

Manual NodeJS

Antonio Guillén

Para empezar nos podremos instalar **NodeJS** desde la página oficial: <https://nodejs.org/en>

Una vez descargado, podremos ejecutarlo en cualquier momento mediante el comando: **node**

Si le indicas un fichero, lo ejecutará, por ejemplo: Tengo un **index.js** que tiene de contenido una función que suma dos números y la imprime por pantalla, pues al poner **node index.js** lanzará la función y mostrará el resultado.

Algo que tiene que quedar claro es que no existe el objeto de JS llamado **window**, por el contrario, existe el objeto llamado **global**

Esto significa que tenemos dos formas de llamar a el “mismo” objeto, para solucionarlo, existe otra variable llamada **globalThis**, la cual sirve para identificar el window y te da igual de donde venga (Si es navegador, servidor, app móvil, etc.)



NodeJS funciona con un **Patrón De Diseño** que utiliza sistemas de **módulos**, antes de nada, hay que saber que cada sistema de módulo se puede indicar con el tipo de extensión (En este tutorial vamos a utilizar CommonJS hasta que veamos como cambiar el **Packpage.json** sin necesidad de cambiar la extensión):

- **CommonJS (.cjs, y por defecto .js):** Este sistema es un módulo estándar que permite la modularidad en las

aplicaciones escritas en JavaScript. Permite dividir el código en módulos separados y organizarlos de manera lógica y reutilizable, es muy común, **no es la forma recomendable a día de hoy.**

No tiene capacidad para usar **Top-Level Await**

Funciona mediante **module.exports** y **require**. Por ejemplo:

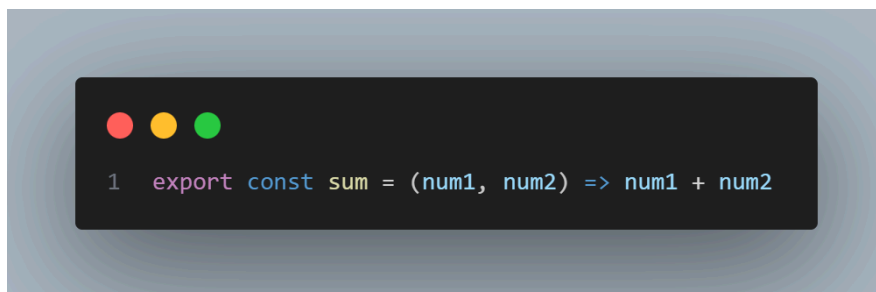
Tengo dos archivos, un index y un sum, en este último, tenemos la función a exportar, para ello tenemos dos maneras, la primera sirve para simplemente exportarla y ya, la segunda, lo exporta y obliga a en el archivo que se llame, hacerlo con el mismo nombre de la función

```
1  const sum = (num1, num2) => num1 + num2
2
3  //Opción 1 - No fuerza el nombre
4  module.exports = sum
5
6  //Opción 2 - Fuerza el nombre
7  module.exports = {
8    sum
9  }
```

Luego a la hora de llamarlo sería de la siguiente manera

```
1  //Opción 1 - Importamos con cualquier nombre
2  const sumFunction = require('./sum')
3
4  //Opción 2 - Importamos obligatoriamente haciendo
   una desestructuración del objeto
5  const { sum } = require('./sum')
6
7  console.log(sumFunction(4, 9)) // 13
8  console.log(sum(4, 9)) // 13
```

- **ES Modules (.mjs):** El sistema de módulos ES (ECMAScript) en Node.js proporciona una forma moderna de modularizar el código en JavaScript, basada en las especificaciones de ES6 (ECMAScript 2015) y posteriores. A diferencia del sistema de CommonJS, los módulos ES son un estándar oficial de JavaScript, lo que significa que están integrados directamente en el lenguaje y no requieren bibliotecas externas como CommonJS. Tiene capacidad para usar **Top-Level Await**. Funciona mediante **export**, **import** y **from**. Con el mismo ejemplo de antes, la utilización de este sistema de módulos se vería de la siguiente manera:

A screenshot of a code editor with a dark background and light blue, green, and yellow accents. It shows a single line of JavaScript code: `1 export const sum = (num1, num2) => num1 + num2`.

```
1 export const sum = (num1, num2) => num1 + num2
```

Para llamarlo sería así (Si se usa esta opción es obligatorio poner la extensión a la hora de llamar al módulo):

A screenshot of a code editor with a dark background and light blue, green, and yellow accents. It shows three lines of JavaScript code: `1 import { sum } from './sum.mjs'`, `2` (empty line), and `3 console.log(sum(5, 9)) // 14`.

```
1 import { sum } from './sum.mjs'
2
3 console.log(sum(5, 9)) // 14
```

Una vez visto los módulos, veremos más cosas.

