MongoDB Shell Cheat Sheet

RED HAT DEVELOPERS

Starting the mongo shell

Mongo shell can be used to connect to local databases, or remote databases running on another server. If connecting to MongoDB locally, make sure it's running in another terminal window or background process (the `` `mongod` `` command).

Code	Description
`mongo`	Connect to a local MongoDB database

`mongo <hostname>:<port> <databaseName> -u <user> -p <password>`
e.g.:
`mongo cianclarke.com:27017/users -u cianclarke -p mYs3cretz`
Protip: The default port MongoDB runs on is 27017.
	Connect to a remote MongoDB server

Navigating around Mongo

Here are the most used commands, operations and queries.

`Help Methods and Commands`	Description
`show dbs`	Shows all databases available on this server
`use acmegrocery`	Switches to a database called acmegrocery. Creates this database if it doesn't already exist.
`show collections`	Show all collections in the current db (first `` `use <someDb>` ``)
`show users`	Show all users for the current DB
`show roles`	Show the roles defined for the current DB

Working with a collection

Now that you have selected a database and listed the collections inside, you can perform operations using the `` `db` `` variable.
Every collection has a property on the `` `db` `` variable - e.g. the `` `apples` `` collection is `` `db.apples` ``.

`JavaScript Database Operations`	Description
`db.apples.find()`	Finds all documents in a collection named "apples".

JavaScript Database Operations	Description
`db[“blood-oranges”].find()`	As above, finds all documents in a collection named “seville-oranges”. Collection names with characters that are reserved in JavaScipt (in this case, your - character) need to be queried this way.
`db.apples.find({type : “granny smith”})`	Finds all documents in a collection called “apples” with type matching “granny smith”.
`db.apples.find({}, { type : 1, price : 1 })`	You can also find all apple documents, but only return the fields we’re interested in - in this case, type and price, by setting them to 1 in the second parameter.
`db.apples.find().sort({ price : -1 })`	You can sort the results from a find() operation using the sort function. `-1` sorts descending, `1` sorts ascending.

Changing Groups of Documents

All of the operations so far have been queries which return many documents - called a cursor. You could use these cursors and iterate, but there’s an easier way.

`db.apples.remove({ bad : true})`	Removes all bad apples! Finds all apples with a property of “bad” set to true, and removes them.
`db.apples.update({ type : “granny smith”}, {$set : { price : 2.99 }})`	Updates the price of all granny smith apples.
`#don’t do this! db.apples.update({ type : “granny smith”}, { price : 2.99 })`	**Important!** Note the $set syntax - this doesn’t work as many would expect. Without putting the updated fields inside a $set clause, we replace the entire document.

Working with Individual Documents

All of these operations do something with many documents, but in the Mongo Shell there’s a really neat way of working with just one document too!

`db.apples.insert({ type : “granny smith”, “price” : 5.99 })`	Inserts a new document into the apples collection. If your apples collection doesn’t exist, it will get created.
`db.apples.findOne({ _id : ObjectId (“54324a5925859afb491a0000”) })`	Looking up a document by ID is special. We can’t just specify the ObjectID as a string - we need to cast it to an ObjectId.

If you haven’t already noticed, the mongo shell is a JavaScript REPL. This means you can use basic JavaScript commands to operate on documents!

`var greenest = db.apples.findOne ({ countryOfOrigin : “Ireland” })`
 	
`greenest.price = 10.99`
 	
`db.apples.save(greenest)`
	Finds the greenest apple of them all! Find just one apple document with a country of origin from Ireland.
	Note that .findOne() is different - it returns a **document**, where .find() returns a **cursor**. We can assign this document to a variable, make changes, then save it back to the collection - an update operation.

Working with Indexes

`db.apples.createIndex ( { countryOfOrigin: 1 } )`	Creates a new index on the countryOfOrigin field of the apples collection in ascending order.
`db.apples.getIndexes()`	List all indexes on the apples collection.

`db.apples.dropIndex ({ countryOfOrigin : 1})`
 	
`db.apples.dropIndexes()`
	Drops the countryOfOrigin index we just created on the apples collection.
	Drops all indexes in the apples collection. (except those on the _id field).

DANGER ZONE!

`db.apples.drop()`	Drops the entire apples collection.

`use acmegrocery`
`db.dropDatabase()`
	Drops the entire acmegrocery database

About the Author

Cian Clarke is a Software Engineer on the Red Hat Mobile Team. An early technologist, he founded his own web consultancy business at 16. Cian was a member of the original FeedHenry mobile team, since acquired by Red Hat.

At Red Hat, Cian builds functionality for the Mobile Application Platform, and also helps with solutions architecture and evangelism. In addition to his day job, he also regularly blogs about the intersection of all things mobile, microservices and Node.js. Cian is originally from Waterford, Ireland, and currently resides in Boston, Massachusetts.