

CAPÍTULO 1

INTRODUCCIÓN AL DESARROLLO CON SYMFONY

Contenidos

- Selección de arquitecturas
- Editores y compiladores
- Sistemas de control de versiones de código fuente
- Gestión de librerías de terceros con Composer

Objetivos

- Reconocer y evaluar las herramientas de programación en entorno servidor
- Identificar las ventajas que proporcionan los gestores y repositorios de librerías
- Verificar los mecanismos de gestión de versiones de código fuente

RESUMEN DEL CAPÍTULO

En este capítulo estudiaremos la configuración del entorno de trabajo que utilizaremos para desarrollar aplicaciones con Symfony. Esto incluye el entorno integrado de desarrollo, gestor de dependencias, sistemas de control de versiones y otras herramientas adicionales.

1.1. INTRODUCCIÓN

Para instalar *Symfony* en nuestros equipos tenemos principalmente dos opciones: por un lado, podemos utilizar el instalador que *SensioLabs* proporciona, y por otro, podemos hacer uso del gestor de paquetes *Composer*¹. En este libro vamos a utilizar la segunda opción, ya que en el futuro nos facilitará la gestión de dependencias de nuestros proyectos.

1.2. INSTALACIÓN DE COMPOSER

Para instalar *Composer* puedes seguir los pasos que se indican en la propia página del software:
<https://getcomposer.org/doc/00-intro.md#installation-windows>

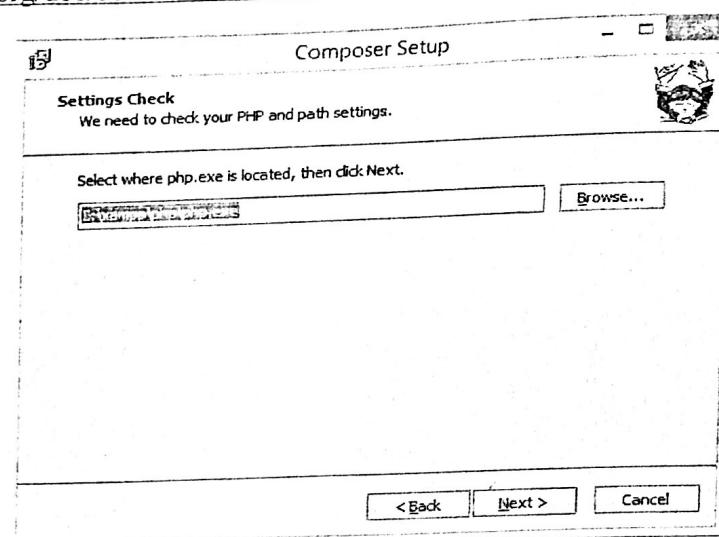


Figura 1.1. Asistente de instalación de Composer

Los pasos a seguir son:

- Descarga el instalador y ejecútalo en tu equipo.
- Avanza en el asistente de instalación con las opciones por defecto hasta que te pida que indiques la ruta en la que está el archivo php.exe en tu equipo. En nuestro caso, y dado que vamos a utilizar *xampp*, estará en C:\xampp\php\php.exe.
- Sigue avanzando en los pasos del asistente con las opciones por defecto. Finalmente *Composer* se instalará y te informará de que ha modificado la variable PATH de tu equipo para que pueda ser utilizado correctamente.

1.3. CREANDO UNA APLICACIÓN SYMFONY CON COMPOSER

Para crear una aplicación *Symfony* mediante *Composer*, debemos utilizar el siguiente comando:

```
composer create-project symfony/framework-standard-edition project_name
```

¹<https://getcomposer.org/>

```
D:\netbeans-projects\symfony3>composer create-project symfony/framework-standard-edition test-project
Installing symfony/framework-standard-edition (v3.1.3)
- Installing symfony/framework-standard-edition (v3.1.3)
  Downloading: 100%
Created project in test-project
Loading composer repositories with package information
Installing dependencies (including require-dev) from lock file
- Installing doctrine/lexer (v1.0.1)
  Downloading: 100%
- Installing doctrine/annotations (v1.2.7)
  Downloading: 100%
- Installing twig/twig (v1.24.1)
  Downloading: 100%
- Installing symfony/polyfill-util (v1.2.0)
  Downloading: 100%
```

Figura 1.2. Creación de un proyecto Symfony con Composer

En los últimos pasos de la creación del proyecto *Composer* pedirá el valor de varios parámetros de la aplicación como la base de datos, servidor de correo, etc. Como de momento no vamos a utilizar nada de esto, puedes simplemente ir asignando los valores por defecto pulsando *Intro*.

