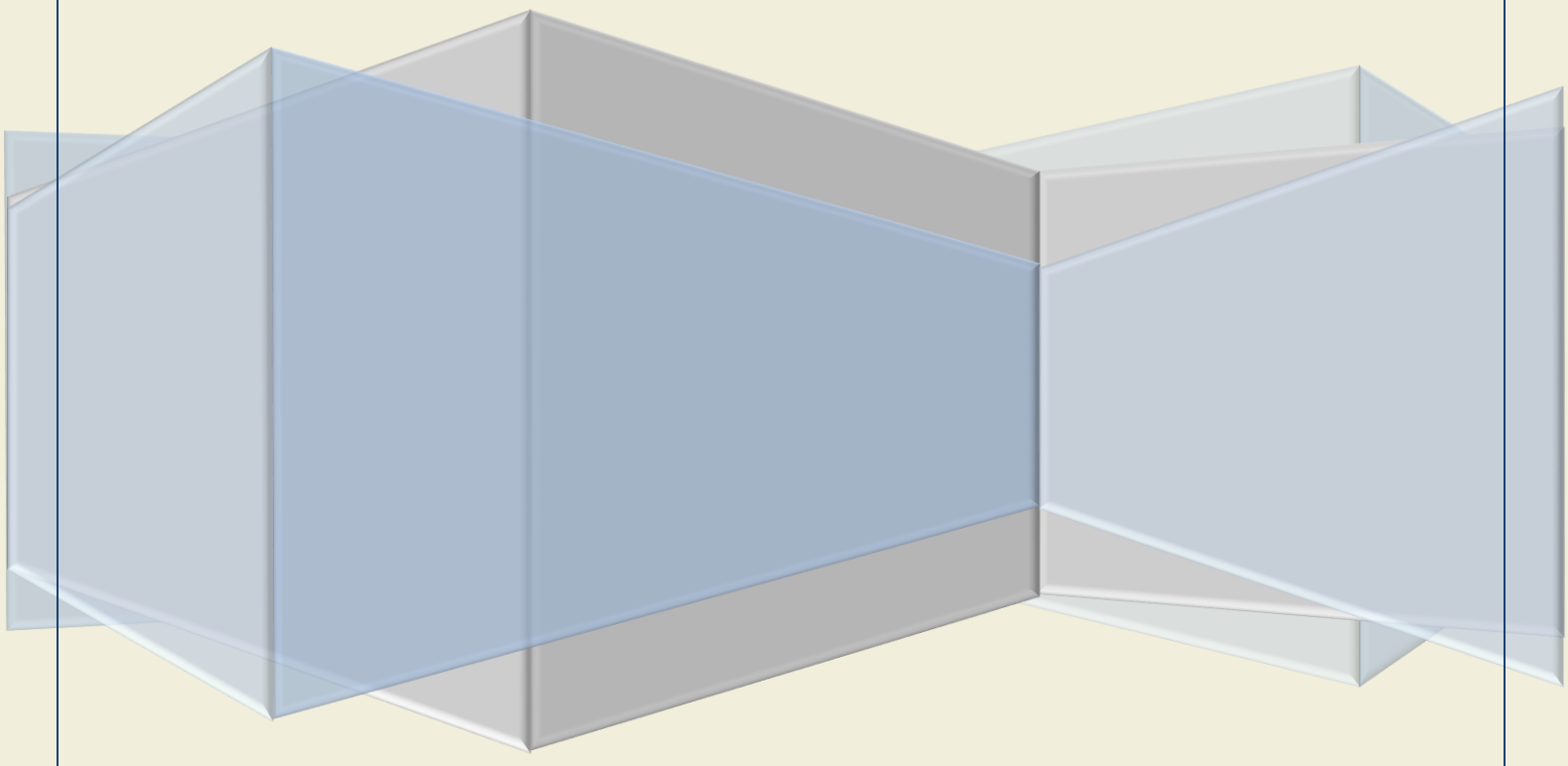
Capítulo 1
Introducción a PHP

Programación: Backend Developer

Potrero Digital



CONTENIDO

QUÉ ES UN LENGUAJE DE PROGRAMACIÓN

El "cerebro" de una computadora es la Unidad Central de Procesamiento (CPU). En general una CPU no puede almacenar más que unos pocos números y realizar operaciones matemáticas básicas con ellos. Además esos números se pueden recibir de o enviar a la memoria o a los distintos dispositivos de entrada y salida (teclado, monitor, disco duro, impresora, etc.). Para cada una de estas operaciones (obtener números de un sitio, hacer cálculos con esos valores, enviar el resultado a otro sitio) existe una instrucción diferente y cada CPU tiene su propio juego de instrucciones, más o menos amplio dependiendo de la complejidad de la CPU.

En la memoria del computadora se puede guardar una secuencia de esas instrucciones, que el computadora es capaz de seguir desde el principio hasta el final (aunque esas instrucciones no tienen por qué ejecutarse siempre en el mismo orden porque existen instrucciones para saltar de un punto a otro de la secuencia de instrucciones). Esas secuencias se llaman programas ejecutables.

Cualquier función que realice una computadora, por sofisticada que parezca, en realidad no es más que una larga secuencia de instrucciones elementales. Para los humanos resulta muy complicado escribir directamente los programas ejecutables porque cualquier tarea requiere muchísimos pasos, que la computadora ejecuta de forma instantánea, pero para que las computadoras funcionen es necesario disponer de programas ejecutables.

Los lenguajes de programación se han creado para facilitar la elaboración de programas ejecutables. Un lenguaje de programación es un lenguaje artificial diseñado para dictar instrucciones a una computadora, pero tienen la ventaja de que no es necesario desmenuzar las tareas tanto como lo requiere la CPU sino que permiten definir tareas de forma más abstracta. Por ejemplo, un lenguaje de programación puede tener una instrucción para calcular raíces cuadradas, aunque haya CPU que no tengan la capacidad de calcular raíces cuadradas.

Existen muchos lenguajes de programación diferentes, pero en general, un programador que quiere conseguir que la computadora realice determinadas operaciones, tiene que seguir estos pasos:

- El programador escribe una secuencia de instrucciones siguiendo las reglas de un lenguaje de programación. Esa secuencia de instrucciones se guarda en uno o varios archivos de texto. A estos archivos se les llama **código fuente del programa**.
- Mediante un programa especial (llamado compilador o intérprete) capaz de realizar esa tarea el computadora convierte los archivos de texto en archivos ejecutables, es decir, traduce la secuencia de instrucciones escritas por el programador en instrucciones. A estos archivos ejecutables se les llama **programas ejecutables**.
- El programador o cualquier otro usuario le puede pedir a la computadora que ejecute el programa ejecutable.

En el ejemplo citado de las raíces cuadradas, el compilador o intérprete tendría que ser capaz de convertir la instrucción de raíz cuadrada que hubiera escrito el programador en una serie de sumas, restas, multiplicaciones o divisiones que acaben dando el resultado deseado.

En Informática, normalmente se utiliza el término **programa** para referirse a las dos cosas:

- al fichero de texto escrito por un programador en algún lenguaje de programación
- al fichero ejecutable que ejecuta el computadora para producir un resultado.

Eso puede causar alguna confusión, pero se espera que el lector/oyente entienda por el contexto de cuál de los dos conceptos se está hablando.

Las ventajas de usar lenguajes de programación son numerosas:

- los lenguajes de programación son infinitamente más comprensibles para los humanos que las secuencias de instrucciones.
- un mismo código fuente se puede acabar ejecutando en computadoras con diferentes juegos de instrucciones, usando compiladores o intérpretes que sean capaces de realizar la traducción.
- mejorando los compiladores o intérpretes, un mismo código fuente se puede acabar convirtiendo en programas ejecutables más rápidos.

Las computadoras siguen al pie de la letra las instrucciones de los programas ejecutables. Por tanto, los programas deben tener en cuenta todos los posibles resultados de todas las instrucciones e incluir las instrucciones a seguir en cada caso, de manera que la computadora no se quede atascada en algún punto del programa. En el ejemplo citado de las raíces cuadradas, el programa tiene que tener en cuenta que no se pueden calcular las raíces cuadradas de números negativos (sin recurrir a los números complejos) y no tiene que pedirle al computadora que lo haga porque seguramente se producirá un error y el programa se interrumpirá.

GENEALOGÍA DE LOS LENGUAJES DE PROGRAMACIÓN

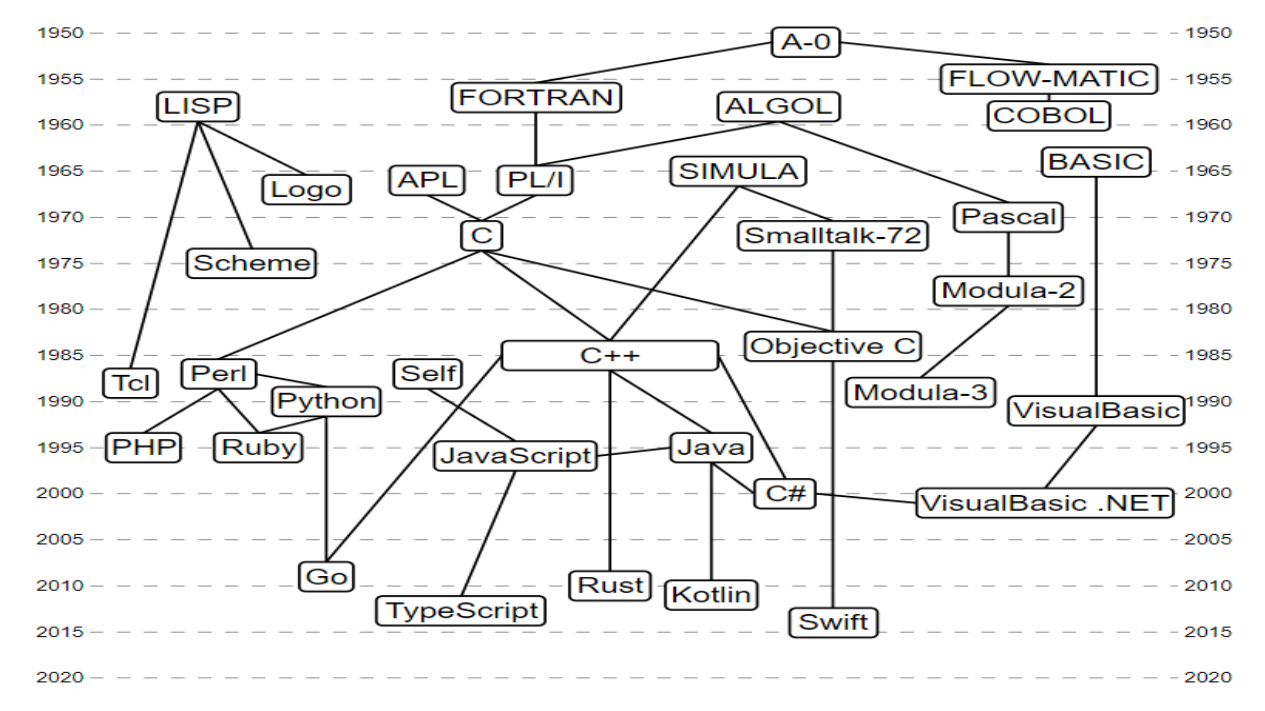
El inicio de la informática moderna se suele situar en 1946, con ENIAC, la primera computadora de propósito general (aunque entre 1936 y 1945 Konrad Zuse construyó en Alemania varios prototipos de computadoras de propósito general que fueron destruidos durante la segunda guerra mundial).

Esas computadoras, como todos los que han venido después, tenían su propio **lenguaje máquina**, el conjunto de instrucciones que puede ejecutar la CPU. Para hacer manejable la tarea de programar las computadoras, se crearon los **lenguajes ensamblador**, que utilizan palabras para referirse a las instrucciones. El problema de estos lenguajes es que están ligados a la CPU utilizada.

En los años 50 aparecieron los lenguajes de programación independientes de una máquina en concreto y en los años 60 se desarrollaron la mayoría de los conceptos en que se basan los actuales lenguajes de programación, perfeccionados a lo largo del tiempo.

Desde entonces, se han inventado muchísimos lenguajes de programación, que han permitido a los programadores resolver problemas cada vez más complejos.

La imagen siguiente, muestra algunos de los lenguajes de programación más importantes de la historia, la mayoría de los cuales se siguen utilizando en la actualidad



TIPOS DE LENGUAJES DE PROGRAMACIÓN

En los años 50, en los inicios de la informática, los lenguajes empleados eran los **lenguajes máquina**, que van íntimamente asociados a cada CPU, puesto que sus instrucciones son las mismas que las de la CPU. En esos años aparecieron los **lenguajes ensamblador**, que utilizan palabras para referirse a las instrucciones, aunque siguen asociados a cada CPU.

En los años 60 se desarrollaron la mayoría de tipos de lenguajes que se usan hoy en día:

Se pueden dar algunos criterios generales para distinguir unos lenguajes de otros, aunque cada vez más los lenguajes incluyen características que antes distinguían unos lenguajes de otros y las frontera están menos definidas. Algunos de esos criterios son los siguientes:

LENGUAJES IMPERATIVOS Y LENGUAJES DECLARATIVOS

En los **lenguajes imperativos** el programador escribe paso a paso las tareas a realizar. Entre ellos, están:

- los **lenguajes modulares**, que permiten definir bloques de instrucciones y reutilizarlos para construir programas más complejos (Algol, COBOL, Fortran, etc.).
- los **lenguajes procedurales**, que permiten definir procedimientos (rutinas, subrutinas, funciones, etc.) y reutilizarlos (BASIC, Pascal, C, etc.).
- los **lenguajes orientados a objetos**, en los que no se permite que cualquier porción del programa acceda a cualquier dato, sino que se agrupan en clases todos los procedimientos que se pueden aplicar a un tipo de datos (Smalltalk, C++, Java, etc.).

En los **lenguajes declarativos** el programador no escribe los pasos a realizar, sino los pasos permitidos en cada situación y es el computadora el que busca la secuencia adecuada. Entre ellos están:

- los **lenguajes funcionales**, que utilizan como elemento básico las funciones y la recursividad (APL, Haskell, Erlang, etc.).
- los **lenguajes lógicos**, que expresan las relaciones en términos lógicos y utilizan reglas de inferencia lógica para conseguir el resultado (Prolog, Datalog, etc.).

Existen lenguajes puros que siguen a rajatabla los principios de uno de estos tipos, pero la mayoría de lenguajes permiten mezclar tipos de programación diferentes.

No se puede afirmar la superioridad intrínseca de un tipo de lenguaje sobre otro, puesto que existen problemas que se abordan mejor con un tipo de lenguaje que otro.

Por ejemplo, en los años finales del siglo XX y en los primeros del siglo XXI, con el aumento de potencia de los microprocesadores, parecía que los lenguajes imperativos habían ganado la partida a los declarativos. Pero en los últimos años el crecimiento de la potencia bruta de las CPU individuales se ha detenido y en su lugar lo que se multiplica es el número de núcleos en cada CPU y en esa situación en la que lo más importante es poder "paralelizar" los programas (que se puedan ejecutar los programas en varios núcleos a la vez), los lenguajes declarativos llevan ventaja, por lo que en los últimos años está aumentando el interés por esos lenguajes, sin que por desgracia se hayan obtenido todavía resultados revolucionarios.

LENGUAJES COMPILADOS E INTERPRETADOS

El programa de computadora que es capaz de convertir el programa fuente escrito por el programador en el programa ejecutable que ejecuta la computadora puede realizar esa tarea de dos formas distintas:

- si es un **compilador**, la tarea de conversión se realiza una sola vez, es decir, el compilador crea un fichero ejecutable, que es independiente del programa fuente. De esta manera, el usuario no necesita el programa fuente, sino únicamente el programa ejecutable.

- si es un **intérprete**, la tarea de conversión se realiza cada vez que se quiere ejecutar el programa, es decir el intérprete lee una instrucción del programa fuente, la convierte en código ejecutable, la ejecuta y pasa a la siguiente. En este caso, el usuario necesita tener el programa fuente para poder ejecutarlo.

En principio, cualquier lenguaje de programación puede ser compilado o interpretado, aunque por tradición hay lenguajes que suelen utilizar compiladores y otros que suelen utilizar intérpretes.

Se solía decir que para un mismo programa fuente, la compilación produce programas ejecutables más rápidos que los programas interpretados, pero la tendencia es a que esas diferencias se reduzcan.

Mi opinión personal es que las preferencias por compiladores o intérpretes pueden deberse a motivaciones no técnicas. Por ejemplo, el programador de software comercial preferirá el uso de compiladores para poder comercializar el programa ejecutable sin necesidad de hacer público el programa fuente, mientras que el programador de software libre no tiene problemas con los intérpretes.

LENGUAJES TIPIFICADOS Y LENGUAJES NO TIPIFICADOS

En programación, los elementos que pueden almacenar información se llaman variables. Dependiendo del tipo de información (números, letras, etc.) que se almacena se usan variables de un tipo o de otro. Una característica que diferencia unos lenguajes de otros es la rigidez o permisividad con respecto a los tipos de variables.

- Los **lenguajes tipificados** (también llamados de tipado estático) son aquellos en los que una variable guarda siempre un mismo tipo de datos. En algunos lenguajes tipificados se exige al programador que declare el tipo de cada variable y en otros lo determina el compilador. En algunos lenguajes tipificados se permite la conversión entre tipos de variables y en otros no.
- Los **lenguajes no tipificados** (también llamados de tipado dinámico) no requieren la declaración de tipo de las variables y una misma variable puede almacenar valores de tipos distintos a lo largo de la ejecución del programa.

Algunas ventajas de los lenguajes tipificados es que permiten detectar errores de tipo (cuando un dato de un tipo se guarda en una variable de otro tipo), que permiten a los compiladores optimizar mejor el código ejecutable y, en el caso de exigir la declaración de tipo de las variables, que facilitan la comprensión de los programas.

Algunas ventajas de los lenguajes no tipificados es la mayor flexibilidad de los programas, y una serie de características que aunque no se derivan necesariamente de la falta de tipificación suelen presentar los lenguajes no tipificados: simplicidad, mayor número de tipos de datos, metaprogramación (introspección, eval, etc.).

QUÉ ES PHP

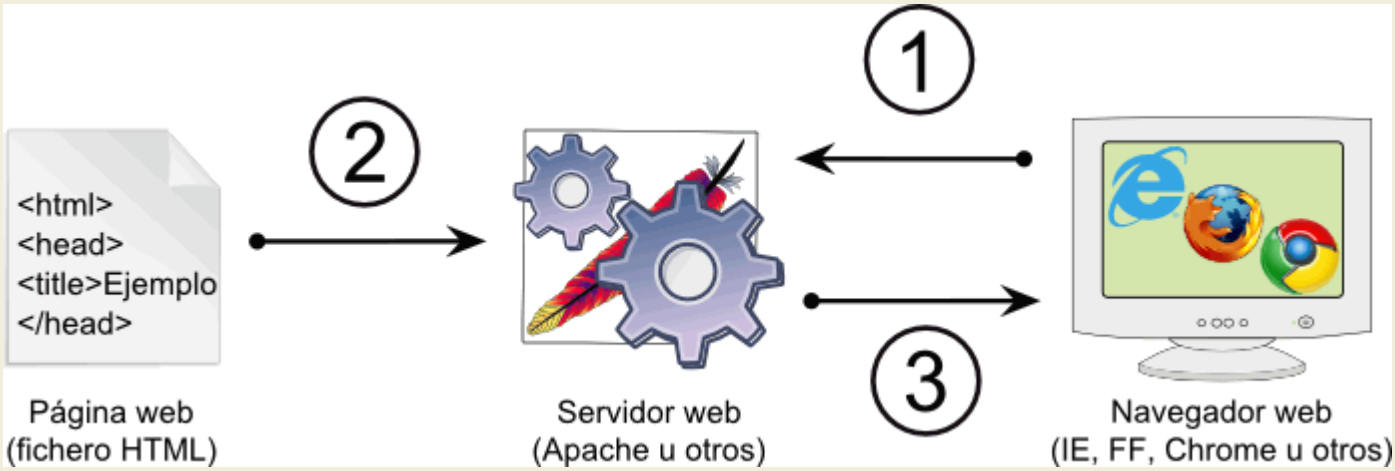
PHP es un lenguaje de programación dirigido a la creación de páginas web. Es un lenguaje de programación procedural, interpretado y no tipificado, con una sintaxis similar a la del lenguaje C, aunque actualmente puede utilizarse una sintaxis de programación orientada a objetos similar a la de Java.

PÁGINAS WEB, LENGUAJES DE PROGRAMACIÓN Y BASES DE DATOS

La World Wide Web (o simplemente, la web) fue ideada por Tim Berners Lee en 1990, empezó a funcionar en el CERN en 1991 y se extendió rápidamente por las Universidades del mundo (en aquel entonces Internet era una red a la que sólo tenían acceso entidades gubernamentales, particularmente entidades educativas). En 1992 algunos proveedores comerciales empezaron a dar acceso a particulares y empresas, lo que convirtió la web en una red de comunicación universal que ha transformado nuestras vidas.

En sus primeros años, las páginas web solían ser documentos de texto guardado en algún directorio de un servidor y a las que se accedía mediante los primeros navegadores web. Cada página web que se veía en el navegador correspondía a un fichero en el servidor.

La imagen siguiente ilustra de forma simplificada el esquema de funcionamiento:



- El usuario escribe la dirección de la página web en su navegador
- (1) El navegador la solicita al servidor web correspondiente (este paso requiere la participación de máquinas intermedias que no se comentan aquí)
- (2) El servidor lee el fichero que corresponde a esa página web
- (3) El servidor envía el fichero al navegador
- El navegador muestra la página web al usuario

Este esquema de funcionamiento es suficiente para sitios web pequeños creados por una sola persona, pero en cuanto un sitio web empieza a crecer, empiezan a surgir los problemas. Por ejemplo:

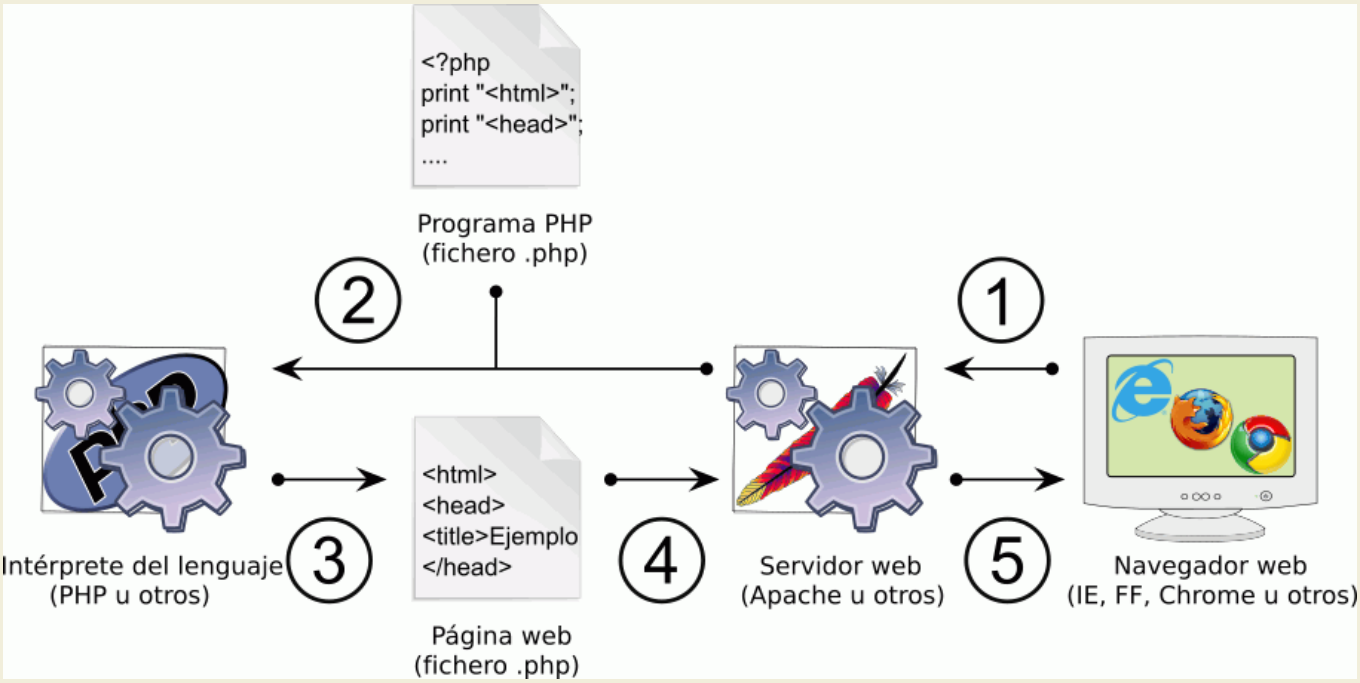
- si el sitio contiene muchas páginas es necesario crear menús que permitan orientarse por el sitio. Como cada página debe contener el menú, cualquier cambio en el menú obliga a modificar todas las páginas.
- si el sitio modifica a menudo su contenido (por ejemplo, la web de un periódico), tener que editar manualmente los ficheros ralentiza el proceso.
- si el sitio es creado por varias personas, cualquiera puede modificar por error los ficheros de otras personas y si varias personas quieren modificar el mismo fichero se pueden producir conflictos.
- si se quiere permitir la participación del público (foros, comentarios en blogs, etc.), esta participación depende de que uno de los autores modifique los ficheros, lo que ralentiza el proceso.
- si personas sin conocimientos técnicos quieren participar en la creación del sitio, pueden cometer errores editando los ficheros.

