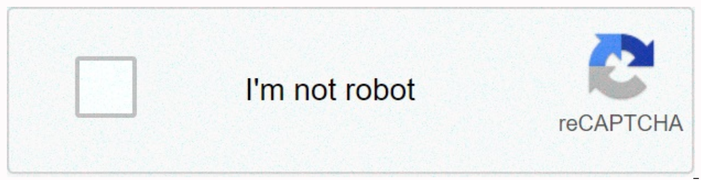

Continue

Redis 4.0.9 cheat sheet

Redis is an in-memory key value store that can be used as a database, cache, and message broker. This project is open source and is currently licensed under a BSD license. Fun fact: Redis means REmote DIctionary server. Redis provides milliseconds or less of response time that enables millions of requests per second to power demanding real-time applications like gaming, ad brokers, financial dashboards, and more! Supports basic data structures such as strings, lists, sets, sorted sets with scope queries, and hashes. Advanced data structures such as geospatial indexes with bitmaps, hyperlogs, and radius queries are also supported. Alex Stanciu, product owner of Auth0's identity governance team, explains one of Redis' use cases. It provides very fast read and write speeds because it stores data in RAM (memory). The response is typically single-digit milliseconds. In this Redis tutorial, you will learn how to set up Redis on your system and how to store data in Redis using core and most frequently used data structures. This foundation will help you learn how to use Redis for caching, session storage, messaging, and real-time analytics in future posts. Let's get started! Redis installation The first thing we need to do is to install Redis. If you're already running on the system, you're free to skip this part of the post. Because redis documentation has no dependencies other than the GCC compiler and libc in which Redis works, it is recommended that you install Redis by compiling it from the source. We redis.io download the latest Redis tarball or always use a special URL that points to the latest stable Redis version: . Windows users: The Redis project does not officially support Windows. However, if you are running Windows 10, you can install and install the Windows subsystem for Linux running Redis. If you are running a Windows subsystem for Linux, please follow all the steps in this post that apply to Linux (if specified) within the Linux shell. Redis to follow these simple steps: create a re-editing directory and check the current working directory: macOS/Linux: mkdir redis & CD redis get the latest redis tarball: macOS /Linux: Curl-O macOS/Linux: Tar xvz f Re-Stability.tar.gz Unpacked Reuse - Stable DirectoryPleased Check current working directory: macOS/Linux: CD Re-Stable macOS/Linux: If the creation package is not installed on your system, follow the instructions provided by the CLI to install it. On macOS, you may need to download XCode to access command-line tools that include the make and C compilers. For example, if you want to install a new Ubuntu, you can run the following command: Update Package Manager and install the core package: Ubuntu: Sudo apt update sudo apt upgrade sudo apt installation build essential sudo apt-get installation tcl8.5 make tcl 8.5 or new required to run Redis test in the next step. Test if the build works correctly: macOS/Linux: Once the compilation is complete, the test, the src directory within the financial stability is populated with other executions that are part of Redis. The Redis article describes the capabilities of each Redis executive: Redis Server: Run Redis Server itself. [Re-ed] Redis Sentinel, the Dispatch=Reporter, monitoring and failover tool, runs. redis-cli: Runs a command-line interface utility that interacts with Redis. Re-benchmark: Check Redis performance. Re-check-ao and re-check dumps: Used in rare cases when corrupted data files are available. We will frequently use redistribution servers and re-run expressions. For convenience, we're copying them all to locations with system-to-system access. macOS/Linux: while sudo cp src/redis-server/usr/local/bin/sudo cp src/redis-cli/usr/local/bin/ reusable as the current working directory, macOS/Linux: sudo can perform the tasks required to restart these shells. Once you do that, you're ready to start running Redis. Redis Startup The easiest way to run Redis is to start the Redis server. Enter in the new Shell window: If everything on the redis server works as expected, the Shell is provided with a giant ASCII Redis logo with the Redis version installed, execution mode, port on which the server runs, and an outline that shows the process identification number (PID). _.-``__ ''-._ _.-`` `. `_. ''-___Redis 4.0.10 (00000000/0) 64-bit .-'. '\/_.-..-_ ` _.-'| Ports: 6379 | `-._ `._ / _.-' | PID: 22394 '-._ '-./ _.-' _.-' _ '-._'-._ _.-'_.-' | '-._ '-._'-._ _.-'_.-' | '-._ '-._'-._ _.-'_.