

INTRODUCCIÓN A NODE.JS



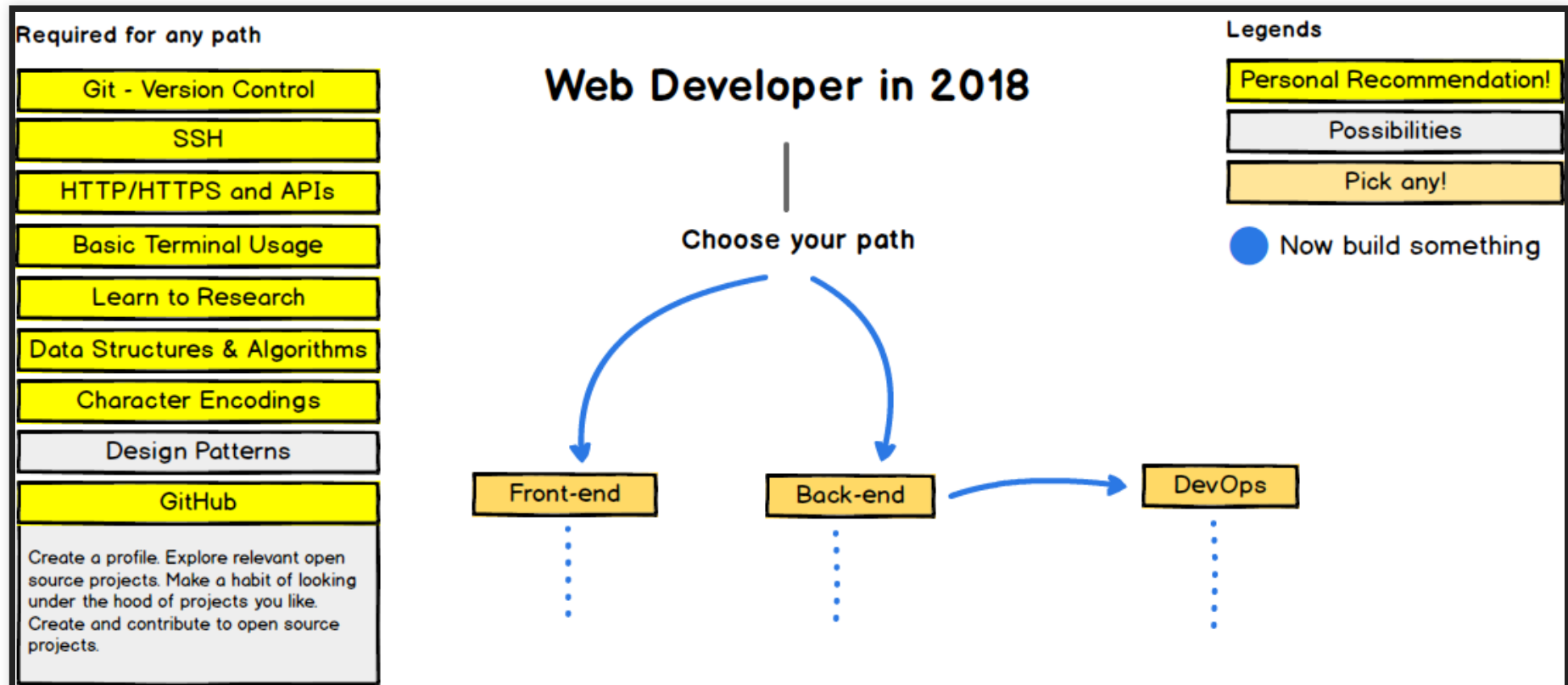
QUÉ ES NODE.JS

- [Sitio web de node.js](https://nodejs.org/):

Node.js[®] is a JavaScript runtime built on Chrome's V8 JavaScript engine.

- Node.js es un intérprete de JavaScript que se ejecuta en servidor (sin navegador).

