Redis cheatsheet

This is a redis quick reference cheat sheet that lists examples of redis commands

Getting Started

Getting started

Start Redis

```
$ redis-server
```

Connect to redis (Redis CLI client)

```
$ redis-cli
```

Connect to redis (telnet)

```
$ telnet 127.0.0.1 6379
```

Hello World

Ping

```
redis> PING
PONG
```

Hello World

```
redis> SET mykey "Hello world"
OK
redis> GET mykey
"Hello world"
```

Basic Data types

- Strings
- Lists
- Hashes
- Sets
- Sorted Sets

Redis supports 5 basic data types

Redis String command

APPEND

```
APPEND key value
```

Example

```
redis> EXISTS mykey
(integer) 0
redis> APPEND mykey "Hello"
(integer) 5
redis> APPEND mykey " World"
(integer) 11
redis> GET mykey
"Hello World"
```

Append a value to a key

BITCOUNT

```
BITCOUNT key [start end]
```

Example

```
redis> SET mykey "foobar"
"OK"
redis> BITCOUNT mykey
(integer) 26
redis> BITCOUNT mykey 0 0
(integer) 4
redis> BITCOUNT mykey 1 1
(integer) 6
```

Count set bits in a string

BITFIELD

```
BITFIELD key [GET type offset] [SET type offset value] [INCRBY type offset increment] [OVERFLOW WRAP|SAT|FAIL]
```

Example

```
redis> BITFIELD mykey INCRBY i5 100 1 GET
1) (integer) 1
2) (integer) 0
```

Perform arbitrary bitfield integer operations on strings

BITOP

```
BITOP operation destkey key [key ...]
```

Example

```
redis> SET key1 "foobar"
"OK"
redis> SET key2 "abcdef"
"OK"
redis> BITOP AND dest key1 key2
(integer) 6
redis> GET dest
"`bc`ab"
```

Perform bitwise operations between strings

BITPOS

```
BITPOS key bit [start] [end]
```

Example

```
redis> SET mykey "\xff\xf0\x00"
"OK"
redis> BITPOS mykey 0
(integer) 12
redis> SET mykey "\x00\xff\xf0"
"OK"
redis> BITPOS mykey 1 0
(integer) 8
redis> BITPOS mykey 1 2
(integer) 16
redis> set mykey "\x00\x00\x00"
"OK"
redis> BITPOS mykey 1
(integer) -1
```

Find first bit set or clear in a string

DECR

```
DECR key
```

Example

```
redis> SET mykey "10"
"OK"
redis> DECR mykey
(integer) 9
redis> SET mykey "23429348239048094802934E
"OK"
redis> DECR mykey
ERR ERR value is not an integer or out of
```

Decrement the integer value of a key by one

DECRBY

```
DECRBY key decrement
```

Example

```
redis> SET mykey "10"
"OK"
redis> DECRBY mykey 3
(integer) 7
```

GET

```
GET key
```

Example

```
redis> GET nonexisting
(nil)
redis> SET mykey "Hello"
"OK"
redis> GET mykey
"Hello"
```

GETBIT

```
GETBIT key offset
```

Example

```
redis> SETBIT mykey 7 1
(integer) 0
redis> GETBIT mykey 0
(integer) 0
redis> GETBIT mykey 7
(integer) 1
```

Decrement the integer value of a key by the given number

Get the value of a key

```
redis> GETBIT mykey 100
(integer) 0
```

Returns the bit value at offset in the string value stored at key

GETRANGE

```
GETRANGE key start end
```

Example

```
redis> SET mykey "This is a string"
"OK"
redis> GETRANGE mykey 0 3
"This"
redis> GETRANGE mykey -3 -1
"ing"
redis> GETRANGE mykey 0 -1
"This is a string"
redis> GETRANGE mykey 10 100
"string"
```

Get a substring of the string stored at a key

GETSET

```
GETSET key value
```

Example

```
redis> INCR mycounter
(integer) 1
redis> GETSET mycounter "0"
"1"
redis> GET mycounter
"0"
```

Set the string value of a key and return its old value

INCR

```
INCR key
```

Example

```
redis> SET mykey "10"
"OK"
redis> INCR mykey
(integer) 11
redis> GET mykey
"11"
```

Increment the integer value of a key by one

MSETNX

```
MSETNX key value [key value ...]
```

Example

```
redis> MSETNX key1 "Hello" key2 "there"
(integer) 1
redis> MSETNX key2 "new" key3 "world"
(integer) 0
redis> MGET key1 key2 key3
1) "Hello"
2) "there"
3) (nil)
```

Set multiple keys to multiple values, only if none of the keys exist

INCRBYFLOAT

```
INCRBYFLOAT key increment
```

Example

```
redis> SET mykey 10.50
"OK"
redis> INCRBYFLOAT mykey 0.1
"10.6"
redis> INCRBYFLOAT mykey -5
"5.6"
redis> SET mykey 5.0e3
"OK"
redis> INCRBYFLOAT mykey 2.0e2
"5200"
```

Increment the float value of a key by the given amount

MGET

```
MGET key [key ...]
```

Example

```
redis> SET key1 "Hello"
"OK"
redis> SET key2 "World"
"OK"
redis> MGET key1 key2 nonexisting
1) "Hello"
2) "World"
3) (nil)
```

Get the values of all the given keys

MSET

```
MSET key value [key value ...]
```

Example

```
redis> MSET key1 "Hello" key2 "World"
"OK"
redis> GET key1
"Hello"
redis> GET key2
"World"
```

Set multiple keys to multiple values

INCRBY

```
INCRBY key increment
```

Example

```
redis> SET mykey "10"
"OK"
redis> INCRBY mykey 5
(integer) 15
```

Increment the integer value of a key by the given amount

PSETEX

```
PSETEX key milliseconds value
```

Example

```
redis> PSETEX mykey 1000 "Hello"
"OK"
redis> PTTL mykey
(integer) 1000
redis> GET mykey
"Hello"
```

Set the value and expiration in milliseconds of a key

SET

```
SET key value [EX seconds|PX
milliseconds|KEEPTTL] [NX|XX] [GET]
```

Example

```
redis> SET mykey "Hello"
"OK"
redis> GET mykey
"Hello"
redis> SET anotherkey "will expire in a mi
"OK"
```

Set the string value of a key

SETBIT

```
SETBIT key offset value
```

Example

```
redis> SETBIT mykey 7 1
(integer) 0
redis> SETBIT mykey 7 0
(integer) 1
redis> GET mykey
"\u0000"
```

Sets or clears the bit at offset in the string value stored at key

SETEX

```
SETEX key seconds value
```

Example

```
redis> SETEX mykey 10 "Hello"
"OK"
redis> TTL mykey
(integer) 10
redis> GET mykey
"Hello"
```

Set the value and expiration of a key

SETNX

```
SETNX key value
```

Example

```
redis> SETNX mykey "Hello"
(integer) 1
redis> SETNX mykey "World"
(integer) 0
redis> GET mykey
```

SETRANGE

```
SETRANGE key offset value
```

Example

```
redis> SET key1 "Hello World"
"OK"
redis> SETRANGE key1 6 "Redis"
(integer) 11
redis> GET key1
```

STRLEN

```
STRLEN key
```

Example

```
redis> SET mykey "Hello world"
"OK"
redis> STRLEN mykey
(integer) 11
redis> STRLEN nonexisting
```

"Hello"

Set the value of a key, only if the key does not exist

"Hello Redis"

Overwrite part of a string at key starting at the specified offset

(integer) 0

Get the length of the value stored in a key

STRALGO

```
STRALGO LCS algo-specific-argument [algo-specific-argument ...]
```

Example

```
redis> STRALGO LCS KEYS key1 key2 IDX
1) "matches"
2) 1) 1) 1) (integer) 4
         2) (integer) 7
      2) 1) (integer) 5
         2) (integer) 8
   2) 1) 1) (integer) 2
         2) (integer) 3
      2) 1) (integer) 0
         2) (integer) 1
3) "len"
4) (integer) 6
```

Run algorithms (currently LCS) against strings

Redis Set command

SADD

```
SADD key member [member ...]
```

Example

```
redis> SADD myset "Hello"
(integer) 1
redis> SADD myset "World"
(integer) 1
redis> SADD myset "World"
(integer) 0
redis> SMEMBERS myset
1) "Hello"
2) "World"
```