La solución es que las páginas no sean ficheros estáticos guardados en el disco, sino que las páginas se generen automáticamente cuando el usuario las solicite.

Por ejemplo, para resolver el problema de los menús comentado anteriormente, una solución sería que el menú estuviera en un fichero aparte y cuando el usuario solicitara una página, el menú se añadiera al principio de cada página (nota: no me refiero a la composición de páginas mediante frames, una solución desaconsejada hace muchos años, sino a componer la página web a partir de varios ficheros).

Esa generación de las páginas se puede hacer de varias maneras. Una de ellas es recurrir a lenguajes de programación generales o específicos (como PHP). Desde sus inicios, los servidores web permiten recurrir a lenguajes de programación para generar las páginas web.

La imagen siguiente ilustra de forma simplificada el esquema de funcionamiento:



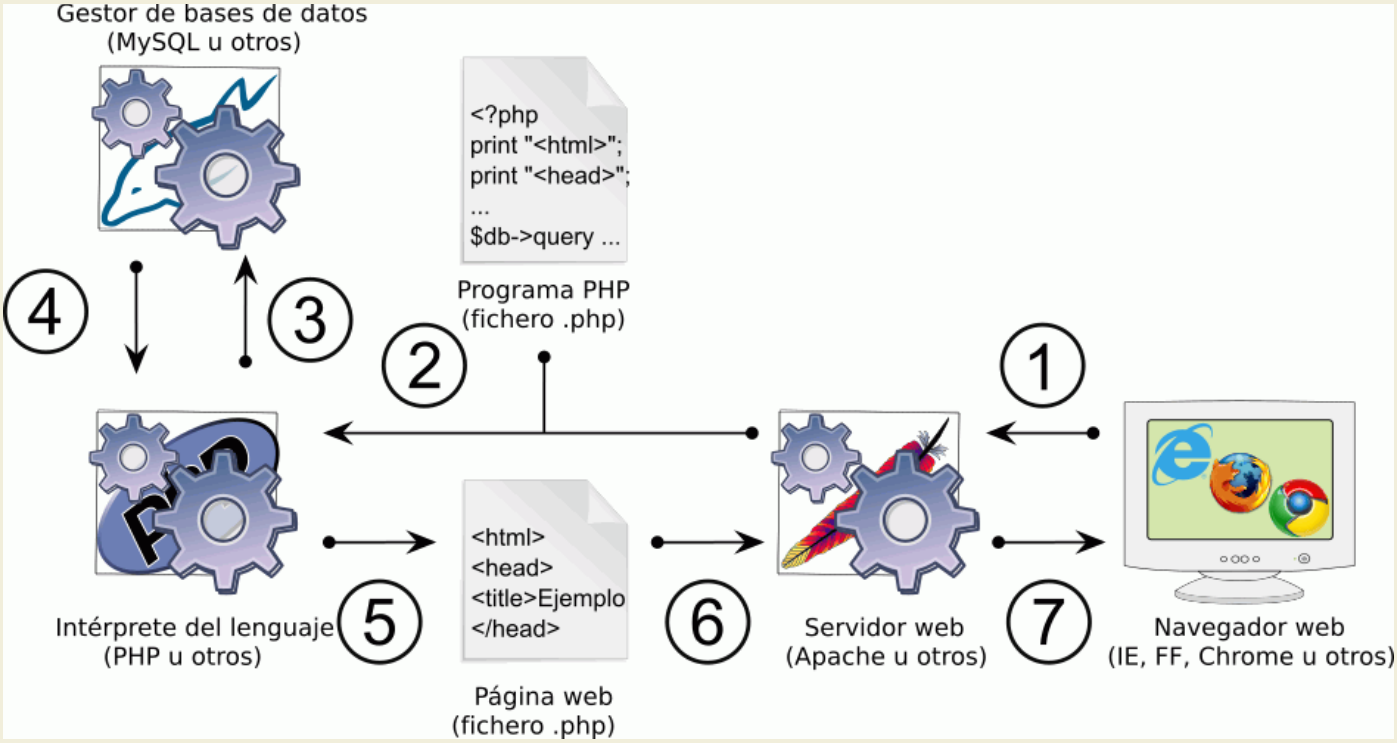
- El usuario escribe la dirección de la página web en su navegador
- (1) El navegador la solicita al servidor web correspondiente (este paso requiere la participación de máquinas intermedias que no se comentan aquí)
- (2) El servidor detecta que es un programa PHP y lo envía al intérprete del lenguaje
- (3) El intérprete del lenguaje completa la ejecución del programa.
- (4) El resultado final del programa (por ejemplo, el código fuente de una página web) se envía al servidor
- (5) El servidor envía el fichero al navegador
- El navegador muestra la página web al usuario

Es importante señalar que el usuario no puede saber si la página web estaba guardada en el disco duro o se ha generado en ese momento, de la misma manera que no puede saber qué lenguaje de programación que ha generado el documento. El navegador recibe el documento en ambos un documento de texto que contiene etiquetas HTML y lo muestra al usuario.

A veces, el usuario puede suponer que la página que se le está sirviendo se ha generado mediante PHP ya que la dirección de la página termina por .php en vez del habitual .HTML, aunque no se puede estar seguro de que sea realmente así.

Pero el uso de lenguajes de programación no suele ser suficiente. Si la información está diseminada en multitud de trozos para organizar y acceder fácilmente a toda esa información, es conveniente utilizar algún sistema gestor de bases de datos. Existen muchos sistemas gestores de bases de datos y los lenguajes de programación pueden conectarse a ellas y realizar consultas.

La imagen siguiente ilustra de forma simplificada el esquema de funcionamiento:



- El usuario escribe la dirección de la página web en su navegador
- (1) El navegador la solicita al servidor web correspondiente (este paso requiere la participación de máquinas intermedias que no se comentan aquí)
- (2) El servidor detecta que es un programa PHP y lo envía al intérprete del lenguaje
- (3) El intérprete del lenguaje ejecuta el programa. Si el programa incluye consultas a la base de datos, estas se realizan.
- (4) La base de datos entrega al intérprete el resultado de las consultas
- (5) El intérprete del lenguaje completa la ejecución del programa.
- (6) El resultado final del programa (por ejemplo, el código fuente de una página web) se envía al servidor
- (7) El servidor envía el fichero al navegador
- El navegador muestra la página web al usuario

Como antes, el usuario no puede saber si se ha accedido o no a un sistema gestor de bases de datos. El navegador recibe el documento en ambos un documento de texto que contiene etiquetas HTML y lo muestra al usuario.

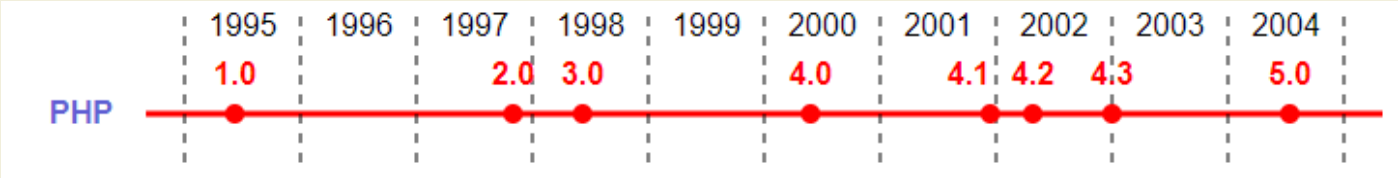
HISTORIA DE PHP

En el caso de PHP no hay ningún organismo oficial encargado de la definición del lenguaje PHP (definición que posteriormente distintas empresas implementan en diferentes compiladores o intérpretes), sino que el lenguaje evoluciona a la vez que lo hace el intérprete "oficial" de PHP publicado, desde la versión 4, por la empresa Zend Technologies. Existen otros motores, como HHVM creado por Facebook, pero son minoritarios. Aunque el lenguaje y el intérprete son cosas conceptualmente distintas, es habitual referirse a ambos como PHP.

PHP (el intérprete) ha sido publicado siempre como software libre, con una licencia llamada licencia PHP. Esta licencia es una licencia libre sin copyleft. Como la licencia impone restricciones al uso del término PHP en productos derivados, esta licencia es incompatible con la licencia GPL, por lo que la Free Software Foundation recomienda que sólo se utilice para programar extensiones del propio PHP.

PHP fue creado por Rasmus Lerdorf en junio de 1995. Las siglas PHP significaban entonces **Personal Home Page** y hacían referencia a que era un lenguaje diseñado para facilitar la generación de páginas web, en primer lugar en el sitio web personal de Rasmus Lerdorf y rápidamente en muchos sitios. Desde entonces, PHP no ha dejado de evolucionar, proporcionando nuevas características.

En los dibujos siguientes, las versiones obsoletas de PHP se muestran en color rojo.



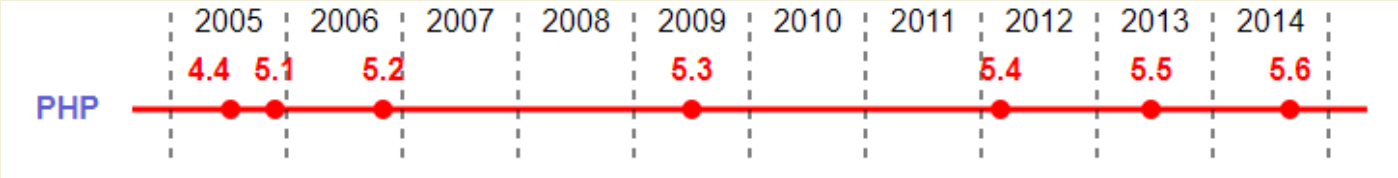
En noviembre de 1997 Rasmus Lerdorf publicó una segunda versión del lenguaje, PHP 2.

En junio de 1998 se publicó PHP 3. Esta versión fue creada por Zeev Suraski y Andi Gutmans, que desde entonces dirigen el desarrollo de PHP (con la colaboración de Lerdorf). A partir de esta versión PHP significa **PHP: Hypertext Preprocessor** (añadiéndose a la lista de acrónimos recursivos como GNU, Lame, Wine, RPM, etc.). La característica más decisiva de esta versión fue la mejora de la extensibilidad del lenguaje, permitiendo que muchos programadores aportaran nuevos módulos, pero esta versión también simplificó la sintaxis del lenguaje e introdujo la posibilidad de utilizar una sintaxis orientada a objetos.

En 1999 Zeev Suraski y Andi Gutmans crearon la empresa Zend Technologies, que desarrolla productos basados en PHP (servidores, editores, etc.).

En mayo de 2000 se publicó PHP 4, que incluía un nuevo motor llamado Zend Engine. Esta versión era bastante más rápida que la anterior y añade nuevas características como las sesiones.

En julio de 2004 se publicó PHP 5, que incluía un nuevo motor llamado Zend Engine 2. La característica más importante de esta versión fue la mejora de la programación orientada a objetos.

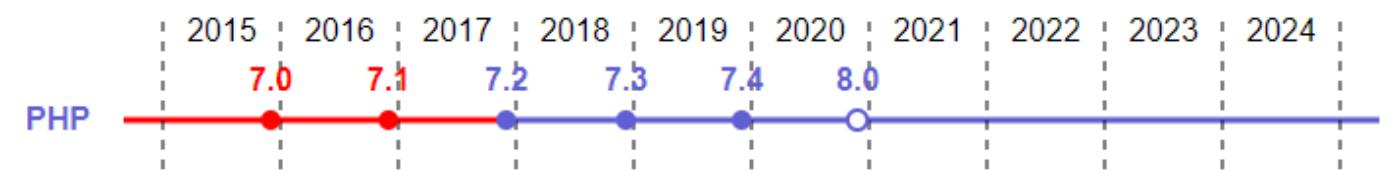


La siguiente versión prevista, que se tendría que haber publicado como PHP 6, se fijó el objetivo de dar a PHP soporte completo de Unicode, el juego de caracteres universal. Pero ese objetivo resultó ser mucho más difícil de conseguir de lo esperado y en 2010 se tiró la toalla.

Desde 2012, se fueron publicando anualmente versiones 5.X que fueron introduciendo novedades en el lenguaje (5.3 en junio de 2009, 5.4 en marzo de 2012, 5.5 en junio de 2013 y 5.6 en agosto de 2014).

En enero de 2014 se inició el desarrollo de PHP-NG, un nuevo motor de PHP enfocado a aumentar la velocidad del lenguaje. Los resultados fueron tan prometedores que en agosto de 2014 PHP-NG se convirtió en la base de la siguiente versión de PHP, que se decidió llamar PHP 7 para evitar la confusión con la frustrada versión PHP 6. PHP 7 se publicó en diciembre de 2015 y el motor PHP-NG se rebautizó como Zend Engine 3.

En su momento se propuso publicar una versión PHP 5.7, de transición entre PHP 5.6 y PHP 7, pero la propuesta fue descartada.



En octubre de 2015, la empresa RogueWave Software compró Zend, pero en un primer momento la mantuvo como empresa independiente. Al principio, esta compra no afectó al desarrollo de PHP. Zeev Suraski siguió trabajando en Zend, pero Andi Gutmans abandonó la empresa para pasar a Amazon Web Services. En octubre de 2018, Suraski anunció que RogueWave iba a dejar de desarrollar Zend Engine y Zend Framework y por tanto Suraski y su equipo abandonaría la empresa. En enero de 2019 Rogue Software fue comprada por Perforce Software, una empresa que comercializa soluciones de desarrollo de software, pero los nuevos propietarios no han anunciado planes específicos para Zend. Actualmente (noviembre de 2019), desconozco bajo qué forma continuará el desarrollo de Zend Engine. Al menos, Zend Framework parece haber encontrado cobijo en la Linux Foundation y cambiará su nombre a Laminas.

En cualquier caso, en PHP 7 la cadencia anual se ha mantenido. Así, PHP 7.1 se publicó en diciembre de 2016, PHP 7.2 en noviembre de 2017, PHP 7.3 en diciembre de 2018 y PHP 7.4 en noviembre de 2019. La próxima versión será PHP 8.0 que se espera publicar a finales de noviembre de 2020.

Durante el período de mantenimiento de una versión de PHP, se publican mensualmente actualizaciones de seguridad (que se numeran con una tercera cifra: 5.6.X, 7.0.X, 7.1.X, 7.2.X, etc.). PHP 5.6, por ser la última versión 5, se ha mantenido hasta el 31 de diciembre de 2018, pero a partir de PHP 7.0, las versiones se mantendrán solamente durante tres años a partir de su publicación. Por eso, PHP 7.0 dejó de mantenerse el 4 de diciembre de 2018 y PHP 7.1 el 1 de diciembre de 2019.

Desde sus inicios, PHP ha sido posiblemente el lenguaje más utilizado en entornos de desarrollo web, y desde 2001 está situado en el Top 10 del índice Tiobe de lenguajes de programación.

Para construir sitios web profesionales y no tener que escribir todo desde cero, se suelen utilizar frameworks de programación en PHP o, si no se quiere programar, se suelen utilizar CMS (Content Management System, Sistema de gestión de contenidos). Muchos de los CMS más populares están escritos en PHP.

CRÍTICAS A PHP

Como cualquier lenguaje de programación, es fácil encontrar en la web hay opiniones muy críticas hacia PHP, sobre todo en referencia a la seguridad. Al leer estas críticas hay que tener en cuenta que:

- PHP ha evolucionado mucho. Muchos defectos del lenguaje han sido corregidos en versiones posteriores, por lo que conviene comprobar si las opiniones que se leen no son opiniones antiguas que ya no tienen sentido. Por el mismo motivo, hay que tener cuidado cuando se leen manuales o tutoriales antiguos, porque pueden estar recomendando técnicas de programación obsoletas.
- PHP es extremadamente popular. Muchos programas de PHP han sido escritos por programadores que intentan compensar con su entusiasmo su falta de conocimientos, pero PHP no es un lenguaje intrínsecamente malo o inseguro.

PALABRAS RESERVADAS (KEYWORDS)

Las palabras reservadas (*keywords*) de un lenguaje de programación son las palabras propias del lenguaje. En general, estas palabras no se pueden utilizar para denominar a los elementos que puede crear el usuario (variables, funciones, clases, etc.). Los lenguajes de programación suelen tener pocas palabras reservadas y la mayoría de los programas utiliza un número muy reducido.

Las palabras reservadas de PHP son las siguientes (se han marcado en negrita las que se utilizan en este curso):

`__halt_compiler()`	`abstract`	**`and`**	**`array()`**	**`as`**
`Break`	`callable`	`case`	**`catch`**	`class`
`clone`	**`const`**	`continue`	`declare`	`default`
`die()`	**`do`**	`echo`	**`else`**	**`elseif`**
`empty()`	`enddeclare`	`endfor`	`endforeach`	`endif`
`endswitch`	`endwhile`	`eval()`	**`exit()`**	`extends`
`final`	`finally`	**`for`**	**`foreach`**	**`function`**
`global`	`goto`	**`if`**	`implements`	**`include`**
`include_once`	`instanceof`	`insteadof`	`interface`	**`isset()`**
`list()`	`namespace`	**`new`**	**`or`**	**`print`**
`private`	`protected`	`public`	**`require_once`**	**`require`**
`return`	`static`	`switch`	`throw`	`trait`
`try`	**`unset()`**	`use`	`var`	**`while`**

INSTALACIÓN Y USO DE XAMPP

¿QUÉ ES XAMPP?

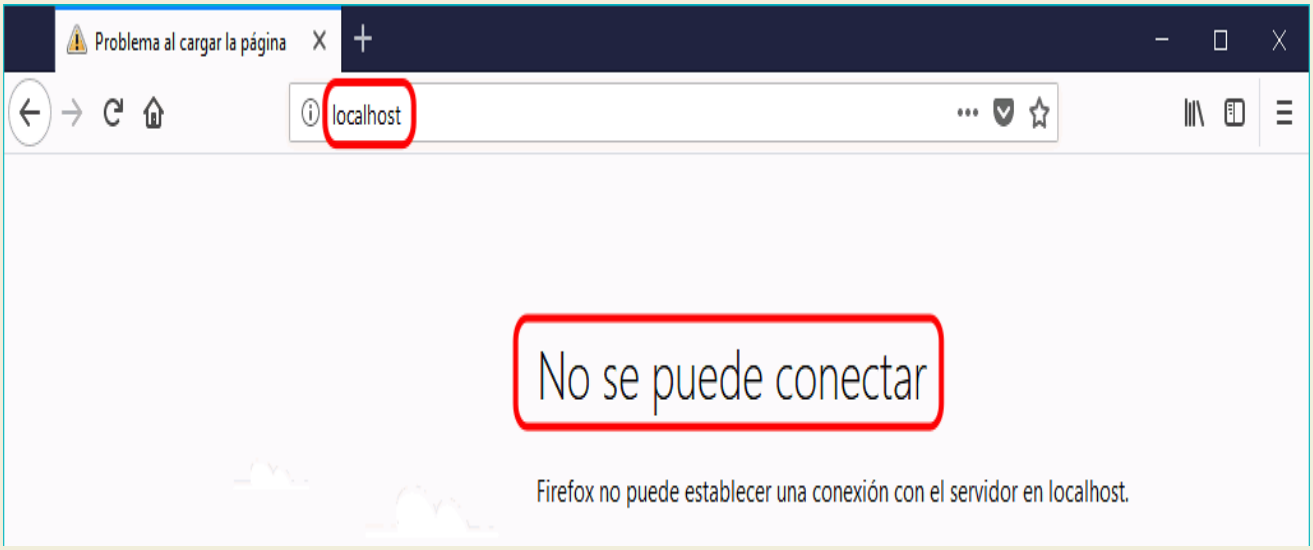
XAMPP es una distribución de Apache que **incluye varios software libres**. El nombre es un acrónimo compuesto por las iniciales de los programas que lo constituyen: el servidor web **A**pache, los sistemas relacionales de administración de bases de datos **M**ySQL y **M**ariaDB, así como los lenguajes de programación **P**erl y **P**HP. La inicial **X** se usa para representar a los sistemas operativos Linux, Windows y Mac OS X.

- **Apache**: el servidor web de código abierto es la aplicación más usada globalmente para la entrega de contenidos web. Las aplicaciones del servidor son ofrecidas como software libre por la Apache Software Foundation.
- **MySQL/MariaDB:** con MySQL, XAMPP cuenta con uno de los sistemas relacionales de gestión de bases de datos más populares del mundo. En combinación con el servidor web Apache y el lenguaje PHP, MySQL sirve para el almacenamiento de datos para servicios web. En las versiones actuales de XAMPP esta base de datos se ha sustituido por MariaDB, una ramificación ("Fork") del proyecto MySQL.
- **PHP:** es un lenguaje de programación de código de lado del servidor que permite crear páginas web o aplicaciones dinámicas. Es independiente de plataforma y soporta varios sistemas de bases de datos.
- **Perl**: este lenguaje de programación se usa en la administración del sistema, en el desarrollo web y en la programación de red. También permite programar aplicaciones web dinámicas.

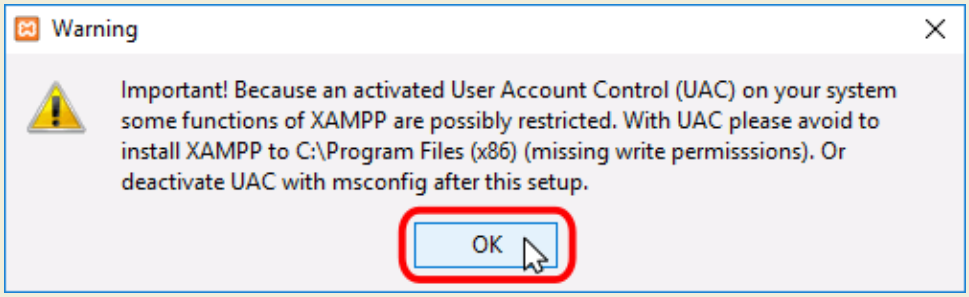
Además de estos componentes principales, esta distribución gratuita también incluye, según el sistema operativo, otras herramientas como el servidor de correo **Mercury**, el programa de administración de bases de datos **phpMyAdmin**, el software de analítica web **Webalizer**, **OpenSSL**, **Apache Tomcat** y los servidores FTP **FileZilla** o **ProFTPd**.

