pyftpsync Documentation

Release 3.1.0

Martin Wendt

Dec 26, 2019

Contents

Synchronize local directories with FTP servers.

Project https://github.com/mar10/pyftpsync/

Version 3.1, Date: Dec 26, 2019

CHAPTER 1

Installation

Requirements: Python 2.7+ or 3.4+ is required.

Releases are hosted on PyPI and can be installed using pip:

```
$ pip install pyftpsync
$ pyftpsync --version -v
pyftpsync/2.0.1 Python/3.6.1 Darwin-17.6.0-x86_64-i386-64bit
```

Note: MS Windows users that only need the command line interface may prefer the MSI installer.

Now the `pyftpsync` command is available:

```
$ pyftpsync --help
```

and the `ftpsync` package can be used in Python code:

```
$ python
>>> from ftpsync import __version__
>>> __version__
'2.0.0'
```

CHAPTER 2

User Guide

Warning: Major version updates (1.0 => 2.0, 2.0 => 3.0, . . .) introduce *breaking changes* to the previous versions. Make sure to adjust your scripts accordingly after update.

2.1 Command Line Interface

Use the `--help` or `-h` argument to get help:

```
$ pyftpsync --help
usage: pyftpsync [-h] [-v | -q] [-V] {upload,download,sync,run,scan} ...

Synchronize folders over FTP.

positional arguments:
{upload,download,sync,run,scan}
                        sub-command help
    upload              copy new and modified files to remote folder
    download            copy new and modified files from remote folder to
                        local target
    sync                synchronize new and modified files between remote
                        folder and local target
    run                 run pyftpsync with configuration from
                        `.pyftpsync.yaml` in current or parent folder
    scan                repair, purge, or check targets

optional arguments:
-h, --help            show this help message and exit
-v, --verbose         increment verbosity by one (default: 3, range: 0..5)
-q, --quiet           decrement verbosity by one
-V, --version         show program's version number and exit
```

(continues on next page)

(continued from previous page)

```
See also https://github.com/mar10/pyftpsync
$
```

2.1.1 *run* command

In addition to the direct invocation of *upload*, *download*, or *sync* commands, version 3.x allows to define a sample_pyftpsync_yaml file in your project's root folder which then can be executed like so:

```
$ pyftpsync run
```

optionally, default settings can be overidden:

```
$ pyftpsync run --dry-run
$ pyftpsync run TASK
```

See the sample_pyftpsync_yaml example for details.

2.1.2 Target URLs

The `local` and `remote` target arguments can be file paths or URLs (currently the `ftp:` and `ftps:` protocols are supported):

```
$ pyftpsync upload ~/temp ftp://example.com/target/folder
```

FTP URLs may contain credentials:

```
$ pyftpsync upload ~/temp ftp://joe:secret@example.com/target/folder
```

Note that *pyftpsync* also supports prompting for passwords and storing passwords in the system keyring.

2.1.3 Authentication

FTP targets often require authentication. There are multiple ways to handle this:

1. Pass credentials with the target URL: `ftp://user:password@example.com/target/folder`
2. Pass only a user name with the target URL: `ftp://user@example.com/target/folder` The CLI will prompt for a password (the library would raise an error).
3. Don't pass any credentials with the URL: `ftp://example.com/target/folder` *pyftpsync* will now
    1. Try to lookup credentials for host ('example.com') in the system keyring storage.
    2. Try to lookup credentials for host ('example.com') in the `.netrc` file in the user's home directory.
    3. CLI will prompt for username and password.
    4. Assume anonymous access.
4. If authentication fails, the CLI will prompt for a password again.

Credential discovery can be controlled by `--no-keyring`, `--no-netrc`, and `--no-prompt` options. `--prompt` will force prompting, even if lookup is possible. `--store-password` will save credentials to the system keyring storage upon successful login.

Note: In order to use *.netrc* on Windows, the *%HOME%* environment variable should be set. If not, try this: `> set HOME=%USERPROFILE%` (see here).

2.1.4 Matching and Filtering

The `--match` option filters processed files using on or more patterns (using the fnmatch syntax). **Note:** These patterns are only applied to files, not directories.

The `--exclude` option is applied after *–match* and removes entries from processing. Unlike *–match*, these patterns are also applied to directories.

Example:

```
$ pyftpsync scan /my/folder --list --match=*.js,*.css --exclude=.git,build,node_
↪modules
```

2.1.5 Upload Files Syntax

Command specific help is available like so:

```
$ pyftpsync upload -h
usage: pyftpsync upload [-h] [--force] [--resolve {local,skip,ask}] [--delete]
                        [--delete-unmatched] [-n] [-v | -q] [--progress]
                        [--no-color] [--ftp-active] [--migrate] [-m MATCH]
                        [-x EXCLUDE] [--prompt | --no-prompt] [--no-keyring]
                        [--no-netrc] [--store-password]
                        LOCAL REMOTE

positional arguments:
  LOCAL                 path to local folder (default: .)
  REMOTE                path to remote folder

optional arguments:
  -h, --help            show this help message and exit
  --force               overwrite remote files, even if the target is newer
                        (but no conflict was detected)
  --resolve {local,skip,ask}
                        conflict resolving strategy (default: 'ask')
  --delete              remove remote files if they don't exist locally
  --delete-unmatched    remove remote files if they don't exist locally or
                        don't match the current filter (implies '--delete'
                        option)
  -n, --dry-run         just simulate and log results, but don't change
                        anything
  -v, --verbose         increment verbosity by one (default: 3, range: 0..5)
  -q, --quiet           decrement verbosity by one
  --progress            show progress info, even if redirected or verbose < 3
  --no-color            prevent use of ansi terminal color codes
  --ftp-active          use Active FTP mode instead of passive
  --migrate             replace meta data files from different pyftpsync
                        versions with current format. Existing data will be
                        discarded.
  -m MATCH, --match MATCH
```

(continues on next page)

(continued from previous page)

```
                        wildcard for file names using fnmatch syntax (default:
                        match all, separate multiple values with ',')
  -x EXCLUDE, --exclude EXCLUDE
                        wildcard of files and directories to exclude (applied
                        after --match, default: '.DS_Store,.git,.hg,.svn')
  --prompt              always prompt for password
  --no-prompt           prevent prompting for invalid credentials
  --no-keyring          prevent use of the system keyring service for
                        credential lookup
  --no-netrc            prevent use of .netrc file for credential lookup
  --store-password      save password to keyring if login succeeds
$
```

2.1.6 Example: Upload Files

Upload all new and modified files from user's temp folder to an FTP server. No files are changed on the local directory:

```
$ pyftpsync upload ~/temp ftp://example.com/target/folder
```

Add the `--delete` option to remove all files from the remote target that don't exist locally:

```
$ pyftpsync upload ~/temp ftp://example.com/target/folder --delete
```

Add the `--dry-run` option to switch to DRY-RUN mode, i.e. run in test mode without modifying files:

```
$ pyftpsync upload ~/temp ftp://example.com/target/folder --delete --dry-run
```

Add one or more `-v` options to increase output verbosity:

```
$ pyftpsync upload ~/temp ftp://example.com/target/folder --delete -vv
```

Mirror current directory to remote folder:

```
$ pyftpsync upload . ftp://example.com/target/folder --force --delete --resolve=local
```

Note: Replace `ftp://` with `ftps://` to enable TLS encryption.

