





















## 1.5 Documentation

<https://python-redis-lock.readthedocs.io/en/latest/>

## 1.6 Development

To run the all tests run:

```
tox
```

## 1.7 Requirements

**OS** Any

**Runtime** Python 2.7, 3.3 or later, or PyPy

**Services** Redis 2.6.12 or later.

## 1.8 Similar projects

- [bbangert/retools](#) - acquire does spinloop
- [distributing-locking-python-and-redis](#) - acquire does polling
- [cezarsa/redis\\_lock](#) - acquire does not block
- [andymccurdy/redis-py](#) - acquire does spinloop
- [mpessas/python-redis-lock](#) - blocks fine but no expiration
- [brainix/pottery](#) - acquire does spinloop

## CHAPTER 2

---

### Installation

---

At the command line:

```
pip install python-redis-lock
```



## CHAPTER 3

---

### Usage

---

To use redis-lock in a project:

```
import redis_lock
```

Blocking lock:

```
conn = StrictRedis()
lock = redis_lock.Lock(conn, "name-of-the-lock"):
if lock.acquire():
    print("Got the lock. Doing some work ...")
    time.sleep(5)
```

Blocking lock with timeout:

```
conn = StrictRedis()
lock = redis_lock.Lock(conn, "name-of-the-lock"):
if lock.acquire(timeout=3):
    print("Got the lock. Doing some work ...")
    time.sleep(5)
else:
    print("Someone else has the lock.")
```

Non-blocking lock:

```
conn = StrictRedis()
lock = redis_lock.Lock(conn, "name-of-the-lock"):
if lock.acquire(blocking=False):
    print("Got the lock. Doing some work ...")
    time.sleep(5)
else:
    print("Someone else has the lock.")
```

Releasing previously acquired lock:

```
conn = StrictRedis()
lock = redis_lock.Lock(conn, "name-of-the-lock")
lock.acquire()
print("Got the lock. Doing some work ...")
time.sleep(5)
lock.release()
```

The above example could be rewritten using context manager:

```
conn = StrictRedis()
with redis_lock.Lock(conn, "name-of-the-lock"):
    print("Got the lock. Doing some work ...")
    time.sleep(5)
```

In cases, where lock not necessarily in acquired state, and user need to ensure, that it has a matching `id`, example:

```
lock1 = Lock(conn, "foo")
lock1.acquire()
lock2 = Lock(conn, "foo", id=lock1.id)
lock2.release()
```

To check if lock with same name is already locked (it can be this or another lock with identical names):

```
is_locked = Lock(conn, "lock-name").locked()
```

You can control the log output by modifying various loggers:

```
logging.getLogger("redis_lock.thread").disabled = True
logging.getLogger("redis_lock").disable(logging.DEBUG)
```

## 4.1 redis\_lock

**exception** `redis_lock.AlreadyAcquired`

**exception** `redis_lock.AlreadyStarted`

**exception** `redis_lock.InvalidTimeout`

**class** `redis_lock.Lock` (*redis\_client, name, expire=None, id=None, auto\_renewal=False, strict=True, signal\_expire=1000*)

A Lock context manager implemented via redis SETNX/BLPOP.

**acquire** (*blocking=True, timeout=None*)

### Parameters

- **blocking** – Boolean value specifying whether lock should be blocking or not.
- **timeout** – An integer value specifying the maximum number of seconds to block.

**extend** (*expire=None*)

Extends expiration time of the lock.

**Parameters** **expire** – New expiration time. If `None` - *expire* provided during lock initialization will be taken.

**locked** ()

Return true if the lock is acquired.

Checks that lock with same name already exists. This method returns true, even if lock have another id.

**release** ()

Releases the lock, that was acquired with the same object.

---

**Note:** If you want to release a lock that you acquired in a different place you have two choices:

- Use `Lock("name", id=id_from_other_place).release()`

- `Use Lock ("name").reset ()`
- 

**reset ()**

Forcibly deletes the lock. Use this with care.

**exception** `redis_lock.NotAcquired`

**exception** `redis_lock.NotExpirable`

**exception** `redis_lock.TimeoutNotUsable`

**exception** `redis_lock.TimeoutTooLarge`

`redis_lock.reset_all (redis_client)`

Forcibly deletes all locks if its remains (like a crash reason). Use this with care.

**Parameters** `redis_client` – An instance of `StrictRedis`.



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

### 5.1 Bug reports

When [reporting a bug](#) please include:

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

### 5.2 Documentation improvements

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

### 5.3 Feature requests and feedback

The best way to send feedback is to file an issue at <https://github.com/ionelmc/python-redis-lock/issues>.

