



# Introducción a Node

Paquete Node

***Implementar un servidor web de contenidos estáticos y dinámicos utilizando motores de plantillas acorde al entorno Node Express para dar solución a un problema.***

- Unidad 1:  
Introducción a Node
- Unidad 2:  
Node y el gestor de paquetes
- Unidad 3:  
Persistencia



Te encuentras aquí



## ¿Qué aprenderás en esta sesión?

- *Identificar los paquetes más utilizados en las aplicaciones hechas con Node para solucionar un problema planteado*
- *Implementar el paquete nodemon para la automatización del levantamiento de aplicaciones Node*

¿Cómo se diferencia el  
manejo de peticiones  
síncronas y asíncronas en  
Node.js y cuál es su  
impacto en el rendimiento  
de la aplicación?



**`/* Paquetes Node */`**

# Conociendo paquetes en Node

*¿Que son?*

Los paquetes en Node.js son unidades de código reutilizable que contienen funcionalidades específicas y se distribuyen a través del registro de paquetes de npm (Node Package Manager). Cada paquete está formado por archivos y metadatos necesarios para que otras aplicaciones de Node puedan utilizarlo de manera sencilla y efectiva.



# Conociendo paquetes en Node

## *¿Para qué sirven?*

Los paquetes en Node.js son fundamentales para el ecosistema del desarrollo en esta plataforma, ya que ofrecen diversas ventajas y funciones esenciales, entre las que destacan:

- Facilitan la reutilización de código
- Mejoran la modularidad
- Comunidad activa
- Actualizaciones y seguridad

# Conociendo paquetes en Node

## *Paquetes más populares*

A continuación se presenta una lista con una breve descripción de los paquetes más utilizados en los proyectos para Node y que podemos gestionarlos a través de NPM.

### **Express**

Es un framework para aplicaciones web de servidor, famoso por la creación de infraestructuras web rápidas, minimalista y flexibles. Su popularidad crece y se ha convertido en un estándar para el desarrollo Backend en Node.

### **Nodemon**

Es una librería de gran utilidad para el desarrollo, ya que permite ir reflejando los cambios que vamos haciendo en nuestro código a medida que vamos guardando nuestro archivo. Cada vez que guardamos nuestro código, el mismo se compila inmediatamente.



# Conociendo paquetes en Node

## *Paquetes más populares*

### Yargs

En nuestros desarrollos, hay ocasiones en que necesitaremos crear algún script que ejecute alguna tarea en específico, como por ejemplo leer un archivo, automatizar una tarea, etc. Normalmente estos script recibirán parámetros y se comportaron de acuerdo a estos valores enviados. Yargs nos ayudará a formatear a objetos los parámetros que se envíen al ejecutar un script por consola.

### Moment

Es una librería que nos permitirá la manipulación de fechas y horas en nuestros desarrollos, trabajarlas en distintos formatos, acceder a la fecha y hora actual, realizar sumas, restas, comparaciones entre fechas, etc. Una gran gama de funcionalidades que todo desarrollo requiere en poca o gran medida.

### Socket.io

Librería basada en WebSocket, que permite la comunicación bidireccional entre Cliente y Servidor. Con esto podrás realizar tareas en el servidor, por ejemplo, avisar a una aplicación de correos la llegada de un correo nuevo, o a una aplicación de chat indicarle que hay un mensaje nuevo por revisar.

# Conociendo paquetes en Node

## *Paquetes más populares*

### Mocha

Mocha es un framework de pruebas o test que nos entrega muchas características para la evaluación de código. Mocha puede ser implementado en Node o por medio de un navegador. Realiza sus pruebas en serie generando reportes flexibles y exactos. Es muy simple, flexible y divertido en su uso.

### Underscore

Es una librería de JavaScript que nos provee de un conjunto de métodos y funcionalidades que se pueden clasificar en manejo de colecciones, arreglos, funciones y objetos. Permite trabajar con funcionalidades avanzadas de búsqueda y filtros en arreglos.

### Lodash

Este paquete nació como una bifurcación de Underscore por lo que está compuesto en gran parte por las mismas propiedades y métodos, no obstante, es demandado hoy en día mucho en el mercado laboral por sus optimizaciones de rendimiento.

# Conociendo paquetes en Node

## *Paquetes más populares*

### **Morgan**

Es un middleware de registro de sucesos o conocidas como logger, que nos permite escribir en la consola peticiones, errores o lo que necesitemos mostrar o monitorear. Es una gran herramienta para utilizarla en el registro de errores, tanto para el desarrollo como para su funcionamiento productivo.

### **Jim**

Es una biblioteca de procesamiento de imágenes para Node escrita completamente en JavaScript sin dependencias nativas. Su API es muy cómoda de ocupar.

