

# Redis - Overview

Redis is an open source, advanced key-value store and an apt solution for building highperformance, scalable web applications.

Redis has three main peculiarities that sets it apart.


- Redis holds its database entirely in the memory, using the disk only for persistence.
- Redis has a relatively rich set of data types when compared to many key-value data stores.
- Redis can replicate data to any number of slaves.

## Redis Advantages

Following are certain advantages of Redis.

- **Exceptionally fast** – Redis is very fast and can perform about 110000 SETs per second, about 81000 GETs per second.
- **Supports rich data types** – Redis natively supports most of the datatypes that developers already know such as list, set, sorted set, and hashes. This makes it easy to solve a variety of problems as we know which problem can be handled better by which data type.
- **Operations are atomic** – All Redis operations are atomic, which ensures that if two clients concurrently access, Redis server will receive the updated value.
- **Multi-utility tool** – Redis is a multi-utility tool and can be used in a number of use cases such as caching, messaging-queues (Redis natively supports Publish/Subscribe), any short-lived data in your application, such as web application sessions, web page hit counts, etc.

## Redis Versus Other Key-value Stores

- Redis is a different evolution path in the key-value DBs, where value contain more complex data types, with atomic operations defined on 

data types.

- Redis is an in-memory database but persistent on disk database, hence it represents a different trade off where very high write and read speed is achieved with the limitation of data sets that can't be larger than the memory.
- Another advantage of in-memory databases is that the memory representation of complex data structures is much simpler to manipulate compared to the same data structure on disk. Thus, Redis can do a lot with little internal complexity.

# Redis - Configuration

In Redis, there is a configuration file (redis.conf) available at the root directory of Redis. Although you can get and set all Redis configurations by Redis **CONFIG** command.

## Syntax

Following is the basic syntax of Redis **CONFIG** command.

```
redis 127.0.0.1:6379> CONFIG GET CONFIG_SETTING_NAME
```

## Example

```
redis 127.0.0.1:6379> CONFIG GET loglevel
1) "loglevel"
2) "notice"
```

To get all configuration settings, use \* in place of CONFIG\_SETTING\_NAME

## Example

```
redis 127.0.0.1:6379> CONFIG GET *
1) "dbfilename"
2) "dump.rdb"
3) "requirepass"
4) ""
5) "masterauth"
6) ""
7) "unixsocket"
8) ""
9) "logfile"
10) ""
11) "pidfile"
12) "/var/run/redis.pid"
13) "maxmemory"
14) "0"
```

- 15) "maxmemory-samples"
- 16) "3"
- 17) "timeout"
- 18) "0"
- 19) "tcp-keepalive"
- 20) "0"
- 21) "auto-aof-rewrite-percentage"
- 22) "100"
- 23) "auto-aof-rewrite-min-size"
- 24) "67108864"
- 25) "hash-max-ziplist-entries"
- 26) "512"
- 27) "hash-max-ziplist-value"
- 28) "64"
- 29) "list-max-ziplist-entries"
- 30) "512"
- 31) "list-max-ziplist-value"
- 32) "64"
- 33) "set-max-intset-entries"
- 34) "512"
- 35) "zset-max-ziplist-entries"
- 36) "128"
- 37) "zset-max-ziplist-value"
- 38) "64"
- 39) "hll-sparse-max-bytes"
- 40) "3000"
- 41) "lua-time-limit"
- 42) "5000"
- 43) "slowlog-log-slower-than"
- 44) "10000"
- 45) "latency-monitor-threshold"
- 46) "0"
- 47) "slowlog-max-len"
- 48) "128"
- 49) "port"
- 50) "6379"
- 51) "tcp-backlog"
- 52) "511"
- 53) "databases"
- 54) "16"
- 55) "repl-ping-slave-period"
- 56) "10"

