**FREE eBook**

# LEARNING redis

Free unaffiliated eBook created from
**Stack Overflow contributors.**

#redis

# Table of Contents

# About

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: redis

It is an unofficial and free redis ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official redis.



Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

# Chapter 1: Getting started with redis

## Remarks

This section provides an overview of what Redis is, and why a developer might want to use it.

It should also mention any large subjects within Redis, and link out to the related topics. Since the documentation for Redis is new, you may need to create initial versions of those related topics.

## Versions

| Version | Release Date |
|---|---|
| 3.2.3 | 2016-08-02 |
| 3.2.2 | 2016-07-28 |

## Examples

### Overview

Redis is an in-memory remote database that offers high performance, replication, and a unique data model to produce a platform for solving problems. Redis is an open source (BSD licensed), in-memory data structure , used as database, cache and message broker. It is categorized as a NoSQL key-value store. It supports data structures such as strings, hashes, lists, sets, sorted sets with range queries, bitmaps, hyperloglogs and geospatial indexes with radius queries. Supporting five different types of data structures,

1. STRING (Operate on the whole string, parts, integers and floats)
2. LIST (Push or pop items from both ends)
3. SET (Add, fetch, remove, check, intersect, union, difference etc)
4. HASH (store, fatch, remove in hash)
5. ZSET (same as set but in ordered way)
6. GEO (Add, update, delete latitude and longitude, get within given redius)

Redis has built-in replication, Lua scripting, LRU eviction, transactions and different levels of on-disk persistence(sync/async).

Prior to version 3, Redis works in master-slave mode and required Redis-Sentinel to provide high-availability.Only master accepts writes and syncs data to its slaves by forking.

From version 3, Redis works & recommends multi-master mode where failover, sharding/paritioning, resharding features are in-built. Redis-Sentinel is not required from version-3. In order for the redis cluster to operate a minimum of 3 master nodes/processes are required.

Additional features are replication, persistence, and client-side sharding. Redis accommodates a wide variety of problems that can be naturally mapped into what Redis offers, allowing you to solve your problems without having to perform the conceptual work required by other databases.

## Redis command line interface

`redis-cli` is the Redis command line interface program that allows to send commands to Redis and read the replies sent by the server, directly from the terminal. Basic command line usage is below:

Access to redis:

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

Access to redis with authentication:

```
$ redis-cli -a myPassword
127.0.0.1:6379>
```

Select database and show database size (default database number is 0):

```
127.0.0.1:6379> dbsize
(integer) 2
127.0.0.1:6379> select 1
OK
127.0.0.1:6379[1]> dbsize
(integer) 20
```

Get information and statistics about the server:

```
127.0.0.1:6379> info
redis_version:2.4.10
redis_git_sha1:00000000
redis_git_dirty:0
arch_bits:64
multiplexing_api:epoll
gcc_version:4.4.6
process_id:947
uptime_in_seconds:873394
uptime_in_days:10
lru_clock:118108
used_cpu_sys:19.55
used_cpu_user:397.46
used_cpu_sys_children:0.00
used_cpu_user_children:0.00
connected_clients:1
connected_slaves:0
client_longest_output_list:0
client_biggest_input_buf:0
blocked_clients:0
used_memory:14295792
used_memory_human:13.63M
used_memory_rss:19853312
```

```
used_memory_peak:14295760
used_memory_peak_human:13.63M
mem_fragmentation_ratio:1.39
mem_allocator:jemalloc-2.2.5
loading:0
aof_enabled:0
changes_since_last_save:0
bgsave_in_progress:0
last_save_time:1468314087
bgrewriteaof_in_progress:0
total_connections_received:2
total_commands_processed:2
expired_keys:0
evicted_keys:0
keyspace_hits:0
keyspace_misses:0
pubsub_channels:0
pubsub_patterns:0
latest_fork_usec:0
vm_enabled:0
role:master
db0:keys=2,expires=0
db1:keys=20,expires=0
```

Exiting from the redis-cli:

```
127.0.0.1:6379> exit
```

## Redis "Hello World"

First you need to install and start your Redis server, check the link below that can help you to install redis on you server or local machine.

### Installation and Setup

Now open your command prompt and run command `redis-cli` :

To save first set >SET 'keyname' then 'value'

```
127.0.0.1:6379> SET hkey "Hello World!"
```

Press Enter you should see

```
OK
```

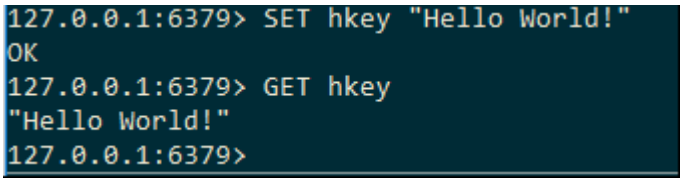
Then enter:

```
GET hkey
```

you should see:

```
"Hello World!"
```

Screen output example:




## Install Redis by using Docker

It is simple to start using Redis using docker:

```
docker pull redis
docker run -p 6379:6379 --rm --name redis redis
```

Now you have running instance on port `6397`

*Attention:* All data will be deleted, when Redis will be stopped.

To connect the redis-cli, start another docker:

```
docker run -it --link redis:redis --rm redis redis-cli -h redis -p 6379
```

Now you can play around with your redis docker.

## Redis installtion on Windows, with Node.js example

Redis has a Windows port provided by 'Microsoft Open Technologies'. You can use the msi installer found on: https://github.com/MSOpenTech/redis/releases

After installation completes you can see 'Redis' is a Windows service (and it's status should be "Started")

To write an 'Hello world' example that uses Redis in Node.js (in windows as well) you can use the following npm module : https://www.npmjs.com/package/redis

code sample:

```
var redis = require('redis'),
    client = redis.createClient();

client.set('mykey', 'Hello World');
client.get('mykey', function(err,res){
    console.log(res);
});
```

Read Getting started with redis online: https://riptutorial.com/redis/topic/1724/getting-started-with-redis

# Chapter 2: Backup

## Introduction

Backing up a remote Redis instance can be achieved with replication. This is useful if you want to take a snapshot of a dataset prior to upgrading, deleting or changing a Redis database.