2.1.7 Synchronize Files Syntax

```
$ pyftpsync sync -h
usage: pyftpsync sync [-h] [--resolve {old,new,local,remote,skip,ask}] [-n]
                      [-v | -q] [--progress] [--no-color] [--ftp-active]
                      [--migrate] [-m MATCH] [-x EXCLUDE]
                      [--prompt | --no-prompt] [--no-keyring] [--no-netrc]
                      [--store-password]
                      LOCAL REMOTE

positional arguments:
  LOCAL                 path to local folder (default: .)
  REMOTE                path to remote folder
```

(continues on next page)

(continued from previous page)

```
optional arguments:
  -h, --help            show this help message and exit
  --resolve {old,new,local,remote,skip,ask}
                        conflict resolving strategy (default: 'ask')
  -n, --dry-run         just simulate and log results, but don't change
                        anything
  -v, --verbose         increment verbosity by one (default: 3, range: 0..5)
  -q, --quiet           decrement verbosity by one
  --progress            show progress info, even if redirected or verbose < 3
  --no-color            prevent use of ansi terminal color codes
  --ftp-active          use Active FTP mode instead of passive
  --migrate             replace meta data files from different pyftpsync
                        versions with current format. Existing data will be
                        discarded.
  -m MATCH, --match MATCH
                        wildcard for file names using fnmatch syntax (default:
                        match all, separate multiple values with ',')
  -x EXCLUDE, --exclude EXCLUDE
                        wildcard of files and directories to exclude (applied
                        after --match, default: '.DS_Store,.git,.hg,.svn')
  --prompt              always prompt for password
  --no-prompt           prevent prompting for invalid credentials
  --no-keyring          prevent use of the system keyring service for
                        credential lookup
  --no-netrc            prevent use of .netrc file for credential lookup
  --store-password      save password to keyring if login succeeds
$
```

2.1.8 Example: Synchronize Folders

Two-way synchronization of a local folder with an FTP server:

```
$ pyftpsync sync --store-password --resolve=ask --execute ~/temp ftps://example.com/
↪target/folder
```

Note that `ftps:` protocol was specified to enable TLS.

2.1.9 Verbosity Level

The verbosity level can have a value from 0 to 6:

Verbosity	Option	Log level	Remarks
0	-qqq	CRITICAL	quiet
1	-qq	ERROR	show errors only
2	-q	WARN	show conflicts and 1 line summary only
3		INFO	show write operations
4	-v	DEBUG	show equal files
5	-vv	DEBUG	diff-info and benchmark summary
6	-vvv	DEBUG	show FTP commands

2.1.10 Exit Codes

The CLI returns those exit codes:

```
0: OK
1: Error (network, internal, ...)
2: CLI syntax error
3: Aborted by user
```

2.2 Script Examples

All options that are available for command line, can also be passed to the synchronizers. For example `--delete-unmatched` becomes `"delete_unmatched": True`.

Upload modified files from local folder to FTP server:

```
from ftpsync.targets import FsTarget
from ftpsync.ftp_target import FtpTarget
from ftpsync.synchronizers import UploadSynchronizer

local = FsTarget("~/temp")
user ="joe"
passwd = "secret"
remote = FtpTarget("/temp", "example.com", username=user, password=passwd)
opts = {"force": False, "delete_unmatched": True, "verbose": 3}
s = UploadSynchronizer(local, remote, opts)
s.run()
```

Synchronize a local folder with an FTP server using TLS:

```
from ftpsync.targets import FsTarget
from ftpsync.ftp_target import FtpTarget
from ftpsync.synchronizers import BiDirSynchronizer

local = FsTarget("~/temp")
user ="joe"
passwd = "secret"
remote = FtpTarget("/temp", "example.com", username=user, password=passwd, tls=True)
opts = {"resolve": "skip", "verbose": 1}
s = BiDirSynchronizer(local, remote, opts)
s.run()
```

2.2.1 Logging

By default, the library initializes and uses a python logger named 'pyftpsync'. This logger can be customized like so:

```
import logging

logger = logging.getLogger("pyftpsync")
logger.setLevel(logging.DEBUG)
```

and replaced like so:

```
import logging
import logging.handlers
from ftpsync.util import set_pyftpsync_logger

custom_logger = logging.getLogger("my.logger")
log_path = "/my/path/pyftpsync.log"
handler = logging.handlers.WatchedFileHandler(log_path)
formatter = logging.Formatter("%(asctime)s - %(name)s - %(levelname)s - %(message)s")
handler.setFormatter(formatter)
custom_logger.addHandler(handler)

set_pyftpsync_logger(custom_logger)
```

Note: The CLI calls `set_pyftpsync_logger(None)` on startup, so it logs to stdout (and stderr).

CHAPTER 3

Reference Guide

3.1 Class Inheritance Diagram

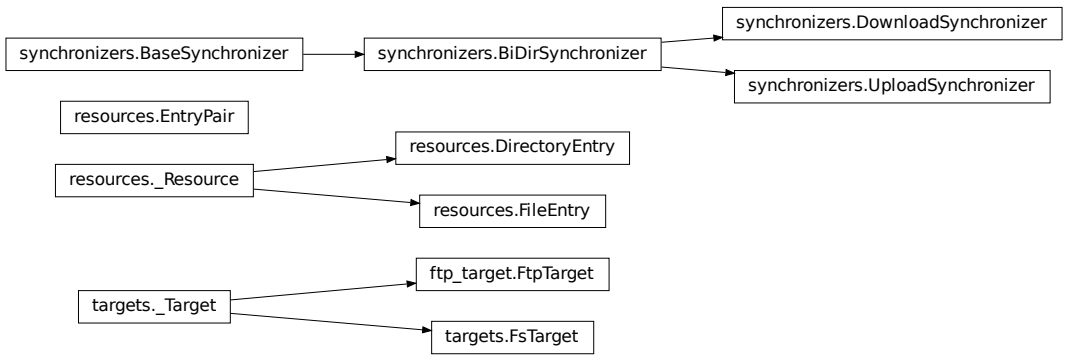


Fig. 1: pyftpsync classes

3.2 Algorithm

See also:

See also the pyftpsync-spec.pdf for details about the algorithm and implementation.

3.3 API Reference

3.3.1 ftpsync

ftpsync package

ftpsync.resources module

(c) 2012-2019 Martin Wendt; see https://github.com/mar10/pyftpsync Licensed under the MIT license: https://www.opensource.org/licenses/mit-license.php

class ftpsync.resources.**DirectoryEntry**(*target*, *rel_path*, *name*, *size*, *mtime*, *unique*)
Bases: *ftpsync.resources._Resource*

__delattr__
x.__delattr__('name') <==> del x.name

__format__()
default object formatter

__getattribute__
x.__getattribute__('name') <==> x.name

__hash__

__reduce__()
helper for pickle

__reduce_ex__()
helper for pickle

__repr__

__setattr__
x.__setattr__('name', value) <==> x.name = value

__sizeof__() → int
size of object in memory, in bytes

as_string(*other_resource=None*)

classify(*peer_dir_meta*)
Classify this entry as 'new', 'unmodified', or 'modified'.