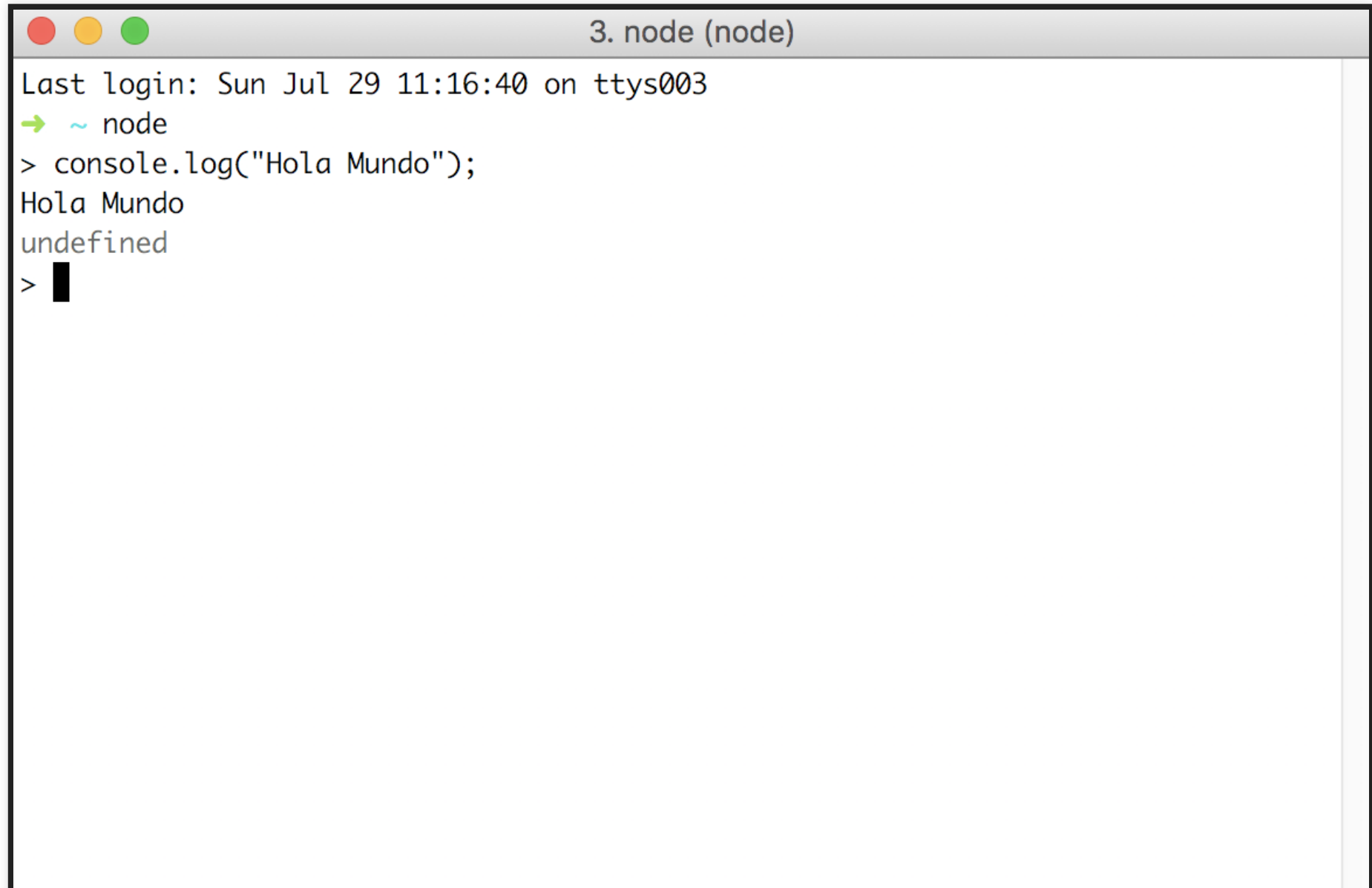
SITUACIÓN ACTUAL DEL DESARROLLO WEB



GOOGLE CHROME V8

- Es el motor de JavaScript que utiliza Google Chrome y node
 - Escrito en C++
 - [Motor de código abierto](#)
 - Compila a código máquina