Para llamar a los módulos del sistema como pueden ser: **os**, **path**, **stream**, **process**, **etc.** se recomienda hacerlo de la siguiente manera: **node:<nombre-módulo>**

Hay varios módulos que son importantes, entre los que se encuentran:

- **os**: Es el módulo del Sistema Operativo, con el que puedes saber cualquier especificación de este
- **fs**: Es el módulo necesario para interactuar con los archivos del sistema, las funciones principales que destacan de este son:
 - **isFile()**: Sirve para saber si es un fichero
 - **isDirectory()**: Sirve para saber si es un directorio
 - **isSymbolicLynk()**: Sirve para saber si es un enlace simbólico
 - **readdir(folder: string)**: Sirve para listar el directorio indicado
 - **readFileSync(file: string, coded: string)**: Sirve para leer un fichero de forma síncrona
 - **readFile(file: string, [coded: string], function: (err, text) => {})**
- **fs/promises**: Es el mismo módulo que el **fs** pero en la parte asíncrona, en vez de utilizar callback, utiliza promesas
- **util**: Es un módulo de utilidades, es bueno saberlo porque por ejemplo tiene funciones para transformar un **readFile** de callback en uno de promesas (Solo en los módulos nativos que no tienen promesas nativas)

- **path**: Es un módulo para trabajar con las rutas del sistema, alguna de las funciones y propiedades que destacan son las siguientes:
 - **sep**: Sirve para ver el separador de carpetas según el SO
 - **join(params: string[])**: Sirve para unir rutas
 - **basename(route: string, [extension: string])**: Sirve para indicar el nombre del fichero que le pases en la ruta, si le pones el parámetro opcional, te quita la extensión que le indiques
 - **extname(route: string)**: Sirve para sacar la extensión de la ruta que le pases

Un objeto global a tener muy en cuenta es el llamado **process** el cual tiene funciones y propiedades que permiten interactuar con el entorno de ejecución de NodeJS e información del relacionada con el proceso actual, como por ejemplo alguno de los siguientes:

- **argv**: Sirve para ver los argumentos de entrada del proceso
- **exit(code: int)**: Sirve para cerrar el proceso y controlar la salida de este
- **on(event: string, function: () => {})**: Sirve para indicarle al proceso que realice una acción dependiendo de lo que le pases
- **cwd()**: Sirve para obtener el directorio donde se está ejecutando el proceso
- **env**: Sirve para ver las variables de entorno, si tengo una que se llame **scan**, pues accedería a ella así: **env.scan**

Vamos a explicar que es [NPM \(Node Packpage Manage\)](#)

Es un administrador de paquetes de NodeJs, es el registro de paquetes más grande de código que existe.

A la vez que es un registro de paquetes, también es una línea de comandos.

Aquí se van a mostrar varios comandos:

- **npm --version | npm -v:** Sirve para ver la versión de npm
- **npm init:** Sirve para crear un **package.json** en el cual van a estar todas las dependencias, la descripción, la licencia, el autor, etc. Es el primer comando que se debe de poner en un proyecto con NodeJS, al ponerlo, nos hará ciertas preguntas y nosotros deberemos de indicarle lo que queramos, si no añadimos nada, se pondrá el valor por defecto en cada campo:

- **package name:** Indicamos el nombre del proyecto
- **version:** Indicamos la versión del proyecto
- **description:** Indicamos la descripción del proyecto
- **entry point:** Es el archivo por el cual arrancaremos todo, por ejemplo si tenemos una API, este archivo sería el que al iniciarlo todo funcione correctamente
- **test command:** Sirve para especificar un comando que se ejecutará automáticamente cuando se ejecuten las pruebas para el proyecto. Este comando es típicamente utilizado por herramientas de prueba como **Mocha, Jest, Jasmine**, entre otras.

Cuando ejecutas el comando **npm test** en el directorio raíz de tu proyecto, npm buscará el valor del campo "test" en package.json y ejecutará el comando que encuentre allí. Este comando podría ser un script que inicie un marco de prueba específico o una tarea personalizada de prueba que hayas definido.

