

Optativa - 2º DAM

Introducción a Node.js

Índice

1. ¿QUÉ ES NODE.JS?
2. MODELO ASÍNCRONO Y NO BLOQUEANTE
3. CARACTERÍSTICAS Y VENTAJAS
4. INSTALACIÓN

Introducción a Node.js

¿Qué es Node.js?

¿Tiene JS?

¿Es backend?

¿Está relacionado con NoSQL?

¿Aplicación típica?

Introducción a Node.js

¿Qué es Node.js?

¿Tiene JS? **SI**

¿Es backend? **SI**

¿Está relacionado con NoSQL? **SI**

¿Aplicación típica? Podremos hacer un típico **chat** y muchas otras cosas (prácticamente de todo).

Introducción a Node.js

¿Qué es Node.js?

¿Es un framework JS?

¿Es una librería?

¿Qué tiene que ver V8? → ¿Sabes lo que es? ¡Búscalos con la palabra chrome!

¿PARA QUÉ SIRVE?

Introducción a Node.js

¿Qué es Node.js?

¿Es un framework JS?

¿Es una librería?

¿Qué tiene que ver [V8](#)? → Nos aprovechamos de la potencia de Google Chrome.

¿PARA QUÉ SIRVE?

Introducción a Node.js

¿Qué es Node.js?

Node.js® is a platform built on Chrome's JavaScript runtime for easily building fast, scalable network applications. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices.

Basada en la **máquina virtual de Google Chrome**, lo que permitirá construir aplicaciones de red **escalables** y **muy rápidas**.

Usa un modelo de **eventos no bloqueantes** y de entradas y salidas, haciéndolo eficiente y potente, perfecto para desarrollar aplicaciones **en tiempo real** y consumiéndose en **dispositivos distribuidos**.

Introducción a Node.js

¿Es Node.js un framework?



Introducción a Node.js

¿Es Node.js un framework?

¡¡NO ES UN FRAMEWORK!!

ES UN ENTORNO DE PROGRAMACIÓN.



Introducción a Node.js

¿Es Node.js un framework?

¡¡NO ES UN FRAMEWORK!!

ES UN ENTORNO DE PROGRAMACIÓN.

¿Cómo podemos saber que es un entorno? ¿Qué características tiene un entorno de programación?

Introducción a Node.js

Características de un lenguaje de programación:

1. Una gramática que define su **sintaxis**:
 - a. Tiene sus formas de crear objetos, crear variables...en PHP las variables van con \$, en JS con var/let....
2. Un **compilador** que lo interpreta y ejecuta:
 - a. En PHP, por ejemplo, tendremos Apache para procesar ese código.
3. Mecanismos para **interactuar con el exterior**:
 - a. Acceder a servidor FTP, correo electrónico...
4. **Librería estándar** :
 - a. Apache instala de manera predeterminada algunas librerías, como para enviar correos electrónicos, acceso a ficheros, generar gráficos...
5. **Utilidades**:
 - a. Intérprete, depurador, paquetes, etc.

Introducción a Node.js

Características de un lenguaje de programación:

1. V8 Posee:

- a. Una gramática que define su **sintaxis**
- b. Un **compilador** que lo interpreta y ejecuta

2. Node.js Posee:

- a. Mecanismos para **interactuar con el exterior**
- b. **Librería estándar**
- c. **Utilidades**

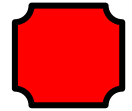
Resumen

1. Plataforma de desarrollo de software creada en 2009.
2. **JavaScript en el lado del servidor.**
3. Basado en la máquina virtual de Google Chrome (**V8** JavaScript Runtime).
4. V8 está escrito en C (es muy rápido).
5. **Multiplataforma** (Windows, Linux, Mac...).
6. Es una marca registrada y es Open Source.
7. **Asíncrono y orientado a eventos.**
8. **Ideal para aplicaciones que consumen datos en tiempo real (geolocalización?) y que se ejecutan a través de dispositivos distribuidos (pc, móvil...).**
9. Lo usan muchas empresas (Uber, PayPal, Linkedin, eBay...).

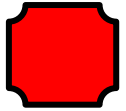
Índice

1. ¿QUÉ ES NODE.JS?
2. **MODELO ASÍNCRONO Y NO BLOQUEANTE**
3. CARACTERÍSTICAS Y VENTAJAS
4. INSTALACIÓN

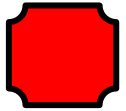
Conceptos necesarios



PARALELISMO Y CONCURRENCIA



BLOQUEANTE Y NO BLOQUEANTE



SÍNCRONO Y ASÍNCRONO

Paralelismo y Concurrency

1. Paralelismo: ¿¿¿???

2. Concurrency: ¿¿¿???

Paralelismo y Concurrency

1. Paralelismo: **Varios objetos** realizando una acción cada uno simultáneamente (síncrono).

a. **Síncrono** → Sincronización, **todos a la vez** →



2. Concurrency: **Un solo objeto**, con varias tareas “activas” entre las que va alternando (asíncrono).

b. **Asíncrono**: Es capaz de llevar el control de muchas tareas diferentes y es capaz de alternar entre ellas.

Paralelismo y Concurrency: Ejemplo

Paralelismo: **Varios objetos** realizando una acción cada uno **simultáneamente** (**síncrono**).

Síncrono → Sincronización, todos a la vez →

Concurrency: **Un solo objeto**, con varias tareas “activas” entre las que va **alternando** (**asíncrono**).

Asíncrono → Es capaz de llevar el control de muchas tareas diferentes y es capaz de alternar entre ellas.



Bart, Lisa, Homer e incluso Maggie son **síncronos**, ya que **de forma paralela** (a la vez) generan el caos en la casa Simpson.

Marge es **asíncrona**, ya que **de forma concurrente** (alternando) es capaz de solucionar los problemas de todos y mantener el control.

Paralelismo:

Paralelismo:
acción cada u

Síncrono → S

Concurrencia
tareas “activa
(**asíncrono**).

Asíncrono →
muchas tare
alternar entre



Maggie son
paralela (a
en la casa

e de forma
s capaz de
de todos y

NODE.JS ES COMO UN CAMARERO/A



NODE.JS ES COMO UN CAMARERO/A

Si vas a comer a un bar y te sientas en la mesa, si dicho bar tiene 12 mesas, puede que ese bar disponga de 3 camareros, los cuales se encargarán cada uno de 4 mesas.

Cuando te sientes, el camarero correspondiente te dará la carta y solicitará las bebidas.

Cuando te sirva las bebidas, te dará tiempo para pensar que vas a comer **y, mientras eliges, se va a atender a otra mesa.**

Esto es concurrencia, es capaz de llevar varios clientes a la vez y servirlos a todos por igual.



NODE.JS ES COMO UN CAMARERO/A

- 1. Para aprovecharlo, tiene que haber varios clientes.**
- 2. Un cliente no termina más rápido por asignarle un camarero exclusivo.**
- 3. Cada cliente termina a su ritmo.**

BLOQUEANTE VS NO BLOQUEANTE



BLOQUEANTE VS NO BLOQUEANTE

Las personas están en mesas independientes, cada mesa va a un ritmo diferente. Los camareros van a las mesas, atendiéndolas en función de la necesidad.

Supongamos que este restaurante **solo sirve con menú**. Los clientes son **bloqueantes**, porque comienzan con los entrantes, primer plato, segundo plato y postre.

Una **acción bloqueante** indica que hasta que la actual no finalice no se pasará a la siguiente. Hasta que no finalices con el primer plato no te servirán el segundo plato.



BLOQUEANTE VS NO BLOQUEANTE

En cambio, **los camareros son NO bloqueantes**, puesto que te trae el plato y no se queda ahí hasta que te lo terminas para traerte el siguiente plato, sino que pasa a atender a otras mesas, va a la cocina...etc.

Su flujo de programación es no bloqueante, **pudiendo ejecutar diferentes tareas de forma simultáneas y que cada una de ellas tarde el tiempo que necesite en finalizar, sin bloquear al resto.**

Mientras menos tareas simultaneas, mayor rendimiento.



RESUMEN

Los **clientes** son:

- **BLOQUEANTES** (van comiendo primer plato, segundo plato... NO TODO AL MISMO TIEMPO)
- **PARALELOS**
- **SÍNCRONOS**

Paralelismo: **Varios objetos** realizando una acción cada uno **simultáneamente** (síncrono).

Síncrono → Sincronización, todos a la vez



Los **camareros** son:

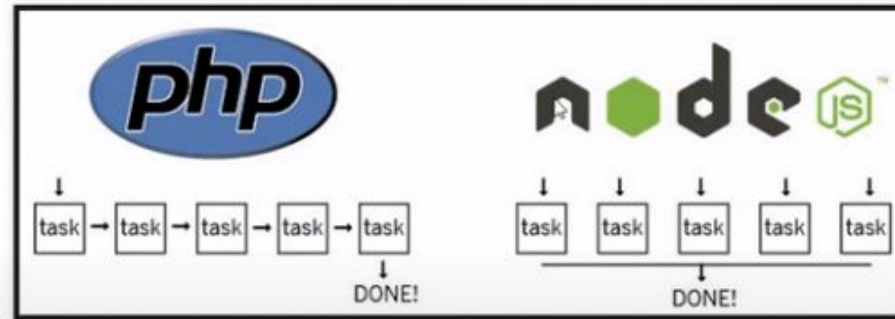
- **NO BLOQUEANTES** (atendiendo varias mesas al mismo tiempo)
- **CONCURRENTES**
- **ASÍNCRONOS**

Concurrencia: **Un solo objeto**, con varias tareas "activas" entre las que va **alternando** (asíncrono).

Asíncrono → Es capaz de llevar el control de muchas tareas diferentes y es capaz de alternar entre ellas.

