

# Package ‘rredis’

July 5, 2015

**Type** Package

**Title** ``Redis" Key/Value Database Client

**Version** 1.7.0

**Date** 2015-07-04

**Author** B. W. Lewis

**Maintainer** B. W. Lewis <blewis@illposed.net>

**Description** R client interface to the ``Redis" key-value database.

**License** Apache License (>= 2.0)

**LazyLoad** yes

**Suggests** RUnit

**NeedsCompilation** yes

**Repository** CRAN

**Date/Publication** 2015-07-05 23:37:49

## R topics documented:

rredis-package . . . . .	4
Increment, Decrement functions . . . . .	4
redisAuth . . . . .	6
redisBgRewriteAOF . . . . .	6
redisBgSave . . . . .	7
redisBitCount . . . . .	8
redisBitOp . . . . .	8
redisBLPop . . . . .	9
redisBRPop . . . . .	10
redisBRPopLPush . . . . .	11
redisClose . . . . .	12
redisCmd . . . . .	13
redisConnect . . . . .	14
redisDBSize . . . . .	16
redisDelete . . . . .	17
redisDiscard . . . . .	18

redisEval	18
redisExec	19
redisExists	20
redisExpire	21
redisExpireAt	22
redisFlushAll	23
redisFlushDB	23
redisGet	24
redisGetBit	25
redisGetResponse	26
redisGetSet	27
redisHDel	28
redisHExists	29
redisHFields	30
redisHGet	31
redisHGetAll	32
redisHIncrBy	33
redisHKeys	34
redisHLen	35
redisHMGet	36
redisHMSet	37
redisHSet	38
redisHVals	39
redisInfo	40
redisKeys	40
redisLIndex	41
redisLLen	42
redisLPop	43
redisLPush	44
redisLRange	45
redisLRem	46
redisLSet	47
redisLTrim	48
redisMGet	49
redisMonitorChannels	50
redisMove	52
redisMSet	53
redisMulti	54
redisPersist	54
redisPublish	55
redisRandomKey	56
redisRename	57
redisRPop	58
redisRPopLPush	59
redisSAdd	60
redisSave	62
redisSCard	62
redisSDiff	63

redisSDiffStore . . . . .	64
redisSelect . . . . .	65
redisSet . . . . .	66
redisSetBit . . . . .	67
redisSetContext . . . . .	68
redisSetPipeline . . . . .	69
redisShutdown . . . . .	70
redisSInter . . . . .	71
redisSInterStore . . . . .	72
redisSIsMember . . . . .	73
redisSlaveOf . . . . .	74
redisSMembers . . . . .	75
redisSMove . . . . .	76
redisSort . . . . .	77
redisSPop . . . . .	78
redisSRandMember . . . . .	79
redisSRem . . . . .	80
redisSubscribe . . . . .	81
redisSUnion . . . . .	82
redisSUnionStore . . . . .	83
redisTTL . . . . .	84
redisType . . . . .	85
redisUnsubscribe . . . . .	86
redisUnwatch . . . . .	88
redisWatch . . . . .	89
redisZAdd . . . . .	89
redisZCard . . . . .	90
redisZCount . . . . .	91
redisZIncrBy . . . . .	91
redisZInterStore . . . . .	92
redisZRange . . . . .	93
redisZRangeByScore . . . . .	94
redisZRank . . . . .	95
redisZRem . . . . .	95
redisZRemRangeByRank . . . . .	96
redisZRemRangeByScore . . . . .	97
redisZScore . . . . .	98
redisZUnionStore . . . . .	98

---

rredis-package      *Redis interface package.*

---

### Description

The rredis package provides a native R interface to Redis. Redis is an in memory key/value database with many innovative features, see <http://redis.io> for details. It supports data persistence, networked client/server operation, command pipelining, structured value types, data expiration, multicast-like publish/subscribe, and it's very fast.

### Details

Package options:

- `options('redis:num'=TRUE)` Set this option to TRUE to return Redis : messages as numeric values. This was the default behavior of the rredis package for all versions prior to 1.6.9. For versions of the R package later than that, `redis : messages` are returned as raw Redis string values to correspond to the data types stored in Redis. Set this option to revert to the old behavior.

Redis commands affected by this option importantly include the increment and decrement operations.

### Author(s)

B. W. Lewis <[blewis@illposed.net](mailto:blewis@illposed.net)>

Maintainer: B. W. Lewis <[blewis@illposed.net](mailto:blewis@illposed.net)>

---

Increment, Decrement functions  
*Increment or decrement Redis values.*

---

### Description

`redisIncr` increments the Redis string value corresponding to the specified key by one.

`redisDecr` decrements the Redis string value corresponding to the specified key by one.

The various `*By` functions increment or decrement values by a specified integer or numeric value.

### Usage

```
redisIncr(key)
redisIncrBy(key, value)
redisIncrByFloat(key, value)
redisDecr(key)
redisDecrBy(key, value)
```

**Arguments**

key	A key corresponding to the value to increment.
value	The value to increment by (integer, numeric, or character).

**Details**

Note that the initial value must be a raw character value (a plain Redis string value), not a serialized R value. See the examples below.

The increment value may be an integer (`redisIncrBy`) or a numeric value (`redisIncrByFloat`), or a raw character representation of an integer or numeric value. If the key does not exist or contains a value of a wrong type, set the key to the value of "0" or similar string representation of an integer before running the function.

**Value**

The new value of key after the increment. Note that the value is returned as a raw Redis string value.

**Author(s)**

B. W. Lewis

**References**

<http://redis.io/commands>

**See Also**

[redisDecr](#)

**Examples**

```
## Not run:
redisSet('x',charToRaw('5')) # Note the value must be a raw string!
redisIncr('x')
redisIncrBy('x','3')
redisIncrBy('x',3) # also works
redisIncrByFloat('x',pi)
redisDecr('x')
redisDecrBy('x',3)

## End(Not run)
```

redisAuth

*Redis authentication.*

---

**Description**

Redis supports a trivially simple and insecure authentication method. This function implements it.

**Usage**

```
redisAuth(pwd)
```

**Arguments**

pwd                      The (required) password.

**Details**

If you use this function, it's probably a good idea to encrypt network traffic between Redis and its client with a program like stunnel. Otherwise, passwords are transmitted over the network in clear text.

**Value**

TRUE if successful, FALSE otherwise.

**Author(s)**

B. W. Lewis

**References**

<http://redis.io/commands>

---

redisBgRewriteAOF*redisBgRewriteAOF*

---

**Description**

Re-write the Redis append-only file in the background.

**Usage**

```
redisBgRewriteAOF()
```

**Details**

BGREWRITEAOF rewrites the Append Only File in the background when it gets too big. The Redis Append Only File is a Journal, so every operation modifying the dataset is logged in the Append Only File (and replayed at startup). This means that the Append Only File always grows. In order to rebuild its content the BGREWRITEAOF creates a new version of the append only file starting directly from the dataset in memory in order to guarantee the generation of the minimal number of commands needed to rebuild the database. (These details are copied verbatim from the Redis master documentation, see the references below.)

**Value**

Nothing is returned.

**Author(s)**

B. W. Lewis

**References**

<http://redis.io/commands>

---

redisBgSave

*redisBgSave*

---

**Description**

Asynchronously save that database to disk.

**Usage**

```
redisBgSave()
```

**Details**

Force Redis to save the database(s) to disk in the background.

**Value**

Nothing is returned. Check the UNIX time of the last completed save operation with the `redisLastsave` function.

**Author(s)**

B. W. Lewis

**References**

<http://redis.io/commands>

---

redisBitCount	<i>Redis BITCOUNT - count all bits in key</i>
---------------	---

---

**Description**

Count the number of set bits (population counting) in a string.

**Usage**

```
redisBitCount(key)
```

**Arguments**

key	redis key
-----	-----------

**Value**

the counted bits as an integer value

---

redisBitOp	<i>Redis BITOP - execute bitoperations on multiple bitsets</i>
------------	--

---

**Description**

Perform a bitwise operation between multiple keys (containing string values) and store the result in the destination key

**Usage**

```
redisBitOp(operation, destkey, sourcekeys, ...)
```

**Arguments**

operation	bit operation as character: 'AND', 'OR', 'XOR', 'NOT'
destkey	destination key where the resulting bit operation will be stored
sourcekeys	one or more source keys subject to the bit operations
...	Optional additional arguments. Specify raw=TRUE to skip de-serialization of the data.

**Value**

the counted bits as an integer value



---

redisBLPop	<i>Blocking List Pop</i>
------------	--------------------------

---

**Description**

Pop the first available value from a key or list of keys, blocking until a value is available.

**Usage**

```
redisBLPop(keys, timeout = 0, ...)
```

**Arguments**

keys	A character key value or vector of key values to monitor.
timeout	A timeout in seconds after which, if no value is available, the function returns NULL. A value of zero indicates block indefinitely.
...	Optional additional arguments. Specify raw=TRUE to skip de-serialization of the data.

**Details**

redisBLPop blocks until at least one of the keys exists and contains a non-empty value, or until the specified timeout period is reached, whichever comes first. Keys are scanned in the order that they are specified.

**Value**

redisBLPop returns NULL after the timeout period, or a list containing:

key	The first key encountered with an available value,
value	The corresponding value.

**Author(s)**

B. W. Lewis

**References**

<http://redis.io/commands>

**See Also**

redisBRpop

**Examples**

```
## Not run:
redisConnect()
redisBLpop('x', 5)
redisLPush('x',runif(5))
redisBLPop('x')

## End(Not run)
```

---

 redisBRPop

*Blocking List Pop*


---

**Description**

Pop the first available value from a key or list of keys, blocking until a value is available.

**Usage**

```
redisBRPop(keys, timeout = 0, ...)
```

**Arguments**

keys	A character key value or vector of key values to monitor.
timeout	A timeout in seconds after which, if no value is available, the function returns NULL. A value of zero indicates block indefinitely.
...	Optional additional arguments. Specify raw=TRUE to skip de-serialization of the data.

**Details**

redisBRPop blocks until at least one of the keys exists and contains a non-empty value, or until the specified timeout period is reached, whichever comes first. Keys are scanned in the order that they are specified.

**Value**

redisBRPop returns NULL after the timeout period, or a list containing:

key	The first key encountered with an available value,
value	The corresponding value.

**Author(s)**

B. W. Lewis

**References**

<http://redis.io/commands>

**See Also**

redisBLpop

**Examples**

```
## Not run:
redisConnect()
redisBLpop('x', 5)
redisLPush('x',runif(5))
redisBRPop('x')

## End(Not run)
```

---

redisBRPopLPush	<i>Remove the tail from a list, blocking if it does not exist, pushing to another.</i>
-----------------	--

---

**Description**

Atomically return and remove the last (tail) element of the src list, blocking if the element does not exist, and push the element as the first (head) element of the dst list.

**Usage**

```
redisBRPopLPush(src, dest, timeout = 0, ...)
```

**Arguments**

src	A key corresponding to the source list.
dest	A key corresponding to the destination list.
timeout	Block for at most timeout seconds. Set to zero to block indefinitely.
...	Optional additional arguments. Specify raw=TRUE to skip de-serialization of the data.

**Details**

Atomically return and remove the last (tail) element of the src list, blocking until the element exists, and push the element as the first (head) element of the dst list. For example if the source list contains the elements "a","b","c" and the destination list contains the elements "foo","bar" after a redisRPopLPush command the content of the two lists will be "a","b" and "c","foo","bar".

If the key does not exist or the list is already empty the special value NULL is returned. If the srckey and dstkey are the same the operation is equivalent to removing the last element from the list and pushing it as first element of the list, so it's a "list rotation" command.

See the Redis reference below for programming examples and discussion.

