




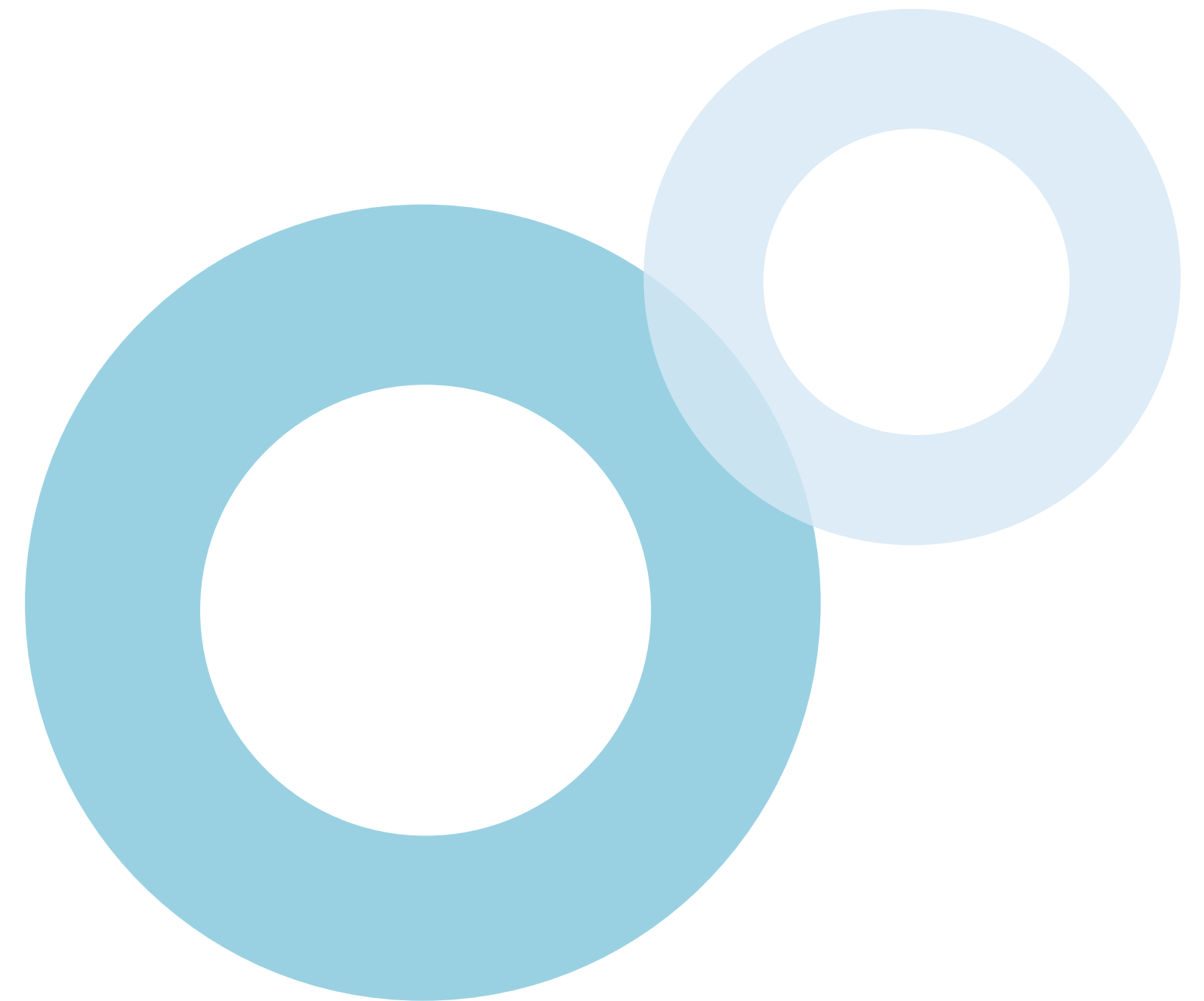


---

## After studying this topic, you should be able to:

-  Identify the main use cases of Node.js CLI commands.
-  Identify how to execute a Node.js application using the command line interface.
-  Identify how to enter the REPL mode for executing JavaScript code.
-  Identify some of the most important npm commands and additional CLI options.
-  Identify the commands that can be used for debugging a Node.js application.