Add one or more members to a set

SCARD

```
SCARD key
```

Example

```
redis> SADD myset "Hello"
(integer) 1
redis> SADD myset "World"
(integer) 1
redis> SCARD myset
(integer) 2
```

Get the number of members in a set

SDIFF

```
SDIFF key [key ...]
```

Example

```
redis> SADD key1 "a"
(integer) 1
redis> SADD key1 "b"
(integer) 1
redis> SADD key1 "c"
(integer) 1
redis> SADD key2 "c"
(integer) 1
redis> SADD key2 "d"
(integer) 1
redis> SADD key2 "e"
(integer) 1
redis> SDIFF key1 key2
1) "a"
2) "b"
```

Subtract multiple sets

SDIFFSTORE

```
SDIFFSTORE destination key [key ...]
```

Example

```
redis> SADD key1 "a"
(integer) 1
redis> SADD key1 "b"
(integer) 1
redis> SADD key1 "c"
(integer) 1
redis> SADD key2 "c"
(integer) 1
redis> SADD key2 "d"
(integer) 1
redis> SADD key2 "e"
(integer) 1
redis> SDIFFSTORE key key1 key2
(integer) 2
redis> SMEMBERS key
1) "a"
2) "b"
```

Subtract multiple sets and store the resulting set in a key

SINTER

```
SINTER key [key ...]
```

Example

```
redis> SADD key1 "a"
(integer) 1
redis> SADD key1 "b"
(integer) 1
redis> SADD key1 "c"
(integer) 1
redis> SADD key2 "c"
(integer) 1
redis> SADD key2 "d"
(integer) 1
redis> SADD key2 "e"
(integer) 1
redis> SINTER key1 key2
1) "c"
```

Intersect multiple sets

SINTERSTORE

```
SINTERSTORE destination key [key ...]
```

Example

```
redis> SADD key1 "a"
(integer) 1
redis> SADD key1 "b"
(integer) 1
redis> SADD key1 "c"
(integer) 1
redis> SADD key2 "c"
(integer) 1
redis> SADD key2 "d"
(integer) 1
redis> SADD key2 "e"
(integer) 1
redis> SINTERSTORE key key1 key2
(integer) 1
redis> SMEMBERS key
1) "c"
```

Intersect multiple sets and store the resulting set in a key

SISMEMBER

```
SISMEMBER key member
```

Example

SMISMEMBER

```
SMISMEMBER key member [member ...]
```

Example

SMEMBERS

```
SMEMBERS key
```

Example

```
redis> SADD myset "one"
(integer) 1
redis> SISMEMBER myset "one"
(integer) 1
redis> SISMEMBER myset "two"
(integer) 0
```

Determine if a given value is a member of a set

```
redis> SADD myset "one"
(integer) 1
redis> SADD myset "one"
(integer) 0
redis> SMISMEMBER myset "one" "notamember"
1) (integer) 1
2) (integer) 0
```

Returns the membership associated with the given elements for a set

```
redis> SADD myset "Hello"
(integer) 1
redis> SADD myset "World"
(integer) 1
redis> SMEMBERS myset
1) "Hello"
2) "World"
```

Get all the members in a set

SMOVE

```
SMOVE source destination member
```

Example

```
redis> SADD myset "one"
(integer) 1
redis> SADD myset "two"
(integer) 1
redis> SADD myotherset "three"
(integer) 1
redis> SMOVE myset myotherset "two"
(integer) 1
redis> SMEMBERS myset
1) "one"
redis> SMEMBERS myotherset
1) "two"
2) "three"
```

Move a member from one set to another

SPOP

```
SPOP key [count]
```

Example

```
redis> SADD myset "one"
(integer) 1
redis> SADD myset "two"
(integer) 1
redis> SADD myset "three"
(integer) 1
redis> SPOP myset
"two"
redis> SMEMBERS myset
1) "one"
2) "three"
redis> SADD myset "four"
(integer) 1
redis> SADD myset "five"
(integer) 1
redis> SPOP myset 3
1) "four"
2) "five"
3) "three"
redis> SMEMBERS myset
1) "one"
```

Remove and return one or multiple random members from a set

SRANDMEMBER

```
SRANDMEMBER key [count]
```

Example

```
redis> SADD myset one two three
(integer) 3
redis> SRANDMEMBER myset
"three"
redis> SRANDMEMBER myset 2
1) "two"
2) "three"
redis> SRANDMEMBER myset -5
1) "one"
2) "two"
3) "three"
4) "three"
5) "one"
```

Get one or multiple random members from a set

SREM

```
SREM key member [member ...]
```

Example

```
redis> SADD myset "one"
(integer) 1
redis> SADD myset "two"
(integer) 1
redis> SADD myset "three"
(integer) 1
redis> SREM myset "one"
(integer) 1
redis> SREM myset "four"
(integer) 0
redis> SMEMBERS myset
1) "two"
2) "three"
```

Remove one or more members from a set

SUNION

```
SUNION key [key ...]
```

Example

```
redis> SADD key1 "a"
(integer) 1
redis> SADD key1 "b"
(integer) 1
redis> SADD key1 "c"
(integer) 1
redis> SADD key2 "c"
(integer) 1
redis> SADD key2 "d"
(integer) 1
redis> SADD key2 "e"
(integer) 1
redis> SUNION key1 key2
1) "a"
2) "c"
3) "e"
4) "b"
5) "d"
```

Add multiple sets

SUNIONSTORE

```
SUNIONSTORE destination key [key ...]
```

Example

```
redis> SADD key1 "a"
(integer) 1
redis> SADD key1 "b"
(integer) 1
redis> SADD key1 "c"
(integer) 1
redis> SADD key2 "c"
(integer) 1
redis> SADD key2 "d"
(integer) 1
redis> SADD key2 "e"
(integer) 1
redis> SUNIONSTORE key key1 key2
(integer) 5
redis> SMEMBERS key
1) "a"
2) "c"
3) "e"
4) "b"
5) "d"
```

Add multiple sets and store the resulting set in a key

Redis List command

Misc

BRPOPLPUSH	Pop an element from a list, push it to another list and return it; or block until one is available
BLMOVE	Pop an element from a list, push it to another list and return it; or block until one is available

BLPOP

```
BLPOP key [key ...] timeout
```

Example

```
redis> DEL list1 list2
(integer) 0
redis> RPUSH list1 a b c
(integer) 3
redis> BLPOP list1 list2 0
1) "list1"
```

BRPOP

```
BRPOP key [key ...] timeout
```

Example

```
redis> DEL list1 list2
(integer) 0
redis> RPUSH list1 a b c
(integer) 3
redis> BRPOP list1 list2 0
1) "list1"
```

```
2) "a"
```

Remove and get the first element in a list, or block until one is available |

```
2) "c"
```

Remove and get the last element in a list, or block until one is available |

LINDEX

```
LINDEX key index
```

Example

```
redis> LPUSH mylist "World"
(integer) 1
redis> LPUSH mylist "Hello"
(integer) 2
redis> LINDEX mylist 0
"Hello"
redis> LINDEX mylist -1
"World"
redis> LINDEX mylist 3
(nil)
```

Get an element from a list by its index

LINSERT

```
LINSERT key BEFORE|AFTER pivot element
```

Example

```
redis> RPUSH mylist "Hello"
(integer) 1
redis> RPUSH mylist "World"
(integer) 2
redis> LINSERT mylist BEFORE "World" "Ther
(integer) 3
redis> LRANGE mylist 0 -1
1) "Hello"
2) "There"
3) "World"
```

Insert an element before or after another element in a list

LLEN

```
LLEN key
```

Example

```
redis> LPUSH mylist "World"
(integer) 1
redis> LPUSH mylist "Hello"
(integer) 2
redis> LLEN mylist
(integer) 2
```

Get the length of a list

LPOP

```
LPOP key [count]
```

Example

```
redis> RPUSH mylist "one"
(integer) 1
redis> RPUSH mylist "two"
(integer) 2
redis> RPUSH mylist "three"
(integer) 3
redis> LPOP mylist
"one"
redis> LRANGE mylist 0 -1
1) "two"
2) "three"
```

