

# Introducción a Node (Parte I)

## ¿Qué es Node?

### Competencias

- Reconocer las características básicas de Node identificando sus componentes y alcances en el desarrollo de aplicaciones
- Ejecutar una aplicación de Node básica por la terminal para devolver un mensaje por consola

### Introducción

En este capítulo vamos a conocer qué es Node, sus características principales, además de su aporte al desarrollo web. Node entre tantas cosas, te ofrece una serie de herramientas para los desarrollos más modernos y sofisticados, esta cualidad lo ha hecho muy popular en la comunidad de desarrolladores.

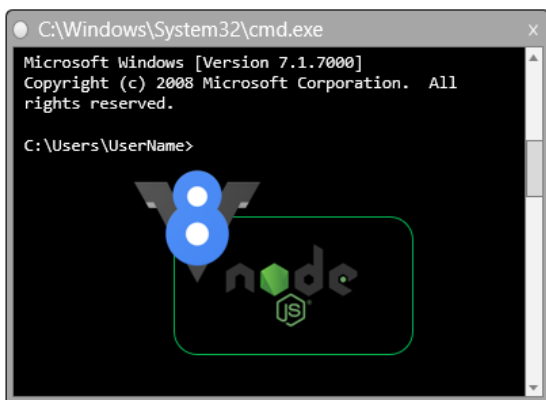
Node funciona bajo el motor V8 de Google, enfocándose en el procesamiento de peticiones asíncronas gestionadas en un único hilo, no obstante, resalta la posibilidad de trabajar con diferentes hilos de procesamiento cuyo funcionamiento representa el ciclo de vida de un proceso.

Aprender a trabajar con Node te ampliará las capacidades como desarrollador y te ayudará a crear proyectos multidisciplinarios, basta con la lógica que has aprendido en el lado del cliente con JavaScript para iniciar a crear aplicaciones en el lado del servidor, en donde aprenderás a interactuar con valores del sistema operativo, incluyendo la gestión de archivos.

## ¿Qué es Node?

Node es una tecnología que nos provee de un entorno para la ejecución de código JavaScript fuera del navegador, quiere decir, que permite ejecutar código JavaScript como si se tratara de un programa instalado en nuestro computador o una aplicación de servidor.

Node js + V8 de Chrome en Servidor



V8 Chrome

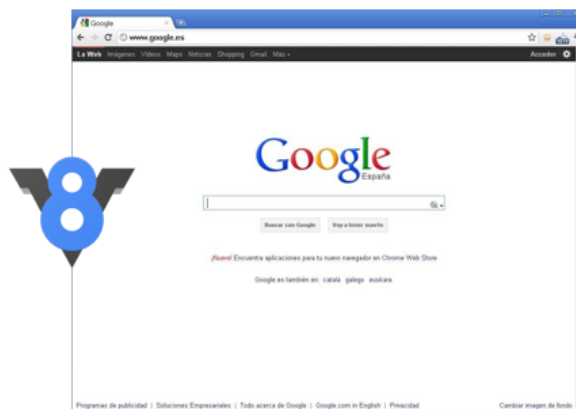


Imagen 1. Procesamiento, máquina o servidor vs Procesamiento en explorador.

Fuente: Desafío Latam

La imagen anterior representa por una lado, Node con el motor V8, el cual se puede ejecutar en una consola de nuestra computadora o de algún servidor, y por otro lado, el motor V8 ejecutándose como es originalmente en el explorador Chrome. ¿Qué es el motor V8 de Google? Revisa el **Material Apoyo Lectura - El motor V8 de Google**, ubicado en "Material Complementario".

El código JavaScript hasta la llegada de Node, solo se podía ejecutar en navegadores que tienen la capacidad de procesarlo y entenderlo, ya que cada navegador posee un motor de interpretación destinado para esta tarea. Node trabaja con el motor de Google Chrome llamado V8, y este es el motor intérprete que se encarga de tomar ese código JavaScript y transformarlo en un lenguaje conocido para la máquina en la cual se requiere ejecutar.

## Características

Node sigue un proceso por cada tarea que procesa, esto se conoce como el "ciclo de vida de un proceso", el cual puedes conocer revisando el **Material Apoyo Lectura - El ciclo de vida de un proceso Node**, ubicado "Material Complementario". Pero para ir un poco más directo y conceptualizar el uso de Node a nivel de desarrollo, a continuación te presento varias ventajas, desventajas y características especiales.

## Ventajas

- **Velocidad y rendimiento:** Gracias a su integración con el motor V8 de Chrome, Node aprovecha todas las características de optimización que V8 posee para el tratamiento veloz y rápido del código Javascript.
- **Basado en el lenguaje JavaScript:** No es un secreto que JavaScript es uno de los lenguajes más populares entre los desarrolladores, es el lenguaje de la web, y esta popularidad en parte se debe al nacimiento de Node. Esto nos da la posibilidad de realizar desarrollo con un mismo lenguaje tanto para el Backend y Frontend.
- **Popularidad:** Como mencionamos, Node y JavaScript gozan de gran popularidad, cada día crece más el interés y la cantidad de desarrolladores que lo utilizan. Esto permitirá que Node crezca y perdure en el tiempo, posee una demanda de una comunidad creciente y que cada día aporta con más contenido que permite la masificación.
- **Multiplataforma:** Puede ejecutarse en máquinas Windows, Mac OS X y Unix.
- **Manejo de alto tráfico:** Es ideal para aplicaciones de alto tráfico como Twitter, WhatsApp, etc.