**Value**

The value moved or rotated across lists, or NULL if the source key does not exist or corresponds to an empty list. An error is thrown if either of the keys does not correspond to a value of 'list' type.

**Author(s)**

B. W. Lewis

**References**

<http://redis.io/commands>

**See Also**

[redisRPopLPush](#)

**Examples**

```
## Not run:  
redisConnect()  
redisLPush('x',1)  
redisBRPopLPush('x','x')  
  
## End(Not run)
```

---

redisClose

*Close an open connection to a Redis server.*

---

**Description**

The redisClose function closes any open connection to a Redis server.

**Usage**

```
redisClose(e)
```

**Arguments**

e (Optional) Redis context. The current context is used if e is not supplied.

**Details**

A running instance of a Redis server is required.

**Value**

Nothing is returned. Errors are displayed if the function fails to close the connection to the Redis server, or if the connection is invalid.

**Author(s)**

B. W. Lewis

**References**

<http://redis.io/commands>

**See Also**

[redisConnect](#) [redisGetContext](#)

**Examples**

```
## Not run:
redisConnect()
redisSet('x',runif(5))
redisGet('x')
redisClose()

## End(Not run)
```

---

redisCmd

*General Redis Interface Function*

---

**Description**

Perform any Redis command.

**Usage**

```
redisCmd(CMD, ..., raw = FALSE)
```

**Arguments**

CMD	The (required) Redis command (character string).
...	Redis command arguments.
raw	If TRUE, return results in raw byte form.

**Details**

Use this low-level function to perform any Redis operation. The ... argument(s) not already in raw format will be converted to raw byte format, and non-character values will be serialized.

**Value**

Output from the Redis command—varies depending on command.

**Author(s)**

B. W. Lewis

**References**<http://redis.io/commands>**Examples**

```
## Not run:
redisCmd('set','x',runif(5))
redisCmd('get','x')

## End(Not run)
```

---

redisConnect	<i>Connect to a Redis server.</i>
--------------	-----------------------------------

---

**Description**

Connect to an available Redis server on the specified port.

**Usage**

```
redisConnect(host = "localhost", port = 6379, password = NULL,
             returnRef = FALSE, nodelay=TRUE, timeout=2678399L)
```

**Arguments**

host	The Redis server host name or inet address (optional, character). The default value is "localhost".
port	The Redis port number (optional, numeric or integer). The default value is 6379L.
password	Redis authentication password.
returnRef	Set returnRef=TRUE to return the environment that contains the Redis connection state (see details). The default value is FALSE.
nodelay	Set nodelay=TRUE to use TCP_NODELAY (that is, to disable the TCP Nagle algorithm). The default value is TRUE, see the details below.
timeout	Set the R connection timeout (in seconds). Beware that some OSes may treat very large values as zero: however the POSIX standard requires values up to 31 days to be supported.

## Details

A running instance of a Redis server is required. Use 'returnRef' to return the Redis connection state in an environment. Then use the 'redisSetContext' function to switch environment state and manage multiple open Redis connections.

Set `nodeLay=TRUE` to use the `TCP_NODELAY` socket setting (disabling the TCP Nagle flow control algorithm) which can improve performance especially for rapid, non-pipelined small-sized transactions. We follow the convention of other popular Redis clients like the `hiredis` C library interface and use `TCP_NODELAY` as the default choice.

Note that Redis pipelining can also increase performance: `redisSetPipeline(TRUE)` (q.v.).

## Value

Nothing is returned by default. Errors are displayed if the function fails to connect to the specified Redis server. Disconnect from a connected server with `redisClose`.

If `returnRef` is set to `TRUE` and no error occurs, a list describing the Redis connection will be returned. A future version of the package will use this feature to support multiple Redis connections with the `attachRedis` function.

## Author(s)

B. W. Lewis

## References

<http://redis.io/commands>

## See Also

[redisClose](#) [redisGetContext](#) [redisSetContext](#) [redisSetPipeline](#) [redisGetResponse](#)

## Examples

```
## Not run:
redisConnect()
redisSet('x',runif(5))
redisGet('x')
redisClose()

## End(Not run)
```

---

redisDBSize	<i>Return the number of keys in the current Redis database.</i>
-------------	---

---

**Description**

Return the number of keys in the current Redis database.

**Usage**

```
redisDBSize()
```

**Details**

Use `redisSelect` to choose a current database from among the available Redis databases.

**Value**

The number of keys in the current database.

**Author(s)**

B. W. Lewis

**References**

<http://redis.io/commands>

**See Also**

[redisSelect](#)

**Examples**

```
## Not run:  
redisDBSize()  
  
## End(Not run)
```



---

`redisDelete`*Delete a key and associated value from Redis.*

---

**Description**

Delete a key and associated value from Redis.

**Usage**

```
redisDelete(key)
```

**Arguments**

`key`            The (required) character identifier to be looked up.

**Details**

The key must not contain spaces or newline characters (otherwise an error will be thrown).

**Value**

Nothing is returned if the key/value pair is successfully deleted. A warning is thrown if the they key could not be found.

**Author(s)**

B. W. Lewis

**References**

<http://redis.io/commands>

**See Also**

[redisSet](#)

**Examples**

```
## Not run:  
redisSet('x',runif(5))  
redisDelete('x')  
  
## End(Not run)
```

---

redisDiscard	<i>redisDiscard</i>
--------------	---------------------

---

**Description**

Discard the current transaction block.

**Usage**

```
redisDiscard()
```

**Details**

Discard the current transaction block.

**Value**

"OK"

**Author(s)**

B. W. Lewis

**References**

<http://redis.io/commands>

---

redisEval	<i>Evaluate a Lua script in the Redis server.</i>
-----------	---

---

**Description**

Evaluate a Lua script in the Redis server.

**Usage**

```
redisEval(script, keys = vector("list",0), SHA = FALSE, ...)
```

**Arguments**

script	A Redis server-side Lua script (character).
keys	An optional list of script key arguments.
SHA	If TRUE, the script is a SHA1-encoded string, otherwise plain text.
...	Optional list of script arguments.

**Value**

"OK" is returned on success. Errors are displayed if the script fails.

**Author(s)**

B. W. Lewis

**References**

<http://redis.io/commands>

**Examples**

```
## Not run:
redisConnect()
redisEval("return redis.call('set','foo','bar')")

# Supply a key argument to the script
redisEval("return redis.call('set',KEYS[1],'bar')", "foo")

# Supply a key and other arguments to the script
redisEval("return redis.call('set',KEYS[1],ARGV[1])", "foo", pi)

## End(Not run)
```

---

redisExec

*redisExec*

---

**Description**

End a transaction block.

**Usage**

```
redisExec()
```

**Details**

End a transaction block. All Redis statements issued after `redisMulti` will be queued locally and then sent to the Redis server en masse once the `redisExec` function is issued.

**Value**

The results of all queued functions are returned in a list.

**Author(s)**

B. W. Lewis

**References**

<http://redis.io/commands>

---

redisExists	<i>Test the existence of a key in the Redis database.</i>
-------------	---

---

**Description**

Test the existence of a key in the Redis database.

**Usage**

```
redisExists(key)
```

**Arguments**

key                   The (required) character identifier to be looked up.

**Details**

The key must not contain spaces or newline characters (otherwise an error will be thrown).

**Value**

Returns FALSE if no matching key, TRUE if matching key exists, and NULL if an error occurred.

**Author(s)**

B. W. Lewis

**References**

<http://redis.io/commands>

**See Also**

[redisSet](#)

**Examples**

```
## Not run:  
redisSet('x',runif(5))  
redisExists('x')  
  
## End(Not run)
```

---

redisExpire	<i>Set a timeout on the specified key.</i>
-------------	--

---

### Description

Set a timeout on the specified key, after which the key and corresponding value will be deleted.

### Usage

```
redisExpire(key, seconds)
redisPexpire(key, milliseconds)
```

### Arguments

key	The character key on which to set the timeout.
seconds	The integer timeout in seconds.
milliseconds	The integer timeout in milliseconds.

### Details

Operations that modify value(s) corresponding to a key subsequent to the `redisExpire` function clear the timeout, removing the expiration. The `redisExpire` function can't set a new timeout value once a timeout has been set on a key.

### Value

Boolean TRUE if the timeout command was successful, FALSE otherwise.

### Author(s)

B. W. Lewis

### References

<http://redis.io/commands>

### See Also

[redisExpireAt](#)

### Examples

```
## Not run:
redisConnect()
redisLPush('x',runif(5))
redisExpire('x', 3)

## End(Not run)
```

---

redisExpireAt	<i>Set a timeout on the specified key.</i>
---------------	--

---

### Description

Set a timeout on the specified key, after which time the key and corresponding value will be deleted.

### Usage

```
redisExpireAt(key, time)
redisPexpireAt(key, time)
```

### Arguments

key	The character key on which to set the timeout.
time	The UNIX time of expiration in seconds or milliseconds.

### Details

Operations that modify value(s) corresponding to a key subsequent to the `redisExpireAt` function clear the timeout, removing the expiration. The `redisExpireAt` function can't set a new timeout value once a timeout has been set on a key.

### Value

Boolean TRUE if the timeout command was successful, FALSE otherwise.

### Author(s)

B. W. Lewis

### References

<http://redis.io/commands>

### See Also

[redisExpire](#)

### Examples

```
## Not run:
redisConnect()
redisLPush('x',runif(5))
redisExpireAt('x', 1266209144)

## End(Not run)
```

---

redisFlushAll	<i>Delete all keys and values from all databases.</i>
---------------	---

---

**Description**

Delete all keys and values from all databases, not just the currently selected one.

**Usage**

```
redisFlushAll()
```

**Value**

Returns TRUE if successful, FALSE otherwise.

**Author(s)**

B. W. Lewis

**References**

<http://redis.io/commands>

**See Also**

[redisFlushDB](#)

**Examples**

```
## Not run:  
redisConnect()  
redisFlushAll()  
  
## End(Not run)
```

---

redisFlushDB	<i>Delete all keys and values from the current database.</i>
--------------	--

---

**Description**

Delete all keys and values from the currently selected database.

**Usage**

```
redisFlushDB()
```

**Value**

Returns TRUE if successful, FALSE otherwise.

**Author(s)**

B. W. Lewis

**References**

<http://redis.io/commands>

**See Also**

[redisFlushAll](#)

**Examples**

```
## Not run:  
redisConnect()  
redisFlushDB()  
  
## End(Not run)
```

---

redisGet

*Retrieve a value from Redis.*

---

**Description**

Retrieve a value identified by a key from the Redis database.

**Usage**

```
redisGet(key, ...)
```

**Arguments**

key	The (required) character identifier for the value to be looked up.
...	Optional additional arguments passed to <code>redisCmd</code> . Specify <code>raw=TRUE</code> to skip de-serialization of the data.

**Details**

The key must not contain spaces or newline characters (otherwise an error will be thrown). The `raw` option is used to retrieve binary data from other languages.



**Value**

The value corresponding to the specified key, or NULL if the matching key contained no value or if no matching key was found.

Use the `raw=TRUE` option to return Redis values in raw binary form, which can optionally later be converted to character using the `rawToChar` function.

A Redis value that does not represent a serialized R value is returned as a character R value with an attribute named "redis string value." Subsequent uploads of that value to Redis will send the character-valued data in raw form (not as a serialized R value), preserving the original nature of the data in Redis.

**Author(s)**

B. W. Lewis

**References**

<http://redis.io/commands>

**See Also**

[redisSet](#)

**Examples**

```
## Not run:
redisSet('x',runif(5))
redisGet('x')

## End(Not run)
```

---

redisGetBit

*Redis BITSET gets - get binary value*

---

**Description**

Returns the bit value at offset in the string value stored at key.

