MongoDB Command Reference (mongosh + CLI)

Practical cheat sheet with common patterns

December 15, 2025

Contents

1 Connect, context, and help (mongosh)

```
# Connect
mongosh
mongosh "mongodb://localhost:27017"
mongosh "mongodb://user:pass@host:27017/db?authSource=admin"
mongosh "mongodb+srv://user:pass@cluster/db?retryWrites=true&w=majority"

# In-shell basics
help
```

```
db.help()
db.<collection>.help()

# Switch dbs
show dbs
use mydb
db
show collections

# Useful info
db.getName()
db.getMongo()
db.getMongo().getDBNames()
db.runCommand({ ping: 1 })
db.runCommand({ buildInfo: 1 })
```

2 Database and collection DDL

2.1 Create, drop, rename, stats

```
db.createCollection("users")
db.users.drop()
db.dropDatabase()

db.users.renameCollection("customers")  # admin command context

db.stats()
db.users.stats()

db.getCollectionNames()
db.getCollectionInfos()
```

2.2 Views and time-series collections

```
# View
db.createView(
  "active_users_view",
  "users",
  [
    { $match: { active: true } },
    { $project: { name: 1, email: 1, _id: 0 } }
  ]
)
db.active_users_view.find({})

# Time-series collection
db.createCollection("sensor_readings", {
  timeseries: {
    timeField: "ts",
    metaField: "device",
    granularity: "seconds"
  }
})
```

3 CRUD: Create (insert)

```
db.users.insertOne({ name: "Ana", total_movies_watched: 12, createdAt: new Date() })

db.users.insertMany([
  { name: "Diogo", total_movies_watched: 3, country: "PT" },
  { name: "Joana", total_movies_watched: 40, country: "PT" }
])

# ordered vs unordered
db.users.insertMany(docs, { ordered: false })

# writeConcern on writes
db.users.insertOne(doc, { writeConcern: { w: "majority", j: true, wtimeout: 5000 } })
```

4 CRUD: Read (find)

4.1 Core find patterns

```
db.users.find({})
db.users.find({ name: "Ana" })
db.users.findOne({ name: "Ana" })

# Projection (include / exclude)
db.users.find({ country: "PT" }, { name: 1, total_movies_watched: 1, _id: 0 })
db.users.find({ country: "PT" }, { legacyField: 0 })   # exclude

# Sorting, limiting, skipping
db.users.find({}).sort({ total_movies_watched: -1 }).limit(10)
db.users.find({}).skip(20).limit(10)

# Cursor helpers
db.users.find({}).pretty()
db.users.find({}).toArray()
db.users.find({}).forEach(doc => printjson(doc))

# Counting
db.users.countDocuments({ total_movies_watched: { $gte: 10 } })
db.users.estimatedDocumentCount()

# Distinct
db.users.distinct("country")

# Existential checks
db.users.find({ email: { $exists: true } })
```

4.2 Read concern, collation, hint, and maxTimeMS

```
db.users.find({ country: "PT" }, { name: 1 })
  .readConcern("majority")

# Collation (case/locale rules)
db.users.find({ name: "ana" }).collation({ locale: "pt", strength: 1 })

# Hint (force index usage)
db.users.find({ country: "PT" }).hint({ country: 1 })
```

```
# Prevent runaway queries
db.users.find({}).maxTimeMS(2000)
```

5 Query operators reference

5.1 Comparison

```
{ field: { $eq: 10 } }
{ field: { $ne: 10 } }
{ field: { $gt: 10 } }
{ field: { $gte: 10 } }
{ field: { $lt: 10 } }
{ field: { $lte: 10 } }
{ field: { $in: [1,2,3] } }
{ field: { $nin: [1,2,3] } }
```

5.2 Logical

```
{ $and: [ { a: 1 }, { b: 2 } ] }
{ $or:  [ { a: 1 }, { b: 2 } ] }
{ $nor: [ { a: 1 }, { b: 2 } ] }
{ a: 1, b: 2 }         # implicit AND
{ a: { $not: { $gt: 3 } } }
```

5.3 Element and type

```
{ field: { $exists: true } }
{ field: { $type: "string" } }    # or numeric type codes
```

5.4 Array

```
{ tags: "mongodb" }                              # contains value
{ tags: { $all: ["a", "b"] } }
{ tags: { $size: 3 } }
{ items: { $elemMatch: { price: { $gt: 10 }, qty: { $lte: 5 } } } }
```

5.5 Bitwise

```
{ flags: { $bitsAllSet: [0, 3] } }
{ flags: { $bitsAnySet: [1] } }
```

5.6 Geospatial

```
# 2dsphere index required
db.places.createIndex({ loc: "2dsphere" })

db.places.find({
  loc: {
    $near: {
      $geometry: { type: "Point", coordinates: [ -9.14, 38.72 ] },
```

```
      $maxDistance: 5000
    }
  }
})
```

