Southwark

Release 0.9.0

Extensions to the Dulwich Git library.

Dominic Davis-Foster

May 18, 2023

Contents

Chapter

ONE

Installation

1.1 from PyPI

```
$ python3 -m pip install Southwark --user
```

1.2 from Anaconda

First add the required channels

```
$ conda config --add channels https://conda.anaconda.org/conda-forge
$ conda config --add channels https://conda.anaconda.org/domdfcoding
```

Then install

```
$ conda install Southwark
```

1.3 from GitHub

```
$ python3 -m pip install git+https://github.com/repo-helper/southwark@master --user
```

Chapter

TWO

southwark

Extensions to the Dulwich Git library.

Classes:

GitStatus(staged, unstaged, untracked)	Represents the output of *status()*.
StagedDict	The values are lists of filenames, relative to the repository root.

Data:

_DR	Invariant TypeVar bound to dulwich.repo.Repo.

Functions:

assert_clean(repo[, allow_config])	Returns True if the working directory is clean.
check_git_status(repo_path)	Check the git status of the given repository.
clone(source[, target, bare, checkout, ...])	Clone a local or remote git repository.
get_tags([repo])	Returns a mapping of commit SHAs to tags.
get_tree_changes(repo)	Return add/delete/modify changes to tree by comparing the index to HEAD.
get_untracked_paths(path, index)	Returns a list of untracked files.
open_repo_closing(path_or_repo)	Returns a context manager which will return dulwich.repo.Repo objects unchanged, but will create a new dulwich.repo.Repo when a filesystem path is given.
status([repo])	Returns staged, unstaged, and untracked changes relative to the HEAD.
windows_clone_helper()	Contextmanager to aid cloning on Windows during tests.

namedtuple GitStatus(*staged*, *unstaged*, *untracked*)

Bases: NamedTuple

Represents the output of *status()*.

New in version 0.6.1.

Fields

0) **staged** (*StagedDict*) – Dict with lists of staged paths.
1) **unstaged** (List[PathPlus]) – List of unstaged paths.
2) **untracked** (List[PathPlus]) – List of untracked, un-ignored & non-.git paths.

__repr__()

Return a nicely formatted representation string

typeddict StagedDict
Bases: dict

The values are lists of filenames, relative to the repository root.

New in version 0.6.1.

Required Keys

- **add** (List[PathPlus])
- **delete** (List[PathPlus])
- **modify** (List[PathPlus])

_DR = TypeVar(_DR, bound=Repo)
Type: TypeVar

Invariant TypeVar bound to dulwich.repo.Repo.

assert_clean(*repo*, *allow_config=()*)
Returns True if the working directory is clean.

If not, returns False and prints a helpful error message to stderr.

Parameters

- **repo** (PathPlus)
- **allow_config** (Sequence[Union[str, Path, PathLike]]) – Default ().

Return type bool

check_git_status(*repo_path*)
Check the git status of the given repository.

Parameters **repo_path** (Union[str, Path, PathLike]) – Path to the repository root.

Return type Tuple[bool, List[str]]

Returns Whether the git working directory is clean, and the list of uncommitted files if it isn't.

clone(*source*, *target=None*, *bare=False*, *checkout=None*, *errstream=<_io.BufferedWriter name='<stderr>'>*, *origin='origin'*, *depth=None*, ***kwargs*)
Clone a local or remote git repository.

Parameters

- **source** (Union[str, bytes]) – Path or URL for source repository.
- **target** (Union[str, Path, PathLike, bytes, None]) – Path to target repository. Default None.
- **bare** (bool) – Whether to create a bare repository. Default False.
- **checkout** (Optional[bool]) – Whether to check-out HEAD after cloning. Default None.
- **errstream** (IO) – Optional stream to write progress to. Default <_io.BufferedWriter name='<stderr>'>.
- **origin** (Union[str, bytes]) – Name of remote from the repository used to clone. Default 'origin'.
- **depth** (Optional[int]) – Depth to fetch at. Default None.

