

## EXERCISES QUE TRABAJAREMOS EN EL CUE

- EXERCISE 1: CREANDO UN PROGRAMA CON YARGS.
- EXERCISE 2: DANDO FUNCIONALIDAD AL PROGRAMA.
- 

### EXERCISE 1: CREANDO UN PROGRAMA CON YARGS

Ahora que ya conocemos la funcionalidad básica de **yargs**, utilizaremos lo aprendido para desarrollar un programa que nos permita: crear, actualizar, leer, y eliminar tareas, las cuales se irán guardando en un archivo de texto plano para generar una capa de persistencia.

En primer lugar, creamos la base para la ejecución de **yargs**, definiendo los cuatro comandos que utilizaremos, éstos serán nombrados en base a las operaciones **CRUD**, y agregaremos una pequeña descripción. Recuerda crear un nuevo proyecto, e instalar **yargs** con **"npm install yargs"**.

```
JS index.js  X
JS index.js > ...
1  const yargs = require('yargs');
2
3  const args = yargs
4  .command('create', 'Crear una nueva tarea')
5  .command('read', 'Mostrar todas las tareas')
6  .command('update', 'Actualizar o modificar una tarea')
7  .command('delete', 'Eliminar una tarea')
8  .help()
9  .argv
10

PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE

C:\Users\nodeporcomandos-ejercicios>node index.js --help
index.js [command]

Comandos:
index.js create  Crear una nueva tarea
index.js read    Mostrar todas las tareas
index.js update  Actualizar o modificar una tarea
index.js delete  Eliminar una tarea

Opciones:
--version  Muestra número de versión  [booleano]
--help     Muestra ayuda              [booleano]

C:\Users\nodeporcomandos-ejercicios>[]
```

Ahora, es momento de definir la configuración para cada uno de nuestros comandos. Al igual que en el ejemplo utilizado en el Text Class, escribiremos el objeto de configuración en su propia variable, y luego usaremos dicha variable para alimentar el método.

Para la configuración del comando "create", definiremos dos opciones, las cuales serán pasadas con un valor, título, y contenido; ambas serán obligatorias, ya que nos interesa que al momento de crear una tarea, ésta tenga un título y un contenido.

```
2
3  const createConfig = {
4    titulo: {
5      describe: 'El nombre de la tarea a realizar',
6      alias: 't',
7      demandOption: true
8    },
9    contenido: {
10     describe: 'Descripcion de la tarea a realizar',
11     alias: 'c',
12     demandOption: true
13   }
14 }
15
```

Ahora, utilizamos este objeto, y lo pasamos como tercer argumento del primer método `command()`.

```
14 }
15
16 const args = yargs
17 .command('create', 'Crear una nueva tarea', createConfig)
18 .command('read', 'Mostrar todas las tareas')
19 .command('update', 'Actualizar o modificar una tarea')
20 .command('delete', 'Eliminar una tarea')
21 .help()
22 .argv
23
24
```

Y al ejecutar `"node index.js create --help"`, observamos que ya tenemos a nuestra disposición las opciones definidas en el objeto de configuración.

Para el comando `read`, no necesitaremos ninguna opción especial, ya que con éste solo queremos mostrar por consola todas las tareas guardadas.

Para el comando **update**, crearemos las opciones título y contenido. Aunque las opciones del comando **create** sean las mismas, esto no causará conflicto, ya que están relacionadas al comando con el cual están definidas. Por lo tanto, cuando lleguemos a definir la funcionalidad de los comandos, cada opción hará lo que le corresponda cuando su comando sea utilizado. En este caso, las opciones no serán requeridas, debido a que es posible que solo necesitemos actualizar el título y no el contenido, o viceversa.

```

16
17   const updateConfig = {
18     titulo: {
19       describe: 'Nuevo nombre de la tarea a realizar',
20       alias: 't',
21     },
22     contenido: {
23       describe: 'Nueva Descripción de la tarea a realizar',
24       alias: 'c',
25     }
26   }
27

