INTRODUCCIÓN

Cuando escribí mi anterior trabajo, *Domine PHP y MySQL*, lo hice siendo consciente de las muchas cosas que me dejaba en el tintero. En realidad, es así como debía ser, puesto que el objetivo de aquel libro era "abrir las puertas" a PHP 5 al lector no iniciado, de modo que adquiriese un conocimiento y una soltura suficientes como para empezar a trabajar en serio. Sin embargo, ahora, ese mismo lector reclama más. Ahora que sabe manejarse en el lado del servidor, ese lector se da cuenta de los conocimientos que posee y, de algún modo, intuye que hay muchos que aún no tiene. Y los necesita, los quiere y los reclama. Y, por lo que a mí respecta, ha llegado el momento de ofrecérselos. Este libro pretende ir más allá, mostrando prestaciones y características del lenguaje no documentadas en la mayor parte de la bibliografía existente. Se han recopilado todos aquellos conceptos que son útiles para la programación de páginas dinámicas de calidad profesional.

Este libro va dirigido, por tanto, a personas que ya tienen un conocimiento de la programación en el lado del servidor, en general, y de PHP 5 en particular. Si usted es recién llegado al mundo de la programación de sitios dinámicos, le recomiendo, encarecidamente, que lea mi anterior trabajo antes de seguir con este, ya que, de otro modo, no le sacará partido. En este libro no hablaremos ya de conceptos como la arquitectura básica de Cliente-Servidor, definición de matrices o uso elemental de sesiones, si no que abordaremos los conceptos más avanzados para el manejo de estas y otras prestaciones. No obstante, repasaré aquellos conceptos que, aun siendo básicos, resulten de utilidad para introducir o afianzar determinados conocimientos.

Por supuesto, se ha trabajado sobre las últimas versiones de PHP y MySQL, de forma que esta documentación está plenamente vigente. PHP 5 es tan diferente en algunos conceptos concretos a sus antecesores, que no podía ser de otro modo.

Este libro se sale un poco de la línea de mis anteriores trabajos. En efecto, en otros textos de programación para Internet he buscado, siempre, ser eminentemente práctico, mostrando usos reales de los conceptos expuestos. Aunque el estudio de cualquier disciplina informática no puede ser de otro modo, a este libro le he dado un enfoque más teórico. Y esto, por una razón. Este libro se ha escrito para que las personas que ya tienen conocimientos previos de PHP 5 tengan, también, una guía de ayuda para optar al examen oficial de Zend. En efecto, el fabricante de PHP 5 ofrece un examen que, una vez superado, otorga una titulación, con validez en todo el mundo, que puede abrirle puertas laborales muy interesantes. Por lo tanto, este libro no es una guía para aprender a programar, sino un compendio de los temas que pueden aparecer en el examen. Eso no significa que, una vez superado el examen (o si usted decide no optar al título), no pueda servirle como guía o referencia en su trabajo cotidiano. Muchos de los conceptos aquí expuestos son extremadamente útiles en el día a día. También encontrará otros conceptos que, probablemente, no llegue a necesitar jamás y cuya teoría es especialmente espesa, para mi gusto. Tal es el caso, por ejemplo, del material contenido en los capítulos 10 y 12, dedicados, respectivamente, a XML y a los streams. Sin embargo, tome nota de los conocimientos teóricos expuestos, aunque de momento no les vea una utilidad específica, para poder afrontar el examen. Una vez superado, usted decide qué partes le interesan para su uso cotidiano y cuáles no.

Si usted decide presentarse al examen, debe saber que este consta de setenta preguntas que debe responder en un período de hora y media (90 minutos). No se preocupe: hay tiempo suficiente. Sin embargo, hay una dificultad añadida: el examen sólo puede efectuarse en inglés o en francés. Esto se debe a que Zend diseñó dicha prueba, inicialmente, para el mercado norteamericano y para el Canadá francófono. Aunque hoy en día existen centros en todo el mundo que realizan este examen, sigue sin estar traducido. Esto es, para muchos de nosotros, un escollo pero, dado que se trata de un inglés técnico y que muchas veces, las preguntas se "cazan" a primer golpe de vista, se puede superar. Por cierto. Si desea saber dónde puede presentarse al examen en su país y ciudad, consulte la siguiente dirección en Internet: **http://vue.com/servlet/vue.web2.core.Dispatcher?webContext=CandidateSite&webApp=TestCenterLocator&requestedAction=register&cid=369** (hay un acceso directo en el CD adjunto).

Si bien en el mercado angloparlante hay algunas obras que facilitan el acceso al examen, esta es la primera guía en español de la que yo tengo conocimiento. Para su redacción se ha recurrido a fuentes de todo tipo. Por supuesto, la mayor fuente de información se encuentra en el propio manual de PHP, pero también en Internet se han recopilado, filtrado y seleccionado documentos públicos con un contenido muy interesante. A lo largo de más de tres meses, se ha estado seleccionando y filtrando información que cubre todas las secciones del examen. Espero y deseo que este libro le resulte especialmente útil, tanto si usted desea presentarse, como si no. Si le ha servido de ayuda, si usted logra superar el examen, me doy por satisfecho.