5.7 Text and regex

```
{ name: { $regex: "^An", $options: "i" } }

db.articles.createIndex({ title: "text", body: "text" })
db.articles.find({ $text: { $search: "mongodb indexing" } })
db.articles.find({ $text: { $search: "\"exact phrase\" -exclude" } })
```

5.8 Dates

```
db.events.find({ ts: { $gte: ISODate("2025-01-01T00:00:00Z") } })
db.events.find({ ts: { $lte: new Date() } })
```

5.9 Field paths, dot notation, and arrays

```
# nested fields
db.users.find({ "profile.city": "Lisbon" })

# array index
db.users.find({ "addresses.0.city": "Lisbon" })

# match any element in an array of objects
db.orders.find({ "items.sku": "ABC-1" })
```

6 CRUD: Update

6.1 Update one / many

```
db.users.updateOne({ name: "Ana" }, { $set: { country: "PT" } })
db.users.updateMany({ active: { $ne: true } }, { $set: { active: false } })

db.users.updateOne({ name: "Ana" }, { $unset: { legacyField: "" } })

# Upsert
db.users.updateOne(
  { email: "a@b.com" },
  { $set: { name: "Ana", country: "PT" } },
  { upsert: true }
)
```

6.2 Update operators (common)

```
{ $set: { a: 1 } }
{ $unset: { a: "" } }
{ $inc: { counter: 1 } }
{ $mul: { score: 1.2 } }
{ $min: { bestRank: 10 } }
```

```
{ $max: { bestRank: 10 } }
{ $rename: { oldName: "newName" } }
{ $currentDate: { updatedAt: true } }

# arrays
{ $push: { tags: "new" } }
{ $push: { tags: { $each: ["a","b"], $slice: 10 } } }
{ $addToSet: { tags: "unique" } }
{ $pull: { tags: "removeMe" } }
{ $pop: { tags: 1 } }     # remove last; -1 removes first
```

6.3 Array positional operators

```
# $ positional operator: first matched array element
db.orders.updateOne(
  { _id: 1, "items.sku": "ABC-1" },
  { $set: { "items.$.qty": 5 } }
)

# arrayFilters: update multiple matched elements
db.orders.updateOne(
  { _id: 1 },
  { $set: { "items.$[it].discount": 0.1 } },
  { arrayFilters: [ { "it.qty": { $gte: 3 } } ] }
)
```

6.4 Replace and find-and-modify

```
db.users.replaceOne({ _id: ObjectId("...") }, { name: "New", total_movies_watched: 0 })

db.users.findOneAndUpdate(
  { name: "Ana" },
  { $inc: { total_movies_watched: 1 } },
  { returnDocument: "after" }   # or "before"
)

db.users.findOneAndReplace(
  { name: "Ana" },
  { name: "Ana", total_movies_watched: 999 },
  { returnDocument: "after" }
)

db.users.findOneAndDelete({ name: "Ana" })
```

6.5 Bulk writes

```
db.users.bulkWrite([
  { insertOne: { document: { name: "A", total_movies_watched: 1 } } },
  { updateOne: { filter: { name: "A" }, update: { $inc: { total_movies_watched: 1 } } }
    },
  { updateMany: { filter: { country: "PT" }, update: { $set: { active: true } } } },
  { deleteOne: { filter: { name: "obsolete" } } }
], { ordered: false })
```

7 CRUD: Delete

```
db.users.deleteOne({ name: "Ana" })
db.users.deleteMany({ inactive: true })
```

8 Indexes

8.1 Create, inspect, explain

```
# Single-field: 1 (asc) or -1 (desc). Single-field indexes can be scanned in both
    directions.
db.users.createIndex({ total_movies_watched: 1 })
db.users.createIndex({ total_movies_watched: -1 })

db.users.getIndexes()

# Explain query usage
db.users.explain("executionStats")
  .find({ country: "PT" })
  .sort({ total_movies_watched: -1 })
  .limit(10)
```

8.2 Compound indexes and sort matching

```
# Order matters
db.users.createIndex({ country: 1, total_movies_watched: -1 })

# Good match for:
# find({country:"PT"}).sort({total_movies_watched:-1})
# and
# find({country:"PT"}).sort({country:1, total_movies_watched:-1})
```

8.3 Unique, partial, sparse, TTL

```
# Unique
db.users.createIndex({ email: 1 }, { unique: true })

# Partial
db.users.createIndex(
  { email: 1 },
  { partialFilterExpression: { email: { $exists: true } } }
)

# Sparse (legacy-ish; prefer partial indexes when possible)
db.users.createIndex({ optionalField: 1 }, { sparse: true })

# TTL (field must be Date)
db.sessions.createIndex({ expiresAt: 1 }, { expireAfterSeconds: 0 })
```