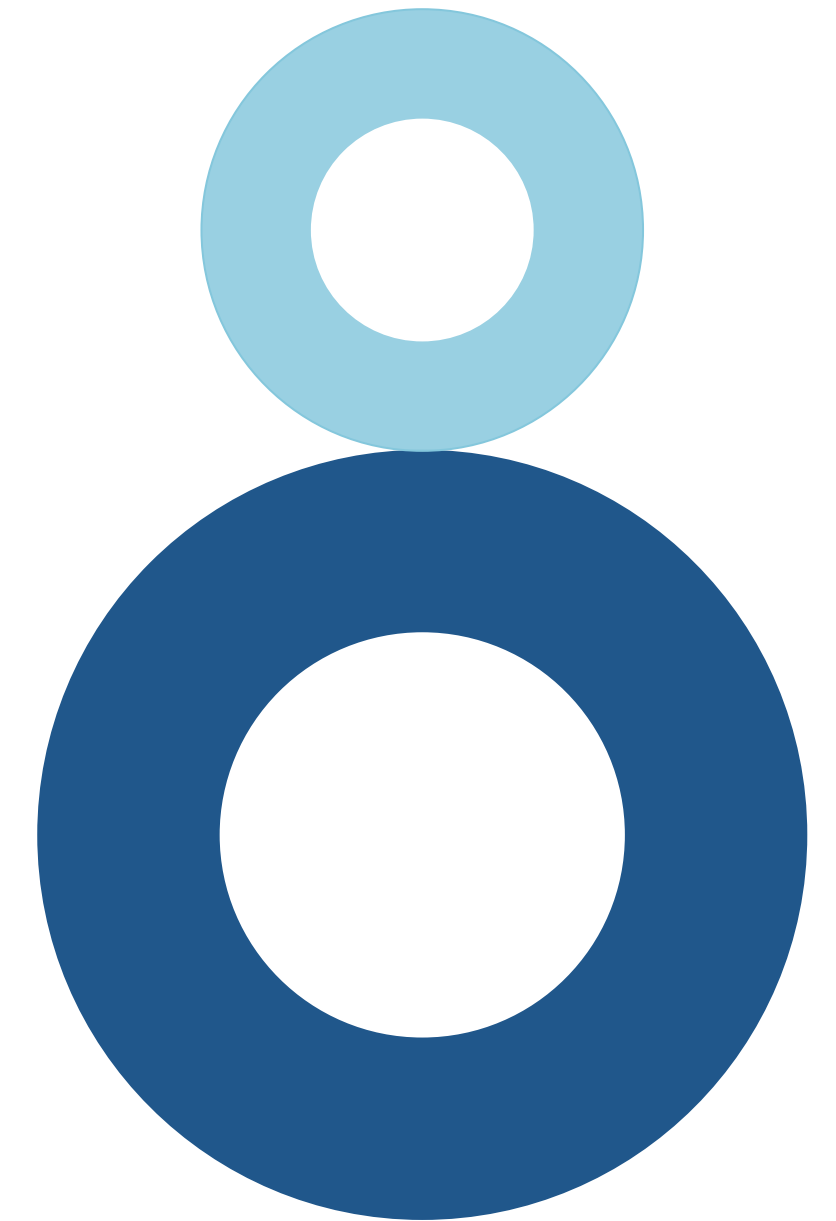


# Introduction

---

Node.js offers various command line options. These include helpful **runtime options**, **debugging commands**, and multiple ways of **executing scripts**. The **node script.js** command can be used to run an application named script.js. The node command can be used to execute JavaScript code in **REPL mode**.

It is also possible to pass command line arguments when executing a Node.js application. Various npm (node package manager) commands are also available, These include **npm init**, **npm install**, **npm update**, and **npm run**. When using the npm install command, it is necessary to consider whether the package should be installed as a development dependency. If so, the **--save-dev** flag is required. Node.js CLI can also be used for the purpose of debugging code. The **node inspect script.js** command can be used to launch the debugger.



# Overview

---

## Node.js CLI

Node.js offers various **command-line options**, which include helpful runtime options, debugging commands, and multiple ways of executing scripts.

## Node Command

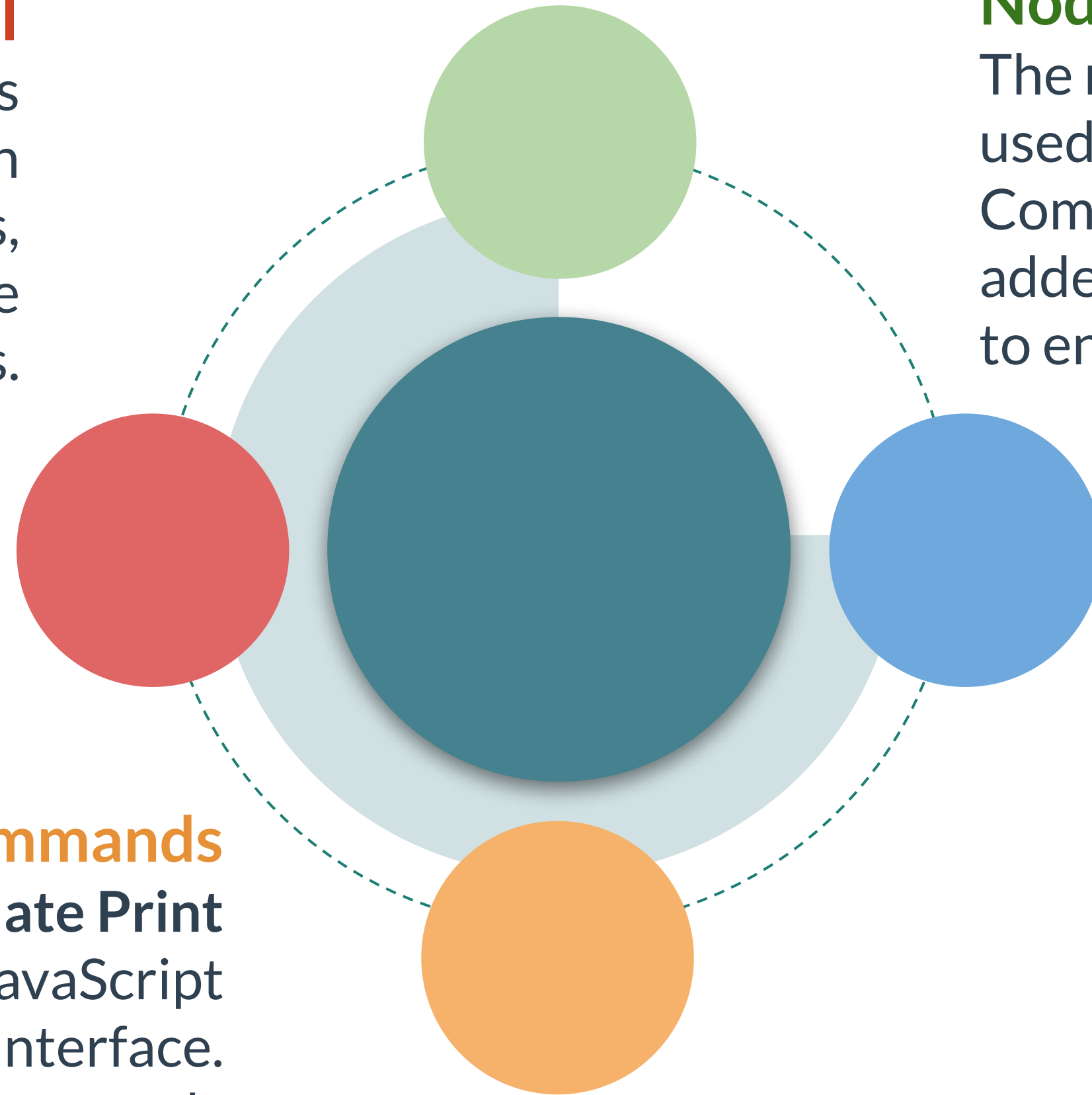
The **node script.js** command can be used to run an application. Command-line parameters can be added. The **node** command can be used to enter **REPL** mode.