CAPÍTULO 1

GENERALIDADES FUNDAMENTALES

Este capítulo incluye una serie de conceptos que, si bien son básicos para el uso diario de PHP, no siempre están disponibles en la documentación que uno pueda encontrar al respecto. También se incluyen aquellos detalles que, de tan básicos, probablemente usted los ha olvidado. Eso es algo que nos sucede a muchos programadores con el tiempo. Nos acostumbramos a olvidar aquello que no usamos con frecuencia y, cuando lo necesitamos, aunque se trate de cosas elementales, tenemos que hacer un pequeño, o no tan pequeño, esfuerzo para recordarlas. Por esta razón, este capítulo le servirá a usted como consulta de cosas que, sin duda, ya conoce pero que seguro que no siempre recuerda. Desde luego, repasar mis notas para escribirlo sí me ha servido para refrescar mi memoria en algunos aspectos. Aunque lo que aparece en las siguientes páginas le resulte obvio, no deje de leerlo, aunque sea a vista de pájaro.

1.1 INCLUYENDO PHP

Antes de seguir adelante, vamos a ver la definición más universalmente aceptada de PHP. ***PHP es un lenguaje incrustado, basado en el motor Zend. Se usa, principalmente, para desarrollar contenido HTML dinámico, aunque puede usarse también para generar documentos XML (entre otros).*** Esta definición es importante por ser la que proporciona Zend Technologies, empresa desarrolladora de PHP. Fíjese que decimos que es un lenguaje incrustado. En efecto, como usted ya sabe, el código PHP se "incrusta" o integra en el de documentos web. Para ello se usan unos tags, o etiquetas, que delimitan donde empieza y acaba el código PHP. A lo largo de la historia de PHP han existido cuatro formatos delimitadores para el código. Estos aparecen recogidos en la siguiente tabla:

ETIQUETAS DELIMITADORAS

Etiquetas	Ejemplo

Estándar	`<?php`
	`  //código PHP`
	`?>`

Abreviadas	`<?`
	`  //código PHP`
	`?>`

Tipo Script	`<script language="php">`
	`  //código PHP`
	`</script>`

Tipo ASP	`<%`
	`  //código PHP`
	`%>`

Usted, hasta ahora, habrá usado cualquiera de estos formatos. En lo sucesivo use, exclusivamente, el primero (etiquetas estándar). Los otros tres formatos se consideran obsoletos y, aunque PHP 5 los reconoce, es muy probable que futuras versiones no lo hagan.

Las etiquetas abreviadas pueden causar conflictos cuando se incorpora el PHP a documentos XML. Las etiquetas abreviadas se han empleado muchísimo cuando, en un documento HTML, es necesario mostrar el valor de una variable PHP, en la forma **`<?= $variable ?>`**. Es una forma cómoda de mostrar un valor en un documento HTML. Si usted se ha acostumbrado a esto, olvídese de ello desde ya. Como le he comentado, estas etiquetas son obsoletas y pueden desaparecer. Además, para que las etiquetas abreviadas funcionen es necesario que esté activada (valor **`on`**) la directiva ***short_open_tag*** en el fichero de configuración de PHP (**`php.ini`**). Por defecto, esta directiva está desactivada en la versión 5.

Las etiquetas de tipo script se introdujeron, principalmente, para su uso en navegadores que no soportaban el formato estándar y que, por lo tanto, ignoraban el código PHP. Hoy día no existe esa situación y estas etiquetas no tienen razón de ser.

Por último, las etiquetas de tipo ASP se implementaron por razones acerca de las cuales nadie se pone de acuerdo (no existe un argumento "oficial" de Zend); sin embargo, hoy también están obsoletas y se comenta que PHP 6 no las reconocerá. En la actualidad, para que funcionen, es necesario activar la directiva ***asp_tags*** en el fichero de configuración. Por defecto, esta directiva está desactivada en la versión 5. Tiene una reseña de esta y otras directivas en el apéndice C.

A partir de ahora use, por lo tanto, las etiquetas estándar exclusivamente, ya que son las únicas que garantizan compatibilidad actual y futura del inicio y finalización de scripts.