8.4 Text, hashed, wildcard, and geo indexes

```
# Text
db.articles.createIndex({ title: "text", body: "text" })

# Hashed (useful for sharding / uniform distribution)
db.users.createIndex({ userId: "hashed" })

# Wildcard (index many fields)
db.events.createIndex({ "attrs.$**": 1 })

# Geo
db.places.createIndex({ loc: "2dsphere" })
```

8.5 Index maintenance

```
db.users.dropIndex("total_movies_watched_1")
db.users.dropIndexes()

# Rebuild / compact (admin operations; may require specific privileges)
db.runCommand({ compact: "users" })
```

9 Aggregation framework

9.1 Common pipeline stages

```
db.orders.aggregate([
  { $match: { status: "PAID" } },
  { $project: {
      userId: 1,
      amount: 1,
      day: { $dateTrunc: { date: "$ts", unit: "day" } }
  }},
  { $group: {
      _id: { userId: "$userId", day: "$day" },
      total: { $sum: "$amount" },
      n: { $sum: 1 }
  }},
  { $sort: { total: -1 } },
  { $limit: 20 }
])
```

9.2 Join-like operations

```
db.orders.aggregate([
  { $lookup: {
      from: "users",
      localField: "userId",
      foreignField: "_id",
      as: "user"
  }},
  { $unwind: "$user" }
])
```

9.3 Facets and bucketing

```
db.users.aggregate([
  { $facet: {
      topWatchers: [
        { $sort: { total_movies_watched: -1 } },
        { $limit: 5 },
        { $project: { name: 1, total_movies_watched: 1, _id: 0 } }
      ],
      byCountry: [
        { $group: { _id: "$country", n: { $sum: 1 } } },
        { $sort: { n: -1 } }
      ]
  }}
])

# $bucket example
db.users.aggregate([
  { $bucket: {
      groupBy: "$total_movies_watched",
      boundaries: [0, 5, 10, 20, 50, 100, 1000000],
      default: "other",
      output: { n: { $sum: 1 } }
  }}
])
```

9.4 Window functions

```
db.orders.aggregate([
  { $setWindowFields: {
      partitionBy: "$userId",
      sortBy: { ts: 1 },
      output: {
        runningTotal: { $sum: "$amount", window: { documents: [ "unbounded", "current" ]
    } }
      }
  }}
])
```

9.5 Aggregation explain

```
db.orders.explain("executionStats").aggregate([
  { $match: { status: "PAID" } },
  { $limit: 10 }
])
```

10 Performance, debugging, and analysis

10.1 Explain plans

```
db.users.explain("queryPlanner").find({ country: "PT" })
db.users.explain("executionStats").find({ country: "PT" }).sort({ total_movies_watched:
    -1 }).limit(10)
db.users.explain("allPlansExecution").find({ country: "PT" })
```

10.2 Profiler (slow query log)

```
db.setProfilingLevel(0)
db.setProfilingLevel(1, { slowms: 100 })  # 0=off, 1=slow ops, 2=all ops
db.getProfilingStatus()

db.system.profile.find({}).sort({ ts: -1 }).limit(10)
```

10.3 Current operations and killing ops

```
db.currentOp()
db.killOp(<opid>)
```

10.4 Stats and server status

```
db.serverStatus()
db.stats()
db.users.stats()
```

11 Schema validation and rules

11.1 JSON Schema validation

```
db.createCollection("users", {
  validator: {
    $jsonSchema: {
      bsonType: "object",
      required: ["name"],
      properties: {
        name: { bsonType: "string", description: "required" },
        total_movies_watched: { bsonType: "int", minimum: 0 },
        country: { bsonType: "string" }
      }
    }
  },
  validationLevel: "moderate",     # strict|moderate
  validationAction: "error"        # error|warn
})
```

11.2 Validation updates

```
db.runCommand({
  collMod: "users",
  validator: { $jsonSchema: { bsonType: "object", required: ["name"] } },
  validationLevel: "strict"
})
```

12 Users, roles, and authentication

12.1 User lifecycle

```
use admin

db.createUser({
  user: "appuser",
  pwd:  "strong_password",
  roles: [
    { role: "readWrite", db: "mydb" }
  ]
})

db.getUsers()

db.updateUser("appuser", {
  roles: [
    { role: "readWrite", db: "mydb" },
    { role: "dbAdmin",   db: "mydb" }
  ]
})

db.changeUserPassword("appuser", "new_password")

db.grantRolesToUser("appuser", [ { role: "read", db: "otherdb" } ])
db.revokeRolesFromUser("appuser", [ { role: "read", db: "otherdb" } ])

db.dropUser("appuser")

db.auth("appuser", "strong_password")
```