## Debugging Commands

Commands like **node inspect script.js** can be used for debugging JavaScript code in a Node.js application. It launches the debugger.

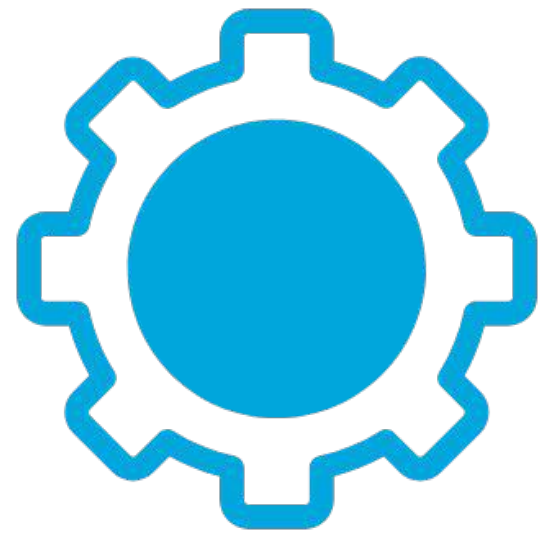
## REPL Commands

**REPL** stands for **Read Evaluate Print Loop**. It allows executing JavaScript code via the command-line interface. It supports some special commands like **.break** and **.clear**.



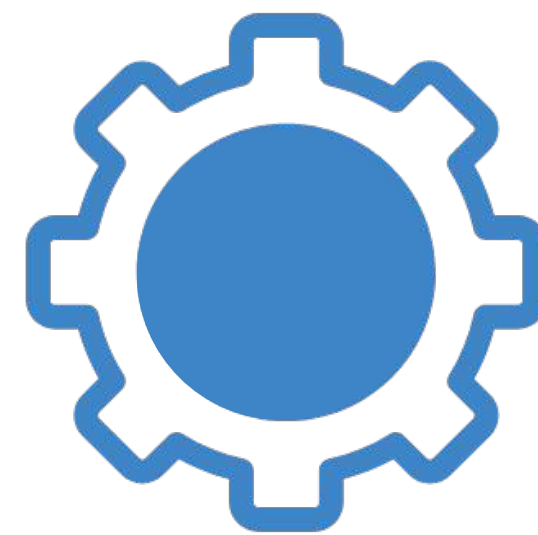
# Node.js CLI

There are various command line options in Node.js. They include helpful **run-time options**, built-in **debugging**, and multiple ways of **executing scripts**.



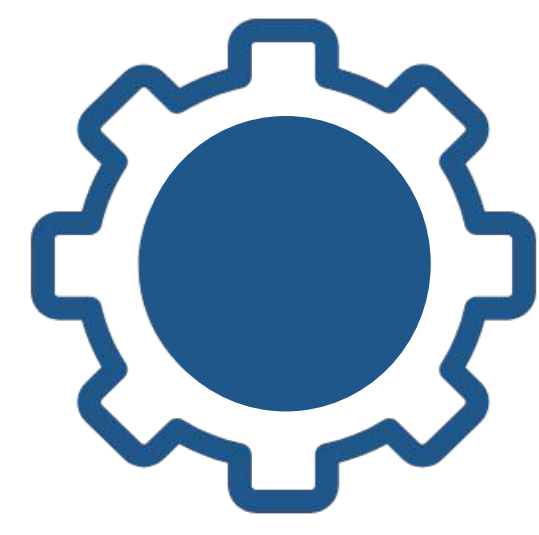
## RUN-TIME OPTIONS

Helpful run-time options include commands such as **node -r**, which preloads the specified module at startup.



## DEBUGGING

Command line options are available for debugging an application. These include the **node inspect** command, which can be used to debug the specified script.



## SCRIPTS

The **node script.js** command can be used to run a Node.js program. In this command, **script.js** is the name of the main Node.js application file.

# Node.js CLI

The **node script.js** command can be used to run a Node.js application named **script.js**. It must be executed from the **same directory** that contains the script.js file.

TERMINAL PROBLEMS OUTPUT DEBUG CONSOLE

1: powershell

Windows PowerShell

Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell <https://aka.ms/pscore6>

PS C:\dev\node-app> **node** script.js

The **node script.js** command can be used to run a Node.js application.



# Node.js REPL

The **node** command can be used to execute JavaScript code in **REPL** mode. REPL is a programming language environment that stands for **Read Evaluate Print Loop**. It takes a single expression as user input and returns the output.

```
PS C:\dev\node-app> node
Welcome to Node.js v14.5.0.
Type ".help" for more information.
> [1, 2, 3].map(e => e*2);
[ 2, 4, 6 ]
```

The **node** command can be used to enter the **REPL** mode to execute JavaScript code.

# Node.js REPL Commands

---

The **REPL** mode in Node.js supports some special commands, all of which start with a dot.

## **.break**

Aborts further input or processing of an expression. It is the same as using Ctrl + C.

## **.clear**

Resets the REPL context to an empty object and clears any multi-line expression being input.

## **.exit**

Closes the I/O stream, which causes the REPL to exit.

## **.help**

Shows a list of REPL commands.

# Node.js REPL Commands

---

The **REPL** mode in Node.js supports some special commands, all of which start with a dot.

## **.save**

Saves the current REPL session to the specified file.

## **.load**

Loads the specified file into the current REPL session.

## **.editor**

Used to enter the editor mode. Ctrl + D is used to finish, and Ctrl + C is used to cancel.