get_rel_path()

get_sync_info(*key=None*)

is_dir()

is_file()

is_local()

set_sync_info(*local_file*)

class ftpsync.resources.**EntryPair**(*local*, *remote*)
Bases: object

__delattr__
x.__delattr__('name') <==> del x.name

__format__()
default object formatter

__getattribute__
x.__getattribute__('name') <==> x.name

__hash__

__reduce__()
helper for pickle

__reduce_ex__()
helper for pickle

__repr__

__setattr__
x.__setattr__('name', value) <==> x.name = value

__sizeof__() → int
size of object in memory, in bytes

any_entry
Return the local entry (or the remote entry if it is None).

classify(*peer_dir_meta*)
Classify entry pair.

is_conflict()

is_dir = None
type: bool

is_same_time()
Return True if local.mtime == remote.mtime.

local_classification = None
type: str

name = None
type: str

operation = None
type: str

override_operation(*operation*, *reason*)
Re-Classify entry pair.

re_class_reason = None
type: str

rel_path = None
type: str

remote_classification = None
type: str

class ftpsync.resources.**FileEntry**(*target*, *rel_path*, *name*, *size*, *mtime*, *unique*)
Bases: *ftpsync.resources._Resource*

EPS_TIME = 2.01

__delattr__
x.__delattr__('name') <==> del x.name

__format__()
default object formatter

__getattribute__
x.__getattribute__('name') <==> x.name

__hash__

__reduce__()
helper for pickle

__reduce_ex__()
helper for pickle

__repr__

__setattr__
x.__setattr__('name', value) <==> x.name = value

__sizeof__() → int
size of object in memory, in bytes

static _eps_compare(*date_1*, *date_2*)

as_string(*other_resource=None*)

classify(*peer_dir_meta*)
Classify this entry as 'new', 'unmodified', or 'modified'.

get_rel_path()

get_sync_info(*key=None*)
Get mtime/size when this resource was last synchronized with remote.

is_dir()

is_file()

is_local()

set_sync_info(*local_file*)

was_modified_since_last_sync()
Return True if this resource was modified since last sync.

None is returned if we don't know (because of missing meta data).

class ftpsync.resources.**_Resource**(*target*, *rel_path*, *name*, *size*, *mtime*, *unique*)
Bases: object

Common base class for files and directories.

__delattr__
x.__delattr__('name') <==> del x.name

__format__()
default object formatter

__getattribute__
x.__getattribute__('name') <==> x.name

__hash__

__reduce__()
helper for pickle

__reduce_ex__()
helper for pickle

__repr__

__setattr__
x.__setattr__('name', value) <==> x.name = value

__sizeof__() → int
size of object in memory, in bytes

as_string(*other_resource=None*)

classification = None
(set by synchronizer._classify_entry()).

Type str

classify(*peer_dir_meta*)
Classify this entry as 'new', 'unmodified', or 'modified'.

get_rel_path()

get_sync_info(*key=None*)

is_dir()

is_file()

is_local()

mtime = None
Current file modification time stamp (for FTP targets adjusted using metadata information).

Type float

mtime_org = None
Modification time stamp (as reported by source FTP server).

Type float

name = None
File name.

Type str

ps_mtime = None
File modification time stamp at the time of last sync operation

Type float

ps_size = None
File size at the time of last sync operation

Type int

ps_utime = None
Time stamp of last sync operation

Type float

rel_path = None
Path relative to *target*

Type str

set_sync_info(*local_file*)

size = None
Current file size

Type int

target = None
Parent target object.

Type _Target

unique = None
Unique id of file/directory.

Type str

ftpsync.synchronizers module

(c) 2012-2019 Martin Wendt; see https://github.com/mar10/pyftpsync Licensed under the MIT license: https://www.opensource.org/licenses/mit-license.php

class ftpsync.synchronizers.**BaseSynchronizer**(*local*, *remote*, *options*)
Bases: object

Synchronizes two target instances in dry_run mode (also base class for other synchronizers).

__delattr__
x.__delattr__('name') <==> del x.name

__format__()
default object formatter

__getattribute__
x.__getattribute__('name') <==> x.name

__hash__

__reduce__()
helper for pickle

__reduce_ex__()
helper for pickle

__repr__

__setattr__
x.__setattr__('name', value) <==> x.name = value

__sizeof__() → int
size of object in memory, in bytes

__str__

_before_sync(*entry*)
Called by the synchronizer for each entry.

Return False to prevent the synchronizer's default action.

_compare_file(*local*, *remote*)
Byte compare two files (early out on first difference).

_copy_file(*src*, *dest*, *file_entry*)

_copy_recursive(*src*, *dest*, *dir_entry*)

_dry_run_action(*action*)
"Called in dry-run mode after call to _log_action() and before exiting function.

_inc_stat(*name*, *ofs=1*)

_log_action(*action*, *status*, *symbol*, *entry*, *min_level=3*)

_match(*entry*)

_remove_dir(*dir_entry*)

_remove_file(*file_entry*)

_resolve_shortcuts = {'l': 'local', 'r': 'remote', 's': 'skip'}

_sync_dir()
Traverse the local folder structure and remote peers.

This is the core algorithm that generates calls to self.sync_XXX() handler methods. _sync_dir() is called by self.run().

_test_match_or_print(*entry*)
Return True if entry matches filter. Otherwise print 'skip' and return False.

_tick()
Write progress info and move cursor to beginning of line.

close()

get_info_strings()

get_stats()

is_script = None
True if this synchronizer is used by a command line script (e.g. pyftpsync.exe)

Type bool

on_conflict(*pair*)
Called when resources have been modified on local *and* remote.

Returns False to prevent visiting of children (if pair is a directory)

on_copy_local(*pair*)
Called when the local resource should be copied to remote.

on_copy_remote(*pair*)
Called when the remote resource should be copied to local.

on_delete_local(*pair*)
Called when the local resource should be deleted.

on_delete_remote(*pair*)
Called when the remote resource should be deleted.

on_equal(*pair*)
Called for (unmodified, unmodified) pairs.

on_error(*e*, *pair*)
Called for pairs that don't match *match* and *exclude* filters.

on_mismatch(*pair*)
Called for pairs that don't match *match* and *exclude* filters.

on_need_compare(*pair*)
Re-classify pair based on file attributes and options.

re_classify_pair(*pair*)
Allow derrived classes to override default classification and operation.

Returns False to prevent default operation.

resolve_all = None
Conflict resolution strategy

Type str

run()

class ftpsync.synchronizers.**BiDirSynchronizer**(*local*, *remote*, *options*)
Bases: *ftpsync.synchronizers.BaseSynchronizer*

Synchronizer that performs up- and download operations as required.

- Newer files override unmodified older files
- When both files are newer than last sync -> conflict! Conflicts may be resolved by these options:

```
--resolve=old:          use the older version
--resolve=new:          use the newer version
--resolve=local:        use the local file
--resolve=remote:       use the remote file
--resolve=ask:          prompt mode
```

- When a file is missing: check if it existed in the past. If so, delete it. Otherwise copy it.

In order to know if a file was modified, deleted, or created since last sync, we store a snapshot of the directory in the local directory.

__delattr__
x.__delattr__('name') <==> del x.name

__format__()
default object formatter

__getattribute__
x.__getattribute__('name') <==> x.name

__hash__

__reduce__()
helper for pickle

__reduce_ex__()
helper for pickle

__repr__

__setattr__
x.__setattr__('name', value) <==> x.name = value

__sizeof__() → int
size of object in memory, in bytes

__str__

_before_sync(*entry*)
Called by the synchronizer for each entry.

