REDIS cheatsheet [v2.0]

starting the server	**cd redis; ./redis-server**
running the client	**./redis-cli <command>** (or with no command, to enter interactive mode)

commands

generic commands for all types

exists *key*	Test if specified key exists. Return: 1 if exists, 0 if not
del *key1 key2 ... keyN*	Remove the specified keys. Return: integer > 0 if keys removed, 0 if none of the keys existed
type *key*	Return the type of the value stored at *key*, as a string. Return: "none", "string", "list", "set"
keys *pattern*	Return all keys matching pattern. Ex: keys h*llo, keys h?llo, keys h[aeo]llo Return: bulk reply string with keys separated by spaces
randomkey	Return a randomly-selected key from the current database. Return: the selected key, or empty string if database is empty
rename *oldkey newkey* **renamenx** *oldkey newkey*	Atomically renames key. **renamenx** fails if *newkey* exists and returns 0. Return: 1 if OK, 0 if *oldkey* doesn't exist or if it equals *newkey*
dbsize	Returns the number of keys in the current database. Return: integer, the number of keys
expire *key seconds* **expireat** *key unixtime*	Sets timeout on the specified key. Return: 1 if timeout set, 0 if key already has a timeout or doesn't exist
ttl *key*	Returns remaining time to live, in seconds, for a key with EXPIRE set. Return: integer number of seconds, or -1 if key doesn't exist or has no expiration
select *db-index*	Selects a database by index (zero-based). Default database is 0. Return: 1 if OK, 0 if error
move *key db-index*	Moves a key from current database to specified database. Return: 1 if OK, 0 if key doesn't exist or is already present in the target database

commands for strings

set *key value* **setnx** *key value*	Sets the value of *key* to the string *value*; setnx will not overwrite an existing value. Return: 1 if OK, 0 if error
get *key*	Gets the value of *key*. Return: string value if OK, "nil" if *key* does not exist
getset *key value*	Atomically sets the value of *key* to the string *value* and returns old value of *key*. Return: value of *key* prior to the new value being set ("nil" if *key* did not exist)
mget *key1 key2 ... keyN*	Gets the values of all specified keys. Return: multi-bulk reply of all values, with "nil" for any keys that do not exist
mset *key1 value1 ... keyN valueN* **msetnx** *key1 value1 ... keyN valueN*	Sets the values of the keys to the string values; msetnx will not overwrite existing values if any key exists. Return: 1 if all keys were set, 0 if none were set
incr *key* **decr** *key*	Increments/decrements value of *key* by 1. Return: New value after increment/decrement operation
incrby *key integer* **decrby** *key integer*	Increments/decrements value of *key* by the integer value specified. Return: New value after increment/decrement operation

transactions

multi **<command1>** ... **<commandn>** **exec** *or* **discard**	Performs set of commands within a transaction, in a single step. Either all or none of the commands will be processed. **exec** will cause the commands to be processed; **discard** will abandon all of the commands. Return: **exec** returns multi bulk reply with the return value of each command **discard** returns OK

REDIS cheatsheet page 2

commands operating on lists

rpush *key string* **lpush** *key string*	Adds the string to the head (rpush) or tail (lpush) of the list at *key*. Return: 1 if exists, 0 if *key* exists but is not a list
llen *key*	Returns the length of the list at *key*. Return: integer length, or error if *key* is not a list
lrange *key start end*	Returns the elements of list at *key*, zero-based. Negative numbers are offset from the end of the list. Return: requested elements or empty list if no match
ltrim *key start end*	Trims list at key to contain only the specified elements. Return: 1 if OK, error if *key* is not a list
lindex *key index*	Returns the element at the specified index of the list *key*. Return: the requested item; empty string if no such element; error if *key* isn't a list
lset *key index value*	Sets the element of list *key* at *index* to the specified value. Return: 1 if OK, error if index out of range or *key* isn't a list
lrem *key count value*	Removes *count* number of items from the list that have the specified value. Count 0 will remove all; negative count starts from the end. Return: # items removed
lpop *key string* **rpop** *key string*	Atomically removes and returns the first (lpop) or last (rpop) element from list *key*. Return: the element, or "nil" if empty/nonexistent list; error if *key* isn't a list
blpop *key1...keyN timeout* **brpop** *key1...keyN timeout*	Blocking pop, returns when a specified list contains an element. Return: key and popped value, or "nil" if operation times out
rpoplpush *srckey destkey*	Atomically returns last element from *srckey* and pushes as first element to *destkey*. Return: element popped/pushed, or "nil" if *srckey* empty or nonexistent

