MongoDB Reporting Tools

This section provides an overview of the reporting methods distributed with MongoDB. It also offers examples of the kinds of questions that each method is best suited to help you address.

**Free Monitoring**

MongoDB offers [free Cloud monitoring](https://docs.mongodb.com/manual/administration/free-monitoring/) for standalones or replica sets.

By default, you can enable/disable free monitoring during runtime using [db.enableFreeMonitoring()](https://docs.mongodb.com/manual/reference/method/db.enableFreeMonitoring/" \l "mongodb-method-db.enableFreeMonitoring) and [db.disableFreeMonitoring()](https://docs.mongodb.com/manual/reference/method/db.disableFreeMonitoring/" \l "mongodb-method-db.disableFreeMonitoring).

Free monitoring provides up to 24 hours of data. For more details, see [Free Monitoring](https://docs.mongodb.com/manual/administration/free-monitoring/).

**Utilities**

The MongoDB distribution includes a number of utilities that quickly return statistics about instances' performance and activity. Typically, these are most useful for diagnosing issues and assessing normal operation.

**mongostat**

[mongostat](https://docs.mongodb.com/database-tools/mongostat/#mongodb-binary-bin.mongostat) captures and returns the counts of database operations by type (e.g. insert, query, update, delete, etc.). These counts report on the load distribution on the server.

Use [mongostat](https://docs.mongodb.com/database-tools/mongostat/" \l "mongodb-binary-bin.mongostat) to understand the distribution of operation types and to inform capacity planning. See the [mongostat manual](https://docs.mongodb.com/manual/reference/program/mongostat/) for details.

**mongotop**

[mongotop](https://docs.mongodb.com/database-tools/mongotop/#mongodb-binary-bin.mongotop) tracks and reports the current read and write activity of a MongoDB instance, and reports these statistics on a per collection basis.

Use [mongotop](https://docs.mongodb.com/database-tools/mongotop/" \l "mongodb-binary-bin.mongotop) to check if your database activity and use match your expectations. See the [mongotop manual](https://docs.mongodb.com/manual/reference/program/mongotop/) for details.

**HTTP Console**

*Changed in version 3.6*: MongoDB 3.6 removes the deprecated HTTP interface and REST API to MongoDB.

**Commands**

MongoDB includes a number of commands that report on the state of the database.

These data may provide a finer level of granularity than the utilities discussed above. Consider using their output in scripts and programs to develop custom alerts, or to modify the behavior of your application in response to the activity of your instance. The [db.currentOp()](https://docs.mongodb.com/manual/reference/method/db.currentOp/" \l "mongodb-method-db.currentOp) method is another useful tool for identifying the database instance's in-progress operations.

**serverStatus**

The [serverStatus](https://docs.mongodb.com/manual/reference/command/serverStatus/" \l "mongodb-dbcommand-dbcmd.serverStatus) command, or [db.serverStatus()](https://docs.mongodb.com/manual/reference/method/db.serverStatus/" \l "mongodb-method-db.serverStatus) from the shell, returns a general overview of the status of the database, detailing disk usage, memory use, connection, journaling, and index access. The command returns quickly and does not impact MongoDB performance.

[serverStatus](https://docs.mongodb.com/manual/reference/command/serverStatus/#mongodb-dbcommand-dbcmd.serverStatus) outputs an account of the state of a MongoDB instance. This command is rarely run directly. In most cases, the data is more meaningful when aggregated, as one would see with monitoring tools including [MongoDB Cloud Manager](https://cloud.mongodb.com/?tck=docs_server) and [Ops Manager](https://www.mongodb.com/products/mongodb-enterprise-advanced?tck=docs_server). Nevertheless, all administrators should be familiar with the data provided by [serverStatus](https://docs.mongodb.com/manual/reference/command/serverStatus/" \l "mongodb-dbcommand-dbcmd.serverStatus).

**dbStats**

The [dbStats](https://docs.mongodb.com/manual/reference/command/dbStats/" \l "mongodb-dbcommand-dbcmd.dbStats) command, or [db.stats()](https://docs.mongodb.com/manual/reference/method/db.stats/" \l "mongodb-method-db.stats) from the shell, returns a document that addresses storage use and data volumes. The [dbStats](https://docs.mongodb.com/manual/reference/command/dbStats/" \l "mongodb-dbcommand-dbcmd.dbStats) reflect the amount of storage used, the quantity of data contained in the database, and object, collection, and index counters.

