Redis - Quick Guide

Previous Page Next Page

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 values can contain more complex data types, with atomic operations defined on those 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 - Environment

In this chapter, you will learn about the environmental setup for Redis.

Install Redis on Ubuntu

To install Redis on Ubuntu, go to the terminal and type the following commands –

```
$sudo apt-get update
$sudo apt-get install redis-server
```

This will install Redis on your machine.

Start Redis

```
$redis-server
```

Check If Redis is Working

```
$redis-cli
```

This will open a redis prompt.

```
redis 127.0.0.1:6379>
```

In the above prompt, **127.0.0.1** is your machine's IP address and **6379** is the port on which Redis server is running. Now type the following **PING** command.

```
redis 127.0.0.1:6379> ping
PONG
```

This shows that Redis is successfully installed on your machine.

Install Redis Desktop Manager on Ubuntu

To install Redis desktop manager on Ubuntu, just download the package from https://redisdesktop.com/download

Open the downloaded package and install it.

Redis desktop manager will give you UI to manage your Redis keys and data.

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 rabitmq
(integer) 3
redis 127.0.0.1:6379> lrange tutoriallist 0 10

1) "rabitmq"
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 rabitmq
(integer) 1
redis 127.0.0.1:6379> sadd tutoriallist rabitmq
(integer) 0
redis 127.0.0.1:6379> smembers tutoriallist

1) "rabitmq"
2) "mongodb"
3) "redis"
```

Note – In the above example, **rabitmq** 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 rabitmq
(integer) 1
redis 127.0.0.1:6379> zadd tutoriallist 0 rabitmq
(integer) 0
redis 127.0.0.1:6379> ZRANGEBYSCORE tutoriallist 0 1000
```

```
1) "redis"
2) "mongodb"
3) "rabitmq"
```

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	**DEL key** This command deletes the key, if it exists.
2	**DUMP key** This command returns a serialized version of the value stored at the specified key.
3	**EXISTS key** This command checks whether the key exists or not.
4	**EXPIRE key** seconds Sets the expiry of the key after the specified time.
5	**EXPIREAT key timestamp** Sets the expiry of the key after the specified time. Here time is in Unix timestamp format.
6	**PEXPIRE key milliseconds** Set the expiry of key in milliseconds.

7	**PEXPIREAT key milliseconds-timestamp** Sets the expiry of the key in Unix timestamp specified as milliseconds.
8	**KEYS pattern** Finds all keys matching the specified pattern.
9	**MOVE key db** Moves a key to another database.
10	**PERSIST key** Removes the expiration from the key.
11	**PTTL key** Gets the remaining time in keys expiry in milliseconds.
12	**TTL key** Gets the remaining time in keys expiry.
13	**RANDOMKEY** Returns a random key from Redis.
14	**RENAME key newkey** Changes the key name.
15	**RENAMENX key newkey** Renames the key, if a new key doesn't exist.
16	**TYPE key** 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	**SET key value** This command sets the value at the specified key.
2	**GET key** Gets the value of a key.
3	**GETRANGE key start end** Gets a substring of the string stored at a key.
4	**GETSET key value** Sets the string value of a key and return its old value.
5	**GETBIT key offset** Returns the bit value at the offset in the string value stored at the key.
6	**MGET key1 [key2..]** Gets the values of all the given keys
7	**SETBIT key offset value** Sets or clears the bit at the offset in the string value stored at the key
8	**SETEX key seconds value** Sets the value with the expiry of a key
9	**SETNX key value** Sets the value of a key, only if the key does not exist
10	**SETRANGE key offset value** Overwrites the part of a string at the key starting at the specified offset

