The gawk-redis extension library

Full documentation of the ”API Redis” of the gawkextlib project

Paulino Huerta

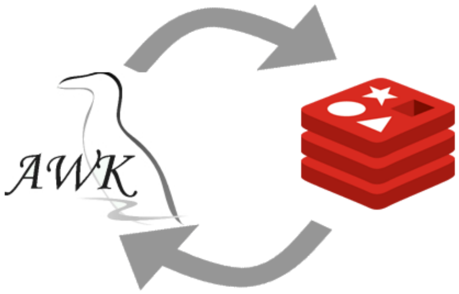
The gawk-redis extension library

Full documentation of the ”API Redis” of the gawkextlib project

Paulino Huerta

Contents

gawk-redis



Arrows denote the functionality or mission of *gawk-redis*

A GAWK[1] (the GNU implementation of the AWK Programming Language) client library for Redis.

The gawk-redis is an extension library that enables GAWK , to process data from a Redis server[2], then provides an API for communicating with the Redis key-value store, using hiredis[3], a C client for Redis.

The prefix "redis_" must be at the beginning of each function name, as shown in the code examples, although the explanations are omitted for clarity.

[1]https://www.gnu.org/software/gawk/
[2]http://redis.io/
[3]https://github.com/redis/hiredis

Installing/Configuring

Everything you should need to install gawk-redis on your system.

- Install hiredis[4], library C client for Redis.
- The README file will explain how to build the Redis extensions for gawk.
- Interested in release candidates or unstable versions? check the repository[5]

You can try running the following gawk script, *myscript.awk*, which uses the extension:

Example: Using gawk-redis extension

```
@load "redis"
BEGIN{
  # the connection with the server: 127.0.0.1:6379
  c=redis_connect()
  if(c==-1) {
    # always you can to use the ERRNO var for checking
    print ERRNO
  }
  # the select redis command
  ret=redis_select(c,4)
  # The above statement assumes that db 4 contains data
  print "select returns "ret
  pong=redis_ping(c) # the ping redis command
  print "The server says: "pong
  # the echo redis command
  print redis_echo(c,"foobared")
  redis_close(c)
}
```

[4]https://github.com/redis/hiredis
[5]https://sourceforge.net/u/paulinohuerta/gawkextlib_d/ci/master/tree/

which must run with:

```
/path-to-gawk/gawk -f myscript.awk /dev/null
```

The API Functions

1. Connection
2. Keys
3. Strings
4. Hashes
5. Lists
6. Sets
7. Sorted Sets
8. Pub/sub
9. Pipelining
10. Scripting
11. Server
12. Transactions
13. HyperLogLog
14. Geolocation

Connection Functions

1. connect - Connect to a Redis server
2. auth - Authenticate to the server
3. select - Change the selected database for the current connection
4. close, disconnect - Close the connection
5. ping - Ping the server
6. echo - Echo the given string

connect

Description: Connects to a Redis instance.

Parameters
host: string, optional
port: number, optional

Return value
connection handle: number, `-1` on error.

Example: Using connect

```
c=redis_connect('127.0.0.1', 6379)
# port 6379 by default
c=redis_connect('127.0.0.1')
# host address 127.0.0.1 and port 6379 by default
c=redis_connect()
```

auth

Description: Authenticate the connection using a password.

Parameters
number: connection
string: password

Return value
`1` if the connection is authenticated, `null string` (empty string) otherwise.

Example: Using auth

```
ret=redis_auth(c,"fooXX")
if(ret) {
  # authenticated
}
else {
  # not authenticated
}
```

select

Description: Change the selected database for the current connection.

Parameters
number: dbindex, the database number to switch to

Return value
`1` in case of success, `-1` in case of failure.

Example: Using select

```
redis_select(c,5)
```

close, disconnect

Description: Disconnects from the Redis instance.

Parameters
number: connection handle

Return value
`1` on success, `-1` on error.

Example: Using close

```
ret=redis_close(c)
if(ret==-1) {
  print ERRNO
}
```

ping

Description: Check the current connection status

Parameters
number: connection handle

Return value
string: `PONG` on success.

echo

Description: Sends a string to Redis, which replies with the same string
Parameters
number: connection
string: The message to send.

Return value
string: the same message.

Keys Functions

1. del - Delete a key
2. dump - Return a serialized version of the value stored at the specified key.
3. exists - Determine if a key exists
4. expire, pexpire - Set a key’s time to live in seconds
5. keys - Find all keys matching the given pattern
6. move - Move a key to another database
7. object - Allows to inspect the internals of Redis Objects
8. persist - Remove the expiration from a key
9. randomkey - Return a random key from the keyspace
10. rename - Rename a key
11. renamenx - Rename a key, only if the new key does not exist
12. sort - Sort the elements in a list, set or sorted set
13. sortLimit - Sort the elements in a list, set or sorted set, using the LIMIT modifier
14. sortLimitStore - Sort the elements in a list, set or sorted set, using the LIMIT and STORE modifiers
15. sortStore - Sort the elements in a list, set or sorted set, using the STORE modifier
16. scan - iterates the set of keys in the currently selected Redis db
17. type - Determine the type stored at key
18. ttl, pttl - Get the time to live for a key
19. restore - Create a key using the provided serialized value, previously obtained with *dump*.

del

Description: Remove specified keys.

Parameters
string or array of string: `key name` or `array name` containing the names of the keys

Return value
number: Number of keys deleted.

Example: **Using del**

```
redis_set(c,"keyX","valX")
redis_set(c,"keyY","valY")
redis_set(c,"keyZ","valZ")
redis_set(c,"keyU","valU")
AR[1]="keyY"
AR[2]="keyZ"
AR[3]="keyU"
redis_del(c,"keyX") # return 1
redis_del(c,AR) # return 3
```

exists

Description: Verify if the specified key exists.

Parameters
number: connection
string: key name

Return value
`1` If the key exists, `0` if the key no exists.

Example: Using exists

```
redis_set(c,"key","value");
redis_exists(c,"key"); # return 1
redis_exists(c,"NonExistingKey") # return 0
```

randomKey

Description: Returns a random key.

Parameters
number: connection

Return value
string: a random key from the currently selected database

Example: Using randomKey

```
print redis_randomkey(c)
```

move

Description: Moves a key to a different database. The key will move only if not exists in destination database.

Parameters
number: connection
string: key, the key to move
number: dbindex, the database number to move the key to

Return value
`1` if key was moved, `0` if key was not moved.

Example: Using move

```
redis_select(c,0)          # switch to DB 0
redis_set(c,"x","42") # write 42 to x
redis_move(c,"x", 1)  # move to DB 1
redis_select(c,1)          # switch to DB 1
redis_get(c,"x");          # will return 42
```

object

Description: allows to inspect the internals of Redis Objects associated with keys. It is useful for debugging or to understand if your keys are using the specially encoded data types to save space. Supports the sub commands: refcount, encoding and idletime. You can to read more about the `object` command[6]

Parameters
number: connection
string: sub command *string*: key

Return value
`number integers` for subcommands refcount and idletime.
`string` for subcommand encoding.
`null string` if the object to inspect is missing. `-1` when the subcommand is non-existent.

[6]http://redis.io/commands/object

Example: Using object

```
@load "redis"
BEGIN {
  c=redis_connect()
  # print "type key students:433:",
        # redis_type(c,"students:433")
  # print "type key foo:",
        # redis_type(c,"foo")
  print "students:433 idletime:",
         redis_object(c,"idletime","students:433")
  print "foo idletime:",
        redis_object(c,"idletime","foo")
  # cob:11 can not exist
  if((value=redis_object(c,"idletime","cob:11"))=="")
    print "Key cob:11 non-existent"
  else
    print value
  if((value=redis_object(c,"refcount","cob:11"))=="")
    print "Key cob:11 non-existent"
  else
    print value
  print "foo refcount:",
        redis_object(c,"refcount","foo")
  print "foo encoding:",
        redis_object(c,"encoding","foo")
  print "students:433 refcount:",
        redis_object(c,"refcount","students:433")
  print "students:433 encoding:",
        redis_object(c,"encoding","students:433")
  # "command"  is not one of the three sub commands
  ret=redis_object(c,"command","students:433")
  if(ret==-1)
     print ERRNO
  redis_close(c)
}
```

Output

```
students:433 idletime: 263
foo idletime: 263
Key cob:11 non-existent
Key cob:11 non-existent
foo refcount: 1
foo encoding: embstr
students:433 refcount: 1
students:433 encoding: ziplist
object need a valid command refcount|encoding|idletime
```

rename

Description: Renames a key. If newkey already exists it is overwritten.

Parameters
number: connection
string: srckey, the key to rename.
string: dstkey, the new name for the key.

Return value
`1` in case of success, `-1` in case of error.

Example: Using rename

```
redis_set(c,"x", "valx");
redis_rename(c,"x","y");
redis_get(c,"y")  # return "valx"
redis_get(c,"x")  # return null string, because x
                  # no longer exists
```

renamenx

Description: Same as rename, but will not replace a key if the destination already exists. This is the same behaviour as set and option nx.

Return value
`1` in case of success, `0` in case not success.

expire, pexpire

Description: Sets an expiration date (a timeout) on an item. pexpire requires a TTL in milliseconds.

Parameters
number: connection
string: key name. The key that will disappear.
number: ttl. The key's remaining Time To Live, in seconds.

Return value
`1` in case of success, `0` if key does not exist or the timeout could not be set

Example: Using expire

```
ret=redis_set(c,"x", "42") # ret value 1; x value "42"
redis_expire(c,"x", 3) # x will disappear in 3 seconds
system("sleep 5") # wait 5 seconds
redis_get(c,"x") # will return null string,
                 # as x has expired
```

keys

Description: Returns the keys that match a certain pattern. Check supported glob-style patterns[7]

Parameters
number: connection
string: pattern
array of strings: the results, the keys that match a certain pattern.

Return value
`1` in case of success, `-1` on error

Example: Using keys

```
redis_keys(c,"*",AR)   # all keys will match this
delete AR
# for matching all keys begining with "user"
redis_keys(c,"user*",AR)
# show AR contains
for(i in AR) {
  print i": "AR[i]
}
```

[7]http://redis.io/commands/keys

type

Description: Returns the type of data pointed by a given key.

Parameters
number: connection
string: key name

Return value
string: the type of the data (string, list, set, zset and hash) or `none` when the key does not exist.

Example: Using type

```
redis_set(c,"keyZ","valZ")
ret=redis_type(c,"keyZ") # ret contains "string"
# showing the "type" all keys of DB 4
redis_select(c,4)
redis_keys(c,"*",KEYS)
for(i in KEYS){
  print i": "KEYS[i]" ---> "redis_type(c,KEYS[i])
}
```

sort

Description: Sort the elements in a list, set or sorted set.

Parameters
number: connection
string: key name
array: the array with the result
optional string: options "desc|asc alpha"

Return value
`1` or `-1`on error

Example: Using sort

```
c=redis_connect()
redis_del(c,"thelist1");
print redis_type(c,"thelist1") # none
redis_lpush(c,"thelist1","bed")
redis_lpush(c,"thelist1","pet")
redis_lpush(c,"thelist1","key")
redis_lpush(c,"thelist1","art")
redis_lrange(c,"thelist1",AR,0, -1)
for(i in AR){
  print i") "AR[i]
}
delete AR
# sort desc "thelist1"
ret=redis_sort(c,"thelist1",AR,"alpha desc")
print "-----"
for(i in AR){
  print i") "AR[i]
}
print "-----"
```

sortLimit

Description: Sort the elements in a list, set or sorted set, using the LIMIT modifier with the sense of limit the number of returned elements.

Parameters
number: connection
string: key name
array: the array with the result *number*: offset *number*: count
optional string: options "desc|asc alpha"

Return value
`1` or `-1`on error

Example: Using type

```
# will return 5 elements of the sorted version of
#   list2, starting at element 0
c=redis_connect()
# assume "list2" with numerical content
ret=redis_sortLimit(c,"list2",AR,0,5)
# or using a sixth argument
# ret=redis_sortLimit(c,"list2",AR,0,5,"desc")
#   for Alphanumeric content should use "alpha"
if(ret==-1) {
  print ERRNO
}
for(i in AR){
  print i") "AR[i]
}
redis_close(c)
```

sortLimitStore

Description: Sort the elements in a list, set or sorted set, using the LIMIT and STORE modifiers with the sense of limit the number of returned elements and ensure that the result is stored as in a new key instead of be returned.

Parameters
number: connection
string: key name
string: the name of the new key *number*: offset *number*: count *otional string*: options "desc|asc alpha"

Return value
`1` or `-1`on error

Example: Using sortLimitStore

```
# will store 5 elements, of the sorted version of list2
# in the list "listb"
c=redis_connect()
# assume "list2" with numerical content
ret=redis_sortLimitStore(c,"list2","listb",0,5)
# or using a sixth argument
# redis_sortLimitStore(c,"list2","listb",0,5,"desc")
```

sortStore

Description: Sort the elements in a list, set or sorted set, using the STORE modifier for that the result to be stored in a new key

*Parameters**_

number: connection
string: key name
string: the name of the new key *optional string*: options "desc|asc alpha"

*Return value**_

`1` or `-1`on error

Example: Using sortStore

```
c=redis_connect()
redis_del(c,"list2")
redis_lpush(c,"list2","John")
redis_lpush(c,"list2","Sylvia")
redis_lpush(c,"list2","Tom")
redis_lpush(c,"list2","Brenda")
redis_lpush(c,"list2","Charles")
redis_lpush(c,"list2","Liza")
ret=redis_sortStore(c,"list2","listb")
# or using a fourth argument
# ret=redis_sortStore(c,"list2","listb","desc alpha")
```

scan

Description: iterates the set of keys. Please read how it works from Redis scan[8] command

Parameters
number: connection
number: the cursor
array: for to hold the results
string (optional): for to `match` a given glob-style pattern, similarly to the behavior of the `keys` function that takes a pattern as only argument

Return value
`1` on success, or `0` on the last iteration (when the returned cursor is equal 0). Returns `-1` on error.

[8] http://redis.io/commands/scan

Example: Using scan

```
@load "redis"
BEGIN{
 c=redis_connect()
 num=0
 while(1){
   # the last parameter (the pattern "s*"), is optional
   ret=redis_scan(c,num,AR,"s*")
   if(ret==-1){
    print ERRNO
    redis_close(c)
    exit
   }
   if(ret==0){
    break
   }
   n=length(AR)
   for(i=2;i<=n;i++) {
     print AR[i]
   }
   num=AR[1]  # AR[1] contains the cursor
   delete(AR)
 }
 for(i=2;i<=length(AR);i++) {
   print AR[i]
 }
 redis_close(c)
}
```

ttl, pttl

Description: Returns the time to live left for a given key in seconds (ttl), or milliseconds (pttl).

Parameters
number: connection
string: key name

Return value
number: The time to live in seconds. If the key has no ttl, `-1` will be returned, and `-2` if the key doesn't exist.

Example: Using ttl

```
redis_ttl(c,"key")
```

persist

Description: Remove the expiration timer from a key.