## Examples

### Backup of a remote Redis instance to a local instance

On the machine where you'd like to make the backup, jump to the Redis CLI:

```
redis-cli
```

## Password?

If your master Redis DB (the one you want to replicate) has a password:

```
config set masterauth <password>
```

## Start replication

Run the following to begin replication:

```
SLAVEOF <host> <port>
```

To check the replication is underway run:

```
INFO replication
```

And you should see output like this:

```
# Replication
role:slave
master_host:some-host.compute-1.amazonaws.com
master_port:6519
master_link_status:up
master_last_io_seconds_ago:3
master_sync_in_progress:0
slave_repl_offset:35492914
slave_priority:100
slave_read_only:1
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
```

Note the `master_link_status` should be `up`.

## Checking sync progress

When the sync is complete, the `INFO replication` should show:

```
master_sync_in_progress:0
```

To check the dataset has been synced you could compare the size of the database:

```
DBSIZE
```

## Saving a data dump to disk

To save the DB to disk asynchronously:

```
BGSAVE
CONFIG GET dir
```

Then you should find a `dump.rdb` file in the directory listed by the config command.

## Halting replication

You can stop replication with:

```
SLAVEOF NO ONE
```

---

Reference: Redis replication guide

Read Backup online: https://riptutorial.com/redis/topic/9369/backup

# Chapter 3: Connecting to redis using Python

## Introduction

Connecting to Redis in Python requires the use of a client library. Many different client libraries exist for Python, but **redis-py** is one of the most popular clients in use.

Once you install your client library, you can then access Redis in your application by importing the appropriate module, establishing a connection, then executing a command.

## Remarks

To connect on redis with python you need to install a **client**. You can install with pip using:

```
pip install redis
```

this will install **redis-py**

Optionally, you may want to install **hiredis-py** which delegates parsing of protocol messages to the C hiredis client. This can provide significant performance improvement in many situations. You can install hiredis with pip by executing:

```
pip install hiredis
```

## Examples

### Add element to list

```
import redis

r = redis.StrictRedis(host='localhost', port=6379, db=0)

r.lpush('myqueue','myelement')
```

### Adding fields to a Hash

There are two main functions in Redis (HSET and HMSET) for adding fields to a hash key. Both functions are available in redis-py.

Using HSET:

```
import redis

r = redis.StrictRedis(host='myserver', port=6379, db=0)
r.hset('my_key', 'field0', 'value0')
```

Using HMSET:

```
import redis

r = redis.StrictRedis(host='myserver', port=6379, db=0)
r.hmset('my_key', {'field0': 'value0', 'field1':'value1', 'field2':'value2'}
```

## Setting up a Connection to Redis

The **redis-py** client provides two classes `StrictRedis` and `Redis` to establish a basic connection to a Redis database. The `Redis` class is provided for backwards compatibility and new projects should use the `StrictRedis` class.

One of the recommended ways to establish a connection, is to define the connection parameters in a dictionary and pass the dictionary to the `StrictRedis` constructor using the `**` syntax.

```
conn_params = {
    "host": "myredis.somedomain.com",
    "port": 6379,
    "password": "sekret",
    "db": 0
}

r = redis.StrictRedis(**config)
```

## Creating a transaction

You can establish a transaction by calling the `pipeline` method on the `StrictRedis`. Redis commands executed against the transaction are performed in a single block.

```
# defaults to transaction=True
tx = r.pipeline()
tx.hincrbyfloat(debit_account_key, 'balance', -amount)
tx.hincrbyfloat(credit_account_key, 'balance', amount)
tx.execute()
```

## Executing Commands Directly

Redis-py provides the `execute_command` method to directly invoke Redis operations. This functionality can be used to access any modules that may not have a supported interface in the redis-py client. For example, you can use the `execute_command` to list all of the modules loaded into a Redis server:

```
r.execute_command('MODULE', 'LIST')
```

Read Connecting to redis using Python online: https://riptutorial.com/redis/topic/9103/connecting-to-redis-using-python

# Chapter 4: Geo

## Introduction

Redis provides the GEO datatype to work with geospatial indexed data.

## Syntax

- GEOADD key longitude latitude member [longitude latitude member ...]
- GEODIST key member1 member2 [unit]

## Examples

### GEOADD

The GEOADD command allows a user to add geospatial information (item name, longitude, latitude) to a particular key.

The GEOADD command can be used to add a single item to a key:

```
GEOADD meetup_cities -122.43 37.77 "San Francisco"
```

or multiple items to a key:

```
GEOADD meetup_cities -122.43 37.77 "San Francisco" -104.99 39.74 "Denver"
```

### GEODIST

The GEODIST command allows a user to determine the distance between two members within a geospatial index while specifying the units.

To find the distance between two meetup cities:

```
GEODIST meetup_cities "San Francisco" "Denver" mi
```

Read Geo online: https://riptutorial.com/redis/topic/9091/geo

# Chapter 5: How to Connect to Redis in Java using Jedis

## Introduction

There are more than ten different client libraries to use with Redis in Java. One of the most popular clients is Jedis.

## Remarks

Further information:

- Java Redis Clients
- Jedis Github Repository
- Jedis Documentation/Wiki

## Examples

### Getting Jedis

The Jedis library is generally added to Java project using a dependency management system built into the build environment of the project. Two popular Java build systems are Maven and Gradle.

