

Desarrollo Web en Entorno Cliente

Tema 18





Node.js

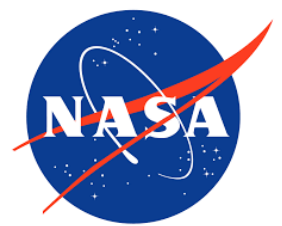
[node.js]



NETFLIX

PayPal

LinkedIn



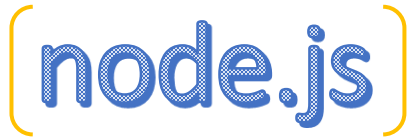
ebay

Node.js (o también NodeJS) es un entorno en tiempo de ejecución multiplataforma, de código abierto, para la capa del servidor basado en el motor JS V8 de Google.

Node.js utiliza muchos paquetes que añaden nuevas características y funcionalidad.

Node.js es conocido por su capacidad de ejecutar código JS en el lado del servidor, la ventaja es que con solo conocer el lenguaje JS se pueden crear aplicaciones tanto *Front End* como *Back End*, todas escritas en JS.

Además, Node.js también permite trabajar con el ordenador como si fuera un servidor de desarrollo, y realizar tareas automáticas, como, por ejemplo, realizar y ejecutar pruebas...

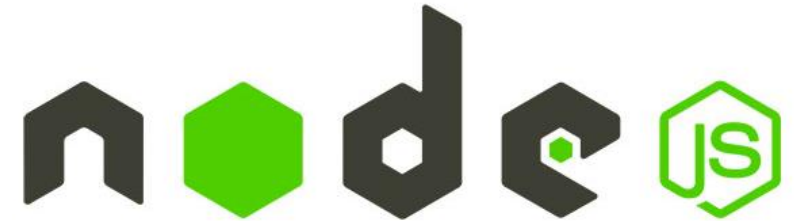


Node.js cuenta con muchas extensiones (complementos, *loaders*, *pluggins...*), más de 470.000, disponibles a través de paquetes.

También permite crear o desarrollar en JS nuevos paquetes, pero es muy probable que ya exista ese paquete en Node.js, por lo que no va a ser necesario crearlo.

Para instalar Node.js hay que ir a la siguiente dirección y descargar la versión que coincida con la plataforma:

<https://nodejs.org/es/download/>



Node.js® es un entorno de ejecución para JavaScript construido con V8, motor de JavaScript de Chrome.

Node.js assessment of OpenSSL 3.0.7 security advisory

Descargar para Windows (x64)

18.12.1 LTS

Recomendado para la mayoría

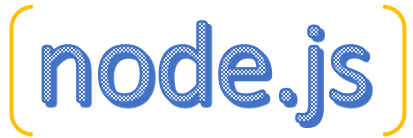
19.3.0 Actual

Últimas características

[Otras Descargas](#) | [Cambios](#) | [Documentación de la API](#)

[Otras Descargas](#) | [Cambios](#) | [Documentación de la API](#)

O eche un vistazo al Programa de soporte a largo plazo (LTS).



Hay disponible dos versiones:

- La versión LTS (*Long Time Support*) que dispone de soporte y corrección de errores, y el software es mantenido durante un largo periodo de tiempo.
- Y la última versión, llamada Actual, que incluye las últimas características experimentales por lo que puede contener errores y bugs.



Se recomienda descargar la versión LTS ya que es una versión depurada y probada.

Una vez descargado el fichero, como con cualquier otro instalador, se hace doble clic sobre el mismo para comenzar la instalación.

Descargas

Versión actual: **18.12.1** (includes npm 8.19.2)

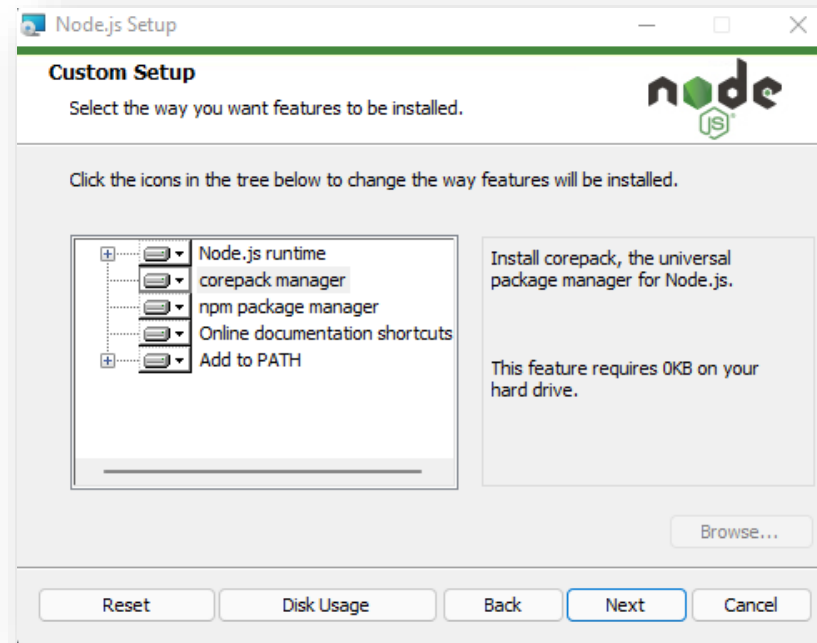
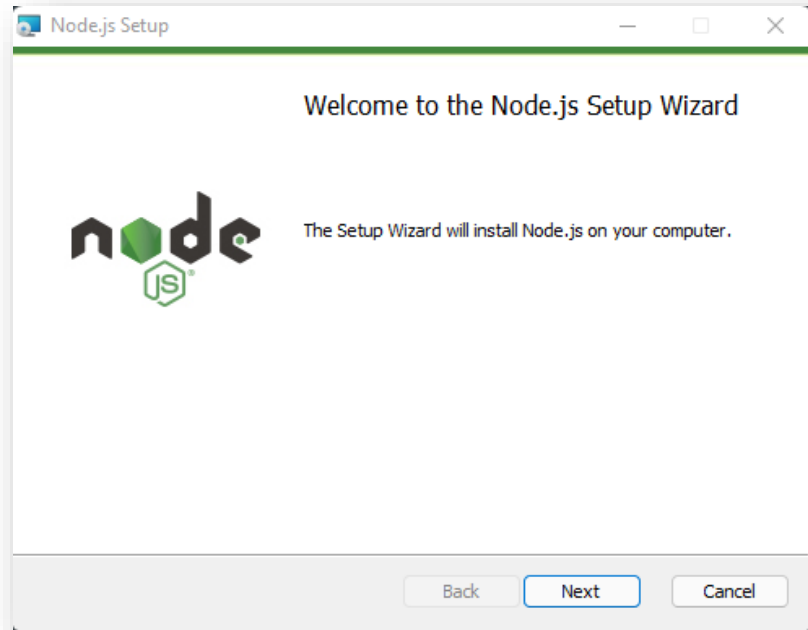
Descargue el código fuente de Node.js o un instalador pre-compilado para su plataforma, y comience a desarrollar hoy.