11	**STRLEN key** Gets the length of the value stored in a key
12	**MSET key value [key value ...]** Sets multiple keys to multiple values
13	**MSETNX key value [key value ...]** Sets multiple keys to multiple values, only if none of the keys exist
14	**PSETEX key milliseconds value** Sets the value and expiration in milliseconds of a key
15	**INCR key** Increments the integer value of a key by one
16	**INCRBY key increment** Increments the integer value of a key by the given amount
17	**INCRBYFLOAT key increment** Increments the float value of a key by the given amount
18	**DECR key** Decrements the integer value of a key by one
19	**DECRBY key decrement** Decrements the integer value of a key by the given number
20	**APPEND key value** 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	**HDEL key field2 [field2]** Deletes one or more hash fields.
2	**HEXISTS key field** Determines whether a hash field exists or not.
3	**HGET key field** Gets the value of a hash field stored at the specified key.
4	**HGETALL key** Gets all the fields and values stored in a hash at the specified key
5	**HINCRBY key field increment** Increments the integer value of a hash field by the given number
6	**HINCRBYFLOAT key field increment** Increments the float value of a hash field by the given amount
7	**HKEYS key** Gets all the fields in a hash
8	**HLEN key** Gets the number of fields in a hash
9	**HMGET key field1 [field2]**

	Gets the values of all the given hash fields
10	**HMSET key field1 value1 [field2 value2]** Sets multiple hash fields to multiple values
11	**HSET key field value** Sets the string value of a hash field
12	**HSETNX key field value** Sets the value of a hash field, only if the field does not exist
13	**HVALS key** Gets all the values in a hash
14	**HSCAN key cursor [MATCH pattern] [COUNT count]** 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	**BLPOP key1 [key2] timeout** Removes and gets the first element in a list, or blocks until one is available
2	**BRPOP key1 [key2] timeout** Removes and gets the last element in a list, or blocks until one is available
3	**BRPOPLPUSH source destination timeout** Pops a value from a list, pushes it to another list and returns it; or blocks until one is available
4	**LINDEX key index** Gets an element from a list by its index
5	**LINSERT key BEFORE|AFTER pivot value** Inserts an element before or after another element in a list
6	**LLEN key** Gets the length of a list
7	**LPOP key** Removes and gets the first element in a list
8	**LPUSH key value1 [value2]** Prepends one or multiple values to a list
9	**LPUSHX key value** Prepends a value to a list, only if the list exists
10	**LRANGE key start stop** Gets a range of elements from a list
11	**LREM key count value** Removes elements from a list
12	**LSET key index value** Sets the value of an element in a list by its index
13	**LTRIM key start stop**

	Trims a list to the specified range
14	**RPOP key** Removes and gets the last element in a list
15	**RPOPLPUSH source destination** Removes the last element in a list, appends it to another list and returns it
16	**RPUSH key value1 [value2]** Appends one or multiple values to a list
17	**RPUSHX key value** 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 repition 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	**SADD key member1 [member2]** Adds one or more members to a set
2	**SCARD key** Gets the number of members in a set
3	**SDIFF key1 [key2]** Subtracts multiple sets
4	**SDIFFSTORE destination key1 [key2]** Subtracts multiple sets and stores the resulting set in a key
5	**SINTER key1 [key2]** Intersects multiple sets
6	**SINTERSTORE destination key1 [key2]** Intersects multiple sets and stores the resulting set in a key
7	**SISMEMBER key member** Determines if a given value is a member of a set
8	**SMEMBERS key** Gets all the members in a set
9	**SMOVE source destination member** Moves a member from one set to another
10	**SPOP key** Removes and returns a random member from a set
11	**SRANDMEMBER key [count]** Gets one or multiple random members from a set
12	**SREM key member1 [member2]** Removes one or more members from a set
13	**SUNION key1 [key2]** Adds multiple sets
14	**SUNIONSTORE destination key1 [key2]**

	Adds multiple sets and stores the resulting set in a key
15	**SSCAN key cursor [MATCH pattern] [COUNT count]** 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	**ZADD key score1 member1 [score2 member2]**