# Command Line Arguments

When executing a Node.js application, **arguments** can be provided. An argument can be **standalone** or have a **key** and a **value**. The **'argv'** property of the **'process'** object can be accessed to obtain all the command line arguments.

```
// The following command can be used to provide an argument when executing a Node.js application.
```

```
PS C:\dev\node-app> node script.js Jon
```

```
/* The 'argv' property of the 'process' object is an array that can be accessed to retrieve all the command line arguments. They can be accessed starting from the third position.*/
```

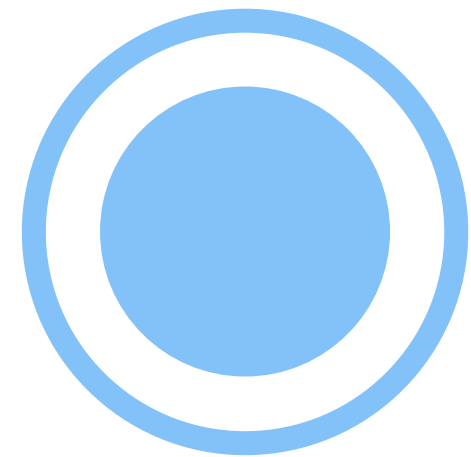
```
console.log(`The employee name is ${process.argv[2]}.`);
```

Output

```
PS C:\dev\node-app> node script.js Jon
The employee name is Jon.
```

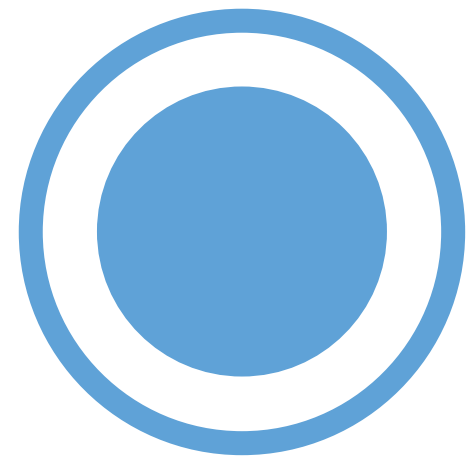
# Environment Variables

**Environment variables** allow a Node.js app to behave differently based on the environment. The **'env'** property of the **'process'** object is used to access them.



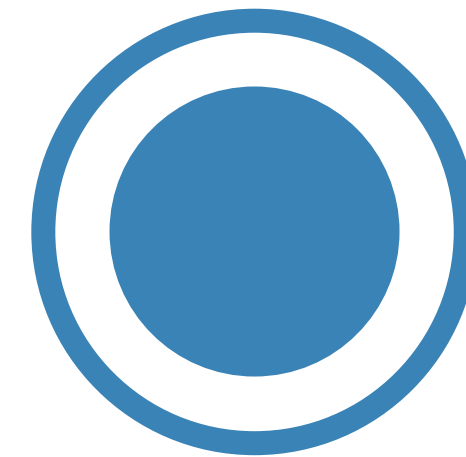
## NODE\_ENV

The environment variable named **NODE\_ENV** is typically used to set the environment, such as **production** or **development**, and can be used to enable or disable features like caching.



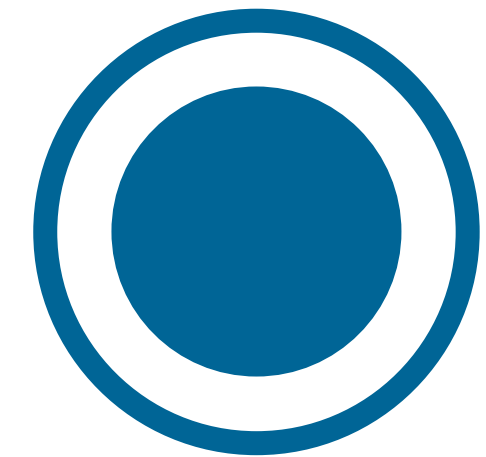
## PORT

The **PORT** variable is used to specify the port on which the server should listen for incoming requests.



## NODE\_DEBUG

The **NODE\_DEBUG** variable is used to list the internal modules that the application developer would like to debug.



## DEBUG

The **DEBUG** variable is used to list the **public modules**, such as npm modules, that need to be debugged. It can be set to **\*** to specify all public modules.

# Using Environment Variables

**Environment variables** can be set using the command line and used in code by accessing **process.env**.

```
/* The following file uses two environment variables named PORT and NODE_ENV.*/
const ENVIRONMENT = process.env.NODE_ENV;
const PORT = process.env.PORT;

const app = require('http').createServer((req, res) => res.end('Success!'));

app.listen(PORT, () => {
  console.log(`The environment is ${ENVIRONMENT}. The server is listening on port ${PORT}.`);
});

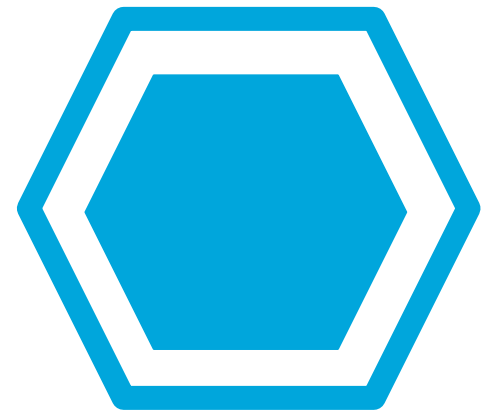
/* The following commands can be used in Windows to set the environment variables named PORT and NODE_ENV prior to running the
Node.js app */.
```

```
PS C:\dev\node-app> $env:PORT=3000
PS C:\dev\node-app> $env:NODE_ENV='development'
PS C:\dev\node-app> node script.js
The environment is development. The server is listening on port 3000.
```

# NPM Commands

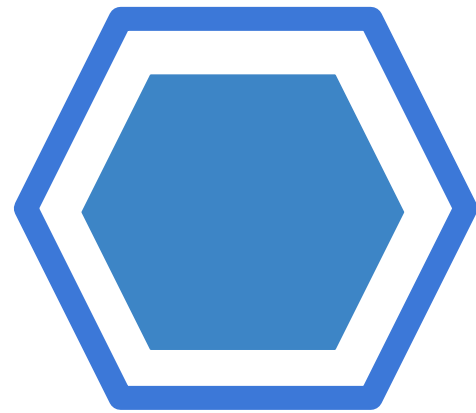
---

There are various **CLI commands** available for npm. Some of the most common commands are as follows.



## npm init

This command can be used to **create a package.json file** in the current project directory.



## npm install

This command can be used to **install the specified package or all dependencies** in package.json.



## npm update

This command can be used to **update the specified package or all dependencies** in package.json.



## npm run

This command can be used to **run a command** from the **'scripts'** object of the package.