Return False to prevent the synchronizer's default action.

_compare_file(*local*, *remote*)
Byte compare two files (early out on first difference).

_copy_file(*src*, *dest*, *file_entry*)

_copy_recursive(*src*, *dest*, *dir_entry*)

_dry_run_action(*action*)
"Called in dry-run mode after call to _log_action() and before exiting function.

_inc_stat(*name*, *ofs=1*)

_interactive_resolve(*pair*)
Return 'local', 'remote', or 'skip' to use local, remote resource or skip.

_log_action(*action*, *status*, *symbol*, *entry*, *min_level=3*)

_match(*entry*)

_print_pair_diff(*pair*)

_remove_dir(*dir_entry*)

_remove_file(*file_entry*)

_resolve_shortcuts = {'l': 'local', 'r': 'remote', 's': 'skip'}

_sync_dir()
Traverse the local folder structure and remote peers.

This is the core algorithm that generates calls to self.sync_XXX() handler methods. _sync_dir() is called by self.run().

_test_match_or_print(*entry*)
Return True if entry matches filter. Otherwise print 'skip' and return False.

_tick()
Write progress info and move cursor to beginning of line.

close()

get_info_strings()

get_stats()

on_conflict(*pair*)
Return False to prevent visiting of children.

on_copy_local(*pair*)
Called when the local resource should be copied to remote.

on_copy_remote(*pair*)
Called when the remote resource should be copied to local.

on_delete_local(*pair*)
Called when the local resource should be deleted.

on_delete_remote(*pair*)
Called when the remote resource should be deleted.

on_equal(*pair*)
Called for (unmodified, unmodified) pairs.

on_error(*e*, *pair*)
Called for pairs that don't match *match* and *exclude* filters.

on_mismatch(*pair*)
Called for pairs that don't match *match* and *exclude* filters.

on_need_compare(*pair*)
Re-classify pair based on file attributes and options.

re_classify_pair(*pair*)
Allow derrived classes to override default classification and operation.

Returns False to prevent default operation.

run()

ftpsync.synchronizers.**DEFAULT_OMIT = ['.DS_Store', '.git', '.hg', '.svn']**
Default for –exclude CLI option Note: DirMetadata.META_FILE_NAME and LOCK_FILE_NAME are always ignored

class ftpsync.synchronizers.**DownloadSynchronizer**(*local*, *remote*, *options*)
Bases: *ftpsync.synchronizers.BiDirSynchronizer*

__delattr__
x.__delattr__('name') <==> del x.name

__format__()
default object formatter

__getattribute__
x.__getattribute__('name') <==> x.name

__hash__

__reduce__()
helper for pickle

__reduce_ex__()
helper for pickle

__repr__

__setattr__
x.__setattr__('name', value) <==> x.name = value

__sizeof__() → int
size of object in memory, in bytes

__str__

_before_sync(*entry*)
Called by the synchronizer for each entry.

Return False to prevent the synchronizer's default action.

_compare_file(*local*, *remote*)
Byte compare two files (early out on first difference).

_copy_file(*src*, *dest*, *file_entry*)

_copy_recursive(*src*, *dest*, *dir_entry*)

_dry_run_action(*action*)
"Called in dry-run mode after call to _log_action() and before exiting function.

_inc_stat(*name*, *ofs=1*)

_interactive_resolve(*pair*)
Return 'local', 'remote', or 'skip' to use local, remote resource or skip.

_log_action(*action*, *status*, *symbol*, *entry*, *min_level=3*)

_match(*entry*)

_print_pair_diff(*pair*)

_remove_dir(*dir_entry*)

_remove_file(*file_entry*)

_resolve_shortcuts = {'l': 'local', 'r': 'remote', 's': 'skip'}

_sync_dir()
Traverse the local folder structure and remote peers.

This is the core algorithm that generates calls to self.sync_XXX() handler methods. _sync_dir() is called by self.run().

_test_match_or_print(*entry*)
Return True if entry matches filter. Otherwise print 'skip' and return False.

_tick()
Write progress info and move cursor to beginning of line.

close()

get_info_strings()

get_stats()

on_conflict(*pair*)
Return False to prevent visiting of children.

on_copy_local(*pair*)
Called when the local resource should be copied to remote.

on_copy_remote(*pair*)
Called when the remote resource should be copied to local.

on_delete_local(*pair*)
Called when the local resource should be deleted.

on_delete_remote(*pair*)
Called when the remote resource should be deleted.

on_equal(*pair*)
Called for (unmodified, unmodified) pairs.

on_error(*e*, *pair*)
Called for pairs that don't match *match* and *exclude* filters.

on_mismatch(*pair*)
Called for pairs that don't match *match* and *exclude* filters.

If –delete-unmatched is on, remove the remote resource.

on_need_compare(*pair*)
Re-classify pair based on file attributes and options.

re_classify_pair(*pair*)
Allow derrived classes to override default classification and operation.

Returns False to prevent default operation.

run()

class ftpsync.synchronizers.**UploadSynchronizer**(*local*, *remote*, *options*)
Bases: *ftpsync.synchronizers.BiDirSynchronizer*

__delattr__
x.__delattr__('name') <==> del x.name

__format__()
default object formatter

__getattribute__
x.__getattribute__('name') <==> x.name

__hash__

__reduce__()
helper for pickle

__reduce_ex__()
helper for pickle

__repr__

__setattr__
x.__setattr__('name', value) <==> x.name = value

__sizeof__() → int
size of object in memory, in bytes

__str__

_before_sync(*entry*)
Called by the synchronizer for each entry.

Return False to prevent the synchronizer's default action.

_compare_file(*local*, *remote*)
Byte compare two files (early out on first difference).

_copy_file(*src*, *dest*, *file_entry*)

_copy_recursive(*src*, *dest*, *dir_entry*)

_dry_run_action(*action*)
"Called in dry-run mode after call to _log_action() and before exiting function.

_inc_stat(*name*, *ofs=1*)

_interactive_resolve(*pair*)
Return 'local', 'remote', or 'skip' to use local, remote resource or skip.

_log_action(*action*, *status*, *symbol*, *entry*, *min_level=3*)

_match(*entry*)

_print_pair_diff(*pair*)

_remove_dir(*dir_entry*)

_remove_file(*file_entry*)

_resolve_shortcuts = {'l': 'local', 'r': 'remote', 's': 'skip'}

_sync_dir()
Traverse the local folder structure and remote peers.

This is the core algorithm that generates calls to self.sync_XXX() handler methods. _sync_dir() is called by self.run().

_test_match_or_print(*entry*)
Return True if entry matches filter. Otherwise print 'skip' and return False.

_tick()
Write progress info and move cursor to beginning of line.

close()

get_info_strings()

get_stats()

on_conflict(*pair*)
Return False to prevent visiting of children.

on_copy_local(*pair*)
Called when the local resource should be copied to remote.

on_copy_remote(*pair*)
Called when the remote resource should be copied to local.

on_delete_local(*pair*)
Called when the local resource should be deleted.

on_delete_remote(*pair*)
Called when the remote resource should be deleted.

on_equal(*pair*)
Called for (unmodified, unmodified) pairs.

on_error(*e*, *pair*)
Called for pairs that don't match *match* and *exclude* filters.

on_mismatch(*pair*)
Called for pairs that don't match *match* and *exclude* filters.