57) "repl-timeout"  
58) "60"  
59) "repl-backlog-size"  
60) "1048576"  
61) "repl-backlog-ttl"  
62) "3600"  
63) "maxclients"  
64) "4064"  
65) "watchdog-period"  
66) "0"  
67) "slave-priority"  
68) "100"  
69) "min-slaves-to-write"  
70) "0"  
71) "min-slaves-max-lag"  
72) "10"  
73) "hz"  
74) "10"  
75) "no-appendfsync-on-rewrite"  
76) "no"  
77) "slave-serve-stale-data"  
78) "yes"  
79) "slave-read-only"  
80) "yes"  
81) "stop-writes-on-bgsave-error"  
82) "yes"  
83) "daemonize"  
84) "no"  
85) "rdbcompression"  
86) "yes"  
87) "rdbchecksum"  
88) "yes"  
89) "activerehashing"  
90) "yes"  
91) "repl-disable-tcp-nodelay"  
92) "no"  
93) "aof-rewrite-incremental-fsync"  
94) "yes"  
95) "appendonly"  
96) "no"  
97) "dir"  
98) "/home/deepak/Downloads/redis-2.8.13/src"

```
99) "maxmemory-policy"
100) "volatile-lru"
101) "appendfsync"
102) "everysec"
103) "save"
104) "3600 1 300 100 60 10000"
105) "loglevel"
106) "notice"
107) "client-output-buffer-limit"
108) "normal 0 0 0 slave 268435456 67108864 60 pubsub 33554432 8388608 60"
109) "unixsocketperm"
110) "0"
111) "slaveof"
112) ""
113) "notify-keyspace-events"
114) ""
115) "bind"
116) ""
```

## Edit Configuration

To update configuration, you can edit **redis.conf** file directly or you can update configurations via **CONFIG set** command.

## Syntax

Following is the basic syntax of **CONFIG SET** command.

```
redis 127.0.0.1:6379> CONFIG SET CONFIG_SETTING_NAME NEW_CONFIG_VALUE
```

## Example

```
redis 127.0.0.1:6379> CONFIG SET loglevel "notice"
OK
redis 127.0.0.1:6379> CONFIG GET loglevel
1) "loglevel"
2) "notice"
```

# Redis - Data Types

Redis supports 5 types of data types.

## Strings

Redis string is a sequence of bytes. Strings in Redis are binary safe, meaning they have a known length not determined by any special terminating characters. Thus, you can store anything up to 512 megabytes in one string.

## Example

```
redis 127.0.0.1:6379> SET name "tutorialspoint"  
OK  
redis 127.0.0.1:6379> GET name  
"tutorialspoint"
```

In the above example, **SET** and **GET** are Redis commands, **name** is the key used in Redis and **tutorialspoint** is the string value that is stored in Redis.

**Note** – A string value can be at max 512 megabytes in length.

## Hashes

A Redis hash is a collection of key value pairs. Redis Hashes are maps between string fields and string values. Hence, they are used to represent objects.

## Example

```
redis 127.0.0.1:6379> HMSET user:1 username tutorialspoint password  
tutorialspoint points 200  
OK  
redis 127.0.0.1:6379> HGETALL user:1  
1) "username"  
2) "tutorialspoint"  
3) "password"  
4) "tutorialspoint"
```

- 5) "points"
- 6) "200"

In the above example, hash data type is used to store the user's object which contains basic information of the user. Here **HMSET**, **HGETALL** are commands for Redis, while **user – 1** is the key.

Every hash can store up to  $2^{32} - 1$  field-value pairs (more than 4 billion).

## Lists

Redis Lists are simply lists of strings, sorted by insertion order. You can add elements to a Redis List on the head or on the tail.

### Example

```
redis 127.0.0.1:6379> lpush tutoriallist redis
(integer) 1
redis 127.0.0.1:6379> lpush tutoriallist mongodb
(integer) 2
redis 127.0.0.1:6379> lpush tutoriallist rabbitmq
(integer) 3
redis 127.0.0.1:6379> lrange tutoriallist 0 10
```

- 1) "rabbitmq"
- 2) "mongodb"
- 3) "redis"

The max length of a list is  $2^{32} - 1$  elements (4294967295, more than 4 billion of elements per list).