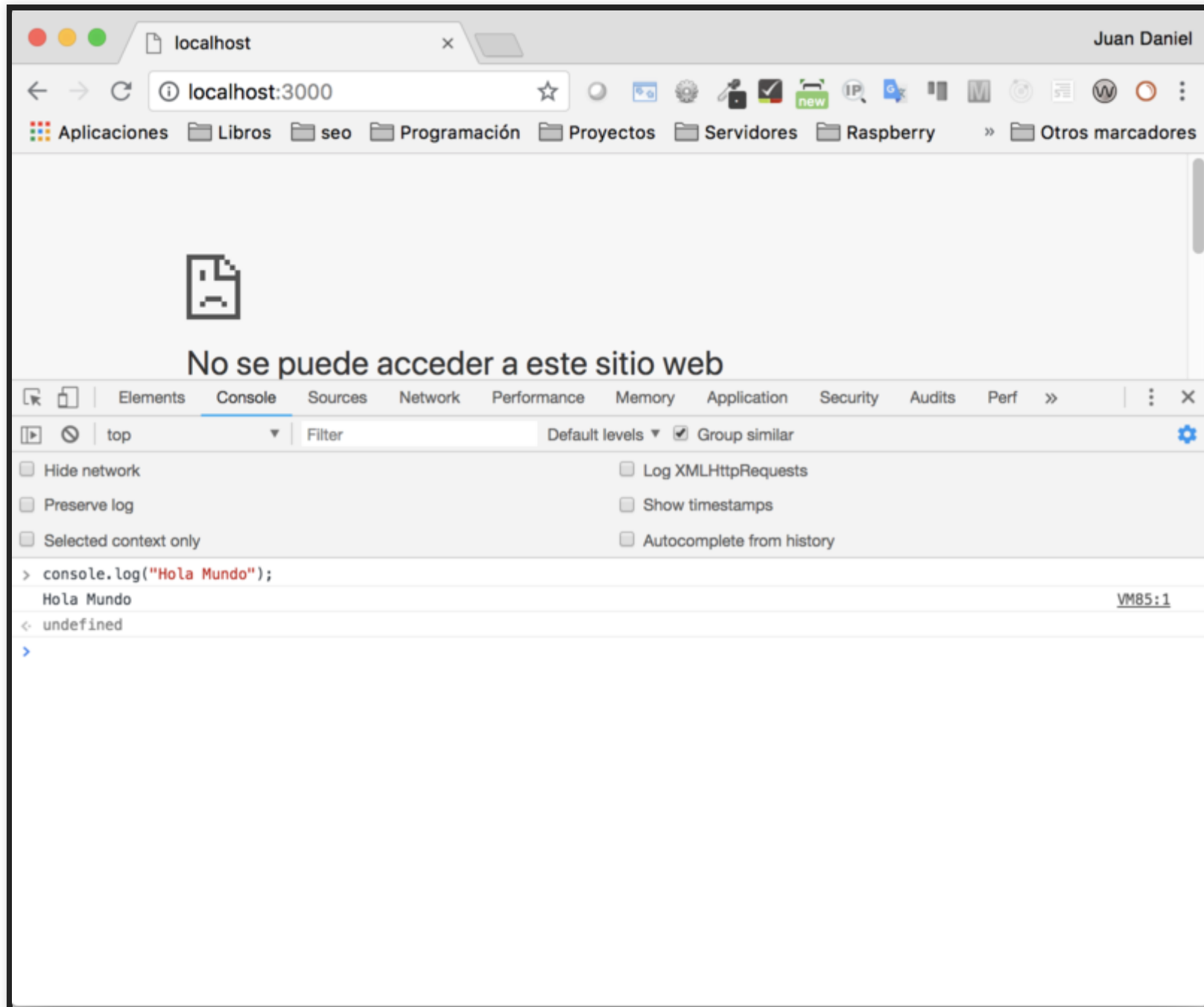
HOLA MUNDO EN NODE

A terminal window with a title bar containing three colored circles (red, yellow, green) and the text "3. node (node)". The terminal content shows a login message, a prompt to run a Node.js command, the execution of the command, and the output of the command.