Remove and get the first elements in a list

LPOS

```
LPOS key element [RANK rank] [COUNT num-matches] [MAXLEN len]
```

Example

```
redis> RPUSH mylist a b c d 1 2 3 4 3 3 3
(integer) 11
redis> LPOS mylist 3
(integer) 6
redis> LPOS mylist 3 COUNT 0 RANK 2
1) (integer) 8
2) (integer) 9
3) (integer) 10
```

Return the index of matching elements on a list

LPUSH

```
LPUSH key element [element ...]
```

Example

```
redis> LPUSH mylist "world"
(integer) 1
redis> LPUSH mylist "hello"
(integer) 2
redis> LRANGE mylist 0 -1
1) "hello"
2) "world"
```

Prepend one or multiple elements to a list

LPUSHX

```
LPUSHX key element [element ...]
```

Example

```
redis> LPUSH mylist "World"
(integer) 1
redis> LPUSHX mylist "Hello"
(integer) 2
redis> LPUSHX myotherlist "Hello"
(integer) 0
redis> LRANGE mylist 0 -1
1) "Hello"
2) "World"
redis> LRANGE myotherlist 0 -1
(empty list or set)
```

Prepend an element to a list, only if the list exists

LRANGE

```
LRANGE key start stop
```

Example

```
redis> RPUSH mylist "one"
(integer) 1
redis> RPUSH mylist "two"
(integer) 2
redis> RPUSH mylist "three"
(integer) 3
redis> LRANGE mylist 0 0
1) "one"
redis> LRANGE mylist -3 2
1) "one"
2) "two"
3) "three"
redis> LRANGE mylist -100 100
1) "one"
2) "two"
3) "three"
redis> LRANGE mylist 5 10
(empty list or set)
```

Get a range of elements from a list

LREM

```
LREM key count element
```

Example

```
redis> RPUSH mylist "hello"
(integer) 1
redis> RPUSH mylist "hello"
(integer) 2
redis> RPUSH mylist "foo"
(integer) 3
redis> RPUSH mylist "hello"
(integer) 4
redis> LREM mylist -2 "hello"
(integer) 2
redis> LRANGE mylist 0 -1
1) "hello"
2) "foo"
```

Remove elements from a list

LSET

```
LSET key index element
```

Example

```
redis> RPUSH mylist "one"
(integer) 1
redis> RPUSH mylist "two"
(integer) 2
redis> RPUSH mylist "three"
(integer) 3
redis> LSET mylist 0 "four"
"OK"
redis> LSET mylist -2 "five"
```

LTRIM

```
LTRIM key start stop
```

Example

```
redis> RPUSH mylist "one"
(integer) 1
redis> RPUSH mylist "two"
(integer) 2
redis> RPUSH mylist "three"
(integer) 3
redis> LTRIM mylist 1 -1
"OK"
redis> LRANGE mylist 0 -1
```

RPOP

```
RPOP key [count]
```

Example

```
redis> RPUSH mylist "one"
(integer) 1
redis> RPUSH mylist "two"
(integer) 2
redis> RPUSH mylist "three"
(integer) 3
redis> RPOP mylist
"three"
redis> LRANGE mylist 0 -1
```

```
"OK"
redis> LRANGE mylist 0 -1
1) "four"
2) "five"
3) "three"
```

Set the value of an element in a list by its index

```
1) "two"
2) "three"
```

Trim a list to the specified range

```
1) "one"
2) "two"
```

Remove and get the last elements in a list

RPOPLPUSH

```
RPOPLPUSH source destination
```

Example

```
redis> RPUSH mylist "one"
(integer) 1
redis> RPUSH mylist "two"
(integer) 2
redis> RPUSH mylist "three"
(integer) 3
redis> RPOPLPUSH mylist myotherlist
"three"
redis> LRANGE mylist 0 -1
1) "one"
2) "two"
redis> LRANGE myotherlist 0 -1
1) "three"
```

Remove the last element in a list, prepend it to another list and return it

LMOVE

```
LMOVE source destination LEFT|RIGHT
LEFT|RIGHT
```

Example

```
redis> RPUSH mylist "one"
(integer) 1
redis> RPUSH mylist "two"
(integer) 2
redis> RPUSH mylist "three"
(integer) 3
redis> LMOVE mylist myotherlist RIGHT LEFT
"three"
redis> LMOVE mylist myotherlist LEFT RIGHT
"one"
redis> LRANGE mylist 0 -1
1) "two"
redis> LRANGE myotherlist 0 -1
1) "three"
2) "one"
```

Pop an element from a list, push it to another list and return it

RPUSH

```
RPUSH key element [element ...]
```

Example

```
redis> RPUSH mylist "hello"
(integer) 1
redis> RPUSH mylist "world"
(integer) 2
redis> LRANGE mylist 0 -1
1) "hello"
2) "world"
```

Append one or multiple elements to a list

RPUSHX

```
RPUSHX key element [element ...]
```

Example

```
redis> RPUSH mylist "Hello"
(integer) 1
redis> RPUSHX mylist "World"
(integer) 2
redis> RPUSHX myotherlist "World"
(integer) 0
redis> LRANGE mylist 0 -1
1) "Hello"
2) "World"
redis> LRANGE myotherlist 0 -1
(empty list or set)
```

Append an element to a list, only if the list exists

Redis Hash command

HDEL

```
HDEL key field [field ...]
```

Example

```
redis> HSET myhash field1 "foo"
(integer) 1
redis> HDEL myhash field1
(integer) 1
redis> HDEL myhash field2
(integer) 0
```

Delete one or more hash fields

HEXISTS

```
HEXISTS key field
```

Example

```
redis> HSET myhash field1 "foo"
(integer) 1
redis> HEXISTS myhash field1
(integer) 1
redis> HEXISTS myhash field2
(integer) 0
```

Determine if a hash field exists

HGET

```
HGET key field
```

Example

```
redis> HSET myhash field1 "foo"
(integer) 1
redis> HGET myhash field1
"foo"
redis> HGET myhash field2
(nil)
```

Get the value of a hash field

HGETALL

```
HGETALL key
```

Example

```
redis> HSET myhash field1 "Hello"
(integer) 1
redis> HSET myhash field2 "World"
(integer) 1
redis> HGETALL myhash
1) "field1"
2) "Hello"
3) "field2"
```

HINCRBY

```
HINCRBY key field increment
```

Example

```
redis> HSET myhash field 5
(integer) 1
redis> HINCRBY myhash field 1
(integer) 6
redis> HINCRBY myhash field -1
(integer) 5
redis> HINCRBY myhash field -10
(integer) -5
```

HINCRBYFLOAT

```
HINCRBYFLOAT key field increment
```

Example

```
redis> HSET mykey field 10.50
(integer) 1
redis> HINCRBYFLOAT mykey field 0.1
"10.6"
redis> HINCRBYFLOAT mykey field -5
"5.6"
redis> HSET mykey field 5.0e3
(integer) 0
```

```
4) "World"
```

Get all the fields and values in a hash

Increment the integer value of a hash field by the given number

```
redis> HINCRBYFLOAT mykey field 2.0e2
"5200"
```

Increment the float value of a hash field by the given amount

HKEYS

```
HKEYS key
```

Example

```
redis> HSET myhash field1 "Hello"
(integer) 1
redis> HSET myhash field2 "World"
(integer) 1
redis> HKEYS myhash
1) "field1"
2) "field2"
```

Get all the fields in a hash

HLEN

```
HLEN key
```

Example

```
redis> HSET myhash field1 "Hello"
(integer) 1
redis> HSET myhash field2 "World"
(integer) 1
redis> HLEN myhash
(integer) 2
```

Get the number of fields in a hash

HMGET

```
HMGET key field [field ...]
```

Example

```
redis> HSET myhash field1 "Hello"
(integer) 1
redis> HSET myhash field2 "World"
(integer) 1
redis> HMGET myhash field1 field2 nofield
1) "Hello"
2) "World"
3) (nil)
```

Get the values of all the given hash fields

HMSET

```
HMSET key field value [field value ...]
```

Example

```
redis> HMSET myhash field1 "Hello" field2
"OK"
redis> HGET myhash field1
"Hello"
redis> HGET myhash field2
"World"
```

Set multiple hash fields to multiple values

HSET

```
HSET key field value [field value ...]
```

Example

```
redis> HSET myhash field1 "Hello"
(integer) 1
redis> HGET myhash field1
"Hello"
```