```

Pasamos el objeto de configuración, como tercer argumento del tercer método **command()**.

```

const args = yargs
.command('create', 'Crear una nueva tarea', createConfig)
.command('read', 'Mostrar todas las tareas')
.command('update', 'Actualizar o modificar una tarea', updateConfig)
.command('delete', 'Eliminar una tarea')
.help()
.argv

```

Y al ejecutar nuestro programa con **"node index.js update --help"**, ya tenemos disponibles nuestras opciones.

```

C:\Users\nodeporcomandos-ejercicios>node index.js update --help
index.js update

Actualizar o modificar una tarea

Opciones:
  --version  Muestra número de versión      [booleano]
  --help     Muestra ayuda                  [booleano]
  -t, --titulo  Nuevo nombre de la tarea a realizar
  -c, --contenido  Nueva Descripción de la tarea a realizar
C:\Users\nodeporcomandos-ejercicios>

```

Por último, el comando **delete** recibirá una opción, la cual tendrá el identificador de la tarea que se quiere eliminar. Ésta será obligatoria, debido a que nuestro programa requiere alguna forma para encontrar la tarea a eliminar.

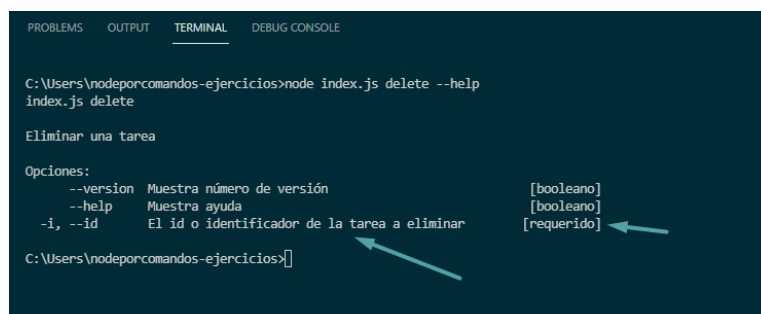
Definimos el objeto de configuración con la opción **id**, su alias y **demandOption**, definida como **true**.

```
26
27 const deleteConfig = {
28   id: {
29     describe: "El id o identificador de la tarea a eliminar",
30     alias: "i",
31     demandOption: true
32   }
33 }
```

Agregamos este objeto como tercer parámetro del cuarto método **command()**, en nuestra cadena de métodos.

```
34
35 const args = yargs
36 .command('create', 'Crear una nueva tarea', createConfig)
37 .command('read', 'Mostrar todas las tareas')
38 .command('update', 'Actualizar o modificar una tarea', updateConfig)
39 .command('delete', 'Eliminar una tarea', deleteConfig)
40 .help()
41 .argv
42
43
```

Y, por último, probaremos el programa con el comando **delete** y la opción **-help**, para confirmar que nuestra opción se encuentra disponible.



```
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

C:\Users\nodeporcomandos-ejercicios>node index.js delete --help
index.js delete

Eliminar una tarea

Opciones:
  --version  Muestra número de versión      [booleano]
  --help     Muestra ayuda                  [booleano]
  -i, --id   El id o identificador de la tarea a eliminar [requerido]

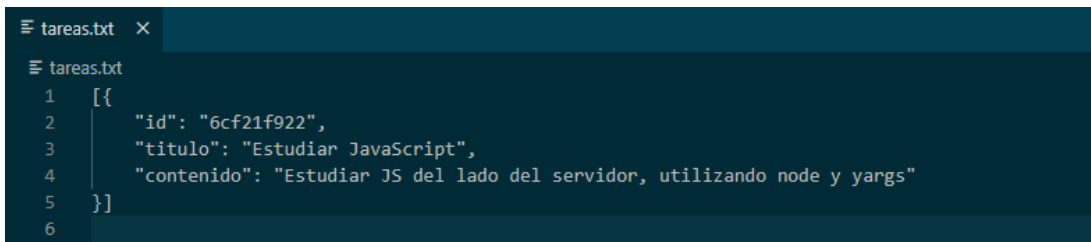
C:\Users\nodeporcomandos-ejercicios>
```

## EXERCISE 2: DANDO FUNCIONALIDAD AL PROGRAMA

Nuestro programa ya tiene comandos y opciones bien definidas, por lo tanto, es momento de darle funcionalidad a éstos.