If –delete-unmatched is on, remove the remote resource.

on_need_compare(*pair*)
Re-classify pair based on file attributes and options.

re_classify_pair(*pair*)
Allow derrived classes to override default classification and operation.

Returns False to prevent default operation.

run()

ftpsync.synchronizers.**match_path**(*entry*, *opts*)
Return True if *path* matches *match* and *exclude* options.

ftpsync.synchronizers.**process_options**(*opts*)
Check and prepare options dict.

ftpsync.targets module

(c) 2012-2019 Martin Wendt; see https://github.com/mar10/pyftpsync Licensed under the MIT license: https://www.opensource.org/licenses/mit-license.php

class ftpsync.targets.**FsTarget**(*root_dir*, *extra_opts=None*)
Bases: *ftpsync.targets._Target*

DEFAULT_BLOCKSIZE = 16384

__delattr__
x.__delattr__('name') <==> del x.name

__format__()
default object formatter

__getattribute__
x.__getattribute__('name') <==> x.name

__hash__

__reduce__()
helper for pickle

__reduce_ex__()
helper for pickle

__repr__

__setattr__
x.__setattr__('name', value) <==> x.name = value

__sizeof__() → int
size of object in memory, in bytes

check_write(*name*)
Raise exception if writing cur_dir/name is not allowed.

close()

copy_to_file(*name*, *fp_dest*, *callback=None*)
Write cur_dir/name to file-like *fp_dest*.

Parameters

- **name** (*str*) – file name, located in self.curdir
- **fp_dest** (*file-like*) – must support write() method
- **callback** (*function, optional*) – Called like *func(buf)* for every written chunk

cwd(*dir_name*)

flush_meta()
Write additional meta information for current directory.

get_base_name()

get_dir()
Return a list of _Resource entries.

get_id()

get_option(*key*, *default=None*)
Return option from synchronizer (possibly overridden by target extra_opts).

get_options_dict()
Return options from synchronizer (possibly overridden by own extra_opts).

get_sync_info(*name*, *key=None*)
Get mtime/size when this target's current dir was last synchronized with remote.

is_local()

is_unbound()

mkdir(*dir_name*)

open()

open_readable(*name*)
Return file-like object opened in binary mode for cur_dir/name.

open_writable(*name*)
Return file-like object opened in binary mode for cur_dir/name.

pop_meta()

push_meta()

pwd()

read_text(*name*)
Read text string from cur_dir/name using open_readable().

remove_file(*name*)
Remove cur_dir/name.

remove_sync_info(*name*)

rmdir(*dir_name*)
Remove cur_dir/name.

set_mtime(*name*, *mtime*, *size*)
Set modification time on file.

set_sync_info(*name*, *mtime*, *size*)
Store mtime/size when this resource was last synchronized with remote.

walk(*pred=None*, *recursive=True*)
Iterate over all target entries recursively.

Parameters

- **pred** (*function, optional*) – Callback(*ftpsync.resources._Resource*) should return *False* to ignore entry. Default: *None*.
- **recursive** (*bool, optional*) – Pass *False* to generate top level entries only. Default: *True*.

Yields *ftpsync.resources._Resource*

write_file(*name*, *fp_src*, *blocksize=16384*, *callback=None*)
Write binary data from file-like to cur_dir/name.

write_text(*name*, *s*)
Write string data to cur_dir/name using write_file().

class ftpsync.targets.**_Target**(*root_dir*, *extra_opts*)
Bases: object

Base class for *FsTarget*, FtpTarget, etc.

DEFAULT_BLOCKSIZE = 16384

__delattr__
x.__delattr__('name') <==> del x.name

__format__()
default object formatter

__getattribute__
x.__getattribute__('name') <==> x.name

__hash__

__reduce__()
helper for pickle

__reduce_ex__()
helper for pickle

__repr__

__setattr__
x.__setattr__('name', value) <==> x.name = value

__sizeof__() → int
size of object in memory, in bytes

__str__

check_write(*name*)
Raise exception if writing cur_dir/name is not allowed.

close()

copy_to_file(*name*, *fp_dest*, *callback=None*)
Write cur_dir/name to file-like *fp_dest*.

Parameters

- **name** (*str*) – file name, located in self.curdir
- **fp_dest** (*file-like*) – must support write() method
- **callback** (*function, optional*) – Called like *func(buf)* for every written chunk

cwd(*dir_name*)

encoding = None
Assumed encoding for this target. Used to decode binary paths.

flush_meta()
Write additional meta information for current directory.

get_base_name()

get_dir()
Return a list of _Resource entries.

get_id()

get_option(*key*, *default=None*)
Return option from synchronizer (possibly overridden by target extra_opts).

get_options_dict()
Return options from synchronizer (possibly overridden by own extra_opts).

get_sync_info(*name*, *key=None*)
Get mtime/size when this target's current dir was last synchronized with remote.

is_local()

is_unbound()

mkdir(*dir_name*)

mtime_compare_eps = None
Maximum allowed difference between a reported mtime and the last known update time, before we classify the entry as 'modified externally'

open()

open_readable(*name*)
Return file-like object opened in binary mode for cur_dir/name.

open_writable(*name*)
Return file-like object opened in binary mode for cur_dir/name.

pop_meta()

push_meta()

pwd(*dir_name*)

read_text(*name*)
Read text string from cur_dir/name using open_readable().

remove_file(*name*)
Remove cur_dir/name.

remove_sync_info(*name*)

rmdir(*dir_name*)
Remove cur_dir/name.

root_dir = None
The target's top-level folder

server_time_ofs = None
Time difference between <local upload time> and the mtime that the server reports afterwards. The value is added to the 'u' time stored in meta data. (This is only a rough estimation, derived from the lock-file.)

set_mtime(*name*, *mtime*, *size*)

set_sync_info(*name*, *mtime*, *size*)
Store mtime/size when this resource was last synchronized with remote.

walk(*pred=None*, *recursive=True*)
Iterate over all target entries recursively.

Parameters

- **pred** (*function, optional*) – Callback(*ftpsync.resources._Resource*) should return *False* to ignore entry. Default: *None*.
- **recursive** (*bool, optional*) – Pass *False* to generate top level entries only. Default: *True*.

Yields *ftpsync.resources._Resource*

write_file(*name*, *fp_src*, *blocksize=16384*, *callback=None*)
Write binary data from file-like to cur_dir/name.

write_text(*name*, *s*)
Write string data to cur_dir/name using write_file().

ftpsync.targets.**_get_encoding_opt**(*synchronizer*, *extra_opts*, *default*)
Helper to figure out encoding setting inside constructors.

ftpsync.targets.**make_target**(*url*, *extra_opts=None*)
Factory that creates *_Target* objects from URLs.

FTP targets must begin with the scheme `ftp://` or `ftps://` for TLS.

Note: TLS is only supported on Python 2.7/3.2+.

Parameters

- **url** (*str*)
- **extra_opts** (*dict, optional*) – Passed to Target constructor. Default: None.

Returns *_Target*

ftpsync.ftp_target module

(c) 2012-2019 Martin Wendt; see https://github.com/mar10/pyftpsync Licensed under the MIT license: https://www.opensource.org/licenses/mit-license.php

class ftpsync.ftp_target.**FtpTarget**(*path*, *host*, *port=0*, *username=None*, *password=None*, *tls=False*, *timeout=None*, *extra_opts=None*)

Bases: *ftpsync.targets._Target*

Represents a synchronization target on an FTP server.

path
Current working directory on FTP server.