**Usage**

```
redisGetBit(key, offset, ...)
```

**Arguments**

key	redis key
offset	integer index
...	Optional additional arguments. Specify <code>raw=TRUE</code> to skip de-serialization of the data.

**Value**

bit binary integer

---

redisGetResponse      *redisGetResponse*

---

**Description**

Service messages from all subscribed Redis message channels.

**Usage**

```
redisGetResponse(all=TRUE)
```

**Arguments**

**all**                      The all argument is ignored. It's left there for backwards compatibility with code based on older package versions.

**Details**

(From the Redis.io documentation): `redisSubscribe`, `redisUnsubscribe` and `redisPublish` implement the Publish/Subscribe messaging paradigm where (citing Wikipedia) senders (publishers) are not programmed to send their messages to specific receivers (subscribers). Rather, published messages are characterized into channels, without knowledge of what (if any) subscribers there may be. Subscribers express interest in one or more channels, and only receive messages that are of interest, without knowledge of what (if any) publishers there are.

The `redisGetResponse` function may be called to service messages on all subscribed channels. When a message is received the a list of three elements is returned containing: the character string 'message', the name of receiving channel, and the message content.

WARNING: The `redisGetResponse` function blocks indefinitely until a message is received.

**Value**

A list containing the string 'message', the channel name from which the message was received, and the message data.

**Author(s)**

B. W. Lewis

**References**

<http://redis.io/commands>

**See Also**

[redisSubscribe](#) [redisPublish](#) [redisUnsubscribe](#) [redisMonitorChannels](#)

**Examples**

```
## Not run:
redisConnect()
# Define a callback function to process messages from channel 1:
channel1 <- function(x) {
  cat("Message received from channel 1: ",x,"\n")
}
# Define a callback function to process messages from channel 2:
channel2 <- function(x) {
  cat("Message received from channel 2: ",x,"\n")
}
redisSubscribe(c('channel1','channel2'))
# Monitor channels for at least one minute (and indefinitely until
# a message is received):
t1 <- proc.time()[[3]]
while(proc.time()[[3]] - t1 < 60)
{
  print(redisGetResponse())
}
redisUnsubscribe(c('channel1','channel2'))

## End(Not run)
```

---

redisGetSet

*Store a value in Redis, returning the previously defined value.*


---

**Description**

Store a value identified by a character key name in the Redis database, returning the previously defined value or NULL if the key was not already associated with a value.

**Usage**

```
redisGetSet(key, value, ...)
```

**Arguments**

key	The (required) character identifier for the value to be stored.
value	The (required) object to associate with the key.
...	Optional additional arguments. Specify raw=TRUE to skip de-serialization of the data.

**Details**

The key must not contain spaces or newline characters (otherwise an error will be thrown).

The value object is copied to the Redis server. The value to be stored may be any serializable R object up to the Redis maximum object size (excluding, for example, external pointer references). References to other R objects or environments inside the value are not guaranteed to be preserved.

In order to store strings that can easily be read by other clients, first convert the character object using the [charToRaw](#) function as shown in the examples.

**Value**

The previous value associated with key or NULL if no previous association exists.

**Author(s)**

B. W. Lewis

**References**

<http://redis.io/commands>

**See Also**

[redisGet](#)

**Examples**

```
## Not run:  
# Store an R object with the key 'x':  
redisGetSet('x',runif(5))  
  
## End(Not run)
```

---

redisHDel

*Delete a hash value.*

---

**Description**

Delete the value associated with the given key/field combination.

**Usage**

```
redisHDel(key, field)
```

**Arguments**

key	A key name.
field	A field name.

**Value**

Nothing is returned if the key/value pair is successfully deleted. A warning is thrown if the they key could not be found.

**Author(s)**

B. W. Lewis

**References**

<http://redis.io/commands>

**See Also**

[redisHSet](#)

**Examples**

```
## Not run:
redisHSet('A',list(x=1,y=2,z=3))
redisHDel('A','x')

## End(Not run)
```

---

redisHExists

*Test the existence of a hash.*

---

**Description**

Test the existence of a hash combination in the Redis database.

**Usage**

```
redisHExists(key, field)
```

**Arguments**

key	A key name.
field	A field name.

**Value**

Returns FALSE if no matching key/field combination, TRUE if matching entry exists, and NULL if an error occurred.

**Author(s)**

B. W. Lewis

**References**

<http://redis.io/commands>

**See Also**[redisHSet](#)**Examples**

```
## Not run:
redisHSet('A','x',runif(5))
redisHExists('A','x')

## End(Not run)
```

---

`redisHFields`*Redis hash fields.*

---

**Description**

Return the fields associated with the given key.

**Usage**

```
redisHFields(key, ...)
```

**Arguments**

<code>key</code>	The key to look up.
<code>...</code>	Optional additional arguments. Specify <code>raw=TRUE</code> to skip de-serialization of the data.

**Details**

Returns the fields in the Redis hash associated with `key`. If the key is not found, or if the hash is empty, `NULL` is returned. If the key is associated with a value of type other than 'hash,' an error is thrown.

**Value**

A list of fields defined for the given key.

**Author(s)**

B. W. Lewis

**References**

<http://redis.io/commands>

**See Also**[redisHSet](#)

**Examples**

```
## Not run:  
redisHSet('A',list(x=1,y=2,z=3))  
redisHFields('A')  
  
## End(Not run)
```

---

redisHGet	<i>Retrieve a hased value from Redis.</i>
-----------	---

---

**Description**

Retrieve a value identified by a key and field from the Redis database.

**Usage**

```
redisHGet(key, field, ...)
```

**Arguments**

key	A key name.
field	A field name.
...	Optional additional agruments. Specify raw=TRUE to skip de-serialization of the data.

**Details**

Redis hash values store values in one or more fields associated with a single key name.

**Value**

The value corresponding to the specified key/field, or NULL if the matching key/field hash contained no value or if no matching key or field was found.

**Author(s)**

B. W. Lewis

**References**

<http://redis.io/commands>

**See Also**

[redisHSet](#)

## Examples

```
## Not run:  
redisHSet('A','x',runif(5))  
redisHGet('A','x')  
  
## End(Not run)
```

---

redisHGetAll	<i>Redis hash fields and values.</i>
--------------	--------------------------------------

---

## Description

Return all fields and values associated with the given key.

## Usage

```
redisHGetAll(key, ...)
```

## Arguments

key	The key to look up.
...	Optional additional arguments. Specify raw=TRUE to skip de-serialization of the data.

## Details

Returns all the fields and their values in the Redis hash associated with key. If the key is not found, or if the hash is empty, NULL is returned. If the key is associated with a value of type other than 'hash,' an error is thrown.

## Value

A list of values defined for the given key, named by the field names.

## Author(s)

B. W. Lewis

## References

<http://redis.io/commands>

## See Also

[redisHSet](#)



**Examples**

```
## Not run:
redisHMSet('A',list(x=1,y=2,z=3))
redisHGetAll('A')

## End(Not run)
```

---

redisHIncrBy	<i>Increment a value.</i>
--------------	---------------------------

---

**Description**

Increment the value corresponding to the given key/field combination by the specified value.

**Usage**

```
redisHIncrBy(key, field, value, ...)
```

**Arguments**

key	A key name.
field	A field name.
value	The value to increment by (integer, numeric, or character).
...	Optional additional arguments. Specify raw=TRUE to skip de-serialization of the data.

**Details**

The value should be an integer (`redisHIncrBy`) or a numeric value (`redisHIncrByFloat`). If the key/field value does not exist or contains a value of a wrong type, set the key to the value of "0" before to perform the operation.

**Value**

The new value of key after the increment, returned as a character string.

**Author(s)**

B. W. Lewis

**References**

<http://redis.io/commands>

**See Also**

[redisHSet](#)

### Examples

```
## Not run:  
# Note initial value must be a raw character string!  
redisHSet('A','x',charToRaw('5'))  
redisHIncrBy('A','x',3)  
  
## End(Not run)
```

---

redisHKeys

*Redis hash fields.*

---

### Description

Return the fields associated with the given key.

### Usage

```
redisHKeys(key, ...)
```

### Arguments

key	The key to look up.
...	Optional additional arguments. Specify raw=TRUE to skip de-serialization of the data.

### Details

Returns the fields in the Redis hash associated with key. If the key is not found, or if the hash is empty, NULL is returned. If the key is associated with a value of type other than 'hash,' an error is thrown.

### Value

A list of fields defined for the given key.

### Author(s)

B. W. Lewis

### References

<http://redis.io/commands>

### See Also

[redisHSet](#)

**Examples**

```
## Not run:
redisHSet('A',list(x=1,y=2,z=3))
redisHKeys('A')

## End(Not run)
```

---

redisHLen	<i>Redis hash length.</i>
-----------	---------------------------

---

**Description**

Return the number of fields associated with the given key.

**Usage**

```
redisHLen(key)
```

**Arguments**

key                    The key to look up.

**Details**

Returns the number of fields in the Redis hash associated with key. If the key is not found, or if the hash is empty, 0 is returned. If the key is associated with a value of type other than 'hash,' an error is thrown.

**Value**

The number of fields defined for the given key.

**Author(s)**

B. W. Lewis

**References**

<http://redis.io/commands>

**See Also**

[redisHSet](#)

### Examples

```
## Not run:  
redisHSet('A',list(x=1,y=2,z=3))  
redisHLen('A')  
  
## End(Not run)
```

---

redisHGet	<i>Retrieve a list of hash values.</i>
-----------	--

---

### Description

Retrieve a list of hash values from specified key/field pairs at once.

### Usage

```
redisHGet(key, fields, ...)
```

### Arguments

key	The (required) character identifier for the key name.
fields	An R string vector of hash fields to retrieve.
...	Optional additional arguments. Specify raw=TRUE to skip de-serialization of the data.

### Details

Redis hash values store values in one or more fields associated with a single key name. The redisHGet function retrieves several fields associated with one key at once. Fields not present return NULL.

### Value

A named list of retrieved values.

### Author(s)

B. W. Lewis

### References

<http://redis.io/commands>

### See Also

[redisHSet](#), [redisHSet](#)

**Examples**

```
## Not run:
redisHMSet('A', list(x=1,y=2,z=3))
redisHMGet('A',c('y','z'))

## End(Not run)
```

---

redisHMSet	<i>Store a list of hash values.</i>
------------	-------------------------------------

---

**Description**

Store a list of hash values at once.

**Usage**

```
redisHMSet(key, values)
```

**Arguments**

key	The (required) character identifier for the key name.
values	An R list of values to be stored. The list names are used as field names and must be nonempty.

**Details**

Redis hash values store values in one or more fields associated with a single key name. The `redisHMSet` function stores several fields associated with one key at once. Values already occupying any specified field slots are replaced.

**Value**

TRUE is returned on success.

**Author(s)**

B. W. Lewis

**References**

<http://redis.io/commands>

**See Also**

[redisHGet](#)

### Examples

```
## Not run:
redisHSet('A', list(x=1,y=2,z=3))
redisHGet('A','y')

## End(Not run)
```

---

redisHSet	<i>Store a hash value in Redis.</i>
-----------	-------------------------------------

---

### Description

Store a value identified by a character key name and field in the Redis database. Any existing value with the same key/field will be replaced by the new value unless NX is set to TRUE.

### Usage

```
redisHSet(key, field, value, NX = FALSE)
```

### Arguments

key	The (required) character identifier for the key name.
field	The (required) character identifier for the field name.
value	The (required) object to associate with the key/field.
NX	If NX = TRUE, existing values will not be replaced.

### Details

Redis hash values store values in one or more fields associated with a single key name.

The value to be stored can be any serializable R object up to the Redis maximum object size (excluding, for example, external pointer references). References to other R objects or environments inside the value are not guaranteed to be preserved.

