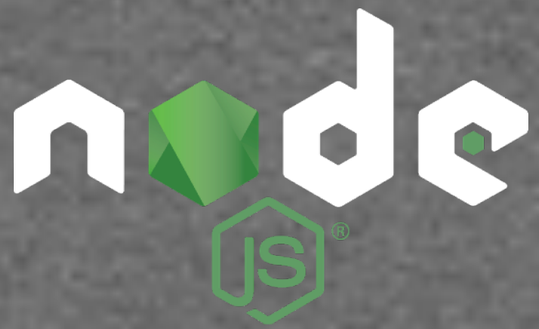
BIENVENIDO



Express.js

Javier Miguel

@JavierMiguelG

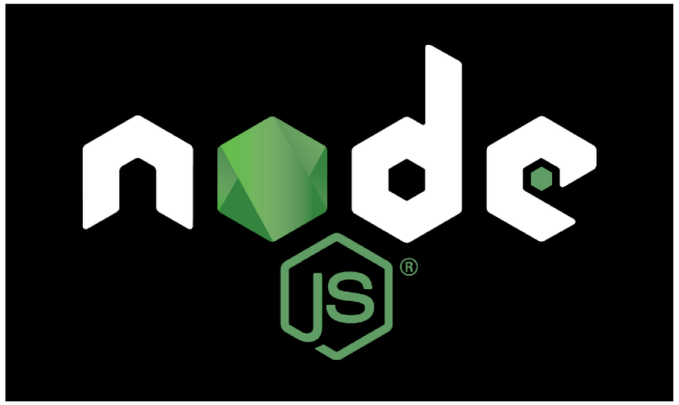
Node.js

Parte I

Javier Miguel

- CTO & Freelance Developer
- Email: jamg44@gmail.com
- Twitter: @javiermiguelg

Que es Node.js?



- Es un interprete de Javascript
- Inicialmente diseñado para correr en servidores
- Asíncrono
- Orientado a eventos
- Basado en el motor V8 de Google

¿Orientado a eventos?

- En la programación secuencial es el programador quien decide el flujo
- En la programación orientada a eventos, el usuario o los programas clientes son quienes definen el flujo

Servidores de aplicaciones

- En otros lenguajes se utiliza un servidor de aplicaciones donde cargamos la nuestra, como Tomcat, IIS, etc.
- También hay múltiples lenguajes que se usan como una extensión de un servidor web como Apache, nginx, etc, que es quien maneja las conexiones, como por ejemplo PHP.
- **En Node.js nuestra aplicación realiza directamente todas las funciones de un servidor web o de aplicaciones.**

Motor V8

Motor Javascript creado por Google para Chrome.

Escrito en C++.

Multiplataforma (Win, Linux, Mac)

Versiones de Node.js

- Versiones 6.x —> LTS (Long Term Service)
- Versiones 7.x —> Estables, con la últimas novedades
- A partir de la versión 4.0 se incorporan muchas de las nuevas características de ECMAScript versión 6

Límite de memoria

Actualmente, por defecto tiene un límite de memoria de 512 MB en sistemas de 32 bits, **1gb en sistemas de 64 bits**.

El límite se puede aumentar mediante el establecimiento de --*max_old_space_size* un máximo de ~ 1,024 (~ 1 GiB) (32 bits) y ~ 1,741 (~ 1.7GiB) (64 bits), pero se recomienda dividir el proceso en varios workers si se llega a los límites.

Ventajas de Node.js

Node.js tiene grandes ventajas reales en:

- Aplicaciones de red, como APIs, servicios en tiempo real, servidores de comunicaciones, etc
- Aplicaciones cuyos clientes están hechos en Javascript, pues compartimos código y estructuras entre el servidor y el cliente.

Instalando Node.js

Distintas formas

Se puede hacer:

- Desde el instalador oficial en nodejs.org
- Con un instalador de paquetes
- Compilando manualmente

Distintas formas - ¿Como decido?