Una vez creado el proyecto, dentro del directorio del mismo, podemos arrancarlo utilizando el servidor interno de php mediante el comando:

```
php bin/console server:run
```

Una vez ejecutado el comando la aplicación se lanzará utilizando el puerto 8000 de nuestra máquina. Para poder visualizar la página de inicio debes ir a la siguiente URL:

<http://localhost:8000/>

1.4. ACTUALIZANDO LAS APLICACIONES EN SYMFONY

Dado que Composer es un gestor de dependencias, nos facilita en gran medida el mantenimiento de las versiones de las librerías externas que nuestros proyectos utilizan. Para actualizar las librerías de un proyecto podemos utilizar el comando (a ejecutar desde la raíz del proyecto):

```
composer update
```

```
^C
D:\netbeans-projects\symfony3>test-project>composer update
Loading composer repositories with package information
Updating dependencies (including require-dev)
- Removing monolog/monolog (1.20.0)
- Installing monolog/monolog (1.21.0)
  Downloading: 100%
Writing lock file
Generating autoload files
> Incenteev\ParameterHandler\ScriptHandler::buildParameters
Updating the "app/config/parameters.yml" file
> Sensio\Bundle\DistributionBundle\Composer\ScriptHandler::buildBootstrap
> Sensio\Bundle\DistributionBundle\Composer\ScriptHandler::clearCache
// Clearing the cache for the dev environment with debug true

[OK] Cache for the "dev" environment (debug=true) was successfully cleared.
```

Figura 1.3. Actualización de dependencias con Composer

4 Symfony. Desarrollo Web en Entorno Servidor

En función de la cantidad de librerías que el proyecto utilice este proceso tardará más o menos tiempo.

1.5. GESTIÓN DEL CÓDIGO FUENTE CON GIT

Git es un sistema de control de versiones distribuido muy extendido y que permite gestionar los cambios y versiones en el código fuente trabajando desde un repositorio ubicado en la propia máquina del usuario. Por tanto, el desarrollador puede aprovecharse de las funcionalidades de Git también offline, algo que no permiten sistemas centralizados tradicionales como Subversion.

Dado que en este libro se utilizará Netbeans como IDE de desarrollo, se va a ilustrar cómo gestionar el código fuente de una aplicación Symfony mediante el plugin de Git para Netbeans.

Una vez que hayas creado un primer proyecto Symfony en Netbeans o hayas importando uno previamente creado mediante comandos, lo primero que debemos hacer es inicializar el repositorio sobre el directorio en el que se encuentra el proyecto. Para ello nos dirigimos a la opción Team > Git > Initialize Repository.