### **Nodemailer**

Nodemailer es un módulo para aplicaciones Node que permite enviar correos electrónicos de forma sencilla. El proyecto comenzó en 2010 cuando no había una opción sensata para enviar mensajes de correo electrónico, hoy es la solución a la que recurren la mayoría de los usuarios de Node de forma predeterminada.

# Conociendo paquetes en Node

## *Paquetes más populares*

### **Axios**

Preferida por la mayoría de los programadores, es una librería para consultas de API REST que te permite hacer peticiones bajo el protocolo HTTP especificando los verbos en forma de métodos, obteniendo en su ejecución una promesa que devuelve como parámetro la respuestas de la consulta.

### **Chalk**

Es un paquete de NPM que ofrece funciones para colorear y personalizar la estética de los mensajes por consola. Contiene una amplia gama de colores y formas de estilizar los mensajes y de esta manera hacer más intuitiva y cómoda la lectura de las reacciones que puedan tener nuestras aplicaciones.

### **UUID**

Por sus siglas en inglés Universally Unique Identifier (Identificador único universal), este paquete es muy usado cuando necesitamos generar un campo identificador de un producto o recurso, su objetivo es generar identificadores para reconocer el objeto dentro de un sistema.

**/\* Nodemon \*/**

# Nodemon



Este paquete no puede faltar en el proceso de desarrollo de servidores con Node. Su objetivo es automatizar el levantamiento del servidor cada vez que salvas cambios en el código, de esta manera no tendrás que preocuparte por cancelar la terminal dando de baja el servidor por cada nueva instrucción o cambio escrito.

Para instalar nodemon deberás ocupar el siguiente comando:

```
npm i nodemon
```

Este paquete tiene la peculiaridad de que no necesita ser importado, simplemente con el hecho de ser instalado podrás ocuparlo como un comando en la terminal.

# Demostración

## "Ups, ¡me equivoqué!"



## Ejercicio guiado

*Ups, ¡me equivoqué!*

Desarrollar una aplicación que imprima por consola un mensaje. La idea será voluntariamente escribir el mensaje que queremos enviar a la consola, al corregirlo y guardar cambios, notar que el servidor fue levantado de nuevo automáticamente arrojando por consola el mensaje corregido

En este ejercicio no serán necesario los pasos porque solo debes tener la siguiente línea de código escrita en tu index.js

```
console.log('Desafio Laram')
```





# Ejercicio guiado

*Ups, ¡me equivoqué!*

Ahora abre la terminal y escribe entonces el siguiente comando

```
nodemon index.js
```

Deberás obtener en la misma terminal lo que te muestro en la siguiente imagen.

```
→ ejercicio lectura git:(master) x nodemon index.js
[nodemon] 1.19.4
[nodemon] to restart at any time, enter `rs`
[nodemon] watching dir(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node index.js`
Desafio Laram
[nodemon] clean exit - waiting for changes before restart
```

**{desafío}**  
latam\_



# Ejercicio guiado

*Ups, ¡me equivoqué!*

Ahora la aplicación está preparada para volver a compilarse cada vez que se guarde un cambio en el código. Para esto cambia la letra “r” por una “t” en la palabra Laram y guarda el cambio. Deberás recibir lo que te muestro en la siguiente imagen por la terminal.

```
→ ejercicio lectura git:(master) x nodemon index.js
[nodemon] 1.19.4
[nodemon] to restart at any time, enter `rs`
[nodemon] watching dir(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node index.js`
Desafio Laram
[nodemon] clean exit - waiting for changes before restart
[nodemon] restarting due to changes...
[nodemon] starting `node index.js`
Desafio Latam
[nodemon] clean exit - waiting for changes before restart
```

**{desafío}**  
latam\_



**/\* CRUD de archivos con File System \*/**

# Gestión de archivos con File System

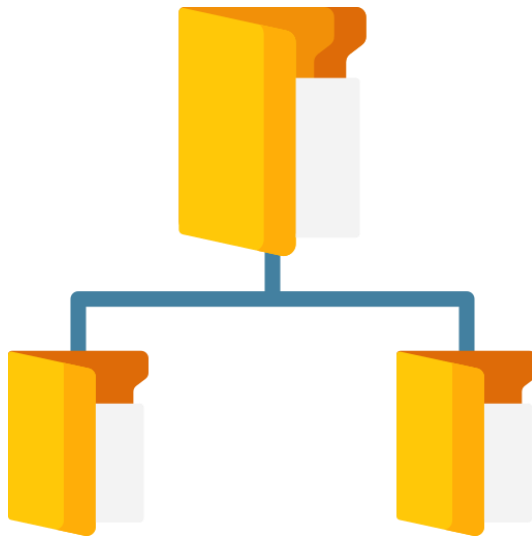
File System es un módulo de Node que permite manipular archivos, es decir, crearlos, editarlos, moverlos, leerlos y eliminarlos.