Instalador:

- Más sencillo de instalar

Homebrew

- Se instala sin sudo
- Más fácil de desinstalar

A través del instalador

- Ir a https://nodejs.org/
- Install —> descargar
- Lanzar .pkg|.exe
- Siguiente, siguiente, …

(Solo OSX y Windows)

Instalará los ejecutables de node y npm en la ruta:

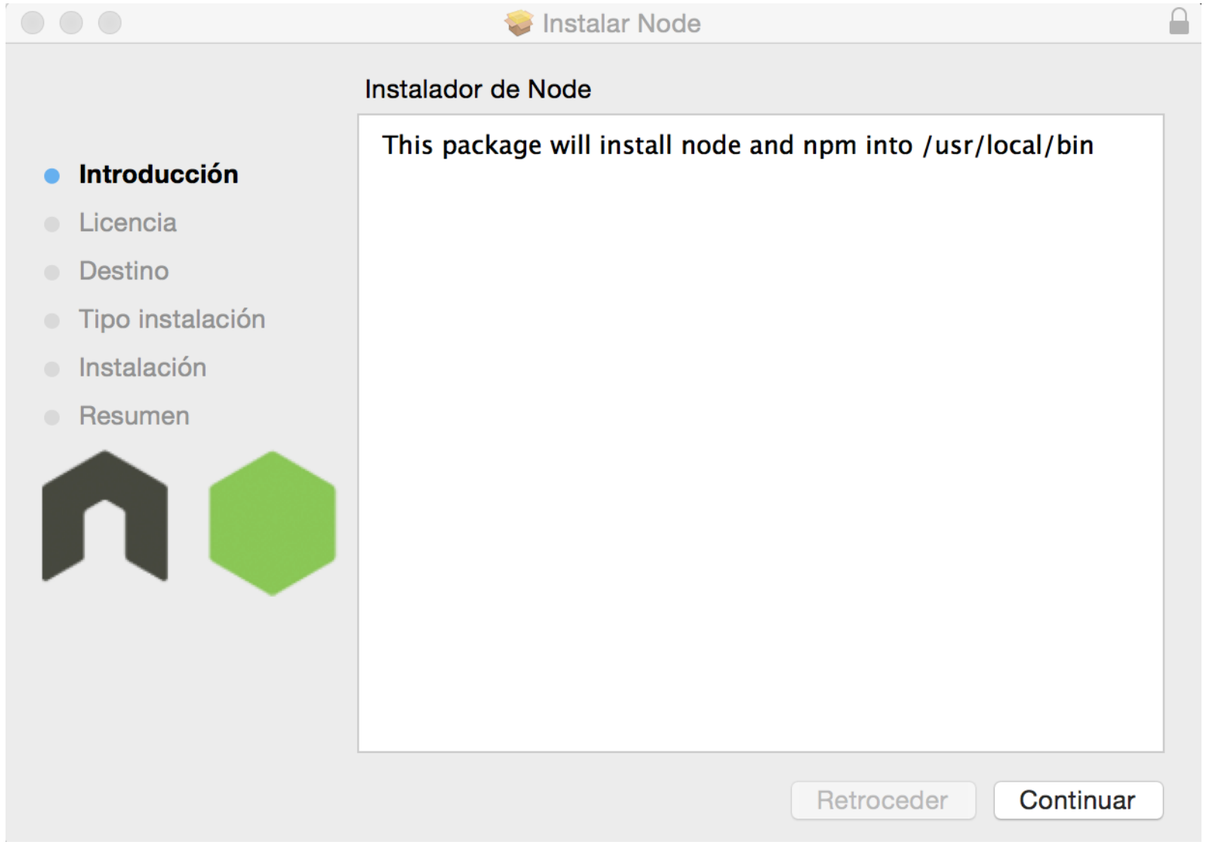
/usr/local/bin

Pondrá la carpeta node_modules global en:

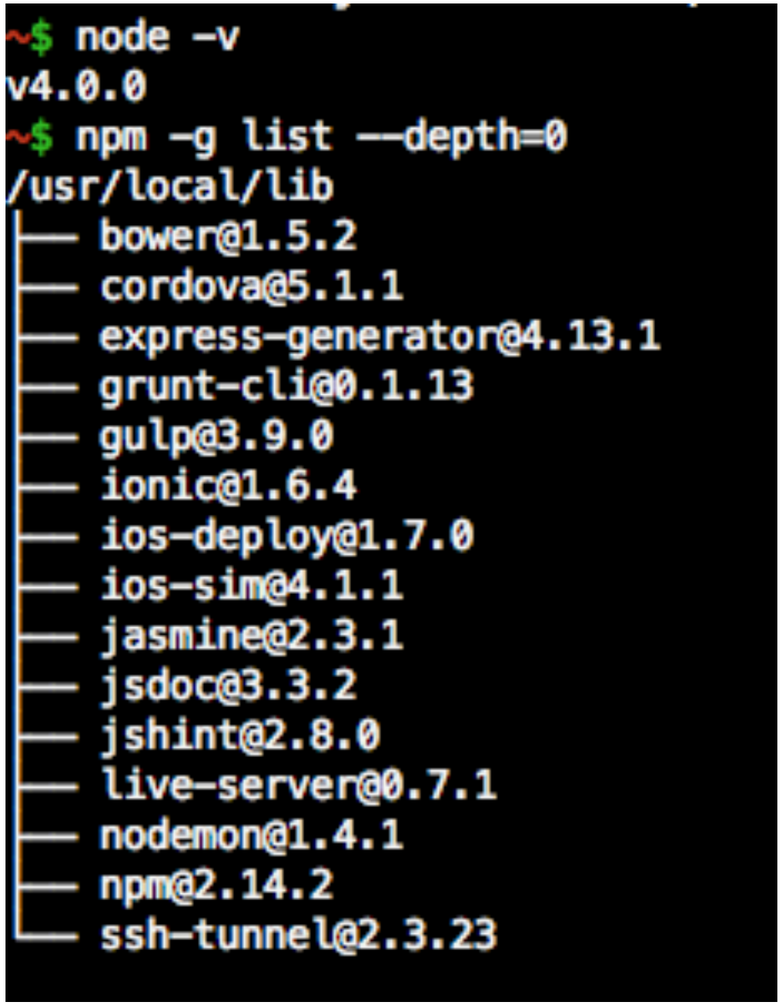
/usr/local/lib

Vínculo dinámico de npm en bin:

```
npm@ -> ../lib/node_modules/npm/bin/npm-cli.js
```



A partir de aquí ya podremos ejecutar:



A través del instalador - actualizar

Para actualizar se repite el proceso:

- Descargar el instalador de la versión elegida
- Instalar encima
- Comprobar la nueva versión

A través del instalador - desinstalar

Se ha de hacer manualmente.

Hay varios scripts que nos ayudan, por ejemplo:
`https://gist.github.com/nicerobot/2697848`

Nota importante:
Siempre que usemos un script de un tercero, **revisar su código y entender que es lo que hace!**
(antes de ejecutarlo)

A través de un instalador de paquetes

Pre-requisitos:

- Xcode

- Homebrew http://brew.sh/

```
ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"
```

(seguir mensajes en la consola para terminar su instalación)

A través de un instalador de paquetes

Instalación:

1. Abre el Terminal
2. Escribe:

```
brew install node
```

Tras la instalación comprueba las versiones:

```
~$ node -v
v0.12.7

~$ npm -g list --depth=0
/usr/local/lib
├── bower@1.4.1
├── cordova@5.1.1
├── grunt-cli@0.1.13
├── gulp@3.9.0
├── ionic@1.6.4
├── ios-deploy@1.7.0
├── ios-sim@4.1.1
├── jasmine@2.3.1
├── live-server@0.7.1
├── nodemon@1.4.1
├── npm@2.11.3
└── ssh-tunnel@2.3.23
```

A través de un instalador de paquetes - actualizar

Actualización:

1. Abre el Terminal
2. Escribe:

```
brew update
```

3. Luego:

```
brew upgrade node
```

Tras la actualización comprueba las versiones.

A través de un instalador de paquetes - desinstalar

Desinstalación:

1. Abre el Terminal
2. Escribe:

```
brew uninstall node
```

io.js junto con node.js

io.js se acaba de mergear recientemente con node.js para producir una única rama de desarrollo.

