

Redis Cheatsheet

Essential commands for in-memory data structure operations

This cheatsheet provides a quick reference to fundamental Redis operations, syntax, and advanced features. Redis is an open-source, in-memory data structure store used as a database, cache, and message broker. It supports various data structures such as strings, hashes, lists, sets, and more. Ideal for both beginners and experienced developers for efficient data processing and caching.

String Operations

Store and manipulate text values

Data Structures

Work with lists, sets, and hashes

Key Management

Manage keys and expiration

Database Operations

Administer Redis instances

Performance Monitoring

Track and optimize Redis performance

Redis Installation & Setup

Get Redis up and running on your system and connect to the server.

Docker: `docker run redis`

Quickest way to get Redis running locally.

```
# Run Redis in Docker
docker run -name my-redis -p 6379:6379 -d redis
# Connect to Redis CLI
docker exec -it my-redis redis-cli
# Run with persistent storage
docker run -name redis-persistent -p 6379:6379 -v redis-data:/data -d redis
```

Linux: `sudo apt install redis`

Install Redis server on Ubuntu/Debian systems.

```
# Install Redis
sudo apt update
sudo apt install redis-server
# Start Redis service
sudo systemctl start redis-server
# Enable auto-start on boot
sudo systemctl enable redis-server
# Check status
sudo systemctl status redis
```

Connect & Test: `redis-cli`

Connect to Redis server and verify installation.

```
# Connect to local Redis
redis-cli
```

```
# Test connection
redis-cli PING
```

```
# Connect to remote Redis
redis-cli -h hostname -p 6379 -a password
```

```
# Execute single command
redis-cli SET mykey "Hello Redis"
```

Basic String Operations

Set & Get: `SET` / `GET`

Store simple values (text, numbers, JSON, etc.)

```
# Set a key-value pair
SET mykey "Hello World"
# Get value by key
GET mykey
# Set with expiration (in seconds)
SET session:123 "user_data" EX 3600
# Set only if key doesn't exist
SET mykey "new_value" NX
```

Number Operations: `INCR` / `DECR`

Increment or decrement integer values stored in Redis.

```
# Increment by 1
INCR counter
# Decrement by 1
DECR counter
# Increment by specific amount
INCRBY counter 5
# Increment float
INCRBYFLOAT price 0.1
```

String Manipulation: `APPEND` / `STRLEN`

Modify and inspect string values.

```
# Append to existing string
APPEND mykey " - Welcome!"
# Get string length
STRLEN mykey
# Get substring
GETRANGE mykey 0 4
# Set substring
SETRANGE mykey 6 "Redis"
```

Multiple Operations: `MSET` / `MGET`

Work with multiple key-value pairs efficiently.

```
# Set multiple keys at once
MSET key1 "value1" key2 "value2" key3 "value3"
# Get multiple values
MGET key1 key2 key3
# Set multiple only if none exist
MSETNX key1 "val1" key2 "val2"
```

List Operations

Lists are ordered sequences of strings, useful as queues or stacks.

Add Elements: `LPUSH` / `RPUSH`

Add elements to the left (head) or right (tail) of a list.

```
# Add to head (left)
LPUSH mylist "first"
# Add to tail (right)
RPUSH mylist "last"
# Add multiple elements
LPUSH mylist "item1" "item2" "item3"
```

Access Elements: `LRANGE` / `LINDEX`

Retrieve elements or ranges from lists.

```
# Get entire list
LRANGE mylist 0 -1
# Get first 3 elements
LRANGE mylist 0 2
# Get specific element by index
LINDEX mylist 0
# Get list length
LLEN mylist
```

Remove Elements: `LPOP` / `RPOP`

Remove and return elements from list ends.

```
# Remove from head
LPOP mylist
# Remove from tail
RPOP mylist
# Blocking pop (wait for element)
BLPOP mylist 10
```

List Utilities: `LSET` / `LTRIM`

Modify list contents and structure.

```
# Set element at index
LSET mylist 0 "new_value"
# Trim list to range
LTRIM mylist 0 99
# Find position of element
LPOS mylist "search_value"
```

Set Operations

Sets are collections of unique, unordered string elements.

