Curso de Back-End con Node.js (Inicial)

Clase 05

Temario

Temario

- Control de Versiones.
- Git.
  - Secciones de un proyecto.
  - Branches.
  - Comandos básicos.
  - Interfaces gráficas.
- Repositorios remotos en la nube.
- Middlewares.

Control de Versiones

Control de versiones (1/5)

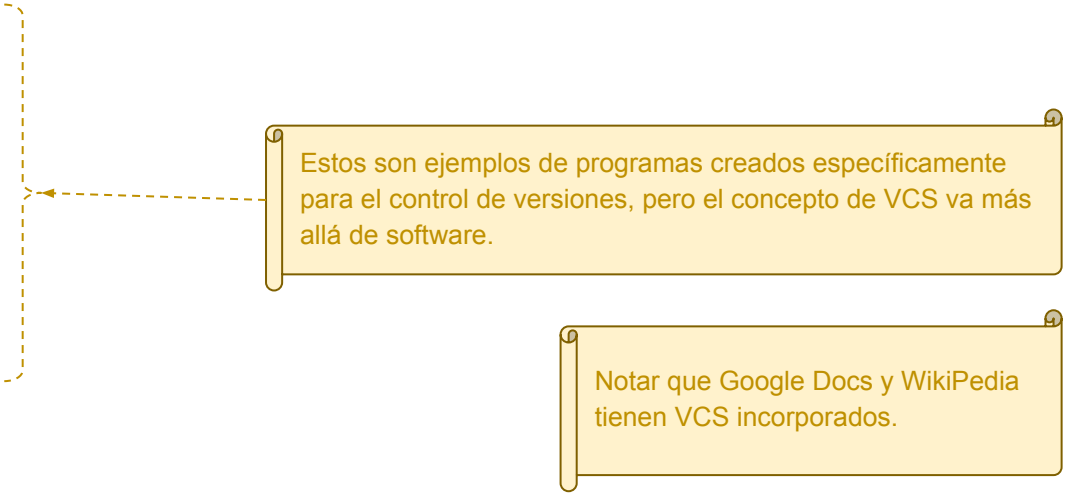
Preguntas que surgen en un proyecto:

- ¿Cuándo se modificó el archivo X la última vez?
- ¿Quién modificó el archivo X el lunes pasado?
- ¿Qué líneas de código se modificaron en el archivo X el lunes pasado?
- ¿El código con el que estoy trabajando es el más nuevo?
- ¿Qué se hace cuando dos desarrolladores modifican el mismo archivo?
- ¿Cómo es posible trabajar en varias versiones paralelas del proyecto?
- ¿Cómo se mantiene el código respaldado?
- Etc.

Control de versiones (2/5)

Para solucionar los problemas anteriores surgen los sistemas de control de versiones (en inglés: VCS). Los más conocidos son:

- Git.
- Subversion (SVN).
- Mercurial.
- Team Foundation Server.



Control de versiones (3/5)

Uno de los conceptos más importantes de los VCS es el de repositorio:

El repositorio es el lugar en el que se almacenan los archivos de los cuales se quieren controlar sus versiones.

Puede ser en un disco duro, en una base de datos, etc..

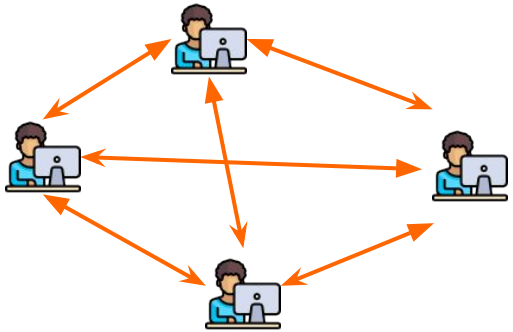
Control de versiones (4/5)

Básicamente, los sistemas de control de versiones se clasifican en:

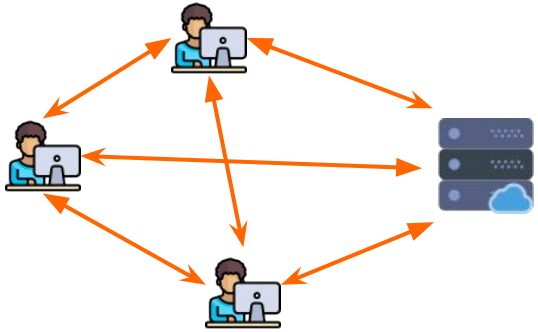
- Distribuidos:
  - Cada usuario tiene su propio repositorio local, por lo cual no se precisa conectividad.
  - Los distintos repositorios pueden intercambiar y mezclar revisiones entre ellos.
  - Es frecuente el uso de un repositorio central (en un servidor) que sirve como punto de sincronización de los distintos repositorios locales. Aquí reside el código "oficial" del proyecto.
- Centralizados:
  - Existe un repositorio centralizado de todo el código, del cual es responsable un único usuario (o conjunto de ellos).
  - Se facilitan las tareas administrativas a cambio de reducir flexibilidad, pues todas las decisiones fuertes (como crear una nueva rama) necesitan la aprobación del responsable.

Control de versiones (5/5)

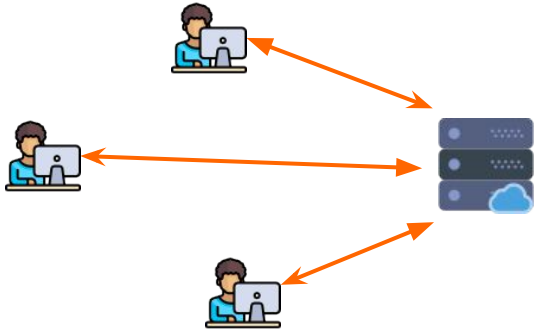
En sistemas distribuidos, cada usuario tiene su propio repositorio, pero lo usual es que una de las máquinas sea un servidor donde el código esté disponible para todo el equipo. De esta forma se emula un sistema centralizado, pero se mantienen las ventajas de los sistemas distribuidos.



Esquema distribuido tradicional (teórico)



Esquema distribuido con repositorio en servidor central



Esquema distribuido con repositorio en servidor central (común en la práctica)

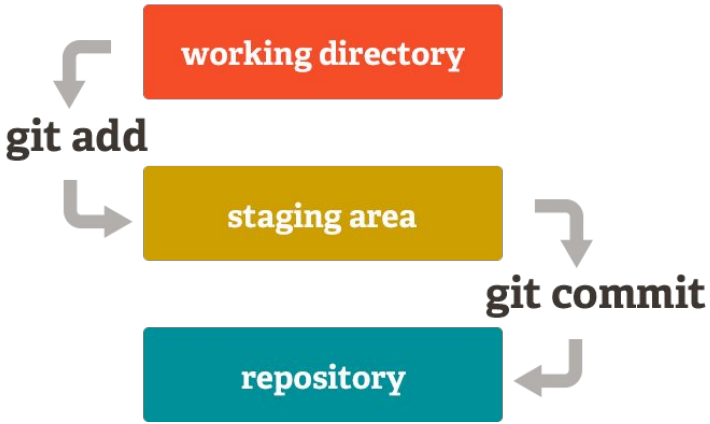
Git

Git

- Es el sistema de control de versiones más utilizado, por lejos.
- Fue creado por Linus Torvalds en 2005 (el creador de Linux).
- Es open source.
- Es distribuido.
- Fue creado pensando en eficiencia, seguridad y flexibilidad.
- Documentación: https://git-scm.com/doc.
- Para practicar: https://try.github.io.

Git – Secciones de un proyecto

En Git, cada archivo se encuentra en alguno de estos estados: `committed`, `modified` o `staged`. Esto conlleva a que existan tres secciones en un proyecto: el directorio de trabajo que contiene los archivos, el staging area (también llamado `Index`) que actúa como una zona intermedia y el repositorio propiamente dicho que apunta al último commit realizado (también llamado `HEAD`).



El *staging area* permite agrupar modificaciones en un mismo `commit`.

Analogía con un fotógrafo:

Stage es preparar la foto, seleccionar qué personas van en la foto y colocar a cada una en su lugar.