## Desventajas

Aunque no son críticas, como cualquier tecnología en el desarrollo, Node tiene sus pros y sus contras, ahora te presento varias de las desventajas que podrías toparte usando Node en tus proyectos.

- **Inestabilidad de la API:** La API cambia frecuentemente y no permanece estable. Una API que cambia frecuentemente, trae cambios incompatibles para versiones anteriores por lo que te obliga a mantenerte atento a estos cambios para hacer las actualizaciones correspondientes.
- **Problemas heredados:** Node hereda los problemas de JavaScript, como por ejemplo la representación del float.
- **Potencia de Cálculo:** No ofrece mayor potencialidad para cálculos que las ofrecidas por otros lenguajes de servidor como C# y Java, aunque esta desventaja sólo aplica a proyectos de procesamiento matemático altamente complejos.

## Características especiales

- **Gestor de Paquetes (NPM):** Posee un gestor de paquetes que nos permite instalar nuevas funcionalidades o módulos, y un manejo de dependencias por separado sin necesidad de una herramienta externa.
- **Cambio de mentalidad:** Requiere un cambio de mentalidad en relación a los lenguajes tradicionales de servidor, debido a la utilización de la asincronía en su programación, lo que conlleva a que la curva de aprendizaje no sea tan rápida en comparación con otros entornos. Por supuesto, esta característica aplica solo para personas que estén aprendiendo Node luego de haber trabajado con otro entorno u otras tecnologías.

En conclusión con Node podemos crear aplicaciones multiplataforma de alto rendimiento capaz de soportar un tráfico elevado de datos, a continuación te muestro una lista de algunas de los problemas o necesidades que podrías resolver desarrollando bajo este entorno:

- **Crear servidores:** De forma nativa con el módulo "http" o con frameworks como "Express.js".
- **Crear y consumir APIs REST:** Para consumirlas podríamos usar "AXIOS" al igual que en el lado del cliente y para crearlas de forma nativa o con Express.js.
- **Conexiones con bases de datos:** Con módulos como "mongoose" o "pg" podremos conectarnos con motores como MongoDB o PostgreSQL, y en sí con diferentes bases de datos tanto relacionales como no relacionales.
- **Aplicaciones multiplataformas:** Con integraciones de diferentes SDK's, librerías y frameworks podremos exportar aplicaciones híbridas de teléfonos y de escritorio creadas con tecnologías web.
- **Chats:** Generando persistencia de datos para los mensajes, podríamos crear un chat en tiempo real con el uso de sockets entre diferentes usuarios con la tecnología "socket.io".
- **Conexiones con electrónica:** Con módulos como "serialport" se pueden hacer conexiones con sistemas arduinos.
- **Gestión de archivos:** Podemos hacer un CRUD con archivos del sistema operativo con el módulo "File System".

- **Testing de aplicaciones:** Al igual que en el lado del frontend, podríamos hacer testing en el lado del backend con las conocidas librerías “mocha” y “jest”.
- **Ejecución de tareas simultáneas:** De forma natural Node permite trabajar con tareas de forma simultánea permitiendo el procesamiento de promesas, callbacks y todo lo relacionado a consultas asíncronas.
- **Algoritmos de inteligencia artificial:** Node y el mundo de JavaScript como tal cuentan con la librería “Tensorflow.js”, y en los últimos años se ha tenido un importante apoyo por parte de 3 de las empresas más grandes del mundo tecnológico: Amazon, Google y Microsoft. Las cuales, facilitan por medio de diferentes APIs en sus servicios un alto computo de datos enfocados en la aplicación de inteligencia artificial.

Cabe destacar que entendiendo que Node es un entorno en sí, la aplicación de estas soluciones se realizan gracias a la instalación, configuración y uso de paquetes o módulos diseñados y creados para estos fines, para eso contamos con una increíble variedad de librerías, frameworks y herramientas disponibles de forma gratuita en una biblioteca internacional de código abierto llamada NPM, la cual conocerás más adelante.

## Ejercicio propuesto (1)

Debate con tus compañeros las siguientes preguntas:

- ¿Node debería ser una alternativa a considerar si necesitara crear una aplicación que se ejecute en diferentes plataformas?
- ¿Node debería ser una alternativa a considerar si necesitara disponibilizar los datos de una base de datos como una API REST?
- ¿Ocuparía Node para hacer cambios estéticos en mis sitios web?

Node es un complemento de diferentes tecnologías que juntas pueden crear grandes cosas, pero ¿Cómo usamos Node? Hagamos nuestro primer “Hola Mundo!”, si no tienes aún instalado Node en tu computadora, revisa el **Material Apoyo Lectura - Instalación y Configuración de Node**, ubicado “Material Complementario”, allí verás el paso a paso en los diferentes sistemas operativos para que puedas instalarlo antes del siguiente ejercicio.

Ahora ejecutaremos una aplicación de Node básica, para esto, con Visual Studio Code abre una carpeta nueva y crea un archivo “index.js” con el siguiente código:

```
console.log('Hola Mundo!')
```

Para ejecutar este documento bastará con abrir una terminal (De preferencia la de VSC) y correr el siguiente comando:

```
node index.js
```

Así como se muestra en la siguiente imagen:

```
→ node git:(master) x node index.js  
Hola Mundo!
```

Imagen 2.Respuesta por terminal del primer Hola Mundo!.