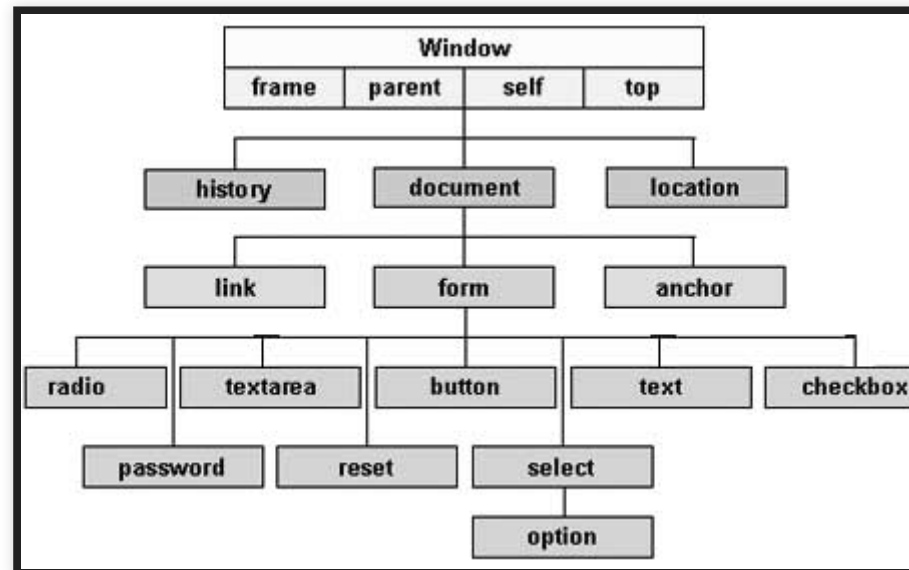
```
Last login: Sun Jul 29 11:16:40 on ttys003
➔ ~ node
> console.log("Hola Mundo");
Hola Mundo
undefined
> █
```



HOLA MUNDO EN BROWSER




HTML DOM



OBJETO WINDOW EN JAVASCRIPT

OBJETO "WINDOW" EN NODE

- El objeto global en Node, se llama global:

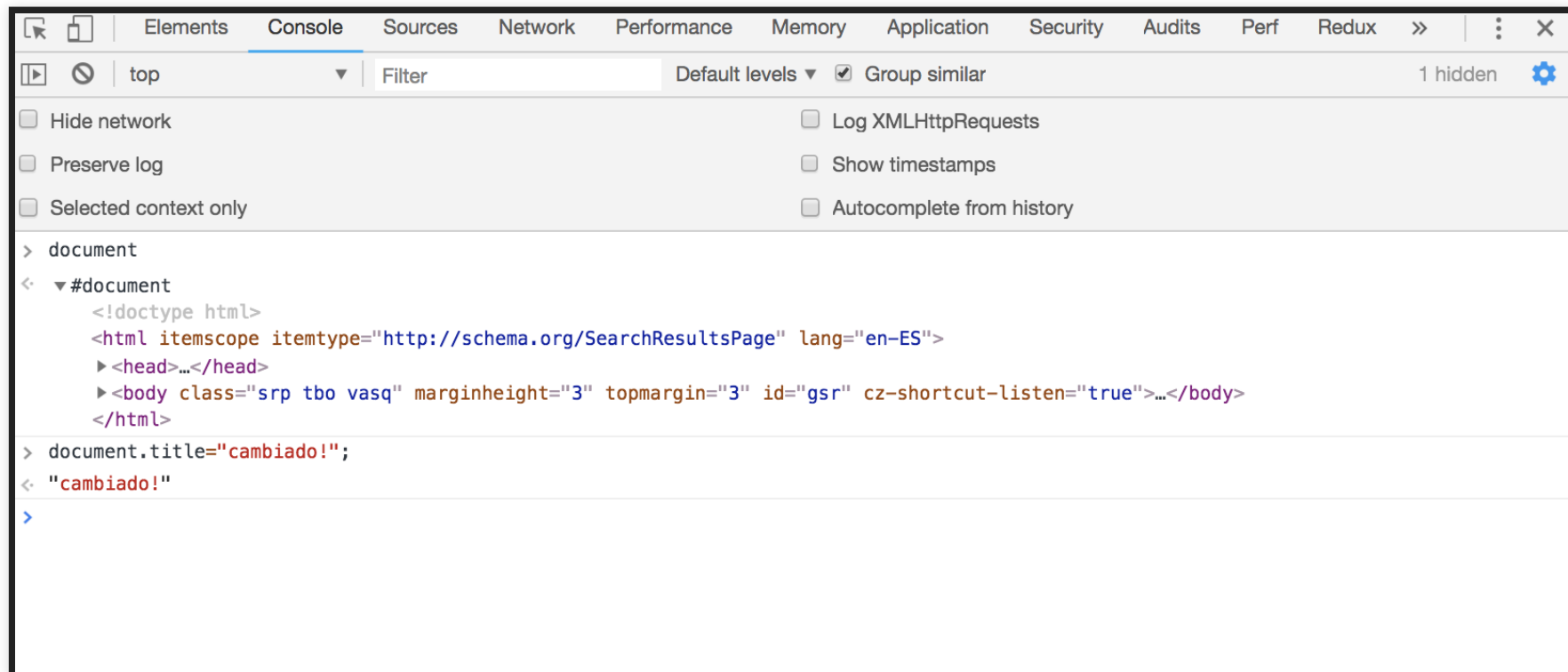


```
2. node (node)
→ ~ node
> global
{ console: [Getter],
  DTRACE_NET_SERVER_CONNECTION: [Function],
  DTRACE_NET_STREAM_END: [Function],
  DTRACE_HTTP_SERVER_REQUEST: [Function],
  DTRACE_HTTP_SERVER_RESPONSE: [Function],
  DTRACE_HTTP_CLIENT_REQUEST: [Function],
  DTRACE_HTTP_CLIENT_RESPONSE: [Function],
  global: [Circular],
  process:
    process {
      title: 'node',
      version: 'v8.11.3',
      moduleLoadList:
        [ 'Binding contextify',
          'Binding natives',
          'Binding config',
```

```
'NativeModule events',  
'Binding async_wrap',  
'Binding icu',  
'NativeModule util',  
'NativeModule internal/errors',  
'NativeModule internal/encoding',  
'NativeModule internal/util',
```

OBJETO DOCUMENT EN JAVASCRIPT

- Lo que se está viendo en el navegador
- Podemos modificarlo en runtime



The screenshot shows a web browser's developer console with the 'Console' tab selected. The console displays the following content:

```
> document
< #document
  <!doctype html>
  <html itemscope itemtype="http://schema.org/SearchResultsPage" lang="en-ES">
    <head>...</head>
    <body class="srp tbo vasq" marginheight="3" topmargin="3" id="gsr" cz-shortcut-listen="true">...</body>
  </html>
> document.title="cambiado!";
< "cambiado!"
>
```

The console also shows a tree view of the document object, with the following structure:

```
> document
< #document
  <!doctype html>
  <html itemscope itemtype="http://schema.org/SearchResultsPage" lang="en-ES">
    <head>...</head>
    <body class="srp tbo vasq" marginheight="3" topmargin="3" id="gsr" cz-shortcut-listen="true">...</body>
  </html>
```


OBJETO PROCESS EN NODE

```
$ node  
> process  
...  
> process.exit(3) // también podríamos hacer .exit(3), o CTRL  
$ echo $?  
3
```

- process (node) = document (JavaScript)
- Vemos la información del proceso node que se está ejecutando:
 - pid
 - versión de node
 - métodos
 - ...

