyway.
[]
```

```
>>> ftpserver.put_files("test_file", return_paths="new", overwrite=True)
["test_file"]
```

```
>>> ftpserver.put_files("test_file3", return_paths="all")
["test_file", "remote_folder/uploaded_file", "test_file.zip"]
```

See also:

get_file_contents(), *get_file_paths()*

reset_tmp_dirs()

Clears all temp files generated on the FTP server. This method is implemented to have more control over the module scoped ftp server.

Examples

filesystem before:

```
filesystem
+---server_home
|    +---test_file1
|    +---test_folder
|        +---test_file2
|
+---anon_root
    +---test_file3
    +---test_folder
        +---test_file4
```

```
>>> ftpserver.reset_tmp_dirs()
```

filesystem after:

```
filesystem
+---server_home
|
+---anon_root
```

stop()

Stops the server, closes all the open ports and deletes all temp files. This is especially useful if you want to test if your code behaves gracefully, when the ftpserver isn't reachable.

> **Warning:** If pytest-localftpserver is run in 'module' (default) or 'session' scope, this should be the last test run using this fixture (in the given test module or suite), Since the server can't be restarted.

Examples

```
>>> ftpserver.stop()
>>> your_code_connecting_to_the_ftp()
RuntimeError: Server is offline/ not reachable.
```

CHAPTER 5

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

5.1 Types of Contributions

5.1.1 Report Bugs

Report bugs at https://github.com/oz123/pytest-localftpserver/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

5.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

5.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

5.1.4 Write Documentation

PyTest FTP Server could always use more documentation, whether as part of the official PyTest FTP Server docs, in docstrings, or even on the web in blog posts, articles, and such.

5.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/oz123/pytest-localftpserver/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

5.2 Get Started!

Ready to contribute? Here's how to set up *pytest_localftpserver* for local development.

1. Fork the *pytest-localftpserver* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/pytest-localftpserver.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv pytest_localftpserver
$ cd pytest-localftpserver/
$ pip install -r requirements_dev.txt
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ tox
```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

5.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 3.5, 3.6 and 3.7 . Check https://github.com/oz123/pytest-localftpserver/actions?query=workflow%3ATests and make sure that the tests pass for all supported Python versions.

5.4 Tips

To run a subset of tests:

```
$ py.test tests/test_pytest_localftpserver.py::<test_name>
```

5.5 Deploying

A reminder for the maintainers on how to deploy. Make sure all your changes are committed (including an entry in HISTORY.rst). Then run:

```
$ bump2version patch # possible: major / minor / patch
$ git push --follow-tags
```

CHAPTER 6

Credits

6.1 Development Lead

- Oz Tiram <oz.tiram@gmail.com>

6.2 Contributors

- Sebastian Weigand <s.weigand.phy@gmail.com>

CHAPTER 7

History

7.1 1.0.1 (2019-12-10)

- Include the certificate in the source package
- Use a bigger certificate

7.2 1.0.0 (2019-09-05)

- Dropped support for Python 2.7 and 3.4

7.3 0.6.0 - released as tag only

- Added fixture scope configuration.
- Added `ftpserver_TLS` as TLS version of the fixture.

7.4 0.5.0 (2018-12-04)

- Added support for Windows.
- Added hightlevel interface.

7.5 0.1.0 (2016-12-09)

- First release on PyPI.

CHAPTER 8

Indices and tables

- genindex
- modindex
- search

Index