Fuente: Desafío Latam

## Child process

### Competencias

- Reconocer el uso del módulo child process para la ejecución de procesos externos
- Construir una aplicación básica con Node para devolver una operación aritmética por la consola

### Introducción

Cuando desarrollamos aplicaciones bajo el entorno de Node, disponemos de varias herramientas poderosas que nos pueden ser de mucha utilidad en diferentes situaciones, estas herramientas son representadas también como módulos nativos que utilizan componentes del sistema operativo para ejecutar una tarea.

En este capítulo tendrás tu primer encuentro con un módulo de Node llamado "child\_process", el cual sirve para emular una terminal usando código Javascript, y ocuparemos esta herramienta para ejecutar una aplicación externa, capturando el mensaje que emita por consola y usandolo en nuestra aplicación principal.

## Tareas delegadas a procesos externos

Node nos provee de `child_process`, un potente módulo que nos permite ejecutar procesos como si se tratara de una terminal o consola de un sistema operativo. Debes entender que esto conlleva una gran responsabilidad porque las terminales tienen los caminos directos al corazón del software.

Ahora podrías preguntarte ¿Y estos famosos módulos cómo se usan? Pues es sencillo, solo hay que importarlos y guardarlos en variables, y veremos cómo hacerlo en breve.

Para el primer encuentro con el módulo `child_process` usaremos el método “`exec`” para ejecutar una línea de comando indicando la siguiente sintaxis “`node <nombre_del_archivo>`” anteriormente expuesta con el “Hola Mundo!”.

## Ejercicio guiado: Aplicando `child_process`

Crear una aplicación que ejecute 2 archivos y manipule su respuesta, en este caso numéricas, para devolver la suma de ambas. Para esto sigue los siguientes pasos:

- **Paso 1:** Crear un archivo con nombre “`num1.js`” y usa el siguiente código para mandar por consola el número 1.

```
console.log(1)
```

- **Paso 2:** Crear un archivo con nombre “`num2.js`” y usa el siguiente código para mandar por consola el número 2.

```
console.log(2)
```

Ahora solo falta crear un tercer archivo cuyo objetivo será sumar los números devueltos por la ejecución de los otros archivos.

- **Paso 3:** Crear un archivo con nombre “`suma.js`”, el cual será nuestra aplicación principal. Escribe el código realizando las siguientes instrucciones:
  1. Importar el módulo “`child_process`” en una constante con el mismo nombre.
  2. Crear las variables `num1` y `num2` de forma global con valores `null`.
  3. Crear una función “`ejecutar`” que reciba como parámetro el nombre de un archivo a ejecutar.
  4. La función creada retornará una promesa.



5. Usar el método "exec" pasando como primer parámetro una concatenación con el parámetro recibido en la función, y como segundo parámetro el callback "resolve" con el resultado de la ejecución en formato Number. Harás esto para poder sumar aritméticamente ambos resultados.
6. Llamar a la función ejecutar pasando como argumento el nombre del primer archivo y en el callback de su método "then", reasignamos el valor de la variable "num1" con el parámetro "numero1", recibido como parámetro por el callback "resolve" de la promesa.
7. Dentro del método "then" de la primera llamada a la función, vuelve a llamar a la función ejecutar pero esta vez especificando el nombre del segundo archivo reasignando en el callback el valor de "num2" con el parámetro "numero2" e imprimiendo por consola la suma de ambas variables.

```
// 1
const child_process = require('child_process')

// 2
let num1 = null
let num2 = null

// 3
function ejecutar(archivo) {

  // 4
  return new Promise((resolve) => {

    // 5
    child_process.exec(`node ${archivo}`, function (err, result) {
      resolve(Number(result))
    })
  })
}

// 6
ejecutar('num1.js').then((numero1) => {
  num1 = numero1
  // 7
  ejecutar('num2.js').then((numero2) => {
    num2 = numero2
    console.log(num1 + num2)
  })
})
```

- **Paso 4:** En la terminal corre el comando "**node suma.js**" y mira lo que sucederá en la siguiente imagen.

```
→ ejercicio lectura git:(master) x node num1.js  
1  
→ ejercicio lectura git:(master) x node num2.js  
2  
→ ejercicio lectura git:(master) x node suma.js  
3
```

Imagen 3. Respuesta por terminal de la aplicación suma.js.

Fuente: Desafío Latam

Si quieres saber más del módulo `child_process`, su API completa la podrás conseguir de forma oficial en el siguiente [link](#).

## Ejercicio propuesto (2)

Basado en el ejercicio de la suma de dos números, crea 3 archivos: `nombre.js`, `apellido.js` y `fullname.js` para imprimir tu nombre completo.

## Argumentos por la línea de comandos

### Competencia

- Hacer una aplicación con Node que reciba argumentos por la línea de comandos.

### Introducción

Se creería que las aplicaciones son lugares cerrados donde todo sucede dentro del código, sin embargo, gracias a Node podemos enviar datos a una aplicación justo en el momento que la levantamos por medio de la línea de comando.

En este capítulo, aprenderás a enviar datos en formato de texto a tus aplicaciones con Node, para ser almacenados en variables y utilizados dentro de las funciones. Aprendiendo a hacer esto, próximamente podrás preparar aplicaciones cuyas entradas de datos sean escritas por la terminal en la que es levantada.

## Capturando argumentos por la línea de comandos

Cada vez que ejecutamos un archivo por la terminal escribimos una sintaxis basada en comandos, direcciones y flags(banderas), en sí todos estos son argumentos, al correr un documento JavaScript con Node no es la excepción.