-' | '-._ '-._ therefore use the internal default configuration. Allow for a range of this blog post: Understand and use the default Redis data structure. How to verify that Redis works as mentioned in the Redis article, an external program uses TCP sockets and Redis-specific protocols to talk to Redis. The Redis protocol is implemented by the Redis client library, which is written in many programming languages, such as JavaScript. However, you do not need to use the client library directly. with Redis. We can use redis-cli to send commands directly. To test that Redis is working properly, let's send a ping command. Open a new shell window and run the following command: If everything works well, we need to get the PONG in response to the shell. When we issue a redis-cli ping, we call the command name, ping, and then back to cli executable. The command name and arguments are sent to the Redis instance running on the local host:6379, where they are processed and a reply is sent. You can change the host and port of an instance. Use the -help option to see all commands that can be used with redis-cli: If you run a redis-cli --help argument, the program starts in interactive mode. Like read-ed-print loops (REPL) in programming languages such as Python, you can enter other Redis commands in the shell and get responses from Redis instances. What those commands are and what they do is the core learning goal of this post! Let's start by learning how to manipulate data in Redis using commands! As we learned before redis created, read, updated, and deleted data, Redis is a key value store that can associate some data called values with keys. If you know the exact key used to store it, you can retrieve the stored data later. If you haven't already done so, run the Redis CLI in interactive mode by running the following command: redis-cli we'll know that the interactive CLI works when we see the Redis instance host and port at a shell prompt: 127.0.0.1:6379> there, we are ready to execute the command. To store a value in data write Redis, you can use the SET command with the following signature: If you set a key value in English, it reads like a key setting to hold the value. If the key already holds a value, SET overwrites whatever it does. Let's look at an example. Interactive shell type: As you enter the SET service auth0, you can see how the interactive shell suggests the required and optional arguments required for the Redis command. To send a command, press enter. When Redis saved auth0 as a service value, they replyed OK so everything went well. Thank you, Redis! Read the data We can get to ask Redis for the value of the key: Let's search for the value of the GET key service: GET service Redis reply with auth0. What if you ask for the value of a key that has never been set? GET user Redis replies with (nil) to let you know that the key doesn't exist in memory. The classic API that connects to the database wants to perform CRUD operations such as build, read, update, and delete. It covers how to create (create) and read data in Redis using the SET and GET commands. Let's cover the rest of the data. Data updates can update key values by simply overwriting the data. I mentioned it earlier. You can create new key value pairs: each of the SET frameworks, but we change our minds and now we want the values to be responsive. We can overwrite it as follows: Did the SET framework get Redis right? Let's ask for it! Get framework Redis actually replies with a response. We are a little indecised and now we want to set the framework key to hold the value of the vue: SET framework vue if we run the GET framework again, we get the vue. Updates/overwrites work as exclusions. But delete the data, but we don't actually want to set up any framework right now and we need to delete that key. What should I do? We use the DEL command: DEL key let's run it: DEL framework Redis (integer) 1 and reply to let you know the number of removed keys. Three commands : SET, GET, and DEL can comply with four CRUD operations! Packing a string in quotes gives you something curious: you don't have to put a quote mark around the single string value we wanted to store. Both set framework angle and SET framework angled are allowed by Redis to store string angles as values in the key framework. Redis automatically lapping single string arguments in quotation marks. Both the key and the value are strings, so the same applies to the key name. We could have used the SET framework angle, and it would have worked as well. However, if you want to use more than one string as a key or value, you must set the framework vue response response to OK to enclose the string in quotation marks. Framework Each vue Response Response (Error) ERR Syntax Error SET Framework Each vue Response Also Response (Error) ERR Syntax Error Finally, to retrieve the value, we must use the correct key string: GET Framework for each vue response. Non-destructive writing Redis is compassionate and can write data with care. Assuming you wanted to create a service key that could hold the value of heroku aws, but instead of entering the SET service heroku aws, you entered the SET service heroku aws. This last command will overwrite