

# Administradores de Paquetes (Package Managers)

## Teoría

Los *Package Managers* (o Administradores de paquetes) sirven para no tener que descargar, instalar y mantener las dependencias de un proyecto a mano. Estas aplicaciones facilitan la descarga e instalación de las librerías que utiliza el proyecto. Para ello, requiere que conozcamos el nombre exacto de la librería (y versión deseada si es necesario), y contar con conexión a Internet. Luego, con solo ejecutar un comando, se descargará de un repositorio centralizado la versión correspondiente de la dependencia especificada y se agregará al proyecto.

NodeJS cuenta con su propio Administrador de Paquetes: *NPM (NodeJS Package Manager)*.

## Instalando dependencias con NPM

Para poder instalar una dependencia usando NPM debemos tener en cuenta lo siguiente: las dependencias pueden instalarse en forma *global* o *local*.

### Global:

```
$ npm install -g nombre-de-la-librería
```

### Local:

```
$ npm install nombre-de-la-librería
```

Si instalamos una dependencia en forma global, todos nuestros programas desarrollados en NodeJS contarán con esa librería, **y con la versión que haya sido instalada**. En cambio, si instalamos en forma local, podremos elegir exactamente qué librería y con qué versión contará cada proyecto que desarrollemos.

Esta segunda opción es la más recomendable, ya que de esta manera podemos tener distintos proyectos usando distintas versiones de una misma librería, sin generar problemas de compatibilidad al actualizar a una nueva versión que no sea retrocompatible con las anteriores.

### **Ejemplo:**

*El programa A usa la librería “fecha” en su versión 1.0, que cuenta con el método “dameFecha()”. El programa B usa la librería “fecha” pero en su versión 2.0, que ya no cuenta con el método “dameFecha()” ya que éste fue reemplazado por el nuevo método “dameFechaLocal()”.*

*Si instalamos “fecha” en forma global, al actualizar la librería fecha, romperemos el programa A, ya que intentará usar un método que ya no existe.*

*Si realizamos dos instalaciones locales (fecha 1.0 para A, fecha 2.0 para B) al actualizar cada una por separado, cada programa seguirá contando con la versión correspondiente).*

Sin embargo, muchas veces es útil instalar en forma **global** librerías **utilitarias** (por ejemplo librerías de *testing*) que son usadas para facilitar las tareas de programación y revisión durante la etapa de desarrollo pero que no son necesarias para el uso de la aplicación.

#### **Advertencia:**

*Es posible que al instalar dependencias en forma global se nos solicite tener permisos de administrador, ya que estaremos editando archivos de configuración y agregando contenidos en carpetas del sistema.*

## El archivo “package.json”

Cómo ya sabemos, los proyectos de NodeJS cuentan con un **archivo de configuración** en formato JSON que, entre otras cosas, permite especificar el nombre del proyecto, versión, nombre del autor, etc.

Una de las cosas que podemos especificar en este archivo es la lista de dependencias. Esto es, una lista con los nombres de las librerías (con sus versiones) que el proyecto usa para su funcionamiento.

Adicionalmente, también permite especificar una lista de dependencias “dev” (developer, desarrollador) que son librerías que si bien no son necesarias para funcionamiento del sistema, son utilizadas por el desarrollador para ir probando sus funcionalidades a medida que avanza el desarrollo (por ejemplo, librerías de testing).

## Manejo automatizado de dependencias

Siempre que hayamos especificado nuestras dependencias en el archivo de configuración (*package.json*) podremos actualizar y mantener de forma fácil y segura las dependencias del proyecto. Con sólo ejecutar un comando, y siempre contando con una conexión a internet funcionando, se descargarán e instalarán/actualizarán las versiones correspondientes de cada dependencia especificada, tanto dependencias de la versión de producción, como de la versión dev.