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

# Gestión de archivos con File System

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.



# 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

- **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.

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

# 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

- 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.

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

# Gestión de archivos con File System

Es importante que sepas que en Node integraron hace varias versiones atrás una nueva API dentro del módulo File System que nos permite manejar los métodos asíncronos como promesas, esto debido a que detectaron problemas de ejecución cuando se mezclan los métodos Sync con funciones asincrónicas. Para acceder a esta nueva API basta con especificar la propiedad promises al momento de requerir el módulo fs en nuestros scripts.

```
const fs = require('fs').promises;
```

Esto nos permitirá además hacer uso de funciones async/await en conjunto con los métodos asíncronos como readFile o writeFile.

**Demostración**  
**“¿Qué tareas tengo  
pendientes para hoy?”**



## Ejercicio guiado

### ¿Qué tareas tengo pendientes para hoy?

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:

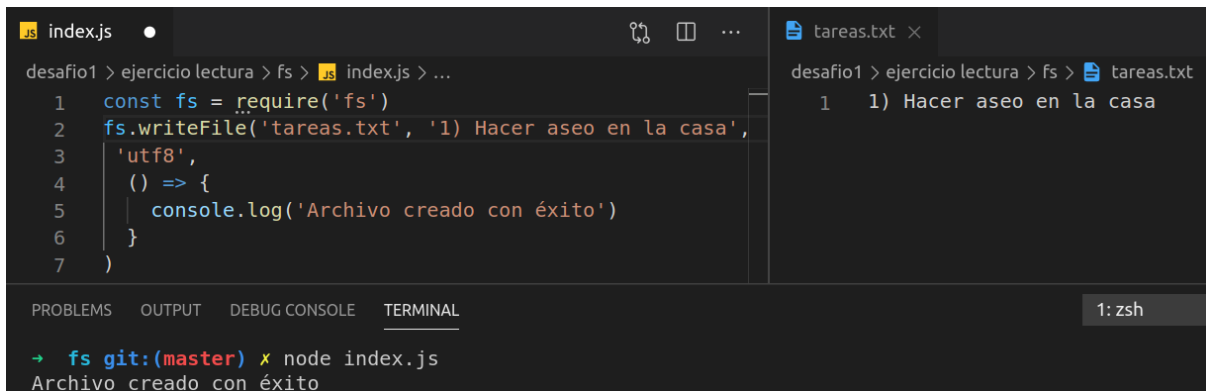
- **Paso 1:** Importar el módulo “fs” en una constante: `const fs = require('fs')`
- **Paso 2:** Usar el método writeFile del módulo “fs” para crear un archivo de nombre “tareas.txt” con una tarea especificando que la codificación será en código utf8 e imprimiendo en pantalla un mensaje de éxito

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

## Ejercicio guiado

*¿Qué tareas tengo pendientes para hoy?*

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 mostramos en la siguiente imagen:



The screenshot shows a VS Code editor with two files open: index.js and tareas.txt. The index.js file contains a script that uses the fs module to write a file named tareas.txt with the content '1) Hacer aseo en la casa' in UTF-8 encoding. The terminal at the bottom shows the command 'node index.js' being executed, resulting in the message 'Archivo creado con éxito'.

```
index.js
desafio1 > ejercicio lectura > fs > JS index.js > ...
1  const fs = require('fs')
2  fs.writeFile('tareas.txt', '1) Hacer aseo en la casa',
3    'utf8',
4    () => {
5      console.log('Archivo creado con éxito')
6    }
7  )

tareas.txt x
desafio1 > ejercicio lectura > fs > tareas.txt
1  1) Hacer aseo en la casa

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL
→ fs git:(master) x node index.js
Archivo creado con éxito
1: zsh
```

## Ejercicio guiado

*¿Qué tareas tengo pendientes para hoy?*

Ahora, intentemos agregar otra tarea al archivo.

- **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.

