MongoDB Command Cheatsheet

Using the MongoDB Shell

The Mongo Shell is a command line utility. You can enter `mongo` (or `mongosh`) in your CMD/Terminal/PowerShell and it will open the shell (you may have to enter connection info for more secured environments). From there you can run MongoDB commands.

On your local machine, you can download the Shell by installing it from here. It is installed by default on the MongoDB server.

Connecting to MongoDB

Description	Command
Connect to a MongoDB instance	`mongo`
Connect to a specific database	`mongo <database_name>`
Connect to a remote MongoDB server	`mongo <hostname>:<port>/<database_name>`
Connect via mongosh	`mongosh`
Connect to specific host and port via mongosh	`mongosh --host <host> --port <port> --authenticationDatabase admin -u <user> -p <pwd>`
Connect via mongosh with connection string	`mongosh "mongodb://<user>:<password>@192.168.1.1:27017"`
Connect to MongoDB Atlas via mongosh	`mongosh "mongodb+srv://cluster-name.abcde.mongodb.net/<dbname>" --apiVersion 1 --username <username>`

Database Operations

Description	Command
Show all databases	`show dbs`
Use a specific database	`use <database_name>`
Get the current database	`db`
Drop the current database	`db.dropDatabase()`

Collection Operations

Description	Command
Show all collections in the current database	`show collections`
Create a collection	`db.createCollection("<collection_name>")`

Drop a collection	`db.<collection_name>.drop()`

CRUD Operations

Create

Description	Command
Insert a document	`db.coll.insertOne({name: "Max"})`
Insert multiple documents (ordered)	`db.coll.insertMany([{name: "Max"}, {name: "Alex"}])`
Insert multiple documents (unordered)	`db.coll.insertMany([{name: "Max"}, {name: "Alex"}], {ordered: false})`
Insert document with current date	`db.coll.insertOne({date: ISODate()})`
Insert document with write concern	`db.coll.insertOne({name: "Max"}, {"writeConcern": {"w": "majority", "wtimeout": 5000}})`

Read

Description	Command
Find one document	`db.coll.findOne()`
Find all documents	`db.coll.find()`
Find all documents (pretty print)	`db.coll.find().pretty()`
Find documents with a query	`db.coll.find({name: "Max", age: 32})`
Find documents with date	`db.coll.find({date: ISODate("2020-09-25T13:57:17.180Z")})`
Explain query execution stats	`db.coll.find({name: "Max", age: 32}).explain("executionStats")`
Get distinct values of a field	`db.coll.distinct("name")`
Count documents with a query	`db.coll.countDocuments({age: 32})`
Get estimated document count	`db.coll.estimatedDocumentCount()`

Update

Description	Command
Update a document (set fields)	`db.coll.updateOne({"_id": 1}, {$set: {"year": 2016, name: "Max"}})`
Update a document (unset fields)	`db.coll.updateOne({"_id": 1}, {$unset: {"year": 1}})`
Rename a field	`db.coll.updateOne({"_id": 1}, {$rename: {"year": "date"}})`
Increment a field	`db.coll.updateOne({"_id": 1}, {$inc: {"year": 5}})`

