The `mongo` Shell

On this page

Introduction

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. Once you have installed and have started MongoDB, connect the `mongo` shell to your running MongoDB instance.

Most examples in the MongoDB Manual use the `mongo` shell; however, many drivers provide similar interfaces to MongoDB.

Start the `mongo` Shell

IMPORTANT
Ensure that MongoDB is running before attempting to start the `mongo` shell.

To start the `mongo` shell and connect to your MongoDB instance running on **localhost** with **default port**:

1. At a prompt in a terminal window (or a command prompt for Windows), go to your `<mongodbinstallation dir>`:

```
2. cd <mongodb installation dir>
```

3. Type `./bin/mongo` to start `mongo`:

```
4. ./bin/mongo
```

   If you have added the `<mongodb installation dir>/bin` to the `PATH` environment variable, you can just type `mongo` instead of `./bin/mongo`.

Options

When you run `mongo` without any arguments, the `mongo` shell will attempt to connect to the MongoDB instance running on the `localhost` interface on port `27017`. To specify a different host or port number, as well as other options, see examples of starting up mongo and mongo reference which provides details on the available options.

`.mongorc.js` File

When starting, `mongo` checks the user's `HOME` directory for a JavaScript file named .mongorc.js. If found,`mongo` interprets the content of `.mongorc.js` before displaying the prompt for the first time. If you use the shell to evaluate a JavaScript file or expression, either by using the *--eval* option on the command line or by specifying a .js file to mongo, `mongo` will read the `.mongorc.js` file *after* the JavaScript has finished processing. You can prevent `.mongorc.js` from being loaded by using the --norc option.

Working with the `mongo` Shell

To display the database you are using, type `db`:

```
db
```

The operation should return `test`, which is the default database. To switch databases, issue the `use <db>`helper, as in the following example:

```
use <database>
```

To list the available databases, use the helper `show dbs`. See also `db.getSiblingDB()` method to access a different database from the current database without switching your current database context (i.e.`db`).

You can switch to non-existing databases. When you first store data in the database, such as by creating a collection, MongoDB creates the database. For example, the following creates both the database`myNewDatabase` and the collection `myCollection` during the `insert()` operation:

```
use myNewDatabase


db.myCollection.insert( { x: 1 } );
```

The `db.myCollection.insert()` is one of the methods available in the mongo shell

- `db` refers to the current database.
- `myCollection` is the name of the collection.

If the `mongo` shell does not accept the name of the collection, for instance if the name contains a space, hyphen, or starts with a number, you can use an alternate syntax to refer to the collection, as in the following:

```
db["3test"].find()




db.getCollection("3test").find()
```

For more documentation of basic MongoDB operations in the `mongo` shell, see:

- Getting Started Guide
- Insert Documents
- Query Documents
- Modify Documents
- Remove Documents
- mongo Shell Methods

Format Printed Results

The `db.collection.find()` method returns a cursor to the results; however, in the `mongo` shell, if the returned cursor is not assigned to a variable using the `var` keyword, then the cursor is automatically iterated up to 20 times to print up to the first 20 documents that match the query. The `mongo` shell will prompt `Typeit` to iterate another 20 times.

To format the printed result, you can add the `.pretty()` to the operation, as in the following:

```
db.myCollection.find().pretty()
```

In addition, you can use the following explicit print methods in the `mongo` shell:

- `print()` to print without formatting
- `print(tojson(<obj>))` to print with JSON formatting and equivalent to `printjson()`
- `printjson()` to print with JSON formatting and equivalent to `print(tojson(<obj>))`

For more information and examples on cursor handling in the `mongo` shell, see Cursors. See also Cursor Help for list of cursor help in the `mongo` shell.

Multi-line Operations in the `mongo` Shell

If you end a line with an open parenthesis (`'('`), an open brace (`'{'`), or an open bracket (`'['`), then the subsequent lines start with ellipsis (`"..."`) until you enter the corresponding closing parenthesis (`')'`), the closing brace (`'}'`) or the closing bracket (`']'`). The `mongo` shell waits for the closing parenthesis, closing brace, or the closing bracket before evaluating the code, as in the following example:

```
> if ( x > 0 ) {

... count++;
```

```
... print (x);

... }
```

You can exit the line continuation mode if you enter two blank lines, as in the following example:

```
> if (x > 0

...

...

>
```

Tab Completion and Other Keyboard Shortcuts

The `mongo` shell supports keyboard shortcuts. For example,

- Use the up/down arrow keys to scroll through command history.
  See .dbshell documentation for more information on the `.dbshell` file.
- Use `<Tab>` to autocomplete or to list the completion possibilities, as in the following example which uses `<Tab>` to complete the method name starting with the letter `'c'`:

```
- db.myCollection.c<Tab>
```

  Because there are many collection methods starting with the letter `'c'`, the `<Tab>` will list the various methods that start with `'c'`.

For a full list of the shortcuts, see Shell Keyboard Shortcuts

Exit the Shell

To exit the shell, type `quit()` or use the `<Ctrl-c>` shortcut.