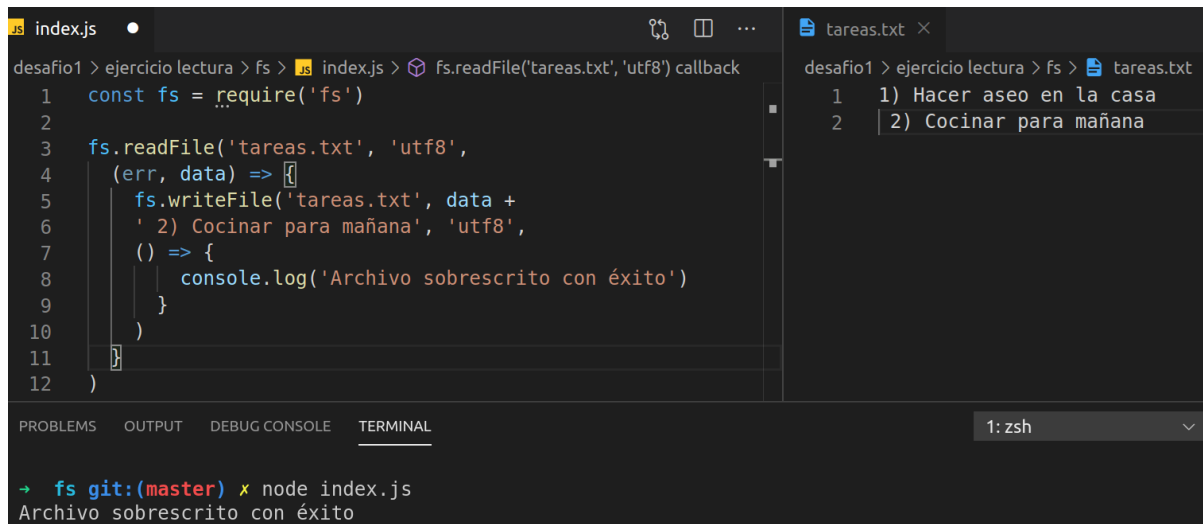
**{desafío}**  
**latam\_**

```
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')
    }
  }
)
```

## Ejercicio guiado

*¿Qué tareas tengo pendientes para hoy?*

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



The screenshot shows a code editor with two files open: `index.js` and `tareas.txt`. The `index.js` file contains JavaScript code that reads `tareas.txt` and appends a new task. The `tareas.txt` file contains a list of tasks. The terminal at the bottom shows the command `node index.js` being executed, resulting in the message "Archivo sobrescrito con éxito".

```
index.js
desafio1 > ejercicio lectura > fs > index.js > fs.readFile('tareas.txt', 'utf8') callback
1  const fs = require('fs')
2
3  fs.readFile('tareas.txt', 'utf8',
4    (err, data) => {
5      fs.writeFile('tareas.txt', data +
6        ' 2) Cocinar para mañana', 'utf8',
7        () => {
8          console.log('Archivo sobrescrito con éxito')
9        }
10     })
11
12 )

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

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL 1: zsh
→ fs git:(master) x node index.js
Archivo sobrescrito con éxito
```

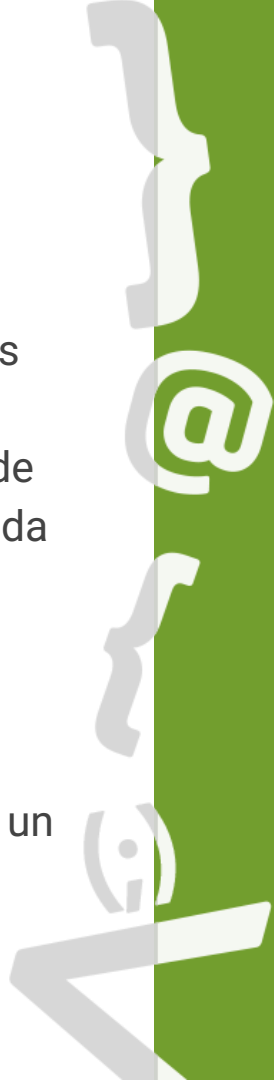
## Ejercicio guiado

*¿Qué tareas tengo pendientes para hoy?*

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, aunque 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.





## Ejercicio guiado

*¿Qué tareas tengo pendientes para hoy?*

- **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('tareass.txt', 'tasks.txt', () => {
  console.log('Archivo renombrado')
  //Paso 2
  fs.unlink('tasks.txt', () => {
    console.log('Archivo eliminado')
  })
})
```



## Ejercicio guiado

*¿Qué tareas tengo pendientes para hoy?*

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



The screenshot shows a VS Code editor with two files open: `index.js` and `tarefas.txt (deleted)`. The `index.js` file contains a script that renames `tarefas.txt` to `tasks.txt` and then deletes it. The terminal at the bottom shows the command `node index.js` being executed, resulting in the messages "Archivo renombrado" and "Archivo eliminado".

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

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

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

¿Cuál es la función de  
Nodemon en el desarrollo de  
aplicaciones con Node.js?



¿Cuáles son los métodos más comunes del módulo "File System" en Node para gestionar archivos?





## Próxima sesión...

- *Describir las características y el rol de un motor de plantillas para el despliegue dinámico de contenidos.*
- *Utilizar motor de plantillas Handlebars para el despliegue de contenido dinámico.*
- *Implementar una vista dinámica utilizando renderización de objetos, variables, páginas parciales y helpers para resolver un problema.*

**{desafío}**  
**latam\_**

*Academia de  
talentos digitales*