LTS Recomendado para la mayoría	Actual Últimas características	
 Instalador Windows <small>node-v18.12.1-x64.msi</small>	 Instalador macOS <small>node-v18.12.1.pkg</small>	 Código Fuente <small>node-v18.12.1.tar.gz</small>

Instalador Windows (.msi)	32-bit	64-bit
Binario Windows (.zip)	32-bit	64-bit
Instalador macOS (.pkg)	64-bit / ARM64	
Binario macOS (.tar.gz)	64-bit	ARM64
Binario Linux (x64)	64-bit	
Binario Linux (ARM)	ARMv7	ARMv8
Código Fuente	node-v18.12.1.tar.gz	

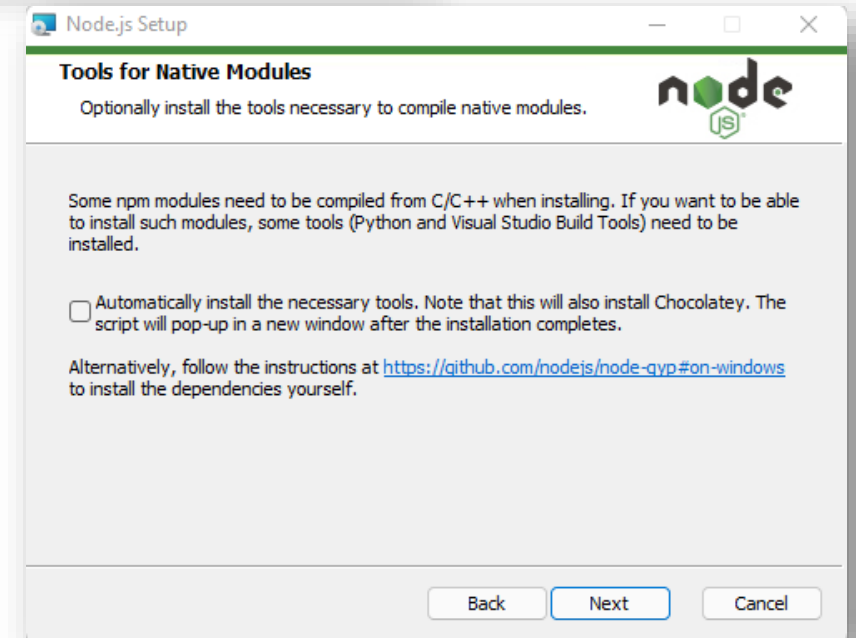
Plataformas adicionales

Imagen Docker	Imagen Docker Oficial Node.js
Linux en Power LE Systems	64-bit
Linux en System z	64-bit
AIX en Power Systems	64-bit



Con las opciones que viene por defecto es suficiente, no es necesario instalar nada más.

“*Tools Native Modules*” son herramientas que permiten crear nuevos módulos para Node.js y *Chocolatey* es un administrador de paquetes de Windows del estilo de “*apt-get*” de Linux.



<https://docs.chocolatey.org/en-us/#what-is-chocolatey>



Para desarrollo web no se necesita Chocolatey.

Chocolatey facilita la instalación y actualización automática de las aplicaciones en el ordenador. Para instalar Chocolatey desde un PowerShell hay que ejecutar el siguiente comando:

```
Administrador: Windows PowerShell
Windows PowerShell
Copyright (C) Microsoft Corporation. Todos los derechos reservados.

Instale la versión más reciente de PowerShell para obtener nuevas características y mejoras. https://aka.ms/PSWindows

PS C:\WINDOWS\system32> Get-ExecutionPolicy
Restricted
```

Si devuelve *Restricted*, entonces se ejecuta:
Set-ExecutionPolicy AllSigned

o mejor:

Set-ExecutionPolicy Bypass -Scope Process

A continuación, se ejecuta el siguiente commando:

```
Set-ExecutionPolicy Bypass -Scope Process -Force; [System.Net.ServicePointManager]::SecurityProtocol = [System.Net.ServicePointManager]::SecurityProtocol -bor 3072; iex ((New-Object System.Net.WebClient).DownloadString('https://community.chocolatey.org/install.ps1'))
```

[node.js]

Cuando finaliza el instalador se comprueba que se ha instalado correctamente abriendo una consola de comandos (*PowerShell, cmd...*) o desde el terminal de *VS Code*, y se ejecutan los siguientes comandos, uno tras de otro:

```
node -v
```

```
npm -v
```

```
PS C:\Users\ibm_h> node -v  
v18.18.0  
PS C:\Users\ibm_h> npm -v  
9.8.1  
PS C:\Users\ibm_h>
```

El sistema responderá con la versión instalada de cada uno de ellos.