Basic Set Operations: `SADD` / `SMEMBERS`

Add unique elements to sets and retrieve all members.

```
# Add elements to set
SADD myset "apple" "banana" "cherry"
# Get all set members
SMEMBERS myset
# Check if element exists
SISMEMBER myset "apple"
# Get set size
SCARD myset
```

Set Operations: `SINTER` / `SUNION`

Perform mathematical set operations.

```
# Intersection of sets
SINTER set1 set2
# Union of sets
SUNION set1 set2
# Difference of sets
SDIFF set1 set2
# Store result in new set
SINTERSTORE result set1 set2
```

Set Modifications: `SREM` / `SPOP`

Remove elements from sets in different ways.

```
# Remove specific elements
SREM myset "banana"
# Remove and return random element
SPOP myset
# Get random element without removing
SRANDMEMBER myset
```

Set Utilities: `SMOVE` / `SSCAN`

Advanced set manipulation and scanning.

```
# Move element between sets
SMOVE source_set dest_set "element"
# Scan set incrementally
SSCAN myset 0 MATCH "a*" COUNT 10
```

Hash Operations

Hashes store field-value pairs, like mini JSON objects or dictionaries.

Basic Hash Operations: `HSET` / `HGET`

Set and retrieve individual hash fields.

```
# Run HSET command
HSET user:123 name "John Doe" age 30
# Get hash field
HGET user:123 name
# Set multiple fields
HMSET user:123 email "john@example.com" city "NYC"
# Get multiple fields
HGETALL user:123 name age email
```

Hash Utilities: `HEXISTS` / `HDEL`

Check existence and remove hash fields.

```
# Check if field exists
HEXISTS user:123 email
# Delete fields
HDEL user:123 age city
# Increment hash field
HINCRBY user:123 age 1
# Increment by float
HINCRBYFLOAT user:123 balance 10.50
```

Hash Inspection: `HKEYS` / `HVALS`

Examine hash structure and contents.

```
# Get all field names
HKEYS user:123
# Get all values
HVALS user:123
# Get all fields and values
HGETALL user:123
# Get number of fields
HLEN user:123
```

Lua Scripting: `EVAL` / `SCRIPT`

Execute custom Lua scripts atomically.

```
# Execute Lua script
EVAL "return redis.call('SET', 'key1', 'value1')"
# Load script and get SHA
SCRIPT LOAD "return redis.call('GET', KEYS[1])"
# Execute by SHA
EVALSHA sha1 mykey
# Check script existence
SCRIPT EXISTS sha1
# Rewrite AOF file
BGREWRITEAOF
```

Sorted Set Operations

Sorted sets combine uniqueness of sets with ordering based on scores.

Basic Operations: `ZADD` / `ZRANGEBYSCORE`

Add sorted members and retrieve ranges.

```
# Add members with scores
ZADD leaderboard 100 "player1" 200 "player2"
# Get members by rank (0-based)
ZRANGE leaderboard 0 -1
# Get with scores
ZRANGE leaderboard 0 -1 WITHSCORES
# Get by score range
ZRANGEBYSCORE leaderboard 100 200
# Watch keys for changes
WATCH mykey
```

Modifications: `ZREM` / `ZINCRBY`

Remove members and modify scores.

```
# Remove members
ZREM leaderboard "player1"
# Increment member score
ZINCRBY leaderboard 10 "player2"
# Remove by rank
ZREMRANGEBYRANK leaderboard 0 2
# Remove by score
ZREMRANGEBYSCORE leaderboard 0 100
```

Database Management

Administrative commands for managing Redis databases and server operations.