Set the string value of a hash field

HSETNX

```
HSETNX key field value
```

Example

```
redis> HSETNX myhash field "Hello"
(integer) 1
redis> HSETNX myhash field "World"
(integer) 0
redis> HGET myhash field
"Hello"
```

Set the value of a hash field, only if the field does not exist

HSTRLEN

```
HSTRLEN key field
```

Example

```
redis> HMSET myhash f1 HelloWorld f2 99 f3
"OK"
redis> HSTRLEN myhash f1
(integer) 10
redis> HSTRLEN myhash f2
(integer) 2
redis> HSTRLEN myhash f3
(integer) 4
```

Get the length of the value of a hash field

HVALS

```
HVALS key
```

Example

```
redis> HSET myhash field1 "Hello"
(integer) 1
redis> HSET myhash field2 "World"
(integer) 1
redis> HVALS myhash
1) "Hello"
2) "World"
```

Get all the values in a hash

Redis Sorted set command

BZPOPMIN

```
BZPOPMIN key [key ...] timeout
```

Example

```
redis> DEL zset1 zset2
(integer) 0
redis> ZADD zset1 0 a 1 b 2 c
(integer) 3
redis> BZPOPMIN zset1 zset2 0
1) "zset1"
2) "a"
3) "0"
```

Remove and return the member with the lowest score from one or more sorted sets, or block until one is available

BZPOPMAX

```
BZPOPMAX key [key ...] timeout
```

Example

```
redis> DEL zset1 zset2
(integer) 0
redis> ZADD zset1 0 a 1 b 2 c
(integer) 3
redis> BZPOPMAX zset1 zset2 0
1) "zset1"
2) "c"
3) "2"
```

Remove and return the member with the highest score from one or more sorted sets, or block until one is available

ZADD

```
ZADD key [NX|XX] [GT|LT] [CH] [INCR]
score member [score member ...]
```

Example

```
redis> ZADD myzset 1 "one"
(integer) 1
redis> ZADD myzset 1 "uno"
(integer) 1
redis> ZADD myzset 2 "two" 3 "three"
(integer) 2
redis> ZRANGE myzset 0 -1 WITHSCORES
1) "one"
2) "1"
3) "uno"
4) "1"
5) "two"
6) "2"
7) "three"
8) "3"
```

ZCARD

```
ZCARD key
```

Example

```
redis> ZADD myzset 1 "one"
```

ZSCORE

```
ZSCORE key member
```

Example

```
redis> ZADD myzset 1 "one"
```

```
(integer) 1
redis> ZADD myzset 2 "two"
(integer) 1
redis> ZCARD myzset
(integer) 2
```

Get the number of members in a sorted set

```
(integer) 1
redis> ZSCORE myzset "one"
"1"
```

Get the score associated with the given member in a sorted set

Add one or more members to a sorted set, or update its score if it already exists

ZCOUNT

```
ZCOUNT key min max
```

Example

```
redis> ZADD myzset 1 "one"
(integer) 1
redis> ZADD myzset 2 "two"
(integer) 1
redis> ZADD myzset 3 "three"
(integer) 1
redis> ZCOUNT myzset -inf +inf
(integer) 3
redis> ZCOUNT myzset (1 3
(integer) 2
```

Count the members in a sorted set with scores within the given values

ZDIFF

```
ZDIFF numkeys key [key ...] [WITHSCORES]
```

Example

```
redis> ZADD zset1 1 "one"
(integer) 1
redis> ZADD zset1 2 "two"
(integer) 1
redis> ZADD zset1 3 "three"
(integer) 1
redis> ZADD zset2 1 "one"
(integer) 1
redis> ZADD zset2 2 "two"
(integer) 1
redis> ZDIFF 2 zset1 zset2
1) "three"
redis> ZDIFF 2 zset1 zset2 WITHSCORES
1) "three"
2) "3"
```

Subtract multiple sorted sets

ZDIFFSTORE

```
ZDIFFSTORE destination numkeys key [key ...]
```

Example

```
redis> ZADD zset1 1 "one"
(integer) 1
redis> ZADD zset1 2 "two"
(integer) 1
redis> ZADD zset1 3 "three"
(integer) 1
redis> ZADD zset2 1 "one"
(integer) 1
redis> ZADD zset2 2 "two"
(integer) 1
redis> ZDIFFSTORE out 2 zset1 zset2
(integer) 1
redis> ZRANGE out 0 -1 WITHSCORES
1) "three"
2) "3"
```

Subtract multiple sorted sets and store the resulting sorted set in a new key

ZINCRBY

```
ZINCRBY key increment member
```

Example

```
redis> ZADD myzset 1 "one"
(integer) 1
redis> ZADD myzset 2 "two"
(integer) 1
redis> ZINCRBY myzset 2 "one"
"3"
redis> ZRANGE myzset 0 -1 WITHSCORES
1) "two"
2) "2"
3) "one"
4) "3"
```

Increment the score of a member in a sorted set

ZINTER

```
ZINTER numkeys key [key ...] [WEIGHTS weight [weight ...]] [AGGREGATE SUM|MIN|MAX] [WITHSCORES]
```

Example

```
redis> ZADD zset1 1 "one"
(integer) 1
redis> ZADD zset1 2 "two"
(integer) 1
redis> ZADD zset2 1 "one"
(integer) 1
redis> ZADD zset2 2 "two"
(integer) 1
redis> ZADD zset2 3 "three"
(integer) 1
redis> ZINTER 2 zset1 zset2
1) "one"
2) "two"
redis> ZINTER 2 zset1 zset2 WITHSCORES
1) "one"
2) "2"
3) "two"
4) "4"
```

Intersect multiple sorted sets

ZINTERSTORE

```
ZINTERSTORE destination numkeys key [key ...] [WEIGHTS weight [weight ...]] [AGGREGATE SUM|MIN|MAX]
```

Example

```
redis> ZADD zset1 1 "one"
(integer) 1
redis> ZADD zset1 2 "two"
(integer) 1
redis> ZADD zset2 1 "one"
(integer) 1
redis> ZADD zset2 2 "two"
(integer) 1
redis> ZADD zset2 3 "three"
(integer) 1
redis> ZINTERSTORE out 2 zset1 zset2 WEIGH
(integer) 2
redis> ZRANGE out 0 -1 WITHSCORES
1) "one"
2) "5"
3) "two"
4) "10"
```

Intersect multiple sorted sets and store the resulting sorted set in a new key

ZLEXCOUNT

```
ZLEXCOUNT key min max
```

Example

```
redis> ZADD myzset 0 a 0 b 0 c 0 d 0 e
(integer) 5
redis> ZADD myzset 0 f 0 g
(integer) 2
redis> ZLEXCOUNT myzset - +
(integer) 7
redis> ZLEXCOUNT myzset [b [f
(integer) 5
```

Count the number of members in a sorted set between a given lexicographical range

ZPOPMAX

```
ZPOPMAX key [count]
```

Example

```
redis> ZADD myzset 1 "one"
(integer) 1
redis> ZADD myzset 2 "two"
(integer) 1
redis> ZADD myzset 3 "three"
(integer) 1
redis> ZPOPMAX myzset
1) "three"
2) "3"
```

Remove and return members with the highest scores in a sorted set

ZPOPMIN

```
ZPOPMIN key [count]
```

Example

```
redis> ZADD myzset 1 "one"
(integer) 1
redis> ZADD myzset 2 "two"
(integer) 1
redis> ZADD myzset 3 "three"
(integer) 1
redis> ZPOPMIN myzset
1) "one"
2) "1"
```

Remove and return members with the lowest scores in a sorted set

ZRANGE

```
ZRANGE key start stop [WITHSCORES]
```

Example

ZRANGEBYLEX

```
ZRANGEBYLEX key min max [LIMIT offset count]
```

ZREVRANGEBYLEX

```
ZREVRANGEBYLEX key max min [LIMIT offset count]
```

```
redis> ZADD myzset 1 "one"
(integer) 1
redis> ZADD myzset 2 "two"
(integer) 1
redis> ZADD myzset 3 "three"
(integer) 1
redis> ZRANGE myzset 0 -1
1) "one"
2) "two"
3) "three"
redis> ZRANGE myzset 2 3
1) "three"
redis> ZRANGE myzset -2 -1
1) "two"
2) "three"
```

Return a range of members in a sorted set, by index

Example

```
redis> ZADD myzset 0 a 0 b 0 c 0 d 0 e 0 f
(integer) 7
redis> ZRANGEBYLEX myzset - [c
1) "a"
2) "b"
3) "c"
redis> ZRANGEBYLEX myzset - (c
1) "a"
2) "b"
redis> ZRANGEBYLEX myzset [aaa (g
1) "b"
2) "c"
3) "d"
4) "e"
5) "f"
```

Return a range of members in a sorted set, by lexicographical range

Example

```
redis> ZADD myzset 0 a 0 b 0 c 0 d 0 e 0 f
(integer) 7
redis> ZREVRANGEBYLEX myzset [c -
1) "c"
2) "b"
3) "a"
redis> ZREVRANGEBYLEX myzset (c -
1) "b"
2) "a"
redis> ZREVRANGEBYLEX myzset (g [aaa
1) "f"
2) "e"
3) "d"
4) "c"
5) "b"
```

Return a range of members in a sorted set, by lexicographical range, ordered from higher to lower strings.