Use this data to monitor the state and storage capacity of a specific database. This output also allows you to compare use between databases and to determine the average [document](https://docs.mongodb.com/manual/reference/glossary/#std-term-document) size in a database.

**collStats**

The [collStats](https://docs.mongodb.com/manual/reference/command/collStats/" \l "mongodb-dbcommand-dbcmd.collStats) or [db.collection.stats()](https://docs.mongodb.com/manual/reference/method/db.collection.stats/" \l "mongodb-method-db.collection.stats) from the shell that provides statistics that resemble [dbStats](https://docs.mongodb.com/manual/reference/command/dbStats/" \l "mongodb-dbcommand-dbcmd.dbStats) on the collection level, including a count of the objects in the collection, the size of the collection, the amount of disk space used by the collection, and information about its indexes.

**replSetGetStatus**

The [replSetGetStatus](https://docs.mongodb.com/manual/reference/command/replSetGetStatus/" \l "mongodb-dbcommand-dbcmd.replSetGetStatus) command ([rs.status()](https://docs.mongodb.com/manual/reference/method/rs.status/" \l "mongodb-method-rs.status) from the shell) returns an overview of your replica set's status. The [replSetGetStatus](https://docs.mongodb.com/manual/reference/command/replSetGetStatus/) document details the state and configuration of the replica set and statistics about its members.

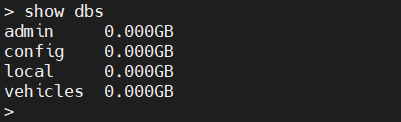
Use this data to ensure that replication is properly configured, and to check the connections between the current host and the other members of the replica set.’

**MongoDB show command**

Let us see how to view objects in a MongoDB database. You can get the existing databases, collections, roles, and users with the **show** command.

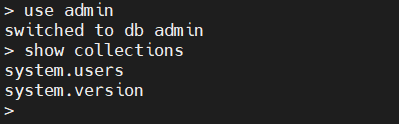
**View all databases**

show dbs

¨

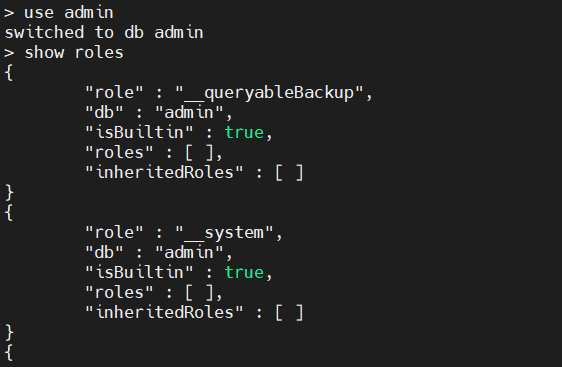
**View collections inside a database**

show collections / db.getCollectionNames()



View roles in a database

show roles



**View users in a database**

show users / db.getUsers()



**User management commands**

One of the most important administrative tasks is to manage permission for users. MongoDB provides this functionality using users and roles, and it has built-in roles for easy access controls.

You have to enable the “authentication” option in the MongoDB config file to use the access control feature. Add the following lines in mongod.conf file and restart the MongoDB service to reflect the changes.

**/etc/mongod.conf**

security:

authorization: "enabled"

### Creating a user

The **createUser** command allows us to create users. Let’s create a user for the vehicles database with only read and write permissions.

Syntax:

db.createUser(

{

user: <username>,

pwd: <passwordPrompt() / Clear Text Password>,

roles: [

{role: <role>, db: <database>}

]

}

)

Example:

use vehicles

db.createUser(

{

user: "barryvehicles",

pwd: passwordPrompt(),

roles: [

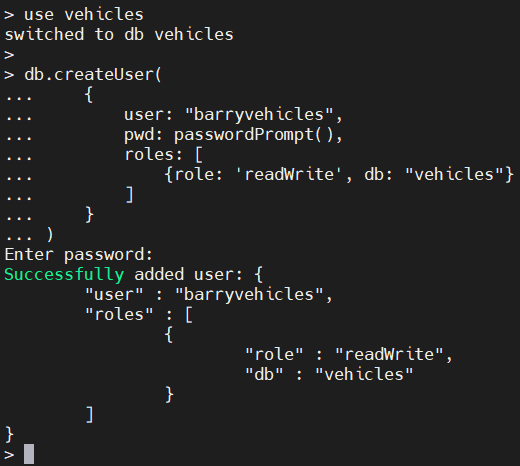
{role: 'readWrite', db: "vehicles"}

]

