

Node.js

Objetivos del tema

- Entender la filosofía detrás de node.js
- Para qué sirve node.js
- Aprender a interactuar con la plataforma
- Aprender a interactuar con servicios externos

Antes de empezar

Requisitos

- editor de texto
- **node.js** versión 10
- familiaridad con **Javascript**
- *paciencia*

Introducción

Fundamentos

- Para **ejecutar código**:
 - escribe el código en un fichero
 - ejecuta el comando:

```
node fichero.js
```

Fundamentos

- Para **depurar código**:
 - puntos de ruptura con **debugger**
 - ejecuta el comando:

```
node inspect fichero.js
```

```
console.log('hola')
```

```
let a = 1
```

```
function suma(a, b) {  
  return a + b  
}
```

```
debugger
```

```
console.log('> a:', a)  
suma(1, 1)  
console.log('adios')
```



```
$ node inspect test.js
< Debugger listening on ws://127.0.0.1:9229/61a499ea
< For help, see: https://nodejs.org/en/docs/inspector
< Debugger attached.
Break on start in test.js:1
> 1 (function (exports, require, module, __filename, __dirname) {
  console.log('hola')
    2 let a = 1
    3 function suma(a, b) {
debug>
```

```
$ node inspect test.js
```

```
< Debugger listening on ws://127.0.0.1:9229/61a499ea
```

```
< For help, see: https://nodejs.org/en/docs/inspector
```

```
< Debugger attached.
```

```
Break on start in test.js:1
```

```
> 1 (function (exports, require, module, __filename, __dirname) {  
  console.log('hola')
```

```
  2 let a = 1
```

```
  3 function suma(a, b) {
```

```
debug>
```

```
$ node inspect test.js
```

```
< Debugger listening on ws://127.0.0.1:9229/61a499ea
```

```
< For help, see: https://nodejs.org/en/docs/inspector
```

```
< Debugger attached.
```

```
Break on start in test.js:1
```

```
> 1 (function (exports, require, module, __filename, __dirname) {
```

```
  console.log('hola')
```

```
    2 let a = 1
```

```
    3 function suma(a, b) {
```

```
debug>
```

```
$ node inspect test.js
< Debugger listening on ws://127.0.0.1:9229/61a499ea
< For help, see: https://nodejs.org/en/docs/inspector
< Debugger attached.
Break on start in test.js:1
> 1 (function (exports, require, module, __filename, __dirname) {
  console.log('hola')
  2 let a = 1
  3 function suma(a, b) {
debug>
```

```
$ node inspect test.js
< Debugger listening on ws://127.0.0.1:9229/61a499ea
< For help, see: https://nodejs.org/en/docs/inspector
< Debugger attached.
Break on start in test.js:1
> 1 (function (exports, require, module, __filename, __dirname) {
  console.log('hola')
    2 let a = 1
    3 function suma(a, b) {
debug>
```

Comandos del depurador

- `list(n)`
 - Muestra n líneas alrededor
- `repl`
 - Arranca el intérprete de node
- `pause`
 - Interrumpe la ejecución

Comandos del depurador

- **cont, c**
 - Continúa la ejecución
- **next, n**
 - Ejecuta una línea (con salto)
- **step, s**
 - Ejecuta una línea (sin salto)
- **out, o**
 - Ejecuta hasta escapar del contexto

Comandos del depurador

- Depurar con DevTools
 - `node --inspect-brk test.js`
 - abre Chrome
 - ve a la url `chrome://inspect`
 - deberías ver el depurador escuchando
 - haz click en inspect

Lenguaje disponible

- Node.js **10.1.0** soporta:
 - **ES2015**: 99%
 - **ES2016**: 100%
 - **ES2017**: 100%
 - **ES2018**: 100%
 - <https://node.green/>

Errores

- Cuando se lanza una excepción y no se captura...
 - El proceso termina
 - Se muestra un resumen de la excepción en la consola

```
function ping(n = 0) {  
  if (n > 5) throw new Error('Oh, noes!');  
  return pong(n)  
}
```

```
function pong(n) {  
  return ping(n + 1)  
}
```

```
ping()
```

```
$ node error.js
  /path/to/error.js:2
if (n > 5) throw new Error('Oh, noes!');
  ^
```

```
Error: Oh, noes!
at ping (/path/to/error.js:2:20)
at pong (/path/to/error.js:7:10)
at ping (/path/to/error.js:3:10)
at pong (/path/to/error.js:7:10)
at ping (/path/to/error.js:3:10)
at pong (/path/to/error.js:7:10)
at ping (/path/to/error.js:3:10)
at pong (/path/to/error.js:7:10)
at ping (/path/to/error.js:3:10)
at pong (/path/to/error.js:7:10)
```

```
$ node error.js
```

```
  /path/to/error.js:2  
if (n > 5) throw new Error('Oh, noes!');  
  ^
```

Error: Oh, noes!

at ping (/path/to/error.js:2:20)

at pong (/path/to/error.js:7:10)

at ping (/path/to/error.js:3:10)

at pong (/path/to/error.js:7:10)

at ping (/path/to/error.js:3:10)

at pong (/path/to/error.js:7:10)

at ping (/path/to/error.js:3:10)

at pong (/path/to/error.js:7:10)

at ping (/path/to/error.js:3:10)

at pong (/path/to/error.js:7:10)

```
$ node error.js
  /path/to/error.js:2
if (n > 5) throw new Error('Oh, noes!');
  ^
```

Error: Oh, noes!

```
at ping (/path/to/error.js:2:20)
at pong (/path/to/error.js:7:10)
at ping (/path/to/error.js:3:10)
at pong (/path/to/error.js:7:10)
at ping (/path/to/error.js:3:10)
at pong (/path/to/error.js:7:10)
at ping (/path/to/error.js:3:10)
at pong (/path/to/error.js:7:10)
at ping (/path/to/error.js:3:10)
at pong (/path/to/error.js:7:10)
```

```
$ node error.js
  /path/to/error.js:2
if (n > 5) throw new Error('Oh, noes!');
  ^
```