ZRANGEBYSCORE

```
ZRANGEBYSCORE key min max [WITHSCORES]
[LIMIT offset count]
```

Example

```
redis> ZADD myzset 1 "one"
(integer) 1
redis> ZADD myzset 2 "two"
(integer) 1
redis> ZADD myzset 3 "three"
(integer) 1
redis> ZRANGEBYSCORE myzset -inf +inf
1) "one"
2) "two"
3) "three"
redis> ZRANGEBYSCORE myzset 1 2
1) "one"
2) "two"
redis> ZRANGEBYSCORE myzset (1 2
1) "two"
redis> ZRANGEBYSCORE myzset (1 (2
(empty list or set)
```

Return a range of members in a sorted set, by score

ZRANK

```
ZRANK key member
```

Example

```
redis> ZADD myzset 1 "one"
(integer) 1
redis> ZADD myzset 2 "two"
(integer) 1
redis> ZADD myzset 3 "three"
(integer) 1
redis> ZRANK myzset "three"
(integer) 2
redis> ZRANK myzset "four"
(nil)
```

Determine the index of a member in a sorted set

ZREM

```
ZREM key member [member ...]
```

Example

```
redis> ZADD myzset 1 "one"
(integer) 1
redis> ZADD myzset 2 "two"
(integer) 1
redis> ZADD myzset 3 "three"
(integer) 1
redis> ZREM myzset "two"
(integer) 1
redis> ZRANGE myzset 0 -1 WITHSCORES
1) "one"
2) "1"
3) "three"
4) "3"
```

Remove one or more members from a sorted set

ZREMRANGEBYLEX

```
ZREMRANGEBYLEX key min max
```

Example

```
redis> ZADD myzset 0 aaaa 0 b 0 c 0 d 0 e
(integer) 5
redis> ZADD myzset 0 foo 0 zap 0 zip 0 ALF
(integer) 5
redis> ZRANGE myzset 0 -1
1) "ALPHA"
 2) "aaaa"
 3) "alpha"
 4) "b"
 5) "c"
 6) "d"
 7) "e"
 8) "foo"
 9) "zap"
10) "zip"
redis> ZREMRANGEBYLEX myzset [alpha [omega
(integer) 6
redis> ZRANGE myzset 0 -1
1) "ALPHA"
2) "aaaa"
3) "zap"
4) "zip"
```

Remove all members in a sorted set between the given lexicographical range

ZREMRANGEBYRANK

```
ZREMRANGEBYRANK key start stop
```

Example

```
redis> ZADD myzset 1 "one"
(integer) 1
redis> ZADD myzset 2 "two"
(integer) 1
redis> ZADD myzset 3 "three"
(integer) 1
redis> ZREMRANGEBYRANK myzset 0 1
(integer) 2
redis> ZRANGE myzset 0 -1 WITHSCORES
1) "three"
2) "3"
```

Remove all members in a sorted set within the given indexes

ZREMRANGEBYSCORE

```
ZREMRANGEBYSCORE key min max
```

Example

```
redis> ZADD myzset 1 "one"
(integer) 1
redis> ZADD myzset 2 "two"
(integer) 1
redis> ZADD myzset 3 "three"
(integer) 1
redis> ZREMRANGEBYSCORE myzset -inf (2
(integer) 1
redis> ZRANGE myzset 0 -1 WITHSCORES
1) "two"
2) "2"
3) "three"
4) "3"
```

Remove all members in a sorted set within the given scores

ZREVRANGE

```
ZREVRANGE key start stop [WITHSCORES]
```

Example

```
redis> ZADD myzset 1 "one"
```

ZREVRANGEBYSCORE

```
ZREVRANGEBYSCORE key max min [WITHSCORES]
[LIMIT offset count]
```

Example

ZREVRANK

```
ZREVRANK key member
```

Example

```
redis> ZADD myzset 1 "one"
```

```
(integer) 1
redis> ZADD myzset 2 "two"
(integer) 1
redis> ZADD myzset 3 "three"
(integer) 1
redis> ZREVRANGE myzset 0 -1
1) "three"
2) "two"
3) "one"
redis> ZREVRANGE myzset 2 3
1) "one"
redis> ZREVRANGE myzset -2 -1
1) "two"
2) "one"
```

Return a range of members in a sorted set, by index, with scores ordered from high to low

```
redis> ZADD myzset 1 "one"
(integer) 1
redis> ZADD myzset 2 "two"
(integer) 1
redis> ZADD myzset 3 "three"
(integer) 1
redis> ZREVRANGEBYSCORE myzset +inf -inf
1) "three"
2) "two"
3) "one"
redis> ZREVRANGEBYSCORE myzset 2 1
1) "two"
2) "one"
redis> ZREVRANGEBYSCORE myzset 2 (1
1) "two"
redis> ZREVRANGEBYSCORE myzset (2 (1
(empty list or set)
```

Return a range of members in a sorted set, by score, with scores ordered from high to low

```
(integer) 1
redis> ZADD myzset 2 "two"
(integer) 1
redis> ZADD myzset 3 "three"
(integer) 1
redis> ZREVRANK myzset "one"
(integer) 2
redis> ZREVRANK myzset "four"
(nil)
```

Determine the index of a member in a sorted set, with scores ordered from high to low

ZUNION

```
ZUNION numkeys key [key ...] [WEIGHTS
weight [weight ...]] [AGGREGATE
SUM|MIN|MAX] [WITHSCORES]
```

Example

```
redis> ZADD zset1 1 "one"
(integer) 1
redis> ZADD zset1 2 "two"
(integer) 1
redis> ZADD zset2 1 "one"
(integer) 1
redis> ZADD zset2 2 "two"
(integer) 1
redis> ZADD zset2 3 "three"
(integer) 1
redis> ZUNION 2 zset1 zset2
1) "one"
2) "three"
3) "two"
redis> ZUNION 2 zset1 zset2 WITHSCORES
1) "one"
2) "2"
3) "three"
4) "3"
5) "two"
6) "4"
```

Add multiple sorted sets

ZMSCORE

```
ZMSCORE key member [member ...]
```

Example

```
redis> ZADD myzset 1 "one"
(integer) 1
redis> ZADD myzset 2 "two"
(integer) 1
redis> ZMSCORE myzset "one" "two" "nofielc
1) "1"
2) "2"
3) (nil)
```

Get the score associated with the given members in a sorted set

ZUNIONSTORE

```
ZUNIONSTORE destination numkeys key [key
...] [WEIGHTS weight [weight ...]]
[AGGREGATE SUM|MIN|MAX]
```

Example

```
redis> ZADD zset1 1 "one"
(integer) 1
redis> ZADD zset1 2 "two"
(integer) 1
redis> ZADD zset2 1 "one"
(integer) 1
redis> ZADD zset2 2 "two"
(integer) 1
redis> ZADD zset2 3 "three"
(integer) 1
redis> ZUNIONSTORE out 2 zset1 zset2 WEIGH
(integer) 3
redis> ZRANGE out 0 -1 WITHSCORES
1) "one"
2) "5"
3) "three"
4) "9"
5) "two"
6) "10"
```

Add multiple sorted sets and store the resulting sorted set in a new key

Redis Geo command

GEOADD

```
GEOADD key longitude latitude member [longitude latitude member
...]
```