In order to store strings that can easily be read by other clients, first convert the character object using the [charToRaw](#) function as shown in the examples.

### Value

TRUE is returned on success. If NX = FALSE and a value already exists, the value will not be replaced and FALSE will be returned.

### Author(s)

B. W. Lewis

### References

<http://redis.io/commands>

**See Also**[redisHGet](#)**Examples**

```
## Not run:  
redisHSet('A', 'x',runif(5))  
  
## End(Not run)
```

---

redisHVals	<i>Redis hash values.</i>
------------	---------------------------

---

**Description**

Return all the values associated with the given hash key.

**Usage**

```
redisHVals(key,...)
```

**Arguments**

key	The key to look up.
...	Optional additional arguments. Specify raw=TRUE to skip de-serialization of the data.

**Details**

Returns the values in the Redis hash associated with key. If the key is not found, or if the list is empty, NULL is returned. If the key is associated with a value of type other than 'hash,' an error is thrown.

**Value**

A list of values defined for the given key.

**Author(s)**

B. W. Lewis

**References**

<http://redis.io/commands>

**See Also**[redisHSet](#)

**Examples**

```
## Not run:
redisHSet('A',list(x=1,y=2,z=3))
redisHVals('A')

## End(Not run)
```

---

redisInfo	<i>redisInfo</i>
-----------	------------------

---

**Description**

Return system information about Redis.

**Usage**

```
redisInfo()
```

**Value**

A list of various Redis system parameters is returned, including at least: the Redis version, connected clients, connected slaves, used memory, changes since last save, last save time (UNIX time), total connections received, total commands processed, uptime in seconds, uptime in days.

**Author(s)**

B. W. Lewis

**References**

<http://redis.io/commands>

---

redisKeys	<i>Return a list of all keys in the active Redis database.</i>
-----------	--

---

**Description**

Return a list of all keys in the active Redis database that match the specified pattern.

**Usage**

```
redisKeys(pattern = "*")
```

**Arguments**

pattern            The character string pattern used to match keys.



**Details**

Basic string-matching wildcards are supported. Use '?' to match any single character and '\*' to match zero or more characters.

**Value**

A space-delimited character string containing all keys that match the pattern in the active Redis database.

**Author(s)**

B. W. Lewis

**References**

<http://redis.io/commands>

**See Also**

[redisSelect](#)

**Examples**

```
## Not run:  
redisKeys()  
  
## End(Not run)
```

---

redisLIndex

*Retrieve a value from a Redis 'list.'*

---

**Description**

Retrieve a value from a Redis 'list' at the specified index without removing it from the list.

**Usage**

```
redisLIndex(key, index, ...)
```

**Arguments**

key	The key (whose value is of the type 'list').
index	The list index to retrieve.
...	Optional additional arguments. Specify raw=TRUE to skip de-serialization of the data.

**Details**

List indices begin at 0. Negative indexes are supported, for example -1 is the last element, -2 the penultimate and so on. If the value stored at key is not of the 'list' type an error is returned. If the index is out of range an empty string is returned.

**Value**

The corresponding value or an empty string if the index is out of bounds.

**Author(s)**

B. W. Lewis

**References**

<http://redis.io/commands>

**See Also**

[redisLPop](#)

**Examples**

```
## Not run:  
redisConnect()  
  
## End(Not run)
```

---

redisLLen

*Redis list length.*

---

**Description**

Return the length of the Redis list associated with the specified key.

**Usage**

```
redisLLen(key)
```

**Arguments**

key            The key to look up.

**Details**

Returns the length of the Redis list associated with key. If the key is not found, or if the list is empty, 0 is returned. If the key is associated with a value of type other than 'list,' an error is thrown.

**Value**

The length if the list.

**Author(s)**

B. W. Lewis

**References**

<http://redis.io/commands>

**See Also**

redisBRpop

**Examples**

```
## Not run:
redisConnect()
redisLPush('list',1)
redisLPush('list',2)
redisLPush('list',3)
redisLLen('list')

## End(Not run)
```

---

redisLPop

*Remove the first element from a list.*

---

**Description**

Atomically return and remove the first element of the list. For example if the list contains the elements "a","b","c" redisLPop will return "a" and the list will become "b","c".

**Usage**

```
redisLPop(key, ...)
```

**Arguments**

key	The desired key associated with a list.
...	Optional additional arguments. Specify raw=TRUE to skip de-serialization of the data.

**Value**

The first element of the list associated with the specified key, or NULL if the list is empty.

**Author(s)**

B. W. Lewis

**References**

<http://redis.io/commands>

**See Also**

[redisRPop](#), [redisLPush](#)

**Examples**

```
## Not run:
redisConnect()
redisLPush('a',1)
redisLPush('a',2)
redisLPush('a',3)
redisLPop('a')

## End(Not run)
```

---

redisLPush

*Add a value to the head tail of a list.*

---

**Description**

Add values to the tail (LPush) or head (RPush) of a list corresponding to the specified key.

**Usage**

```
redisLPush(key, value, ...)
```

**Arguments**

key	The desired key corresponding to a list.
value	The element to add to the list.
...	Optional additional values to add to the list.

**Details**

If the key does not exist an empty list is created just before the append operation. If the key exists but is not a list an error is returned.

**Value**

The length of the list after the push operation.

**Author(s)**

B. W. Lewis

**References**

<http://redis.io/commands>

**See Also**

[redisRPush](#)

**Examples**

```
## Not run:
redisConnect()
redisLPush('x',1)
redisLPush('x',2)
redisLPush('x',3)
redisRPush('x',11)
redisRPush('x',21)
redisRPush('x',31)
redisLLen('x')

## End(Not run)
```

---

redisLRange	<i>Copy values from a list.</i>
-------------	---------------------------------

---

**Description**

Return the elements of a list between the start and end indices, inclusively.

**Usage**

```
redisLRange(key, start, end, ...)
```

**Arguments**

key	A key corresponding to a list.
start	The beginning index from which to read list elements.
end	The ending index.
...	Optional additional arguments. Specify raw=TRUE to skip de-serialization of the data.

**Details**

List indices begin with 0. If the start or end indices are beyond the bounds of the list, return the list elements up to the index bounds.

**Value**

An indexed list containing the returned values. An error will be thrown if the specified key does not correspond to a list or if the key can't be found.

**Author(s)**

B. W. Lewis

**References**

<http://redis.io/commands>

**See Also**

[redisLPop](#)

**Examples**

```
## Not run:
redisConnect()
redisLPush('x',1)
redisLPush('x',2)
redisLPush('x',3)
redisLRange('x',0,2)

## End(Not run)
```

---

redisLRem	<i>Remove elements from a list.</i>
-----------	-------------------------------------

---

**Description**

Remove the first count occurrences of the given value from a list corresponding to the specified key.

**Usage**

```
redisLRem(key, count, value)
```

**Arguments**

key	The desired key corresponding to a list.
count	The maximum number of occurrences to remove.
value	Remove elements matching this value from the list.

**Details**

Remove the first count occurrences of the value element from the list. If count is zero all the elements are removed. If count is negative elements are removed from tail to head, instead of the default removal behavior from head to tail. So for example `redisLRem` with count -2 and value "hello" applied to the list (a,b,c,hello,x,hello,hello) will yield the list (a,b,c,hello,x). The number of removed elements is returned as an integer upon success. Note that non-existing keys are considered as empty lists.

**Value**

The number of occurrences removed.

**Author(s)**

B. W. Lewis

**References**

<http://redis.io/commands>

**See Also**

[redisLPush](#)

**Examples**

```
## Not run:  
redisConnect()  
  
## End(Not run)
```

---

redisLSet	<i>Set a value within a list.</i>
-----------	-----------------------------------

---

**Description**

Set the value of an element at the given index in a list corresponding to the specified key.

**Usage**

```
redisLSet(key, index, value)
```

**Arguments**

key	A key corresponding to a Redis list.
index	The index within the list of the element to write to.
value	The value to set.

**Details**

Indices begin at zero. Out of range indices throw an error.

**Value**

TRUE if successful, FALSE otherwise.

**Author(s)**

B. W. Lewis

**References**

<http://redis.io/commands>

**See Also**

[redisLPush](#)

**Examples**

```
## Not run:
redisConnect()
redisLPush('list',pi)
redisLSet('list',0,5)
redisLIndex('list',0)

## End(Not run)
```

---

redisLTrim

*Trim a list.*

---

**Description**

Trim an existing list so that it will contain only the specified range of elements specified.

**Usage**

```
redisLTrim(key, start, end)
```

**Arguments**

key	A key corresponding to a Redis list object.
start	The starting list index at which to trim.
end	The ending list index at which to trim.



**Details**

Start and end are zero-based indexes. 0 is the first element of the list (the list head), 1 the next element and so on.

**Value**

TRUE if successful. An error is thrown if the key does not exist or correspond to a list value.

**Author(s)**

B. W. Lewis

**References**

<http://redis.io/commands>

**See Also**

[redisLPush](#)

**Examples**

```
## Not run:  
redisConnect()  
redisLPush('x',1)  
redisLPush('x',2)  
redisLPush('x',3)  
redisLTrim('x',0,1)  
redisLRange('x',0,99)  
  
## End(Not run)
```

---

redisMGet

*Retrieve one or more values from Redis.*

---

**Description**

Retrieve values corresponding to the specified list of keys.

**Usage**

```
redisMGet(keys, ...)
```

**Arguments**

keys            A vector or list of character strings corresponding to keys to retrieve.  
...             Optional arguments. Specify raw=TRUE to skip de-serialization of the returned values.

**Details**

Values are returned in a list with names corresponding to keys. The raw argument is used to retrieve binary data from other languages.

**Value**

A list of retrieved key/value pairs. Missing values return NULL.

**Author(s)**

B. W. Lewis

**References**

<http://redis.io/commands>

**See Also**

[redisMSet](#)

**Examples**

```
## Not run:  
redisSet('x',runif(5))  
redisSet('y',runif(5))  
redisMGet(c('x','y'))  
  
## End(Not run)
```

---

redisMonitorChannels *redisMonitorChannels*

---

**Description**

Subscribe to one or more Redis message channels.

**Usage**

```
redisMonitorChannels()
```

**Details**

(From the Redis.io documentation): `redisSubscribe`, `redisUnsubscribe` and `redisPublish` implement the Publish/Subscribe messaging paradigm where (citing Wikipedia) senders (publishers) are not programmed to send their messages to specific receivers (subscribers). Rather, published messages are characterized into channels, without knowledge of what (if any) subscribers there may be. Subscribers express interest in one or more channels, and only receive messages that are of interest, without knowledge of what (if any) publishers there are.

The `redisMonitorChannels` function may be called repeatedly in an event loop to service messages on all subscribed channels. When a message is received, the `redisMonitorChannels` function will attempt to evaluate a callback function with same name as the channel, with the message as its single argument. If no such function can be found, the message is returned. See the help page for `redisGetResponse` for a description of the message format.

**WARNING:** The `redisMonitorChannels` function blocks indefinitely until a message is received. Use the lower-level `redisGetResponse` function to simply poll channels for messages without evaluating function callbacks.

### Value

The result of an evaluated function callback message, or if no matching callback exists, the message.

### Author(s)

B. W. Lewis

### References

<http://redis.io/commands>

### See Also

[redisSubscribe](#) [redisPublish](#) [redisUnsubscribe](#) [redisGetResponse](#)  
[redisMonitorChannels](#)

### Examples

```
## Not run:
redisConnect()
# Define a callback function to process messages from channel 1:
channel1 <- function(x) {
  cat("Message received from channel 1: ",x,"\n")
}
# Define a callback function to process messages from channel 2:
channel2 <- function(x) {
  cat("Message received from channel 2: ",x,"\n")
}
redisSubscribe(c('channel1','channel2'))
# Monitor channels for at least 1 minute:
t1 <- proc.time()[[3]]
while(proc.time()[[3]] - t1 < 60)
{
  redisMonitorChannels()
  Sys.sleep(0.05)
}
redisUnsubscribe(c('channel1','channel2'))

## End(Not run)
```

---

redisMove	<i>Move the specified key/value pair to another database.</i>
-----------	---

---

**Description**

Move the specified key/value pair in the currently selected database to another database.