Return type *Repo*

Returns The cloned repository.

New in version 0.6.1.

Changed in version 0.7.2:

- `target` now accepts `domdf_python_tools.typing.PathLike` objects.
- `origin` now accepts `str` objects.

get_tags(*repo='.'*)
Returns a mapping of commit SHAs to tags.

Parameters **repo** (`Union[Repo, str, Path, PathLike]`) – Default `'.'`.

Return type `Dict[str, str]`

get_tree_changes(*repo*)
Return add/delete/modify changes to tree by comparing the index to HEAD.

Parameters **repo** (`Union[str, Path, PathLike, Repo]`) – repo path or object.

Return type *StagedDict*

Returns Dictionary containing changes for each type of change.

New in version 0.6.1.

get_untracked_paths(*path*, *index*)
Returns a list of untracked files.

Parameters

- **path** (`Union[str, Path, PathLike]`) – Path to walk.
- **index** (`Index`) – Index to check against.

Return type `Iterator[str]`

open_repo_closing(*path_or_repo*)
Returns a context manager which will return `dulwich.repo.Repo` objects unchanged, but will create a new `dulwich.repo.Repo` when a filesystem path is given.

New in version 0.7.0.

Parameters **path_or_repo** – Either a `dulwich.repo.Repo` object or the path of a repository.

Return type `AbstractContextManager[+T_co]`

Overloads

- *open_repo_closing*(**path_or_repo**: *~_DR*) -> `AbstractContextManager[~_DR]`
- *open_repo_closing*(**path_or_repo**: `Union[str, PathLike]`) -> `AbstractContextManager[Repo]`

status(*repo='.'*)
Returns staged, unstaged, and untracked changes relative to the HEAD.

Parameters **repo** (`Union[Repo, str, Path, PathLike]`) – Path to repository or repository object.
Default `'.'`.

Return type *GitStatus*

windows_clone_helper()

Contextmanager to aid cloning on Windows during tests.

New in version 0.8.0.

> **Attention:** This function is intended only for use in tests.

Usage:

```
with windows_clone_helper():
    repo = clone(...)
```

Return type Iterator[None]

Chapter

THREE

southwark.click

Extensions to click.

New in version 0.5.0.

Functions:

commit_message_option(default)	Decorator to add the `-m / --message` option to a click command.
commit_option(default)	Decorator to add the `--commit / --no-commit` option to a click command.

commit_message_option (*default*)
Decorator to add the `-m / --message` option to a click command.

New in version 0.5.0.

Parameters **default** (`str`) – The default commit message.

Return type `Callable`

commit_option (*default*)
Decorator to add the `--commit / --no-commit` option to a click command.

New in version 0.5.0.

Parameters **default** (`Optional`[`bool`]) – Whether to commit automatically.

- `None` – Ask first
- `True` – Commit automatically
- `False` – Don't commit

Return type `Callable`

Chapter

FOUR

southwark.config

Utilities for repository configuration.

New in version 0.5.0.

Functions:

get_remotes(config)	Returns a dictionary mapping remote names to URLs.
set_remote_http(config, domain, username, repo)	Set the remote url for the repository, using HTTP.
set_remote_ssh(config, domain, username, repo)	Set the remote url for the repository, using SSH.

get_remotes(*config*)
Returns a dictionary mapping remote names to URLs.

Parameters **config** (ConfigFile)

Return type Dict[str, str]

set_remote_http(*config*, *domain*, *username*, *repo*, *name='origin'*)
Set the remote url for the repository, using HTTP.

Parameters

- **config** (ConfigFile)
- **domain** (str)
- **username** (str)
- **repo** (str)
- **name** (str) – The name of the remote to set. Default 'origin'.

set_remote_ssh(*config*, *domain*, *username*, *repo*, *name='origin'*)
Set the remote url for the repository, using SSH.

