

Colección
Recursos Informáticos

PHP 5.5

**Desarrollar un sitio Web
dinámico e interactivo**

Olivier HEURTEL

Descarga
www.ediciones-eni.com

 **INFORMÁTICA TÉCNICA**


ediciones

Buscar

Índice

Notas y marca páginas

Favorito

Índice

Información

Título, autor...

Introducción

Objetivo del libro

Breve historia de PHP

¿Dónde conseguir PHP?

Convenciones de escritura

Información general sobre PHP

Variables, constantes, tipos y matrices

Operadores

Estructuras de control

Funciones y clases

Gestión de formularios

Acceder a las bases de datos

Administrar las sesiones

Enviar un correo electrónico

Gestión de archivos

Administrar los errores en un script PHP

Anexo

PHP 5.5

Desarrollar un sitio Web dinámico e interactivo

Este libro sobre PHP 5.5 está dirigido a diseñadores y desarrolladores que desean utilizar PHP para desarrollar un **sitio web dinámico e interactivo**.

Después de presentar los **principios básicos** del lenguaje, el autor se centra en las necesidades específicas del desarrollo de sitios dinámicos e interactivos, proporcionando respuestas completas y precisas a las cuestiones más habituales (gestión de **formularios**, acceso a **bases de datos**, gestión de **sesiones**, envío de **mensajes de correo electrónico**...).

Para todas las características detalladas, se presentan y comentan **numerosos ejemplo de código**. Este libro de referencia, a la vez completo y conciso, le permite ir directamente al grano: es el libro ideal para iniciarse en PHP.

Los ejemplos que aparecen en el libro se pueden descargar en esta página.

Los capítulos del libro:

Introducción - Información general sobre PHP - Variables, constantes, tipos y matrices - Operadores - Estructuras de control - Funciones y clases - Gestión de formularios - Acceder a las bases de datos - Administrar las sesiones - Enviar un correo electrónico - Gestión de archivos - Administrar los errores en un script PHP - Anexo

Olivier HEURTEL

Después de más de ocho años en una empresa de servicios, donde sucesivamente desempeñó labores de desarrollador, administrador de proyectos y director de proyectos, **Olivier HEURTEL** inició una actividad como consultor/formador independiente especializado en bases de datos (Oracle), desarrollo Web (PHP) y los sistemas de apoyo a las decisiones. También es poseedor del certificado Oracle Certified Professional.

Material de
descarga

Archivos adicionales

Publicación: febrero 2014
Ref. ENI : RIT5.5PHP
ISBN : 9782746087989



Comprar
la versión
impresa



Constantes

1. Definición

La función `define` o la palabra clave `const` (desde la versión 5.3) permiten definir una constante.

Una constante es un área de memoria identificada por un nombre que contiene un valor legible, pero no modificable en el programa.

Sintaxis

```
booleano define(cadena nombre, mixto valor[, booleano sensible_mayúsculas)
```

```
const nombre = valor
```

nombre	Nombre de la constante (ver sección Estructura básica de una página PHP - Normas de denominación en el capítulo Información general sobre PHP).
valor	Valor de la constante.
sensible_mayúsculas	Indica si el nombre de la constante es sensible a mayúsculas y minúsculas (TRUE - valor predeterminado) o no (FALSE).

La función `define` devuelve TRUE en caso de éxito y FALSE en caso de error.

Cualquier tipo de dato escalar (véase la sección Tipos de datos en este capítulo) se puede utilizar como tipo de datos de una constante.

El nombre de una constante no puede comenzar con un \$ ya que este prefijo está reservado al nombre de las variables (véase la sección Variables en este capítulo). Definir una constante cuyo nombre comienza por \$ no genera un error (`define` devuelve TRUE). Sin embargo, cuando se utiliza, la constante se verá como una variable no inicializada.

Una vez creada, una constante no se puede cambiar, ni por una nueva llamada a `define` (devuelve FALSE y deja el valor de la constante inalterado), ni por asignación directa (genera un error de análisis de la secuencia de comandos).

Ejemplo

```
<?php
// Definir una constante (cuyo nombre es por defecto
// sensible a mayúsculas y minúsculas).
define('CONSTANTE','valor de la CONSTANTE');
// Mostrar el valor de la CONSTANTE (=> OK).
echo 'CONSTANTE = ',CONSTANTE,'<br />';
// Mostrar el valor de la constante (=> vacío)
echo 'constante = ',constante;
echo ' => interpretada literalmente<br />';
// Utilización de la palabra clave const (desde la versión 5.3)
const OTRA_CONSTANTE = 'PHP 5.5';
echo 'OTRA_CONSTANTE = ', OTRA_CONSTANTE;
?>
```

Resultado

```
CONSTANTE = valor de CONSTANTE  
constante = constante => interpretada literalmente  
OTRA_CONSTANTE = PHP 5.5
```

Tradicionalmente, los nombres de las constantes se definen en mayúsculas.

Utilizar una constante no definida (o una variable no inicializada) o intentar redefinir una constante ya definida genera un error de nivel `E_NOTICE`. El nivel de error reportado por PHP depende de las directivas de configuración en el archivo `php.ini` (véase el capítulo Administrar los errores en un script PHP). El resultado anterior corresponde a una configuración en la que los errores de nivel `E_NOTICE` no se muestran.

2. Alcance

El alcance de una constante es la secuencia de comandos en la que se define. Por lo tanto, una constante se puede definir en una primera sección de código PHP y utilizarse en otra sección de código PHP del mismo script.

Ejemplo

```
<?php  
// Definir una constante.  
define('NOMBRE','Olivier');  
?>  
<html>  
<body>  
<p>;Hola <b><?php echo NOMBRE; ?></b>!</p>  
</body>  
</html>
```

Resultado

```
;;Hola Olivier!
```

3. Funciones útiles

La función `defined` permite saber si una constante está definida o no.

Sintaxis

```
booleano defined(cadena nombre)
```

`nombre` Nombre de la constante.

`defined` devuelve `TRUE` si la constante está definida y `FALSE` en caso contrario.

Ejemplo

```
<?php  
// Probar si la constante CONSTANTE está definida.  
$ok = defined('CONSTANTE');  
if ($ok) {  
    echo 'CONSTANTE está definida.<br />';  
} else {
```

```

    echo 'CONSTANTE no está definida.<br />';
};
// Definir la constante CONSTANTE
define('CONSTANTE','valor de la CONSTANTE');
// Probar si la constante CONSTANTE está definida.
$ok = defined('CONSTANTE');
if ($ok) {
    echo 'CONSTANTE está definida.<br />';
} else {
    echo 'CONSTANTE no está definida.<br />';
};
?>

```

Resultado

```

CONSTANTE no está definida.
CONSTANTE está definida.

```



En este ejemplo se utiliza la estructura de control `if`, que permite probar una condición y actuar en consecuencia (véase el capítulo Estructuras de control).

La función `constant` devuelve el valor de una constante cuyo nombre se pasa como parámetro.

Sintaxis

```

mixto constant(cadena nombre)

```

Con

`nombre` Nombre de la constante.

Esta función es útil para recuperar el valor de una constante cuyo nombre no se conoce a priori.

Ejemplo

```

<?php
// definir el nombre de la constante en una variable
$nombreConstante = 'CONSTANTE';
// definir el valor de la constante
define($nombreConstante,'valor de la CONSTANTE');
// mostrar el valor de la constante
echo $nombreConstante,' = ',constant($nombreConstante);
?>

```

Resultado

```

CONSTANTE = valor de la CONSTANTE

```

Otras funciones permiten conocer el tipo de una constante (véase la sección Tipos de datos - Funciones útiles).

Variables

Una variable es un área de memoria identificada por un nombre que contiene un valor legible o modificable en el programa.

1. Inicialización y asignación

En PHP, las variables se identifican por el prefijo \$ seguido de un nombre que cumple con las reglas de denominación presentadas en el capítulo Información general de PHP.

El nombre de las variables es sensible a mayúsculas y minúsculas: PHP considera \$nombre y \$Nombre como variables diferentes. Este comportamiento es peligroso ya que, en caso de utilizar una sintaxis incorrecta, se crea una nueva variable vacía con un simple error de nivel E_NOTICE que no se puede mostrar (véase el capítulo Administrar los errores en un script PHP). Por tanto, es esencial adoptar una convención de denominación y respetarla. Algunas sugerencias:

- todo en minúsculas (\$nombre).
- primera letra en mayúscula y el resto en minúsculas (\$Nombre).
- primera letra de cada palabra en mayúscula y el resto en minúsculas (\$NombreDePila).

Las variables de PHP se definen automáticamente la primera vez que se utilizan. No hay instrucciones específicas para crear una variable.

Las variables de PHP se escriben de forma automática; cada vez que se asigne un valor a una variable, el tipo de variable se define o se redefine automáticamente (véase la sección Tipos de datos).


Un valor se puede asignar a una variable mediante el operador de asignación "=" (véase el capítulo Operadores para obtener una lista de todos los operadores).

Ejemplo

```
<?php
// Inicializar una variable $nombre.
$nombre = 'Olivier';
// Mostrar la variable $nombre.
echo '$nombre = ', $nombre, '<br />';
// Mostrar la variable $Nombre.
echo '$<b>N</b>ombre = ', $Nombre;
echo ' => vacío (es otra variable)<br />';
// Modificar el valor (y el tipo) de la variable $nombre.
$nombre = 123;
// Mostrar la variable $nombre.
echo '$nombre = ', $nombre, '<br />';
?>
```

Resultado (los errores de nivel E_NOTICE no se muestran)

```
$nombre = Olivier
$Nombre = => vacío (es otra variable)
$nombre = 123
```

 A lo largo de este libro, tendremos la oportunidad de conocer las variables definidas automáticamente por PHP y que contienen valores relativos al entorno, a PHP, a formularios, cookies...

2. Alcance y duración

El alcance de una variable es la secuencia de comandos en la que se define. Por lo tanto, una variable se puede definir en una primera sección de código PHP y utilizarse en otra sección de código PHP del mismo script.

La duración de una variable es el tiempo de ejecución del script. Cuando el script termina, las variables se eliminan. Si más adelante se llama al mismo script, se definen nuevas variables.

Ejemplo

```
<?php
// Mostrar el contenido de la variable $nombre.
echo '$nombre = ', $nombre, '<br />';
// Inicializar la variable $nombre.
$nombre = 'Olivier';
// Mostrar de nuevo el contenido de la variable $nombre.
echo '$nombre = ', $nombre, '<br />';
?>
```

Resultado de la primera llamada del script

```
$nombre =
$nombre = Olivier
```

Resultado de la segunda llamada del script

```
$nombre =
$nombre = Olivier
```

Entre las dos llamadas, se ha eliminado la variable. Al comienzo de la segunda llamada, no contiene el valor que tenía al final de la primera llamada (no es la misma variable).



En el capítulo Gestión de sesiones, veremos cómo conservar el valor de una variable más allá de la ejecución del script o cómo transmitir el valor de una variable de un script a otro.

3. Funciones útiles

PHP ofrece una serie de funciones útiles en las variables:

Nombre	Función
empty	Indica si una variable está vacía o no.
isset	Indica si una variable está definida o no.
unset	Elimina una variable.
var_dump	Muestra la información sobre una variable (tipo y valor).

empty

La función empty permite probar si una variable está vacía o no.

Sintaxis

booleano `empty(mixto variable)`

`variable` Variable que se va a probar.

`empty` devuelve TRUE si la variable está definida y FALSE en caso contrario.

Una variable se considera vacía si no ha sido asignada o si contiene una cadena vacía (""), una cadena igual a 0 ("0"), 0, NULL o FALSE.

Desde la versión 5.5, la función `empty` se puede utilizar para probar si una expresión está vacía o no.

Ejemplo

```
<?php
// Prueba de una variable no inicializada.
$está_vacia = empty($variable);
echo '$variable no inicializada<br />';
if ($está_vacia) {
    echo '=> $variable está vacía.<br />';
} else {
    echo '=> $variable no está vacía.<br />';
}
// Prueba de una variable que contiene una cadena vacía.
$variable = '';
$está_vacia = empty($variable);
echo '$variable = \'\'<br />';
if ($está_vacia) {
    echo '=> $variable está vacía.<br />';
} else {
    echo '=> $variable no está vacía.<br />';
}
// Prueba de una variable que contiene una cadena igual a 0.
$variable = '0';
$está_vacia = empty($variable);
echo '$variable = \'\'',$variable,'\'<br />';
if ($está_vacia) {
    echo '=> $variable está vacía.<br />';
} else {
    echo '=> $variable no está vacía.<br />';
}
// Prueba de una variable que contiene 0.
$variable = 0;
$está_vacia = empty($variable);
echo '$variable = \'',$variable,'\'<br />';
if ($está_vacia) {
    echo '=> $variable está vacía.<br />';
} else {
    echo '=> $variable no está vacía.<br />';
}
// Prueba de una variable que contiene una cadena no vacía.
$variable = 'x';
$está_vacia = empty($variable);
echo '$variable = \'\'',$variable,'\'<br />';
if ($está_vacia) {
    echo '=> $variable está vacía.<br />';
} else {
    echo '=> $variable no está vacía.<br />';
}
```



```
?>
```

Resultado

```
$variable no inicializada
=> $variable está vacía.
$variable = ''
=> $variable está vacía.
$variable = '0'
=> $variable está vacía.
$variable = 0
=> $variable está vacía.
$variable = 'x'
=> $variable no está vacía.
```

isset

La función `isset` permite probar si una variable está definida o no.

Sintaxis

```
booleano isset(mixto variable[,...])
```

`variable` Variable que se va a probar; pueden ser varias, separadas por una coma.

`isset` devuelve `TRUE` si la variable está definida y `FALSE` en caso contrario.

Si se facilitan varios parámetros, la función devuelve `TRUE` únicamente si se definen todas las variables.

Una variable se considera como no definida si no se ha visto asignado o si contiene `NULL`. A diferencia de la función `empty`, una variable que contiene una cadena vacía (""), una cadena igual a 0 ("0"), un 0 o `FALSE`, no se considera como no definida.

Ejemplo

```
<?php
// Prueba de una variable no inicializada.
$está_definida = isset($variable);
echo '$variable no inicializada<br />';
if ($está_definida) {
    echo '=> $variable está definida.<br />';
} else {
    echo '=> $variable no está definida.<br />';
}
// Prueba de una variable que contiene una cadena vacía.
$variable = '';
$está_definida= isset($variable);
echo '$variable = \'\'<br />';
if ($está_definida) {
    echo '=> $variable está definida.<br />';
} else {
    echo '=> $variable no está definida.<br />';
}
// Prueba de una variable que contiene una cadena igual a 0.
$variable = '0';
$está_definida = isset($variable);
echo '$variable = \''.$variable.'\'<br />';
if ($está_definida) {
```

```

    echo '=> $variable está definida.<br />';
} else {
    echo '=> $variable no está definida.<br />';
}
// Prueba de una variable que contiene 0.
$variable = 0;
$está_definida = isset($variable);
echo '$variable = ', $variable, '<br />';
if ($está_definida) {
    echo '=> $variable está definida.<br />';
} else {
    echo '=> $variable no está definida.<br />';
}
// Prueba de una variable que contiene una cadena no vacía.
$variable = 'x';
$está_definida = isset($variable);
echo '$variable = \'', $variable, '\'  


```

Resultado

```

$variable no inicializada
=> $variable no está definida.
$variable = ''
=> $variable está definida.
$variable = '0'
=> $variable está definida.
$variable = 0
=> $variable está definida.
$variable = 'x'
=> $variable está definida.

```

unset

La función `unset` permite eliminar una variable.

Sintaxis

```
unset(mixto variable[, ...])
```

`variable` Variable que se va a eliminar (para eliminar varias, deben estar separadas por una coma).

`unset` acepta una lista de variables.

Después de la eliminación, la variable se encuentra en el mismo estado que si no hubiera sido asignada. El uso de la función `isset` en una variable eliminada devuelve `FALSE`.

Ejemplo

```

<?php
// Definir una variable.
$variable = 1;
// Mostrar la variable y probar si está definida.

```

```

$está_definida = isset($variable);
echo '$variable = ', $variable, '<br />';
if ($está_definida) {
    echo '=> $variable está definida.<br />';
} else {
    echo '=> $variable no está definida.<br />';
}
// Eliminar la variable.
unset($variable);
// Mostrar la variable y probar si está definida.
$está_definida = isset($variable);
echo '$variable = ', $variable, '<br />';
if ($está_definida) {
    echo '=> $variable está definida.<br />';
} else {
    echo '=> $variable no está definida.<br />';
}
?>

```

Resultado

```

$variable = 1
=> $variable está definida.
$variable =
=> $variable no está definida.

```



Al asignar un 0 o una cadena vacía a una variable, no se borra.

var_dump

La función `var_dump` muestra información sobre una variable (tipo y contenido).

Sintaxis

```
var_dump(mixto variable)
```

variable	Variable que se va a mostrar (pueden ser varias, separadas por una coma).
----------	---

La función `var_dump` es especialmente interesante en las fases de desarrollo.

Ejemplo

```

<?php
// mostrar la información sobre una variable no inicializada
var_dump($variable);
// inicializar la variable con un número entero
$variable = 10;
// mostrar la información sobre una variable
echo '<br />';
var_dump($variable);
// modificar el valor (y el tipo) de la variable
$variable = 3.14; // número decimal
// mostrar la información sobre una variable
echo '<br />';
var_dump($variable);
// modificar el valor (y el tipo) de la variable

```


```
$variable = 'abc'; // cadena de caracteres
// mostrar la información sobre la variable
echo '<br />';
var_dump($variable);
?>
```

Resultado

```
NULL
int(10)
float(3.14)
string(3) "abc"
```

Para una variable no inicializada, `var_dump` devuelve `NULL`. Para un número, `var_dump` indica el tipo (`int` = entero, `float` = número decimal) seguido por el valor entre paréntesis. Para una cadena, `var_dump` indica el tipo (`string`), seguido de la longitud entre paréntesis, seguido por el valor entre comillas.

PHP también ofrece las funciones `print_r` y `var_export`, que son similares a la función `var_dump`. La función `print_r` muestra o devuelve el contenido de la variable en una forma más legible, sin mencionar el tipo de datos. La función `var_export` muestra o devuelve una cadena que ofrece un código PHP de definición de la variable; esta función apareció en la versión 5.

 En la sección Tipos de datos de este capítulo, estudiaremos otras funciones que permiten determinar el tipo de una variable y realizar conversiones de tipos (de número a cadena, de cadena a número...).

4. Variable dinámica (o variable variable)

PHP ofrece una función de variable dinámica (también llamada variable variable) útil en determinadas situaciones.

El principio consiste en utilizar una variable que almacena el nombre de otra variable y, a continuación se hace referencia a ella con una notación, del tipo `$$variable` o `${$variable}`. Con esta notación, la `$variable` "interior" se sustituye por el valor de la variable `$variable` (valor por ejemplo) que se utiliza como un nombre de variable por la `$` "exterior" (es decir, `$valor` en nuestro ejemplo).

Ejemplo

```
<?php
$una_variable = 10;
$nombre_variable = 'una_variable';
echo '$una_variable = ', $una_variable, '<br />';
echo '$nombre_variable = ', $nombre_variable, '<br />';
echo '$$nombre_variable = ', $$nombre_variable, '<br />';
?>
```

Resultado

```
$una_variable = 10
$nombre_variable = una_variable
$$nombre_variable = 10
```

Tipos de datos

1. Tipos disponibles

PHP dispone de cuatro tipos de datos escalares (sólo pueden contener un valor), dos tipos compuestos (pueden contener varios valores) y dos tipos especiales:

- Tipos escalares:
 - número entero
 - número de punto flotante
 - cadena de caracteres
 - booleano
- Tipos compuestos:
 - matriz (véase la sección Matrices de este capítulo)
 - objeto (véase el capítulo Funciones y clases)
- Tipos especiales:
 - NULL
 - Recurso

Entero

El tipo entero (*integer*) permite almacenar un número entero de 32 bits, cuyos valores estén comprendidos entre -2 147 483 648 (-2^{31}) y +2 147 483 647 ($+2^{31}-1$).

En caso de desbordamiento de capacidad en un cálculo, el resultado se convierte automáticamente a un número de punto flotante.

Número de punto flotante

El tipo número de punto flotante (*float*) permite almacenar un número decimal en un rango de valores dependiente de la plataforma (normalmente del orden de 10^{-308} a 10^{+308}).

Este número se puede expresar en notación decimal x.y (por ejemplo, 123.456) o en notación científica x.yEz o x.yez (por ejemplo 1.23456E2).

En caso de conversión de un número de punto flotante en entero, el número se trunca (no se redondea) al entero más próximo (1.9 da 1, por ejemplo). En caso de desbordamiento de capacidad, no aparece ningún mensaje, pero el valor a la llegada está definido (normalmente 0).



PHP ofrece bibliotecas especiales (también llamadas Librerías) para manejar los números de gran tamaño (bibliotecas BC o GMP).

Cadena de caracteres

El tipo cadena de caracteres (string) permite almacenar cualquier secuencia de caracteres de un byte

(código ASCII entre 0 y 255), sin limitación de tamaño.

Una expresión literal de tipo cadena de caracteres se puede especificar entre comillas ("esto es una cadena") o entre apóstrofes ('esto también es una cadena') con importantes diferencias en el comportamiento, que se describen a continuación.

Las comillas presentes en una cadena delimitada por comillas o los apóstrofes presentes en una cadena delimitada por apóstrofes deben "escaparse", es decir, deben ir precedidos por una barra invertida (\). Además, una barra invertida al final de esta cadena, justo antes de la comilla o el apóstrofo final, también debe escaparse mediante una barra invertida.

Ejemplo

```
<?php
echo 'It\'s raining.<br />';
echo "Yo digo \"hola\".<br />";
?>
```

Resultado

```
It's raining.
Yo digo "hola".
```

Una cadena se puede introducir en varias líneas, pero se muestra en una sola línea en el navegador. Para mostrar una cadena en varias líneas en el navegador, debe insertar una etiqueta
.

```
<?php
$cadena = '<p>Yo me llamo Olivier
y vivo en Francia.</p>';
echo $cadena;
$cadena = '<p>Yo me llamo Olivier<br />
y vivo en Francia.</p>';
echo $cadena;
?>
```

Resultado

```
Yo me llamo Olivier y vivo en Francia.
```

```
Yo me llamo Olivier
y vivo en Francia.
```

Cuando una cadena está delimitada por comillas, cualquier secuencia de caracteres que comience con el signo \$ se interpreta como una variable y se sustituye por el valor de la variable: es el mecanismo de sustitución de variables por su valor. Esta característica, muy práctica, no funciona con las cadenas delimitadas por apóstrofes (primera diferencia entre los dos tipos de cadenas).

Ejemplo

```
<?php
$nombre = 'Olivier';
echo "Yo me llamo $nombre.<br />";
echo 'Yo me llamo $nombre.<br />'; // no funciona
?>
```

Resultado

```
Yo me llamo Olivier.
```

Yo me llamo \$nombre.

En algunos casos, es posible que este comportamiento no sea el deseado. Basta con escapar el signo \$ con la barra invertida (\) para que se comporte como un \$.

Ejemplo

```
<?php
$nombre = 'Olivier';
echo "\$nombre = $nombre";
?>
```

Resultado

\$nombre = Olivier

En otros casos, el comportamiento puede ser deseado si existe la necesidad de adjuntar un texto adicional después del nombre de la variable.

Ejemplo

```
<?php
$fruta = 'manzana';
echo "Una $fruta no es cara.<br />";
echo "Dos $frutas cuestan el doble.<br />";
?>
```

Resultado

Una manzana no es cara.
Dos cuestan el doble.

En este ejemplo, PHP interpreta la "s" plural como perteneciente a la secuencia de caracteres situada detrás del \$ y, por lo tanto, a la variable \$frutas que se reconoce y se sustituye por su valor (vacío, ya que la variable no se ha inicializado).

La solución consiste en delimitar el nombre de la variable por llaves en forma de {\$variable} o \${variable}. Antes de la versión 5.1.1, el carácter de escape barra diagonal inversa se podía utilizar antes de la primera llave para cancelar esta interpretación de las llaves. Desde la versión 5.1.1, este carácter de escape ya no funciona para la llave, por lo tanto, a partir de esta versión utilizamos la notación {{ \$fruta }} en lugar de \{ \$fruta } (véase el ejemplo más adelante). De todos modos, una llave que no va precedida o seguida por un \$ no escapado se deja sin cambios.

Ejemplo

```
<?php
$fruta = 'manzana';
echo "Una $fruta no es cara.<br />";
echo "Dos {$fruta}s cuestan el doble.<br />";
echo "Tres ${fruta}s cuestan el triple.<br />";
echo "{\$fruta} = {{ $fruta }}.<br />";
?>
```

Resultado

Una manzana no es cara.
Dos manzanas cuestan el doble.
Tres manzanas cuestan el triple.

```
{ $fruta } = { manzana }.
```

- No hay mecanismo de sustitución equivalente para las constantes; esta es una razón válida para utilizar variables en lugar de constantes reales.

Además, se pueden utilizar otras secuencias de escape en las cadenas delimitadas por comillas, pero no en las delimitadas por apóstrofes (segunda diferencia entre los dos tipos de cadenas).

Secuencia	Valor
\n	Salto de línea (= LF = código ASCII 10)
\r	Retorno de carro (= CR = código ASCII 13)
\t	Tabulación (= HT = código ASCII 9)
\v	Tabulación vertical (= VT = código ASCII 11)
\e	Escape (= ESC = código ASCII 27)
\f	Página siguiente (= FF = código ASCII 12)
\\	\ (ya explicado)
\\$	\$ (ya explicado)
\nnn	El carácter designado por el código ASCII nnn expresado en octal
\xnn	El carácter designado por el código ASCII nn expresado en hexadecimal

Ejemplo

```
<?php
echo "Yo me llamo Olivier.<br />\n";
echo "Yo me llamo \117\154\151\166\151\145\162.";
?>
```

Resultado

```
Yo me llamo Olivier.
Yo me llamo Olivier.
```

- Recordatorio: un salto de línea en la fuente de la página enviada al navegador no causa ningún salto de línea en la página. Este es el caso de la secuencia "\n", utilizada en nuestro ejemplo. Aquí, la etiqueta
 provoca el salto de línea en la página actual.

Es posible acceder al enésimo carácter en una cadena usando la notación `$x[i]`, `$x`, designando la variable de tipo cadena e `i` el número del carácter (el primer carácter lleva el número 0). También se pueden utilizar las llaves.

Ejemplo

```
<?php
$nombre = 'Olivier';
echo $nombre[0], $nombre{6};
?>
```


Resultado

Or

PHP es capaz de convertir una cadena en un número (entero o decimal) con las siguientes reglas:

- Si el primer carácter no "blanco" (que no sea el espacio, la tabulación, LF, CR) no es una cifra, ni un punto ni el signo menos, la cadena se evalúa a 0 (entero).
- De lo contrario, PHP extraerá todos los caracteres no "blancos" del principio de la cadena hasta que encuentre un carácter no numérico (es decir, que no esté entre 1 y 9, que no sea un punto, el signo "menos" y el símbolo científico "e" o "E"); la secuencia obtenida se convierte en un entero (sin punto ni símbolo científico) o decimal (en caso de presencia de un punto o un símbolo científico).