Error: Oh, noes!

```
at ping (/path/to/error.js:2:20)
at pong (/path/to/error.js:7:10)
at ping (/path/to/error.js:3:10)
at pong (/path/to/error.js:7:10)
at ping (/path/to/error.js:3:10)
at pong (/path/to/error.js:7:10)
at ping (/path/to/error.js:3:10)
at pong (/path/to/error.js:7:10)
at ping (/path/to/error.js:3:10)
at pong (/path/to/error.js:7:10)
```

Módulos

Módulos

- Cada fichero es **un módulo independiente**
- Se ejecuta en un **ámbito aislado**
- Puede **importar** otros módulos
- Puede **exportar** nombres

Módulos

- El código de un módulo se **ejecuta** cuando:
 - Lo pasamos como parámetro al invocar a **node**
 - **La primera vez** que es importado

colors.js

```
exports.red = '#d30139'  
exports.green = '#01d339'  
exports.blue = '#0139d3'
```

colors.js

```
exports.red = '#d30139'  
exports.green = '#01d339'  
exports.blue = '#0139d3'
```

index.js

```
const colors = require('./colors.js')  
console.log('Red:', colors.red)
```

index.js

```
const colors = require('./colors.js')  
console.log('Red:', colors.red)
```

index.js

```
const colors = require('./colors.js')  
console.log('Red:', colors.red)
```

Módulos

- Dentro del ámbito del módulo tenemos 4 **variables**
 - **module**: *referencia al módulo actual*
 - **require**: *función para importar otros módulos*
 - **__filename**: *string con la ruta del fichero del módulo*
 - **__dirname**: *string con la ruta del directorio del módulo*

index.js

```
console.log(`Gracias por ejecutar ${__filename}`)
```

Módulos

- Los módulos solo se ejecutan **una vez**
 - Se cachea todo lo que haya exportado
 - La siguiente vez que se requiera, se utilizan los valores cacheados

module.js

console.log(*'Running!'*)

index.js

```
require('./module.js')  
require('./module.js')  
require('./module.js')  
require('./module.js')
```

Módulos

- La ruta que pasamos a **require**...
 - Si empieza con **/** es una ruta absoluta
 - Si empieza con **./** o **../** es una ruta relativa
 - En caso contrario, ruta relativa a **\$NODE_PATH**
 - `node_modules`

Módulos

- Si la ruta **es un directorio**....
 - node busca en el interior del directorio
 - intenta cargar el fichero **index.js**
 - si no existe, levanta un error

module.js

```
console.log(module.exports) // {}  
exports.prop = 1  
console.log(module.exports) // { prop: 1 }
```

module.js

```
module.exports.otherProp = 2  
console.log(exports.otherProp) // 2  
console.log(module.exports === exports) // true
```


module.js

```
module.exports = { prop: 'oh, noes!' }  
console.log(module.exports === exports) // FALSE!
```

Módulos

- La expresión **require(...)**
 - devuelve el valor de **module.exports** del módulo importado
 - habitualmente es **un objeto**
 - pero puede ser **cualquier valor**

module.js

module.exports = *'Something else'*

index.js

```
const what = require('./module.js')  
console.log(what) // Something else
```

npm

npm

- **N**ode **P**ackage **M**anager
- `package.json`
- Múltiples usos:
 - gestor de dependencias
 - configuración del proyecto
 - comandos de compilación, testing y despliegue

npm

- Inicializamos un proyecto con: **npm init**
 - Un asistente para crear la configuración inicial
 - Genera el fichero **package.json**

```
{
  "name": "test",
  "version": "1.0.0",
  "description": "Just a test package",
  "main": "index.js",
  "scripts": {
    "test": "mocha ./tests"
  },
  "repository": {
    "type": "git",
    "url": "http://www.github.com/test/test"
  },
  "keywords": [
    "test"
  ],
  "author": "Elias Alonso",
  "license": "ISC"
}
```



```
{  
  "name": "test",  
  "version": "1.0.0",  
  "description": "Just a test package",  
  "main": "index.js",  
  "scripts": {  
    "test": "mocha ./tests"  
  },  
  "repository": {  
    "type": "git",  
    "url": "http://www.github.com/test/test"  
  },  
  "keywords": [  
    "test"  
  ],  
  "author": "Elias Alonso",  
  "license": "ISC"  
}
```

```
{  
  "name": "test",  
  "version": "1.0.0",  
  "description": "Just a test package",  
  "main": "index.js",  
  "scripts": {  
    "test": "mocha ./tests"  
  },  
  "repository": {  
    "type": "git",  
    "url": "http://www.github.com/test/test"  
  },  
  "keywords": [  
    "test"  
  ],  
  "author": "Elias Alonso",  
  "license": "ISC"  
}
```

```
{  
  "name": "test",  
  "version": "1.0.0",  
  "description": "Just a test package",  
  "main": "index.js",  
  "scripts": {  
    "test": "mocha ./tests"  
  },  
  "repository": {  
    "type": "git",  
    "url": "http://www.github.com/test/test"  
  },  
  "keywords": [  
    "test"  
  ],  
  "author": "Elias Alonso",  
  "license": "ISC"  
}
```

```
{
  "name": "test",
  "version": "1.0.0",
  "description": "Just a test package",
  "main": "index.js",
  "scripts": {
    "test": "mocha ./tests"
  },
  "repository": {
    "type": "git",
    "url": "http://www.github.com/test/test"
  },
  "keywords": [
    "test"
  ],
  "author": "Elias Alonso",
  "license": "ISC"
}
```

npm

- Los “scripts” son comandos que se pueden ejecutar...
 - `npm run <script>`
 - útil para comandos de complicación, testing, despliegue....
 - `node_modules/.bin` está en el `$PATH`

npm

- Podemos **instalar paquetes** con...
 - `npm install <paquete>`
 - el paquete se convierte en una *dependencia*
 - se añade a `package.json`

```
$ npm install lodash
```

```
{
  "name": "test",
  "version": "1.0.0",
  "description": "Just a test package",
  "main": "index.js",
  "scripts": {
    "test": "mocha ./tests"
  },
  "repository": {
    "type": "git",
    "url": "http://www.github.com/test/test"
  },
  "author": "Elias Alonso",
  "license": "ISC",
  "dependencies": {
    "lodash": "^4.17.10"
  }
}
```



```
$ ls  
node_modules  
package.json  
package-lock.json
```

project

```
|
|├─ node_modules
|├─ package-lock.json
|├─ package.json
|├─ public
|├─ src
|├─   └─ index.js
└─ tests
```