Parameters

- **config** (ConfigFile)
- **domain** (str)
- **username** (str)
- **repo** (str)
- **name** (str) – The name of the remote to set. Default 'origin'.

Chapter

FIVE

southwark.log

Python implementation of `git log`.

Classes:

Log([repo])	Python implementation of `git log`.

class Log(*repo='.'*)

Bases: `object`

Python implementation of `git log`.

Parameters **repo** (`Union`[`Repo`, `str`, `Path`, `PathLike`]) – The git repository. Default `'.'`.

Attributes:

current_branch	The name of the current branch
local_branches	Mapping of local branches to the SHA of the latest commit in that branch.
refs	Mapping of git refs to commit SHAs.
remote_branches	Mapping of remote branches to the SHA of the latest commit in that branch.
repo	The git repository.
tags	Mapping of commit SHAs to tags.

Methods:

format_commit(commit)	Return a human-readable commit log entry.
log([max_entries, reverse, from_date, ...])	Return the formatted commit log.

current_branch

Type: `str`

The name of the current branch

format_commit(*commit*)

Return a human-readable commit log entry.

Parameters **commit** (`Commit`) – A *Commit* object

Return type `StringList`

local_branches
Type: Dict[str, str]

Mapping of local branches to the SHA of the latest commit in that branch.

log(*max_entries=None*, *reverse=False*, *from_date=None*, *from_tag=None*, *colour=True*)
Return the formatted commit log.

Parameters

- **max_entries** (Optional[int]) – Maximum number of entries to display. Default all entries.
- **reverse** (bool) – Print entries in reverse order. Default False.
- **from_date** (Optional[datetime]) – Show commits after the given date. Default None.
- **from_tag** (Optional[str]) – Show commits after the given tag. Default None.
- **colour** (bool) – Show coloured output. Default True.

Return type str

refs
Type: Dict[str, str]

Mapping of git refs to commit SHAs.

remote_branches
Type: Dict[str, str]

Mapping of remote branches to the SHA of the latest commit in that branch.

repo
Type: Repo

The git repository.

tags
Mapping of commit SHAs to tags.

Chapter

SIX

southwark.repo

Modified Dulwich repository object.

New in version 0.3.0.

Classes:

Repo(root[, object_store, bare])	Modified Dulwich repository object.

Data:

_R	Invariant TypeVar bound to *southwark.repo.Repo*.

Functions:

get_user_identity(config[, kind])	Determine the identity to use for new commits.

class Repo(*root*, *object_store=None*, *bare=None*)

Bases: Repo

Modified Dulwich repository object.

A git repository backed by local disk.

To open an existing repository, call the constructor with the path of the repository.

To create a new repository, use the Repo.init class method.

Parameters **root** (str)

Methods:

do_commit([message, committer, author, ...])	Create a new commit.
init(path[, mkdir])	Create a new repository.
init_bare(path[, mkdir])	Create a new bare repository.
list_remotes()	Returns a mapping of remote names to remote URLs, for the repo's current remotes.
reset_to(sha)	Reset the state of the repository to the given commit sha.

do_commit(*message=None*, *committer=None*, *author=None*, *commit_timestamp=None*, *commit_timezone=None*, *author_timestamp=None*, *author_timezone=None*, *tree=None*, *encoding=None*, *ref=b'HEAD'*, *merge_heads=None*)

Create a new commit.

If not specified, *committer* and *author* default to *get_user_identity(..., 'COMMITTER')* and *get_user_identity(..., 'AUTHOR')* respectively.