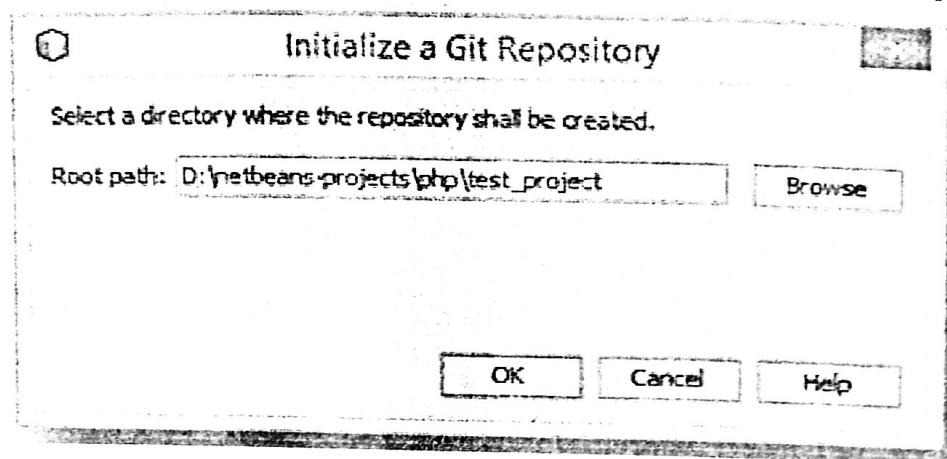


Figura 1.4. Inicialización de un repositorio Git sobre un proyecto

Cuando creamos un proyecto Symfony con Composer, automáticamente se crea un archivo `.gitignore`. Este archivo tiene como misión establecer qué archivos o directorios van a ser omitidos por Git a la hora de hacer operaciones de commit. Si abres este archivo verás que, por ejemplo, la carpeta `vendor` será omitida. Esta carpeta contiene las librerías que el proyecto utiliza, y su tamaño suele ser bastante elevado. Por tanto, la excluimos de los commit estaremos evitando sobrecargar el espacio en disco del repositorio y el tráfico en red. Además, Composer es capaz de restaurar el contenido de esta carpeta leyendo el archivo `composer.json`, que contiene un listado de todas las librerías con sus números de versión.

Netbeans también tiene en cuenta el archivo `.gitignore`, y como puedes observar sombrean en gris todos aquellos que serán ignorados al hacer commit.

Una vez hecho esto, podemos proceder a hacer el primer commit sobre el proyecto. Para ello, hacemos clic con el botón derecho del ratón sobre el proyecto y vamos a `Git > Commit`. Se abrirá una ventana en la que podremos introducir un comentario descriptivo (es una buena práctica) y un listado de los archivos que se añadirán al control de código fuente.

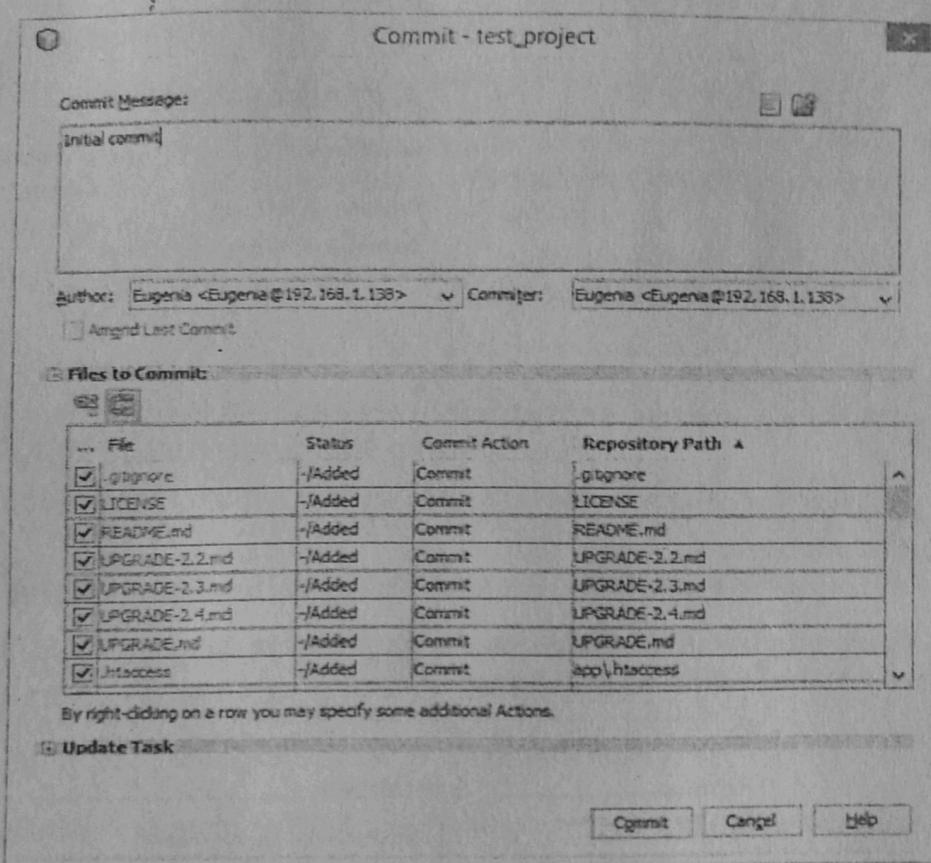


Figura 1.5. Almacenar un repositorio git en un servidor remoto

El repositorio local recién creado puede ser trasladado a un servidor remoto fácilmente mediante la operación *Push*. Existen múltiples servicios online que permiten crear repositorios *Git*, destacando principalmente *Github* y *Bitbucket*. Utilizaremos este último por la posibilidad de crear repositorios privados sin coste.