Realicemos un ejercicio con otra operación aritmética parecida al ejercicio del módulo "child\_process". Es importante que sepas que para esto, no será necesario usar ningún módulo, porque los argumentos de cada proceso ejecutado se almacenan como un arreglo en la propiedad "argv", dentro del objeto "process", el cual parecido a una variable global es posible acceder a él desde cualquier entorno de desarrollo en nuestros diferentes sistemas operativos.

Crea un archivo llamado index.js y escribe el siguiente código.

```
console.log(process.argv)
```

Ahora ejecutando este archivo por la terminal obtendrás la siguiente imagen, cabe recordar que la ruta de directorios puede variar para cada usuario:

```
→ argv git:(master) ✗ node index.js
[
  '/home/brian/.nvm/versions/node/v12.18.3/bin/node',
  '/home/brian/Escritorio/node/Unidad 1/desafio1/ejercicio lectura/argv/index.js'
]
```

Imagen 4. Mensaje por consola del process.argv

Fuente: Desafío Latam

Como puedes apreciar en la consola hemos recibido un arreglo con 2 elementos tipo String que reflejan la dirección del recurso que se está ocupando en el argumento, en caso de no tener un soporte en ninguna localidad la propiedad "argv" captura el puro valor escrito.

Vuelve a correr el documento pero agregale un "Desafío Latam" luego de la mención del archivo y obtendrás lo que verás en la siguiente imagen.

```
→ argv git:(master) ✗ node index.js Desafío Latam
[
  '/home/brian/.nvm/versions/node/v12.18.3/bin/node',
  '/home/brian/Escritorio/node/Unidad 1/desafio1/ejercicio lectura/argv/index.js',
  'Desafío',
  'Latam'
]
```

Imagen 5. Mensaje por consola de todos los argumentos recibidos por línea de comando.

Fuente: Desafío Latam

Como puedes ver ahora obtenemos un arreglo de 4 elementos, siendo los últimos 2 el argumento textualmente escrito como texto.

Ahora si, tenemos todo listo para poner a prueba esta herramienta y así capturar los argumentos en la línea de comandos.

## Ejercicio guiado: Operación aritmética

Crear una mini aplicación que devuelva la multiplicación de 2 números, los cuales serán tomados al momento de correr el documento. En el siguiente código te muestro cómo hacerlo.

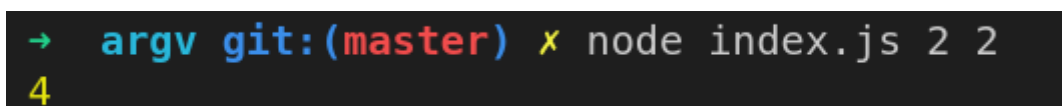
```
const argumentos = process.argv.slice(2)

let num1 = Number(argumentos[0])
let num2 = Number(argumentos[1])

console.log(num1 * num2)
```

Como verás estoy usando el método slice para recortar el arreglo, omitiendo los primeros 2 elementos porque no serán usados en este ejercicio, además de convertir los datos a tipo Number para poder multiplicarlos matemáticamente.

Al ejecutar este archivo obtendremos lo que te muestro en la siguiente imagen.



```
→ argv git:(master) x node index.js 2 2
4
```

Imagen 6. Respuesta por la terminal usando argumento de la línea de comandos.

Fuente: Desafío Latam

## Ejercicio propuesto (3)

Basado en el ejercicio de la multiplicación de dos números por línea de comando, crea una mini aplicación que devuelva por consola la división del primero con respecto al segundo.

## Consulta a una API desde Node

### Competencia

- Codificar un programa para la consulta y el procesamiento de la data en un servicio REST utilizando el entorno Node

### Introducción

Así como en el desarrollo frontend en donde las consultas a APIs son muy comunes y claves para trabajar en sistemas basados en microservicios, a nivel del backend podemos hacerlo con las mismas dependencias o paquetes, incluso con herramientas propias y nativas.

Node tiene integrado varios módulos que permiten hacer diferentes cosas, y uno de los más conocidos es el módulo `https`, el cual cuenta con los métodos representativos de los verbos de consultas HTTP, para consultar recursos externos basados en la arquitectura REST.

En este capítulo, aprenderás a consultar APIs desde una aplicación con Node y usar esa información dentro de tus funciones, para construir sistemas que se comuniquen con datos alojados en un servidor externo.

## Consultando una API con Node

Bien, hasta ahora has podido probar la interacción entre la terminal y Node, ahora empecemos a sacarle provecho a nuestro lenguaje y usemos datos externos como bien has aprendido a hacerlo en desarrollo frontend, pero ¿Es posible hacer consultas a APIs desde el backend? ¡Claro que sí! Desde Node podemos hacer consultas a recursos externos gestionados por APIs bajo el protocolo HTTP. A nivel de frontend esto es muy sencillo con el método “fetch”, con la librería “AXIOS” o usando la tecnología AJAX con jQuery, en el lado backend no es muy diferente porque también se pueden ocupar estas librerías, no obstante, con Node no contamos con el método fetch de forma nativa, sin embargo, disponemos de módulos para realizar la misma tarea, entre ellos tal vez el más conocido para consultas seguras es el módulo “https”, cuya sintaxis te la muestro en el siguiente código y verás el parecido con los métodos que ya conoces.

```
// https.get(url[, options][, callback])

const https = require('https')

https
  .get(<url>, (resp) => {
    resp.on('data', (respuesta) => {
      console.log(JSON.parse(respuesta)) // Impresión por consola de la
data
    })
  })

  .on('error', (err) => {
    console.log('Error: ' + err.message) // Impresión por consola del
error
  })
```

Como puedes notar, el primer parámetro es la url que deseamos consultar y como segundo parámetro tenemos una función callback con el parámetro “resp”, que contiene un método “on” con otros dos parámetros: “data”, que refiere a cuando la data haya llegado con éxito al documento, y una función callback que recibe como parámetro la “respuesta” de la url consultada.