Si aún así quisiéramos tener instalado conjuntamente node.js v0.12 con node.js v4.0, podemos usar por ejemplo nvm para que nos ayude a gestionarlo.

nvm

Instalar

- En linux/mac - https://github.com/creationix/nvm
- En windows - https://github.com/coreybutler/nvm-windows

```
nvm list

nvm install <version>

nvm use <version>

nvm use system # en linux
```

Un servidor básico

Ejercicio

Ejecutar

```
$ node index.js
```

Con nodemon:

```
$ npm install nodemon -g
$ nodemon
```

NPM

NPM

Node Package Manager es un gestor de paquetes que nos ayuda a gestionar las dependencias de nuestro proyecto.

Entre otras cosas nos permite:

- Instalar librerías o programas de terceros
- Eliminarlas
- Mantenerlas actualizadas

Generalmente se instala conjuntamente con Node.js de forma automática.

NPM - package.json

Se apoya en un fichero llamado package.json para guardar el estado de las librerías.

```
npm init
```

Crea este fichero.

Documentación en https://docs.npmjs.com/files/package.json

NPM - package.json

```
$ npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sane defaults.

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

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

Press ^C at any time to quit.
name: (myapp)
version: (1.0.0)
description: Esta es la descripción
entry point: (index.js) index.js
test command:
git repository:
keywords:
author: Javier Miguel
license: (ISC)
About to write to \Users\javi\www\cursonode\keepcoding\myapp\package.json:
```

NPM - package.json

```
{
  "name": "myapp",
  "version": "1.0.0",
  "description": "Esta es la descripción",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "Javier Miguel",
  "license": "ISC"
}
```

NPM - package.json

Instalar una librería (como por ejemplo chance)

```
npm install chance --save
```

```
{
  "name": "myapp",
  "version": "1.0.0",
  "description": "Esta es la descripción",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "Javier Miguel",
  "license": "ISC"
  "dependencies": {
    "chance": "^0.8.0"
  }
}
```

NPM - global o local

Instalación local, en la carpeta del proyecto

```
npm install <paquete> [--save]
```

Instalación global (en */usr/local/lib/node_modules*)

```
npm install -g <paquete>
```

Si el paquete tiene ejecutables hará un vínculo a ellos en /usr/local/bin

NPM - modificadores de versión

Cada librería lleva un identificador de versión.

```
"dependencies": {
  "chance": "^0.7.6"
},
```

NPM las parsea usando Semver (https://github.com/npm/node-semver)

Por defecto se establece un **Caret Range**:

- Ejemplos **"^1.2.3" "^0.2.5" "^0.0.4"**
- **Allows changes that do not modify the left-most non-zero digit in the [major, minor, patch] tuple.**

NPM - versiones

Fijar versiones de dependencias y subdependencias

```
npm shrinwrap
```

Se incluye habitualmente en git

Instalando un módulo

Ejercicio

Instalando un módulo

Instalar el módulo chance (https://github.com/victorquinn/chancejs) y usar su generador de nombres en el servidor básico.

```
[npm init]
```

```
npm install chance [--save]
```

Process

Process

El objeto global process tiene muchas propiedades que nos serán útiles, como `process.platform`, que en OS X nos dirá 'darwin', en linux 'linux', etc.

También tiene métodos útiles como `process.exit(int)` que para node estableciendo un exit code.

O eventos como `process.on('exit', callback)` donde podemos hacer cosas antes de salir.

ejemplos/process.js

Process

Tiene un método muy interesante, `process.nextTick`, que recibe un solo argumento, un callback.

```
process.nextTick(function () {
    console.log('Siguiente vuelta del event loop, whoooouuu!')
});
```

Lo que hará es colocar la función de nuestro callback al principio de la siguiente vuelta del event loop (ahora vemos que es eso).

Event loop

Event loop

Node usa un solo hilo.

Tiene un bucle interno, que podemos llamar 'event loop' donde en cada vuelta ejecuta todo lo que tiene en esa 'fase', dejando los callbacks pendientes para otra vuelta.

La siguiente vuelta mira a ver si ha terminado algún callback y si es así ejecuta su handlers.

Event loop

El equivalente de esta construcción

```
process.nextTick(function () {
    console.log('Siguiente vuelta del event loop, whoooouuu!')
});
```

Es como decir a node "cuando vuelvas a comprobar los callbacks finalizados haz esto!".

Non blocking

Si node se quedara esperando hasta que termine una query o una petición a Facebook, acumularía demasiados eventos pendientes y dejaría de atender a las siguientes peticiones, ya que como dijimos **usa un solo hilo**.