## Sets

Redis Sets are an unordered collection of strings. In Redis, you can add, remove, and test for the existence of members in  $O(1)$  time complexity.

### Example

```
redis 127.0.0.1:6379> sadd tutoriallist redis
(integer) 1
```



```
redis 127.0.0.1:6379> sadd tutoriallist mongodb
(integer) 1
redis 127.0.0.1:6379> sadd tutoriallist rabbitmq
(integer) 1
redis 127.0.0.1:6379> sadd tutoriallist rabbitmq
(integer) 0
redis 127.0.0.1:6379> smembers tutoriallist
```

- 1) "rabbitmq"
- 2) "mongodb"
- 3) "redis"

**Note** – In the above example, **rabbitmq** is added twice, however due to unique property of the set, it is added only once.

The max number of members in a set is  $2^{32} - 1$  (4294967295, more than 4 billion of members per set).

## Sorted Sets

Redis Sorted Sets are similar to Redis Sets, non-repeating collections of Strings. The difference is, every member of a Sorted Set is associated with a score, that is used in order to take the sorted set ordered, from the smallest to the greatest score. While members are unique, the scores may be repeated.

## Example

```
redis 127.0.0.1:6379> zadd tutoriallist 0 redis
(integer) 1
redis 127.0.0.1:6379> zadd tutoriallist 0 mongodb
(integer) 1
redis 127.0.0.1:6379> zadd tutoriallist 0 rabbitmq
(integer) 1
redis 127.0.0.1:6379> zadd tutoriallist 0 rabbitmq
(integer) 0
redis 127.0.0.1:6379> ZRANGEBYSCORE tutoriallist 0 1000
```

- 1) "redis"
- 2) "mongodb"
- 3) "rabbitmq"



# Redis - Commands

Redis commands are used to perform some operations on Redis server.

To run commands on Redis server, you need a Redis client. Redis client is available in Redis package, which we have installed earlier.

## Syntax

Following is the basic syntax of Redis client.

```
$redis-cli
```

## Example

Following example explains how we can start Redis client.

To start Redis client, open the terminal and type the command **redis-cli**. This will connect to your local server and now you can run any command.

```
$redis-cli  
redis 127.0.0.1:6379>  
redis 127.0.0.1:6379> PING  
PONG
```

In the above example, we connect to Redis server running on the local machine and execute a command **PING**, that checks whether the server is running or not.

## Run Commands on the Remote Server

To run commands on Redis remote server, you need to connect to the server by the same client **redis-cli**

## Syntax

```
$ redis-cli -h host -p port -a password
```



## Example

Following example shows how to connect to Redis remote server, running on host 127.0.0.1, port 6379 and has password mypass.

```
$redis-cli -h 127.0.0.1 -p 6379 -a "mypass"  
redis 127.0.0.1:6379>  
redis 127.0.0.1:6379> PING  
PONG
```

# Redis - Keys

Redis keys commands are used for managing keys in Redis. Following is the syntax for using redis keys commands.

## Syntax

```
redis 127.0.0.1:6379> COMMAND KEY_NAME
```

## Example

```
redis 127.0.0.1:6379> SET tutorialspoint redis
OK
redis 127.0.0.1:6379> DEL tutorialspoint
(integer) 1
```

In the above example, **DEL** is the command, while **tutorialspoint** is the key. If the key is deleted, then the output of the command will be (integer) 1, otherwise it will be (integer) 0.

## Redis Keys Commands

Following table lists some basic commands related to keys.