Example

```
redis> GEOADD Sicily 13.361389 38.115556 "Palermo" 15.087269 37.50
(integer) 2
redis> GEODIST Sicily Palermo Catania
"166274.1516"
redis> GEORADIUS Sicily 15 37 100 km
1) "Catania"
redis> GEORADIUS Sicily 15 37 200 km
1) "Palermo"
2) "Catania"
```

Add one or more geospatial items in the geospatial index represented using a sorted set

GEOHASH

```
GEOHASH key member [member ...]
```

Example

```
redis> GEOADD Sicily 13.361389 38.115556 "Palermo" 15.087269 37.50
(integer) 2
redis> GEOHASH Sicily Palermo Catania
1) "sqc8b49rny0"
2) "sqdtr74hyu0"
```

Returns members of a geospatial index as standard geohash strings

GEOPOS

```
GEOPOS key member [member ...]
```

Example

```
redis> GEOADD Sicily 13.361389 38.115556 "Palermo" 15.087269 37.50
(integer) 2
redis> GEOPOS Sicily Palermo Catania NonExisting
1) 1) "13.3613893897018433"
   2) "38.11555639549629859"
```

GEODIST

```
GEODIST key member1 member2 [m|km|ft|mi]
```

Example

```
redis> GEOADD Sicily 13.361389 38.115556 "Palermo" 15.087269 37.50
(integer) 2
redis> GEODIST Sicily Palermo Catania
"166274.1516"
redis> GEODIST Sicily Palermo Catania km
```

```
2) 1) "15.08726745843887329"
   2) "37.50266842333162032"
3) (nil)
```

Returns longitude and latitude of members of a geospatial index

```
"166.2742"
redis> GEODIST Sicily Palermo Catania mi
"103.3182"
redis> GEODIST Sicily Foo Bar
(nil)
```

Returns the distance between two members of a geospatial index

GEORADIUS

```
GEORADIUS key longitude latitude radius m|km|ft|mi [WITHCOORD]
[WITHDIST] [WITHHASH] [COUNT count] [ASC|DESC] [STORE key]
[STOREDIST key]
```

Example

```
redis> GEOADD Sicily 13.361389 38.115556 "Palermo" 15.087269 37.50
(integer) 2
redis> GEORADIUS Sicily 15 37 200 km WITHDIST
1) 1) "Palermo"
   2) "190.4424"
2) 1) "Catania"
   2) "56.4413"
redis> GEORADIUS Sicily 15 37 200 km WITHCOORD
1) 1) "Palermo"
   2) 1) "13.36138933897018433"
      2) "38.11555639549629859"
2) 1) "Catania"
   2) 1) "15.08726745843887329"
      2) "37.50266842333162032"
redis> GEORADIUS Sicily 15 37 200 km WITHDIST WITHCOORD
1) 1) "Palermo"
   2) "190.4424"
   3) 1) "13.36138933897018433"
      2) "38.11555639549629859"
2) 1) "Catania"
   2) "56.4413"
   3) 1) "15.08726745843887329"
      2) "37.50266842333162032"
```

Query a sorted set representing a geospatial index to fetch members matching a given maximum distance from a point

GEORADIUSBYMEMBER

```
GEORADIUSBYMEMBER key member radius m|km|ft|mi [WITHCOORD]
[WITHDIST] [WITHHASH] [COUNT count] [ASC|DESC] [STORE key]
[STOREDIST key]
```

Example

```
redis> GEOADD Sicily 13.583333 37.316667 "Agrigento"
(integer) 1
redis> GEOADD Sicily 13.361389 38.115556 "Palermo" 15.087269 37.50
(integer) 2
redis> GEORADIUSBYMEMBER Sicily Agrigento 100 km
1) "Agrigento"
2) "Palermo"
```

Query a sorted set representing a geospatial index to fetch members matching a given maximum distance from a member

GEOSEARCH

```
GEOSEARCH key [FROMMEMBER member] [FROMLONLAT longitude latitude]
[BYRADIUS radius m|km|ft|mi] [BYBOX width height m|km|ft|mi]
[ASC|DESC] [COUNT count] [WITHCOORD] [WITHDIST] [WITHHASH]
```

Example

```
redis> GEOADD Sicily 13.361389 38.115556 "Palermo" 15.087269 37.50
(integer) 2
redis> GEOADD Sicily 12.758489 38.788135 "edge1"   17.241510 38.78
(integer) 2
redis> GEOSEARCH Sicily FROMLONLAT 15 37 BYRADIUS 200 km ASC
1) "Catania"
2) "Palermo"
redis> GEOSEARCH Sicily FROMLONLAT 15 37 BYBOX 400 400 km ASC
1) "Catania"
2) "Palermo"
3) "edge2"
4) "edge1"
```

Query a sorted set representing a geospatial index to fetch members inside an area of a box or a circle.

Misc

GEOSEARCHSTORE — Query a sorted set representing a geospatial index to fetch members inside an area of a box or a circle, and store the result in another key.

Redis Hyperloglog command

PFADD

```
PFADD key element [element ...]
```

Example

```
redis> PFADD hll a b c d e f g
(integer) 1
redis> PFCOUNT hll
(integer) 7
```

PFCOUNT

```
PFCOUNT key [key ...]
```

Example

```
redis> PFADD hll foo bar zap
(integer) 1
redis> PFADD hll zap zap zap
(integer) 0
redis> PFADD hll foo bar
(integer) 0
redis> PFCOUNT hll
(integer) 3
redis> PFADD some-other-hll 1 2 3
(integer) 1
redis> PFCOUNT hll some-other-hll
```

PFMERGE

```
PFMERGE destkey sourcekey [sourcekey ...]
```

Example

```
redis> PFADD hll1 foo bar zap a
(integer) 1
redis> PFADD hll2 a b c foo
(integer) 1
redis> PFMERGE hll3 hll1 hll2
"OK"
redis> PFCOUNT hll3
(integer) 6
```

Adds the specified elements to the specified HyperLogLog.

```
(integer) 6
```

Return the approximated cardinality of the set(s) observed by the HyperLogLog at key(s).

Merge N different HyperLogLogs into a single one.

Redis Server command

COMMAND

```
COMMAND
```

Example

```
redis> COMMAND
1) 1) "georadius_ro"
     2) (integer) -6
     3) 1) "readonly"
        2) "movablekeys"
     4) (integer) 1
     5) (integer) 1
     6) (integer) 1
     7) 1) "@read"
        2) "@geo"
        3) "@slow"
  2) 1) "zpopmin"
     2) (integer) -2
     3) 1) "write"
        2) "fast"
  ........
```

Get array of Redis command details

COMMAND COUNT

```
COMMAND COUNT
```

Example

```
redis> COMMAND COUNT
(integer) 217
```

Get total number of Redis commands

COMMAND GETKEYS

```
COMMAND GETKEYS
```

Example

```
redis> COMMAND GETKEYS MSET a b c d e f
1) "a"
2) "c"
3) "e"
redis> COMMAND GETKEYS EVAL "not consultec
1) "key1"
2) "key2"
3) "key3"
redis> COMMAND GETKEYS SORT mylist ALPHA S
1) "mylist"
2) "outlist"
```

Extract keys given a full Redis command

COMMAND INFO

```
COMMAND INFO command-name [command-name
...]
```

Example

```
redis> COMMAND INFO get set eval
1) 1) "get"
   2) (integer) 2
   3) 1) "readonly"
      2) "fast"
   4) (integer) 1
   5) (integer) 1
   6) (integer) 1
   7) 1) "@read"
      2) "@string"
      3) "@fast"
2) 1) "set"
   2) (integer) -3
   3) 1) "write"
      2) "denyoom"
```