1.2 CARACTERES ESPECIALES

Cuando se incluye código PHP en un documento HTML, todo lo que aparece fuera de los delimitadores de PHP es enviado, directamente, al navegador… incluyendo los caracteres "invisibles" de nueva línea. Cuando se trabaja en el cuerpo de la página, esto no representa ningún problema, ya que, como usted sabe, HTML ignora los saltos de línea a menos que se indiquen, específicamente, mediante el tag
. Sin embargo, cuando trabajamos con cabeceras HTTP, la cosa cambia. Un salto de línea inesperado puede causar efectos como que una cabecera no se envíe, generando, en su lugar, un mensaje de error. Por eso es importante prestar especial atención a no incluir saltos de línea adicionales después del tag de cierre de PHP, ?>. Algunos procesadores de texto plano usados por los programadores incluyen una línea en blanco al final del código. Pensando en ello, el intérprete ignora, específicamente, el primer salto de línea después del tag de cierre. No obstante, acostúmbrese a evitar estos saltos, para ahorrarse problemas.

Otro punto a tener en cuenta son los espacios en blanco. PHP no es sensible a estos, de modo que no importa cuantos use. Por lo tanto, las sentencias **`$v1=10;`** y **`$v1 = 10;`** son equivalentes y tienen el mismo efecto. Sin embargo, hay tres puntos donde usted debe tener especial cuidado en no introducir espacios en blanco indeseables. Estos son:

- Los tags de inicio y finalización de PHP. Usted no puede, por ejemplo, escribir **`<? php`** o **`? >`** (con espacios en medio).
- Las instrucciones. Podemos escribir, por ejemplo, **`mysql_query`**, pero no **`my sql_query`** o **`mysql _query`**, etc.
- Los nombres de las variables. Podremos usar **`$miNombre`**, pero no **`$mi Nombre`**.

Otros caracteres con los que tiene que tener cuidado es con los comentarios cuando se acabe un script. Como usted sabe, PHP reconoce tres maneras de incluir comentarios en su listado, como se ve a continuación:

```
<?php
      // Esta línea es un comentario.
      # Esta también lo es.
      /* Esto es un bloque de comentarios.
      Puede abarcar varias líneas. */
?>
```

Tenga en cuenta que, una vez hallado el tag de cierre de PHP, el navegador envía a la salida estándar (el monitor) todo lo que encuentre a continuación, como si fuera parte de HTML. Por lo tanto, el tag de cierre no debe aparecer en medio de una línea o bloque de comentarios. Para entender a que me refiero, suponga el siguiente fragmento de script:

```
<?php
      // Esto es un comentario
      // En este aparece el tag ?> de cierre.
```

En el navegador aparecerán las palabras **`de cierre`**, que están después del correspondiente tag.

1.3 TIPOS DE DATOS

PHP soporta varios tipos de datos, que podemos clasificar en dos categorías: ***escalares*** y ***compuestos***. Los datos escalares son aquellos que sólo pueden contener un valor en un momento dado (las variables "normales", para entendernos). Los datos compuestos son los que pueden tener dos o más valores en un mismo instante de tiempo (matrices y objetos).

1.3.1 Los datos escalares

Los datos escalares pueden ser de cuatro tipos fundamentales, recogidos en la tabla que vemos a continuación:

TIPOS DE DATOS ESCALARES

Dato	Tipo de valor
int	Un número entero con signo.
float	Un número fraccionario (en coma flotante) con signo.
string	Una secuencia de datos binarios.
booleano	Un valor de tipo true o false (verdadero o falso).

PHP reconoce, como acabamos de ver en esta tabla, dos tipos de valores numéricos: los enteros y los fraccionarios, o en coma flotante. Una cosa que no debe olvidar es que los números fraccionarios emplean el punto como separador de decimales, y no existe separador de millares. Así pues, el valor 1.258,23, expresado para un script PHP, será 1258.23. Esto, en sí, no es nada nuevo, sino que es una característica común de todos los lenguajes de programación. Esto se debe a que es la notación anglosajona, usada en informática. Otra cosa es que, al obtener un valor, lo formateemos antes de la salida, para que el usuario lo vea de otro modo, pero eso es harina de otro costal.

Para anotar números enteros, PHP nos permite manejar varias bases de numeración. Sin embargo, la tres que manejaremos con mayor frecuencia son: la

decimal (base 10), la hexadecimal (base16) y la octal (base 8). Cuando establecemos un valor numérico entero, PHP asume, por defecto, que está en decimal (por ejemplo, **1517**). Para indicar que un valor está expresado en hexadecimal lo precederemos con **0x** (por ejemplo, **0x5ED**). Si queremos que PHP entienda que el valor está expresado en octal, lo precederemos con un **0** (por ejemplo, **02755**). En ocasiones, estas notaciones pueden inducir a error a las personas no familiarizadas con ellas. En concreto, es fácil confundir un número octal con uno decimal. También puede inducir a confusión el hecho de que, en la notación hexadecimal, no tiene importancia que las letras aparezcan en mayúsculas o en minúsculas. Preste especial atención a estos puntos.

Los números en coma flotante se pueden expresar con la notación clásica de un punto separando la parte entera de la fraccionaria (por ejemplo, **763747.56**) o en forma de mantisa y exponente (por ejemplo **17e23**). En este último caso es indiferente que la letra **e** que separa ambos términos sea mayúscula o minúscula.