Ejemplo

```
<?php
echo '1 + "1" = ',var_dump(1 + "1"),'<br />';
echo '1 + "1.5" = ',var_dump(1 + "1.5"),'<br />';
echo '1 + "1.5E2" = ',var_dump(1 + "1.5E2"),'<br />';
echo '1 + "1e3" = ',var_dump(1 + "1e3"),'<br />';
echo '1 + 1abc = ',var_dump(1 + "1abc"),'<br />';
echo '1 + "1.5abcd" = ',var_dump(1 + "1.5abcd"),'<br />';
echo '1 + "1.5 abcd" = ',var_dump(1 + "1.5 abcd"),'<br />';

echo '1 + ".5" = ',var_dump(1 + ".5"),'<br />';
echo '1 + "-5" = ',var_dump(1 + "-5"),'<br />';
echo '1 + " \t\n\r 5" = ',var_dump(1 + " \t\n\r 5"),'<br />';
echo '1 + "abc1" = ',var_dump(1 + "abc1"),'<br />';
?>
```

Resultado

```
1 + "1" = int(2)
1 + "1.5" = float(2.5)
1 + "1.5E2" = float(151)
1 + "1e3" = float(1001)
1 + 1abc = int(2)
1 + "1.5abcd" = float(2.5)
1 + "1.5 abcd" = float(2.5)
1 + ".5" = float(1.5)
1 + "-5" = int(-4)
1 + " \t\n\r 5" = int(6)
1 + "abc1" = int(1)
```

El último ejemplo muestra que una cadena que no comienza con un carácter numérico se convierte a un número entero igual a 0.

Estos mecanismos de conversión son muy prácticos en algunos casos, pero también pueden conducir a problemas difíciles de detectar porque PHP rechaza cantidades tan pequeñas PHP se niegue a compilar la secuencia de comandos.


Booleano

El tipo booleano (*boolean*) puede tomar dos valores: TRUE (o *true*) y FALSE (o *false*).

Este tipo de datos se utiliza principalmente en las estructuras de control para probar una condición (ver capítulo Estructuras de control).

PHP es capaz de convertir cualquier tipo de datos en booleano según las siguientes reglas:

Valor	Resultado de la conversión
número entero 0 número decimal 0.000... cadena vacía ("") cadena igual a 0 ("0") matriz vacía objeto vacío constante NULL (véase el "tipo" Null)	FALSE
todo lo demás	TRUE

 Un valor igual a -1 se convierte en TRUE con PHP.

Por el contrario, PHP es capaz de hacer las conversiones siguientes:

	TRUE	FALSE
Booleano -> Número	1	0
Booleano -> Cadena	"1"	" " (cadena vacía)

Teniendo en cuenta la lógica de conversión que se ha indicado anteriormente, es posible probar cualquier variable como un valor lógico (PHP se encarga de la conversión). Este funcionamiento suele ser práctico, pero puede dar lugar a errores difíciles de detectar.

El "tipo" NULL

Este tipo, introducido en PHP 4, es algo especial y corresponde al tipo de una variable utilizada sin haber sido inicializada. Tiene un valor único, el valor `NULL` definido por la constante `NULL` (`onull`).

Si se convierte en booleano, `NULL` toma el valor `FALSE`.

Ejemplo

```
<?php
var_dump($variable_no_definida);
?>
```

Resultado

```
NULL
```

El tipo recurso (recurso)

Este tipo genérico, introducido en PHP 4 es un poco particular y es una referencia a un recurso externo: archivo abierto, con base de datos, etc.

En varias ocasiones en este libro, tendremos la oportunidad de presentar funciones que permiten manipular estos datos de tipo recurso.

2. Conversiones

PHP es capaz de realizar las conversiones automáticas implícitas de tipo, de acuerdo con las normas presentadas en la sección anterior Tipos disponibles.

Cuando un valor/expresión se asigna a una variable, la variable pasa a ser el tipo de valor/expresión.

Para determinar el tipo de una expresión compuesta de operandos de tipos diferentes, PHP evalúa (pero no convierte) los operandos en función de los operadores procesados en el orden de precedencia (véase el capítulo Operadores - Precedencia de los operadores). Por ejemplo, los dos operandos utilizados en una suma se evalúan en número, mientras que dos operandos utilizados con el operador de concatenación (véase el capítulo Operadores - El operador de cadena) se evalúan en una cadena.

Ejemplo

```
<?php
$número = 123;
$cadena = '456abc';
echo '$número + $cadena= ';
var_dump($número + $cadena);
echo '<br />';
echo '$número . $cadena = ';
var_dump($número . $cadena);
echo '<br />';
echo '$número = ';
var_dump($número);
echo '<br />';
echo '$cadena = ';
var_dump($cadena);
?>
```

Resultado

```
$número + $cadena = int(579)
$número . $cadena = string(9) "123456abc"
$número = int(123)
$cadena = string(6) "456abc"
```

En el primer ejemplo, la variable \$cadena se evalúa en número por ser del tipo esperado por el operador "+", mientras que en el segundo ejemplo, es \$número el que se evalúa en cadena por ser del tipo esperado por el operador "." (concatenación). Por contra, las últimas dos visualizaciones muestran que las variables en cuestión no se han convertido durante las operaciones: conservan su tipo respectivo original.

Además, PHP ofrece una notación y una función para realizar una conversión manual explícita.

Notación

La notación consiste en indicar el nombre del tipo deseado entre paréntesis antes de la expresión que se desea convertir. Los valores permitidos son los siguientes:

Notación	Conversión en
(int) o (integer)	entero
(bool) o (boolean)	booleano
(real), (double) o (float)	número de punto flotante

(string)	cadena
(array)	matriz
(object)	objeto

Ejemplo

```
<?php
echo ' (float)"1abc" = ',var_dump((float)"1abc"),'<br />';
echo ' (float)"1.5abc" = ',var_dump((float)"1.5abc"),'<br />';
echo ' (float)"abc1" = ',var_dump((float)"abc1"),'<br />';
echo ' (int)1.7 = ',var_dump((int)1.7),'<br />';
echo ' (int)TRUE = ',var_dump((int)TRUE),'<br />';
echo ' (int)FALSE = ',var_dump((int)FALSE),'<br />';
echo ' (bool)-1 = ',var_dump((bool)-1),'<br />';
echo ' (bool)0 = ',var_dump((bool)0),'<br />';
echo ' (bool)1 = ',var_dump((bool)1),'<br />';
echo ' (bool)"" = ',var_dump((bool)""),'<br />';
echo ' (bool)"0" = ',var_dump((bool)"0"),'<br />';
echo ' (bool)"1" = ',var_dump((bool)"1"),'<br />';
echo ' (bool)"a" = ',var_dump((bool)"a"),'<br />';
?>
```

Resultado

```
(float)"1abc" = float(1)
(float)"1.5abc" = float(1.5)
(float)"abc1" = float(0)
(int)1.7 = int(1)
(int)TRUE = int(1)
(int)FALSE = int(0)
(bool)-1 = bool(true)
(bool)0 = bool(false)
(bool)1 = bool(true)
(bool)"" = bool(false)
(bool)"0" = bool(false)
(bool)"1" = bool(true)
(bool)"a" = bool(true)
```

Estos ejemplos permiten encontrar las reglas de conversión mencionadas anteriormente.

Función de conversión

La función `settype` permite convertir una variable de un tipo a otro.

Sintaxis

```
booleano settype (mixto variable, cadena tipo)
```

variable Variable que se va a convertir.

tipo Tipo deseado usando uno de los siguientes valores:
 boolean o bool (conversión en booleano)
 integer o int (conversión en entero)
 double o float (conversión en número de punto
 flotante)
 string (conversión en cadena de caracteres)

array (conversión en matriz)
 object (conversión en objeto)
 null (conversión en NULL)

settype devuelve TRUE en caso de éxito y FALSE en caso de error.

Ejemplo

```
<?php
$x = '1abc';
settype($x,'integer');
echo '\1abc\' convertido en entero = ',var_dump($x),'<br />';
$x = 1.7;
settype($x,'integer');
echo '1.7 convertido en entero = ',var_dump($x),'<br />';
$x = TRUE;
settype($x,'string');
echo 'TRUE convertido en cadena = ',var_dump($x),'<br />';
$x = '0';
settype($x,'boolean');
echo '\0\' convertido en booleano = ',var_dump($x),'<br />';
$x = -1;
settype($x,'boolean');
echo '-1 convertido en booleano = ',var_dump($x),'<br />';
?>
```

Resultado

```
'1abc' convertido en entero = int(1)
1.7 convertido en entero = int(1)
TRUE convertido en cadena = string(1) "1"
'0' convertido en booleano = bool(false)
-1 convertido en booleano = bool(true)
```



En general, es recomendable elegir la conversión explícita: el código es más legible, más fácil de mantener y desarrollar.

3. Funciones útiles

Además, PHP tiene varias funciones útiles para el tipo de variables:

Nombre	Función
is_*	Indica si la variable es del tipo dado por *: array = matriz; bool = booleano; double, float, real = número de punto flotante; int, integer, long = entero; null = tipo NULL; numeric = entero o número de punto flotante o una cadena que contiene un número (entero o decimal); object = objeto; string = cadena;

	resource = recurso; scalar = tipo escalar.
strval	Convierte una variable en cadena.
floatval doubleval	Convierte una variable en número de punto flotante.
Intval	Convierte una variable en entero.
Boolval	Convierte una variable en booleano.

is_*

La función `is_*` permite probar si una variable es de un tipo particular.

Sintaxis

```
booleano is_* (mixto variable)
```

variable Variable que se va a probar.

Las declinaciones son las siguientes:

Función	Tipo probado
is_array	matriz
is_bool	booleano
is_double is_float is_real	número de punto flotante
is_int is_integer is_long	entero
is_null	tipo NULL
is_numeric	entero o número de punto flotante o una cadena que contiene un número (entero o decimal).
is_object	objeto
is_string	cadena
is_resource	recurso
is_scalar	tipo escalar

La función devuelve TRUE si la variable está definida y FALSE en caso contrario.

Ejemplo

```
<?php
if (is_null($x)) {
    echo 'De momento, $x es del tipo NULL.<br />';
}
$x = (1 < 2);
if (is_bool($x)) {
    echo '$x = (1 < 2) es del tipo booleano.<br />';
}
```

```

}
$x = '123abc';
if (is_string($x)) {
    echo '$x = "123abc" es del tipo cadena...<br />';
}
if (! is_numeric($x)) {
    echo '... pero no del «tipo» <i>numeric</i>.<br />';
}
$x = '1.23e45';
if (is_numeric($x)) {
    echo 'Por el contrario, $x = "1.23e45" es del «tipo»
<i>numeric</i>.<br />';
}
?>

```

Resultado

De momento, \$x es del tipo NULL.
 \$x = (1 < 2) es del tipo booleano.
 \$x = "123abc" es del tipo cadena...
 ... pero no del «tipo» *numeric*.
 Por el contrario, \$x = "1.23e45" es del «tipo» *numeric*.

En este ejemplo, la función `is_numeric` no aplica las mismas reglas para determinar si una cadena contiene un número que a las utilizadas para la conversión. Con la función `is_numeric`, la cadena no debe contener caracteres no numéricos.

strval

La función `strval` devuelve el valor de una variable después de la conversión en cadena.

Sintaxis

cadena `strval(mixto variable)`

`variable` Variable que se va a procesar.

Esta función sólo se aplica a las variables de tipo escalar (no es válida para las matrices u objetos). El tipo de la variable permanece sin cambios.

Ejemplo

```

<?php
$x = TRUE;
echo var_dump($x), ' => ', var_dump(strval($x)), '<br />';
$x = 1.2345;
echo var_dump($x), ' => ', var_dump(strval($x)), '<br />';
?>

```

Resultado

```

bool(true) => string(1) "1"
float(1.2345) => string(6) "1.2345"

```

floatval (o doubleval)

La función `floatval` devuelve el valor de una variable después de la conversión en número de punto

flotante. La función `doubleval` es un alias de la función `floatval`.

Sintaxis

número `floatval(mixto variable)`

variable Variable que se va a procesar.

Esta función sólo se aplica a las variables de tipo escalar (no es válida para las matrices u objetos). El tipo de la variable permanece sin cambios.

Ejemplo

```
<?php
$x = TRUE;
echo var_dump($x), ' => ', var_dump(floatval($x)), '<br />';
$x = 123;
echo var_dump($x), ' => ', var_dump(floatval($x)), '<br />';
$x = "1.23e45";
echo var_dump($x), ' => ', var_dump(floatval($x)), '<br />';
$x = "123abc";
echo var_dump($x), ' => ', var_dump(floatval($x)), '<br />';
$x = " \n\t\r 123.45abc";
echo var_dump($x), ' => ', var_dump(floatval($x)), '<br />';
?>*
```

Resultado

```
bool(true) => float(1)
int(123) => float(123)
string(7) "1.23e45" => float(1.23E+45)
string(6) "123abc" => float(123)
string(14) " \n\t\r 123.45abc" => float(123.45)
```

Se han respetado las reglas de conversión evocadas.

intval

La función `intval` devuelve el valor de una variable después de la conversión en entero.

Sintaxis

número `intval(mixto variable)`

Variable Variable que se va a procesar.

Esta función sólo se aplica a las variables de tipo escalar (no es válida para las matrices u objetos).

El tipo de la variable permanece sin cambios.

Ejemplo

```
<?php
$x = TRUE;
echo var_dump($x), ' => ', var_dump(intval($x)), '<br />';
$x = 123.9;
echo var_dump($x), ' => ', var_dump(intval($x)), '<br />';
$x = "1.23e45";
echo var_dump($x), ' => ', var_dump(intval($x)), '<br />';
```



```
$x = "123abc";  
echo var_dump($x), ' => ', var_dump(intval($x)), '<br />';  
$x = " \n\t\r 123.45abc";  
echo var_dump($x), ' => ', var_dump(intval($x)), '<br />';  
?>
```

Resultado

```
bool(true) => int(1)  
float(123.9) => int(123)  
string(7) "1.23e45" => int(1)  
string(6) "123abc" => int(123)  
string(14) " 123.45abc" => int(123)
```

Se han respetado las reglas de conversión evocadas anteriormente. Una vez más, debemos recordar que un número de punto flotante convertido a un entero se trunca y no se redondea: en nuestro ejemplo, 123.9 da 123 y no 124. Para convertir un número de punto flotante en entero, con redondeo, es necesario utilizar la función `round()`.

Ejemplo

```
<?php  
$x = 123.9;  
echo "round($x) => ", var_dump(round($x)), '<br />';  
echo "intval(round($x)) => ",  
    var_d
```

Matrices

1. Definición

En PHP, una matriz es una colección (lista de elementos) ordenada por la pareja clave/valor.

La clave puede ser de tipo número entero o de tipo cadena. En el primer caso, se dice que la matriz es numérica y la clave se designa por el término Índice. En el segundo caso, se dice que la matriz es asociativa: las claves no son necesariamente consecutivas, ni ordenadas y esta matriz puede tener claves enteras y claves de tipo cadena.

El valor asociado a la clave puede ser de cualquier tipo, incluyendo el tipo matriz, en cuyo caso se dice que la matriz es multidimensional.

Ejemplo

- Matriz numérica (índices ordenados consecutivos)

Clave/Índice	Valor
0	cero
1	uno
2	dos
3	tres

Matriz numérica (índices no ordenados, no consecutivos)

Clave/Índice	Valor
20	veinte
30	treinta
10	diez

Matriz mixta

Clave/Índice	Valor
0	cero
cero	0
uno	1
1	uno
dos	2
2	dos
tres	3
3	tres

Matriz multidimensional (lista de ciudades por país)

Clave/Índice	Valor
--------------	-------

ESPAÑA	Clave/Índice	Valor
	0	Madrid
	1	León
	2	Barcelona
ITALIA	Clave/Índice	Valor
	0	Roma
	1	Venecia

2. Creación

Una variable de tipo matriz se puede definir explícitamente a través de la función `array` o implícitamente mediante una notación entre corchetes (`[]`).

Desde la versión 5.4, es posible utilizar una sintaxis abreviada para definir explícitamente una matriz sin utilizar la función `array`.

Notación entre corchetes (`[]`)

Una variable utilizada por primera vez con una notación de la forma `$variable[...]`, se crea automáticamente con el tipo matriz.

Si se efectúa la misma operación en una variable ya definida, con un tipo escalar, produce un mensaje de error.

El contenido de una matriz puede estar bien definido por varias asignaciones de tipo `$matriz[...]` = valor.

Con una asignación del tipo `$matriz[] = valor`, PHP busca el índice entero mayor utilizado y asocia el valor al índice inmediatamente superior. Si la tabla está vacía, el elemento se colocará en el índice 0.

Con una asignación del tipo `$matriz[clave] = valor`, PHP asocia el valor a la clave especificada (que puede ser de tipo entero o de tipo cadena).

Ambas notaciones se pueden mezclar en una secuencia de asignación.

Ejemplo

```
<?php
$números[] = 'cero'; // => índice 0
$números[] = 'uno'; // => índice max (0) + 1 = 1
$números[] = 'dos'; // => índice max (1) + 1 = 2
$números[] = 'tres'; // => índice max (2) + 1 = 3
$números[5] = 'cinco'; // => índice 5
$números[] = 'seis'; // => índice max (5) + 1 = 6
$números['uno'] = 1; // índice "un"
$números[] = 'siete'; // => índice max (6) + 1 = 7
$números[-1] = 'menos uno'; // => -1
?>
```

Resultado

Clave/Índice	Valor
0	cero

1	uno
2	dos
3	tres
5	cinco
6	seis
uno	1
7	siete
-1	menos uno

Estas notaciones se pueden utilizar para construir una matriz multidimensional en forma de `$matriz[...] = $matriz_interior` o `$matriz[...] [...] = valor`. La primera notación permite almacenar una matriz en una ubicación de otra matriz, y la segunda notación, almacenar un valor directamente en una ubicación dentro de otra matriz.

Ejemplo

- Primer método:

```
<?php
// creación de una tabla que contiene las ciudades de España
$ciudades_españa[] = 'Madrid';
$ciudades_españa[] = 'León';
$ciudades_españa[] = 'Barcelona';
// almacenamiento de la tabla de ciudades de España en la tabla
// de ciudades
$ciudades['ESPAÑA'] = $ciudades_españa;
// idem con las ciudades de Italia
$ciudades_italia[] = 'Roma';
$ciudades_italia[] = 'Venecia';
$ciudades['ITALIA'] = $ciudades_italia;
?>
```

- Segundo método:

```
<?php
// almacenamiento directo de las ciudades en la tabla
// - para España
$ciudades['ESPAÑA'][] = 'Madrid';
$ciudades['ESPAÑA'][] = 'León';
$ciudades['ESPAÑA'][] = 'Barcelona';
// - para Italia
$ciudades['ITALIA'][] = 'Roma';
$ciudades['ITALIA'][] = 'Venecia';
?>
```

Resultado (en ambos casos)

Clave/Índice	Valor	
ESPAÑA	Clave/Índice	Valor
	0	Madrid
	1	León
	2	

ITALIA	Clave/Índice	Valor
	0	Roma
	1	Venecia

La función array

La función `array` permite crear una matriz a partir de una lista de elementos.

Sintaxis

```
matriz array([mixto valor[, ...]])
```

o

```
matriz array([{cadena | entero} clave => mixto valor[, ...]])
```

`valor` Elemento de la matriz.

`clave` Valor de la clave.

En la primera sintaxis, las claves/índices no se han especificado y se crea una matriz numérica con índices consecutivos empezando en 0: el primer argumento de la función se almacena en el índice 0, el segundo en el índice 1, etc.

En la segunda sintaxis, el índice o la clave se especifican ya sea mediante un número entero o una cadena, y se le asocia un valor mediante el operador `=>`.

Ambas sintaxis se pueden mezclar. En este caso, cuando no se ha especificado el índice o la clave, PHP busca el índice entero mayor utilizado y asocia el valor al índice inmediatamente superior; si no hay índices enteros, el elemento se coloca en el índice 0.

La función `array`, llamada sin argumentos, crea una matriz vacía.

Ejemplo

```
<?php
$numeros = array('cero','uno','dos','tres',
    5 => 'cinco','seis','uno' => 1,'siete',-1 => 'menos uno');
?>
```

Resultado

Clave/Índice	Valor
0	cero
1	uno
2	dos
3	tres
5	cinco
6	seis
uno	1

7	siete
-1	menos uno

La función `array` acepta como argumentos los datos de tipo matriz (ya sea una variable o una llamada anidada en `array`), lo que permite crear una matriz multidimensional.

Ejemplo

- Primer método:

```
<?php
// creación de una matriz que contiene las ciudades de España
$ciudades_españa = array('Madrid', 'León', 'Barcelona');
// idem con las ciudades de Italia
$ciudades_italia = array('Roma', 'Venecia');
// almacenamiento de las 2 matrices en la tabla de ciudades
$ciudades = array('ESPAÑA' => $ciudades_españa,
                  'ITALIA' => $ciudades_italia);
?>
```

- Segundo método:

```
<?php
// creación por anidamiento de llamadas a array
$ciudades = array('ESPAÑA' => array('Madrid', 'León', 'Barcelona'),
                  'ITALIA' => array('Roma', 'Venecia'));
?>
```

Resultado (en ambos casos)

Clave/Índice	Valor	
ESPAÑA	Clave/Índice	Valor
	0	Madrid
	1	León
	2	Barcelona
ITALIA	Clave/Índice	Valor
	0	Roma
	1	Venecia

Sintaxis corto

Desde la versión 5.4, es posible definir una matriz explícitamente utilizando una notación entre corchetes (`[]`) en lugar de la función `array()`.

Ejemplo

```
<?php
$numeros = ['cero', 'uno', 'dos', 'tres',
            5 => 'cinco', 'seis', 'uno' => 1, 'siete', -1 => 'menos uno'];
?>
```

Al igual que al utilizar la función `array()`, otra matriz puede definirse como elemento, ya sea con la función `array()`, o bien con la sintaxis corta como en el siguiente ejemplo.

Ejemplo

```
<?php
$ciudades = ['FRANCIA' => ['París', 'Lyon', 'Nantes'],
             'ITALIA' => ['Roma', 'Venecia']];
?>
```

Esta sintaxis corta permite aligerar la escritura en un cierto número de situaciones (pasar de una matriz en parámetro a una función, definición de matrices multidimensionales)..

3. Manipulación

En el manejo de matrices existen dos necesidades comunes:

- Acceder a un elemento individual de la matriz.
- Examinar la matriz.

Acceder a un elemento individual de la matriz

La notación entre paréntesis se utiliza para acceder, leer o escribir, un elemento individual de la matriz:

```
$matriz[{cadena | entero} clave]
```

\$matriz Matriz correspondiente.

clave Valor de la clave/índice.

Para las matrices multidimensionales, se deben utilizar varias series de corchetes.

Ejemplo

```
<?php
$números = array('cero', 'uno', 'dos', 'tres',
                 5 => 'cinco', 'seis', 'uno' => 1, 'siete', -1 => 'menos uno');
echo $números[1], '<br />';
echo $números['un'], '<br />';
$ciudades = array('ESPAÑA' => array('Madrid', 'León', 'Barcelona'),
                  'ITALIA' => array('Roma', 'Venecia'));
echo $ciudades['ESPAÑA'][0], '<br />';
echo $ciudades['ITALIA'][1], '<br />';
?>
```

Resultado

```
uno
1
Madrid
Venecia
```

PHP acepta que omita el delimitador de cadena (comillas o apóstrofo) cuando especifica una clave de tipo cadena en una matriz asociativa.

Ejemplo

```
<?php
$números = array('uno' => 1, 'dos' => 2);
```

```
// utilización de $números[uno] y no $números['uno']
echo $números[uno];
?>
```

Resultado (si los errores de tipo E_NOTICE no se muestran)

1

Resultado (si los errores de tipo E_NOTICE se muestran)

Notice: Use of undefined constant uno - assumed 'uno' in /app/scripts/index.php
on line 4
1

Esta sintaxis se acepta (y funciona), pero genera un error de tipo E_NOTICE. PHP considera que la constante uno no existe y la sustituye por la cadena 'uno' ..., lo que permite obtener el resultado correcto... hasta el día en que defina una constante uno con valor 1, por ejemplo. En consecuencia, no se recomienda utilizar esta sintaxis "simplificada".

El principio de sustitución de variables en las cadenas delimitadas por comillas funciona con las matrices. Es necesario utilizar llaves para delimitar la expresión en dos casos:

- Para especificar una clave de tipo cadena expresada en forma de un literal: `{ $matriz['...'] }`
- Para una matriz multidimensional: `{ $matriz[...] [...] }`

Ejemplo

```
<?php
$números = array('cero','uno','dos','tres',
    5 => 'cinco','seis','uno' => 1,'siete',-1 => 'menos uno');
echo "\$números[1] = $números[1]<br />";
echo "\$números['un'] = { $números['uno'] }<br />";
$ciudades = array('ESPAÑA' => array('Madrid','León','Barcelona'),
    'ITALIA' => array('Roma','Venecia'));
echo "\$ciudades['ESPAÑA'][0] = { $ciudades['ESPAÑA'][0] }<br />";?>
```

Resultado

```
$números[1] = uno
$números['un'] = 1
$ciudades['ESPAÑA'][0] = Madrid
```

Examinar la matriz

Se pueden utilizar multitud de métodos para examinar una matriz con las siguientes construcciones:

- la estructura de control iterativa `for`
- la estructura de control iterativa `while`
- la estructura de examen de matriz `foreach`

En este capítulo, sólo estudiaremos el uso de la estructura `foreach`, que es sin duda la forma más fácil de examinar una matriz. Este método no requiere ningún conocimiento especial sobre la naturaleza de la matriz (numérica, asociativa, rango de índices/claves...).

Sintaxis


```
foreach(matriz as variable_valor)
{ instrucciones }
```

o

```
foreach(matriz as variable_clave => variable_valor)
{ instrucciones }
```

La primera sintaxis permite examinar la matriz de principio a fin; en cada iteración, el valor actual de la matriz se almacena en la variable `variable_valor` y las instrucciones entre llaves se ejecutan. Esta sintaxis es suficiente si el procesamiento no necesita hacer referencia a los valores de la clave.

La segunda sintaxis funciona en base al mismo principio, pero en cada iteración, la clave actual se almacena en la variable `variable_clave` y el valor en la variable `variable_valor`. Esta sintaxis es útil si el procesamiento necesita hacer referencia a los valores de la clave.

Ejemplo

```
<?php
// Inicialización de una matriz.
$ números = array('cero','uno','dos',
                  'cero' => 0,'uno' => 1,'dos' => 2);
// Examen de la matriz con la primera sintaxis.
echo 'Primera sintaxis:<br />';
foreach($ números as $ número) {
    echo "$ número<br />";
}
// Examen de la matriz con la segunda sintaxis
echo 'Segunda sintaxis:<br />';
foreach($ números as $ clave => $ número) {
    echo "$ clave => $ número<br />";
}
?>
```

Resultado

```
Primera sintaxis:
cero
uno
dos
0
1
2
```

```
Segunda sintaxis:
0 => cero
1 => uno
2 => dos
cero => 0
uno => 1
dos => 2
```

Estos dos ejemplos demuestran que no es necesario ningún conocimiento previo de la matriz para examinarla: ni su tamaño, ni su estructura de claves.

Desde la versión 5.5, la estructura `foreach` permite examinar una matriz de matrices y recuperar los elementos de la matriz anidada en variables con la ayuda de la función `list`.

Sintaxis

```
foreach(matriz as list(variable[,... ]))
```

En cada iteración, los elementos de la matriz anidada actual se almacenan en las variables enumeradas en la función `list` (primer elemento en la primera variable, segundo elemento en la segunda variable, etc.).

Ejemplo

```
<?php
$capitales = [['FRANCIA','París'], ['ITALIA','Roma']];
foreach ($capitales as list($país,$ciudad)) {
    echo "$país: $ciudad<br />";
}
?>
```

Resultado

```
FRANCIA: París
ITALIA: Roma
```

En la función `list`, si hay menos variables que elementos en la tabla anidada, los elementos sobrantes de esta última se ignorarán. Por el contrario, si hay muchas variables en la función `list`, se generará una alerta de nivel `NOTICE` y las variables sobrantes no se inicializarán.