Misc

ACL LOAD	Reload the ACLs from the configured ACL file
ACL SAVE	Save the current ACL rules in the configured ACL file
ACL LIST	List the current ACL rules in ACL config file format
ACL USERS	List the username of all the configured ACL rules
ACL GETUSER	Get the rules for a specific ACL user
ACL SETUSER	Modify or create the rules for a specific ACL user
ACL DELUSER	Remove the specified ACL users and the associated rules
ACL CAT	List the ACL categories or the commands inside a category
ACL GENPASS	Generate a pseudorandom secure password to use for ACL users
ACL WHOAMI	Return the name of the user associated to the current connection
ACL LOG	List latest events denied because of ACLs in place
ACL HELP	Show helpful text about the different subcommands
BGREWRITEAOF	Asynchronously rewrite the append-only file
BGSAVE	Asynchronously save the dataset to disk
CONFIG GET	Get the value of a configuration parameter
CONFIG REWRITE	Rewrite the configuration file with the in memory configuration
CONFIG SET	Set a configuration parameter to the given value
CONFIG RESETSTAT	Reset the stats returned by INFO
DBSIZE	Return the number of keys in the selected database
DEBUG OBJECT	Get debugging information about a key
DEBUG SEGFAULT	Make the server crash
FLUSHALL	Remove all keys from all databases
FLUSHDB	Remove all keys from the current database
LOLWUT	Display some computer art and the Redis version
LASTSAVE	Get the UNIX time stamp of the last successful save to disk
MEMORY DOCTOR	Outputs memory problems report
MEMORY HELP	Show helpful text about the different subcommands
MEMORY MALLOC-STATS	Show allocator internal stats
MEMORY PURGE	Ask the allocator to release memory
MEMORY STATS	Show memory usage details
MEMORY USAGE	Estimate the memory usage of a key
MODULE LIST	List all modules loaded by the server
MODULE LOAD	Load a module
MODULE UNLOAD	Unload a module
MONITOR	Listen for all requests received by the server in real time
SAVE	Synchronously save the dataset to disk
SHUTDOWN	Synchronously save the dataset to disk and then shut down the server
SLAVEOF	Make the server a replica of another instance, or promote it as master. Deprecated starting with Redis 5. Use REPLICAOF instead.
REPLICAOF	Make the server a replica of another instance, or promote it as master.
SLOWLOG	Manages the Redis slow queries log
SWAPDB	Swaps two Redis databases
SYNC	Internal command used for replication
PSYNC	Internal command used for replication

```
   4) (integer) 1
   5) (integer) 1
   6) (integer) 1
   7) 1) "@write"
      2) "@string"
      3) "@slow"
3) 1) "eval"
   2) (integer) -3
   3) 1) "noscript"
      2) "movablekeys"
   4) (integer) 0
   5) (integer) 0
   6) (integer) 0
   7) 1) "@slow"
      2) "@scripting"
```

Get array of specific Redis command details

LATENCY DOCTOR	Return a human readable latency analysis report.
LATENCY GRAPH	Return a latency graph for the event.
LATENCY HISTORY	Return timestamp-latency samples for the event.
LATENCY LATEST	Return the latest latency samples for all events.
LATENCY RESET	Reset latency data for one or more events.
LATENCY HELP	Show helpful text about the different subcommands.

INFO

```
INFO [section]
```

Example

```
redis> INFO
# Server
redis_version:6.1.240
redis_git_sha1:00000000
redis_git_dirty:0
redis_build_id:a26db646ea64a07c
redis_mode:standalone
os:Linux 5.4.0-1017-aws x86_64
......
```

Get information and statistics about the server

ROLE

```
ROLE
```

Example

```
redis> ROLE
1) "master"
2) (integer) 0
3) (empty list or set)
```

Return the role of the instance in the context of replication

TIME

```
TIME
```

Example

```
redis> TIME
1) "1609040690"
2) "558952"
redis> TIME
1) "1609040690"
2) "559206"
```

Return the current server time

Redis Generic command

Misc

COPY	Copy a key
MIGRATE	Atomically transfer a key from a Redis instance to another one.
MOVE	Move a key to another database
OBJECT	Inspect the internals of Redis objects
RESTORE	Create a key using the provided serialized value, previously obtained using DUMP.
SORT	Sort the elements in a list, set or sorted set
WAIT	Wait for the synchronous replication of all the write commands sent in the context of the current connection
SCAN	Incrementally iterate the keys space

DEL

```
DEL key [key ...]
```

Example

```
redis> SET key1 "Hello"
"OK"
redis> SET key2 "World"
"OK"
redis> DEL key1 key2 key3
(integer) 2
```

Delete a key

DUMP

```
DUMP key
```

Example

```
redis> SET mykey 10
"OK"
redis> DUMP mykey
"\u0000\xC0\n\t\u0000\xBEm\u0006\x89Z(\u00
```

Return a serialized version of the value stored at the specified key.

EXISTS

```
EXISTS key [key ...]
```

Example

```
redis> SET key1 "Hello"
"OK"
redis> EXISTS key1
(integer) 1
redis> EXISTS nosuchkey
(integer) 0
redis> SET key2 "World"
"OK"
redis> EXISTS key1 key2 nosuchkey
(integer) 2
```

Determine if a key exists

EXPIRE

```
EXPIRE key seconds
```

Example

```
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
```

Set a key's time to live in seconds

EXPIREAT

```
EXPIREAT key timestamp
```

Example

```
redis> SET mykey "Hello"
"OK"
redis> EXISTS mykey
(integer) 1
redis> EXPIREAT mykey 1293840000
(integer) 1
redis> EXISTS mykey
(integer) 0
```

Set the expiration for a key as a UNIX timestamp

KEYS

```
KEYS pattern
```

Example

```
redis> MSET firstname Jack lastname Stuntm
```

PERSIST

```
PERSIST key
```

Example

```
redis> SET mykey "Hello"
```

PEXPIRE

```
PEXPIRE key milliseconds
```

Example

```
redis> SET mykey "Hello"
```

```
"OK"
redis> KEYS *name*
1) "firstname"
2) "lastname"
redis> KEYS a??
1) "age"
redis> KEYS *
1) "firstname"
2) "age"
3) "lastname"
```

Find all keys matching the given pattern

```
"OK"
redis> EXPIRE mykey 10
(integer) 1
redis> TTL mykey
(integer) 10
redis> PERSIST mykey
(integer) 1
redis> TTL mykey
(integer) -1
```

Remove the expiration from a key

```
"OK"
redis> PEXPIRE mykey 1500
(integer) 1
redis> TTL mykey
(integer) 1
redis> PTTL mykey
(integer) 1499
```

Set a key's time to live in milliseconds

PEXPIREAT

```
PEXPIREAT key milliseconds-timestamp
```

Example

```
redis> SET mykey "Hello"
"OK"
redis> PEXPIREAT mykey 1555555555005
(integer) 1
redis> TTL mykey
(integer) -2
redis> PTTL mykey
(integer) -2
```

Set the expiration for a key as a UNIX timestamp specified in milliseconds

PTTL

```
PTTL key
```

Example

```
redis> SET mykey "Hello"
"OK"
redis> EXPIRE mykey 1
(integer) 1
redis> PTTL mykey
(integer) 1000
```

Get the time to live for a key in milliseconds

RENAME

```
RENAME key newkey
```

Example

```
redis> SET mykey "Hello"
"OK"
redis> RENAME mykey myotherkey
"OK"
redis> GET myotherkey
"Hello"
```

Rename a key

RENAMENX

```
RENAMENX key newkey
```

Example

```
redis> SET mykey "Hello"
"OK"
redis> SET myotherkey "World"
"OK"
redis> RENAMENX mykey myotherkey
(integer) 0
redis> GET myotherkey
"World"
```

Rename a key, only if the new key does not exist

TOUCH

```
TOUCH key [key ...]
```

Example

```
redis> SET key1 "Hello"
"OK"
redis> SET key2 "World"
"OK"
redis> TOUCH key1 key2
(integer) 2
```

Alters the last access time of a key(s). Returns the number of existing keys specified.

TTL

```
TTL key
```

Example

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

Get the time to live for a key

TYPE

```
TYPE key
```

Example

```
redis> SET key1 "value"
"OK"
redis> LPUSH key2 "value"
(integer) 1
redis> SADD key3 "value"
(integer) 1
redis> TYPE key1
"string"
redis> TYPE key2
"list"
redis> TYPE key3
"set"
```

Determine the type stored at key

UNLINK

```
UNLINK key [key ...]
```

Example

```
redis> SET key1 "Hello"
"OK"
redis> SET key2 "World"
"OK"
redis> UNLINK key1 key2 key3
(integer) 2
```

Delete a key asynchronously in another thread. Otherwise it is just as DEL, but non blocking.

