MongoDBSpain.com
Shell Cheat Sheet for mongoDB 2.4

César Trigo Esteban
info@mongodbspain.com

Command Line

mongo | Start the shell from the command line

Command line options:
--hostname localhost | hostname to connect
--port 27017 | connect to port 27017 (default)
-u foo | username foo
-p bar | password bar
--athenticationDatabase arg | database to authenticate

---

mongoimport | import a data from a file into MongoDB
> mongoimport --db <db> -c <coll> --file <filename> --type <type>
mongodump | Dumps contents of a db to a file
> mongodump --db <db> -c <coll>
mongorestore | restore from a dump to MongoDB
> mongorestore --db <db> -c <coll> <bson file>

Basic Shell commands

help| get help for the context you are in
exit | exit the shell

---

use <database> | select and use database
db | show selected database
show dbs | show databases on server
show collections (show tables) | show collections in current db
show users | show users in current database

Collection commands

help() | show a list of help commands for a collection

---

copyTo(name) | copy a collection to a new collection name
count() | get number of documents in a collection
drop() | remove the collection from the database
mapReduce() | performs map-reduce data aggregate
renameCollection(name) | Rename a collection
stats() | get stats about the collection

Cursors (db.collection.find().)

it | iterate on a cursor
count() | return a count of the documents on a cursor
explain() | get query execution plan for a cursor
hasNext() | true if cursor has more docs and can be iterated
hint({age: 1}) | force to use an specific index for a query
limit(5) | limits the size of the cursor result set
next() | return the next document in a cursor
skip() | skip through some documents and the return cursor
sort({age:-1}) | return results ordered ascending/descending
toArray() | return an array of all documents in the cursor

---

pretty() | pretty print the returned documents

DB Commands (db.)

help() | show a list of help commands for a dbIndexing

---

copyDatabase() | copies a db to another db
> db.copyDatabase('records', 'archive_records')
dropDatabase() | remove the current db
getLastError() | get status of last error
hostInfo() | getinfo about the host system
serverStatus() | get an overview of server status
shutdownServer() | shutdown current server
stats() | get stats on the current db selected
version() | get the current version of the server

Authentication (db.)

addUser() | add a user to system.users or admin collection
> db.addUser({user: "username", pwd: "password",roles: ["readWrite"] })
changeUserPassword() | change an existing users password
> db.changeUserPassword("reporting", "SOhSS3")
removeUser() | remove a user from a database
> db.changeUserPassword("reporting")

---

auth() | authenticates a user to a database
> db.auth("reporting", "SOhSS3")
logout() | logout from a database

Top & Stats System Commands

./mongotop | Shows time spent per operation per collection
./mongostat | Shows snapshot on the MongoDB System

Query commands (db.collection.)

Finding documents:
find().pretty()| Finds all documents using nice formatting
find({}, {name:true, _id:false}) | retrieve only *name* field
findOne() | Finds one arbitrary document
findOne({name:'abc'}) | Finds one document by attribute

Inserting document:
insert({name:'a'}) | Insert new document in the collection

Removing document:
remove() |Remove all collection documents
remove({name:'a'}) |Remove by criteria

Updating documents:
update({name:'a'}, {age:25}) | replaces the whole document
update({name:'a'}, {$set:{age:25}}) | change certain attribute
update({name:'b'}, {$unset:{age:1}}) | unset attribute
findAndModify({query:{...}, sort:{...},update:{...}}) | Automatically find and update

Query operators ($)

Comparison:
Ex: db.<collection>.find({ qty: { $gt: 20 } })
$gt| matches values greater than the value
$in | matches values supplied in an array
$gte | matches values greater than or equal the value
$lte | matches values less than or equal the value
$ne | matches all values that are not equal to given
$lt | matches values less than the value
$nin | matches values that do not exist in an array

Logical:
Ex: db.<coll>.find({ price:9, $or: [{ qty: { $lt:20 } },{ sale: true}]})
$or | joins query clauses with a logical OR
$and | joins query clauses with a logical AND
$not | returns documents that do not match
$nor | joins query clauses with a logical NOR

Evaluation:
$exists | matches documents that have a field
> db.inventory.find({ qty: { $exists: true, $nin: [5, 15] } })
$type | matches a field if it is of a given BSON type
> db.inventory.find({ price: { $type : 1 } })

Evaluation:
$mod | perform a modulo on a field and select if 0
> db.inventory.find({ qty: { $mod: [4, 0] } })
$regex | matches a regex expression on a field
> db.collection.find({ field : /acme.*corp/i })
$where | matches against a JavaScript expression
> db.myCollection.find({ $where: "this.credits - this.debits < 0" })