Database Selection: `SELECT` / `FLUSHDB`

Manage multiple databases within Redis.

```
SELECT 0
# Clear current database
FLUSHDB
# Clear all databases
FLUSHALL
# Get current database size
DBSIZE
```

Persistence: `SAVE` / `BGSAVE`

Control Redis data persistence and backups.

```
# Synchronous save (blocks server)
SAVE
# Background save (non-blocking)
BGSAVE
# Get last save time
LASTSAVE
# Rewrite AOF file
BGREWRITEAOF
```

Server Info: `INFO` / `PING`

Get server statistics and test connectivity.

```
# Test server connection
PING
# Get server information
INFO
# Get specific info section
INFO memory
INFO replication
# Get server time
TIME
```

Client Information: `CLIENT LIST`

Monitor connected clients and connections.

```
# List all clients
CLIENT LIST
# Get client info
CLIENT INFO
# Kill client connection
CLIENT KILL ip:port
# Set client name
CLIENT SETNAME "my-app"
```

Performance Monitoring

Monitor Redis performance, track slow commands, and analyze server metrics.

Real-time Monitoring: `MONITOR` / `SLOWLOG`

Track commands and identify performance bottlenecks.

```
# Monitor all commands in real-time
MONITOR
# Get slow query log
SLOWLOG GET 10
# Get slow log length
SLOWLOG LEN
# Reset slow log
SLOWLOG RESET
```

Hash Scanning: `HSCAN`

Iterate through large hashes incrementally.

```
# Scan hash fields
HSCAN user:123 0 MATCH "addr*" COUNT 10
```

Memory Analysis: `MEMORY USAGE` / `MEMORY STATS`

Analyze memory consumption and optimization.

```
# Get key memory usage
MEMORY USAGE mykey
# Get memory statistics
MEMORY STATS
# Get memory doctor report
MEMORY DOCTOR
# Purge memory
MEMORY PURGE
```

Streams: `XADD` / `XREAD`

Work with Redis streams for log-like data.

```
# Add entry to stream
XADD mystream * field1 value1 field2 value2
# Read from stream
XREAD STREAMS mystream 0
# Get stream length
XLEN mystream
# Create consumer group
XGROUP CREATE mystream mygroup 0
```

Advanced Features

Explore Redis advanced capabilities including transactions, pub/sub, and scripting.

Transactions: `MULTI` / `EXEC`

Execute multiple commands atomically.

```
# Start transaction
MULTI
# Add to transaction
SET key1 "value1"
# Execute transaction
EXEC
```

Scripting: `EVAL` / `SCRIPT`

Execute custom Lua scripts atomically.

```
# Execute Lua script
EVAL "return redis.call('SET', 'key1', 'value1')"
# Load script and get SHA
SCRIPT LOAD "return redis.call('GET', KEYS[1])"
# Execute by SHA
EVALSHA sha1 mykey
# Check script existence
SCRIPT EXISTS sha1
# Rewrite AOF file
BGREWRITEAOF
```

Redis Configuration Tips

Essential configuration settings for production Redis deployments.

Memory Management

Configure memory limits and eviction policies.

```
# Set memory limit
CONFIG SET maxmemory 2gb
# Set eviction policy
CONFIG SET maxmemory-policy allkeys-lru
# Check memory usage
INFO memory
```

Security Settings

Basic security configurations for Redis.

```
# Set password
CONFFIG SET requirepass mypassword
# Authenticate
AUTH mypassword
# Disable dangerous commands
CONFFIG SET rename-command FLUSHALL ...
```

Advanced Features

Quick reference for Redis data type capabilities and use cases.

String Operations

Can store text, numbers, JSON, binary data. Max size: 512MB. Use for: caching, counters, flags.

```
SET user:123:name "John"
GET user:123:name
INCR page:views
```

Lists: Ordered collections

Linked lists of strings. Use for: queues, stacks, activity feeds, recent items.

```
LPUISH queue:jobs "job1"
RPOP queue:jobs
LRANGE recent:posts 0 9
```

Sorted Set Info: `ZCARD` / `ZSCORE`

Get information about sorted set members.

```
# Get set size
ZCARD leaderboard
# Get member score
ZSCORE leaderboard "player1"

```