Redis Connection command

Misc

AUTH	Authenticate to the server
CLIENT CACHING	Instruct the server about tracking or not keys in the next request
CLIENT KILL	Kill the connection of a client
CLIENT LIST	Get the list of client connections
CLIENT GETNAME	Get the current connection name
CLIENT GETREDIR	Get tracking notifications redirection client ID if any

CLIENT ID

```
CLIENT ID
```

Example

```
redis> CLIENT ID
ERR Unknown or disabled command 'CLIENT'
```

CLIENT INFO

```
CLIENT INFO
```

Example

```
redis> CLIENT INFO
"id=55542 addr=127.0.0.1:58710 laddr=127.0
```

CLIENT PAUSE	Stop processing commands from clients for some time
CLIENT REPLY	Instruct the server whether to reply to commands
CLIENT SETNAME	Set the current connection name
CLIENT TRACKING	Enable or disable server assisted client side caching support
CLIENT UNBLOCK	Unblock a client blocked in a blocking command from a different connection
HELLO	switch Redis protocol
QUIT	Close the connection
RESET	Reset the connection
SELECT	Change the selected database for the current connection

Returns the client ID for the current connection

Returns information about the current client connection.

ECHO

```
ECHO message
```

Example

```
redis> ECHO "Hello World!"
"Hello World!"
```

Echo the given string

PING

```
PING [message]
```

Example

```
redis> PING
"PONG"
redis> PING "hello world"
"hello world"
```

Ping the server

Redis Stream command

Misc

XINFO	Get information on streams and consumer groups
XDEL	Removes the specified entries from the stream. Returns the number of items actually deleted, that may be different from the number of IDs passed in case certain IDs do not exist.
XREAD	Return never seen elements in multiple streams, with IDs greater than the ones reported by the caller for each stream. Can block.
XGROUP	Create, destroy, and manage consumer groups.
XREADGROUP	Return new entries from a stream using a consumer group, or access the history of the pending entries for a given consumer. Can block.
XCLAIM	Changes (or acquires) ownership of a message in a consumer group, as if the message was delivered to the specified consumer.
XPENDING	Return information and entries from a stream consumer group pending entries list, that are messages fetched but never acknowledged.

XADD

```
XADD key [MAXLEN [=|~] length] [NOMKSTREAM] *|ID field value
[field value ...]
```

Example

```
redis> XADD mystream * name Sara surname OConnor
"1609040574632-0"
redis> XADD mystream * field1 value1 field2 value2 field3 value3
"1609040574632-1"
redis> XLEN mystream
(integer) 2
redis> XRANGE mystream - +
1) 1) "1609040574632-0"
   2) 1) "name"
      2) "Sara"
      3) "surname"
      4) "OConnor"
2) 1) "1609040574632-1"
   2) 1) "field1"
      2) "value1"
      3) "field2"
      4) "value2"
      5) "field3"
      6) "value3"
```

Appends a new entry to a stream

XTRIM

```
XTRIM key MAXLEN [=|~] length
```

Example

```
redis> XADD mystream * field1 A field2 B field3 C field4 D
"1609040575750-0"
redis> XTRIM mystream MAXLEN 2
(integer) 0
redis> XRANGE mystream - +
1) 1) "1609040575750-0"
   2) 1) "field1"
      2) "A"
      3) "field2"
      4) "B"
      5) "field3"
      6) "C"
      7) "field4"
```

XRANGE

```
XRANGE key start end [COUNT count]
```

Example

```
redis> XADD writers * name Virginia surname Woolf
"1609040578002-0"
redis> XADD writers * name Jane surname Austen
"1609040578002-1"
redis> XADD writers * name Toni surname Morrison
"1609040578003-0"
redis> XADD writers * name Agatha surname Christie
"1609040578003-1"
redis> XADD writers * name Ngozi surname Adichie
"1609040578003-2"
redis> XLEN writers
(integer) 5
redis> XRANGE writers - + COUNT 2
```

```
      8) "D"
```

Trims the stream to (approximately if '~' is passed) a certain size

```
1) 1) "1609040578002-0"
   2) 1) "name"
      2) "Virginia"
      3) "surname"
      4) "Woolf"
2) 1) "1609040578002-1"
   2) 1) "name"
      2) "Jane"
      3) "surname"
      4) "Austen"
```

Return a range of elements in a stream, with IDs matching the specified IDs interval

XREVRANGE

```
XREVRANGE key end start [COUNT count]
```

Example

```
redis> XADD writers * name Virginia surname Woolf
"1609040579130-0"
redis> XADD writers * name Jane surname Austen
"1609040579130-1"
redis> XADD writers * name Toni surname Morrison
"1609040579130-2"
redis> XADD writers * name Agatha surname Christie
"1609040579131-0"
redis> XADD writers * name Ngozi surname Adichie
"1609040579131-1"
redis> XLEN writers
(integer) 5
redis> XREVRANGE writers + - COUNT 1
1) 1) "1609040579131-1"
   2) 1) "name"
      2) "Ngozi"
      3) "surname"
      4) "Adichie"
```

Return a range of elements in a stream, with IDs matching the specified IDs interval, in reverse order (from greater to smaller IDs) compared to XRANGE

XLEN

```
XLEN key
```

Example

```
redis> XADD mystream * item 1
"1609040580250-0"
redis> XADD mystream * item 2
"1609040580250-1"
redis> XADD mystream * item 3
"1609040580251-0"
redis> XLEN mystream
(integer) 3
```

Return the number of entries in a stream

XACK

```
XACK key group ID [ID ...]
```

Example

```
redis> XACK mystream mygroup 1526569495631-0
ERR Unknown or disabled command 'XACK'
```

Marks a pending message as correctly processed, effectively removing it from the pending entries list of the consumer group. Return value of the command is the number of messages successfully acknowledged, that is, the IDs we were actually able to resolve in the PEL.

Miscellaneous

Cluster

CLUSTER ADDSLOTS	Assign new hash slots to receiving node
CLUSTER BUMPEPOCH	Advance the cluster config epoch
CLUSTER COUNT-FAILURE-REPORTS	Return the number of failure reports active for a given node
CLUSTER COUNTKEYSINSLOT	Return the number of local keys in the specified hash slot
CLUSTER DELSLOTS	Set hash slots as unbound in receiving node
CLUSTER FAILOVER	Forces a replica to perform a manual failover of its master.
CLUSTER FLUSHSLOTS	Delete a node's own slots information
CLUSTER FORGET	Remove a node from the nodes table
CLUSTER GETKEYSINSLOT	Return local key names in the specified hash slot
CLUSTER INFO	Provides info about Redis Cluster node state
CLUSTER KEYSLOT	Returns the hash slot of the specified key
CLUSTER MEET	Force a node cluster to handshake with another node

Transactions

DISCARD	Discard all commands issued after MULTI
EXEC	Execute all commands issued after MULTI
MULTI	Mark the start of a transaction block
UNWATCH	Forget about all watched keys
WATCH	Watch the given keys to determine execution of the MULTI/EXEC block

CLUSTER MYID	Return the node id
CLUSTER NODES	Get Cluster config for the node
CLUSTER REPLICATE	Reconfigure a node as a replica of the specified master node
CLUSTER RESET	Reset a Redis Cluster node
CLUSTER SAVECONFIG	Forces the node to save cluster state on disk
CLUSTER SET-CONFIG-EPOCH	Set the configuration epoch in a new node
CLUSTER SETSLOT	Bind a hash slot to a specific node
CLUSTER SLAVES	List replica nodes of the specified master node
CLUSTER REPLICAS	List replica nodes of the specified master node
CLUSTER SLOTS	Get array of Cluster slot to node mappings
READONLY	Enables read queries for a connection to a cluster replica node
READWRITE	Disables read queries for a connection to a cluster replica node

Scripting

EVAL	Execute a Lua script server side
EVALSHA	Execute a Lua script server side
SCRIPT DEBUG	Set the debug mode for executed scripts.
SCRIPT EXISTS	Check existence of scripts in the script cache.
SCRIPT FLUSH	Remove all the scripts from the script cache.
SCRIPT KILL	Kill the script currently in execution.
SCRIPT LOAD	Load the specified Lua script into the script cache.

Pubsub

PSUBSCRIBE	Listen for messages published to channels matching the given patterns
PUBSUB	Inspect the state of the Pub/Sub subsystem
PUBLISH	Post a message to a channel
PUNSUBSCRIBE	Stop listening for messages posted to channels matching the given patterns
SUBSCRIBE	Listen for messages published to the given channels
UNSUBSCRIBE	Stop listening for messages posted to the given channels