Sr.No	Command & Description
1	<b>DEL key</b> This command deletes the key, if it exists.
2	<b>DUMP key</b> This command returns a serialized version of the value stored at the specified key.
3	<b>EXISTS key</b> This command checks whether the key exists or not.
4	<b>EXPIRE key</b> seconds Sets the expiry of the key after the specified time.

5	<b>EXPIREAT key timestamp</b> Sets the expiry of the key after the specified time. Here time is in Unix timestamp format.
6	<b>PEXPIRE key milliseconds</b> Set the expiry of key in milliseconds.
7	<b>PEXPIREAT key milliseconds-timestamp</b> Sets the expiry of the key in Unix timestamp specified as milliseconds.
8	<b>KEYS pattern</b> Finds all keys matching the specified pattern.
9	<b>MOVE key db</b> Moves a key to another database.
10	<b>PERSIST key</b> Removes the expiration from the key.
11	<b>PTTL key</b> Gets the remaining time in keys expiry in milliseconds.
12	<b>TTL key</b> Gets the remaining time in keys expiry.
13	<b>RANDOMKEY</b> Returns a random key from Redis.
14	<b>RENAME key newkey</b> Changes the key name.
15	<b>RENAMENX key newkey</b> Renames the key, if a new key doesn't exist.
16	<b>TYPE key</b> Returns the data type of the value stored in the key.

# Redis - Strings

Redis strings commands are used for managing string values in Redis. Following is the syntax for using Redis string commands.

## Syntax

```
redis 127.0.0.1:6379> COMMAND KEY_NAME
```

## Example

```
redis 127.0.0.1:6379> SET tutorialspoint redis
OK
redis 127.0.0.1:6379> GET tutorialspoint
"redis"
```

In the above example, **SET** and **GET** are the commands, while **tutorialspoint** is the key.

## Redis Strings Commands

Following table lists some basic commands to manage strings in Redis.

Sr.No	Command & Description
1	<b>SET key value</b> This command sets the value at the specified key.
2	<b>GET key</b> Gets the value of a key.
3	<b>GETRANGE key start end</b> Gets a substring of the string stored at a key.
4	<b>GETSET key value</b> Sets the string value of a key and return its old value.
5	<b>GETBIT key offset</b>



	Returns the bit value at the offset in the string value stored at the key.
6	<b>MGET key1 [key2..]</b> Gets the values of all the given keys
7	<b>SETBIT key offset value</b> Sets or clears the bit at the offset in the string value stored at the key
8	<b>SETEX key seconds value</b> Sets the value with the expiry of a key
9	<b>SETNX key value</b> Sets the value of a key, only if the key does not exist
10	<b>SETRANGE key offset value</b> Overwrites the part of a string at the key starting at the specified offset
11	<b>STRLEN key</b> Gets the length of the value stored in a key
12	<b>MSET key value [key value ...]</b> Sets multiple keys to multiple values
13	<b>MSETNX key value [key value ...]</b> Sets multiple keys to multiple values, only if none of the keys exist
14	<b>PSETEX key milliseconds value</b> Sets the value and expiration in milliseconds of a key
15	<b>INCR key</b> Increments the integer value of a key by one
16	<b>INCRBY key increment</b> Increments the integer value of a key by the given amount
17	<b>INCRBYFLOAT key increment</b> Increments the float value of a key by the given amount
18	<b>DECR key</b> Decrements the integer value of a key by one
19	<b>DECRBY key decrement</b> Decrements the integer value of a key by the given number
20	<b>APPEND key value</b> Appends a value to a key





# Redis - Hashes

Redis Hashes are maps between the string fields and the string values. Hence, they are the perfect data type to represent objects.

In Redis, every hash can store up to more than 4 billion field-value pairs.

## Example

```
redis 127.0.0.1:6379> HMSET tutorialspoint name "redis tutorial"
description "redis basic commands for caching" likes 20 visitors 23000
OK
redis 127.0.0.1:6379> HGETALL tutorialspoint
1) "name"
2) "redis tutorial"
3) "description"
4) "redis basic commands for caching"
5) "likes"
6) "20"
7) "visitors"
8) "23000"
```

In the above example, we have set Redis tutorials detail (name, description, likes, visitors) in hash named 'tutorialspoint'.