commands operating on sets

sadd *key member*	Adds *member* to the set stored at *key*. Return: 1 if OK, 0 if element was already a set member; error if *key* isn't a set
srem *key member*	Removes *member* from set *key*. Return: 1 if OK, 0 element not a set member; error if *key* isn't a set
spop *key* **srandmember** *key*	Returns random element from set *key*. spop will remove the element. Return: element, or nil object if *key* is empty or doesn't exist
smove *srckey dstkey member*	Atomically moves *member* from set *srckey* to set *dstkey*. Return: 1 if OK, 0 if element not found in *srckey*; error if either key isn't a set
scard *key*	Returns the number of elements in set *key*. Return: integer number of elements; 0 if empty or *key* doesn't exist
sismember *key member*	Return whether *member* is in set *key*. Return: 1 if element is a member, 0 if not or if *key* doesn't exist
sinter *key1 key2...keyN* **sinterstore** *dstkey key1...keyN*	Returns the members resulting from intersection of sets specified. sinterstore will store results in new set and return status code.
sunion *key1 key2...keyN* **sunionstore** *dstkey key1...keyN*	Returns the members resulting from union of sets specified. sunionstore will store results in new set and return status code.
sdiff *key1 key2...keyN* **sdiffstore** *dstkey key1...keyN*	Returns the members resulting from the difference between the first set and the rest. sdiffstore will store results in new set and return status code.
smembers *key*	Returns all of the members of set *key*. This is sinter, for only one set. Return: the members

sort *key [**by** pattern] [**limit** start count] [**get** pattern] [**asc**|**desc**] [**alpha**] [**store** dstkey]*

Sorts the elements in the list, set, or sorted set at *key*. Default sort is numeric, ascending. Specifying **asc** or **desc** will sort in ascending or descending order. Specifying **alpha** will sort alphabetically. **limit** will return *count* number of elements beginning at offset *start* (zero-based). **store** will put the results of the sort into a list with key *dstkey*.

Specifying "**by** *pattern*" will sort using the values at keys generated using the pattern. For example, if the list/set being sorted contains the values 1, 2, 3 then "sort by weight_*" will sort using the values at keys "weight_1", "weight_2", "weight_3".

Specifying "**get** *pattern*" will retrieve the values stored at keys generated using the pattern. For example, "get items_*" will return the values at keys items_1, items_2, items_3 if the list/set being sorted contains the values 1, 2, 3.

REDIS cheatsheet page 3

commands operating on sorted sets

zadd *key score member*	Adds *member* to zset *key*, with specified score. Return: 1 if added, 0 if element was already a member and score was updated
zrem *key member*	Removes *member* from zset *key*. Return: 1 if removed, 0 if element was not a member
zincrby *key incr member*	Increments score of *member* by *incr* and updates element's position in zset. Return: integer, the new score of *member* after the increment
zrank *key member* **zrevrank** *key member*	Returns the rank of the member in the sorted set, zero-based; returns nil if member doesn't exist. **zrevrank** returns the rank in high-to-low order.
zrange *key start end* **zrevrange** *key start end*	Returns elements in zset *key* within the specified index range, sorted in order (or reverse with zrevrange). Option: "withscores" will also return scores.
zrangebyscore *key min max* *[***limit** *offset count]* *[***withscores***]*	Returns elements in zset *key* with scores within the specified range. Option "withscores" will also return scores with the elements.
zremrangebyrank *key start end*	Removes elements from zset *key* with rank between *start* and *end*. Negative numbers will start from the end. Return: integer, number of elements removed
zremrangebyscore *key min max*	Removes elements from zset *key* with scores between *min* and *max*. Return: integer, number of elements removed
zcard *key*	Returns the number of elements in the zset *key*. Return: integer, the number of elements; returns 0 if *key* doesn't exist
zscore *key element*	Returns the score of the specified element in zset *key*. Return: the score, as a string; or "nil" if *key* or *element* don't exist
zunion/zinter *dstkey N k1...kN* *[* ***weights*** *w1...wN]* *[* ***aggregate sum|min|max*** *]*	Creates union or intersection of N sorted sets named by keys *k1...kN*, and stores it in key *dstkey*. The number of input keys, N, must be specified. The **weights** option will multiply the scores by the provided weights. The **aggregate** option changes how the scores are aggregated into the new set. Returns the number of elements in the new set at *dstkey*.

