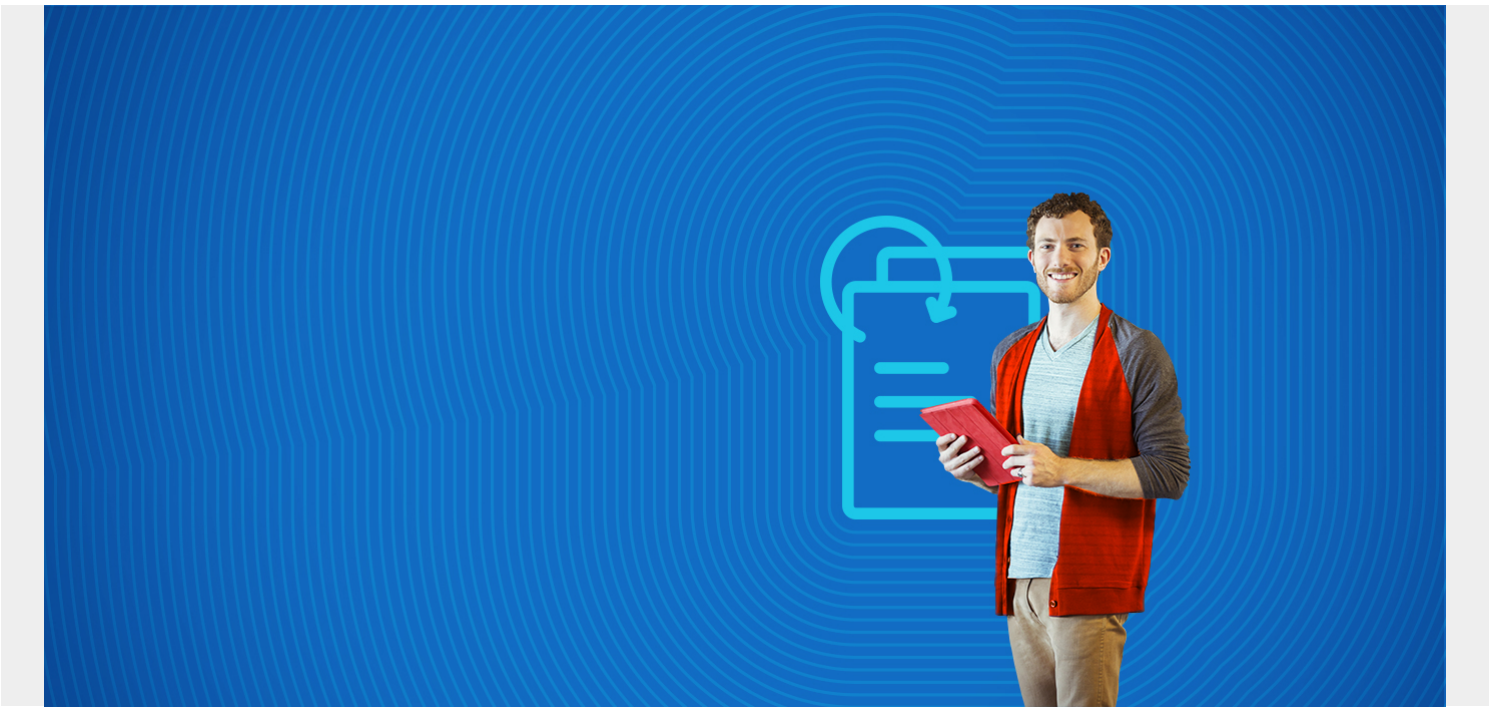
REDIS: BASIC COMMANDS FOR THE IN-MEMORY DATABASE



This tutorial introduces some basic commands for Redis.

(This tutorial is part of our Redis Guide. Use the right-hand menu to navigate.)

Redis architecture

Redis is a distributed in-memory database. It stores data in key, value pairs. There are no tables, schema, or collections.

Redis processes data in memory but stores it on disk. Processing it in memory means it is very fast, because there are no mechanical moving parts as there is no disk I/O.

Of course in memory processes that spill over to other servers in the cluster would be limited by the speed of the ethernet connection between the servers (which is usually 1 GBPS). But that is true with any distributed architecture.

Redis makes the analogy between their distributed memory system and **memcached**, which its authors say "Memcached is an in-memory key-value store for small chunks of arbitrary data (strings, objects) from results of database calls, API calls, or page rendering." So, memcached is temporary storage and Redis is permanent.

Installing Redis

You can try Redis online from here or install it.

Redis does not promote installing their server from packages. Instead make sure you

have installed build-essential, so that you can compile code, then download from source then issue this command to compile the code:

```
make
```

if you have any problems look here.

Now, start it:

```
src/redis-server
```

open the command line interface (cli):

```
src/redis-cli
```

Add key->Value pairs

Add a key value pair then retrieve it. Here the key is **walker** and the value is **walker**.

```
set walker "walker"
OK
127.0.0.1:6379> get walker
"walker"
```

Redis will let you add to numeric values using the incr function. The function is atomic meaning it will work even when two people are adding to the same key at the same time.

```
set age 10
OK
127.0.0.1:6379> incr age
(integer) 11
```

Lists

Lists are a data structure that contains lists of items. The items are stored in the order you add them.

```
rpush students "walker" "stephen" "ellen"
(integer) 3
```

You can remove the right-most element by popping it from the list:

```
rpop students
"ellen"
```

Now look at the list. First we need to get the length so that we can list then as the lrange function requires that you tell it where to stop. List commands start with **l** while set commands start with **s**. The first element is 0.

```
llen students
(integer) 2lrange students 0 1
1) "walker"
2) "stephen"
```

You can also list elements starting from the end of the list using negative numbers. This lists the last element:

```
lrange students -1 1
1) "stephen"
```

Sets

Sets are lists that do not allow duplicate values. Use **SADD** to add elements.

See below. Here we show you cannot add the same element twice to a set as the second command returns **0** elements added:

```
sadd numbers 1
(integer) 1
127.0.0.1:6379> sadd numbers 1
(integer) 0
```

Here is the union of two sets. We use **odd** and **even** numbers as the set names.

```
sadd odd 1sadd odd 3sadd odd 5sadd even 2sadd even 4sadd even 6sunion odd
even
1) "1"
2) "2"
3) "3"
4) "4"
5) "5"
6) "6"
```

And the intersection should be empty since no number can be both even and odd:

```
sinter odd even
(empty list or set)
```

Redis also supports sorted sets and many other data structures.

Scripting

Redis scripting uses the Lua programming language. Lua's authors say:

"Lua is a powerful and fast programming language that is easy to learn and use and to embed into your application."

In the next blog post we will explore how to use Lua with Redis. But if you don't want to use Lua you can use any of the many Redis clients listed here.