Type str

ftp
Instance of ftplib.FTP.

Type FTP

host
hostname of FTP server

Type str

port
FTP port (defaults to 21)

Type int

username

Type str

password

Type str

DEFAULT_BLOCKSIZE = 8192

MAX_SPOOL_MEM = 102400

__delattr__
x.__delattr__('name') <==> del x.name

__format__()
default object formatter

__getattribute__
x.__getattribute__('name') <==> x.name

__hash__

__reduce__()
helper for pickle

__reduce_ex__()
helper for pickle

__repr__

__setattr__
x.__setattr__('name', value) <==> x.name = value

__sizeof__() → int
size of object in memory, in bytes

_ftp_nlst(*dir_name*)
Variant of *self.ftp.nlst()* that supports encoding-fallback.

_ftp_pwd()
Variant of *self.ftp.pwd()* that supports encoding-fallback.

Returns Current working directory as native string.

_ftp_retrlines_native(*command*, *callback*, *encoding*)
A re-implementation of ftp.retrlines that returns lines as native *str*.

This is needed on Python 3, where *ftp.retrlines()* returns unicode *str* by decoding the incoming command response using *ftp.encoding*. This would fail for the whole request if a single line of the MLSD listing cannot be decoded. FtpTarget wants to fall back to Cp1252 if UTF-8 fails for a single line, so we need to process the raw original binary input lines.

On Python 2, the response is already bytes, but we try to decode in order to check validity and optionally re-encode from Cp1252.

Parameters

- **command** (*str*) – A valid FTP command like 'NLST', 'MLSD', ...
- **callback** (*function*) –

  Called for every line with these args: status (int): 0:ok 1:fallback used, 2:decode failed
  line (str): result line decoded using *encoding*.

  If *encoding* is 'utf-8', a fallback to cp1252 is accepted.
- **encoding** (*str*) – Coding that is used to convert the FTP response to *str*.

Returns None

_lock(*break_existing=False*)
Write a special file to the target root folder.

_probe_lock_file(*reported_mtime*)
Called by get_dir

_rmdir_impl(*dir_name*, *keep_root_folder=False*, *predicate=None*)

_unlock(*closing=False*)
Remove lock file to the target root folder.

check_write(*name*)
Raise exception if writing cur_dir/name is not allowed.

close()

copy_to_file(*name*, *fp_dest*, *callback=None*)
Write cur_dir/name to file-like *fp_dest*.

Parameters

- **name** (*str*) – file name, located in self.curdir
- **fp_dest** (*file-like*) – must support write() method
- **callback** (*function, optional*) – Called like *func(buf)* for every written chunk

cwd(*dir_name*)

flush_meta()
Write additional meta information for current directory.

get_base_name()

get_dir()
Return a list of _Resource entries.

get_id()

get_option(*key*, *default=None*)
Return option from synchronizer (possibly overridden by target extra_opts).

get_options_dict()
Return options from synchronizer (possibly overridden by own extra_opts).

get_sync_info(*name*, *key=None*)
Get mtime/size when this target's current dir was last synchronized with remote.

is_local()

is_unbound()

lock_data = None
written to ftp target root folder before synchronization starts. set to False, if write failed. Default: None

Type dict

mkdir(*dir_name*)

open()

open_readable(*name*)
Open cur_dir/name for reading.

Note: we read everything into a buffer that supports .read().

Parameters **name** (*str*) – file name, located in self.curdir

Returns file-like (must support read() method)

open_writable(*name*)
Return file-like object opened in binary mode for cur_dir/name.

pop_meta()

push_meta()

pwd()
Return current working dir as native *str* (uses fallback-encoding).

read_text(*name*)
Read text string from cur_dir/name using open_readable().

remove_file(*name*)
Remove cur_dir/name.

remove_sync_info(*name*)

rmdir (*dir_name*)
Remove cur_dir/name.

server_time_ofs = None
Time difference between <local upload time> and the mtime that the server reports afterwards. The value is added to the 'u' time stored in meta data. (This is only a rough estimation, derived from the lock-file.)

set_mtime (*name*, *mtime*, *size*)

set_sync_info (*name*, *mtime*, *size*)
Store mtime/size when this resource was last synchronized with remote.

support_utf8 = None
True if server reports FEAT UTF8

walk (*pred=None*, *recursive=True*)
Iterate over all target entries recursively.

Parameters

- **pred** (*function, optional*) – Callback(*ftpsync.resources._Resource*) should return *False* to ignore entry. Default: *None*.
- **recursive** (*bool, optional*) – Pass *False* to generate top level entries only. Default: *True*.

Yields *ftpsync.resources._Resource*

write_file (*name*, *fp_src*, *blocksize=8192*, *callback=None*)
Write file-like *fp_src* to cur_dir/name.

Parameters

- **name** (*str*) – file name, located in self.curdir
- **fp_src** (*file-like*) – must support read() method
- **blocksize** (*int, optional*)
- **callback** (*function, optional*) – Called like *func(buf)* for every written chunk

write_text (*name*, *s*)
Write string data to cur_dir/name using write_file().

3.3.2 Index

CHAPTER 4

Development

4.1 Install for Development

First off, thanks for taking the time to contribute!

This small guideline may help takinf the first steps.

Happy hacking :)

4.1.1 Fork the Repository

Clone pyftpsync to a local folder and checkout the branch you want to work on:

```
$ git clone git@github.com:mar10/pyftpsync.git
$ cd pyftpsync
$ git checkout my_branch
```

4.1.2 Work in a Virtual Environment

Install Python

We need Python 2.7, Python 3.5+, and pip on our system.

If you want to run tests on *all* supported platforms, install Python 2.7, 3.5, 3.6, 3.7, and 3.8.

Create and Activate the Virtual Environment

Linux / macOS

On Linux/OS X, we recommend to use pipenv to make this easy:

```
$ cd /path/to/pyftpsync
$ pipenv shell
bash-3.2$
```

Windows

Alternatively (especially on Windows), use virtualenv to create and activate the virtual environment. For example using Python's builtin `venv` (instead of `virtualenvwrapper`) in a Windows PowerShell:

```
> cd /path/pyftpsync
> py -3.6 -m venv c:\env\pyftpsync_py36
> c:\env\pyftpsync_py36\Scripts\Activate.ps1
(pyftpsync_py36) $
```

Install Requirements

Now that the new environment exists and is activated, we can setup the requirements:

```
$ pip install -r requirements-dev.txt
```

and install pyftpsync to run from source code:

```
$ pip install -e .
```

The code should now run:

```
$ pyftpsync --version
$ 2.0.0
```

The test suite should run as well:

```
$ python setup.py test
$ pytest -v -rs
```

Build Sphinx documentation:

```
$ python setup.py sphinx
```

4.2 Run Tests

The unit tests create fixtures in a special folder. By default, a temporary folder is created on every test run, but it is recommended to define a location using the `PYFTPSYNC_TEST_FOLDER` environment variable, for example:

```
export PYFTPSYNC_TEST_FOLDER=/Users/USER/pyftpsync_test
```

Run all tests with coverage report. Results are written to <pyftpsync>/htmlcov/index.html:

```
$ pytest -v -rsx --cov=ftpsync --cov-report=html
```

Run selective tests:

```
$ pytest -v -rsx -k FtpBidirSyncTest
$ pytest -v -rsx -k "FtpBidirSyncTest and test_default"
$ pytest -v -rsx -m benchmark
```

Run tests on multiple Python versions using tox (need to install those Python versions first):

```
$ tox
$ tox -e py36
```

In order to run realistic tests through an FTP server, we need a setup that publishes a folder that is also accessible using file-system methods.