	Adds one or more members to a sorted set, or updates its score, if it already exists
2	**ZCARD key** Gets the number of members in a sorted set
3	**ZCOUNT key min max** Counts the members in a sorted set with scores within the given values
4	**ZINCRBY key increment member** Increments the score of a member in a sorted set
5	**ZINTERSTORE destination numkeys key [key ...]** Intersects multiple sorted sets and stores the resulting sorted set in a new key
6	**ZLEXCOUNT key min max** Counts the number of members in a sorted set between a given lexicographical range
7	**ZRANGE key start stop [WITHSCORES]** Returns a range of members in a sorted set, by index
8	**ZRANGEBYLEX key min max [LIMIT offset count]** Returns a range of members in a sorted set, by lexicographical range
9	**ZRANGEBYSCORE key min max [WITHSCORES] [LIMIT]** Returns a range of members in a sorted set, by score
10	**ZRANK key member** Determines the index of a member in a sorted set
11	**ZREM key member [member ...]** Removes one or more members from a sorted set
12	**ZREMRANGEBYLEX key min max** Removes all members in a sorted set between the given lexicographical range
13	**ZREMRANGEBYRANK key start stop** Removes all members in a sorted set within the given indexes
14	**ZREMRANGEBYSCORE key min max**

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

Redis - HyperLogLog

Redis HyperLogLog is an algorithm that uses randomization in order to provide an approximation of the number of unique elements in a set using just a constant, and small amount of memory.

HyperLogLog provides a very good approximation of the cardinality of a set even using a very small amount of memory around 12 kbytes per key with a standard error of 0.81%.

There is no limit to the number of items you can count, unless you approach 2^{64} items.

Example

Following example explains how Redis HyperLogLog works.

```
redis 127.0.0.1:6379> PFADD tutorials "redis"
1) (integer) 1
redis 127.0.0.1:6379> PFADD tutorials "mongodb"
1) (integer) 1
redis 127.0.0.1:6379> PFADD tutorials "mysql"
1) (integer) 1
```

```
redis 127.0.0.1:6379> PFCOUNT tutorials
(integer) 3
```

Redis HyperLogLog Commands

Following table lists some basic commands related to Redis HyperLogLog.

Sr.No	Command & Description
1	**PFADD key element [element ...]** Adds the specified elements to the specified HyperLogLog.
2	**PFCOUNT key [key ...]** Returns the approximated cardinality of the set(s) observed by the HyperLogLog at key(s).
3	**PFMERGE destkey sourcekey [sourcekey ...]** Merges N different HyperLogLogs into a single one.

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

Redis - Transactions

Redis transactions allow the execution of a group of commands in a single step. Following are the two properties of Transactions.

All commands in a transaction are sequentially executed as a single isolated operation. It is not possible that a request issued by another client is served in the middle of the execution of a Redis transaction.

Redis transaction is also atomic. Atomic means either all of the commands or none are processed.

Sample

Redis transaction is initiated by command **MULTI** and then you need to pass a list of commands that should be executed in the transaction, after which the entire transaction is executed by **EXEC** command.

```
redis 127.0.0.1:6379> MULTI
OK
List of commands here
redis 127.0.0.1:6379> EXEC
```

Example

Following example explains how Redis transaction can be initiated and executed.

```
redis 127.0.0.1:6379> MULTI
OK
redis 127.0.0.1:6379> SET tutorial redis
QUEUED
redis 127.0.0.1:6379> GET tutorial
QUEUED
redis 127.0.0.1:6379> INCR visitors
QUEUED
redis 127.0.0.1:6379> EXEC
1) OK
2) "redis"
3) (integer) 1
```

Redis Transaction Commands

Following table shows some basic commands related to Redis transactions.

Sr.No	Command & Description
1	**DISCARD** Discards all commands issued after MULTI
2	**EXEC** Executes all commands issued after MULTI
3	**MULTI** Marks the start of a transaction block
4	**UNWATCH** Forgets about all watched keys

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

Redis - Scripting

Redis scripting is used to evaluate scripts using the Lua interpreter. It is built into Redis starting from version 2.6.0. The command used for scripting is **EVAL** command.

Syntax

Following is the basic syntax of **EVAL** command.

```
redis 127.0.0.1:6379> EVAL script numkeys key [key ...] arg [arg ...]
```

Example

Following example explains how Redis scripting works.

```
redis 127.0.0.1:6379> EVAL "return {KEYS[1],KEYS[2],ARGV[1],ARGV[2]}" 2 key1
key2 first second
1) "key1"
2) "key2"
3) "first"
4) "second"
```

Redis Scripting Commands

Following table lists some basic commands related to Redis Scripting.