Para la captura del error en caso de fracaso, la primera función callback ofrece también un método “on” pero en este caso con el evento “error” y otro callback que recibe este error ocasionado en la consulta.

## Ejercicio guiado: Consultando una API

Utilizar la siguiente [API](#) para consultar el clima de las ciudades más representativas de Chile, para eso debemos usar el siguiente endpoint: <https://api.gael.cloud/general/public/clima>

Aplicando el módulo https sigue los siguientes pasos:

- **Paso 1:** Importar el módulo https.
- **Paso 2:** Usar el método get del módulo https para consultar la API.
- **Paso 3:** Usar el método "on" del parámetro recibido en el callback del método get especificando el evento "data" e imprimiendo la data obtenida de la API formateada a JSON.
- **Paso 4:** Usar el método "on" del callback del método get especificando el evento "error" para imprimir por consola en su callback el posible error generado por la consulta.

```
// Paso 1
const https = require('https')
// Paso 2
https
  .get('https://api.gael.cloud/general/public/clima', (resp) => {
    // Paso 3
    resp.on('data', (data) => {
      console.log(JSON.parse(data))
    })
  })
// Paso 4
  .on('error', (err) => {
    console.log('Error: ' + err.message)
  })
```

- **Paso 5:** Si ejecutas este código desde la terminal obtendrás una respuesta como la siguiente imagen.



```
https git:(master) x node index.js
{
  Codigo: 'SCFA',
  Estacion: 'Antofagasta',
  HoraUpdate: '10:00',
  Temp: '14',
  Humedad: '63',
  Estado: 'Nublado',
  Icono: 'parcial.png'
},
{
  Codigo: 'SCIR',
  Estacion: 'Arch. Juan Fernández',
  HoraUpdate: '10:00',
  Temp: '12',
  Humedad: '94',
  Estado: 'Cubierto y niebla',
  Icono: 'nieblacubierto.png'
},
{
  Codigo: 'SCAR',
  Estacion: 'Arica',
  HoraUpdate: '10:00',
  Temp: '16',
  Humedad: '68',
  Estado: 'Cubierto'
}
```

Imagen 7. Mensaje por consola con el consumo de la API de climas.

Fuente: Desafío Latam

Como ves, estamos imprimiendo por consola la data de la API ya formateada a JSON.

### Ejercicio guiado: ¿Cuál es el clima de mi ciudad?

Ahora apliquemos lo que hemos aprendido, prosigue con los siguientes pasos para hacer un ejercicio que llamaremos **¿Cuál es el clima de mi ciudad?**, donde se solicita desarrollar una aplicación en Node que consulte el clima de Concepción.

- **Paso 1:** Almacenar en una constante “ciudad” el string “Concepción”.
- **Paso 2:** Almacenar en una variable la data obtenida de la consulta en formato JSON.
- **Paso 3:** Usar un simple método “find” encuentra el objeto que contiene en su propiedad “Estacion” el valor equivalente a la constante ciudad.

```
// Paso 1
const ciudad = 'Concepción'

const https = require('https')

https
  .get('https://api.gael.cloud/general/public/clima', (resp) => {
    resp.on('data', (data) => {
      // Paso 2
      let climas = JSON.parse(data)
      // Paso 3
      let miClima = climas.find((c) => c.Estacion == ciudad)
      console.log(miClima)
    })
  })

  .on('error', (err) => {
    console.log('Error: ' + err.message)
  })
```

- **Paso 4:** Ahora abre la terminal y corre el “index.js” para obtener el resultado que te muestro en la siguiente imagen.

```
→ clima git:(master) x node index.js
{
  Codigo: 'SCIE',
  Estacion: 'Concepción',
  HoraUpdate: '11:00',
  Temp: '11',
  Humedad: '94',
  Estado: 'Nublado y chubascos',
  Icono: 'chubascos.png'
}
```

Imagen 8. Mensaje por consola con la data de los climas filtrada.  
Fuente: Desafío Latam

## Ejercicio propuesto (4)

Basado en el ejercicio de climas, consulta e imprime por consola todos los climas cuyo “Estado” incluye la palabra “Despejado”.

## CRUD de archivos con File System

### Competencias

- Identificar los métodos básicos del módulo File System para la gestión de archivos en el servidor
- Construir una aplicación con Node que pueda crear, leer, renombrar y eliminar con el módulo File System

### Introducción

Node por ser ejecutado como entorno de desarrollo en nuestros sistemas operativos, nos provee de varias herramientas o módulos potentes, capaces de gestionar los archivos y servicios. En este capítulo aprenderás a crear un CRUD (Crear, Leer, Actualizar, Eliminar) de archivos locales desde una aplicación hecha con Node, pero también verás cómo podemos hacer una consulta a una API de forma nativa y cómo desglosar los comandos descritos en la terminal como argumentos, manipularlos a nivel del código para mezclarlo con nuestras funciones, operaciones y condicionales.