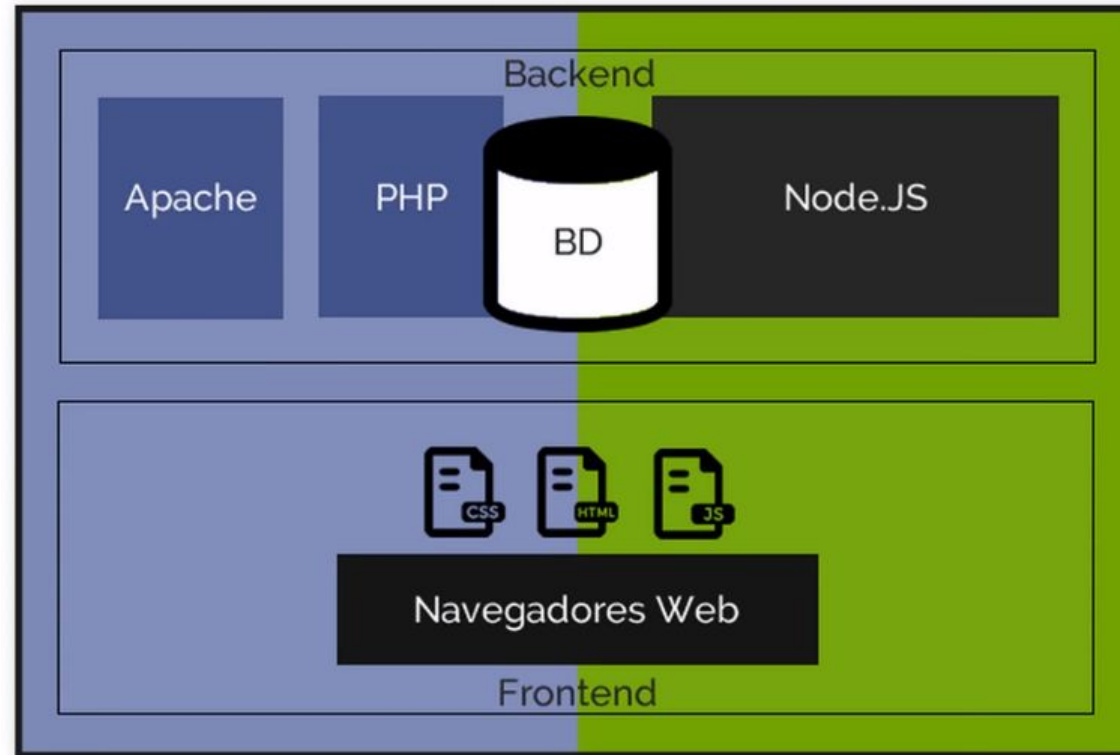


PHP VS NODE.JS



BLOCKING VS NON BLOCKING

- En **PHP** podemos lanzar tareas y, **en orden de lanzamiento**, se van ejecutando **una detrás de otra**. **Hasta que no finaliza una no pasa a la otra**.
- En **Node.js** podemos lanzar tantas como queramos y **cada una se ejecutará en el tiempo que necesite**, **sin bloquearse unas a otras**.



- En **PHP** necesitaremos el servidor **Apache** para compilar el código.
- En **Node.js** no necesitamos de ningún servidor, porque corre bajo V8 de Chrome.

EJEMPLO: BLOQUEANTE VS NO BLOQUEANTE

ABRIENDO UN FICHERO

BLOCKING

- Abrir el archivo
- Leer el archivo
- Imprimir contenido
- Hacer algo más

NON BLOCKING

- Abrir el archivo
- Leer el archivo
 - Cuando termines, imprimir contenido
- Hacer algo más

EJEMPLO: BLOQUEANTE VS NO BLOQUEANTE

PHP CÓDIGO BLOQUEANTE

1. Crearemos un nuevo fichero PHP que vamos a llamar “**1-blocking.php**”.
2. Lo pondremos en nuestra carpeta principal de Apache (**htdocs**).
3. Crearemos el “**archivo.txt**” en la misma carpeta.
4. Lo ejecutamos, desde localhost.

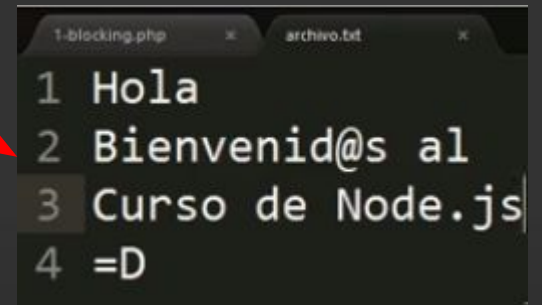
```
<?php
printf("Abriendo Archivo...<br>");
$fichero_url = fopen("archivo.txt", "r");

$texto = "";

while ($trozo = fgets($fichero_url)) {
    $texto .= $trozo;
}

printf($texto);

printf("<br>Haciendo otra cosa");
?>
```



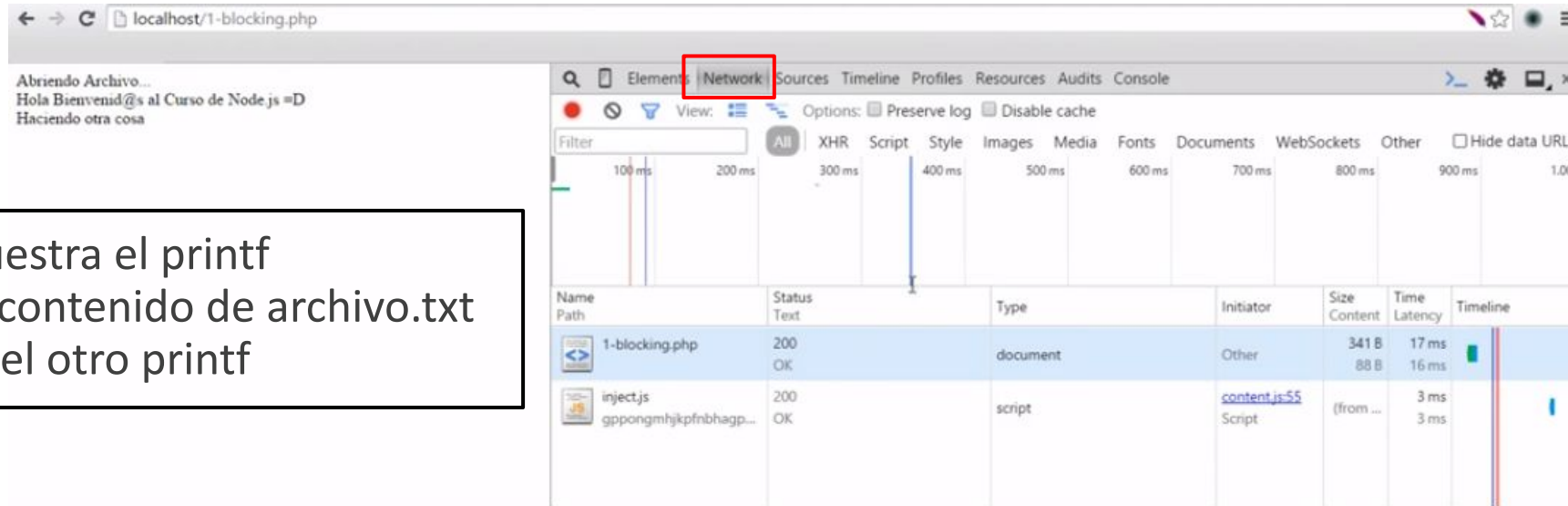
1-blocking.php x archivo.txt x

```
1 Hola
2 Bienvenid@s al
3 Curso de Node.js
4 =D
```

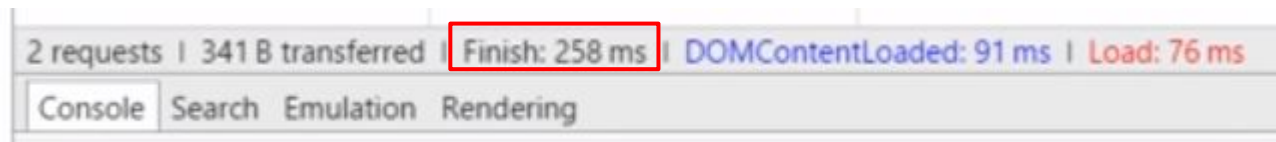
EJEMPLO: BLOQUEANTE VS NO BLOQUEANTE

PHP CÓDIGO BLOQUEANTE

1. Primero muestra el printf
2. Después el contenido de archivo.txt
3. Por último, el otro printf



- Lo ejecutamos con la herramienta para desarrollador, accedemos a la pestaña de “Network” y abajo del todo podemos ver **lo que tardó en ejecutar todo**:



EJEMPLO: BLOQUEANTE VS NO BLOQUEANTE

NODE.JS CÓDIGO BLOQUEANTE

Node.js puede ejecutarse de manera síncrona (bloqueante) o de manera asíncrona (no bloqueante).

Como ya hemos dicho, **su naturaleza es ser asíncrono (no bloqueante)**, pero podemos forzar a lanzar un código para que trabaje de manera síncrona (bloqueante).

Para el siguiente ejemplo, crearemos una nueva carpeta, en **C:**, por ejemplo, de nombre “**Node**”. Dentro, otra carpeta llamada “**00ModeloAsincrono**”. Dentro, crearemos “**2-blocking.js**” y copiamos el anterior “*archivo.txt*”.

EJEMPLO: BLOQUEANTE VS NO BLOQUEANTE

NODE.JS CÓDIGO BLOQUEANTE

```
JS 2-blocking.js •
C: > Users > danya > OneDrive > Desktop > Node > 00ModeloAsincrono > JS 2-blocking.js
1  var fs = require('fs')
2  console.log('\nAbriendo Archivo...')
3
4  var content = fs.readFileSync('archivo.txt', 'utf8')
5  console.log(content)
6
7  console.log('\nHaciendo otra cosa\n')
8
9  console.log( process.uptime() )
```

Con **process.uptime()** podremos ver el tiempo dedicado a ejecutarse todo el código anterior.

Declaramos una variable llamada **fs**. En Node, con **require("fs");** podremos **importar módulos** (como librerías o clases). En este caso, el módulo "**fs**" → **FileSystem**, nos permite trabajar con ficheros locales del sistema.

Declaramos una variable llamada **content** con el contenido de llamar a la función "Leer un archivo **de manera Síncrona**" del módulo **fs**.

EJEMPLO: BLOQUEANTE VS NO BLOQUEANTE

NODE.JS CÓDIGO BLOQUEANTE

Para ejecutar en Node, abrimos un PowerShell, accedemos (cd) a la carpeta donde tenemos el código y escribimos: “**node nombrefichero.js**”

```
Desktop\Node\00ModeloAsincrono> node 2-blocking.js
```

```
Abriendo Archivo...  
Hola  
Bienvenid@s al  
Curso de Node.js  
=D  
  
Haciendo otra cosa  
  
0.238
```

Podemos ver que ha tardado **0,238** milisegundos.