Por eso todos las llamadas a funciones que usan IO (por ejemplo escritura o lectura en disco, la red, bases de datos, etc) se hacen de forma asíncrona. Se aparcan para que nos avisen cuando terminen.

Evented IO

Evented IO

Node nos proporciona una forma de manejar IO en forma de eventos.

Usando el EventEmitter podemos colgar eventos de un identificador.

```
eventEmitter.on('llamar telefono', suenaTelefono);

// suenaTelefono es una función
```

Evented IO

Y podemos emitir el identificador cuando queremos que sus eventos 'salten'

```
eventEmitter.emit('llamar telefono');
```

Evented IO

Incluso podemos darle argumentos al identificador para que se los dé a los manejadores.

```
eventEmitter.emit('llamar telefono', 'madre');
```

Nuestro manejador recibirá el parametro

```
var suenaTelefono = function(quien) {
    ...
}
```

ejemplos/eventemitter.js

Evented IO

Son cómodos para procesar streams.

Mandando a un fichero todo lo que se escriba en la consola.

```
process.stdin.on('data', function(data) {
  file.write(data);
});
```

ejemplos/eventedio.js

Módulos

Módulos

Los módulos de Node.js se basan en el estándar **CommonJS**.

- Los módulos usan **`exports`** para exportar cosas.
- Quien quiere usar un módulo lo carga con **`require`**.
- La instrucción `require` es **síncrona**.

Módulos

Un módulo básico:

```
// modulo.js
console.log('Hola desde un módulo!');


// index.js
require('./modulo.js');
```

ejemplos/modulos/basico.js

Módulos

Donde busca Node.js los módulos

1. Si es un módulo del core
2. Si empieza por './' or '/' or '../' va a la ruta calculada desde la situación del fichero actual
3. Módulos en la carpeta node_modules local
4. Módulos en la carpeta node_modules global

Módulos

```
require('../../../../../lib/files/modulo.js');
```

Como evitar esto? Asignando NODE_PATH a la ruta donde meteremos nuestras librerías en el arranque de nuestra app.

NODE_PATH=lib node index.js

```
require("files/modulo.js"); // mejor así!
```

Módulos

O si usamos npm a partir de 2.0 podemos anotarlas en el package.json

```
{
  "name": "baz",
  "dependencies": {
    "bar": "file:../foo/bar"
  }
}
```

https://docs.npmjs.com/files/package.json#local-paths

Módulos

Formas de hacer disponible el contenido del módulo

Módulos

Exportar una función anónima:

```
module.exports = function () {
    console.log('Hola desde una función anónima!');
}
```

Exportar una función con nombre:

```
exports.nombre = function () {
    console.log('Ahora con nombre!');
}
```

ejemplos/modulos/index.js

Módulos

Exportar un objeto anónimo:

```
var Objeto = function () {};

Objeto.prototype.salta = function () {
  console.log('Objeto anonimo salta!');
};

module.exports = new Objeto();
```

ejemplos/modulos/objeto_anonimo.js

Módulos

Exportar un objeto con nombre:

```
var Objeto = function () {};

Objeto.prototype.salta = function () {
  console.log('Objeto anonimo salta!');
};

exports.objeto = new Objeto();
```

ejemplos/modulos/objeto_nombre.js

Módulos

Exportar un constructor anónimo:

```
var Constructor = function () {};

Constructor.prototype.salta = function () {
    console.log('Contructor anÃ³nimo salta!');
}

module.exports = Constructor;
```

ejemplos/modulos/ConstructorAnonimo.js

Módulos

Exportar un constructor con nombre:

```
var Constructor = function () {};

Constructor.prototype.salta = function () {
    console.log('Contructor con nombre salta!');
}

exports.Constructor = Constructor;
```

ejemplos/modulos/ConstructorNombre.js

Módulos

¿Cual elegir?

- Exports con nombre —> un solo módulo, varias cosas exportadas
- Exports anónimos —> interfaz de cliente más sencilla

Módulos

¿module.exports o exports?

