



## EXPIRE key seconds

Available since 1.0.0.

Time complexity:  $O(1)$

Set a timeout on key. After the timeout has expired, the key will automatically be deleted. A key with an associated timeout is often said to be *volatile* in Redis terminology.

The timeout will only be cleared by commands that delete or overwrite the contents of the key, including [DEL](#), [SET](#), [GETSET](#) and all the \*STORE commands. This means that all the operations that conceptually *alter* the value stored at the key without replacing it with a new one will leave the timeout untouched. For instance, incrementing the value of a key with [INCR](#), pushing a new value into a list with [LPUSH](#), or altering the field value of a hash with [HSET](#) are all operations that will leave the timeout untouched.

The timeout can also be cleared, turning the key back into a persistent key, using the [PERSIST](#) command.

If a key is renamed with [RENAME](#), the associated time to live is transferred to the new key name.

If a key is overwritten by [RENAME](#), like in the case of an existing key Key\_A that is overwritten by a call like `RENAME Key_B Key_A`, it does not matter if the original Key\_A had a timeout associated or not, the new key Key\_A will inherit all the characteristics of Key\_B.

Note that calling [EXPIRE/PEXPIRE](#) with a non-positive timeout or [EXPIREAT/PEXPIREAT](#) with a time in the past will result in the key being [deleted](#) rather than expired (accordingly, the emitted [key event](#) will be `del`, not `expired`).

### Refreshing expires

It is possible to call [EXPIRE](#) using as argument a key that already has an existing expire set. In this case the time to live of a key is *updated* to the new value. There are many useful applications for this, an example is documented in the *Navigation session* pattern section below.

### Differences in Redis prior 2.1.3

In Redis versions prior **2.1.3** altering a key with an expire set using a command altering its value had the effect of removing the key entirely. This semantics was needed because of limitations in the replication layer that are now fixed.

[EXPIRE](#) would return 0 and not alter the timeout for a key with a timeout set.

## Return value

[Integer reply](#), specifically:

- 1 if the timeout was set.
- 0 if key does not exist.

## Examples

```
redis> SET mykey "Hello"
"OK"
redis> EXPIRE mykey 10
(integer) 1
redis> TTL mykey
(integer) 10
redis> SET mykey "Hello World"
"OK"
redis> TTL mykey
(integer) -1
redis>
```

## Pattern: Navigation session

Imagine you have a web service and you are interested in the latest N pages *recently* visited by your users, such that each adjacent page view was not performed more than 60 seconds after the previous. Conceptually you may consider this set of page views as a *Navigation session* of your user, that may contain interesting information about what kind of products he or she is looking for currently, so that you can recommend related products.

You can easily model this pattern in Redis using the following strategy: every time the user does a page view you call the following commands:

```
MULTI
RPUSH pagewviews.user:<userid> http://.....
EXPIRE pagewviews.user:<userid> 60
EXEC
```

If the user will be idle more than 60 seconds, the key will be deleted and only subsequent page views that have less than 60 seconds of difference will be recorded.

This pattern is easily modified to use counters using [INCR](#) instead of lists using [RPUSH](#).

## Appendix: Redis expires

## Keys with an expire

Normally Redis keys are created without an associated time to live. The key will simply live forever, unless it is removed by the user in an explicit way, for instance using the [DEL](#) command.

The [EXPIRE](#) family of commands is able to associate an expire to a given key, at the cost of some additional memory used by the key. When a key has an expire set, Redis will make sure to remove the key when the specified amount of time elapsed.

The key time to live can be updated or entirely removed using the [EXPIRE](#) and [PERSIST](#) command (or other strictly related commands).

## Expire accuracy

In Redis 2.4 the expire might not be pin-point accurate, and it could be between zero to one seconds out.

Since Redis 2.6 the expire error is from 0 to 1 milliseconds.

## Expires and persistence

Keys expiring information is stored as absolute Unix timestamps (in milliseconds in case of Redis version 2.6 or greater). This means that the time is flowing even when the Redis instance is not active.

For expires to work well, the computer time must be taken stable. If you move an RDB file from two computers with a big desync in their clocks, funny things may happen (like all the keys loaded to be expired at loading time).

Even running instances will always check the computer clock, so for instance if you set a key with a time to live of 1000 seconds, and then set your computer time 2000 seconds in the future, the key will be expired immediately, instead of lasting for 1000 seconds.

## How Redis expires keys

Redis keys are expired in two ways: a passive way, and an active way.

A key is passively expired simply when some client tries to access it, and the key is found to be timed out.

Of course this is not enough as there are expired keys that will never be accessed again. These keys should be expired anyway, so periodically Redis tests a few keys at random among keys with an expire set. All the keys that are already expired are deleted from the keyspace.

Specifically this is what Redis does 10 times per second:

1. Test 20 random keys from the set of keys with an associated expire.
2. Delete all the keys found expired.

3. If more than 25% of keys were expired, start again from step 1.

This is a trivial probabilistic algorithm, basically the assumption is that our sample is representative of the whole key space, and we continue to expire until the percentage of keys that are likely to be expired is under 25%

This means that at any given moment the maximum amount of keys already expired that are using memory is at max equal to max amount of write operations per second divided by 4.

## How expires are handled in the replication link and AOF file

In order to obtain a correct behavior without sacrificing consistency, when a key expires, a [DEL](#) operation is synthesized in both the AOF file and gains all the attached replicas nodes. This way the expiration process is centralized in the master instance, and there is no chance of consistency errors.

However while the replicas connected to a master will not expire keys independently (but will wait for the [DEL](#) coming from the master), they'll still take the full state of the expires existing in the dataset, so when a replica is elected to master it will be able to expire the keys independently, fully acting as a master.


Related commands

- [COPY](#)
- [DEL](#)
- [DUMP](#)
- [EXISTS](#)
- **[EXPIRE](#)**
- [EXPIREAT](#)
- [KEYS](#)
- [MIGRATE](#)
- [MOVE](#)
- [OBJECT](#)
- [PERSIST](#)
- [PEXPIRE](#)
- [PEXPIREAT](#)
- [PTTL](#)
- [RANDOMKEY](#)
- [RENAME](#)
- [RENAMENX](#)
- [RESTORE](#)
- [SCAN](#)
- [SORT](#)
- [TOUCH](#)
- [TTL](#)

- [TYPE](#)
- [UNLINK](#)
- [WAIT](#)

---

This website is open source software. See all credits.

Sponsored by  **redislabs**  
HOME OF REDIS