4. Alcance

Las variables de tipo matriz siguen las mismas reglas de alcance y de duración que las variables de tipo escalar (véase sección Variables - Alcance y duración).

5. Funciones útiles

PHP ofrece un gran número de funciones que permiten manipular las matrices.

Las funciones utilizadas con mayor frecuencia son:

Nombre	Función
<code>count</code>	Cuenta el número de elementos de una matriz.
<code>in_array</code>	Comprueba si un valor está presente en una matriz.
<code>array_search</code>	Busca un valor en una matriz.
<code>array_replace</code>	Reemplaza valores de una matriz.
<code>[a k][r]sort</code>	Ordena una matriz (varias variantes posibles).
<code>explode</code>	Divide una cadena según un separador y almacena los elementos en una matriz.
<code>implode</code>	Reagrupa los elementos de una matriz en una cadena mediante un separador.
<code>str_split</code>	Divide una cadena en fragmentos de longitud fija y almacena los elementos en una matriz.
<code>Array_column</code>	Devuelve los valores de una columna de una matriz

	multidimensional.
--	-------------------

La función `is_array` (véase la sección Tipos de datos - Funciones útiles) permite conocer si una variable es de tipo matriz. Recuerde.

Existen muchas otras funciones y puede consultar la descripción de cada función en línea en www.php.net. Ahí encontrará, especialmente, funciones para:

- realizar cálculos (suma...)
- extraer una submatriz de una matriz
- fusionar matrices
- deduplicar una matriz...

count

La función `count` permite conocer el número de elementos en una variable en general, una matriz en particular.

Sintaxis

```
entero count (mixto variable)
```

variable Variable en cuestión.

Algunas funciones útiles sobre las cadenas de caracteres, las fechas y los números

1. Preámbulo


El objetivo de esta sección es presentar las funciones más útiles, relacionadas con la manipulación de cadenas de caracteres y de fechas, tipos de datos que se utilizan con mucha frecuencia en las aplicaciones.

PHP ofrece muchas funciones y puede consultar la descripción de cada función en línea en www.php.net.

2. Manipulación de cadenas de caracteres

Las funciones más útiles para la manipulación de cadenas de caracteres son las siguientes:

Nombre	Función
<code>strlen</code>	Devuelve el número de caracteres de una cadena.
<code>strtolower</code> <code>strtoupper</code> <code>ucfirst</code> <code>ucwords</code> <code>lcfirst</code>	Las conversiones de minúsculas/mayúsculas pueden limitarse a la(s) primera(s) palabra(s).
<code>strcmp</code> <code>strcasecmp</code>	Comparación de cadenas (sensible a mayúsculas y minúsculas o no).
<code>[s]printf</code> <code>v[s]printf</code>	Formato a una cadena (idéntica a las funciones de C equivalentes).
<code>number_format</code>	Formato de un número.
<code>[l r]trim</code>	Eliminación de caracteres "blancos".
<code>substr</code>	Extracción de una subcadena de una cadena.
<code>str_repeat</code>	Construcción de una cadena por repetición de caracteres.
<code>str[r][i]pos</code>	Búsqueda de la posición de una ocurrencia (carácter o cadena) dentro de una cadena.
<code>str[i]str</code> <code>strrchr</code>	Extracción de la subcadena dentro de una cadena a partir de una ocurrencia determinada de un carácter o una cadena.
<code>str_[i]replace</code>	Sustitución de las ocurrencias de una cadena por otra cadena.
<code>strtr</code>	Sustitución de las ocurrencias de un carácter por otro carácter o de una cadena por otra cadena.

 Recuerde las funciones `explode`, `implode` y `str_split` anteriormente presentadas (véanse sección Matrices - Funciones útiles). Otras funciones, más específicamente relacionadas con la gestión de formularios se estudian en el capítulo Gestionar los formularios.

strlen

La función `strlen` devuelve el número de caracteres de una cadena.

Sintaxis

```
entero strlen(cadena cadena)
```

cadena Cadena en cuestión.

strtolower - strtoupper - ucfirst - ucwords - lcfirst

Estas funciones permiten realizar conversiones de minúsculas/mayúsculas, posiblemente limitadas a la(s) primera(s) palabra(s) de la cadena.

Sintaxis

```
cadena strtolower(cadena cadena)
cadena strtoupper(cadena cadena)
cadena ucfirst(cadena cadena)
cadena ucwords(cadena cadena)
cadena lcfirst(cadena cadena)
```

cadena Cadena que se va a procesar.

La función `strtolower` convierte todos los caracteres de una cadena en minúsculas.

La función `strtoupper` convierte todos los caracteres de una cadena en mayúsculas.

La función `ucfirst` convierte el primer carácter de una cadena en mayúsculas.

La función `ucwords` convierte el primer carácter de cada palabra de una cadena en mayúsculas.

La función `lcfirst` convierte el primer carácter de una cadena en minúscula (a partir de la versión 5.3).

Ejemplo

```
<?php
$x = 'OLIVIER HEURTEL';
$y = 'olivier heurtel';
echo "strtolower('$x') = ",strtolower($x),'<br />';
echo "strtoupper('$y') = ",strtoupper($y),'<br />';
echo "ucfirst('$y') = ",ucfirst($y),'<br />';
echo "lcfirst('$x') = ",lcfirst($x),'<br />';
echo "ucwords('$y') = ",ucwords($y),'<br />';
?>
```

Resultado

```
strtolower('OLIVIER HEURTEL') = olivier heurtel
strtoupper('olivier heurtel') = OLIVIER HEURTEL
ucfirst('olivier heurtel') = Olivier heurtel
lcfirst('OLIVIER HEURTEL') = oLIVIER HEURTEL
ucwords('olivier heurtel') = Olivier Heurtel
```

strcmp - strcasecmp

Estas funciones permiten comparar dos cadenas teniendo en cuenta o no las mayúsculas y minúsculas.

Sintaxis

```
entero strcmp(cadena cadena1, cadena cadena2)
entero strcasecmp(cadena cadena1, cadena cadena2)
```

cadena1 y cadena2 Cadenas que se van a comparar.

Ambas funciones devuelven un número negativo si `cadena1` es menor que `cadena2`, un número igual a 0 si son iguales y un número positivo si `cadena1` es mayor que `cadena2`.

`strcmp` es sensible a mayúsculas y minúsculas, mientras que `strcasecmp` no lo es.

Ejemplo

```
<?php
$x = 'Olivier';
$y = 'olivier';
echo "strcmp('$x','$y') = ", strcmp($x,$y), '<br />';
echo "strcasecmp('$x','$y') = ", strcasecmp($x,$y), '<br />';
?>
```

Resultado

```
strcmp('Olivier','olivier') = -1
strcasecmp('Olivier','olivier') = 0
```

[s]printf

Las funciones `printf` y `sprintf` permiten dar formato a una cadena (idénticas a las funciones de C equivalentes).

Sintaxis

```
cadena sprintf(cadena formato[, mixto valor[, ...]])
entero printf(cadena formato[, mixto valor[, ...]])
```

formato Cadena de formato que presenta varias directivas según las especificaciones que figuran a continuación.

valor Valor que se integrará en la cadena.

`sprintf` devuelve el resultado en forma (o FALSE en caso de error), mientras que `printf` muestra directamente el resultado (como la instrucción `echo`) y devuelve la longitud de la cadena con formato en caso de éxito o FALSE en caso de error.

La cadena `format` debe contener una directiva de formato para cada argumento `valor`; esta directiva de formato especifica la ubicación y el formato del valor correspondiente. La correspondencia entre una directiva de formato y un valor es de posicionamiento (primera directiva para el primer valor...).

Las instrucciones de formato comienzan con el carácter % seguido de una a cinco informaciones, siendo la última la única obligatoria:

```
%[relleno][alineación][longitud][precisión]tipo
```

Las informaciones son las siguientes:

relleno	Especifica el carácter utilizado para el relleno. El carácter por defecto es el espacio. Se puede utilizar cualquier otro carácter si se menciona precedido de un apóstrofo (únicamente el carácter cero se puede indicar directamente): 'x indica que el carácter de relleno es la "x".
alineación	Especifica la alineación. De forma predeterminada, la alineación es a la derecha. El carácter menos ("-") permite obtener una alineación a la izquierda.
longitud	Especifica el número mínimo de caracteres del elemento formateado.
precisión	Indica el número de dígitos usados para el formato de un número de punto flotante (válido sólo si el elemento asociado es un número).
tipo	Da el tipo del valor que se va a insertar: c: entero que se sustituye por el carácter cuyo código ASCII tiene este valor; d: entero que se representará como tal; f: número de punto flotante que se representará como tal (teniendo en cuenta la configuración regional); F: número de punto flotante que se representará como tal (sin tener en cuenta la configuración regional); apareció con la versión 5.0.3; s: ninguno, se representará como una cadena.

Para obtener un carácter "%" en el resultado final, debe duplicarlo en el formato.

Algunos ejemplos:

Directiva	Valor	Resultado	Explicación
%d	1	1	Número entero sin un formato especial.
%02d	1	01	02 = completar con el carácter cero, en una longitud mínima de dos.
%f	1/3	0.333333	Número de punto flotante sin un formato especial.
%.2f	1/3	0.33	.2 = dos dígitos después del separador decimal.
%02.3f	1/3	00.333	.3 = tres dígitos después del separador decimal. 02 = completar con el carácter cero, con el separador decimal, en una longitud mínima de dos.
%s	Olivier!	Olivier!	Cadena sin un formato especial.
%'.10s	Olivier	...Olivier	' .10 = completar con un punto para llegar a una longitud mínima de diez caracteres (alineación predeterminada).
%'.-10s	Olivier	Olivier...	' .-10 = completar con un punto para llegar a una longitud mínima de diez

			caracteres (signo - = alineación a la izquierda).
%'.5.2f	9.99.90	.2 = dos dígitos después del separador decimal. ' .5 = completar con un punto para llegar a una longitud mínima de cinco caracteres antes del punto decimal (alineación predeterminada).

Ejemplo

```
<?php
echo 'Formato de una fecha: ',
    sprintf('%02d/%02d/%04d',1,1,2001),'<br />';
echo 'Formato de números: ',
    sprintf('%01.2f - %01.2f',1/3,12345678.9),'<br />';
echo 'Porcentaje: ',
    sprintf('%01.2f %%',12.3),'<br />';
echo 'Utilización de las opciones de relleno:<br />';
echo '<tt>'; // fuente no proporcional
printf("%'.-10s%'.5.2f<br />",'Libros',9.35); // printf direct
printf("%'.-10s%'.5.2f<br />",'Discos',99.9); // printf direct
echo '</tt>';
?>
```

Resultado

```
Formato de una fecha: 01/01/2001
Formato de números: 0.33 - 12345678.90
Porcentaje: 12.30 %
Utilización de las opciones de relleno:
Libros.....9.35
Discos...99.90
```

v[s]printf

Las funciones `vprintf` y `vsprintf` son idénticas a las funciones `printf` y `sprintf`, pero aceptan como segundo parámetro una matriz que contiene los distintos valores que se van a utilizar (en lugar de varios parámetros).

Sintaxis

```
cadena vsprintf(cadena formato[, matriz valores])
entero vprintf(cadena formato[, matriz valores])
```

Con

formato Cadena de formato que presenta varias directivas según las especificaciones dadas anteriormente.

valores Matriz que da los valores que se van a integrar en la cadena.

Ejemplo

```
<?php
$datos = array(array('Libros',9.35),array('Discos',99.9));
echo '<tt>'; // fuente no proporcional
```



```
foreach($datos as $línea) {
vprintf("%'-.10s%' .5.2f<br />", $línea); // printf direct
}
echo '</tt>';
?>
```

Resultado

```
Libros.....9.35
Discos.....99.90
```

number_format

La función `number_format` permite dar formato a un número.

Sintaxis

```
cadena number_format(número valor[, entero decimales[,
cadena separador_decimal, cadena separador_millares]])
```

valor	Número que se va a formatear.
decimales	Número de decimales (ninguna parte decimal por defecto).
separador_decimal	Separador decimal (punto por defecto).
separador_millares	Separador de millares (coma por defecto).

Se puede llamar a la función con uno, dos o cuatro argumentos, pero no tres: si se da un tercero, el cuarto es obligatorio.

Si el número tiene una precisión superior a la solicitada (parámetro decimales), el número se redondea a la precisión solicitada.

Ejemplo

```
<?php
$x = 1234.567;
echo "number_format($x) = ", number_format($x), '<br />';
echo "number_format($x,1) = ", number_format($x,1), '<br />';
echo "number_format($x,2,' ',' ') = ",
    number_format($x,2,' ',' '), '<br />';
?>
```

Resultado

```
number_format(1234.567) = 1,235
number_format(1234.567,1) = 1,234.6
number_format(1234.567,2,' ',' ') = 1 234,57
```

Observe, en estos ejemplos, los redondeos automáticos cuando la precisión solicitada es inferior a la precisión del número.

ltrim - rtrim - trim

Estas funciones permiten eliminar los caracteres "blancos" u otros caracteres al principio de la cadena, al final de la cadena o en ambos lados.

Sintaxis

```
cadena ltrim(cadena cadena[, cadena caracteres])
cadena rtrim(cadena cadena[, cadena caracteres])
cadena trim(cadena cadena[, cadena caracteres])
```

Con

cadena	Cadena que se va a procesar.
caracteres	Cadena que indica la lista de caracteres que se va a eliminar. Si este parámetro está ausente, los caracteres "blancos" se eliminan.

Las tres funciones devuelven una cadena igual a la cadena inicial en la que los caracteres "blancos" o los caracteres especificados se han eliminado al principio (`ltrim` con `l = left` = a la izquierda) al final (`rtrim` con `r = right` = a la derecha) o en ambos lados (`trim`).

Los caracteres "blancos" son el salto de línea (`\n` = código ASCII 10), el retorno de carro (`\r` = código ASCII 13), la tabulación (`\t` = código ASCII 9), el carácter NULL (`\0` = código ASCII 0) y el espacio.

Ejemplo

```
<?php
$x = "\t\t\t x \n\r";
echo 'strlen($x) = ',strlen($x),'<br />';
echo 'strlen(ltrim($x)) = ',strlen(ltrim($x)),'<br />';
echo 'strlen(rtrim($x)) = ',strlen(rtrim($x)),'<br />';
echo 'strlen(trim($x)) = ',strlen(trim($x)),'<br />';
$x = '***+-Olivier-***';
echo "trim('$x','*+-') = ",trim($x,'*+-'), '<br />';
?>
```

Resultado

```
strlen($x) = 8
strlen(ltrim($x)) = 4
strlen(rtrim($x)) = 5
strlen(trim($x)) = 1
trim('***+-Olivier-***','*+-') = Olivier
```

substr

La función `substr` permite extraer una subcadena de una cadena.

Sintaxis

```
cadena substr(cadena cadena, entero inicio[, entero longitud])
```

cadena	Cadena que se va a procesar.
inicio	Posición del primer carácter de la subcadena que se va a extraer (atención 0 = 1er carácter).
longitud	Número de caracteres que se van a extraer (por defecto, hasta el final de la cadena).

Si el argumento `inicio` es positivo, la subcadena extraída comienza en el

carácter inicio(0 = 1er carácter).

- Si el argumento `inicio` es negativo, la subcadena extraída comienza en el carácter `inicio` partiendo desde el final (-1 = último carácter).
- Si no se especifica el argumento `longitud`, la subcadena extraída termina al final de la cadena.
- Si el argumento `longitud` se especifica y es positivo, `substr` extrae el número de caracteres indicado por el argumento `longitud`.
- Si el argumento `longitud` se especifica y es negativo, la subcadena extraída termina al final de la cadena, menos el número de caracteres indicado por el valor absoluto del argumento `longitud`.

Ejemplo

```
<?php
// 0123456 => para el control
$x = 'Olivier';
echo "substr('$x',3) = ",substr($x,3),'<br />';
echo "substr('$x',3,2) = ",substr($x,3,2),'<br />';
echo "substr('$x',-4) = ",substr($x,-4),'<br />';
echo "substr('$x',-4,3) = ",substr($x,-4,3),'<br />';
?>
```

Resultado

```
substr('Olivier',3) = vier
substr('Olivier',3,2) = vi
substr('Olivier',-4) = vier
substr('Olivier',-4,3) = vie
```

str_repeat

La función `str_repeat` permite construir una cadena por repetición de caracteres.

Sintaxis

`cadena str_repeat(cadena secuencia, entero repeticiones)`

secuencia	Secuencia de caracteres a repetir.
repeticiones	Número de repeticiones deseadas.

Ejemplo

```
<?php
echo str_repeat('abc',3);
?>
```

Resultado

```
abcbabcbcb
```

strpos - strrpos - stripos - strrpos

Estas funciones permiten buscar la posición de una ocurrencia (carácter o cadena) dentro de una

cadena.

Sintaxis

```
entero strpos(cadena a_procesar, cadena buscar[, entero inicio])
entero strrpos(cadena a_procesar, cadena buscar[, entero inicio])
entero strpos(cadena a_procesar, cadena buscar[, entero inicio])
entero strrpos(cadena a_procesar, cadena buscar[, entero inicio])
```

Con

a_procesar	Cadena que se va a procesar.
buscar	Elemento buscado.
inicio	Número del carácter (0 = primer carácter) a partir del cual se debe llevar a cabo la búsqueda (por defecto, el inicio de la cadena).

strpos busca, en la cadena a_procesar, la primera ocurrencia de la cadena de buscar, comenzando a apartir del carácter número inicio (0 = primer carácter).

strrpos busca, en la cadena a_procesar, la última ocurrencia de la cadena de buscar, comenzando a apartir del carácter número inicio (0 = primer carácter). Si el inicio es negativo (-n), los n últimos caracteres de la cadena a_procesar se ignoran. El parámetro inicio se añadió en la versión 5. En la versión 4, sólo se tenía en cuenta el primer carácter de la cadena buscar.

Las dos funciones son sensibles a mayúsculas y minúsculas (en mayúsculas no es igual que en minúsculas). Las funciones strpos y strrpos son idénticas respectivamente a las funciones strpos y strrpos, pero no son sensibles a mayúsculas y minúsculas; esto se introdujo en la versión 5.

Estas cuatro funciones devuelven la posición de la ocurrencia encontrada (0 = primer carácter) o FALSE si el elemento de búsqueda no se encuentra.

FALSE es equivalente a 0, por lo tanto es fácil confundir el caso en el que el elemento no se ha encontrado y en el que se encontró al principio de la cadena. La técnica consiste en utilizar el operador de comparación "===" (tres signos igual), que permite comparar el valor y el tipo de dos expresiones (para más detalles, véase el capítulo Operadores).

Ejemplo

```
<?php
//      0123456789 ... => para el control
$correo = 'contacto@olivier-heurtel.fr';
// strpos
$posición = strrpos($mail, '@');
echo "@ está en la posición $posición en $correo<br />";
// strpos
$posición = strpos($correo, 'olivier');
echo "'olivier' está en la posición $posición en $correo<br />";
// Ocurrencia al principio de la cadena
$posición = strpos($correo, 'contacto');
if (! $posición) { // prueba no superada
    echo "'contacto' no se puede encontrar en $correo<br />";
} else {
    echo "'contacto' está en la posición $posición
    en $correo<br />";
}
if ($posición === FALSE) { // prueba superada: ===
    echo "'contacto' no se puede encontrar en $correo<br />";
}
```

```

} else {
    echo "'contacto' está en la posición $posición
    en $correo<br />";
}
// Ocurrencia no encontrada
$posición = strpos($correo, 'información');
if ($posición === FALSE) { // prueba superada: ===
    echo "'información' no se encuentra en $correo<br />";
} else {
    echo "'información' está en la posición $posición
    en $correo<br />";
}
?>

```

Resultado

```

@ está en la posición 7 en contacto@olivier-heurtel.fr
'olivier' está en la posición 8 en contacto@olivier-heurtel.fr
'contacto' no se puede encontrar en contacto@olivier-heurtel.fr
'contacto' está en la posición 0 en contacto@olivier-heurtel.fr
'información' no se puede encontrar en contacto@olivier-heurtel.fr

```

strstr - stristr - strrchr

Estas funciones permiten extraer la subcadena comenzando a partir de una ocurrencia determinada de un carácter o de una cadena.

Sintaxis

```

cadena strstr(cadena a_procesar, cadena buscar, booleano antes)
cadena stristr(cadena a_procesar, cadena buscar, booleano antes)
cadena strrchr(cadena a_procesar, carácter buscar)

```

<code>a_procesar</code>	Cadena que se va a procesar.
<code>buscar</code>	Elemento buscado.
<code>antes</code>	Indica si la función devuelve la cadena situada antes (valor TRUE) o después (valor FALSE, por defecto) de la cadena buscada.

Las funciones `strstr` y `stristr` buscan, en la cadena `a_procesar`, la primera ocurrencia de la cadena `buscar` y devuelven la porción final o inicial de la cadena comenzando desde esta ocurrencia (incluida). La función `strstr` es sensible a mayúsculas y minúsculas (una mayúscula es diferente de una minúscula), mientras que `stristr` no lo es.

`strrchr` busca, en la cadena `a_procesar`, la última ocurrencia del carácter `buscar` y devuelve la porción final o inicial de la cadena comenzando desde esta ocurrencia (incluida). Si `buscar` es una cadena de varios caracteres, sólo el primero se tiene en cuenta. `strrchr` es sensible a mayúsculas y minúsculas.

Estas tres funciones devuelven FALSE si no se encuentra el elemento buscado.

Ejemplo

```

<?php
$correo = 'Olivier-Heurtel@olivier-heurtel.fr';
echo "Resto del $correo comenzando por:<br />";
// strrchr
$resto = strrchr($correo, '-');

```

```
echo "- la última ocurrencia de '-'<br />----> $resto <br />";  
// strstr  
$resto = strstr($correo,'olivier');  
echo "- la primera ocurrencia de 'olivier'  
      (sensible a mayúsculas y minúsculas)<br />----> $resto <br />";  
// stristr  
$resto = stristr($correo,'olivier');  
echo "- la primera ocurrencia de 'olivier'  
      (no sensible a mayúsculas y minúsculas)<br />----> $resto <br />";  
echo "Inicio de $correo terminando por:<br />";  
// strstr&nbs
```

El operador de asignación por valor

El operador de asignación es el signo de igual (=).

Sintaxis

```
$variable = expresión;
```

expresión puede ser un valor literal de cualquier tipo (123, 'Hola', TRUE...), otra variable o cualquier expresión que combina valores literales, variables con funciones y operadores.

Ejemplo

```
<?php
$nombre = 'Olivier';
$indice = 1;
?>
```

Con esta sintaxis, la asignación se efectúa por valor, es decir, que el valor de la expresión situada a la derecha del signo igual se copia en la variable mencionada a la izquierda. Al realizar una asignación de una variable en otra, la modificación posterior de la primera variable no tiene efecto en la segunda.

Ejemplo

```
<?php
// Inicialización de una variable.
$x = 1;
// Asignación de la variable $x en la variable $y.
$y = $x;
// Modificación de la variable $x.
$x = 2;
// Visualización del resultado.
echo "\$x = $x<br />";
echo "\$y = $y<br />";
?>
```

Resultado

```
$x = 2
$y = 1
```

La operación de asignación es una expresión que tiene un valor igual al valor asignado y que se puede utilizar directamente en otra expresión. Por ejemplo, el valor de la expresión `$x = 1` es 1 y se puede escribir una instrucción de tipo `$y = ($x=1) + 2` que asigna al valor 3 a `$y`.

Ejemplo

```
<?php
// Asignación de una instrucción de $x y $y.
$y = ($x = 1) + 2;
// Visualización del resultado.
echo "\$x = $x<br />";
echo "\$y = $y<br />";
?>
```

Resultado

```
$x = 1
```

\$y = 3

Esta técnica es muy práctica, pero puede afectar a la legibilidad del código.



Para todos los operadores que se estudian en este capítulo, pueden existir espacios en todo el operador.

Índice

Información general sobre PHP

- ¿Qué es PHP?
- Estructura básica de una página PHP
- Configuración de PHP
- Utilizar PHP desde la línea de comandos

Variables, constantes, tipos y matrices

- Constantes
- Variables
- Tipos de datos
- Matrices
- Algunas funciones útiles sobre las cadenas de caracteres, las fechas y los números

Operadores

- El operador de asignación por valor
- El operador de asignación por referencia
- Los operadores aritméticos
- El operador de cadena
- Los operadores de comparación
- Los operadores lógicos
- El operador ternario
- Los operadores combinados
- Precedencia de los operadores

Estructuras de control

Funciones y clases

Gestión de formularios

Acceder a las bases de datos

Administrar las sesiones

Enviar un correo electrónico

Gestión de archivos

Administrar los errores en un script PHP

Anexo

Condiciones generales de uso

El operador de asignación por referencia

A partir de PHP 4, es posible realizar una asignación por referencia utilizando el operador &.

Sintaxis

```
$variable2 = &$variable1;
```

Con esta sintaxis, el valor de la variable `$variable1` no se copia en la variable `$variable2`. La variable `$variable2` hace referencia a la variable `$variable1`; las dos variables apuntan a la misma zona de memoria y la modificación de una variable afecta a la otra.

Ejemplo

```
<?php
// Inicialización de una variable.
$apellido = 'Olivier';
// Asignación en otra variable por referencia.
$patronimico = &$apellido;
// Visualización del resultado.
echo "<b>Inicialmente:</b><br />";
echo "\$apellido = $apellido<br />";
echo "\$patronimico = $patronimico<br />";
// Modificación de la primera variable.
$apellido = 'Heurtel';
// Visualización del resultado.
echo "<b>Después de la modificación de \$apellido:</b><br />";
echo "\$apellido = $apellido<br />";
echo "\$patronimico = $patronimico<br />";
?>
```

Resultado

```
Inicialmente:
$apellido = Olivier
$patronimico = Olivier
Después de la modificación de $apellido:
$apellido = Heurtel
$patronimico = Heurtel
```

[Subir](#)

Índice

Información general sobre PHP

[¿Qué es PHP?](#)

[Estructura básica de una página PHP](#)

[Configuración de PHP](#)

[Utilizar PHP desde la línea de comandos](#)

Variables, constantes, tipos y matrices

[Constantes](#)

[Variables](#)

[Tipos de datos](#)

[Matrices](#)

[Algunas funciones útiles sobre las cadenas de caracteres, las fechas y los números](#)

Operadores

[El operador de asignación por valor](#)

[El operador de asignación por referencia](#)

[Los operadores aritméticos](#)

[El operador de cadena](#)

[Los operadores de comparación](#)

[Los operadores lógicos](#)

[El operador ternario](#)

[Los operadores combinados](#)

[Precedencia de los operadores](#)

Estructuras de control

Funciones y clases

Gestión de formularios

Acceder a las bases de datos

Administrar las sesiones

Enviar un correo electrónico

Gestión de archivos

Administrar los errores en un script PHP

Anexo

Los operadores aritméticos

Los operadores aritméticos son los siguientes:

Operación	Operador	Ejemplo (\$x=13 y \$y=8)
Suma	+	echo \$x + \$y; => 21
Resta	-	echo \$x - \$y; => 5
Multiplicación	*	echo \$x * \$y; => 104
División	/	echo \$x / \$y; => 1.625
Módulo (resto de la división entera del primer operando por el segundo)	%	echo \$x % \$y; => 5
Contrario	-	echo -\$x; => -13
Preincremento (incrementa la variable <u>antes</u> de devolver el valor de la variable)	++ antes del operando	echo ++\$x; => 14
Postincremento (incrementa la variable <u>después</u> de haber devuelto el valor de la variable)	++ después del operando	echo \$x++; => 13 echo \$x; => 14
Predecremento (decrementa la variable <u>antes</u> de devolver el valor de la variable)	-- antes del operando	echo --\$x; => 12
Postdecremento (decrementa la variable <u>después</u> de haber devuelto el valor de la variable)	-- después del operando	echo \$x--; => 13 echo \$x; => 12

[Subir](#)

Índice

Información general sobre PHP

- ¿Qué es PHP?
- Estructura básica de una página PHP
- Configuración de PHP
- Utilizar PHP desde la línea de comandos

Variables, constantes, tipos y matrices

- Constantes
- Variables
- Tipos de datos
- Matrices
- Algunas funciones útiles sobre las cadenas de caracteres, las fechas y los números

Operadores

- El operador de asignación por valor
- El operador de asignación por referencia
- Los operadores aritméticos
- El operador de cadena
- Los operadores de comparación
- Los operadores lógicos
- El operador ternario
- Los operadores combinados
- Precedencia de los operadores

Estructuras de control

Funciones y clases

Gestión de formularios

Acceder a las bases de datos

Administrar las sesiones

Enviar un correo electrónico

Gestión de archivos

Administrar los errores en un script PHP

Anexo

El operador de cadena

El único operador de cadena es el operador de concatenación, igual que el punto (.).