Parameters
number: connection
string: key name

Return value
`1` if a timeout was removed, `0` if key does not exist or does not have an associated timeout

Example: Using persist

```
redis_exists(c,"key) # returns 1
# returns -1 if has no associated expire
redis_ttl(c,"key")
redis_expire(c,"key",100)  # returns 1
redis_persist(c,"key")     # returns 1
redis_persist(c,"key")     # returns 0
```

dump

Description: Dump a key out of a redis database, the value of which can later be passed into redis using the RESTORE command. The data that comes out of DUMP is a binary representation of the key as Redis stores it.

Parameters
number: connection
string: key name

Return value
The Redis encoded value of the key, or `string` `null` if the key doesn't exist

Example: Using dump

```
redis_set(c,"foo","bar")
val=redis_dump(c,"foo")
 # val will be the Redis encoded key value
```

restore

Description: Restore a key from the result of a DUMP operation.

Parameters
number: connection
string: key name.
number: ttl number. How long the key should live (if zero, no expire will be set on the key).
string: value string (binary). The Redis encoded key value (from DUMP).

Return value
`1` on sucess, `-1` on error

Example: Using restore

```
redis_set(c,"foo","bar")
val=redis_dump(c,"foo")
# The key "bar", will now be equal to the key "foo"
redis_restore(c,"bar",0,val)
```

Strings Functions

1. append - Append a value to a key
2. bitcount - Count set bits in a string
3. bitop - Perform bitwise operations between strings
4. decr, decrby - Decrement the value of a key
5. get - Get the value of a key
6. getbit - Returns the bit value at offset in the string value stored at key
7. getrange - Get a substring of the string stored at a key
8. getset - Set the string value of a key and return its old value
9. incr, incrby - Increment the value of a key
10. incrbyfloat - Increment the float value of a key by the given amount
11. mget - Get the values of all the given keys
12. mset - Set multiple keys to multiple values
13. set - Set the string value of a key
14. setbit - Sets or clears the bit at offset in the string value stored at key
15. setrange - Overwrite part of a string at key starting at the specified offset
16. strlen - Get the length of the value stored in a key

---

get

Description: Get the value related to the specified key

Parameters
number: connection
string: the key

Return value
string: `key value` or `null string` (empty string) if key didn't exist.

Example: Using get

```
value=redis_get(c,"key1")
```

set

Description: Set the string value in argument as value of the key. If you're using Redis `>=` 2.6.12, you can pass extended options as explained below

Parameters
number: connection
string: key
string: value
and optionally: "EX",timeout,"NX" or "EX",timeout,"XX" or "PX" instead of "EX"

Return value
`1` if the command is successful `string null` if no success, or `-1` on error.

Example: Using set

```
# Simple key -> value set
redis_set(c,"key","value");

# Will redirect, and actually make an SETEX call
redis_set(c,"mykey1","myvalue1","EX",10)

# Will set the key, if it doesn't exist, with a ttl
# of 10 seconds
redis_set(c,"mykey1","myvalue1","EX",10,"NX")
```

```
# Will set a key, if it does exist, with a ttl
# of 10000 miliseconds
redis_set(c,"mykey1","myvalue1","PX",10000,"XX")
```

incr, incrby

Description: Increment the number stored at key by one. If the second argument is filled, it will be used as the integer value of the increment.

Parameters
number: connection
string: key name
number: value that will be added to key (only for incrby)

****Return value***
number: the new value

Example: Using incr

```
redis_incr(c,"key1")
# key1 didn't exists, set to 0 before the increment
# and now has the value 1
redis_incr(c,"key1") #  value 2
redis_incr(c,"key1") #  value 3
redis_incr(c,"key1") #  value 4
redis_incrby(c,"key1",10) #  value 14
```

incrbyfloat

Description: Increment the key with floating point precision.

Parameters
number: connection
string: key name
value: (float) value that will be added to the key

Return value
number: the new value

Example: Using incrbyfloat

```
redis_incrbyfloat(c,"key1", 1.5)
# key1 didn't exist, so it will now be 1.5
redis_incrbyfloat(c,"key1", 1.5)  # 3
redis_incrbyfloat(c,"key1", -1.5) # 1.5
redis_incrbyfloat(c,"key1", 2.5)  # 4
```

decr, decrby

Description: Decrement the number stored at key by one. If the second argument is filled, it will be used as the integer value of the decrement.

Parameters
number: connection
string: key name
number: value that will be substracted to key (only for decrby)

Return value
number: the new value

Example: Using decr

```
redis_decr(c,"keyXY")
# keyXY didn't exists, set to 0 before the increment
# and now has the value -1
redis_decr(c,"keyXY") # -2
redis_decr(c,"keyXY") # -3
redis_decrby(c,"keyXY",10)  # -13
```

mget

Description: Get the values of all the specified keys. If one or more keys dont exist, the array will contain `null string` at the position of the key.

Parameters
number: connection
Array: Array containing the list of the keys
Array: Array of results, containing the values related to keys in argument

Return value
`1` success `-1` on error

Example: Using mget

```
@load "redis"
BEGIN{
 null="\"\""
 c=redis_connect()
 redis_set(c,"keyA","val1")
 redis_set(c,"keyB","val2")
 redis_set(c,"keyC","val3")
 redis_set(c,"keyD","val4")
 redis_set(c,"keyE","")
 AR[1]="keyA"
 AR[2]="keyB"
 AR[3]="keyZ" # this key no exists
 AR[4]="keyC"
 AR[5]="keyD"
 AR[6]="keyE"
 ret=redis_mget(c,AR,K) # K is the array with results
 for(i=1; i<=length(K); i++){
   if(!K[i]) {
     # function exists was described previously
     if(redis_exists(c,AR[i])){
       print i": "AR[i]" ----> "null
     }
     else {
       print i": "AR[i]" ----> not exists"
     }
   }
   else {
     print i": "AR[i]" ----> ""\""K[i]"\""
   }
 }
 redis_close(c)
}
```

getset

Description: Sets a value and returns the previous entry at that key.

Parameters
number: connection
string: key name
string: key value

Return value
A string, the previous value located at this key

Example: Using getset

```
redis_set(c,"x", "42")
exValue=redis_getset(c,"x","lol")
 # returns "42", now the value of x is "lol"
 #
newValue = redis_get(c,"x") # return "lol"
```

append

Description: Append specified string to the string stored in specified key.

Parameters
number: connection
string: key name
string: value

Return value
number: Size of the value after the append

Example: Using append

```
redis_set(c,"key","value1")
redis_append(c,"key","value2") # 12
redis_get(c,"key") # "value1value2"
```

getrange

Description: Return a substring of a larger string

Parameters
number: connection
string: key name
number: start
number: end

Return value
string: the substring

Example: Using getrange

```
redis_set(c,"key","string value");
print redis_getrange(c,"key", 0, 5)  # "string"
print redis_getrange(c,"key", -5, -1)  # "value"
```

setrange

Description: Changes a substring of a larger string.

Parameters
number: connection
string: key name
number: offset
string: value

Return value
string: the length of the string after it was modified.

Example: Using setrange

```
redis_set(c,"key1","Hello world")
ret=redis_setrange(c,"key1",6,"redis") # ret value 11
redis_get(c,"key1") # "Hello redis"
```

strlen

Description: Get the length of a string value.

Parameters
number: connection
string: key name

Return value
number: length of string value

Example: Using strlen

```
redis_set(c,"key","value")
redis_strlen(c,"key")  # 5
```

getbit

Description: Return a single bit out of a larger string

Parameters
number: connection
string: key name
number: offset

Return value
number: the bit value (0 or 1)

Example: Using getbit

```
redis_set(c,"key", "\x7f") # this is 0111 1111
redis_getbit(c,"key", 0) # 0
redis_getbit(c,"key", 1) # 1
redis_set(c,"key", "s")  # this is 0111 0011
print redis_getbit(c,"key", 5) # 0
print redis_getbit(c,"key", 6) # 1
print redis_getbit(c,"key", 7) # 1
```

setbit

Description: Changes a single bit of a string.

Parameters
number: connection
string: key name
number: offset
number: value (1 or 0)

Return value
number: 0 or 1, the value of the bit before it was set.

Example: Using setbit

```
@load "redis"
BEGIN{
  c=redis_connect()
  redis_set(c,"key", "*") # ord("*") = 42 = "0010 1010"
  redis_setbit(c,"key", 5, 1) #  returns 0
  redis_setbit(c,"key", 7, 1) # returns 0
  print redis_get(c,"key") #  "/" = "0010 1111"
  redis_set(c,"key1","?") # 00111111
  print redis_get(c,"key1")
  print "key1: changing bit 7, it returns ",
         setbit(c,"key1", 7, 0) # returns 1
  print "key1: value actual is 00111110"
  print redis_get(c,"key1") # retorna ">"
  redis_close(c)
}
```

bitop

Description: Bitwise operation on multiple keys.

Parameters
number: connection
operator: either "AND", "OR", "NOT", "XOR"
ret_key: result key
array or string: array containing the keys or only one string (in case of using the NOT operator).

Return value
number: The size of the string stored in the destination key.

bitcount

Description: Count bits in a string.

Parameters
number: connection
string: key name

Return value
number: The number of bits set to `1` in the value behind the input key.

mset, msetnx

Description: Sets multiple key-value pairs in one atomic command. msetnx only returns `1` if all the keys were set (see set and option nx).

Parameters
number: connection
array: keys and their respectives values

Return value
`1` in case of success, `-1` on error. while msetnx returns `0` if no key was set (at least one key already existed).

Example: Using mset

```
@load "redis"
BEGIN {
 AR[1]="q1"
 AR[2]="vq1"
 AR[3]="q2"
 AR[4]="vq2"
 AR[5]="q3"
 AR[6]="vq3"
 AR[7]="q4"
 AR[8]="vq4"
 c=redis_connect()
 ret=redis_mset(c,AR)
 print ret" returned by mset"
 redis_keys(c,"q*",R)
 for(i in R){
   print i") "R[i]
 }
 redis_close(c)
}
```

Output

```
1 returned by mset
1) q2
2) q3
3) q4
4) q1
```

Hashes Functions

1. hdel - Deletes one or more hash fields
2. hexists - Determines if a hash field exists
3. hget - Gets the value of a hash field
4. hgetAll - Gets all the fields and values in a hash
5. hincrby - Increments the integer value of a hash field by the given number
6. hincrbyfloat - Increments the float value of a hash field by the given amount
7. hkeys - Gets all the fields in a hash
8. hlen - Gets the number of fields in a hash
9. hmget - Gets the values of all the given hash fields
10. hmset - Sets multiple hash fields to multiple values
11. hset - Sets the string value of a hash field
12. hsetnx - Sets the value of a hash field, only if the field does not exist
13. hscan - Iterates elements of Hash types
14. hvals - Gets all the values in a hash

hset

Description: Adds a value to the hash stored at key. If this value is already in the hash, `FALSE` is returned.

Parameters
number: connection
string: key name.
string: hash Key
string: value

Return value
`1` if value didn't exist and was added successfully, `0` if the value was already present and was replaced, `-1` if there was an error.

Example: Using hset

```
@load "redis"
BEGIN{
 c=redis_connect()
 redis_del(c,"thehash")
 redis_hset(c,"thehash","key1","hello") # returns 1
 redis_hget(c,"thehash", "key1") # returns "hello"
 redis_hset(c,"thehash", "key1", "plop")
  # returns 0, value was replaced
  #
 redis_hget(c,"thehash", "key1") # returns "plop"
 redis_close(c)
}
```

hsetnx

Description: Adds a value to the hash stored at key only if this field isn't already in the hash.

Return value
`1` if the field was set, `0` if it was already present.

Example: Using hsetnx

```
redis_del(c,"thehash")
redis_hsetnx(c,"thehash","key1","hello") # returns 1
redis_hget(c,"thehash", "key1") # returns "hello"
redis_hsetnx(c,"thehash", "key1", "plop")
 # returns 0. No change, value wasn't replaced
 #
redis_hget(c,"thehash", "key1") # returns "hello"
```

hget

Description: Gets a value associated with a field from the hash stored it key.

Parameters
number: connection
string: key name
string: hash field

Return value
string: the value associated with field, or `string null` when field is not present in the hash or the key does not exist.

hlen

Description: Returns the length of a hash, in number of items

Parameters
number: connection
string: key name

Return value
number: the number of fields in the hash, or `0` when key does not exist. `-1` on error (by example if key exist and isn't a hash).

Example: Using hlen

```
redis_hsetnx(c,"thehash","key1","hello1") # returns 1
redis_hsetnx(c,"thehash","key2","hello2") # returns 1
redis_hsetnx(c,"thehash","key3","hello3") # returns 1
redis_hlen(c,"thehash")  # returns 3
```

hdel

Description: Removes the specified fields from the hash stored at key.

Parameters
number: connection
string: key name
string or array: field name, or array name containing the field names

Return value
number: the number of fields that were removed from the hash, not including specified but non existing fields.

hkeys

Description: Obtains the keys in a hash.

Parameters
number: connection
Key: key name
array: containing field names results

Return value
`1` on success, `0` if the hash is empty or no exists

Example: Using hkeys

```
@load "redis"
BEGIN{
  c=redis_connect()
  redis_hkeys(c,"thehash",A) # returns 1
  for(i in A){
    print i": "A[i]
  }
  redis_close(c)
}
```

The order is random and corresponds to redis' own internal representation of the structure.

hvals

Description: Obtains the values in a hash.

Parameters
number: connection
Key: key name
array: contains the result with values

Return value
`1` on success, `0` if the hash is empty or no exists

Example: Using hvals

```
@load "redis"
BEGIN{
 c=redis_connect()
 redis_hvals(c,"thehash",A) # returns 1
 for(i in A){
   print i": "A[i]
 }
 redis_close(c)
}
```

The order is random and corresponds to redis' own internal representation of the structure.

hgetall

Description: Returns the whole hash.

Parameters
number: connection
Key: key name
array: for the result, contains the entire sequence of field/value

Return value
`1` on success, `0` if the hash is empty or no exists

Example: Using hgetall

```
@load "redis"
BEGIN{
c=redis_connect()
redis_hgetall(c,"thehash",A) # returns 1
n=length(A)
for(i=1;i<=n;i+=2){
  print i": "A[i]" ---> "A[i+1]
}
redis_close(c)
}
```

The order is random and corresponds to redis' own internal representation of the structure.

hscan

Description: iterates elements of Hash types. Please read how it works from Redis hscan[9] command.

Parameters
number: connection
string: key name
number: the cursor
array: for to hold the results
string (optional): for to `match` a given glob-style pattern, similarly to the behavior of the `keys` function that takes a pattern as only argument

Return value
`1` on success, or `0` on the last iteration (when the returned cursor is equal `0`). Returns `-1` on error (by example a WRONGTYPE Operation).