If you are proposing a feature:

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

## 5.4 Development

To set up *python-redis-lock* for local development:

1. Fork [python-redis-lock](#) (look for the “Fork” button).
2. Clone your fork locally:

```
git clone git@github.com:YOURGITHUBNAME/python-redis-lock.git
```

3. Create a branch for local development:

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

Now you can make your changes locally.

4. When you’re done making changes run all the checks and docs builder with `tox` one command:

```
tox
```

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

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

6. Submit a pull request through the GitHub website.

### 5.4.1 Pull Request Guidelines

If you need some code review or feedback while you’re developing the code just make the pull request.

For merging, you should:

1. Include passing tests (run `tox`).
2. Update documentation when there’s new API, functionality etc.
3. Add a note to `CHANGELOG.rst` about the changes.
4. Add yourself to `AUTHORS.rst`.

### 5.4.2 Tips

To run a subset of tests:

```
tox -e envname -- pytest -k test_myfeature
```

To run all the test environments in *parallel*:

```
tox -p auto
```

## CHAPTER 6

---

### Authors

---

- Ionel Cristian Mărieș - <https://blog.ionelmc.ro>
- Rob Terhaar - <https://github.com/robbyt>
- Corey Farwell - <http://rwell.org>
- Andrey Kobyshev - <https://github.com/yokotoka>
- Jardel Weyrich - <https://twitter.com/jweyrich>
- Victor Torres - <https://github.com/victor-torres>
- Andrew Pashkin - <https://github.com/AndreiPashkin>
- Tero Vuotila - <https://github.com/tvuotila>
- Joel Höner - <https://github.com/athre0z>
- Julie MacDonell - <https://github.com/juliemacdonell>
- Julien Heller - <https://github.com/flux627>
- Przemysław Suliga - <https://github.com/suligap>
- Artem Slobodkin - <https://github.com/artslob>
- Salomon Smeke Cohen - <https://github.com/SalomonSmeke>



### 7.1 4.0.0 (2022-10-17)

- Dropped support for Python 2.7 and 3.6.
- Switched from Travis to GitHub Actions.
- Made logging messages more consistent.
- Replaced the `redis_lock.refresh.thread.*` loggers with a single `redis_lock.refresh.thread` logger.
- Various testing cleanup (mainly removal of hardcoded tmp paths).

### 7.2 3.7.0 (2020-11-20)

- Made logger names more specific. Now can have granular filtering on these new logger names:
  - `redis_lock.acquire` (emits *DEBUG* messages)
  - `redis_lock.acquire` (emits *WARN* messages)
  - `redis_lock.acquire` (emits *INFO* messages)
  - `redis_lock.refresh.thread.start` (emits *DEBUG* messages)
  - `redis_lock.refresh.thread.exit` (emits *DEBUG* messages)
  - `redis_lock.refresh.start` (emits *DEBUG* messages)
  - `redis_lock.refresh.shutdown` (emits *DEBUG* messages)
  - `redis_lock.refresh.exit` (emits *DEBUG* messages)
  - `redis_lock.release` (emits *DEBUG* messages)

Contributed by Salomon Smeke Cohen in [PR #80](#).

- Fixed few CI issues regarding doc checks. Contributed by Salomon Smeke Cohen in [PR #81](#).

## 7.3 3.6.0 (2020-07-23)

- Improved `timeout/expire` validation so that:
  - `timeout` and `expire` are converted to `None` if they are falsy. Previously only `None` disabled these options, other falsy values created buggy situations.
  - Using `timeout` greater than `expire` is now allowed, if `auto_renewal` is set to `True`. Previously a `TimeoutTooLarge` error was raised. See [#74](#).
  - Negative `timeout` or `expire` are disallowed. Previously such values were allowed, and created buggy situations. See [#73](#).
- Updated benchmark and examples.
- Removed the custom script caching code. Now the `register_script` method from the redis client is used. This will fix possible issue with redis clusters in theory, as the redis client has some specific handling for that.

## 7.4 3.5.0 (2020-01-13)

- Added a `locked` method. Contributed by Artem Slobodkin in [PR #72](#).

## 7.5 3.4.0 (2019-12-06)

- Fixed regression that can cause deadlocks or slowdowns in certain configurations. See: [#71](#).

## 7.6 3.3.1 (2019-01-19)

- Fixed failures when running python-redis-lock 3.3 alongside 3.2. See: [#64](#).