#### Using Gradle

To add the Jedis library to a Gradle project, you will need configure a repository and add a dependency. The following snippet shows how to add version 2.9.0 of the Jedis library to a Gradle project.

```
repositories {
    mavenCentral()
}

dependencies {
    compile 'redis.clients:jedis:2.9.0'
}
```

#### Using Maven

To add Jedis to a Maven project, you need to add a dependency to your dependency list and provide the coordinates of the library. The following snippet would be added to your pom.xml file:

```
<dependencies>
    <dependency>
        <groupId>redis.clients</groupId>
        <artifactId>jedis</artifactId>
```

```
        <version>2.9.0</version>
    </dependency>
</dependencies>
```

## Connecting to Redis

### Using a Pool

Most code will want to connect to Redis using a pool of shared connection objects. Connecting to Redis using a pool involves two different code block. At initialization time, your application needs to create the connection pool:

```
    JedisPoolConfig poolCfg = new JedisPoolConfig();
    poolCfg.setMaxTotal(3);

    pool = new JedisPool(poolCfg, hostname, port, 500, password, false);
```

The `JedisPoolConfig` provides options for tuning the pool.

As your application processes it's workload, you will need to get a connection from the shared pool using the following code:

```
    try (Jedis jedis = pool.getResource()) {

        ...
    }
```

Best practice is to get the `Jedis` connection object from the pool within a try-with-resources block.

### Without Pools

In some cases, such as a simple application or an integration test, you may not want to deal with shared pools and instead create the `Jedis` connection object directly. That can be accomplished with the following code:

```
try (Jedis jedis = new Jedis(hostname, port)) {
    jedis.connect();
    jedis.auth(password);
    jedis.select(db);

    . . .
}
```

Again, best practice is to create the Jedis client object within a try-with-resources block.

## Executing Basic Get/Set Commands

Once you have established a connection to Redis you can get and set values using the `Jedis` connection object:

### Get

```
String value = jedis.get(myKey);
```

**Set**

```
jedis.put(myKey, "some value");
```

## Executing Commands

To execute a Redis command using Jedis, you make method calls against the `Jedis` object you created from the pool. Jedis exposes Redis commands as method calls, some example are:

```
- String get(String key)
- Long geoadd(String key, double longitude, double latitude, String member)
- List<String> hmget(String key, String... fields)
- Long hsetnx(String key, String field, String value)
```

If you wanted to set the value of a String key in Redis you would use a code block similar to:

```
try (Jedis jedis = pool.getResource()) {

   String myKey = "users:20";
   String myValue = "active";

   jedis.set(myKey, myValue);
}
```

Read How to Connect to Redis in Java using Jedis online:
https://riptutorial.com/redis/topic/9712/how-to-connect-to-redis-in-java-using-jedis

# Chapter 6: Installation and Setup

## Examples

### Installing Redis

```
wget http://download.redis.io/redis-stable.tar.gz
tar xvzf redis-stable.tar.gz
cd redis-stable
make
```

### Starting Redis

```
redis-server
```

### Check if Redis is working

```
redis-cli ping
```

This should return `PONG`

### Access Redis Cli

Assuming that you are running redis server on localhost you can type command

```
redis-cli
```

After this command appear redis command line prompt

```
127.0.0.1:6379>
```

### Redis data types

The following is the list of all the data structures supported by Redis:

- **Binary-safe strings**
- **Lists**: collections of string elements sorted according to the order of insertion.
- **Sets**: collections of unique, unsorted string elements.
- **Sorted sets**: similar to Sets but where every string element is associated to a floating number value, called score.
- **Hashes**: are maps composed of fields associated with values.
- **HyperLogLogs**: this is a probabilistic data structure which is used in order to estimate the cardinality of a set.

*Based on **redis.io** official documentation*

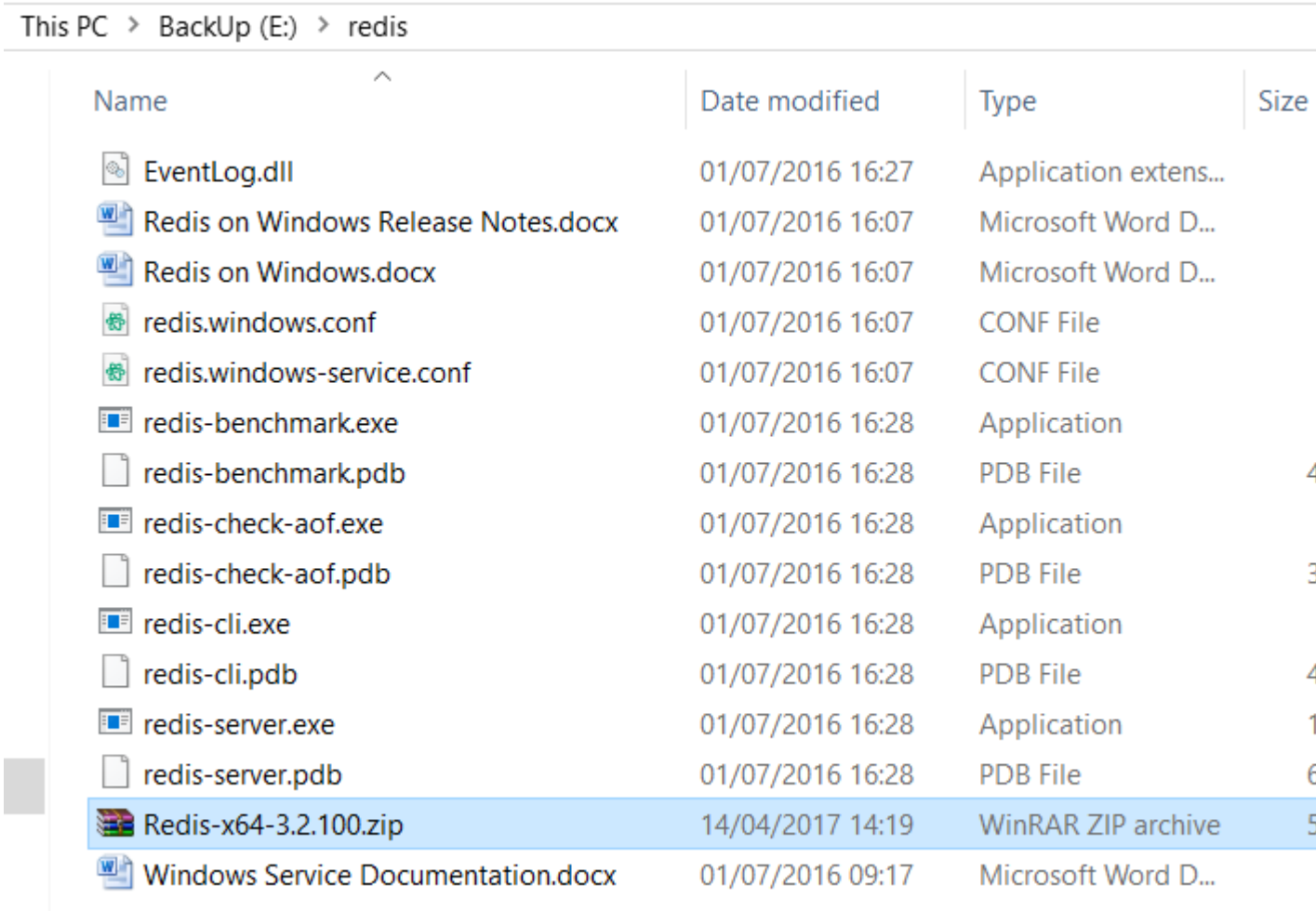
## Installing and running Redis Server on Windows