Primero, crearemos un archivo de texto llamado `tareas.txt`, y partiremos con una tarea inicial. El formato que utilizaremos será el de un Array de objetos.

```
1 [{
2   "id": "6cf21f922",
3   "titulo": "Estudiar JavaScript",
4   "contenido": "Estudiar JS del lado del servidor, utilizando
5 node y yargs"
6 }]
7
```



```
tareas.txt x
tareas.txt
1  [{
2    "id": "6cf21f922",
3    "titulo": "Estudiar JavaScript",
4    "contenido": "Estudiar JS del lado del servidor, utilizando node y yargs"
5  }]
6
```

Crearemos una función para el comando `create`, la cual se encargará de definir el id de nuestra nueva tarea, y escribirla en el archivo `tareas.txt`.

Definiremos el id, utilizando el paquete `uuid(npm install uuid)`, pero esta vez solo los primeros 8 caracteres del id generado por `uuid`.

```
const funcionCreate = () => {
  const id = uuidv4().slice(0,8);
}
```

Recuerda importar el paquete `uuid`, tal como está definido en la documentación.

```
JS index.js  X
JS index.js > ...
1  const yargs = require('yargs');
2  const { v4: uuidv4 } = require('uuid') ←
3
4
```

Para que nuestra función sea capaz de recibir el valor contenido en las opciones título y contenido, debemos definir su parámetro, que también recibirá el objeto entregado por **yargs**, y que contiene todas las opciones y sus valores, en este caso: título y contenido.

```
const funcionCreate = (argv) => {
  const id = uuidv4().slice(0,8);
  const titulo = argv.titulo; ←
  const contenido = argv.contenido; ←
}
```

Y como hemos visto anteriormente, podemos utilizar **Destructuring** en la definición de parámetros de la función.

```
37
38  const funcionCreate = ({titulo, contenido}) => {
39    const id = uuidv4().slice(0,8);
40  }
41
42
```

Ahora, debemos definir el nuevo objeto que será insertado en el Array de objetos, contenido en el archivo de texto.

```

43
44 const funcionCreate = ({titulo, contenido}) => {
45   const id = uuidv4().slice(0,8);
46   const nuevaTarea = { id: id, titulo: titulo, contenido: contenido }; ←
47
48
49 }
50

```

También debemos obtener el contenido actual del archivo de texto, para así poder sumar nuestra nueva tarea a las ya existentes. Transformaremos nuestra función en una asíncrona con la palabra clave **async**, leeremos el archivo de texto, y guardaremos su contenido en una variable, transformándolo a objeto para poder trabajar con él.

```

43
44 const funcionCreate = async ({titulo, contenido}) => {
45   const id = uuidv4().slice(0,8);
46   const nuevaTarea = { id: id, titulo: titulo, contenido: contenido };
47
48   const tareas = await fs.readFile('tareas.txt');
49   const arrayTareas = JSON.parse(tareas);
50
51 }
52

```

Ahora, agregaremos la nueva tarea al Array, y utilizaremos el Array resultante para escribir la información al archivo tareas.txt, empleando: **JSON.stringify()**.

```

43
44 const funcionCreate = async ({titulo, contenido}) => {
45   const id = uuidv4().slice(0,8);
46   const nuevaTarea = { id: id, titulo: titulo, contenido: contenido };
47
48   const tareas = await fs.readFile('tareas.txt');
49   const arrayTareas = JSON.parse(tareas);
50
51   arrayTareas.push(nuevaTarea)
52
53   await fs.writeFile('tareas.txt', JSON.stringify(arrayTareas, null, 2));
54   console.log('Nueva tarea agregada');
55 }
56