Ejercicio

- Inicializa un proyecto con **npm init**
- Instala los paquetes **lodash** y **nodemon**
- Monta la estructura de directorios sugerida
- Crea un script “*start*” que ejecute **src/index.js**
- Crea un script “*dev*” que ejecute **src/index.js** con **nodemon**

concurrency

Concurrencia

- Node.js ejecuta nuestro código en **una sola hebra**
 - Por tanto, **NO HAY CONCURRENCIA**

```
console.log('empezamos')  
  
for (let i=1e10; i--;)   
    void 0  
  
console.log('terminamos')
```



```
console.log('empezamos')
```

```
for (let i=1e10; i--;)
```

```
    void 0
```

```
console.log('terminamos')
```

```
console.log('empezamos')
```



```
for (let i=1e10; i--:)
```

```
void 0
```

```
console.log('terminamos')
```



```
console.log('empezamos')
```

```
for (let i=1e10; i--;)
  void 0
```

➡ console.log('terminamos')

```
console.log('empezamos')  
  
for (let i=1e10; i--;)   
    void 0  
  
console.log('terminamos')
```



Concurrencia

- Node.js ejecuta nuestro **código** en **una sola hebra**
 - Por tanto, **NO HAY CONCURRENCIA**
- Node.js ejecuta las operaciones de **I/O** en **hebras distintas**
 - El código js **no** es concurrente
 - La **plataforma si es concurrente!**



```
console.log('empezamos')
```

```
setTimeout(() => console.log('cuándo?'), 1000)
```

```
console.log('terminamos')
```

```
console.log('empezamos')
```



```
setTimeout(() => console.log('cuándo?'), 1000)
```

```
console.log('terminamos')
```

```
console.log('empezamos')
```



```
setTimeout(() => console.log('cuándo?'), 1000)
```

```
console.log('terminamos')
```



Cuando hayan pasado 1000ms...

```
console.log('empezamos')
```

➡ setTimeout(() => console.log('cuándo?'), 1000)

```
console.log('terminamos')
```

Cuando hayan pasado 1000ms...

console.log('cuándo?')

```
console.log('empezamos')
```

```
setTimeout(() => console.log('cuándo?'), 1000)
```

➡ console.log('terminamos')

Cuando hayan pasado 1000ms...

```
console.log('cuándo?')
```




```
console.log('empezamos')
```

```
setTimeout(() => console.log('cuándo?'), 1000)
```

```
console.log('terminamos')
```

Cuando hayan pasado 1000ms...



```
console.log('cuándo?')
```



```
console.log('empezamos')
```

```
setTimeout(() => console.log('cuándo?'), 1000)
```

```
console.log('terminamos')
```



Cuando hayan pasado 1000ms...



```
console.log('cuándo?')
```

```
console.log('empezamos')
```

```
setTimeout(() => console.log('cuándo?'), 1000)
```

```
console.log('terminamos')
```



Cuando hayan pasado 1000ms...

```
console.log('cuándo?')
```



➡ console.log('empezamos')
setTimeout(() => console.log('cuánto tarda?'), 1)
for (let i=1e10; i--;) void 0
console.log('terminamos')

```
console.log('empezamos')
```

➡ setTimeout(() => console.log('cuánto tarda?'), 1)

```
for (let i=1e10; i--;) void 0
```

```
console.log('terminamos')
```

```
console.log('empezamos')
```

➡ `setTimeout(() => console.log('cuánto tarda?'), 1)`

```
for (let i=1e10; i--;) void 0
```

```
console.log('terminamos')
```

Cuando haya pasado 1ms...

```
console.log('cuánto tarda?')
```

```
console.log('empezamos')
```

```
setTimeout(() => console.log('cuánto tarda?'), 1)
```



```
for (let i=1e10; i--;) void 0
```

```
console.log('terminamos')
```



Cuando haya pasado 1ms...

```
console.log('cuánto tarda?')
```




```
console.log('empezamos')
```

```
setTimeout(() => console.log('cuánto tarda?'), 1)
```

➡ for (let i=1e10; i--;) void 0

```
console.log('terminamos')
```

Ya ha pasado 1ms!

```
console.log('cuánto tarda?')
```

```
console.log('empezamos')
```

```
setTimeout(() => console.log('cuánto tarda?'), 1)
```

```
for (let i=1e10; i--;) void 0
```

➡ console.log('terminamos')

Ya han pasado muchos ms!

```
console.log('cuánto tarda?')
```

```
console.log('empezamos')  
  
setTimeout(() => console.log('cuánto tarda?'), 1)  
  
for (let i=1e10; i--;) void 0  
  
console.log('terminamos')
```



Ya han pasado muchos ms!



```
console.log('cuánto tarda?')
```

```
console.log('empezamos')  
  
setTimeout(() => console.log('cuánto tarda?'), 1)  
  
for (let i=1e10; i--;) void 0  
  
console.log('terminamos')
```

Ya han pasado muchos ms!

➡ console.log('cuánto tarda?')

```
console.log('empezamos')  
  
setTimeout(() => console.log('cuánto tarda?'), 1)  
  
for (let i=1e10; i--;) void 0  
  
console.log('terminamos')
```

Ya han pasado muchos ms!

```
console.log('cuánto tarda?')
```



Concurrencia


- Node.js ejecuta nuestro código en **una sola hebra**
 - Solo podemos ejecutar **una cosa cada vez**
 - La **plataforma** es **concurrente**
 - Nuestro **código NO**

Concurrencia

- Es una **decisión de diseño**
 - Para **simplificar** la escritura de código
 - La visión: miles de clientes **simultáneos** con código menos propenso a errores y más fácil de mantener
 - Mantener la concurrencia “oculta”
 - Increíblemente eficaz en casos donde **I/O > cpu**



```
console.log('empezamos')  
setTimeout(() => console.log('uno'), 1)  
setTimeout(() => console.log('dos'), 1)  
setTimeout(() => console.log('tres'), 1)  
console.log('terminamos')
```




```
console.log('empezamos')  
setTimeout(() => console.log('uno'), 1)  
setTimeout(() => console.log('dos'), 1)  
setTimeout(() => console.log('tres'), 1)  
console.log('terminamos')
```

