1 Start with MongoDB

Log in to one of your machines.
Start the MongoDB daemon

```
$ cd ~/mongodb-tutorial
$ less launch
$ less mongodb.conf
$ ps aux | grep mongo
$ ./launch
about to fork child process, waiting until server is ready for connections.
forked process: 4751
child process started successfully, parent exiting
$ ps aux | grep mongo
```

Launch the mongo shell

```
$ mongo
MongoDB shell version: 2.6.10
connecting to: test
> db
> show dbs
> use local
> show collections
> db.system.indexes.find()
```

2 Create in MongoDB

Create with insert

```
> use ourlibrary
> show collections
> db.createCollection('whatever_you_like', {
    capped: true,
    size: 5242880,
    max: 5000
  })
> db.authors.insert({
    id: 2,
    first_name: 'Arthur',
    last_name:  'Miller',
    age: 25
  })
> show collections
```

Create with save

```
> post = {
    id: 2,
    first_name: 'John',
    last_name: 'Smith',
```

```
    written_books: 4
  }
> db.authors.save(post)
> db.authors.find()
> exit
```

Create with `mongoimport`. We will use this data in the following examples.

```
# CITIES.json was created earlier using mongoexport
$ mongoimport -d germany -c cities --file CITIES.json
$ mongo
> show dbs
```

Create with `mongorestore` (optional)

```
makes a dump only of the specified collection
$ mongodump --collection authors --db ourlibrary
now use the resulting dump to populate a different collection
$ mongorestore --collection authors_copy --db ourlibrary dump/ourlibrary/authors.bson
makes a dump of the whole instance
$ mongodump
```

3 Read from MongoDB

```
> use germany
// counts number of all documents in the germany.cities collection
> db.cities.find().count()
// shows only the first document in the collection
> db.cities.findOne()
// shows all documents in the collection (20 documents at a time)
> db.cities.find()
// the same but in more readable form
> db.cities.find().pretty()

// shows documents where the city attribute is equal KARLSRUHE
> db.cities.find({'city': 'KARLSRUHE'})
// finds all of the documents in the collection,
// but only shows their city attribute which
// is sorted backward from z to a
// and skips first 20 documents
> db.cities.find({}, {
    '_id': 0,
    'city': 1
  }).sort({'city': -1}).skip(20)
// finds all documents that their city attribute has ka in its
// name, with i option for case insensitivity
> db.cities.find({
    'city': {
      $regex: 'ka',
      $options: 'i'
    }
```

```
  }, {
    '_id': 0,
    'city': 1
  }).sort({'city': -1}).skip(20)
// the same but written in a more compact form
> db.cities.find({
    'city': /ka/i
  }, {
    '_id': 0,
    'city': 1
  }).sort({'city': -1}).skip(20)

// finds all documents that their loc attribute is in 0.1 km radius
// of a central coordinate point at [8.4, 49.017]
> db.cities.find({'loc': {
    $geoWithin: {$center: [[8.4, 49.017], 0.1]}
  }})
// finds all documents that either
//  the first element of their loc.coordinates attribute is greater than 8 and
//  the second element of their loc.coordinate attribute is smaller than 50
// or
//  their city attribute name contains karl (case-insensitive)
> db.cities.find({
    $or: [
      {$and: [
        {'loc.coordinates.0': {$gt: 8}},
        {'loc.coordinates.1': {$lt: 50}}
      ]}, {
        'city': /karl/i
      }
    ]
  })

> use ourlibrary
// finds all documents in ourlibrary.authors collection which
// does not contain written_books attribute
> db.authors.find({'written_books': {$exists: 0}})
```

4 Update in MongoDB

```
> use germany
// first finds documents with the given condition and then updates
// all of them (multi: true), but it will not insert a new document
// if it does not find any with this condition (upsert: false)
> db.cities.update({
    'loc.coordinates.0': {$gt: 10.5}},
    {$set: {'region': 'East'}
  }, {
    upsert: false, multi: true
```

```
  })
> db.cities.find({'region': {$exists: 1}})

// just to see the results
> db.cities.find({
    $and: [
      {'checkpoint': {$exists: 1}},
      {'checkpoint': {$gt: 10.5}}
    ]
  })
// a small script in shell that changes checkpoint attribute of
// every document that doesn't have a region attribute and adds
// a region attribute with value West to it
> db.cities.find({'region': {$exists: 0}}).forEach(function(doc) {
    doc.checkpoint = doc.loc.coordinates.0;
    doc.region = 'West';
    db.cities.save(doc);
  })
> db.cities.find({
    $and: [
      {'checkpoint': {$exists: 1}},
      {'checkpoint': {$gt: 10.5}}
    ]
  }) //should be empty

> db.cities.update({}, {$inc: {'checkpoint': 10}}, {
    upsert: false,
    multi: true
  }) //will set checkpoint for all

> db.cities.update({
    $and: [
      {'region': 'East'},
      {'checkpoint': {$exists: 1}}
    ]
  }, {
    $unset: {'checkpoint': ''}
  }, {
    multi: true
  })
> db.cities.find({
    $and: [{'region': 'East'
  }, {
    'checkpoint': {$exists: 1}}]
  })
```