}

)

Result:

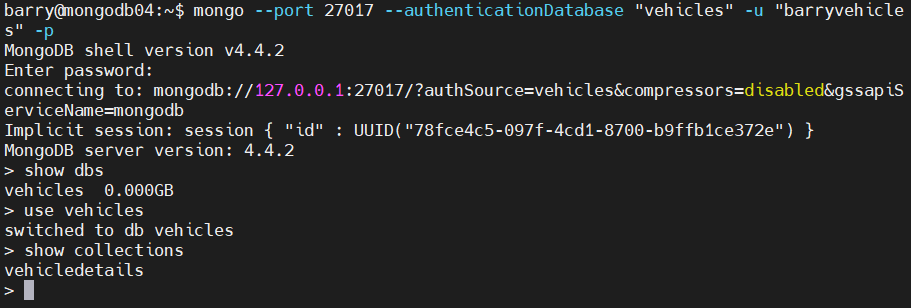


The **passwordPromt()** function will ask for the password when running the **createUser** command. The user is created in the “vehicles” database.

So, when we authenticate using this user, we must specify the “vehicles” database as the “authenticationDatabase”. A database user can be created in any database while defining permissions for other databases.

mongo --port 27017 --authenticationDatabase "vehicles" -u "barryvehicles" -p

Result:



**Updating user details**

We can update the details of the user using the **updateUser()** command. When updating user roles, we need to specify all the desired roles—because updateUser()will overwrite any existing rules.

In this example, we will update the “barryvehicles” user with a custom field and give read permission to the admin database. The “customData” section allows us to create any custom key pair. This has no effect on user roles; custom fields can be considered more of an informative section where we can add additional details for the user.

Syntax:

db.updateUser(

<"username">,

{

customData : { <custom fields> },

roles: [

{role: <role>, db: <database>}

]

}

)

Example:

db.updateUser(

"barryvehicles",

{

customData : { usertype: 'dbadmin' },

roles: [

{role: 'readWrite', db: "vehicles"},

{role: 'read', db: "admin"}

]

}

)

Result:



### Deleting a user

We can delete a user using the **dropUser()** command. In the following example, we will delete the user “barryvehicles”.

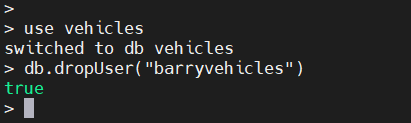
Syntax:

db.dropUser(<username>)

Example:

>db.dropUser("barryvehicles")

Result:



**Checking logs**

We have two methods for checking logs in MongoDB. We can:

* Check the mongod log file
* Use the **getLog()** command

**getLog()** returns the most recent logged events. This command will read the recent 1024 MongoDB log events in the RAM cache. In earlier versions of MongoDB, logs were returned in plaintext format. However, in MongoDB 4.4, the logs are formatted in [Extended JSON v2.0](https://docs.mongodb.com/manual/reference/mongodb-extended-json/).

Syntax:

db.adminCommand( { getLog: <value> } )

There are three possible values for the **getLog()** command. Those are

* **\*** returns the list of available values for getLog() command.
* **global** returns all the recent log entries.
* **startupWarnings** returns log entries that may contain errors or warnings since the start of the current process.

Example:

db.adminCommand( { getLog: "\*" } )

db.adminCommand( { getLog : "global" } )

Result:



**Database management commands**

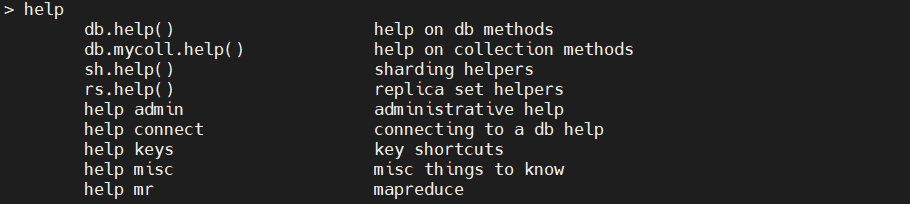
In this section, we will cover basic database management commands. These can help determine the server stats, collection stats, collection size, etc.

