mongosh MongoDB shell

START HERE

`mongosh "<uri>"`
Connect

`db.<coll>.find()`
Read documents

`show dbs`
List databases

`db.<coll>.insertOne({})`
Add a document

`use <db>`
Switch database

`db.<coll>.updateOne({}, {$set: {}})`
Change a document

`show collections`
List collections

`db.<coll>.aggregate([])`
Run a pipeline

CONCEPTS

mongosh is a JavaScript shell: db is the current database, db.<name> a collection, and any JS works.

Deployment server or cluster
- **Database** db / use <db>
  - **Collection** db.users
    - **Document** BSON object

A database or collection is created lazily on the first insert; use <db> alone creates nothing.

Methods return cursors; the shell prints 20 results, type it for the next batch.

CONNECT

`mongosh`
Connect to localhost:27017

`mongosh "mongodb://<host>:27017/<db>"`
Connect with a URI

`mongosh "mongodb+srv://<host>/<db>"`
Connect to an Atlas style SRV URI

`mongosh --host <host> --port <port> -u <user> -p`
Connect with options, prompt for password

`--authenticationDatabase admin`
Database holding the user

`--tls`
Connect over TLS

`--eval "<js>" <uri>`
Run JavaScript and exit

`mongosh <uri> <file.js>`
Run a script file then exit

`exit / quit()`
Leave the shell

INSPECT

`show dbs`
List databases

`use <db>`
Switch database

`db`
Print the current database name

`show collections`
List collections

`show users`
List users of this database

`db.<coll>.countDocuments({})`
Count matching documents

`db.<coll>.estimatedDocumentCount()`
Fast approximate count

INSPECT (cont.)

`db.<coll>.stats()`
Size and storage figures

`db.<coll>.getIndexes()`
List indexes

`db.stats()`
Database size summary

`help / db.help() / db.<coll>.help()`
List available commands

READ

`db.<coll>.find()`
All documents (first batch)

`db.<coll>.find({ <field>: <value> })`
Match on equality

`db.<coll>.findOne({ <filter> })`
First match as one document

`db.<coll>.find({}, { <f>: 1 })`
Projection: pick fields

`db.<coll>.find().sort({ <f>: -1 })`
Sort, 1 ascending, -1 descending

`db.<coll>.find().limit(<n>)`
Cap results; chain .skip(<n>) to page

`db.<coll>.distinct("<field>")`
Unique values of a field

`db.<coll>.find(<filter>).explain("executionStats")`
Show how the query ran

`it`
Show the next batch of a cursor

Query operators

Operator	Meaning
`$eq $ne`	equals, not equals
`$gt $gte $lt $lte`	comparison
`$in $nin`	value in / not in list
`$and $or $nor`	combine conditions
`$not`	negate an operator
`$exists`	field is present
`$type`	BSON type check
`$regex`	pattern match
`$elemMatch`	one array element matches all
`$size`	array length
`$all`	array holds every value
`$expr`	compare fields, use aggregation expressions

WRITE

`db.<coll>.insertOne({ <doc> })`
Insert one document

WRITE (cont.)

`db.<coll>.insertMany([{ }, { }])`
Insert several documents

`db.<coll>.updateOne(<filter>, { $set: { <f>: <v> } })`
Change first match

`db.<coll>.updateMany(<filter>, { $set: { } })`
Change all matches

`db.<coll>.updateOne(<filter>, <update>, { upsert: true })`
Insert if nothing matches

`db.<coll>.deleteOne(<filter>)`
Delete first match

`db.<coll>.deleteMany(<filter>)`
Delete all matches

deleteMany({}) empties the collection but keeps its indexes.

`db.<coll>.drop()`
Delete the collection

`db.dropDatabase()`
Delete the current database

Update operators

Operator	Effect
`$set $unset`	set or remove a field
`$inc $mul`	add to or multiply a number
`$min $max`	keep the lower or higher value
`$rename`	rename a field
`$push $addToSet`	append to array, only if new
`$pull $pop`	remove from array
`$currentDate`	set to now
`$setOnInsert`	only applied on upsert insert

An update without an operator like $set is rejected; use replaceOne to swap a document.

AGGREGATION

`db.<coll>.aggregate([<stage>])`
Run a pipeline of stages

`{ $match: { <filter> } }`
Filter documents early

`{ $group: { _id: "$<f>", n: { $sum: 1 } } }`
Group and count

`{ $sort: { <f>: -1 } }`
Order the stream

`{ $project: { <f>: 1, <new>: "$<g>" } }`
Reshape documents

`{ $lookup: { from, localField, foreignField, as } }`
Left join another collection

AGGREGATION (cont.)

Put $match (and $sort) first so the pipeline can use indexes and handle fewer documents.

INDEXES

`db.<coll>.createIndex({ <f>: 1 })`
Create an ascending index

`db.<coll>.createIndex({ <a>: 1, <b>: -1 })`
Compound index, order matters

`db.<coll>.createIndex({ <f>: 1 }, { unique: true })`
Enforce uniqueness

USERS & ADMIN

`db.createUser({ user: "<u>", pwd: passwordPrompt(), roles: ["readWrite"] })`
Create a user, ask for password

DUMP & RESTORE

`mongodump --uri="<uri>" --out=<dir>`
Dump databases to BSON files

`mongorestore --uri="<uri>" <dir>`
Restore from a dump directory

mongodump, mongorestore, mongoexport and mongoimport are separate Database Tools, not part of mongosh.

RECIPES

Top 5 values of a field

```
db.orders.aggregate([
  { $group: { _id: "$status", n: { $sum: 1 } } },
  { $sort: { n: -1 } },
  { $limit: 5 }
])
```

Scripted query from the shell

```
mongosh "$MONGO_URI" --quiet \
  --eval 'printjson(db.users.find({active:true}).limit(3).toArray())'
```

Find and remove duplicates by key

```
db.users.aggregate([
  { $group: { _id: "$email", ids: { $push: "$_id" }, n: { $sum: 1 } } },
  { $match: { n: { $gt: 1 } } }
])
```

Backup then restore into another server

```
mongodump --uri="mongodb://old:27017/app" --gzip --archive=app.gz
mongorestore --uri="mongodb://new:27017" --gzip --archive=app.gz
```