INSTALAR XAMPP EN WINDOWS 7/10

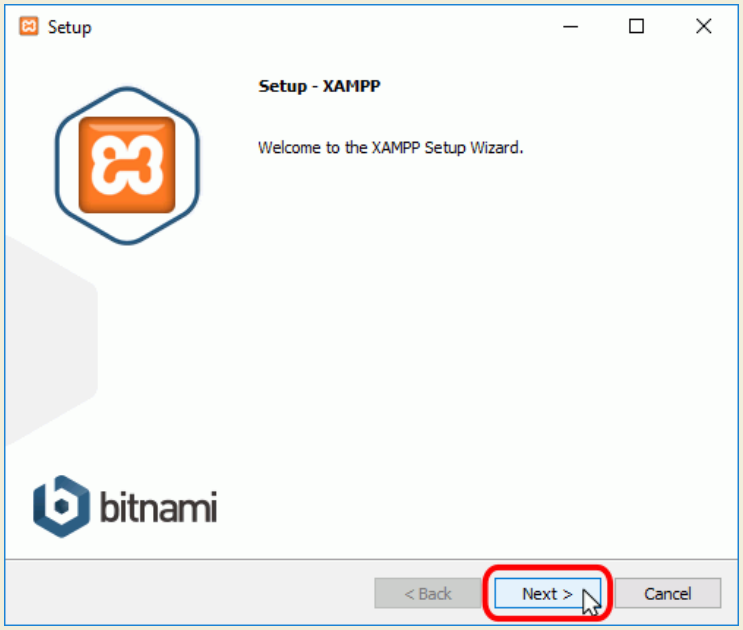
Antes de instalar un servidor de páginas web es conveniente comprobar si no hay ya uno instalado, o al menos si no está en funcionamiento. Para ello, es suficiente con abrir el navegador y escribir la dirección **http://localhost**. Si se obtiene un mensaje de error es que no hay ningún servidor de páginas web en funcionamiento (aunque podría haber algún servidor instalado, pero no estar en funcionamiento).



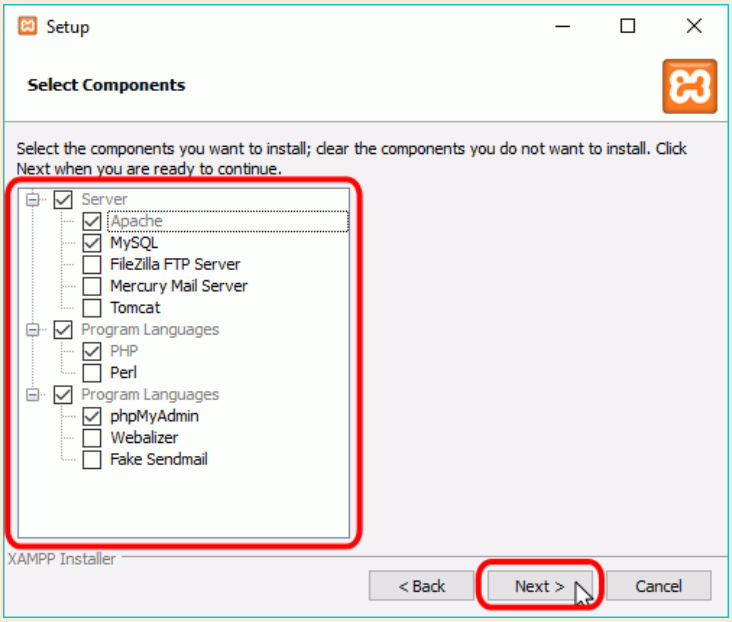
Una vez obtenido el archivo de instalación de XAMPP, hay que hacer doble clic sobre él para ponerlo en marcha. Al poner en marcha el instalador XAMPP nos muestra un aviso que aparece si está activado el Control de Cuentas de Usuario y recuerda que algunos directorios tienen permisos restringidos:



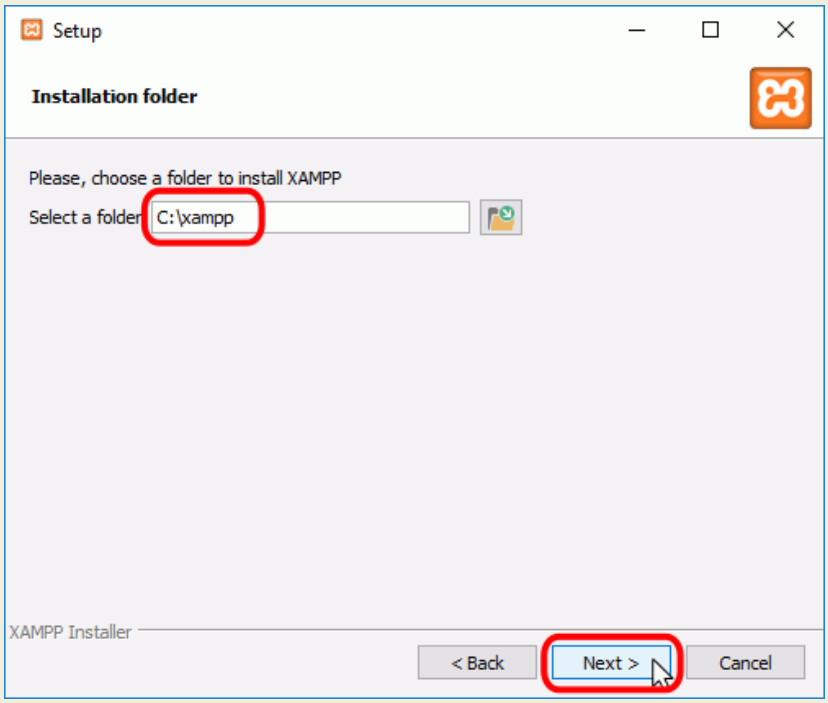
A continuación se inicia el asistente de instalación. Para continuar, haga clic en el botón "Next".



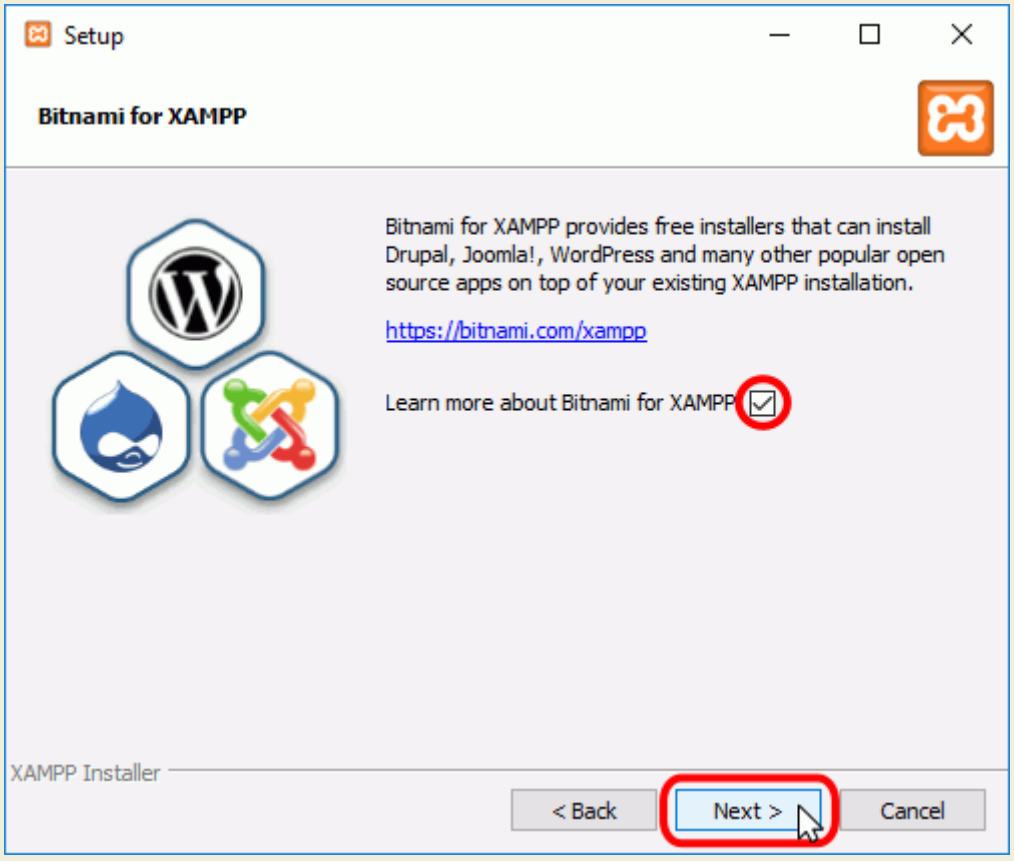
Los componentes mínimos que instala XAMPP son el servidor Apache y el lenguaje PHP, pero XAMPP también instala otros elementos. En la pantalla de selección de componentes puede elegir la instalación o no de estos componentes. Para seguir estos apuntes se necesita al menos instalar MySQL y phpMyAdmin.



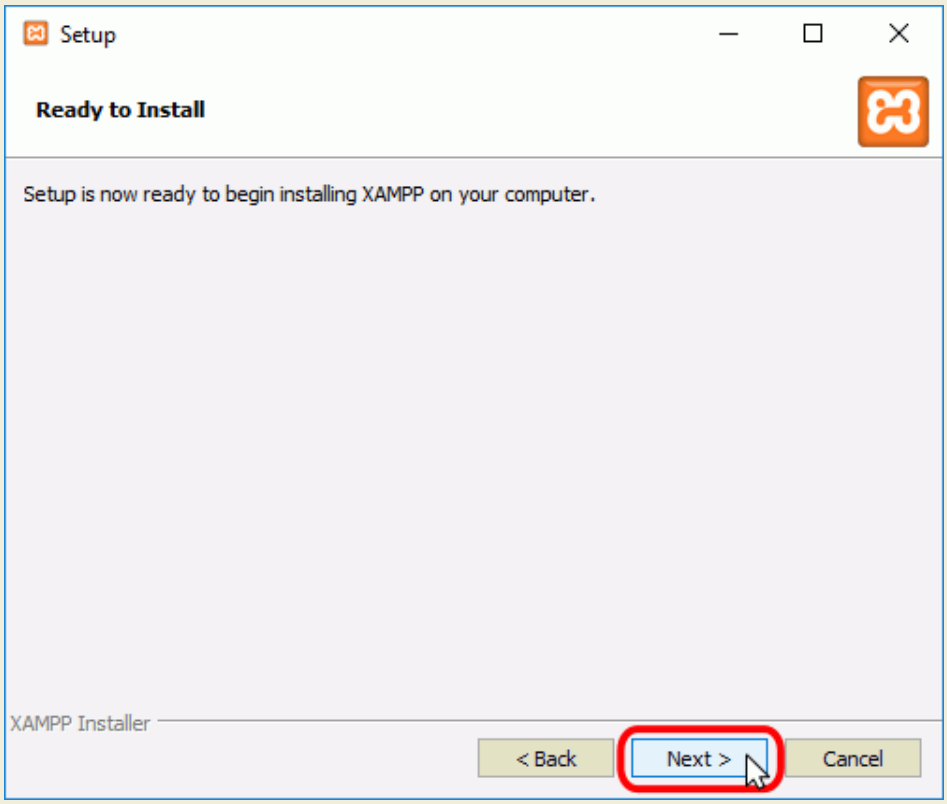
En la siguiente pantalla puede elegir la carpeta de instalación de XAMPP. La carpeta de instalación predeterminada es **C:\xampp**. Si quiere cambiarla, haga clic en el icono de carpeta y seleccione la carpeta donde quiere instalar XAMPP. Para continuar la configuración de la instalación, haga clic en el botón "Next".



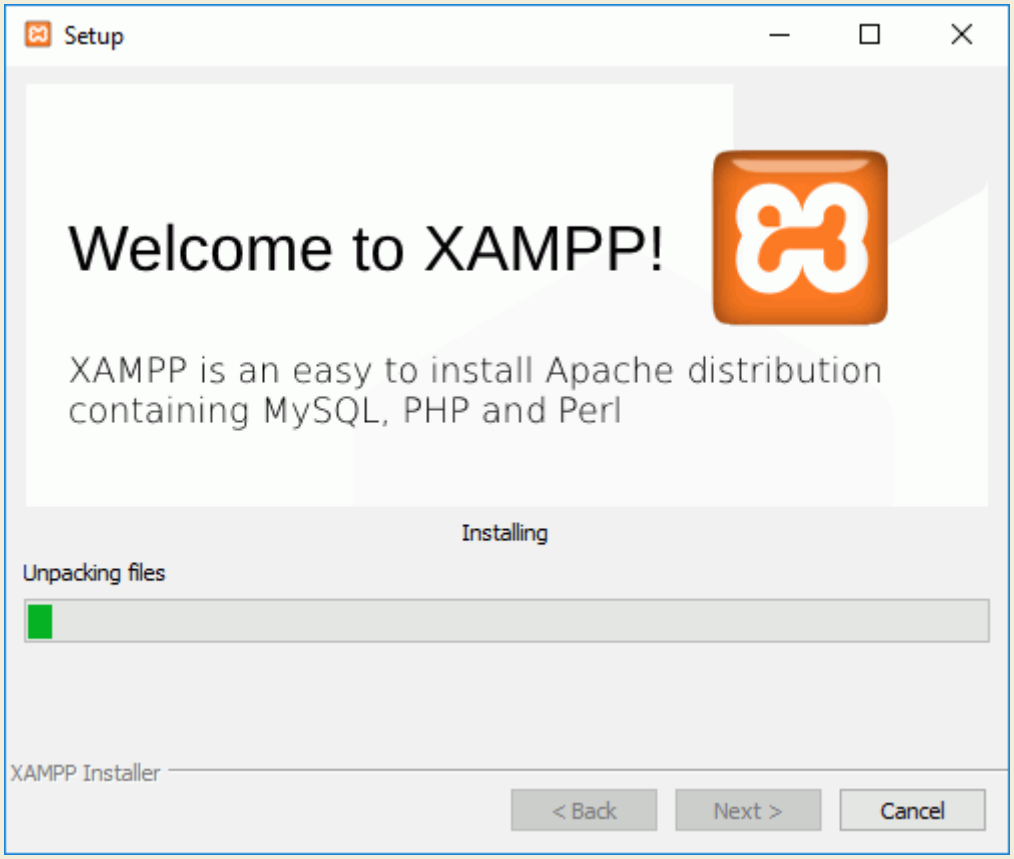
La siguiente pantalla ofrece información sobre los instaladores de aplicaciones para XAMPP creados por Bitnami. Haga clic en el botón "Next" para continuar. Si deja marcada la casilla, se abrirá una página web de Bitnami en el navegador.



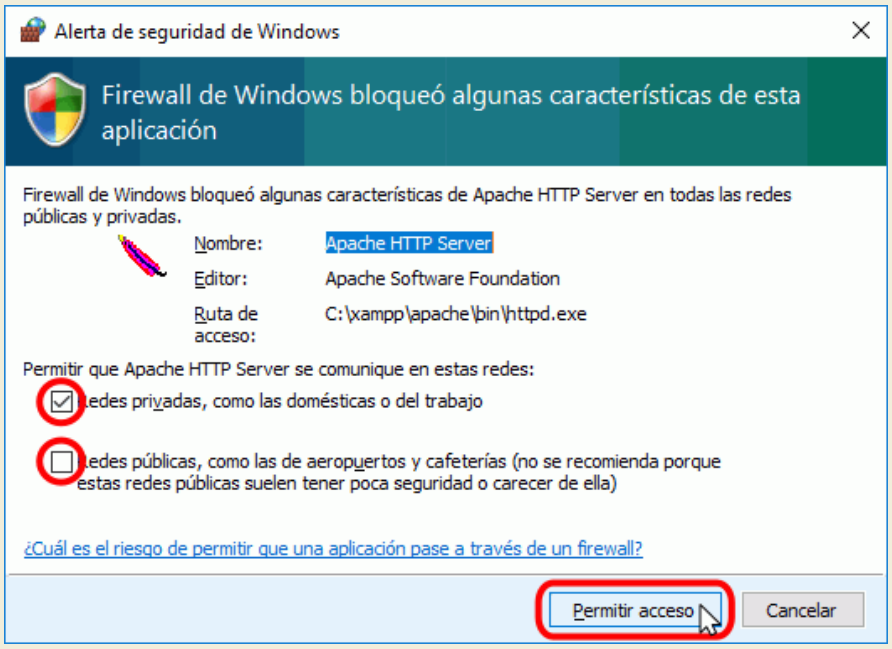
Una vez elegidas las opciones de instalación en las pantallas anteriores, esta pantalla es la pantalla de confirmación de la instalación. Haga clic en el botón "Next" para comenzar la instalación en el disco duro.



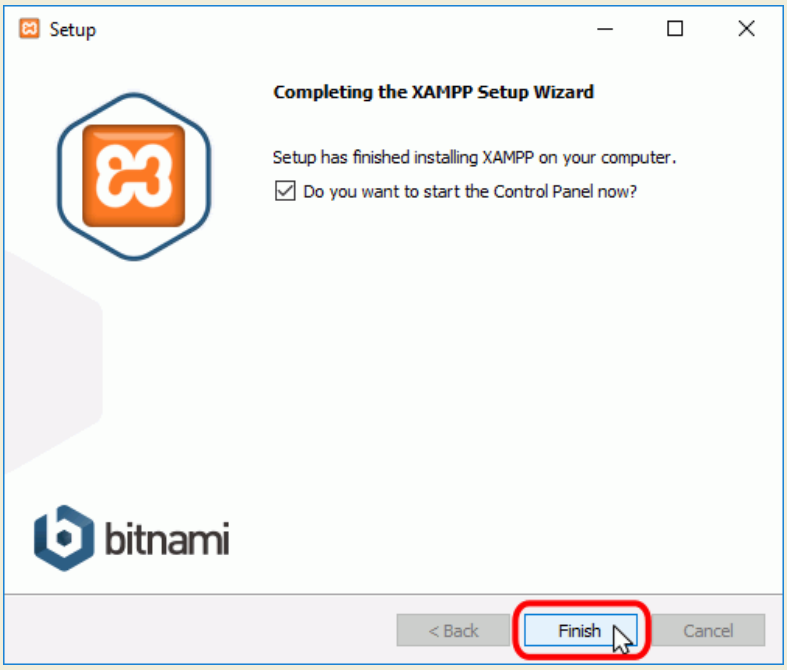
El proceso de copia de archivos puede durar unos minutos.



Durante la instalación, si en la computadora no se había instalado Apache anteriormente, en algún momento se mostrará un aviso del cortafuego de Windows para autorizar a Apache a comunicarse en las redes privadas o públicas. Una vez elegidas las opciones deseadas (en estos apuntes se recomienda permitir las redes privadas y denegar las redes públicas), haga clic en el botón "Permitir acceso".



Una vez terminada la copia de archivos, la pantalla final confirma que XAMPP ha sido instalado. Si se deja marcada la casilla, se abrirá el panel de control de XAMPP. Para cerrar el programa de instalación, haga clic en el botón "Finish".



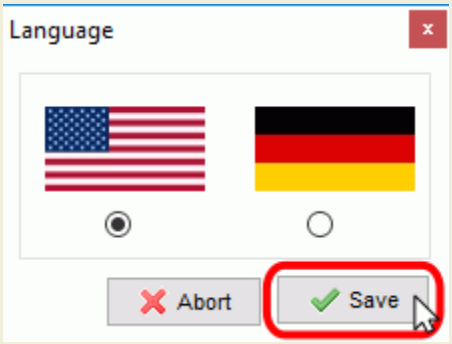
Parece ser que el instalador de XAMPP tiene un problema en Windows 10 y no crea la carpeta del menú inicio.

Por ello, una vez completada la instalación, compruebe si se ha creado la carpeta en el menú de inicio. Si no se ha creado, abra el explorador de archivos, abra el directorio de instalación de XAMPP (en principio, C:\xampp\), haga clic derecho sobre el programa xampp-control.exe y elija la opción "Anclar a inicio" o "Anclar a la barra de tareas". Se añadirá un icono al menú de inicio (o a la barra de tareas) que permite abrir el panel de control de XAMPP.

ABRIR Y CERRAR EL PANEL DE CONTROL

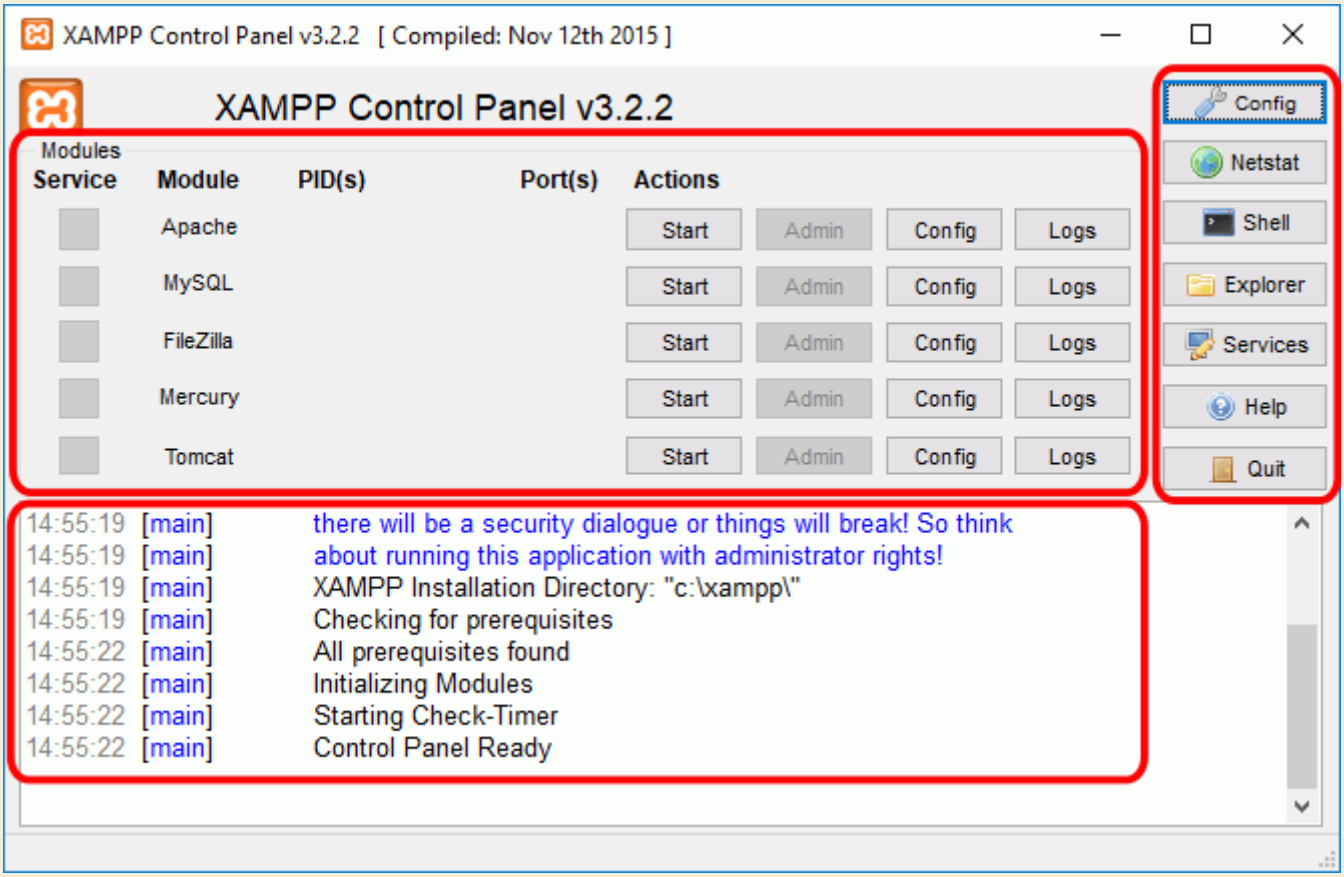
Al panel de control de XAMPP se puede acceder mediante el menú de inicio "Todos los programas > XAMPP > XAMPP Control Panel" o, si ya está iniciado, mediante el icono del área de notificación.

La primera vez que se abre el panel de control de XAMPP, se muestra una ventana de selección de idioma que permite elegir entre inglés y alemán.