Para subir nuestro repositorio local a *Bitbucket*, lo primero que hay que hacer es ir a *Bitbucket* y crear un repositorio. Este paso es siempre igual.

Figura 1.6. Creación de un nuevo repositorio en bitbucket

En la siguiente pantalla indica que tienes un repositorio que te gustaría subir (*push up*).

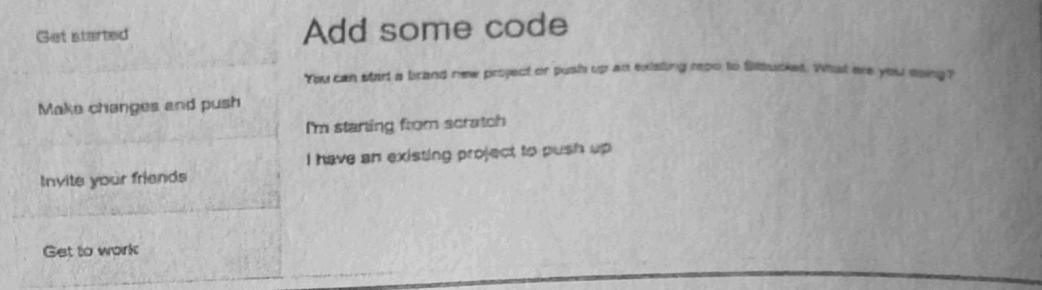


Figura 1.7. Subiendo un repositorio a Bitbucket

Al hacerlo, el propio *Bitbucket* te dice qué comandos Git debes utilizar para hacerlo:

```
git remote add origin
https://eugenia_perez@bitbucket.org/eugenia_perez/symfony.test_project.git
```

Como estamos utilizando *Netbeans*, simplemente debemos hacer clic con el botón derecho del ratón en proyecto y nos vamos a *Git > Remote > Push*.

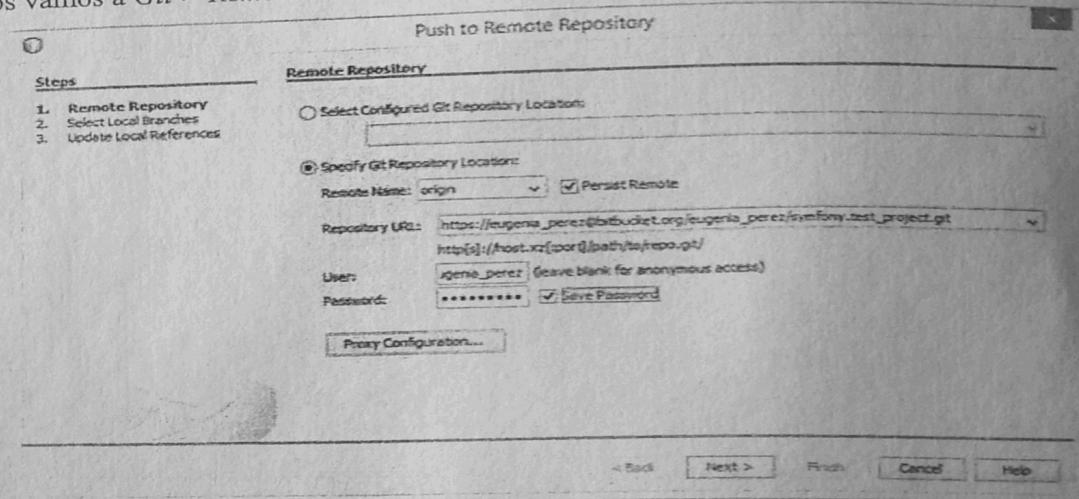


Figura 1.8. Push de un repositorio local con Git y Netbeans

Las operaciones anteriores podrían realizarse igualmente desde la consola en caso de que dispusiésemos de un cliente gráfico o una herramienta como *Netbeans*. A continuación se explica cómo.

Lo primero que debes hacer es instalar el software de Git:

<http://git-scm.com/download/>

A continuación, ejecuta el fichero descargado e instala Git.