[9]http://redis.io/commands/hscan

Example: Using hscan

```
@load "redis"
BEGIN{
  c=redis_connect()
  num=0
  while(1){
    ret=redis_hscan(c,"myhash",num,AR)
    if(ret==-1){
     print ERRNO
     redis_close(c)
     exit
    }
    if(ret==0){
     break
    }
    n=length(AR)
    for(i=2;i<=n;i++) {
      print AR[i]
    }
    num=AR[1]  # AR[1] contains the cursor
    delete(AR)
  }
  for(i=2;i<=length(AR);i++) {
    print AR[i]
  }
  redis_close(c)
}
```

hexists

Description: Verify if the specified member exists in a hash.

Parameters
number: connection

string: key name
string: field or member

Return value
`1` If the member exists in the hash, otherwise return `0`.

Example: Using hexists

```
@load "redis"
BEGIN {
  c=redis_connect()
  if(redis_hexists(c,"hashb","cl1")==1) {
    print "Key cl1 exists in the hash hashb"
  }
  if(redis_hexists(c,"hashb","cl1")==0) {
    print "Key cl1 does not exist in the hash hashb"
  }
  if(redis_hexists(c,"hashb","cl1")==-1) {
    print ERRNO
  }
  redis_close(c)
}
```

hincrby

Description: Increments the value of a member from a hash by a given amount.

Parameters
number: connection
string: key name
string: member or field
number: (integer) value that will be added to the member's value

Return value
number: the new value

Example: Using hincrby

```
print redis_hset(c,"hashb","field", 5)     # returns 1
print redis_hincrby(c,"hashb","field", 1)  # returns 6
print redis_hincrby(c,"hashb","field", -1) # returns 5
print redis_hincrby(c,"hashb","field", -10)# returns -5
```

hincrbyfloat

Description: Increments the value of a hash member by the provided float value

Parameters
number:connection
string: key name
string: field name
number: (float) value that will be added to the member's value

Return value
number: the new value

Example: Using hincrbyfloat

```
redis_del(c,"h");
redis_hincrbyfloat(c,"h","x",1.5)
  # returns 1.5: field x = 1.5 now
redis_hincrbyfloat(c,"h","x", 1.5)
  # returns 3.0: field x = 3.0 now
redis_hincrbyfloat(c,"h","x",-3.0)
  # returns 0.0: field x = 0.0 now
```

hmset

Description: Fills in a whole hash. Overwriting any existing fields in the hash. If key does not exist, a new key holding a hash is created.

Parameters
number connection
string: key name
array: contains field names and their respective values

Return value
`1` on success, `-1` on error

Example: Using hmset

```
c=redis_connect()
AR[1]="a0"
AR[2]="value of a0"
AR[3]="a1"
AR[4]="value of a1"
ret=redis_hmset(c,"hash1",AR1)
```

hmget

Description: Retrieve the values associated to the specified fields in the hash.

Parameters
number: connection
string: key name
array or string: an array contains field names, or only one string that containing the name of field
array: contains results, a sequence of values associated with the given fields, in the same order as they are requested. For every field that does not exist in the hash, a null string (empty string) is associated.

Return value
`1` on success, `-1` on error

Example: Using hmget

```
load "redis"
BEGIN{
 c=redis_connect()
 J[1]="c2"
 J[2]="k3"
 J[3]="cl1"
 J[4]="c1"
 J[5]="c6"
 ret=redis_hmget(c,"thash",J,T)
 if(ret==-1) {
   print ERRNO
 }
 print "hmget: Results and requests"
 for (i in T) {
   print i": ",T[i], " ........ ",J[i]
 }
 ret=redis_hgetall(c,"thash",AR)
 print "hgetall from the hash thash"
 for (i in AR) {
   print i": "AR[i]
 }
 # other use allowed for hmget
 ret=redis_hmget(c,"thash","cl1",OTH)
 print "is cl1 a field?"
 for(i in OTH){
   print i": "OTH[i]
 }
 redis_close(c);
}
```

Output

```
hmget: Results and requests
1:     ........  c2
2:  vk3  ........  k3
3:  vcl1  ........  cl1
4:     ........  c1
5:     ........  c6
hgetall from the hash thash
1: k1
2: vk1
3: k3
4: vk3
5: cl1
6: vcl1
7: cl2
8: vcl2
is cl1 a field?
1: vcl1
```

Lists Functions

1. lindex - Returns the element at index index in the list.
2. linsertBefore - Inserts value in a list key before the reference value pivot.
3. linsertAfter - Inserts value in a list key after the reference value pivot.
4. llen - Gets the length/size of a list
5. lpop - Remove and get the first element in a list
6. lpush - Insert all the specified values at the head of a list
7. lpushx - Inserts a value at the head of the list, only if the key already exists and holds a list.
8. lrange - Get a range of elements from a list
9. lrem - Remove elements from a list
10. lset - Set the value of an element in a list by its index
11. ltrim - Trim a list to the specified range
12. rpop - Remove and get the last element in a list
13. rpoplpush - Returns and removes the last element (tail) of a list, and pushes the element at the first element (head) of other list.
14. rpush - Insert all the specified values at the tail of a list
15. rpushx - Inserts a value at the tail of the list, only if the key already exists and holds a list.
16. blpop - Is a blocking list pop primitive. Pops elements from the head of a list.
17. brpop - Is a blocking list pop primitive. Pops elements from the tail of a list.
18. brpoplpush - Is the blocking variant of RPOPLPUSH.

lindex

Description: Returns the element at index index in the list.

Parameters
number: connection
string: key name
number: the index

Return value
string: the requested element, or `null string` when index is out of range. `-1` on error

Example: Using lindex

```
@load "redis"
BEGIN{
  c=redis_connect()
  redis_del(c,"mylist99")
  redis_lpush(c,"mylist99","s1")
  redis_lpush(c,"mylist99","s0")
   # gets element index 0
  print redis_lindex(c,"mylist99",0)
   # gets the last element
  print redis_lindex(c,"mylist99",-1)
   # gets null string
  print redis_lindex(c,"mylist99",3)
  redis_close(c)
}
```

linsertBefore

Description: Inserts value in a list key before the reference value pivot.

Parameters
number: connection

string: key name
string: pivot *string*: value

Return value
number: the length of the list after the insert, or `-1` when the value pivot was not found.

Example: Using linsertBefore

```
redis_del(c,"mylist")
print redis_rpush(c,"mylist","Hello")
print redis_rpush(c,"mylist","World")
print redis_linsertBefore(c,"mylist","Hello","Hi")
print redis_linsertBefore(c,"mylist","OH","Mmm")
 # to use 'redis_lrange' for show the list
```

Output

```
1
2
3
-1
```

linsertAfter

Description: Inserts value in a list key after the reference value pivot

Parameters
number: connection
string: key name
string: pivot
string: value

Return value
number: the length of the list after the insert, or `-1` when the value pivot was not found.

Example: Using linsertAfter

```
redis_del(c,"mylist")
print redis_rpush(c,"mylist","Hello")
print redis_rpush(c,"mylist","World")
redis_linsertAfter(c,"mylist","Hello","--") # Returns 3
redis_linsertAfter(c,"mylist","World","OK") # Retuns 4
 # to use 'redis_lrange' for to show the list
```

rpop

Description: Return and remove the last element of the list.

Parameters
number: connection
string: key name

Return value
string: the value, `null string` in case of empty list or no exists

Example: Using rpop

```
@load "redis"
BEGIN{
  c=redis_connect()
  redis_del(c,"mylist")
  C[1]="push1";C[2]="push2";C[3]="push3"
  C[4]="push4";C[5]="push5";C[6]="pushs6"
  print redis_rpush(c,"mylist",C)
  redis_lrange(c,"mylist",AR,0,-1)
  for(i in AR) {
    print i") "AR[i]
  }
  print "RPOP to empty the list 'mylist'"
  while(redis_exists(c,"mylist")) {
    print redis_rpop(c,"mylist")
  }
  redis_close(c)
}
```

Output

```
6
1) push1
2) push2
3) push3
4) push4
5) push5
6) pushs6
RPOP to empty the list 'mylist'
pushs6
push5
push4
push3
push2
push1
```

rpoplpush

Description: Atomically returns and removes the last element (tail) of a source list, and pushes the element at the first element (head) of a destination list.

Parameters
number: connection
string: the source list name
string: the destination list name

Return value
string: the element being popped and pushed. If source key does not

exist, `null string` is returned and no operation is performed. `-1` on error (if any of the key names exist and is not a list).

Example: Using rpoplpush

```
@load "redis"
BEGIN{
  c=redis_connect()
  redis_del(c,"mylist")
  C[1]="a";C[2]="b";C[3]="c";C[4]="d"
  print redis_rpush(c,"mylist",C)
   # mylist before rpoplpush is executed
  redis_lrange(c,"mylist",AR,0,-1)
  for(i in AR) {
    print i") "AR[i]
  }
  redis_del(c,"mylist0")
  print redis_rpoplpush(c,"mylist","mylist0")
  delete AR
   # mylist after rpoplpush is executed
  redis_lrange(c,"mylist",AR,0,-1)
  for(i in AR) {
    print i") "AR[i]
  }
  print "Elements in 'mylist0':"
  delete AR
  redis_lrange(c,"mylist0",AR,0,-1)
  for(i in AR) {
    print i") "AR[i]
  }
  redis_close(c)
}
```

Output

```
4
1) a
2) b
3) c
4) d
d
1) a
2) b
3) c
Elements in 'mylist0':
1) d
```

brpoplpush

Description: Is the blocking variant of RPOPLPUSH. When the source list contains elements, this function behaves exactly like RPOPLPUSH, if the source list is empty, Redis will block the connection until another client pushes to it or until timeout is reached.

Parameters
number: connection
string: the source list name
string: the destination list name
number: timeout

Return value
string: the element being popped and pushed. If timeout is reached, a `null string` is returned. `-1` on error (if any of the key names exist and is not a list).

Example: Using brpoplpush

```
print redis_brpoplpush(c,"mylist","mylist0",10)
```

lpop

Description: Return and remove the first element of the list.

Parameters
number: connection
string key name

Return value
string: the value, `null string` in case of empty list or no exists

Example: Using lpop

```
@load "redis"
BEGIN{
 c=redis_connect()
 ret=redis_del(c,"list1")
 print "return del="ret
 ret=redis_lpush(c,"list1","AA")
 print "return lpush="ret
 ret=redis_lpush(c,"list1","BB")
 print "return lpush="ret
 ret=redis_lpush(c,"list1","CC")
 print "return lpush="ret
 ret=redis_lrange(c,"list1",AR,0,-1)
 print "return lrange="ret
 for(i in AR) {
   print i": "AR[i]
 }
 ret=redis_lpop(c,"list1")
 print "return lpop="ret
 delete AR
 ret=redis_lrange(c,"list1",AR,0,-1)
 print "return lrange="ret
 for(i in AR) {
   print i": "AR[i]
 }
 redis_close(c)
}
```

Output

```
return del=1
return lpush=1
return lpush=2
return lpush=3
return lrange=1
1: CC
2: BB
3: AA
return lpop=CC
return lrange=1
1: BB
2: AA
```

lpush

Description: Adds all the specified values to the head (left) of the list. Creates the list if the key didn't exist.

Parameters
number: connection
key: key name
string or array: the string value to push in key, or if is an array, it's containing all values.

Return value
number: The new length of the list in case of success, `-1` on error (if the key exists and is not a list).

Example: Using lpush

```
redis_lpush(c,"list1","dd")
# being the array 'A' that containing the values
redis_lpush(c,"list1",A)
# to see example code of rpush function
```

lpushx

Description: Inserts a value at the head of the list, only if the key already exists and holds a list, no operation will be performed when key does not yet exist.

Parameters
number: connection
string: key name
string: the value to push in key

Return value
number: The new length of the list in case of success. `0` when no operation is executed. `-1` on error (if the key exists and is not a list).

Example: Using lpushx

```
@load "redis"
BEGIN{
  c=redis_connect()
  redis_del(c,"mylist99")
  redis_lpush(c,"mylist99","s1")
  redis_lpushx(c,"mylist99","s2") # returns 2
  redis_del(c,"mylist99")
   # The next returns 0. The list not exist
  redis_lpushx(c,"mylist99","a")
  redis_lpush(c,"mylist99","a")  # returns 1
  redis_close(c)
}
```

rpushx

Description: Inserts a value at the tail of the list, only if the key already exists and holds a list, no operation will be performed when key does not yet exist.

Parameters
number: connection
string: key name
string: the value to push in key

Return value
number: The new length of the list in case of success. `0` when no operation is executed. `-1` on error (if the key exists and is not a list).

Example: Using rpushx

```
redis_del(c,"mylist99")
 # the next returns 0 because 'mylist99' not exist
redis_rpushx(c,"mylist99","ppp")
redis_rpush(c,"mylist99","s0")
redis_lpush(c,"mylist99","s1")
redis_rpushx(c,"mylist99","s2")  # returns 3
```

rpush

Description: Adds all the specified values to the tail (right) of the list. Creates the list if the key didn't exist.

Parameters
number: connection

string: key name
string or array: the string value to push in key, or if is an array, it's containing all values.

Return value
number: The new length of the list in case of success, `-1` on error (if the key exists and is not a list).

Example 1: Using rpush

```
@load "redis"
BEGIN{
  c=redis_connect()
  redis_del(c,"mylist")
  C[1]="Hello";C[2]="World"
  print redis_rpush(c,"mylist",C)
  redis_lrange(c,"mylist",AR,0,-1)
  for(i in AR) {
    print i") "AR[i]
  }
  C[1]="push1";C[2]="push2"
  print redis_lpush(c,"mylist",C)
  delete AR
  redis_lrange(c,"mylist",AR,0,-1)
  for(i in AR) {
    print i") "AR[i]
  }
  redis_close(c)
}
```

Output

```
2
1) Hello
2) World
4
1) push2
2) push1
3) Hello
4) World
```

Example 2: Using rpush

```
@load "redis"
BEGIN{
  c=redis_connect()
  redis_del(c,"mylist")
  r=redis_rpush(c,"mylist","Hello")
  print r
  r=redis_rpush(c,"mylist","World")
  print r
  redis_lrange(c,"mylist",AR,0,-1)
  for(i in AR) {
    print i") "AR[i]
  }
  redis_close(c)
}
```

Output

```
1
2
1) Hello
2) World
```

lrange

Description: Returns the specified elements of the list stored at the specified key in the range [start, end]. start and stop are interpreted as indices: 0 the first element, 1 the second … -1 the last element, -2 the penultimate …

Parameters
number: connection
string: key name
array: for the result. It will contain the values in specified range
number: start
number: end

Return value
`1` on success, `0` in case of empty list or no exists

Example: Using lrange

```
# it range includes all values.
redis_lrange(c,"list1",AR,0,-1)
```

lrem

Description: Removes the first count occurences of the value element from the list. If count is zero, all the matching elements are removed. If count is negative, elements are removed from tail to head.