- ❑ *node* se corresponde con Node.js
- ❑ *npm* es *node package management* o el administrador de paquetes de Node.js, con el que se puede realizar la instalación de diferentes complementos y *pluggins*, como *Webpack* o *Gulp*.



Cuando se utiliza Node.js para desarrollo web hay que crear una configuración, esta es una tarea que puede resultar pesada, pero una vez realizada puede guardarse para reutilizarla como configuración por defecto en futuros proyectos.

Para ello se utiliza *npm*.

Primero se debe crear una carpeta, que contendrá los ficheros de configuración por defecto y los diferentes paquetes y configuraciones que se vayan a usar.

Se crea la carpeta llamada, por ejemplo, *Proyecto*, y se arrastra o se abre con VS Code.

Se abre un terminal y se crea el fichero de configuración de Node.js con el siguiente comando:

```
npm init
```

o también

```
npm init -y
```

Mediante este comando se crea un archivo llamado *package.json* que indica Node.js el comportamiento de la aplicación: que debe ejecutarse cuando la aplicación se genera para producción, etc.



`npm init -y`, a diferencia de `npm init`, responde *yes* de manera automática a todas las preguntas o su valor por defecto.

El proceso de creación de la aplicación web consiste en responder a una serie de preguntas:

package name: (proyecto) pregunta por el nombre del proyecto, por defecto, le asigna el nombre del directorio.

version: (1.0.0) la versión de la aplicación que se está generando del proyecto, por defecto, la versión 1.0.0.

description: la descripción que explique en que consiste la aplicación u otra información que se considere de interés.

entry point: (index.js) es el punto de entrada a la aplicación, como también suele ser: *index.html*, *index.php*... Por defecto, *index.js*.



test command: si se quiere ejecutar algún comando para realizar pruebas de forma automática.

git repository: es el repositorio de GitHub que se va a usar, si es el caso.

keywords: son el conjunto de palabras claves que permiten encontrar la aplicación en el caso de que fuera un paquete para Node.js y se subiera a su repositorio.

author: es el programador/es o compañía que va a realizar la aplicación.

Por último, mostrará la configuración que se va a almacenar dentro del fichero *package.json* y que contiene las respuestas que se han ido indicando previamente. A continuación, preguntará si se está de acuerdo, en cuyo caso creará el fichero *package.json*, en caso contrario no lo creará y finalizará.

Este fichero se crea en la raíz de la carpeta, siempre puede borrarse y volverse a crear con el comando: *npm init*.



En resumen, el contenido de *package.json* es el siguiente:

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



La sección *scripts*, del fichero *package.json*, contiene los comandos que se pueden ejecutar desde el terminal mediante el comando *npm*:

```
"scripts": {  
  "test": "echo \"Error: no test specified\" && exit 1"  
}
```

Para ejecutar estos comandos se pone *npm run* seguido del comando a ejecutar:

```
npm run nombre_comando
```

npm init crea un comando por defecto llamado *test* que, en principio, no hace nada salvo mostrar un mensaje en la consola indicando que no se ha especificado test alguno. Se ejecuta de la siguiente manera:

```
npm run test
```



En la sección *scripts*, del fichero *package.json*, también puede existir *start* que *npm* puede ejecutar directamente:

```
"scripts": {  
  "test": "echo \"Error: no test specified\" && exit 1",  
  "start": "echo \"Error: no start specified\" && exit 1",  
}
```

Para ejecutar este comando basta con escribir *npm start*, seguido de los argumentos si los hubiera precedidos de *--* en la terminal:

```
npm start [--<args>]
```

Realmente es un alias del comando *npm run start*, por lo que estos dos comandos hacen lo mismo. Existen otros alias como *npm test*, *npm start*, *npm restart* y *npm stop*.



Una vez creada la configuración se pueden instalar los diferentes paquetes que se van a utilizar mediante el gestor de paquetes *npm* con el parámetro *install* o simplemente *i*.

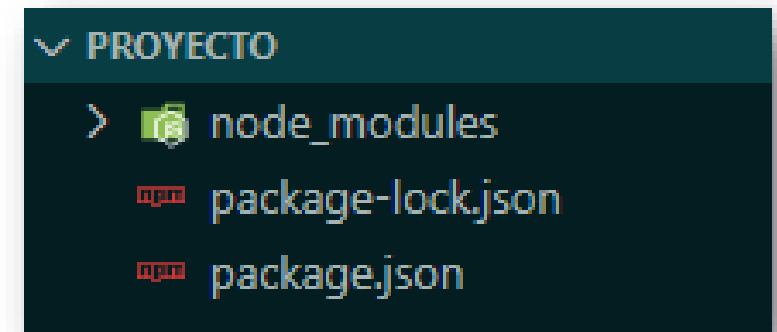
Desde un terminal de VS Code o desde cualquier otra consola de comandos (*PowerShell*, *cmd*...) se ejecuta el siguiente comando, pero hay que asegurarse que se hace desde el directorio raíz que contiene la aplicación:

```
npm i -D paquete1 [paquete2... ]
```

La opción *-D*, o también *--save-dev*, indica que el paquete se instalará como una dependencia de desarrollo y no de producción, esto significa que se utilizará el paquete como ayuda en el desarrollo de la aplicación y no como un paquete que necesita la aplicación en producción para funcionar en producción.