Sr.No	Command & Description
1	**EVAL script numkeys key [key ...] arg [arg ...]** Executes a Lua script.
2	**EVALSHA sha1 numkeys key [key ...] arg [arg ...]** Executes a Lua script.
3	**SCRIPT EXISTS script [script ...]** Checks the existence of scripts in the script cache.
4	**SCRIPT FLUSH** Removes all the scripts from the script cache.

5	**SCRIPT KILL** Kills the script currently in execution.
6	**SCRIPT LOAD script** Loads the specified Lua script into the script cache.

Redis - Connections

Redis connection commands are basically used to manage client connections with Redis server.

Example

Following example explains how a client authenticates itself to Redis server and checks whether the server is running or not.

```
redis 127.0.0.1:6379> AUTH "password"
OK
redis 127.0.0.1:6379> PING
PONG
```

Redis Connection Commands

Following table lists some basic commands related to Redis connections.

Sr.No	Command & Description
1	**AUTH password** Authenticates to the server with the given password
2	**ECHO message** Prints the given string
3	**PING** Checks whether the server is running or not
4	**QUIT** Closes the current connection
5	**SELECT index** Changes the selected database for the current connection

Redis - Server

Redis server commands are basically used to manage Redis server.

Example

Following example explains how we can get all statistics and information about the server.

```
redis 127.0.0.1:6379> INFO

# Server
redis_version:2.8.13
redis_git_sha1:00000000
redis_git_dirty:0
redis_build_id:c2238b38b1edb0e2
redis_mode:standalone
os:Linux 3.5.0-48-generic x86_64
arch_bits:64
multiplexing_api:epoll
gcc_version:4.7.2
process_id:3856
run_id:0e61abd297771de3fe812a3c21027732ac9f41fe
tcp_port:6379
uptime_in_seconds:11554
uptime_in_days:0 hz:10
lru_clock:16651447
config_file:

# Clients
connected_clients:1
client_longest_output_list:0
client_biggest_input_buf:0
blocked_clients:0

# Memory
used_memory:589016
used_memory_human:575.21K
used_memory_rss:2461696
used_memory_peak:667312
used_memory_peak_human:651.67K
used_memory_lua:33792
mem_fragmentation_ratio:4.18
mem_allocator:jemalloc-3.6.0

# Persistence
loading:0
rdb_changes_since_last_save:3
```

```
rdb_bgsave_in_progress:0
rdb_last_save_time:1409158561
rdb_last_bgsave_status:ok
rdb_last_bgsave_time_sec:0
rdb_current_bgsave_time_sec:-1
aof_enabled:0
aof_rewrite_in_progress:0
aof_rewrite_scheduled:0
aof_last_rewrite_time_sec:-1
aof_current_rewrite_time_sec:-1
aof_last_bgrewrite_status:ok
aof_last_write_status:ok

# Stats
total_connections_received:24
total_commands_processed:294
instantaneous_ops_per_sec:0
rejected_connections:0
sync_full:0
sync_partial_ok:0
sync_partial_err:0
expired_keys:0
evicted_keys:0
keyspace_hits:41
keyspace_misses:82
pubsub_channels:0
pubsub_patterns:0
latest_fork_usec:264

# Replication
role:master
connected_slaves:0
master_repl_offset:0
repl_backlog_active:0
repl_backlog_size:1048576
repl_backlog_first_byte_offset:0
repl_backlog_histlen:0

# CPU
used_cpu_sys:10.49
used_cpu_user:4.96
used_cpu_sys_children:0.00
used_cpu_user_children:0.01
```

```
# Keyspace
db0:keys = 94,expires = 1,avg_ttl = 41638810
db1:keys = 1,expires = 0,avg_ttl = 0
db3:keys = 1,expires = 0,avg_ttl = 0
```

Redis Server Commands

Following table lists some basic commands related to Redis server.