Cuando hayan pasado 1ms...



```
console.log('uno')
```




```
console.log('empezamos')
setTimeout(() => console.log('uno'), 1)
setTimeout(() => console.log('dos'), 1)
setTimeout(() => console.log('tres'), 1)
console.log('terminamos')
```

Cuando hayan pasado 1ms...

```
console.log('uno')
```

Cuando hayan pasado 1ms...

```
console.log('dos')
```



```
console.log('empezamos')
setTimeout(() => console.log('uno'), 1)
setTimeout(() => console.log('dos'), 1)
setTimeout(() => console.log('tres'), 1)
console.log('terminamos')
```

Cuando hayan pasado 1ms...


```
console.log('uno')
```

Cuando hayan pasado 1ms...

```
console.log('dos')
```

Cuando hayan pasado 1ms...

```
console.log('tres')
```



```
console.log('empezamos')
setTimeout(() => console.log('uno'), 1)
setTimeout(() => console.log('dos'), 1)
setTimeout(() => console.log('tres'), 1)
console.log('terminamos')
```

Cuando hayan pasado 1ms...

```
console.log('uno')
```

Cuando hayan pasado 1ms...

```
console.log('dos')
```

Cuando hayan pasado 1ms...

```
console.log('tres')
```

```
console.log('empezamos')  
setTimeout(() => console.log('uno'), 1)  
setTimeout(() => console.log('dos'), 1)  
setTimeout(() => console.log('tres'), 1)  
console.log('terminamos')
```

 Cuando hayan pasado 1ms...

```
console.log('uno')
```

Cuando hayan pasado 1ms...

```
console.log('dos')
```

Cuando hayan pasado 1ms...

```
console.log('tres')
```



```
console.log('empezamos')  
setTimeout(() => console.log('uno'), 1)  
setTimeout(() => console.log('dos'), 1)  
setTimeout(() => console.log('tres'), 1)  
console.log('terminamos')
```

Ya ha pasado 1 ms!



```
console.log('uno')
```

Ya ha pasado 1 ms!

```
console.log('dos')
```

Ya ha pasado 1 ms!

```
console.log('tres')
```



```
console.log('empezamos')  
setTimeout(() => console.log('uno'), 1)  
setTimeout(() => console.log('dos'), 1)  
setTimeout(() => console.log('tres'), 1)  
console.log('terminamos')
```

Ya ha pasado 1 ms!

```
console.log('uno')
```

Ya ha pasado 1 ms!

➡ console.log('dos')

Ya ha pasado 1 ms!

```
console.log('tres')
```

```
console.log('empezamos')  
setTimeout(() => console.log('uno'), 1)  
setTimeout(() => console.log('dos'), 1)  
setTimeout(() => console.log('tres'), 1)  
console.log('terminamos')
```

Ya ha pasado 1 ms!

```
console.log('uno')
```

Ya ha pasado 1 ms!

```
console.log('dos')
```

Ya ha pasado 1 ms!

```
console.log('tres')
```

```
console.log('empezamos')  
setTimeout(() => console.log('uno'), 1)  
setTimeout(() => console.log('dos'), 1)  
setTimeout(() => console.log('tres'), 1)  
console.log('terminamos')
```

Ya ha pasado 1 ms!

```
console.log('uno')
```

Ya ha pasado 1 ms!

```
console.log('dos')
```

Ya ha pasado 1 ms!

```
console.log('tres')
```

Concurrencia

- El **patrón** es siempre el mismo:
 - El **código principal** arranca una serie de operaciones
 - Con bloques de código asociados
 - Y **termina!**
 - *El programa se ejecuta de arriba a abajo sin parar!*
 - *Tiene que terminar para que la hebra se quede libre*

Concurrencia

- **NUNCA BLOQUEAR LA HEBRA!**
 - No podemos responder a los eventos asíncronos!
 - El programa se queda “colgado”

Ejercicio

- Escribe una función **tirarDado(ncaras, cb)**
 - Simula el lanzamiento de un dado de n caras
 - Tarda un número aleatorio de **ms** entre 100 y 500
 - Pasa el resultado como primer parámetro a **cb**

Ejercicio

```
tirarDado(6, (result) => {  
  console.log(result) // 2  
})
```

```
tirarDado(10, (result) => {  
  console.log(result) // 5  
})
```

Ejercicio

- Escribe una función **tirarDados(listaCaras, cb)**
 - Recibe un **array** de caras
 - Llama a **tirarDado** con *cada elemento* del array
 - Llama a **cb** con los resultados **de todos los dados!**
 - cuando haya *terminado todas las llamadas*

Ejercicio

```
tirarDados([4, 4, 6, 6], (results) => {  
  console.log(results) // [2, 1, 4, 3]  
})
```

plataforma

Plataforma

- Node.js es una plataforma **back-end**
 - no tenemos **interfaz de usuario**
 - **interacción** con **el sistema**
 - y sus **servicios**
 - escribiendo **scripts** o **servidores**

Plataforma

- Node.js no es sólo el intérprete de js
 - también es **la librería** estándar
 - para interactuar con el **sistema**
 - de una **manera peculiar**
 - para **sacar provecho** de su **modelo de concurrencia**

Plataforma: módulo “fs”

- Interactuar con el **sistema de ficheros**
 - crear, leer, modificar y borrar ficheros
 - listar directorios
 - consultar detalles de un fichero
 - cambiar permisos
 -

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

fs.readdir('.', (err, files) => {
  if (err)
    console.error(err)
  else
    console.log(files)
})

console.log('antes o después?')
```

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

```
fs.readdir('.', (err, files) => {  
  if (err)  
    console.error(err)  
  else  
    console.log(files)  
})
```

```
console.log('antes o después?')
```

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

```
fs.readdir('.', (err, files) => {  
  if (err)  
    console.error(err)  
  else  
    console.log(files)  
}))
```