*Note:* The Redis project does not officially support Windows.

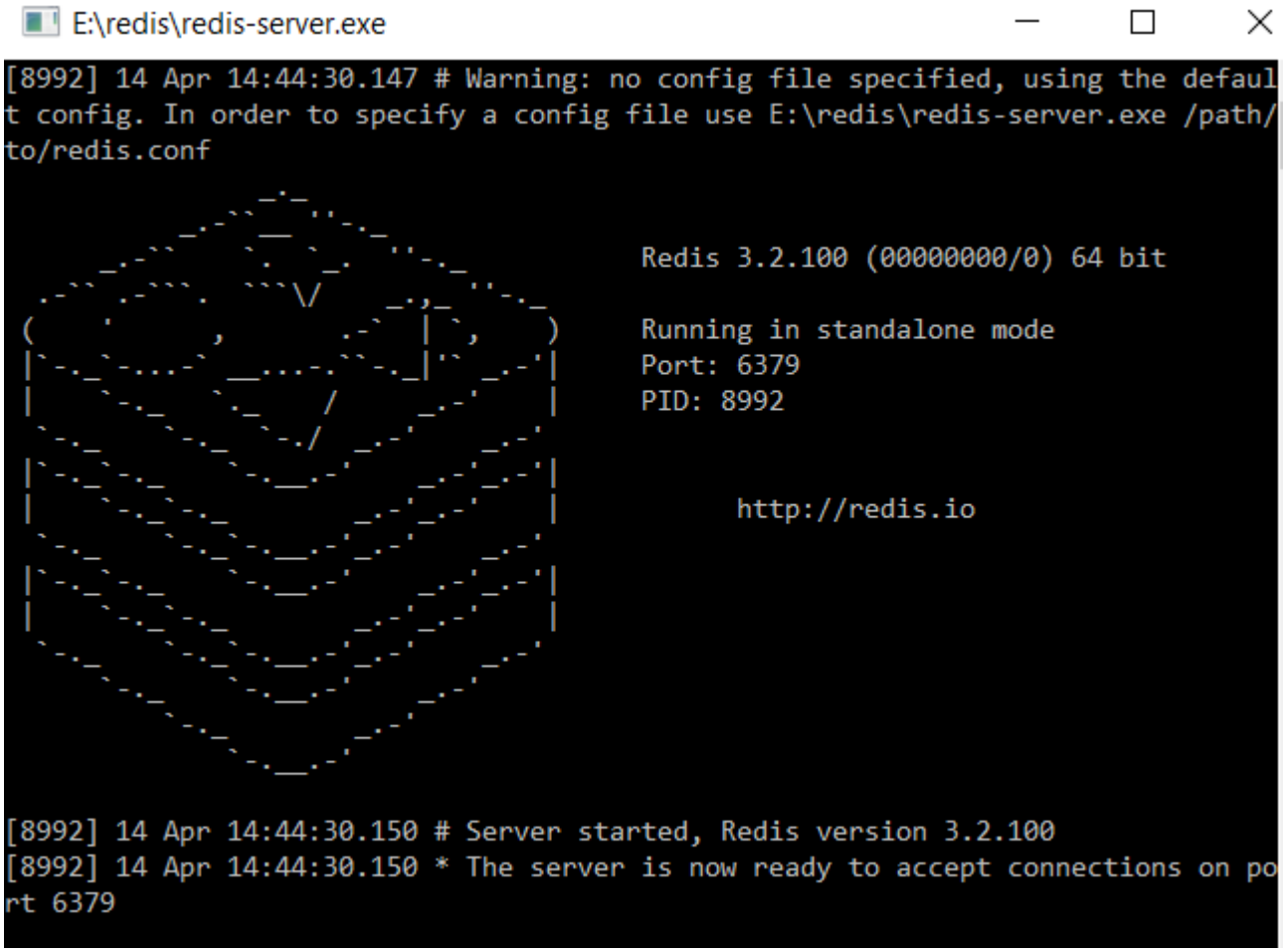
However, the **Microsoft Open Tech group** develops and maintains this Windows port targeting Win64. **Official redis.io/download**

You can choose to download different versions or the latest version of Redis
github.com/MSOpenTech/redis/releases

1. **Download** either .msi or .zip file, this tutorial will let you download latest zip file **Redis-x64-3.2.100.zip**.
2. **Extract the zip file** to prepared directory.