Sr.No	Command & Description
1	**BGREWRITEAOF** Asynchronously rewrites the append-only file
2	**BGSAVE** Asynchronously saves the dataset to the disk
3	**CLIENT KILL [ip:port] [ID client-id]** Kills the connection of a client
4	**CLIENT LIST** Gets the list of client connections to the server
5	**CLIENT GETNAME** Gets the name of the current connection
6	**CLIENT PAUSE timeout** Stops processing commands from the clients for a specified time
7	**CLIENT SETNAME connection-name** Sets the current connection name
8	**CLUSTER SLOTS** Gets an array of Cluster slot to node mappings
9	**COMMAND** Gets an array of Redis command details
10	**COMMAND COUNT** Gets total number of Redis commands
11	**COMMAND GETKEYS** Extracts the keys given a full Redis command

12	**BGSAVE** Asynchronously saves the dataset to the disk
13	**COMMAND INFO command-name [command-name ...]** Gets an array of specific Redis command details
14	**CONFIG GET parameter** Gets the value of a configuration parameter
15	**CONFIG REWRITE** Rewrites the configuration file with the in-memory configuration
16	**CONFIG SET parameter value** Sets a configuration parameter to the given value
17	**CONFIG RESETSTAT** Resets the stats returned by INFO
18	**DBSIZE** Returns the number of keys in the selected database
19	**DEBUG OBJECT key** Gets debugging information about a key
20	**DEBUG SEGFAULT** Makes the server crash
21	**FLUSHALL** Removes all the keys from all databases
22	**FLUSHDB** Removes all the keys from the current database
23	**INFO [section]** Gets information and statistics about the server
24	**LASTSAVE** Gets the UNIX time stamp of the last successful save to the disk
25	**MONITOR** Listens for all the requests received by the server in real time

26	**ROLE** Returns the role of the instance in the context of replication
27	**SAVE** Synchronously saves the dataset to the disk
28	**SHUTDOWN [NOSAVE] [SAVE]** Synchronously saves the dataset to the disk and then shuts down the server
29	**SLAVEOF host port** Makes the server a slave of another instance, or promotes it as a master
30	**SLOWLOG subcommand [argument]** Manages the Redis slow queries log
31	**SYNC** Command used for replication
32	**TIME** Returns the current server time

Redis - Backup

Redis **SAVE** command is used to create a backup of the current Redis database.

Syntax

Following is the basic syntax of redis **SAVE** command.

```
127.0.0.1:6379> SAVE
```

Example

Following example creates a backup of the current database.

```
127.0.0.1:6379> SAVE
OK
```

This command will create a **dump.rdb** file in your Redis directory.

Restore Redis Data

To restore Redis data, move Redis backup file (dump.rdb) into your Redis directory and start the server. To get your Redis directory, use **CONFIG** command of Redis as shown

below.

```
127.0.0.1:6379> CONFIG get dir
1) "dir"
2) "/user/tutorialspoint/redis-2.8.13/src"
```

In the output of the above command **/user/tutorialspoint/redis-2.8.13/src** is the directory, where Redis server is installed.

Bgsave

To create Redis backup, an alternate command **BGSAVE** is also available. This command will start the backup process and run this in the background.

Example

```
127.0.0.1:6379> BGSAVE
Background saving started
```

Redis - Security

Redis database can be secured, such that any client making a connection needs to authenticate before executing a command. To secure Redis, you need to set the password in the config file.

Example

Following example shows the steps to secure your Redis instance.

```
127.0.0.1:6379> CONFIG get requirepass
1) "requirepass"
2) ""
```

By default, this property is blank, which means no password is set for this instance. You can change this property by executing the following command.

```
127.0.0.1:6379> CONFIG set requirepass "tutorialspoint"
OK
127.0.0.1:6379> CONFIG get requirepass
1) "requirepass"
2) "tutorialspoint"
```

After setting the password, if any client runs the command without authentication, then **(error) NOAUTH Authentication required**. error will return. Hence, the client needs to use **AUTH** command to authenticate himself.

Syntax

Following is the basic syntax of **AUTH** command.

```
127.0.0.1:6379> AUTH password
```

Example

```
127.0.0.1:6379> AUTH "tutorialspoint"
OK
127.0.0.1:6379> SET mykey "Test value"
OK
127.0.0.1:6379> GET mykey
"Test value"
```

Redis - Benchmarks

Redis benchmark is the utility to check the performance of Redis by running n commands simultaneously.

