MongoDB on mac

Install:

https://stackoverflow.com/questions/18452023/installing-and-running-mongodb-on-osx

Mac Installation:

1. Install brew

```
ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/ir
```

2. Update and verify you are good with

```
brew update
brew doctor
```

3. Install mongodb with

```
brew install mongodb
```

4. Create folder for mongo data files:

```
mkdir -p /data/db
```

5. Set permissions

```
sudo chown -R `id -un` /data/db
```

6. Open another terminal window & run and keep running a mongo server/daemon

```
mongod
```

7. Return to previous terminal and run a mongodb shell to access data

```
mongo
```

To quit each of these later:

1. The Shell:

```
quit()
```

2. The Server

```
ctrl-c
```

Steps to start mongodb server in your mac

1. Open Terminal
2. Run the command `sudo su`
3. Enter your administrator password
4. run the command `mongod`
5. MongoDb Server starts

Hope it helps you. Thanks

How to check if mongo db is running on Mac?

```
ps -ef | grep mongod | grep -v grep | wc -l | tr -d ' '
```

Run the following in your Terminal:

```
ps -ef | grep mongod | grep -v grep | wc -l | tr -d ' '
```

This will get you the number of `MongoDB` processes running, thus if it is other than `0`, then you have `MongoDB` running on your system.

Step-by-Step

- The **`ps -ef | grep mongod`** part return all the running processes, that have any relation to the supplied string, i.e. `mongod`, e.g. have the string in the executable path, have the string in the username, etc.
- When you run the previous command, the `grep mongod` also becomes a process containing the string `mongod` in the `COMMAND` column of `ps` output, so it will also appear in the output. For that reason you need to eliminate it by piping **`grep -v grep`**, which filters all the lines from the input that contain the string `grep`.
- So now you have all possible lines that contain string `mongod` and are not the instances of `grep`. What to do? Count them, and do that with **`wc -l`**.
- `wc -l` output contains additional formatting, i.e. spaces, so just for the sake of the beauty, run **`tr -d ' '`** to remove the redundant spaces.

As a result you will get a single number, representing the number of processes you `grep`'ed for.

```
mongo
```

The shell will give you status of mongodb.

MongoDB folder location in mac (server)

□Macintosh HD□ ▸ □usr□ ▸ □local□ ▸ □var□ ▸ □mongodb□

MongoDB Collections location in mac (data)

□Macintosh HD□ ▸ □data□ ▸ □db□

collection-0--8069778669997908300.wt
collection-2--8069778669997908300.wt

what is the mongo shell

The **mongo shell** is an interactive JavaScript interface to **MongoDB**. You can use the **mongo shell** to query and update data as well as perform administrative operations. The **mongo shell** is a component of the **MongoDB** distributions.

To start mongoDB server

New Terminal:

mongod

To start mongoDB shell

New Terminal:

Mongo

On shell terminal

```
> show dbs
admin   0.000GB
config  0.000GB
local   0.000GB

> use classes
switched to db classes
> db.createCollection('class')
{ "ok" : 1 }

> db.createCollection('student')
{ "ok" : 1 }

> db.createCollection('students')
{ "ok" : 1 }



> db student.drop()



> db.class.insert({code:'FBW1', mamber:12, isRunning:true})



> db.class.find().pretty()
{
        "_id" : ObjectId("5cb457a1c1d9bcafc2e3d87b"),
        "code" : "FBW1",
        "mamber" : 12,
        "isRunning" : true
}
>
```

```
> db.teacher.insert({firstName:'kostas', lastNmame:'Diakogiannis', age:31, nickName:'Jake'})
WriteResult({ "nInserted" : 1 })
```

```
> db.student.insert({firstName:'Mauro', lastName:'Navaroo', favTopics:['swimming',
'runnung'], scores:[{html:78,css:88,js:77}]})
WriteResult({ "nInserted" : 1 })
```

```
> db.student.find().pretty()
{
        "_id" : ObjectId("5cb45bd4c1d9bcafc2e3d87d"),
        "firstName" : "Mauro",
        "lastName" : "Navaroo",
        "favTopics" : [
                "swimming",
```

```
            "runnung"
        ],
        "scores" : [
            {
                "html" : 78,
                "css" : 88,
                "js" : 77
            }
        ]
}
>



> db.student.find({firstName:'mohammed'})



> db.student.find({firstName:'mohammed'}).pretty()
{
        "_id" : ObjectId("5cb45d90c1d9bcafc2e3d87e"),
        "firstName" : "mohammed",
        "lastName" : "wahba",
        "favTopics" : [
            "swimming2",
            "runnung2"
        ],
        "scores" : [
            {
                "html" : 44,
                "css" : 48,
                "js" : 33
```

```
            }
        ]
}
>
```

```
> db.student.find({firstName:'mohammed'},{lastName: 1}).pretty()
{ "_id" : ObjectId("5cb45d90c1d9bcafc2e3d87e"), "lastName" : "wahba" }
>
> db.student.find({firstName:'mohammed'},{lastName: 1, _id:0}).pretty()
{ "lastName" : "wahba" }
>
```

```
> db.student.update({firstName:'mohammed'},{$set:{firstName:'Mmohammed',
lastName:'wahbee'}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
>
```

```
> db.student.update({},{$set:{instute:'DCI'}}, {multi:true})
WriteResult({ "nMatched" : 3, "nUpserted" : 0, "nModified" : 2 })
>
```