**Usage**

```
redisMove(key, dbindex)
```

**Arguments**

key	The key to move.
dbindex	The destination database index number.

**Details**

This command returns TRUE only if the key was successfully moved, and FALSE if the target key was already there or if the source key was not found at all. It is possible to use redisMove as a locking primitive.

**Value**

Returns TRUE if the key/value pair was moved, or FALSE otherwise.

**Author(s)**

B. W. Lewis

**References**

<http://redis.io/commands>

**See Also**

[redisSelect](#)

**Examples**

```
## Not run:
redisConnect()
redisSelect(1)
redisSet('x',5)
redisMove('x',2)
redisSelect(2)
redisGet('x')

## End(Not run)
```

---

redisMSet	<i>Set one or more key/value pairs in the Redis database.</i>
-----------	---

---

**Description**

Set one or more key/value pairs in the Redis database.

**Usage**

```
redisMSet(keyvalues, NX = FALSE, ...)
```

**Arguments**

keyvalues	A list of values to set of the form list(key1=value1, key2=value2, ...)
NX	If NX = TRUE, existing values will not be replaced, otherwise they will be.
...	Optional additional arguments. Specify raw=TRUE to skip de-serialization of the data.

**Details**

Set one or more key/value pairs in the Redis database. Existing values will be replaced.

**Value**

“OK” upon success.

**Author(s)**

B. W. Lewis

**References**

<http://redis.io/commands>

**See Also**

[redisGet](#)

**Examples**

```
## Not run:  
redisMSet(list(x=5, y=6))  
  
## End(Not run)
```

`redisMulti`*redisMulti*

---

**Description**

Start a transaction block.

**Usage**

```
redisMulti()
```

**Details**

Begin a transaction block. All Redis statements issued after `redisMulti` will be queued locally and then sent to the Redis server en masse once the `redisExec` function is issued.

**Value**

"OK" indicates the transaction has begun.

**Author(s)**

B. W. Lewis

**References**

<http://redis.io/commands>

---

`redisPersist`*Clear expiration flags for a key*

---

**Description**

Clear expiration flags for a key.

**Usage**

```
redisPersist(key)
```

**Arguments**

`key`            The (required) character identifier for the value to be looked up.

**Details**

Remove the existing timeout on key, turning the key from volatile (a key with an expire set) to persistent (a key that will never expire as no timeout is associated).

**Value**

1 if the timeout was removed, 0 if the key does not exists or does not have a timeout set.

**Author(s)**

B. W. Lewis

**References**

<http://redis.io/commands>

**See Also**

[redisTTL](#), [redisExpire](#)

**Examples**

```
## Not run:  
redisSet('x',runif(5))  
redisExpire('x',10)  
redisPersist('x')  
  
## End(Not run)
```

---

redisPublish

*redisPublish*

---

**Description**

Publish data to a Redis message channel.

**Usage**

```
redisPublish(channel, message)
```

**Arguments**

channel	The channel name (character).
message	The message (any type, use RAW type for communication with non-R subscribers).

## Details

(From the Redis.io documentation): `redisSubscribe`, `redisUnsubscribe` and `redisPublish` implement the Publish/Subscribe messaging paradigm where (citing Wikipedia) senders (publishers) are not programmed to send their messages to specific receivers (subscribers). Rather, published messages are characterized into channels, without knowledge of what (if any) subscribers there may be. Subscribers express interest in one or more channels, and only receive messages that are of interest, without knowledge of what (if any) publishers there are.

Use the Redis function `redisUnsubscribe` to unsubscribe from one or more channels. Service incoming messages on the channels with either `redisGetResponse` or `redisMonitorChannels`.

Use of any other Redis after `redisSubscribe` prior to calling `redisUnsubscribe` will result in an error.

## Value

A list conforming to the Redis subscribe response message. Each subscribed channel corresponds to three list elements, the header 'subscribe' followed by the channel name and a count indicating the total number of subscriptions.

## Author(s)

B. W. Lewis

## References

<http://redis.io/commands>

## See Also

[redisSubscribe](#) [redisUnsubscribe](#) [redisPublish](#) [redisGetResponse](#)  
[redisMonitorChannels](#)

## Examples

```
## Not run:  
redisConnect()  
redisPublish('channel1', charToRaw('A raw character data message example'))  
  
## End(Not run)
```

---

`redisRandomKey`

*Return a randomly selected key from the currently selected database.*

---

## Description

Return a randomly selected key from the currently selected database.



**Usage**

```
redisRandomKey()
```

**Details**

Note that this function returns key names, not values.

**Value**

A character string corresponding to a randomly-selected key from the currently selected database, or a zero-length string if no keys are defined.

**Author(s)**

B. W. Lewis

**References**

<http://redis.io/commands>

**Examples**

```
## Not run:  
redisConnect()  
redisRandomKey()  
  
## End(Not run)
```

---

redisRename	<i>Rename a key.</i>
-------------	----------------------

---

**Description**

Atomically rename a key.

**Usage**

```
redisRename(old, new, NX = FALSE)
```

**Arguments**

old	Original key name.
new	New key name.
NX	Set NX = TRUE to prevent overwriting a key that already exists.

**Details**

If the source and destination name are the same an error is returned. If newkey already exists it is overwritten unless `NX = TRUE`, in which case the operation fails.

**Value**

Returns the Redis status "OK" unless `NX = TRUE`. When `NX = TRUE`, returns 1 if successful, 0 otherwise.

**Author(s)**

B. W. Lewis

**References**

<http://redis.io>

**See Also**

[redisKeys](#)

**Examples**

```
## Not run:  
redisConnect()  
redisSet('x',5)  
redisRename('x','y')  
  
## End(Not run)
```

---

redisRPop

*Remove the last element from a list.*

---

**Description**

Atomically return and remove the last element of the list associated with the specified key. For example if the list contains the elements "a","b","c" `redisRPop` will return "c" and the list will become "a","b".

**Usage**

```
redisRPop(key, ...)
```

**Arguments**

key	The desired key associated with a list.
...	Optional additional arguments. Specify <code>raw=TRUE</code> to skip de-serialization of the data.

**Value**

The first element of the list associated with the specified key, or NULL if the list is empty.

**Author(s)**

B. W. Lewis

**References**

<http://redis.io/commands>

**See Also**

[redisLPop](#), [redisLPush](#)

**Examples**

```
## Not run:
redisConnect()
redisLPush('a',1)
redisLPush('a',2)
redisLPush('a',3)
redisRPop('a')

## End(Not run)
```

---

redisRPopLPush	<i>Remove the tail from a list, pushing to another.</i>
----------------	---

---

**Description**

Atomically return and remove the last (tail) element of the src list, and push the element as the first (head) element of the dst list.

**Usage**

```
redisRPopLPush(src, dest, ...)
```

**Arguments**

src	A key corresponding to the source list.
dest	A key corresponding to the destination list.
...	Optional additional arguments. Specify raw=TRUE to skip de-serialization of the data.

## Details

Atomically return and remove the last (tail) element of the src list, and push the element as the first (head) element of the dst list. For example if the source list contains the elements "a","b","c" and the destination list contains the elements "foo","bar" after a `redisRPopLPush` command the content of the two lists will be "a","b" and "c","foo","bar".

If the key does not exist or the list is already empty the special value `NULL` is returned. If the `srckey` and `dstkey` are the same the operation is equivalent to removing the last element from the list and pushing it as first element of the list, so it's a "list rotation" command.

See the Redis reference below for programming examples and discussion.

## Value

The value moved or rotated across lists, or `NULL` if the source key does not exist or corresponds to an empty list. An error is thrown if either of the keys does not correspond to a value of 'list' type.

## Author(s)

B. W. Lewis

## References

<http://redis.io/commands>

## See Also

[redisLPush](#)

## Examples

```
## Not run:
redisConnect()
redisLPush('x',1)
redisLPush('x',2)
redisLPush('x',3)
redisRPopLPush('x','x')
redisRPopLPush('x','x')

## End(Not run)
```

---

redisSAdd

*Add an element to a set.*

---

## Description

Add the element to the specified set.

**Usage**

```
redisSAdd(set, element)
```

**Arguments**

set	The string set identifier.
element	The object to add to the set.

**Details**

If member is already a member of the set no operation is performed. If key does not exist a new set with the specified member as sole member is created. If the key exists but does not hold a set value an error is returned.

**Value**

TRUE if the element was successfully added, FALSE otherwise (including cases in which the element already belonged to the set).

**Author(s)**

B. W. Lewis

**References**

<http://redis.io/commands>

**See Also**

[redisSRem](#)

**Examples**

```
## Not run:  
redisConnect()  
redisSAdd("set", 5)  
  
## End(Not run)
```

redisSave

*redisSave*

---

**Description**

Synchronously save that database to disk.

**Usage**

```
redisSave()
```

**Details**

Force Redis to save the database(s) to disk.

**Value**

Nothing is returned. Check the UNIX time of the last completed save operation with the `redisLastsave` function.

**Author(s)**

B. W. Lewis

**References**

<http://redis.io/commands>

---

redisSCard

*Set cardinality*

---

**Description**

Return the number of elements contained in the specified set.

**Usage**

```
redisSCard(set)
```

**Arguments**

`set`            A string set identifier.

**Value**

The function returns an integer corresponding to the number of elements in the set. Zero is returned if the set is empty or if the set does not exist.

**Author(s)**

B. W. Lewis

**References**

<http://redis.io/commands>

**See Also**

[redisSAdd](#)

**Examples**

```
## Not run:
redisConnect()
redisSAdd("set",5)
redisSCard("set")

## End(Not run)
```

---

redisSDiff

*Return the difference of two or more sets.*

---

**Description**

Return the (set) difference of two or more sets.

**Usage**

```
redisSDiff(keys, ...)
```

**Arguments**

keys	A vector or list of keys corresponding to sets. May also be a single key.
...	Additional keys corresponding to sets. See the examples below.

**Details**

The first argument may be a vector of set names, a list of set names, or a single set name. If only a single set name is specified, specify more sets as additional function arguments as shown in the examples.

**Value**

A list of elements in the difference of the specified sets, or NULL if the intersection is the empty set.

**Author(s)**

B. W. Lewis

**References**

<http://redis.io/commands>

**See Also**

[redisSDiff](#)

**Examples**

```
## Not run:
redisConnect()
redisSAdd('A',1)
redisSAdd('A',2)
redisSAdd('B',4)
redisSDiff('A','B')
redisSDiff(c('A','B'))
redisSDiff(list('A','B'))

## End(Not run)
```

---

redisSDiffStore	<i>Store the difference of two or more sets.</i>
-----------------	--

---

**Description**

Store the (set) difference of two or more sets in the specified set.

**Usage**

```
redisSDiffStore(dest, keys, ...)
```

**Arguments**

dest	The destination set in which to store the result.
keys	A vector or list of keys corresponding to sets. May also be a single key.
...	Additional keys corresponding to sets. See the examples below.

**Details**

The keys argument may be a vector of set names, a list of set names, or a single set name. If only a single set name is specified, specify more sets as additional function arguments as shown in the examples.



**Value**

A redis status code.