```

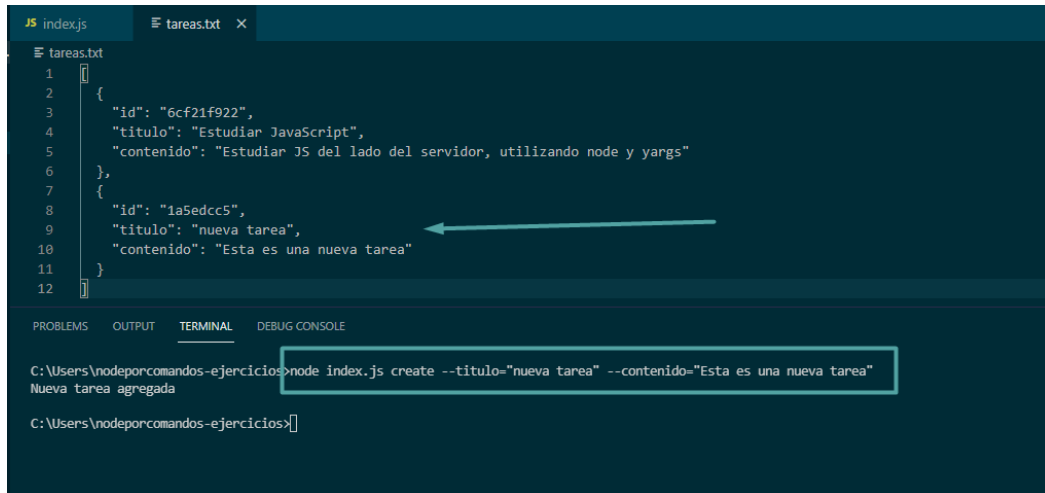
Ahora, ya podemos agregar nuestra nueva función al primer método **command**, pasándola como cuarto parámetro.

```

56
57
58 const args = yargs
59 .command('create', 'Crear una nueva tarea', createConfig, (argv) => funcionCreate(argv))
60 .command('read', 'Mostrar todas las tareas')
61 .command('update', 'Actualizar o modificar una tarea', updateConfig)
62 .command('delete', 'Eliminar una tarea', deleteConfig)
63 .help()
64 .argv
65
66

```

Y si ejecutamos nuestro programa con el comando **create**, y escribimos las opciones con una nueva tarea, obtenemos el siguiente resultado.



```

JS index.js  tareass.txt x
tareass.txt
1  [
2  {
3    "id": "6cf21f922",
4    "titulo": "Estudiar JavaScript",
5    "contenido": "Estudiar JS del lado del servidor, utilizando node y yargs"
6  },
7  {
8    "id": "1a5edcc5",
9    "titulo": "nueva tarea",
10   "contenido": "Esta es una nueva tarea"
11  }
12 ]

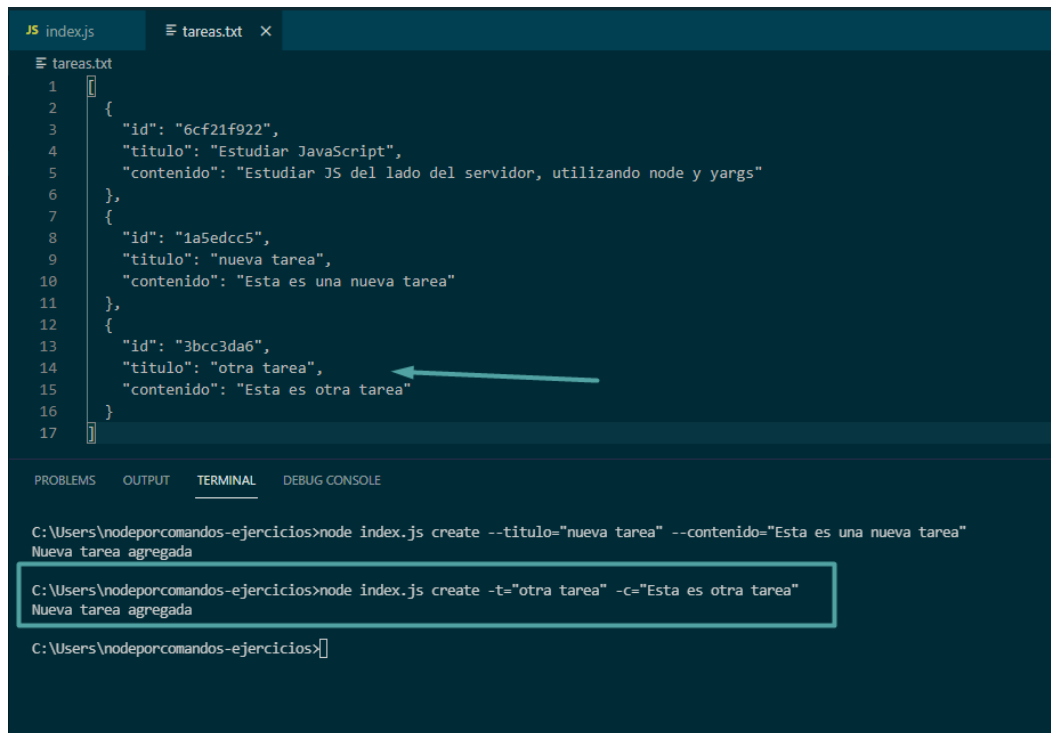
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE
C:\Users\nodeporcomandos-ejercicios>node index.js create --titulo="nueva tarea" --contenido="Esta es una nueva tarea"
Nueva tarea agregada

C:\Users\nodeporcomandos-ejercicios>