Parameters
number: connection
string: key name
number: count
string: value

Return value
number: the number of removed elements, `-1` on error

Example: Using lrem

```
@load "redis"
BEGIN{
  c=redis_connect()
  redis_del(c,"list1")
  redis_lpush(c,"list1","AA")
  redis_lpush(c,"list1","BB")
  redis_lpush(c,"list1","CC")
  redis_lpush(c,"list1","BB")
  redis_lrange(c,"list1",AR,0,-1)
  for(i in AR) {
    print i": "AR[i]
  }
   # count is 4 but removes only two (existing values)
  ret=redis_lrem(c,"list1",4,"BB")
  print "return redis_lrem="ret
  if(ret==-1) print ERRNO
  delete AR
  ret=redis_lrange(c,"list1",AR,0,-1)
  for(i in AR) {
     print i": "AR[i]
  }
  redis_close(c)
}
```

lset

Description: Set the list at index with the new value.

Parameters
number: connection
string: key name
number: index
string: value

Return value
`1` if the new value is setted. `-1` on error (if the index is out of range, or data type identified by key is not a list).

Example: Using lset

```
@load "redis"
BEGIN{
 c=redis_connect()
 # set "28" in list2 with index 7
 ret=redis_lset(c,"list2",7,"28")
 print "lset returns "ret
 redis_close(c)
}
```

ltrim

Description: Trims an existing list so that it will contain only a specified range of elements. It is recommended that you consult on possibles uses[10] of this function in the main page of Redis project.

Parameters
number: connection
string: key name

[10]http://redis.io/commands/ltrim

number: start
number: stop

Return value

`1` on success, `-1` on error (by example a WRONGTYPE Operation). Out of range indexes will not produce an error.

Example: Using ltrim

```
@load "redis"
BEGIN{
  c=redis_connect()
  ret=redis_lrange(c,"list2",AR,0,-1)
  if(ret==1) {
    print "--->Values in list2<---"
    for(i in AR) {
      print i": "AR[i]
    }
    ret=redis_ltrim(c,"list2",2,5)
    if(ret==1) {
      delete AR
      redis_lrange(c,"list2",AR,0,-1)
      print "--->Values in list2 after applying",
  print "'ltrim'<---"
      for(i in AR) {
        print i": "AR[i]
      }
    }
  }
  redis_close(c)
}
```

Output

```
--->Values in list2<---
1: 96
2: 5
3: 63
4: 60
5: 12
6: 69
7: 162
--->Values in list2 after applying 'ltrim'<---
1: 63
2: 60
3: 12
4: 69
```

brpop

Description: Is a blocking list pop primitive. Pops elements from the tail of a list. To see Redis site[11] for a more detailed explanation

Parameters
number: connection
string or array: key name (the list name), or an array containing the list names
array: for the results. If a elemment be popped then, this array is two-element where the first element is the name of the key where it value was popped and the second element is the value of the popped element
number: timeout

Return value
`1`, if popped a element. A `string null` when no element could be popped and the timeout expired.

[11]http://redis.io/commands/brpop

Example: Using brpop

```
@load "redis"
BEGIN {
  c=redis_connect()
  redis_del(c,"listb")
  redis_lpush(c,"listb","hello")
  redis_lpush(c,"listb","Sussan")
  redis_lpush(c,"listb","nice")
  LIST[1]="listbbb"; LIST[2]="listbb"; LIST[3]="listb"
   # knowing that "listbbb" and "listbb" does not exist
   # brpop will get a element from 'listb'
  redis_brpop(c,LIST,AR,10) # return is 1
  for(i in AR) {
    print i": "AR[i]
  }
  redis_close(c)
}
```

Output

```
1: listb
2: hello
```

blpop

Description: Is a blocking list pop primitive. Pops elements from the head of a list. To see Redis site[12] for a more detailed explanation

Parameters
number: connection
string or array: key name (the list name), or an array containing the list names

[12]http://redis.io/commands/blpop

array: for the results. If a elemment be popped then, this array is two-element where the first element is the name of the key where it value was popped and the second element is the value of the popped element.
number: timeout

Return value
`1`, if popped a element. A `string null` when no element could be popped and the timeout expired.

Example: Using blpop

```
# the same example code in brpop
# ...
redis_blpop(c,LIST,AR,10) # returns is 1
```

Output

```
1: listb
2: nice
```

llen

Description: Returns the size of a list identified by Key.

If the list didn't exist or is empty, the command returns `0`. If the data type identified by Key is not a list, the command returns `-1`.

Parameters
number: connection
string: key name

Return value
number: the size of the list identified by key, `0` if the key no exist or is empty, `-1` on error (if the data type identified by key is not list)

Example: Using llen

```
print "Length of 'mylist': "redis_llen(c,"mylist")
```

Sets Functions

1. sadd - Adds one or more members to a set
2. scard - Gets the number of members in a set
3. sdiff - Subtracts multiple sets
4. sdiffstore - Subtracts multiple sets and store the resulting set in a key
5. sinter - Intersects multiple sets
6. sinterstore - Intersects multiple sets and store the resulting set in a key
7. sismember - Determines if a given value is a member of a set
8. smembers - Gets all the members in a set
9. smove - Moves a member from one set to another
10. spop - Removes and returns one or more random members from a set
11. sscan - Iterates elements of Sets types
12. srandmember - Gets one or multiple random members from a set
13. srem - Removes one or more members from a set
14. sunion - Adds multiple sets
15. sunionstore - Adds multiple sets and store the resulting set in a key

srandmember

Description: Get one or multiple random members from a set, (not remove it). See Redis site[13] for the use of additional parameters.

Parameters
number: connection

[13]http://redis.io/commands/srandmember

string: key name
(optional) number: `count` distinct elements if count is positive. If count is negative, the number of elements is the absolute value of the specified count and can obtain the same element multiple times in the result
(optional) array: containing results, when count parameter is used.

Return value
string: the randomly selected element, or `null string` if key not exist. If count is used, returns `1` and the array parameter, will contain the results.

Example: Using srandmember

```
@load "redis"
BEGIN {
  c=redis_connect()
  redis_del(c,"myset")
  A[1]="55"; A[2]="c16"; A[3]="89"; A[4]="c15"
  redis_sadd(c,"myset",A)
  r=redis_smembers(c,"myset",MEMB)
  if(r!=-1) {
     print "Members in set 'myset'"
     for( i in MEMB) {
        print MEMB[i]
     }
  }
  print "srandmember gets:",
        redis_srandmember(c,"myset")
  print "Members in set 'myset',"
  print "after to applicate 'srandmember' function."
  delete MEMB
  r=redis_smembers(c,"myset",MEMB)
  print "smembers returns: "r
  for( i in MEMB) {
        print MEMB[i]
  }
  r=redis_srandmember(c,"myset",3,B)
   # Members obtained using srandmember with
   # the additional count argument
  for( i in B) {
      print "              "B[i]
  }
  redis_close(c)
}
```

Output

```
Members in set 'myset'
55
c15
89
c16
srandmember gets: c16
Members in set 'myset',
after to applicate 'srandmember' function.
smembers returns: 1
55
c15
89
c16
          55
          c16
          c15
```

spop

Description: Removes and returns a random member from a set

Parameters
number: connection
string: key name

Return value
string: the removed element, or `null string` if key not exist.

Example: Using spop

```
BEGIN {
  c=redis_connect()
  redis_del(c,"myset")
  A[1]="55"; A[2]="c16"; A[3]="89"; A[4]="c15"
  redis_sadd(c,"myset",A)
  r=redis_smembers(c,"myset",MEMB)
  if(r!=-1) {
     print "Members in set 'myset'"
     for( i in MEMB) {
        print MEMB[i]
     }
  }
  print "spop gets: "redis_spop(c,"myset")
  print "Members in set 'myset',"
  print "after to applicate 'spop' function."
  delete MEMB
  r=redis_smembers(c,"myset",MEMB)
  print "smembers returns: "r
  for( i in MEMB) {
        print MEMB[i]
  }
  redis_close(c)
}
```

Output

```
Members in set 'myset'
55
89
c15
c16
spop gets: 55
Members in set 'myset',
after to applicate 'spop' function.
smembers returns: 1
89
c15
c16
```

sdiff

Description: Subtract multiple sets

Parameters
number: connection
array: containing set names
array: containing the members (strings) of the result

Return value
number: `1` on sucess, `-1` on error.

Example: Using sdiff

```
redis_del(c,"myset1")
A[1]="55"; A[2]="c16"; A[3]="89"; A[4]="c15"
redis_sadd(c,"myset1",A)
redis_del(c,"myset2")
delete A
A[1]="89"
redis_sadd(c,"myset2",A)
redis_del(c,"myset3")
delete A
A[1]="9"; A[2]="c16"; A[3]="89"
redis_sadd(c,"myset3",A)
delete A
A[1]="myset1"; A[2]="myset2"; A[3]="myset3"
redis_sdiff(c,A,RE)
 # expected members in array RE: 55, c15
```

sinter

Description: Obtains the intersection of the given sets.

Parameters
number: connection
array: containing set names
array: containing the members (strings) of the result

Return value
number: `1` on sucess, `-1` on error.

Example: Using sinter

```
redis_del(c,"myset1")
A[1]="55"; A[2]="c16"; A[3]="89"; A[4]="c15"
redis_sadd(c,"myset1",A)
redis_del(c,"myset2")
delete A
A[1]="89"
redis_sadd(c,"myset2",A)
redis_del(c,"myset3")
delete A
A[1]="9"; A[2]="c16"; A[3]="89"
redis_sadd(c,"myset3",A)
delete A
A[1]="myset1"; A[2]="myset2"; A[3]="myset3"
redis_sinter(c,A,RE)
 # expected members in array RE: 89
```

sunion

Description: Obtains the union of the given sets.

Parameters
number: connection
array: containing set names
array: containing the members (strings) of the result

Return value
number: `1` on sucess, `-1` on error.

Example: Using sunion

```
redis_del(c,"myset1")
A[1]="55"; A[2]="c16"; A[3]="89"; A[4]="c15"
redis_sadd(c,"myset1",A)
redis_del(c,"myset2")
delete A
A[1]="89"
redis_sadd(c,"myset2",A)
redis_del(c,"myset3")
delete A
A[1]="9"; A[2]="c16"; A[3]="89"
redis_sadd(c,"myset3",A)
delete A
A[1]="myset1"; A[2]="myset2"; A[3]="myset3"
redis_sunion(c,A,RE)
 # expected members in array RE: 55, c15, c16, 89, 9
```

sunionstore

Description: Adds multiple sets and store the resulting set in a key.

Parameters
number: connection
string: new key name (a new set), where is stored the result.
array: containing set names

Return value
number: the number of elements in the resulting set, or `-1` on error.

Example: Using sunionstore

```
redis_del(c,"myset1")
A[1]="55"; A[2]="c16"; A[3]="89"; A[4]="c15"
redis_sadd(c,"myset1",A)
redis_del(c,"myset2")
delete A
A[1]="89"
redis_sadd(c,"myset2",A)
redis_del(c,"myset3")
delete A
A[1]="9"; A[2]="c16"; A[3]="89"
redis_sadd(c,"myset3",A)
delete A
A[1]="myset1"; A[2]="myset2"; A[3]="myset3"
redis_sunionstore(c,"mysetUnion",A)
 # expected members in set mysetUnion:
 # 55, c15, c16, 89, 9
```

sdiffstore

Description: Substracts multiple sets and store the resulting set in a key.

Parameters
number: connection
string: new key name (a new set), where is stored the result.
array: containing set names

Return value
number: the number of elements in the resulting set, or `-1` on error.

Example: Using sdiffstore

```
@load "redis"
BEGIN{
  c=redis_connect()
  print ERRNO
  redis_del(c,"myset1")
  A[1]="55"; A[2]="c16"; A[3]="89"; A[4]="c15"
  redis_sadd(c,"myset1",A)
  redis_del(c,"myset2")
  delete A
  A[1]="89"
  redis_sadd(c,"myset2",A)
  redis_del(c,"myset3")
  delete A
  A[1]="9"; A[2]="c16"; A[3]="89"
  redis_sadd(c,"myset3",A)
  delete A
  A[1]="myset1"; A[2]="myset2"; A[3]="myset3"
   # the next sdiffstore should returns 2
  redis_sdiffstore(c,"mysetDiff",A)
   # expected members in set mysetDiff: 55,c15
   #
   # for to show the results
  ret=redis_smembers(c,"mysetDiff",MEMB)
   # 'ret' contains the return of 'smembers' and the
   # array MEMB contains the results
  for(i in MEMB) {
    print MEMB[i]
  }
  redis_close(c)
```

}

sinterstore

Description: Intersects multiple sets and store the resulting set in a key.

Parameters
number: connection
string: new key name (a new set), where is stored the result.
array: containing set names

Return value
number: the number of elements in the resulting set, or `-1` on error.

Example: Using sinterstore

```
redis_del(c,"myset1")
A[1]="55"; A[2]="c16"; A[3]="89"; A[4]="c15"
redis_sadd(c,"myset1",A)
redis_del(c,"myset2")
delete A
A[1]="89"
redis_sadd(c,"myset2",A)
redis_del(c,"myset3")
delete A
A[1]="9"; A[2]="c16"; A[3]="89"
redis_sadd(c,"myset3",A)
delete A
A[1]="myset1"; A[2]="myset2"; A[3]="myset3"
redis_sinterstore(c,"mysetInter",A)
 # expected members in set mysetInter: 89
```

sscan

Description: iterates elements of Sets types. Please read how it works from Redis sscan[14] command.

[14]http://redis.io/commands/sscan

Parameters
number: connection
string: key name
number: the cursor
array: for to hold the results
(optional) string: for to `match` a given glob-style pattern, similarly to the behavior of the `keys` function that takes a pattern as only argument

Return value
`1` on success or `0` on the last iteration (when the returned cursor is equal `0`). Returns `-1` on error (by example a WRONGTYPE Operation).

Example: **Using sscan**

```
@load "redis"
BEGIN{
  c=redis_connect()
  num=0
  while(1){
   ret=redis_sscan(c,"myset",num,AR)
   if(ret==-1){
    print ERRNO
    redis_close(c)
    exit
   }
   if(ret==0){
     break
   }
   n=length(AR)
   for(i=2;i<=n;i++) {
     print AR[i]
   }
   num=AR[1]  # AR[1] contains the cursor
   delete(AR)
  }
  for(i=2;i<=length(AR);i++) {
     print AR[i]
  }
  redis_close(c)
}
```

sadd

Description: Add one or more members to a set.

Parameters
number: connection
string: key name
string or array: containing the value, and if it is an array containing the set of values

Return value
`the number of members` added to the set in this operation. Returns `-1` on error (by example a WRONGTYPE Operation).

Example: Using sadd

```
@load "redis"
BEGIN {
  c=redis_connect()
  redis_del(c,"myset")
  r=redis_sadd(c,"myset","c15")
  print r
  A[1]="55"; A[2]="c16"; A[3]="89"
  print redis_sadd(c,"myset",A)
  r=redis_smembers(c,"myset",MEMB)
  if(r!=-1) {
     print "Members in set 'myset'"
     for( i in MEMB) {
        print MEMB[i]
     }
  }
  redis_close(c)
}
```

Output

```
1
3
Members in set 'myset'
89
c16
55
c15
```

srem

Description: Remove one or more members from a set.