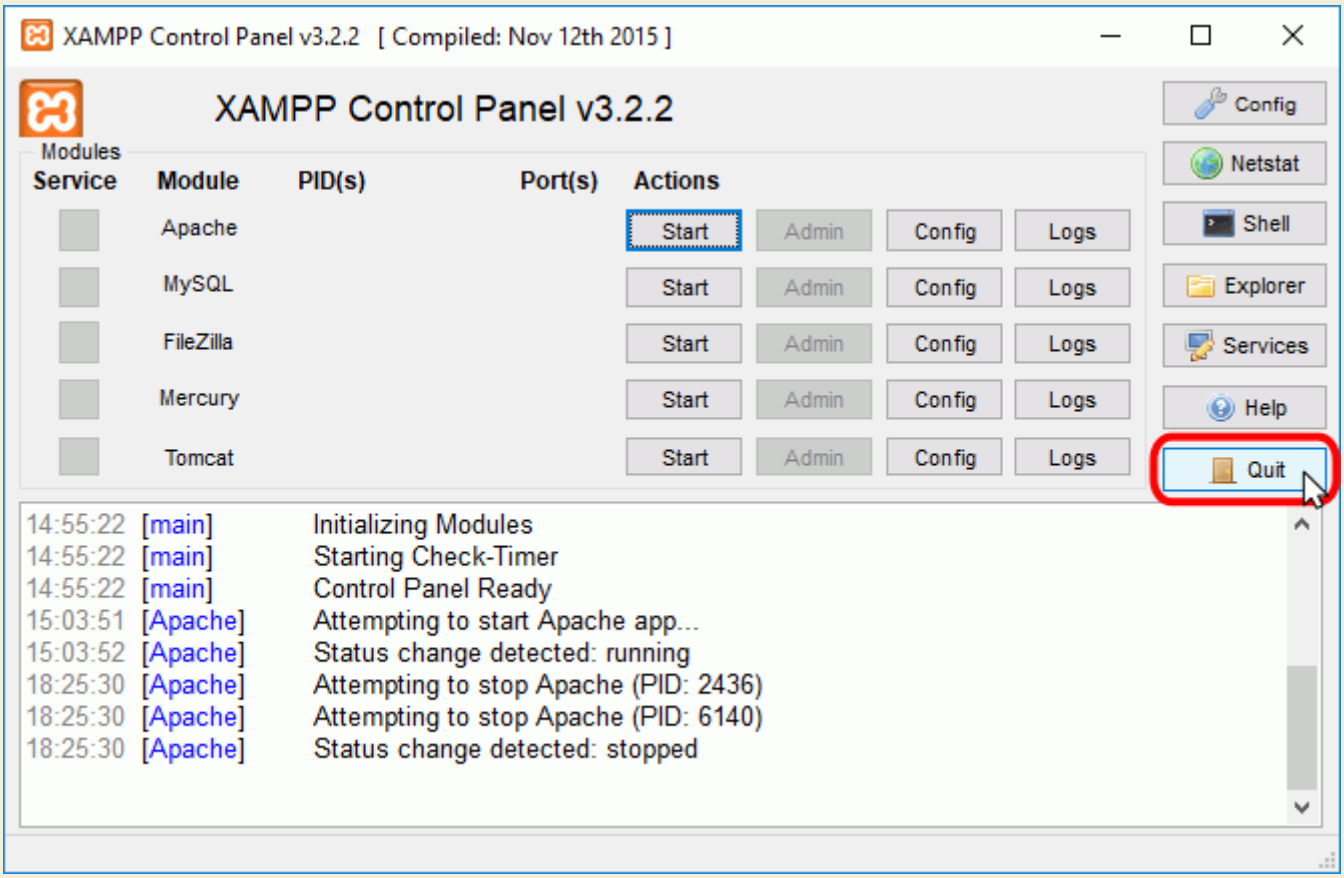


El panel de control de XAMPP se divide en tres zonas:

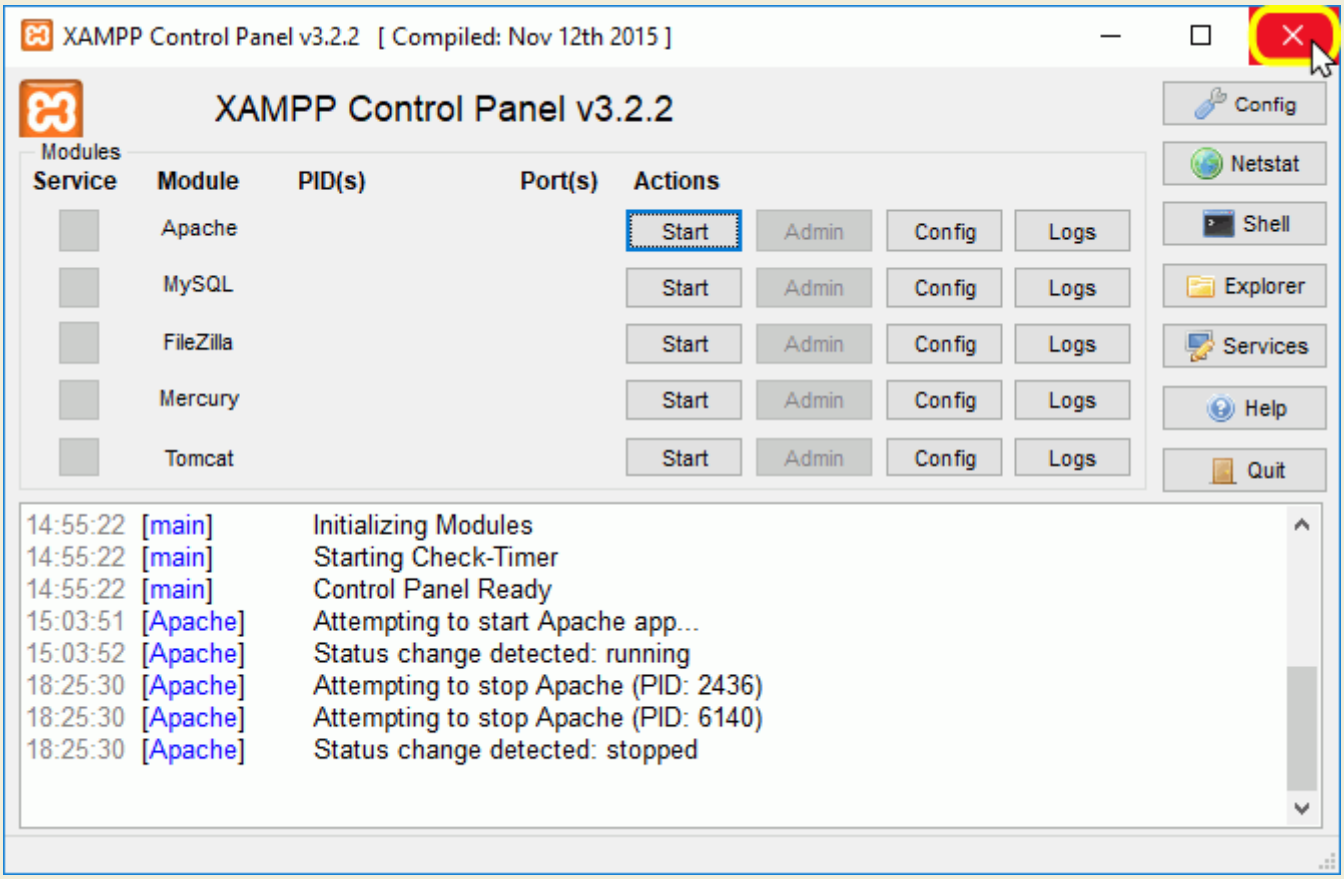
- la zona de módulos, que indica para cada uno de los módulos de XAMPP: si está instalado como servicio, su nombre, el identificador de proceso, el puerto utilizado e incluye unos botones para iniciar y detener los procesos, administrarlos, editar los archivos de configuración y abrir los archivos de registro de actividad.
- la zona de notificación, en la que XAMPP informa del éxito o fracaso de las acciones realizadas
- la zona de utilidades, para acceder rápidamente



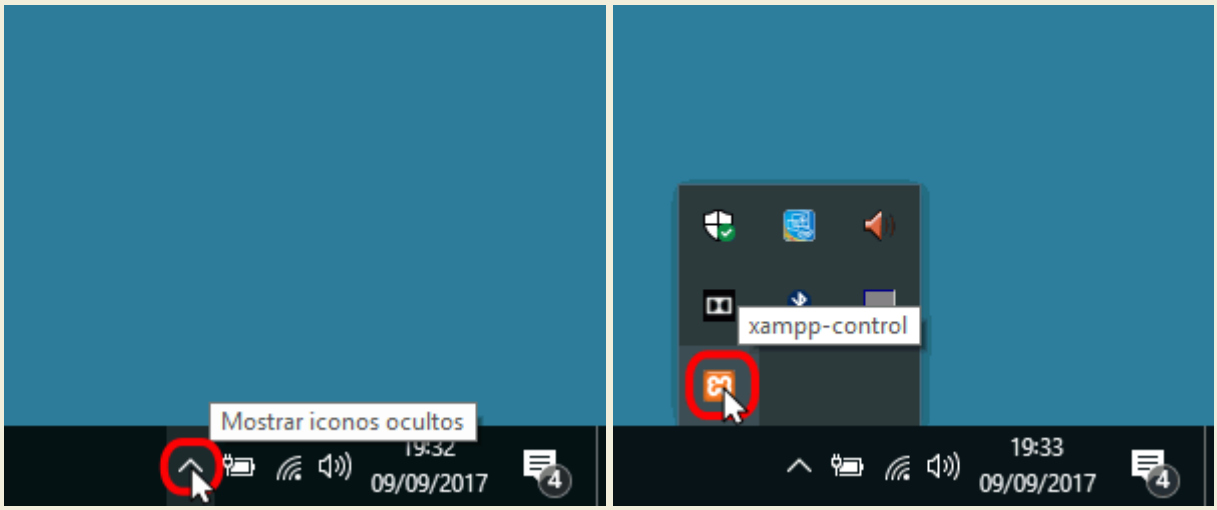
Para cerrar el panel de control de XAMPP hay que hacer clic en el botón Quit (al cerrar el panel de control no se detienen los servidores):



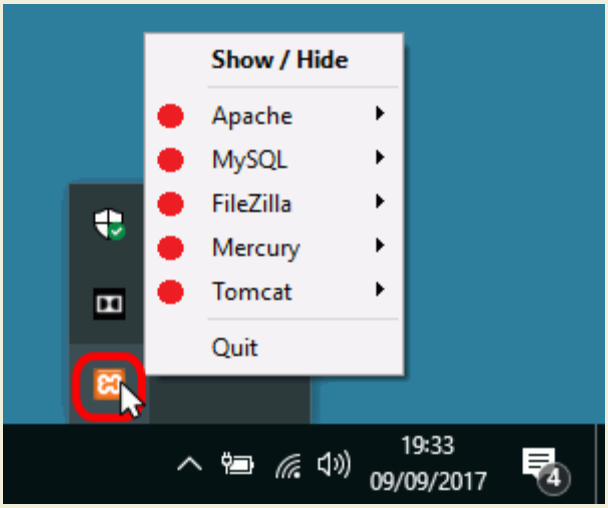
El botón Cerrar en forma de aspa no cierra realmente el panel de control, sólo lo minimiza:



Si se ha minimizado el panel de control de XAMPP, se puede volver a mostrar haciendo doble clic en el icono de XAMPP del área de notificación.



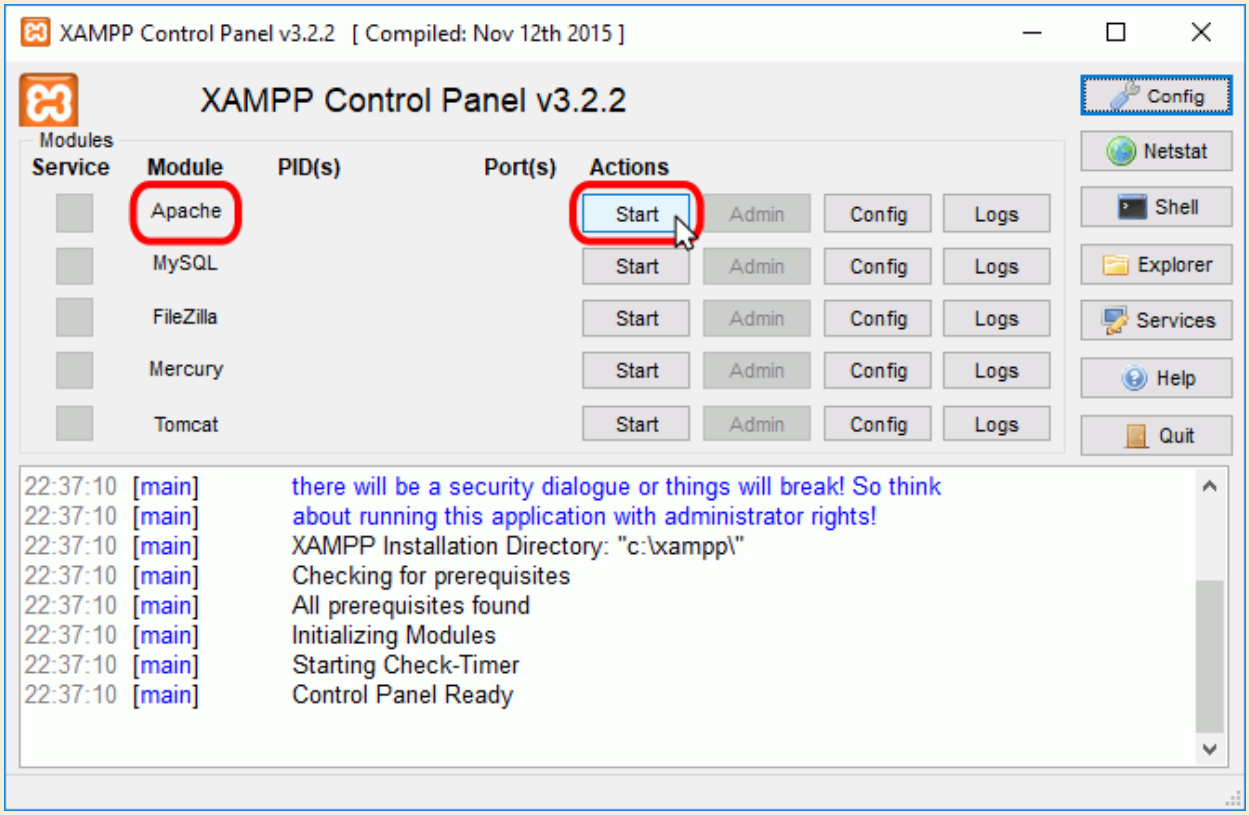
Haciendo clic derecho en el icono de XAMPP del área de notificación se muestra un menú que permite mostrar u ocultar el panel de control, arrancar o detener servidores o cerrar el panel de control.



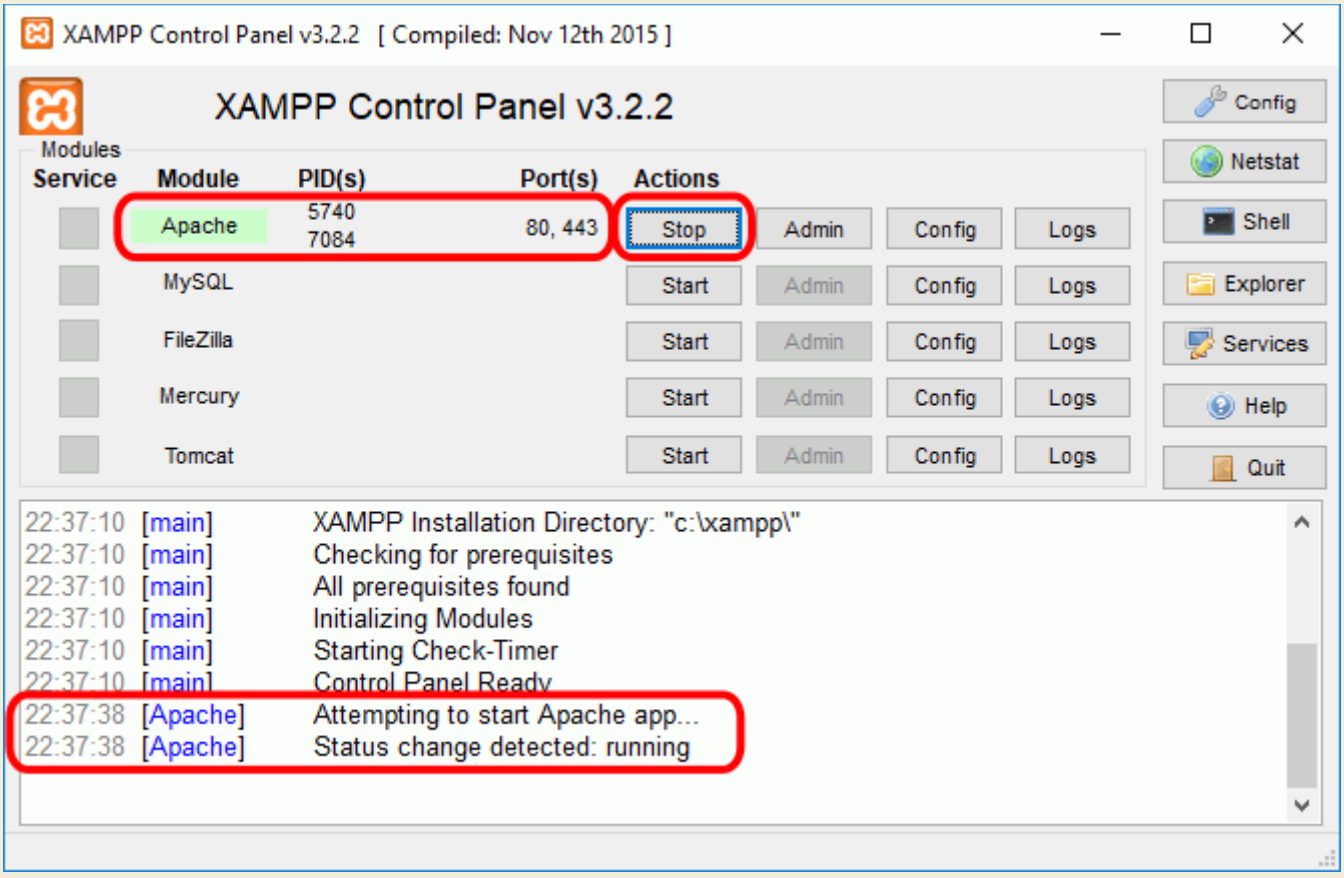
Se pueden abrir varios paneles de control simultáneamente y cualquiera de ellos puede iniciar o detener los servidores, pero no es aconsejable hacerlo ya que puede dar lugar a confusiones (por ejemplo, al detener un servidor desde un panel de control los otros paneles de control interpretan la detención como un fallo inesperado y muestran un mensaje de error).

INICIAR SERVIDORES

Para poner en funcionamiento Apache (u otro servidor), hay que hacer clic en el botón "Start" correspondiente:

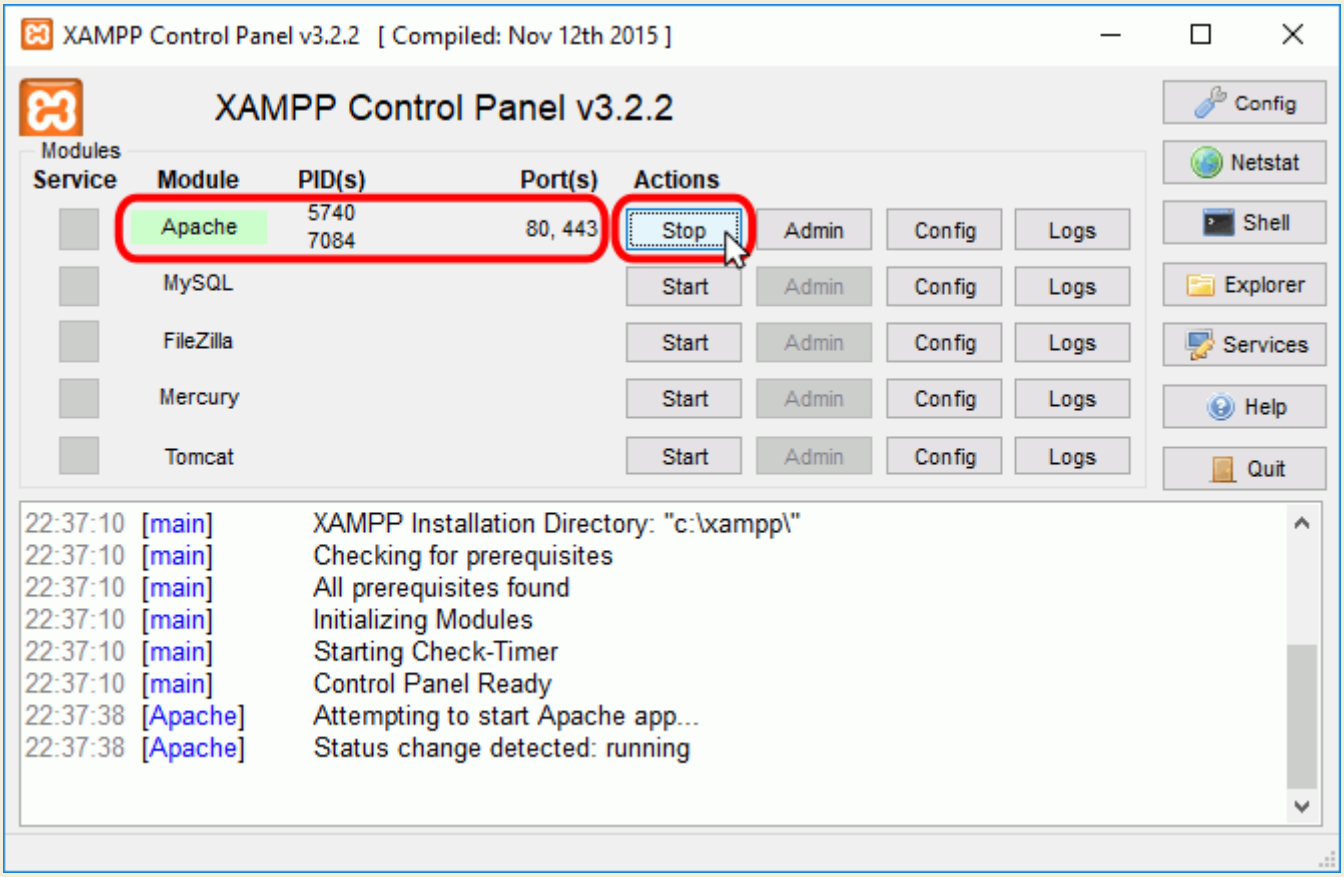


Si el arranque de Apache tiene éxito, el panel de control mostrará el nombre del módulo con fondo verde, su identificador de proceso, los puertos abiertos (http y https), el botón "Start" se convertirá en un botón "Stop" y en la zona de notificación se verá el resultado de las operaciones realizadas.

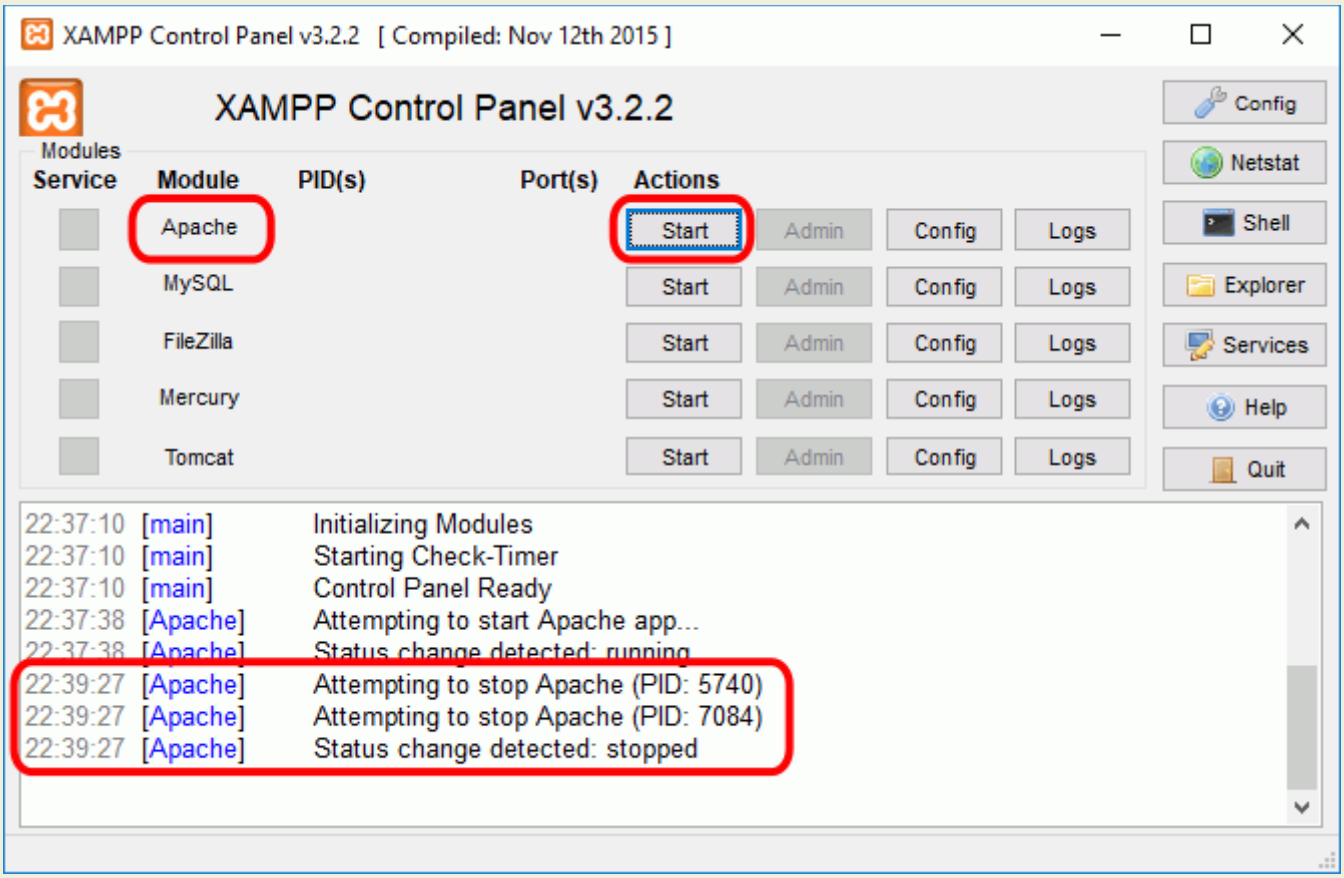


DETENER SERVIDORES

Para detener Apache (u otro servidor), hay que hacer clic en el botón "Stop" correspondiente a Apache.