Vamos a suponer que tenemos una carpeta *testSite* con dos ficheros, un HTML y un JS. Lo primero debemos hacer es crear un repositorio local sobre esa carpeta. Para ello, vamos a la ruta donde tengas carpeta con el proyecto e introduce el siguiente comando:

```
git init testSite
```

La respuesta que se producirá será:

```
Initialized empty Git repository in /Users/eugenia/Documents/testSite/.git/
```

Con esto ya tenemos creado el repositorio local. Ahora, debemos añadir los ficheros de nuestro proyecto al repositorio, para decirle a Git que son nuevos, y que cuando hagamos un *commit* los queremos incluir. Para ello se introduce el comando:

```
cd testSite //para asegurarnos que estamos en el directorio del proyecto
git add -A
```

Por tanto, ahora ya puedes hacer *commit*.

```
MacBook-Air-de-Eugenia:testSite eugenias git commit -m 'Initial commit'
[master (root-commit) 287b621] Initial commit
  Committer: Eugenia <eugenia@MacBook-Air-de-Eugenia.local>
  Your name and email address were configured automatically based
  on your username and hostname. Please check that they are accurate.
  You can suppress this message by setting them explicitly:
    git config --global user.name "Your Name"
    git config --global user.email you@example.com

After doing this, you may fix the identity used for this commit with:
  git commit --amend --reset-author

  2 files changed, 11 insertions(+)
  create mode 100644 test.html
  create mode 100644 test.js
MacBook-Air-de-Eugenia:testSite eugenias$ git status
# On branch master
nothing to commit, working directory clean
```

Se hace el *commit* y se indica en un mensaje que dos ficheros han sido guardados en el repositorio. Ahora podríamos modificar cualquiera de los dos ficheros de nuestro proyecto (o ambos) y guardarlos. Para agregarlos al repositorio, habría que repetir los comandos *git add -A* para actualizarlos puesto que añadidos ya están, y *git commit* para guardarlos en el repositorio.

1.6. ESTRUCTURA BÁSICA DE UNA APLICACIÓN

Las aplicaciones web interactúan con los usuarios mediante páginas web. Estas páginas son el resultado de una respuesta HTTP que el servidor devuelve al cliente tras una petición previa de éste. Para determinar qué página debe enviar el servidor en función de la petición recibida, debe haber una tabla de rutas que relacionen ambas entidades.

Para ilustrar este proceso vamos a comenzar desarrollando una aplicación *Symfony* sencilla que reciba una petición del cliente con su edad y le devuelva una página en la que pueda visualizar su año de nacimiento.

En *Symfony* todo el código de la aplicación debe residir en un *bundle*, pudiendo existir varios de estos en la misma aplicación. Cuando creamos una aplicación *Symfony* con *Composer* o mediante *Netbeans*, automáticamente se crea ya con un bundle por defecto, que no es más que un módulo reutilizable dentro de la aplicación.

Si necesitas crear un *bundle* puedes utilizar el siguiente comando:

```
php bin/console generate:bundle --namespace=BundleName --format=yml
```

Para realizar esta aplicación de ejemplo utilizaremos el esqueleto generado en el capítulo inicial con *Composer*. Una aplicación Symfony tiene la siguiente estructura:

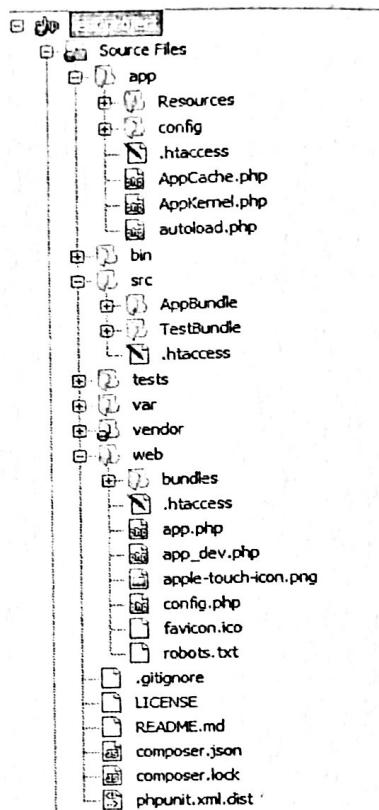


Figura 1.6. Estructura de una aplicación Symfony3

El propósito de cada directorio es el siguiente:

- **app/Resources:** almacena las vistas y archivos de internacionalización de la aplicación.
- **app/config:** contiene toda la configuración del entorno.
- **src/AppBundle:** almacena los controladores y rutas de *Symfony*, código del modelo de dominio (por ejemplo *Doctrine*) y la lógica de negocio general de la aplicación.
- **tests:** contiene los archivos para generar pruebas unitarias.
- **var:** este directorio está destinado a almacenar aquellos archivos sobre los que el sistema necesitará permisos de escritura para realizar las diferentes operaciones que la aplicación requiera.
- **vendor:** es el directorio donde *Composer* instala las dependencias de la aplicación. Este directorio nunca debe ser modificado manualmente.
- **web:** almacena todos los recursos estáticos de la aplicación, incluyendo imágenes, hojas de estilo, JavaScript, etc.