12.2 Roles quick reference (common)

```
# read, readWrite
# dbAdmin, userAdmin, dbOwner
# clusterMonitor, clusterAdmin (admin db)
```

13 Transactions (replica set / sharded cluster)

```
const session = db.getMongo().startSession()
const sdb = session.getDatabase("mydb")

session.startTransaction({
  readConcern: { level: "snapshot" },
  writeConcern: { w: "majority" }
})

try {
  sdb.accounts.updateOne({ _id: 1 }, { $inc: { balance: -10 } })
  sdb.accounts.updateOne({ _id: 2 }, { $inc: { balance:  10 } })
  session.commitTransaction()
} catch (e) {
  session.abortTransaction()
  throw e
} finally {
  session.endSession()
}
```

14 Change streams (watch real-time changes)

```
# Requires replica set or sharded cluster
const cursor = db.users.watch([
  { $match: { "fullDocument.country": "PT" } }
], { fullDocument: "updateLookup" })

cursor.hasNext()
cursor.next()
```

15 Replica set and sharding admin (quick reference)

15.1 Replica sets

```
rs.initiate()
rs.status()
rs.conf()
rs.reconfig(rs.conf())

rs.stepDown()
rs.freeze(60)

db.isMaster()           # older name; may still be present in some contexts
db.hello()              # modern command (depending on version)
```

15.2 Sharding

```
sh.status()
sh.enableSharding("mydb")

# shard key examples
sh.shardCollection("mydb.users", { userId: "hashed" })  # hashed
sh.shardCollection("mydb.orders", { ts: 1, _id: 1 })    # ranged (example)

# balancer
sh.getBalancerState()
sh.setBalancerState(true)
```

16 Read/write concerns, preferences, and sessions

```
# Read preference (driver-ish concept, but available in shell)
db.getMongo().setReadPref("primaryPreferred")

# Read concern (consistency)
db.users.find({}).readConcern("majority")

# Write concern (durability)
db.users.insertOne({ a: 1 }, { writeConcern: { w: 1, j: true } })
```

17 Import, export, backup, restore (CLI tools)

17.1 Import / export JSON and CSV

```
# Import JSON array
mongoimport --db mydb --collection users --file users.json --jsonArray

# Import CSV (you must provide fields)
mongoimport --db mydb --collection users --type csv --headerline --file users.csv

# Export JSON
mongoexport --db mydb --collection users --out users.json

# Export CSV with fields
mongoexport --db mydb --collection users --type=csv --fields name,country,
    total_movies_watched --out users.csv
```

17.2 Backup / restore

```
mongodump --db mydb --out dump/
mongorestore --db mydb dump/mydb/

# Auth examples
mongodump --uri="mongodb://user:pass@host:27017/mydb?authSource=admin" --out dump/
mongorestore --uri="mongodb://user:pass@host:27017/mydb?authSource=admin" dump/mydb/
```

18 GridFS (store large files)

```
# CLI helper
mongofiles -d mydb put ./bigfile.zip
mongofiles -d mydb list
mongofiles -d mydb get bigfile.zip
```

19 Extended operator and stage reference (compact)

19.1 Query operators

Category	Operators
Comparison	`$eq, $ne, $gt, $gte, $lt, $lte, $in, $nin`
Logical	`$and, $or, $nor, $not`
Element	`$exists, $type`
Evaluation	`$regex, $expr, $jsonSchema, $mod, $text, $where (avoid)`
Array	`$all, $elemMatch, $size`
Geospatial	`$geoWithin, $geoIntersects, $near, $nearSphere`
Bitwise	`$bitsAllSet, $bitsAnySet, $bitsAllClear, $bitsAnyClear`

19.2 Update operators

Category	Operators
Field	`$set, $unset, $rename, $setOnInsert`

Arithmetic	`$inc, $mul, $min, $max`
Dates	`$currentDate`
Arrays	`$push, $push/$each, $addToSet, $pull, $pop`
Bitwise	`$bit`

19.3 Aggregation stages

Stage	Typical use
`$match`	Filter documents early
`$project` / `$set`	Select/compute fields
`$unset`	Remove fields
`$group`	Aggregations by key
`$sort`	Order results
`$limit` / `$skip`	Pagination
`$unwind`	Explode arrays
`$lookup`	Join other collections
`$facet`	Multiple pipelines in parallel
`$bucket` / `$bucketAuto`	Histogram-style grouping
`$setWindowFields`	Window functions (running totals etc.)
`$merge` / `$out`	Write pipeline results

20 Notes

- Single-field index direction (`1` vs `-1`) rarely matters for performance; MongoDB can scan forward/backward. It matters more in compound indexes and sort patterns.
- `0` is not a valid index direction.
- Some commands depend on MongoDB version, deployment mode, and privileges (Atlas vs self-hosted, replica set vs standalone, etc.).