Si la parada de Apache tiene éxito, el panel de control mostrará el nombre del módulo con fondo gris, sin identificador de proceso ni puertos abiertos (http y https), el botón "Stop" se convertirá en un botón "Start" y en la zona de notificación se verá el resultado de las operaciones realizadas.



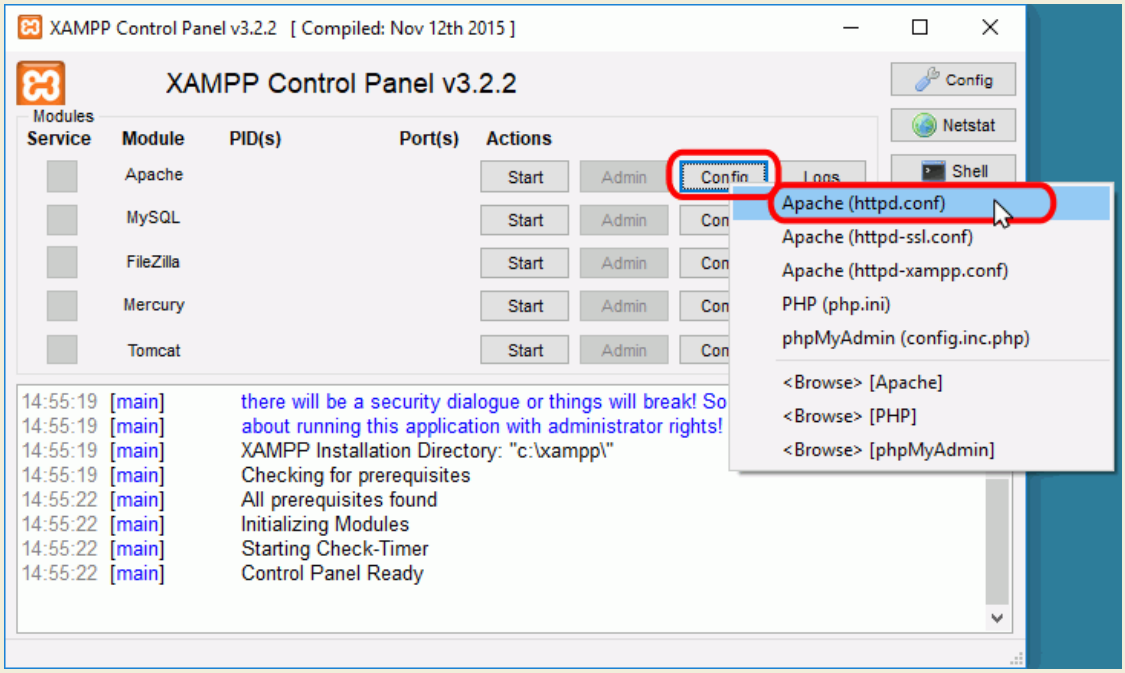
Para reiniciar de nuevo Apache habría que volver a hace clic en el botón "Start" correspondiente a Apache.

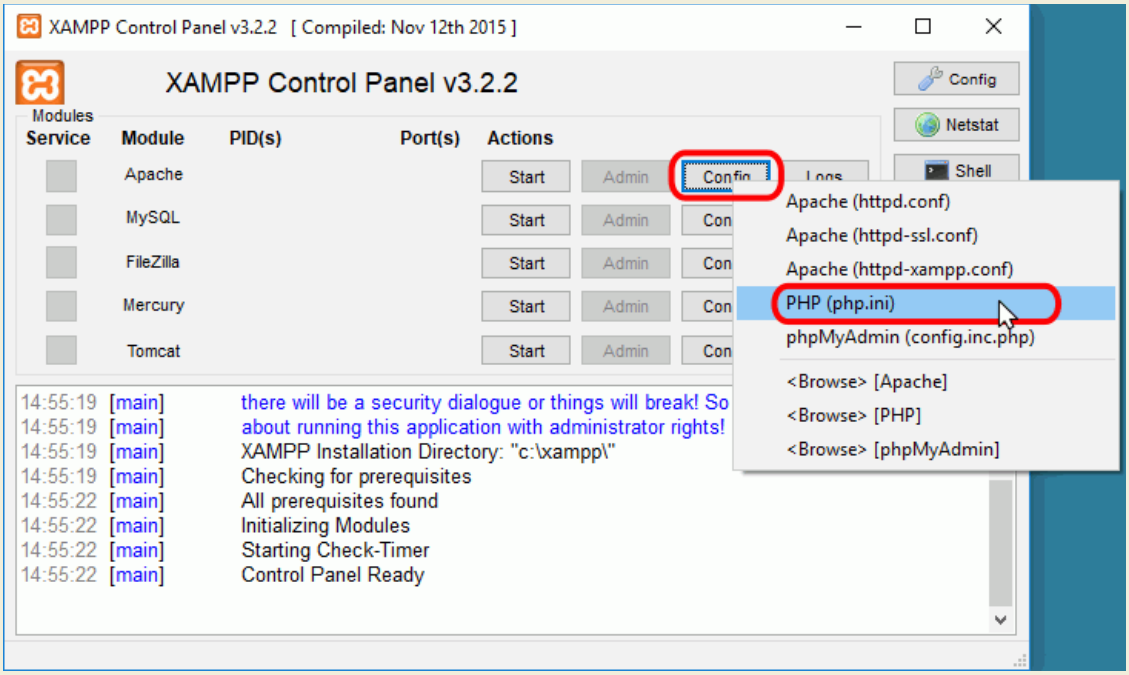
Nota:

- A veces es necesario detener y reiniciar los servidores. Por ejemplo, los archivos de configuración de Apache se cargan al iniciar Apache. Si se modifica un archivo de configuración de Apache (httpd.conf, php.ini u otro) mientras Apache está en marcha, para recargar los archivos de configuración es necesario detener y reiniciar el servidor Apache.
- Si al modificar el archivo de configuración hemos introducido errores, el servidor no será capaz de iniciarse. Si no sabemos encontrar el origen del problema, se recomienda restaurar los archivos de configuración originales, de los que se aconseja tener una copia de seguridad.

EDITAR ARCHIVOS DE CONFIGURACIÓN DE APACHE O PHP

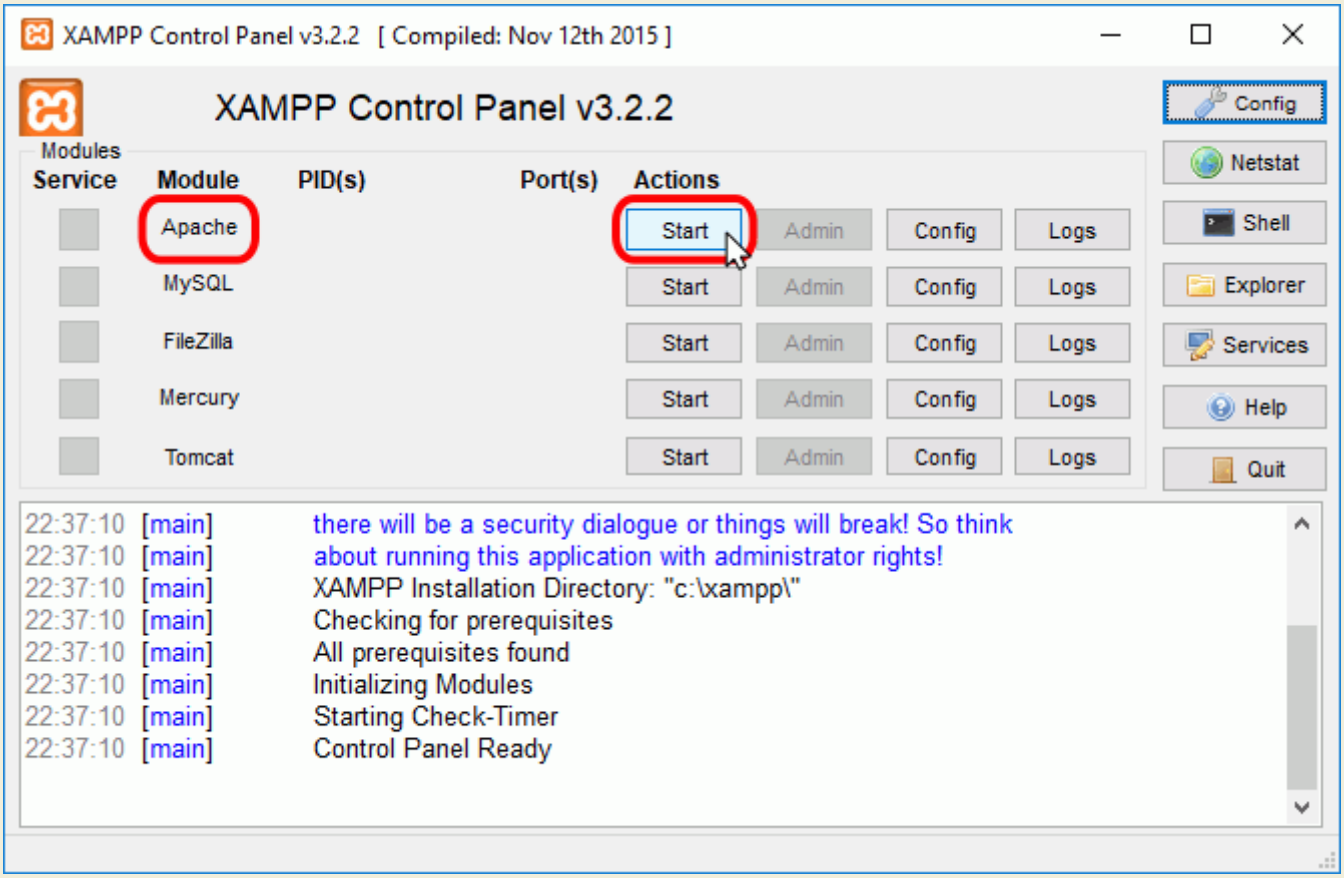
Los dos archivos principales de configuración son los archivos httpd.conf (Apache) y php.ini (PHP). Para editarlos se puede utilizar el panel de control de XAMPP, que los abre directamente en el bloc de notas. Para ello hay que hacer clic en el botón "Config" correspondiente a Apache y hacer clic en el archivo que se quiere editar.



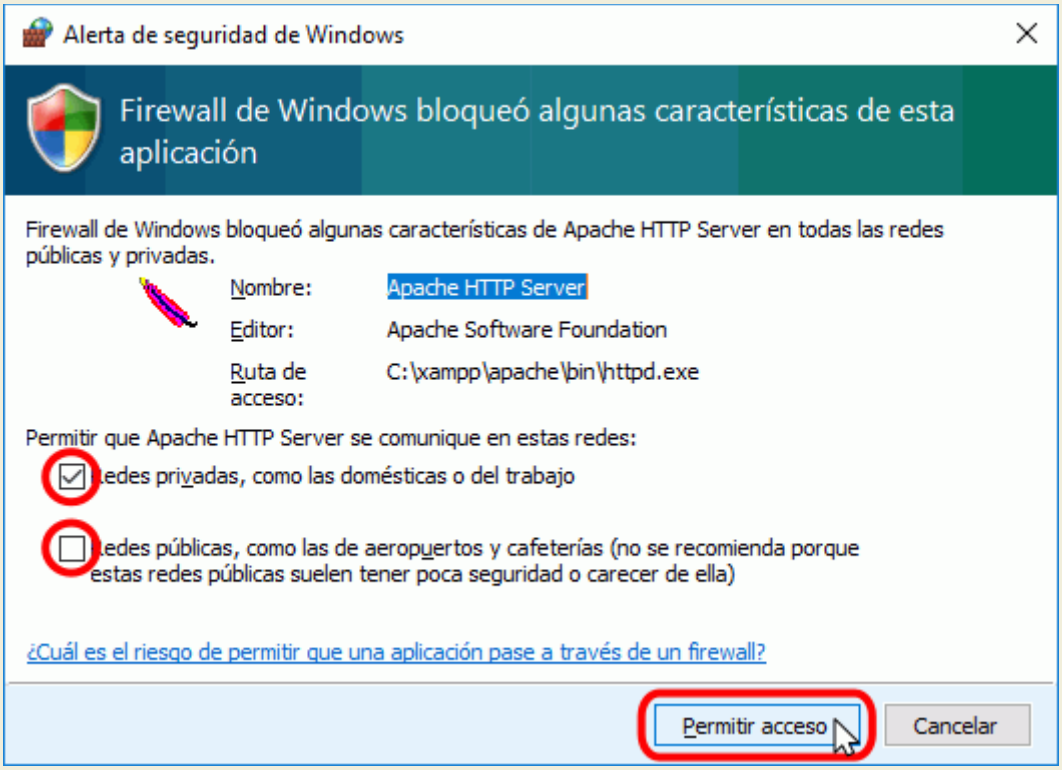


Cuando se pone en marcha por primera vez cualquiera de los servidores que instala XAMPP, el cortafuego de Windows pide al usuario confirmación de la autorización.

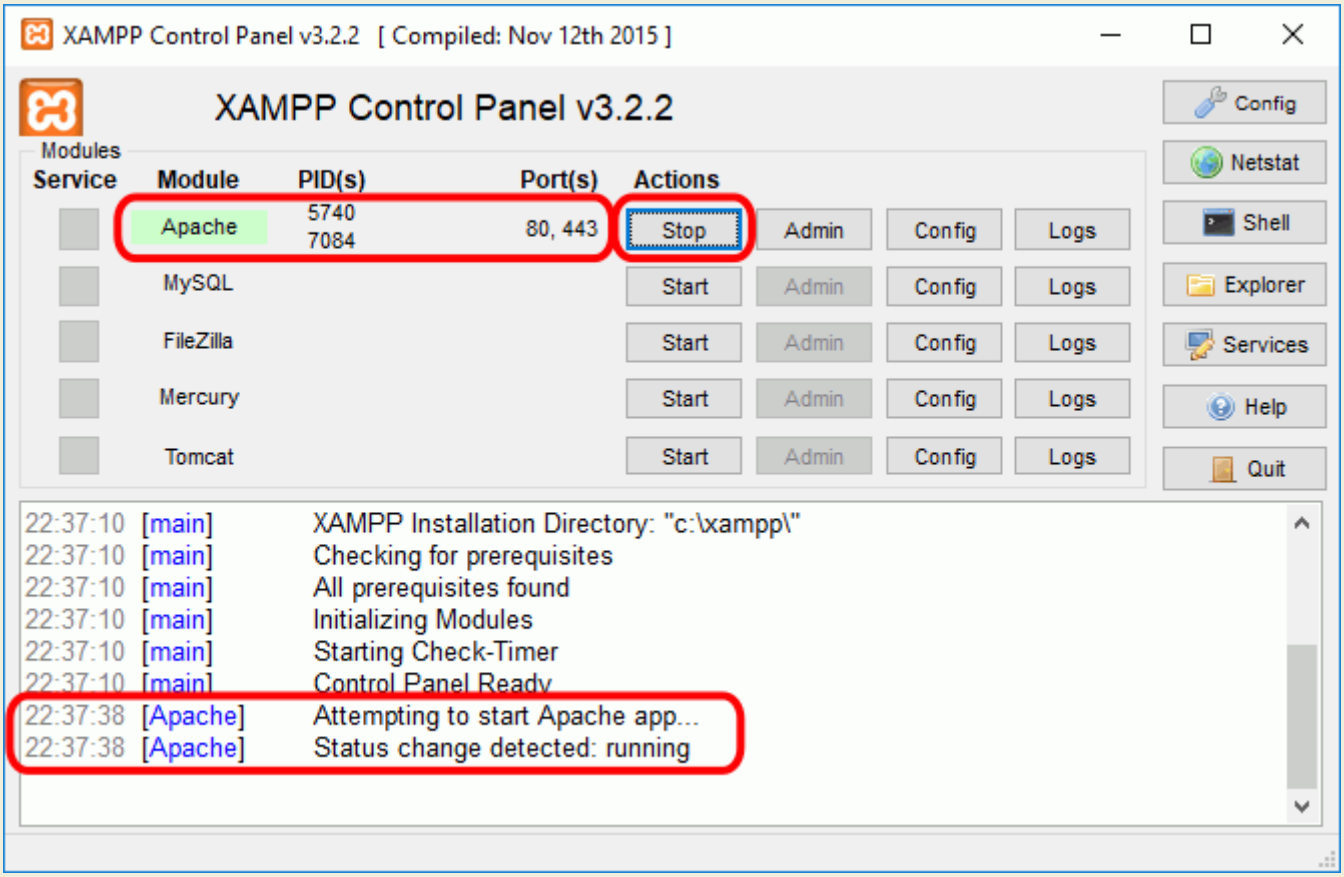
Por ejemplo, la primera vez que se pone en marcha Apache mediante el botón Start correspondiente...



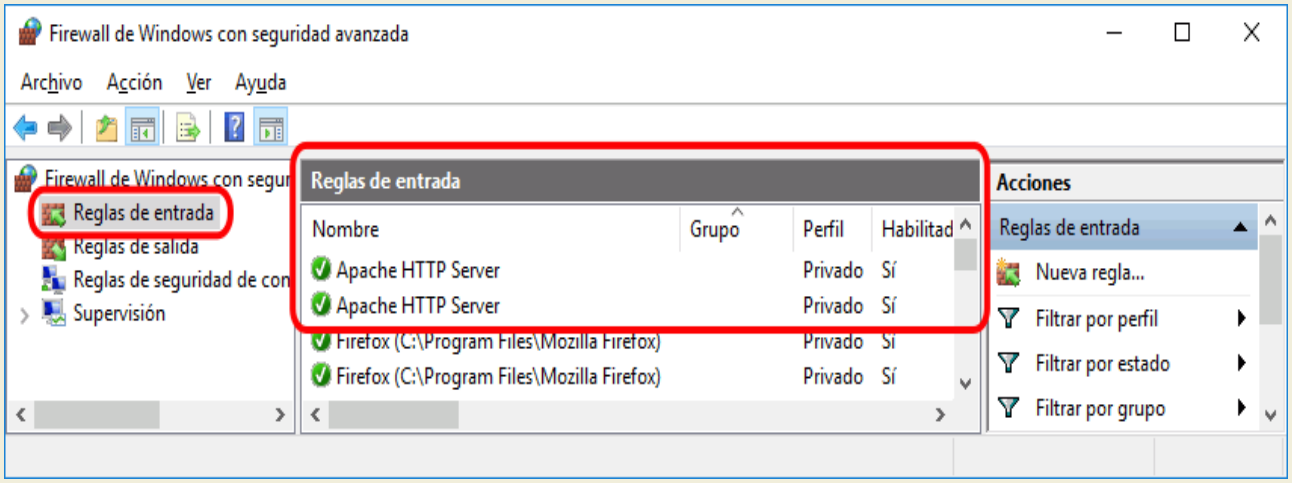
... como Apache abre puertos en la computadora (por primera vez), el cortafuego de Windows pide al usuario confirmación. Para poder utilizarlo hace falta al menos autorizar el acceso en redes privadas:



Si el arranque de Apache tiene éxito, el panel de control mostrará el nombre del módulo con fondo verde, su identificador de proceso, los puertos abiertos (http y https), el botón "Start" se convertirá en el botón "Stop" y en la zona de notificación se verá el resultado de las operaciones realizadas.



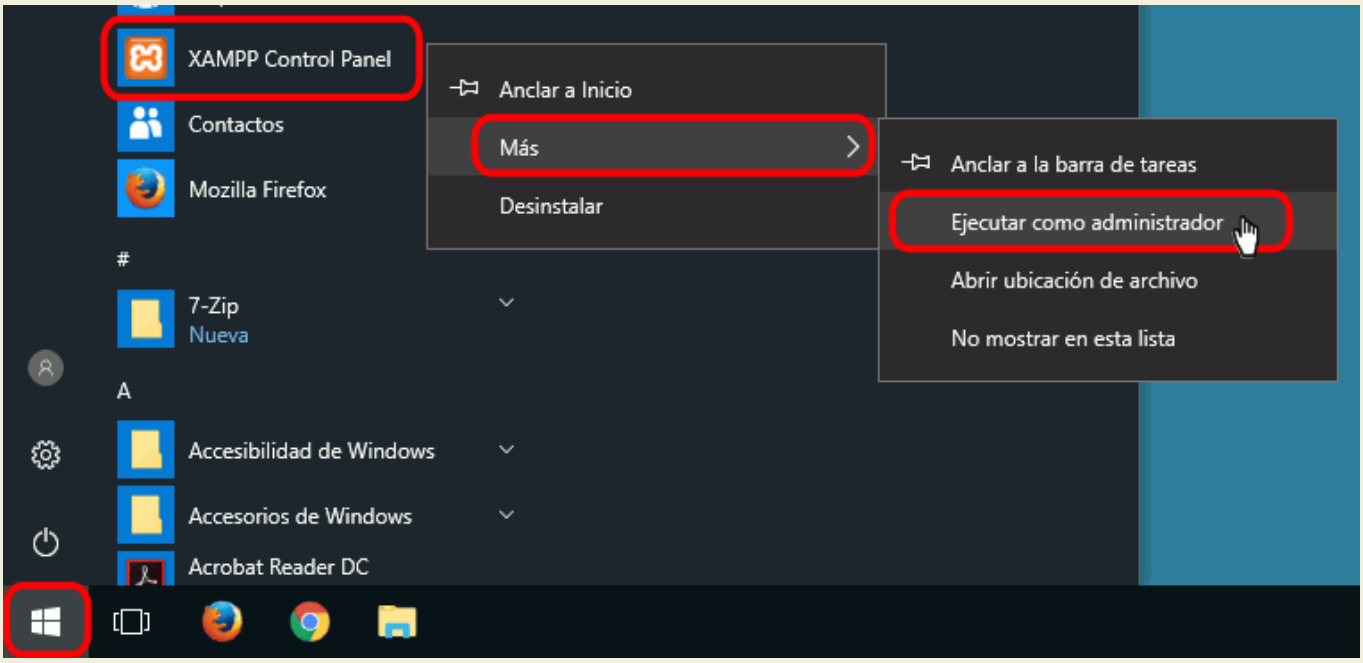
Si se abre el programa "Firewall de Windows con seguridad avanzada", en el apartado de Reglas de entrada se puede ver las nuevas reglas añadidas.



EJECUTAR EL PANEL DE CONTROL COMO ADMINISTRADOR

En algunas situaciones es necesario ejecutar el panel de control como administrador, por ejemplo, para configurar los servidores como servicios o deshabilitarlos.