Parameters
number: connection
string: key name
string or array: containing the member (a string), and if it is an array containing the set of members (one or more strings)

Return value
number: the number of members that were removed from the set. Returns `-1` on error.

Example: Using srem

```
redis_del(c,"myset")
A[1]="55"; A[2]="c16"; A[3]="89"; A[4]="c26"; A[5]="12"
redis_sadd(c,"myset",A)
r1=redis_srem(c,"myset","89")
B[1]=55; B[2]="c16"; B[3]="12"
r2=redis_srem(c,"myset",B)
print "r1="r1" - r2="r2
redis_smembers(c,"myset",MEMB)
 # expected members in 'myset': c26
```

sismember

Description: Determines if a given value is a member of a set.

Parameters
number: connection
string: key name, (the set)
string: member

Return value
number: `1` if the element is a member of the set. `0` if the element is not a member of the set, or if key does not exist.

Example: Using sismember

```
redis_del(c,"myset")
A[1]="55"; A[2]="c16"; A[3]="89"; A[4]="c26"; A[5]="12"
redis_sadd(c,"myset",A)
redis_sismember(c,"myset","c26") # returns 1
redis_sismember(c,"myset","66") # returns 0
```

smove

Description: Move a member from one set to another.

Parameters
number: connection
string: key name, the source set
string: key name, the destination set
string: member

Return value
`1` if the elemment is moved, `0` if the element is not a member of source and no operation was performed. Returns `-1` on error.

Example: Using smove

```
redis_del(c,"myset")
A[1]="55"; A[2]="c16"; A[3]="89"; A[4]="c26"; A[5]="12"
redis_sadd(c,"myset",A)
 # execute "smove" and display its return
print "Member 'c26' from 'myset' to 'newset':",
      redis_smove(c,"myset","newset","c26")
 # now, the expected return value is 0
print "Member 'ccc' from 'myset' to 'newset':",
      redis_smove(c,"myset","newset","ccc")
```

scard

Description: Gets the cardinality (number of elements) of the set.

Parameters
number: connection
string: key name

Return value
`the cardinality` or 0 if key does not exist. `-1` on error.

Example: Using scard

```
print "Cardinality of 'myset': "redis_scard(c,"myset")
```

smembers

Description: Gets all the members in a set.

Parameters
number: connection
string: key name
array: will contain the results, a set of strings.

Return value
`1` on success, `-1` on error.

```
To see example `sadd function`
```

Sorted Sets Functions

1. zadd - Adds one or more members to a sorted set or updates its score if it already exists
2. zcard - Gets the number of members in a sorted set
3. zcount - Counts the members in a sorted set with scores between the given values
4. zincrby - Increments the score of a member in a sorted set
5. zinterstore - Intersects multiple sorted sets and store the resulting sorted set in a new key
6. zlexcount - Returns the number of elements with a value in a specified range, forcing lexicographical ordering
7. zrange - Returns a range of members in a sorted set. The elements are sorted from the lowest to the highest score
8. zrangebylex - Returns all the elements with a value in a specified range, forcing a lexicographical ordering
9. zrangebyscore - Returns all the elements with a score between min and max specified
10. zrangeWithScores - Return a range of members in a sorted set, by score
11. zrank - Determines the index of a member in a sorted set
12. zrem - Removes one or more members from a sorted set
13. zremrangebylex - Removes all elements in the sorted set between the lexicographical range specified by min and max
14. zremrangebyrank - Removes all elements in the sorted set with rank into a specified rango
15. zremrangebyscore - Removes all elements in the sorted set with a score into a specified range
16. zrevrange - Returns a specified range of elements in the sorted set. The elements are sorted from highest to lowest score

17. zrevrangebyscore - Returns all the elements in the sorted set with a score between max and min.
18. zrevrangeWithScores - Executes zrevrange with the option ‘withscores’, gettings the scores together with the elements
19. zrevrank - Returns the rank of a member in the sorted set, with the scores ordered from high to low
20. zscan - Iterates elements of Sorted Set types
21. zscore - Gets the score associated with the given member in a sorted set
22. zunionstore - Adds multiple sorted sets and stores the resulting sorted set in a new key

zcard

Description: Gets the number of members in a sorted set.

Parameters
number: connection
string: key name

Return value
number: `the cardinality` or number de elements, `0` if key does not exist. `-1` on error.

Example: Using zcard

```
print "Cardinality of 'zmyset':",
       redis_zcard(c,"zmyset")
```

zrevrank

Description: Returns the rank of a member in the sorted set, with the scores ordered from high to low. The rank (or index) is 0-based, which means that the member with the highest score has rank 0.

Parameters
number: connection
string: key name
string: member name

Return value
number: the rank of member, if member exists in the sorted set. Returns `null string` if member does not exist in the sorted set or key does not exist. `-1` on error.

Example: Using zrevrank

```
@load "redis"
BEGIN{
  c=redis_connect()
  redis_del(c,"myzset")
  A[1]="1"; A[2]="one"; A[3]="2"; A[4]="two"
  A[5]="3"; A[6]="three"; A[7]="4"; A[8]="four"
  redis_zadd(c,"myzset",A)
  redis_zrevrank(c,"myzset","four") # returns 0
  redis_zrevrank(c,"myzset","seven")
   # and returns null string
  redis_zrevrank(c,"myzset","two") # returns 2
  redis_close(c)
}
```

zcount

Description: Count the members in a sorted set with a score between min and max (two values given as arguments).

Parameters
number: connection
string: key name
number: min value *number*: max value

Return value
number: the number of elements in the specified score range, `0` if key does not exist. `-1` on error (by example a WRONGTYPE Operation).

Example: Using zcount

```
redis_del(c,"zmyset")
r1=redis_zadd(c,"zmyset",1,"one")
r2=redis_zadd(c,"zmyset",1,"uno")
AR[1]="2"; AR[2]="two"; AR[3]="3"; AR[4]="three"
r3=redis_zadd(c,"zmyset",AR)
print r1, r2, r3
print "Zcount with score between 1 and 2:",
      redis_zcount(c,"zmyset",1,2) # returns 3
```

zinterstore

Description: Intersects multiple sorted sets and store the resulting sorted set in a new key. To see Redis site[15] for to know how use additionals parameters "weights" and "aggregate"

Parameters
number: connection
string: new key name (a new sorted set), where is stored the result.
array: containing the sorted sets names
`and optionally...` *array*: containing the weights *string*: containing "agregate sum|min|max"

Return value
number: the number of elements in the resulting sorted set at destination, or `-1` on error

[15] http://redis.io/commands/zinterstore

Example: Using zinterstore

```
redis_del(c,"zmyset1")
A[1]="1"; A[2]="one"; A[3]="3"; A[4]="three"
A[5]="5"; A[6]="five"
redis_zadd(c,"zmyset1",A)
redis_del(c,"zmyset2")
delete A
A[1]="3"; A[2]="three"; A[3]="4"; A[4]="four"
redis_zadd(c,"zmyset2",A)
redis_del(c,"zmyset3")
delete A
A[1]="3"; A[2]="three"; A[3]="4"; A[4]="four"
A[5]="5"; A[6]="five"
redis_zadd(c,"zmyset3",A)
delete A
A[1]="zmyset1"; A[2]="zmyset2"; A[3]="zmyset3"
redis_zinterstore(c,"zmysetInter",A)
# expected members in the sorted set zmysetInter:
# 'three' with score 9
#
W[1]=2; W[2]=3; W[3]=4
redis_zinterstore(c, \
        "zmysetInterWeights",A,W,"aggregate sum")
# 'three' with score 27
redis_zinterstore(c, \
        "zmysetInterWeights", A,W, "aggregate min")
# 'three' with score 6
```

zunionstore

Description: Adds multiple sorted sets and store the resulting sorted set in a new key. To see Redis site[16] for to know how use

[16]http://redis.io/commands/zunionstore

additionals parameters “weights” and “aggregate”

Parameters
number: connection
string: new key name (a new sorted set), where is stored the result.
array: containing the sorted sets names
and optionally:
array: containing the weights *string*: containing “agregate sum|min|max”

Return value
number: the number of elements in the resulting sorted set at destination, or `-1` on error

Example: Using zunionstore

```
redis_del(c,"zmyset1")
A[1]="1"; A[2]="one"; A[3]="3"; A[4]="three"
A[5]="5"; A[6]="five"
redis_zadd(c,"zmyset1",A)
redis_del(c,"zmyset2")
delete A
A[1]="3"; A[2]="three"; A[3]="4"; A[4]="four"
redis_zadd(c,"zmyset2",A)
redis_del(c,"zmyset3")
delete A
A[1]="3"; A[2]="three"; A[3]="4"; A[4]="four"
A[5]="5"; A[6]="five"
redis_zadd(c,"zmyset3",A)
delete A
A[1]="zmyset1"; A[2]="zmyset2"; A[3]="zmyset3"
W[1]=2; W[2]=3; W[3]=4
redis_zunionstore(c,"zmysetUW",A,W,"aggregate sum")
 # one,2  three,27  four,28  five,30
redis_zunionstore(c,"zmysetUW",A,W,"aggregate min")
 # one,2  three,6  four,12  five,10
```

zrange

Description: Returns a range of members in a sorted set. The members are considered to be ordered from the lowest to the highest score.

Parameters
number: connection
string: key name
array: array name for the results
number: start of range
number: stop of range

Return value
`1` on success, `0` if the result is empty, `-1` on error (by example a WRONGTYPE Operation)

Example: Using zrange

```
redis_del(c,"zmyset")
AR[1]="2"; AR[2]="two"; AR[3]="3"; AR[4]="three";
AR[5]="1"; AR[6]="one"; AR[7]="1"; AR[8]="uno"
redis_zadd(c,"zmyset",AR)
redis_zrange(c,"zmyset",RET,6,-1)
 # returns 0, and array RET is empty
redis_zrange(c,"zmyset",RET,1,2)
 # returns 1, and array RET contains members
 #
 # shows the results
for( i in RET ) {
  print RET[i]
}
```

zrevrange

Description: Returns the specified range of elements in the sorted set. The elements are considered to be ordered from the highest to the lowest score

Parameters
number: connection
string: key name
array: array name for the results
number: start of range
number: stop of range

Return value
`1` on success, `0` if the result is empty, or the key not exists. `-1` on error (by example a WRONGTYPE Operation)

Example: **Using zrevrange**

```
@load "redis"
BEGIN{
  c=redis_connect()
  redis_del(c,"myzset")
  redis_zadd(c,"myzset","1","t7")
  redis_zadd(c,"myzset","2","t0")
  redis_zadd(c,"myzset","5","t1")
  redis_zadd(c,"myzset","4","t9")
   #  zrevrange(c,"myzset",RES,6,-1)  # returns 0
  print redis_zrevrange(c,"myzset",RES,0,-1)# returns 1
  for (i in RES) {
     print i": "RES[i]
  }
  redis_close(c)
}
```

Output

```
1
1: t1
2: t9
3: t0
4: t7
```

zrevrangeWithScores

Description: Returns the specified range of elements in the sorted set. The elements are considered to be ordered from the highest to the lowest score. Returns the scores of the elements together with the elements.

Parameters
number: connection
string: key name
array: array name for the results. It will contain value1,score1,..., valueN,scoreN instead value1,...valueN
number: start of range
number: stop of range

Return value
`1` on success, `0` if the result is empty, or the key not exists. `-1` on error (by example a WRONGTYPE Operation)

Example: Using zrevrangeWithScores

```
@load "redis"
BEGIN{
  c=redis_connect()
  redis_del(c,"myzset")
  redis_zadd(c,"myzset","1","t7")
  redis_zadd(c,"myzset","2","t0")
  redis_zadd(c,"myzset","5","t1")
  redis_zadd(c,"myzset","4","t9")
  redis_zrevrangeWithScores(c, \
       "myzset",RES,1,-1)  # returns 1
  for (i in RES) {
     print i": "RES[i]
  }
  redis_close(c)
}
```

Output

```
1: t9
2: 4
3: t0
4: 2
5: t7
6: 1
```

zlexcount

Description: When all the elements in a sorted set are inserted with the same score, returns the number of elements with a value between min and max specified, forcing lexicographical ordering.

To see the Redis command[17] to know how to specify intervals and others details.

Parameters
number: connection
string: key name
string: min
string: max

Return value
number: the number of elements in the specified score range. `-1` on error

Example: Using zlexcount

```
@load "redis"
BEGIN{
  c=redis_connect()
  redis_del(c,"zset")
  A[1]="0"; A[2]="a"; A[3]="0"; A[4]="b"; A[5]="0"
  A[6]="c"; A[7]="0"; A[8]="d"; A[9]="0"; A[10]="e"
  A[11]="0"; A[12]="f"; A[13]="0"; A[14]="g"
  redis_zadd(c,"zset",A)
  redis_zlexcount(c,"zset","-","+")  # return 7
  redis_zlexcount(c,"zset","[b","(d")  # returns 2
  redis_close(c)
}
```

zremrangebylex

Description: When all the elements in a sorted set are inserted with the same score, removes all elements in the sorted set between the lexicographical range specified by min and max To see the Redis command[18] to know how to specify intervals and others details.

[17]http://redis.io/commands/zlexcount
[18]http://redis.io/commands/zremrangebylex

Parameters
number: connection
string: key name
string: min
string: max

Return value
number: the number of elements removed. `-1` on error

Example: Using zremrangebylex

```
@load "redis"
BEGIN{
  c=redis_connect()
  redis_del(c,"myzset")
  redis_zadd(c,"myzset","2","one")
  redis_zadd(c,"myzset","2","two")
  redis_zadd(c,"myzset","2","three")
  redis_zremrangebylex(c, \
       "myzset","[g","(tkz") # returns 2
  redis_zrange(c,"myzset",RES,0,-1) # returns 1
  for (i in RES) {
    print i": "RES[i]
  }
}
```

zremrangebyscore

Description: Removes all elements in the sorted set with a score into a specified range with a min and a maxm (inclusive). To see the Redis command[19] to know how to specify intervals and others details.

Parameters
number: connection

[19] http://redis.io/commands/zremrangebyscore

string: key name
string: min
string: max

Return value
number: the number of elements removed. `-1` on error

Example: Using zremrangebyscore

```
@load "redis"
BEGIN{
  c=redis_connect()
  redis_del(c,"myzset")
  redis_zadd(c,"myzset","1","t7")
  redis_zadd(c,"myzset","2","t0")
  redis_zadd(c,"myzset","5","t1")
  redis_zadd(c,"myzset","4","t9")
   # redis_zremrangebyscore(c,"myzset","-inf","(5")
   # returns 3
   # redis_zremrangebyscore(c,"myzset",1,3)# returns 2
  redis_zremrangebyscore(c,"myzset","(2","4")#returns 1
  redis_zrangeWithScores(c,"myzset",RES,0,-1)
   # returns 1, and the results in array RES
  for (i in RES) {
     print i": "RES[i]
  }
  redis_close(c)
}
```

Output

```
1: t7
2: 1
3: t0
4: 2
5: t1
6: 5
```

zremrangebyrank

Description: Removes all elements in the sorted set with rank between start and stop. Both start and stop are `0` -based indexes with `0` being the element with the lowest score.