## npm start

This command can be used to **run a command** specified in the **'start'** property of the package's **'scripts'** object.



# NPM Commands

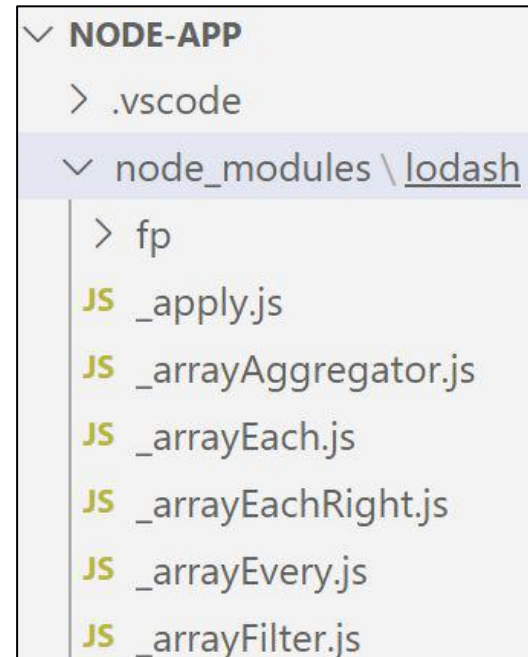
When using the **npm install** command, the **-g** or **--global** flag can be appended to the command in order to install the package globally. The **--save-dev** flag can be used to install the package under **devDependencies** in **package.json**.

```
// The following command can be used to install an npm package to a global location on the computer.
```

```
PS C:\dev\node-app> npm install -g lodash
```

```
/* When a package is installed locally, it appears in the node_modules subfolder of the project directory and also under dependencies in the package.json file. */
```

```
PS C:\dev\node-app> npm install lodash
```



```
▼ NODE-APP
  > .vscode
  ▼ node_modules \ lodash
    > fp
    JS _apply.js
    JS _arrayAggregator.js
    JS _arrayEach.js
    JS _arrayEachRight.js
    JS _arrayEvery.js
    JS _arrayFilter.js
```



```
{ } package.json > ...
1  {
2    "name": "node-app",
3    "version": "1.0.0",
4    "description": "",
5    "main": "script.js",
6    "scripts": {
7      "test": "echo \"Error: no test specified\" && exit 1"
8    },
9    "author": "",
10   "license": "ISC",
11   "dependencies": {
12     "lodash": "^4.17.20"
13   }
14 }
```

# NPM Commands

The **npm run** command **runs the specified command** from the **'scripts'** object of the package. If no command has been specified, it **lists the available commands**.

// The 'npm run' command can be used to run a script named 'build' that has been specified in the package.json file.

```
PS C:\dev\node-app> npm run build
```

```
> node-app@1.0.0 build C:\dev\node-app
> webpack
```

```
[webpack-cli] Compilation finished
asset main.js 1.27 KiB [compared for emit] [minimized] (name: main)
runtime modules 657 bytes 3 modules
cacheable modules 1.31 KiB
  ./index.js 186 bytes [built] [code generated]
  ./helper.js 1.13 KiB [built] [code generated]
webpack 5.4.0 compiled successfully in 328 ms
```

```
package.json X
package.json > ...
1  {
2    "name": "node-app",
3    "version": "1.0.0",
4    "description": "Web application",
5    "main": "script.js",
6    "scripts": {
7      "test": "echo \"Error: no test specified\" && exit 1",
8      "build": "webpack"
9    },
10   "author": "",
11   "license": "ISC",
12   "repository": {
13     "github": "localhost:3000"
14   },
15   "dependencies": {
16     "lodash": "^3.2.0"
17   },
18   "devDependencies": {
19     "webpack": "^5.4.0",
20     "webpack-cli": "^4.2.0"
21   }
22 }
```



# NPM Commands

The **npm start** command **runs the command** specified in the **'start'** property of the **'scripts'** object. If no command has been specified, it runs **node script.js**, where **script.js** is the main JavaScript file.

// The 'npm start' command can be used to run a script specified in the 'start' property of the 'scripts' object.

```
PS C:\dev\node-app> npm start

> node-app@1.0.0 start C:\dev\node-app
> node script.js

The customer's email address is valid.
```

```
package.json X
package.json > ...
1  {
2    "name": "node-app",
3    "version": "1.0.0",
4    "description": "Web application",
5    "main": "script.js",
6    "scripts": {
7      "start": "node script.js"
8    },
9    "author": "",
10   "license": "ISC",
11   "repository": {
12     "github": "localhost:3000"
13   },
14   "dependencies": {
15     "validator": "^13.1.17"
16   },
17   "devDependencies": {
18     "webpack": "^5.4.0",
19     "webpack-cli": "^4.2.0"
20   }
21 }
```

# Debugging Commands

Command-line options are available for debugging Node.js code. The **node inspect** command can be used to launch the debugger. Commands can be executed to **set breakpoints, step through code, watch expressions**, etc.

// This example shows how to launch the debugger in Node.js and set a breakpoint.