- **git repository:** Sirve para indicar el repositorio al cual está vinculado
- **keywords:** Sirve para indicar unas palabras clave para que otros desarrolladores encuentren el proyecto
- **author:** Sirve para indicar quién es el autor del proyecto
- **license:** Sirve para indicar el tipo de licencia del proyecto

```
PS C:\Users\Antonio Guillén\Documents\HispaSec\TutorialNodeJS> npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults.

See `npm help init` for definitive documentation on these fields
and exactly what they do.

Use `npm install <pkg>` afterwards to install a package and
save it as a dependency in the package.json file.

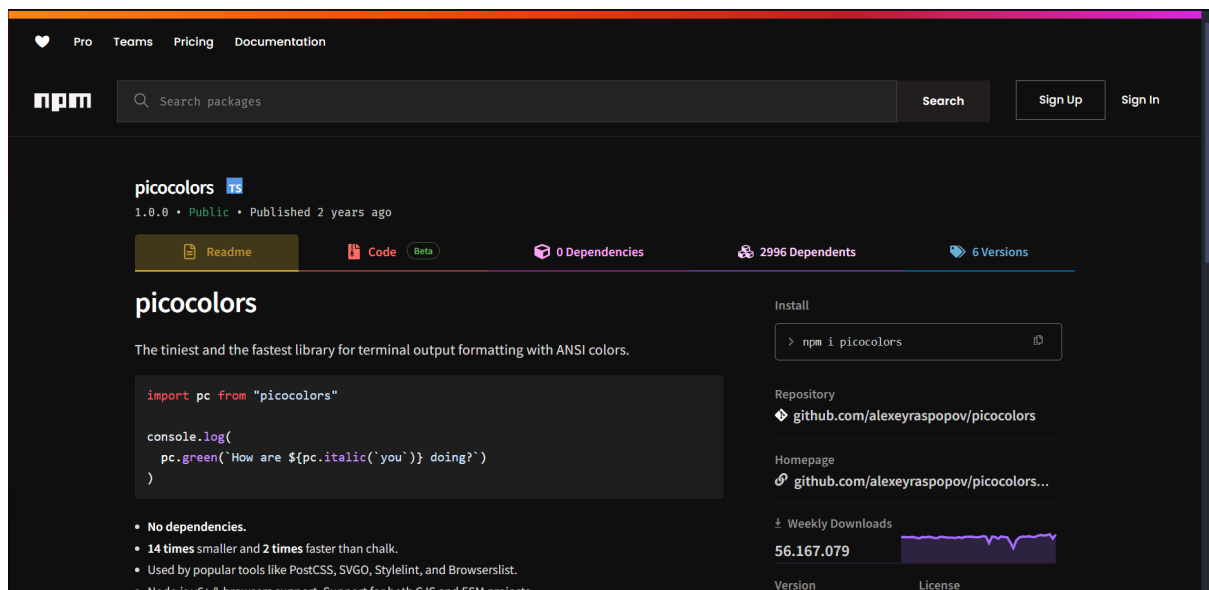
Press ^C at any time to quit.
package name: (tutorialnodejs)
version: (1.0.0)
description: El mejor curso de NodeJS
entry point: (index.js)
test command:
git repository: https://github.com/AntonioGuillen123/Node.js-Course-from-Scratch
keywords: curso, node, nodejs, js, npm
author: Antonio Guillén García
license: (ISC) MIT
About to write to C:\Users\Antonio Guillén\Documents\HispaSec\TutorialNodeJS\package.json:

{
  "name": "tutorialnodejs",
  "version": "1.0.0",
  "description": "El mejor curso de NodeJS",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "repository": {
    "type": "git",
    "url": "git+https://github.com/AntonioGuillen123/Node.js-Course-from-Scratch.git"
  },
  "keywords": [
    "curso",
    "node",
    "nodejs",
    "js",
    "npm"
  ],
  "author": "Antonio Guillén García",
  "license": "MIT",
  "bugs": {
    "url": "https://github.com/AntonioGuillen123/Node.js-Course-from-Scratch/issues"
  },
  "homepage": "https://github.com/AntonioGuillen123/Node.js-Course-from-Scratch#readme"
}

Is this OK? (yes)
```

Vamos a instalar una dependencia para que en la consola aparezcan colores, así que vamos a enseñar como se hace, para ello, tendremos que irnos a la página de npm y buscar la que queremos, una vez la tengamos, iremos a la consola y la instalaremos.

Cabe aclarar que si instalamos la dependencia con el parámetro **-D** se instalará en el entorno de desarrollo y no en el de producción.



```
PS C:\Users\Antonio Guillén\Documents\HispaSec\TutorialNodeJS> npm install picocolors
added 1 package, and audited 2 packages in 682ms
found 0 vulnerabilities
```

Una vez instalada nos saldrán líneas nuevas en el package.json.

Si le dejamos el símbolo de ^ lo que hará será que cuando haya una nueva versión se instalará automáticamente, esto no es muy recomendado de hacerlo.

```
"dependencies": {
  "picocolors": "^1.0.0"
}
```


Al instalar una dependencia se nos creará la carpeta llamada **node_modules** en la cual van a estar todo el código fuente de cada dependencia que vayamos instalando.

Ahora veremos un módulo muy importante necesario para crear un servidor web, estamos hablando del módulo **http**. Con este módulo podremos crear un servidor web de la siguiente manera:

- **createServer(function: (req: request, res: response) => {})**

Una vez creado el servidor, podremos indicarle que puerto escucha:

- **listen(port: int, function: () => {})**

Al recibir peticiones, podemos hacer varias cosas con ellas, tendremos nuestra **req** como una **request** y nuestro **res** como una **response** como por ejemplo:

- **req.on(event: string, function: (param: Depende del evento) => {})**
- **res.statusCode = <value>**: Sirve para indicar el código que tendrá la respuesta
- **res.setHeader(nameHeader: string, valueHeader)**: Sirve para indicar la cabecera de la respuesta
- **res.writeHead(code: int, header: object<header>)**: Combina la creación de código y cabecera de respuesta
- **res.end(value: string)**: Sirve para mandar una solicitud

Un comando muy importante de npm es el **npm run <command>** el cual sirve para lanzar comandos que hayamos puesto en la sección de scripts del package.json

Hasta aquí llegaría el manual general de node, hemos visto que es, lo básico para instalarlo y alguna funciones importantes que serán necesarias en algún futuro proyecto :)