Syntax

Following is the basic syntax of Redis benchmark.

```
redis-benchmark [option] [option value]
```

Example

Following example checks Redis by calling 100000 commands.

```
redis-benchmark -n 100000

PING_INLINE: 141043.72 requests per second
PING_BULK: 142857.14 requests per second
SET: 141442.72 requests per second
GET: 145348.83 requests per second
INCR: 137362.64 requests per second
LPUSH: 145348.83 requests per second
LPOP: 146198.83 requests per second
SADD: 146198.83 requests per second
SPOP: 149253.73 requests per second
LPUSH (needed to benchmark LRANGE): 148588.42 requests per second
LRANGE_100 (first 100 elements): 58411.21 requests per second
LRANGE_300 (first 300 elements): 21195.42 requests per second
LRANGE_500 (first 450 elements): 14539.11 requests per second
LRANGE_600 (first 600 elements): 10504.20 requests per second
MSET (10 keys): 93283.58 requests per second
```

Following is a list of available options in Redis benchmark.

Sr.No	Option	Description	Default Value
1	**-h**	Specifies server host name	127.0.0.1
2	**-p**	Specifies server port	6379
3	**-s**	Specifies server socket	
4	**-c**	Specifies the number of parallel connections	50
5	**-n**	Specifies the total number of requests	10000
6	**-d**	Specifies data size of SET/GET value in bytes	2
7	**-k**	1=keep alive, 0=reconnect	1
8	**-r**	Use random keys for SET/GET/INCR, random values for SADD	
9	**-p**	Pipeline <numreq> requests	1
10	**-h**	Specifies server host name	
11	**-q**	Forces Quiet to Redis. Just shows query/sec values	
12	**--csv**	Output in CSV format	
13	**-l**	Generates loop, Run the tests forever	
14	**-t**	Only runs the comma-separated list of tests	
15	**-I**	Idle mode. Just opens N idle connections and wait	

Example

Following example shows the multiple usage options in Redis benchmark utility.

```
redis-benchmark -h 127.0.0.1 -p 6379 -t set,lpush -n 100000 -q

SET: 146198.83 requests per second
LPUSH: 145560.41 requests per second
```

Redis - Client Connection

Redis accepts clients' connections on the configured listening TCP port and on the Unix socket, if enabled. When a new client connection is accepted, the following operations are performed –

The client socket is put in non-blocking state since Redis uses multiplexing and non-blocking I/O.

The TCP_NODELAY option is set in order to ensure that we don't have delays in our connection.

A readable file event is created so that Redis is able to collect the client queries as soon as new data is available to be read on the socket.

Maximum Number of Clients

In Redis config (redis.conf), there is a property called **maxclients**, which describes the maximum number of clients that can connect to Redis.

Following is the basic syntax of command.

```
config get maxclients


1) "maxclients"
2) "10000"
```

By default, this property is set to 10000 (depending upon the maximum number of file descriptors limit of OS), although you can change this property.

Example

In the following example, we have set the maximum number of clients to 100000, while starting the server.

```
redis-server --maxclients 100000
```

Client Commands

Sr.No	Command	Description
1	**CLIENT LIST**	Returns the list of clients connected to Redis server
2	**CLIENT SETNAME**	Assigns a name to the current connection
3	**CLIENT GETNAME**	Returns the name of the current connection as set by CLIENT SETNAME
4	**CLIENT PAUSE**	This is a connections control command able to suspend all the Redis clients for the specified amount of time (in milliseconds)
5	**CLIENT KILL**	This command closes a given client connection.

Redis - Pipelining

Redis is a TCP server and supports request/response protocol. In Redis, a request is accomplished with the following steps –

The client sends a query to the server, and reads from the socket, usually in a blocking way, for the server response.

The server processes the command and sends the response back to the client.

Meaning of Pipelining

The basic meaning of pipelining is, the client can send multiple requests to the server without waiting for the replies at all, and finally reads the replies in a single step.

Example

To check the Redis pipelining, just start the Redis instance and type the following command in the terminal.

```
$(echo -en "PING\r\n SET tutorial redis\r\nGET tutorial\r\nINCR
visitor\r\nINCR visitor\r\nINCR visitor\r\n"; sleep 10) | nc localhost 6379
+PONG
+OK
redis
:1
:2
:3
```

In the above example, we will check Redis connection by using **PING** command. We have set a string named **tutorial** with value **redis**. Later, we get that keys value and increment the visitor number three times. In the result, we can see that all commands are submitted to Redis once, and Redis provides the output of all commands in a single step.