EJEMPLO: BLOQUEANTE VS NO BLOQUEANTE

NODE.JS CÓDIGO BLOQUEANTE

Por tanto, en un modelo BLOQUEANTE, siempre va a tardar menos Node.js que PHP:

```
1 PHP - 258ms  
2 Node - 238ms
```

En el anterior código JavaScript de Node, BLOQUEANTE, se ejecutan las líneas unas debajo de otras, esperando en todo caso que finalice la actual para pasar a la siguiente.

Ahora haremos el ejemplo con un código más natural de Node.js, siendo asíncrona (**NO BLOQUEANTE**).

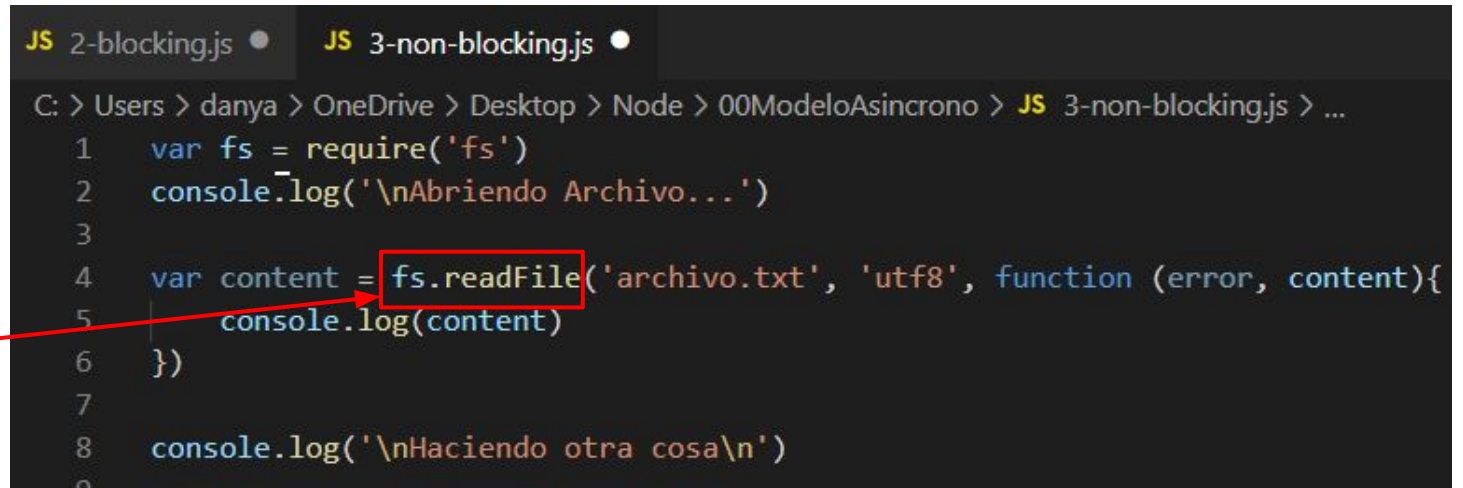
EJEMPLO: BLOQUEANTE VS NO BLOQUEANTE

NODE.JS CÓDIGO **NO** BLOQUEANTE

Crearemos un nuevo fichero JS que vamos a llamar “**3-non-blocking.js**”.

Ahora, en el código NO BLOQUEANTE, el método de entrada se llama “Lee Fichero”. Esta función recibe un tercer parámetro (a diferencia de la anterior función síncrona), **que es una función anónima de tipo callback** (es una función que recibe como argumento otra función y la ejecuta).

Con esto conseguimos la asincronía.



```
JS 2-blocking.js • JS 3-non-blocking.js •
C: > Users > danya > OneDrive > Desktop > Node > 00ModeloAsincrono > JS 3-non-blocking.js > ...
1  var fs = require('fs')
2  console.log('\nAbriendo Archivo...')
3
4  var content = fs.readFile('archivo.txt', 'utf8', function (error, content){
5    console.log(content)
6  })
7
8  console.log('\nHaciendo otra cosa\n')
```

EJEMPLO: BLOQUEANTE VS NO BLOQUEANTE

NODE.JS CÓDIGO **NO** BLOQUEANTE

Ejecutamos el código → `\Desktop\Node\00ModeloAsincrono> node .\3-non-blocking.js`

El resultado es:

```
Abriendo Archivo...
Haciendo otra cosa
Hola
Bienvenid@s al
Curso de Node.js
=D
```

Podemos comprobar que **no ha seguido de forma secuencial** el código escrito, ya que:

- **primero** ha indicado el primer `console.log`
- **segundo** ha indicado el tercer `console.log`
- **tercero** ha mostrado el contenido de "archivo.txt"

```
var fs = require('fs')
console.log('\nAbriendo Archivo...')

var content = fs.readFile('archivo.txt', 'utf8', function(error, content){
  console.log(content)
});

console.log('\nHaciendo otra cosa\n')
```

EJEMPLO: BLOQUEANTE VS NO BLOQUEANTE

NODE.JS CÓDIGO **NO** BLOQUEANTE

Si comprobamos el tiempo de ejecución:

`console.log (process.uptime())`

En este caso hemos obtenido un resultado de 0,230 milisegundos.

```
Abriendo Archivo...  
  
Haciendo otra cosa  
  
0.23  
Hola  
Bienvenid@s al  
Curso de Node.js  
=D
```

Por tanto, podemos comprobar que:

- PHP síncrono (bloqueante) → 258ms
- Node síncrono (bloqueante) → 238ms
- Node asíncrono (no bloqueante) → 230ms

¿CUÁL ES LA DIFERENCIA?

LAS CALLBACK

Es una pieza de código ejecutable que se pasa como argumento a otro código.



Con **la callback** estamos logrando un comportamiento **asíncrono**, es decir, **no bloqueante**.

El código se ejecuta y no espera a que la lectura del fichero finalice.

CALLBACK...¿ES TODO POSITIVO?

CALLBACK HELL

```
1 function hell (win) {  
2   // for listener purpose  
3   return function () {  
4     loadLink(win, REMOTE_SRC+'assets/css/style.css', function () {  
5       loadScript(win, REMOTE_SRC+'lib/async.js', function () {  
6         loadScript(win, REMOTE_SRC+'lib/easyXDM.js', function () {  
7           loadScript(win, REMOTE_SRC+'lib/json2.js', function () {  
8             loadScript(win, REMOTE_SRC+'lib/underscore.min.js', function () {  
9               loadScript(win, REMOTE_SRC+'lib/backbone.min.js', function () {  
10                loadScript(win, REMOTE_SRC+'dev/base.dev.js', function () {  
11                 loadScript(win, REMOTE_SRC+'assets/js/deps.js', function () {  
12                  loadScript(win, REMOTE_SRC+'src/'+win.loader_path+'/loader.js', function () {  
13                    async.eachSeries(SERIALS, function (src, callback) {  
14                      loadScript(win, BASE_URL+src, callback);  
15                    });  
16                  });  
17                });  
18              });  
19            });  
20          });  
21        });  
22      });  
23    });  
24  });  
25 }  
26 }
```



Podemos escribir una función dentro de la otra, y dentro de la otra, y dentro de la otra.....

CALLBACK...¿UNA FORMA MEJOR?

```
JS 2-blocking.js • JS 3-non-blocking.js • JS 4-callback-definida.js ×
C: > Users > danya > OneDrive > Desktop > Node > 00ModeloAsincrono > JS 4-callback-definida.js >
1  var fs = require('fs')
2  console.log('\nAbriendo Archivo...')
3
4  function imprimir(error, content){
5  |   console.log(content)
6  | }
7
8  var content = fs.readFile('archivo.txt', 'utf8', imprimir)
9
10 console.log('\nHaciendo otra cosa\n')
11
12 console.log( process.uptime() )
```

Para solucionar los *callback hell* y tener un código mucho más legible, **declararemos las funciones aparte** y por parámetro al “readFile” (o la que sea), se le pasará la llamada a dicha función.

Índice

1. ¿QUÉ ES NODE.JS?
2. MODELO ASÍNCRONO Y NO BLOQUEANTE
3. CARACTERÍSTICAS Y VENTAJAS
4. INSTALACIÓN

SINGLE THREAD

En Node no disponemos del multihilo, tan solo disponemos de un solo hilo principal.

¿Quiere decir esto que solo se podrá ejecutar una tarea a la vez?

