**1) To start mongodb after installation:**

**Mongo**

**2) How to Exit From Mongodb**

**In the mongo shell, type:**

**Exit**

**3) To Get a List of All Commands of Mongodb:**

In mongo shell, use

**db.help()**

Now, in my case, (mongodb 2.6.12 is installed) the command shows the following list:

**db.adminCommand(nameOrDocument) - switches to 'admin' db, and runs command [ just calls db.runCommand(...) ]**

**db.auth(username, password)**

**db.cloneDatabase(fromhost)**

**db.commandHelp(name) returns the help for the command**

**db.copyDatabase(fromdb, todb, fromhost)**

**db.createCollection(name, { size : ..., capped : ..., max : ... } )**

**db.createUser(userDocument)**

**db.currentOp() displays currently executing operations in the db**

**db.dropDatabase()**

**db.eval(func, args) run code server-side**

**db.fsyncLock() flush data to disk and lock server for backups**

**db.fsyncUnlock() unlocks server following a db.fsyncLock()**

**db.getCollection(cname) same as db['cname'] or db.cname**

**db.getCollectionInfos()**

**db.getCollectionNames()**

**db.getLastError() - just returns the err msg string**

**db.getLastErrorObj() - return full status object**

**db.getMongo() get the server connection object**

**db.getMongo().setSlaveOk() allow queries on a replication slave server**

**db.getName()**

**db.getPrevError()**

**db.getProfilingLevel() - deprecated**

**db.getProfilingStatus() - returns if profiling is on and slow threshold**

**db.getReplicationInfo()**

**db.getSiblingDB(name) get the db at the same server as this one**

**db.getWriteConcern() - returns the write concern used for any operations on this db, inherited from server object if set**

**db.hostInfo() get details about the server's host**

**db.isMaster() check replica primary status**

**db.killOp(opid) kills the current operation in the db**

**db.listCommands() lists all the db commands**

**db.loadServerScripts() loads all the scripts in db.system.js**

**db.logout()**

**db.printCollectionStats()**

**db.printReplicationInfo()**

**db.printShardingStatus()**

**db.printSlaveReplicationInfo()**

**db.dropUser(username)**

**db.repairDatabase()**

**db.resetError()**

**db.runCommand(cmdObj) run a database command. if cmdObj is a string, turns it into { cmdObj : 1 }**

**db.serverStatus()**

**db.setProfilingLevel(level,<slowms>) 0=off 1=slow 2=all**

**db.setWriteConcern( <write concern doc> ) - sets the write concern for writes to the db**

**db.unsetWriteConcern( <write concern doc> ) - unsets the write concern for writes to the db**

**db.setVerboseShell(flag) display extra information in shell output**

**db.shutdownServer()**

**db.stats()**

**db.version() current version of the server**

And in mongodb 3.4,

It gives use the following:

**db.adminCommand(nameOrDocument) - switches to 'admin' db, and runs command [ just calls db.runCommand(...) ]**

**db.auth(username, password)**

**db.cloneDatabase(fromhost)**

**db.commandHelp(name) returns the help for the command**

**db.copyDatabase(fromdb, todb, fromhost)**

**db.createCollection(name, { size : ..., capped : ..., max : ... } )**

**db.createView(name, viewOn, [ { $operator: {...}}, ... ], { viewOptions } )**

**db.createUser(userDocument)**

**db.currentOp() displays currently executing operations in the db**

**db.dropDatabase()**

**db.eval() - deprecated**

**db.fsyncLock() flush data to disk and lock server for backups**

**db.fsyncUnlock() unlocks server following a db.fsyncLock()**

**db.getCollection(cname) same as db['cname'] or db.cname**

**db.getCollectionInfos([filter]) - returns a list that contains the names and options of the db's collections**

**db.getCollectionNames()**

**db.getLastError() - just returns the err msg string**

**db.getLastErrorObj() - return full status object**

**db.getLogComponents()**

**db.getMongo() get the server connection object**

**db.getMongo().setSlaveOk() allow queries on a replication slave server**

**db.getName()**

**db.getPrevError()**

**db.getProfilingLevel() - deprecated**

**db.getProfilingStatus() - returns if profiling is on and slow threshold**

**db.getReplicationInfo()**

**db.getSiblingDB(name) get the db at the same server as this one**

**db.getWriteConcern() - returns the write concern used for any operations on this db, inherited from server object if set**