Geospatial:
$geoWithin | matches within a bounding GeoJSON geometry
> db.coll.find({ loc : { $geoWithin : { $geometry : { type : "Polygon" , coordinates : [[[0,0] , [3,6] ,[6,1] , [0,0]]] } } })
$geoIntersects | matches all docs that intersect with a GeoJSON object
> db.coll.find({ loc : { $geoIntersects: { $geometry : { type : "Polygon" , coordinates : [[[0,0] , [3,6] ,[6,1] , [0,0]]] } } })
$near | matches near a geospatial point
> db.place.find({ loc :{ $near : [40 , 5] ,$maxDistance : 10 } })
> db.place.find({ loc : { $near :{ $geometry :{ type : "Point" , coordinates: [40 , 5] } },$maxDistance : 500} })
$nearSphere | matches near a point on a sphere
> db.place.find({ loc :{ $nearSphere : [40 , 5] ,$maxDistance : 10 } })
> db.place.find({ loc : { $nearSphere :{ $geometry :{ type : "Point" , coordinates: [40 , 5] } },$maxDistance : 500} })
Geospatial Operators:
$geometry | specifies a GeoJSON for a geospatial query
$maxDistance | specifies the distance for $near & $nearSphere
$center | return documents within the circle center + radius
> db.place.find({ loc: { $geoWithin :{ $center : [[-74, 40], 10] } } })
$centerSphere | return documents within spherical geometry
> db.pl.find({loc:{$geoWithin :{ $centerSphere: [[88 , 30] , 10 /3959]} } })
$box | returns all documents that are within the box
> db.place.find({ loc: { $geoWithin : { $box :[[0 , 0] , [100 , 100]] } } })
$polygon | returns all documents that are within the polygon
> db.place.find({loc :{ $geoWithin :{ $polygon : [[0,0] ,[3,6] ,[6,0]] } } })
$uniqueDocs:true | returns a document only once

Array:
$all | matches arrays that contain all elements given
> db.inventory.find({ tags: { $all: ["appliance", "school", "book"] } })
$elemMatch | matches multiple conditions in array
> db.collection.find({array: {$elemMatch: {value1: 1, value2: { $gt: 1 } } } })
$size | matches if the array is of specified size
> db.collection.find({ field: { $size: 1 } })

Other:
$hint | force to use an specific index

Update operators ($)

Field operators:
$inc | Increment a value by a specified amount
$rename | rename a field
$setOnInsert | set a value only if inserting
$set | set the value of a field on an existing document
$unset | remove the field from an existing document

Array operators:
$ | update the first element in an array that matches
> db.coll.update({ _id: 1, grades: 80 }, { $set: { "grades.$" : 82 } })
> db.coll.update({ _id: 4,"grades.stack": 85 }, {$set:{"grades.$.std":6}})

Ex: db.coll.update({ name: "joe" }, { $push: { scores: 89 } })
$addToSet | add element to array if it doesn't exist
$push | adds an item to an array
$pop | update the first element in an array that matches
$pull | remove items which match a query statement
$pullAll | remove multiple values from an array

Array modifiers:
$each | modify $push and $addToSet to add many
> db.coll.update({ name: "joe" }, { $push: { scores: { $each: [90,85] } } })
$slice | modify $push to limit size of updated array
> db.coll.update({ _id: 2 }, {$push: { grades: {$each:[80,78], $slice: -5} } })
$sort | modify $push to reorder documents in array
> db.coll.update({ _id: 2 }, {$push: { grades: 81 , $sort: { grades: 1 } } })

Bitwise:
$bit | performs a bitwise update of a field
> db.coll.update({field:NumberInt(1) }, {$bit: {field:{and:NumberInt(5)}}})

Isolation:
$isolated | isolates a write operation for multiple documents
> db.coll.update({field1:1,$isolated:1}, {$inc:{field2 : 1 }} , {multi: true})

Indexes

Indexing - db.collection.
ensureIndex({a:1}) | Cretes an ascending Index
dropIndex({'a':1}) | Removes a index from a collection
getIndexes() | Get indexes details of a collections
reIndex() | Rebuild all indexes on a collections
compact() | Defragment a collection and rebuild indexes

Properties- db.collection.<IndexOp>({...},{<option>})
expireAfterSeconds:300 | delete docs after set time
unique:true | only unique data
sparse:true | only documents with the index field

Options - db.collection.<IndexOp>({...},{<option>})
background:true | create index in the background
dropDups:true | Drop duplicates on unique index creation

Info - db.collection.
totalIndexSize() | get index size

Projection (db.)

$ | project the first element in an array that matches
$elemMatch | project only the first element match
$slice | limit number of elements projected from array

Durability of writes

w | Tells the driver to wait for the write to be acknowledged. Ensures no indexes are violated. Neverless the data can still be lost as it is not necessarilly already persisted to disc.

---

j | Stands the journal-mode. Tells the driver to wait until the journal has been commited to disk. Once this has happened it is quite sure that the write be persistent unless there are any disc-failures.