Este comando creará una carpeta llamada *node_modules* con los módulos del paquete y todos los módulos que se necesitan para que funcione (dependencias), además se habrá modificado el contenido del fichero del fichero *package.json* con la configuración pertinente.

Además, se crea el fichero *package-lock.json* que contiene las dependencias que necesitan los módulos instalados. Este fichero no debe modificarse.





Si ahora se observa el contenido del fichero `package.json`, se verá que se ha añadido *devDependencies*.

Esta sección indica a Node.js que el nombre y versión de los paquetes que en ella figura son necesarios para la fase de desarrollo (*development*) pero no cuando se generen los ficheros de la aplicación para producción.

```
{
  "name": "proyecto",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "devDependencies": {
    "paquete1": "^5.67.0",
    "paquete2": "^4.9.2"
  }
}
```



```
{
  "name": "proyecto",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "devDependencies": {
    "webpack": "^5.67.0",
    "webpack-cli": "^4.9.2"
  },
  "dependencies": {
    "@tailwindcss/jit": "^0.1.18",
    "autoprefixer": "^10.4.2",
    "postcss-cli": "^9.1.0",
    "tailwindcss": "^3.0.18"
  }
}
```



Cuando se instalan paquetes con *npm*, sin utilizar la opción *-D* o *--save-dev*, se indica que los paquetes son necesarios en producción y que serán necesarios al generar la aplicación final, como en el ejemplo de la izquierda.

En este caso el nombre y versión de los paquetes se encuentran en la sección *dependencies* del fichero *package.json*.



El comando *npm install* o *npm i* siempre va a instalar la última versión estable del paquete o paquetes que lo acompañan como argumento.

No obstante, a veces se hace necesario instalar una versión en concreto de un paquete del repositorio npm, por ejemplo, debido a que la última versión presenta errores o se conoce que una versión tiene alguna funcionalidad que en versiones superiores se ha perdido. En este caso, se puede utilizar el carácter *@* justo detrás del nombre del paquete y seguido por la versión que se quiere instalar:

```
npm i paquete@version
```

Por ejemplo:

```
npm i -D gulp@4.0.2
```

También, puede querer actualizarse o instalar un paquete concreto a la última versión:

```
npm i -D paquete@latest
```

Visualizar la versión instalada y si hay alguna versión nueva de los paquetes instalados:

```
npm outdated
```

O actualizar todos los paquetes instalados:

```
npm update
```



Cuando se use la configuración por defecto como punto de inicio de un nuevo proyecto, *npm* instalará todos los paquetes que se necesiten con el comando: *npm install*, satisfaciendo todas las dependencias, lo que incluye los paquetes de *devDependencies* y los de producción.

Una vez creada la aplicación para producción, dicha aplicación no necesita paquetes de desarrollo y otros módulos para funcionar, por lo que no se incluirán y, además, se hará un **Tree shaking** para eliminar todo lo superfluo, como los comentarios, útiles solo para el desarrollador, y así reducir el tamaño de la aplicación.



Tree shaking (sacudir el árbol) es un término que se utiliza en JavaScript para eliminar cosas innecesarias del código, como, por ejemplo, los comentarios, cuando se generan los ficheros de la de la aplicación para producción.



Debido a que Node.js es un entorno de ejecución multiplataforma para ejecutar JS, no sólo en un navegador web sino fuera de él, como en sistemas de escritorio o servidores web, existen algunas opciones específicas de los paquetes que son realmente utilidades o herramientas, que no se utilizan solo en proyectos web, y que se ejecutan como aplicaciones de línea de comandos (CLI) desde el terminal.

En estas situaciones se deben instalar los paquetes de forma global.

En esta modalidad, los paquetes se instalan a nivel del sistema (no en la carpeta del proyecto), por lo que estarán disponibles siempre que el usuario quiera utilizarlos, sin necesidad de tener que volverlos a instalar en cada proyecto.

Para instalar un paquete de forma global se utiliza el modificador `-g`:

`npm install -g paquete` o también `npm install --global paquete`



Si se dispone de una carpeta de proyecto web con la configuración y los ficheros fuentes ya creados y se quiere trabajar en otra máquina o crear una copia de seguridad, no es necesario copiar la carpeta que contiene los paquetes que utiliza el proyecto, *node_modules*, ya que son pesados y pueden bajarse e instalarse desde Internet de manera sencilla; para satisfacer estos paquetes en otra máquina o al restablecer una copia de seguridad se utiliza *npm* con el parámetro *install* sin acompañarlo de otras opciones:

```
npm install
```

De igual manera, para actualizar los paquetes de un proyecto se utiliza *npm* con el parámetro *update*:

```
npm update
```

Para desinstalar un paquete se utiliza el parámetro *uninstall* en vez de *install*:

```
npm uninstall paquete
```

Con el parámetro *-g* se indica que se desinstale de manera global:

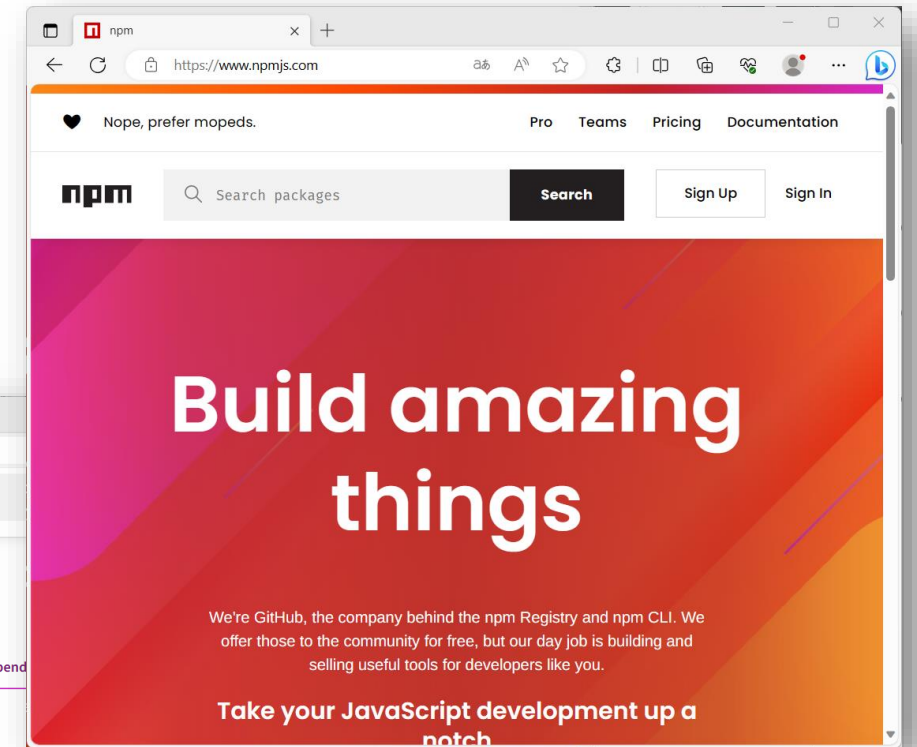
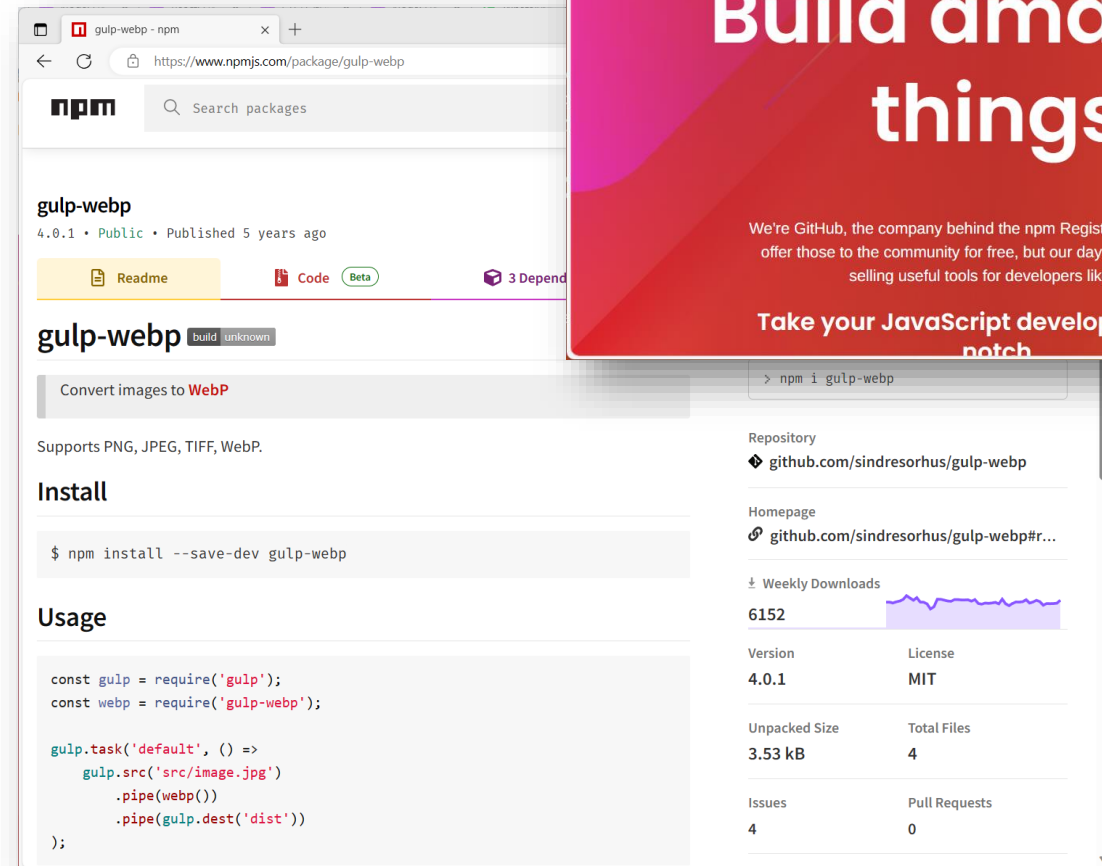
```
npm uninstall -g paquete
```



El sitio web oficial del repositorio de paquetes de Node.js es npmjs.com

En este repositorio existe mucha información de los paquetes disponibles para Node.js

Muestra la ayuda para la instalación, cómo usarlo o configurarlo,... e información variada como acceso al código fuente, el número de descargas del paquete, etc.





Junto con la instalación de *npm* se instala *npx* (*Node Package Executor*), una herramienta de línea de comandos cuyo propósito es facilitar la instalación y la gestión de los paquetes del repositorio *npm*.

npx, a diferencia de *npm*, permite ejecutar archivos binarios de paquetes, pero sin la necesidad de instalarlos de forma global para que funcionen correctamente.



Por ejemplo, cuando se quiere trabajar en más de un proyecto que depende de un paquete instalado globalmente hay que decidir si se debe actualizar la versión instalada o quedarse con la versión previamente instalada.