Aprender a crear servidores cuyas funcionalidades estén a disposición de usuarios es indispensable para un desarrollador backend y parte de las habilidades del perfil de un full stack developer.

## Gestión de archivos con File System

¿Qué dirías si te dijera que con Node pudieses eliminar todos los archivos de tu sistema operativo? Sería interesante saber cómo hacerlo ¿Cierto?. Pues esto es posible gracias a File System, el cual se describe como un módulo de Node que permite manipular archivos, es decir, crearlos, editarlos, moverlos, leerlos y eliminarlos.

A continuación te presento una tabla con los métodos más comunes de este módulo:

Método	Descripción
writeFile	Crea un archivo
readFile	Devuelve el contenido de un archivo
rename	Renombra/Mueve un archivo.
unlink	Elimina un archivo

Tabla 1. Métodos comunes del módulo fs  
Fuente: Desafío Latam

La sintaxis de estos métodos son bastante parecidos, es importante que sepas que todos estos métodos son asíncronos, por lo que su resultado es obtenido por medio de una función callback, no obstante, incluyen una versión síncrona con la palabra “Sync” luego de la mención del método. Este último por ser síncrono no necesitará tomar el resultado en un callback sino que puedes almacenar directamente su resultado en una variable.

A continuación, te muestro las sintaxis y un ejemplo de los 4 métodos de la tabla 1.

writeFile

```
// fs.writeFile(file, data[, options], callback)
fs.writeFile('message.txt', 'Hola hola Node.js', 'utf8', callback);
```

- **Primer parámetro:** Nombre del archivo.
- **Segundo parámetro:** Contenido del archivo.
- **Tercer parámetro:** Codificación del contenido del archivo, de preferencia será “utf8”.
- **Cuarto parámetro:** Función callback.

## readFile

```
// fs.readFile(file, data[, options], callback)
fs.readFile('message.txt', 'utf8', callback);
```

- **Primer parámetro:** Nombre del archivo.
- **Segundo parámetro:** Codificación del contenido del archivo, de preferencia será "utf8".
- **Tercer parámetro:** Función callback que incluye el error y la data de la consulta.

## rename

```
//fs.rename(oldPath, newPath, callback)
fs.rename('mensaje.txt', 'message.txt', callback)
```

- **Primer parámetro:** Ubicación del archivo original incluyendo el nombre del mismo. Si te encuentras en la misma carpeta puedes colocar directamente el nombre como ves en el ejemplo.
- **Segundo parámetro:** Ubicación del nuevo destino del archivo con el nombre nuevo. Si quieres hacerlo en la misma carpeta puedes colocar directamente el nuevo nombre como ves en el ejemplo.
- **Tercer parámetro:** Función callback.

## unlink

```
//fs.unlink(path, callback)
fs.unlink('message.txt', callback)
```

- **Primer parámetro:** Ubicación del archivo a eliminar incluyendo el nombre del mismo. Si quieres hacerlo en la misma carpeta puedes colocar directamente el nombre.
- **Segundo parámetro:** Función callback.

Escribir los callbacks no son opcionales, sin embargo puedes crear una función anónima o de flecha vacía, es decir, no estás obligado a generar un código de respuesta a la acción que quieras realizar, no obstante sería una buena práctica para tener un feedback del proceso realizado.

## Ejercicio guiado: ¿Qué tareas tengo pendientes para hoy?

Realizar un ejercicio de estos métodos, el cual llamaremos **¿Qué tareas tengo pendientes para hoy?**, donde se solicita desarrollar una aplicación en Node que cree, devuelva el contenido, renombre y elimine un archivo con las tareas pendientes descritas por el usuario.

Crea un archivo "index.js" y sigue los siguiente pasos:

- **Paso 1:** Importar el módulo "fs" en una constante.
- **Paso 2:** Usar el método writeFile del módulo "fs" para crear un archivo de nombre "tareas.txt" con una tarea especificando que la codificación será en código utf8 e imprimiendo en pantalla un mensaje de éxito

```
// Paso 1
const fs = require('fs')
// Paso 2
fs.writeFile('tareas.txt', '1) Hacer aseo en la casa', 'utf8', () => {
  console.log('Archivo creado con éxito')
})
```

Esto creará entonces un archivo llamado tareas.txt en la carpeta en la que te encuentres, con el contenido especificado en el segundo parámetro del método writeFile. Tal y como te muestro en la siguiente imagen.

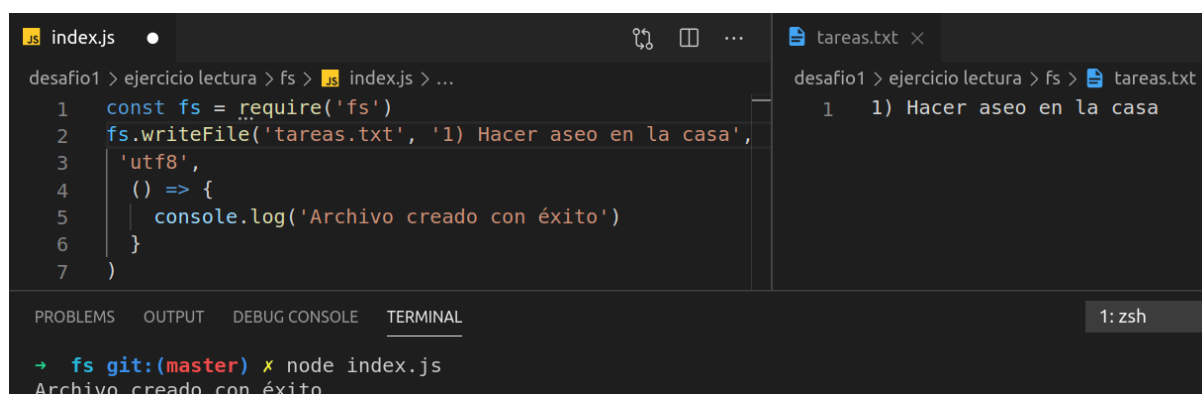


Imagen 9. Mensaje por consola indicando que fue creado con éxito el archivo tareas.txt.