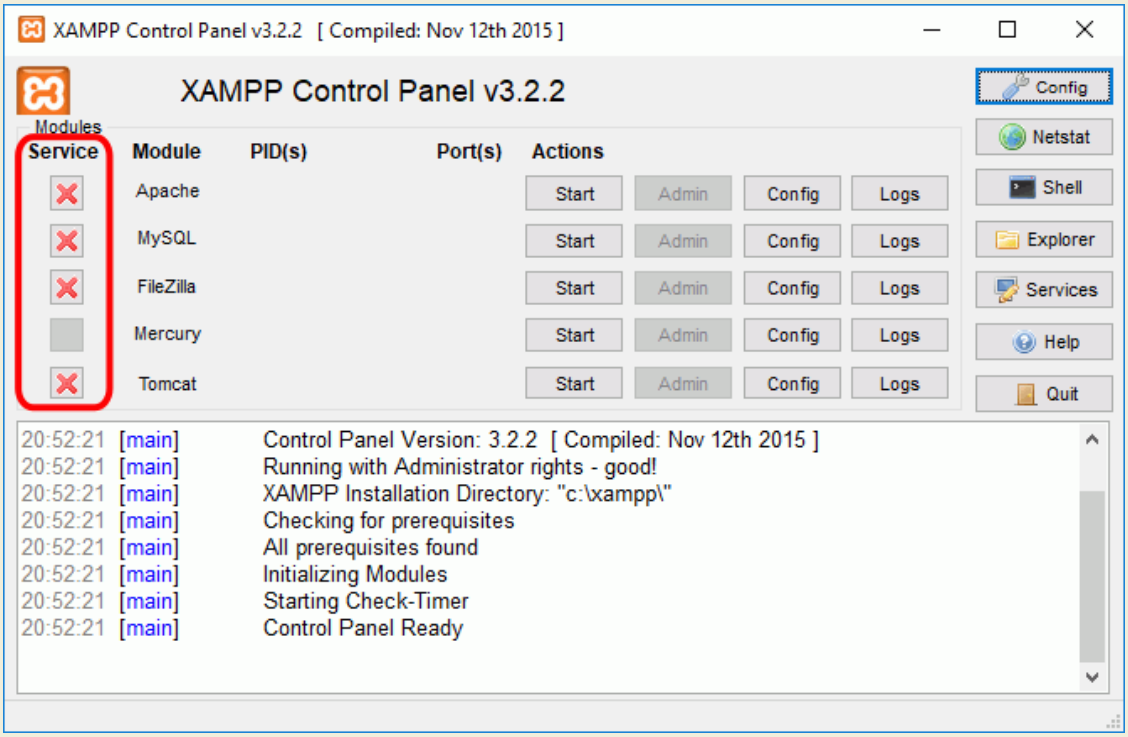
Para ejecutar el panel de control como administrador, hay que hacer clic derecho sobre el icono de acceso directo (Inicio > XAMPP Control Panel > y elegir la opción "Más > Ejecutar como administrador".



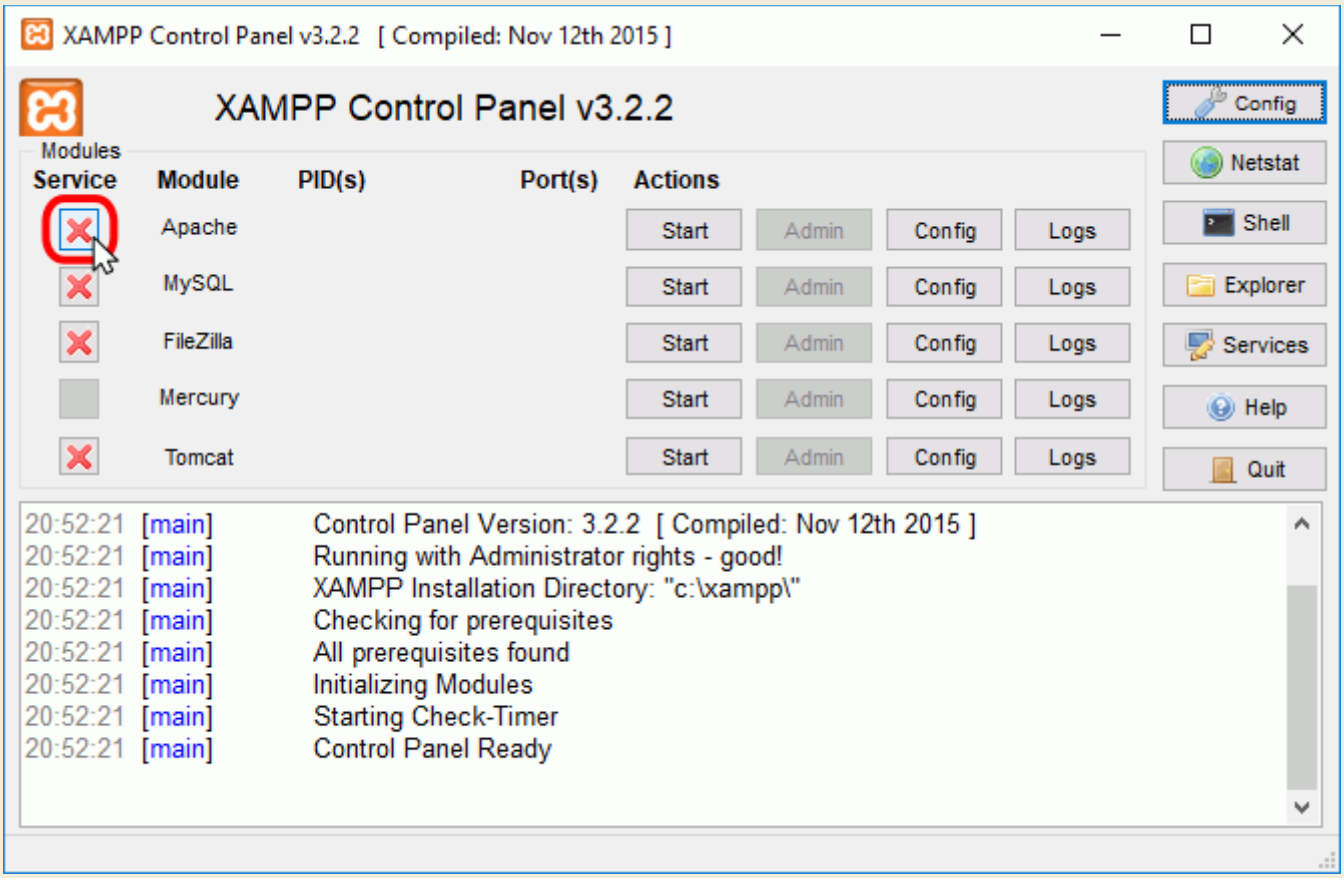
INSTALAR LOS SERVIDORES COMO SERVICIOS

Si queremos instalar un servidor como servicio, es decir, que se ponga en marcha cada vez que arrancamos el computadora, hay que marcar la casilla Service correspondiente. Para ello, es necesario iniciar XAMPP como administrador.

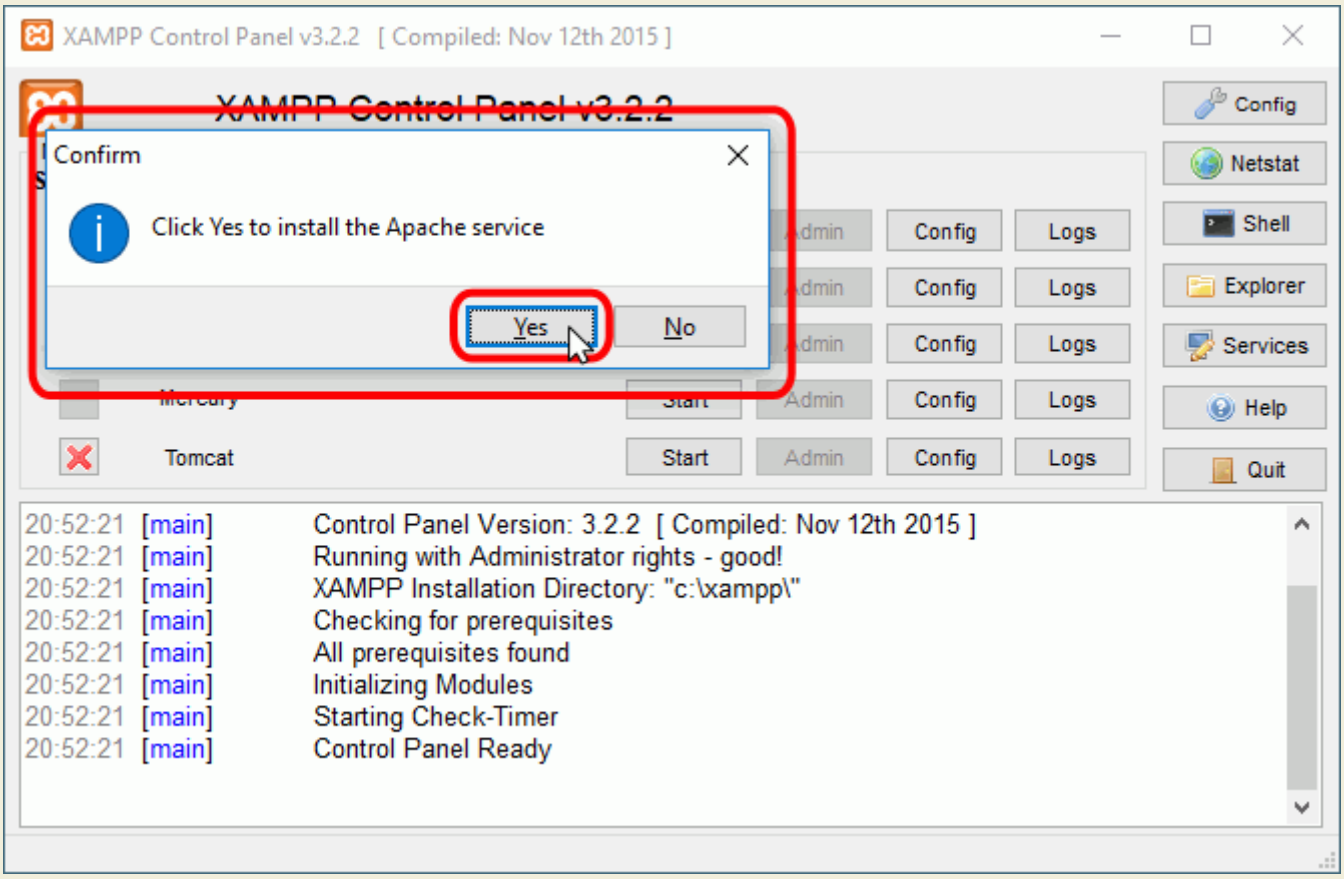
Iniciando XAMPP como administrador, las casillas de la columna Service muestran el estado de los servicios:



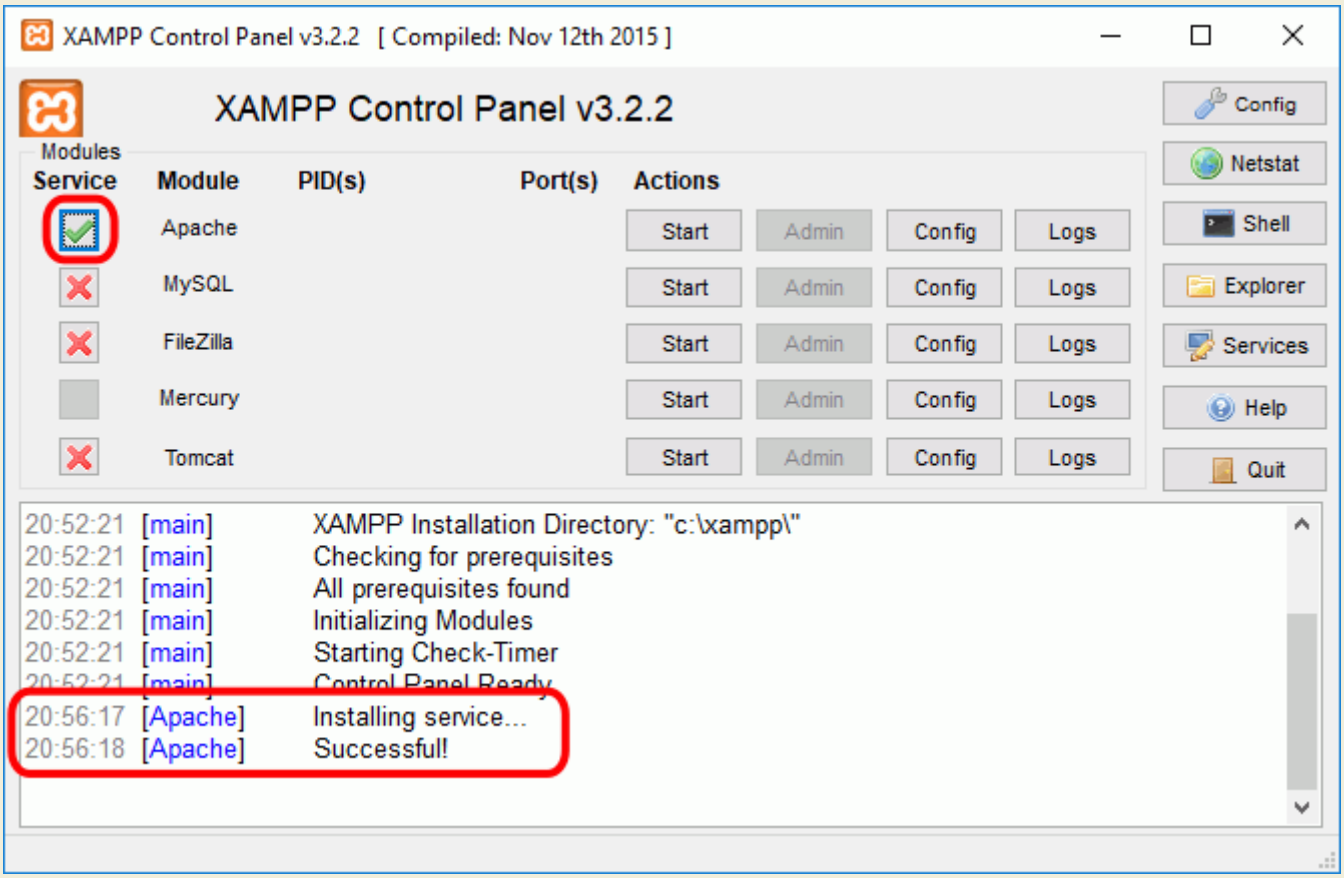
Para instalar un servicio, haga clic en la casilla Service correspondiente:



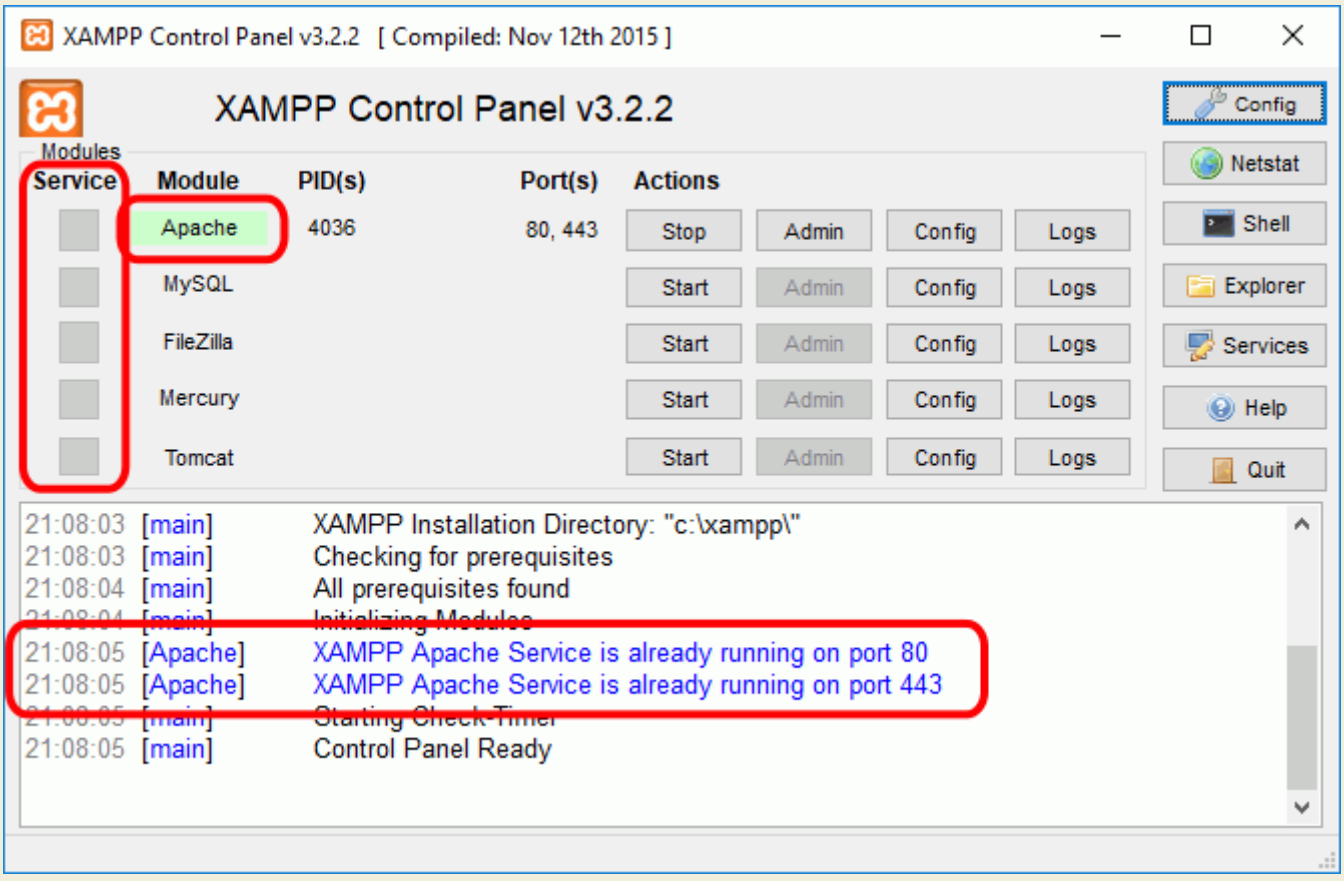
Se mostrará una ventana de confirmación. Para instalar el servicio, haga clic en Yes:

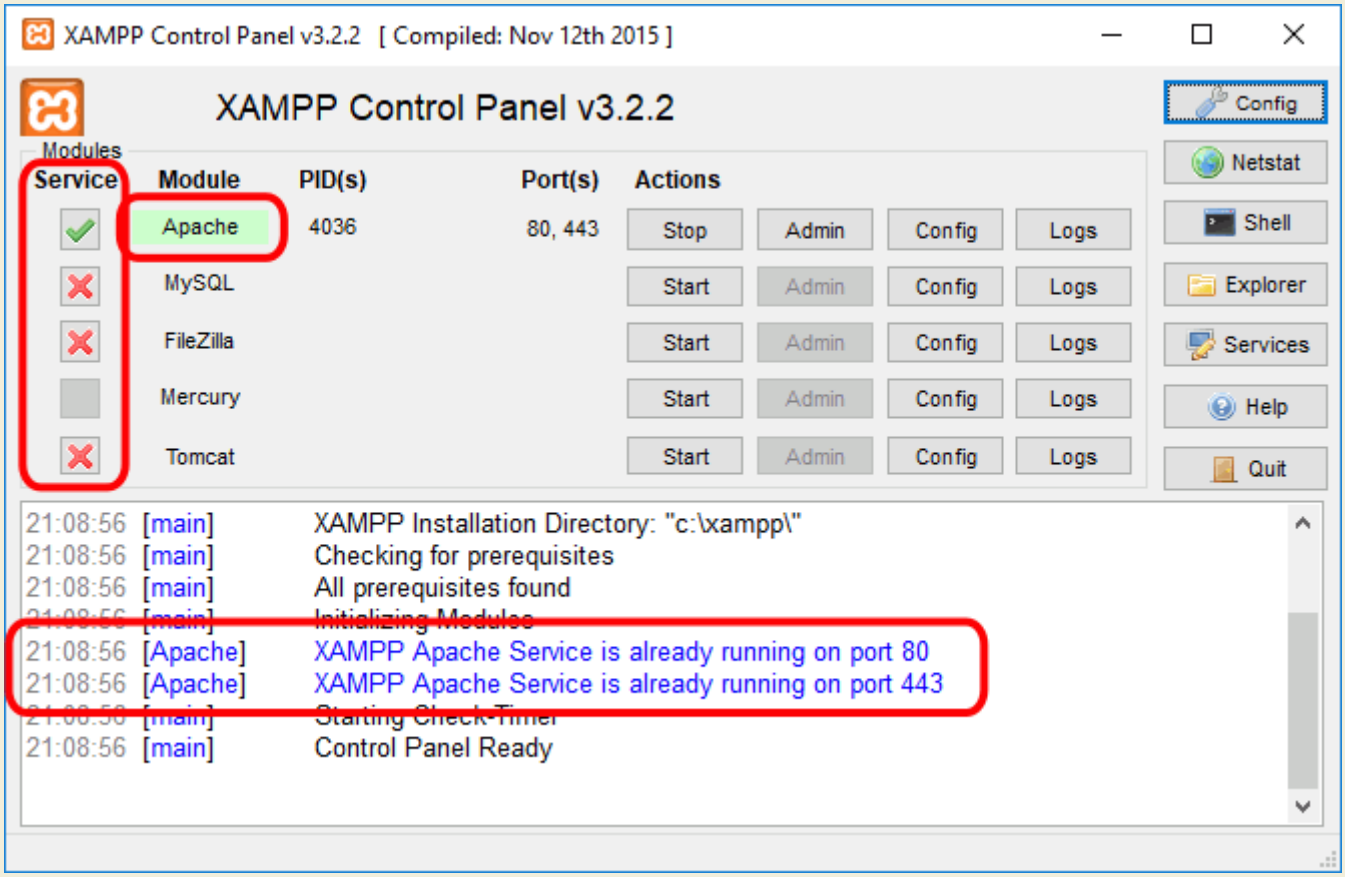


Si el servicio se instala correctamente, se indicará en el panel inferior y se mostrará una marca verde en la columna Service:



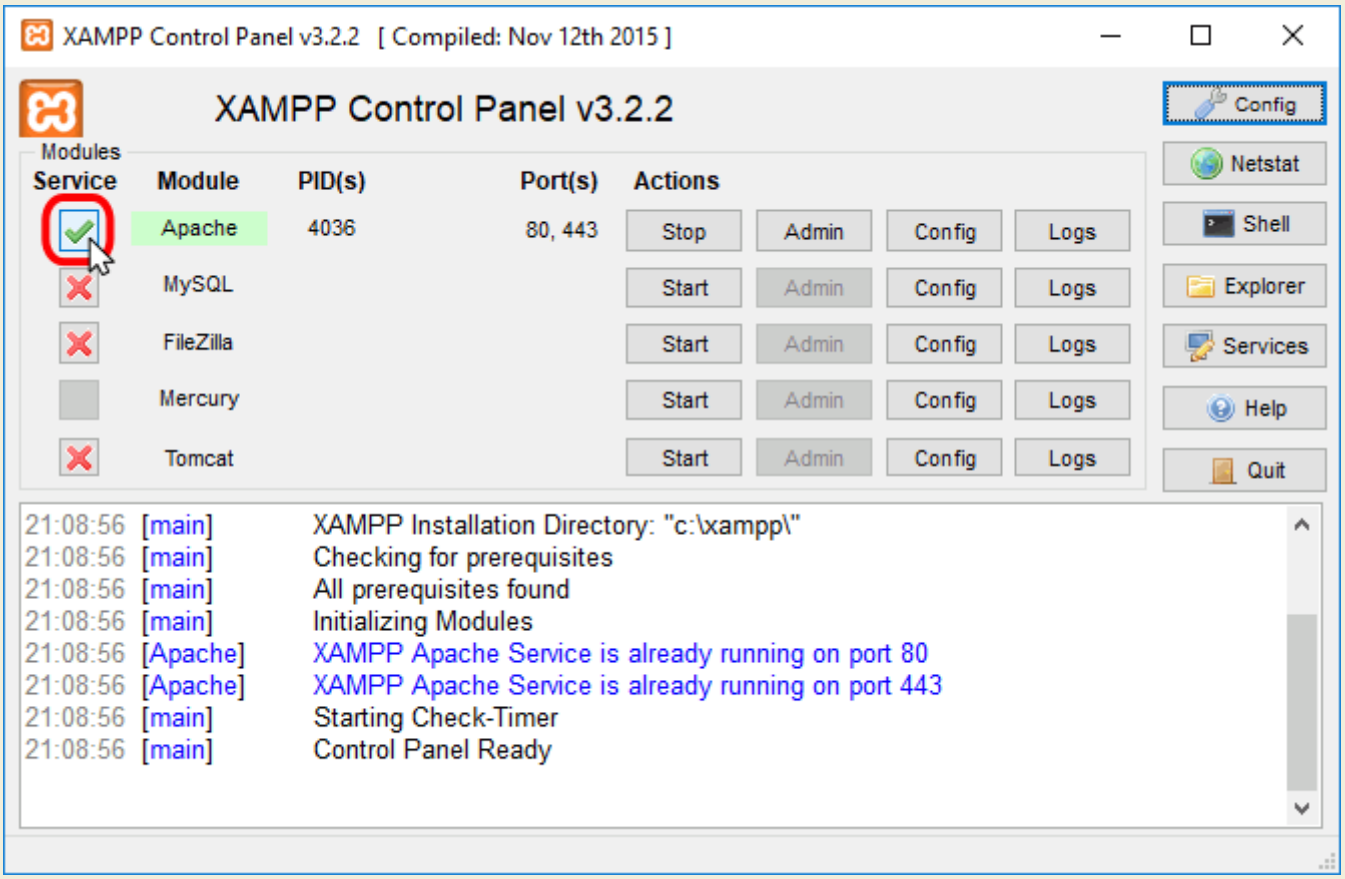
Al reiniciar la computadora, tanto si se hace como administrador como si no, el panel de control de XAMPP indica los servicios arrancados. La columna Service sólo mostrará los iconos si se ha iniciado como administrador.