Parameters
number: connection
string: key name
string: min
string: max

Return value
number: the number of elements removed. `-1` on error

Example: Using zremrangebyrank

```
@load "redis"
BEGIN{
  c=redis_connect()
  redis_del(c,"myzset")
  redis_zadd(c,"myzset","1","t7")
  redis_zadd(c,"myzset","2","t0")
  redis_zadd(c,"myzset","5","t1")
  redis_zadd(c,"myzset","4","t9")
  redis_zremrangebyrank(c,"myzset",0,1) # returns 2
  redis_zrangeWithScores(c, \
      "myzset",RES,0,-1)  # returns 1
  for (i in RES) {
    print i": "RES[i]
  }
  redis_close(c)
}
```

Output

```
1: t9
2: 4
3: t1
4: 5
```

zrangebylex

Description: When all the elements in a sorted set are inserted with the same score, returns all the elements with a value between min and max specified, forcing a lexicographical ordering. To see the Redis command[20] to know how to specify intervals and others details.

Parameters
number: connection
string: key name
array: for the results, will be a list of elements with value in the specified range. *string*: min
string: max

Return value
`1` when obtains results,`0` when list empty (no elements in the score range) or the key name no exists, `-1` on error (by example a WRONGTYPE Operation)

[20]http://redis.io/commands/zrangebylex

Example: Using zrangebylex

```
c=redis_connect()
redis_del(c,"zset")
A[1]="0"; A[2]="a"; A[3]="0"; A[4]="b"; A[5]="0"
A[6]="c"; A[7]="0"; A[8]="d"; A[9]="0"; A[10]="e"
A[11]="0"; A[12]="f"; A[13]="0"; A[14]="g"
redis_zadd(c,"zset",A)  # returns 7
redis_zrangebylex(c,"zset",AR,"[aaa","(g") # returns 1
 # AR contains b,c,d,e,f
 # to show the result contained in array AR
for(i in AR){
  print i": "AR[i]
}
 # the next return is 0
redis_zrangebylex(c,"zset",AR,"[pau","(ra")
 # the array has not content
```

zrangebyscore

Description: Returns all the elements with a score between min and max specified. The elements are considered to be ordered from low to high scores. To see the Redis command[21] to know how to specify intervals and others details.

Parameters
number: connection
string: key name
array: for the results, will be a list of elements in the specified score range. *string*: min
string: max

Return value
`1` when obtains results,`0` when list empty (no elements in the

[21]http://redis.io/commands/zrangebyscore

score range) or the key name no exists, `-1` on error (by example a WRONGTYPE Operation)

Example: Using zrangebyscore

```
@load "redis"
BEGIN{
  c=redis_connect()
  redis_del(c,"myzset")
  redis_zadd(c,"myzset","1","one")
  redis_zadd(c,"myzset","2","two")
  redis_zadd(c,"myzset","3","three")
  redis_zrangebyscore(c,"myzset",RES,"-inf","+inf")
    # returns 1
  for (i in RES) {
     print i": "RES[i]
  }
  delete RES
  redis_zrangebyscore(c,"myzset",RES,1,2) # returns 1
  for (i in RES) {
     print i": "RES[i]
  }
  redis_zrangebyscore(c,"myzset",RES,"(1","(2")
  # returns 0
  redis_close(c)
}
```

zrevrangebyscore

Description: Returns all the elements in the sorted set with a score between max and min (including elements with score equal to max or min). The elements are sorted from highest to lowest score. To see the Redis command[22] to know how to specify intervals and others details.

[22]http://redis.io/commands/zrevrangebyscore

Parameters
number: connection
string: key name
array: for the results, will be a list of elements in the specified score range. *string*: max
string: min

Return value
`1` when obtains results,`0` when list empty (no elements in the score range) or the key name no exists, `-1` on error (by example a WRONGTYPE Operation)

Example: Using zrevrangebyscore

```
@load "redis"
BEGIN{
  c=redis_connect()
  redis_del(c,"myzset")
  redis_zadd(c,"myzset","1","one")
  redis_zadd(c,"myzset","2","two")
  redis_zadd(c,"myzset","3","three")
  redis_zrevrangebyscore(c,"myzset",RES,"+inf","-inf")
   # returns 1
  for (i in RES) {
     print i": "RES[i]
  }
  delete RES
  redis_zrevrangebyscore(c,"myzset",RES,"2","1")
   # returns 1
  print
  for (i in RES) {
     print i": "RES[i]
  }
  redis_close(c)
}
```

Output

```
1: three
2: two
3: one

1: two
2: one
```

zrangeWithScores

Description: Returns the scores of the elements together with the elements in a range, in a sorted set.

Parameters
number: connection
string: key name
string: array name for the results. It will contain value1,score1,..., valueN,scoreN instead value1,...valueN
number: start of range
number: stop of range

Return value
number: `1` on success, `0` if the result is empty, `-1` on error (by example a WRONGTYPE Operation)

Example: Using zrangeWithScores

```
redis_del(c,"zmyset")
AR[1]="2"; AR[2]="two"; AR[3]="3"; AR[4]="three";
AR[5]="1"; AR[6]="one"; AR[7]="1"; AR[8]="uno"
redis_zadd(c,"zmyset",AR)
redis_zrange(c,"zmyset",RET,0,-1) #  gets only elements
 # use RET ... and then remove
delete RET
redis_zrangeWithScores(c,"zmyset",RET,0,-1)
```

```
 # gets all elements with their respectives scores
 #
 # shows the results
for( i in RET ) {
  print RET[i]
}
```

zrem

Description: Removes one or more members from a sorted set.

Parameters
number: connection
string: key name
string or array: the member (a string) or the set of members that containing the array

Return value
number: The number of members removed from the sorted set, `-1` on error.

Example: Using zrem

```
redis_del(c,"zmyset")
AR[1]="2"; AR[2]="two"; AR[3]="3"; AR[4]="three"
AR[5]="1"; AR[6]="one"
redis_zadd(c,"zmyset",AR)
redis_zrem(c,"zmyset","three") # returns 1
R[1]="uno"; R[2]="two"; R[3]="five"
redis_zrem(c,"zmyset",R) # returns 2
```

zrank

Description: Determines the index or rank of a member in a sorted set.

Parameters
number: connection
string: key name
string: the member

Return value
`the rank of member`, if the member exists in the key. `string null`, if the member does not exist in the key or the key does not exist, `-1` on error.

Example: Using zrank

```
redis_del(c,"zmyset")
redis_zadd(c,"zmyset",1,"uno")
AR[1]="2"; AR[2]="two"; AR[3]="3"; AR[4]="three"
AR[5]="1"; AR[6]="one"
redis_zadd(c,"zmyset",AR)
redis_zrank(c,"zmyset","three") # returns 3
redis_zrank(c,"zmyset","one") # returns 0
```

zscore

Description: Gets the score associated with the given member in a sorted set.

Parameters
number: connection
string: key name
string: the member

Return value
`the score of member represented as string`, if the member exists in the key. `string null`, if the member does not exist in the key or the key does not exist. `-1` on error.

Example: Using zscore

```
redis_del(c,"zmyset")
redis_zadd(c,"zmyset",1,"uno")
AR[1]="2"; AR[2]="two"; AR[3]="3"; AR[4]="three"
AR[5]="1"; AR[6]="one"
redis_zadd(c,"zmyset",AR)
redis_zscore(c,"zmyset","three") # returns 3
redis_zscore(c,"zmyset","one") # returns 1
```

zincrby

Description: Increments the score of a member in a sorted set.

Parameters
number: connection
string: key name
number: the increment
string: the member

Return value
number: the new score of member.

Example: Using zincrby

```
redis_del(c,"zmyset")
redis_zadd(c,"zmyset",1,"uno")
AR[1]="2"; AR[2]="two"; AR[3]="3"; AR[4]="three"
AR[5]="1"; A[6]="one"
redis_zadd(c,"zmyset",AR)
# redis_zincrby increments '3' the score of the
# member 'one' of key 'zmyset'
redis_zincrby(c,"zmyset",3,"one") # returns 4
```

zadd

Description: Adds one or more members to a sorted set or updates its score if it already exists.

Parameters
number: connection
string: key name
number: score *string or array*: containing the member, and if it is an array containing the set of score and members

Return value
`the number of elements` added to the sorted set, not including elements already existing. Returns `-1` on error (by example a WRONGTYPE Operation).

Example: Using zadd

```
@load "redis"
BEGIN {
  c=redis_connect()
  redis_del(c,"zmyset")
  r1=redis_zadd(c,"zmyset",1,"one")
  r2=redis_zadd(c,"zmyset",1,"uno")
  AR[1]="2"; AR[2]="two"; AR[3]="3"; AR[4]="three"
  r3=redis_zadd(c,"zmyset",AR)
  print r1, r2, r3
  redis_close(c)
}
```

Output

```
1 1 2
```

zscan

Description: iterates elements of Sets types. Please read how it works from Redis zscan[23] command.

Parameters
number: connection
string: key name
number: the cursor
array: for to hold the results
string (optional): for to `match` a given glob-style pattern, similarly to the behavior of the `keys` function that takes a pattern as only argument

Return value
`1` on success or `0` on the last iteration (when the returned cursor is equal `0`). Returns `-1` on error (by example a WRONGTYPE Operation).

Example: **Using zscan**

```
@load "redis"
BEGIN{
  c=redis_connect()
  num=0
  while(1){
   ret=redis_zscan(c,"myzset1",num,AR)
   if(ret==-1){
     print ERRNO
     redis_close(c)
     exit
   }
   if(ret==0){
     break
   }
```

[23] http://redis.io/commands/zscan

```
    n=length(AR)
    for(i=2;i<=n;i++) {
      print AR[i]
    }
    num=AR[1]  # AR[1] contains the cursor
    delete(AR)
   }
   for(i=2;i<=length(AR);i++) {
      print AR[i]
   }
   redis_close(c)
}
```

Pub/sub Functions

Recommended reading about the paradigm Pub/Sub[24] and the implemetation

- publish - Post a message to the given channel
- subscribe - Subscribes the client to the specified channels.
- psubscribe - Subscribes the client to the given patterns. Supported glob-style patterns.
- unsubscribe - Unsubscribes the client from the given channels, or from all of them if none is given.
- punsubscribe - Unsubscribes the client from the given patterns, or from all of them if none is given.
- getMessage - Way in which a subscriber consumes a message

publish

Description: Publish messages to channels.

Parameters
number: connection
string: a channel to publish to
string: a string messsage

Return value
number: the number of clients that received the message

[24]http://redis.io/topics/pubsub

Example: Using publish

```
redis_publish(c,"chan-1", "hello, world!")
# send message
```

subscribe

Description: Subscribe to channels.

Parameters
number: connection
string or array: the channel name or the array containing the names of channels
array(three elements): contains the strings returned: "message", the `channel name` and "1"

Return value
`1` on success, `-1` on error

Example: Using subscribe

```
redis_subscribe(c,"chan-2")  # returns 1,
# subscribes to chan-2
# array RET will contain "message",
# "chan-2", "1"
#
CH[1]="chan-1"
CH[2]="chan-2"
CH[3]="chan-3"
#
redis_subscribe(c,CH)  # returns 1,
# subscribes to chan-1, chan-2 and chan-3
```

unsubscribe

Description: Unsubscribes the client from the given channels, or from all of them if none is given.

Parameters
number: connection
string or array (This parameter could not be): the channel name or the array containing the names of channels

Return value
`1` on success, `-1` on error

Example: Using unsubscribe

```
redis_unsubscribe(c,"chan-2")
 # returns 1, unsubscribes to chan-2
CH[1]="chan-1"; CH[2]="chan-2"; CH[3]="chan-3"
 # unsubscribes to chan-1, chan-2 and chan-3
redis_unsubscribe(c,CH)  # returns 1
# unsubscribing from all the previously
# subscribed channels
```

punsubscribe

Description: Unsubscribes the client from the given patterns, or from all of them if none is given.

Parameters
number: connection
string or array (This parameter could not be): the pattern or the array containing the patterns.

Return value
`1` on success, `-1` on error

Example: Using punsubscribe

```
@load "redis"
BEGIN {
  c=redis_connect()
  redis_psubscribe(c,"ib*")
  redis_subscribe(c,"channel1")
  while(ret=redis_getMessage(c,A)) {
    for(i in A) {
      print i": "A[i]
    }
    if(A[4]=="exit" && A[3]=="ibi") {
      redis_punsubscribe(c,"ib*")
    }
    if(A[3]=="exit" && A[2]=="channel1") {
      break
    }
    delete A
  }
  redis_unsubscribe(c)
  redis_close(c)
}
```

psubscribe

Description: Subscribes the client to the given patterns. Supported glob-style patterns.

Parameters
number: connection
string or array: the pattern, or the array containing the patterns. *array(three elements)*: contains the strings returned: “pmessage”, the `channel name` and “1”

Return value
`1` on success, `-1` on error

Example: Using psubscribe

```
# subscribes to channels that match the pattern 'ib'
# to the begin of the name
redis_psubscribe(c,"ib*")  # returns 1
CH[1]="chan[ae]-1"
CH[2]="chan[ae]-2"
redis_psubscribe(c,CH)  # returns 1,
# subscribes to chana-1, chane-1, chana-2, chane-2
```

getMessage

Description: Gets a message from any of the subscribed channels, (based at hiredis API redisGetReply for to consume messages).

Parameters
number: connection
array(three or four elements): containing the messages received. When the subscription is by `subscribe` the strings are "message", `channel name` and `message`. While with subscription realized by `psubscribe` the strings are "pmessage", the `pattern channel name`, the `channel name` and `message`.

Return value
`1` on success, `-1` on error

Example mple: Using getMessage

```
A[1]="c1"
A[2]="c2"
ret=redis_subscribe(c,A)
while(ret=redis_getMessage(c,B)) {
   for(i in B){
     print i") "B[i]
   }
```

```
    delete B
}
```

Pipelining Functions

Recommended reading for to know as this is supported: Redis pipelining[25] and hiredis pipelining[26], who works in a more low layer.

- pipeline - To create a pipeline, allowing buffered commands
- getReply - To get or receive the result of each command buffered
- getReplyInfo - To get the result when the `info` command is the command buffered
- getReplyMass - To perform a massive insertion data

pipeline

Description: To create a pipeline, allowing buffered commands.

Parameters
number: connection

Return value *number*: `pipe handle` on success, `-1` on error

[25]http://redis.io/topics/pipelining
[26]https://github.com/redis/hiredis#pipelining

Example: Using pipeline

```
@load "redis"
BEGIN{
  c=redis_connect()
  p=redis_pipeline(c)  # 'p' is a new pipeline
   # The following SET commands are buffered
  redis_select(p,4) # changing db, using select
  redis_set(p,"newKey","newValue") # set command
  redis_type(p,"newKey") # type command
  redis_setrange(p,"newKey",6,"123") # setrange command
  redis_dump(p,"newKey") # dump command
  redis_keys(p,"n*",AR) # keys command
   # To execute all commands buffered, and store
   # the return values
  for( ; ERRNO=="" ; RET[++i]=redis_getReply(p,REPLY) )
    ;
  ERRNO=""
   # To use the value returned by 'redis_dump'
  redis_restore(c,"newKey1","0",RET[5])
   # Actually the array REPLY stores the result
   # the last command buffered. Then, for know the
   # result of 'redis_keys':
  for( j in REPLY ) {
    print j": "REPLY[j]
  }
  redis_close(c)
}
```

getReplyInfo

Description: This function is exactly like `getReply` with the only difference that has been designed for replies of the `info` command.