Parameters

- **message** (Union[str, bytes, None]) – Commit message. Default None.
- **committer** (Union[str, bytes, None]) – Committer fullname. Default None.
- **author** (Union[str, bytes, None]) – Author fullname. Default None.
- **commit_timestamp** (Optional[float]) – Commit timestamp (defaults to now). Default None.
- **commit_timezone** (Optional[float]) – Commit timestamp timezone (defaults to GMT). Default None.
- **author_timestamp** (Optional[float]) – Author timestamp (defaults to commit timestamp). Default None.
- **author_timezone** (Optional[float]) – Author timestamp timezone (defaults to commit timestamp timezone). Default None.
- **tree** (Optional[Any]) – SHA1 of the tree root to use (if not specified the current index will be committed). Default None.
- **encoding** (Union[str, bytes, None]) – Encoding. Default None.
- **ref** (bytes) – Optional ref to commit to (defaults to current branch). Default b'HEAD'.
- **merge_heads** (Optional[Any]) – Merge heads (defaults to .git/MERGE_HEADS). Default None.

Return type bytes

Returns New commit SHA1

classmethod init(*path*, *mkdir=False*)

Create a new repository.

Parameters

- **path** (Union[str, Path, PathLike]) – Path in which to create the repository.
- **mkdir** (bool) – Whether to create the directory if it doesn't exist. Default False.

Return type *~_R*

classmethod init_bare(*path*, *mkdir=False*)

Create a new bare repository.

Parameters

- **path** (Union[str, Path, PathLike]) – Path in which to create the repository.
- **mkdir** (bool) – Default False.

Return type *~_R*

list_remotes()
Returns a mapping of remote names to remote URLs, for the repo's current remotes.

New in version 0.7.0.

Return type Dict[str, str]

reset_to(*sha*)
Reset the state of the repository to the given commit sha.

Any files added in subsequent commits will be removed, any deleted will be restored, and any modified will be reverted.

New in version 0.8.0.

Parameters **sha** (Union[str, bytes])

_R = TypeVar(_R, bound=Repo)
Type: TypeVar

Invariant TypeVar bound to *southwark.repo.Repo*.

get_user_identity(*config*, *kind=None*)
Determine the identity to use for new commits.

If kind is set, this first checks GIT_${KIND}_NAME and GIT_${KIND}_EMAIL.

If those variables are not set, then it will fall back to reading the user.name and user.email settings from the specified configuration.

If that also fails, then it will fall back to using the current users' identity as obtained from the host system (e.g. the gecos field, $EMAIL, $USER@$(hostname -f).

Parameters

- **config** (StackedConfig)
- **kind** (Optional[str]) – Optional kind to return identity for, usually either 'AUTHOR' or 'COMMITTER'. Default None.

Return type bytes

Returns A user identity

Chapter

SEVEN

southwark.targit

Archive where the changes to the contents are recorded using git.

Exceptions:

BadArchiveError()	Exception to indicate an archive contains files utilising path traversal.

Data:

Modes	Valid modes for opening *TarGit* archives in

Classes:

Status	alias of *southwark.StagedDict*
TarGit(filename[, mode])	A "TarGit" (pronounced "target") is a `tar.gz` archive where the changes to the contents are recorded using git.
SaveState(id, user, device, time, timezone)	Represents a save event in a *TarGit* archive's history.

Functions:

check_archive_paths(archive)	Checks the contents of an archive to ensure it does not contain any filenames with absolute paths or path traversal.

exception BadArchiveError

Bases: OSError

Exception to indicate an archive contains files utilising path traversal.

Modes

Valid modes for opening *TarGit* archives in

- `'r'` – Read only access. The archive must exist.
- `'w'` – Read and write access. The archive must not exist.
- `'a'` – Read and write access to an existing archive.

Alias of `Literal['r', 'w', 'a']`

Status

alias of *southwark.StagedDict*

class TarGit (*filename*, *mode='r'*)

Bases: PathLike

A "TarGit" (pronounced "target") is a `tar.gz` archive where the changes to the contents are recorded using git.

Parameters

- **filename** (Union[str, Path, PathLike]) – The filename of the archive.
- **mode** (Literal['r', 'w', 'a']) – The mode to open the file in. Default `'r'`.

Raises

- **FileNotFoundError** – If the file is opened in read or append mode, but it does not exist.
- **FileExistsError** – If the file is opened in write mode, but it already exists.
- **ValueError** – If an unknown value for `mode` is given.