## Redis Hash Commands

Following table lists some basic commands related to hash.

Sr.No	Command & Description
1	<b>HDEL key field2 [field2]</b> Deletes one or more hash fields.
2	<b>HEXISTS key field</b> Determines whether a hash field exists or not.
3	<b>HGET key field</b> Gets the value of a hash field stored at the specified key.

4	<b>HGETALL key</b> Gets all the fields and values stored in a hash at the specified key
5	<b>HINCRBY key field increment</b> Increments the integer value of a hash field by the given number
6	<b>HINCRBYFLOAT key field increment</b> Increments the float value of a hash field by the given amount
7	<b>HKEYS key</b> Gets all the fields in a hash
8	<b>HLEN key</b> Gets the number of fields in a hash
9	<b>HMGET key field1 [field2]</b> Gets the values of all the given hash fields
10	<b>HMSET key field1 value1 [field2 value2 ]</b> Sets multiple hash fields to multiple values
11	<b>HSET key field value</b> Sets the string value of a hash field
12	<b>HSETNX key field value</b> Sets the value of a hash field, only if the field does not exist
13	<b>HVALS key</b> Gets all the values in a hash
14	<b>HSCAN key cursor [MATCH pattern] [COUNT count]</b> Incrementally iterates hash fields and associated values

# Redis - Lists

Redis Lists are simply lists of strings, sorted by insertion order. You can add elements in Redis lists in the head or the tail of the list.

Maximum length of a list is  $2^{32} - 1$  elements (4294967295, more than 4 billion of elements per list).

## Example

```
redis 127.0.0.1:6379> LPUSH tutorials redis
(integer) 1
redis 127.0.0.1:6379> LPUSH tutorials mongodb
(integer) 2
redis 127.0.0.1:6379> LPUSH tutorials mysql
(integer) 3
redis 127.0.0.1:6379> LRANGE tutorials 0 10
1) "mysql"
2) "mongodb"
3) "redis"
```

In the above example, three values are inserted in Redis list named 'tutorials' by the command **LPUSH**.

## Redis Lists Commands

Following table lists some basic commands related to lists.

Sr.No	Command & Description
1	<b>BLPOP key1 [key2 ] timeout</b> Removes and gets the first element in a list, or blocks until one is available
2	<b>BRPOP key1 [key2 ] timeout</b> Removes and gets the last element in a list, or blocks until one is available
3	<b>BRPOPLPUSH source destination timeout</b> Pops a value from a list, pushes it to another list and returns it; or blocks until one is available

4	<b>LINDEX key index</b> Gets an element from a list by its index
5	<b>LINSERT key BEFORE AFTER pivot value</b> Inserts an element before or after another element in a list
6	<b>LLEN key</b> Gets the length of a list
7	<b>LPOP key</b> Removes and gets the first element in a list
8	<b>LPUSH key value1 [value2]</b> Prepends one or multiple values to a list
9	<b>LPUSHX key value</b> Prepends a value to a list, only if the list exists
10	<b>LRANGE key start stop</b> Gets a range of elements from a list
11	<b>LREM key count value</b> Removes elements from a list
12	<b>LSET key index value</b> Sets the value of an element in a list by its index
13	<b>LTRIM key start stop</b> Trims a list to the specified range
14	<b>RPOP key</b> Removes and gets the last element in a list
15	<b>RPOPLUSH source destination</b> Removes the last element in a list, appends it to another list and returns it
16	<b>RPUSH key value1 [value2]</b> Appends one or multiple values to a list
17	<b>RPUSHX key value</b> Appends a value to a list, only if the list exists

# Redis - Sets

Redis Sets are an unordered collection of unique strings. Unique means sets does not allow repetition of data in a key.