**Author(s)**

B. W. Lewis

**References**

<http://redis.io/commands>

**See Also**

[redisSDiff](#)

**Examples**

```
## Not run:
redisConnect()
redisSAdd('A',1)
redisSAdd('A',2)
redisSAdd('A',3)
redisSAdd('B',2)
redisSDiffStore('C','A','B')

## End(Not run)
```

---

redisSelect	<i>Select a redis database.</i>
-------------	---------------------------------

---

**Description**

Select a database by numeric index value.

**Usage**

```
redisSelect(index)
```

**Arguments**

index            The nonnegative integer value of the database to select.

**Details**

Redis supports multiple databases indexed by nonnegative integers. The number of supported databases is configurable via the redis server configuration file. New connections are assigned to database 0 by default.

**Value**

A character string status code. An error is thrown if the index value is invalid.

**Author(s)**

B. W. Lewis

**References**

<http://redis.io/commands>

**Examples**

```
## Not run:  
redisConnect()  
redisSelect(1)  
  
## End(Not run)
```

---

redisSet	<i>Store a value in Redis.</i>
----------	--------------------------------

---

**Description**

Store a value identified by a character key name in the Redis database. Any existing value with the same key will be replaced by the new value unless NX is set to TRUE.

**Usage**

```
redisSet(key, value, NX = FALSE)
```

**Arguments**

key	The (required) character identifier for the value to be stored.
value	The (required) object to associate with the key.
NX	If NX = TRUE, existing values will not be replaced.

**Details**

The value is copied to the Redis server. The value to be stored can be any serializable R object up to the Redis maximum object size (excluding, for example, external pointer references). References to other R objects or environments inside the value are not guaranteed to be preserved.

In order to store strings that can easily be read by other clients, first convert the character object using the [charToRaw](#) function as shown in the examples.

**Value**

The value "OK" is returned upon success (conforming to the usual Redis behavior).

**Author(s)**

B. W. Lewis

**References**

<http://redis.io/commands>

**See Also**

[redisGet](#)

**Examples**

```
## Not run:
# Store an R object with the key 'x':
redisSet('x',runif(5))

## End(Not run)
## Not run:
# Store a string that can be easily read by other clients:
redisSet('x',charToRaw('Hello Redis clients'))

## End(Not run)
```

---

redisSetBit	<i>Redis BITSET - set binary value</i>
-------------	--

---

**Description**

Sets or clears the bit at offset in the string value stored at key.

**Usage**

```
redisSetBit(key, offset, bit)
```

**Arguments**

key	redis key
offset	integer index to be updated
bit	binary number to be set

**Value**

bit binary number with previous value, or '0' if it had not been set before.

---

redisSetContext	<i>redisSetContext</i>
-----------------	------------------------

---

### Description

Get or set the current active Redis connection environment.

### Usage

```
redisSetContext( e = NULL)
redisGetContext()
```

### Arguments

**e** An environment representing the new Redis server connection context returned by `redisConnect(returnRef=TRUE)`. The default value of `NULL` sets the context to the most recently established connection.

### Details

The `redis` package stores information associated with a connection to a Redis server in an environment. The `redisSetContext` and `redisGetContext` functions help manage simultaneous connection to multiple Redis servers.

The `redisSetContext` function returns an environment representing the current active Redis connection. If there is no current active Redis connection, an environment is still returned, but without connection information.

The `redisSetContext` function replaces the current active environment. Any number of simultaneous connections to multiple Redis servers may be managed in this manner.

### Value

`NULL` is invisibly returned.

### Author(s)

B. W. Lewis

### See Also

[redisGetContext](#) [redisConnect](#)

### Examples

```
## Not run:
# Open a connection to a Redis server on HOST1 and store its context:
HOST1 <- redisConnect(host='HOST1', returnRef=TRUE)
print(redisInfo())
```

```

# Initiate a new Redis context:
HOST2 <- redisConnect(host='HOST2', returnRef=TRUE)
# The connection to HOST2 is now active:
print(redisInfo())

# We may now switch back and forth between the two active connections:
redisSetContext(HOST1)
print(redisInfo())
redisSetContext(HOST2)
print(redisInfo())
redisClose()
redisSetContext(HOST1)
redisClose()

## End(Not run)

```

---

redisSetPipeline	<i>Set the Redis message blocking state.</i>
------------------	--

---

### Description

Use `redisSetBlocking` to set the `rredis` client to blocking (default) or non-blocking Redis communication mode.

### Usage

```

redisSetBlocking( value = TRUE )
redisSetPipeline( value = FALSE )

```

### Arguments

value	TRUE indicates Redis pipelined mode, FALSE non-pipelined mode. Just to be extra confusing, it's the opposite for the older, now deprecated function <code>redisSetBlocking</code> . Just use the new <code>redisSetPipeline</code> .
-------	--

### Details

The `rredis` client blocks for a response from a connected Redis server after each transaction in non-pipelined mode (the default). When in pipelined mode, transactions are issued without servicing Redis server responses, and server responses must be manually serviced with the `redisGetResponse` function.

Pipelined mode can improve performance in some circumstances like lots of repeated `redisSet` operations. When using pipelined mode, don't forget to periodically service responses from the Redis server (results are cached on the server until requested).

Note that use of commands like `redisMSet` can sometimes obviate the need to use non-blocking mode.

**Value**

The new pipeline mode (TRUE/FALSE) is invisibly returned.

**Note**

The function name `redisSetBlocking` is deprecated and will be replaced in a future version by `redisSetPipeline`.

**Author(s)**

B. W. Lewis

**See Also**

[redisGetResponse](#)

**Examples**

```
## Not run:  
redisConnect()  
redisSetBlocking(FALSE)  
redisLPush('x',pi)  
redisLPush('x',exp(1))  
redisGetResponse()  
redisSetBlocking(TRUE)  
  
## End(Not run)
```

---

redisShutdown

*redisShutdown*

---

**Description**

Request the currently currently connected Redis server to shutdown, then close the connection.

**Usage**

```
redisShutdown()
```

**Details**

This will terminate the connected Redis server process in an orderly way.

**Value**

Nothing is returned. An error is thrown if no server is connected.

**Author(s)**

B. W. Lewis

**References**

<http://redis.io/commands>

---

redisSInter

*Find and return the intersection of two or more sets.*

---

**Description**

Find and return the intersection of two or more sets.

**Usage**

```
redisSInter(keys, ...)
```

**Arguments**

keys	A vector or list of keys corresponding to sets. May also be a single key.
...	Additional keys corresponding to sets. See the examples below.

**Details**

The first argument may be a vector of set names, a list of set names, or a single set name. If only a single set name is specified, specify more sets as additional function arguments as shown in the examples.

**Value**

A list of elements in the intersection of the specified sets, or NULL if the intersection is the empty set.

**Author(s)**

B. W. Lewis

**References**

<http://redis.io/commands>

**See Also**

[redisSUnion](#)

**Examples**

```
## Not run:
redisConnect()
redisSAdd('A',1)
redisSAdd('A',2)
redisSAdd('A',3)
redisSAdd('B',2)
redisSInter('A','B')
redisSInter(c('A','B'))
redisSInter(list('A','B'))

## End(Not run)
```

---

redisSInterStore	<i>Store intersection of two or more sets.</i>
------------------	--

---

**Description**

Find the intersection of two or more sets, storing the result in the specified set.

**Usage**

```
redisSInterStore(dest, keys, ...)
```

**Arguments**

dest	The destination set in which to store the result.
keys	A vector or list of keys corresponding to sets. May also be a single key.
...	Additional keys corresponding to sets. See the examples below.

**Details**

The keys argument may be a vector of set names, a list of set names, or a single set name. If only a single set name is specified, specify more sets as additional function arguments as shown in the examples.

**Value**

A redis status code.

**Author(s)**

B. W. Lewis

**References**

<http://redis.io/commands>



**See Also**[redisSUnionStore](#)**Examples**

```
## Not run:
redisConnect()
redisSAdd('A',1)
redisSAdd('A',2)
redisSAdd('A',3)
redisSAdd('B',2)
redisSInterStore('C','A','B')

## End(Not run)
```

---

redisSIsMember	<i>Test for set membership</i>
----------------	--------------------------------

---

**Description**

Test if an element belongs to a set.

**Usage**

```
redisSIsMember(set, element)
```

**Arguments**

set	A string set identifier.
element	The element whose membership is to be tested.

**Value**

TRUE if the element is a member of the set, FALSE otherwise.

**Author(s)**

B. W. Lewis

**References**

<http://redis.io/commands>

**See Also**[redisSAdd](#)

**Examples**

```
## Not run:
redisConnect()
redisSAdd("set",5)
redisSIsMember("set",5)

## End(Not run)
```

---

 redisSlaveOf

*redisSlaveOf*


---

**Description**

Change Redis replication settings.

**Usage**

```
redisSlaveOf(host, port)
```

**Arguments**

host	The host name of a master Redis server.
port	The port of a master Redis server.

**Details**

(The following details are adapted from the Redis Wiki manual referenced below.)

The `redisSlaveOf` command can change the replication settings of a slave on the fly. If a Redis server is already acting as slave, the function `redisSlaveOf(host="no",port="one")` will turn off the replication turning the Redis server into a MASTER. In the proper form `redisSlaveOf(hostname,port)` will make the server a slave of the specific server listening at the specified hostname and port.

If a server is already a slave of some master, `redisSlaveOf(hostname,port)` will stop the replication against the old server and start the synchronization against the new one discarding the old dataset.

The form `redisSlaveOf(host="no",port="one")` will stop replication, turning the server into a MASTER but will not discard the replication. So if the old master stop working it is possible to turn the slave into a master and set the application to use the new master in read/write. Later when the other Redis server will be fixed it can be configured in order to work as slave.

**Value**

A Redis status message is returned.

**Author(s)**

B. W. Lewis

## References

<http://redis.io/commands>

---

redisSMembers	<i>List elements of a set.</i>
---------------	--------------------------------

---

## Description

Return a list containing all the members (elements) of the specified set.

## Usage

```
redisSMembers(set, ...)
```

## Arguments

set	The set key name (character).
...	Optional additional arguments. Specify raw=TRUE to skip de-serialization of the data.

## Value

A list containing the set elements. An error is thrown if the set does not exist.

## Author(s)

B. W. Lewis

## References

<http://redis.io/commands>

## See Also

[redisSAdd](#)

## Examples

```
## Not run:  
redisConnect()  
redisSAdd('set',runif(5))  
redisSMembers('set')  
  
## End(Not run)
```

---

redisSMove	<i>Move a set element.</i>
------------	----------------------------

---

**Description**

Move the specified element from one set to another.

**Usage**

```
redisSMove(setA, setB, element)
```

**Arguments**

setA	The set to move from (character identifier).
setB	The set to move to (character identifier).
element	The element to move.

**Details**

Note that the set element is specified as the actual set element object (aka set member), not as an index value.

**Value**

TRUE if successful, FALSE otherwise.

**Author(s)**

B. W. Lewis

**References**

<http://redis.io/commands>

**See Also**

[redisSAdd](#)

---

redisSort	<i>redisSort</i>
-----------	------------------

---

**Description**

Sort a list, set or sorted set (zset).

**Usage**

```
redisSort(key, decreasing = FALSE, alpha = FALSE, by = NULL,
          start = NULL, count = NULL, get = NULL, store = NULL, ...)
```

**Arguments**

key	The key name of the list, set or zset to be sorted.
decreasing	Set the direction of the sort.
alpha	Lexicographically sort if true, otherwise try numeric sorting.
by	The BY option takes a pattern that is used in order to generate the key names of the weights used for sorting. Weight key names are obtained substituting the first occurrence of * with the actual value of the elements on the list. See the Redis documentation for examples.
start	Starting index of the sort.
count	Number of entries past start to use for the sort.
get	Retrieve external keys. See the Redis documentation.
store	Store the results in the specified key.
...	Optional additional arguments. Specify raw=TRUE to skip de-serialization of the data.