Multiply a field	`db.coll.updateOne({"_id": 1}, {$mul: {price: NumberDecimal("1.25"), qty: 2}})`
Set minimum value of a field	`db.coll.updateOne({"_id": 1}, {$min: {"imdb": 5}})`
Set maximum value of a field	`db.coll.updateOne({"_id": 1}, {$max: {"imdb": 8}})`
Set current date	`db.coll.updateOne({"_id": 1}, {$currentDate: {"lastModified": true}})`
Set current date as timestamp	`db.coll.updateOne({"_id": 1}, {$currentDate: {"lastModified": {$type: "timestamp"}}})`
Array updates (push)	`db.coll.updateOne({"_id": 1}, {$push: {"array": 1}})`
Array updates (pull)	`db.coll.updateOne({"_id": 1}, {$pull: {"array": 1}})`
Array updates (addToSet)	`db.coll.updateOne({"_id": 1}, {$addToSet: {"array": 2}})`
Array updates (pop last element)	`db.coll.updateOne({"_id": 1}, {$pop: {"array": 1}})`
Array updates (pop first element)	`db.coll.updateOne({"_id": 1}, {$pop: {"array": -1}})`
Array updates (pullAll)	`db.coll.updateOne({"_id": 1}, {$pullAll: {"array": [3, 4, 5]}})`
Array updates (push with each)	`db.coll.updateOne({"_id": 1}, {$push: {"scores": {$each: [90, 92]}}})`
Array updates (push with sort)	`db.coll.updateOne({"_id": 2}, {$push: {"scores": {$each: [40, 60], $sort: 1}}})`
Update array element	`db.coll.updateOne({"_id": 1, "grades": 80}, {$set: {"grades.$": 82}})`
Update all array elements	`db.coll.updateMany({}, {$inc: {"grades.$[]": 10}})`
Update array elements with filter	`db.coll.updateMany({}, {$set: {"grades.$[element]": 100}}, {multi: true, arrayFilters: [{"element": {$gte: 100}}]})`
Find and update	`db.coll.findOneAndUpdate({"name": "Max"}, {$inc: {"points": 5}}, {returnNewDocument: true})`
Upsert (update or insert)	`db.coll.updateOne({"_id": 1}, {$set: {item: "apple"}, $setOnInsert: {defaultQty: 100}}, {upsert: true})`
Replace a document	`db.coll.replaceOne({"name": "Max"}, {"firstname": "Maxime", "surname": "Beugnet"})`
Update with write concern	`db.coll.updateMany({}, {$set: {"x": 1}}, {"writeConcern": {"w": "majority", "wtimeout": 5000}})`

Delete

Description	Command

Delete one document	`db.coll.deleteOne({name: "Max"})`
Delete multiple documents	`db.coll.deleteMany({name: "Max"}, {"writeConcern": {"w": "majority", "wtimeout": 5000}})`
Delete all documents	`db.coll.deleteMany({})`
Find and delete	`db.coll.findOneAndDelete({"name": "Max"})`

Indexing

Description	Command
Create an index on a field	`db.<collection_name>.createIndex({<field>: 1})`
Create a unique index	`db.<collection_name>.createIndex({<field>: 1}, {unique: true})`
List all indexes on a collection	`db.<collection_name>.getIndexes()`
Drop an index	`db.<collection_name>.dropIndex("<index_name>")`
Hide an index	`db.coll.hideIndex("name_1")`
Unhide an index	`db.coll.unhideIndex("name_1")`

Aggregation

Description	Command
Aggregation framework	`db.<collection_name>.aggregate([ { $match: { <field>: <value> } }, { $group: { _id: "$<group_field>", total: { $sum: "$<sum_field>" } } }, { $sort: {total: -1} } ])`

Backup and Restore

Description	Command
Backup a database	`mongodump --db <database_name> --out <backup_directory>`
Restore a database	`mongorestore --db <database_name> <backup_directory>/<database_name>`

User Management

Description	Command
Create a new user	`db.createUser({ user: "<username>", pwd: "<password>", roles: [ { role: "<role>", db: "<database>" } ] })`
Show users	`show users`
Drop a user	`db.dropUser("<username>")`

Server Administration

Description	Command
Server status	`db.serverStatus()`
Database statistics	`db.stats()`
Collection statistics	`db.<collection_name>.stats()`
Current operations	`db.currentOp()`
Kill an operation	`db.killOp(<operation_id>)`
Lock the database	`db.fsyncLock()`
Unlock the database	`db.fsyncUnlock()`
Get collection names	`db.getCollectionNames()`
Get collection info	`db.getCollectionInfos()`
Print collection stats	`db.printCollectionStats()`
Replication info	`db.getReplicationInfo()`
Print replication info	`db.printReplicationInfo()`
Server info	`db.hello()`
Host info	`db.hostInfo()`
Shutdown server	`db.shutdownServer()`
Profiling status	`db.getProfilingStatus()`
Set profiling level	`db.setProfilingLevel(1, 200)`
Enable free monitoring	`db.enableFreeMonitoring()`
Disable free monitoring	`db.disableFreeMonitoring()`
Get free monitoring status	`db.getFreeMonitoringStatus()`