// db.collection.deleteOne() for Delelt the first one in matches the criteria

```
> db.student.deleteOne({firsName:'ali'})
```

{ "acknowledged" : true, "deletedCount" : 1 }

>

// db.collection.deleteMany() for method deletes all documents that match the criteria.

> db.student.deleteMany({firsName:'ali'})

{ "acknowledged" : true, "deletedCount" : 3 }

>

$or:[{option1}, {option2}]

```
@(shell):1:1
> db.teacher.find({ $or:[{firstName:'Tamer'},{ nickName:'Jake'}] }).pretty()
{
        "_id" : ObjectId("5cb45a79c1d9bcafc2e3d87c"),
        "firstName" : "Kostas",
        "lastNmame" : "Diakogiannis",
        "age" : 31,
        "nickName" : "Jake"
}
{
        "_id" : ObjectId("5cb58b1c49d5c2a2fd2c4581"),
        "firstName" : "Tamer",
        "lastName" : "Alsamer",
        "age" : 33
}
>
```

$gt greater than
$gte greater than or equal
$eg equal
$lt. lower than
$lte. Lower than or equal
$neq not equal

```
> db.teacher.find( { age: { $gt: 30 } } )
{ "_id" : ObjectId("5cb45a79c1d9bcafc2e3d87c"), "firstName" : "Kostas", "lastNmame" : "Diakogiannis", "age" : 31, "nickName" : "Jake" }
{ "_id" : ObjectId("5cb58b1c49d5c2a2fd2c4581"), "firstName" : "Tamer", "lastName" : "Alsamer", "age" : 33 }
>
```

```
> db.class.find({ members:{$gt:15}}, {code:1,_id:0})
{ "code" : "FBW2" }
>
```

.count

db.student.find().count()

```
> db.student.find().count()
3
>
```

Import json file to MngogDB. **Via my mac**

1

New terminal

2

/usr/local/mongodb/bin

3

mongoimport --db classes --collection student –file

/Users/mohammed/Desktop/students_json_file.json

```
sh-3.2# mongoimport --db classes --collection student /Users/mohammed/Desktop/students_json_file.js
2019-04-16T12:03:55.232+0200    connected to: localhost
2019-04-16T12:03:55.237+0200    imported 8 documents
```

MongoDB deals with array {$elemMatch: }

https://docs.mongodb.com/manual/reference/operator/update/each/

```
{
        "_id" : ObjectId("5cb5a88b90f51e76c1d2fae9"),
        "firstName" : "Mohammed",
        "lastName" : "Wahba",
        "favTopics" : [
                "Java",
                "Express"
        ],
        "scores" : [
                {
                        "html" : 90,
                        "css" : 58,
                        "js" : 88
                }
        ],
        "profession" : "Student"
}
```

```
> db.student.find({ favTopics:{$elemMatch:{$eq:'Java'} } })
{ "_id" : ObjectId("5cb5a88b90f51e76c1d2fae9"), "firstName" : "Mohammed", "lastName" : "Wahba",
s" ], "scores" : [ { "html" : 90, "css" : 58, "js" : 88 } ], "profession" : "Student" }
>
```

```
> db.student.find({ favTopics:{$elemMatch:{$eq:'Java'} } },{firstName:1,lastName:1,favTopics:1,_id:0})
{ "firstName" : "Mohammed", "lastName" : "Wahba", "favTopics" : [ "Java", "Express" ] }
>
```

```
> db.student.find({ favTopics:{$elemMatch:{$eq:'Java'} } },{firstName:1,lastName:1,favTopics:1,_id:0}).pretty()
{
        "firstName" : "Mohammed",
        "lastName" : "Wahba",
        "favTopics" : [
                "Java",
                "Express"
        ]
}
>
```

```
> db.student.find({ favTopics:{$elemMatch:{$eq:'Java'} } },{firstName:1,lastName:1,favTopics:1,_id:0}).count()
1
>
```