Benefits of Pipelining

The benefit of this technique is a drastically improved protocol performance. The speedup gained by pipelining ranges from a factor of five for connections to localhost up to a factor of at least one hundred over slower internet connections.

Redis - Partitioning

Partitioning is the process of splitting your data into multiple Redis instances, so that every instance will only contain a subset of your keys.

Benefits of Partitioning

It allows for much larger databases, using the sum of the memory of many computers. Without partitioning you are limited to the amount of memory that a single computer can support.

It allows to scale the computational power to multiple cores and multiple computers, and the network bandwidth to multiple computers and network adapters.

Disadvantages of Partitioning

Operations involving multiple keys are usually not supported. For instance, you can't perform the intersection between two sets if they are stored in the keys that are mapped to different Redis instances.

Redis transactions involving multiple keys cannot be used.

The partitioning granuliary is the key, so it is not possible to shard a dataset with a single huge key like a very big sorted set.

When partitioning is used, data handling is more complex. For instance, you have to handle multiple RDB/AOF files, and to get a backup of your data you need to aggregate the persistence files from multiple instances and hosts.

Adding and removing the capacity can be complex. For instance, Redis Cluster supports mostly transparent rebalancing of data with the ability to add and remove nodes at runtime. However, other systems like client-side partitioning and proxies don't support this feature. A technique called **Presharding** helps in this regard.

Types of Partitioning

There are two types of partitioning available in Redis. Suppose we have four Redis instances, R0, R1, R2, R3 and many keys representing users like user:1, user:2, ... and so forth.

Range Partitioning

Range partitioning is accomplished by mapping ranges of objects into specific Redis instances. Suppose in our example, the users from ID 0 to ID 10000 will go into instance R0, while the users from ID 10001 to ID 20000 will go into instance R1 and so forth.

Hash Partitioning

In this type of partitioning, a hash function (eg. modulus function) is used to convert the key into a number and then the data is stored in different-different Redis instances.

Redis - Java

Before you start using Redis in your Java programs, you need to make sure that you have Redis Java driver and Java set up on the machine. You can check our Java tutorial for Java installation on your machine.

Installation

Now, let us see how to set up Redis Java driver.

> You need to download the jar from the path **Download jedis.jar** . Make sure to download the latest release of it.
>
> You need to include the **jedis.jar** into your classpath.

Connect to Redis Server

```
import redis.clients.jedis.Jedis;

public class RedisJava {
   public static void main(String[] args) {
      //Connecting to Redis server on localhost
      Jedis jedis = new Jedis("localhost");
      System.out.println("Connection to server sucessfully");
      //check whether server is running or not
      System.out.println("Server is running: "+jedis.ping());
   }
}
```

Now, let's compile and run the above program to test the connection to Redis server. You can change your path as per your requirement. We are assuming the current version of **jedis.jar** is available in the current path.

```
$javac RedisJava.java
$java RedisJava
Connection to server sucessfully
Server is running: PONG
```

Redis Java String Example

```
import redis.clients.jedis.Jedis;

public class RedisStringJava {
   public static void main(String[] args) {
      //Connecting to Redis server on localhost
      Jedis jedis = new Jedis("localhost");
      System.out.println("Connection to server sucessfully");
      //set the data in redis string
      jedis.set("tutorial-name", "Redis tutorial");
      // Get the stored data and print it
      System.out.println("Stored string in redis:: "+ jedis.get("tutorialname"));
   }
}
```

Now, let's compile and run the above program.

```
$javac RedisStringJava.java
$java RedisStringJava
Connection to server sucessfully
Stored string in redis:: Redis tutorial
```

Redis Java List Example

```
import redis.clients.jedis.Jedis;

public class RedisListJava {
   public static void main(String[] args) {

      //Connecting to Redis server on localhost
      Jedis jedis = new Jedis("localhost");
      System.out.println("Connection to server sucessfully");

      //store data in redis list
      jedis.lpush("tutorial-list", "Redis");
      jedis.lpush("tutorial-list", "Mongodb");
      jedis.lpush("tutorial-list", "Mysql");
      // Get the stored data and print it
      List<String> list = jedis.lrange("tutorial-list", 0 ,5);

      for(int i = 0; i<list.size(); i++) {
         System.out.println("Stored string in redis:: "+list.get(i));
      }
   }
}
```