**Help**

**help** is an essential command in any administrator’s toolbox. The help command will give you a list of help options available in MongoDB.

Normal help:

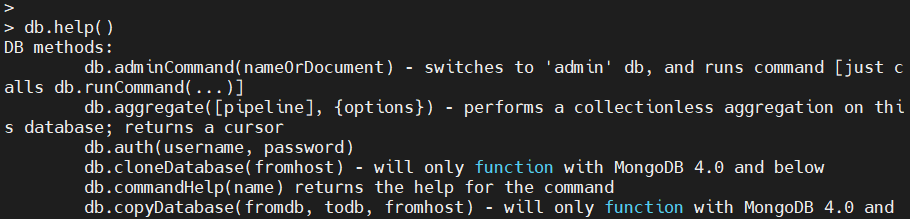
Result:



Here, you can see all the help options available in MongoDB. If you want to get all the help commands needed to work with databases, execute the **db.help()** command.

db.help()

Result:



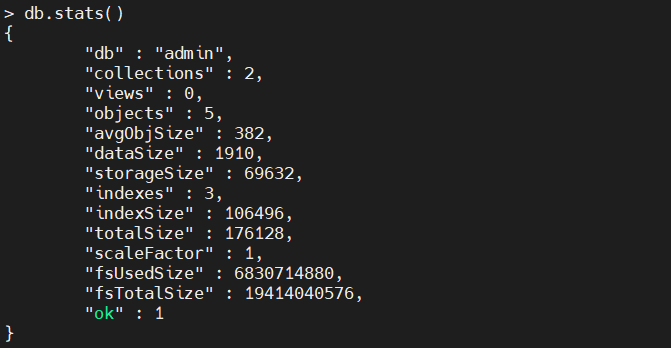
### Get database details

The **stats()** command provides statistics of the database. The information provided ranges from the number of collections and objects (documents), database sizes to indexes.

The **scaleFactor** reflects how data sizes are represented. The default scaleFactor is set to 1, which shows data in bytes. For example, we can change the scaleFactor to 1024 to show the sizes in kilobytes.

db.stats()

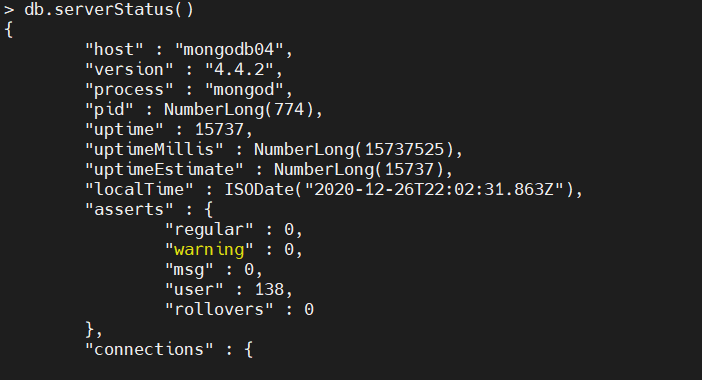
Result:



If you want to get the server details, use the **db.serverStats()** command.

db.serverStats()

Result:



To get a list of connection names, use **getCollectionNames()**command.

db.getCollectionNames()

Result:



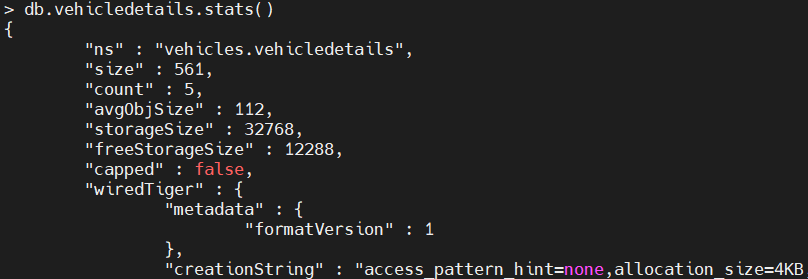
**Obtaining and returning collection details**

**Get collection statistics**

The **status()** function will provide a comprehensive overview of the collection.

