
Southwark

Release 0.9.0

Extensions to the Dulwich Git library.

Dominic Davis-Foster

May 15, 2024

Contents

1	Installation	1
1.1	from PyPI	1
1.2	from Anaconda	1
1.3	from GitHub	1
2	southwark	3
2.1	GitStatus	3
2.2	StagedDict	4
2.3	_DR	4
2.4	assert_clean	4
2.5	check_git_status	4
2.6	clone	4
2.7	get_tags	5
2.8	get_tree_changes	5
2.9	get_untracked_paths	5
2.10	open_repo_closing	5
2.11	status	5
2.12	windows_clone_helper	6
3	southwark.click	7
3.1	commit_message_option	7
3.2	commit_option	7
4	southwark.config	9
4.1	get_remotes	9
4.2	set_remote_http	9
4.3	set_remote_ssh	9
5	southwark.log	11
5.1	Log	11
6	southwark.repo	13
6.1	Repo	13
6.2	_R	15
6.3	get_user_identity	15
7	southwark.targit	17
7.1	BadArchiveError	17
7.2	Modes	17
7.3	TarGit	18
7.4	check_archive_paths	19
7.5	SaveState	20

8	Downloading source code	21
8.1	Building from source	22
9	License	23
	Python Module Index	25
	Index	27

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
```


Extensions to the Dulwich Git library.

Classes:

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

Data:

<code>_DR</code>	Invariant <code>TypeVar</code> bound to <code>dulwich.repo.Repo</code> .
------------------	--

Functions:

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

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.BufferedReader
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.BufferedReader 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 `ContextManager`

Overloads

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

<pre>with windows_clone_helper(): repo = clone(...)</pre>

Return type `Iterator[None]`

southwark.click

Extensions to `click`.

New in version 0.5.0.

Functions:

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

southwark.config

Utilities for repository configuration.

New in version 0.5.0.

Functions:

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

southwark.log

Python implementation of `git log`.

Classes:

<code>Log([repo])</code>	Python implementation of <code>git log</code> .
--------------------------	---

class `Log` (*repo*='.')

Bases: `object`

Python implementation of `git log`.

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

Attributes:

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

Methods:

<code>format_commit(commit)</code>	Return a human-readable commit log entry.
<code>log([max_entries, reverse, from_date, ...])</code>	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.

southwark.repo

Modified Dulwich repository object.

New in version 0.3.0.

Classes:

<code>Repo(root[, object_store, bare])</code>	Modified Dulwich repository object.
---	-------------------------------------

Data:

<code>_R</code>	Invariant <code>TypeVar</code> bound to <code>southwark.repo.Repo</code> .
-----------------	--

Functions:

<code>get_user_identity(config[, kind])</code>	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:

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

southwark.targit

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

Exceptions:

<code>BadArchiveError()</code>	Exception to indicate an archive contains files utilising path traversal.
--------------------------------	---

Data:

<code>Modes</code>	Valid modes for opening <code>TarGit</code> archives in
--------------------	---

Classes:

<code>Status</code>	alias of <code>southwark.StagedDict</code>
<code>TarGit(filename[, mode])</code>	A “TarGit” (pronounced “target”) is a <code>tar.gz</code> archive where the changes to the contents are recorded using <code>git</code> .
<code>SaveState(id, user, device, time, timezone)</code>	Represents a save event in a <code>TarGit</code> archive’s history.

Functions:

<code>check_archive_paths(archive)</code>	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:

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

Attributes:

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

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`

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

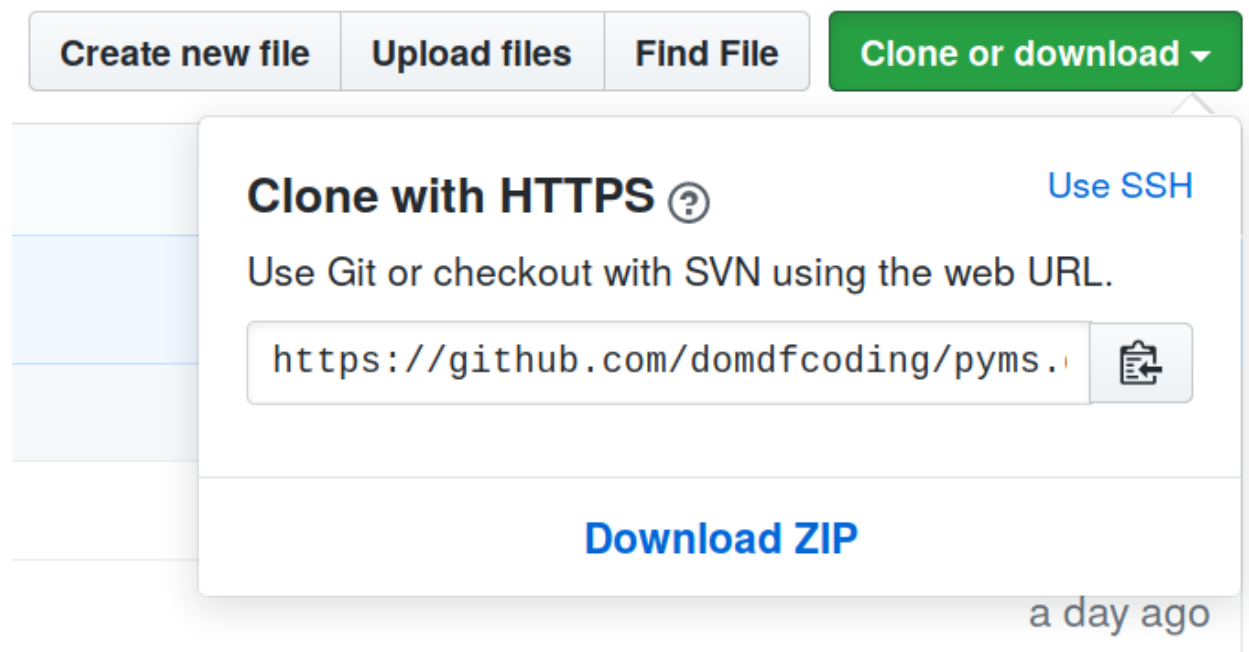


Fig. 1: Downloading a 'zip' file of the source code

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.

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](#) ⇒