SINGLE THREAD

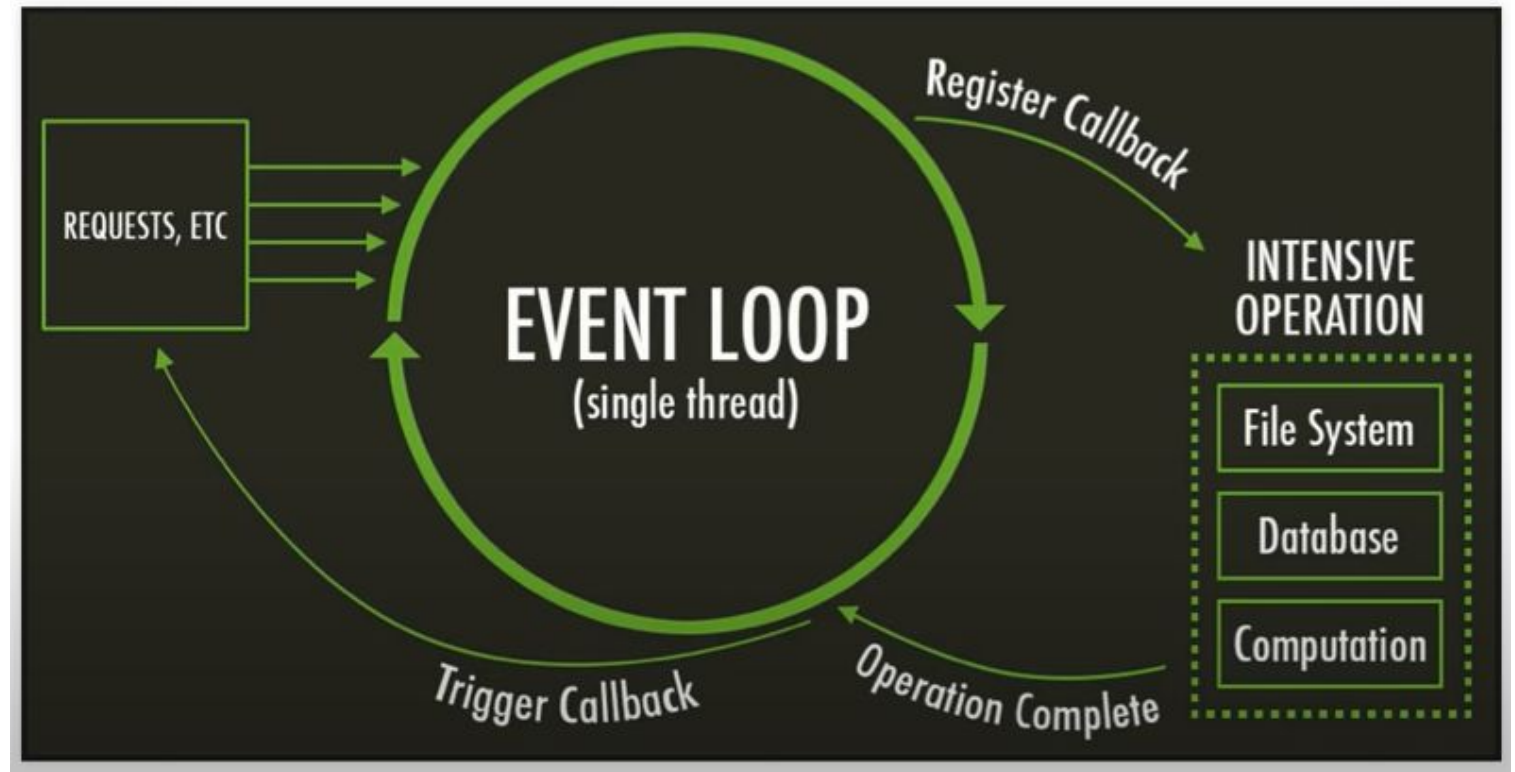
En Node no disponemos del multihilo, tan solo disponemos de un solo hilo principal.

¿Quiere decir esto que solo se podrá ejecutar una tarea a la vez?

No, como ya hemos visto, el **modelo asíncrono (no bloqueante)** nos permitirá lanzar diferentes tareas y que se vayan ejecutando y finalizando según les corresponda.

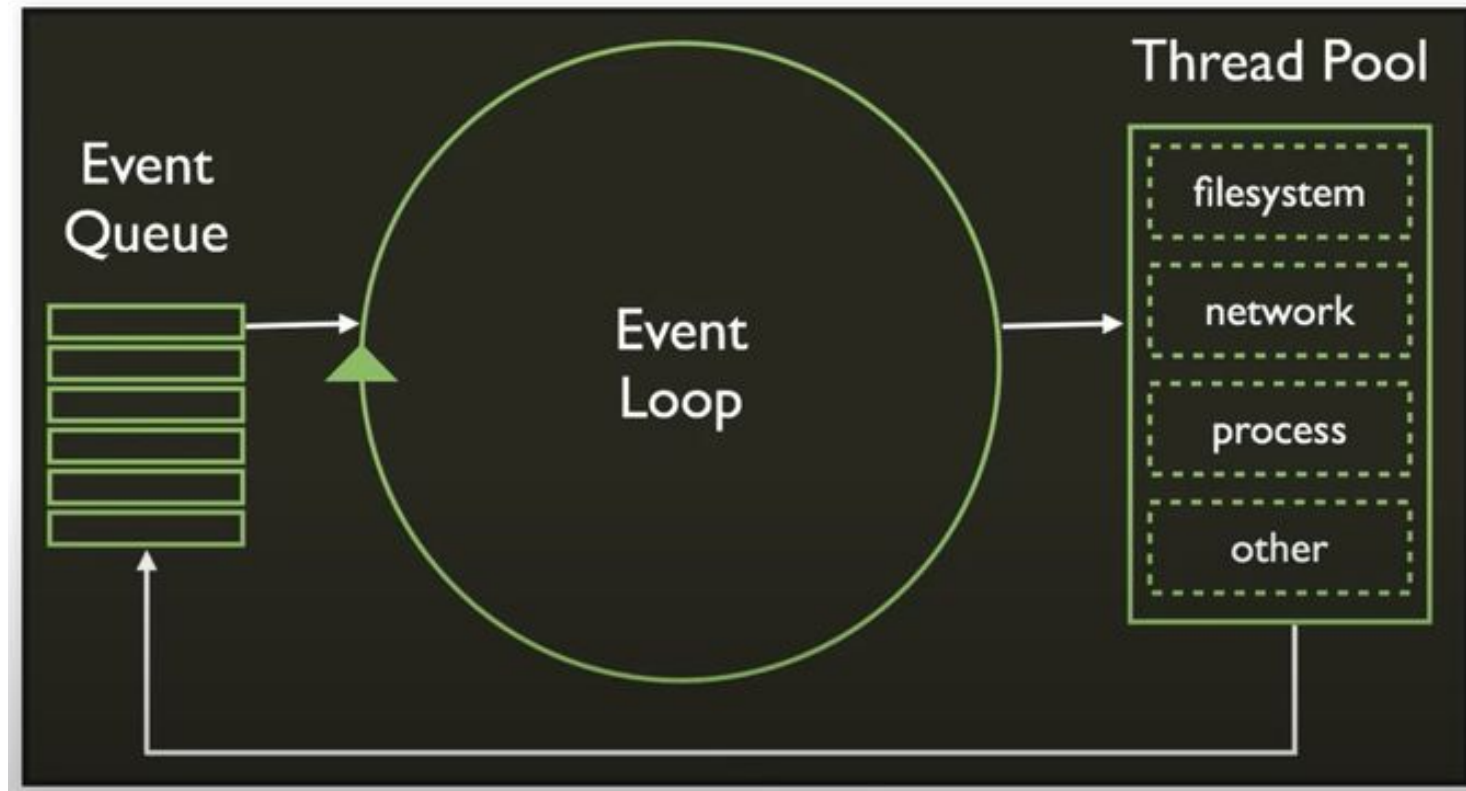
EVENT LOOP

El navegador hace X peticiones, que se van agregando al hilo unico que pasan, mediante los callback, a ejecutarse mediante los módulos. Una vez terminan, vuelven al hilo único.



EVENT LOOP

En Node no disponemos del multihilo, tan solo disponemos de un solo hilo principal.



CARACTERÍSTICAS DE NODE.JS

- Es concurrente sin paralelismo.
- Es asíncrono y no bloqueante.
- Es orientado a eventos.
- Single thread basado en callbacks.



¿QUÉ PODEMOS HACER CON NODE.JS?

- Aplicaciones cliente - servidor
- Aplicaciones servidor - servidor (networking)
- Chats, Juegos (en línea), Clientes de correo, Traductores...
- Aplicaciones colaborativas (tipo Google Drive, Dropbox, etc.) y Redes Sociales.
- Estadísticas y analítica.
- Aplicaciones de red.
- Sitios Web, Blogs, CMS's (Sistemas de Gestión de Contenidos).
- Controlar Hardware.

RESUMEN DE BENEFICIOS DE USAR NODE.JS

- **Arquitectura unificada**
 - **Javascript Full Stack** → Usamos el mismo lenguaje (JavaScript) en todo el proceso de una aplicación compleja, pasando por el frontend, backend y Base de Datos.
- **Código reutilizable** → (en el frontend, backend y en la base de datos).
- **Excelente rendimiento y escalabilidad.**
- **Fuerte y apasionada comunidad.**

DIFERENCIA ENTRE FRONT Y BACK

JavaScript en el FRONT

- No me importa el SO.
- Apenas hay E/S.
- Un único usuario.
- Todo es accesible con objetos (document, window...) y valores primitivos.

JavaScript en el BACK

- Llamadas al SO (en el anterior ejemplo, las llamadas al FileSystem).
- Muchas E/S.
- Usuarios y permisos.
- Puertos y servidores.
- ...

Índice

1. ¿QUÉ ES NODE.JS?
2. MODELO ASÍNCRONO Y NO BLOQUEANTE
3. CARACTERÍSTICAS Y VENTAJAS
4. **INSTALACIÓN**

INSTALACIÓN

nodejs.org



Node.js® is a JavaScript runtime built on Chrome's V8 JavaScript engine.

New security releases now available for Node.js 12, 14, 16, and 17 release lines

Download for Windows (x64)

16.13.2 LTS

Recommended For Most Users

17.3.1 Current

Latest Features

[Other Downloads](#) | [Changelog](#) | [API Docs](#) [Other Downloads](#) | [Changelog](#) | [API Docs](#)

Or have a look at the [Long Term Support \(LTS\) schedule](#)