This can be achieved by configuring an FTP server to allow access to the *remote* folder:

```
<PYFTPSYNC_TEST_FOLDER>/
  local/
    folder1/
      file1_1.txt
      ...
    file1.txt
    ...
  remote/  # <- FTP server should publish this folder as <PYFTPSYNC_TEST_FTP_URL>
    ...
```

The test suite checks if `PYFTPSYNC_TEST_FTP_URL` is defined and accessible. Otherwise FTP tests will be skipped.

For example, environment variables may look like this, assuming the FTP server is rooted at the user's home directory:

```
export PYFTPSYNC_TEST_FOLDER=/Users/USER/pyftpsync_test
export PYFTPSYNC_TEST_FTP_URL=ftp://USER:PASSWORD@localhost/pyftpsync_test/remote
```

This environment variable may be set to generate `.pyftpsync-meta` files in a larger, but more readable format:

```
export PYFTPSYNC_VERBOSE_META=True
```

4.2.1 .pyftpsyncrc

Instead of using environment variables, it is recommended to create a `.pyftsyncrc` file in the user's home directory:

```
[test]
folder = /Users/USER/pyftpsync_test
ftp_url = ftp://USER:PASSWORD@localhost/pyftpsync_test/remote

[debug]
verbose_meta = True
```

Settings from environment variables still take precedence.

4.2.2 Run Manual Tests

In order to run the command line script against a defined test scenario, we can use the `test.fixture_tools` helper function to set up the default fixture:

```
$ python -m test.fixture_tools
Created fixtures at /Users/USER/test_pyftpsync

$ ls -al /Users/USER/test_pyftpsync
total 0
drwxrwxrwx   4 martin  staff  136  7 Okt 15:32 .
drwxr-xr-x   7 martin  staff  238 20 Aug 20:26 ..
drwxr-xr-x  19 martin  staff  646  7 Okt 15:32 local
drwxr-xr-x  18 martin  staff  612  7 Okt 15:32 remote
```

The fixture set's up files with defined time stamps (2014-01-01) and already contains meta data, so conflicts can be detected:

```
                        Local (UTC)     Remote (UTC)
------------------------------------------------------------------------------
file1.txt               12:00           12:00        (unmodified)
file2.txt               13:00           12:00
file3.txt                 x             12:00
file4.txt               12:00           13:00
file5.txt               12:00             x
file6.txt               13:00           13:00:05     CONFLICT!
file7.txt               13:00:05        13:00        CONFLICT!
file8.txt                 x             13:00        CONFLICT!
file9.txt               13:00             x          CONFLICT!

folder1/file1_1.txt     12.00           12:00        (unmodified)
folder2/file2_1.txt     13.00           12:00
folder3/file3_1.txt       x             12:00        (folder deleted)
folder4/file4_1.txt       x             13:00        (*) undetected CONFLICT!
folder5/file5_1.txt     12:00           13:00
folder6/file6_1.txt     12:00             x          (folder deleted)
folder7/file7_1.txt     13:00             x          (*) undetected CONFLICT!

new_file1.txt           13:00             -
new_file2.txt             -             13:00
new_file3.txt           13:00           13:00        (same size)
new_file4.txt           13:00           13:00        CONFLICT! (different size)
new_file5.txt           13:00           13:00:05     CONFLICT!
new_file6.txt           13:00:05        13:00        CONFLICT!

NOTE: (*) currently conflicts are NOT detected, when a file is edited on one
target and the parent folder is removed on the peer target.
The folder will be removed on sync!
```

Now run pyftpsync with arbitrary options, passing local and remote folders as targets, for example:

```
$ pyftpsync -v sync /Users/USER/test_pyftpsync/local /Users/USER/test_pyftpsync/remote
```

If an FTP server was configured, we can also run the script against it:

```
$ pyftpsync -v sync /Users/USER/test_pyftpsync/local ftp://localhost/Users/USER/test_
↪pyftpsync/remote
```

Run `python -m test.fixture_tools` again to reset the test folders.

4.2.3 Run FTP Server

Run `pylibdftp` FTP Server Locally

In develpoment mode, pyftpsync installs pyftpdlib which can be used to run an FTP server for testing. We allow anonymous access and use a custom port > 1024, so we don't need to sudo:

```
$ python -m pyftpdlib  -p 8021 -w -d /Users/USER/test_pyftpsync/remote
```

or:

```
$ python -m test.ftp_server
```

Also set the test options accordingly in `.pyftpsyncrc`:

```
[test]
folder = /Users/USER/pyftpsync_test
ftp_url = ftp://anonymous:@localhost:8021
```

Run Built-in FTP Server on macOS Sierra

Note: This does **not** work anymore with macOS *High* Sierra.

On OSX (starting with Sierra) the built-in FTP server needs to be activated like so:

```
$ sudo -s launchctl load -w /System/Library/LaunchDaemons/ftp.plist
```

It can be stopped the same way:

```
$ sudo -s launchctl unload -w /System/Library/LaunchDaemons/ftp.plist
```

The FTP server exposes the whole file system, so the URL must start from root:

```
[test]
folder = /Users/USER/pyftpsync_test
ftp_url = ftp://USER:PASSWORD@localhost/Users/USER/pyftpsync_test/remote
```

Warning: Exposing the file system is dangerous! Make sure to stop the FTP server after testing.

Run FTP Server on Windows

On Windows the Filezilla Server may be a good choice.

4.3 Code

Note: Follow the Style Guide, basically PEP 8.

Failing tests or not follwing PEP 8 will break builds on travis, so run `$ pytest`, `$ flake8`, and `$ tox` frequently and before you commit!

4.4 Create a Pull Request

Todo: TODO

CHAPTER 5

Release Info

5.1 3.1.1 (unreleased)

5.2 3.1.0 (2020-12-26)

- Drop support for Python 3.4 (end-of-life: 2019-03-18)
- Add support for Python 3.8
- Fix #38 Remove trailing '/' before checking PWD response

5.3 3.0.0 (2019-04-20)

- This release addresses some known **encoding-related issues**: - The internal path format are now native strings (i.e. unicode on Python 3

  or UTF-8 bytes on Python 2)

  - FTP targets are now assumed to support UTF-8.
  - #30: Fallback to CP-1252 encoding when FTP server returns non-UTF-8
  - Local filesystem targets now consider the OS encoding.
  - Modified format of *.pyftpsync-meta.json*: File names are now stored as UTF-8 (was the unmodified binary format of the target platform before).
  - See also the 'encoding' section in the [spec](https://github.com/mar10/pyftpsync/blob/master/docs/sphinx/pyftpsync-spec.pdf).
- **New 'run' command** reads and executes settings from a configuration file *.pyftpsync.yaml*
- Remove trailing garbage from output lines

Breaking Changes:

- Modified format of *.pyftpsync-meta.json*. Pass *–migrate* option to convert from a prvious version (note that this cannot be undone)

5.4 2.1.0 (2018-08-25)

- Allow *-v* with *–version* option.
- Fix #26: Crash when not setting verbose option.
- Print SYST and FEAT when -vv is passed
- Accept list type options for *exclude* argument in CLI mode
- Apply and enforce Black formatter
- Fix #27: Download- and UploadSynchronizer honor *–delete* flag for all conditions.
 NOTE: default settings will no longer delete files for up- and downloads.