## 7.7 3.3.0 (2019-01-17)

- Fixed deprecated use of `warnings` API. Contributed by Julie MacDonell in [PR #54](#).
- Added `auto_renewal` option in `RedisCache.lock` (the Django cache backend wrapper). Contributed by c in [PR #55](#).
- Changed log level for “%(script)s not cached” from `WARNING` to `INFO`.
- Added support for using `decode_responses=True`. Lock keys are pure ascii now.

## 7.8 3.2.0 (2016-10-29)

- Changed the signal key cleanup operation do be done without any expires. This prevents lingering keys around for some time. Contributed by Andrew Pashkin in [PR #38](#).

- Allow locks with given *id* to acquire. Previously it assumed that if you specify the *id* then the lock was already acquired. See [#44](#) and [#39](#).
- Allow using other redis clients with a `strict=False`. Normally you're expected to pass in an instance of `redis.StrictRedis`.
- Added convenience method `locked_get_or_set` to Django cache backend.

## 7.9 3.1.0 (2016-04-16)

- Changed the auto renewal to automatically stop the renewal thread if lock gets garbage collected. Contributed by Andrew Pashkin in [PR #33](#).

## 7.10 3.0.0 (2016-01-16)

- Changed `release` so that it expires signal-keys immediately. Contributed by Andrew Pashkin in [PR #28](#).
- Resetting locks (`reset` or `reset_all`) will release the lock. If there's someone waiting on the reset lock now it will acquire it. Contributed by Andrew Pashkin in [PR #29](#).
- Added the `extend` method on `Lock` objects. Contributed by Andrew Pashkin in [PR #24](#).
- Documentation improvements on `release` method. Contributed by Andrew Pashkin in [PR #22](#).
- Fixed `acquire(block=True)` handling when `expire` option was used (it wasn't blocking indefinitely). Contributed by Tero Vuotila in [PR #35](#).
- Changed `release` to check if lock was acquired with the same `id`. If not, `NotAcquired` will be raised. Previously there was just a check if it was acquired with the same instance (`self._held`). **BACKWARDS INCOMPATIBLE**
- Removed the `force` option from `release` - it wasn't really necessary and it only encourages sloppy programming. See [#25](#). **BACKWARDS INCOMPATIBLE**
- Dropped tests for Python 2.6. It may work but it is unsupported.

## 7.11 2.3.0 (2015-09-27)

- Added the `timeout` option. Contributed by Victor Torres in [PR #20](#).

## 7.12 2.2.0 (2015-08-19)

- Added the `auto_renewal` option. Contributed by Nick Groenen in [PR #18](#).

## 7.13 2.1.0 (2015-03-12)

- New specific exception classes: `AlreadyAcquired` and `NotAcquired`.
- Slightly improved efficiency when non-waiting acquires are used.

## 7.14 2.0.0 (2014-12-29)

- Rename `Lock.token` to `Lock.id`. Now only allowed to be set via constructor. Contributed by Jardel Weyrich in [PR #11](#).

## 7.15 1.0.0 (2014-12-23)

- Fix Django integration. (reported by Jardel Weyrich)
- Reorganize tests to use `py.test`.
- Add test for Django integration.
- Add `reset_all` functionality. Contributed by Yokotoka in [PR #7](#).
- Add `Lock.reset` functionality.
- Expose the `Lock.token` attribute.

## 7.16 0.1.2 (2013-11-05)

- ?

## 7.17 0.1.1 (2013-10-26)

- ?

## 7.18 0.1.0 (2013-10-26)

- ?

## 7.19 0.0.1 (2013-10-25)

- First release on PyPI.



## CHAPTER 8

---

### Indices and tables

---

- `genindex`
- `modindex`
- `search`



**r**

`redis_lock`, [11](#)



### A

`acquire()` (*redis\_lock.Lock method*), 11  
`AlreadyAcquired`, 11  
`AlreadyStarted`, 11

### E

`extend()` (*redis\_lock.Lock method*), 11

### I

`InvalidTimeout`, 11

### L

`Lock` (*class in redis\_lock*), 11  
`locked()` (*redis\_lock.Lock method*), 11

### N

`NotAcquired`, 12  
`NotExpirable`, 12

### R

`redis_lock` (*module*), 11  
`release()` (*redis\_lock.Lock method*), 11  
`reset()` (*redis\_lock.Lock method*), 12  
`reset_all()` (*in module redis\_lock*), 12

### T

`TimeoutNotUsable`, 12  
`TimeoutTooLarge`, 12