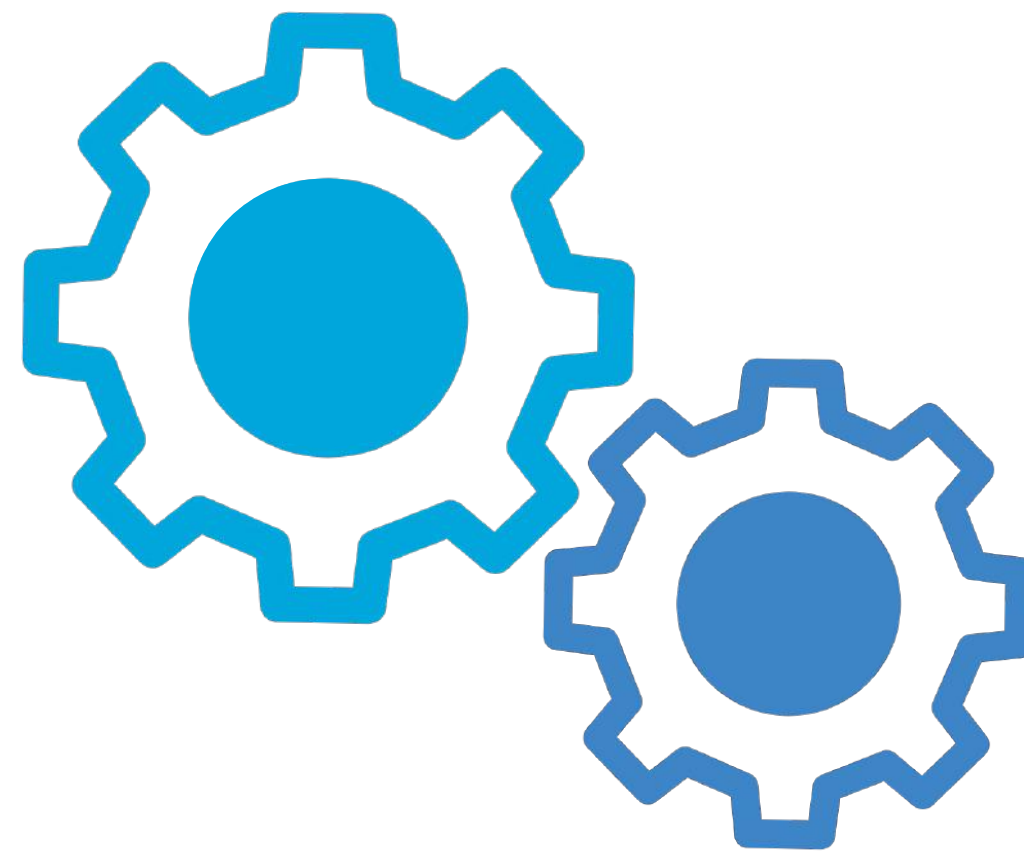
```
PS C:\dev\node-app> node inspect script.js
< Debugger listening o
< n ws://127.0.0.1:9229/2fdb3e25-f33b-4178-bfa1-e06c9d2955a8
< For help, see: https://nodejs.org/en/docs/inspector
< Debugger attached.
Break on start in script.js:1
> 1 const https = require('https');
  2 const fs = require('fs');
  3 const options = {
debug> setBreakpoint(8)
  3 const options = {
  4   hostname: 'cosmicsolutionsemployees.free.beeceptor.com',
  5   path: '/' + 713,
  6   method: 'GET'
  7 };
> 8 const request = https.request(options, response => {
```

# Breakpoints and Watchers

Commands are available to **set** and **clear breakpoints**. It is also possible to use a command to **watch expression** and **variable values** while debugging.

## BREAKPOINTS

The **setBreakpoint()** or **sb()** command can be used to set a breakpoint on the **current line**, a **specific line**, the **first line**, or the **first statement of a function's body**. A **conditional breakpoint** can also be set. The **clearBreakpoint()** or **cb()** command is used to clear a breakpoint.



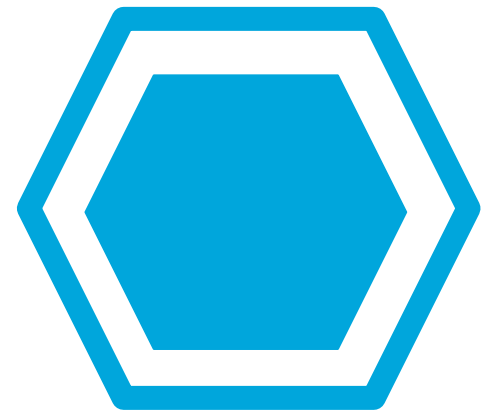
## WATCHERS

The **watch('expression')** command can be used to watch an expression. The **watchers** command can be used to print the active watchers. The **unwatch('expression')** command is used to remove an active watcher.



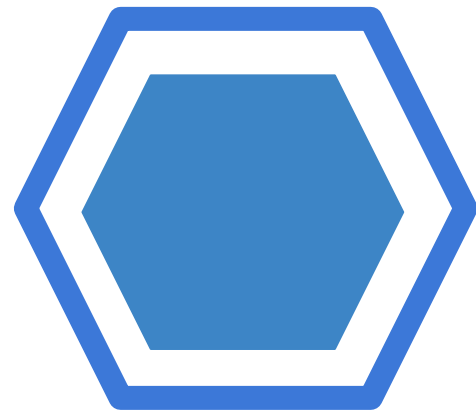
# Stepping Commands

Commands are available for **stepping through code** while debugging.



**cont(c)**

The **cont** or **c** command can be used to continue execution of code while debugging.



**next(n)**

The **next** or **n** command can be used to step to the next line of code while debugging.



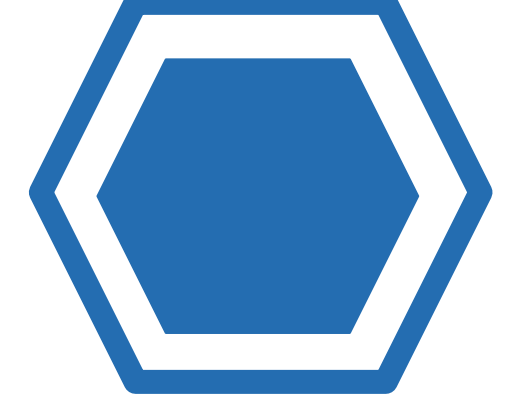
**step(s)**

The **step** or **s** command can be used to step into a function while debugging.



**out(o)**

The **out** or **o** command can be used to step out of a function while debugging.