Sintaxis

```
cadena1.cadena2;
```

Esta sintaxis devuelve una cadena igual a la primera cadena inmediatamente seguida de la segunda; no se coloca ningún separador entre las dos cadenas.

Ejemplo

```
<?php
// Utilización del operador de concatenación con
// variables y expresiones literales.
$nombre = 'Olivier';
$apellido = 'Heurtel';
echo $apellido.', '.$nombre.'  

```

Resultado

```
Heurtel, Olivier
```

[Subir](#)

Índice

Información general sobre PHP

- ¿Qué es PHP?
- Estructura básica de una página PHP
- Configuración de PHP
- Utilizar PHP desde la línea de comandos

Variables, constantes, tipos y matrices

- Constantes
- Variables
- Tipos de datos
- Matrices
- Algunas funciones útiles sobre las cadenas de caracteres, las fechas y los números

Operadores

- El operador de asignación por valor
- El operador de asignación por referencia
- Los operadores aritméticos
- El operador de cadena
- Los operadores de comparación
- Los operadores lógicos
- El operador ternario
- Los operadores combinados
- Precedencia de los operadores

Estructuras de control

Funciones y clases

Gestión de formularios

Acceder a las bases de datos

Administrar las sesiones

Enviar un correo electrónico

Gestión de archivos

Administrar los errores en un script PHP

Anexo

Los operadores de comparación

Los operadores de comparación son los siguientes:

Operación	Operador	Ejemplo (\$x=13, \$y=8, \$z="8")
Igualdad	==	\$x == \$y => FALSE \$y == \$z => TRUE
Igualdad y tipos idénticos	===	\$x === \$y => FALSE \$y === \$z => FALSE
Diferente	!=	\$x != \$y => TRUE \$y != \$z => FALSE
Diferente o tipos diferentes	!==	\$x !== \$y => TRUE \$y !== \$z => TRUE
Inferior	<	\$x < \$y => FALSE \$y < \$x => TRUE \$y < \$z => FALSE
Inferior o igual	<=	\$x <= \$y => FALSE \$y <= \$x => TRUE \$y <= \$z => TRUE
Superior	>	\$x > \$y => TRUE \$y > \$x => FALSE \$y > \$z => FALSE
Superior o igual	>=	\$x >= \$y => TRUE \$y >= \$x => FALSE \$y >= \$z => TRUE

➡ No se debe confundir el operador de asignación (=) con el operador de comparación (==).

Subir

Índice

Información

[Título, autor...](#)

Introducción

[Objetivo del libro](#)[Breve historia de PHP](#)[¿Dónde conseguir PHP?](#)[Convenciones de escritura](#)

Información general sobre PHP

[Variables, constantes, tipos y matrices](#)[Operadores](#)[Estructuras de control](#)[Funciones y clases](#)[Gestión de formularios](#)[Acceder a las bases de datos](#)[Administrar las sesiones](#)[Enviar un correo electrónico](#)[Gestión de archivos](#)[Administrar los errores en un script PHP](#)[Anexo](#)

Objetivo del libro

El objetivo de este libro es aprender a desarrollar un sitio Web dinámico e interactivo usando PHP 5.

Para cumplir con este objetivo, este libro presenta de manera rápida las funciones básicas del lenguaje PHP antes de estudiar en detalle las características necesarias para desarrollar un sitio Web dinámico e interactivo:

- Gestión de formularios.
- Acceso a bases de datos, como MySQL, Oracle y SQLite.
- Gestión de sesiones (autenticación, gestión de un contexto, el uso de "cookies").
- Envío de correos electrónicos en formato HTML y con datos adjuntos.
- Gestión de archivos (incluyendo la transferencia de archivos desde la estación de trabajo del usuario al servidor "file upload").

Este libro está dirigido a gestores de proyecto, diseñadores y desarrolladores con un conocimiento básico de programación Web en HTML (*HyperText Markup Language*) y algunos conocimientos de SQL (*Structured Query Language* - lenguaje estándar de acceso a las bases de datos relacionales) para el capítulo sobre las bases de datos.

Este libro aborda la versión 5 de PHP; las nuevas características, específicas para esta versión, están claramente indicadas y se basa en la versión 5.5.0 publicada en junio de 2013.

[Subir](#)

Índice

Información general sobre PHP

[¿Qué es PHP?](#)

[Estructura básica de una página PHP](#)

[Configuración de PHP](#)

[Utilizar PHP desde la línea de comandos](#)

Variables, constantes, tipos y matrices

[Constantes](#)

[Variables](#)

[Tipos de datos](#)

[Matrices](#)

[Algunas funciones útiles sobre las cadenas de caracteres, las fechas y los números](#)

Operadores

[El operador de asignación por valor](#)

[El operador de asignación por referencia](#)

[Los operadores aritméticos](#)

[El operador de cadena](#)

[Los operadores de comparación](#)

[Los operadores lógicos](#)

[El operador ternario](#)

[Los operadores combinados](#)

[Precedencia de los operadores](#)

Estructuras de control

Funciones y clases

Gestión de formularios

Acceder a las bases de datos

Administrar las sesiones

Enviar un correo electrónico

Gestión de archivos

Administrar los errores en un script PHP

Anexo

Los operadores lógicos

Los operadores lógicos son los siguientes:

Operación	Operador(es)	Ejemplo
Y lógica	and &&	TRUE and TRUE => TRUE TRUE and FALSE => FALSE FALSE and FALSE => FALSE
O lógico	or 	TRUE or TRUE => TRUE TRUE or FALSE => TRUE FALSE or FALSE => FALSE
O lógico exclusivo (FALSE si ambos operandos son TRUE)	xor	TRUE xor TRUE => FALSE TRUE xor FALSE => TRUE FALSE xor FALSE => FALSE
No lógico	!	! TRUE => FALSE ! FALSE => TRUE

Los operadores `and` y `&`, así como `or` y `||` son idénticos, pero no tienen la misma precedencia (véase la sección Precedencia de los operadores).

[Subir](#)

Índice

Información general sobre PHP

- ¿Qué es PHP?
- Estructura básica de una página PHP
- Configuración de PHP
- Utilizar PHP desde la línea de comandos

Variables, constantes, tipos y matrices

- Constantes
- Variables
- Tipos de datos
- Matrices
- Algunas funciones útiles sobre las cadenas de caracteres, las fechas y los números

Operadores

- El operador de asignación por valor
- El operador de asignación por referencia
- Los operadores aritméticos
- El operador de cadena
- Los operadores de comparación
- Los operadores lógicos
- El operador ternario
- Los operadores combinados
- Precedencia de los operadores

Estructuras de control

Funciones y clases

Gestión de formularios

Acceder a las bases de datos

Administrar las sesiones

Enviar un correo electrónico

Gestión de archivos

Administrar los errores en un script PHP

Anexo

El operador ternario

Otro operador condicional, el operador ternario `?`, funciona como en el lenguaje C.

Sintaxis

`expresión1?expresión2:expresión3`

Esta instrucción devuelve el valor de `expresión2` si `expresión1` se evalúa como `TRUE` y el valor de `expresión3`, si `expresión1` se evalúa como `FALSE`. Si `expresión1` no es de tipo booleano, se realiza una conversión de acuerdo a las reglas descritas en el capítulo Variables, constantes, tipos y matrices.

Desde la versión 5.3, es posible omitir `expresión2:expresión1?: expresión3`. En este caso, la instrucción devuelve el valor de `expresión1` si `expresión1` se evalúa como `TRUE` y el valor de `expresión3` en caso contrario. Es el equivalente de `expresión1?expresión1:expresión3`.

Ejemplo

```
<?php
// Inicialización de una variable.
$nombre= '';
// Visualización de un mensaje que depende del valor de $nombre.
echo '¡Hola ' . (($nombre=='')?'desconocido':$nombre) . ' ! <br />';
echo '¡Hola ' . ($nombre?'desconocido') . ' !<br />'; //versión 5.3
// Asignación de un valor a la variable $nombre.
$nombre = 'Olivier';
// nuevo intento
echo '¡Hola ' . (($nombre=='')?'desconocido':$nombre) . ' ! <br />';
echo '¡Hola ' . ($nombre?'desconocido') . ' !<br />'; //versión 5.3
?>
```

Resultado

```
¡Hola desconocido!
¡Hola desconocido!
¡Hola Olivier!
¡Hola Olivier!
```

[Subir](#)

Índice

Información general sobre PHP

[¿Qué es PHP?](#)
[Estructura básica de una página PHP](#)
[Configuración de PHP](#)
[Utilizar PHP desde la línea de comandos](#)

Variables, constantes, tipos y matrices

[Constantes](#)
[Variables](#)
[Tipos de datos](#)
[Matrices](#)
[Algunas funciones útiles sobre las cadenas de caracteres, las fechas y los números](#)

Operadores

[El operador de asignación por valor](#)
[El operador de asignación por referencia](#)
[Los operadores aritméticos](#)
[El operador de cadena](#)
[Los operadores de comparación](#)
[Los operadores lógicos](#)
[El operador ternario](#)
[Los operadores combinados](#)
[Precedencia de los operadores](#)

Estructuras de control

Funciones y clases

Gestión de formularios

Acceder a las bases de datos

Administrar las sesiones

Enviar un correo electrónico

Gestión de archivos

Administrar los errores en un script PHP

Anexo

Los operadores combinados

Los operadores suma (+), diferencia (-), multiplicación (*), división (/), módulo (%) y concatenación (.) se pueden combinar con el operador de asignación (=) siguiendo la siguiente sintaxis:

Sintaxis	Equivalente a
<code>\$variable += expresión</code>	<code>\$variable = \$variable + expresión</code>
<code>\$variable -= expresión</code>	<code>\$variable = \$variable - expresión</code>
<code>\$variable *= expresión</code>	<code>\$variable = \$variable * expresión</code>
<code>\$variable /= expresión</code>	<code>\$variable = \$variable / expresión</code>
<code>\$variable %= expresión</code>	<code>\$variable = \$variable % expresión</code>
<code>\$variable .= expresión</code>	<code>\$variable = \$variable . expresión</code>

[Subir](#)

Índice

Información general sobre PHP

- ¿Qué es PHP?
- Estructura básica de una página PHP
- Configuración de PHP
- Utilizar PHP desde la línea de comandos

Variables, constantes, tipos y matrices

- Constantes
- Variables
- Tipos de datos
- Matrices
- Algunas funciones útiles sobre las cadenas de caracteres, las fechas y los números

Operadores

- El operador de asignación por valor
- El operador de asignación por referencia
- Los operadores aritméticos
- El operador de cadena
- Los operadores de comparación
- Los operadores lógicos
- El operador ternario
- Los operadores combinados
- Precedencia de los operadores

Estructuras de control

Funciones y clases

Gestión de formularios

Acceder a las bases de datos

Administrar las sesiones

Enviar un correo electrónico

Gestión de archivos

Administrar los errores en un script PHP

Anexo

Precedencia de los operadores

La precedencia de los operadores designa el orden en que se procesan los operadores en una expresión completa.

Al igual que en todos los lenguajes, se pueden utilizar los paréntesis para modificar el orden en el procesamiento de las operaciones. En la práctica, no dude en utilizar los paréntesis para evitar problemas y mejorar la legibilidad de las expresiones.

La precedencia de los operadores es la siguiente, desde el menos prioritario (se procesa el último) al más prioritario (se procesa el primero):

Operador

or

xor

and

= += -= *= /= %= . =

? :

||

&&

== != ===

< <= > >=

+ - .

* / %

! ++ -- (int) (double) (string) (array) (object)

Subir

Los controles condicionales

1. If

La estructura de control `if` permite una ejecución condicional de instrucciones.

Esta estructura tiene dos sintaxis:

Primera sintaxis

```
if (condición_1) {  
    instrucciones_1;  
[ ] elseif (condición_2) {  
    instrucciones_2; ]  
[ ... ]  
[ ] else {  
    instrucciones_n; ]  
}
```

El principio de funcionamiento de la estructura de control `if` es el siguiente:

- Si la `condición_1` es verdadera, las instrucciones `instrucciones_1` se ejecutan y, a continuación, el control se pasa a las instrucciones que siguen la estructura de control. Por ejemplo, la ejecución del programa sigue a la instrucción que sigue directamente al final de la estructura de control.
- Si la `condición_1` no es verdadera, el proceso se repite para las posibles parejas `condición_i/instrucciones_i` siguientes, introducidas por la palabra clave `elseif`.
- Si ninguna condición es verdadera, las instrucciones `instrucciones_n`, introducidas por la palabra clave `else` se ejecutan y luego se pasa el control a las instrucciones que siguen la estructura de control.

Puede haber varias cláusulas `elseif`.

Si las expresiones que definen las condiciones no son de tipo booleano, se realiza una conversión de acuerdo a las reglas descritas en el capítulo Variables, constantes, tipos y matrices.

Ejemplo

```
<?php  
// Estructura if / elseif / else  
$nombre = 'Olivier';  
$edad = NULL;  
if ($nombre == NULL) {  
    echo "¡Hola desconocido!<br />";  
} elseif ($edad == NULL) {  
    echo "¡Hola $nombre! No sé tu edad.<br />";  
} else {  
    echo "¡Hola $nombre! Tienes $edad años.<br />";  
}  
?>
```

Resultado

```
¡Hola Olivier! No sé tu edad.
```

La segunda sintaxis se utiliza principalmente para escribir una estructura de control en varios bloques PHP entre los que se inserta código HTML.

Segunda sintaxis (con código HTML incrustado)

```
<?php if (condición_1): ?>
    código_HTML_1
[ <?php elseif (condición_2): ?>
    código_HTML_2 ]
[ ... ]
[ <?php else: ?>
    código_HTML_n ]
<?php endif; ?>
```

El principio de análisis de la estructura `if - elseif - else` es el mismo que con la primera estructura, pero en lugar de la ejecución de instrucciones PHP, el motor incorpora en el resultado el código HTML asociado con la condición.

Ejemplo

```
<?php
// Un poco de aleatoriedad para definir las variables $nombre y $edad.
$nombre = rand(0,1)?'Olivier':NULL;
$edad = rand(0,1)?rand(7,77):NULL;
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
    <head>
        <title>Ejemplo de página PHP</title>
        <style type="text/css" media="all">
            .ko {font-weight: bold; color: red;}
            .ok {font-weight: bold; color: green;}
        </style>
    </head>
    <body>
        <div>
            <?php if ($nombre == NULL) : // condición PHP ?>
                <!-- Código HTML -->
                ;Hola desconocido!<br />
            <?php elseif ($edad == NULL) : // después de la condición ?>
                <!-- Código HTML -->
                Conozco tu <span class="ok">nombre</span>
                pero no tu <span class="ko">edad</span>.<br />
            <?php else : // después de la condición PHP ?>
                <!-- Código HTML -->
                Conozco tu <span class="ok">nombre</span>
                y tu <span class="ok">edad</span>,
                pero no se lo diré a nadie.<br />
            <?php endif; // fin de la condición PHP ?>
        </div>
    </body>
</html>
```

Resultado (depende de la asignación aleatoria de las variables)

Conozco tu **nombre** pero no tu **edad**.

Esta sintaxis es realmente muy práctica para realizar una construcción condicional de una página HTML,

evitando el uso pesado de un solo bloque PHP que genera todo el código HTML con la instrucción `echo`.

2. Switch

La estructura de control `switch`, equivalente a múltiples `if - elseif` se utiliza para comparar el resultado de una expresión con varios resultados.

Esta estructura tiene dos sintaxis:

Primera sintaxis

```
switch (expresión) {
    case expresión_1:
        instrucciones_1;
        [break;]
    [ case expresión_2:
        instrucciones_2;
        [break;] ]
    [ ... ]
    [ default:
        instrucciones_n;
        [break;] ]
}
```

El principio de funcionamiento de la estructura de control `switch` es el siguiente:

- Si la expresión es igual a `expresión_i` las instrucciones asociadas a `instrucciones_i` se ejecutan y se realizan las comparaciones si no hay una instrucción `break`.
- Si no se encuentra ninguna igualdad, se ejecutan todas las instrucciones `instrucciones_n` introducidas por la palabra clave `default`.

Puede haber varias cláusulas `case`.

Cuando se verifica una igualdad y se ejecutan las instrucciones correspondientes, la instrucción `switch` no se interrumpe y se evalúan las expresiones `casos` siguientes. Para interrumpir la ejecución de la instrucción `switch` y la evaluación de las cláusulas `case`, es necesario utilizar la instrucción `break` (fuerza la salida de la estructura de control).

Ejemplo

```
<?php
$nombre = rand(0,1)?'Olivier':NULL;
switch ($nombre) {
    case NULL :
        echo '¡Hola desconocido! ',
            'Voy a llamarte Olivier.<br />';
        $nombre = 'Olivier';
        break;
    case 'Olivier' :
        echo "¡Hola Maestro $nombre!<br />";
        break;
    default :
        echo "¡Hola alumno $nombre!<br />";
}
?>
```

Resultado (depende de la asignación aleatoria de las variables)

¡Hola desconocido! Voy a llamarte Oliver.

Segunda sintaxis (con código HTML incrustado)

```
<?php switch (expresión) :
  case (condición_1) : ?>
    código_HTML_1
  [ <?php case (condición_2) : ?>
    código_HTML_2 ]
  [ ... ]
  [ <?php default : ?>
    código_HTML_n ]
<?php endswitch; ?>
```

El primer case debe estar escrito en el bloque PHP switch.

Ejemplo

```
<?php
$idioma = 'es';
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Ejemplo de página PHP</title>
    <style type="text/css" media="all">
      .en {font-weight: bold; color: green;}
      .es {font-weight: bold; color: orange;}
      .fr {font-weight: bold; color: blue;}
      .desconocido {font-weight: bold; color: red;}
    </style>
  </head>
  <body>
    <div>
      <?php switch ($idioma) : // switch
        case 'en' : // primer case
      ?>
        <!-- Código HTML -->
        Hello <span class="en">my friend</span>!<br />
      <?php break; // break primer case ?>
      <?php case 'fr' : // segundo case ?>
        <!-- Código HTML -->
        Bonjour <span class="sp">mon pote</span> !<br />
      <?php break; // break segundo case ?>
      <?php case 'es' : // tercer case ?>
        <!-- Código HTML -->
        ¡Hola <span class="es">amigo</span>!<br />
      <?php break; // break tercer case ?>
      <?php default : // por defecto ?>
        <!-- Código HTML -->
        <span class="desconocido">?????</span>
      <?php endswitch; // fin del switch ?>
    </div>
  </body>
</html>
```

Resultado

;Hola **amigo!**

Índice

Utilizar PHP desde la línea de comandos

Variables, constantes, tipos y matrices

Constantes

Variables

Tipos de datos

Matrices

Algunas funciones útiles sobre las cadenas de caracteres, las fechas y los números

Operadores

El operador de asignación por valor

El operador de asignación por referencia

Los operadores aritméticos

El operador de cadena

Los operadores de comparación

Los operadores lógicos

El operador ternario

Los operadores combinados

Precedencia de los operadores

Estructuras de control

Los controles condicionales

Los controles iterativos

Incluir un archivo

Interrumpir el script

Funciones y clases

Gestión de formularios

Acceder a las bases de datos

Administrar las sesiones

Enviar un correo electrónico

Gestión de archivos

Administrar los errores en un script PHP

Anexo

Los controles iterativos

1. While

La estructura de control `while` permite ejecutar en bucle una serie de instrucciones siempre que una condición es verdadera.

Como para las estructuras de control condicionales, hay dos sintaxis disponibles.

Primera sintaxis

```
while (condición) {  
    instrucciones;  
}
```

El principio de funcionamiento de la estructura de control `while` es el siguiente: mientras que la condición condición sea verdadera, las instrucciones instrucciones se ejecutan.

Si la expresión que define la condición no es de tipo booleano, se realiza una conversión de acuerdo a las reglas descritas en el capítulo Variables, constantes, tipos y matrices.

Ejemplo

```
<?php  
// Inicializar dos variables.  
$nombre = 'OLIVIER';  
$longitud = strlen($nombre);  
// Inicializar un índice.  
$indice = 0;  
// Mientras el índice es inferior a la longitud de la cadena  
while ($indice < $longitud) {  
    // Mostrar el carácter correspondiente al índice seguido  
    // de un punto.  
    echo "$nombre[$indice].";  
    // Incrementar el índice  
    $indice++;  
}  
?>
```

Resultado

O.L.I.V.I.E.R.

Tradicionalmente, esta estructura adopta las siguientes conductas:

- Si la condición es falsa en la primera iteración, las instrucciones situadas dentro del bucle nunca se ejecutan.
- Si la condición no es nunca falsa, las instrucciones situadas dentro del bucle se ejecutan

Incluir un archivo

1. Funcionamiento

Las funciones `include`, `include_once`, `require` y `require_once` permiten incluir un archivo en un script PHP.

Sintaxis

```
entero include(archivo)
entero include_once(archivo)
require(archivo)
require_once(archivo)
```

`archivo` Nombre del archivo que se va a incluir (se puede especificar con una ruta absoluta o relativa).

Las funciones `include` e `include_once` devuelven 1 en caso de éxito y `FALSE` en caso de error. Las funciones `require` y `require_once` no tienen código de retorno.

En caso de error, las funciones `include` e `include_once` generan un simple error de nivel `E_WARNING` que no interrumpe la ejecución del script. Este no es el caso de las funciones `require` y `require_once` que causan un error fatal interrumpiendo la ejecución del script.

El archivo incluido puede contener código HTML, código PHP o ambos. Si contiene código PHP, este código debe estar escrito entre las etiquetas PHP habituales. El código HTML, presente en el archivo incluido, se integra tal cual en la página enviada al navegador, como si se encontrase en el script que realiza la llamada. El código PHP, presente en el archivo incluido, se ejecuta también, como si se encontrase en el script que realiza la llamada.

Cuando se incluye código PHP, variables y constantes, definidas en el archivo incluido se pueden usar en el script de llamada y viceversa. Todo sucede como si hubiera un sólo script tras la inclusión y, por lo tanto, un rango igual a este script para las variables y constantes.

Es posible incluir varios archivos en un script, o integrar las inclusiones (incluir un archivo que, a su vez, incluye un archivo en otro).

Con las funciones `include` y `require`, el proceso de inclusión se repite varias veces, si el mismo archivo se incluye en varias ocasiones.

En algunos casos este puede ser un comportamiento no deseado, especialmente cuando un archivo se incluye por primera vez directamente en un script y una segunda vez indirectamente a través de la inclusión de otro archivo.

Este comportamiento se puede evitar usando las funciones `include_once` y `require_once`, que garantizan que un archivo se incluye una sola vez, aunque se le llame varias veces.

➤ La extensión del archivo que se va a incluir es totalmente libre. Por ejemplo, no es obligatorio el uso de la extensión `.php` para incluir código PHP: el archivo incluido no se ejecuta directamente por el motor PHP (es el script de llamada el que se ejecuta). Para los archivos incluidos que no pueden o no deben ejecutarse directamente, es posible utilizar una extensión diferente (`.inc.`, por ejemplo).

Ejemplo: script principal


```

<?php
// Inclusión de un archivo
include('común.inc');
// Declaración de una variable $x en el script principal.
$x = 1;
// Visualización de la variable $x.
echo "Valor de \$x en el script principal: $x<br />";
// Visualización de la variable $y (definida en el archivo
// incluido).
echo "Valor de \$y en el script principal: $y<br />";
?>

```

Archivo incluido común.inc

```

<!-- Inicio del archivo de inclusión en modo HTML (apertura
---- de una etiqueta<b> que se cierra al final del archivo -->
<b>Inicio del archivo común.inc<br />
<?php // algo de código PHP
// Declaración de una variable $y en el script incluido.
$y = 2;
// Visualización de la variable $y.
echo "Valor de \$y en el archivo incluido: $y<br />";
?>
Fin del archivo común.inc</b><br />

```

Resultado

```

Inicio del archivo común.inc
Valor de $y en el archivo incluido: 2
Fin del archivo común.inc
Valor de $x en el script principal: 1
Valor de $y en el script principal: 2

```

- En el archivo `php.ini`, la directiva `include_path` permite definir rutas de búsqueda para la inclusión de archivos.

2. Utilización

La técnica de inclusión es práctica para dos tipos principales de uso:

- Incluir definiciones estáticas: constantes, definiciones de funciones. En este caso, es necesario sobre todo utilizar las funciones `include_once` y `require_once` para evitar una posible doble inclusión (lo que provocaría un error en la definición de las funciones).
- Incluir código PHP o HTML dinámico que se ejecuta efectivamente en el momento de la inclusión: sección HTML común a varias páginas (encabezado, pie de página) o código común a varias páginas (aunque en este caso, es más adecuado definir una función). En este caso, es necesario sobre todo utilizar las funciones `include` o `require` para garantizar que la inclusión se produce en cada llamada.

- La creación de funciones personalizadas se estudia en el capítulo Funciones y clases.