```

También podemos utilizar los alias que hemos definido para las opciones, la diferencia es que éstos son usados con un solo guion "-".





```

JS index.js  tareas.txt x
tareas.txt
1  {
2    {
3      "id": "6cf21f922",
4      "titulo": "Estudiar JavaScript",
5      "contenido": "Estudiar JS del lado del servidor, utilizando node y yargs"
6    },
7    {
8      "id": "1a5edcc5",
9      "titulo": "nueva tarea",
10     "contenido": "Esta es una nueva tarea"
11   },
12   {
13     "id": "3bcc3da6",
14     "titulo": "otra tarea",
15     "contenido": "Esta es otra tarea"
16   }
17 }

PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE

C:\Users\nodeporcomandos-ejercicios>node index.js create --titulo="nueva tarea" --contenido="Esta es una nueva tarea"
Nueva tarea agregada

C:\Users\nodeporcomandos-ejercicios>node index.js create -t="otra tarea" -c="Esta es otra tarea"
Nueva tarea agregada

C:\Users\nodeporcomandos-ejercicios>
  
```

Para el comando **read**, solo debemos implementar una función que lea el contenido del archivo, y devuelva las tareas en el archivo de texto. Luego, podemos imprimir por consola.

```

56
57  const funcionRead = async () => {
58    const tareasArchivo = fs.readFile('tareas.txt');
59    const arrayTareas = JSON.parse(tareasArchivo);
60
61    console.log(arrayTareas);
62  }
63
64
65
  
```

Debido a que el comando **read** no tiene configuraciones asociadas, debemos pasar un objeto vacío como tercer argumento, y luego la función que hemos creado.

```

65
66 const args = yargs
67 .command('create', 'Crear una nueva tarea', createConfig, (argv) => funcionCreate(argv))
68 .command('read', 'Mostrar todas las tareas', {}, (argv) => funcionRead())
69 .command('update', 'Actualizar o modificar una tarea', updateConfig)
70 .command('delete', 'Eliminar una tarea', deleteConfig)
71 .help()
72 .argv
73
  