the current value of the service without mercy. However, Redis provides a nondestructive version of the SET called SETNX: the SETNX key value SETNX only creates a key in memory if the key does not already exist (SET if it is not eXis). If the key already exists, Redis replies with 0, failing to save the key value pair and using 1 to indicate success. Let's try this previous scenario, but let's use SETNX instead of SET: the SETNX service heroku aws response is 0 (integer) as expected. The SETNX service heroku awsThix has a reply of (integer) 1. Good! You can use SETNX to prevent accidental discoloration of your data. When you create a key with a lidis, the keymanry can specify how much that key is stored in memory. You can use the Expiration command to set the time time. Automatically deleted when start and time settings expire: Create a notification key that you want to delete after 30 seconds: Notification more than 30 Notifications 30 Notifications Schedule a notification key to be deleted after 30 seconds. We can look at the clock and see after 30 seconds if the notification is still

available, but we don't need to do that! Redis provides a TTL command that tells you a few seconds before the key expires and is deleted: the TTL key has already passed for more than 30 seconds, so you can try the example above again. This time, you can call the TTL as soon as it expires by calling the TTL: Notification Detection Over 30 TTL Notification Indicates that Redis still responded to a 27-second (integering) notification that responded to the notification 30 TTL notification (integering) 27 seconds in detail. Wait a bit and let the TTL run again: this time with the TTL notification, Ledis answered with -2 (integer). Starting with Redis 2.8, the TTL is returned: the time time remains in seconds. -2 No key (key not generated or deleted). The -1 key exists but does not have an expiration set. I expected -2 because I made sure it was 30 seconds old. If you have a key but expiration is not set, I'm going to send you an error message: SET dialog continue? The TTL dialog box, as expected, without the expiration setting, Redis replies with (integer) -1. It is important to note that we can reset the time time again using the key and SET: SET notification detection expiration notification 30 TTL notification // (integer) 27 SET notification no detection over TTL notification // (integer) -1 We learned previously that using SET is the same as re-creating the key. We now have a solid foundation in manipulating data from Redis. With this knowledge under your belt, you're now ready to explore the types of data redis has to offer. The Redis data type is far from a regular key value store, and Redis is a physical data structure server that supports different kinds of values. In general, the key value store allows you to map string keys to anything different from string values. In Redis, string keys can be mapped to more than just strings. Because it is a data structure server, you can also refer to the data type as a data structure. You can use these more complex data structures to store multiple values in a key at once. Look at these types at a high level. In a later section, we'll look at each format in detail. The most basic kind of Redis value. Because it is binary safe, a string can contain any type of data that is represented by a string. In essence, the Redis list is a linked list. A group of string elements arranged in the order in which they were inserted. Indicates a unique collection. An unalted string element. Like a set, it indicates its own collection of string elements. However, each string element is linked to a floating numeric value called an element score. When querying an ordered set, elements are always sorted by score, so that various data in the set can be presented consistently. A map consisting of string fields associated with string values. Also called bitmaps. Make the string value treated like a bit array. The probability data structure used to estimate the cardinality of a set measures the number of elements in a set. Redis has already covered strings in the Write, Read, Update, and Delete Data sections. The rest of this tutorial will focus on all Redis types except bitmaps and hyperlogrons. We visit people in future posts that handle advanced Redis use cases. Whenever any site is slashed and can no longer be reached, I think in a small blog of 5$/mo VM that is remotely unaffected and resists HN top position pressure. And I think a lot of people are sincerely missing out on the opportunity to use Redis. — ANTIREZ (@antirez) July 12, 2018 List A list is the order of sorted elements. For example, 1 2 4 5 6 90 19 3 is a list of numbers. It is important to note that in Redis, lists are implemented as linked lists. This has some significant impact on performance. Adding elements to the head and