**pause**

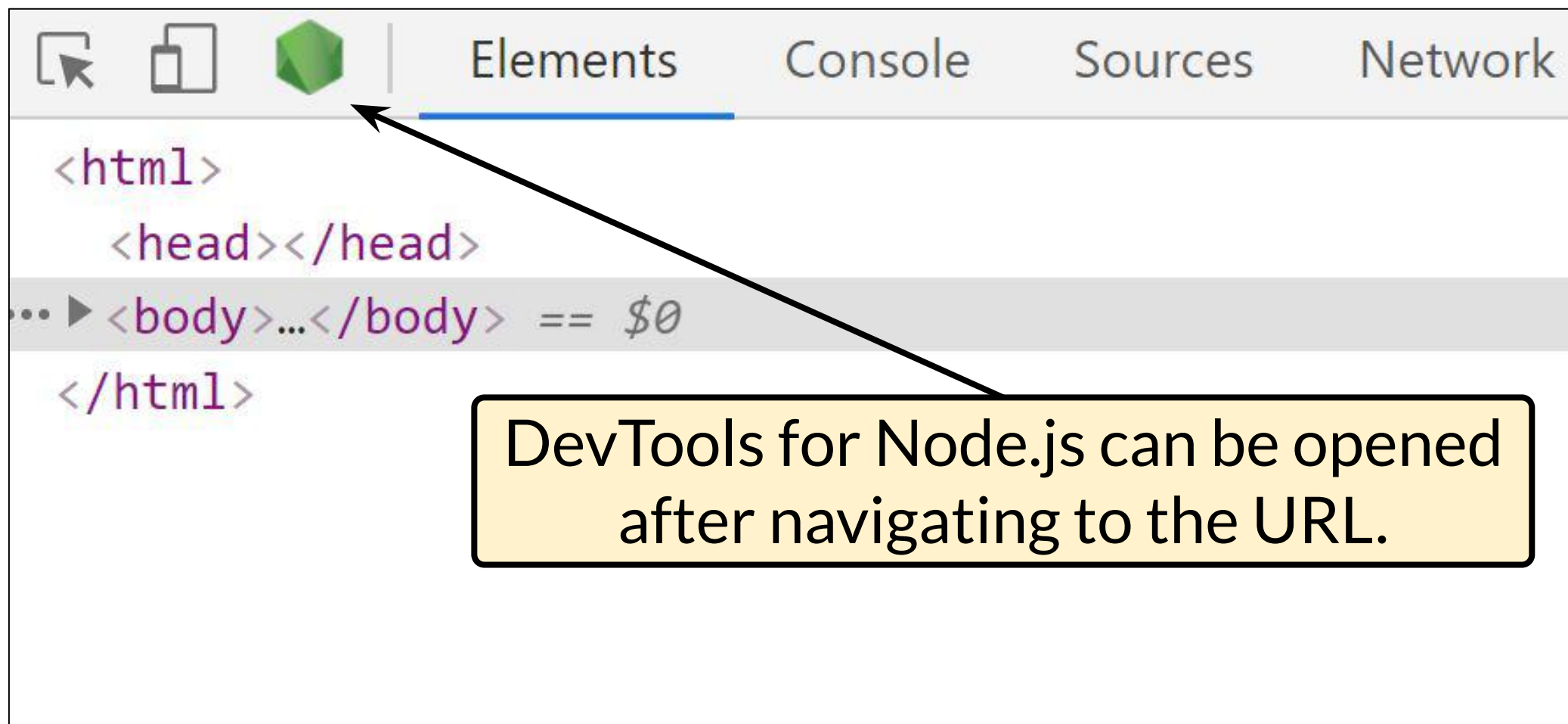
The **pause** command allows pausing the running code while debugging.

# Node Inspector and DevTools

For debugging and profiling, **Chrome DevTools** can be attached to a Node.js instance. The **V8 Inspector** can be enabled by using the **--inspect** flag when starting an app. The **--inspect-brk** flag can be used instead to break on the first line of application code.

// This example shows how to use the --inspect-brk flag to enable the V8 Inspector.

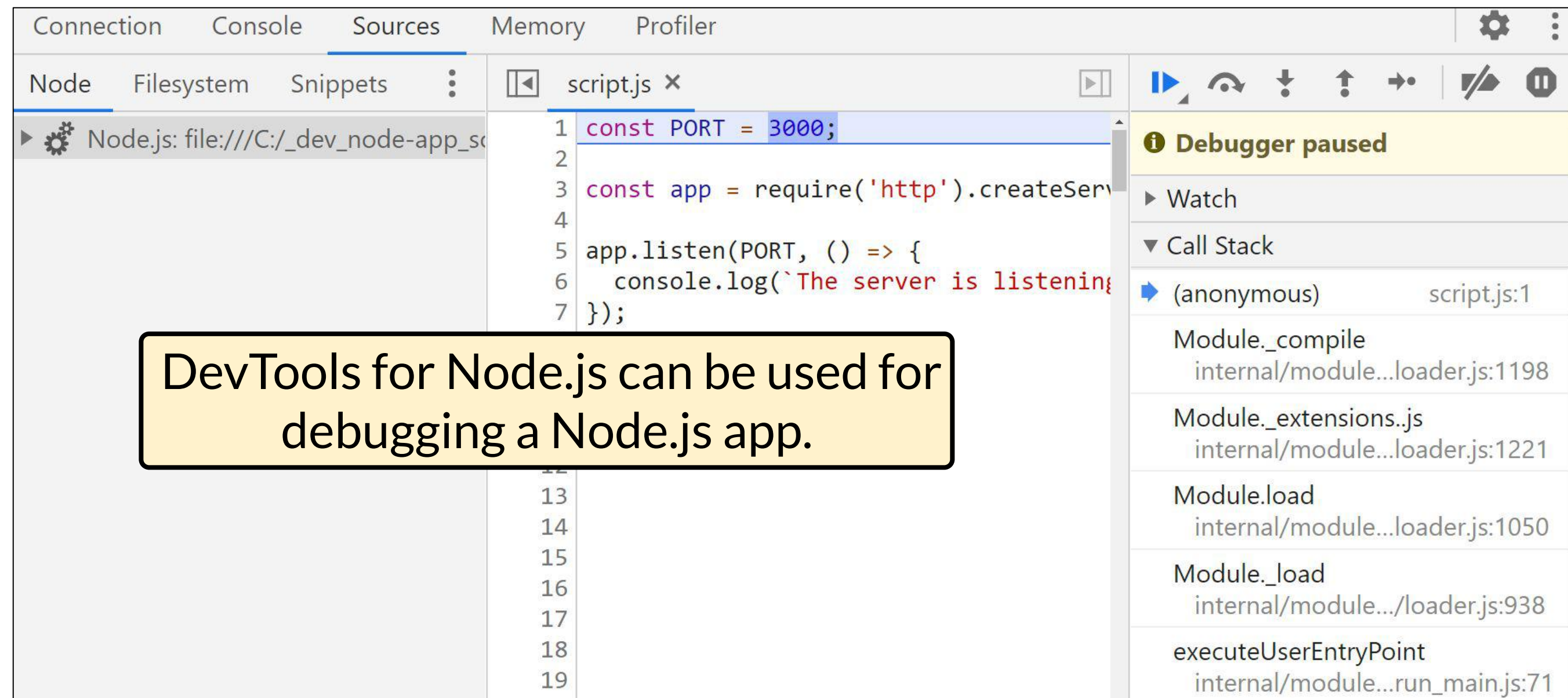
```
PS C:\dev\node-app> node --inspect-brk script.js
Debugger listening on ws://127.0.0.1:9229/97a4c693-4eb6-4d02-ac20-a82cc7e3339d
For help, see: https://nodejs.org/en/docs/inspector
```