Parameters *number*: pipeline handle *array*: for results of the `info` command

Return value *string or number*: allways `1` or `-1` on error (if not exist results buffered)

Example: Using getReplyInfo

```
@load "redis"
BEGIN{
 c=redis_connect()
 p=redis_pipeline(c)
 print redis_info(p,AR,"clients")
 print redis_getReplyInfo(p,AR)
 for(i in AR) {
   print i" ==> "AR[i]
 }
 redis_close(c)
}
```

getReply

Description: To receive the replies, the first time sends all buffered commands to the server, then subsequent calls get replies for each command.

Parameters
number: pipeline handle
array: for results. Will be used or no, according to command in question

Return value *string or number*: the return value of the following command in the buffer, `-1` on error (if not exist results buffered)

Example: Using getReply

```
c=redis_connect()
p=redis_pipeline(c)
redis_hset(p,"thehash","field1","25")
redis_hset(p,"thehash","field2","26")
 # To execute all and obtain the return of the first
r1=redis_getReply(p,REPLY)
 # To get the reply of second 'hset'
r2=redis_getReply(p,REPLY)
print r1,r2
 # Now there are no results in the buffer, and
 # using 'the pipeline handle' can be reused, no need
 # to close the pipeline once completed their use
```

getReplyMass

Description: This function was designed in order to perform mass insertion

Parameters *number*: pipeline handle

Return value *number*: the replies received from server or `-1` on error (if not exist results buffered)

Example: Using getReplyMass

```
BEGIN {
 FS = ","
 c=redis_connect()
 p=redis_pipeline(c)
}
{
  redis_set(p,$1,$2)
}
END {
  r=redis_getReplyMass(p) # "r" contains
    #how many data was transferred
}

# one-liner script
gawk -lredis -F, 'BEGIN{c=redis_connect(); \
  p=redis_pipeline(c)}{redis_set(p,$1,$2)} \
  END{redis_getReplyMass(p)}' file.csv
```

Scripting Funtions

Recommended reading Redis Lua scripting[27]

- evalRedis - Executes a Lua script server side
- evalsha - Executes a Lua script server side. The script had must been cached previously
- script exists - Checks existence of scripts in the scripts cache
- script flush - Removes all the scripts from the scripts cache
- script kill - Kills the script currently in execution
- script load - Loads the specified Lua script into the scripts cache

evalRedis

Description: Evaluates scripts using the Lua interpreter built into Redis.

Parameters
number: connection
string: the Lua script
number: the number of arguments, that represent Redis key names
array: containing the arguments
array: to store the results, but it not be always will contain results (read the example). Also this array may contain subarrays

Return value
number or *string*: `1` when it puts the results in the arrray. `-1` on error: `NOSCRIPT` No matching script.

[27]http://redis.io/commands/eval

Example: Using evalRedis

```
@load "redis"
BEGIN{
  c=redis_connect()
  ARG[1]="hset"
  ARG[2]="thehash"
  ARG[3]="field3"
  ARG[4]="value3"
  ret=redis_evalRedis(c, \
     "return redis.call( \
     KEYS[1],ARGV[1],ARGV[2],ARGV[3])",1,ARG,R)
  print "Function 'evalRedis' returns: "ret
  print "Elements in arrray of results: "length(R)
  print redis_hget(c,"thehash","field3")
  redis_close(c)
}
```

Output

```
Function 'evalRedis' returns: 1
Elements in arrray of results: 0
value3
```

script exists

Description: Returns information about the existence of the scripts in the script cache. Accepts one or more SHA1 digests. For detailed information about Redis Lua scripting[28]

Parameters
number: connection handle
string: “exists” *array*: containing the SHA1 digests

[28]http://redis.io/commands/eval

array: an array of integers that correspond to the specified SHA1 digest. It stores `1` for a script that actually exists in the script cache, otherwise `0` is stored.

Return value

number: `1` on success, `0` if array of SHA1 digests (third argument) is empty. `-1` on error.

Example: Using script exists

```
@load "redis"
BEGIN{
  c=redis_connect()
   #  'script load' returns SHA1 digest if success
  A[1]=redis_script(c, \
        "load","return {1,2,{7,'Hello World!',89}}")
  A[2]=redis_script(c, \
        "load","return redis.call('set','foo','bar')")
  A[3]=redis_script(c, \
        "load","return redis.call(KEYS[1],ARGV[1])")
  ret=redis_script(c,"exists",A,R)
  print "Obtain information of existence for these"
  print "three scripts whose keys are:"
  for(i in A) {
   print A[i]
  }
  print "script exists returns: "ret
  print "The results of command are:"
  for(i in R) {
    print i") "R[i]
  }
  redis_close(c)
}
```

Output

```
Obtain information of existence for these
three scripts whose keys are:
4647a689ee8af8debe9fd50a6fb9fee93ef92e43
2fa2b029f72572e803ff55a09b1282699aecae6a
24598a5b88e25cb396a4de4afbd1f5509c537396
script exists returns: 1
The results of command are:
1) 1
2) 1
3) 1
```

script load

Description: Loads a script into the scripts cache, without executing it. For detailed information about Redis Lua scripting[29]

Parameters
number: connection handle
string: “load”
string: the Lua script

Return value
string: returns the SHA1 digest of the script added into the script cache

[29]http://redis.io/commands/eval

Example: Using script load

```
c=redis_connect()
k1=redis_script(c, \
    "load","return redis.call('set','foo','bar')")
 # 'k1' stores the SHA1 digest
```

script kill

Description: Kills the currently executing Lua script For detailed information about Redis Lua scripting[30]

Parameters
number: connection handle
string: “kill”

Return value
number: `1` on sucess, `-1` on error, by example: NOTBUSY No scripts in execution right now.

Example: Using script kill

```
c=redis_connect()
redis_script(c,"kill")
```

script flush

Description: Flush the Lua scripts cache For detailed information about Redis Lua scripting[31]

Parameters
number: connection handle
string: “flush”

[30]http://redis.io/commands/eval
[31]http://redis.io/commands/eval

Return value
number: `1` on success

Example: Using script flush

```
c=redis_connect()
redis_script(c,"flush")
```

evalsha

Description: evalsha works exactly like evalRedis, but instead of having a script as the first argument it has the SHA1 digest of a script.

Parameters
number: connection
string: the SHA1 digest of a script
number: the number of arguments, that represent Redis key names
array: containing the arguments
array: to store the results, but it not be always will contain results (read the example). Also this array may contain subarrays

Return value
number or *string*: `1` when it puts the results in the arrray. `-1` on error: `NOSCRIPT` No matching script.

Example: Using evalsha

```
@load "redis"
BEGIN{
  c=redis_connect()
   #  loading into the scripts cache
  cmd1=redis_script(c, \
        "load","return {1,2,{7,'Hello World!',89}}")
  cmd2=redis_script(c, \
        "load","return redis.call('set','foo','bar')")
  cmd3=redis_script(c, \
        "load","return redis.call(KEYS[1],ARGV[1])")
   #  executing the scripts
  print "Returns cmd1:",
        redis_evalsha(c,cmd1,0,ARG,R)
  print "Elements in arrray R (the results): "length(R)
   # Elements in R are
   # R[1], R[2], R[3][1], R[3][2], R[3][3]
  delete R
  print "Returns cmd2:",
        redis_evalsha(c,cmd2,0,ARG,R)
  print "Elements in arrray R (the results): "length(R)
   # the arguments for the next
  ARG[1]="hvals"
  ARG[2]="thehash"
  print "Returns cmd3: "redis_evalsha(c,cmd3,1,ARG,R)
  print "Elements in arrray R (the results): "length(R)
   # Compare the return value of the next command
  ARG[1]="type"
  delete R
  print "Now cmd3 returns a string:",
         redis_evalsha(c,cmd3,1,ARG,R)
  print "Elements in arrray R (the results): "length(R)
  redis_close(c)
}
```

Output

```
Returns cmd1: 1
Elements in arrray R (the results): 3
Returns cmd2: 1
Elements in arrray R (the results): 0
Returns cmd3: 1
Elements in arrray R (the results): 30
Now cmd3 returns a string: hash
Elements in arrray R (the results): 0
```

Server Functions

- dbsize - Returns the number of keys in the currently-selected database
- flushdb - Deletes all the keys of the currently selected DB
- info - Returns information and statistics about the server

dbsize

Description: Returns the number of keys in the currently-selected database

Parameters
number: connection handle

Return value
number: `the number of keys` in the DB

Example: Using dbsize

```
@load "redis"
BEGIN{
  c=redis_connect()
  redis_select(c,5) # DB 5 selected
  print "DBSIZE:",
        redis_dbsize(c) # number of keys into DB 5
  print "FLUSHDB:",
        redis_flushdb(c) # delete all the keys of DB 5
  print redis_keys(c,"*",AR)
  print "DBSIZE: "redis_dbsize(c)
  redis_close(c)
}
```

Output

```
DBSIZE: 3
FLUSHDB: 1
0
DBSIZE: 0
```

flushdb

Description: Delete all the keys of the currently selected DB

Parameters
number: connection handle

Return value
`1` on success

Example: Using flushdb

```
c=redis_connect()
redis_flushdb(c)
 # deletes all the keys of the currently DB
```

info

Description: Returns information and statistics about the server. If is executed as pipelined command, the return is an string; this string is an collection of text lines. Lines can contain a section name (starting with a # character) or a property. All the properties are in the form of field:value terminated by \r\n

Parameters
number: connection handle
array: is an associative array and stores the results
string: is optional, and admits a name of section to filter out the scope of this section

Return value

`1` on success, `-1` on error

Example 1: Using info

```
@load "redis"
BEGIN{
 c=redis_connect()
 r=redis_info(c,AR)
 for(i in AR) {
   print i" ==> "AR[i]
 }
 redis_close(c)
}
```

Example 2: Using info with pipelining

```
@load "redis"
BEGIN {
 c=redis_connect()
 p=redis_pipeline(c)
 redis_info(p,AR,"replication") # asks a section
 # here others commands to pipeline
 #
 string_result=redis_getReply(p,ARRAY)
  # string_result contains the result as an
  # string multiline
 n=split(string_result,ARRAY,"\r\n")
 for(i in ARRAY) {
   n=split(ARRAY[i],INFO,":")
   if(n==2) {
     print INFO[1]" ==> "INFO[2]
   }
 }
 redis_close(c)
}
```

Transactions Functions

Recommended reading Redis Transactions topic[32]

- exec - Executes all previously queued commands in a transaction and restores the connection state to normal.
- multi - Marks the start of a transaction block
- watch - Marks the given keys to be watched for conditional execution of a transaction
- discard - Flushes all previously queued commands in a transaction
- unwatch - Flushes all the previously watched keys for a transaction

multi

Description: Marks the start of a transaction block

Parameters
number: connection

Return value
number: `1` always.

[32]http://redis.io/topics/transactions

Example: Using multi

```
@load "redis"
BEGIN{
  c=redis_connect()
  redis_multi(c)
  print redis_set(c,"SK1","valSK1")
  print redis_lrange(c,"list1",B,0,-1)
  print redis_llen(c,"list2")
  redis_exec(c,R)
   # do somthing with array R
  redis_close(c)
}
```

Output

```
QUEUED
QUEUED
QUEUED
```

exec

Description: Executes all previously queued commands in a transaction and restores the connection state to normal.

Parameters
number: connection
array: for the results. Each element being the reply to each of the commands in the atomic transaction

Return value
number: `1` on success, `0` if the execution was aborted (when using WATCH).

Example: Using exec

```
redis_exec(c,R)
```

watch

Description: Marks the given keys to be watched for conditional execution of a transaction.

Parameters
number: connection
string or array: a key name or an array containing the key names

Return value
number: always `1`

Example: Using watch

```
@load "redis"
BEGIN{
  c=redis_connect()
  AR[1]="list1"
  AR[2]="list2"
  redis_del(c,"list1")
  redis_del(c,"list2")
  LVAL[1]="one";
  LVAL[2]="two";
  LVAL[3]="three";
  redis_lpush(c,"list1",LVAL)
  redis_watch(c,AR)
  redis_multi(c)
  redis_set(c,"SK1","valSK1")
  redis_lrange(c,"list1",B,0,-1)
  redis_llen(c,"list2")
  redis_exec(c,R)
  print R[1]
  print R[2][1]"  "R[2][2]"  "R[2][3]
  print R[3]
  redis_close(c)
}
```

Output

```
1
three  two  one
0
```

unwatch

Description: Flushes all the previously watched keys for a transaction. No need to use when was used EXEC or DISCARD

Parameters
number: connection

Return value
number: always `1`

Example: Using unwatch

```
redis_unwatch(c)
```

discard

Description: Flushes all previously queued commands in a transaction and restores the connection state to normal. Unwatches all keys, if WATCH was used.

Parameters
number: connection

Return value
number: always `1`

Example: Using discard

```
redis_discard(c)
```

HyperLogLog Functions

Recommended reading Redis HyperLogLog[33]

- pfadd - Adds elements to the HyperLogLog data structure.
- pfcount - Returns the approximated cardinality computed by the HyperLogLog data structure stored at the specified key.
- pfmerge - Merge multiple HyperLogLog keys into an unique key.

pfadd

Description: Adds elements to the HyperLogLog data structure stored at the key specified.

Parameters
number: connection
string: key name
string or array: a element or an array containing the elements

Return value
number: `1` if at least 1 HyperLogLog internal register was altered. `0` otherwise.

[33]http://redis.io/commands/pfadd

Example: Using pfadd

```
@load "redis"
BEGIN {
  c=redis_connect()
  AR[1]="a"; AR[2]="b"; AR[3]="c"
  AR[4]="d"; AR[5]="e"; AR[6]="f"
  redis_pfadd(c,"hll",AR) # returns 1
  redis_pfcount(c,"hll")  # returns 6
  redis_close(c)
}
```

pfcount

Description: Returns the approximated cardinality computed by the HyperLogLog data structure stored at the specified key.

Parameters
number: connection
string or array: a key name or an array containing the key names

Return value
number: The approximated number of unique elements observed via PFADD. `0` if the key does not exist.