```
Copyright (c) 2020 Dominic Davis-Foster
```

```
Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:
```

```
The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.
```

```
THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.
IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM,
DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR
OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE
OR OTHER DEALINGS IN THE SOFTWARE.
```


Python Module Index

S

- `southwark`, [3](#)
- `southwark.click`, [7](#)
- `southwark.config`, [9](#)
- `southwark.log`, [11](#)
- `southwark.repo`, [13](#)
- `southwark.targit`, [17](#)

Symbols

`_DR` (in module *southwark*), 4
`_R` (in module *southwark.repo*), 15
`__fspath__` () (*TarGit* method), 19
`__repr__` () (*GitStatus* method), 3
`__repr__` () (*SaveState* method), 20
`__repr__` () (*TarGit* method), 19
`__str__` () (*TarGit* method), 19
`__truediv__` () (*TarGit* method), 19

A

`assert_clean` () (in module *southwark*), 4

B

`BadArchiveError`, 17

C

`check_archive_paths` () (in module *southwark.targit*), 19
`check_git_status` () (in module *southwark*), 4
`clone` () (in module *southwark*), 4
`close` () (*TarGit* method), 19
`closed` () (*TarGit* property), 19
`commit_message_option` () (in module *southwark.click*), 7
`commit_option` () (in module *southwark.click*), 7
`current_branch` (*Log* attribute), 11

D

`device` (*namedtuple* field)
 `SaveState` (*namedtuple* in *southwark.targit*), 20
`do_commit` () (*Repo* method), 13

E

`exists` () (*TarGit* method), 18

F

`format_commit` () (*Log* method), 11
`format_time` () (*SaveState* method), 20

G

`get_remotes` () (in module *southwark.config*), 9
`get_tags` () (in module *southwark*), 5

`get_tree_changes` () (in module *southwark*), 5
`get_untracked_paths` () (in module *southwark*), 5
`get_user_identity` () (in module *southwark.repo*), 15
`GitStatus` (*namedtuple* in *southwark*), 3
 `staged` (*namedtuple* field), 3
 `unstaged` (*namedtuple* field), 3
 `untracked` (*namedtuple* field), 3

H

`history` () (*TarGit* property), 19

I

`id` (*namedtuple* field)
 `SaveState` (*namedtuple* in *southwark.targit*), 20
`init` () (*Repo* class method), 14
`init_bare` () (*Repo* class method), 14

L

`list_remotes` () (*Repo* method), 14
`local_branches` (*Log* attribute), 11
`Log` (class in *southwark.log*), 11
`log` () (*Log* method), 12

M

MIT License, 23
`mode` () (*TarGit* property), 19
`Modes` (in module *southwark.targit*), 17
`module`
 southwark, 3
 southwark.click, 7
 southwark.config, 9
 southwark.log, 11
 southwark.repo, 13
 southwark.targit, 17

O

`open_repo_closing` () (in module *southwark*), 5

P

Python Enhancement Proposals
 PEP 517, 22

R

`refs` (*Log attribute*), 12
`remote_branches` (*Log attribute*), 12
`Repo` (*class in southwark.repo*), 13
`repo` (*Log attribute*), 12
`reset_to()` (*Repo method*), 15

S

`save()` (*TarGit method*), 18
`SaveState` (*namedtuple in southwark.targit*), 20
 `device` (*namedtuple field*), 20
 `id` (*namedtuple field*), 20
 `time` (*namedtuple field*), 20
 `timezone` (*namedtuple field*), 20
 `user` (*namedtuple field*), 20
`set_remote_http()` (*in module southwark.config*), 9
`set_remote_ssh()` (*in module southwark.config*), 9
`southwark`
 `module`, 3
`southwark.click`
 `module`, 7
`southwark.config`
 `module`, 9
`southwark.log`
 `module`, 11
`southwark.repo`
 `module`, 13
`southwark.targit`
 `module`, 17
`staged` (*namedtuple field*)
 `GitStatus` (*namedtuple in southwark*), 3
`StagedDict` (*typeddict in southwark*), 3
`Status` (*in module southwark.targit*), 17
`status()` (*in module southwark*), 5
`status()` (*TarGit method*), 18

T

`tags` (*Log attribute*), 12
`TarGit` (*class in southwark.targit*), 17
`time` (*namedtuple field*)
 `SaveState` (*namedtuple in southwark.targit*), 20
`timezone` (*namedtuple field*)
 `SaveState` (*namedtuple in southwark.targit*), 20

U

`unstaged` (*namedtuple field*)
 `GitStatus` (*namedtuple in southwark*), 3
`untracked` (*namedtuple field*)
 `GitStatus` (*namedtuple in southwark*), 3
`user` (*namedtuple field*)
 `SaveState` (*namedtuple in southwark.targit*), 20

W

`windows_clone_helper()` (*in module southwark*), 6