db.vehicledetails.stats()

Result:

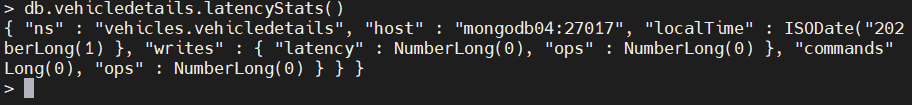


### Get collection latency

Use the **latencyStatus()** command to obtain the average latency of the read, write operations and the number of read and write operations.

db.vehicledetails.latencyStats()

Result:



### Get collection sizes

The following commands are used to find out the sizes of Collections in various ways:

* **dataSize()** shows the size of data within the collection.
* **storageSize()**indicates the total amount of storage allocated to the documents in the collection.
* **totalSize()** indicates the total size of the collection, including documents and the indexes.
* **totalIndexSize()** provides the indexed size of the collection.

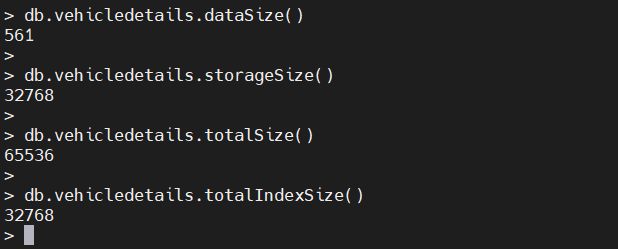
db.vehicledetails.dataSize()

db.vehicledetails.storageSize()

db.vehicledetails.totalSize()

db.vehicledetails.totalIndexSize()

Result:



Because the “vehicledetails” is a small collection, the **storageSize** is equal to **totalIndexSize** as the indexed data is equal to the raw data in the collection, and further compression is unfeasible.

**Renaming collections**

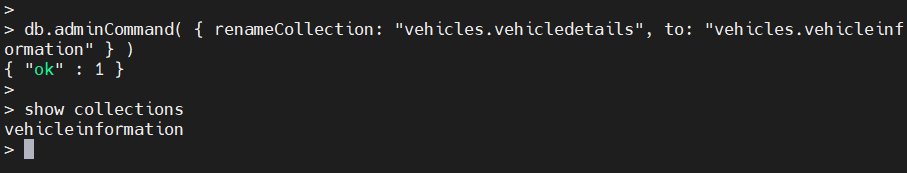
We can rename an existing collection with the **renameCollection** function. (This function is not compatible with [sharded collections](https://www.bmc.com/blogs/mongodb-sharding-explained/).)

When renaming a collection, we need to specify the source namespace and the destination namespace correctly. In MongoDB, namespace relates to the unique name in which we can identify database objects.

In the below example, we are renaming the “vehicledetails” collection in the vehicle database. We have defined the namespaces as <database>.<collectionname> to differentiate between the old and new collections.

db.adminCommand( { renameCollection: "vehicles.vehicledetails", to: "vehicles.vehicleinformation" } )

Result:



**Terminating the server**-If we want to completely terminate the MongoDB instance, we can use the built-in **shutdownServer()** command. **shutdownServer()** will clean up all the resources used by databases before terminating the MongoDB process.

The command must be issued against the admin database to be executed. We can achieve this by using the **getSiblingDB** function to indicate the admin database.

Syntax:

db.shutdownServer({

force: <boolean>,

timeoutSecs: <int>

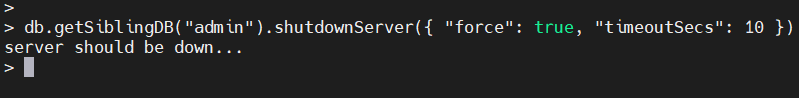
})

The **force** option forces a shutdown operation and interrupts any ongoing operations to terminate the MongoDB instance. The **timeoutSec** option can be used to set the time in seconds before a shutdown occurs. In an authenticated environment, the user must have the shutdown privilege to run this command.

In the following example, we will force a shutdown of the MongoDB instance in 10 seconds. The **getSiblingDB** function allows us to point the shutdown function to the admin database.

db.getSiblingDB("admin").shutdownServer({ "force": true, "timeoutSecs": 10 })

Result:



# **Delete Documents**

➤ Use the **Select your language** drop-down menu in the upper-right to set the language of the following examples.