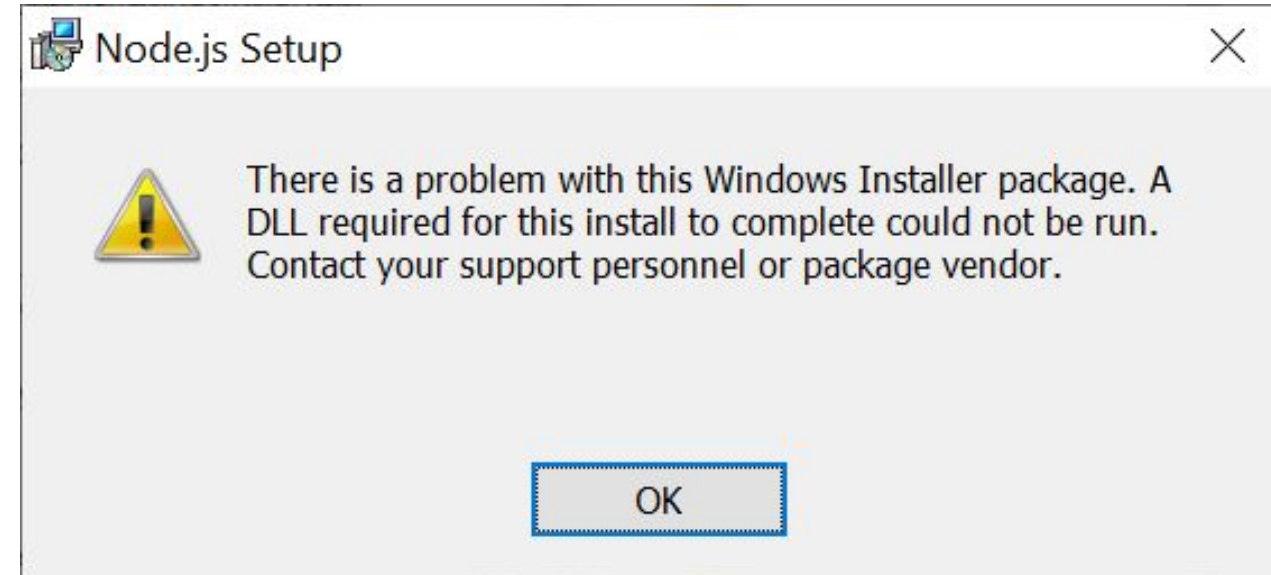
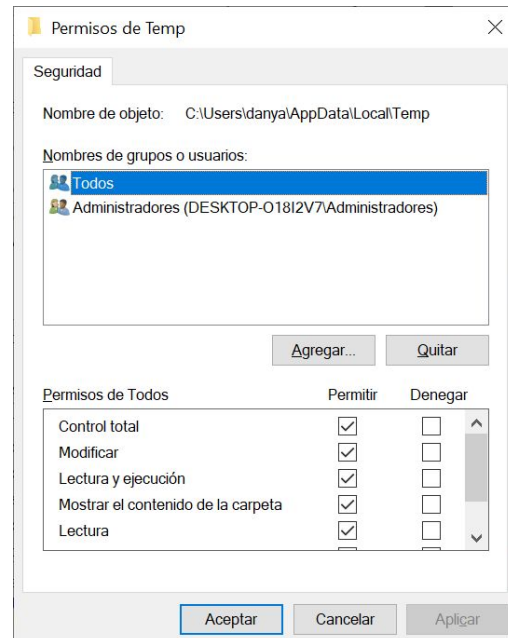
INSTALACIÓN

Si obtienes este problema al instalar:

Deberás cambiar los permisos a la carpeta **Temp** que está en:

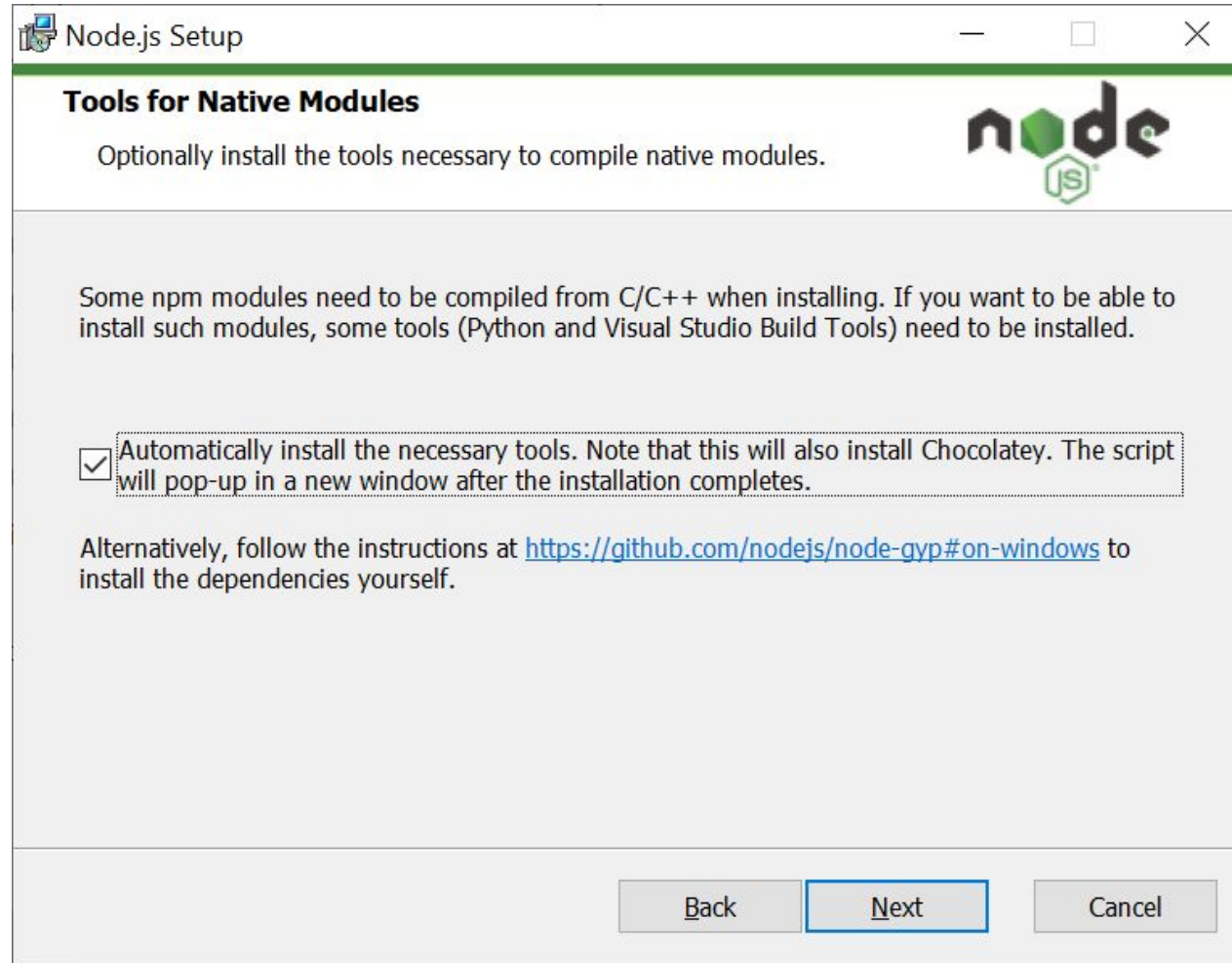
C:\Users**USUARIO**\AppData\Local

Con **Control Total**



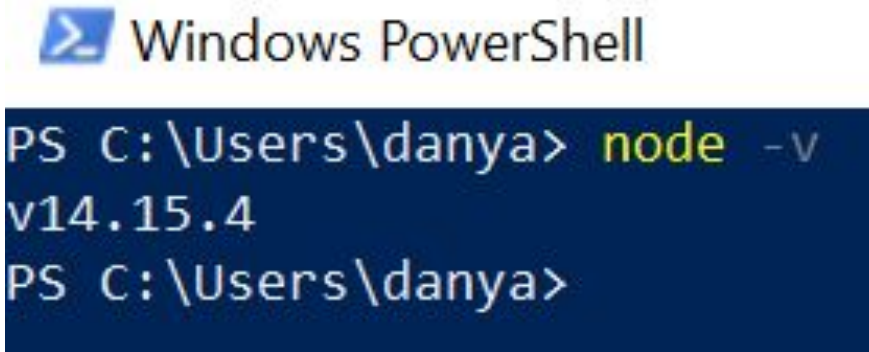
INSTALACIÓN

Recomiendo marcar esta opción



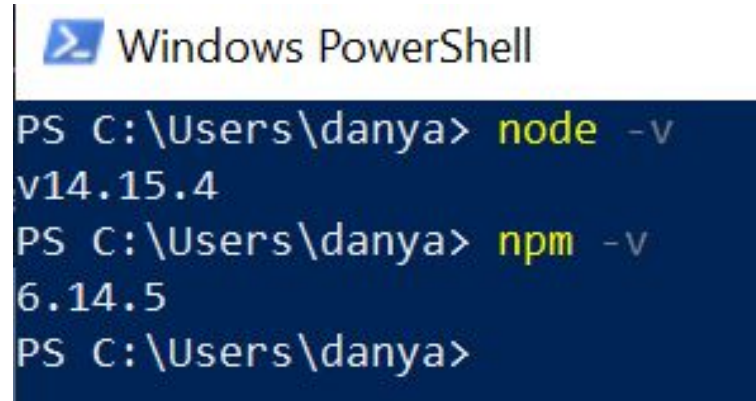
INSTALACIÓN

Una vez finalice la instalación, abriremos PowerShell de Windows y lanzaremos el comando **node -v**



```
Windows PowerShell  
PS C:\Users\danya> node -v  
v14.15.4  
PS C:\Users\danya>
```

También comprobaremos el gestor de paquetes que incorpora node, llamado NPM, mediante el comando **npm -v**

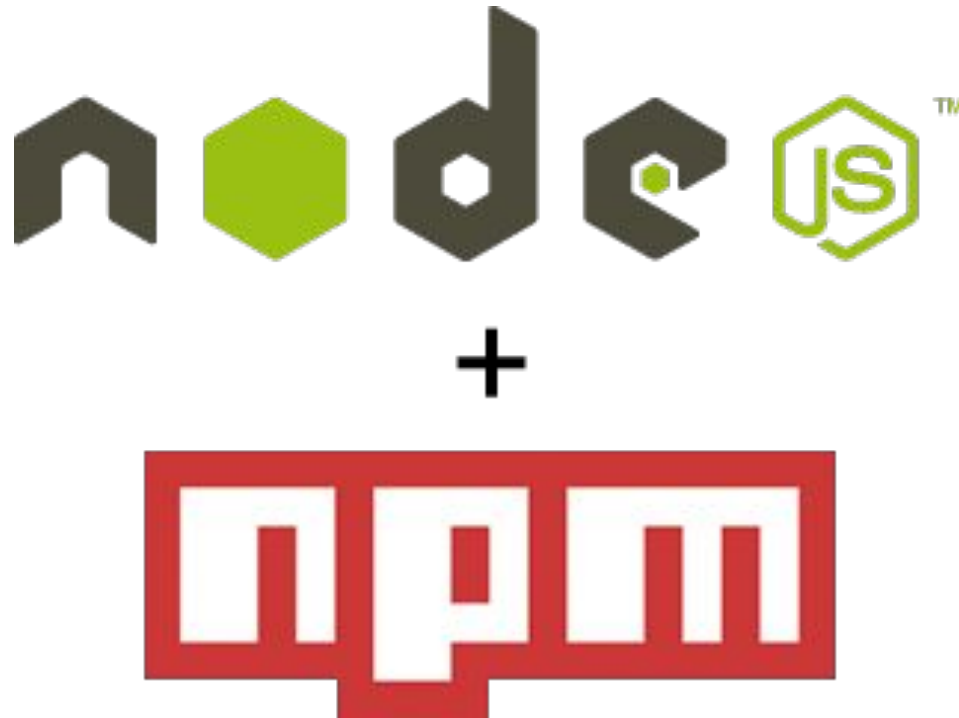


```
Windows PowerShell  
PS C:\Users\danya> node -v  
v14.15.4  
PS C:\Users\danya> npm -v  
6.14.5  
PS C:\Users\danya>
```

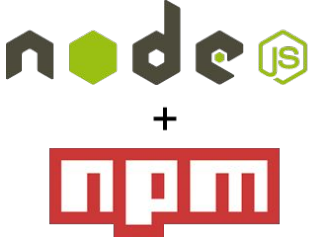

NPM

Ya hemos visto algunos de los módulos más importantes del núcleo de Node.js.