Con *npx* pueden convivir varias versiones del mismo paquete ya que el paquete se instala de modo local:

```
npm install --save-dev gulp
```

y cada vez que se quiera ejecutar la versión local:

```
npx gulp
```



npx permite ejecutar una herramienta que no está instalada y que solo se necesita una vez, sin tener que instalarla:

npx comando

Si el comando no está instalado entonces se instalará automáticamente desde el repositorio de npm y se ejecutará de inmediato.

Al terminar el comando ya no estará ahí y no hay que preocuparse porque queden archivos de instalación olvidados en el sistema.

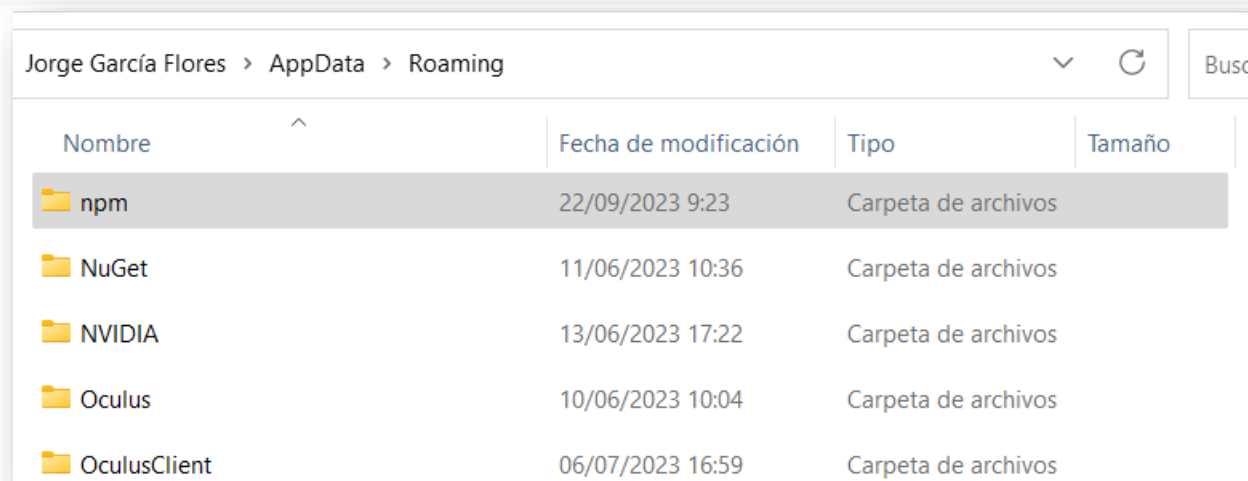




El siguiente error se soluciona creando una carpeta llamada *npm* en la ruta que presenta el problema:

```
PS C:\Users\ibm_h\Desktop> npx create-react-app hola-mundo
npm ERR! code ENOENT
npm ERR! syscall lstat
npm ERR! path C:\Users\ibm_h\AppData\Roaming\npm
npm ERR! errno -4058
npm ERR! enoent ENOENT: no such file or directory, lstat 'C:\Users\ibm_h\AppData\Roaming\npm'
npm ERR! enoent This is related to npm not being able to find a file.
npm ERR! enoent

npm ERR! A complete log of this run can be found in: C:\Users\ibm_h\AppData\Local\npm-cache\_logs\2023-09-22T07_20_32_807Z-debug-0.log
```



Jorge García Flores > AppData > Roaming

Nombre	Fecha de modificación	Tipo	Tamaño
npm	22/09/2023 9:23	Carpeta de archivos	
NuGet	11/06/2023 10:36	Carpeta de archivos	
NVIDIA	13/06/2023 17:22	Carpeta de archivos	
Oculus	10/06/2023 10:04	Carpeta de archivos	
OculusClient	06/07/2023 16:59	Carpeta de archivos	



yarn es un gestor de paquetes enfocado en la velocidad y la seguridad.

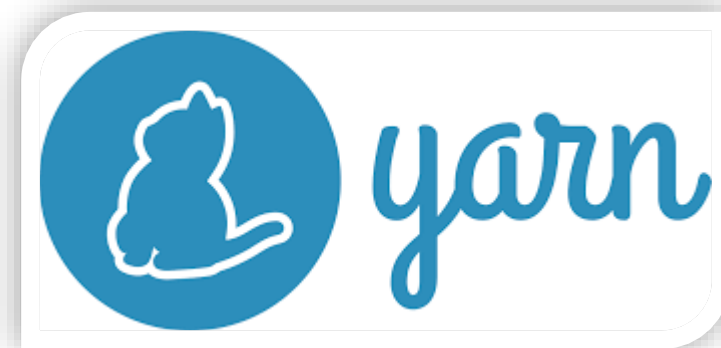
A diferencia de otros gestores, como *npm*, es muy rápido y muy fácil de usar.

Fue creado por los equipos de Facebook y ofrecido a la comunidad *open source* en 2016.

yarn utiliza, por defecto, el registro de *npm* por lo que no necesita ninguna configuración adicional.

Es muy rápido porque almacena en la cache los paquetes descargados y realiza operaciones en paralelo para maximizar la utilización de recursos.

Ofrece al usuario una interfaz muy vistosa y fácil de leer, con gran uso de colores e iconos a pesar de ser una herramienta de consola.



yarn se traduce como hilo en español.

yarn permite usar y compartir código con otros desarrolladores de cualquier parte del mundo.