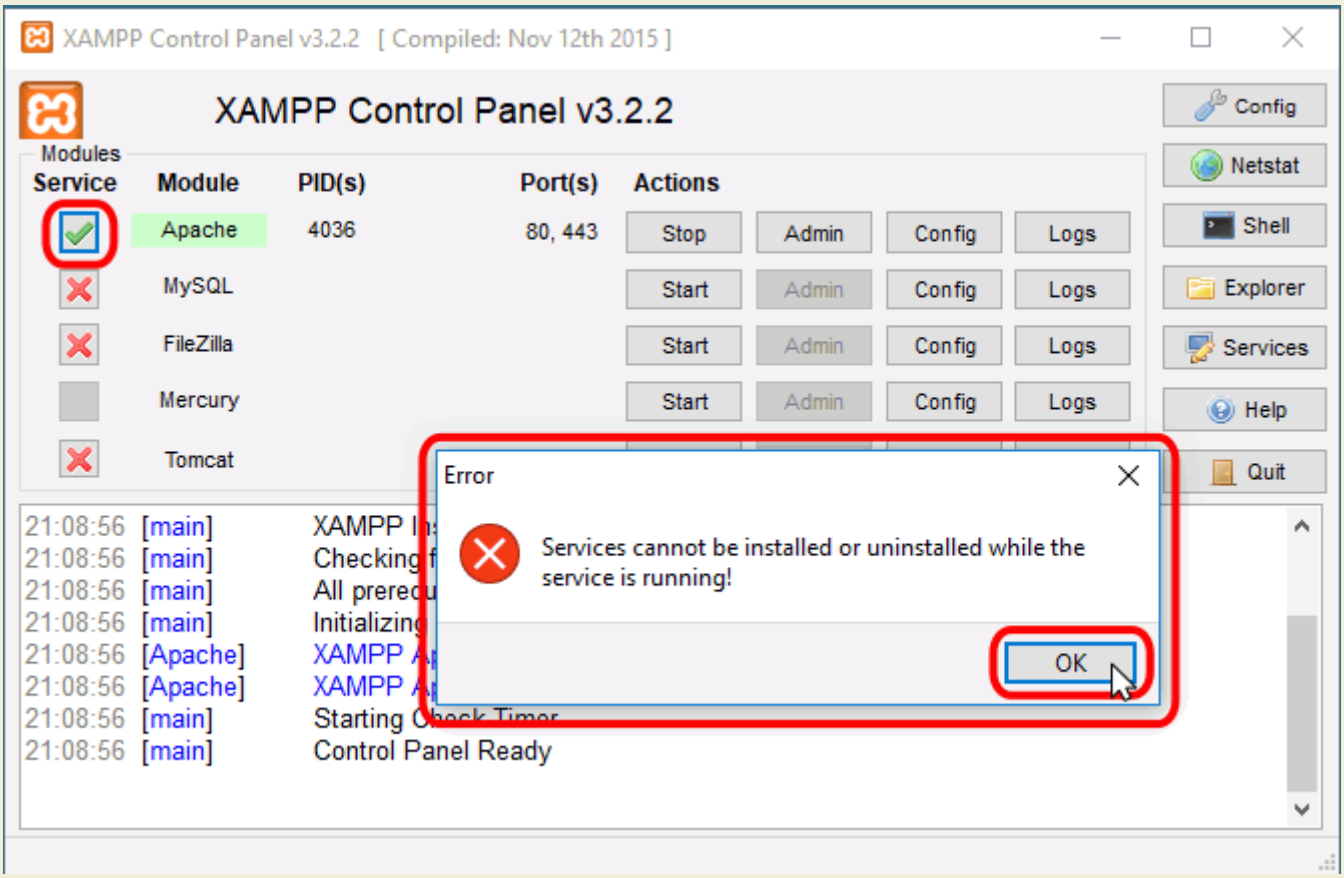


Si queremos desinstalar un servidor como servicio, es decir, que **no** se ponga en marcha cada vez que arrancamos el computadora, hay que desmarcar la casilla Service correspondiente. Para ello, es necesario iniciar XAMPP como administrador.

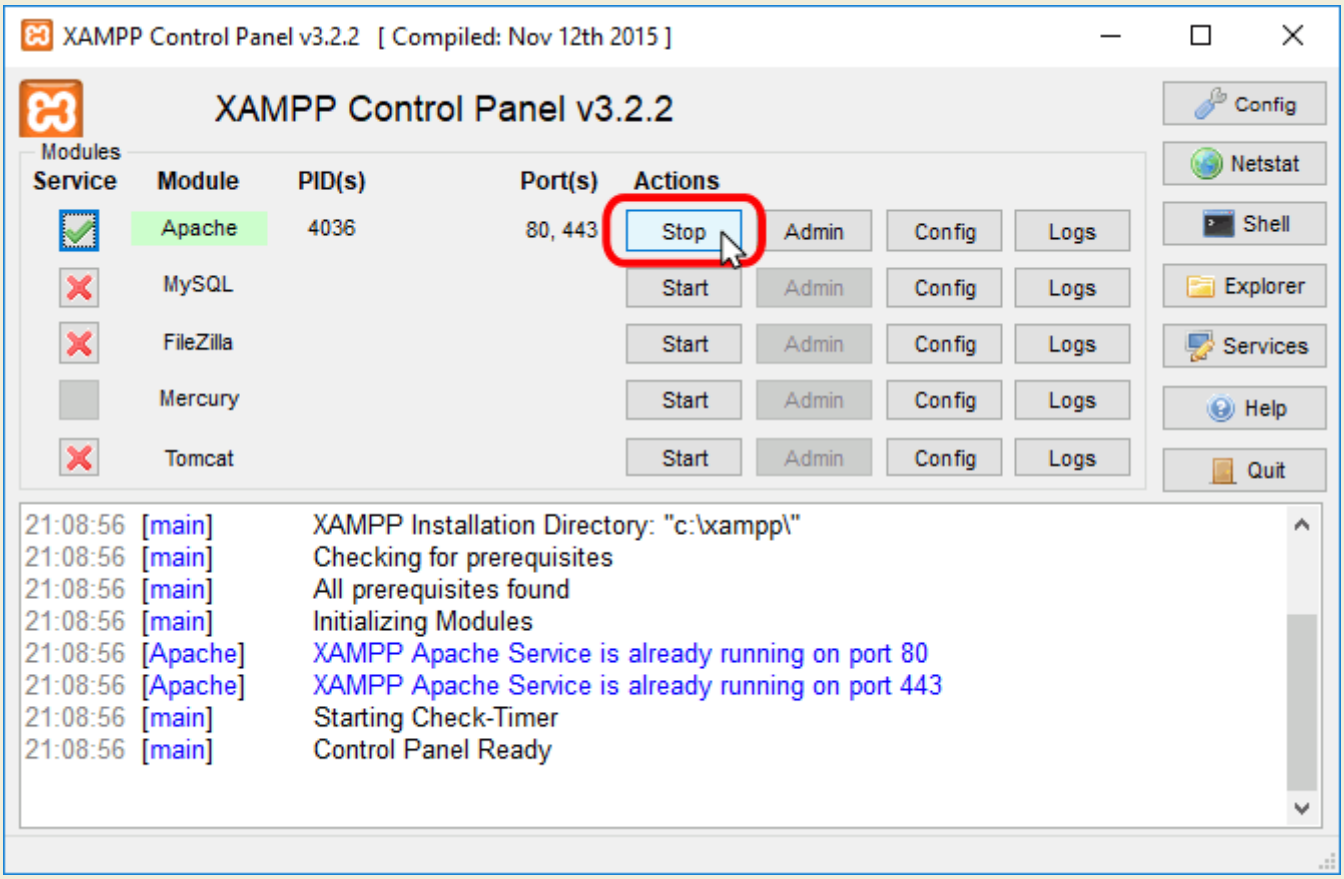
Para eliminar un servicio, haga clic en la casilla Service correspondiente:



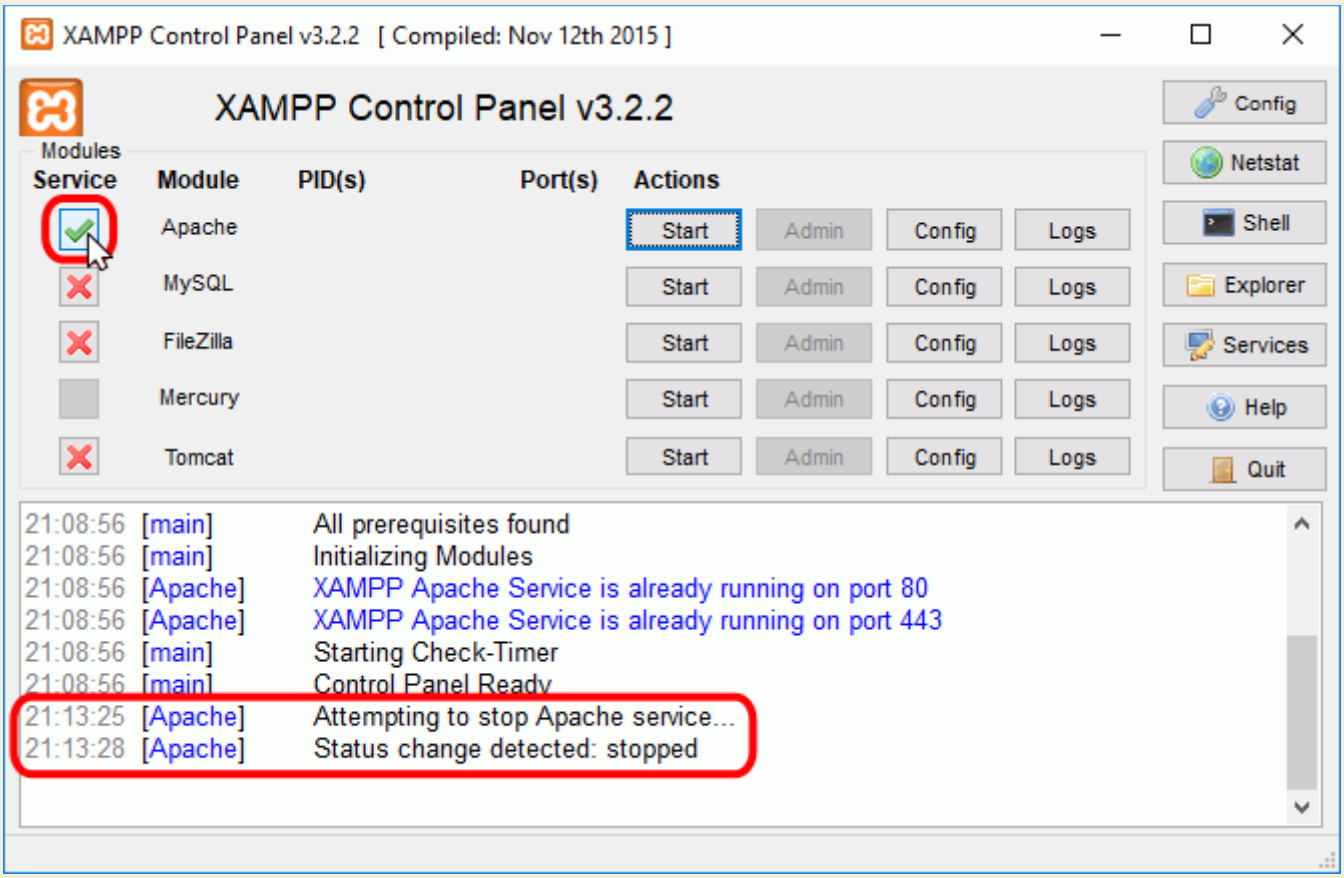
Se mostrará una ventana de aviso, que indica que para desinstalar un servicio, antes debe detenerse el servidor. Haga clic en OK para continuar:



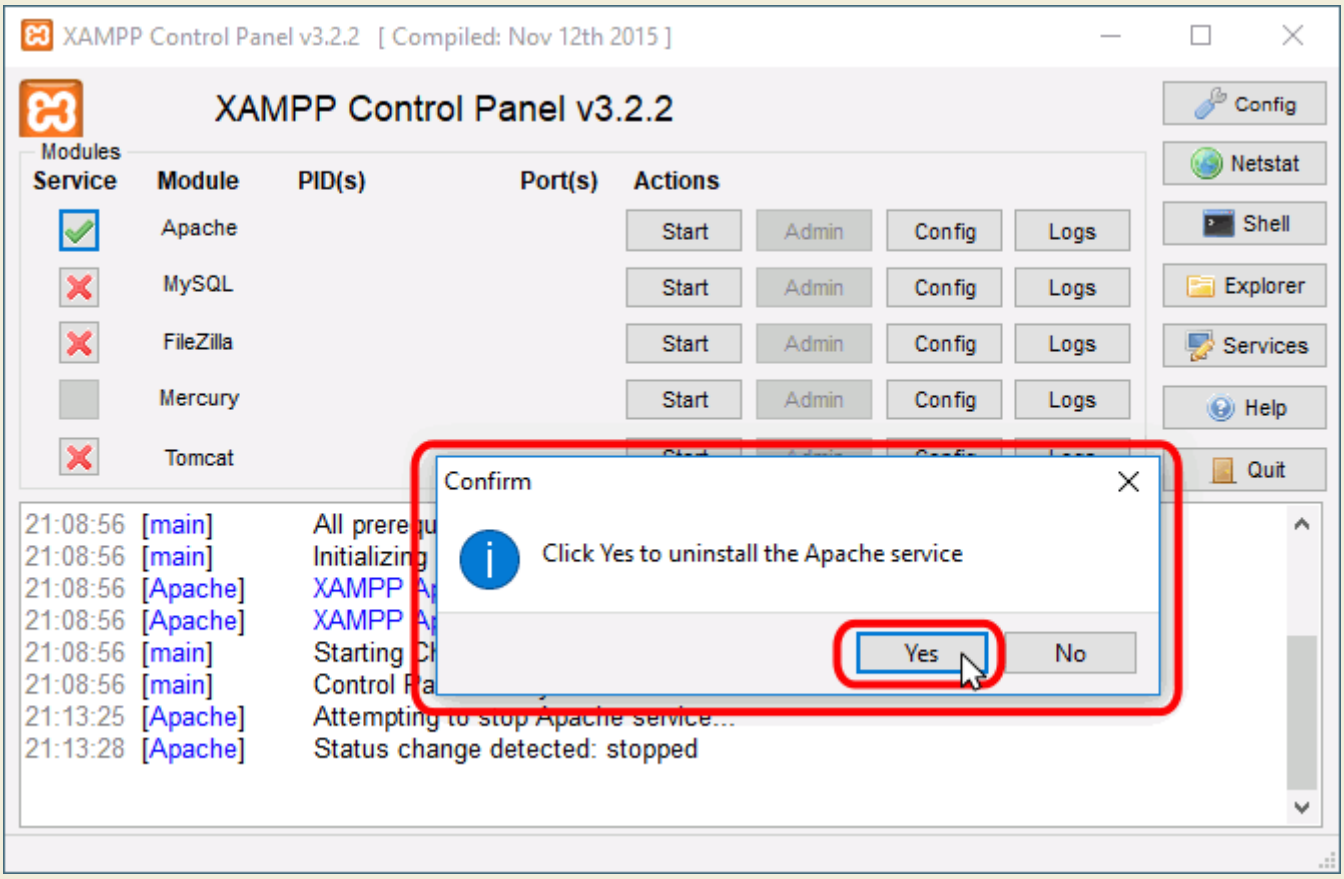
Detenga el servidor haciendo clic en el botón Stop correspondiente:



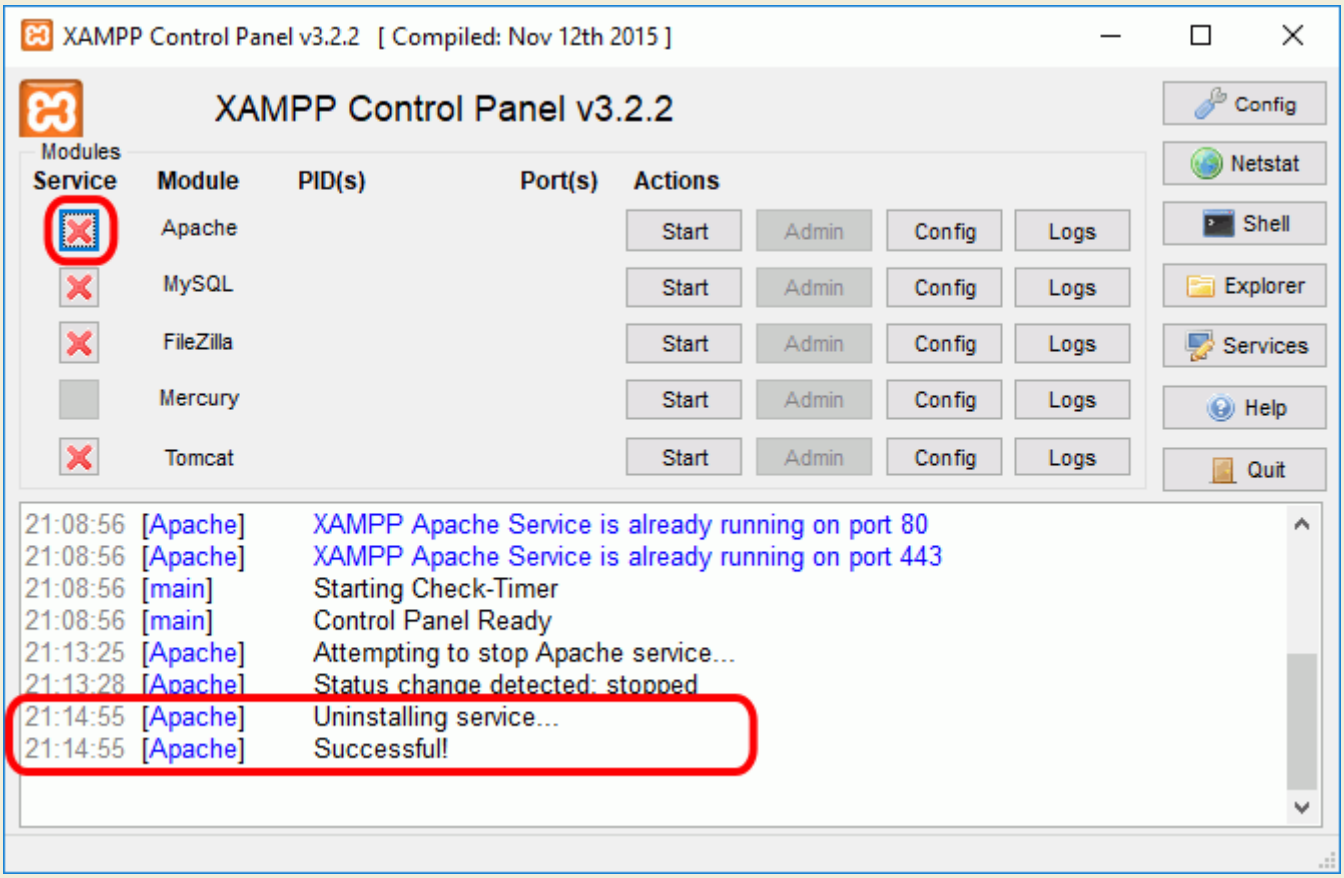
Una vez detenido el servidor, haga clic en la casilla Service correspondiente:



Se mostrará una ventana de confirmación. Para desinstalar el servicio, haga clic en Yes:

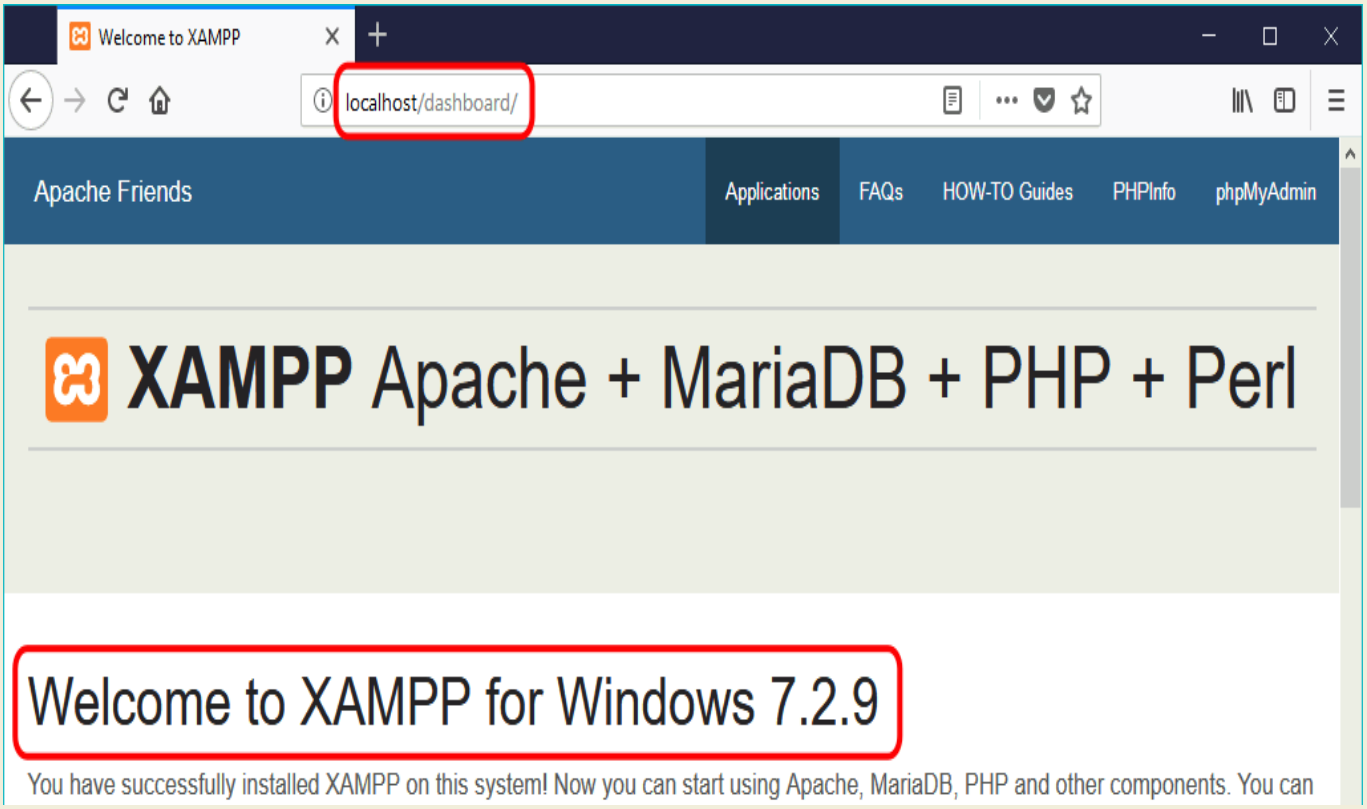


Si el servicio se desinstala correctamente, se indicará en el panel inferior y se mostrará una marca roja en la columna Service:



EL PANEL DE ADMINISTRACIÓN WEB DE XAMPP

Si se ha iniciado el servidor Apache, para comprobar que todo funciona correctamente, hay que escribir en el navegador la dirección **http://localhost**. XAMPP abrirá el nuevo panel de administración web (dashboard), que todavía se encuentra en desarrollo:



ESTRUCTURA DE UNA PÁGINA PHP

Una página PHP es un archivo de texto que contiene uno o varios fragmentos de código PHP y que también puede contener fragmentos de código HTML.