tail of a list is quick, but it is slow to retrieve elements within the list because you don't have indexed access to the elements, as in the array. The list generates a key name using the Redis command that pushes the data. There are two commands: RBUSH and LPUSH. Without keys, these commands return a new list with passed arguments as elements. If the key already exists or is not a list, an error is returned. RBUSH RRUSH inserts a new element at the end of the list (tail): RBUSH key value [...] RBUSH Engineer Alice// 1 RBUSH Engineer Bob// 2 RBUSH Engineer Carmen// 3 Each time an element is inserted, Redis replies to the length of the list after that insertion. We would expect the user list to resemble this: Alice Bob Carmen How can we check it out? We can use the LRANGE command. LRANGE LRANGE returns a subset of the list based on the specified start and stop indexes. These indexes are 0-based, but they are not the same as array indexes. Given the full list, they simply indicate where to split the list: check the slice here (start) here (stop): LRANGE key start stop to see the full list, we can use a neat trick: go to the element just before, -1. LRANGE Engineer 0 -1 Redis Return: 1) Alice 2) Bob 3) Carmen Index -1 Indicates the last element in the list. To get the first two elements of an engineer, you can issue the following command: LRANGE Engineer 0 1 LPUSH LPUSH LPUSH works the same as RRUSH except to insert an element at the front of the list (in the header): LPUSH key value [value ...] Let's insert Daniel at the front of the engineer list: LRUSH Engineer Daniel// 4 We now have 4 engineers. Let's make sure the order is correct: LRANGE Engineer 0 -1 Reply with Redis: 1) Daniel2) Alice3) Bob4) CarmenIt's the same list as we did before, but as the first element daniel and exactly what to expect. Insert multiple elements we saw in the signatures of RRUSH and LPUSH that we can insert one or more elements through each command. Let's see the action. Based on our existing engineer list, let's issue this command: RBUSH engineer Yves Francis Gary // 7 Because we inserted them at the end of the list, we expect these three new elements to appear in the same order listed as arguments. Let's check: LRANGE Engineer 0 -1 Redis on what to return: 1) Daniel2) Alice3) Bob4) Carmen5) Eve6) Francis7) How about if Garyuri does the same with LPUSH: LPUSH engineer HugoJes/10 Reds will insert these three new elements once in a while? Let's see: LRANGE 0 -1 reply: 1) Jess 2) Ivan3) Hugo 4) Daniel 5) Alice 6) Bob 7) Carmen 8) Eve 9) Francis 10) Gary LPUSH, and when listing multiple arguments for RBUSH, Redis inserts elements one by one so that Hugo, Ivan and Jess appear in reverse order listed as arguments. LLEN We can find the length of the list at any time using the LLEN command: Let's see if the length of the LLEN key engineer is actually 10: LLEN Engineer Redis (integer) 10 times and reply. Perfect. If you remove an element from the Redis list, similar to how it pops up in an array, you can pop the element from the head or tail of the red list. LPOP removes and returns the first element in the list: the LPOP key we can use to remove Jess, the first element, from the list: LPOP engineer Redis actually indicates the removed element replying with Jess. Remove the RPOP and return the last element of the list: the RPOP key is the time it says goodbye to Gary, the last element in the list: the response from RPOP engineer Redis is Gary. It is very useful to be able to obtain elements that have been removed from the list. Redis List is implemented as a linked list because the engineering team envisioned that database systems should be able to add elements to a very long list in a very fast way. Redis, which is a set, is similar to a list, except that it does not maintain a specific order for that element. The element must be unique. SADD We create a set using the SADD command that adds the specified member to the key: The specified member that is already part of the SADD key member [member ...] set is ignored. If no key exists, a new set is created and a unique specified member is added. If the key already exists or is not a set, an error is returned. Let's create a language set of: SADD language English // 1 SADD language Spanish // 1 SADD language French // 1 In this case, each member returns the number of members added with the Additional RedisSAdd command, not the size of the set. Let's take a look at this task: Sadd language Chinese Japanese German // 3 SADD language English // 0 The first command returned 3 while adding three unique members to the set. The second command returned 0 in English and was already a member of the set. You can use the SREM SREM command to remove a member from a set: SREM key member [Member ...] SREM Language English French // 2 SREM Language German // 0 SREM returns the number of members