```
console.log('antes o después?')
```



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

```
fs.readdir('.', (err, files) => {  
  if (err)  
    console.error(err)  
  else  
    console.log(files)  
})
```

```
console.log('antes o después?')
```

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

```
fs.readdir('.', (err, files) => {  
  if (err)  
    console.error(err)  
  else  
    console.log(files)  
})
```

```
console.log('antes o después?')
```

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

```
fs.readdir('.', (err, files) => {  
  if (err)  
    console.error(err)  
  else  
    console.log(files)  
})
```

```
console.log('antes o después?')
```

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

```
fs.readdir('.', (err, files) => {  
  if (err)  
    console.error(err)  
  else  
    console.log(files)  
})
```

```
console.log('antes o después?')
```

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

fs.readdir('.', (err, files) => {
  if (err)
    console.error(err)
  else
    console.log(files)
})
```

```
console.log('antes o después?')
```

Ejercicio

- Escribe un programa que...
 - reciba **un directorio** como parámetro
 - en la invocación por línea de comandos
 - y calcule recursivamente su **tamaño**

Ejercicio

```
$ node index.js ~/Downloads/  
Total: 281.96 Mb
```

Ejercicio

- **process.argv**
 - lista con los componentes de la invocación
- **fs.readdir**
 - lista los elementos de un directorio
- **path.join**
 - concatenar segmentos de una ruta

Ejercicio

- **fs.stat**
 - consultar si una ruta es directorio o fichero
 - consultar el tamaño de un fichero

Ejercicio

- Escribe un programa que...
 - reciba **un directorio** como parámetro
 - en la invocación por línea de comandos
 - busque recursivamente un **substring** en el contenido de los ficheros
 - un clon de grep

Ejercicio

```
$ node index.js "const" ~/code/  
./path1/file.js: const fs = require('fs')  
./path2/index.js: const next = (remaining)  
./path2/index.js: const [head, ...tail]
```

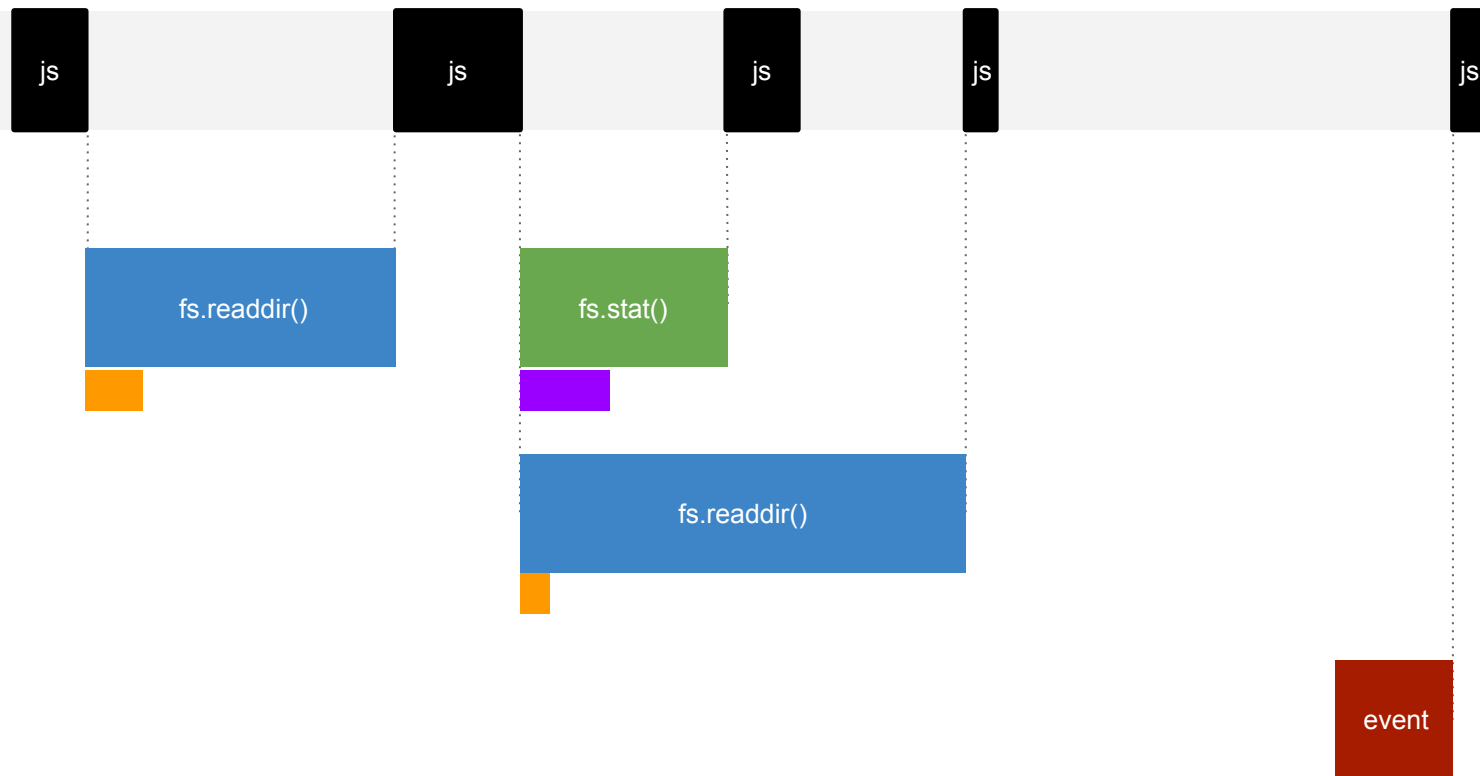
Ejercicio

- **fs.readFile**
 - leer el contenido de un fichero

streams

Streams

- El modelo de operaciones asíncronas que hemos visto...
 - funciona muy bien para gestiones “pequeñas”



Streams

- Este modelo de operaciones asíncronas...
 - funciona muy bien para gestiones “pequeñas”
 - pero es problemático para volúmenes más grandes
 - mucha memoria consumida
 - mucho tiempo de bloque de hebra