**Details**

Sort the elements contained in the List, Set, or Sorted Set value at key. By default sorting is numeric with elements being compared as double precision floating point numbers.

**Value**

A list of sorted values, unless store is specified in which case the results are stored in the specified key and TRUE is returned.

**Author(s)**

B. W. Lewis

**References**

<http://redis.io/commands>

---

`redisSPop`*Remove and return an element from a set.*

---

**Description**

Remove and return an element, chosen at random, from the specified set.

**Usage**

```
redisSPop(set, ...)
```

**Arguments**

<code>set</code>	The set name (character).
<code>...</code>	Optional additional arguments. Specify <code>raw=TRUE</code> to skip de-serialization of the data.

**Value**

A random element of the set, or NULL if the set is empty or does not exist.

**Author(s)**

B. W. Lewis

**References**

<http://redis.io/commands>

**See Also**

[redisSAdd](#)

**Examples**

```
## Not run:
redisConnect()
redisSAdd('set',runif(2))
redisSAdd('set',runif(3))
redisSAdd('set',runif(4))
redisSPop('set')
redisSPop('set')
redisSPop('set')
redisSPop('set')

## End(Not run)
```

---

redisSRandMember	<i>Choose a random element from a set</i>
------------------	---

---

### Description

Select and return an element of a set at random.

### Usage

```
redisSRandMember(set, ...)
```

### Arguments

set	The set key name (character).
...	Optional additional arguments. Specify raw=TRUE to skip de-serialization of the data.

### Value

An element from the set selected randomly. An error is thrown if the set does not exist.

### Author(s)

B. W. Lewis

### References

<http://redis.io/commands>

### See Also

[redisSAdd](#)

### Examples

```
## Not run:  
redisConnect()  
redisSAdd('set',1)  
redisSAdd('set',2)  
redisSAdd('set',3)  
redisSRandMember('set')  
  
## End(Not run)
```

redisSRem

*Remove an element from a set.*

---

**Description**

Remove an element from a set.

**Usage**

```
redisSRem(set, element)
```

**Arguments**

set	The set name (character) from which to remove the element.
element	The element to remove (not an index, but the actual element value).

**Details**

The set element matching the specified element will be removed from the set.

**Value**

TRUE upon successful removal, FALSE otherwise.

**Author(s)**

B. W. Lewis

**References**

<http://redis.io/commands>

**See Also**

[redisSAdd](#)

**Examples**

```
## Not run:
redisConnect()
redisSAdd('set', 5)
redisSAdd('set', 7)
redisSMembers('set')
redisSRem('set',5)
redisSMembers('set')

## End(Not run)
```



---

redisSubscribe	<i>redisSubscribe</i>
----------------	-----------------------

---

**Description**

Subscribe to one or more Redis message channels.

**Usage**

```
redisSubscribe(channels, pattern=FALSE)
```

**Arguments**

channels	A character vector or list of channel names to subscribe to.
pattern	If TRUE, allow wildcard pattern matching in channel names, otherwise names indicate full channel names.

**Details**

(From the Redis.io documentation): `redisSubscribe`, `redisUnsubscribe` and `redisPublish` implement the Publish/Subscribe messaging paradigm where (citing Wikipedia) senders (publishers) are not programmed to send their messages to specific receivers (subscribers). Rather, published messages are characterized into channels, without knowledge of what (if any) subscribers there may be. Subscribers express interest in one or more channels, and only receive messages that are of interest, without knowledge of what (if any) publishers there are.

Use the Redis function `redisUnsubscribe` to unsubscribe from one or more channels. Service incoming messages on the channels with either `redisGetResponse` or `redisMonitorChannels`.

Use of any other Redis after `redisSubscribe` prior to calling `redisUnsubscribe` will result in an error.

**Value**

A list conforming to the Redis subscribe response message. Each subscribed channel corresponds to three list elements, the header 'subscribe' followed by the channel name and a count indicating the total number of subscriptions.

**Author(s)**

B. W. Lewis

**References**

<http://redis.io/commands>

**See Also**

[redisSubscribe](#) [redisPublish](#) [redisUnsubscribe](#) [redisGetResponse](#)  
[redisMonitorChannels](#)

**Examples**

```
## Not run:
redisConnect()
# Define a callback function to process messages from channel 1:
channel1 <- function(x) {
  cat("Message received from channel 1: ",x,"\n")
}
# Define a callback function to process messages from channel 2:
channel2 <- function(x) {
  cat("Message received from channel 2: ",x,"\n")
}
redisSubscribe(c('channel1','channel2'))
# Monitor channels for at least 1 minute:
t1 <- proc.time()[[3]]
while(proc.time()[[3]] - t1 < 60)
{
  redisMonitorChannels()
  Sys.sleep(0.05)
}
redisUnsubscribe(c('channel1','channel2'))

## End(Not run)
```

---

redisUnion

*Return the union of two or more sets.*


---

**Description**

Return the union of two or more sets.

**Usage**

```
redisUnion(keys, ...)
```

**Arguments**

**keys**            A vector or list of keys corresponding to sets. May also be a single key.  
**...**            Additional keys corresponding to sets. See the examples below.

**Details**

The first argument may be a vector of set names, a list of set names, or a single set name. If only a single set name is specified, specify more sets as additional function arguments as shown in the examples.

**Value**

A list of elements in the union of the specified sets, or NULL if the intersection is the empty set.

**Author(s)**

B. W. Lewis

**References**

<http://redis.io/commands>

**See Also**

[redisSInter](#)

**Examples**

```
## Not run:
redisConnect()
redisSAdd('A',1)
redisSAdd('A',2)
redisSAdd('B',4)
redisSUnion('A','B')
redisSUnion(c('A','B'))
redisSUnion(list('A','B'))

## End(Not run)
```

---

redisSUnionStore	<i>Store the union of two or more sets.</i>
------------------	---

---

**Description**

Store the union of two or more sets in the specified set.

**Usage**

```
redisSUnionStore(dest, keys, ...)
```

**Arguments**

dest	The destination set in which to store the result.
keys	A vector or list of keys corresponding to sets. May also be a single key.
...	Additional keys corresponding to sets. See the examples below.

**Details**

The keys argument may be a vector of set names, a list of set names, or a single set name. If only a single set name is specified, specify more sets as additional function arguments as shown in the examples.

**Value**

A redis status code.

**Author(s)**

B. W. Lewis

**References**

<http://redis.io/commands>

**See Also**

[redisSInterStore](#)

**Examples**

```
## Not run:
redisConnect()
redisSAdd('A',1)
redisSAdd('A',2)
redisSAdd('A',3)
redisSAdd('B',2)
redisSUnionStore('C','A','B')

## End(Not run)
```

---

redisTTL

*Return the time to live for a key set to expire.*

---

**Description**

She the time left to live in seconds (redisTTL) or milliseconds (redisPTTL) for the specified key.

**Usage**

```
redisTTL(key)
redisPTTL(key)
```

**Arguments**

key            The key to look up.

**Details**

Redis keys may be set to expire at or after a given time with the `redisExpire` and `redisExpireAt` functions. This function shows the time left before expiration in seconds for such a key.

**Value**

A Redis string value representation of the integer time left to live or -1 if the key is not set to expire or -2 if not found.

**Author(s)**

B. W. Lewis

**References**

<http://redis.io/commands>

**See Also**

[redisExpire](#), [redisExpireAt](#)

**Examples**

```
## Not run:
redisConnect()
redisSet('x',5)
redisExpire('x',100)
redisTTL('x')

## End(Not run)
```

---

redisType

*Query a Redis value type.*

---

**Description**

Return a description of the type of the value corresponding to the specified key.

**Usage**

```
redisType(key)
```

**Arguments**

key            The key to look up.

### Details

The Redis database differentiates three types of values. The value types determine how Redis operates on the key/value pairs, not what kind of data may be contained within. Each value type may store generic R objects, binary blobs, or character strings. The Redis value types are:

string: A single value is associated with the key

list: A stack of values is associated with the key

set: A set of values is associated with the key.

Values of "list" and "set" types have available to them special stack and set operations, respectively.

### Value

A descriptive character string that may be one of "none", "string", "list", or "set". "none" is returned if the key does not exist.

### Author(s)

B. W. Lewis

### References

<http://redis.io/commands>

### See Also

[redisKeys](#)

### Examples

```
## Not run:
redisConnect()
redisSet('x',5)
redisLPush('y',6)
redisType('x')
redisType('y')

## End(Not run)
```

---

redisUnsubscribe

*redisUnsubscribe*

---

### Description

Subscribe to one or more Redis message channels.

### Usage

```
redisUnsubscribe(channels, pattern=FALSE)
```

## Arguments

channels	A character vector or list of channel names to subscribe to.
pattern	If TRUE, allow wildcard pattern matching in channel names, otherwise names indicate full channel names.

## Details

(From the Redis.io documentation): `redisSubscribe`, `redisUnsubscribe` and `redisPublish` implement the Publish/Subscribe messaging paradigm where (citing Wikipedia) senders (publishers) are not programmed to send their messages to specific receivers (subscribers). Rather, published messages are characterized into channels, without knowledge of what (if any) subscribers there may be. Subscribers express interest in one or more channels, and only receive messages that are of interest, without knowledge of what (if any) publishers there are.

Use the Redis function `redisUnsubscribe` to unsubscribe from one or more channels. Service incoming messages on the channels with either `redisGetResponse` or `redisMonitorChannels`.

Use of any other Redis after `redisSubscribe` prior to calling `redisUnsubscribe` will result in an error.

## Value

A list conforming to the Redis subscribe response message. Each subscribed channel corresponds to three list elements, the header element 'unsubscribe' followed by the channel name and a count indicating the total number of subscriptions.

## Author(s)

B. W. Lewis

## References

<http://redis.io/commands>

## See Also

[redisSubscribe](#) [redisUnsubscribe](#) [redisPublish](#) [redisGetResponse](#)  
[redisMonitorChannels](#)

## Examples

```
## Not run:
redisConnect()
# Define a callback function to process messages from channel 1:
channel1 <- function(x) {
  cat("Message received from channel 1: ",x,"\n")
}
# Define a callback function to process messages from channel 2:
channel2 <- function(x) {
  cat("Message received from channel 2: ",x,"\n")
}
```

```
redisSubscribe(c('channel1','channel2'))
# Monitor channels for at least 1 minute:
t1 <- proc.time()[[3]]
while(proc.time()[[3]] - t1 < 60)
{
  redisMonitorChannels()
  Sys.sleep(0.05)
}
redisUnsubscribe(c('channel1','channel2'))

## End(Not run)
```

---

redisUnwatch

*redisUnwatch*

---

### Description

Stop watching the specified keys for changes.

### Usage

```
redisUnwatch(keys)
```

### Arguments

keys            A character vector or list of key names to stop watching.

### Details

Stop watching the specified keys. See `redisWatch` for details.

### Value

"OK"

### Author(s)

B. W. Lewis

### References

<http://redis.io/commands>



---

`redisWatch`*redisWatch*

---

**Description**

Condition a transaction block on value changes.

**Usage**

```
redisWatch(keys)
```

**Arguments**

keys            A character vector or list of key names to watch for changes.

**Details**

Watch the specified keys for changed values. If any of the values change for the watched keys, discard the current transaction block and unwatch all watched keys.

**Value**

"OK"

**Author(s)**

B. W. Lewis

**References**

<http://redis.io/commands>

---

`redisZAdd`*redisZAdd*

---

**Description**

Add an element to a Redis sorted set. Sorted sets order their elements by numeric weights.

**Usage**

```
redisZAdd(key, score, member)
```

**Arguments**

key	The set name.
score	The numeric score associated with the new element (member).
member	The new object to add to the set.