removed. To verify that the SISMEMBER member is part of a set, the SISMEMBER command: If the SISMEMBER key member member is part of a set, this command can use Return1. Otherwise, the SISMEMBER language returns 0 because you removed German from the last section of Spanish// 1 SISMEMBER language German// 0. S members can use the S member command to display all members that exist in the set: SMEMBERS key Let's look at the language values in the current language set: S member language Redis return: 1) Chinese 2) Japanese 3) As Spanish sets are not sorted, Redis can return elements in any order on all calls. There is no guarantee of element order. SUNION A really powerful thing we can do with a very fast set is to combine them using the SUNION command: SUNION Key [...] Each argument to SUNION indicates a set that can be merged into a larger set. It is important to notice that overlapping members are listed once. To see this work, let's first create an ancient language set: ancient languages Greek SADD ancient languages Latin Smembers ancient languages can create a combination of languages and ancient languages that can see them all at once: SUNION language ancient languages we get the following responses: 1) Greek 2) Spanish 3) Chinese 5) Latin, if we pass a non-existent key to SUNION, it considers that key as an empty set (a set with nothing). Redis, a hash, is a data structure that maps string keys with a pair of field values. Therefore, hashes are useful for representing objects. This key is the name of the hash, and the value represents the sequence of field name field value items. We are Object as follows: The computer name MacBook Pro year 2015 disk 512 RAM 16 object's properties are defined by the name of the object, the name of the computer, and then the sequence of property names and property values. Because Redis is about sequential strings, you should be careful when creating string objects that correctly define objects using appropriate string sequencing. Use string-like commands to manipulate hashes. The hash data HSET write and read HSET command HSET set the field of the hash to a value. If no key exists, a new key is created to store the hash. If the field already exists in the hash, it is overwritten. HSET key field value, as you create the computer hash: HSET computer name MacBook Pro // 1 HSET computer year 2015 // 1 HSET computer disk 512 // 1 HSET computer RAM 16 // Reply to 1 HSET command, Redis replies with the following integer: 1 field is a new field in hash and value. The 0 field already exists in the hash and the value is updated. HSET Computer Year 2018 // 0 HGET HGET returns a value related to the hash field: Let's see if the HGET key field is getting 2018 as the value of the year instead of 2015: HGET Computer Year Redis replies to 2018. It makes good. A quick way to get all the fields with their own values from the HGETALL hash is to use HGETALL: LET'S TEST IT: HGETALL COMPUTER REPLY: 1) NAME2) MACBOOK PRO 3) YEAR 4) 2018 5) DISK 6) 512 7) RAM 8) REPLY TO BLANK LIST WHEN 16 HGETALL PROVIDED KEY ARGUMENTS DO NOT EXIST. HMSET HMSET key field value [field value ...] let's create a tablet hash with it: HMSET tablet name iPad year 2016 disk 64 ram 4 HMSET return confirmed that we let you know that the tablet hash was made successful. HMGET How do we get just two fields? Use HMGET to specify which field of hash you want to get the value from: HMGET key field [field ...] you can get the disk and ram field of the tablet hash: HMGET tablet disk ram effectively we get the value of disk and ram in reply:1) 642) 4This is almost the point of using hash in redis. You can browse and try the full list of hash commands. The sorted set introduced in Redis 1.2 is essentially a set of sorted sets. However, although the members of the set are not aligned (Redis can return elements in order from any call in the set), each member of the sorted set is linked to a floating point value called a score used by Redis to determine the order of the sorted set members. Because all elements of an ordered set are mapped to values, there is also an architecture similar to hashes. In Redis, the sorted set A hybrid of set and hash. How is the order of members of the sorted set determined? As stated in the Redis document: If A and B are two members with different scores, if the A.score is b.score > if A> B and B have exactly the same score, there is A> B if the A string is more than the B string. The A and B strings can't be the same because the sorted set has only its own elements. Some of the commands you use to interact with an ordered set are similar to the commands you used with a set that replaces S in the set and replaces it with Z. For example, SADD => ZADD. However, both have their own commands. Let's check it out. Zadd using ZADD adds all specified members with the specified score to the sorted set: ZADD