- `exports` es un alias de `module.exports`.
- Node lo crea automáticamente
- Para hacer `exports` con nombre se pueden usar los dos indistintamente

Si asignamos algo directamente a `exports` deja de ser un alias. Solo podemos usarlo con `exports.loquesea`

Módulos

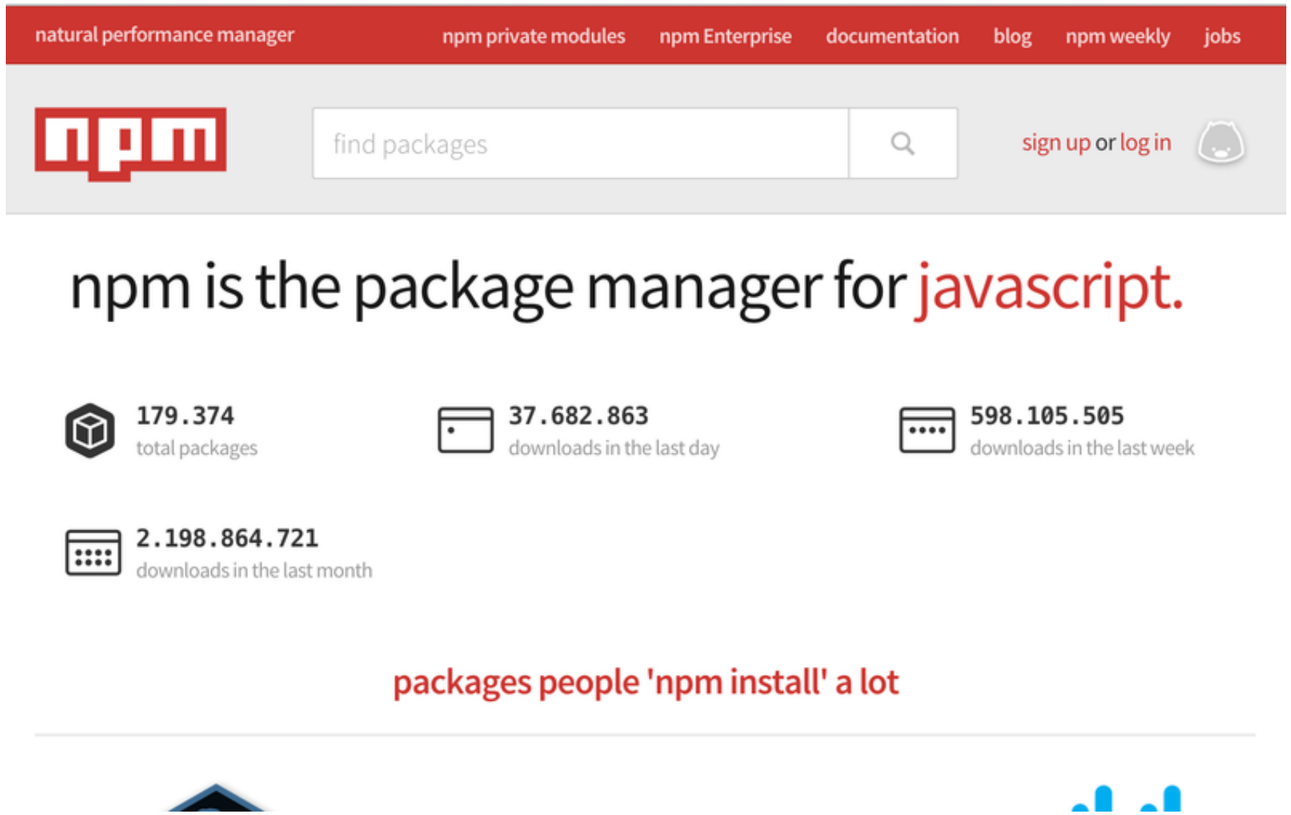
Node carga un módulo una sola vez. En el primer require.

Las siguientes llamadas a require con la misma ruta devolverán el mismo objeto que se creo en la primera llamada.

Esto nos vendrá muy bien con las conexiones a bases de datos, por ejemplo.

Módulos

Principalmente podemos encontrar módulos de terceros en npmjs.com



Ejercicio

Creando un módulo

Creando un módulo

Crear un módulo con la función versionModulo que consiga la versión de un módulo instalado en node_modules local.

Creando un módulo

Crear otro módulo con la función versionModulos que consiga las versiones de los módulos instalados en node_modules local.