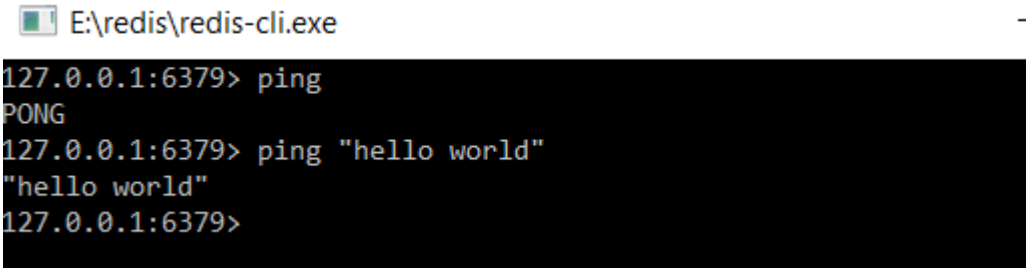



3. **Run redis-server.exe**, you can either directly run redis-server.exe by clicking or run via command prompt.




4. **Run redis-cli.exe**, after successfully running the redis-server. You can access it and test commands by running redis-cli.exe Te


**PING** command is used to test if a connection is still alive.




You can now start using Redis , please refer for more commands in official documentations

Read Installation and Setup online: https://riptutorial.com/redis/topic/2898/installation-and-setup

# Chapter 7: Lua Scripting

## Introduction

Redis provides a couple of mechanisms for extending the functionality of the database. One mechanism is through the use of server-side LUA scripts that can be executed to manipulate data. Lua scripts can be useful to perform expensive operations or to implement atomic operations that require logic.

## Examples

### Commands For Scripting

Redis provides seven different operations for working with scripts:

- Eval operations (EVAL, EVALSHA)
- SCRIPT operations (DEBUG, EXISTS, FLUSH, KILL, LOAD)

The EVAL command evaluates a script provided as a string argument to the server. Scripts can access the specified Redis keys named as arguments to the command and and additional string parameters that the user wants to pass to the script.

For example, the command:

```
EVAL "return {KEYS[1],KEYS[2],ARGV[1],ARGV[2]}" 2 key1 key2 first second
```

causes the execution of a user defined Lua script that simply returns the values supplied. The call is involved with 2 Redis keys (key1 and key2) and two parameters.

Another way to execute a Lua script is to first load it into the database then execute it using a SHA hash of the script.:

```
> script load "return {KEYS[1],KEYS[2],ARGV[1],ARGV[2]}"
"a42059b356c875f0717db19a51f6aaca9ae659ea"
> evalsha "a42059b356c875f0717db19a51f6aaca9ae659ea" 2 key1 key2 foo bar
1) "key1"
2) "key2"
3) "foo"
4) "bar"
```

The script load command loads the script and stores it in the database. A sha signature of the script is returned so it can be referenced by future calls. The EVALSHA function takes the sha and executes the corresponding script from the database.

Read Lua Scripting online: https://riptutorial.com/redis/topic/9112/lua-scripting

# Chapter 8: Pub/Sub

## Introduction

Redis provides an implementation of the Publish/Subscribe (Pub/Sub) messaging pattern. Instead of sending messages to specific receivers, Publishers send messages to interested receivers via some indirection mechanism. Receivers specify interest in particular messages. In Redis this functionality is accessed using the PUBLISH and SUBSCRIBE commands on channels.

## Syntax

- SUBSCRIBE channel [channel ...]
- UNSUBSCRIBE [channel [channel ...]]
- PUBLISH channel message
- PSUBSCRIBE pattern [pattern ...]
- PUNSUBSCRIBE [pattern [pattern ...]]

## Remarks

To handle the pub/sub in redis, need to have **one client for subscribe & different client for publish**. Both can't be handled by same client. Though all other commands can be still handled with same client.

## Examples

### Publish & subscribe with redis

Redis has publish/subscribe for sending messages. This is handled by subscribing to a channel & publishing to channel. Yes, subscribers will subscribe to one or more channels. Publisher need not know who are all subscribers. Instead, publisher will publish to specific channel. All the subscribers who are subscribed for that channel will get the message. This decoupling of publishers and subscribers can allow for greater scalability and a more dynamic network topology.

**Example:** User is subscribing to 2 channels say foo & boo

```
SUBSCRIBE foo boo
```

In console of redis-client1:

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

```
2) "boo"
3) (integer) 2
```

It will start to listen for message. On publish will get data for corresponding channel.

**For example:** When want to send message to all subscribers who are connected with boo, need to publish to that channel.

```
PUBLISH boo "Hello Boo"
```

In console of redis-client1:

```
1) "message"
2) "boo" //channel name
3) "Hello Boo" //Actual data
```

To unsubscribe from channel at any point, use

```
UNSUBSCRIBE // to unsubscribe from all channels
UNSUBSCRIBE foo // to unsubscribe from specific channel
```

Can do subscribe based on pattern too. When the channel name is not sure/want to subscribe based on pattern use **PSUBSCRIBE**.

Similarly to unsubscribe based on pattern use **PUNSUBSCRIBE**

Read Pub/Sub online: https://riptutorial.com/redis/topic/5071/pub-sub

# Chapter 9: Redis Keys

## Introduction

The Redis keyspace can be thought of as a hash table or dictionary mapping keys to data structures in the database.

Redis provides a wide range of commands that work with keys to manage the keyspace, including the ability to remove keys, inspect key metadata, search for keys, and modify certain properties of keys.