Ejemplo

- Archivo de definición de constantes (constantes.inc)

```
<?php
// Definición de constantes
// por ejemplo, el nombre del sitio.
DEFINE(NOMBRE_SITIO,'miSitio.com');
?>
```

- Archivo que contiene el principio de cada página (principio.inc)

```
<?php
// Inclusión del archivo de constantes.
include_once('constantes.inc');
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head><title><?php echo NOMBRE_SITIO; ?></title></head>
<body>
```

- Archivo que contiene el final de cada página (final.inc)

```
</body>
</html>
```

- Script de una página

```
<?php
// Inclusión del principio de la página.
include('principio.inc');
?>
<p>Contenido de la página ...</p>
<?php
// Inclusión del final de la página
include('final.inc');
?>
```

Resultado (código fuente de la página en el navegador)

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head><title>miSitio.com</title></head>
<body>
<p>Contenido de la página ...</p>
</body>
</html>
```

Interrumpir el script

Las instrucciones `exit` y `die` permiten interrumpir la ejecución del script (`die` es un alias de `exit`).

Sintaxis

```
exit[(cadena mensaje)];  
exit[(entero estado)];  
die[(cadena mensaje)];  
die[(entero estado)];
```

`mensaje` Mensaje que se va a mostrar antes de detener el script.

`estado` Estado de retorno (entero entre 1 y 254).

El script se detiene de repente y la visualización se queda "como está". Por lo tanto, la página HTML que se envía al navegador puede ser incoherente o estar vacía.

Si se llama a la instrucción o a la función dentro de un archivo incluido, el script principal se interrumpe.

Ejemplo (sin mensaje)

```
<?php  
// Generar el principio de la página.  
echo '¡Hola ';  
// No se ha verificado una condición, se interrumpe el script.  
if ($nombre == NULL) {  
    exit(1); // sin mensaje ...  
}  
// Continuar generando la página.  
echo $nombre;  
?>
```

Resultado

```
¡Hola
```

Ejemplo (con mensaje)

```
<?php  
// Generar el principio de la página.  
echo '¡Hola ';  
// No se ha verificado una condición, se interrumpe el script.  
if ($nombre == NULL) {  
    exit('<b>Usuario desconocido. No se puede continuar.</b>');  
}  
// Continuar la generación de la página.  
echo $nombre;  
?>
```

Funciones

1. Introducción

Al igual que en otros lenguajes de programación, PHP ofrece la posibilidad de definir sus propias funciones (llamadas funciones del "usuario") con todas las ventajas asociadas (modularidad, uso de mayúsculas...). Una función es un conjunto de instrucciones identificadas por un nombre, cuya ejecución devuelve un valor y culla llamada se puede utilizar como operando en una expresión. Un procedimiento es un conjunto de instrucciones identificadas por un nombre que puede ser llamado como una instrucción.

2. Declaración y llamada

La palabra clave `función` permite introducir la definición de una función.

Sintaxis

```
función nombre_función([parámetro]) {  
    instrucciones;  
}
```

<code>nombre_función</code>	Nombre de la función (debe respetar las reglas de denominación presentes en el capítulo Información general sobre PHP).
-----------------------------	---

<code>parámetro</code>	Parámetros posibles de la función expresados como una lista de variables (véase la sección Parámetros): <code>\$parámetro1</code> , <code>\$parámetro2</code> , ...
------------------------	---

<code>instrucciones</code>	Conjunto de instrucciones que componen la función.
----------------------------	--

Si la función devuelve un valor, es posible utilizar la instrucción `return` para definir el valor de retorno de la función (véase la sección Funciones - Valor de retorno).

El nombre de la función no debe ser una palabra reservada de PHP (nombre de función nativa, de instrucción) o ser igual al nombre de otra función definida de antemano.

Una función de usuario se puede llamar como una función nativa de PHP: en una asignación, en una comparación...


Ejemplo

```
<?php  
// Función sin parámetro que muestra "¡Hola!"  
// sin valor de retorno.  
función mostrar_hola() {  
    echo '¡Hola!<br />';  
}  
// Función con 2 parámetros que devuelve el producto  
// de los dos parámetros.  
función producto($valor1,$valor2) {  
    return $valor1 * $valor2;  
}  
// Llamada de la función mostrar_hola.  
mostrar_hola();
```

```
// Usos de la función producto:
// - en una asignación
$resultado = producto(2,4);
echo "2 x 4 = $resultado<br />";
// - en una comparación
if (producto(10,12) > 100) {
    echo '10 x 12 es superior a 100.<br />';
}
?>
```

Resultado

```
;Hola!
2 x 4 = 8
10 x 12 es superior a 100.
```

 En el lenguaje PHP, no existe un procedimiento real. Para definir algo equivalente a un procedimiento, basta con definir una función que no devuelva ningún valor y llamar a la función como si se tratara de una instrucción (como la función `mostrar_hola`, por ejemplo).

Desde la versión 5.4, cuando una función devuelve una matriz, es posible acceder directamente a un elemento de la matriz llamando a la función con una sintaxis de tipo `función(...)[clave]`.

Ejemplo

```
<?php
// Definición de una función que devuelve una matriz.
function quien() {
    return ['Olivier','Heurtel'];
}
// Llamada a la función y recuperación directa del nombre almacenado
// en el índice 0 de la matriz devuelta.
$nombre = quien()[0];
echo "quien()[0] = $nombre<br />";
?>
```

Resultado

```
quien()[0] = Olivier
```

Esta técnica funciona también cuando la función devuelve una matriz multidimensional con una sintaxis de tipo `función(...)[clave1][clave2]`.

Es posible utilizar una función antes de definirla.

Ejemplo

```
<?php
// Utilización de la función producto.
echo producto(5,5);
// Definición de la función producto.
función producto($valor1,$valor2) {
    return $valor1 * $valor2;
}
?>
```

Resultado

Por lo tanto, no hay ningún problema para definir funciones que llamen entre ellas.

- Una función se puede utilizar sólo en el script donde se define. Para utilizarla en varios scripts, es necesario, bien copiar su definición en los diferentes scripts (se pierde el interés por definir una función), o bien definirla en un archivo incluido donde la función sea necesaria.

Ejemplo

- Archivo `funciones.inc` que contiene la definición de funciones:

```
<?php
// Definición de la función producto.
función producto($valor1,$valor2) {
    return $valor1 * $valor2;
}
?>
```

- Script que utiliza las funciones definidas en `funciones.inc`:

```
<?php
// Inclusión del archivo contenedor de la definición de las funciones.
include('funciones.inc');
// Utilización de la función producto.
echo producto(5,5);
?>
```

Función variable

PHP ofrece una "función variable" que permite almacenar un nombre de función en una variable y llamar a la variable en una instrucción, como si se tratara de una función, con la notación `$variable()`. En tal escritura, PHP sustituye la variable por su valor y trata de ejecutar la función correspondiente (que debe, por supuesto, existir).

Ejemplo

```
<?php
// Función que efectúa un producto.
función producto($valor1,$valor2) {
    return $valor1 * $valor2;
}
// Función que efectúa una suma.
función suma($valor1,$valor2) {
    return $valor1 + $valor2;
}
// Función que efectúa un cálculo, el nombre del cálculo
// ('suma' o 'producto') se pasa como parámetro.
función calcular($operación,$valor1,$valor2) {
    // $operación contiene el nombre de la función
    // a ejecutar => llamada $operación().
    return $operación($valor1,$valor2);
}
// Utilización de la función calcular.
echo '3 + 5 = ',calcular('suma',3,5).'  
';
echo '3 x 5 = ',calcular('producto',3,5).'  
';
?>
```

Resultado

```
3 + 5 = 8
3 x 5 = 15
```

3. Parámetros

Los parámetros (también llamados argumentos) posibles de la función se definen en forma de una lista de variables.

En este tercer punto, vamos a estudiar las siguientes posibilidades:

- definir un valor predeterminado para un parámetro;
- pasar un parámetro por referencia;
- utilizar una lista variable de parámetros.

Valor predeterminado

Es posible indicar que un parámetro tenga un valor predeterminado con la siguiente sintaxis:

```
$parámetro = expresión literal
```

El valor predeterminado de un parámetro debe ser una expresión literal y puede ser ni una variable, ni una función, ni una expresión compuesta.

El valor predeterminado se utiliza como valor de un parámetro cuando se llama a la función, sin mencionar el valor para el parámetro en cuestión.

Ejemplo

```
<?php
// Definición de una constante
define('UNO',1);
// Definición de la función producto con valores
// predeterminados para los parámetros (incluyendo una constante
// para el primer parámetro).
función producto($valor1=UNO,$valor2=2) {
    return $valor1* $valor2;
}
// Llamadas
// - sin parámetro
echo 'producto() = ',producto(),'<br />';
// - con un único parámetro = obligatoriamente el primero
echo 'producto(3) = ',producto(3),'<br />';
?>
```

Resultado

```
producto() = 2
producto(3) = 6
```

No dar ningún valor a un parámetro que tenga un valor predeterminado sólo es posible desde la derecha. Pase un valor "vacío" ("" o NULL) no resuelve el problema porque PHP convierte el valor en cuestión en el tipo apropiado (en este caso, entero igual a 0).

- Pasar un número insuficiente de parámetros y no tener ningún valor predeterminado genera un error. Pasar demasiados parámetros no genera ningún error; los parámetros en exceso se pasan por alto.

Pase por referencia

Por defecto, el pase de parámetros se realiza por valor: se trata de una copia del valor que se pasa a la función. Por lo tanto, la modificación de los parámetros dentro de la función no tiene efecto sobre los valores en el script de llamada.

Ejemplo

```
<?php
// Definición de una función que toma un parámetro.
función por_valor($parámetro) {
    // Incrementación del parámetro.
    $parámetro++;
    // Visualización del parámetro en el interior de la función.
    echo "\$parámetro = $parámetro<br />";
}
// Inicialización de una variable.
$x = 1;
// Visualización de la variable antes de la llamada a la función.
echo "\$x antes de la llamada = $x<br />";
// Llamada de la función utilizando la variable como valor
// del parámetro.
por_valor($x);
// Visualización de la variable después de la llamada a la función.
echo "\$x después de la llamada = $x<br />";
?>
```

Resultado

```
$x antes de la llamada = 1
$parámetro = 2
$x después de la llamada = 1
```

Si es necesario, es posible tener un pase de referencia utilizando el operador de referencia & (véase el capítulo Operadores - sección El operador de asignación por referencia) delante del nombre del parámetro en la definición de la función. Con esta definición, a la función se le pasa una referencia a la variable (no una copia); esta última trabaja directamente en la variable del script de llamada.

Ejemplo

```
<?php
// Definición de una función que toma un parámetro.
función por_referencia(&$parámetro) {
    // Incrementación del parámetro.
    $parámetro++;
    // Visualización del parámetro en el interior de la función.
    echo "\$parámetro = $parámetro<br />";
}
// Inicialización de una variable.
$x = 1;
// Visualización de la variable antes de la llamada a la función.
echo "\$x antes de la llamada = $x<br />";
// Llamada de la función utilizando la variable como valor
```



```
// del parámetro.
por_referencia($x);
// Visualización de la variable después de la llamada a la función.
echo "\$x después de la llamada = <b>$x</b><br />";
?>
```

Resultado

```
$x antes de la llamada = 1
$parámetro = 2
$x después de la llamada = 2
```

Con esta definición, no es posible pasar constante o una expresión como valor del parámetro.

Ejemplos prohibidos

```
por_referencia(2);
por_referencia(1+1);
por_referencia(UNO); // UNO = constante definida anteriormente
```

Lista variables de parámetros

Dentro de una función, es posible utilizar las siguientes tres funciones de PHP:

Función	Función
func_num_args	Indica el número de parámetros pasados a la función.
func_get_args	Devuelve la lista de parámetros pasados a la función (en una matriz).
func_get_arg	Devuelve el valor de un parámetro cuyo número se especifica.

Sintaxis

```
entero func_num_args()
matriz func_get_args()
mixto func_get_arg(entero número)
```

número Número del parámetro solicitado (0 = primer parámetro).

Gracias a estas funciones nativas, es muy fácil escribir una función que acepta un número variable de parámetros. Los principios son los siguientes:

- declarar la función sin parámetros;
- recuperar, en el cuerpo de la función, los parámetros con las funciones `func_get_args` o `func_get_arg` y utilizarlos (normalmente en un bucle).

En la práctica, nada impide que los parámetros sean de diferentes tipos.

Por otra parte, es posible declarar explícitamente los parámetros y luego aceptar una lista de variables de parámetros adicionales. En este caso, los parámetros explícitamente declarados se retoman en el recuento y en la lista de parámetros. Por tanto, es necesario eliminarlos del procesamiento.

Ejemplo

```
<?php
// Función que acepta un primer parámetro por referencia
```

```
// y que almacena el producto de los demás parámetros.
función producto(&$resultado) {
    switch (func_num_args()) {
        caso 1:
            // Un único parámetro (la variable para el resultado)
            // Devolver 0 (elección arbitraria).
            $resultado = 0;
            break;
        default:
            // Recuperar los parámetros en una matriz
            // y eliminar el primer elemento (el primer
            // parámetro).
            $parámetros = func_get_args();
            unset($parámetros[0]);
            // Inicializar el resultado a 1.
            $resultado = 1;
            // Hacer un bucle en la matriz de parámetros
            // para multiplicar el resultado por el parámetro.
            foreach($parámetros as $parámetro) {
                $resultado *= $parámetro;
            }
            break;
    }
}

// llamadas
producto($resultado);
echo 'producto($resultado) => ', $resultado, '<br />';
producto($resultado, 1, 2, 3);
echo 'producto($resultado, 1, 2, 3) => ', $resultado, '<br />';
?>
```

Resultado

```
producto($resultado) => 0
producto($resultado, 1, 2, 3) => 6
```

4. Valor de retorno

El uso de la instrucción `return` en el interior de una función, permite definir el valor de retorno de la función y detener su ejecución.

Sintaxis

```
return expresión;
```

expresión Expresión cuyo resultado constituye el valor de retorno de la función.

El resultado de una función puede ser de cualquier tipo (cadena, número, matriz...).

Si hay varias instrucciones `return` presentes en la función, es la primera encontrada en el curso de las instrucciones la que define el valor de retorno y provoca la interrupción de la función.

Ejemplo

```
<?php
// Definición de una función con dos llamadas a return.
función valor_retorno($parámetro) {
```

```

    if ($parámetro == 1) {
        return 'Primer return';
    }
    return 'Segundo return';
}
// Llamadas a la función.
echo 'valor_retorno(1) = '.valor_retorno(1).'<br />';
echo 'valor_retorno(0) = '.valor_retorno(0).'<br />';
?>

```

Resultado

```

valor_retorno(1) = Primer return
valor_retorno(0) = Segundo return

```

Si la función no tiene ninguna instrucción `return` (o si no se ejecuta ninguna instrucción `return`), el valor de retorno de la función es `NULL`.

5. Consideraciones sobre las variables utilizadas en las funciones

Variable locale/global

Las variables utilizadas dentro de una función son locales: no se definen fuera de la función y se inicializan en cada llamada a la función. Lo mismo ocurre con los parámetros de la función.

Por el contrario, una variable definida fuera de la función (en el script de llamada) no está definida dentro de la función.

PHP ofrece un concepto de variable global para acceder, en una función, a las variables definidas dentro del contexto del script de llamada.

Para ello, dentro de la función, es necesario declarar las variables globales que la función se utiliza con la instrucción `global` o utilizar la matriz asociativa predefinida `$GLOBALS`.

Sintaxis

```
global $variable[, ...];
```

`$variable` Variable del script de llamada que la función quiere usar. Se pueden especificar varias variables, separadas por una coma.

La matriz predeterminada `$GLOBALS` es una matriz asociativa. En esta matriz asociativa, la clave es igual al nombre de la variable global (sin el \$) y el valor asociado es igual al valor de la variable global.



La matriz asociativa `$GLOBALS` es una matriz "superglobal". Una matriz superglobal está automáticamente disponible en todos los entornos de ejecución, sin tener que haberla declarado como global. En este libro, utilizaremos otras matrices superglobales.

Ejemplo

```

<?php
// Objetivo: escribir una función que efectúe el producto
//           de las variables $x y $y y que almacene el resultado
//           en la variable $z.

```

```
// Inicialización de dos variables en el script de llamada.
$x = 2;
$y = 5;
echo '<b>Caso 1: sin utilización de variables ',
    'globales</b><br />';
función producto1() {
    // $x y $y están vacías en el interior de la función.
    echo "\$x = $x<br />";
    echo "\$y = $y<br />";
    $z = 0 + $x * $y;
}
producto1();
// $z está vacía en el script principal.
echo "\$z = $z<br />";
// Resolución del problema utilizando variables globales:
// - con la palabra clave global para $x y $y
// - con la matriz $GLOBALS para $z
echo '<b>Caso 2: utilización de variables globales</b><br />';
función producto2() {
    global $x, $y;
    echo "\$x = $x<br />";
    echo "\$y = $y<br />";
    $GLOBALS['z'] = 0 + $x * $y;
}
producto2();
echo "\$z = $z<br />";
?>
```

Resultado

Caso 1: sin utilización de variables globales

```
$x =
$y =
$z =
```

Caso 2: utilización de variables globales

```
$x = 2
$y = 5
$z = 10
```



Un parámetro de una función se comporta como una variable local a la función, a menos que se pase por referencia, en cuyo caso es equivalente a una variable global.

Variable estática

De forma predeterminada, las variables locales de una función se restablecen en cada llamada a la función.

La instrucción `static` permite definir variables locales estáticas que tienen la propiedad de conservar el valor de una llamada a otra función, durante la duración del script.

Sintaxis

```
static $variable = expresión_literal[, ...];
```

`$variable` Variable en cuestión.

`expresión_literal` Valor inicial asignado a la variable durante la primera

llamada a la función dentro del script. Sólo se aceptan las expresiones literales y las constantes; las expresiones compuestas o las funciones no están permitidas.

Ejemplo

```
<?php
// Definición de una función.
función variable_estática() {
    // Inicialización de una variable estática.
    static $variable_estática = 0;
    // Inicialización de otra variable.
    $otra_variable = 0;
    // Visualización de las dos variables.
    echo "\$variable_estática = $variable_estática <br />";
    echo "\$otra_variable = $otra_variable<br />";
    // Incrementación de las dos variables.
    $variable_estática++;
    $otra_variable++;
}
// Primera llamada de la función.
echo '<b>Primera llamada de la función:</b><br />';
variable_estática();
// ...
// Segunda llamada de la función.
echo '<b>Segunda llamada de la función:</b><br />';
variable_estática();
//...
// Tercera llamada de la función.
echo '<b>Tercera llamada de la función:</b><br />';
variable_estática();
?>
```

Resultado

Primera llamada de la función:

```
$variable_estática = 0
$otra_variable = 0
```

Segunda llamada de la función:

```
$variable_estática = 1
$otra_variable = 0
```

Tercera llamada de la función:

```
$variable_estática = 2
$otra_variable = 0
```

Este ejemplo muestra que el valor de `$variable_estática` se conserva de una llamada a otra, la primera llamada permite inicializarlo. Por contra, la variable `$otra_variable` se pone a 0 en cada llamada de la función.



El valor sólo se conserva por la duración del script: cuando éste termina, el valor se pierde y, en la siguiente llamada del script, la variable estática se restablece.

6. Las constantes y las funciones

En el capítulo Variables, constantes, tipos y matrices, se observó que el alcance de las constantes es el

script en el que se definen.

A diferencia de las variables, el ámbito de aplicación se extiende a las funciones llamadas en el script: una constante se puede utilizar dentro de la función sin ser declarada global.

Por el contrario, una constante definida en una función se puede utilizar en un script, después de llamar a la función.

Ejemplo

```
<?php
// Definición de una constante en el script.
define(CONSTANTE_SCRIPT,'constante script');
// Definición de una función.
función constante() {
    // Que define una constante.
    define(CONSTANTE_FUNCIÓN,'constante función');
    // Y que muestra una constante del script de llamada.
    echo 'En la función, CONSTANTE_SCRIPT = ',
        CONSTANTE_SCRIPT,'<br />';
}
// Llamada de la función.
constante();
// Visualización de la constante definida en la función.
echo 'En el script, CONSTANTE_FUNCIÓN= ',
    CONSTANTE_FUNCIÓN,'<br />';
?>
```

Resultado

En la función, CONSTANTE_SCRIPT = constante script
 En el script, CONSTANTE_FUNCIÓN = constante función

7. Recursividad

Al igual que muchos lenguajes, PHP permite la recursividad, es decir, la posibilidad de que una función se llame a sí misma.

Para ilustrar este concepto, vamos a escribir una función genérica que muestra que permite mostrar el contenido de una matriz, posiblemente multidimensional.

Fuente

```
<?php
función mostrar_matriz($matriz,$título='', $nivel=0) {
    // parámetros
    // - $matriz = matriz cuyo contenido se debe mostrar
    // - $título = título que se debe mostrar sobre el contenido
    // - $nivel = nivel de visualización
    // Si hay un título, mostrarlo.
    if ($título != '') {
        echo "<br /><b>$título</b><br />";
    }
    // Probar si hay datos.
    if (isset($matriz)) { // hay datos
        // Examinar la matriz pasada en parámetro.
        reset ($matriz);
        foreach ($matriz as $clave => $valor) {
```

```
// Mostrar la clave (con sangría en función del nivel).
echo
    str_pad('',12*$nivel, ' &nbsp;'),
    htmlentities($clave),' = ';
// Mostrar el valor
if (is_array($valor)) { // es una matriz ...
    // etiquetar <br />
    echo '<br />';
    // Y llamar de manera recursiva mostrar_matriz para
    // mostrar la matriz en cuestión (sin título y
    // al nivel siguiente de sangría)
    mostrar_matriz($valor,'',$nivel+1);
} else { // es un valor escalar
    // Mostrar el valor.
    echo htmlentities($valor),'<br />';
}
}
} else { // sin datos
    // Poner una etiqueta simple <br />
    echo '<br />';
}
}
// Mostrar una matriz de colores.
$colores = array('Azul','Blanco','Rojo');
mostrar_matriz($colores,'Colores');
// Mostrar una matriz de dos dimensiones (país/ciudad).
$país = array('España' => array('Madrid','León','Barcelona'),
              'Italia' => array('Roma','Venecia'));
mostrar_matriz($país,'País/Ciudades');
?>
```

Resultado

Colores

0 = Azul
1 = Blanco
2 = Rojo

País/Ciudades

España =
0 = Madrid
1 = León
2 = Barcelona
Italia =
0 = Roma
1 = Venecia

Para ser rigurosos, debemos comprobar que la variable pasada inicialmente en el primer parámetro es una matriz.

Vamos a utilizar esta función varias veces en este libro, asumiendo que es parte de un conjunto de funciones genéricas definidas en un archivo incluido en el script si es necesario.

8. Función anónima

Una función anónima, también llamada de cierre o *closure* en inglés, es una función a la que no se le ha asignado ningún nombre en su creación. Este concepto apareció en la versión 5.3.

Una función anónima se puede utilizar como el valor de una variable (función variable) o como una

función de devolución de llamada a otra función.

Ejemplo

```
<?php
// Definición de una función anónima almacenada en una variable.
$función_anónima = función ($nombre) {
    echo "¡Hola $nombre!<br />";
};
// Llamada a la función anónima.
$función_anónima('todo el mundo');
// Utilización de la función anónima como función de devolución de llamada.
$nombrres = array('Olivier', 'David', 'Tomás');
array_walk($nombrres, $función_anónima);
?>
```

Resultado

```
¡Hola todo el mundo!
¡Hola Olivier!
¡Hola David!
¡Hola Tomás!
```

La función `array_walk` es una función que permite ejecutar una función (llamada función de devolución de llamada) en los elementos de una matriz.

La función anónima se puede definir directamente en la llamada.

Ejemplo

```
<?php
$nombrres = array('Olivier', 'David', 'Tomás');
array_walk
(
    $nombrres,
    función ($nombre) {echo "¡Hola $nombre!<br />";}
);
?>
```

Una función anónima puede importar variables del contexto padre. Para ello, tras la firma de la función, basta con utilizar la palabra clave `use` seguida de la lista de variables que deben importarse entre paréntesis.

Ejemplo

```
<?php
// Definición de una variable.
$nombre = 'Olivier';
// Definición de una función anónima almacenada en una variable
// esta función importa la variable $nombre.
$función_anónima = function () use ($nombre) {
    echo "¡Hola $nombre!<br />";
};
// Llamada de la función anónima.
$función_anónima();
?>
```

Resultado


```
;Hola Olivier!
```

Si la variable importada debe modificarse por la función anónima, es necesario importarla con el operador & (use (&\$nombre)).

9. Función generadora

Un generador (*generator* en inglés) o función generadora es una función que genera una lista de valores que podrán explorarse en el programa mediante una llamada con la ayuda de la estructura `foreach` (como para una matriz). Esta técnica, aparecida en la versión 5.5, es interesante ya que permite generar una lista de valores sin pasar por un almacenamiento en una matriz que podría consumir mucha memoria. Antes de la versión 5.5, era posible obtener el mismo resultado, pero a cambio de una mayor complejidad a la hora de definir una clase que implementase la interfaz `iterator`.

Una función generadora no devuelve ningún valor y, por tanto, no comporta instrucción `return`. En su lugar, para proporcionar un valor a la persona que realiza la llamada, la función utiliza la palabra clave `yield`.

Sintaxis

```
yield [clave =>] valor;
```

valor Expresión cuyo resultado constituye el valor proporcionado por la función.

Clave Expresión cuyo resultado constituye la clave asociada al valor proporcionado por la función. Si no se especifica esta cláusula, los valores proporcionados se indexan por defecto a partir de 0.

Cuando se llama a una función generadora, en realidad devuelve un objeto de la clase interna `Generator` que implementa la interfaz `Iterator`. Cuando este objeto se explora por un bucle `foreach`, PHP llama a la función para recuperar un valor, hasta que no haya más valores (como para una matriz). En cada iteración, la función proporciona el valor con la ayuda de la palabra clave `yield` y su ejecución se suspende hasta la siguiente iteración.

Primer ejemplo: generación de una lista de valores

```
<?php
// definición de una función que genera números aleatorios
función lanzador_de_dados($numero=3) {
    for ($i = 1; $i <= $numero; $i++) {
        yield rand(1,6);
    }
}
// llamada de la función en una variable
$valores = lanzador_de_dados();
// dump de la variable (para ver)
echo '<b>Dump de la variable:</b><br />';
var_dump($valores);
echo '<b> // es un objeto</b><br />';
// exploración de los valores
echo '<b>Exploración de los valores:</b><br />';
foreach ($valores as $valor) {
    echo "$valor ";
}
echo '<br />';
```

```
// nueva llamada (con 5 valores) y exploración directamente
// en la estructura foreach
echo '<b>Nueva generación (5 valores):</b><br />';
foreach (lanzador_de_dados(5) as $valor) {
    echo "$valor ";
}
?>
```

Resultado

Dump de la variable:

```
object(Generator)#1 (0) { } // es un objeto
```

Exploración de los valores:

```
5 6 3
```

Nueva generación (5 valores):

```
4 4 1 4 3
```

Segundo ejemplo: generación de una lista de parejas clave/valor

```
<?php
// definición de una función que proporciona una a una las letras de un
// texto utilizando el código ASCII de la letra como índice
function letras($texto) {
    for ($i = 0; $i < strlen($texto); $i++) {
        yield ord($texto[$i]) => $texto[$i];
    }
}
foreach (letras('OLIVIER') as $code => $letra) {
    echo "$letra ($code) ";
}
?>
```

Resultado

```
O (79) L (76) I (73) V (86) I (73) E (69) R (82)
```

Clases

1. Concepto

En la versión 5, la gestión de objetos en PHP se ha reescrito completamente para proporcionar un mejor rendimiento y más características.

PHP ofrece ahora características clásicas de programación orientada a objetos:

- definición de clase.
- uso de métodos constructor y destructor.
- conceptos de atributo o de método público, privado, protegido.
- legado.
- conceptos de clase o método abstracto, de clase o método final, de interfaz, de atributo o método estático (de clase).
- excepciones.

Una clase es un tipo compuesto que reagrupa variables (llamadas atributos de la clase) y funciones (llamadas métodos de la clase). Por sí misma, una clase no contiene ningún dato; es sólo un modelo, una definición.

A partir de la clase, es posible definir ("instanciar") objetos que tienen la estructura de la clase y que contienen datos.

En esta sección, vamos a presentar las características básicas más utilizadas: es una introducción práctica a las características orientadas a objetos de PHP. Para obtener más información, consulte la documentación de PHP.

2. Definir una clase

La palabra clave `class` permite introducir la definición de una clase.

Sintaxis