```

Esto, al ejecutar nuestro programa con la opción **read**, nos entrega el siguiente resultado.

```

C:\Users\nodeporcomandos-ejercicios>node index.js read
[
  {
    id: '6cf21f922',
    titulo: 'Estudiar JavaScript',
    contenido: 'Estudiar JS del lado del servidor, utilizando node y yargs'
  },
  {
    id: '1a5edcc5',
    titulo: 'nueva tarea',
    contenido: 'Esta es una nueva tarea'
  },
  {
    id: '3bcc3da6',
    titulo: 'otra tarea',
    contenido: 'Esta es otra tarea'
  }
]
C:\Users\nodeporcomandos-ejercicios>
  
```

Debido a que solo estamos imprimiendo el Array, el formato de la respuesta puede no ser tan amigable, desde el punto de vista de un usuario que no es programador. Entonces, podemos darle un formato que si lo considere, pensando en un usuario que no esté familiarizado con el formato **Json**. Definiremos un contador para enumerar cada tarea, utilizaremos un ciclo **for of** para iterar sobre el Array, y luego imprimiremos las propiedades de cada objeto dentro de éste.

```

61
62 const funcionRead = async () => {
63   const tareasArchivo = await fs.readFile('tareas.txt');
64   const arrayTareas = JSON.parse(tareasArchivo);
65   let contador = 0;
66
67   for (tareas of arrayTareas){
68     const { titulo, contenido, id } = tareas;
69     contador++;
70     console.log(`Tarea numero ${contador}:`);
71     console.log(`- Titulo: ${titulo}`);
72     console.log(`- Contenido: ${contenido}`);
73     console.log(`- id: ${id}`);
74     console.log("");
75   }
76
77 }
  
```

Este nuevo código nos da una respuesta más amigable desde el punto de vista de un usuario.



```

PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE

C:\Users\nodeporcomandos-ejercicios>node index.js read
Tarea numero 1:
- Titulo: Estudiar JavaScript
- Contenido: Estudiar JS del lado del servidor, utilizando node y yargs
- id: s6cf21f922

Tarea numero 2:
- Titulo: nueva tarea
- Contenido: Esta es una nueva tarea
- id: s1a5edcc5

Tarea numero 3:
- Titulo: otra tarea
- Contenido: Esta es otra tarea
- id: s3bcc3da6
  
```

Para el comando **créate**, construiremos una nueva función, y al igual que en la anterior, partiremos por leer el contenido actual en nuestro archivo de texto, y lo guardaremos en una variable utilizando **JSON.parse()**.

```

73
74  const funcionUpdate = async () => {
75      const tareasArchivo = await fs.readFile('tareas.txt');
76      const arrayTareas = JSON.parse(tareasArchivo);
77  }
78
79
  
```

Para poder actualizar una tarea, debemos obtener su id. Por lo tanto, agregaremos una opción "id" a la configuración del comando **update**, con la opción **demandOption** como true, y un alias **"i"**.

```

24
25  const updateConfig = {
26      titulo: {
27          describe: 'Nuevo nombre de la tarea a realizar',
28          alias: 't',
29      },
30      contenido: {
31          describe: 'Nueva Descripción de la tarea a realizar',
32          alias: 'c',
33      },
34      id: {
35          describe: 'El id de la tarea a actualizar o modificar',
36          alias: 'i',
37          demandOption: true
38      }
39  }
40
  
```

Ahora, debemos modificar nuestra función para que reciba el objeto con las opciones ingresadas. Usaremos **destructuring** para recibir el contenido de éstas.

```

77 }
78
79 const funcionUpdate = async ({id, titulo, contenido}) => {
80   const tareasArchivo = await fs.readFile('tareas.txt');
81   const arrayTareas = JSON.parse(tareasArchivo);
82
83
84
85 }

```

También debemos obtener el objeto dentro del Array que deseamos buscar. Para ello, utilizaremos el método `findIndex()` de **JavaScript**. El resultado de éste contendrá el **index** del objeto que coincida con la condición de búsqueda.

```

78
79 const funcionUpdate = async ({id, titulo, contenido}) => {
80   const tareasArchivo = await fs.readFile('tareas.txt');
81   const arrayTareas = JSON.parse(tareasArchivo);
82   const tareaActual = arrayTareas.findIndex( tarea => tarea.id === id);
83
84
85

```

Debido a que las opciones título y contenido no son obligatorias, debemos controlar que el valor de éstas tenga información, pues de lo contrario, será **“undefined”**. Para esto, definiremos dos variables, y su valor dependerá de si su “título” y “contenido” se encuentran indefinidas, o no. Si la opción se encuentra como indefinida, entonces utilizaremos el valor de la tarea actual, y de lo contrario, emplearemos el nuevo.