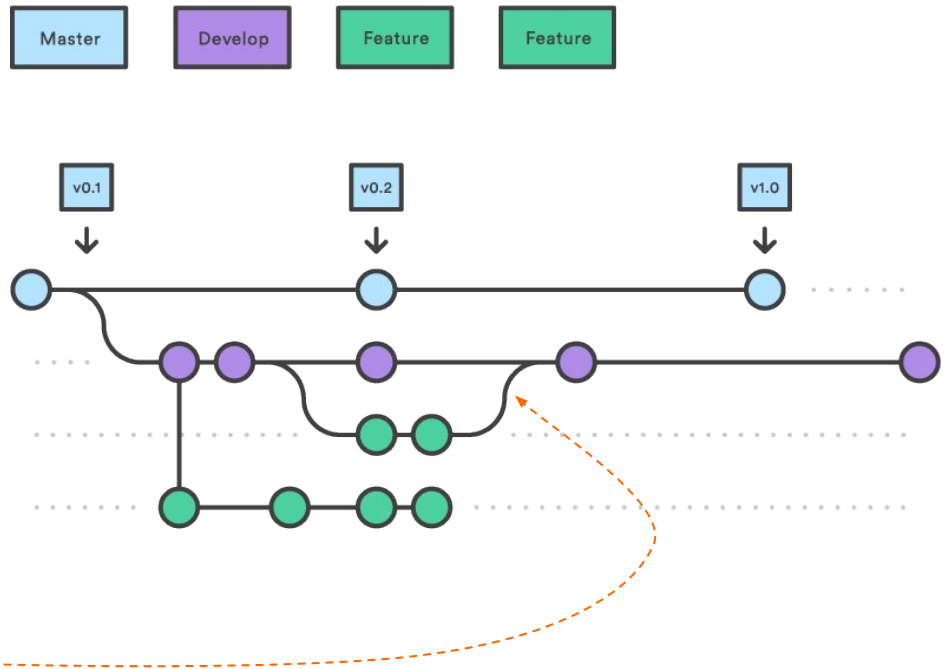
Commit es sacar la foto (snapshot).

Git – Branches

Una de las ventajas de Git es su capacidad para realizar branches.

Cada repositorio cuenta con una branch principal llamada `master` o `main`. donde se encuentra el código para producción.

Cada desarrollador puede crear branches paralelas donde desarrollar nuevas funcionalidades. Luego es posible hacer `merge` (combinar) branches.



Git – Comandos Básicos

Crear un proyecto nuevo (crea un directorio para el repositorio):

```
git init mi-proyecto
```

Colocar todos los archivos modificados a un zona de *staging*:

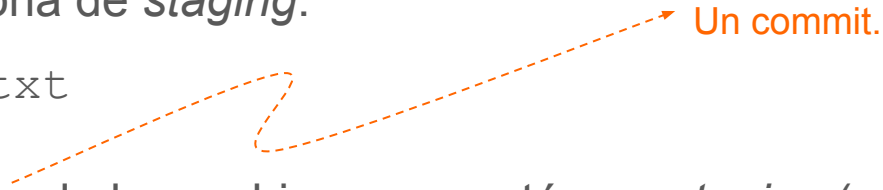
```
git add .
```

Colocar un archivo en la zona de *staging*:

```
git add archivo.txt
```

Grabar una versión histórica de los archivos que están en *staging* (snapshot):

```
git commit -m "Un comentario descriptivo..."
```



Git – Comandos para trabajo colaborativo

Descargar un proyecto existente y toda su historia:

```
git clone [url]
```

Subir todos los commits de cierto branch al repositorio remoto (Ej: Github, Bitbucket):

```
git push origin [branch]
```

Incorporar cambios de repositorio remoto en branch actual:

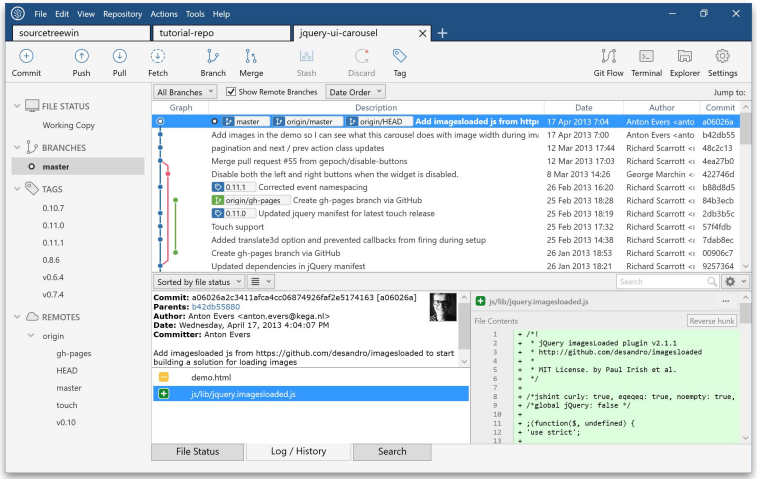
```
git pull
```

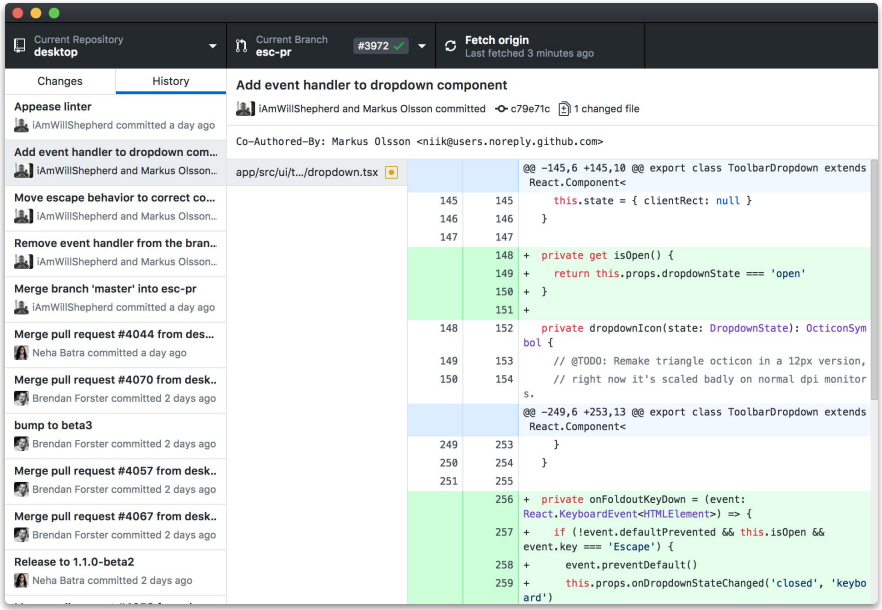
Combinar la historia de cierto branch con el branch actual:

```
git merge [branch]
```

Git – Interfaces Gráficas

Existen interfaces gráficas que facilitan mucho el trabajo con Git, sobre todo para usuarios principiantes. Ejemplos: SourceTree y GitHub Desktop.

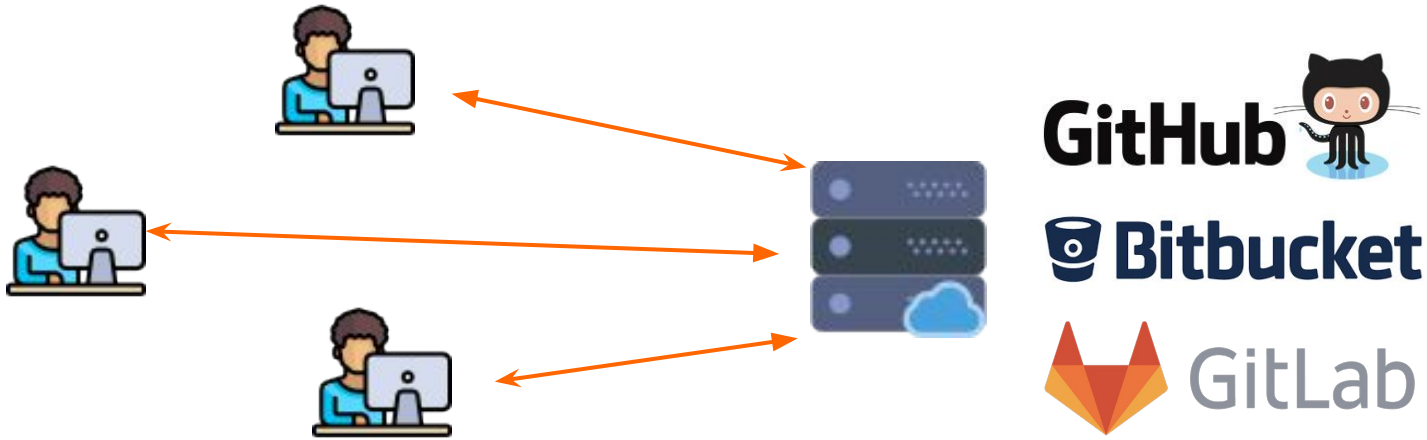




Repositorios remotos en la nube

Repositorios en la nube (1/3)