FRAGMENTOS PHP

Los fragmentos de código PHP están delimitados por las etiquetas <?php (etiqueta inicial) y ?> (etiqueta final) y contienen las instrucciones de PHP. Más adelante se comentan otros posibles delimitadores de fragmentos de código PHP.

```
<?php
print "<p>Hola</p>\n";
?>
```

```
<p>Hola</p>
```

Los fragmentos de PHP no pueden anidarse, es decir, no puede haber un fragmento dentro de otro:

```
<?php
print "<p>Hola</p>\n";

<?php
print "<p>Adios</p>\n";
?>
```

Parse error: syntax error, unexpected '<' in **ejemplo.php** on line **4**

Si en una misma página hay varios fragmentos PHP, se tratan como un único programa. Los siguientes programas son equivalentes:

```
<?php
$saludo = "Hola";          // Se define una variable
print "<p>$saludo</p>";    // Se escribe el valor de la variable
?>
```

```
<p>Hola</p>
```

```
<?php
$saludo = "Hola";              // Se define una variable
?>
<?php
print "<pre>$saludo<pre>";    // Se escribe el valor de la variable
?>
```

```
<p>Hola</p>
```

Si el programa termina con un fragmento PHP, ese último fragmento PHP no necesita cerrarse. Los siguientes programas son correctos:

```
<?php
print "<p>Hola</p>\n";
```

```
<p>Hola</p>
```

```
<?php
  print "<p>Hola</p>\n";
?>
<?php
print "<p>Adios</p>\n";
```

```
<p>Hola</p>
<p>Adios</p>
```

FRAGMENTOS PHP Y FRAGMENTOS HTML

En una página PHP todo lo que no son fragmentos PHP son fragmentos HTML. Los fragmentos de código HTML no están delimitados por ninguna etiqueta.

```
<?php
print "<p>Hola</p>\n";
?>
<p>¿Cómo estás?</p>
```

```
<p>Hola</p>
<p>¿Cómo estás?</p>
```

Cuando un navegador solicita una página PHP a un servidor, el servidor lee el archivo secuencialmente y va generando el resultado:

- si la página contiene fragmentos HTML, el código HTML se incluye sin modificados en el resultado.
- si la página contiene fragmentos PHP, se ejecutan las instrucciones que se encuentran en los fragmentos de código PHP. Si esas instrucciones generan texto, el texto se incluye en el resultado.

Cuando el servidor termina de leer el archivo, el servidor envía al navegador el resultado.

Es importante señalar que:

- El navegador recibe una página web (es decir, únicamente código HTML), no recibe código PHP.
- Los fragmentos PHP tiene que generar las etiquetas HTML que se necesiten para una correcta visualización de la página web en el navegador.
- El navegador no puede distinguir en la página recibida, qué parte ha sido generada en un fragmento PHP y qué parte se encontraba ya en un fragmento HTML.
- Como la página PHP se lee secuencialmente, el código HTML generado por los fragmentos PHP y el incluido en los fragmentos HTML se encuentran en el mismo orden en que se encontraban los fragmentos en la página PHP.

Los ejemplos siguientes muestran diferentes programas PHP que generan el mismo resultado:

- Programa con un único fragmento PHP

```php
<?php
print "<p>Hola</p>\n";
print "<p>¿Cómo estás?</p>\n";
print "<p>Adios</p>\n";
?>
```

```
<p>Hola</p>
<p>¿Cómo estás?</p>
<p>Adios</p>
```

- Programa con un fragmento PHP y un fragmento HTML

```
<p>Hola</p>
<?php
print "<p>¿Cómo estás?</p>\n";
print "<p>Adios</p>\n";
?>
```

```
<p>Hola</p>
<p>¿Cómo estás?</p>
<p>Adios</p>
```

- Programa con dos fragmentos PHP y un fragmento HTML

```
<?php
print "<p>Hola</p>\n";
?>
<p>¿Cómo estás?</p>
<?php
print "<p>Adios</p>\n";
?>
```

```
<p>Hola</p>
<p>¿Cómo estás?</p>
<p>Adios</p>
```

En un fragmento PHP no pueden escribirse etiquetas HTML sueltas; el código HTML debe generarse siempre con instrucciones de PHP. El programa siguiente no puede ni siquiera ejecutarse y produce un error de sintaxis:

```
<?php
<p>Hola</p>
?>
```

Parse error: syntax error, unexpected '<' in **ejemplo.php** on line **2**

De la misma manera, no se deben escribir instrucciones de PHP fuera de fragmentos PHP. No se producen errores, pero el resultado no es el esperado. Como el texto fuera de fragmentos PHP se envía tal cual al navegador, el navegador recibirá las instrucciones de PHP, que el navegador no sabe ejecutar y se mostrarán en la pantalla.

```
<?php
print "<p>Hola</p>\n";
?>
<p>¿Cómo estás?</p>

print "<p>Adios</p>\n";
```

```
<p>Hola</p>
<p>¿Cómo estás?</p>

print "<p>Adios</p>\n";
```

EL DELIMITADOR <? ... ?>

Nota: El delimitador **<? ... ?>** no se debe utilizar. Se comenta en esta lección porque todavía se puede encontrar en código escrito hace muchos años.

El delimitador **<?** se podía utilizar antiguamente como delimitador de bloques PHP, pero desde hace años **se desaconseja completamente** su uso porque provoca conflictos, entre otros, con la instrucción de procesamiento xml (<?xml) de los documentos XML. En su lugar, se debe utilizar el delimitador **<?php .. ?>**.

Mediante la directiva short_open_tag (etiqueta de apertura abreviada) se puede permitir el uso del delimitador **<?** además del delimitador **<?php**. Si la directiva short_open_tag tiene el valor On (lo que está desaconsejado), el siguiente programa no daría errores:

```
<?
print "<p>Hola</p>\n";
?>
```

```
<p>Hola</p>
```

Pero si la directiva short_open_tag tiene el valor Off (valor recomendado), el mismo programa produciría un resultado incorrecto y que puede resultar chocante, ya que en el navegador se puede ver el programa original:

```
<?
print "<p>Hola</p>\n";
?>
```

```
<?
print "<p>Hola</p>\n";
?>
```

Este resultado se debe a que como el programa no contiene el delimitador <?php, su contenido no se reconoce como fragmento PHP. Como no se reconoce como fragmento PHP, el servidor lo envía tal cual al navegador, que no puede mostrarlo correctamente puesto que no es HTML correcto. Abriendo la vista de código fuente en el navegador, se podría ver mejor el programa completo.

EL DELIMITADOR <?= ... ?>

El delimitador **<?=** es una abreviatura de la expresión **<?php echo** que se utiliza en muchos frameworks de PHP en los documentos que sirven de plantilla de generación de interfaces, sobre todo cuando estas

plantillas las van a editar usuarios que pueden no tener conocimientos de programación (diseñadores gráficos, etc.).

```
<?php
$saludo = "Hola";
$despedida = "Adios";
?>
<p><?= $saludo ?></p>
<p><?= $despedida ?></p>
```

```
<p>Hola</p>
<p>Adios</p>
```

Antes de PHP 5.4, publicado en marzo de 2012, para poder usar este delimitador se necesitaba activar la directiva short_open_tag. El problema era que, como se comenta en el apartado anterior, desde hace años se aconseja no activar esa directiva para impedir el uso del delimitador <?. A partir de PHP 5.4 se resolvió este problema desvinculado el delimitador <?= de la directiva short_open_tag. por tanto, desde PHP 5.4, el delimitador <?= se puede utilizar sin limitaciones.

COMENTARIOS EN PÁGINAS PHP

En un fragmento PHP se pueden comentar líneas de código utilizando:

- // para comentar el resto de la línea (como en C++)
- # para comentar el resto de la línea (como en la shell de Unix o en Perl)
- /* */ para delimitar varias líneas (como en C)

Estos comentarios no se añaden al código HTML generado por la página, por lo que no pueden verse en el navegador.

```
<p><strong>
<?php
// La instrucción print escribe texto en la página web
print "Hola"; // El comentario se puede escribir al final de la línea
?>
</strong></p>
```

```
<p><strong>Hola</strong></p>
```

```
<p><strong>
<?php
# La instrucción print escribe texto en la página web
print "Hola"; # El comentario se puede escribir al final de la línea
?>
</strong></p>
```

```
<p><strong>Hola</strong></p>
```

```
<?php
print "<p><strong>";
/* Dentro de un fragmento PHP no se pueden escribir
   etiquetas HTML sueltas, tienen que estar siempre
   incluidas en instrucciones print
*/
?>
```

```
Hola
<?php
print "</strong></p>";
?>
```

```
<p><strong>Hola</strong></p>
```

Si se quieren escribir comentarios en los fragmentos HTML, hay que utilizar la etiqueta de comentarios de HTML <!-- .... -->.

Estos comentarios, como todo el código HTML situado en los fragmentos HTML, se incluyen sin modificaciones en el resultado, por lo que pueden verse en el navegador.

```
<p>
<?php
print "Hola";
?>
</p>
<!-- El texto anterior ha sido generado por PHP -->
```

```
<p>Hola</p>
<!-- El texto anterior ha sido generado por PHP -->
```

Si se quieren escribir comentarios en las hojas de estilo CSS, hay que utilizar la etiqueta de comentarios de C /* .... */.

Estos comentarios, como todo el código CSS situado en las hojas de estilo, se incluyen sin modificaciones en el resultado, por lo que pueden verse en el navegador.

```
HTML {
  background-color: #FF0000; /* #FF000000 es rojo */
}
```

```
HTML {
  background-color: #FF0000; /* #FF000000 es rojo */
}
```

CÓDIGO HTML DE PÁGINAS HTML

Si queremos crear páginas HTML válidas debemos generar código HTML válido, es decir, generar todas las etiquetas HTML necesarias respetando las reglas sintácticas. En este curso se generarán páginas HTML 5 (siguiendo la sintaxis XHTML), y se utilizará el juego de caracteres UTF-8.

El ejemplo siguiente muestra el código HTML de una página web HTML 5 válida con las líneas numeradas.

```
<!DOCTYPE HTML>
<HTML lang="es">
<head>
  <meta charset="utf-8">
  <title>Página HTML 5 válida</title>
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <link rel="stylesheet" href="estilo.css" title="Color">
</head>
<body>
  <p>Esta página es una página HTML 5 válida.</p>
</body>
</HTML>
```

- La etiqueta <!DOCTYPE ... > (línea 1) es obligatoria. Esta etiqueta indica el tipo de documento (DOCTYPE) de la página. El tipo HTML sin información adicional corresponde a HTML 5. En versiones anteriores a HTML 5 esta etiqueta contenía la dirección web de la DTD (definición de tipo de documento) que especifica la versión del lenguaje de marcas utilizado en el documento (cuál es la estructura, qué etiquetas existen y qué atributos pueden tener).
- La etiqueta <HTML> ... </HTML> (líneas 2 a 13) engloba todo el documento HTML. El atributo lang no es obligatorio e indica el idioma en que está escrito el documento. El documento HTML se divide a su vez en dos partes, el encabezado (<head> ... </head>, líneas 3 a 8) y el cuerpo (<body> ... </body>, líneas 10 a 12).
- El encabezado <head> ... </head> (líneas 3 a 8) contiene información de identificación y control que en general no se muestra en la ventana del navegador, aunque puede afectar a la presentación (por ejemplo, los enlaces a hojas de estilo).
- Las etiquetas <meta> están pensadas para proporcionar información sobre el documento a los programas que analicen la página y por ello existen muchas etiquetas <meta> diferentes. Por ejemplo, la etiqueta <meta name="keywords" content="..."> está pensada para informar a los buscadores de los temas tratados en la página pero, por desgracia, se ha abusado mucho de esta etiqueta así que los buscadores no la toman prácticamente en cuenta).
    - La línea 4 del ejemplo informa del juego de caracteres empleado en la página, en este caso UTF-8, que es el juego de caracteres recomendado para HTML 5.
    - La línea 6 del ejemplo se utiliza para mejorar la presentación de las páginas webs en las pantallas de los teléfonos móviles y, en general, en las pantallas de alta densidad.
- La etiqueta de título <title> ... </title> (línea 5) contiene el texto que se muestra en la barra de título de la ventana del navegador.
  La etiqueta title es obligatoria y debe incluirse en todas las páginas web.
- La línea 7 contiene el enlace a una hoja de estilo. Esta etiqueta no es obligatoria.
- El cuerpo (<body> .... </body>, líneas 10 a 12) contiene lo que se verá en la ventana del navegador. En el ejemplo, contiene un único párrafo
    - La etiqueta de párrafo ... contiene el texto que se muestra en la ventana del navegador

GENERAR EL CÓDIGO HTML EN PÁGINAS PHP

La página PHP debe incluir o generar todo el código HTML necesario. El ejemplo siguiente genera una página válida mediante un único fragmento PHP.

Nota: Dentro de la cadena todas las comillas están precedidas del carácter contrabarra (\).

```
<?php
print "<!DOCTYPE HTML>
<HTML lang=\"es\">
<head>
  <meta charset=\"utf-8\">
  <title>Página HTML 5 válida</title>
  <meta name=\"viewport\" content=\"width=device-width, initial-scale=1.0\">
</head>

<body>
  <p>Esta página es una página HTML 5 válida.</p>
</body>
</HTML>";
?>
```

En principio, el mismo código HTML podría generarse combinando fragmentos PHP y fragmentos HTML, como en el siguiente ejemplo con dos fragmentos PHP y tres fragmentos HTML:

Nota: El carácter especial \n incluido en los print es para generar un salto de línea en el código HTML.

```
<!DOCTYPE HTML>
<HTML lang="es">
<head>
  <meta charset="utf-8">
<?php
print "  <title>Página HTML 5 válida</title>\n";
?>
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
</head>

<body>
<?php
print "  <p>Esta página es una página HTML 5 válida.</p>\n";
?>
</body>
</HTML>
```

ENLACES A HOJAS DE ESTILO

Una página PHP puede enlazar a una hoja de estilo CSS exactamente igual que una página HTML, es decir, enlazando a la hoja de estilo mediante la etiqueta <link>.

Los siguientes ejemplos muestran el enlace a la hoja de estilo incluido en un fragmento HTML o generado por un fragmento PHP:

```
<head>
  <meta charset="utf-8">
  <title>Página HTML 5 válida</title>
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <link rel="stylesheet" href="estilo.css" title="Color">
</head>
```

```
<head>
  <meta charset="utf-8">
  <title>Página HTML 5 válida</title>
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
<?php
print "  <link rel=\"stylesheet\" href=\"estilo.css\" title=\"Color\">\n";
?>
</head>
```

ERRORES Y MENSAJES DE ERROR

ERRORES EN UN PROGRAMA

Las computadoras no poseen inteligencia ni sentido común, se limitan a ejecutar las instrucciones contenidas en el programa. Por ello, los programas no pueden contener errores ni ambigüedades y tienen que tener previstos todas las posibles situaciones.

Algunos errores de programación no pueden ser detectados por los compiladores o intérpretes (por ejemplo, si un programa tiene que calcular una suma y el programador ha escrito una multiplicación en vez de la suma el programa no dará el resultado correcto pero el compilador no puede saberlo), pero otros sí. En esta lección se comentan los errores que detectan los compiladores o intérpretes, avisando al programador o al usuario de su existencia.

Podemos distinguir dos tipos de errores:

- Errores sintácticos (parse errors): Un programa debe estar escrito según las reglas del lenguaje de programación. Si un programa no cumple alguna regla, se dice que el programa tiene un error sintáctico.
- Errores no sintácticos (errors): Aunque un programa siga las reglas del lenguaje, el computadora puede ser incapaz de ejecutar una instrucción determinada (dependiendo del error, el computadora y de continuar con la ejecución del programa. En ese caso, se dice que el programa tiene errores no sintácticos.

Entre los errores no sintácticos, se pueden distinguir varios tipos de errores:

- Advertencias (warnings): El computadora es incapaz de ejecutar una instrucción determinada, pero la computadora puede continuar con la ejecución del programa. En ese caso, se dice que el compilador da advertencias.
- Avisos (notices): El computadora es capaz de ejecutar una instrucción determinada, pero detecta que el programador puede haberse equivocado. En ese caso, se dice que el compilador da avisos. Los avisos no siempre son señales de equivocaciones, pero deben evitarse.
- Obsolescencias (deprecated): El compilador puede detectar que un programa utiliza características del lenguaje que actualmente funcionan correctamente, pero que no estarán (o pueden no estar) disponibles en futuras versiones, y generar un aviso de obsolescencia.

MENSAJES DE ERROR

Cuando PHP encuentra errores en un programa, PHP puede generar un mensaje de error que se incluye en la página en el punto en que se ha encontrado el error. El error incluye una descripción del error y el número de línea en el que PHP ha encontrado el error, lo que facilita al programador su corrección (el número de línea indica la línea en que PHP se da cuenta de que hay un error, pero el error puede estar en una línea anterior a la indicada por PHP).

La directiva **error_reporting** del fichero de configuración de PHP php.ini permite establecer qué tipos de error generarán un mensaje de error. Como se comenta en la lección de configuración de PHP:

- En un entorno de producción (es decir, en un servidor accesible a usuarios en Internet o en una red local) se recomienda que PHP no muestre mensajes de error al usuario, pues esos mensajes de error pueden ser aprovechados por usuarios maliciosos para sus propios fines (provocar la caída del servidor, obtener información privada, etc.).
- En un entorno de desarrollo (es decir, en un servidor utilizado por el programador o para probar el funcionamiento de la aplicación) se recomienda que PHP muestre el mayor número de errores posible, para tener conocimiento de ellos y corregirlos lo antes posible

Antes de ejecutar un programa, PHP analiza el programa en busca de errores sintácticos (llaves, paréntesis o comillas sin cerrar, etc.). Un error sintáctico es un error fatal, que detiene el propio análisis y lo único que obtendremos será el mensaje de error. Si un programa contiene varios errores sintácticos, PHP sólo muestra el mensaje de error del primer error. Una vez corregido el primer error, al volver a ejecutar el programa se mostrará el mensaje del segundo error y así sucesivamente hasta corregir todos los errores sintácticos.

Si un programa no contiene errores sintácticos, PHP ejecuta el programa. Durante la ejecución del programa PHP puede encontrar un error que impida la ejecución del resto del programa o encontrarse con un error que no impida la ejecución del programa. En el primer caso, el navegador mostrará el contenido de la página generado hasta el error, el mensaje de error y nada más. En el segundo caso, el navegador mostrará el contenido de la página completo, con el mensaje de error insertado en el lugar en el que se produjo (dependiendo de la estructura de la página, el mensaje de error puede ser visible únicamente en el código fuente).

Por supuesto, que un programa no genere mensajes de error no significa que el programa funcione correctamente, pero si genera mensajes de error probablemente no funcionará correctamente.

ERRORES SINTÁCTICOS (PARSE ERRORS)

Un programa está formado por las palabras reservadas del lenguaje, cadenas, variables, constantes o funciones, paréntesis, llaves, corchetes, signos de puntuación, operadores, etc., que además deben sucederse en un orden determinado. Los errores sintácticos se producen cuando el compilador no encuentra los símbolos (tokens) esperados o encuentra otros en su lugar.

Cuando encuentra un símbolo no esperado, PHP indica el tipo de símbolo que ha encontrado sin esperarlo.

unexpected character

```
<?php
print <p>Hola</p>\n
?>
```

Parse error: syntax error, unexpected '<' in **ejemplo.php** on line **2**

En este caso, el motivo del error es que un trozo de texto debe estar contenido en una cadena (rodeado por comillas simples o dobles) y cuando encuentra el carácter < (que representa el operador matemático de desigualdad), no puede entenderlo como tal.

unexpected T_CONSTANT_ENCAPSED_STRING

```
<?php
primt "<p>Hola</p>\n";
?>
```

Parse error: syntax error, unexpected T_CONSTANT_ENCAPSED_STRING in **ejemplo.php** on line **2**

En este caso, el motivo del error es que **print** no es una palabra reservada del lenguaje ni una función (debería decir **print**) y cuando encuentra la cadena no sabe cómo seguir.

unexpected T_PRINT

```
<?php
print "<p>Hola</p>\n"
print "<p>Adios</p>\n"
?>
```

Parse error: syntax error, unexpected T_PRINT in **ejemplo.php** on line **3**

En este caso, el motivo de error es que la primera sentencia no finaliza en punto y coma (;) y no se esperaba el segundo print. Por cierto, la última sentencia de un bloque php puede no finalizar en punto y coma, aunque se recomienda hacerlo.

Los paréntesis, llaves y corchetes deben estar emparejados (por cada signo abierto debe hacer uno cerrado).

Final inesperado

```
<?php
$edad = 15;
if ($edad < 19) {
    print "<p>Menor de edad</p>\n";
?>
```

Parse error: syntax error, unexpected $end in **ejemplo.php** on line **5**

En este caso, el motivo de error es que no se ha cerrado la llave y PHP llega al final del fichero sin haber cerrado el bloque.

Paréntesis inesperado

```
<?php
$resultado = 2 * 3 + 4);
print "<p>Resultado: $resultado</p>\n";
?>
```

Parse error: syntax error, unexpected ')' in **ejemplo.php** on line **2**

En este caso, el motivo de error es que no se ha abierto el paréntesis y PHP no se espera el paréntesis cerrado.

ADVERTENCIAS (WARNINGS)

Estos avisos indican que se ha producido un error de algún tipo que no impide la ejecución del resto del programa, aunque seguramente el programa no producirá el resultado esperado.

División por cero

```
<?php
$numero = 0;
$resultado = 1 / $numero;
print "<p>Resultado: $resultado</p>";
?>
```

Warning: Division by zero in **ejemplo.php** on line **3**

Resultado:

En este caso, el motivo del aviso es que se intenta realizar un cálculo matemático imposible (dividir entre cero).

Uso de cabeceras después de escribir texto

```
<?php
print "<p>Hola</p>\n";
header("Location:index.php");
exit();
?>
```

Hola

Warning: Cannot modify header information - headers already sent by (output started at ejemplo.php:2) in **ejemplo.php** on line **3**

En este caso, el motivo del aviso es que se intenta enviar una cabecera después de haber enviado texto (dependiendo de la configuración de PHP, concretamente de la directiva output_buffering, puede no dar aviso de error).

AVISOS (NOTICES)

Los avisos no impiden la ejecución de un programa, aunque seguramente el programa no producirá el resultado esperado.

Variable no definida

```php
<?php
print "<p>Edad: $edad</p>\n";
?>
```

Notice: Undefined variable: edad in **ejemplo.php** on line **2**

Edad:

En este caso, el motivo del aviso es que se utiliza una variable a la que no se había asignado valor previamente. El programa funciona puesto que PHP le asigna el valor de cadena vacía, pero se desaconseja utilizar variables no establecidas previamente.

Ausencia del símbolo $ en el nombre de una variable

```php
<?php
$edad = 18;
print "<p>Edad: " . edad . "</p>\n";
?>
```

Notice: Use of undefined constant edad - assumed 'edad' in **ejemplo.php** on line **3**

Edad: edad

En este caso, el motivo del aviso es que PHP encuentra una palabra (edad) que no es un término del lenguaje, ni una variable (porque no empieza por $) ni una cadena (porque no está entre comillas. PHP genera un aviso y supone que se trata de una cadena, y por eso escribe "edad".

OBSOLESCENCIAS (STRICT / DEPRECATED)

Constantes insensibles a mayúsculas/minúsculas

```php
<?php
define("TEST", 99.9, true);
?>
```

Deprecated: define(): Declaration of case-insensitive constants is deprecated in **ejemplo.php** on line **3**

En este caso, el motivo del aviso es que desde la versión PHP 7.0 no se pueden definir constantes insensibles a mayúsculas (es decir, que se pueden escribir tanto en minúsculas como en mayúsculas).