**db.hostInfo() get details about the server's host**

**db.isMaster() check replica primary status**

**db.killOp(opid) kills the current operation in the db**

**db.listCommands() lists all the db commands**

**db.loadServerScripts() loads all the scripts in db.system.js**

**db.logout()**

**db.printCollectionStats()**

**db.printReplicationInfo()**

**db.printShardingStatus()**

**db.printSlaveReplicationInfo()**

**db.dropUser(username)**

**db.repairDatabase()**

**db.resetError()**

**db.runCommand(cmdObj) run a database command. if cmdObj is a string, turns it into { cmdObj : 1 }**

**db.serverStatus()**

**db.setLogLevel(level,<component>)**

**db.setProfilingLevel(level,<slowms>) 0=off 1=slow 2=all**

**db.setWriteConcern( <write concern doc> ) - sets the write concern for writes to the db**

**db.unsetWriteConcern( <write concern doc> ) - unsets the write concern for writes to the db**

**db.setVerboseShell(flag) display extra information in shell output**

**db.shutdownServer()**

**db.stats()**

**db.version() current version of the server**

**4) To Use mongodb In Admin Mode:**

use admin

1. **To Create A User As AdminOfAnyDatabase/dbOwner/root  
     
   To Create A User as AdminOfAnyDataBase:**

**db.createUser(  
  {  
    user: "admin",  
    pwd: "admin123",  
    roles: [ { role: "userAdminAnyDatabase", db: "admin" } ]  
  }  
)**

**To Create A User As Admin:**

**db.createUser(  
  {  
    user: "admin",  
    pwd: "admin123",  
    roles: [ { role: "root", db: "admin" } ]  
  }  
)**

Now, both provides a centralised authentication.

To provide the authentication for a particular database in MongoDB:

**db.createUser(  
  {  
    user: "dbadmin",  
    pwd: "dbadmin123",  
    roles: [ { role: "dbOwner", db: "serverApplicationDB" } ]  
  }  
)**

1. **Show All DataBases:  
     
   show dbs**

Use this in mongo shell.

1. **Drop A DataBase:**

Suppose, you want to drop a database named mydb

**use mydb**

**db.dropDatabase()**

1. **Create A Collection(Collection is table equivalent of mongodb)**

Suppose, you want to create just a collection in mydb database named as mytable:

**use mydb**

**db.createCollection(“mytable”)**

I won’t discuss about options which can be used during createCollection method.

**9) Insert Data In A Specific Collection  
  
db.mytable.insert( { name: "Sayak", Age: "23" } )**

And if write is successful, it would generate the message like following:

**WriteResult({ "nInserted" : 1 })**

**10) Drop a Specific Collection  
  
db.COLLECTION\_NAME.drop()**

If the collection name you want to drop is COLLECTION\_NAME

**11) Find Document(s) In a Specific Collection:**

Suppose, the data you want to see is from collection **COLLECTION\_NAME.** Simply do:

**db.COLLECTION\_NAME.find()**

**12) Limit The Number Of Documents Displayed As The Result Of The Specific Collection:**That is done by using limit() method additionally with the find method.  
  