## Syntax

- KEYS pattern
- PERSIST key
- EXPIRE key seconds
- EXPIREAT key timestamp
- TTL key
- PEXPIRE key milliseconds
- PEXPIREAT key milliseconds-timestamp
- PTTL key
- UNLINK key [key ...]
- DEL key [key ...]
- SCAN cursor [MATCH pattern] [COUNT count]

## Remarks

For *valid characters in Redis keys*, the manual explains this completely:

> Redis keys are binary safe, this means that you can use any binary sequence as a key, from a string like "foo" to the content of a JPEG file. The empty string is also a valid key.
>
> A few other rules about keys:
>
> Very long keys are not a good idea, for instance a key of 1024 bytes is a bad idea not only memory-wise, but also because the lookup of the key in the dataset may require several costly key-comparisons. Even when the task at hand is to match the existence of a large value, to resort to hashing it (for example with SHA1) is a better idea, especially from the point of view of memory and bandwidth.
>
> Very short keys are often not a good idea. There is little point in writing "u1000flw" as a key if you can instead write "user:1000:followers". The latter is more readable and the added space is minor compared to the space used by the key object itself and the value object. While short keys will obviously consume a bit less memory, your job is to

find the right balance.

Try to stick with a schema. For instance "object-type:id" is a good idea, as in "user:1000". Dots or dashes are often used for multi-word fields, as in "comment:1234:reply.to" or "comment:1234:reply-to".

The maximum allowed key size is 512 MB.

Be careful with using the KEYS command against a production system, it can cause serious performance problems. If you need to do a search against the keyspace the SCAN commands are a better alternative.

# Examples

## Valid Keys

Redis keys are binary-safe, so literally anything can be used as a key. The only limitations are that they must be less than 512MB.

Examples of valid keys:

```
7
++++
`~!@#$%^&*()-_=+
user:10134
search/9947372/?query=this%20is%20a%28test%29%20query
<div id="div64">

Any other string less than 512MB in size.
The raw binary content of an image or other binary file.
An entire multi-line text document.
An entire SQL query.
Any integer, hexadecimal, octal, or binary value.
Anything else you can think of less than 512MB in size.
```

Invalid Redis keys:

```
Anything larger than 512MB.
```

## Key Naming Schemes

For clarity and maintainability, it is often recommended to develop a system or schema for naming your Redis keys. Here are some examples of common and maintainable systems for naming your keys:

```
user:10134
user:10134:favorites
user:10134:friends
user:10134:friends-of-friends

user:10134
```

```
user:10134/favorites
user:10134/friends
user:10134/friends.of.friends

user/10134
user/10134/favorites
user/10134/friends
user/10134/friends of friends
```

Note that, while allowed, larger keys use more memory and result in slower lookup times, so using a 500MB key might not be a great idea for performance. A better idea might be to use a SHA-1, SHA-256, or MD5 hash of a large binary object as a key instead:

```
image/9517bb726d33efdc503a43582e6ea2eea309482b
image52e9df0577fca2ce022d4e8c86b1eccb070d37bef09dec36df2fabbfa7711f5c
```

## Listing all keys

You can list all of the keys in a Redis database by executing the following commands from redis-cli:

```
KEYS *
```

The parameter to KEYS is a glob-style pattern matching expression. Examples of suppored patterns include:

```
h?llo matches hello, hallo and hxllo
h*llo matches hllo and heeeello
h[ae]llo matches hello and hallo, but not hillo
h[^e]llo matches hallo, hbllo, ... but not hello
h[a-b]llo matches hallo and hbllo
```

Using the KEYS * command can have adverse affects on performance, so it is not recommended against production instances. Use the SCAN operation to search for keys in production code.

## TTL and Key Expiration

The expiration values of a key can be managed by a user outside of the update commands. Redis allows a user to determine the current time to live (TTL) of a key using the TTL command:

```
TTL key
```

This command will return the TTL of a key in seconds or will return the special values -1 or -2. A -1 indicates that the key is persistent (won't expire) and a -2 indicates that the key does not exist.

An expiring key can be made persistent using the PERSIST command:

```
PERSIST KEY
```

and a persistent key can be made to expire using the EXPIRE command:

```
EXPIRE KEY seconds
```

Expire can also be used to modify the TTL of an existing key. Alternatively, you can use the EXPIREAT command with a UNIX timestamp to set an expire time.

There are millisecond versions of TTL, EXPIRE and EXPIREAT commands that are prefixed with a P.

## Deleting Keys

Redis provides two functions for removing keys from the database: del and unlink.

The del function removes one or more keys from the database. The del command causes Redis to immediately reclaim the memory for the deleted key on the current thread of execution. The execution time for del is proportional to the number of individual elements deleted from all the keys.

The unlink function acts like the del command, it removes one or more keys from the database. However, unlike the del command, any memory used by those keys is reclaimed asynchronously on another thread.

## Scanning the Redis Keyspace

Redis provides the SCAN command to iterate over the keys in the database matching a particular pattern. Redis supports glob style pattern matching in the SCAN command.

The SCAN command provides a cursor-based iterator over the Redis keyspace. The iterative call sequence to SCAN starts with the user making a call with the cursor argument set to 0. The result of that call is a batch of items and an updated cursor which is supplied to the next call to SCAN. This iteration continues until Redis returns a 0 cursor.

The following Python function demonstrates the basic usage of SCAN:

```
def scan_keys(r, pattern):
    "Returns a list of all the keys matching a given pattern"

    result = []
    cur, keys  = r.scan(cursor=0, match=pattern, count=2)
    result.extend(keys)
    while cur != 0:
        cur, keys = r.scan(cursor=cur, match=pattern, count=2)
        result.extend(keys)

    return result
```

The SCAN command is the recommended way to search for keys in the database, and is recommended over the `KEYS *` command.

Read Redis Keys online: https://riptutorial.com/redis/topic/3916/redis-keys

# Chapter 10: Redis List Datatype

## Introduction

The List datatype in Redis is an ordered collection of items referenced by a Redis key. Redis allows you to access and modify a list by index or push/pop operations. In Redis, the two ends of a list are referred to as the left and the right. The left corresponds to the first element or head of a list and the right coresponds to the last element or tail of a list.

## Syntax

- LPUSH key value [value ...]
- RPUSH key value [value ...]
- LPOP key
- RPOP key
- LLEN key

## Remarks

More detail on the List datatype and all the commands that can be used in conjunction with them can be found in the official Redis documentation at Redis.io.