This page uses the following [mongo](https://docs.mongodb.com/manual/reference/program/mongo/#mongodb-binary-bin.mongo) shell methods:

* [db.collection.deleteMany()](https://docs.mongodb.com/manual/reference/method/db.collection.deleteMany/#mongodb-method-db.collection.deleteMany)
* [db.collection.deleteOne()](https://docs.mongodb.com/manual/reference/method/db.collection.deleteOne/#mongodb-method-db.collection.deleteOne)

The examples on this page use the inventory collection. To populate the inventory collection, run the following:

|  |
| --- |
| db.inventory.insertMany( [ |
| { item: "journal", qty: 25, size: { h: 14, w: 21, uom: "cm" }, status: "A" }, |
| { item: "notebook", qty: 50, size: { h: 8.5, w: 11, uom: "in" }, status: "P" }, |
| { item: "paper", qty: 100, size: { h: 8.5, w: 11, uom: "in" }, status: "D" }, |
| { item: "planner", qty: 75, size: { h: 22.85, w: 30, uom: "cm" }, status: "D" }, |
| { item: "postcard", qty: 45, size: { h: 10, w: 15.25, uom: "cm" }, status: "A" }, |
| ] ); |

You can run the operation in the web shell below:

## Delete All Documents

To delete all documents from a collection, pass an empty [filter](https://docs.mongodb.com/manual/core/document/#std-label-document-query-filter) document {} to the [db.collection.deleteMany()](https://docs.mongodb.com/manual/reference/method/db.collection.deleteMany/" \l "mongodb-method-db.collection.deleteMany) method.

The following example deletes all documents from the inventory collection:

|  |
| --- |
| db.inventory.deleteMany({}) |

The method returns a document with the status of the operation. For more information and examples, see [deleteMany()](https://docs.mongodb.com/manual/reference/method/db.collection.deleteMany/" \l "mongodb-method-db.collection.deleteMany).

## Delete All Documents that Match a Condition

You can specify criteria, or filters, that identify the documents to delete. The [filters](https://docs.mongodb.com/manual/core/document/#std-label-document-query-filter) use the same syntax as read operations.

To specify equality conditions, use <field>:<value> expressions in the [query filter document](https://docs.mongodb.com/manual/core/document/#std-label-document-query-filter):

|  |
| --- |
| { <field1>: <value1>, ... } |

A [query filter document](https://docs.mongodb.com/manual/core/document/#std-label-document-query-filter) can use the [query operators](https://docs.mongodb.com/manual/reference/operator/query/#std-label-query-selectors) to specify conditions in the following form:

|  |
| --- |
| { <field1>: { <operator1>: <value1> }, ... } |

To delete all documents that match a deletion criteria, pass a [filter](https://docs.mongodb.com/manual/core/document/#std-label-document-query-filter) parameter to the [deleteMany()](https://docs.mongodb.com/manual/reference/method/db.collection.deleteMany/" \l "mongodb-method-db.collection.deleteMany) method.

The following example removes all documents from the inventory collection where the status field equals "A":

|  |
| --- |
| db.inventory.deleteMany({ status : "A" }) |

The method returns a document with the status of the operation. For more information and examples, see [deleteMany()](https://docs.mongodb.com/manual/reference/method/db.collection.deleteMany/" \l "mongodb-method-db.collection.deleteMany).

## Delete Only One Document that Matches a Condition

To delete at most a single document that matches a specified filter (even though multiple documents may match the specified filter) use the [db.collection.deleteOne()](https://docs.mongodb.com/manual/reference/method/db.collection.deleteOne/" \l "mongodb-method-db.collection.deleteOne) method.

The following example deletes the first document where status is "D":

|  |
| --- |
| db.inventory.deleteOne( { status: "D" } ) |

## Delete Behavior

### Indexes

Delete operations do not drop indexes, even if deleting all documents from a collection.

### Atomicity

All write operations in MongoDB are atomic on the level of a single document. For more information on MongoDB and atomicity, see [Atomicity and Transactions](https://docs.mongodb.com/manual/core/write-operations-atomicity/).

### Write Acknowledgement

With write concerns, you can specify the level of acknowledgement requested from MongoDB for write operations. For details, see [Write Concern](https://docs.mongodb.com/manual/reference/write-concern/).