Como se comentó, por más de que Git sea un VCS distribuido, lo más común es trabajar con un repositorio central. Además, es muy común que dicho repositorio se encuentre alojado en un servicio en la nube como GitHub, Gitlab o Bitbucket.



Este tipo de repositorios también funciona como una gran herramienta de backups.

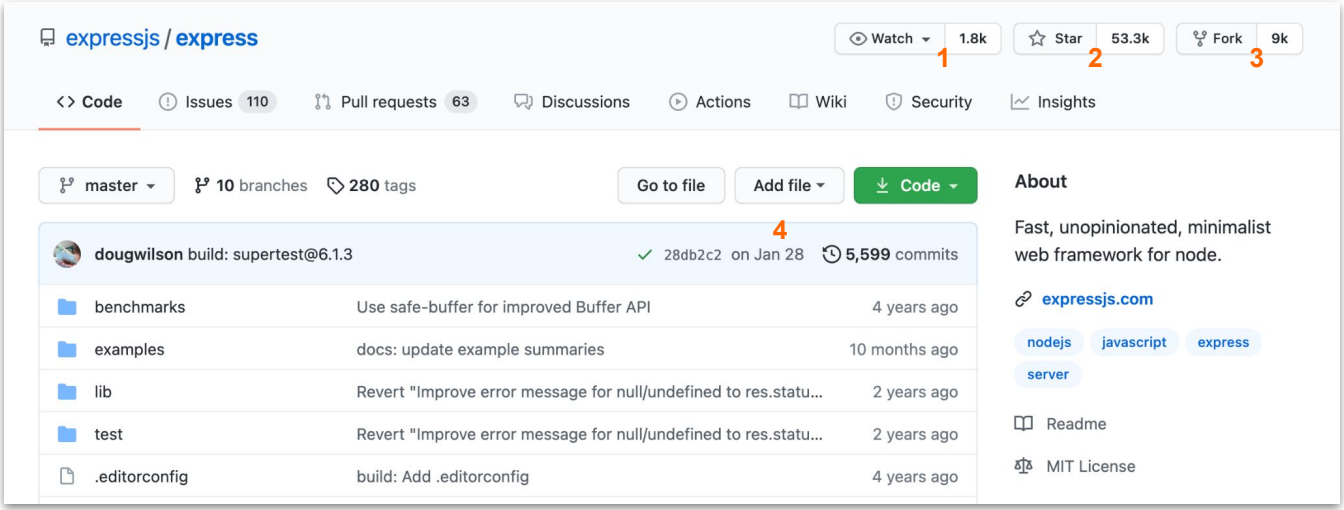
Repositorios en la nube (2/3)

GitHub es el servicio más conocido, sobre todo por la gran cantidad de repositorios de proyectos open source que residen allí. Ejemplos:

- Node.js: https://github.com/nodejs/node.
- React: https://github.com/facebook/react.
- Laravel: https://github.com/laravel/laravel.
- Bootstrap: https://github.com/twbs/bootstrap.
- Vue.js: https://github.com/vuejs/vue.

Crear repositorios en GitHub es gratuito, pero hay ciertas limitaciones.

Repositorios en la nube (3/3) – GitHub – Indicadores

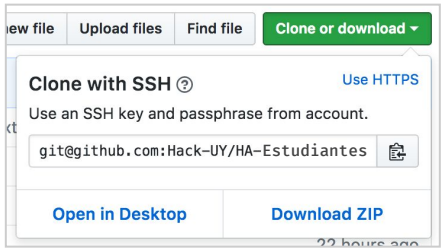


1. Cantidad de personas que están pendientes de cambios en el repositorio (lo están *mirando*).
2. Cantidad de estrellas (o *likes*) del repositorio.
3. Cantidad de veces que otros desarrolladores crearon (clonaron) su propia versión del repositorio.
4. Tiempo transcurrido desde el último commit.