Una cosa que debe tener en cuenta cuando maneje valores aritméticos grandes es que el rango de estos tipos de números depende, en gran parte, de la plataforma sobre la que se esté ejecutando el script. Así, un equipo de 64 bits podrá, en principio, manejar números más grandes que uno de 32 bits. Con esto puede haber un problema: PHP no maneja desbordamientos, con lo que el resultado de algo tan simple como una suma o una multiplicación puede, si se supera la capacidad de proceso de la máquina, arrojar resultados erróneos y es fácil pasarlos por alto.

Aún hay otra cosa que debe tener en cuenta. El manejo de datos en coma flotante por parte de PHP no siempre es como se espera. Observe el siguiente script:

```
<?php
      echo (int)((0.1 +0.7)*10);
?>
```

En principio usted piensa que el resultado a visualizar será 8. Es decir, 0.1+0.7=0.8. Al multiplicarlo por 10, el resultado es, internamente, 8.0 (puesto que trabajamos con números en coma flotante) y al aplicarle el casting **(int)** obtendremos 8. Sin embargo, si prueba el script verá que el resultado es, sorprendentemente, 7. Sin embargo, si usted suprime el casting **(int)** verá que el resultado es correcto. Esto se debe a que el resultado obtenido de la operación aritmética no es, realmente, 8.0, sino 7.999999. Al aplicar el casting, simplemente se trunca la parte fraccionaria, dejando el resultado 7.

Para concluir esta avanzadilla al mundo de los números, únicamente recordarle que tenga siempre presente qué tipos de datos va a manejar y cómo. En un capítulo posterior hablaremos de las prestaciones que PHP ofrece para trabajar con números de precisión arbitraria, de modo que logremos siempre los resultados adecuados al trabajo a realizar.

Los datos de tipo String, también comprendidos dentro de la categoría de datos escalares, se usan, habitualmente, para almacenar y manipular cadenas de texto. En otros lenguajes de programación, sólo pueden usarse para esto. Sin embargo PHP gestiona los datos String como secuencias de datos binarios. Esto quiere decir que en una variable de este tipo se puede almacenar una cadena de texto (lo más usual) pero también un recurso del sistema, un archivo de imagen, sonido o vídeo, etc. En general, cualesquiera datos binarios. Más adelante le dedicaremos un capítulo completo a este tipo de datos.

Los datos booleanos sólo pueden asumir, como usted sabe, el valor **`true`** (verdadero) o **`false`** (falso). Por esta razón, constituyen la base de todas las operaciones lógicas. Dado que PHP es un lenguaje no tipado (es decir, que una variable puede cambiar de tipo simplemente cambiándole su valor) permite conversiones de tipos de datos con cierta flexibilidad. Uno de los casos más habituales es la interpretación de datos numéricos como booleanos. Por ejemplo, un número 0 se evalúa como false y cualquier otro valor numérico se evalúa como true. También las cadenas son susceptibles de ser interpretadas de este modo: una cadena vacía o con un único carácter 0 se evalúa como false. Si tiene cualquier otro contenido (incluso varios ceros) es evaluada como true. Suponga, por ejemplo, el siguiente script:

```
<?php
      $cadena = "0";
      if ($cadena){
            echo "La cadena se evalúa como true.";
      } else {
            echo "La cadena se evalúa como false.";
      }
?>
```

El resultado en la salida del navegador será: **La cadena se evalúa como false.**

En un capítulo posterior entraremos a fondo en las estructuras de control (condicionales y bucles).

Por otro lado, también es posible la interpretación inversa de los datos, es decir, interpretar un valor booleano como una cadena o un número. Si es un valor true, se registra como 1 y si es false, como 0.

1.3.2 Los datos compuestos

Como contraparte de los datos escalares que acabamos de ver, existen otros datos conocidos, genéricamente, como compuestos. Son aquellos que pueden, como mencionábamos anteriormente, almacenar varios valores simultáneamente. Se trata de las **matrices** y los **objetos**, que se estudiarán en sus respectivos capítulos.

1.3.3 Otros datos

PHP gestiona, además, otro tipo de información no incluida en las anteriores categorías. Estos valores no son, estrictamente, datos escalares ni compuestos, por lo que no aparecen en la clasificación que hemos visto. En realidad, muchos programadores, incluido yo mismo, los consideramos inclasificables, pero son unidades de información que existen y, por ello, están en este apartado específico.

Por una parte, está el valor NULL. Esto significa, realmente, "ningún valor". Es decir, no representa un cero, ni una cadena vacía, ni un false, ni nada. Es el valor de una variable cuando aún no se le ha asignado nada. Suponga el siguiente script:

```
<?php
      echo gettype ($miVariable);
?>
```

En él se pretende, como ve, mostrar el tipo de dato de una variable que no existe. El resultado es **NULL**. También obtendremos este resultado de una variable que ya se haya usado en el script, si esta ha sido borrada con **`unset()`**.