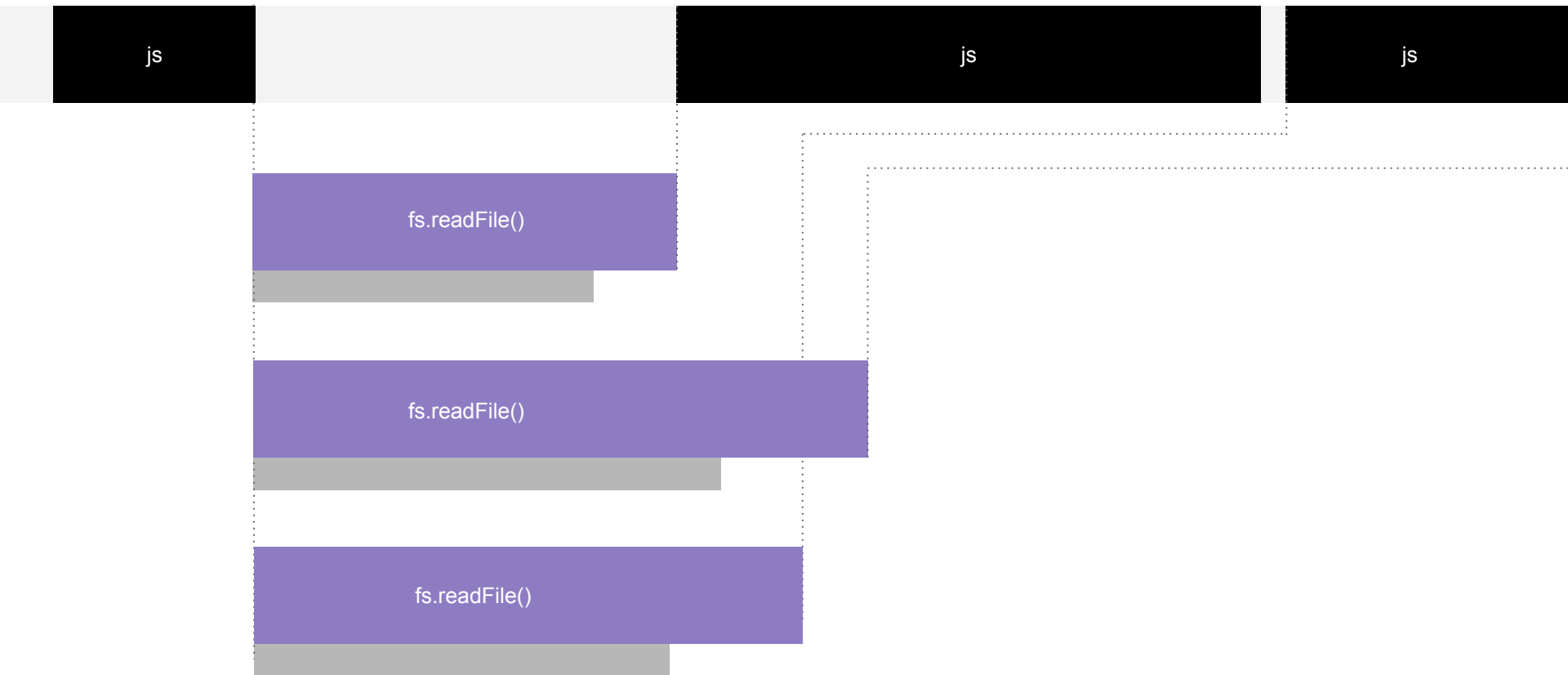
js

fs.readFile()

js

js

fs.readFile()



Streams

- ¿Qué hacemos para procesar un fichero de 200gb?

Streams

- ¿Qué hacemos para procesar un fichero de 200gb?
 - ¡Lo procesamos por partes!
 - Leemos un fragmento y lo procesamos...
 - Una y otra vez
 - Hasta que hayamos procesado el fichero entero

Streams

- **Streams**
 - **Observables** que representan un **flujo** de datos
 - Cuatro tipos:
 - lectura
 - escritura
 - dúplex
 - transformación

Streams

- **Streams**
 - Son **complejos**
 - mecanismo fundamental en node
 - realizan una tarea delicada
 - Interfaces diferentes de **lectura** y **escritura**

Streams de Lectura

- Un flujo de datos “entrantes”
- Procesarlos poco a poco
- Mediante eventos:
 - data
 - end
 - error

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

const fileStream = fs.createReadStream(__filename)

fileStream.on('data', (data) => {
  console.log('read:', data)
})

fileStream.on('end', () => {
  console.log('done!')
})

fileStream.on('error', (err) => {
  console.error(err)
})
```



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

```
const fileStream = fs.createReadStream(__filename)
```

```
fileStream.on('data', (data) => {  
  console.log('read:', data)  
})
```

```
fileStream.on('end', () => {  
  console.log('done!')  
})
```

```
fileStream.on('error', (err) => {  
  console.error(err)  
})
```

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

```
const fileStream = fs.createReadStream(__filename)
```

```
fileStream.on('data', (data) => {  
  console.log('read:', data)  
})
```

```
fileStream.on('end', () => {  
  console.log('done!')  
})
```

```
fileStream.on('error', (err) => {  
  console.error(err)  
})
```

```
$ node test.js
```

```
read: <Buffer 63 6f 6e 73 74 20 66 73 20 3d 20 72 65  
71 75 69 72 65 28 27 66 73 27 29 0a 0a 63 6f 6e 73  
74 20 66 69 6c 65 53 74 72 65 61 6d 20 3d 20 66 73  
2e 63 72 ... >  
done!
```

```
$ node test.js
```

```
read: <Buffer 63 6f 6e 73 74 20 66 73 20 3d 20 72 65  
71 75 69 72 65 28 27 66 73 27 29 0a 0a 63 6f 6e 73  
74 20 66 69 6c 65 53 74 72 65 61 6d 20 3d 20 66 73  
2e 63 72 ... >
```

```
done!
```

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

const fileStream = fs.createReadStream(__filename)

fileStream.on('data', (data) => {
  console.log('read:', data.toString())
})

fileStream.on('end', () => {
  console.log('done!')
})

fileStream.on('error', (err) => {
  console.error(err)
})
```