Methods:

save()	Saves the contents of the archive.
status()	Returns the status of the TarGit archive.
exists()	Returns whether the *TarGit* archive exists.
close()	Closes the *TarGit* archive.
__truediv__(filename)	Returns a PathPlus object representing the given filename relative to the archive root.
__repr__()	Returns a string representation of the *TarGit*.
__fspath__()	Returns the filename of the *TarGit* archive.
__str__()	Returns the filename of the *TarGit* archive.

Attributes:

closed	Returns whether the *TarGit* archive is closed.
mode	Returns the mode the *TarGit* archive was opened in.
history	Returns an iterable over the historic save states of the *TarGit*.

save()

Saves the contents of the archive.

Does nothing if there are no changes to be saved.

Return type bool

Returns Whether there were any changes to save.

Raises **IOError** – If the file is closed, or if it was opened in read-only mode.

status()

Returns the status of the TarGit archive.

The values in the dictionary are lists of filenames, relative to the TarGit root.

Raises **IOError** – If the file is closed.

Return type *StagedDict*

exists()
Returns whether the *TarGit* archive exists.

Return type bool

close()
Closes the *TarGit* archive.

property closed
Returns whether the *TarGit* archive is closed.

Return type bool

property mode
Returns the mode the *TarGit* archive was opened in.

This defaults to 'r'. After the archive is closed this will show the last mode until the archive is opened again.

Return type Literal['r', 'w', 'a']

__truediv__(*filename*)
Returns a PathPlus object representing the given filename relative to the archive root.

Parameters **filename**

__repr__()
Returns a string representation of the *TarGit*.

Return type str

__fspath__()
Returns the filename of the *TarGit* archive.

Return type str

__str__()
Returns the filename of the *TarGit* archive.

Return type str

property history
Returns an iterable over the historic save states of the *TarGit*. :rtype: Iterator[*SaveState*] :return:

check_archive_paths(*archive*)
Checks the contents of an archive to ensure it does not contain any filenames with absolute paths or path traversal.

For example, the following paths would raise an *BadArchiveError*:

- /usr/bin/malware.sh – this is an absolute path.
- ~/.local/bin/malware.sh – this tries to put the file in the user's home directory.
- ../.local/bin/malware.sh – this uses path traversal to try to get to a parent directory.

See also: The warning for tarfile.TarFile.extractall() in the Python documentation.

Parameters **archive** (TarFile)

Return type bool

namedtuple SaveState(*id*, *user*, *device*, *time*, *timezone*)

Bases: NamedTuple

Represents a save event in a *TarGit* archive's history.

Fields

0) **id** (str) – The SHA id of the underlying commit.
1) **user** (str) – The name of the user who made the changes.
2) **device** (str) – The hostname of the device the changes were made on.
3) **time** (float) – The time the changes were saved, in seconds from epoch.
4) **timezone** (int) – The timezone the changes were made in, as a GMT offset in seconds.

format_time()

Format the save state's time in the following format:

```
Thu Oct 29 2020 15:53:52 +0000
```

where +0000 represents GMT.

Return type str

__repr__()

Return a string representation of the *SaveState*.

Return type str

Chapter

EIGHT

Downloading source code

The `Southwark` source code is available on GitHub, and can be accessed from the following URL: `https://github.com/repo-helper/southwark`

If you have `git` installed, you can clone the repository with the following command:

```
$ git clone https://github.com/repo-helper/southwark
```

```
Cloning into 'southwark'...
remote: Enumerating objects: 47, done.
remote: Counting objects: 100% (47/47), done.
remote: Compressing objects: 100% (41/41), done.
remote: Total 173 (delta 16), reused 17 (delta 6), pack-reused 126
Receiving objects: 100% (173/173), 126.56 KiB | 678.00 KiB/s, done.
Resolving deltas: 100% (66/66), done.
```