In Redis set add, remove, and test for the existence of members in  $O(1)$  (constant time regardless of the number of elements contained inside the Set). The maximum length of a list is  $2^{32} - 1$  elements (4294967295, more than 4 billion of elements per set).

## Example

```
redis 127.0.0.1:6379> SADD tutorials redis
(integer) 1
redis 127.0.0.1:6379> SADD tutorials mongodb
(integer) 1
redis 127.0.0.1:6379> SADD tutorials mysql
(integer) 1
redis 127.0.0.1:6379> SADD tutorials mysql
(integer) 0
redis 127.0.0.1:6379> SMEMBERS tutorials
1) "mysql"
2) "mongodb"
3) "redis"
```

In the above example, three values are inserted in Redis set named 'tutorials' by the command **SADD**.

## Redis Sets Commands

Following table lists some basic commands related to sets.

Sr.No	Command & Description
1	<b>SADD key member1 [member2]</b> Adds one or more members to a set
2	<b>SCARD key</b> Gets the number of members in a set



3	<b>SDIFF key1 [key2]</b> Subtracts multiple sets
4	<b>SDIFFSTORE destination key1 [key2]</b> Subtracts multiple sets and stores the resulting set in a key
5	<b>SINTER key1 [key2]</b> Intersects multiple sets
6	<b>SINTERSTORE destination key1 [key2]</b> Intersects multiple sets and stores the resulting set in a key
7	<b>SISMEMBER key member</b> Determines if a given value is a member of a set
8	<b>SMEMBERS key</b> Gets all the members in a set
9	<b>SMOVE source destination member</b> Moves a member from one set to another
10	<b>SPOP key</b> Removes and returns a random member from a set
11	<b>SRANDMEMBER key [count]</b> Gets one or multiple random members from a set
12	<b>SREM key member1 [member2]</b> Removes one or more members from a set
13	<b>SUNION key1 [key2]</b> Adds multiple sets
14	<b>SUNIONSTORE destination key1 [key2]</b> Adds multiple sets and stores the resulting set in a key
15	<b>SSCAN key cursor [MATCH pattern] [COUNT count]</b> Incrementally iterates set elements

# Redis - Sorted Sets

Redis Sorted Sets are similar to Redis Sets with the unique feature of values stored in a set. The difference is, every member of a Sorted Set is associated with a score, that is used in order to take the sorted set ordered, from the smallest to the greatest score.

In Redis sorted set, add, remove, and test for the existence of members in  $O(1)$  (constant time regardless of the number of elements contained inside the set).

Maximum length of a list is  $2^{32} - 1$  elements (4294967295, more than 4 billion of elements per set).

## Example

```
redis 127.0.0.1:6379> ZADD tutorials 1 redis
(integer) 1
redis 127.0.0.1:6379> ZADD tutorials 2 mongodb
(integer) 1
redis 127.0.0.1:6379> ZADD tutorials 3 mysql
(integer) 1
redis 127.0.0.1:6379> ZADD tutorials 3 mysql
(integer) 0
redis 127.0.0.1:6379> ZADD tutorials 4 mysql
(integer) 0
redis 127.0.0.1:6379> ZRANGE tutorials 0 10 WITHSCORES
1) "redis"
2) "1"
3) "mongodb"
4) "2"
5) "mysql"
6) "4"
```

In the above example, three values are inserted with its score in Redis sorted set named 'tutorials' by the command **ZADD**.

## Redis Sorted Sets Commands

Following table lists some basic commands related to sorted sets.



