Primeros pasos con Node.js

Antes de comenzar a dar nuestros primeros pasos con Node.js, es necesario tenerlo instalado. Si has descargado la máquina virtual con todo el software instalado, o si has seguido los pasos del documento de instalación, ya lo deberías tener listo para usarse.

1. Comprobando la instalación

Una vez instalado Node.js, podemos comprobar la instalación y la versión instalada desde línea de comandos, con el comando `node`. Para ello, abrimos un terminal en el sistema en que estemos y escribimos este comando:

```
node -v
```

También podemos utilizar `node --version` en su lugar. En ambos casos, nos deberá aparecer la versión que hemos instalado, que en nuestro caso será algo parecido a esto (el número de versión puede variar):

```
v16.14.0
```

> **IMPORTANTE:** debemos asegurarnos de que podemos escribir este comando y obtener la salida esperada antes de continuar, ya que de lo contrario tampoco podremos ejecutar nuestros programas Node.

2. Probando nuestro primer programa

El comando `node` también se emplea para ejecutar desde el terminal un archivo fuente Node.js. Por ejemplo, podemos editar un programa llamado `prueba.js` con este contenido:

```
console.log("Hola mundo");
```

Y después ejecutarlo con este comando (desde la misma carpeta donde tengamos el archivo fuente):

```
node prueba.js
```

La salida en este caso será:

```
Hola mundo
```

3. Integración de Node.js y Visual Studio Code

Visual Studio Code es uno de los muchos editores que podemos emplear para desarrollar aplicaciones Node. Disponéis de una versión ya instalada en la máquina virtual que se os proporciona, y si lo queréis instalar en vuestro sistema, también tenéis los pasos explicados en el documento de instalación.

Este IDE ofrece una integración muy interesante con Node.js, de manera que podemos editar, ejecutar y depurar nuestras aplicaciones desde el propio IDE. Veamos qué pasos seguir para ello.

3.1. Preparar el espacio de trabajo

Cada proyecto Node que hagamos irá contenido en su propia carpeta y, por otra parte, Visual Studio Code y otros editores similares que podamos utilizar (como Atom o Sublime Text) trabajan por carpetas (es decir, les indicamos qué carpeta abrir y nos permiten gestionar todos los archivos de esa carpeta). Por lo tanto, y para centralizar de alguna forma todo el trabajo del curso, lo primero que haremos será crear una carpeta llamada "**ProyectosNode**" en nuestro espacio de trabajo (por ejemplo, en nuestra carpeta personal). Dentro de esta carpeta, podemos crear dos subcarpetas:

- **Pruebas**, donde guardaremos todos los proyectos de prueba y ejemplo que hagamos durante las sesiones.
- **Ejercicios**, donde almacenaremos los ejercicios propuestos de cada sesión para entregar.

La estructura de carpetas quedará entonces como sigue:

- ProyectosNode
  - Pruebas
  - Ejercicios

3.2. Crear y editar un proyecto básico

Dentro de la carpeta *ProyectosNode/Pruebas*, vamos a crear otra subcarpeta llamada "*PruebasSimples*" donde definiremos pequeños archivos para probar algunos conceptos básicos, especialmente en las primeras sesiones.

Una vez creada la carpeta, abrimos Visual Studio Code y vamos al menú *Archivo* > *Abrir carpeta* (o *Archivo* > *Abrir...*, dependiendo de la versión de Visual Studio Code que tengamos). Elegimos la carpeta

"*PruebasSimples*" dentro de *ProyectosNode/Pruebas* y se abrirá en el editor. También podemos abrir la carpeta arrastrándola desde algún explorador de carpetas hasta una instancia abierta de Visual Studio Code.

De momento la carpeta está vacía, pero desde el panel izquierdo podemos crear nuevos archivos y carpetas. Para empezar, vamos a crear un archivo "prueba.js" como el de un ejemplo previo, con el código que se muestra a continuación:

3.3. Ejecutar un archivo Node desde Visual Studio Code

Si quisiéramos ejecutar el programa anterior con lo visto hasta ahora, deberíamos abrir un terminal, navegar hasta la carpeta del proyecto y ejecutar el comando `node prueba.js`, como hicimos en un ejemplo anterior.

Sin embargo, Visual Studio Code cuenta con un terminal incorporado, que podemos activar yendo al menú *Ver* > *Terminal*, o *Ver* > *Terminal integrado*, según la versión instalada. Aparecerá en un panel en la zona inferior. Observemos cómo automáticamente dicho terminal se sitúa en la carpeta de nuestro proyecto actual, por lo que podemos directamente escribir `node prueba.js` en él y se ejecutará el archivo, mostrando el resultado en dicho terminal:

4. Depuración de código

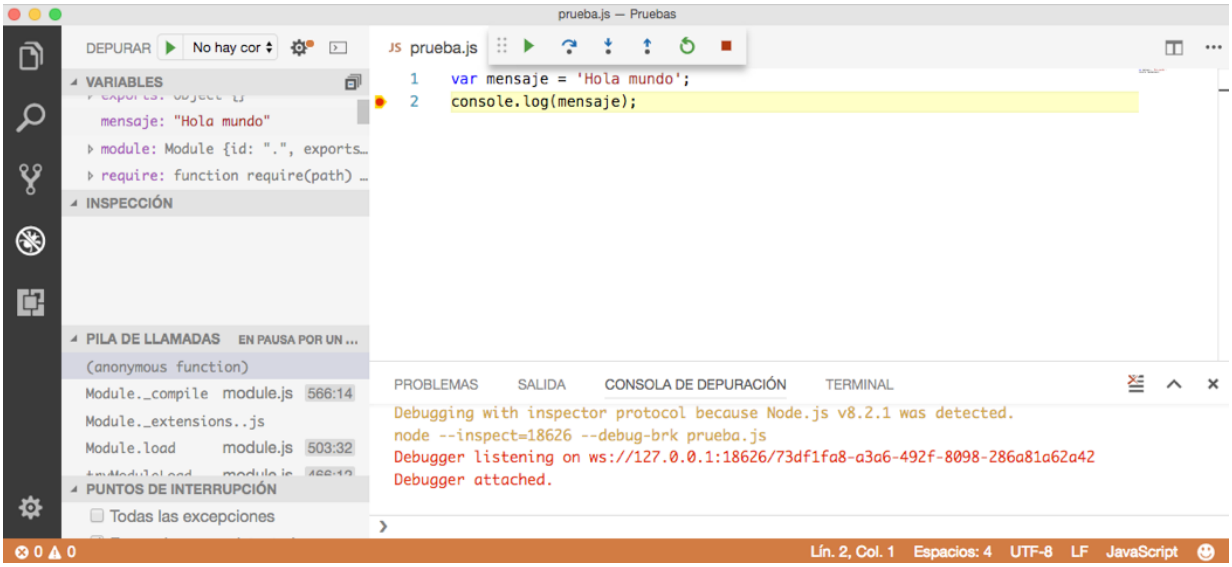
Existen diferentes alternativas para depurar el código de nuestras aplicaciones Node.js. En esta sección veremos dos:

- Utilizando nuestro propio IDE (Visual Studio Code)
- Utilizando Google Chrome

4.1. Depurar desde Visual Studio Code

Visual Studio Code ofrece un depurador para nuestras aplicaciones Node. Para entrar en modo depuración, hacemos clic en el icono de depuración del panel izquierdo (el que tiene forma de *bug* o chinche).

Podemos establecer *breakpoints* en nuestro código haciendo clic en la línea en cuestión, en el margen izquierdo (como en muchos otros editores de código). Después, podemos iniciar la depuración con `F5`, o bien con el menú *Ejecutar* > *Iniciar depuración* o *Run* > *Start Debugging*, o con el icono de la flecha verde de *play* de la barra superior. Al llegar a un breakpoint, podemos analizar en el panel izquierdo los valores de las variables y el estado de la aplicación.



También podemos continuar hasta el siguiente breakpoint (botón de flecha verde), o ejecutar paso a paso con el resto de botones de la barra de depuración. Para finalizar la depuración, hacemos clic en el icono del cuadrado rojo de stop (si no se ha detenido ya la aplicación), y volvemos al modo de edición haciendo clic en el botón de explorador del panel izquierdo.

La "consola de depuración" que aparece en el terminal inferior realmente es un terminal REPL (*Read Eval Print Loop*), lo que significa que desde ella podemos acceder a los elementos de nuestro programa (variables,

objetos, funciones) y obtener su valor o llamarlos. Por ejemplo, en el caso anterior, podríamos teclear "mensaje" en el terminal y ver cuánto vale esa variable:

4.2. Depurar desde Google Chrome

Si tenemos disponible Google Chrome, podemos valernos de las herramientas para desarrolladores que incorpora (*Developer Tools*) para utilizar el depurador. Para ello, desde un terminal ejecutaremos este comando para poner en marcha el archivo que queremos depurar (desde la carpeta donde esté dicho archivo):

```
node --inspect-brk archivo.js
```

Una vez hecho esto, la aplicación queda a la espera de ser depurada. Si abrimos Chrome y accedemos a la URL `chrome://inspect`, podemos acceder al depurador desde el enlace *Open dedicated DevTools for Node*.

Se abrirá una ventana con el depurador. Desde la pestaña *Sources* podemos examinar el código fuente del programa:

Además, podemos establecer *breakpoints* de forma visual haciendo clic izquierdo sobre el número de línea (se marcará en azul, como en la imagen anterior), o anteponiendo la instrucción `debugger;` a la línea donde queramos fijar el *breakpoint*.

```
debugger;
// Línea donde queremos parar
...
```

Después, podemos ejecutar la aplicación de forma continuada o paso a paso con los controles de la parte superior derecha. Finalmente, si abrimos la consola inferior de esta pestaña *Sources*, accedemos a un terminal REPL como el que tiene Visual Studio Code, para examinar el valor de variables, objetos, o llamar a funciones. Notar cómo examinamos la variable `mensaje` en el ejemplo anterior.