Se puede instalar *yarn* de dos maneras diferentes:

- ☐ Descargando e instalando el paquete de instalación:

<https://yarnpkg.com/latest.msi>

<https://yarnpkg.com/latest.rpm>

<https://yarnpkg.com/latest.deb>

- ☐ Mediante *npm*:

```
npm install --global yarn
```

Para comprobar que *yarn* está instalado en el sistema:

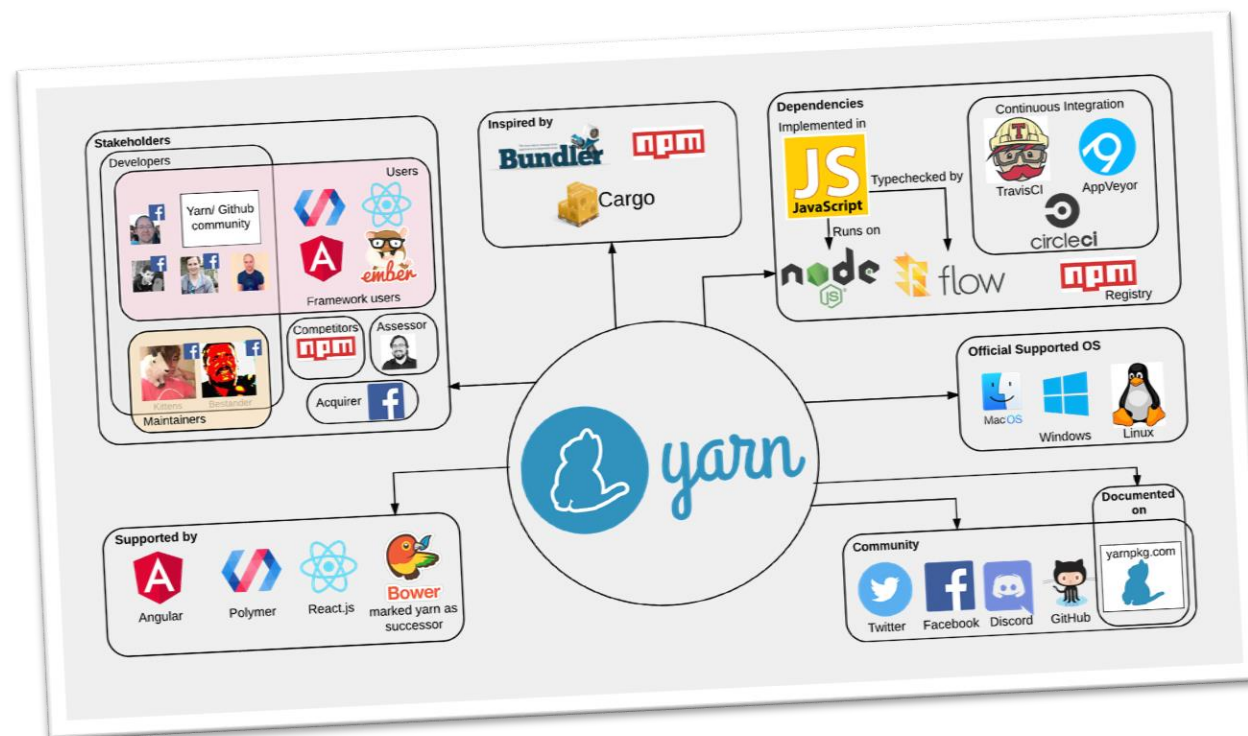
```
yarn --version
```



Para crear un nuevo proyecto con *yarn* se ejecuta el comando:

`yarn init`

```
PS C:\Users\ibm_h\Desktop> yarn init
yarn init v1.22.19
question name (Desktop): Prueba
question version (1.0.0):
question description:
question entry point (index.js):
question repository url:
question author: jgarciafl@educa.jcyl.es
question license (MIT):
question private:
success Saved package.json
Done in 30.88s.
PS C:\Users\ibm_h\Desktop>
```



Se puede utilizar *yarn* incluso si se estaba trabajando con *npm*, lo único que hay que hacer es borrar el archivo *package-lock.json*.



Instalar todas las dependencias de un proyecto se puede usar cualquiera de los dos comandos:

`yarn`

`yarn install`

Para instalar dependencias:

`yarn [global] add [package]` ejemplo: `yarn global add nodemon`
`yarn add [package]@[version]` ejemplo: `yarn add vue@[2.6.8]`
`yarn add [package] [--dev | -D]` ejemplo: `yarn add vue --dev`

Actualizar dependencias:

`yarn upgrade [package]` ejemplo: `yarn upgrade vue`

Eliminar dependencias :

`yarn remove [package]` ejemplo: `yarn remove vue`

[PNpM]

pnpm (*Performant NPM*) es un gestor de paquetes que es hasta dos veces más rápido que NPM, dedicado principalmente al ahorro de espacio en disco.

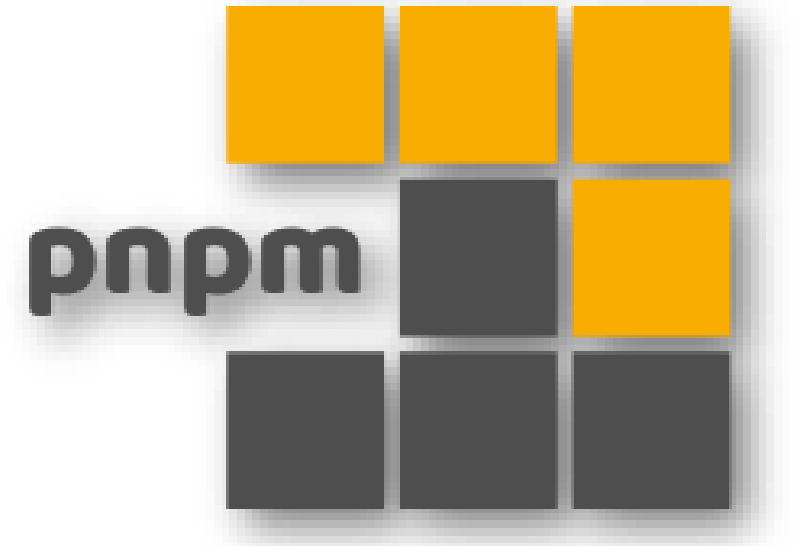
Cuando se usa *npm* o *yarn* si se tienen 100 proyectos que usan una misma dependencia se tendrían 100 copias de esa dependencia guardadas en el disco.

pnpm guarda la dependencia en un almacén direccionable por contenido, por lo que solo existirá un paquete para satisfacer la dependencia en todos los proyectos. Además, todos los paquetes se guardan en un solo lugar en el disco, por lo que no se consume espacio adicional en el disco.