Now, let's compile and run the above program.

```
$javac RedisListJava.java
$java RedisListJava
Connection to server sucessfully
Stored string in redis:: Redis
Stored string in redis:: Mongodb
Stored string in redis:: Mysql
```

Redis Java Keys Example

```
import redis.clients.jedis.Jedis;

public class RedisKeyJava {
   public static void main(String[] args) {

      //Connecting to Redis server on localhost
      Jedis jedis = new Jedis("localhost");
      System.out.println("Connection to server sucessfully");
      //store data in redis list
      // Get the stored data and print it
      List<String> list = jedis.keys("*");
```

```
      for(int i = 0; i<list.size(); i++) {
         System.out.println("List of stored keys:: "+list.get(i));
      }
   }
}
```

Now, let's compile and run the above program.

```
$javac RedisKeyJava.java
$java RedisKeyJava
Connection to server sucessfully
List of stored keys:: tutorial-name
List of stored keys:: tutorial-list
```

Redis - PHP

Before you start using Redis in your PHP programs, you need to make sure that you have Redis PHP driver and PHP set up on the machine. You can check PHP tutorial for PHP installation on your machine.

Installation

Now, let us check how to set up Redis PHP driver.

You need to download the phpredis from github repository https://github.com/nicolasff/phpredis . Once you've downloaded it, extract the files to phpredis directory. On Ubuntu, install the following extension.

```
cd phpredis
sudo phpize
sudo ./configure
sudo make
sudo make install
```

Now, copy and paste the content of "modules" folder to the PHP extension directory and add the following lines in **php.ini**.

```
extension = redis.so
```

Now, your Redis PHP installation is complete

Connect to Redis Server

```
<?php
   //Connecting to Redis server on localhost
   $redis = new Redis();
   $redis->connect('127.0.0.1', 6379);
   echo "Connection to server sucessfully";
   //check whether server is running or not
```

```
   echo "Server is running: ".$redis->ping();
?>
```

When the program is executed, it will produce the following result.

```
Connection to server sucessfully
Server is running: PONG
```

Redis PHP String Example

```
<?php
   //Connecting to Redis server on localhost
   $redis = new Redis();
   $redis->connect('127.0.0.1', 6379);
   echo "Connection to server sucessfully";
   //set the data in redis string
   $redis->set("tutorial-name", "Redis tutorial");
   // Get the stored data and print it
   echo "Stored string in redis:: " .$redis→get("tutorial-name");
?>
```

When the above program is executed, it will produce the following result.

```
Connection to server sucessfully
Stored string in redis:: Redis tutorial
```

Redis php List Example

```
<?php
   //Connecting to Redis server on localhost
   $redis = new Redis();
   $redis->connect('127.0.0.1', 6379);
   echo "Connection to server sucessfully";
   //store data in redis list
   $redis->lpush("tutorial-list", "Redis");
   $redis->lpush("tutorial-list", "Mongodb");
   $redis->lpush("tutorial-list", "Mysql");

   // Get the stored data and print it
   $arList = $redis->lrange("tutorial-list", 0 ,5);
   echo "Stored string in redis:: ";
   print_r($arList);
?>
```

When the above program is executed, it will produce the following result.

```
Connection to server sucessfully
Stored string in redis::
Redis
Mongodb
Mysql
```

Redis PHP Keys Example

```php
<?php
   //Connecting to Redis server on localhost
   $redis = new Redis();
   $redis->connect('127.0.0.1', 6379);
   echo "Connection to server sucessfully";
   // Get the stored keys and print it
   $arList = $redis->keys("*");
   echo "Stored keys in redis:: "
   print_r($arList);
?>
```

When the program is executed, it will produce the following result.

```
Connection to server sucessfully
Stored string in redis::
tutorial-name
tutorial-list
```

Previous Page Next Page

Enter email for newsletter go