# Node Inspector and DevTools

For debugging and profiling, **Chrome DevTools** can be attached to a Node.js instance. The **V8 Inspector** can be enabled by using the **--inspect** flag when starting an app. The **--inspect-brk** flag can be used instead to break on the first line of application code.

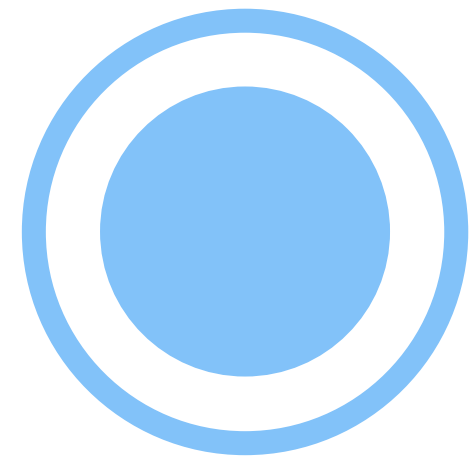
// The screenshot below shows DevTools for Node.js.





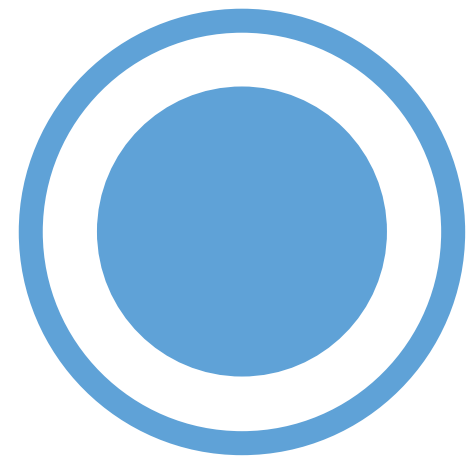
# Additional CLI Options

Some additional Node.js CLI commands are as follows.



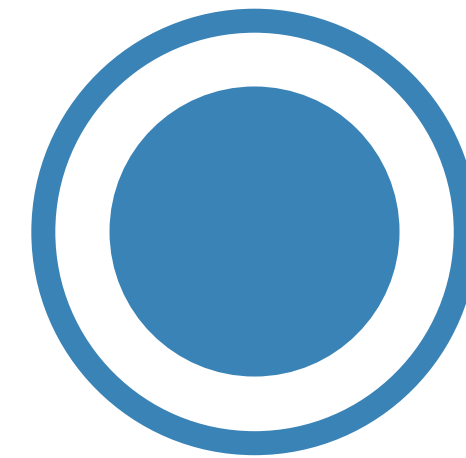
## **node --version**

This command can be used to get the version of Node.js.



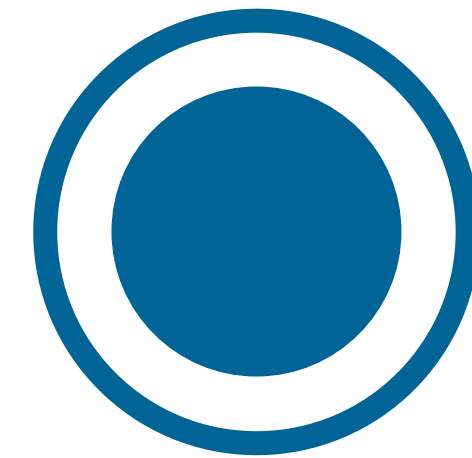
## **node --eval**

This command can be used to evaluate the specified JavaScript code.



## **node --check**

This command can be used to check the syntax of the code without execution.



## **node --help**

This command can be used to print the Node command-line options.

# Learn More

---



[Node.js REPL](#)



[Node.js Command Line Options](#)

# Scenario and Solution

# Scenario

A developer of Cosmic Software Solutions is building a Node.js web application for which she requires certain libraries. Some libraries, such as lodash and express, are required for running the application. On the other hand, there are libraries like jest and webpack which are required for the purpose of development only. She needs to install these libraries while making sure that only the libraries that are required for running the application can be easily installed in production later.

Furthermore, once lodash is installed, she would like to quickly evaluate the following JavaScript code using the command line to check if the library could be used to meet a specific requirement.

```
const data1 = ['Jon Williams', 'Jon Williams', 'Olivia Smith', 'Ashley James'];
const data2 = ['Jen Kirkman', 'Jane Liu', 'Jon Williams', 'Ashley James'];
const deduped_data1 = _.uniq(data1);
const commonData = _.intersection(deduped_data1, data2);
console.log(commonData);
```

It should result in the following output:

```
[ 'Jon Williams', 'Ashley James' ]
```

# Solution

The developer can use the `npm install <library>` command to install a library such as `lodash` as a dependency of the application. This adds the library to the 'dependencies' list in the `package.json` file. She can use the `npm install <library> --save-dev` command to install a library such as `jest` as a development dependency. This adds the library to the 'devDependencies' list in the `package.json` file.

```
npm install lodash
```

```
npm install webpack --save-dev
```

When only the dependencies that are required for running the application need to be installed, the `npm install --production` command can be used.

To evaluate the given JavaScript code, the developer can enter the REPL mode by executing the `node` command. However, in the REPL node, the following statement must be entered before the given code in order to require the `lodash` library before executing its methods.

```
const _ = require('lodash');
```