Ahora vamos a ver **NPM**, el gestor de paquetes de Node.js (y todo JavaScript).



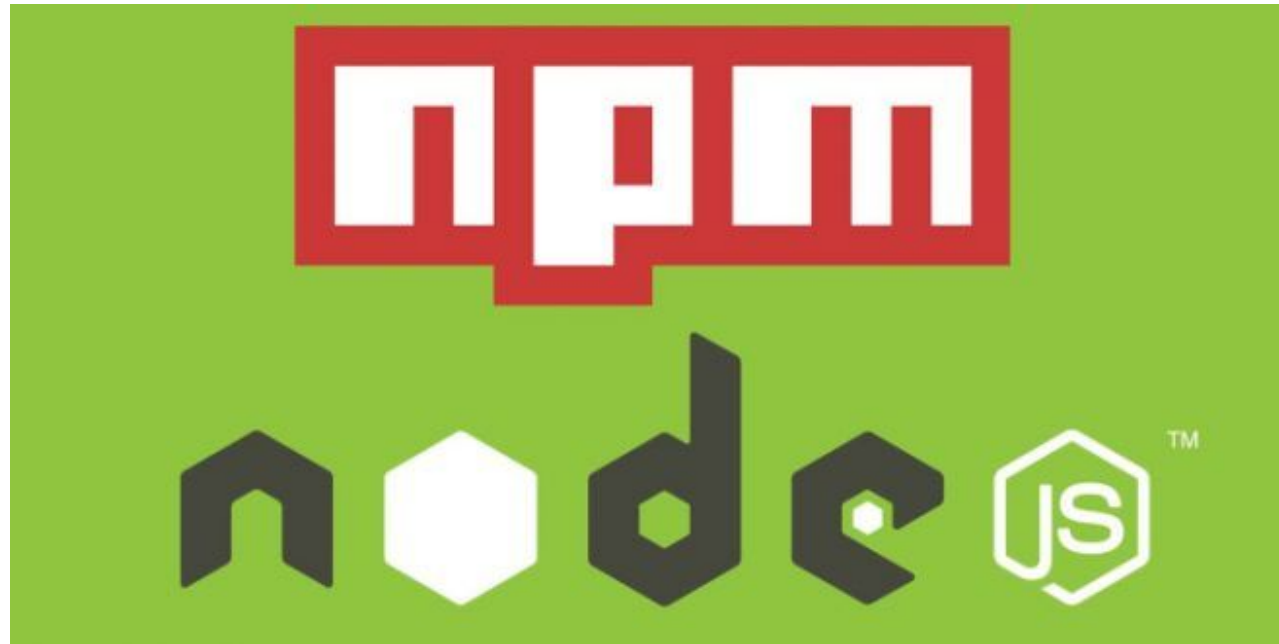
NPM

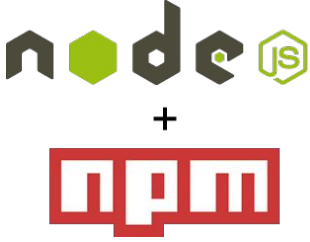


¿Cómo se instala?

¿Lo tenemos ya instalado?

¿Cómo lo comprobamos?





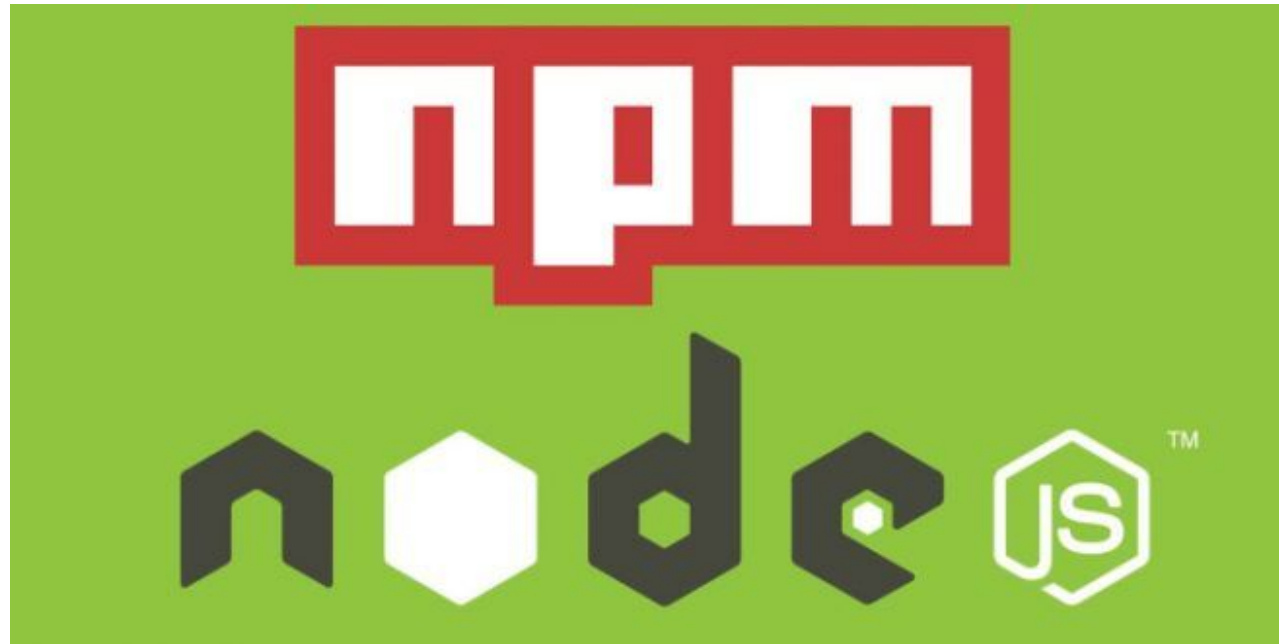
NPM

¿Cómo se instala? **Junto con el instalador de Node.js**

¿Lo tenemos ya instalado? **Si, se instaló junto con Node.js**

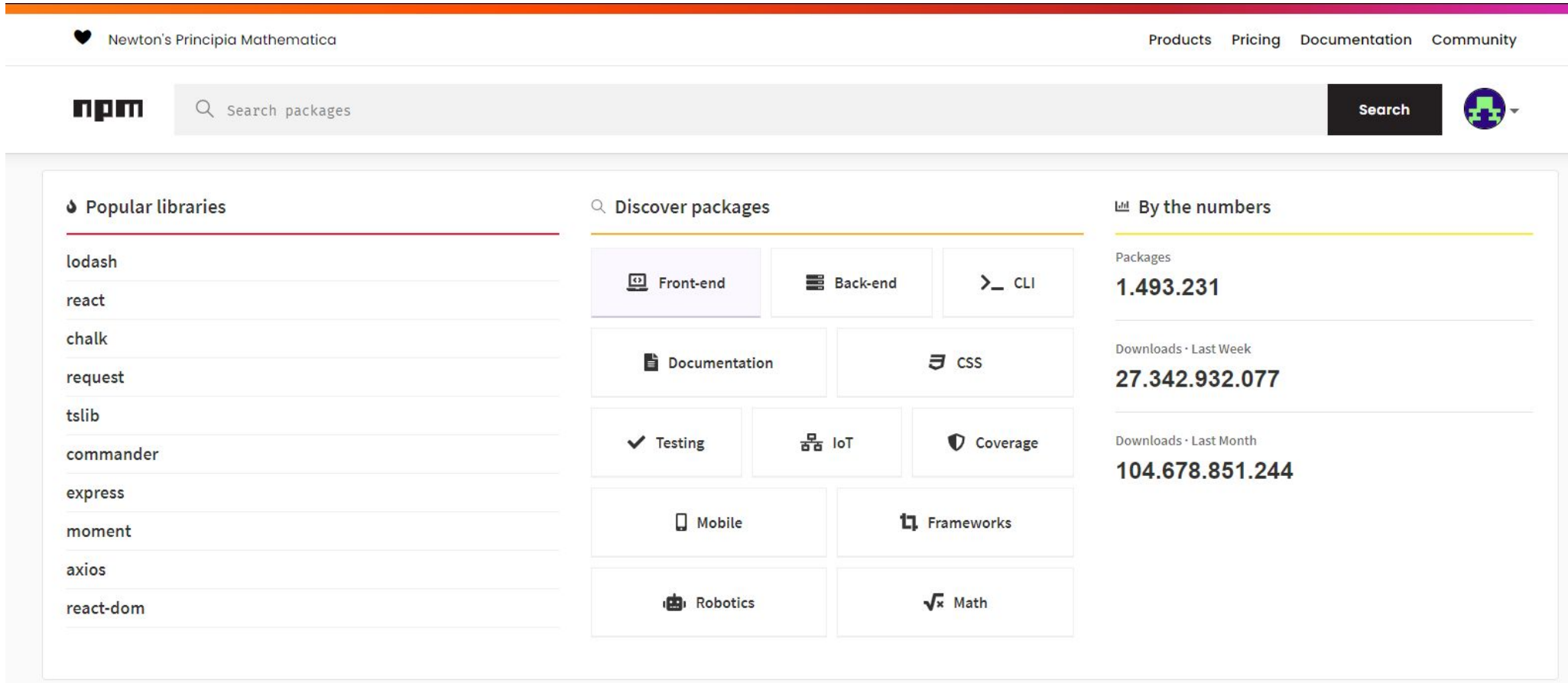
¿Cómo lo comprobamos? En PowerShell → ***npm -v***

```
PS C:\Node\01ModulosCore> npm -v  
6.14.10  
PS C:\Node\01ModulosCore>
```