Combinations
w=0 j=0 | Fire and forget
w=1 j=0 | Waits for an acknowledgement that the write was recived and no indexes have been violated. Data can be lost.
w=1 j=1 | Most safe configuration by waiting for the write to the journal to be completed.
w=0 j=1 | Wait for the for the journal. Indexes could be violated

MongoDBSpain.com
Shell Cheat Sheet for mongoDB 2.4

César Trigo Esteban
info@mongodbspain.com

Aggregation Framework ($)

Pipeline Operators:
$project | reshapes a document stream
$match | match documents against a query
$limit | restrict the number of documents returned
$skip | skip over some documents and return the rest
$unwind | open elements of an array into documents
$group | group on a field and aggregate values
$sort | sort on a specified field
$geoNear | get documents near a geospatial point

Group Operators:
$addToSet | return a unique array of values for group
$first | return the first value in a group
$last | return the last value in a group
$max | return the highest value in a group
$min | return the lowest value in a group
$avg | return an average of all values in a group
$push | return an array of values for a grouped field
$sum | return the sum of all values in a group

Boolean Operators:
$and | returns true when all values in array are true
$or | returns true when any value in its array are true
$not | returns boolean value that is opposite of input

Comparison Operators:
$cmp | return the result of a compare as an integer
$eq | return true if two values are equal
$gt | return true if first value greater than 2nd
$gte | return true if first value greater or equal to 2nd
$lt | return true if first value less than 2nd
$lte | return true if first value less than or equal 2nd
$ne | return true if two values are not equal

Arithmetic Operators:
$add | return the sum of an array of numbers
$divide | return the result of dividing two numbers
$mod | return the modulo of dividing two numbers
$multiply | return the product of an array of numbers
$subtract | return the result of subtracting 2 numbers

String Operators:
$concat | concatenate two strings
$strcasecmp | return an int reflecting a comparison
$substr | return a portion of a string
$toLower | convert a string to lowercase
$toUpper | convert a string to uppercase

Date Operators:
$dayOfYear | return an int between 1 and 366
$dayOfMonth | return an int between 1 and 31
$dayOfWeek | return an int between 1 and 7
$year | return the full year from a date
$month | return an int between 1 and 12
$week | return an int between 0 and 53
$hour | return an int between 0 and 23
$minute | return an int between 0 and 59
$second | return an int between 0 and 60
$millisecond | return millisecond portion of a date

Conditional Expresions:
$cond | ternary style operator, takes 3 expressions
$ifNull | eval 1st expression, if null eval 2nd, return

Example:

```
> db.orders.aggregate([
            { $match: { status: "A" } },
            {
                $group: {
                    _id: "$cust_id",
                    total: { $sum: "$amount" }
                }
            },
            { $sort: { total: -1 } }
          ])
```

Replication (rs.)

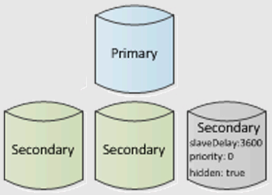
help() | get basic help for replica set functionsSharding
add() | adds a member to a replica set
addArb() | adds an arbiter to a replica set
conf() | returns the replica set config document
freeze() | prevents a member from becoming primary
initiate() | initializes a new replica set
reconfig() | reconfigure a replica set with a new config
remove() | remove a member from a replica set
slaveOk() | allow reads to happen on a seconary
status() | return a document with status of replica set
stepDown() | force primary to step down
syncFrom() | specify the member to sync from

Read preferences:
primary | read only from the primary in a replica set
primaryPreferred | prefer primary but can read from secondary
secondary | read only from a secondary in a replica set
secondaryPreferred | prefer a secondary, read from primary last
nearest | read from the nearest member in a replica set

Replica set nodes:
Regular | allowed to vote | Can become Primary
Most typical kind of node. Acts as primary o secondary
Arbiter | allowed to vote | Can't become Primary
Only for voting purposes.
Delayed | allowed to vote | Can't become Primary
For disaster recovery. Data stored is a few hours delayed
Hidden | not allowed to vote | Can't become Primary
For analytics in the replica set

Replica Set properties (members[n].):
priority | higher values to make a member more eligible to become primary, and lower values to make the member less eligible to become primary
hidden | hide this member
slaveDelay | number of seconds of lag "behind" the primary
arbiterOnly | Identifies an arbiter
buildIndexes | builds indexes on this member
votes | number of votes a server will cast
chainingAllowed | allows secondary members to replicate from other secondary members

Replica Set setup:

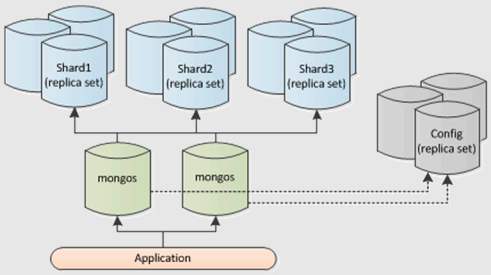


Create data directories
> mkdir -p rs0-0 rs0-1 rs0-2 rs0-hddb
Start mongod instances
> mongod --port 27017 --dbpath rs0-0 --repSet rs0
> mongod --port 27018 --dbpath rs0-1 --repSet rs0
> mongod --port 27019 --dbpath rs0-2 --repSet rs0
> mongod --port 27020 --dbpath rs0-hddb --repSet rs0
* Optionally use --smallfiles --oplogSize 128 for dev environments
Connect to one mongod instance
> mongod --port 27017
Create replica set config and initiate
> rsconfig = { _id:"rs0",members:[{ _id : 0, host : "<hostname>:27017" }] }
> rs.initiate(rsconfig)
Add more members to the RS
> rs.add("<hostname>:27018")
> rs.add("<hostname>:27019")
> rs.add("<hostname>:27020")
Configure member[3] hddb as a hidden & delayed (1h) member
> cfg = rs.conf() //to configured a hidden & delayed node
> cfg.members[3].priority = 0
> cfg.members[3].hidden = true
> cfg.members[3].slaveDelay = 3600
> rs.reconfig(cfg)

Sharding (sh.)

help() | returns help text for the sh methods
addShard() | add a shard to the cluster
addShardTag() | associate a shard with a tag
addTagRange() | associate range of shard keys with tag
disableBalancing() | disable balancing on a collection
enableBalancing() | re-enable balancing on a collection
enableSharding() | enables sharding on a database
getBalancerHost() | get the mongos doing balancing
getBalancerState() | true if the balancer is enabled
isBalancerRunning() | true if the balancer is migrating
moveChunk() | migrates a chunk in a sharded cluster
removeShardTag() | disassociate a shard with a tag
setBalancerState() | enable or disable the balancer
shardCollection() | enables sharding for a collection
splitAt() | divide a chunk in 2 based on shard key value
splitFind() | divide a chunk in half based on a query
startBalancer() | enable balancer and wait until started
status() | reports on the status of a sharded cluster
stopBalancer() | stop balancer and wait until stoppedIndex

Sharded Cluster Deploy:



Config server
> mkdir /data/configdb
Start mongod instance for config server on default port
> mongod --configsvr --dbpath /data/configdb --port 27019
Start mongos instances
> mongos --configdb cfg0.example.net:27019,cfg1.example.net:27019, cfg2.example.net:27019
Connect to mongos instance
> mongo --host mongos0.example.net --port 27017
Add Shards to the Cluster
> sh.addShard("rs1/db0.example.net:27017,db1.example.net:27017, db2.example.net:27017")
Enable Sharding for a Database
> sh.enableSharding("<database>")
Enable Sharding for a Collection
> sh.shardCollection("<database>.<collection>", shard-key-pattern)

Map Reduce (db.coll.mapReduce())

Map Reduce Commands:
db.runCommand({mapReduce:"<collection>", ...})
run map-reduce aggregation operations over a collection
db.<collection>.mapReduce()
Wrapper around the mapReduce command

db.runCommand() parameters for Map Reduce:
mapReduce | name of the collection
map | a JavaScript function that associates or "maps" a value with a key and emits the key and value pair.
reduce | a JavaScript function that "reduces" to a single object all the values associated with a particular key.
out | location of the result of the map-reduce operation
query | specifies the selection criteria using query operators
sort | sorts the input documents
limit | maximum number of documents to return
finalize | a JavaScript function that follows the reduce method and modifies the output
scope | specifies global variables that are accessible in the map, reduce and finalize functions
jsMode | convert intermediate data into BSON between the execution of the map and reduce functions. Default false.
verbose | timing information in the result information

Example:

```
> var mapFunction = function() { emit( this.customerId, this.amount ) };
> var reduceFunction = function(key, values) { return Array.sum(values) };
> db.runCommand(
        {
          mapReduce: 'sales',
          map: mapFunction,
          reduce: reduceFunction,
          out: { merge: 'map_reduce_results', db: 'test' },
          query: { orderDate: { $gt: new Date('01/01/2014') } }
        }
      )
```

Data Modeling

References:
Store the relationships between data by including links or references from one document to another.

Manual references
Save the _id field of one document in another document as a reference.

DBRefs (DBRef(<collection>, <ObjectId>, <db>))
References from one document to another using the value of the first document's _id field collection, and optional database name. The application must perform additional queries to return the referenced documents.

Embedded Data:
Capture relationships between data by storing related data in a single document structure

```
{
    _id : <ObjectId1>,
    username: "username1",
    address:{
        street: "Street name",
        number: 24,
        city: "Madrid",
    },
    access:{
        group: "dev", level: 5
    }
}
```