Handy Commands

Description	Command
Use admin database	`use admin`
Create root user	`db.createUser({"user": "root", "pwd": passwordPrompt(), "roles": ["root"]})`
Drop root user	`db.dropUser("root")`
Authenticate user	`db.auth( "user", passwordPrompt() )`
Switch to test database	`use test`
Get sibling database	`db.getSiblingDB("dbname")`
Get current operations	`db.currentOp()`

Kill operation	`db.killOp(123)`
Get collection stats	`db.printCollectionStats()`
Get server status	`db.serverStatus()`
Create a view	`db.createView("viewName", "sourceColl", [{$project:{department: 1}}])`

Change Streams

Description	Command
Watch for changes	`watchCursor = db.coll.watch([ { $match : {"operationType" : "insert" } } ])`
Iterate change stream	`while (!watchCursor.isExhausted()){ if (watchCursor.hasNext()){ print(tojson(watchCursor.next())); } }`

Replica Set

Description	Command
Replica set status	`rs.status()`
Initiate replica set	`rs.initiate({"_id": "RS1", members: [ { _id: 0, host: "mongodb1.net:27017" }, { _id: 1, host: "mongodb2.net:27017" }, { _id: 2, host: "mongodb3.net:27017" }]})`
Add a member	`rs.add("mongodb4.net:27017")`
Add an arbiter	`rs.addArb("mongodb5.net:27017")`
Remove a member	`rs.remove("mongodb1.net:27017")`
Get replica set config	`rs.conf()`
Replica set hello	`rs.hello()`
Print replication info	`rs.printReplicationInfo()`
Print secondary replication info	`rs.printSecondaryReplicationInfo()`
Reconfigure replica set	`rs.reconfig(config)`
Set read preference	`db.getMongo().setReadPref('secondaryPreferred')`
Step down primary	`rs.stepDown(20, 5)`

Sharded Cluster

Description	Command
Print sharding status	`db.printShardingStatus()`
Sharding status	`sh.status()`
Add a shard	`sh.addShard("rs1/mongodb1.example.net:27017")`
Shard a collection	`sh.shardCollection("mydb.coll", {zipcode: 1})`
Move a chunk	`sh.moveChunk("mydb.coll", { zipcode: "53187" }, "shard0019")`
Split a chunk at a key	`sh.splitAt("mydb.coll", {x: 70})`
Split a chunk by find query	`sh.splitFind("mydb.coll", {x: 70})`
Start balancer	`sh.startBalancer()`
Stop balancer	`sh.stopBalancer()`
Disable balancing	`sh.disableBalancing("mydb.coll")`
Enable balancing	`sh.enableBalancing("mydb.coll")`
Get balancer state	`sh.getBalancerState()`
Set balancer state	`sh.setBalancerState(true/false)`
Is balancer running	`sh.isBalancerRunning()`
Start auto merger	`sh.startAutoMerger()`
Stop auto merger	`sh.stopAutoMerger()`
Enable auto merger	`sh.enableAutoMerger()`
Disable auto merger	`sh.disableAutoMerger()`
Update zone key range	`sh.updateZoneKeyRange("mydb.coll", {state: "NY", zip: MinKey }, { state: "NY", zip: MaxKey }, "NY")`
Remove range from zone	`sh.removeRangeFromZone("mydb.coll", {state: "NY", zip: MinKey }, { state: "NY", zip: MaxKey })`
Add shard to zone	`sh.addShardToZone("shard0000", "NYC")`
Remove shard from zone	`sh.removeShardFromZone("shard0000", "NYC")`