Streams de Escritura

- Un flujo de datos “salientes”
- Escribir data poco a poco
- Dos métodos esenciales:
 - `write(chunk, [encoding, callback])`
 - `end([chunk, encoding, callback])`

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

const fileStream = fs.createReadStream(__filename)
const outputStream = fs.createWriteStream('/tmp/copy.out')

fileStream.on('data', (data) => {
  outputStream.write(data)
})

fileStream.on('end', () => {
  outputStream.end()
})

fileStream.on('error', (err) => {
  console.error(err)
})
```

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

```
const fileStream = fs.createReadStream(__filename)
```

```
const outputStream = fs.createWriteStream('/tmp/copy.out')
```

```
fileStream.on('data', (data) => {  
  outputStream.write(data)  
}))
```

```
fileStream.on('end', () => {  
  outputStream.end()  
}))
```

```
fileStream.on('error', (err) => {  
  console.error(err)  
}))
```



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

```
const fileStream = fs.createReadStream(__filename)
```

```
const outputStream = fs.createWriteStream('/tmp/copy.out')
```

```
fileStream.on('data', (data) => {  
  outputStream.write(data)  
})
```

```
fileStream.on('end', () => {  
  outputStream.end()  
})
```

```
fileStream.on('error', (err) => {  
  console.error(err)  
})
```

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

const fileStream = fs.createReadStream(__filename)
const outputStream = fs.createWriteStream('/tmp/copy.out')

fileStream.pipe(outputStream)
```

Ejercicio

- reescribe el clon de grep...
 - utilizando **streams**
 - para que funcione con ficheros de cualquier tamaño

http

Http

- `http.createServer(requestListener)`
 - crea una instancia de `http.Server`
 - *requestListener* se invoca **en cada petición** con dos parámetros:
 - **req**: stream de lectura con info sobre la petición
 - **res**: stream de escritura para enviar la respuesta

```
const http = require('http')

const server = http.createServer((req, res) => {
  console.log('> conexión!')
  res.write('Hola, Mundo!')
  res.end()
})

server.listen(3000)
console.log('* Ready http://localhost:3000')
```

```
const http = require('http')
```

```
const server = http.createServer((req, res) => {  
  console.log('> conexión!')  
  res.write('Hola, Mundo!')  
  res.end()  
})
```

```
server.listen(3000)  
console.log('* Ready http://localhost:3000')
```

```
const http = require('http')
```

```
const server = http.createServer((req, res) => {  
  console.log('> conexión!')  
  res.write('Hola, Mundo!')  
  res.end()  
})
```

```
server.listen(3000)  
console.log('* Ready http://localhost:3000')
```



```
const http = require('http')

const server = http.createServer((req, res) => {
  console.log('> conexión!')
  res.write('Hola, Mundo!')
  res.end()
})

server.listen(3000)
console.log('* Ready http://localhost:3000')
```

```
const http = require('http')

const server = http.createServer((req, res) => {
  console.log('> conexión!')
  res.write('Hola, Mundo!')
  res.end()
})

server.listen(3000)
console.log('* Ready http://localhost:3000')
```

```
const http = require('http')

const server = http.createServer((req, res) => {
  console.log('> conexión!')
  res.write('Hola, Mundo!')
  res.end()
})
```

```
server.listen(3000)
console.log('* Ready http://localhost:3000')
```

Ejercicio

- Escribe un **programa** que sirva **su propio código** por **HTTP**
 - <http://localhost:3000/>
 - ruta del script en **__filename**

Http

- `req.url`
 - dirección solicitada por el cliente
- `req.headers`
 - cabeceras proporcionadas por el cliente
- `req` leído como *stream*
 - cuerpo de la petición (POST)

Http

- `res.writeHead(statusCode, headerObject)`
 - status code y las cabeceras de la respuesta
- `res.statusCode`
 - asignar un número para especificar un status code *sin especificar ninguna cabecera*

```
const http = require('http')
const { parse } = require('url')

const server = http.createServer((req, res) => {
  const url = parse(req.url, true)
  const responseStr = JSON.stringify(url, null, 4)
  res.writeHead(200, {
    'Content-Type': 'text/plain',
    'Content-Length': Buffer.byteLength(responseStr)
  })
  res.end(responseStr)
})

server.listen(3000)
console.log('* Ready http://localhost:3000')
```

```
const http = require('http')
const { parse } = require('url')
```

```
const server = http.createServer((req, res) => {
  const url = parse(req.url, true)
  const responseStr = JSON.stringify(url, null, 4)
  res.writeHead(200, {
    'Content-Type': 'text/plain',
    'Content-Length': Buffer.byteLength(responseStr)
  })
  res.end(responseStr)
})
```

```
server.listen(3000)
console.log('* Ready http://localhost:3000')
```



```
const http = require('http')
const { parse } = require('url')

const server = http.createServer((req, res) => {
  const url = parse(req.url, true)
  const responseStr = JSON.stringify(url, null, 4)
  res.writeHead(200, {
    'Content-Type': 'text/plain',
    'Content-Length': Buffer.byteLength(responseStr)
  })
  res.end(responseStr)
})

server.listen(3000)
console.log('* Ready http://localhost:3000')
```

```
const http = require('http')
const { parse } = require('url')

const server = http.createServer((req, res) => {
  const url = parse(req.url, true)
  const responseStr = JSON.stringify(url, null, 4)
  res.writeHead(200, {
    'Content-Type': 'text/plain',
    'Content-Length': Buffer.byteLength(responseStr)
  })
  res.end(responseStr)
})

server.listen(3000)
console.log('* Ready http://localhost:3000')
```

```
const http = require('http')
const { parse } = require('url')

const server = http.createServer((req, res) => {
  const url = parse(req.url, true)
  const responseStr = JSON.stringify(url, null, 4)
  res.writeHead(200, {
    'Content-Type': 'text/plain',
    'Content-Length': Buffer.byteLength(responseStr)
  })
  res.end(responseStr)
})

server.listen(3000)
console.log('* Ready http://localhost:3000')
```