1.6.1. Rutas

La definición de rutas en *Symfony* reside en el archivo `app/config/routing.yml`. Las dos maneras más extendidas de definir las rutas son o bien indicándolas de manera explícita en este archivo, o mediante anotaciones. Un ejemplo del primer caso sería:

```
app:  
    path: /calculator/{age}  
    defaults: {_controller: AppBundle:Calculator:index}
```

Mediante el fragmento de código anterior estamos indicando que cuando llegue una petición de tipo GET a la ruta `/controlador/{age}`, ésta será tratada por el controlador *CalculatorController* del *bundle* de nuestra aplicación, y dentro de este se invocará a la acción *indexAction* pasándole como parámetro la edad indicada por el usuario. Veamos pues qué aspecto tiene este controlador:

```
class CalculatorController extends Controller {  
  
    public function indexAction($age) {  
        $currentYear = date('Y');  
  
        return new Response('<html><body>Current year: ' . $currentYear .  
            '<br/>Year of birth: ' . ($currentYear - $age) . '</body></html>');  
    }  
}
```

El código es bastante trivial. El método *indexAction* recibe la edad y se la resta al año en el que estamos actualmente. Por último, envía la respuesta en forma de fragmento HTML.

Para probar el funcionamiento de esta sencilla aplicación despliega el código en un servidor capaz de interpretar PHP, como por ejemplo Apache. A continuación, para ir a la acción recién implementada inserta en tu navegador la siguiente URL:

http://localhost/test_project/web/app_dev.php/calculator/20

Fíjate que tanto en el fichero de *routing* como en la URL del navegador no ha sido necesario indicar las palabras *controller* o *action*. Esto es porque por convención, *Symfony*, como muchos otros *frameworks* MVC asume determinadas convenciones, entre ellas esta.

1.6.2. Plantillas y vistas

En la acción anterior, por tanto, hemos visto cómo retornar un fragmento de código HTML para ser mostrado al usuario. No obstante, esto no es lo más elegante ni mantenible. Lo ideal es definir las páginas (vistas) por separado, y desde los controladores enviar redirecciones a estas pasándoles los datos necesarios a mostrar.

Para ello, utilizaremos el método *render*, que nos permitirá indicar la plantilla a la que queremos redirigir la petición y, un array asociativo con los datos que el controlador desea enviar a la plantilla para su renderización. La acción anteriormente vista quedaría de la siguiente manera:

```
public function indexAction($age) {  
    $currentYear = date('Y');  
    $year = $currentYear - $age;  
  
    return $this->render('calculator/index.html.twig',  
        array('year' => $year));  
}
```

En la última línea del método se está invocando a la plantilla `index.html.twig`. La extensión `.twig` denota que es una plantilla que será ejecutada por el motor de plantillas de *Symfony Twig*. Además, como se puede ver, le estamos enviando el año que debe mostrar.

La plantilla invocada se encuentra en el directorio `app > Resources > views > calculator`. Esta plantilla tiene el siguiente contenido:

```
{% extends 'base.html.twig' %}

{% block body %}
    Year of birth: {{year}}
{% endblock %}
```

La primera línea sirve para especificar que esta plantilla, a su vez, utiliza otra que se usará como *layout*. Es decir, `base.html.twig` será una plantilla que definirá una estructura común a muchas otras para evitar tener que repetir el mismo código HTML una y otra vez. El resto del código es bastante sencillo. Se define un bloque en el que se imprimirá el año recibido desde el controlador. Para obtener ese valor se hace uso de la notación `{{valor}}`, debiendo coincidir *valor* con el nombre de la variable enviada por el controlador.