El otro tipo de dato al que aludimos en este apartado son los recursos. Se trata de búferes que PHP emplea para gestionar bases de datos, ficheros, imágenes, etc. Por ejemplo, cuando se establece una conexión con una base de datos, o se abre un fichero de texto, creando un manejador, se genera un recurso. Si, por ejemplo, intentamos convertir un manejador de un fichero de disco a una variable numérica, obtendremos el número que PHP le ha asignado a ese recurso. Si intentamos convertirlo a una variable de tipo String, obtendremos la secuencia **`Resource id #`** seguida del número de recurso. Por último, y esto es lo más importante que debemos tener en cuenta sobre los recursos, podemos convertirlo a un valor booleano. Si el recurso se ha podido crear sin problemas, obtendremos un valor **`true`**; si no se ha podido crear, se obtiene un valor **`false`**. Por esta razón, muchas funciones de PHP aparecen documentadas en el manual oficial del lenguaje especificando que devuelven **`false`** si no se han podido ejecutar adecuadamente. En realidad, si usted se para a echarle un vistazo al manual, verá que esta no es una característica exclusiva de las funciones que generan un recurso como pueda ser, por ejemplo, **`fopen()`**. Hay muchas otras (por ejemplo, **`define()`**) que también devuelven un valor booleano **`false`** si no se han podido ejecutar adecuadamente.

1.4 NOMBRES DE VARIABLES Y FUNCIONES

Usted ya conoce las reglas para establecer los nombres de las variables que utilice en sus scripts. A estas alturas no vamos a entrar en esos detalles. Y, sin duda, conoce también el uso de las llamadas variables variables, o nombres variables de

variables. Así, pues, sabe que puede asignar, a una variable, el nombre de otra a la que desee acceder. Por ejemplo:

```
$cadena = "Esto es una cadena";
$numero = 25;
$var1 = "cadena";
echo $$var1;
```

Esto mostrará en la página **Esto es una cadena**, basándose en la llamada que se hace mediante el doble signo de dólar, tal como aparece en la línea resaltada. Es posible que usted no use este sistema de llamar a variables con mucha frecuencia. En realidad, nadie lo usa con demasiada frecuencia, sino sólo cuando es útil.

Lo que ya no es tan conocido es que PHP permite usar la misma técnica para llamar a funciones, tanto aquellas que están definidas en el propio lenguaje como las que define el usuario. Por ejemplo, observe el fragmento siguiente:

```
function f1(){
      echo "Esto es f1";
}
function f2(){
      echo "Esto es f2";
}
$llamada = "f1";
$llamada();
```

Como ve, podemos asignar el nombre de una función a una variable y luego llamarla mediante el nombre de dicha variable. Observe que, en este caso, al contrario que en los nombres variables de variables, no recurrimos al doble signo de dólar. La razón de esto es obvia: los nombres de variables ya incluyen, de por sí, un signo de dólar, mientras que los nombres de funciones no.

1.5 CONSTANTES

Usted conoce el uso normal de constantes en PHP, así como su creación mediante **`define()`**. Sin embargo, he querido incluir aquí un par de detalles que debe saber, y que no aparecen en la mayoría de la documentación existente. Por una parte, debe saber que las constantes sólo admiten valores escalares. No sé si alguna vez, llevado por la curiosidad, ha intentado usted definir una matriz constante, por ejemplo. Si lo ha intentado, ya sabe que no es posible en modo alguno. Y otra cosa. Las constantes son siempre, al contrario que las variables, de ámbito global, es decir, accesibles desde cualquier parte del script donde se encuentran definidas. Es decir, aunque estén definidas dentro del cuerpo de una función, su contenido es visible, sin más, desde fuera de la función. En el capítulo dedicado a funciones hablaremos más a fondo de los ámbitos.

1.6 MOSTRAR VALORES EN LA PÁGINA

PHP contempla, como usted sabe, varios modos de mostrar valores numéricos, de cadena, etc. en una página. Los dos sistemas más habituales son el uso de **`print()`** y de **`echo`**. Mucha gente considera estas dos palabras clave como idénticas, como si una de ellas fuese un alias de la otra. Y, en la mayoría de los casos, lo son. Usted puede mostrar un valor, directamente en la página, con cualquiera de estas dos instrucciones. Sin embargo, hay una diferencia, sutil pero significativa, desde el punto de vista de un programador experimentado. ***Mientras que*** `print()` ***puede formar parte de expresiones complejas,*** `echo` ***no puede***. He remarcado esta breve definición porque es pregunta de examen cuando se aspira a obtener el certificado de Zend y muchos programadores, incluso con experiencia, la fallan. Observe la siguiente línea de código:

```
$miVariable = print ("Esto es una cadena");
```

Si todo va bien (si no se produce un error), en la página se mostrará la cadena y en la variable se almacenará el valor 1.