5 Delete in MongoDB

```
> use ourlibrary
// copies the contents of authors collection to backauthors collection
> db.authors.copyTo('backauthors')
// removes all documents of authors
> db.authors.remove({})
// an index on authors should still exists
> db.system.indexes.find()
> db.backauthors.copyTo('authors')
> db.authors.drop()
// by dropping a collection, its index will be deleted automatically
> db.system.indexes.find()

> db.dropDatabase()
> show dbs
> db
```

6 Rename database and collections

Rename a collection

```
> use germany
> show collections
> db.cities.renameCollection('germancities')
```

Rename a database

```
> use admin
> db.copyDatabase('germany','theGermany')
> show dbs
> use germany
> db.dropDatabase()
```

7 Authorisation

Stop mongod, edit the config file to enable authentication, restart. Connect to MongoDB as before. Why can you still use the system despite not having provided any credentials?

Add an user “fred” to the database “germany” with read-only access to “germany” and “test”

```
> use germany
> db.createUser({
    user: 'fred',
    pwd: 'mySecretPassword',
    roles: [
      {
        role: 'read',
        db: 'germany'
```

```
      },
      {
        role: 'read',
        db: 'test'
      }
    ]
  })
```

Add an user “joe” to the administrative database with read-only access to **all** databases

```
> use admin
> db.createUser({
    user: 'joe',
    pwd: 'drowssap',
    roles: [
      {
        role: 'readAnyDatabase',
        db: 'admin'
      }
    ]
  })
```

Grant joe additional read-write access to the database “test”

```
> use admin
> db.grantRolesToUser(
   'joe',
   [
     {
       role: 'readWrite',
       db: 'test'
     }
   ]
  )
```

Change fred’s password in the database “germany”

```
> use germany
> db.changeUserPassword('fred', 'verySecure123')
```

Try connecting to MongoDB as different users; see the output of `mongo --help` for how to specify user, password and authentication database.

Try to make joe the MongoDB root:

```
> use admin
> db.grantRolesToUser(
'joe',
[
{
  role: 'root',
  db: 'admin'
}
]
)
```

Why hasn't it worked? Can you still use the localhost exception to bypass the problem? What can you do to succeed here?

Generate a random 1024-byte key and enable it in configuration (optional)

```
$ openssl rand -base64 754 >mongodb-keyfile
$ chmod 600 mongodb-keyfile
$ vi mongodb.conf
```

Find the keyFile line, point it to your file, save the configuration and restart mongod.

8 Profiling a Database

Note that at this point you must have either succeeded in granting joe the root role or disabled authentication again.

Check current profiling level

```
> db.getProfilingStatus()
```

Begin collecting profiling data for operations slower than 20 ms

```
> db.setProfilingLevel(1, 20)
```

Begin collecting profiling data for all operations

```
> db.setProfilingLevel(2)
```

With profiling enabled, display last 5 operations which took at least 1 ms to execute (if profiling disabled or to show any other results, query the *system.profile* collection)

```
> show profile
```

Disable profiling

```
> db.setProfilingLevel(0)
```

9 Indexing in MongoDB

```
> use germany
> db.system.indexes.find()
> db.system.find().count()

// Single Key query:
> db.cities.find({
    'loc': { $geoWithin: { $center: [[8.4, 49.017 ], 0.1] }}
  }).explain()
// Multi Key query:
> db.cities.find({$or: [
    {$and: [{'loc.coordinates.0': {$gt: 8}}, {'loc.coordinates.1':{$lt:50}}]},
    {'city': /karl/i}
  ]}).explain()

// Create an index
> db.cities.ensureIndex({'loc': '2dsphere'})
> db.system.indexes.find()
```

Compare results of single and compound indexes with each other for different queries.

10 MongoDB and Node.js

10.1 Initial Set-up

Install the Node.js modules we will need (note that they will only be available to scripts placed *in the same directory* as where you ran this command):

```
$ npm install mongoskin express body-parser
```

10.2 Mongoskin

First step: load Mongoskin and create a database object

```
var mongoskin = require('mongoskin');
var db = mongoskin.db('mongodb://[user:password@]localhost:27017/germany');
```

Create a new document

```
db.collection('cities').insert({
                                  name: 'Karlsruhe',
                                  population: 296033
                                }, function (err, result) {
                                  if (err) {
                                    throw err;
                                  }
                                  if (result) {
                                    console.log('Document added');
                                  }
                                });
```