NPM

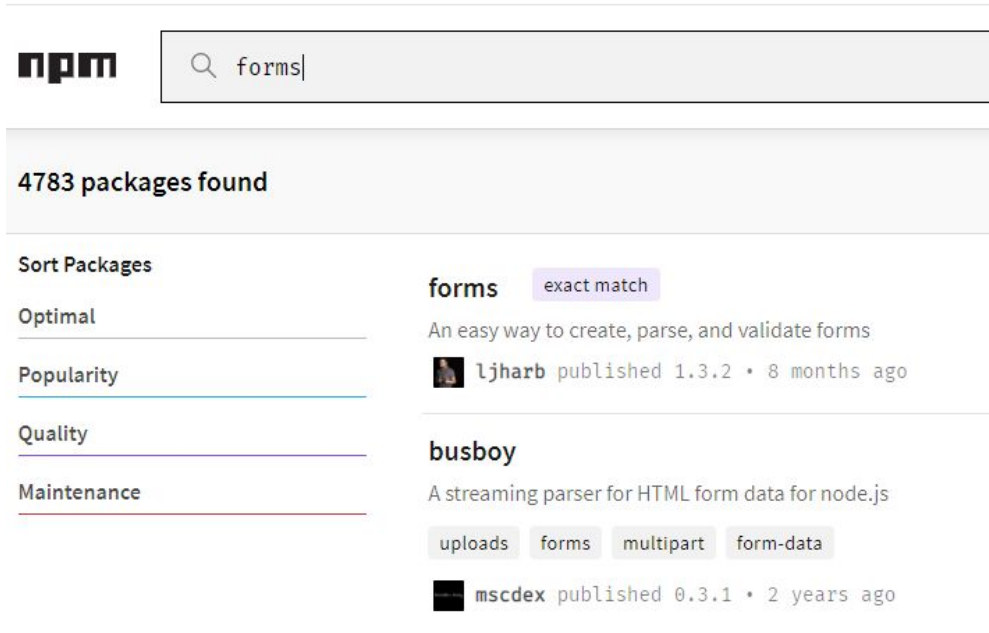
Lo primero que haremos será acceder a la [web oficial de NPM](https://www.npmjs.com/) y crearnos una cuenta (**no es obligatorio**). Deberemos verificarla en nuestro email.



The screenshot shows the NPM website homepage. At the top, there's a navigation bar with a heart icon and the text "Newton's Principia Mathematica" on the left, and links for "Products", "Pricing", "Documentation", and "Community" on the right. Below this is a search bar with the NPM logo, a search icon, the text "Search packages", a "Search" button, and a user profile icon. The main content area is divided into three columns. The left column, titled "Popular libraries", lists several packages: lodash, react, chalk, request, tslib, commander, express, moment, axios, and react-dom. The middle column, titled "Discover packages", features a grid of category tiles: Front-end, Back-end, CLI, Documentation, CSS, Testing, IoT, Coverage, Mobile, Frameworks, Robotics, and Math. The right column, titled "By the numbers", displays statistics: "Packages" with a count of 1.493.231, "Downloads · Last Week" with a count of 27.342.932.077, and "Downloads · Last Month" with a count of 104.678.851.244.

NPM

Por ejemplo, si buscamos un paquete destinado a trabajar con formularios, podemos buscar por “*forms*”:



The screenshot shows the npm search interface. At the top, the 'npm' logo is on the left, and a search bar contains the text 'forms'. Below the search bar, it says '4783 packages found'. On the left side, there is a 'Sort Packages' section with four options: 'Optimal', 'Popularity', 'Quality', and 'Maintenance'. The 'Optimal' option is selected. The search results are displayed in a list. The first result is for the 'forms' package, which is marked as an 'exact match'. It is described as 'An easy way to create, parse, and validate forms' and was published by 'ljharb' 8 months ago. The second result is for the 'busboy' package, described as 'A streaming parser for HTML form data for node.js', with tags for 'uploads', 'forms', 'multipart', and 'form-data'. It was published by 'mscdex' 2 years ago.

npm

forms

4783 packages found

Sort Packages

Optimal

Popularity

Quality

Maintenance

forms exact match

An easy way to create, parse, and validate forms

ljharb published 1.3.2 • 8 months ago

busboy

A streaming parser for HTML form data for node.js

uploads forms multipart form-data

mscdex published 0.3.1 • 2 years ago

Si accedemos el primer paquete que vemos, podemos leer en la descripción:

“Construir un buen formulario a mano es mucho trabajo. Los frameworks populares como Ruby on Rails y Django contienen código para hacer que este proceso sea menos doloroso. Este módulo es un intento de proporcionar el mismo tipo de ayuda para node.js.”

NPM

Para instalarlo, podemos ver que debemos poner “*npm install forms*”

Forms ^{v1.3.2}

build **passing** dependencies **up to date** devDependencies **up to date** license **MIT** downloads **5.7k/month**


19 ★

npm install forms
11 dependencies
version 1.3.2
updated 8 months ago

Constructing a good form by hand is a lot of work. Popular frameworks like Ruby on Rails and Django contain code to make this process less painful. This module is an attempt to provide the same sort of helpers for node.js.

```
$ npm install forms
```

Pero, antes de ejecutar este comando. ¿Dónde se instalará?

Abrir la CMD como administradores: **Set-ExecutionPolicy -ExecutionPolicy Bypass -Scope CurrentUser**

NPM

Se instala en el directorio en el que me encuentre en el PowerShell. Por eso, en nuestro espacio de trabajo de Node (C:\Node\), desde PowerShell vamos a crear una nueva carpeta llamada **02ModulosNPM** y accederemos a ella.

```
Windows PowerShell
PS C:\Node> md 02ModulosNPM

Directorio: C:\Node

Mode                LastWriteTime         Length Name
----                -
d-----          31/01/2021    16:37             02ModulosNPM

PS C:\Node> cd .\02ModulosNPM\
PS C:\Node\02ModulosNPM>
```

Ahora podremos lanzar el comando para instalar el módulo “**forms**”:

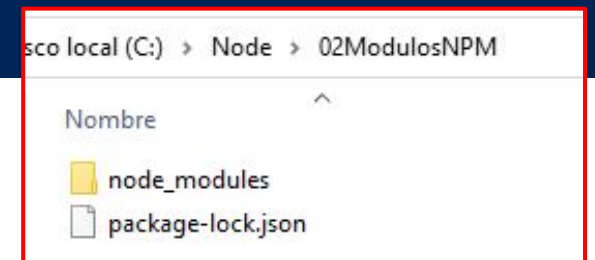
```
PS C:\Node\02ModulosNPM> npm install forms
npm WARN ENOENT: no such file or directory, open 'C:\Node\02ModulosNPM\package.json'
npm WARN created a lockfile as package-lock.json. You should commit this file.
npm WARN ENOENT: no such file or directory, open 'C:\Node\02ModulosNPM\package.json'
npm WARN 02ModulosNPM No description
npm WARN 02ModulosNPM No repository field.
npm WARN 02ModulosNPM No README data
npm WARN 02ModulosNPM No license field.

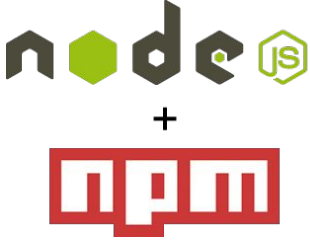
+ forms@1.3.2
added 30 packages from 19 contributors and audited 30 packages in 5.669s

20 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```

Podemos ver en la carpeta que ahora hay nuevas carpetas y ficheros. **Luego hablamos del “package-lock.json”**



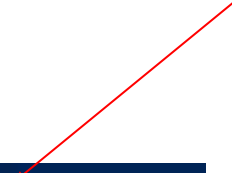


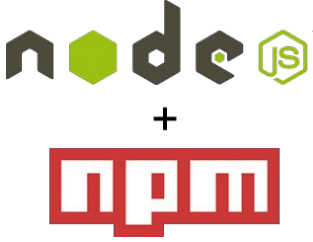
NPM

Hay otra forma de instalar dicho módulo, que es hacerlo de **manera global**. Esto no va a instalarlo en la carpeta local, sino en el propio ordenador donde se lanza el comando de instalación, quedando disponible para poder usarlo en el futuro en cualquier proyecto nuevo (tal y como se utilizan los módulos del core de Node).

Para instalarlo de esta forma, simplemente **deberemos agregar un “-g”** al anterior comando:

```
PS C:\Node\02ModulosNPM> npm install forms -g
+ forms@1.3.2
added 30 packages from 19 contributors in 3.075s
PS C:\Node\02ModulosNPM>
```

A red arrow points from the text “-g” in the paragraph above to the -g flag in the terminal command.



NPM

Para desinstalar un módulo, podemos hacerlo con el comando “*npm **uninstall** forms*”, por ejemplo.

Para desinstalarlo de forma global, deberemos incluir el “*-g*”:

```
PS C:\Node\02ModulosNPM> npm uninstall forms -g
removed 30 packages in 0.476s
PS C:\Node\02ModulosNPM>
```

NPM - PACKAGE.JSON

Es un fichero que, principalmente, nos servirá para **mantener un orden** de nuestros paquetes y dependencias. Mantendrá un listado de los paquetes y sus versiones. [Recomiendo revisar la documentación](#)

Suponiendo que nuestra carpeta **02ModulosNPM** corresponde al directorio de nuestro proyecto, podemos lanzar el comando: ***npm init -y***

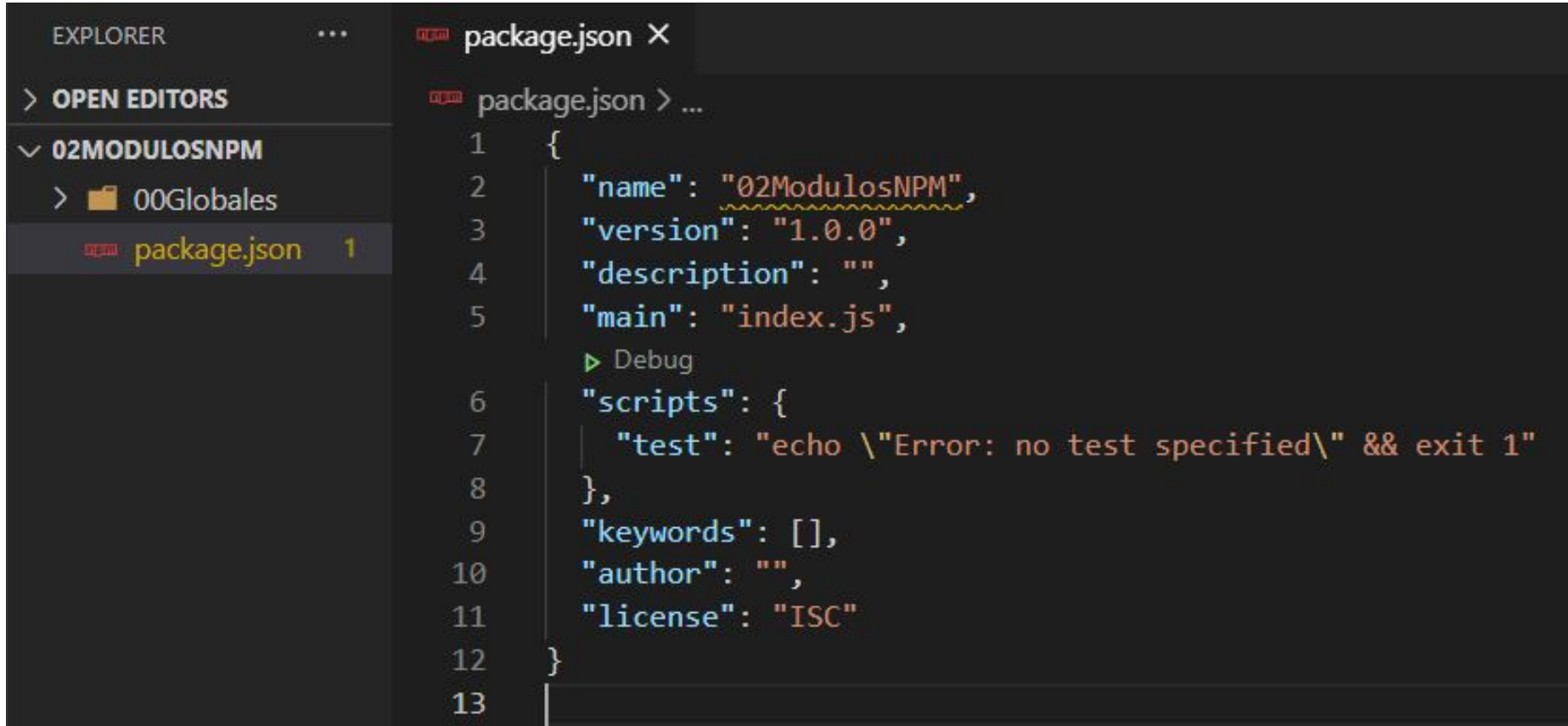
```
PS C:\Node\02ModulosNPM> npm init -y
Wrote to C:\Node\02ModulosNPM\package.json:

{
  "name": "02ModulosNPM",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```

Esto creará un archivo con información configurable sobre nuestro proyecto, como sus **dependencias** y scripts.

NPM - PACKAGE.JSON

Si abrimos el fichero, podremos ver:




```
EXPLORER  ...  package.json X
> OPEN EDITORS
02MODULOSNPM
  > 00Globales
    package.json 1

1  {
2    "name": "02ModulosNPM",
3    "version": "1.0.0",
4    "description": "",
5    "main": "index.js",
6    "scripts": {
7      "test": "echo \"Error: no test specified\" && exit 1"
8    },
9    "keywords": [],
10   "author": "",
11   "license": "ISC"
12 }
13
```

NPM - PACKAGE.JSON

De momento no tiene ninguna dependencia, pero vamos a instalar un nuevo paquete con NPM para ver cómo se crean automáticamente. Vamos a buscar un famoso módulo llamado “***cowsay***”, utilizado típicamente para realizar pruebas.

cowsay 

1.4.0 • Public • Published 2 years ago

[Readme](#) [Explore](#) [4 Dependencies](#) [157 Dependents](#) [19 Versions](#)

cowsay

```
< srsly dude, why? >
  \  ^__^
   \  (oo)\_______
      (__)\       )\/\
         ||----w |
         ||     ||
```

cowsay is a configurable talking cow, originally written in Perl by **Tony Monroe**

This project is a translation in JavaScript of the original program and an attempt to bring the same silliness to node.js.

Install

```
npm install -g cowsay
```

Install
`> npm i cowsay`

Weekly Downloads
7837

Version	License
1.4.0	MIT
Unpacked Size	Total Files
264 kB	201
Issues	Pull Requests
7	4

Homepage
github.com/piuccio/cowsay

Repository

Como bien indica, otra forma de instalar los paquetes **de forma local** es mediante el comando “***npm i paquete***”

```
7   "test": "echo \\"Err
8   },
9   "keywords": [],
10  "author": "",
11  "license": "ISC",
12  "dependencies": {
13    "cowsay": "^1.4.0"
14  }
15 }
16
```

NPM - PACKAGE.JSON

Vemos que ha aparecido una nueva dependencia. Para probarlo, vamos a crear un nuevo fichero con nombre “*cowsay.js*”. Para ello, vamos a la [documentación](#) del módulo:

Usage as a module

cowsay can be used as any other npm dependency

```
var cowsay = require("cowsay");

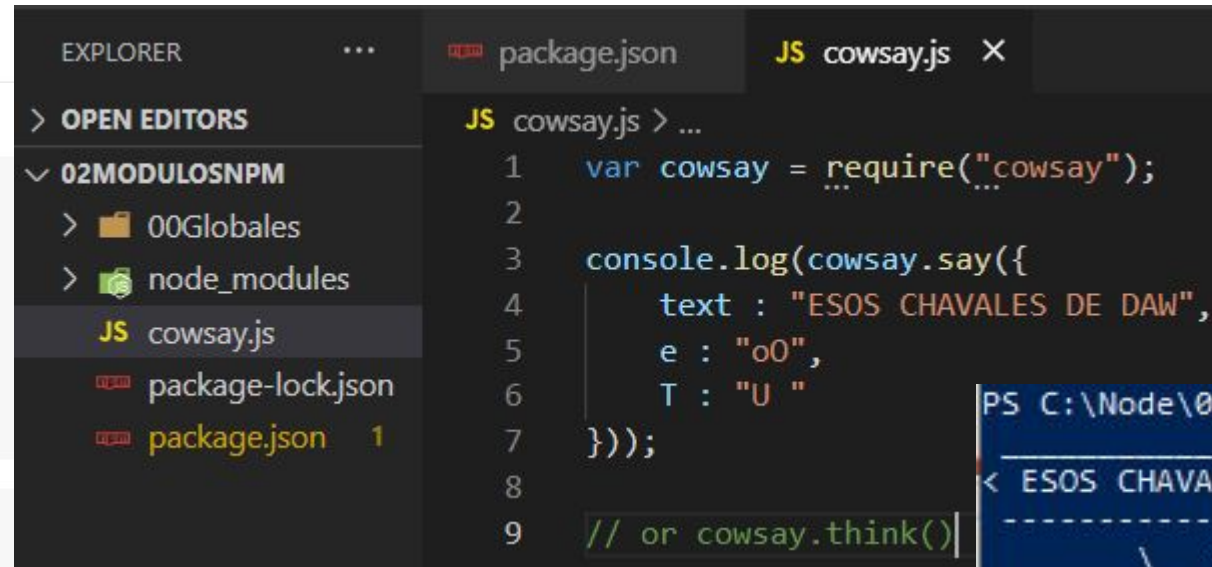
console.log(cowsay.say({
  text : "I'm a moooodule",
  e : "o0",
  T : "U "
}));

// or cowsay.think()
```

```
( I'm a moooodule )
```

```

  o  ^__^
  o  (oo)\_______
      (__)\       )\/\
         U     ||----w |
               ||     ||
```



The screenshot shows the VS Code interface with two files open: `package.json` and `cowsay.js`. The `package.json` file is open in the Explorer view, showing the `dependencies` section with `"cowsay": "4.0.1"`. The `cowsay.js` file is open in the Editor view, showing the code that uses the `cowsay` module.

```

package.json
{
  "name": "02modulosnpm",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" & exit 1"
  },
  "dependencies": {
    "cowsay": "4.0.1"
  }
}

cowsay.js
var cowsay = require("cowsay");

console.log(cowsay.say({
  text : "ESOS CHAVALES DE DAW",
  e : "o0",
  T : "U "
}));

// or cowsay.think()
```

```

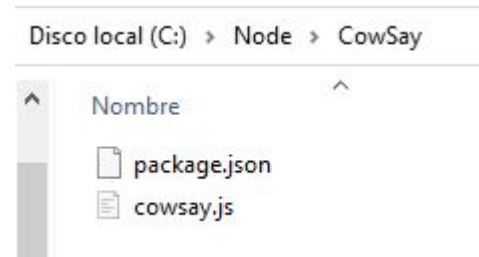
PS C:\Node\02ModulosNPM> node .\cowsay.js
< ESOS CHAVALES DE DAW >
-----
      \   ^__^
       \  (oo)\_______
          (__)\       )\/\
             U     ||----w |
                   ||     ||

PS C:\Node\02ModulosNPM>
```


NPM - PACKAGE.JSON

Por último, podemos probar como funciona a la perfección nuestro paquete y su configuración. Vamos a simular que este proyecto está finalizado y lo subo a GitHub. **NUNCA SE SUBE EL DIRECTORIO “node_modules”**, tan solo los ficheros “*.js*” y el “*package.json*”.

Para ello, crearemos una nueva carpeta llamada “**CowSay**” y dentro copiaremos el fichero “*cowsay.js*” y el “*package.json*”.



Ahora accedemos a ese directorio mediante el terminal y lanzamos el comando para **inicializar** el proyecto [**npm i**] mediante el *package.json*:

```
PS C:\Node> cd .\CowSay\  
PS C:\Node\CowSay> npm i  
npm WARN create-lockfile created a lockfile as package-lock.json. You should commit this file.  
npm WARN 02ModulosNPM@1.0.0 No description  
npm WARN 02ModulosNPM@1.0.0 No repository field.  
  
added 10 packages from 3 contributors and audited 10 packages in 0.65s  
found 1 low severity vulnerability  
  run `npm audit fix` to fix them, or `npm audit` for details  
PS C:\Node\CowSay>
```