Sin embargo, tenga presente una cosa. Tal como dice el manual de PHP, **`print()`** no es una función, por lo que los paréntesis, al igual que en el caso de **`echo`**, son opcionales, aunque mucha gente, por convencionalismo, los incluye. Al no ser, ninguna de ambas, funciones, no pueden ser referenciadas por el sistema de nombres variables que veíamos en el apartado 1.4.

1.7 GESTIÓN DE ERRORES

La captura y tratamiento de errores es un aspecto fundamental de muchos lenguajes de programación incluyendo, como no podía ser de otro modo, PHP. Este proporciona varias prestaciones con esta finalidad. En este capítulo hablaremos de aquellas que forman, tradicionalmente, parte del lenguaje. Más adelante, en el capítulo reservado a Programación Orientada a Objetos conoceremos el nuevo tratamiento de excepciones de PHP 5.

Los errores que pueden producirse en un script de PHP se pueden clasificar, de un modo genérico, en cinco categorías básicas:

- **Errores en tiempo de compilación (Compile-time errors)**. Son aquellos que se producen durante la compilación de un script, cuando este es tratado de ese modo (lo que no es habitual cuando se programan sitios web, ya que lo normal es la interpretación en lugar de la compilación). Estos errores no pueden capturarse desde el propio script.

- **Errores fatales (Fatal errors)**. Son errores que detienen la ejecución de un script. No pueden ser capturados.
- **Errores recuperables (Recoverable errors)**. Representan fallos significativos, pero pueden manejarse de forma segura.
- **Avisos (Warnings)**. Errores recuperables que indican un problema en tiempo de ejecución, pero que no detienen la misma.
- **Notificaciones (Notices)**. Indican una posible situación de error, no necesariamente significativa. No detienen la ejecución del script.

1.7.1 Gestión estándar de errores

Podemos, mediante las prestaciones normalizadas de PHP, establecer el nivel de errores que queremos que se nos muestre durante la ejecución de un script, de modo que los detectemos y podamos corregir lo que sea necesario. Para ello recurrimos a la directiva ***error_reporting*** del fichero de configuración (**`php.ini`**). Esta recibe valores constantes según la siguiente tabla:

CONSTANTES DE ERROR		
VALOR	**CONSTANTE**	**ERRORES DETECTADOS**
1	E_ERROR	Errores fatales en tiempo de ejecución. Se detiene la ejecución del script.
2	E_WARNING	Avisos en tiempo de ejecución. La ejecución del script no se detiene, pero puede dar resultados erróneos.
4	E_PARSE	Errores como la falta de un punto y coma, paréntesis que no se cierran, etc.
8	E_NOTICE	Notificaciones en tiempo de ejecución.
16	E_CORE_ERROR	Errores fatales que ocurren durante el arranque inicial de PHP, debido a fallos del propio intérprete.
32	E_CORE_WARNING	Advertencias (errores no-fatales) que ocurren durante el arranque inicial de PHP.
64	E_COMPILE_ERROR	Errores fatales en tiempo de compilación. Es como un **`E_ERROR`**, excepto que es generado por el Motor de Scripting de Zend. (1)

CONSTANTES DE ERROR (Cont.)		
VALOR	**CONSTANTE**	**ERRORES DETECTADOS**
128	E_COMPILE_WARNING	Advertencias en tiempo de compilación (errores no fatales). Es como un **E_WARNING**, excepto que es generado por el Motor de Scripting de Zend. (1)
256	E_USER_ERROR	Error generado por el usuario.
512	E_USER_WARNING	Advertencia generada por el usuario.
1024	E_USER_NOTICE	Anotación generada por el usuario.
2047	E_ALL	Todos los errores y advertencias excepto los correspondientes a la constante **E_STRICT**.
2048	E_STRICT	Noticias de tiempo de ejecución. Habilite este valor para hacer que PHP sugiera cambios en su código que velarán por la mejor interoperabilidad y por mantener la compatibilidad de su código. Este nivel de error está disponible desde la versión 5 del lenguaje.
(1) La tecnología Zend se emplea para compilar scripts de PHP para usos específicos y queda fuera del alcance de este volumen.		

Podemos fijar una constante de la tabla como valor de error_reporting. Entonces el sistema mostrará todos los errores que se produzcan del nivel especificado, o de niveles inferiores, según el valor numérico que aparece en la columna de la izquierda. Así pues, suponga el siguiente valor para la directiva:

```
error_reporting = E_NOTICE
```

Se mostrarán los errores que se engloben en las categorías E_NOTICE, E_WARNING, E_PARSE y E_ERROR, pero no los de niveles superiores.

