# Redis Command Reference

Keys Strings Hashes Lists Sets Sorted Sets Pub/Sub Transactions Scripting Connection Server

**APPEND** `key value`
Append a value to a key

**AUTH** `password`
Authenticate to the server

**BGREWRITEAOF**
Asynchronously rewrite the append-only file

**BGSAVE**
Asynchronously save the dataset to disk

**BITCOUNT** `key [start] [end]`
Count set bits in a string

**BITOP** `operation destkey key [ke…`
Perform bitwise operations between strings

**BLPOP** `key [key ...] timeout`
Remove and get the first element in a list, or block until one is available

**BRPOP** `key [key ...] timeout`
Remove and get the last element in a list, or block until one is available

**BRPOPLPUSH** `source destination…`
Pop a value from a list, push it to another list and return it; or block until one is available

**CLIENT KILL** `ip:port`
Kill the connection of a client

**CLIENT LIST**
Get the list of client connections

**CLIENT GETNAME**
Get the current connection name

**CLIENT SETNAME** `connection-na…`
Set the current connection name

**CONFIG GET** `parameter`
Get the value of a configuration parameter

**CONFIG REWRITE**
Rewrite the configuration file with the in memory configuration

**CONFIG SET** `parameter value`
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** `key`
Get debugging information about a key