zsets available in redis 1.1 and later. zrank, zrevrank, zrangebyscore [withscores], zremrangebyrank in redis 1.3.4. zunion/zinter in redis 1.3.5.

commands operating on hashes

hset *key field value* **hsetnx** *key field value*	Sets hash *field* to the *value*. Will create a new hash if *key* does not exist. hsetnx will do nothing if field already exists. Return: 1 if new field was created, 0 if existing field was updated or already exists
hget *key field*	Returns value of *field* stored in hash *key*. Return: value of *field*, or nil if *field* or *key* do not exist
hmset *key field1 value1 .. fieldN valueN*	Sets fields to the values provided, replacing existing values if any. Creates new hash if none exists at *key*.
hincrby *key field value*	Increments value in field. Negative values will decrement. If field does not exist or holds a string, will reset to zero first. Return: the new value of field
hexists *key field*	Returns 1 if *field* exists in hash at *key*. Returns 0 if *key* or *field* don't exist.
hdel *key field*	Removes *field* from hash stored at *key*. Return: 1 if field removed, 0 if field was not present
hlen *key*	Returns the number of fields in hash stored at *key*, or 0 if *key* does not exist.
hkeys *key* **hvals** *key*	Returns the keys or values of the hash stored at *key*.
hgetall *key*	Returns the keys and values of the hash stored at *key*, as a multi bulk reply in the form field1, value1, ..., fieldN, valueN.

hash commands available in redis 1.3.10.

REDIS cheatsheet page 4

publish / subscribe commands

subscribe *channel1 ... channelN*	Subscribes the client to the specified channels.
psubscribe *pattern1 ... patternN*	Subscribes the client to any channels matching the specified patterns.
unsubscribe *[channel1 ... channelN]*	Unsubscribes the client from the specified channels, or from all channels if no channels are specified. Returns a message for each unsubscribed channel, and the number of channels still subscribed to.
punsubscribe *[pattern1 ... patternN]*	Unsubscribes the client from channels matching the specified patterns, or from all channels if no patterns are specified. Returns a message for each unsubscribed channel, and the number of channels still subscribed to.
publish *channel message*	Sends a message that will be received by all clients subscribed to the specified channel. Returns the number of clients that received the message.

When a message is received by a client, the message type will be *message* or *pmessage* to indicate how the client is subscribed to the originating channel. If *pmessage*, the pattern matched will be provided. For both, the originating channel and the message are then provided.

publish/subscribe commands in redis 1.3.8.

persistence and control commands

save	Saves all databases to disk. Connection requests will not be served during the save. Returns OK when complete.
bgsave	Saves all databases to disk in the background. Redis forks and writes so the parent process continues to process connection requests.
lastsave	Returns integer unix time of last successful save. This can be used following a **bgsave** to check if it was successful.
bgrewriteaof	Rewrites the Append Only File in the background.
shutdown	Stops all clients, saves databases, and quits.
info	Returns information and statistics about the server.
monitor	Used to view commands for debugging. Telnet to redis server then enter **monitor** command. Enter **quit** to end the session.
slaveof *host port* **slaveof no one**	Makes server the replication slave of the redis server at *host*/*port*. The "no one" form turns off replication, making the server a master.
quit	Tells server to close the connection immediately.
auth *password*	Authorizes client using the provided password, if redis server is configured with *requirepass*. Returns OK or error if *password* is incorrect.
flushdb	Deletes all keys in the currently-selected database. Return: 1 -- this command never fails
flushall	Deletes all keys in all existing databases. Return: 1 -- this command never fails

redis site: http://code.google.com/p/redis/
mailing list: http://groups.google.com/group/redis-db