Sr.No	Command & Description
1	<b>ZADD key score1 member1 [score2 member2]</b> Adds one or more members to a sorted set, or updates its score, if it already exists
2	<b>ZCARD key</b> Gets the number of members in a sorted set
3	<b>ZCOUNT key min max</b> Counts the members in a sorted set with scores within the given values
4	<b>ZINCRBY key increment member</b> Increments the score of a member in a sorted set
5	<b>ZINTERSTORE destination numkeys key [key ...]</b> Intersects multiple sorted sets and stores the resulting sorted set in a new key
6	<b>ZLEXCOUNT key min max</b> Counts the number of members in a sorted set between a given lexicographical range
7	<b>ZRANGE key start stop [WITHSCORES]</b> Returns a range of members in a sorted set, by index
8	<b>ZRANGEBYLEX key min max [LIMIT offset count]</b> Returns a range of members in a sorted set, by lexicographical range
9	<b>ZRANGEBYSCORE key min max [WITHSCORES] [LIMIT]</b> Returns a range of members in a sorted set, by score
10	<b>ZRANK key member</b> Determines the index of a member in a sorted set
11	<b>ZREM key member [member ...]</b> Removes one or more members from a sorted set
12	<b>ZREMRANGEBYLEX key min max</b> Removes all members in a sorted set between the given lexicographical range
13	<b>ZREMRANGEBYRANK key start stop</b> Removes all members in a sorted set within the given indexes
14	<b>ZREMRANGEBYSCORE key min max</b>

	Removes all members in a sorted set within the given scores
15	<b>ZREVRANGE key start stop [WITHSCORES]</b> Returns a range of members in a sorted set, by index, with scores ordered from high to low
16	<b>ZREVRANGEBYSCORE key max min [WITHSCORES]</b> Returns a range of members in a sorted set, by score, with scores ordered from high to low
17	<b>ZREVRANK key member</b> Determines the index of a member in a sorted set, with scores ordered from high to low
18	<b>ZSCORE key member</b> Gets the score associated with the given member in a sorted set
19	<b>ZUNIONSTORE destination numkeys key [key ...]</b> Adds multiple sorted sets and stores the resulting sorted set in a new key
20	<b>ZSCAN key cursor [MATCH pattern] [COUNT count]</b> Incrementally iterates sorted sets elements and associated scores

# Redis - Publish Subscribe

Redis Pub/Sub implements the messaging system where the senders (in redis terminology called publishers) sends the messages while the receivers (subscribers) receive them. The link by which the messages are transferred is called **channel**.

In Redis, a client can subscribe any number of channels.

## Example

Following example explains how publish subscriber concept works. In the following example, one client subscribes a channel named 'redisChat'.

```
redis 127.0.0.1:6379> SUBSCRIBE redisChat
Reading messages... (press Ctrl-C to quit)
1) "subscribe"
2) "redisChat"
3) (integer) 1
```

Now, two clients are publishing the messages on the same channel named 'redisChat' and the above subscribed client is receiving messages.

```
redis 127.0.0.1:6379> PUBLISH redisChat "Redis is a great caching technique"
(integer) 1
redis 127.0.0.1:6379> PUBLISH redisChat "Learn redis by tutorials point"
(integer) 1
1) "message"
2) "redisChat"
3) "Redis is a great caching technique"
1) "message"
2) "redisChat"
3) "Learn redis by tutorials point"
```

## Redis PubSub Commands

Following table lists some basic commands related to Redis Pub/Sub.

Sr.No	Command & Description
-------	-----------------------

1	<b>PSUBSCRIBE pattern [pattern ...]</b> Subscribes to channels matching the given patterns.
2	<b>PUBSUB subcommand [argument [argument ...]]</b> Tells the state of Pub/Sub system. For example, which clients are active on the server.
3	<b>PUBLISH channel message</b> Posts a message to a channel.
4	<b>PUNSUBSCRIBE [pattern [pattern ...]]</b> Stops listening for messages posted to channels matching the given patterns.
5	<b>SUBSCRIBE channel [channel ...]</b> Listens for messages published to the given channels.
6	<b>UNSUBSCRIBE [channel [channel ...]]</b> Stops listening for messages posted to the given channels.