## Examples

### Adding Items to a List

Redis allows you to add items to either the right or the left of a list.

If I was working with a list, my_list and I wanted to prepend 3 to the list, I could do that using the Redis LPUSH command:

```
LPUSH my_list 3
```

If I wanted to append 3 to my_list, I would instead use the RPUSH command:

```
RPUSH my_list 3
```

Both the LPUSH and RPUSH command will automatically create a new list for you if the supplied key doesn't exist. Two alternative commands LPUSHX and RPUSHX can be used to only operate on the list key if it already exists.

### Getting Items from a List

Redis provides the LPOP and RPOP commands as a counterpart to the LPUSH and RPUSH

commands for fetching data items.

If I was working with a list my_list that had several data items in it already, I can get the first item in the list using the LPOP command:

```
LPOP my_list
```

The result of this command will return the value of the first element from the list and remove it from my_list. For example, if I had the list [1, 3, 2, 4] and I applied LPOP to it, I would have the list [3, 2, 4] in memory afterwards.

Similarly, I can remove from the end of the list using RPOP:

```
RPOP my_list
```

would return the value fo the last element form the list and then remove it from my_list. Using our example, [1, 2, 3, 4] after calling RPOP on this list, the list in memory would be [1, 2, 3].

### Size of a List

The size of a Redis list can be deterimed using the LLEN command. If I have a four element list stored at the key my_list, I can get the size using:

```
LLEN my_list
```

which will return 4.

If a user specifies a key that doesn't exist to LLEN, it will return a zero, but if a key is used that points to an item of a different datatype, an error will be returned.

Read Redis List Datatype online: https://riptutorial.com/redis/topic/9107/redis-list-datatype

# Chapter 11: Redis Persistence Storage

## Introduction

Redis supports two main modes of persistence: RDB and AOF. The RDB mode of persistence takes a snapshot of your database at a point in time. In the RDB mode, Redis forks off a process to persist the database to disk. AOF logs every operation executed against the server into a replay log that can be processed at startup to restore the state of the database.

## Examples

### Disable all persistence storage in Redis

There are two kinds of persistent storage modes in Redis: AOF and RDB. To temporarily disable RDB execute the following commands on the Redis command line:

```
config set save ""
```

to temporarily disable AOF execute the following from the Redis command line:

```
config set appendonly no
```

The changes will persist until the server is restarted, then the server will revert back to whatever modes are configured in the server's redis.conf file.

The `CONFIG REWRITE` command can be used to modify the redis.conf file to reflect any dynamic changes to the configuration.

### Get persistence storage status

The following code will get the current configuration for the persistent storage state. These values can be modified dynamically, so they may differ from the configuration in redis.conf:

```
# get
config get appendonly
config get save
```

Read Redis Persistence Storage online: https://riptutorial.com/redis/topic/7871/redis-persistence-storage

# Chapter 12: Redis Set Datatype

## Introduction

Redis supports a set datatype analogous to mathematical sets for modeling data in the database. Sets are a compound datatype consisting of a group of unique, unordered members. Sets support adding and removing members, size operations, as well as combination operations that take two sets and generate a third set. Sets in Redis are similar to Sets in most programming languages.

## Syntax

- SADD key member [member ...]
- SISMEMBER key member
- SCARD key
- SADD key member [member ...]

## Remarks

The full documentation on the Redis set datatype can be found at Redis.io.

## Examples

### Size of a Set

The size of a set can be determined using the SCARD command. SCARD will return the cardinality of a set or the number of members in the set. For example, if I had a Redis set my_set stored in the database that looked like (Apple, Orange, Banana), I could get the size using the following code:

```
SCARD my_set
```

In the case of my example set, this would return 3. If the user executes an SCARD command on a key that does not exist, Redis will return 0.

### Adding Items to a Set

The basic Redis command for adding an item to a set is SADD. It takes a key and one or more members and adds them to the set stored at the given key.

For example, lets say that I wanted to create a set with the items apple, pear and banana. I could execute either of the following:

```
SADD fruit apple
SADD fruit pear
```

```
SADD fruit banana
```

or

```
SADD fruit apple pear banana
```

After executing either, I will have the set fruit with 3 items.

Attempting to add an item that is already in the set will have no effect. After setting up my fruit set using the code above, if I try to add apple again:

```
SADD fruit apple
```

Redis will attempt to add apple to the fruit set, but since it is already in the set nothing will change.

The result of the SADD command is always the number of items Redis added to a set. So attempting to re-add apple, will return a result of 0.

Member items in Redis are case sensitive, so apple and Apple are treated as two separate items.

### Testing for Membership

Redis supplies the SISMEMBER command to test if a particular item is already a member of a set. Using the SISMEMBER command I can test and see if apple is already a member of my fruit set.

If I construct my fruit set from the previous example, I can check and see if it contains apple using the following test:

```
SISMEMBER fruit apple
```

SISMEMBER will return a 1 since the item is already there.

If I tried to see if dog is a member of my fruit set:

```
SISMEMBER fruit dog
```

Redis will return a 0 since dog isn't in the fruit set.

If a user attempts to use the SISMEMBER command with a key that doesn't exist, Redis will return a 0 indicating no membership, but if you use SISMEMBER with a key that already holds a non-set datatype, Redis will return an error.

Read Redis Set Datatype online: https://riptutorial.com/redis/topic/9109/redis-set-datatype

# Chapter 13: Redis String datatype

## Introduction

Redis provides a string datatype that is used to associate data with a particular key. Redis string are the most basic datatype available in Redis and one of the first datatypes that users learn to work with.

Strings are often associated with text data, but Redis strings are more like buffers that can be used to store a wide range of different data. Redis strings can be used to represent integers, floating point numbers, bitmaps, text, and binary data.