Example: Using pfcount

```
@load "redis"
BEGIN {
  c=redis_connect()
  AR[1]="foo"
  AR[2]="bar"
  AR[3]="zap"
  redis_pfadd(c,"hll",AR) # returns 1
  AR[1]=AR[2]="zap"
  redis_pfadd(c,"hll",AR) # returns 0
  BR[1]="foo"
  BR[2]="bar"
  redis_pfadd(c,"hll",BR) # returns 0
  print redis_pfcount(c,"hll")
  #
  CR[1]=1; CR[2]=2; CR[3]=3
  redis_pfadd(c,"other-hll",CR) # returns 1
  K[1]="hll"
  K[2]="other-hll"
  print redis_pfcount(c,K)
  redis_close(c)
}
```

Output

```
3
6
```

pfmerge

Description: Merge multiple HyperLogLog keys into an unique key that will approximate the cardinality of the union of the observed Sets of the source HyperLogLog structures.

Parameters
number: connection
string: a destination key name
string or array: a source key name or an array containing the source key names

Return value
number: returns `1`.

Example: Using pfmerge

```
@load "redis"
BEGIN {
  c=redis_connect()
  AR[1]="foo"; AR[2]="bar"; AR[3]="zap"; AR[4]="a"
  redis_pfadd(c,"hll1",AR) # returns 1
  BR[1]="a"; BR[2]="b"; BR[3]="c"; BR[4]="foo"
  redis_pfadd(c,"hll2",BR) # returns 1
  K[1]="hll1"; K[2]="hll2"
  redis_pfmerge(c,"hll3",K) # returns 1
  redis_pfcount(c,"hll3") # returns 6
  redis_close(c)
}
```

Geolocation Functions

Recommended reading Redis Geolocation[34].
Geospatial data (latitude, longitude, name) are stored into a key as a sorted set, in a way that makes it possible to later retrieve items using a query by radius or member.

1. geoadd - Adds the specified geospatial items to one specified key.
2. geodist - Obtains the distance between two members with information geospatial.
3. geohash - Returns members of a geospatial index as standard geohash strings.
4. geopos - Returns longitude and latitude of members of a geospatial index.
5. georadius - Obtains the members with geospatial information which are within the borders of the area specified with the center and the maximum distance from the center.
6. georadiusWD - This is like `georadius`, adding `distance` to the results.
7. georadiusWC - This is like `georadius`, adding coordinates (longitude and latitude) to the results.
8. georadiusWDWC - This is like `georadius`, adding distance and coordinates to the results.
9. georadiusbymember - This is like `georadius` with the same results. It takes the name of a member existing in a geospatial index
10. georadiusbymemberWD - This is like `georadiusbymember`, adding `distance` to the results.

[34]http://redis.io/commands/geoadd

11. georadiusbymemberWC - This is like `georadiusbymember`, adding coordinates (longitude and latitude) to the results.
12. georadiusbymemberWDWC - This is like `georadiusbymember`, adding distance and coordinates to the results.

geoadd

Description: Adds the specified geospatial items (latitude, longitude, name) to the specified key.

Parameters *number*: connection
string: key name
array: it contains three elements (longitude, latitude, name) per item

Return value *number*: the number of elements added to the sorted set, not including elements already existing for which the score was updated.

Example: Using geoadd

```
@load "redis"
BEGIN {
  c=redis_connect()
  A[1]="-118.2436800"
  A[2]="34.0522300"
  A[3]="la"
  A[4]="-74.0059700"
  A[5]="40.7142700"
  A[6]="nyc"
  redis_geoadd(c,"US",A)
   # returns 2, are two items added to a zset
  print "la-nyc kms:"
  redis_geodist(c,"US","la","nyc", "km")
  print "la-nyc miles:"
  redis_geodist(c,"US","la","nyc","mi")
  redis_close(c)
}
```

Output

```
la-nyc kms:
3936.8457102104558
la-nyc miles:
2446.248592721523
```

geodist

Description: Returns the distance between two members in the geospatial index represented by the sorted set.

Parameters *number*: connection
string: key name
string: name member
string: name member
optional string: the unit, must be one of the following values, m, km, mi, ft. Defaults to meters.

Return value *number*: represented as a string in the specified unit, or null string if one or both the members are missing

Example: Using geodist

```
@load "redis"
BEGIN {
  c=redis_connect()
  print redis_geodist(c,"US","la","nyc","m")
  redis_close(c)
}
```

Output

```
3936845.7102104556
```

geohash

Description: Returns members of a geospatial index as standard geohash strings.

Parameters *number*: connection
string: key name
array: it contains the names of members
array: will contain the results. Each element is the Geohash corresponding to each member name passed as argument

Return value `1` on success. `0` if not exists the key. `-1` on error.

Example: Using geohash

```
@load "redis"
BEGIN {
  A[1]="Trapani"
  A[2]="Catania"
  c=redis_connect()
  redis_geohash(c,"sicilia",A,RESP)
  for(i=1; i<=2; i++) {
    print i") "RESP[i]
  }
  redis_close(c)
}
```

Output

```
1) sqbbm2ck9f0
2) sqdtr74hyu0
```

geopos

Description: Returns longitude and latitude of members of a geospatial index.

Parameters *number*: connection
string: key name
array: it contains the names of members
array: will contain the results where each element is a two elements array representing longitude and latitude (x,y) of each member name passed as argument. Non existing elements are reported as NULL elements of the array.

Return value `1` on success. `0` if not exists the key. `-1` on error.

Example: Using geopos

```
@load "redis"
BEGIN {
  c=redis_connect()
  B[1]="Trapani"; B[2]="Catanzaro"; B[3]="Catania"
  redis_geopos(c,"sicilia",B,AR) # returns 1
  redis_close(c)
  if(length(AR)>0) {
     dumparray(AR,"NN")
  }
}

function dumparray(array,e, i) {
  for (i in array){
    if (isarray(array[i])){
      dumparray(array[i],e "[\"" i "\"]")
    }
    else {
        printf("%s[\"%s\"] = %s\n",e,i, array[i])
    }
  }
}
```

Output

```
@load "redis"
BEGIN {
  c=redis_connect()
  print redis_geodist(c,"US","la","nyc","m")
  redis_close(c)
}
```

Output

```
3936845.7102104556
```

georadius

Description: Returns the members of a sorted set populated with geospatial information using `geoadd`, which are within the borders of the area specified with the center location and the maximum distance from the center (the radius).

Parameters *number*: connection
string: key name
array: will contain the results, a set of strings.
number: longitud
number: latitud

number: radius
string: with a value between m|km|ft|mi *optional string*: the order with values desc or asc
optional number: to limit the results to the first N matching items. N is passed to `count` option of the command.

Return value `1` if is at least one result. `0` if there is no result. `-1` on error.

Example: Using georadius

```
@load "redis"
BEGIN {
  A[1]="13.361389"
  A[2]="38.115556"
  A[3]="Palermo"
  A[4]="15.087269"
  A[5]="37.502669"
  A[6]="Catania"
  A[7]="12.5372"
  A[8]="38.0176"
  A[9]="Trapani"
  c=redis_connect()
  redis_geoadd(c,"sicilia",A)
  print redis_geodist(c,"sicilia","Catania","Trapani",
  "km")
  redis_georadius(c,"sicilia",AR,15,37,200,"km")
    # georadius using optionals argumments
      # using count
    # redis_georadius(c,"sicilia",AR,15,37,200,"km",1)
      # using order and count
    # redis_georadius(c,"sicilia",AR,15,37,200,"km",
    # "desc", 1)
      # only order
    # redis_georadius(c,"sicilia",AR,15,37,200,"km",
    # "desc")
  dumparray(AR,"NN") # function defined in the
                     # geopos example
  redis_close(c)
}
```

Output

```
231.42622077769485
1) Palermo
2) Catania

NN["1"]["1"] = 12.537200152873993
NN["1"]["2"] = 38.017599561572482
NN["3"]["1"] = 15.087267458438873
NN["3"]["2"] = 37.502668423331613
```

georadiusWD

Description: This is like `georadius`, adding `distance` to the results.

Parameters *number*: connection
string: key name
array: will contain the results, a set of strings.
number: longitud
number: latitud
number: radius
string: with a value between m|km|ft|mi *optional string*: order
optional number: count

Return value `1` if is at least one result. `0` if there is no result. `-1` on error.

Example: Using georadiusWD

```
@load "redis"
BEGIN {
  c=redis_connect()
  redis_zrange(c,"sisu",RET,0,-1) # returns 1
  print "zset sisu contains index geospatial for"
  for(i in RET) {
    print i": "RET[i]
  }
  redis_georadius(c,"sisu",AR,"14",37.9,150,"km",1)
  for(i in AR) {
    print i") "AR[i]
  }
  print"georadiusWD radius 2500 output in km desc order"
  delete(AR)
  redis_georadiusWD(c,"sisu",AR,"14",37.9,2500,"km",
  "desc") # returns 1
  dumparray(AR,"NN") # function defined in the
                     # geopos example
  redis_close(c)
}
```

georadiusWC

Description: This is like `georadius`, adding `coordinates` to the results.

Parameters *number*: connection
string: key name
array: will contain the results, a set of strings.
number: longitud
number: latitud
number: radius

string: with a value between m|km|ft|mi *optional string*: order *optional number*: count

Return value `1` if is at least one result. `0` if there is no result. `-1` on error.

Example: Using georadiusWC

```
@load "redis"
BEGIN {
  c=redis_connect()
  print"georadiusWC radius 2500 output in km desc order"
  delete(AR)
  redis_georadiusWC(c,"sisu",AR,"14",37.9,2500,"km",
  "desc") # returns 1
  dumparray(AR,"NN")  # function defined in the
                      # geopos example
  redis_close(c)
}
```

georadiusWDWC

Description: This is like `georadius`, adding `distance` and `coordinates` to the results.

Parameters *number*: connection
string: key name
array: will contain the results, a set of strings.
number: longitud
number: latitud
number: radius
string: with a value between m|km|ft|mi
optional string: order *optional number*: count

Return value `1` if is at least one result. `0` if there is no result. `-1` on error.

Example: Using georadiusWDWC

```
@load "redis"
BEGIN {
  c=redis_connect()
  print "georadiusWDWC radius 2500 output in km desc"
  delete(AR)
  redis_georadiusWDWC(c,"sisu",AR,"14",37.9,2500,"km",
  "desc") # returns 1
  dumparray(AR,"NN")  # function defined in the
                      #  geopos example
  redis_close(c)
}
```

georadiusbymember

Description: This command is exactly like `georadius`. The difference is that instead of to take a longitude and latitude as the center of the area, it takes the name of a member already existing inside the geospatial index.

Parameters *number*: connection
string: key name
array: will contain the results, a set of strings.
string: name member
number: radius
string: with a value between m|km|ft|mi

Return value `1` if is at least one result. `0` if there is no result. `-1` on error.

Example: Using georadiusbymember

```
@load "redis"
BEGIN {
  c=redis_connect()
  redis_zrangeWithScores(c,"sisu",RES,0,-1) # returns 1
  for(i in RES){
    print i") "RES[i]
  }
  print ""
  redis_georadiusbymember(c,"sisu",AR,"Ecija",350,
  "km") # returns 1
  dumparray(AR,"NN")
  delete AR
  print ""
  redis_georadiusbymember(c,"sisu",AR,"Ecija",2800,
  "km") # returns 1
  redis_close(c)
  dumparray(AR,"NN")
}
```

Output

```
1) Sevilla
2) 1966655518805908
3) Ecija
4) 1968142286694693
5) Palermo
6) 3479099956230698
7) Catania
8) 3479447370796909

NN["1"] = Sevilla
NN["2"] = Ecija
```

```
NN["1"] = Sevilla
NN["2"] = Ecija
NN["3"] = Palermo
NN["4"] = Catania
```

georadiusbymemberWD

Description: Returns the members of a sorted set populated with geospatial information using `geoadd`, adding `distance` to the results.

Parameters *number*: connection
string: key name
array: will contain the results, a set of strings.
string: name member
number: radius
string: with a value between m|km|ft|mi

Return value `1` if is at least one result. `0` if there is no result. `-1` on error.

Example: Using georadiusbymemberWD

```
@load "redis"
BEGIN {
 c=redis_connect()
 redis_georadiusbymemberWD(c,"sisu",AR,"Ecija",2800,
 "km")
 redis_close(c)
 dumparray(AR,"NN")
}
```

Output

```
NN["1"]["1"] = Sevilla
NN["1"]["2"] = 81.3977
NN["2"]["1"] = Ecija
NN["2"]["2"] = 0.0000
NN["3"]["1"] = Palermo
NN["3"]["2"] = 1618.9443
NN["4"]["1"] = Catania
NN["4"]["2"] = 1775.8787
```

georadiusbymemberWC

Description: Returns the members of a sorted set populated with geospatial information using `geoadd`, adding `coordinates` to the results.

Parameters *number*: connection
string: key name
array: will contain the results, a set of strings.
string: name member
number: radius
string: with a value between m|km|ft|mi

Return value `1` if is at least one result. `0` if there is no result. `-1` on error.

Example: Using georadiusbymemberWC

```
@load "redis"
BEGIN {
 c=redis_connect()
 redis_georadiusbymemberWC(c,"sisu",AR,"Ecija",2800,
 "km")
 redis_close(c)
 dumparray(AR,"NN")
}
```

Output

```
NN["1"]["1"] = Sevilla
NN["1"]["2"]["1"] = -5.9844991564750671
NN["1"]["2"]["2"] = 37.389100241787432
NN["2"]["1"] = Ecija
NN["2"]["2"]["1"] = -5.0826975703239441
NN["2"]["2"]["2"] = 37.541500351492687
NN["3"]["1"] = Palermo
NN["3"]["2"]["1"] = 13.361389338970184
NN["3"]["2"]["2"] = 38.115556395496299
NN["4"]["1"] = Catania
NN["4"]["2"]["1"] = 15.087267458438873
NN["4"]["2"]["2"] = 37.502668423331613
```

georadiusbymemberWDWC

Description: Returns the members of a sorted set populated with geospatial information using `geoadd`, adding `distances` and `coordinates` to the results.

Parameters *number*: connection
string: key name

array: will contain the results, a set of strings.
string: name member
number: radius
string: with a value between m|km|ft|mi

Return value `1` if is at least one result. `0` if there is no result. `-1` on error.

Example: Using georadiusbymemberWDWC

```
@load "redis"
BEGIN {
 c=redis_connect()
 redis_georadiusbymemberWDWC(c,"sisu",AR,"Ecija",2800,
 "km")
 redis_close(c)
 dumparray(AR,"NN")
}
```

Output

```
NN["1"]["1"] = Sevilla
NN["1"]["2"] = 81.3977
NN["1"]["3"]["1"] = -5.9844991564750671
NN["1"]["3"]["2"] = 37.389100241787432
NN["2"]["1"] = Ecija
NN["2"]["2"] = 0.0000
NN["2"]["3"]["1"] = -5.0826975703239441
NN["2"]["3"]["2"] = 37.541500351492687
NN["3"]["1"] = Palermo
NN["3"]["2"] = 1618.9443
NN["3"]["3"]["1"] = 13.361389338970184
NN["3"]["3"]["2"] = 38.115556395496299
NN["4"]["1"] = Catania
NN["4"]["2"] = 1775.8787
```

```
NN["4"]["3"]["1"] = 15.087267458438873
NN["4"]["3"]["2"] = 37.502668423331613
```