Sort () **1 creasing** **-1 decreasing**

.sort({firstName:1})

```
> db.student.find({},{firstName:1,_id:0}).sort({firstName:1})
{ "firstName" : "Ali" }
{ "firstName" : "Daniel" }
{ "firstName" : "Marcelo" }
{ "firstName" : "Mauro" }
{ "firstName" : "Meir" }
{ "firstName" : "Milad" }
{ "firstName" : "Mmohammed" }
{ "firstName" : "Mohamad" }
{ "firstName" : "Mohammed" }
{ "firstName" : "Murhaf" }
{ "firstName" : "murhaf" }
```

.sort({firstName:-1})

```
> db.student.find({},{firstName:1,_id:0}).sort({firstName:-1})
{ "firstName" : "murhaf" }
{ "firstName" : "Murhaf" }
{ "firstName" : "Mohammed" }
{ "firstName" : "Mohamad" }
{ "firstName" : "Mmohammed" }
{ "firstName" : "Milad" }
{ "firstName" : "Meir" }
{ "firstName" : "Mauro" }
{ "firstName" : "Marcelo" }
{ "firstName" : "Daniel" }
{ "firstName" : "Ali" }
>
```

.limit(3) **to show the first 3**

```
> db.student.find({},{firstName:1,_id:0}).sort({firstName:-1}).limit(3)
{ "firstName" : "murhaf" }
{ "firstName" : "Murhaf" }
{ "firstName" : "Mohammed" }
>
```

Ex:
Sort, find and return the top 3 JS scores, return the firstname and lastanme and score

db.student.find({ },{firstName:1,_id:0,lastName:1}).sort({"scores.js":1}).limit(3)

```
> db.student.find({ },{firstName:1,_id:0,lastName:1}).sort({"scores.js":1}).limi
t(3)
{ "firstName" : "Mmohammed", "lastName" : "wahbee" }
{ "firstName" : "murhaf", "lastName" : "orfali" }
{ "firstName" : "Marcelo", "lastName" : "Ramirez" }
>
```

Skip()

```
> db.student.find({ },{firstName:1,_id:0,lastName:1}).sort({"scores.js":1}).limit(3)
{ "firstName" : "Mmohammed", "lastName" : "wahbee" }
{ "firstName" : "murhaf", "lastName" : "orfali" }
{ "firstName" : "Marcelo", "lastName" : "Ramirez" }
> db.student.find({ },{firstName:1,_id:0,lastName:1}).sort({"scores.js":1}).limit(3).skip(1)
{ "firstName" : "murhaf", "lastName" : "orfali" }
{ "firstName" : "Marcelo", "lastName" : "Ramirez" }
{ "firstName" : "Meir", "lastName" : "Overfeirst" }
```

Mongo update query to find field within array of objects

db.student.update(

{firstName:'Mohammed', "array.object":Curretvalue},

{$set:{"array.$.object":NewValue}}

)

```
> db.student.update({firstName:'Mohammed', "scores.js":101},{$set:{"scores.$.js":90}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
>
```

Why to define "array.object":Curretvalue, only when I need update the object inside array.

But not use to update only simple array without object.

Why to write .$, to find automatically the element index in array the match the needed field ("array.object":Curretvalue).