También podemos configurar esta directiva estableciendo diferentes niveles de error. Por ejemplo, suponga que la asignamos los siguientes valores:

```
error_reporting  =  E_ALL & ~E_NOTICE
```

En este caso se determina que se muestren los errores comprendidos en el nivel E_ALL e inferiores, pero excluyendo, específicamente, los que correspondan al nivel E_NOTICE. Fíjese que ambas constantes van unidas mediante el operador a

nivel de bit &. Se usa este porque, en definitiva, estas constantes son bits de un byte que es el que se almacena en la directiva **`error_reporting`**. Además, el valor E_NOTICE va precedido de un operador de negación para excluirlo de los errores que deben mostrarse. En un capítulo posterior hablaremos de los operadores a nivel de bit que proporciona PHP.

Existen otras dos directivas relativas al tratamiento de errores, en el fichero de configuración, que usted debe conocer. La primera es ***display_errors***. Su valor por defecto es **`On`**, lo que permite que se muestren los errores en la página. Eso está muy bien mientras estamos depurando un script. Sin embargo, una vez que este se pone a disposición del público, conviene desactivar esta directiva. No queda bien que, si se produce un error que se nos haya pasado por alto, este sea visto por los usuarios. Además, los mensajes de error proporcionan cierta información, por otra parte necesaria para la depuración, que una persona con los conocimientos adecuados podría usar como una vulnerabilidad de nuestro sitio.

Por último, considere la directiva ***log_errors***. Cuando se activa (valor **`On`**), los errores se almacenan en un fichero específico en el servidor, llamado, por defecto, ***php_error.log***. El nombre de este fichero, así como su ruta y tamaño máximo, también pueden establecerse en el fichero de configuración.

Pero las directivas de configuración no son la única manera de gestionar errores. Con independencia del valor que tenga la directiva error_reporting, usted puede usar una función homónima en su script para cambiar el nivel de detección de errores para ese script en concreto, así:

```
error_reporting (E_WARNING);
```

Si incluye esta línea al principio de su script (es el sitio más lógico), cuando este se ejecute sólo se mostrarán los errores de nivel E_WARNING y E_ERROR.

1.7.2 Gestión personalizada de errores

Podemos crear funciones de usuario que se ejecuten como respuesta a un error. Para ello usamos la función **`set_error_handler()`**. En su modo más simple, esta recibe el nombre de la función que queramos establecer como gestor de errores. Por ejemplo, suponga el siguiente fragmento de un script:

```
function miGestorPersonalizado(){
      echo "Se ha producido un error.";
}
set_error_handler ("miGestorPersonalizado");

// Resto del script PHP.
```

Cuando se produzca un error, se ejecutará la función definida por el programador **miGestorPersonalizado()**.

Hay dos aspectos importantes que usted debe tener siempre presentes cuando use este sistema:

- En primer lugar, esto inhabilita el gestor estándar de errores de PHP. Ya no importa lo que se le haya asignado a la directiva **error_reporting**.
- Tal como nos dice el manual de PHP, hay determinados niveles de error que no pueden ser tratados mediante un gestor personalizado. Son los siguientes: **E_ERROR**, **E_PARSE**, **E_CORE_ERROR**, **E_CORE_WARNING**, **E_COMPILE_ERROR**, **E_COMPILE_WARNING** y la mayoría de **E_STRICT.**

La función que se usa como gestor de errores recibe, cuando se produce un error, cinco valores, generados por la propia función **set_error_handler()**. Estos se pueden usar para tratar el error de la manera más conveniente. Suponga el siguiente fragmento de código:

```
      function miGestorPersonalizado ($numero, $descripcion,
$fichero, $linea, $contexto){
            echo "Se ha producido un error $descripcion en
$fichero, en la línea $linea. ";
            echo "El número de error es $numero. <br/>";
      }
      set_error_handler ("miGestorPersonalizado");
```

Como ve, mediante el uso de estos parámetros es posible obtener información acerca del error. El quinto parámetro, asignado a la variable **$contexto**, es una matriz con todas las variables implicadas.

Durante la ejecución de un script es posible que necesitemos usar diferentes gestores de error. En ese caso, los iremos activando según sean necesarios, con la función **set_error_handler()** que acabamos de ver, y los desactivaremos con la función ***restore_error_handler()***. Observe el siguiente fragmento de ejemplo:

```
      function miPrimerGestor (){
            // Cuerpo de la función
      }
      function miSegundoGestor (){
            // Cuerpo de la función
      }
      set_error_handler ("miPrimerGestor");
      //Desde aquí, cuando se produzca un error, se ejecutará
miPrimerGestor().
```

```
      /*****
      Código PHP
      *****/
      set_error_handler ("miSegundoGestor");
      //Desde aquí, cuando se produzca un error, se ejecutará
miSegundoGestor().
      /*****
      Más código PHP
      *****/
      restore_error_handler();
      //Desde aquí, cuando se produzca un error, se ejecutará,
de nuevo, miPrimerGestor().
      /*****
      Más código PHP
      *****/
      restore_error_handler();
      // Desde aquí, cuando se produzca un error, se ejecutará
el gestor de errores estándar de PHP.
```

Observe cómo se activa la función **`miPrimerGestor()`** como gestor de errores. Después, tras algunas instrucciones, se decide que los errores serán tratados por la función **`miSegundoGestor()`**. Cuando usamos por primera vez **`restore_error_handler()`** se vuelve un paso atrás, estableciendo, de nuevo, **`miPrimerGestor()`** como gestor de errores. Al usar de nuevo **`restore_error_handler()`**, como ya no se han establecido más gestores personalizados, se pasa el control al gestor estándar del que hablamos en el apartado anterior.