**DEBUG SEGFAULT**
Make the server crash

**DECR** `key`
Decrement the integer value of a key by one

**DECRBY** `key decrement`
Decrement the integer value of a key by the given number

**DEL** `key [key ...]`
Delete a key

**DISCARD**
Discard all commands issued after MULTI

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

**ECHO** `message`
Echo the given string

**EVAL** `script numkeys key [key ....`
Execute a Lua script server side

**EVALSHA** `sha1 numkeys key [key …`
Execute a Lua script server side

**EXEC**
Execute all commands issued after MULTI

**EXISTS** `key`
Determine if a key exists

**EXPIRE** `key seconds`
Set a key's time to live in seconds

**EXPIREAT** `key timestamp`
Set the expiration for a key as a UNIX timestamp

**FLUSHALL**
Remove all keys from all databases

**FLUSHDB**
Remove all keys from the current database

**GET** `key`
Get the value of a key

**GETBIT** `key offset`
Returns the bit value at offset in the string value stored at key

**GETRANGE** `key start end`
Get a substring of the string stored at a key

**GETSET** `key value`
Set the string value of a key and return its old value

**HDEL** `key field [field ...]`
Delete one or more hash fields

**HEXISTS** `key field`
Determine if a hash field exists

**HGET** `key field`
Get the value of a hash field

**HGETALL** `key`
Get all the fields and values in a hash

**HINCRBY** `key field increment`
Increment the integer value of a hash field by the given number

**HINCRBYFLOAT** `key field increm…`
Increment the float value of a hash field by the given amount

**HKEYS** `key`
Get all the fields in a hash

**HLEN** `key`
Get the number of fields in a hash

**HMGET** `key field [field ...]`
Get the values of all the given hash fields

**HMSET** `key field value [field va…`
Set multiple hash fields to multiple values

**HSET** `key field value`
Set the string value of a hash field

**HSETNX** `key field value`
Set the value of a hash field, only if the field does not exist

**HVALS** `key`
Get all the values in a hash

**INCR** `key`
Increment the integer value of a key by one

**INCRBY** `key increment`
Increment the integer value of a key by the given amount

**INCRBYFLOAT** `key increment`
Increment the float value of a key by the given amount

**INFO** `[section]`
Get information and statistics about the server

**KEYS** `pattern`
Find all keys matching the given pattern

**LASTSAVE**
Get the UNIX time stamp of the last successful save to disk

**LINDEX** `key index`
Get an element from a list by its index

**LINSERT** `key BEFORE|AFTER pivot…`
Insert an element before or after another element in a list

**LLEN** `key`
Get the length of a list

**LPOP** `key`
Remove and get the first element in a list

**LPUSH** `key value [value ...]`
Prepend one or multiple values to a list

**LPUSHX** `key value`
Prepend a value to a list, only if the list exists

**LRANGE** `key start stop`
Get a range of elements from a list

**LREM** `key count value`
Remove elements from a list

**LSET** `key index value`
Set the value of an element in a list by its index

**LTRIM** `key start stop`
Trim a list to the specified range

**MGET** `key [key ...]`
Get the values of all the given keys

**MIGRATE** `host port key destinat…`
Atomically transfer a key from a Redis instance to another one.

**MONITOR**
Listen for all requests received by the server in real time

**MOVE** `key db`
Move a key to another database

**MSET** `key value [key value ...]`
Set multiple keys to multiple values

**MSETNX** `key value [key value ...]`
Set multiple keys to multiple values, only if none of the keys exist

**MULTI**
Mark the start of a transaction block

**OBJECT** `subcommand [arguments [a…`
Inspect the internals of Redis objects

**PERSIST** `key`
Remove the expiration from a key

**PEXPIRE** `key milliseconds`
Set a key's time to live in milliseconds

**PEXPIREAT** `key milliseconds-tim…`
Set the expiration for a key as a UNIX timestamp specified in milliseconds

**PING**
Ping the server

**PSETEX** `key milliseconds value`
Set the value and expiration in milliseconds of a key

**PSUBSCRIBE** `pattern [pattern ...]`
Listen for messages published to channels matching the given patterns

**PUBSUB** `subcommand [argument [ar...]`
Inspect the state of the Pub/Sub subsystem

**PTTL** `key`
Get the time to live for a key in milliseconds

**PUBLISH** `channel message`
Post a message to a channel

**PUNSUBSCRIBE** `[pattern [patter...]`
Stop listening for messages posted to channels matching the given patterns

**QUIT**
Close the connection

**RANDOMKEY**
Return a random key from the keyspace

**RENAME** `key newkey`
Rename a key

**RENAMENX** `key newkey`
Rename a key, only if the new key does not exist

**RESTORE** `key ttl serialized-val...`
Create a key using the provided serialized value, previously obtained using DUMP.

**RPOP** `key`
Remove and get the last element in a list

**RPOPLPUSH** `source destination`
Remove the last element in a list, append it to another list and return it

**RPUSH** `key value [value ...]`
Append one or multiple values to a list

**RPUSHX** `key value`
Append a value to a list, only if the list exists

**SADD** `key member [member ...]`
Add one or more members to a set

**SAVE**
Synchronously save the dataset to disk

**SCARD** `key`
Get the number of members in a set

**SCRIPT EXISTS** `script [script...]`
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** `script`
Load the specified Lua script into the script cache.

**SDIFF** `key [key ...]`
Subtract multiple sets

**SDIFFSTORE** `destination key [k...`
Subtract multiple sets and store the resulting set in a key

**SELECT** `index`
Change the selected database for the current connection

**SET** `key value [EX seconds] [PX m...`
Set the string value of a key

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

**SETEX** `key seconds value`
Set the value and expiration of a key

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

**SETRANGE** `key offset value`
Overwrite part of a string at key starting at the specified offset

**SHUTDOWN** `[NOSAVE] [SAVE]`
Synchronously save the dataset to disk and then shut down the server

**SINTER** `key [key ...]`
Intersect multiple sets

**SINTERSTORE** `destination key [...`
Intersect multiple sets and store the resulting set in a key

**SISMEMBER** `key member`
Determine if a given value is a member of a set

**SLAVEOF** `host port`
Make the server a slave of another instance, or promote it as master

**SLOWLOG** `subcommand [argument]`
Manages the Redis slow queries log

**SMEMBERS** `key`
Get all the members in a set

**SMOVE** `source destination member`
Move a member from one set to another

**SORT** `key [BY pattern] [LIMIT of...`
Sort the elements in a list, set or sorted set

**SPOP** `key`
Remove and return a random member from a set

**SRANDMEMBER** `key [count]`
Get one or multiple random members from a set

**SREM** `key member [member ...]`
Remove one or more members from a set

**STRLEN** `key`
Get the length of the value stored in a key

**SUBSCRIBE** `channel [channel ...]`
Listen for messages published to the given channels

**SUNION** `key [key ...]`
Add multiple sets

**SUNIONSTORE** `destination key [...`
Add multiple sets and store the resulting set in a key

**SYNC**
Internal command used for replication

**TIME**
Return the current server time

**TTL** `key`
Get the time to live for a key

**TYPE** `key`
Determine the type stored at key

**UNSUBSCRIBE** `[channel [channel...`
Stop listening for messages posted to the given channels

**UNWATCH**
Forget about all watched keys

**WATCH** `key [key ...]`
Watch the given keys to determine execution of the MULTI/EXEC block

**ZADD** `key score member [score me...`
Add one or more members to a sorted set, or update its score if it already exists

**ZCARD** `key`
Get the number of members in a sorted set

**ZCOUNT** `key min max`
Count the members in a sorted set with scores within the given values

**ZINCRBY** `key increment member`
Increment the score of a member in a sorted set

**ZINTERSTORE** `destination numke...`
Intersect multiple sorted sets and store the resulting sorted set in a new key

**ZRANGE** `key start stop [WITHSCOR...`
Return a range of members in a sorted set, by index

**ZRANGEBYSCORE** `key min max [W...`
Return a range of members in a sorted set, by score

**ZRANK** `key member`
Determine the index of a member in a sorted set

**ZREM** `key member [member ...]`
Remove one or more members from a sorted set

**ZREMRANGEBYRANK** `key start st...`
Remove all members in a sorted set within the given indexes

**ZREMRANGEBYSCORE** `key min max`
Remove all members in a sorted set within the given scores

**ZREVRANGE** `key start stop [WITH...`
Return a range of members in a sorted set, by index, with scores ordered from high to low

**ZREVRANGEBYSCORE** `key max mi...`
Return a range of members in a sorted set, by score, with scores ordered from high to low

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

**ZSCORE** `key member`
Get the score associated with the given member in a sorted set

**ZUNIONSTORE** `destination numke...`
Add multiple sorted sets and store the resulting sorted set in a new key

**SCAN** `cursor [MATCH pattern] [CO...`
Incrementally iterate the keys space

**SSCAN** `key cursor [MATCH pattern...`
Incrementally iterate Set elements

**HSCAN** `key cursor [MATCH pattern...`
Incrementally iterate hash fields and associated values

**ZSCAN** `key cursor [MATCH pattern...`
Incrementally iterate sorted sets elements and associated scores