## Syntax

- SET key value [EX seconds] [PX milliseconds] [NX|XX]
- INCR key
- INCRBY key increment
- INCRBYFLOAT key increment
- DECR key
- DECRBY key decrement

## Examples

### Working with String as Integers

Several commands allow you to work with Strings representing integer values.

A user can set the integer value of a key using the command:

```
SET intkey 2
```

The set command will create the key if necessary or update it if it already exists.

The value of an integer key can be updated on the server using either the INCR or INCRBY commands. INCR will increase the value of a key by 1 and INCRBY will increase the value of the key by the provided step value.

```
INCR intkey
INCRBY intkey 2
```

If the value of the key specified to INCR or INCRBY can't be expressed as an integer, Redis will return an error. If the key doesn't exist, the key will be created and the operation will be applied to the default value of 0.

The DECR and DECRBY ccommands work in reverse to decrement the value.

### Working with Strings as Floating Point Numbers

Redis allow you to use the String data type to store floating point numbers.

A user can set the float value of a key using the command:

```
SET floatkey 2.0
```

The set command will create the key if necessary or update it if it already exists.

The value of the key can be updated on the server using either the INCRBYFLOAT command. INCRBYFLOAT will increase the value of a key by the provided increment value.

```
INCRBYFLOAT floatkey 2.1
```

If the value of the key specified to INCRBYFLOAT can't be expressed as a floating point, Redis will return an error. If the key doesn't exist, the key will be created and the operation will be applied to the default value of 0.0.

Keys can be decremented by passing a negative increment to the INCRBYFLOAT command.

Read Redis String datatype online: https://riptutorial.com/redis/topic/9507/redis-string-datatype

# Chapter 14: Sorted Sets

## Introduction

The Sorted Set datatype in Redis is an ordered version of the Set datatype. A Redis sorted set consists of a collection of unique members. Each member in the sorted set can be thought of as a pair consisting of the member and a score. The score is used to order the members within the set in ascending order.

## Syntax

- ZADD key [NX|XX] [CH] [INCR] score member [score member ...]
- ZCARD key
- ZCOUNT key min max
- ZLEXCOUNT key min max

## Remarks

The official documentation for Sorted Sets can be found at the Redis.io site.

Sorted sets are sometimes referred to as zsets. If you use the TYPE command on a sorted set key, the value zset will be returned.

## Examples

### Adding Items to a Sorted Set

Redis provides the ZADD command to add items to a sorted set. The basic form of the ZADD command is to specify the set, the item to add and it's score. For example, if I wanted to construct an ordered set of my favorite food (from least to most), I could use either of:

```
zadd favs 1 apple
zadd favs 2 pizza
zadd favs 3 chocolate
zadd favs 4 beer
```

or alternatively:

```
zadd favs 1 apple 2 pizza 3 chocolate 4 beer
```

The ZADD function operates very similarly to the unsorted set function SADD. The result of the ZADD command is the number of items that were added. So after creating my set as above, if I attempted to ZADD beer again:

```
ZADD favs 4 beer
```

I would get a 0 result, if I decided I like chocolate better than beer, I could execute:

```
ZADD favs 3 beer 4 chocolate
```

to update my preferences, but I would still get a 0 return result since both beer and chocolate are already in the set.

### Counting Items in a Sorted Set

Redis provides three commands to count the items within a sorted set: ZCARD, ZCOUNT, ZLEXCOUNT.

The ZCARD command is the basic test for the cardinality of a set. (It is analogous to the SCARD command for sets.) . ZCARD returns the count of the members of a set. Executing the following code to add items to a set:

```
zadd favs 1 apple
zadd favs 2 pizza
zadd favs 3 chocolate
zadd favs 4 beer
```

running ZCard:

```
zcard favs
```

returns a value of 4.

The ZCOUNT and ZLEXCOUNT commands allow you to count a subset of the items in a sorted set based on a range of values. ZCOUNT allows you to count items within a particular range of scores and ZLEXCOUNT allowes you to count the number of items within a particular lexographic range.

Using our set above:

```
zcount favs 2 5
```

would return a 3, since there are three items (pizza, chocolate, beer) that have scores between 2 and 5 inclusive.

ZLEXCOUNT is designed to work with sets where every item has the same score, forcing and ordering on the elemement names. If we created a set like:

```
zadd favs 1 apple
zadd favs 1 pizza
zadd favs 1 chocolate
zadd favs 1 beer
```

we could use ZLEXCOUNT to get the number of elements in particular lexographical range (this is done by byte-wise comparison using the memcpy function).

```
zlexcount favs [apple (chocolate
```

would return 2, since two elements (apple, beer) fall within the range apple (inclusive) and chocolate (exclusive). We could alternatively make both ends inclusive:

```
zlexcount favs [apple [chocolate
```

and get the result 3.

Read Sorted Sets online: https://riptutorial.com/redis/topic/9111/sorted-sets

# Credits

| S. No | Chapters | Contributors |
|---|---|---|
| 1 | Getting started with redis | Ahamed Mustafa M, Alexander V., Aminadav, Community, Florian Hämmerle, Itamar Haber, Prashant Barve, RLaaa, Sagar Ranglani, sirin |
| 2 | Backup | odlp |
| 3 | Connecting to redis using Python | Gianluca D'Ardia, Tague Griffith, ystark |
| 4 | Geo | Tague Griffith |
| 5 | How to Connect to Redis in Java using Jedis | Tague Griffith |
| 6 | Installation and Setup | Cristiana214, Gianluca D'Ardia, Sagar Ranglani |
| 7 | Lua Scripting | Tague Griffith |
| 8 | Pub/Sub | jerry, Tague Griffith |
| 9 | Redis Keys | Tague Griffith, Will |
| 10 | Redis List Datatype | Tague Griffith |
| 11 | Redis Persistence Storage | Aminadav, Tague Griffith |
| 12 | Redis Set Datatype | JonyD, Tague Griffith |
| 13 | Redis String datatype | Tague Griffith |
| 14 | Sorted Sets | Tague Griffith |