Ejercicio

- Escribe una **aplicación web** para “acortar” enlaces con tres “rutas”:
 - un formulario para introducir el target
 - una página de confirmación con el link creado
 - una ruta de redirección que recibe el token como parámetro

mysql

MySQL

- Node.js **no** trae soporte para bases de datos...
 - ni para ningún otro servicio que no sea general del SO
- Interfaz C++ para integrar **librerías externas**
 - extender la plataforma
 - transparente para nosotros
 - simplemente instalamos el módulo adecuado

MySQL

- Los paquetes que integran librerías externas...
 - no siempre siguen la misma filosofía de diseño
 - no siempre siguen el interfaz `callback(err, data...)`
 - no siempre son estables o están bien mantenidos
 - a veces requieren un **compilador de c++**
 - no siempre están bien **documentados**

MySQL

- El paquete **mysql**
 - 270k descargas semanales
 - 12k estrellas en github
 - múltiples commits en el último mes
 - documentación adecuada
 - interfaz bastante fiel al estilo de Node.js


```
$ npm install mysql
```

```
CREATE DATABASE test;  
USE test;
```

```
CREATE TABLE users (  
    id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,  
    name VARCHAR(256),  
    email VARCHAR(256),  
    password VARCHAR(256)  
);
```

```
INSERT INTO USERS VALUES (  
    null, 'Elias', 'eliasagc@gmail.com', 'supersecret'  
);
```

```
const mysql = require('mysql')
const connection = mysql.createConnection({
  host: 'localhost',
  user: 'root',
  database: 'test'
})

connection.connect();

connection.query('SELECT * FROM users', (err, results, fields) => {
  if (err)
    console.error(err)
  else
    console.log(results)
})

connection.end()
```

```
const mysql = require('mysql')
```

```
const connection = mysql.createConnection({  
  host: 'localhost',  
  user: 'root',  
  database: 'test'  
})
```

```
connection.connect();
```

```
connection.query('SELECT * FROM users', (err, results, fields) => {  
  if (err)  
    console.error(err)  
  else  
    console.log(results)  
})
```

```
connection.end()
```

```
const mysql = require('mysql')
const connection = mysql.createConnection({
  host: 'localhost',
  user: 'root',
  database: 'test'
})
```

```
connection.connect();
```

```
connection.query('SELECT * FROM users', (err, results, fields) => {
  if (err)
    console.error(err)
  else
    console.log(results)
})
```

```
connection.end()
```

```
const mysql = require('mysql')
const connection = mysql.createConnection({
  host: 'localhost',
  user: 'root',
  database: 'test'
})
```

```
connection.connect();
```

```
connection.query('SELECT * FROM users', (err, results, fields) => {
  if (err)
    console.error(err)
  else
    console.log(results)
})
```

```
connection.end()
```

```
const mysql = require('mysql')
const connection = mysql.createConnection({
  host: 'localhost',
  user: 'root',
  database: 'test'
})
```

```
connection.connect();
```

```
connection.query('SELECT * FROM users' (err, results, fields) => {
  if (err)
    console.error(err)
  else
    console.log(results)
})
```

```
connection.end()
```

```
const mysql = require('mysql')
const connection = mysql.createConnection({
  host: 'localhost',
  user: 'root',
  database: 'test'
})

connection.connect();

connection.query('SELECT * FROM users', (err, results, fields) => {
  if (err)
    console.error(err)
  else
    console.log(results)
})

connection.end()
```



```
const mysql = require('mysql')
const connection = mysql.createConnection({
  host: 'localhost',
  user: 'root',
  database: 'test'
})

connection.connect();

connection.query('SELECT * FROM users', (err, results, fields) => {
  if (err)
    console.error(err)
  else
    console.log(results)
})

connection.end()
```

```
connection.query(  
  'INSERT INTO users VALUES (null, "Homer", "h.simpson@fox.com", "rosquilla")',  
  (err, result) => console.log('Inserted row ID:', result.insertId)  
)
```

```
connection.query(  
  'UPDATE users SET email = "homer@sprngfld.usa" WHERE name = "Homer"',  
  (err, result) => console.log('Changed rows:', result.changedRows)  
)
```

```
connection.query(  
  'DELETE FROM users WHERE name = "Homer"',  
  (err, result) => console.log('Affected rows:', result.affectedRows)  
)
```

```
connection.query(  
  'INSERT INTO users VALUES (null, "Homer", "h.simpson@fox.com", "rosquilla")',  
  (err, result) => console.log('Inserted row ID:', result.insertId)  
)
```

```
connection.query(  
  'UPDATE users SET email = "homer@sprngfld.usa" WHERE name = "Homer"',  
  (err, result) => console.log('Changed rows:', result.changedRows)  
)
```

```
connection.query(  
  'DELETE FROM users WHERE name = "Homer"',  
  (err, result) => console.log('Affected rows:', result.affectedRows)  
)
```

MySQL

- Seguridad básica: escapa todo lo que venga del usuario!
 - **mysql.escape(*anyValue*)**
 - placeholders en **mysql.query(...)**

```
connection.query(  
    'SELECT * FROM users WHERE email = ?',  
    ['eliasagc@gmail.com'],  
    (err, results, fields) => {  
        err ? console.error(err) : console.log(results)  
    })  
)
```

```
connection.query(  
  'SELECT * FROM users WHERE email = ?',  
  ['eliasagc@gmail.com'],  
  (err, results, fields) => {  
    err ? console.error(err) : console.log(results)  
  })  
)
```

```
const data = {  
  name: 'Homer',  
  email: 'h.simpson@fox.com',  
  password: 'rosquilla'  
}
```

```
connection.query('INSERT INTO users SET ?', data)
```

MySQL

- Para trabajar con gran volumen de datos...
 - podemos convertir el resultado en un *stream*
 - que se trae los resultados poco a poco
 - para poder procesarlos según llegan


```
const readStream = connection
    .query('SELECT * FROM users')
    .stream()

readStream.on('data', row => console.log('row:', row))
readStream.on('end', () => console.log('done!'))
```

MySQL

- La librería tiene funcionalidad más avanzada
 - pool de conexiones
 - transacciones
 - procedimientos almacenados
 - gestión de usuarios
 - etc....

Ejercicio

- Modifica el acortador de urls para que guarde los enlaces en **mysql**
 - y que cuente cuantas veces se visita cada enlace