```
class nombre_clase {  
    // definición de los atributos  
    [  
    public | private | protected $atributo [= expresión_literal];  
    ...  
    ]  
    // definición de los métodos  
    [  
    [public | private | protected] function método() {  
        ...  
    }  
    ...  
    ]  
}
```

nombre_classe

Nombre de la clase (debe respetar las reglas de denominación presentes en el capítulo Información general sobre PHP).

<code>\$atributo</code>	Nombre de una variable que corresponde a un atributo de la clase.
<code>expresión_literal</code>	Valor inicial del atributo. Sólo se aceptan las expresiones literales y las constantes; las expresiones compuestas o las funciones no están permitidas.
<code>método</code>	Definición de una función que corresponde a uno de los métodos de la clase.

La visibilidad de los atributos y los métodos se define por una de las siguientes palabras clave:

<code>public</code>	El atributo o método es público y se puede acceder a él desde el exterior de la clase.
<code>private</code>	El atributo o método es privado y sólo se puede acceder a él desde el interior de la clase.
<code>protected</code>	El atributo o el método está protegido y sólo se puede acceder a él desde el interior de la clase o de las clases derivadas de la clase (ver el concepto de legado).

De forma predeterminada, un método es público. Por contra, la visibilidad del atributo debe especificarse.

Los métodos se definen como las funciones de usuario clásicas (con parámetros, instrucción `return...`), pero dentro de la clase. Un método de una clase puede tener el mismo nombre que una función de usuario o que otro método de otra clase.

Uno de los métodos puede tener el mismo nombre que la clase en la que se define. A este método particular, llamado método constructor, se le llama automáticamente al crear ("instanciar") un nuevo objeto. En general, este método se utiliza para inicializar los atributos que no puede serlo por una simple expresión literal. El método constructor de una clase puede ser llamado de manera unificada `__construct`. Este método de denominación es recomendable porque facilita la llamada del método constructor de una clase padre en una clase derivada (ver el concepto de legado). En primer lugar, PHP siempre busca un método llamado `__construct`; si no lo encuentra, busca un método con el mismo nombre que la clase.

Además, es posible especificar un método destructor llamado `__destruct` (sin parámetros). A este método destructor se le llama automáticamente cuando se elimina la última referencia a un objeto. Este método se puede utilizar para liberar los recursos utilizados por el objeto (del tipo archivo, información en una base de datos, etc.).

Por último, es posible especificar un método llamado `__toString` (sin parámetros), que permite convertir un objeto en cadena. A este método se le llama automáticamente cada vez que se utiliza el objeto en un contexto en el que PHP espera una cadena (por ejemplo, en un `echo`). Este método debe devolver una cadena en la que se puede integrar la información que desee sobre el objeto.

Dentro de los métodos, el objeto actual (o instancia actual) se puede referenciar por la variable `$this`; para acceder a los atributos o a los métodos, basta con utilizar el operador `->` seguido del nombre del atributo o del nombre del método, detrás del nombre de la variable `$this`.

Sintaxis

```
$this->atributo
$this->método([valor[, ...]])
```


El signo `$` va en el nombre de la variable `this`, pero no en el nombre del atributo.

Ejemplo

```

<?php
// Definición de una clase destinada a almacenar información
// sobre un usuario.
class usuario {
    // Definición de los atributos.
    public $apellido; // nombre del usuario
    public $nombre; // nombre del usuario
    public $idioma = 'es_ES'; // idioma del usuario
                                // español por defecto
    private $timestamp; // fecha/hora de creación (privado)
    // Definición de los métodos:
    // - método constructor
    public function __construct($nombre,$apellido) {
        // Inicializar el apellido y el nombre
        // con los valores parametrados.
        $this->nombre = $nombre;
        $this->apellido = $apellido;
        // Inicializar la marca de tiempo con la función time().
        $this->timestamp = time();
    }
    // - método destructor
    public function __destruct() {
        // Basta con mostrar un mensaje.
        echo "<p><b>Eliminación de $this->apellido</b></p>";
    }
    // - método de conversión del objeto en cadena
    public function __toString() {
        // Devuelve el apellido y el nombre.
        return "__toString = $this->apellido - $this->nombre";
    }
    // - método que modifica el idioma del usuario.
    public function idioma($idioma) {
        $this->idioma = $idioma;
    }
    // - método (privado) que da formato a la fecha/hora
    // de creación del usuario.
    private function FormatoDeLaMarcaDeTiempo() {
        setlocale(LC_TIME,$this->idioma);
        return strftime('%c', $this->timestamp);
    }
    // - método que da información sobre el usuario
    public function información() {
        $creación = $this->FormatoDeLaMarcaDeTiempo();
        return "$this->nombre $this->apellido - $creación";
    }
}
?>

```

 Una clase se puede utilizar sólo en el script donde se define. Para poder utilizarla en varios scripts, es necesario, bien copiar su definición en los diferentes scripts (se pierde el interés por definir una clase), o bien definirla en un archivo incluido donde la clase sea necesaria.

3. Instanciar una clase

Una instancia significa crear un objeto basado en la definición de la clase. Hasta cierto punto, esto equivale a la definición de una variable que tiene como "tipo" la clase.

La creación de instancias se realiza mediante el operador `new`.

Sintaxis

```
$nombre_objeto = new nombre_clase[(valor[, ...])]
```

<code>\$nombre_objeto</code>	Variable para almacenar el objeto.
<code>nombre_clase</code>	Nombre de la clase que sirve como "modelo" para el objeto.
<code>valor</code>	Posible parámetro que se pasa al método constructor de la clase llamada durante la creación del objeto.

Después de la creación del objeto, los atributos y métodos públicos del objeto se pueden obtener con el operador `->`, en la variable `$nombre_objeto`, como con la variable `$this`.



No hay protección de los atributos públicos de los objetos; se pueden manipular directamente.

Ejemplo

```
<?php
// Inclusión del archivo que contiene la definición de la
// clase del usuario presentada anteriormente.
include('clases.inc');
// Instanciación de un objeto.
$yo = new usuario('Olivier','Heurtel');
// La variable $yo contiene ya un objeto basado en la
// clase del usuario. Los métodos son accesibles por el
// operador ->.
// Utilización de los métodos del objeto.
echo "{$yo->información()} <br/>";
$yo->idioma('en_US'); // modificación del idioma
echo "{$yo->información()} <br/>";
// Modificación y lectura directa de un atributo público
$yo->apellido = strtoupper($yo->apellido);
echo "$yo->apellido <br />";
// Visualización directa del objeto => utilización de __toString
echo "$yo <br />";
?>
```

Resultado

```
Olivier Heurtel - mar 25 jun 2013 16:23:13 CET
Olivier Heurtel - Tue 25 Jun 2013 04:23:13 PM CET
HEURTEL
__toString = HEURTEL - Olivier
Eliminación de HEURTEL
```

Como muestra este ejemplo, las variables objeto pueden sustituirse igual que el resto (ver capítulo Variables, constantes, tipos y matrices - sección Tipos de datos) en las cadenas de caracteres delimitadas con comillas dobles. Si es necesario, se pueden utilizar llaves para delimitar la variable dentro de la cadena.

Desde la versión 5.4, es posible acceder a un miembro de una clase (atributo o método) durante la instanciación, con una sintaxis de tipo `(new clase(...))->miembro`.

Ejemplo


```
<?php
// Inclusión del archivo que contiene la definición de la
// clase de usuarios presentada anteriormente.
include('clases.inc');
// Llamada a un método durante la instanciación de un objeto.
$información = (new usuario('Víctor','Hugo'))->información();
echo "$información <br/>";
?>
```

Resultado

Supresión de Hugo

Víctor Hugo - mar 25 jun 2013 17:15:03 CET

Con este tipo de llamada, el nuevo objeto se suprime inmediatamente después de su creación, lo que explica por qué el mensaje de supresión se muestra antes de la información.

-  Las mismas reglas de alcance y duración de las variables se aplican a los objetos (ver capítulo Variables, constantes, tipos y matrices - sección Variables).

4. Legado

Es posible definir una nueva clase que hereda de una clase existente con la palabra clave `extends`.

Sintaxis

```
class nombre_clase extends nombre_clase_de_base{
// Definición de atributos adicionales.
[
public | private | protected $atributo [= expresión_literal];
...

]

// Definición de métodos adicionales.
[
public | private | protected function método() {
...
}
...
]
}
```

El significado de los diferentes elementos es el mismo que para la definición de una clase.

La nueva clase creada se llama subclase y la clase base (que sirve como un "molde" en esta creación) se denomina superclase.

La nueva clase tiene implícitamente los atributos y los métodos de la clase base y puede definir otras nuevas, incluyendo una clase constructora de llamada `__construct`. Si la subclase no tiene un método constructor se llama al método del constructor de la superclase cuando se instancia un objeto de la subclase.

Cuando el método constructor existe en la subclase, no hay ninguna llamada automática al método constructor de la superclase: si es necesario, se le debe llamar explícitamente utilizandoparent::__construct().

Ejemplo

```
<?php
// Definición de una clase base.
class usuario {
    // Definición de los atributos.
    public $apellido; // nombre del usuario
    public $nombre; // nombre del usuario
    // Definición de los métodos:
    // - método constructor
    public function __construct($nombre,$apellido) {
        // Inicializar el apellido y el nombre
        // con los valores parametrados.
        $this->nombre = $nombre;
        $this->apellido = $apellido;
    }
    // - método que da la información sobre el usuario
    public function información() {
        return "$this->nombre $this->apellido";
    }
}
// Definición de una clase que hereda de la primera.
class usuario_color extends usuario{
    // Definición de atributos complementarios.
    public $colores; // colores preferidos de usuarios
    // Definición de métodos complementarios.
    // - método constructor
    public function __construct($nombre,$colores) {
        // Llamada al constructor de la superclase
        // para la primera parte de la inicialización.
        parent::__construct ($nombre,'X');
        // Inicialización específica complementaria.
        $this->colores = explode(",",$colores);
    }
    // - lista de los colores preferidos del usuario
    public function colores() {
        return implode(',',$this->colores);
    }
}
// Instanciación de un objeto en la subclase.
$yo = new usuario_color('Olivier','azul,blanco,rojo');
// Utilización de métodos:
// - de la superclase
echo "{$yo->información()}<br />"; // existe por legado
// - de la subclase
echo "{$yo->colores()}<br />"; // existe en la clase
?>
```

Resultado

```
Olivier X
azul,blanco,rojo
```



En una subclase, es posible volver a declarar un método o un atributo existe en la superclase. En este caso, el operador parent:: se puede utilizar para hacer referencia explícitamente a

los atributos (`parent::atributo`) o métodos (`parent::método()`) de la superclase y eliminar la ambigüedad de los nombres (este operador se puede utilizar incluso si no hay ambigüedad).

5. Otras características de las clases

a. Clase o método abstracto

Una clase abstracta es una clase que no se puede instanciar (no se crea ningún objeto en la clase). Por contra, esta clase puede servir de base para definir una subclase que se puede instanciar (excepto si es a su vez abstracta).

Un método abstracto es un método que se define en una clase (que a su vez es necesariamente abstracta), pero no se ha implementado (el código del método no está presente). Este método se puede implementar en una subclase. Un método abstracto no puede ser privado.

En cuanto a la sintaxis, basta con poner la palabra clave `abstract` antes de la definición de la clase o del método.

Una subclase que no implementa todos los métodos abstractos de la superclase es implícitamente abstracta (aunque la palabra clave `abstract` no esté presente).

Ejemplo

```
<?php
// Definición de una clase abstracta.
abstract class superclase {
    // atributo protegido
    protected $x;
    // dos métodos para acceder al atributo protegido
    // - para leer
    public function get() {
        return "GET = $this->x";
    }
    // - para escribir
    // > método abstracto
    abstract public function put($valor);
}
// Definición de una subclase que hereda de la superclase.
class subclase extends superclase {
    // Implementación del método de escritura.
    public function put($valor) {
        $this->x = $valor;
    }
}
// Utilización de la subclase.
$objeto = new subclase();
$objeto->put(123);
echo $objeto->get(), '<br />';
?>
```

Resultado

GET = 123

b. Clase o método final

No se puede heredar de una clase final.

Un método final no puede ser redefinido en una subclase.

En cuanto a la sintaxis, basta con poner la palabra clave `final` antes de la definición de la clase o del método.

Ejemplo 1

```
<?php
// Definición de una superclase con un método final.
class superclase {
    final public function métodoFinal() {
        echo 'Método final en la superclase';
    }
}
// Definición de una subclase que hereda de la superclase.
class subclase extends superclase {
    // Intento de modificación de la clase final.
    public function métodoFinal() {
        echo 'Método final en la subclase';
    }
}
?>
```

Resultado

```
Fatal error: Cannot override final method
superclase::métodoFinal() in /app/scripts/index.php on line 14
```

Ejemplo 2

```
<?php
// Definición de una superclase final.
final class superclase {
    public $x;
}
// Intento de definición de una subclase que hereda
// de la superclase.
class subclase extends superclase {
    public $y;
}
?>
```

Resultado

```
Fatal error: Class subclase may not inherit from final class
(superclase) in /app/scripts/index.php on line 10
```

c. Interface

Una interfaz es una clase que sólo contiene las especificaciones de los métodos sin implementación. Una interfaz tampoco incluye ningún atributo.

Otras clases se pueden definir y, a continuación aplicar una o más interfaces, es decir, aplicar los métodos de una o más interfaces.

Sintaxis para la definición de una interfaz

```
interface nombre_interfaz {
    // Definición de los métodos.
    [public] function método();
    ...
}
```

nombre_interfaz Nombre de la interfaz (debe respetar las reglas de denominación presentes en el capítulo Información general sobre PHP).

método Especificación de un método de la interfaz.

Los métodos de una interfaz son siempre públicos, por lo que se puede omitir la palabra clave `public`.

Es posible definir una nueva clase que implemente una o varias interfaces con la palabra clave `implements`.

Sintaxis

```
class nombre_clase implements nombre_interfaz1,nombre_interfaz2,... {
    ...
}
```

Esta clase debe implementar los distintos métodos de las interfaces que implementa. Si no se implementa uno de los métodos de las interfaces, la clase debe declararse como abstracta.

Ejemplo

```
<?php
// Definición de dos interfaces.
interface lectura {
    function get();
}
interface escritura {
    function put($valor);
}

// Definición de una clase que implementa las dos interfaces.
class unaClase implements lectura,escritura {
    // Definición de cualquier atributo.
    private $x;
    // Implementación del método de lectura.
    public function get() {
        return $this->x;
    }
    // Implementación del método de escritura.
    public function put($valor) {
        $this->x = $valor;
    }
}
?>
```

d. Atributo o método estático - Constante de clase

Un atributo o un método estático se puede utilizar directamente sin necesidad de instanciar antes un

objeto. También conocido como atributo o método de clase.

Para definir un atributo o un método estático, basta con colocar la palabra clave `static` antes de la definición del atributo o del método.

Para hacer referencia a un atributo o un método estático, debe utilizar la sintaxis `nombre_clase::$nombre_atributo` o `nombre_clase::nombre_método()`.

Una constante de clase es una constante definida en una clase que se puede utilizar directamente sin necesidad de instanciar antes un objeto (como un atributo de clase, pero constante).

Sintaxis para definir una constante de clase

```
const nombre_constante = valor;
```

`nombre_constante` Nombre de la constante.

`valor` Valor de la constante.

Una constante de clase es implícitamente pública.

Para hacer referencia a una constante de clase, debe utilizar la sintaxis `nombre_clase::nombre_constante`.

Ejemplo

```
<?php
// Definición de una clase.
class unaClase {
    // Cualquier atributo privado.
    private $x;
    // Atributo privado estático para almacenar
    // el número de objetos instanciados.
    static private $número = 0;
    // Constante de clase para definir un valor predeterminado.
    const PREDETERMINADO = 'X';
    // Función pública estática que devuelve el número de objetos.
    static public function númeroObjetos() {
        return unaClase::$número;
    }
    // Método constructor
    // - recuperar el valor del atributo (valor predeterminado
    //   = la constante de clase)
    // - incrementar el número de objetos
    public function __construct($valor = unaClase::PREDETERMINADO) {
        $this->x = $valor;
        unaClase::$número++;
        echo "Creación del objeto: $this->x<br />";
    }
    // Método destructor.
    // - reducir el número de objetos
    public function __destruct() {
        unaClase::$número--;
        echo "Eliminación del objeto: $this->x<br />";
    }
}
// Crear dos objetos.
$desconocido = new unaClase();
$abc = new unaClase ('ABC');
// Mostrar el número de objetos
echo unaClase::númeroObjetos(), ' objeto(s)<br />';
```

```
// "Eliminar" un objeto.
unset($desconocido);
// Mostrar el número de objetos
echo unaClase::numeroObjetos(), ' objeto(s)<br />';
?>
```

Resultado

```
Creación del objeto: X
Creación del objeto: ABC
2 objeto(s)
Eliminación del objeto: X
1 objeto(s)
Eliminación del objeto: ABC
```

e. Trazos

Un trazo es un tipo de clase que reagrupa un conjunto de métodos o de atributos que pueden a continuación utilizarse en otras clases sin legado. Es un medio fácil de utilizar del código en un lenguaje como PHP que no autoriza el legado múltiple. Esta característica se introdujo en la versión 5.4.

Sintaxis de definición de un trazo

```
trazo nombre_trazo {
    // Definición de atributos y/o de métodos
    ...
}
```

nombre_trazo Nombre del trazo (debe respetar las reglas de nomenclatura presentadas en el capítulo Información general sobre PHP).

En la definición del trazo, los atributos y métodos se definen como en una clase normal.

Entonces, es posible definir una nueva clase que utilice uno o varios trazos con la palabra clave `use`.

Sintaxis

```
class nombre_clase [extiende el nombre de la clase de base] {
    // Utilización de uno o varios trazos
    use nombre_trazo [...];
    // Seguido de la definición de la clase.
    ...
}
```

Ejemplo

```
<?php
// Definición de un trazo que contiene métodos de cálculo.
trazo YoSéCalcular {
    function suma($a,$b) {
        return $a+$b;
    }
    function producto($a,$b) {
        return $a*$b;
    }
}
// Definición de un trazo que contiene un método que
```

```
// muestra un mensaje.
trazo YoSoyListo {
    // El método prueba si existe un atributo 'nombre' en
    // la clase y si es el caso lo utiliza en el mensaje.
    function decirHola() {
        if (isset($this->nombre)) {
            echo ";Hola {$this->nombre}!<br />";
        } else {
            echo ';Hola!<br />';
        }
    }
}

// Definición de una clase que utiliza los dos trazos.
class usuario{
    use YoSéCalcular,YoSoyListo;
    // Atributos y métodos de la clase.
    private $nombre; // nombre del usuario
    public function __construct($nombre) {
        // Inicializar el nombre con el valor pasado como parámetro.
        $this->nombre = $nombre;
        // Decir hola (llamada de un método de uno de los trazos).
        $this->decirHola();
    }
}

// Instanciar un nuevo objeto.
$yo = new usuario('Olivier');
// Hacer un cálculo (llamada de un método de uno de los trazos)
echo 'Yo sé calcular: ';
echo '10821 x 11409 = ', $yo->producto(10821,11409);
?>
```

Resultado

```
;Hola Olivier!
Yo sé calcular: 10821 x 11409 = 123456789
```

Un método heredado de una clase base se reemplaza por un método con el mismo nombre surgido de un trazo, que a su vez se reemplaza por un método del mismo nombre de la clase actual.

Por el contrario, si dos trazos insertan un método con el mismo nombre en una clase, se produce un error fatal, salvo si el conflicto se resuelve explícitamente utilizando los operadores `insteadof` o `as` (véase la documentación a este respecto).

Un trazo puede utilizar otros trazos.

6. Excepciones

Los lenguajes de programación basados en objetos como C++ y Java utilizan el concepto de excepción para administrar los errores. Esta característica se introdujo en la gestión de objetos en la versión PHP 5.

El principio consiste en incluir el código susceptible de provocar errores en un bloque `try` y asociarlo a un bloque `catch` destinado a detectar errores y procesarlos:

Estructura general

```
try {
    // código susceptible de generar errores
```

```

...
} catch (Exception $e) {
    // código destinado a procesar los errores
    ...
[
} finally {
    // código ejecutado en todos los casos
    ...
]
}

```

Desde la versión 5.5, es posible incluir un bloque `finally` después del bloque `catch`. El código presente en el bloque `finally` siempre se ejecuta después de los bloques `try` y `catch`, independientemente de si se produce una excepción o no.

La clase `Exception` es una clase que contiene los siguientes métodos:

<code>__construct</code>	Método constructor que acepta dos parámetros: un mensaje de error y un código de error (opcional).
<code>getMessage</code>	Método que permite recuperar el mensaje de error.
<code>getCode</code>	Método que permite recuperar el código de error.

Dentro del bloque `try`, se puede producir una excepción por una instrucción de tipo `throw new Exception(mensaje, [código])`. En el bloque `catch`, los métodos `getMessage` y `getCode` permiten recuperar información sobre el error y procesarla. En caso de excepción en un bloque `try`, el procesamiento se ramifica directamente en el bloque `catch`: el resto del bloque `try` no se ejecuta.

Ejemplo

```

<?php
// Definición de una clase.
class unaClase {
    // Cualquier atributo.
    private $x;
    // Método constructor.
    public function __construct($valor) {
        $this->x = $valor;
    }
    // Método que lleva a cabo cualquier acción.
    public function acción() {
        // Por alguna razón, se prohíbe la acción
        // si el atributo es negativo: se produce una excepción.
        if ($this->x < 0) {
            throw new Exception('Acción prohibida',123);
        }
    }
}
// Crear dos objetos.
$objeto = new unaClase(1);
try {
    echo 'Objeto 1: ';
    $objeto->acción(); // no va a provocar ninguna excepción
    echo 'OK<br />';
} catch (Exception $e) {
    echo 'ERROR ', $e->getCode(), ' - ', $e->getMessage(), '<br />';
}

```

```
$objeto = new unaClase(-1);
try {
    echo 'Objeto 2: ';
    $objeto->acción(); // va a provocar una excepción
    echo 'OK<br />';
} catch (Exception $e) {
    echo 'ERROR ', $e->getCode(), ' - ', $e->getMessage(), '<br />';
}
echo '--<br />';
// Lo mismo con un bloque finally.
$objeto = new unaClase(1);
try {
    echo 'Objeto 1: ';
    $objeto->action(); // no se producirá una excepción
    echo 'OK<br />';
} catch (Exception $e) {
    echo 'ERROR ', $e->getCode(), ' - ', $e->getMessage(), '<br />';
} finally {
    echo 'FINALLY<br />';
}
$objeto = new unaClase(-1);
try {
    echo 'Objeto 2: ';
    $objeto->action(); // producirá una excepción
    echo 'OK<br />';
} catch (Exception $e) {
    echo 'ERROR ', $e->getCode(), ' - ', $e->getMessage(), '<br />';
} finally {
    echo 'FINALLY<br />';
}
?>
```

Resultado

```
Objeto 1: OK
Objeto 2: ERROR 123 - Acción prohibida
--
Objeto 1: OK
FINALLY
Objeto 2: ERROR 123 - Acción prohibida
FINALLY
```


Índice

Información

[Título, autor...](#)

Introducción

[Objetivo del libro](#)[Breve historia de PHP](#)[¿Dónde conseguir PHP?](#)[Convenciones de escritura](#)

Información general sobre PHP

[Variables, constantes, tipos y matrices](#)[Operadores](#)[Estructuras de control](#)[Funciones y clases](#)[Gestión de formularios](#)[Acceder a las bases de datos](#)[Administrar las sesiones](#)[Enviar un correo electrónico](#)[Gestión de archivos](#)[Administrar los errores en un script PHP](#)[Anexo](#)

Breve historia de PHP

El lenguaje PHP (*Personal Home Page* históricamente, oficialmente acrónimo recursivo de PHP: *Hypertext Preprocessor*) fue diseñado en 1994 por Rasmus Lerdorf para sus necesidades personales antes de su lanzamiento a principios de 1995.

En 1995 se publicó una nueva versión completamente reescrita bajo el nombre de PHP/FI versión 2. Esta versión, capaz de manejar formularios y de acceder a la base de datos mSQL, permite al lenguaje crecer rápidamente.

En 1997, el desarrollo del lenguaje recae en un equipo liderado por Rasmus Lerdorf y conduce al lanzamiento de la versión 3.

En 2000, el analizador PHP se migra al motor de análisis de Zend para proporcionar un mejor rendimiento y admitir un mayor número de extensiones: se trata de la versión 4 de PHP.

En 2004 nace la versión 5. Esta nueva versión, basada en la versión del motor Zend 2, aporta varias características nuevas, la mayoría de ellas relacionadas con el desarrollo orientado a objetos.

A día de hoy, los analistas creen que más del 80% de los sitios Web utilizan PHP en el mundo (en número de dominios).

[Subir](#)

Los espacios de nombres

Los espacios de nombres (*namespace* en inglés) permiten resolver dos problemas frecuentes en el uso de clases o de bibliotecas de funciones:

- Uso de un mismo nombre (clase, función, constante) en dos bibliotecas.
- Manipulación de nombres especialmente largos que hacen que el código sea difícil de escribir.

Este concepto se añadió en la versión 5.3.

Un espacio de nombres se declara con la palabra clave `namespace` al principio de un archivo (de lo contrario, se produce un error grave).

Ejemplo

```
<?php
// Definición del espacio de nombres.
namespace MiBiblioteca;
// Definición de una constante.
const UNO = 1;
// Definición de una clase.
class unaClase {
    /*
    ...
    */
}
// Definición de una función.
function unaFunción() {
    /*
    ...
    */
}
?>
```

El mismo espacio de nombres se puede definir en varios archivos, lo que permite organizar el código en varios archivos, agrupándolo dentro del mismo espacio de nombres.



Es posible definir varios espacios de nombres en un mismo archivo, pero no es una buena práctica de codificación.

Un espacio de nombres se puede definir con subniveles, utilizando el separador de la barra invertida (`\`): `MiBiblioteca\Sub\Nivel`.

De forma predeterminada, si no se define un espacio de nombres, todas las definiciones (clases, funciones, constantes) se colocan en el espacio de nombres global.

La constante `__NAMESPACE__` da el nombre del espacio de nombres actual (cadena vacía en el espacio global).

El espacio de nombres se puede utilizar para cualificar un identificador (constante, función, clase) con el separador de barra invertida (`\`) y así precisar su origen. El nombre cualificado es relativo al espacio de nombres actual si no comienza con una barra invertida; de lo contrario, es absoluto.

Si el nombre no está cualificado, está implícitamente resuelto en el espacio de nombres actual (o en el espacio de nombres global si no se ha definido un espacio de nombres).

La palabra clave `namespace` se puede utilizar para hacer referencia explícitamente a un elemento del

espacio de nombres actual, o a un subespacio.

Suponiendo que el espacio de nombres actual es `MiProyecto`, tenemos las siguientes resoluciones:

Referencia	Se convierte en
<code>f()</code>	<code>\MiProyecto\f()</code> (nombre no cualificado = espacio de nombres actual)
<code>Biblioteca\f()</code>	<code>\MiProyecto\Biblioteca\f()</code> (nombre relativo)
<code>\MiProyecto\Biblioteca\f()</code>	<code>\MiProyecto\Biblioteca\f()</code> (nombre absoluto)
<code>namespace\f()</code>	<code>\MiProyecto\f()</code> (utilización de la palabra clave <code>namespace</code>)
<code>namespace\Biblioteca\f()</code>	<code>\MiProyecto\Biblioteca\f()</code> (utilización de la palabra clave <code>namespace</code>)

Para hacer referencia a un identificador del espacio global, se puede utilizar un nombre absoluto `\nombre` (por ejemplo, `\time()`).

Para facilitar el uso de espacios de nombres en el código, es posible importar un espacio de nombres o hacer referencia a un nombre absoluto con un alias. El alias de nombre y la importación sólo son posibles para las clases y el espacio de nombres, no para las funciones ni para las constantes.

El alias se crea con el operador `use`.

Sintaxis

```
use nombre [as alias][, ...]
```

nombre Nombre cualificado absoluto del espacio de nombres o de la clase que se debe importar. La barra invertida inicial del nombre absoluto no es necesario.

alias Nombre del alias. Igual al nombre "corto" de nombre (sin la ruta) si no se especifica.

El nombre importado debe ser absoluto y no se resuelve desde el espacio de nombres actual.

Ejemplos