NODE VS JAVASCRIPT

- El código en ambos es JavaScript
- Ambos se ejecutan con el mismo motor (si usamos Chrome)
- Node utiliza el motor fuera del contexto del navegador
 - No hay un *browser sandbox*
 - Tenemos funcionalidad añadida:
 - Acceso al file system
 - Acceso a bbdd *completa*
 - Incluso montar un servidor web

FUNCIONAMIENTO CÓDIGO

- Se lee el código en JavaScript
- Se compila a código máquina por el V8 y se ejecuta:

V8 compiles JavaScript directly to native machine code before executing it, instead of more traditional techniques such as interpreting bytecode or compiling the whole program to machine code and executing it from a filesystem. The compiled code is additionally optimized (and re-optimized) dynamically at runtime, based on heuristics of the code's execution profile. ([wikipedia](#))

CARACTERÍSTICAS DE NODE

- **Mismo lenguaje en cliente y servidor**
 - Permite a cualquier persona desarrollar en backend o en frontend
 - Permite reusar código o incluso mover código de cliente a servidor o al revés
- **No bloqueante (asíncrono) por naturaleza**
 - Los métodos síncronos, llevan el sufijo sync

- El mayor repositorio de código disponible: [npm](#)
 - composer/php o jpm/java) están basados en npm
- Orientado a eventos
- Es monohilo
 - Utiliza un solo procesador
 - Si queremos usar toda la potencia de la CPU, tendremos que levantar varias instancias de node y utilizar un balanceador de carga ([por ejemplo con pm2](#))

- Llamadas síncronas en servidor serían fatales:
 - ¡Bloquearíamos las conexiones al servidor hasta que acabase la instrucción bloqueante!
 - Al ser asíncrono podremos tener muchas sesiones concurrentes

EJEMPLO CÓDIGO NO BLOQUEANTE

```
const fs = require('fs')
fs.readFile('./prueba.txt', 'utf-8', (err, data) => {
  if (err) throw err
  console.log(`El contenido del fichero es este: ${data}`)
})
console.log(`Aquí todavía no tenemos el valor de fs.readFile`)
```

EJEMPLO CÓDIGO BLOQUEANTE

```
const fs = require('fs');  
const data = fs.readFileSync('./prueba.txt', 'utf-8');  
console.log(`El contenido del fichero es este: ${data}`)
```


CONSULTA DE API DE NODE

- El módulo fs pertenece a los core modules de node, no es necesario instalarlo
- Para consultar la API:
 - Desde el navegador
 - Desde el terminal (plugin node de zsh)

```
$ node-docs
```

BLOQUEANTE VS NO BLOQUEANTE

- El código asíncrono tiene un **throughput** mucho mayor.
- Se puede volver **complejo** el trabajar con el resultado de una función asíncrona.
 - El código asíncrono no se ejecuta de forma secuencial, más difícil de seguir
 - El método asíncrono recibe como último parámetro una **función de callback**

¿QUÉ SERÍA LO IDEAL?

- Utilizar **código secuencial y asíncrono**
 - Para ello utilizaremos **promesas y async/await**
- Evitaremos la anidación de funciones de callback, conocido como **callback hell**