Ejercicio 1

Ejercicio 1

1. Crearse una cuenta en GitHub: https://github.com.
2. Descargar GitHub Desktop.
3. Crear un repositorio privado en GitHub.
4. Clonar dicho repositorio. Se puede hacer haciendo click en “Open in Desktop”.



Esto hace una copia del código del proyecto en su computadora (repositorio local). GitHub Desktop les preguntará dónde quieren crear la carpeta del proyecto dentro de su equipo.

Ejercicio 1 (cont)

5. Crear algún archivo (ej: un `index.html`) dentro de la carpeta del proyecto.
6. Hacer commit del archivo.
7. Hacer push de los cambios.
8. Ver el repositorio en GitHub (en la web).
9. Realizar modificaciones al archivo.
10. Hacer commit y push nuevamente. Ver los cambios en GitHub.

EXTRA

Ejercicio 2

EXTRA

Ejercicio 2

1. Darle acceso a su repositorio (el que crearon en el ejercicio anterior) a un compañero de la clase. Darle permisos de escritura (no solamente de lectura).
2. A su vez, otro compañero les dará acceso a ustedes.
3. La idea es que se hagan cambios cruzados en sus proyectos.

👉 ¿Qué pasa si dos personas hacen cambios sobre el mismo archivo? ¿Cómo se soluciona? ¡Pruébenlo!

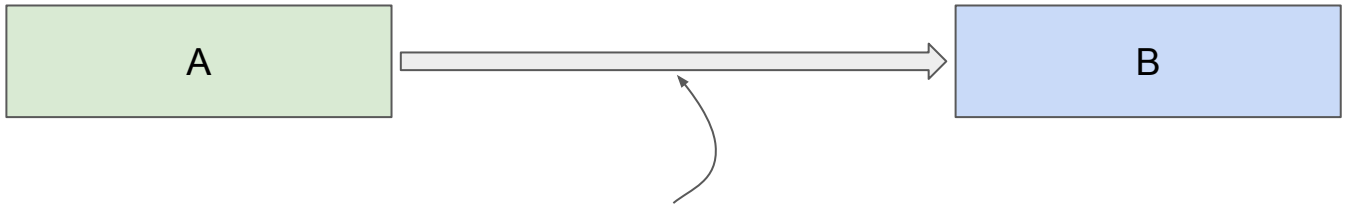
Middlewares

Middlewares (1/2)

En el ambiente de Express se habla mucho de *middlewares*, aunque no es un concepto exclusivo de Express y probablemente hayan oído hablar de *middlewares* en otros ámbitos.

Como dice el nombre, el *middleware* es un "pedazo de código" que se ejecuta en medio de una comunicación, con el fin de agregar cierta funcionalidad en el proceso.

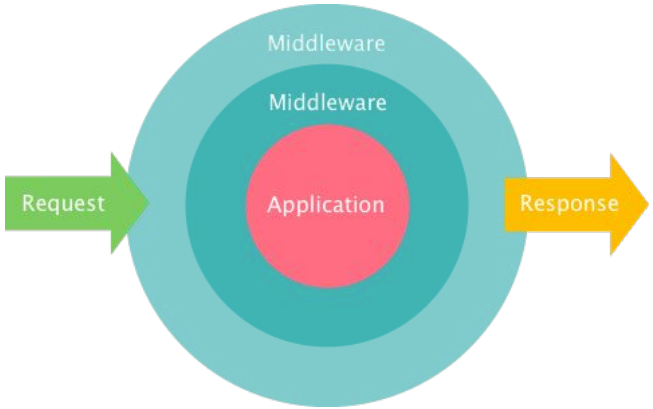
Ejemplo, un sistema A le quiere enviar datos a un sistema B:



Podríamos crear un *middleware* que se "meta" en medio de esa comunicación con el fin de, por ejemplo, manipular los datos que están siendo enviados. Por ejemplo, podríamos hacer un corrección ortográfica de los datos. Y podríamos "encadenar" otros *middlewares*.