key [NX| As with the XX] [CH] [INR] score member Score Member [Score Member ...] set, if no key is found, a new sort set is created in which only the specified members are created. If you have a key but do not have an sorted set, an error is returned. Starting with Redis 3.0.2, ZADD provides optional arguments to control insertion: XX: Update only members that already exist. Do not add members. NX: Do not update existing members already. Always add a new member. CH: Modifies the return value from the number of new members added to the total number of members changed (CH is a changed abbreviation). The changed member is a new member that has been added, and a member whose score has been updated already exists. Therefore, it is not calculated that the member assigned to the command line has the same score as in the past. INCR: If this option is specified, ZADD will work like ZINCRBY. In this mode, you can specify only one pair of score members. There are these optional arguments and it's good to know what they're doing, but for this introduction we'll focus on adding members without using either of them, but we're free to explore! In future posts, we'll be back in more complex use cases! To save help desk support tickets, we're creating a sorted set. Support tickets are unique but need to be sorted, but they need to be sorted, so this data structure is a great choice: ZADD Ticket 100 HELP204 // 1 ZADD Ticket 90 HELP004 // 1 ZADD Ticket 180 HELP330 // 1 ZADD returns the number of new elements added. In the command above, we used the location of the ticket in the queue as the table value, followed by the ticket number (all fiction). RANGE We want to see what our sorted set looks like now. Used a set to list unalted members using the S member. Sorted sets use commands that are more coordinated with the list, which is a command that shows various elements. RANGE returns the specified range of members in the sorted set: It behaves much like LRANGE in the STOP ZRANGE Key Start [WITHSCORES] list. It can be used to obtain a subset of an ordered set. You can use 0 -1 to get the full sort set. Re: ZRANGE Tickets 0 -1 Redis Reply: 1) HELP004 2) HELP204 3) HELP330 We can pass the RANGE WITHSCORE argument and also take over to include each member's score: RANGE ticket 0 -0 1 WITHSCORE REPLY: 1) HELP004 2) 90 3) HELP204 4) 100 5) HELP330 6) YOU CAN SEE HOW 180 MEMBERS AND SCORES ARE LISTED IN ORDER, NOT SIDE BY SIDE. As we can see, members are saved on tickets in ascending order according to their scores. When you use Redis as a session store, the most relevant use of Redis in the authentication and authorization workflow of a Web application is to role as a session store. Recognized by Amazon Web Services, Redis' in-memory architecture is a popular choice for storing and managing session data for Internet-scale applications by providing developers with high availability and persistence. Lightning-fast performance provides the very low latency, optimal scaling, and resiliency required to manage session data such as user profiles, user settings, session status, and credential management. Roshan Kumar of Redis Labs explains in his Cache vs. Session Store article that session-oriented web applications start sessions when users sign in. The session is activated until the user logs out or the session time is out. During the session lifecycle, the Web application stores all session-related data in basic memory (RAM) or session storage where no data is lost when the application goes down. This session store can be implemented using Redis, and even though it is memory storage, transaction logs can be created sequentially on disk to persist data. Source: Redis Labs: Cache vs. Session Store Roshan explains that session storage relies on in-memory databases to read and write data. Session storage data is not temporary and is the only source of truth when the session is live. For this reason, session storage must meet the data durability requirements of a true database. Bottom line Redis is a powerful, agile, and flexible database that can speed up your architecture. It has a lot to offer, including caching, data replication, pub/sub-messaging systems, session storage, and more. Redis has a number of clients that cover all popular programming languages. I hope you try it whenever there is a use case that fits that value offer. About Auth0 Auth0 provides a platform for authentication, approval, and secure access for applications, devices, and users. Security and application teams leverage Auth0's simplicity, scalability, and expertise to get identities up and down for everyone. Protecting billions of monthly sign-in transactions, Auth0 protects identities and enables global enterprises to deliver reliable, superior digital experiences to customers around the world. For more information, see twitter or @auth0 twitter. Twitter.