Y una cosa más. Si una función va precedida del operador de control de errores (**`@`**), no generará un error. Si no puede ejecutarse adecuadamente, no dará el resultado esperado, pero este fallo no será capturado ni por el gestor estándar ni por los gestores personalizados, si los hubiera.

1.7.3 Depuración en tiempo de ejecución

PHP 5 incluye algunas herramientas útiles para la depuración de un script en tiempo de ejecución. Una de estas es la función ***debug_backtrace()***. Esta no recibe ningún argumento, pero devuelve una matriz asociativa con información acerca del script en curso. Los elementos de dicha matriz son los siguientes:

- Clave **function**. Contiene el nombre de la función actual, si se está ejecutando alguna en el momento de invocar a **`debug_backtrace()`**.
- Clave **line**. Contiene el número de línea actual.
- Clave **file**. Contiene el nombre y la ruta del script en curso.

- Clave **class**. Contiene el nombre de la clase con la que se está trabajando, si es el caso (véase el capítulo 8 para saber lo que necesita sobre clases).
- Clave **type**. El tipo de método que se está ejecutando, si estamos trabajando con un objeto (véase el capítulo 8 para saber lo que necesita sobre Programación Orientada a Objetos). Si es un método normal, esta clave contiene el valor `->`, si es un método estático, contiene `::` y, por último, si es una función, en lugar de un método, no contiene nada.
- Clave **args**. Si la llamada a **`debug_backtrace()`** se efectúa dentro de una función, esta clave contiene una matriz con los argumentos de la misma. Si se ejecuta en un archivo incluido (con **`include()`** o **`require()`**) contiene una lista de los archivos incluidos en el que hizo la llamada.

Para ver cómo funciona, suponga el siguiente fragmento de ejemplo:

```
function miFuncion($cadena){
    echo "Hola: $cadena. <br />";
    var_dump(debug_backtrace());
}

miFuncion("Pepe");
```

El resultado será similar al siguiente:

```
array(1) {
  [0]=>
  array(4) {
    ["file"]=>
    string(77) "C:\Documents and Settings\Jose\Mis
documentos\Mis webs dinamicas\comillas.php"
    ["line"]=>
    int(8)
    ["function"]=>
    string(9) "miFuncion"
    ["args"]=>
    array(1) {
      [0]=>
      &string(4) "Pepe"
    }
  }
}
```

Como vemos, esta función se usa para determinar el estado de un script en un momento dado. Como complemento existe una función similar, llamada ***debug_print_backtrace()***. Esta tampoco recibe ningún argumento. Como resultado,

genera lo que se conoce como un ***backtrace PHP***, que es una lista de las llamadas a función, archivos incluidos y el material evaluado mediante **`eval()`**. Se usa para determinar el flujo de un script, hasta llegar a un punto determinado de la ejecución.

1.8 COMANDOS DEL SISTEMA OPERATIVO

Desde un script de PHP se pueden ejecutar comandos del sistema operativo y dirigir el resultado a la salida del navegador o a donde convenga. Para ello encerramos el comando deseado entre dos signos de acento grave (`` ` ``). Este se corresponde con el código ASCII 96 y no debe confundirse con el acento agudo, al que estamos habituados los hispano-parlantes, ni con la comilla simple. Para saber más acerca de códigos ASCII consulte el apéndice A. Esta prestación es especialmente útil cuando se tienen ciertas tareas programadas en el propio sistema operativo del servidor y deben ejecutarse como respuesta a un evento de usuario, del sitio, cronológico, etc.

Para saber cómo funciona esto, suponga la siguiente línea de código en un script:

```
echo `ls -l -a`;
```

En este caso, se ejecutará un listado de archivos del directorio en curso, pues tal es la instrucción de Linux incluida en el script PHP. Y esto nos lleva a un comentario importante, que no queremos que se nos pase por alto: al ejecutarse una instrucción del sistema operativo, deberá estar escrita, como es lógico, siguiendo la sintaxis propia del sistema que tengamos corriendo en el servidor, no en el cliente. Si el script va a correr, por ejemplo, sobre un servidor Linux, las instrucciones deberán ser de este sistema operativo, con independencia de que un cliente pueda conectarse desde una plataforma Windows, o Mac, o cualquier otra.

El acento grave empleado para esta finalidad se conoce como ***operador de ejecución***, por razones obvias.