Alternatively, the code can be downloaded in a 'zip' file by clicking:
Clone or download –> Download Zip

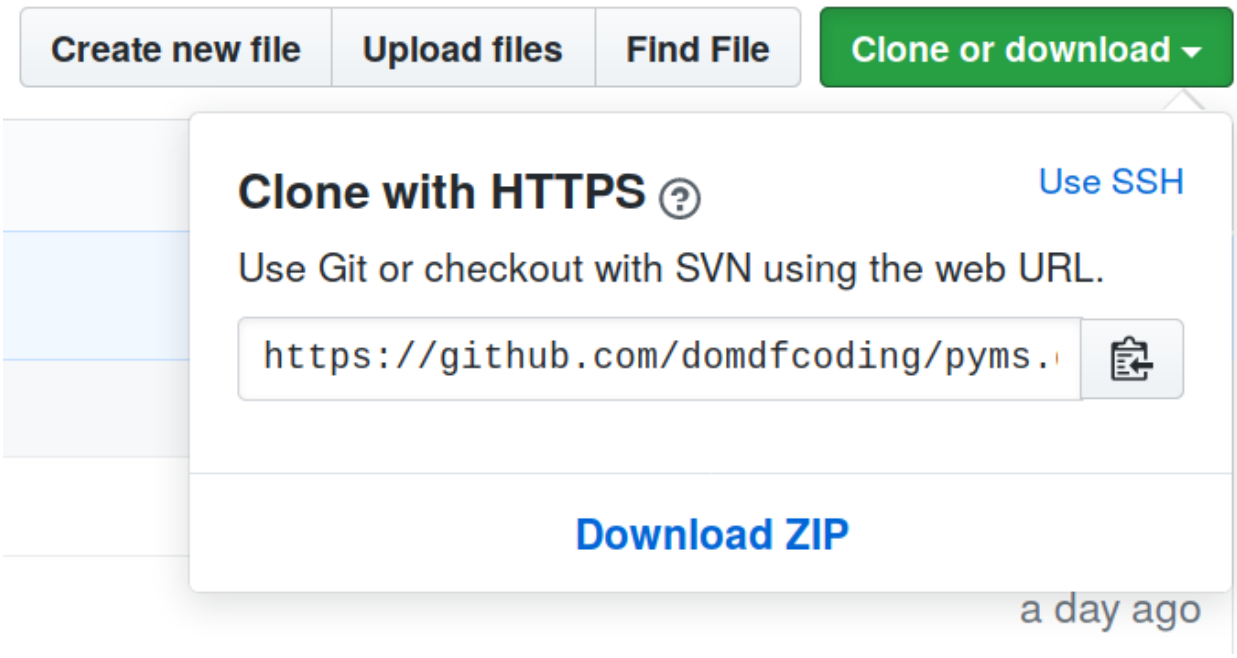

Fig. 1: Downloading a 'zip' file of the source code

8.1 Building from source

The recommended way to build `Southwark` is to use tox:

```
$ tox -e build
```

The source and wheel distributions will be in the directory `dist`.

If you wish, you may also use pep517.build or another **PEP 517**-compatible build tool.

Chapter

NINE

License

`Southwark` is licensed under the MIT License

A short and simple permissive license with conditions only requiring preservation of copyright and license notices. Licensed works, modifications, and larger works may be distributed under different terms and without source code.

Permissions

- Commercial use – The licensed material and derivatives may be used for commercial purposes.
- Modification – The licensed material may be modified.
- Distribution – The licensed material may be distributed.
- Private use – The licensed material may be used and modified in private.

Conditions

- License and copyright notice – A copy of the license and copyright notice must be included with the licensed material.

Limitations

- Liability – This license includes a limitation of liability.
- Warranty – This license explicitly states that it does NOT provide any warranty.

See more information on choosealicense.com ⇒

Python Module Index

S

Index

Symbols

A

B

C

D

E

F

G

H

I

L

M

O

P

R

S

T

U

W