TEMA 1 INTRODUCCIÓN A NODE.JS

4- El gestor de paquetes npm

Gestor de Paquetes NPM

- Es un gestor de paquetes para JavaScript (Node, jQuery…), como el Play Store de Android o App Store de Apple para app's de móviles.
- Se instala en el momento que instalamos el Node.js.
- Para comprobarlo, igual que hicimos con node. npm –v ó npm –-version.
- Página web: npmjs.com.

Modo de instalación

- LOCAL: Lo normal y más habitual que instalemos los módulos que necesitemos en cada momento y en cada proyecto en particular.
- GLOBAL: Pero también disponemos de la posibilidad de hacerlo global para siempre.

NPM. LOCAL. Package.json

- Crea una carpeta “PruebaNPM” en tu carpeta “Pruebas”
- La configuración básica se almacena en un fichero llamado: “package.json”
- Para crearlo 2 formas:
  - npm init –-yes: Creará el fichero con los valores por defecto
  - npm init: Iniciará el asistente para que rellenemos nosotros los valores.

LOCAL. Añadir módulos I

- Una vez creado el fichero de configuración podemos proceder a añadir módulos (nunca antes).
- npm install --save <nombre_módulo>
- Podemos instalar una versión concreta:
- npm install --save <nombre_módulo>@<versión>

LOCAL. Añadir módulos II

- Tras ejecutar la instalación, se creará automáticamente una subcarpeta "node_modules" y dentro, cada uno de los módulos que vayamos instalando en el proyecto.
- El flag --save sirve para que además de instalarlo modifique automáticamente el fichero de configuración package.json añadiendo dicho elemento.
- También se creará o modificará un fichero llamado "package-lock.json" que es un backup de lo que vamos haciendo para que podamos volver atrás si es necesario.

LOCAL. Utilización

- Una vez instalado el módulo podemos usarlo, como siempre con REQUIRE. Veamos un ejemplo con el módulo LODASH:
- npm install --save lodash
- Y dentro del js:

```
const lodash = require('lodash');
console.log(lodash.difference([1, 2, 3], [1]));
```

--Elimina un elemento de un vector.

EJERCICIO 3

- Crea una carpeta llamada "Tema1_EnlazarLista" en tu espacio de trabajo, en la carpeta "Ejercicios". Dentro, crea un archivo "package.json" utilizando el comando npm init visto antes. Deja los valores por defecto que te plantea el asistente, y pon tu nombre como autor.
- Después, instala el paquete "lodash" como se ha explicado en el ejemplo anterior, y consulta su documentación (http://lodash.com/docs/4.17.10), para hacer un programa en un archivo "index.js" que, dado un vector de nombres de personas, los muestre por pantalla separados por comas. Deberás definir a mano el array de nombres dentro del código. Por ejemplo, para el array ["Alex", "Arturo", “Pablo", "Nacho"], la salida del programa deberá ser:
- Alex,Arturo,Pablo,Nacho
- NOTA: revisa el método join dentro de la documentación de "lodash", puede serte muy útil para este ejercicio.

Gestión de Versiones

- Puede que las versiones de los módulos cambien y afecten a nuestro proyecto. Para limitar esto podemos usar una serie de modificadores de versiones.
- Cambiaremos manualmente el fichero package.json en la sección “dependencies” y ejecturíamos el comando:
  - npm update --save

Modificadores de Versiones

- "lodash": "1.0.0" indicaría que la aplicación sólo es compatible con la versión 1.0.0 de la librería
- "lodash": "1.0.x" indica que nos sirve cualquier versión 1.0
- "lodash": "*" indica que queremos tener siempre la última versión disponible del paquete. Si dejamos una cadena vacía "", se tiene el mismo efecto. No es una opción recomendable en algunos casos, al no poder controlar lo que contiene esa versión.
- "lodash": "> = 1.0.2" indica que nos sirve cualquier versión a partir de la 1.0.2
- "lodash": "< 1.0.9" indica que sólo son compatibles las versiones de la librería hasta la 1.0.9 (sin contar esta última).
- "lodash": "^1.1.2" indica cualquier versión desde la 1.1.2 (inclusive) hasta el siguiente salto mayor de versión (2.0.0, en este caso, sin incluir este último).
- "lodash": "~1.3.0" indica cualquier versión entre la 1.3.0 (inclusive) y la siguiente versión menor (1.4.0, exclusive).

Instalación manual módulos

- También podemos tocar el fichero package.json directamente, y añadirlos manualmente. Añadamos “express”.

```
{
    ...
    "dependencies": {
        "lodash": "^4.17.4",
        "express": "*"
    }
}
```

- Una vez añadidos, tendríamos que ejecutar el comando y añadiría el módulo:
  - npm install

Desinstalar módulos

- Se eliminará del fichero package.json.
- npm uninstall --save <nombre_módulo>

Compartir nuestro proyecto

- Para compartir nuestro proyecto, o para subirlo al profesor cuando tengamos que entregar prácticas.
- No se debe añadir la carpeta “node_modules”, puede ocupar demasiado y se regenera facilmente con el fichero package.json.

GLOBAL

- Sirve para que lo usen todos nuestros proyectos.
- Sintaxis:
- npm install –g <nombre_módulo>
- Estos módulos quedan instalados de forma global, no se pueden incluir con require, para eso deberíamos hacerlo de forma local.

GLOBAL II

- Nodemon sirve para que ante cualquier cambio que hagamos en nuestro código el problema se reinicia solo, ahorrándonos tener que pararlo y reiniciarlo.
- npm install –g nodemon
- Y luego para ejecutar el programa que queramos, en vez de usar node, usaremos:
- nodemon index.js
- Para desinstalar: npm unistall –g <nombre>

EJERCICIO 4

- Instala el módulo "nodemon" a nivel global en tu sistema, y ejecuta con él el ejercicio anterior. Prueba a modificar los nombres del vector y comprueba cómo se actualiza automáticamente la salida por pantalla.