Para esto haremos uso del operador ternario. Si no estás familiarizado con éste, revisa la sección Hints, donde encontrarás una amplia explicación al respecto.

```

78
79 const funcionUpdate = async ({id, titulo, contenido}) => {
80   const tareasArchivo = await fs.readFile('tareas.txt');
81   const arrayTareas = JSON.parse(tareasArchivo);
82   const tareaActual = arrayTareas.find( tarea => tarea.id === id);
83
84   const tituloNuevo = titulo ? titulo : tareaActual.titulo;
85   const contenidoNuevo = contenido ? contenido : tareaActual.contenido;
86
87
88
89

```

Ahora que ya tenemos nuestras nuevas propiedades, podemos definirlas en el objeto.

```

78
79 const funcionUpdate = async ({id, titulo, contenido}) => {
80   const tareasArchivo = await fs.readFile('tareas.txt');
81   const arrayTareas = JSON.parse(tareasArchivo);
82   const tareaActual = arrayTareas.findIndex( tarea => tarea.id === id);
83
84   const tituloNuevo = titulo ? titulo : arrayTareas[tareaActual].titulo;
85   const contenidoNuevo = contenido ? contenido : arrayTareas[tareaActual].contenido;
86
87   arrayTareas[tareaActual].titulo = tituloNuevo;
88   arrayTareas[tareaActual].contenido = contenidoNuevo;
89
90
91
92
93

```

Y utilizamos el método `writeFile()` del module `fs`, para escribir el Array resultante en nuestro archivo `tareas.txt`.

```

78
79 const funcionUpdate = async ({id, titulo, contenido}) => {
80   const tareasArchivo = await fs.readFile('tareas.txt');
81   const arrayTareas = JSON.parse(tareasArchivo);
82   const tareaActual = arrayTareas.findIndex( tarea => tarea.id === id);
83
84   const tituloNuevo = titulo ? titulo : arrayTareas[tareaActual].titulo;
85   const contenidoNuevo = contenido ? contenido : arrayTareas[tareaActual].contenido;
86
87   arrayTareas[tareaActual].titulo = tituloNuevo;
88   arrayTareas[tareaActual].contenido = contenidoNuevo;
89
90   await fs.writeFile('tareas.txt', JSON.stringify(arrayTareas, null, 2));
91   console.log('Tu tarea ha sido actualizada')
92
93
94

```

Luego, agregamos nuestra función como cuarto parámetro del tercer método `command()` de la cadena de métodos en `yargs`.

```

95
96 const args = yargs
97 .command('create', 'Crear una nueva tarea', createConfig, (argv) => funcionCreate(argv))
98 .command('read', 'Mostrar todas las tareas', {}, (argv) => funcionRead())
99 .command('update', 'Actualizar o modificar una tarea', updateConfig, (argv) => funcionUpdate(argv))
100 .command('delete', 'Eliminar una tarea', deleteConfig)
101 .help()
102 .argv
103
104

```

Y para probar esta función, utilizaremos la primera tarea en la lista, y su id (6cf21f922).



```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE

C:\Users\nodeporcomandos-ejercicios>node index.js update -i=6cf21f922 -c-"Estudiando JavaScript del lado del cliente, utilizando frameworks"
Tu tarea ha sido actualizada

C:\Users\nodeporcomandos-ejercicios>node index.js read
Tarea numero 1:
- Titulo: Estudiar JavaScript
- Contenido: Estudiando JavaScript del lado del cliente, utilizando frameworks
- id: s6cf21f922

Tarea numero 2:
- Titulo: nueva tarea
- Contenido: Esta es una nueva tarea
- id: s1a5edcc5

Tarea numero 3:
- Titulo: otra tarea
- Contenido: Esta es otra tarea
- id: s3bcc3da6

C:\Users\nodeporcomandos-ejercicios>
```

También podemos actualizar el título.