Read all documents from collection, as a cursor which can then be iterated through.

```
db.collection('cities').find({}, function (err, result) {
  if (err) {
    throw err;
  }
  // Now iterate over results
  result.each(function (err, entry) {
    if (entry) {
      // Do something with one document.
      // Here we just display it.
      console.log(JSON.stringify(entry, null, '  '));
    }
  });
});
```

Read documents matching specific criteria, this time as an array

```
db.collection('cities').find({name: 'Frankfurt'}).toArray(
  function (err, result) {
    if (err) {
      throw err;
    }
    // Do something with an array of documents.
    // Here we just display it.
    console.log(JSON.stringify(result, null, '  '));
  });
```

Update two fields, an array and a scalar, in an existing document

```
db.collection('states').update({name: 'Pfalz'},
                               {'$push': {formerCapitals: 'Heidelberg'}},
                               function (err) {
  if (err) {
    throw err;
  }
  db.collection('states').update({name: 'Pfalz'},
                                 {capital: 'Mannheim'},
                                 function (err) {
    if (err) {
      throw err;
    }
    console.log('Entry updated');
  });
});
```

Delete a document

```
db.collection('cities').remove({name: 'Strassburg'},
                               function (err, result) {
  if (err) {
    throw err;
  }
  if (result) {
    console.log('Document deleted');
  }
});
```

Working with values of the `_id` field — using strings directly does *not* work

```
var idObject =
  db.ObjectID.createFromHexString('0123456789abcdef01234');
db.collection('cities').findOne({_id: idObject},
                                function (err, result) {
  if (err) {
    throw err;
  }
  doSomethingWithOneDocument(result);
});
```

10.3 REST API using Express

First step: load and initialise the required modules

```
var express = require('express');
var bodyParser = require('body-parser');
var app = express();
```

Allow Express to handle HTTP message bodies consisting of JSON data

```
app.use(bodyParser.json());
```

Define a custom handler for the request parameter (see below) "collectionName". Such a handler can do any sort of pre-processing, here we obtain a Mongoskin collection object and assign it to a custom property of HTTP-request objects which feature this parameter.

```
app.param('collectionName', function (request, response, next, cn)
{
  request.collection = db.collection(cn);
  return next();
});
```

Define a handler for HTTP POST requests matching the given pattern. Colons indicate respective parts of the URI to be parameters. N.B. The same method pattern works for other HTTP verbs — `app.get()`, `app.put()`, `app.delete()` and so on.

```
app.post('/collections/:collectionName/:id',
        function (request, response, next) {
          // Among other things we can now access the JSON-parsed
          // body of the request, and a dictionary of all
          // parameters used in the URI regardless of whether they
          // have custom handlers or not.
          var result = doSomethingWithRequest(request.body,
                                              request.params.collectionName,
                                              request.params.id);
          // Respond to the request, with valid JSON data
          response.send(result);
        });
```

Finally, have your application listen for requests on given TCP port

```
app.listen(3000);
```

11 Replication

Activate Mongo on both machines.
On the first machine:

```
$ mongo --host <ipOfSecondMachine>
```

On the second machine:

```
$ mongo --host <ipOfFirstMachine>
```

On both of your machines:

```
> rs.status()
> exit()
$ ps aux | grep mongodb
$ kill -TERM <mongoprocessId>
$ vi mongodb.conf
```

```
# Replication Options
replSet = satRset
oplogSize = 1024
```

```
$ mongo
> rs.status()
```

Just on the one to be Primary:

```
> rs.initiate()
satRset:OTHER> rs.status()
satRset:PRIMARY> rs.conf()
satRset:PRIMARY> rs.add('<name-ipofTheOtherMachine>:27017')
```

On the secondary machine:

```
> rs.status()
```

```
satRset:SECONDARY> show dbs
```

Create new database and collection on primary and observe the changes on secondary.
Try to do the same on secondary.
Force Primary to shutdown and monitor changes:

```
satRset:PRIMARY> db.adminCommand({shutdown: 1, force: true})
> exit()
$ ./launch
```

12 Sharding

Start the Config Server Database Instances
1. Create data directories for each of the three config server instances

```
 > mongod --configsvr --dbpath <path> --port <port>
```

The default port for config servers is 27019
Start the `mongos` Instances

```
mongos --configdb <config server hostnames:port>
```

Add Shards to the Cluster
From `mongo`, connect to the `mongos` instance.

```
> sh.addShard("rs1/mongodb0.example.net:27017,\
    mongodb1.example.net:27017,mongodb2.example.net:27017")
```

Enable Sharding for a Database

```
$ mongo --host <hostname of machine running mongos> --port <port mongos listens on>
> sh.enableSharding("<database>")
```

Enable Sharding for a Collection

If the collection already contains data you must create an index on the shard key using ensureIndex().
If the collection is empty then MongoDB will create the index as part of the sh.shardCollection() step.

```
sh.shardCollection("<database>.<collection>", shard-key-pattern)
```