Middlewares (2/2)

En el ámbito de las aplicaciones web, típicamente se utilizan *middlewares* para filtrar los *requests* HTTP, los cuales se hacen pasar por varias capas de *middlewares* antes de llegar al “core” de la aplicación.



Uso de middlewares en Express

Uso de middlewares en Express (1/4)

En Express, un *middleware* es una función que recibe tres parámetros:

1. Objeto Request: `req`.
2. Objeto Response: `res`.
3. Función `next`.

La función *middleware* puede hacer una o varias de las siguientes acciones:

- Ejecutar cierto código (en principio, cualquier código).
- Hacer modificaciones sobre los objetos *Request* y *Response*.
- Finalizar el ciclo de *request-response*.
- Llamar al próximo *middleware* que hay para ejecutar.

Uso de middlewares en Express (2/4)

Ejemplo de un *middleware* que se encarga de hacer un *log* cada vez que llega un *request* y de agregarle el atributo `requestTime` al *request* que llegó.

```
const miLogger = (req, res, next) => {

  console.log("Se recibió un request");

  req.requestTime = new Date();

  next(); // Para dar paso al siguiente middleware.

};
```

Notar que por ahora sólo se creó una función. Aún falta decirle a Express cuándo se debería ejecutarla.

Uso de middlewares en Express (3/4)

Siguiendo con el ejemplo anterior, ahora que tenemos creada la función *middleware* tenemos que indicarle a Express cuándo usarla.

Hay más de una forma de hacerlo.

Si queremos que el *middleware* se utilice para toda la aplicación, podemos hacerlo así:

```
const express = require("express");
const app = express();
app.use(miLogger);
```

Notar que el orden en que se hacen los *bindings* será el orden en que se ejecuten los *middlewares*.

Uso de middlewares en Express (4/4)

En caso de sólo se quiera ejecutar el *middleware* para todos los métodos de determinada ruta, se puede especificar de la siguiente manera:

```
app.use("/productos", miLogger);
```

También se puede agregar un *middleware* para determinada ruta y método:

```
app.post("/productos", miLogger, function (req, res) {
   // Código normal del handler...
});
```

Para ver otras formas de usar *middlewares*, consultar la documentación oficial.

Middlewares integrados

Middlewares integrados (1/2)

Express ya viene con una serie de *middlewares* integrados que podemos usar.

Por ejemplo:

- `express.static` para servir archivos estáticos como imágenes, CSS, etc.
- `express.json` para *parsear requests* que vienen con contenido de tipo JSON.
- `express.urlencoded` para *parsear requests* que vienen con contenido de tipo URL-Encoded (como cuando nos están enviando datos que vienen desde un formulario).

👉 Definición: “Parsear” significa analizar un texto sintácticamente y convertirlo en otra estructura como, por ejemplo, un objeto. Un servidor, cuando recibe un *request* con contenido, recibe simplemente un texto, una cadena de caracteres. Por lo tanto hay que indicarle a Express cómo queremos *parsear* dicho texto.

Middlewares integrados (2/2)

Ejemplo de uso:

```
const express = require("express");
const app = express();


app.use(express.urlencoded({ extended: true }));
```

Con este código le estamos diciendo a Express que cuando lleguen datos de tipo “URL-Encoded” se debe crear un atributo `body` dentro del objeto *request*:

```
req.body
```

Ejercicio 3

Ejercicio 3

1. Crear una carpeta llamada `ejercicio_listaFrutas`.
2. Inicializar un proyecto con el comando `npm init -y`.
3. Verificar que se haya creado el archivo `package.json`.
4. Crear un archivo: `index.js`.
5. Instalar los módulos `express` y `nunjucks`.
6. Crear las siguientes rutas:
    - [GET] http://localhost:3000/frutas.
    - [POST] http://localhost:3000/frutas.

Ejercicio 3 (cont)

7. En la página de frutas debe mostrarse:
    - Un título `<h1>`.
    - Un listado `<ul>`
      (inicialmente con 3 frutas: "Manzana", "Pera" y "Frutilla", es decir, con 3 elementos `<li>`).
    - Un formulario con un campo de texto y un botón llamado “Agregar”.
8. El `action` del formulario debe ser la URL anterior y el método POST.
9. Al hacer click en el botón se debe mostrar las misma página, pero con la nueva fruta.