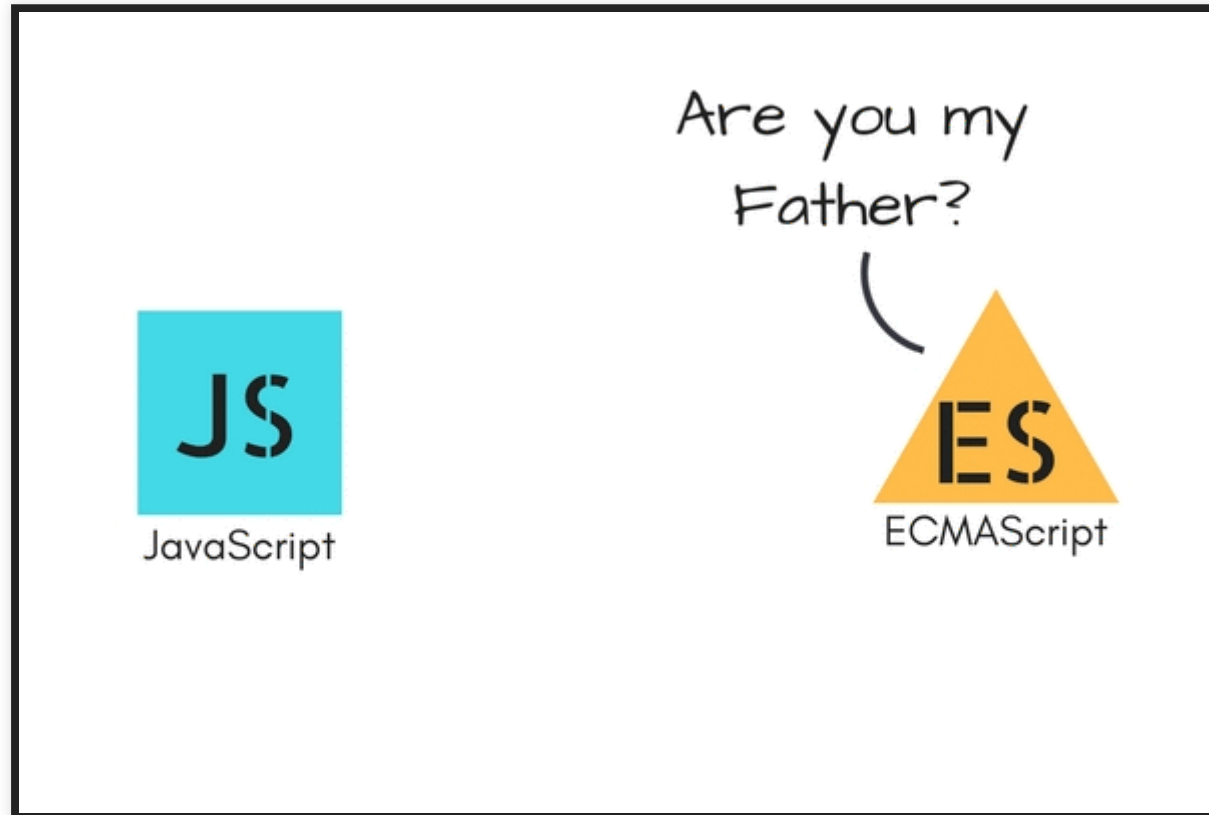
VERSIONES DE JS

ECMAScript

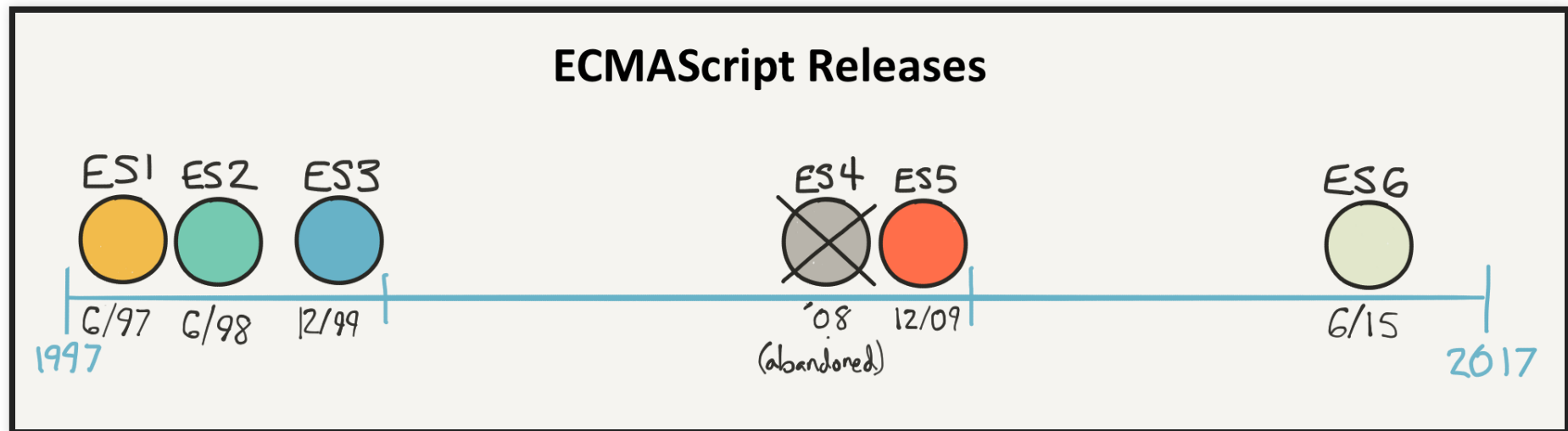
- Estándarización del JavaScript de NetScape (1997)
- Implementaciones de ECMAScript:
 - JavaScript
 - JScript (Microsoft)
 - ActionScript (Adobe)
 - ..