d**b.COLLECTION\_NAME.find().limit(5)**

Suppose, I have a collection named validData2, and when I use

**db.validData2.find()**  
It generates the following result:  
  
**{ "\_id" : ObjectId("58f41133c4aa3b935cbd42d9"), "Name" : "Sayak" }**

**{ "\_id" : ObjectId("58f42510bdda0d1c0baa2a2f"), "Name" : "Sayan", "Age" : "24" }**

**{ "\_id" : ObjectId("58f4252cbdda0d1c0baa2a30"), "Name" : "Sayantan", "Age" : "22" }**

**{ "\_id" : ObjectId("58f42546bdda0d1c0baa2a31"), "Name" : "Suman", "Age" : "24" }**

**{ "\_id" : ObjectId("58f42578bdda0d1c0baa2a32"), "Name" : "Saptarshi", "Age" : "23" }**

Now, if I use,

**db.validData2.find().limit(4)**You will see only the first four results are displayed.

**{ "\_id" : ObjectId("58f41133c4aa3b935cbd42d9"), "Name" : "Sayak" }**

**{ "\_id" : ObjectId("58f42510bdda0d1c0baa2a2f"), "Name" : "Sayan", "Age" : "24" }**

**{ "\_id" : ObjectId("58f4252cbdda0d1c0baa2a30"), "Name" : "Sayantan", "Age" : "22" }**

**{ "\_id" : ObjectId("58f42546bdda0d1c0baa2a31"), "Name" : "Suman", "Age" : "24" }**

**13) Skip a Specific Number of Documents From the Beginning Displayed As The Result Of The Specific Collection:**

**db.COLLECTION\_NAME.find().skip(1)**

Suppose, I have a collection named validData2, and when I use

**db.validData2.find()**  
It generates the following result:  
  
**{ "\_id" : ObjectId("58f41133c4aa3b935cbd42d9"), "Name" : "Sayak" }**

**{ "\_id" : ObjectId("58f42510bdda0d1c0baa2a2f"), "Name" : "Sayan", "Age" : "24" }**

**{ "\_id" : ObjectId("58f4252cbdda0d1c0baa2a30"), "Name" : "Sayantan", "Age" : "22" }**

**{ "\_id" : ObjectId("58f42546bdda0d1c0baa2a31"), "Name" : "Suman", "Age" : "24" }**

**{ "\_id" : ObjectId("58f42578bdda0d1c0baa2a32"), "Name" : "Saptarshi", "Age" : "23" }**

Now, if I use,

**db.validData2.find().limit(4)**You will see only the last document is displayed as first four are skipped.

**{ "\_id" : ObjectId("58f42578bdda0d1c0baa2a32"), "Name" : "Saptarshi", "Age" : "23" }**

You can combine limit() and skip() together, too.

**db.validData2.find().limit(2).skip(2)**

Will display:

**{ "\_id" : ObjectId("58f4252cbdda0d1c0baa2a30"), "Name" : "Sayantan", "Age" : "22" }**

**{ "\_id" : ObjectId("58f42546bdda0d1c0baa2a31"), "Name" : "Suman", "Age" : "24" }**

1. **Display Find Query Result In A Sorted Manner:**To sort documents in mongodb, sort function is uses. We can sort either in ascending or descending order.

sort(1) is used to sort documents in ascending order during display .  
Sort(-1) is used to sort documents in descending order during display.   
  
The basic syntax of sort query is:  
  
**db.COLLECTION\_NAME.find().sort({KEY:1})**

Where key is the field.

1. **Delete Documents From A Collection:**

**Deleting All Entries:**

Suppose, all entries are to be deleted from a table named validData:

**db.validData.remove()**

**Removing One Entry:**

Suppose, db.validData.find() generates the following records:

**{ "\_id" : ObjectId("58f41133c4aa3b935cbd42d9"), "Name" : "Sayak" }**

**{ "\_id" : ObjectId("58f42510bdda0d1c0baa2a2f"), "Name" : "Sayan", "Age" : "24" }**

**{ "\_id" : ObjectId("58f4252cbdda0d1c0baa2a30"), "Name" : "Sayantan", "Age" : "22" }**

**{ "\_id" : ObjectId("58f42546bdda0d1c0baa2a31"), "Name" : "Suman", "Age" : "24" }**

**{ "\_id" : ObjectId("58f42578bdda0d1c0baa2a32"), "Name" : "Saptarshi", "Age" : "23" }**

Now, to remove only one entry (the first one with age 24:

**db.validData.remove({"Age" : "24"},1);**

**Removing All Entries Of a Certain Criteria:**

**db.validData.remove({"Age" : "24"})**

This will remove all records from the collection validData whose age are 24.