Fuente: Desafío Latam

Ahora intentemos agregar otra tarea al archivo, para esto debemos obtener su contenido y simplemente concatenarle otra tarea como un String, básicamente estaríamos sobrescribiendo su contenido. A continuación te muestro los pasos necesarios para elaborarlo:

- **Paso 1:** Ejecutar el método `readFile` declarando el nombre del archivo `tareas.txt` y la codificación `utf8`.
- **Paso 2:** Dentro del callback del `readFile` ejecuta el método `writeFile` declarando de nuevo el nombre del archivo `tareas.txt`, pero en este método concatena el parámetro "data" con la nueva tarea.
- **Paso 3:** Dentro del callback del `writeFile` imprime en consola un mensaje de éxito.

```
const fs = require('fs')
//Paso 1
fs.readFile('tareas.txt', 'utf8',
  (err, data) => {
    //Paso 2
    fs.writeFile('tareas.txt', data +
      ' 2) Cocinar para mañana', 'utf8',
    () => {
      //Paso 3
      console.log('Archivo sobrescrito con éxito')
    }
  }
)
```

Si ejecutas ahora el archivo `index.js` obtendrás lo que te muestro en la siguiente imagen.

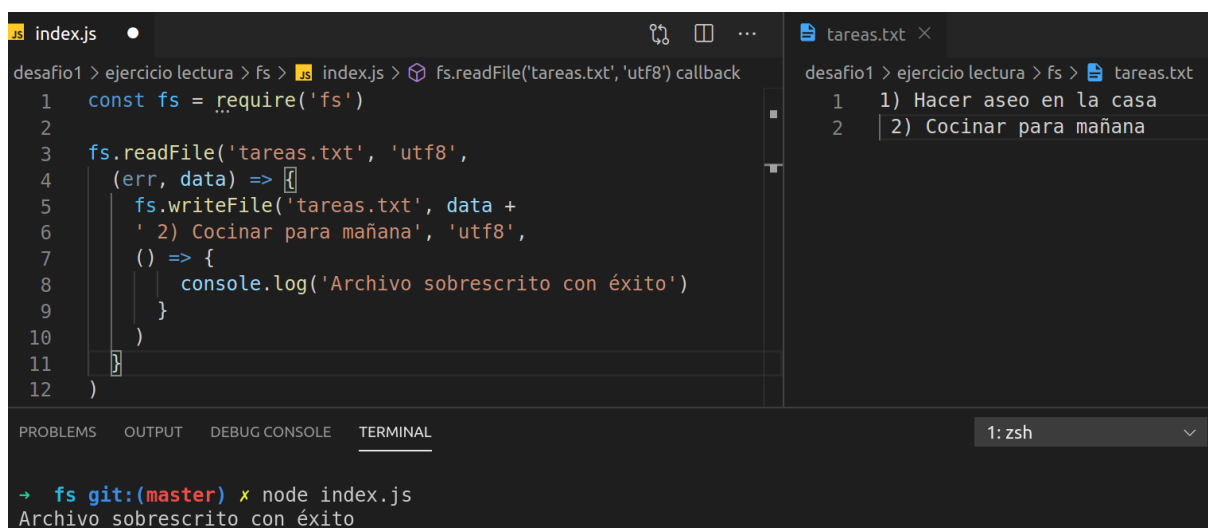


Imagen 10. Mensaje por consola indicando que fue sobrescrito con éxito el archivo.

Fuente: Desafío Latam

## Ejercicio propuesto (5)

Desarrollar una aplicación en Node que cree un archivo llamado "Saludo.txt" y tenga como contenido el mensaje "Hola Mundo!". Justo luego de ser creado este archivo con el método `writeFile`, imprime el contenido de este mismo dentro de su callback con el método `readFile`.

Ahora si quisiéramos renombrar o eliminar el archivo `tareas.txt` debemos usar los métodos `rename` y `unlink`, no obstante te recomiendo comentar o eliminar el código escrito hasta ahora, exceptuando la importación del módulo "fs", porque de ser ejecutado de nuevo sobrescribirá el contenido con una repetición de la segunda tarea, no obstante no es de suma importancia pues al final será eliminado.