**Author(s)**

B. W. Lewis

**References**

<http://redis.io/commands>

---

redisZCard

*redisZCard.Rd*

---

**Description**

Report the number of elements in a sorted set.

**Usage**

redisZCard(key)

**Arguments**

key	The name of the set.
-----	----------------------

**Details**

See the Redis documentation for details.

**Value**

The number of elements in the set or zero if the set does not exist. An error is thrown if the key name exists but is not a set.

**Author(s)**

B. W. Lewis

**References**

<http://redis.io/commands>

---

redisZCount	<i>redisZCount</i>
-------------	--------------------

---

**Description**

Returns the number of elements in the sorted set at key with a score between min and max. The min and max arguments have the same semantic as described for `redisZRangeByScore`.

**Usage**

```
redisZCount(key, min, max)
```

**Arguments**

key	The set name.
min	Minimum score.
max	Maximum score.

**Value**

See the Redis documentation for more information.

**Author(s)**

B. W. Lewis

**References**

<http://redis.io/commands>

---

redisZIncrBy	<i>redisZIncrBy</i>
--------------	---------------------

---

**Description**

Increment the weight associated with an ordered set element.

**Usage**

```
redisZIncrBy(key, member, increment)
```

**Arguments**

key	The set name.
member	The set element.
increment	The numeric value by which to increase the weight associated with the element.

**Details**

See the Redis documentation for details.

**Value**

The new weight associated with the element.

**Author(s)**

B. W. Lewis

**References**

<http://redis.io/commands>

---

redisZInterStore	<i>redisZInterStore</i>
------------------	-------------------------

---

**Description**

Store the intersection of two sorted sets into another sorted set.

**Usage**

```
redisZInterStore(dstkey, keys, weights = c(), aggregate = NULL, ...)
```

**Arguments**

dstkey	The destination set name.
keys	A vector or list of sorted set names.
weights	A set of weights equal in length to the keys argument, or NULL (default). If specified, the weights associated with the members of each set will be multiplied by the specified weight values.
aggregate	A character value of either "SUM" "MIN" or "MAX" indicating how the resulting weights should be aggregated.
...	Optional additional arguments. Specify raw=TRUE to skip de-serialization of the data.

**Details**

See the Redis documentation for details.

**Value**

The number of elements stored in the destination set.

**Author(s)**

B. W. Lewis

**References**<http://redis.io/commands>

---

`redisZRange`*redisZRange*

---

**Description**

Return a subset of an ordered set.

**Usage**

```
redisZRange(key, start = 0, end = -1, decreasing = FALSE,
            withscores = FALSE, ...)
```

**Arguments**

<code>key</code>	The set name.
<code>start</code>	Starting index value.
<code>end</code>	Ending index value.
<code>decreasing</code>	Set TRUE to sort in decreasing order of scores.
<code>withscores</code>	If TRUE, also return set element scores.
<code>...</code>	Optional additional arguments. Specify <code>raw=TRUE</code> to skip de-serialization of the data.

**Details**

Return the specified elements of the sorted set at the specified key. The elements are considered sorted from the lowest to the highest score if `decreasing = FALSE`, and from highest to lowest score if `decreasing = TRUE`. Start and end are zero-based indexes. 0 is the first element of the sorted set (the one with the lowest score), 1 the next element by score and so on. start and end can also be negative numbers indicating offsets from the end of the sorted set. For example -1 is the last element of the sorted set, -2 the penultimate element and so on.

Indexes out of range will not produce an error: if start is over the end of the sorted set, or start is greater than end, an empty list is returned. If end is over the end of the sorted set Redis will treat it just like the last element of the sorted set.

**Value**

A list of returned set elements. If `withscores = TRUE`, two lists called 'elements' and 'scores' are returned.

**Author(s)**

B. W. Lewis

**References**

<http://redis.io/commands>

---

redisZRangeByScore     *redisZRangeByScore*

---

**Description**

Return the all the elements in the sorted set at key with a score between min and max (including elements with score equal to min or max).

**Usage**

```
redisZRangeByScore(key, min, max, offset = NULL, count = NULL,  
                  withscores = FALSE, ...)
```

**Arguments**

key	The set name.
min	Minimum score.
max	Maximum score.
offset	Limit the results by excluding the first offset items. Requires also the use of count.
count	Limit the results to the first count items. Requires also the use of offset.
withscores	Also return the scores.
...	Optional additional arguments. Specify raw=TRUE to skip de-serialization of the data.

**Details**

The offset and count arguments must both be specified if used.

**Value**

See the Redis documentation for more information.

**Author(s)**

B. W. Lewis

**References**

<http://redis.io/commands>

---

redisZRank	<i>redisZRank</i>
------------	-------------------

---

**Description**

Return the rank of an element of an ordered set.

**Usage**

```
redisZRank(key, member, decreasing = FALSE)
```

**Arguments**

key	The set name.
member	The set element.
decreasing	Set to TRUE to rank in reverse order of score.

**Details**

Returns the rank of the member in the sorted set, with scores ordered from low to high by default, or from high to low if `decreasing = TRUE`. The returned rank (or index) of the member is 0-based.

**Value**

The numeric rank of the set element.

**Author(s)**

B. W. Lewis

**References**

<http://redis.io/commands>

---

redisZRem	<i>redisZRem</i>
-----------	------------------

---

**Description**

Delete an element from an ordered set.

**Usage**

```
redisZRem(key, member)
```

**Arguments**

key	The set name.
member	The element to delete.

**Details**

See the Redis documentation for details.

**Value**

0 or 1: 0 indicates that the element could not be removed, 1 that it was removed.

**Author(s)**

B. W. Lewis

**References**

<http://redis.io/commands>

---

redisZRemRangeByRank *redisZRemRangeByRank.Rd*

---

**Description**

redisZRemRangeByRank.Rd

**Usage**

```
redisZRemRangeByRank(key, start, end)
```

**Arguments**

key  
start  
end

**Details**

See the Redis documentation for details.

**Value**

See the Redis documentation for more information.

**Author(s)**

B. W. Lewis



## References

<http://redis.io/commands>

---

redisZRemRangeByScore *redisZRemRangeByScore.Rd*

---

## Description

redisZRemRangeByScore.Rd

## Usage

redisZRemRangeByScore(key, min, max)

## Arguments

key

min

max

## Details

See the Redis documentation for details.

## Value

See the Redis documentation for more information.

## Author(s)

B. W. Lewis

## References

<http://redis.io/commands>

redisZScore            *redisZScore*

---

**Description**

Return the score of an ordered set element.

**Usage**

```
redisZScore(key, element)
```

**Arguments**

key	The set name.
element	The set element.

**Value**

The numeric score.

**Author(s)**

B. W. Lewis

**References**

<http://redis.io/commands>

---

redisZUnionStore            *redisZUnionStore*

---

**Description**

Store the union of two sorted sets into another sorted set.

**Usage**

```
redisZUnionStore(dstkey, keys, weights = c(), aggregate = NULL, ...)
```

**Arguments**

<code>dstkey</code>	The destination set name.
<code>keys</code>	A vector or list of sorted set names.
<code>weights</code>	A set of weights equal in length to the <code>keys</code> argument, or NULL (default). If specified, the weights associated with the members of each set will be multiplied by the specified weight values.
<code>aggregate</code>	A character value of either "SUM" "MIN" or "MAX" indicating how the resulting weights should be aggregated.
<code>...</code>	Optional additional arguments. Specify <code>raw=TRUE</code> to skip de-serialization of the data.

**Details**

See the Redis documentation for details.

**Value**

The number of elements stored in the destination set.

**Author(s)**

B. W. Lewis

**References**

<http://redis.io/commands>

# Index

- \*Topic **package**
  - rredis-package, 4
- charToRaw, 28, 38, 66
- Increment, Decrement functions, 4
- redisAuth, 6
- redisBgRewriteAOF, 6
- redisBgSave, 7
- redisBitCount, 8
- redisBitOp, 8
- redisBLOp, 9
- redisBRPop, 10
- redisBRPopLPush, 11
- redisClose, 12, 15
- redisCmd, 13
- redisConnect, 13, 14, 68
- redisDBSize, 16
- redisDecr, 5
- redisDecr (Increment, Decrement functions), 4
- redisDecrBy (Increment, Decrement functions), 4
- redisDelete, 17
- redisDiscard, 18
- redisEval, 18
- redisExec, 19
- redisExists, 20
- redisExpire, 21, 22, 55, 85
- redisExpireAt, 21, 22, 85
- redisFlushAll, 23, 24
- redisFlushDB, 23, 23
- redisGet, 24, 28, 53, 67
- redisGetBit, 25
- redisGetContext, 13, 15, 68
- redisGetContext (redisSetContext), 68
- redisGetResponse, 15, 26, 51, 56, 70, 81, 87
- redisGetSet, 27
- redisHDel, 28
- redisHExists, 29
- redisHFields, 30
- redisHGet, 31, 37, 39
- redisHGetAll, 32
- redisHIncrBy, 33
- redisHIncrByFloat (redisHIncrBy), 33
- redisHKeys, 34
- redisHLen, 35
- redisHMGet, 36
- redisHMSet, 36, 37
- redisHSet, 29–36, 38, 39
- redisHVals, 39
- redisIncr (Increment, Decrement functions), 4
- redisIncrBy (Increment, Decrement functions), 4
- redisIncrByFloat (Increment, Decrement functions), 4
- redisInfo, 40
- redisKeys, 40, 58, 86
- redisLIndex, 41
- redisLLen, 42
- redisLPop, 42, 43, 46, 59
- redisLPush, 44, 44, 47–49, 59, 60
- redisLRange, 45
- redisLRem, 46
- redisLSet, 47
- redisLTrim, 48
- redisMGet, 49
- redisMonitorChannels, 26, 50, 51, 56, 81, 87
- redisMove, 52
- redisMSet, 50, 53
- redisMulti, 54
- redisPersist, 54
- redisPexpire (redisExpire), 21
- redisPexpireAt (redisExpireAt), 22
- redisPTTL (redisTTL), 84
- redisPublish, 26, 51, 55, 56, 81, 87
- redisRandomKey, 56

redisRename, 57  
redisRPop, 44, 58  
redisRPopLPush, 12, 59  
redisRPush, 45  
redisRPush (redisLPush), 44  
redisSAdd, 60, 63, 73, 75, 76, 78–80  
redisSave, 62  
redisSCard, 62  
redisSDiff, 63, 64, 65  
redisSDiffStore, 64  
redisSelect, 16, 41, 52, 65  
redisSet, 17, 20, 25, 66  
redisSetBit, 67  
redisSetBlocking (redisSetPipeline), 69  
redisSetContext, 15, 68  
redisSetPipeline, 15, 69  
redisShutdown, 70  
redisSInter, 71, 83  
redisSInterStore, 72, 84  
redisSIsMember, 73  
redisSlaveOf, 74  
redisSMembers, 75  
redisSMove, 76  
redisSort, 77  
redisSPop, 78  
redisSRandMember, 79  
redisSRem, 61, 80  
redisSubscribe, 26, 51, 56, 81, 81, 87  
redisSUnion, 71, 82  
redisSUnionStore, 73, 83  
redisTTL, 55, 84  
redisType, 85  
redisUnsubscribe, 26, 51, 56, 81, 86, 87  
redisUnwatch, 88  
redisWatch, 89  
redisZAdd, 89  
redisZCard, 90  
redisZCount, 91  
redisZIncrBy, 91  
redisZInterStore, 92  
redisZRange, 93  
redisZRangeByScore, 94  
redisZRank, 95  
redisZRem, 95  
redisZRemRangeByRank, 96  
redisZRemRangeByScore, 97  
redisZScore, 98  
redisZUnionStore, 98  
rredis (rredis-package), 4  
rredis-package, 4