```
use MiProyecto\Biblioteca as lib; // alias de un espacio de nombres
lib\unaFunción(); // llama a MiProyecto\Biblioteca\unaFunción()
use MiProyecto\Biblioteca\unaClase as cl; // alias de una clase
$objeto = new cl; // instanciación de un objeto de la clase
                // MiProyecto\Biblioteca\unaClase
use MiProyecto\Biblioteca; // equivalente a
                // use MiProyecto\Biblioteca as Biblioteca
```

Para ilustrar los diferentes conceptos presentados anteriormente, vamos a considerar el uso de la biblioteca `biblioteca.inc` siguiente:

```
<?php
// Declaración del espacio de nombres.
namespace MiProyecto\Biblioteca;
// Definición de una constante.
```

```

const UNO = 1;
// Definición de una clase.
class unaClase {
    static function información() {
        echo 'Biblioteca<br />';
    }
}
// Definición de una función.
function unaFunción() {
    echo __FUNCIÓN__, '<br />';
}
?>

```

Esta biblioteca se utiliza en el siguiente script:

```

<?php
// Definición del espacio de nombres.
namespace MiProyecto;
// Inclusión de la biblioteca.
include('biblioteca.inc');
// Definición de una constante.
const UNO = 'uno';
// Definición de una clase.
class unaClase {
    static function información() {
        echo 'MiProyecto<br />';
    }
}
// Definición de una función.
function unaFunción() {
    echo __FUNCIÓN__, '<br />';
}
// Visualización del espacio de nombres actual.
echo 'Espacio de nombres actual = ', __NAMESPACE__, '<br />';
// Llamada de unaFunción():
// nombre no cualificado = espacio de nombres actual
echo 'unaFunción() = ';
unaFunción();
// Llamada de Biblioteca\unaFunción():
// nombre cualificado relativo
echo 'Biblioteca\unaFunción() = ';
Biblioteca\unaFunción();
// Visualización de \MiProyecto\Biblioteca\UNO:
// nombre cualificado absoluto
echo '\MiProyecto\Biblioteca\UNO = ',
    \MiProyecto\Biblioteca\UNO,
    '<br />';
// Visualización de namespace\UNO:
// utilización de la palabra clave 'namespace' (espacio actual)
echo 'namespace\UNO = ',
    namespace\UNO,
    '<br />';
// Definición de un alias de clase.
use \MiProyecto\Biblioteca\unaClase as cl;
echo 'cl::información() = ';
cl::información();
// Definición de un alias de espacio de nombres.
use MiProyecto\Biblioteca as lib;
echo ' lib\unaFunción() = ';
lib\unaFunción();

```

?>

Resultado

```
Espacio de nombres actual = MiProyecto
unaFunción() = MiProyecto\unaFunción
Biblioteca\unaFunción() = MiProyecto\Biblioteca\unaFunción
\MiProyecto\Biblioteca\UNO = 1
namespace\UNO = uno
cl::información() = Biblioteca
lib\unaFunción() = MiProyecto\Biblioteca\unaFunción
```

Información general

1. Rápido recordatorio sobre los formularios

El formulario es un instrumento básico necesario para los sitios Web dinámicos, ya que permite al usuario introducir información y, por lo tanto, interactuar con el sitio.

Un formulario HTML se define entre las etiquetas `<form>` y `</form>`.

Sintaxis simplificada

```
<form
  [ action="url_de_procesamiento" ]
  [ method="GET"|"POST" ]
  [ id="identificador_formulario" ]
  [ target="destino" ]
>
...
</form>
```

Los atributos de la etiqueta `<form>` son los siguientes:

<code>action</code>	URL relativa o absoluta (<i>Uniform Resource Locator</i>) que procesará el formulario, en nuestro caso, un script PHP. Este atributo es obligatorio para cumplir con la estricta recomendación XHTML.
<code>method</code>	Modo de transmisión al servidor de la información introducida en el formulario. GET (valor predeterminado): los datos del formulario se transmiten en la URL. POST: los datos del formulario se transmiten en el cuerpo de la consulta.
<code>id</code>	Identificador del formulario. Si la página HTML contiene varios formularios, el identificador permite diferenciarlos. En nuestro caso, este identificador no tiene ningún valor porque no se recupera en el script de procesamiento del formulario. Por contra, se puede utilizar del lado del cliente, en JavaScript, por ejemplo.
<code>target</code>	Destino (por ejemplo, otra ventana) en el que se abrirá la dirección URL de destino.

Entre las etiquetas `<form>` y `</form>`, es posible colocar etiquetas `<input>`, `<select>` o `<textarea>` para definir los campos de entrada de datos.

Ejemplo (formulario de HTML completo)

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Entrada de datos</title>
  </head>
  <body>
    <form action="" method="post">
      <div>
        Nombre:
        <input type="text" name="nom" value=""
```

```

        size="20" maxlength="20" />
Contraseña:
<input type="password" name="contraseña" value=""
        size="20" maxlength="20" />
<br />Sexo:
<input type="radio" name="sexo" value="M" />Masculino
<input type="radio" name="sexo" value="F" />Feminino
<input type="radio" name="sexo" value="?"
        checked="checked" />No lo sé
<br />Foto:
<input type="file" name="foto" value="" size="50" />
<br />Colores favoritos:
<input type="checkbox" name="colores[azul]" />Azul
<input type="checkbox" name="colores[blanco]" />Blanco
<input type="checkbox" name="colores[rojo]" />Rojo
<input type="checkbox" name="colores[ninguno]"
        checked="checked" />No lo sé
<br />Idioma:
<select name="idioma">
    <option value="E">Español</option>
    <option value="F" selected="selected" >Francés</option>
    <option value="I">Italiano</option>
</select>
<br />Frutas favoritas:<br />
<select name="fruta[]" multiple="multiple" size="8">
    <option value="A">Albaricoques</option>
    <option value="C">Cerezas</option>
    <option value="F">Fresas</option>
    <option value="M">Melocotones</option>
    <option value="?" selected="selected">
        No lo sé</option>
</select>
<br />Comentarios:<br />
<textarea name="comentarios" rows="4" cols="50"></textarea>
<br />
<input type="hidden" name="invisible" value="123" /><br />
<input type="submit" name="enviar" value="OK" />
<input type="image" name="validar" src="validar.gif" />
<input type="reset" name="borrar" value="Borrar" />
<input type="button" name="acción" value="No hacer nada" />
</div>
</form>
</body>
</html>

```

Resultado

Nombre: Contraseña:

Sexo: ☐ Masculino ☐ Feminino ☒ No lo sé

Foto:

Colores favoritos: ☐ Azul ☐ Blanco ☐ Rojo ☒ No lo sé

Idioma: ▼

Frutas favoritas:

- Albaricoques
- Cerezas
- Fresas
- Melocotones
- No lo sé**

Comentarios:

☒

2. Interacción entre un formulario y un script PHP

PHP puede intervenir en dos lugares con respecto al formulario:

- Para la construcción del formulario, si éste debe incluir información dinámica.
- Para el procesamiento del formulario (es decir, los datos introducidos por el usuario en el formulario).

Hay tres métodos principales que se pueden utilizar para interactuar con un formulario y un script PHP:

- Colocar el formulario en un documento HTML "puro" (.htm o .html), el formulario no contiene ningún elemento dinámico e indicar el nombre del script PHP que debe procesar el formulario en el atributo `action` de la etiqueta `<form>`.
- Colocar el formulario en un script PHP (por ejemplo, para construir una parte del formulario de forma dinámica) y hacer procesar el formulario por otro script PHP (mencionado en el atributo `action` de la etiqueta `<form>`).
- Colocar el formulario en un script PHP (por ejemplo, para construir una parte del formulario de forma dinámica) y hacerlo procesar por el mismo script PHP (mencionado en el atributo `action` de la etiqueta `<form>` o llamado por defecto si este atributo no está presente).

Además, también se puede insertar cualquier parte en otra página, un enlace (Entrada de datos, por ejemplo) para llamar al formulario de entrada de datos:

- Formulario HTML:

```
<a href="entrada.htm">Entrada de datos</a>
```


- Formulario PHP:

`Entrada de datos`

Primer método

Documento HTML entrada.htm

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head><title>Entrada</title></head>
  <body>
    <form action="procesamiento.php" method="post">
      <div>
        Nombre: <input type="text" name="nombre" value="" />
        <input type="submit" name="ok" value="OK" />
      </div>
    </form>
  </body>
</html>
```

Script PHP procesamiento.php

```
<?php
/* Por hacer...
   - recuperar la información introducida
   - realizar el procesamiento
   - mostrar una nueva página
*/
?>
```

Resultado

- Presentación inicial del formulario:



- Entrada de información:



- El resultado al hacer clic en el botón **OK** es una página en blanco, porque por ahora, el script de procesamiento no hace nada.

Segundo método

Documento PHP entrada.php

Un poco de código PHP (en negrita) se utiliza para generar una parte dinámica del formulario.

```
<?php
// Incluir un archivo que contiene definiciones de
// constantes, incluido el título de página (TITULO_PAGINA_ENTRADA).
```

```

require('constantes.inc');
// Inicialización de una variable que contiene el valor
// inicial del campo de entrada de datos (en la práctica este
// valor proviene sin duda de otro lugar y no está codificado de forma
// rígida).
$nombre = 'X';
// En el código HTML siguiente, inclusión de dos pequeñas
// porciones de código PHP para mostrar respectivamente el título
// de la página y el valor inicial del campo de entrada de datos.
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title><?php echo TITULO_PAGINA_ENTRADA; ?></title>
</head>
<body>
<form action="procesamiento.php" method="post">
<div>
Nombre: <input type="text" name="nombre"
value="<?php echo $nombre; ?>" />
<input type="submit" name="ok" value="OK" />
</div>
</form>
</body>
</html>

```

Script PHP procesamiento.php

```

<?php
/* Por hacer...
- recuperar la información introducida
- realizar el procesamiento
- mostrar una nueva página
*/
?>

```

Resultado

- Presentación inicial del formulario (se propone un valor inicial dinámico para el área de entrada):



- Entrada de información:



- El resultado al hacer clic en el botón **OK** es una página en blanco, porque por ahora, el script de procesamiento no hace nada.

Tercer método

Documento PHP entrada.php

Es el mismo script que el anterior, simplemente cambia el atributo `action` de la etiqueta `<form>` para indicar que el formulario debe ser procesado por el mismo `entrada.php`.

```
<?php
// Incluir un archivo que contiene definiciones de
// constantes, incluido el título de página (TITULO_PAGINA_ENTRADA).
require('constantes.inc');
// Inicialización de una variable que contiene el valor
// inicial del campo de entrada de datos (en la práctica este
// valor proviene sin duda de otro lugar y no está codificado de forma
// rígida).
$nombre = 'X';
// En el código HTML siguiente, inclusión de dos pequeñas
// porciones de código PHP para mostrar respectivamente el título
// de la página y el valor inicial del campo de entrada de datos.
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title><?php echo TITULO_PAGINA_ENTRADA; ?></title>
</head>
<body>
<form action="entrada.php" method="post">
<div>
Nombre: <input type="text" name="nombre"
value="<?php echo $nombre; ?>" />
<input type="submit" name="ok" value="OK" />
</div>
</form>
</body>
</html>
```

Resultado

- Presentación inicial del formulario (se propone un valor inicial dinámico para el área de entrada):



- Entrada de información:



- El resultado al hacer clic en el botón **OK** es la misma página mostrada de nuevo, porque por ahora, el script de procesamiento no hace nada más:



¿Qué método elegir?

La elección de un método en particular depende de la complejidad del sitio y de las preferencias de cada uno.

Algunas consideraciones generales:

- Separar la página HTML (o el script PHP que genera el formulario) del script PHP tiene una desventaja en términos de mantenimiento: si se realizan cambios en el formulario, hay dos archivos que modificar (con los consiguientes riesgos de error, olvido...).
- Por el contrario, si el formulario no tiene ninguna parte dinámica, escribirlo en un archivo HTML separado del script PHP que lo procesa permite separar la interfaz de usuario (la capa de "presentación") del procesamiento.
- En la práctica, para facilitar el mantenimiento, es conveniente definir ciertos valores presentados en varias ocasiones (nombre de la empresa, por ejemplo) en las constantes o variables y utilizar estas constantes y variables en las páginas: todas las páginas se vuelven en un poco dinámicas y el tercer método parece ser el mejor.

En el resto de este capítulo, vamos a entrar en los detalles del procesamiento de formularios en PHP, utilizando ejemplos contruidos sobre el modelo del tercer método.

Recuperar los datos introducidos en el formulario

1. Principio

A diferencia de los scripts CGI, no hay necesidad de realizar análisis complejos de cadenas de caracteres ("parser") para recuperar los valores introducidos por el usuario; estos valores se recuperan con facilidad en el script de procesamiento.

Por defecto, todos los campos del formulario se almacenan automáticamente en el script PHP que procesa el formulario, en una matriz asociativa `$_POST` para los formularios POST y `$_GET` para los formularios GET: la clave de la matriz es igual al nombre del campo en el formulario (atributo `name` de la etiqueta `<input>`, `<select>` o `<textarea>`) y el valor igual al valor introducido en el campo. Esta información también está disponible en la matriz asociativa `$_REQUEST` que agrupa el contenido de las matrices `$_GET` y `$_POST` (y las veremos más adelante de la matriz `$_COOKIE` que contiene información sobre las cookies).

Ejemplo

Documento HTML entrada.htm

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head><title>Entrada</title></head>
  <body>
    <form action="procesamiento.php" method="post">
      <div>
        Nombre: <input type="text" name="nombre" value="" />
        <input type="submit" name="ok" value="OK" />
      </div>
    </form>
  </body>
</html>
```

Script PHP procesamiento.php

```
<?php
// Visualización de la información contenida en las
// matrices $_POST y $_REQUEST.
echo '$_POST[\'nombre\'] -> ', $_POST['nombre'], '<br \>';
echo '$_REQUEST[\'nombre\'] -> ', $_REQUEST['nombre'], '<br \>';
?>
```

Resultado

- Presentación inicial del formulario



- Entrada de un valor



- Resultado al hacer clic en el botón **OK**

```
$_POST['nombre'] -> Olivier
$_REQUEST['nombre'] -> Olivier
```

- Las matrices `$_POST`, `$_GET` y `$_REQUEST` son matrices superglobales. Están disponibles en todos los contextos de ejecución.
- En el capítulo Administrar las sesiones, después de haber visto otras matrices similares, haremos un breve resumen sobre las variables GPCS (Get/Post/Cookie/Session).

El script siguiente presenta una forma estándar de recuperar los valores introducidos en un formulario utilizando ambos métodos.

```
<?php
// Recuperación en variables PHP de los valores
// introducidos en el formulario utilizando una
// matriz $_* ($_POST aquí).
$form_apellido = (isset($_POST['apellido']))?$_POST['apellido']:'';
$form_nombre = (isset($_POST['nombre']))?$_POST['nombre']:'';
// Etc. para cada campo.
// A continuación, se utilizan las variables en los procesamientos
// y, posiblemente, al volver a mostrar el formulario.
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head><title>Entrada</title></head>
  <body>
    <form action="entrada.php" method="post">
      <div>
        Apellidos: <input type="text" name="apellido"
          value="<?php echo $form_apellido ?>" /><br/>
        Nombre: <input type="text" name="nombre"
          value="<?php echo $form_nombre ?>" /><br/>
        <input type="submit" name="ok" value="OK" />
      </div>
    </form>
  </body>
</html>
```

- Antes de la versión 5.4, la información introducida en el formulario podía importarse automáticamente en variables PHP gracias a la función `import_request_variables`. Esto ya no es posible desde la versión 5.4 y esta función se ha eliminado. Del mismo modo, al poner la directiva de configuración `register_globals` en `on`, todos los campos del formulario podían registrarse en variables en el script de procesamiento del formulario. Esto ya no es posible desde la versión 5.4 y la directiva se ha eliminado.

¿Qué sucede si dos campos comparten el mismo nombre?

Simplemente, el último campo encontrado en el formulario es el que determina el valor.

Ejemplo

```
<form action="entrada.php" method="POST"><div>
Apellido: <input type="text" name="nombre"><br />
Nombre: <input type="text" name="nombre"><br />
<input type="submit" name="ok" value="OK">
</div></form>
```

El texto HEURTEL en la primera zona y Olivier en la segunda, dan una sola variable igual aOlivier en la matriz \$_POST.

¿Qué ocurre si hay dos formularios en la página HTML?

Las variables sólo se crean y se rellenan en el formulario que ha sido validado.

Ejemplo

```
<form action="entrada.php" method="POST"><div>
Nombre 1: <input type="text" name="nombre1"><br />
<input type="submit" name="ok1" value="OK1">
</div></form>
<form action="entrada.php" method="POST"><div>
Nombre 2: <input type="text" name="nombre2"><br />
<input type="submit" name="ok2" value="OK2">
</div></form>
```

Si el usuario valida el primer formulario, la variable nombre1 estará disponible. Si el usuario valida el segundo formulario, la variable nombre2 estará disponible.

Utilización de una matriz para recuperar los campos rellenos

Es posible utilizar una notación de tipo matriz en el atributo name de las etiquetas <input>, <select> y <textarea>.

Ejemplo

```
<form action="entrada.php" method="POST"><div>
Apellido: <input type="text" name="entrada[]"><br />
Nombre: <input type="text" name="entrada[]"><br />
<input type="submit" name="ok" value="OK">
</div></form>
```

El texto HEURTEL en el primer campo y Olivier en el segundo, dan una sola variable \$entrada, de tipo de matriz, que contiene las líneas siguientes:

Clave	Valor
0	HEURTEL
1	Olivier

PHP completa la matriz, añadiendo una línea para cada campo, con un índice entero consecutivo comenzando en 0 (como en la notación [] estudiada en el capítulo Variables, constantes, tipos y matrices).

Esta técnica es interesante, pero en el código, debe saber que el índice 0 corresponde al apellido y el índice 1 al nombre. Por otra parte, puede surgir un problema si el orden de los campos cambia.

Para mejorar esta técnica, es posible establecer la clave bien con un número o con una cadena de caracteres.

Ejemplo

```
<form action="entrada.php" method="POST"><div>
Apellido: <input type="text" name="entrada[1]"><br />
Nombre: <input type="text" name="entrada[2]"><br />
<input type="submit" name="ok" value="OK">
</div></form>
```

El texto HEURTEL en el primer campo y Olivier en el segundo, dan el siguiente resultado en la matriz \$entrada:

Clave	Valor
1	HEURTEL
2	Olivier

Ejemplo

```
<form action="entrada.php" method="POST"><div>
Apellido: <input type="text" name="entrada[apellido]"><br />
Nombre: <input type="text" name="entrada[nombre]"><br />
<input type="submit" name="ok" value="OK">
</div></form>
```

El texto HEURTEL en el primer campo y Olivier en el segundo, dan el siguiente resultado en la matriz \$entrada:

Clave	Valor
nombre	HEURTEL
nombre	Olivier

Más adelante veremos otras situaciones donde se requiere este tipo de notación.

2. Los diferentes tipos de campos

Tomamos nuestro formulario completo de salida y vemos la información recuperada en el script PHP.

Script entrada.htm

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Entrada</title>
</head>
<body>
<form action="entrada.php" method="post">
<div>
Nombre:
<input type="text" name="nombre" value=""
size="20" maxlength="20" />
Contraseña:
<input type="password" name="contraseña" value=""
size="20" maxlength="20" />
```



```

<br />Sexo:
<input type="radio" name="sexo" value="M" />Masculino
<input type="radio" name="sexo" value="F" />Femenino
<input type="radio" name="sexo" value="?"
        checked="checked" />No lo sé

<br />Foto:
<input type="file" name="foto" value="" size="50" />
<br />Colores favoritos:
<input type="checkbox" name="colores[azul]" />Azul
<input type="checkbox" name="colores[blanco]" />Blanco
<input type="checkbox" name="colores[rojo]" />Rojo
<input type="checkbox" name="colores[ninguno]"
        checked="checked" />No lo sé

<br />Idioma:
<select name="idioma">
    <option value="E">Español</option>
    <option value="F" selected="selected" >Francés</option>
    <option value="I">Italiano</option>
</select>
<br />Fruta favorita:<br />
<select name="fruta[]" multiple="multiple" size="8">
    <option value="A">Albaricoques</option>
    <option value="C">Cerezas</option>
    <option value="F">Fresas</option>
    <option value="M">Melocotones</option>
    <option value="?" selected="selected">
        No lo sé</option>
</select>
<br />Comentarios:<br />
<textarea name="comentarios" rows="4" cols="50"></textarea>
<br />
<input type="hidden" name="invisible" value="123" /><br />
<input type="submit" name="enviar" value="OK" />
<input type="image" name="validar" src="validar.gif" />
<input type="reset" name="borrar" value="Borrar" />
<input type="button" name="action" value="No hacer nada" />
</div>
</form>
</body>
</html>

```

Script entrada.php

```

<?php
// Inclusión de un archivo que contiene funciones genéricas
// (incluida la función mostrar_matriz definida en el
// capítulo Funciones y clases)
include('funciones.inc') ;
mostrar_matriz($_POST,'$_POST :');
?>

```

Resultado

- Presentación inicial y entrada de los diferentes valores:

Nombre: HEURTEL Contraseña: ●●●●●●

Sexo: ☒ Masculino ☐ Feminino ☐ No lo sé

Foto: C:\Fotos\identidad.jpg

Colores favoritos: ☒ Azul ☐ Blanco ☒ Rojo ☒ No lo sé

Idioma: Francés ▼

Frutas favoritas:

- Albaricoques
- Cerezas
- Fresas
- Melocotones
- No lo sé

Comentarios:

Consultor en sistemas informáticos

- Resultado al hacer clic en el botón **OK** (visualización del contenido de \$_POST):

\$_POST:

```

apellido = HEURTEL
contraseña = olivier
sexo = M
foto = identidad.jpg
colores =
    azul = on
    rojo = on
idioma = F
fruta =
    0 = A
    1 = F
comentarios = Consultor en sistemas informáticos
invisible = 123
enviar = OK

```

Basándonos en este ejemplo, vamos a ofrecer algunas explicaciones.

Campo que contiene texto

Para los campos que contienen texto, es decir los campos `<input>` de tipo `text` (apellido), `password` (contraseña), `file` (foto) y `hidden` (invisible), así que para el campo `<textarea>` (comentarios), las variables asociadas contienen el texto introducido.

Ejemplo

\$_POST:

```

apellido = HEURTEL

```

```

contraseña = olivier
foto = identidad.jpg
comentario = Consultor en sistemas informáticos
invisible = 123

```

- Por ahora, con el campo de tipo file, hemos recuperado sólo el nombre del archivo, pero no el archivo en sí (véase el capítulo Gestión de archivos).

Grupo de botones de opción

Para tener un grupo de botones de opción, los campos deben tener el mismo nombre.

Para un grupo de botones de opción, es decir, para los campos `<input>` de tipo `radio`, la variable asociada contiene el valor contenido en el atributo `value` de la etiqueta `input` del botón seleccionado. Si el atributo `value` está ausente, el valor por defecto es `on`, lo cual es molesto ya que es imposible conocer la opción seleccionada. En la práctica, debemos rellenar el atributo `value`.

Ejemplo

```

<br />Sexo:
<input type="radio" name="sexo" value="M" /> Masculino
<input type="radio" name="sexo" value="F" /> Femenino
<input type="radio" name="sexo" value="?"
      checked="checked" /> No lo sé

```

Opción seleccionada

Sexo: ☒ Masculino ☐ Femenino ☐ No lo sé

Resultado

```

$_POST:
sexo = M

```

Casilla de verificación

Para las casillas de verificación, es decir, para los campos `<input>` de tipo `checkbox`, la variable asociada contiene el valor contenido en el atributo `value` de la etiqueta `input`; si el atributo `value` está ausente, el valor predeterminado es `on`. En ambos casos, la variable asociada se define sólo para las casillas de verificación.

Ejemplo

```

<br />Colores favoritos:
<input type="checkbox" name="azul" value="b"/> Azul
<input type="checkbox" name="blanco" /> Blanco
<input type="checkbox" name="rojo" /> Rojo
<input type="checkbox" name="nolosé"
      checked="checked" /> No lo sé

```

Opciones seleccionadas

Colores favoritos:	<input checked="" type="checkbox"/> Azul	<input type="checkbox"/> Blanco	<input checked="" type="checkbox"/> Rojo	<input checked="" type="checkbox"/> No lo sé
--------------------	--	---------------------------------	--	--

Resultado

\$_POST:

azul = b

rojo = on

Con la casilla de verificación, el atributo `value` es menos importante por lo general, ya que es posible determinar si una casilla está marcada sólo por el hecho de que la variable asociada tiene un valor (independientemente de este valor). Por contra, es importante que cada casilla tenga un nombre diferente.

Varios enfoques son posibles en relación al valor del atributo `value`:

- El atributo `value` almacena el valor deseado al nivel de la lógica de la aplicación en caso de que la casilla esté marcada (1, sí...), sabiendo que, si la casilla no está marcada, la variable no existe.
- El atributo `value` se omite y el código interpreta la existencia de la variable de acuerdo a las necesidades de la lógica de la aplicación.

En el código, si desea recuperar el hecho de que una casilla esté marcada en forma de un booleano, puede escribir una instrucción como la siguiente:

```
$casilla_marcada = isset($_POST['nombre_casilla'])?TRUE:FALSE;
```

Si lo desea, puede utilizar una matriz para no tener una variable por casilla de verificación. Para poder determinar cuáles son las casillas marcadas, existen dos opciones:

- Utilizar una matriz sin índice, pero rellenar el atributo `value`:

```
<br />Colores favoritos:
<input type="checkbox" name="colores[]" value="azul"/> Azul
<input type="checkbox" name="colores[]" value="blanco" /> Blanco
<input type="checkbox" name="colores[]" value="rojo" /> Rojo
<input type="checkbox" name="colores[]" value="nolosé"
checked="checked" />No lo sé
```

- No rellenar el atributo `value`, pero definir los índices o las claves en la matriz:

```
// Índices numéricos
<br />Colores favoritos:
<input type="checkbox" name="colores[1]"/> Azul
<input type="checkbox" name="colores[2]" /> Blanco
<input type="checkbox" name="colores[3]" /> Rojo
<input type="checkbox" name="colores[4]"
checked="checked" />No lo sé

// claves alfanuméricas
<br />Colores favoritos:
<input type="checkbox" name="colores[azul]"/> Azul
<input type="checkbox" name="colores[blanco]" /> Blanco
<input type="checkbox" name="colores[rojo]" /> Rojo
<input type="checkbox" name="colores[nolosé]"
checked="checked" />No lo sé
```

Con este último ejemplo, si las casillas "Azul" y "Rojo" están marcadas, la matriz `color` contendrá las

siguientes líneas:

```
$_POST:
colores =
  azul = on
  rojo = on
```

Si se utiliza una matriz para todo el formulario, puede utilizar soluciones diferentes, incluyendo:

- Utilizar una matriz sin índice, pero rellenar el atributo `value`:

```
<br />Colores favoritos:
<input type="checkbox" name="entrada[colores][]" value="azul"/> Azul
<input type="checkbox" name="entrada[colores][]" value="blanco" /> Blanco
<input type="checkbox" name="entrada[colores][]" value="rojo" /> Rojo
<input type="checkbox" name="entrada[colores][]" value="nolosé"
      checked="checked" />No lo sé
```

- No rellenar el atributo `value`, pero definir los índices o las claves en la matriz:

```
// Índices numéricos
<br />Colores favoritos:
<input type="checkbox" name="entrada[colores][1]"/> Azul
<input type="checkbox" name="entrada[colores][2]" /> Blanco
<input type="checkbox" name="entrada[colores][3]" /> Rojo
<input type="checkbox" name="entrada[colores][4]"
      checked="checked" />No lo sé