1.6.3. El directorio web

Como ya se ha mencionado brevemente unas líneas más arriba, el directorio web sirve principalmente para almacenar recursos estáticos como JavaScript, hojas de estilo o imágenes que la aplicación necesita. Además, también se guardan los *front controllers*. Los front controller (por ejemplo, `app_dev.php`) son los ficheros PHP que se ejecutan para lanzar e inicializar una aplicación Symfony.

Una consecuencia de utilizar un front controller es que las URLs de la aplicación variarán ligeramente con respecto a las aplicaciones PHP tradicionales. En el ejemplo anterior vimos que la URL utilizada era:

`http://localhost/test_project/web/app_dev.php/calculator/20`

Donde `app_dev.php` es un *front controller*. En caso de que queramos utilizar URLs omitiendo el *front controller*, deberíamos trabajar en la implementación de redirecciones (por ejemplo con el módulo `mod_rewrite` de Apache) para conseguir que se ejecute este archivo sin necesidad de incluirlo de manera explícita en la URL.

1.6.4. Configuración de varios entornos de ejecución

En el desarrollo profesional de aplicaciones, disponer de distintos entornos de ejecución es absolutamente imprescindible. Así, durante las diferentes etapas en las que se divide el desarrollo, una aplicación irá escalando entornos hasta finalmente alcanzar el de producción. *Symfony* define por defecto 3 entornos: `dev`, `test` y `prod`, que se corresponden con desarrollo, pruebas y producción.

Es muy conveniente que los *frameworks* faciliten la gestión de la aplicación en distintos entornos, ya que existirán diferencias sobre todo en lo que respecta a la configuración de la aplicación. Por ejemplo, una aplicación puede generar *logs* en desarrollo y prueba, pero no en producción, la base de datos a utilizar es diferente en distintos entornos, etc.

Para ejecutar la aplicación en modo `dev`, simplemente basta con utilizar la siguiente URL:

`http://localhost/test_project/web/app_dev.php/calculator/20`

Mientras que la URL equivalente en producción sería:

`http://localhost/test_project/web/app.php/calculator/20`

Fíjate que la diferencia radica en el *front controller* utilizado (`app_dev.php` contra `app.php`). Por tanto, si deseas crear un nuevo entorno, lo único que deberás hacer es copiar cualquiera de estos dos ficheros y modificar la siguiente línea:

```
$kernel = new AppKernel('NUEVO_ENTORNO', true);
```

Además del *front controller* mencionado anteriormente, existen más archivos en un proyecto *Symfony* dependientes del entorno. Por ejemplo, en `app/config` existen varios archivos `config.yml`, uno por entorno. Si los inspeccionas, podrás ver que hay configuración de base de datos, motor de plantillas a utilizar, *logging*, etc.

COMPRUEBA TU APRENDIZAJE

1º) Crea un nuevo proyecto con Composer. A continuación descárgate las dependencias. Ahora edita el archivo `composer.json` para que tu proyecto pase a utilizar la última versión 7.1.x disponible de PHP. Es decir, debes evitar que Composer instale versiones mayores o iguales que la 7.2, en caso de que estuvieran disponibles.

2º) Crea un repositorio local con Git sobre el proyecto recién implementado en el ejercicio 1. Realiza un primer *commit* a tu repositorio local, a continuación añade el archivo `.gitignore` al directorio raíz y haz los cambios necesarios para que se evite hacer commit de:

- Todo lo que haya en el directorio `nbproject`.
- Todo el contenido de los directorios `/var/cache` y `/var/logs`.

ACTIVIDADES DE AMPLIACIÓN

En *Symfony* los proyectos se estructuran en bundles. Al crear el esqueleto de una aplicación con Composer, se crea automáticamente un bundle. Estos componentes vienen a ser plugins que se pueden activar y desactivar. Así, podríamos crear diferentes bundles en nuestra aplicación para agrupar funcionalidad, o crear un bundle con código reutilizable que pueda ser utilizado en otros proyectos. Además, también podemos importar bundles de terceros en nuestros proyectos que nos aporten funcionalidad extra sin necesidad de implementarla nosotros mismos.

Prueba a crear un proyecto e importar un bundle de terceros. Puedes buscar uno por Internet, o por ejemplo, utilizar el *EasyAdminBundle*, pensado para proporcionar una interfaz de administración a tus aplicaciones Web. Puedes leer cómo incluirlo a través del siguiente enlace:

<https://github.com/javiereguiluz/EasyAdminBundle/blob/master/Resources/doc/getting-started.md>