So we can replace $ by index 0 or 1 2 3 4 … :

{firstName:'Mohammed', "array.object":Curretvalue},

{$set:{"array.0.object":NewValue}}

$in

`$in`

The `$in` operator selects the documents where the value of a field equals any value in the specified array. To specify an `$in` expression, use the following prototype:

For comparison of different BSON type values, see the specified BSON comparison order.

copy

```
{ field: { $in: [<value1>, <value2>, ... <valueN> ] } }
```

Ex. Find all student that has CSS or sass

```
> db.student.find({favTopics:{$in:["CSS","sass"]} },{firstName:1,_id:0,lastName:1})
{ "firstName" : "Mohamad", "lastName" : "Lahham" }
{ "firstName" : "Ali", "lastName" : "Pudina" }
>
```

Complex ex. Return the firstName and the lastName of every student whose JS score is more than 80, and it's favorite topic is one of : 'JS', 'JQuery', 'React' or 'Express'. Sort the results by the JS score.

db.student.find({"scores.0.js":{$gt:80},favTopics:{$in:['Express','React','JS','JQuery']}},{"scores.js":1,firstName:1,lastName:1,_id:0}).sort({"scores.js":-1})

```
> db.student.find({"scores.0.js":{$gt:80},favTopics:{$in:['Express','React','JS','J
{ "firstName" : "Mohammed", "lastName" : "Wahba", "scores" : [ { "js" : 92 } ] }
{ "firstName" : "Milad", "lastName" : "Khoshkaran", "scores" : [ { "js" : 88 } ] }
{ "firstName" : "Murhaf", "lastName" : "Orfali", "scores" : [ { "js" : 88 } ] }
>
```

Push to array $push

The following example appends 89 to the `scores` array:

```
db.students.update(
   { _id: 1 },
   { $push: { scores: 89 } }
)
```

Push to array $push

Remove All Items That Match a Specified `$pull` Condition

Given the following document in the `profiles` collection:

```
{ _id: 1, votes: [ 3, 5, 6, 7, 7, 8 ] }
```

The following operation will remove all items from the `votes` array that are greater than or equal to (`$gte`) 6:

```
db.profiles.update( { _id: 1 }, { $pull: { votes: { $gte: 6 } } } )
```

After the update operation, the document only has values less than 6:

```
{ _id: 1, votes: [  3,  5 ] }
```

To create a document referenced relationship in MongoDB

Document Referenced Relationships

You can use a document reference to create a relationship. Rather than embedding the child document into the parent document (like we did above), you separate the child document out into to its own stand alone document.

So we could do this:

Parent Document

```
db.artists.insert(
    {
        _id : 4,
        artistname : "Rush"
    }
)
```

Child Documents

We'll insert 3 child documents — one for each band member:

```
db.musicians.insert(
    {
        _id : 9,
        name : "Geddy Lee",
        instrument : [ "Bass", "Vocals", "Keyboards" ],
        artist_id : 4
    }
)
```

```
db.musicians.insert(
    {
        _id : 10,
        name : "Alex Lifeson",
        instrument : [ "Guitar", "Backing Vocals" ],
        artist_id : 4
    }
)
```

Querying the Relationship

After inserting the above two documents, you can use `$lookup` to perform a left outer join on the two collections.

This, in conjunction with the `aggregate()` method, and `$match` to specify the specific artist you're interested in, will return parent and child documents in one.

```
db.artists.aggregate([
    {
      $lookup:
        {
          from: "musicians",
          localField: "_id",
          foreignField: "artist_id",
          as: "band_members"
        }
   },
   { $match : { artistname : "Rush" } }
]).pretty()
```

Result:

```
{
  "_id" : 4,
  "artistname" : "Rush",
  "band_members" : [
    {
      "_id" : 9,
      "name" : "Geddy Lee",
      "instrument" : [
        "Bass",
        "Vocals",
        "Keyboards"
      ],
      "artist_id" : 4
    },
    {
      "_id" : 10,
      "name" : "Alex Lifeson",
      "instrument" : [
        "Guitar",
        "Backing Vocals"
      ],
      "artist_id" : 4
    },
```

You can see that the first two fields are from the artists collection, and the rest of it is from the musicians collection.

So if you only query the artists collection by itself:

```
db.artists.find( { artistname : "Rush" } )
```

You'd only get this:

```
{ "_id" : 4, "artistname" : "Rush" }
```

No related data is returned.

