# Módulos del núcleo de Node.js

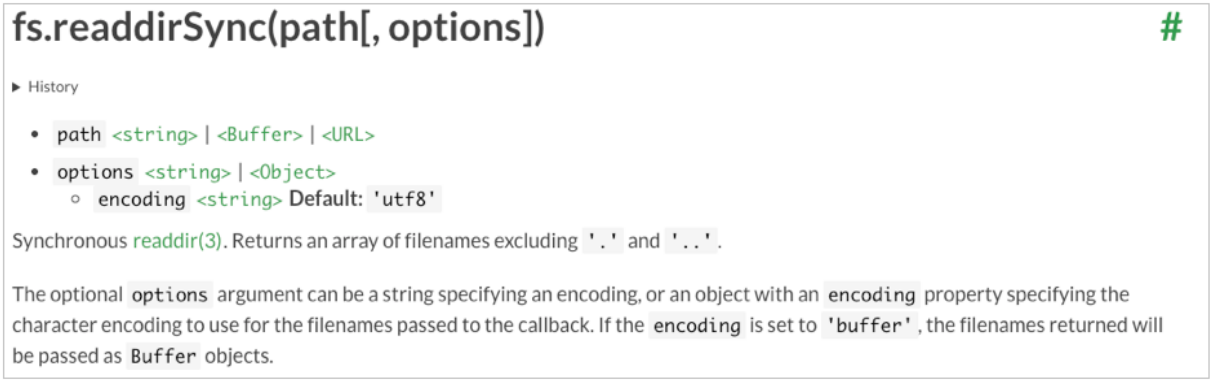



Node.js es un framework muy modularizado, es decir, está subdividido en numerosos módulos, librerías o paquetes (a lo largo de estos apuntes utilizaremos estos tres términos indistintamente para referirnos al mismo concepto). De esta forma, sólo añadimos a nuestros proyectos aquellos módulos que necesitemos.

## 1. Ejemplos de módulos

El propio núcleo de Node.js ya incorpora algunas librerías de uso habitual. Por ejemplo:

- **http** y **https**, para hacer que nuestra aplicación se comporte como un servidor web, o como un servidor web seguro o cifrado, respectivamente.
- **fs** para acceder al sistema de archivos
- **utils**, con algunas funciones de utilidad, tales como formato de cadenas de texto.
- ... etc. Para una lista detallada de módulos, podemos acceder aquí. Es una API de todos los módulos incorporados en el núcleo de Node.js, con documentación sobre todos los métodos disponibles en cada uno. Por ejemplo, aquí podemos ver la documentación sobre el método `readdirSync` del módulo `fs`, que utilizaremos en un ejemplo a continuación:



fs.readdirSync(path[, options]) #

▸ History

- `path` <string> | <Buffer> | <URL>
- `options` <string> | <Object>
  - `encoding` <string> Default: `'utf8'`

Synchronous readdir(3). Returns an array of filenames excluding `'.'` and `'..'`.

The optional `options` argument can be a string specifying an encoding, or an object with an `encoding` property specifying the character encoding to use for the filenames passed to the callback. If the `encoding` is set to `'buffer'`, the filenames returned will be passed as `Buffer` objects.

## 2. Utilizar módulos del núcleo de Node

En primer lugar, vamos a aprender cómo incluir en nuestros proyectos módulos o librerías que ya vienen incorporadas en el núcleo de Node.js, y por tanto, ya tenemos disponibles tras instalarlo.

Para utilizar cualquier módulo (propio de Node o hecho por terceras partes) en una aplicación es necesario incluirlo en nuestro código con la instrucción `require`. Recibe como parámetro el nombre del módulo a

añadir, como una cadena de texto.

## 2.1. Ejemplo sencillo: listar ficheros con *fs*

Por ejemplo, vamos a crear un archivo llamado `listado.js` en nuestro proyecto de "*ProyectosNode/Pruebas/PruebasSimples*". En él vamos a hacer un pequeño programa que utilice el módulo `fs` incorporado en el núcleo de Node para obtener un listado de todos los archivos y subcarpetas de una carpeta determinada. El código de este archivo puede ser más o menos así:

```
const ruta = '/Users/nacho';
const fs = require('fs');
fs.readdirSync(ruta).forEach(fichero => {console.log(fichero);});
```

Si ejecutamos este programa en el terminal (recordemos que podemos usar el terminal integrado de Visual Studio Code), obtendremos el listado de la carpeta indicada. Antes de ejecutarlo, recuerda cambiar el valor de la constante `ruta` en el código por el de una carpeta válida en tu sistema.

Notar que en el código hemos declarado dos constantes ( `const` ), en lugar de variables ( `var` / `let` ), ya que no necesitamos manipular o modificar el código que se almacenará en ellas una vez cargadas (no vamos a cambiar la ruta, ni el contenido del módulo `fs` en nuestro código). En ejemplos que podáis encontrar en Internet es habitual, no obstante, el uso de `var` o `let` en estos casos.

> **NOTA**: Además de la instrucción `require` para incorporar módulos en aplicaciones Node.js, es muy habitual utilizar la sintaxis `import` en aplicaciones JavaScript en general (especialmente cuando se emplean frameworks cliente, como Angular o React). A lo largo de este curso optaremos por la primera opción, pero dejamos aquí indicado cómo se importarían los módulos con `import` (aunque su uso comporta algunas diferencias con respecto a `require` ):

```
import fs from 'fs';
fs.readdirSync(...);
```