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE

C:\Users\nodeporcomandos-ejercicios>node index.js update -i=6cf21f922 -t=JS
Tu tarea ha sido actualizada

C:\Users\nodeporcomandos-ejercicios>node index.js read
Tarea numero 1:
- Titulo: JS
- Contenido: Estudiando JavaScript del lado del cliente, utilizando frameworks
- id: s6cf21f922

Tarea numero 2:
- Titulo: nueva tarea
- Contenido: Esta es una nueva tarea
- id: s1a5edcc5

Tarea numero 3:
- Titulo: otra tarea
- Contenido: Esta es otra tarea
- id: s3bcc3da6

C:\Users\nodeporcomandos-ejercicios>
```

Para nuestro comando **delete**, solo necesitaremos recibir el id, aunque también debemos leer el archivo, y guardar su contenido en una variable. Además, lo podemos recibir directamente utilizando **destructuring** en los parámetros de entrada

```
94
95 const funcionDelete = async ({id}) => {
96   const tareasArchivo = await fs.readFile('tareas.txt');
97   const arrayTareas = JSON.parse(arrayTareas);
98 }
99
100
```

Para eliminar el objeto con el id recibido, utilizaremos el método **filter**, el cual devuelve todos los objetos que cumplan con la condición entregada. Por lo tanto, filtraremos las tareas, buscando todas aquellas en donde el id no sea igual al que se entregó.

```
94
95 const funcionDelete = async ({id}) => {
96   const tareasArchivo = await fs.readFile('tareas.txt');
97   const arrayTareas = JSON.parse(tareasArchivo);
98   console.log(id);
99
100   const nuevasTareas = arrayTareas.filter(tareas => tareas.id !== id);
101 }
102
103
```

Ahora, ya solo nos queda utilizar el nuevo Array de tareas para escribir la información a nuestro archivo de texto, utilizando el método **writeFile()** del módulo **fs**.

```
94
95 const funcionDelete = async ({id}) => {
96   const tareasArchivo = await fs.readFile('tareas.txt');
97   const arrayTareas = JSON.parse(tareasArchivo);
98
99   const nuevasTareas = arrayTareas.filter(tareas => tareas.id !== id);
100
101   await fs.writeFile('tareas.txt', JSON.stringify(nuevasTareas, null, 2));
102   console.log("La tarea ha sido eliminada exitosamente");
103 }
104
105
```

Y por último, probamos nuestro programa.



```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE

C:\Users\nodeporcomandos-ejercicios>node index.js read
Tarea numero 1:
- Titulo: JS
- Contenido: Estudiando JavaScript del lado del cliente, utilizando frameworks
- id: 6cf21f922

Tarea numero 2:
- Titulo: nueva tarea
- Contenido: Esta es una nueva tarea
- id: 1a5edcc5

Tarea numero 3:
- Titulo: otra tarea
- Contenido: Esta es otra tarea
- id: 3bcc3da6

C:\Users\nodeporcomandos-ejercicios>node index.js delete --id=3bcc3da6
La tarea ha sido eliminada exitosamente

C:\Users\nodeporcomandos-ejercicios>node index.js read
Tarea numero 1:
- Titulo: JS
- Contenido: Estudiando JavaScript del lado del cliente, utilizando frameworks
- id: 6cf21f922

Tarea numero 2:
- Titulo: nueva tarea
- Contenido: Esta es una nueva tarea
- id: 1a5edcc5

C:\Users\nodeporcomandos-ejercicios>
```

En este punto hemos logrado crear un CLI, el cual permite una funcionalidad **CRUD**, utilizando persistencia de datos mediante archivos de texto plano.