To apply on our example:

```
> db.student.update({},{$set:{classcode:'FBW1'}},{multi:true } )
WriteResult({ "nMatched" : 11, "nUpserted" : 0, "nModified" : 11 })
> db.class.aggregate(
```

```
> db.class.aggregate([ { $lookup: { from:"student" , localField:"code" ,
    foreignField:"classcode"  , as:"participates"}}]).pretty()
```

Aggregate group

```
> db.movies.aggregate([{
... $group:{
... _id:'$director',
... average_movies:{$avg:'$imdb_ratio'}
... }
... }])
{ "_id" : "Quentin Tarantino", "average_movies" : 8.6 }
{ "_id" : "Christopher Nolan", "average_movies" : 8.675 }
>
```

MongoDB Compass

Trello

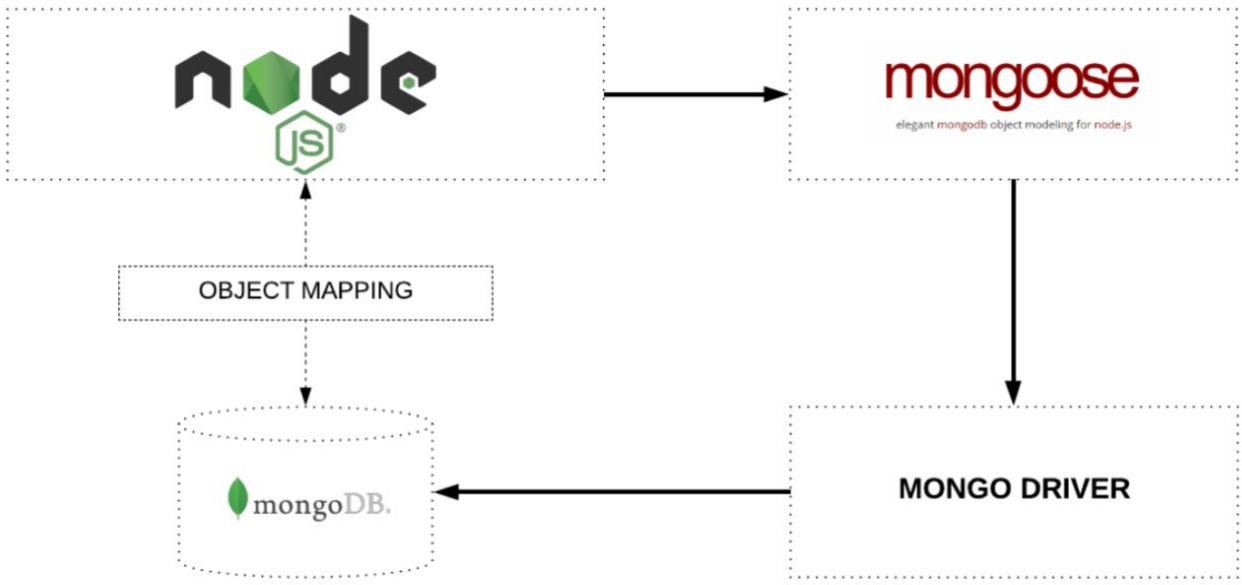
Trello is a task management app that gives you a visual **overview** of what is being worked on and who is working on it. It used the Kanban system, which was

developed in Toyota as a system to keep production levels high and maintain flexibility.

---

mongoose

Mongoose is an Object Data Modeling (ODM) library for MongoDB and Node.js. It manages relationships between data, provides schema validation, and is used to translate between objects in code and the representation of those objects in MongoDB.



Object Mapping between Node and MongoDB managed via Mongoose

Install this in your project:

npm install --save mongoose

---

mongoose.set('useNewUrlParser', true);

The `useNewUrlParser` Option

By default, `mongoose.connect()` will print out the below warning:

```
DeprecationWarning: current URL string parser is deprecated, and will be
removed in a future version. To use the new parser, pass option
{ useNewUrlParser: true } to MongoClient.connect.
```

The MongoDB Node.js driver rewrote the tool it uses to parse MongoDB connection strings. Because this is such a big change, they put the new connection string parser behind a flag. To turn on this option, pass the `useNewUrlParser` option to `mongoose.connect()` or `mongoose.createConnection()`.

```
mongoose.connect(uri, { useNewUrlParser: true });
mongoose.createConnection(uri, { useNewUrlParser: true });
```

You can also set the global `useNewUrlParser` option to turn on `useNewUrlParser` for every connection by default.

```
// Optional. Use this if you create a lot of connections and don't want
// to copy/paste `{ useNewUrlParser: true }`.
mongoose.set('useNewUrlParser', true);
```

To test your app with `{ useNewUrlParser: true }`, you only need to check whether your app successfully connects. Once Mongoose has successfully connected, the URL parser is no longer important. If you can't connect with `{ useNewUrlParser: true }`, please open an issue on GitHub.

https://mongoosejs.com/docs/deprecations.html

css Example
Css Example

css Example
Css Example