Para realizar el renombrado y eliminación realiza los siguientes pasos:

- **Paso 1:** Renombrar el archivo `tareas.txt` por `tasks.txt` e imprime por consola un mensaje de éxito.
- **Paso 2:** Dentro del callback del método `rename`, elimina el archivo recién renombrado como `tasks.txt` e imprime por consola un mensaje de éxito.

```
//Paso 1
fs.rename('tareas.txt', 'tasks.txt', () => {
  console.log('Archivo renombrado')
//Paso 2
  fs.unlink('tasks.txt', () => {
    console.log('Archivo eliminado')
  })
})
```

Ahora si ejecutas nuevamente el archivo `index.js` obtendrás lo que te muestro en la siguiente imagen.





```
index.js
desafio1 > ejercicio lectura > fs > index.js > ...
1  const fs = require('fs')
2
3  fs.rename('tareas.txt', 'tasks.txt', () => {
4    console.log('Archivo renombrado')
5    fs.unlink('tasks.txt', () => {
6      console.log('Archivo eliminado')
7    })
8  })

tareas.txt (deleted)
desafio1 > ejercicio lectura > fs > tareas.txt
1  1) Hacer aseo en la casa
2  2) Cocinar para mañana

TERMINAL
1: zsh
→ fs git:(master) x node index.js
Archivo renombrado
Archivo eliminado
```

Imagen 11. Mensaje por consola indicando que fue renombrado y eliminado el archivo.  
Fuente: Desafío Latam

## Ejercicio propuesto (6)

Basado en el ejercicio de tareas, crea una aplicación en Node que cree, lea, renombre y elimine un archivo llamado “shopping.js”, cuyo contenido debe ser una lista de productos a comprar en un minimarket, siéntete libre de poner los valores que quieras para este ejercicio propuesto.

Luego del método writeFile, cada método debe estar dentro de la función callback del anterior y debe ser ejecutado dentro de un setTimeout con un tiempo de 2 segundos.

## Resumen

A lo largo de esta lectura cubrimos:

- Las características de Node.
- Ejecución de otra aplicación desde un archivo JavaScript obteniendo su impresión por consola con el módulo child\_process.
- Captura de los argumentos escritos en la línea de comandos al levantar una aplicación.
- Consultar APIs con el módulo nativo https.
- Gestión de archivos con el módulo File System.

## Solución de los ejercicios propuestos

1. Debate con tus compañeros las siguientes preguntas:

- ¿Node debería ser una alternativa a considerar si necesitara crear una aplicación que se ejecute en diferentes plataformas?  
**Claro que sí, como aprendistes Node es multiplataforma.**
- ¿Node debería ser una alternativa a considerar si necesitara disponibilizar los datos de una base de datos como una API REST?  
**Si, ya que sirve para crear y consumir APIs REST.**
- ¿Ocuparía Node para hacer cambios estéticos en mis sitios web?  
**No, Node no es una alternativa para realizar cambios estéticos en sitios web.**

2. Basado en el ejercicio de la suma de dos números, crea 3 archivos: nombre.js, apellido.js y fullname.js para imprimir tu nombre completo.

**nombre.js:**

```
console.log('Brian')
```

**apellido.js:**

```
console.log('Habib')
```

**fullname.js:**

```
const child_process = require('child_process')

let nombre = null
let apellido = null

function ejecutar(archivo) {
  return new Promise((resolve) => {
    child_process.exec(`node ${archivo}`, function (err, result) {
      resolve(result)
    })
  })
}

ejecutar('nombre.js').then((name) => {
  nombre = name
  ejecutar('apellido.js').then((lastname) => {
    apellido = lastname
    console.log(nombre + lastname)
  })
})
```

3. Basado en el ejercicio de la multiplicación de dos números por línea de comando, crea una mini aplicación que devuelva por consola la división del primero con respecto al segundo.

```
const argumentos = process.argv.slice(2)

let num1 = Number(argumentos[0])
let num2 = Number(argumentos[1])

console.log(num1 / num2)
```

4. Basado en el ejercicio de climas, consulta e imprime por consola todos los climas cuyo "Estado" incluye la palabra "Despejado".

```
const estado = 'Despejado'
const https = require('https')

https
  .get('https://api.gael.cl/general/public/clima', (resp) => {
    resp.on('data', (data) => {
      let climas = JSON.parse(data)
      let miClima = climas.filter((c) => c.Estado.includes(estado))
      console.log(miClima)
    })
  })

  .on('error', (err) => {
    console.log('Error: ' + err.message)
  })
```

5. Desarrollar una aplicación en Node que cree un archivo llamado "Saludo.txt" y tenga como contenido el mensaje "Hola Mundo!". Justo luego de ser creado este archivo con el método WriteFile, imprime el contenido de este mismo dentro de su callback con el método readFile.

```
const fs = require('fs')
fs.writeFile('Saludo.txt', 'Hola Mundo!', 'utf8', () => {
  fs.readFile('Saludo.txt', 'utf8', (err, data) => {
    console.log(data)
  })
})
```

6. Basado en el ejercicio de tareas, crea una aplicación en Node que cree, lea, renombre y elimine un archivo llamado "shopping.js", cuyo contenido debe ser una lista de productos a comprar en un minimarket, siéntete libre de poner los valores que quieras para este ejercicio propuesto.

Luego del método writeFile, cada método debe estar dentro de la función callback del anterior y debe ser ejecutado dentro del callback de un setTimeout con un tiempo de 2 segundos.

```
const fs = require('fs')

fs.writeFile('shopping.txt', '1) Flips 2) Savoy', 'utf8', () => {
  console.log('Archivo creado')
  setTimeout(() => {
    fs.readFile('shopping.txt', 'utf8', (err, data) => {
      console.log('Contenido del archivo: ' + data)
      setTimeout(() => {
        fs.rename('shopping.txt', 'compras.txt', () => {
          console.log('Archivo renombrado')

          setTimeout(() => {
            fs.unlink('compras.txt', () => {
              console.log('Archivo eliminado')
            })
          }, 2000)
        })
      }, 2000)
    })
  }, 2000)
})
```