Por defecto, *pnpm* usa enlaces simbólicos para agregar solo las dependencias directas del proyecto en el directorio *node_modules*.

Se instala mediante el comando:

```
npm install -g pnpm
```



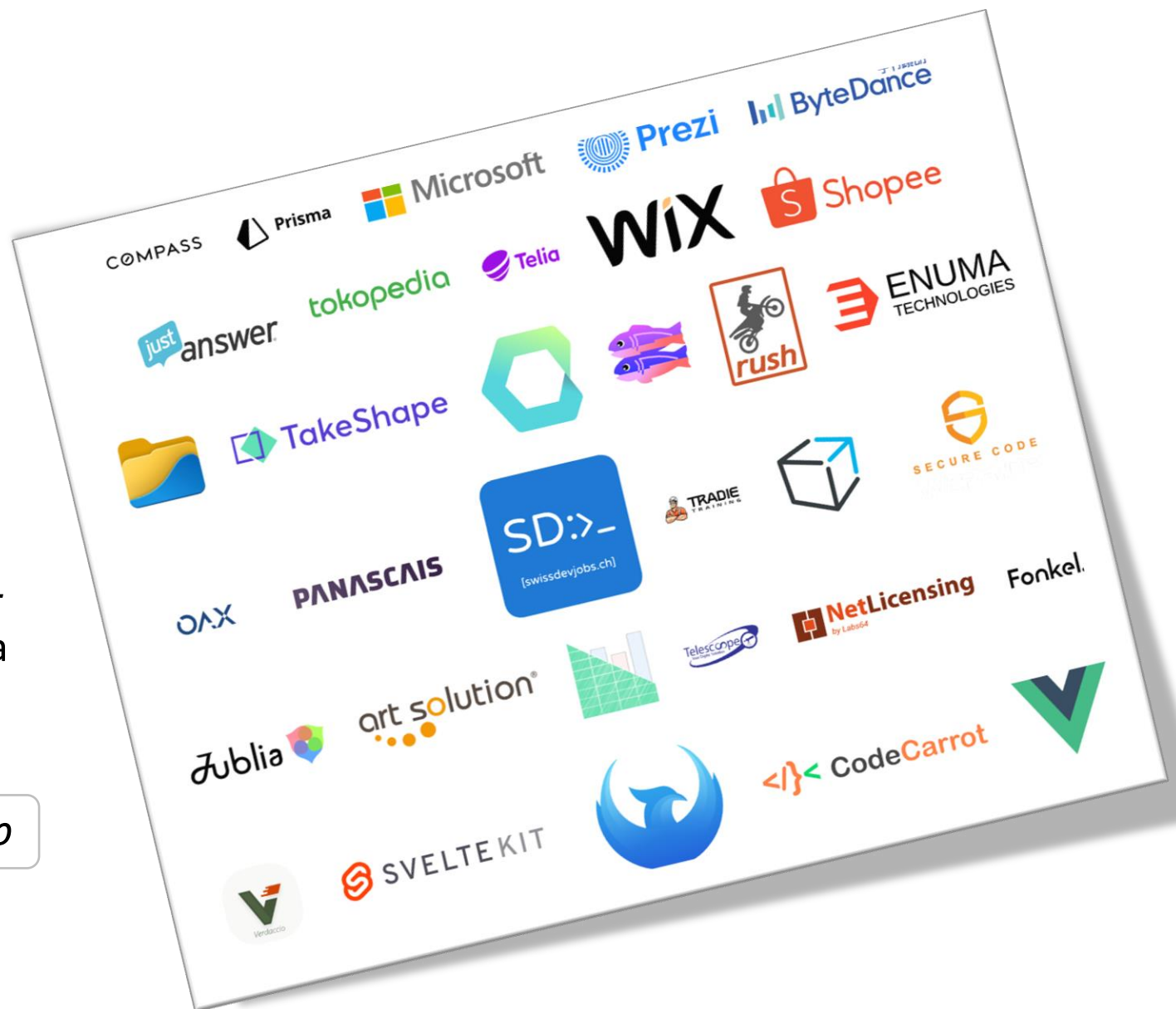
[PNpM]

comando npm	equivalente pnpm
<code>npm install</code>	<code>pnpm install</code>
<code>npm i <pkg></code>	<code>pnpm add <pkg></code>
<code>npm run <cmd></code>	<code>pnpm <cmd></code>

Ejemplo, para instalar el paquete *create-react-app* y crear una aplicación llamada *my-app*:

```
pnpm install -g create-react-app
```

```
pnpx create-react-app my-app
```



Bibliografía y recursos online

- <https://nodejs.org/es/>
- <https://opensource.org/licenses/ISC>
- <https://blog.logrocket.com/install-node-windows-chocolatey/>
- <https://creativecommons.org/>
- <https://yarnpkg.com/>
- <https://dev.to/juandelgado06/guia-rapida-de-yarn-5alj>
- <https://pnpm.io/>