Para instalar y/o actualizar las dependencias de un proyecto debemos ejecutar:

```
$ npm install
```

Si solo queremos instalar las dependencias imprescindibles de un proyecto (es decir, dejando afuera las dependencias de desarrollador) debemos agregar el flag *--production*, así:

```
$ npm install --production
```

Además, podemos hacer que npm agregue como dependencia al package.json un módulo que estamos instalando, todo en una misma acción. Si lo queremos como dependencia del proyecto, agregaremos al comando 'install' la opción --save. En cambio, si sólo es una dependencia del entorno de desarrollo, agregaremos --save-dev. Ejemplo:

```
$ npm install --save <algún-módulo>
$ npm install --save-dev <algún-módulo-del-desarrollador>
```

## Versionado

Las librerías de NPM siguen un estándar de versionado que sigue la siguiente semántica: Consta de 3 números, separados entre sí por un punto:

- El primer número corresponde a actualizaciones grandes/significativas (*Major Release*), que incluyen muchas nuevas características, o que cambian de manera significativa el funcionamiento de las existentes.
- El segundo número corresponde a actualizaciones pequeñas (*Minor Release*), que agregan pocas cosas nuevas o actualizan algún detalle del funcionamiento de la librería.
- El tercer número corresponde a arreglos o parches (*Patches*), que corrigen defectos en las funcionalidades de la librería.

## Ejemplo de archivo package.json con dependencias

```
{
  "type": "module",
  "name": "mi-proyecto",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "express": "^4.16.4",
    "joi": "~14.3.1",
    "sleep": "*6.0.0"
  },
  "devDependencies": {
```

```
"jest": "latest"
}
```

Siguiendo este criterio, NPM ofrece una semántica especial para especificar los módulos que el proyecto necesita tener instalados para poder funcionar (sus dependencias) y la versión de cada uno de ellos, todo esto definido en su archivo de configuración *package.json*.

Como notarán en el ejemplo anterior, cada una de las versiones de las dependencias está precedida por un símbolo. Este símbolo condiciona la versión del módulo que se va a instalar cada vez que ejecutemos `npm install` en el proyecto, siguiendo la siguiente tabla:

#### Sin símbolos

Si no se pone ningún símbolo, se descarga **únicamente** la versión especificada.

#### ~ (permite instalar también patches a la version especificada)

Si escribimos en nuestro *package.json*: `~0.13.0`

- Si existe la versión *0.13.1* **se descargará la 0.13.1, ya que es un Patch**
- Si existe la versión *0.14.0* **se descargará la 0.13.0, ya que la 0.14.0 es una Minor Release**
- Si existe la versión *1.1.0* **se descargará la 0.13.0, ya que la 1.1.0 es una Major Release**

#### ^ (permite patches y actualizaciones menores)

Si escribimos en nuestro *package.json*: `^0.13.0`

- Si existe la versión *0.13.1* **se descargará la 0.13.1, ya que es un Patch**
- Si existe la versión *0.14.0* **se descargará la 0.14.0, ya que es una Minor Release**
- Si existe la versión *1.1.0* **se descargará la 0.13.0, ya que la 1.1.0 es una Major Release**

#### \* (permite todas las actualizaciones)

Si escribimos en nuestro *package.json*: `*0.13.0`

- Si existe la versión *0.13.1* **se descargará la 0.13.1, ya que es un Patch**
- Si existe la versión *0.14.0* **se descargará la 0.14.0, ya que es una Minor Release**
- Si existe la versión *1.1.0* **se descargará la 1.1.0, ya que es una Major Release**

#### Si en lugar de escribir una versión, se escribe 'latest'

- Se descargará siempre la última versión disponible.

#### Más símbolos:

- >**: descargar/actualizar a cualquier versión posterior a la dada
- >=**: descargar/actualizar a cualquier versión igual o posterior a la dada
- <=**: descargar/actualizar a cualquier versión igual o anterior a la dada
- <**: descargar/actualizar a cualquier versión anterior a la dada

Adicionalmente, se pueden crear combinaciones con los criterios anteriores, por ejemplo:

`1.0.0 || >=1.1.0 <1.2.0`

Aquí se usará la versión 1.0.0 (si la encuentra) o alguna a partir de 1.1.0 pero anteriores a 1.2.0.