5.5 2.0.0 (2018-01-01)

Note: the command line options have changed: **Be careful with existing shell scripts after updating from v1.x!**

New Features: - New *scan* command to list, purge, etc. remote targets. - Add FTPS (TLS) support. - Support Active FTP. - Support for *.netrc* files. - CLI returns defined error codes. - Use configurable logger for output when not in CLI mode. - Release as Wheel.

Breaking Changes: - Write mode is now on by default.

The *-x*, *–execute* option was removed, use *–dry-run* instead.

- *-f*, *–include-files* option was renamed to *-m*, *–match*.
 -o, *–omit* option was renamed to *-x*, *–exclude*.
- Modified format of *.pyftpsync-meta.json*.
- Dropped support for Python 2.6 and 3.3.

Fixes and Improvements: - Remove lock file on Ctrl-C. - Refactored and split into more modules. - Improved test framework and documentation. - Enforce PEP8, use flake8.

5.6 1.0.4 (unreleased)

- Add FTPS (TLS) support on Python 2.7/3.2+

5.7 1.0.3 (2015-06-28)

- Add conflict handling to upload and download commands
- Move documentation to Read The Docs
- Use tox for tests

5.8 1.0.2 (2015-05-17)

- Bi-directional synchronization
- Detect conflicts if both targets are modified since last sync
- Optional resolve strategy (e.g. always use local)
- Distinguish whether a resource was added on local or removed on remote
- Optionally prompt for username/password
- Optionally store credentials in keyring
- Custom password file (~/pyftpsync.pw) is no longer supported
- Colored output
- Interactive mode
- Renamed _pyftpsync-meta.json to .pyftpsync-meta.json
- MSI installer for MS Windows

5.9 0.2.1 (2013-05-07)

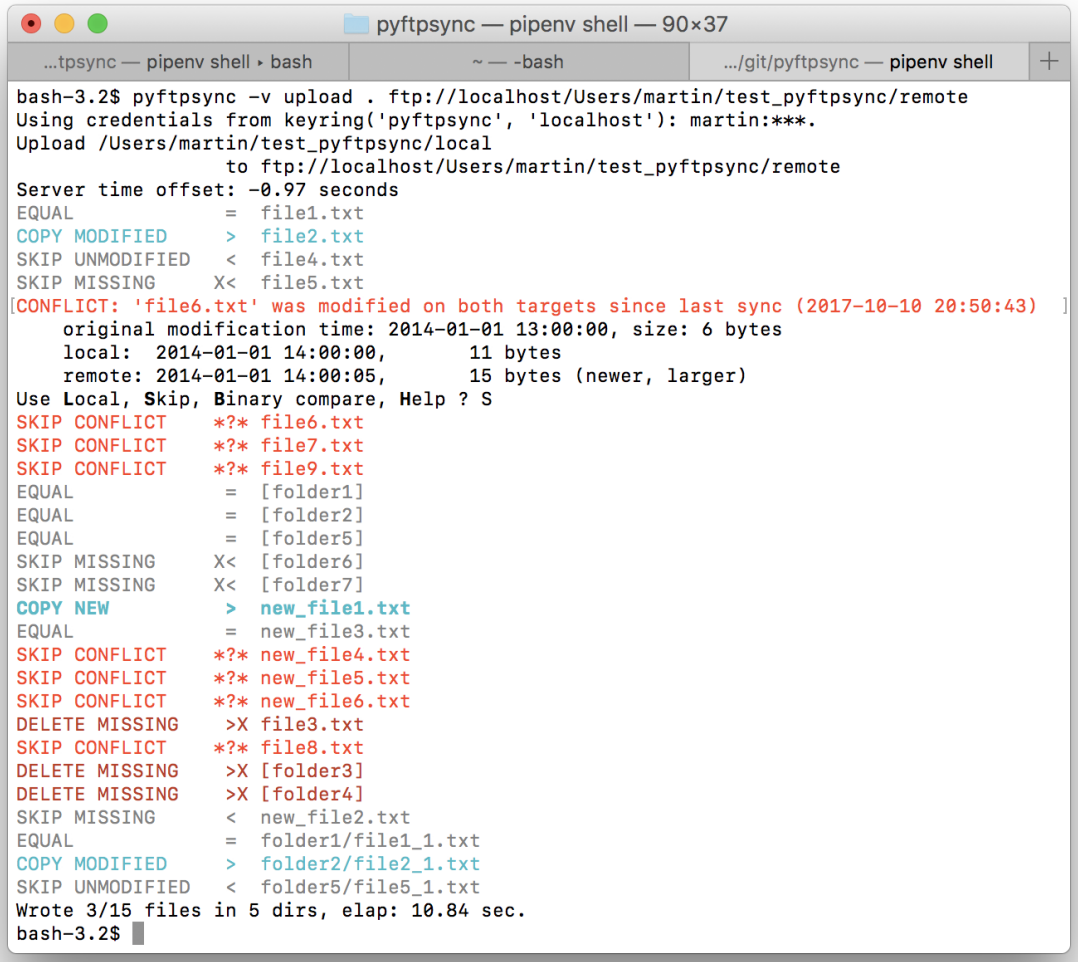
- Fixes for py3

5.10 0.2.0 (2013-05-06)

- Improved progress info
- Added *–progress* option

5.11 0.1.0 (2013-05-04)

First release



Warning: Major version updates (1.0 => 2.0, 2.0 => 3.0, . . .) introduce *breaking changes* to the previous versions. Make sure to adjust your scripts accordingly after update.

CHAPTER 6

Features

- This is a command line tool...
- ... and a library for use in custom Python projects.
- Recursive synchronization of folders on file system and/or FTP targets.
- Upload, download, and bi-directional synchronization mode.
- Configurable conflict resolution strategies.
- Unlike naive implementations, pyftpsync maintains additional meta data to detect conflicts and decide whether to replicate a missing file as deletion or addition.
- Unlike more complex implementations, pyftpsync does not require a database or a service running on the targets.
- Optional FTPS (TLS) support.
- Architecture is open to add other target types.

The command line tool adds:

- Runs on Linux, OS X, and Windows.
- Remember passwords in system keyring.
- Interactive conflict resolution mode.
- Dry-run mode.

Note: Known Limitations

- The FTP server must support the MLSD command.
- pyftpsync uses file size and modification dates to detect file changes. This is efficient, but not as robust as CRC checksums could be.
- pyftpsync tries to detect conflicts (i.e. simultaneous modifications of local and remote targets) by storing last sync time and size in a separate meta data file inside the local folders. This is not bullet proof and may fail under some conditions.

- Currently conflicts are *not* detected, when a file is edited on one target and the parent folder is removed on the peer target: The folder will be removed on sync.

In short: Make sure you have backups.

CHAPTER 7

Quickstart

Releases are hosted on PyPI and can be installed using pip:

```
$ pip install pyftpsync --upgrade
$ pyftpsync --help
```

Python Module Index

f

Index

Symbols

A

B

C

D

E

F

G

H

I

L

M

N

O

P

R

S

T

U

W