TC39

- Comité técnico encargado de la especificación de ECMAScript
 - Yahoo
 - Paypal
 - Google
 - Microsoft
 -



JAVASCRIPT TIMELINE



ECMAScript 4

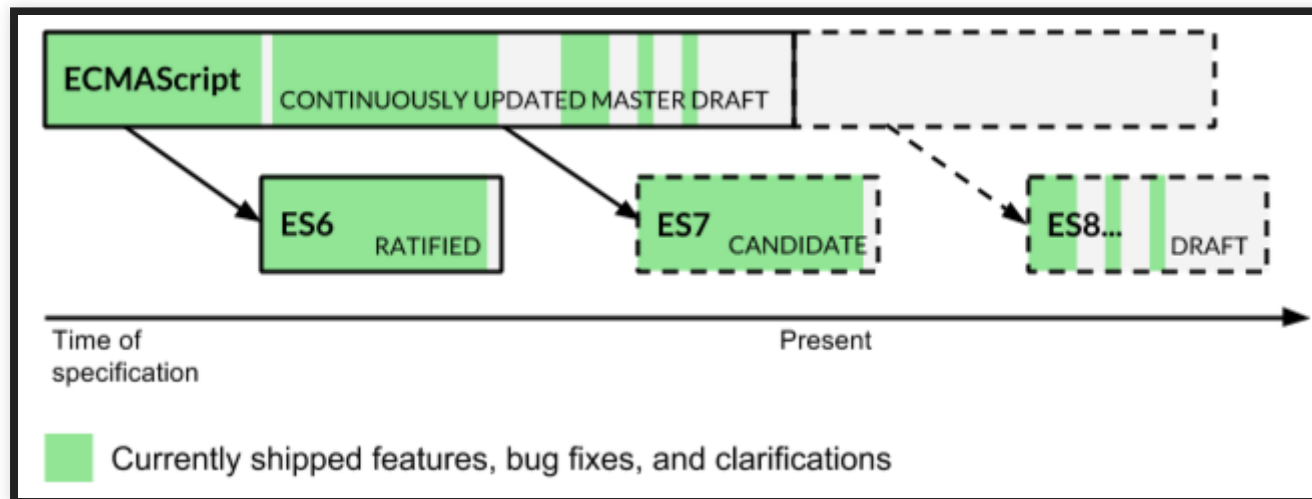
- A favor de grandes cambios:
 - Microsoft (Silverlight con C#)
 - Adobe (Adobe air con ActionScript)
- A favor de mantener compatibilidad con lo anterior:
 - Google
 - Yahoo

ECMAScript 5

- No tiene tantas grandes novedades
- jQuery nace en el 2006 para paliar las diferencias entre navegadores

ECMAScript 6

- Grandes novedades
- Se pueden usar **transpilers**



VERSIONES ACTUALES

- ES6: Jun 2015
 - El comite decidió publicar especificaciones ECMAScript de forma anual
 - Se renombro a **ES2015**
- **ES2016** (ES7)
- **ES2017** (ES8)
- **ES.Next**: Término dinámico, se refiere a la próxima versión de ECMAScript.

¿QUÉ PUEDO USAR?

- En Web tenemos que vivir con la fragmentación
 - V8 - Google Chrome (Chromium, MongoDB)
 - SpiderMonkey Firefox (GNOME, Adobe).
 - Chakra - Microsoft IE y Edge
- En Node es más sencillo porque:
 - Solo hay un motor
 - Nosotros elegimos su versión

TABLAS DE COMPATIBILIDADES

- Node:
 - <http://kangax.github.io/compat-table/es5/>
 - <https://node.green/>
- Web:
 - <https://caniuse.com>

¿EMPEZAMOS EL CÓDIGO?

- Primer proyecto