// claves alfanuméricas
<br />Colores favoritos:
<input type="checkbox" name="entrada[colores][azul]"/> Azul
<input type="checkbox" name="entrada[colores][blanco]" /> Blanco
<input type="checkbox" name="entrada[colores][rojo]" /> Rojo
<input type="checkbox" name="entrada[colores][nolosé]"
      checked="checked" />No lo sé
```

Con este último ejemplo, si las casillas "Azul" y "Rojo" están marcadas, la matriz `color` contendrá las siguientes líneas:

```
$_POST:
entrada =
  colores =
    azul = on
    rojo = on
```

Lista de selección única

Para las listas de selección única, es decir, para un campo `<select>` sin atributo `multiple` (idioma en el siguiente ejemplo), la variable asociada contiene el valor contenido en el atributo `value` de la etiqueta `<option>`, o, si no hay atributo `value`, el valor mostrado en la lista (es decir, detrás de la etiqueta `<option>`).

Ejemplo (con atributo value)

```
<br />Idioma:
<select name="idioma">
  <option value="E">Español</option>
  <option value="F" selected="selected" >Francés</option>
```

```
<option value="I">Italiano</option>
</select>
```

Opción seleccionada



Resultado

```
$_POST:
idioma = F
```

Ejemplo (sin atributo value)

```
<br />Idioma:
<select name="idioma">
  <option>Español</option>
  <option selected="selected" >Francés</option>
  <option>Italiano</option>
</select>
```

Opción seleccionada



Resultado

```
$_POST:
idioma = Francés
```

Se puede elegir una u otra posibilidad de acuerdo a las necesidades de la lógica de la aplicación. A menudo, se utiliza el atributo `value` para crear un código que se almacena en la base de datos en lugar del valor mostrado. Este enfoque tiene un inconveniente: la parte de interfaz de usuario (capa de presentación) debe conocer los códigos, lo cual no es una buena idea. La solución óptima consiste en generar dinámicamente el formulario a partir de la base.

Lista de selección múltiple

Para las listas de selección múltiple, es decir, para un campo `<select>` sin atributo `multiple` (fruta en el siguiente ejemplo), la variable asociada contiene el valor contenido en el atributo `value` de la etiqueta `<option>`, o, si no hay atributo `value`, el valor mostrado en la lista (es decir, detrás de la etiqueta `<option>`). Tenga en cuenta que esto es válido sólo para la última opción seleccionada, si la variable es una variable escalar. En consecuencia, para obtener una lista de selección múltiple, es necesario utilizar una matriz.

Ejemplo (con atributo value)

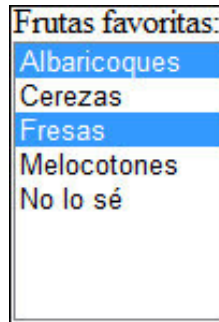
```
<br />Fruta favorita:<br />
<select name="fruta[]" multiple="multiple" size="8">
  <option value="A">Albaricoques</option>
  <option value="C">Cerezas</option>
  <option value="F">Fresas</option>
  <option value="M">Melocotones</option>
  <option value="?" selected="selected">
```

```

        No lo sé</option>
    </select>

```

Opciones seleccionadas



Resultado

```

$_POST:
fruta =
    0 = A
    1 = F

```

Como ya hemos visto, PHP rellena la tabla con una línea para cada opción seleccionada y numera estas líneas. En el caso de la lista de selección múltiple, este modo de funcionamiento no plantea ningún problema.

Ejemplo (sin opción value)

```

<br />Fruta favorita:<br />
<select name="fruta[]" multiple="multiple" size="8">
    <option>Albaricoques</option>
    <option>Cerezas</option>
    <option>Fresas</option>
    <option>Melocotones</option>
    <option selected="selected">
        No lo sé</option>
</select>

```

Resultados (con la misma selección anteriormente)

```

$_POST:
fruta =
    0 = Albaricoques
    1 = Fresas

```

Se puede elegir una u otra posibilidad de acuerdo a las necesidades de la lógica de la aplicación (los mismos principios que para la lista de selección única).

Utilizar una matriz para todo el formulario no es un problema.

Ejemplo (con atributo value)

```

<br />Fruta favorita:<br />
<select name="entrada[fruta][]" multiple="multiple" size="8">
    <option value="A">Albaricoques</option>
    <option value="C">Cerezas</option>
    <option value="F">Fresas</option>

```

```

<option value="M">Melocotones</option>
<option value="?" selected="selected">
    No lo sé</option>
</select>

```

Resultados (con la misma selección anteriormente)

```

$_POST:
entrada =
fruta =
    0 = A
    1 = F

```

Botón de validación

Para un botón de validación, es decir, para un campo `<input>` de tipo submit (OK en el siguiente ejemplo), PHP crea una variable que lleva el nombre del botón (atributo `name`) y tiene como valor el del atributo `value`, sólo si se pulsa el botón.

Ejemplo

```
<input type="submit" name="enviar" value="OK" />
```

Resultado (si se pulsa el botón)

```

$_POST:
enviar = OK

```

Si el botón no tiene nombre, no se crea ninguna variable. Esto no tiene importancia si:

- no es necesario saber cómo se llama el script (presentación inicial o procesamiento del formulario).
- hay un solo botón de validación.

En otros casos, se deben nombrar los botones.

El script siguiente muestra cómo diferenciar entre la llamada del script para la presentación inicial y la llamada del script para el procesamiento del formulario (ver sección Información general - Interacción entre un formulario y un script PHP, tercer método).

Ejemplo

```

<?php
...
// Probar cómo se llama el script if (isset($_POST['enviar'])) {
    // Existe una línea en la variable $_POST
    // correspondiente al botón OK llamada "enviar":
    // el script se llama en la validación del formulario.
    // => Procesar el formulario ...
    ...
} else {
    // El script no se llama por el clic en el
    // botón OK. Si no hay otro botón "submit", el
    // script se llama para la presentación inicial.
    // => Inicializar el formulario ...
    ...
}
?>

```


Surge un problema si el formulario tiene un único cuadro de texto, ningún botón y el usuario pulsa ENTER o RETURN. En este caso, el formulario se envía correctamente, pero no hay botón de validación para completar la prueba en el script PHP. La solución consiste en probar si la variable `$_*` está vacía o no (con `empty` pero no `isset` ya que la matriz no existe).

Si el formulario contiene dos botones de validación con nombres diferentes (atributo `name`), el primero `ok` y el segundo `cancelar`, es posible determinar en qué contexto se llama al script.

Ejemplo

```
// Probar cómo se llama al script
if (isset($_POST['ok'])) {
    // botón OK
} elseif (isset($_POST['cancelar'])) {
    // botón Cancelar
} else {
    // Presentación inicial
}
```

Si el formulario contiene dos botones de validación con el mismo nombre (atributo `name="enviar"` por ejemplo), pero valores (atributo `value`) distintos, el primero `OK` y el segundo `Cancelar`, es posible determinar en qué contexto se llama al script de la siguiente manera.

Ejemplo

```
// Probar cómo se llama al script
if ($_POST['enviar'] == 'OK') {
    // botón OK
} elseif ($_POST['enviar'] == 'Cancelar') {
    // botón Cancelar
} else {
    // Presentación inicial
}
```

Botón de imagen

Para un botón de imagen, es decir, para un campo `<input>` de tipo `image` (ejemplo validar a continuación), PHP crea dos variables que llevan el nombre del botón (atributo `name`) seguido de `_x` y `_y` y dando la posición relativa, en píxeles, del clic con respecto al ángulo situado en la parte superior izquierda de la imagen (sólo si se hace clic en la imagen). Si el botón no tiene ningún nombre, las dos variables se llaman `x` e `y`.

Ejemplo

```
<input type="image" name="validar" src="validar.gif" />
```

Resultado si se hace clic en la imagen

```
$_POST:
validar_x = 5
validar_y = 8
```

Entonces es posible procesar la posición del clic si es significativa desde el punto de vista de la lógica de la aplicación.

Si hay varios botones de imagen en el formulario, es posible determinar qué botón provoca el envío del formulario.

Ejemplo

```
// Probar cómo se llama el script
if (isset($_POST['validar_x'])) {
    // botón Validar
} else ...
```

Botón "reset" o "button"

Un clic en los botones correspondientes a los campos `<input>` de tipo `reset` o `button` no provoca el envío del formulario, ni la llamada del script de procesamiento. Estos botones permiten realizar una acción simple del lado del navegador (por ejemplo, en JavaScript).

3. Resumen

Debemos acostumbrarnos a designar correctamente (atributo `name`) todos los campos del formulario con nombres distintos o utilizar una denominación de tipo matriz, especialmente asociativa, para facilitar el mantenimiento y la legibilidad del código.

Para los grupos de botones de opción, se debe especificar un atributo `value` distinto para cada botón.

Para las casillas de verificación, se deben utilizar diferentes nombres (atributo `name`) y/o valores diferentes (atributo `value`), para garantizar la diferencia en la llegada; en caso de utilizar una matriz, no deje que PHP realice la numeración (sin []).

Para las listas de selección múltiple, se debe utilizar una denominación de tipo matriz para poder recuperar la lista de valores seleccionados; utilice el atributo `value` para recuperar un valor diferente del que aparece en la lista.

Del mismo modo, es necesario nombrar los botones de validación para poder saber cómo se llama al script PHP. Si se utilizan varios botones, es posible utilizar el mismo nombre, siempre y cuando los valores (atributo `value`) sean diferentes.

Los diferentes tipos de campo del formulario utilizado en esta sección son un buen ejemplo.

Mismo ejemplo con una matriz

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Entrada</title>
  </head>
  <body>
    <form action="entrada.php" method="post">
      <div>
        Nombre:
        <input type="text" name="entrada[apellido]" value=""
          size="20" maxlength="20" />
        Contraseña:
        <input type="password" name="entrada[contraseña]" value=""
          size="20" maxlength="20" />
        <br />Sexo:
        <input type="radio" name="entrada[sexo]" value="M" /> Masculino
        <input type="radio" name="entrada[sexo]" value="F" /> Femenino
        <input type="radio" name="entrada[sexo]" value="?"
          checked="checked" /> No lo sé
```

```

<br />Foto:
<input type="file" name="entrada[foto]" value="" size="50" />
<br />Colores favoritos:
<input type="checkbox" name="entrada[colores][azul]" />Azul
<input type="checkbox" name="entrada[colores][blanco]" />Blanco
<input type="checkbox" name="entrada[colores][rojo]" />Rojo
<input type="checkbox" name="entrada[colores][ninguno]"
        checked="checked" />No lo sé

<br />Idioma:
<select name="entrada[idioma]">
    <option value="E">Español</option>
    <option value="F" selected="selected" >Francés</option>
    <option value="I">Italiano</option>
</select>
<br />Fruta favorita:<br />
<select name="entrada[fruta][]" multiple="multiple" size="8">
    <option value="A">Albaricoques</option>
    <option value="C">Cerezas</option>
    <option value="F">Fresas</option>
    <option value="P">Melocotones</option>
    <option value="?" selected="selected">
        No lo sé</option>
</select>
<br />Comentarios:<br />
<textarea name="entrada[comentario]"
        rows="4" cols="50"></textarea>

<br />
<input type="hidden" name="entrada[invisibles]" value="123" />
<br />
<input type="submit" name="enviar" value="OK" />
<input type="image" name="validar" src="validar.gif" />
<input type="reset" name="borrar" value="Borrar" />
<input type="button" name="action" value="No hacer nada" />
</div>
</form>
</body>
</html>

```

Resultado (con la misma entrada que en el ejemplo original)

\$_POST:

```

entrada =
  nombre = HEURTEL
  contraseña = olivier
  sexo = M
  foto = identidad.jpg
  colores =
    azul = on
    rojo = on
  idioma = F
  fruta =
    0 = A
    1 = F
  comentarios = Consultor en sistemas informáticos
  invisible = 123
  enviar = OK

```

Construir un formulario de forma dinámica

Al igual que el resto de la página, la totalidad o parte de un formulario se puede construir de forma dinámica. En esta sección se abordan tres casos:

- generar todo el formulario;
- generar valores iniciales en los campos de entrada;
- generar una lista de opciones.

Generar todo el formulario

Si existe una descripción del formulario de una forma u otra, es posible generar todo el formulario.

En el siguiente ejemplo simplificado, se supone que se recupera (en un archivo, en una base de datos...) una descripción del formulario como una matriz de dos dimensiones: cada línea de la matriz contiene una descripción del campo en forma de matriz con el título, el tipo, el nombre y el valor.

```
<?php
// Matriz que contiene la descripción del formulario.
$formulario = array(
    array('Apellido: ', 'text', 'apellido', 'HEURTEL'),
    array(' ', 'submit', 'ok', 'OK') );
// Generación del formulario mediante un bucle
// en la matriz.
echo '<form action="entrada.php" method="POST">';
foreach($formulario as $campo) {
    echo "$campo[0]<input type=\"$campo[1]\" ",
        "name=\"$campo[2]\" value=\"$campo[3]\"><br />";
}
echo '</form>';
?>
```

Resultado en pantalla



Resultado en el código fuente de la página del navegador (todo está en una línea)

```
<form action="entrada.php" method="POST">Apellido: <input
type="text" name="apellido" value="HEURTEL"><br /><input
type="submit" name="ok" value="OK"><br /></form>
```

Generar valores iniciales en los campos de entrada

Ya hemos hablado de esta posibilidad en diferentes ejemplos.

Ejemplo

```
<form action="entrada.php" method="POST">
Apellido: <input type="text" name="apellido"
        value="<?php echo $apellido?>"><br />
<input type="submit" name="ok" value="OK">
</form>
```

En este caso, suponemos que \$apellido es una variable inicializada en el resto del script PHP.

Generar una lista de opciones

Se puede utilizar código PHP para generar listas de opciones, ya sea en un campo <select> (lista de selección única o múltiple), o bien en un campo <input> de tipo radio (grupo de botones de opción), o bien en un campo <input> de tipo checkbox (casilla de verificación).

Los datos mostrados proceden a menudo a una base de datos y es interesante poder construir este campo del formulario de forma dinámica a partir de los datos existentes en la base de datos.

Ejemplo con una lista de selección múltiple

```
<?php
// Lista de frutas a mostrar en la lista, en
// forma de matriz asociativa que da el código de la
// fruta (clave de la matriz) y el título de la fruta.
$fruta_del_mercado = array(
    'A' => 'Albaricoques',
    'C' => 'Cerezas',
    'F' => 'Fresas',
    'M' => 'Melocotones',
    '?' => 'No lo sé');
// Lista de frutas favoritas del usuario, en
// forma de una matriz que da el código de las frutas correspondientes.
$frutas_favoritas = array('A','F');
// Nota: más adelante veremos cómo recuperar
// esta información en una base de datos.
?>
<!-- construcción del formulario -->
<form action="entrada.php" method="POST">
Fruta favorita:<br />
<select name="fruta[]" multiple size="8">
<?php
// Código PHP que genera la parte dinámica del formulario.
// Examinar la lista que se va a mostrar y recuperar el código
// y el título.
foreach($fruta_del_mercado as $código => $título) {
    // Determinar si la línea debe estar seleccionada
    // - o si el código figura en la lista de frutas
    //   favoritas del usuario => búsqueda de $código
    //   en $fruta_favorita con la función in_array
    // - si es el caso, incluir el atributo "selected" en
    //   la etiqueta "option", en caso contrario no incluir nada.
    $selección =
        in_array($código,$fruta_favorita)?'selected="selected"' : '';
    // Generar la etiqueta "option" con la variable $código para
    // el atributo "value", la variable $selección para
    // la indicación de selección y la variable $título
    // para el texto mostrado en la lista.
    echo "<option value=\"$código\" $selección>$título</option>";
}
?>
</select>
</form>
```

Resultado en pantalla

Código fuente en el navegador

```
<!-- construcción del formulario -->
<form action="entrada.php" method="POST">
Fruta favorita:<br />
<select name="fruta[]" multiple size="8">
<option value="A"
selected="selected">Albaricoques</option><option value="C"
>Cerezas</option><option value="F"
selected="selected">Fresas</option><option value="M"
>Melocotones</option><option value="?" >No lo
sé</option></select>
</form>
```

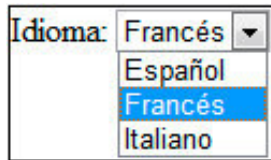
Este ejemplo es muy fácil de adaptar si el atributo value no se utiliza.

Ejemplo con una lista de selección única

```
<?php
// Lista de idiomas a mostrar en la lista, en
// forma de matriz asociativa que da el código del
// idioma (clave de la matriz) y el título del idioma.
$idiomas_disponibles = array(
    'E' => 'Español',
    'F' => 'Francés',
    'I' => 'Italiano');
// Código del idioma del usuario
$idioma = 'E';
?>
<!-- construcción del formulario -->
<form action="entrada.php" method="POST">
Idioma:<br />
<select name="idioma">
<?php
// Código PHP que genera la parte dinámica del formulario.
// Examinar la lista que se va a mostrar y recuperar el código
// y el título.
foreach($idiomas_disponibles as $código => $título) {
    // Determinar si la línea debe estar seleccionada
    // - o si el código es igual al código del idioma del
    // usuario
    // - si es el caso, incluir el atributo "selected" en
    // la etiqueta "option", en caso contrario no incluir nada
    $selección = ($código == $idioma)?'selected="selected"':'';
    // Generar la etiqueta "option" con la variable $código para
    // la opción "value", la variable $selección para
    // la indicación de selección y la variable $título
```

```
// para el texto mostrado en la lista.  
echo "<option value=\"\$código\" \$selección>$título</option>";  
}  
?>  
</select>  
</form>
```

Resultado en pantalla



Idioma:	Francés ▼
	Español
	Francés
	Italiano

Código fuente en el navegador

```
<!-- construcción del formulario -->  
<form action="entrada.php" method="POST">  
Idioma:<br />  
<select name="idioma">  
<option value="E" >Español</option><option value="F"  
selected="selected">Francés</option><option value="I"  
>Italiano</option></select>  
</form>
```

Se pueden utilizar técnicas similares para construir una lista de casillas de verificación, un grupo de botones de opción...

Controlar los datos introducidos

1. Información general

En la primera parte de este capítulo, vimos cómo recuperar los datos introducidos.

A continuación, es necesario comprobar que los datos introducidos son correctos, es decir, que respetan las normas de gestión definidas para la aplicación.

- Para la seguridad del sitio, es necesario no fiarse de los datos procedentes del exterior (formulario, URL, pero esto también lo veremos más adelante, cookie, etc.). Estos datos se deben controlar, filtrar, para evitar posibles ataques de un usuario malintencionado.

El objetivo de este apartado es proporcionar una orientación sobre las técnicas más utilizadas en PHP para realizar esta comprobación. Otro posible enfoque consiste en realizar un control en JavaScript en el navegador, se trata de evitar un viaje de ida y vuelta al servidor.

2. Comprobaciones clásicas

Limpieza de los espacios en blanco

Para los campos de entrada libre (<input> de tipo text o password, <textarea>), la función `trim` (ver capítulo Variables, constantes, tipos y matrices - sección Algunas funciones útiles sobre las cadenas de caracteres, las fechas y los números) se puede utilizar para eliminar los espacios en blanco no deseados al principio y/o al final de la cadena.

Ejemplo

```
// Recuperar el valor introducido en el campo "apellido" y limpiar
// los espacios en blanco (al principio y al final)
$apellido = trim($_POST['apellido']);
```

Campos obligatorios

Comprobar si un campo es obligatorio resulta muy simple: basta con comprobar si la variable asociada contiene un valor.

Ejemplo

```
$apellido = trim($_POST['apellido']);
if ($apellido == '') {
    // $apellido vacío = campo "apellido" no rellenado => hacer algo
}
```

Longitud máxima de una cadena

Para los campos de entrada libre, la longitud de la información introducida se puede controlar con la función `strlen` (ver capítulo Variables, constantes, tipos y matrices - sección Algunas funciones útiles sobre las cadenas de caracteres, las fechas y los números).

Ejemplo


```
$apellido = trim($_POST['apellido']);
if (strlen($apellido) > 20) {
    // $apellido demasiado largo => hacer algo
}
```

El atributo `maxlength` de la etiqueta `input` permite un control adicional en el momento de la entrada de datos (el navegador es el responsable).

Caracteres permitidos para una cadena - Formato

Si es necesario, el uso de expresiones regulares (ver capítulo Variables, constantes, tipos y matrices - sección Algunas funciones útiles sobre las cadenas de caracteres, las fechas y los números) permite controlar muy fácilmente que sólo ciertos caracteres están presentes y, posiblemente, si la cadena introducida respeta un formato específico.

Por ejemplo, supongamos que la contraseña debe verificar la siguiente regla: comenzar con una letra, seguida de letras, números o caracteres `_#$` con una longitud mínima de 4.

Ejemplo

```
$contraseña = trim($_POST['contraseña']);
$motivo = '/^[a-z][a-z0-9_#$]{3,}/i';
if (preg_match($patrón,$contraseña) == 0) {
    // Contraseña no válida
}
```

Algunas explicaciones sobre la expresión regular utilizada (`/^[a-z][a-z0-9_#$]{3,}/i`):

- carácter delimitador `=/`
- opción `i` utilizada para no hacer diferenciación entre mayúsculas y minúsculas.
- `^` = comienza por ...
- `[a-z]` = una letra entre a y z (o A y Z con la opción `i`) ...
- `[a-z0-9_#$]{3,}` = seguido de al menos tres (`{3,}`) caracteres entre los que figuran: a a z (y por tanto A a Z), 0 a 9 y los caracteres `_#$`.

Otros ejemplos se presentan más adelante en esta sección con fechas y números.

Validez de una fecha - Rango de valores

Por lo general, para una fecha es necesario comprobar si:

- cumple con un formato (DD/MM/AAAA, por ejemplo);
- validez (sin 32/13/2001).

La verificación del cumplimiento con el formato y los caracteres permitidos se puede realizar de manera muy simple con una expresión regular.

Ejemplo

```
$fecha_nacimiento = trim($_POST['fecha_nacimiento']);
$formato_fecha = '#^([0-9]{1,2})/([0-9]{1,2})/([0-9]{4})$#';
if (preg_match($formato_fecha,$fecha_nacimiento) == 0) {
    // Formato de fecha incorrecto.
}
```

Algunas explicaciones sobre la expresión regular utilizada `(#^([0-9]{1,2})/([0-9]{1,2})/([0-9]{4}))$#`:

- carácter delimitador = #
- ^ = comienza por ...
- `([0-9]{1,2})` = uno o dos dígitos (subpatrón de captura) ...
- / = seguido por el carácter "/" ...
- `([0-9]{1,2})` = seguido de uno o dos dígitos (subpatrón de captura) ...
- / = seguido por el carácter "/" ...
- `([0-9]{4})` = seguido de cuatro dígitos (subpatrón de captura) ...
- \$ = seguido de ... inada! La cadena debe terminar inmediatamente.

Para comprobar la validez de la fecha, es posible utilizar la función `checkdate` (ver capítulo Variables, constantes, tipos y matrices - Algunas funciones útiles sobre las cadenas de caracteres, las fechas y los números).

Al principio, se debe recuperar los componentes de la fecha introducida. Hay varias opciones disponibles:

- Con la función `explode`:

```
$jma = explode('/', $fecha_nacimiento);
// $jma[0] contiene el día
// $jma[1] contiene el mes
// $jma[2] contiene el año
if (! checkdate($jma[1], $jma[0], $jma[2])) {
// Fecha no válida.
}
```

- Variante con las funciones `explode` y `list`:

```
list($día, $mes, $año) = explode('/', $fecha_nacimiento);
// Recuperación de componentes en variables independientes.
if (! checkdate($mes, $día, $año)) {
// Fecha no válida.
}
```

- Con el tercer parámetro de la función `preg_match`, que permite recuperar partes de la cadena, y esta directamente en la prueba de cumplimiento del formato:

```
$fecha_nacimiento = trim($_POST['fecha_nacimiento']);
$formato_fecha = '#^([0-9]{1,2})/([0-9]{1,2})/([0-9]{4}))$#';
if (preg_match($formato_fecha, $fecha_nacimiento) == 0) {
// Formato de fecha incorrecto.
} else {
// Formato de fecha correcto.
// $jma contiene los distintos componentes con
// índices distintos en relación al ejemplo anterior:
// $jma[1] contiene el día
// $jma[2] contiene el mes
// $jma[3] contiene el año
if (! checkdate($jma[2], $jma[1], $jma[3])) {
// Fecha no válida.
}
```

}

Para probar el rango de valores, la manera más fácil es hacer una comparación numérica (o alfabética) sobre un número (o cadena) construida en formato AAAAMMDD (20010828 para el 28/08/2001).

Ejemplo

```
// Recuperar los componentes de la fecha.
list($día,$mes,$año) = explode('/', $fecha_nacimiento);
// Construir una cadena con el formato AAAAMMDD
$aaaammdd = sprintf('%04d%02d%02d', $año, $mes, $día);
// Definir las fechas mini y maxi según el mismo formato.
$fecha_mini = '19000101'; // 01/01/1900
$fecha_maxi = date('Ymd'); // fecha del día
// Comparar.
if ($aaaammdd < $fecha_mini o $aaaammdd > $fecha_maxi) {
    // Fecha fuera del rango permitido
}
```

Validez de un número - Rango de valores

Se pueden utilizar varias técnicas para comprobar que un número tiene un formato adecuado:

- las expresiones regulares;
- la función `is_numeric`;
- la conversión de los datos introducidos y la comprobación del resultado obtenido.

Ejemplos con expresiones regulares

```
// Comprobar que un dato es un número entero.
preg_match('/^[+-]?[0-9]+$/', $variable)
// Comprobar que un dato es un número entero y controlar
// el número de cifras (entre 1 y 3 por ejemplo)
preg_match('/^[+-]?[0-9]{1,3}$/', $variable)
// Comprobar que un dato es un número decimal (en este
// ejemplo la coma y el punto se aceptan como
// separador decimal).
preg_match('/^[+-]?[0-9]+[.,]?[0-9]*$/', $variable)
```

Para el rango de valores, basta con una simple prueba del siguiente tipo.

Ejemplo

```
if ($variable < mínimo or $variable > maximum) {
    // $variable fuera del rango permitido
}
```

Validez de una dirección de correo electrónico

Una vez más, las expresiones regulares serán útiles. El tema ha sido objeto de numerosos estudios y se pueden encontrar muchas soluciones en Internet.

La siguiente solución funciona bien para las estructuras de dirección actuales:

```
preg_match(
    '/^[a-z0-9]+([._-]?[a-z0-9]+)*' .           // inicio
    '@' .                                       // continuación
```

```
'[a-z0-9]+([.-]?[a-z0-9]+)*\.[a-z]{2,4}$/' // fin
$dirección_correo
)
```

Para facilitar la lectura, la expresión racional se construye por concatenación de tres cadenas.

Algunas explicaciones sobre la expresión regular utilizada:

- Carácter delimitador = /
- Opción i utilizada para no hacer diferenciación entre mayúsculas y minúsculas.
- ^ = comienza por ...
- [0-9a-z]+ = una secuencia que incluye letras o números ...
- ([.-]?[a-z0-9]+)* = posiblemente seguido de una o varias secuencias, cada una puede comenzar por un guión, un guión bajo o un punto seguido de letras o de números ...
- @ = seguido de una @ ...
- [0-9a-z]+ = seguido de una secuencia que incluye letras o números ...
- ([.-]?[a-z0-9]+)* = seguido posiblemente de una o varias secuencias, cada una puede comenzar por un guión o un punto seguido de letras o de números ...
- \. = seguido de un punto ...
- [a-z]{2,4} = seguido de dos a cuatro letras
- \$ = seguido de ... inada! La cadena debe terminar.



La expresión regular permite verificar que la dirección es sintácticamente correcta, pero no controlar que realmente existe.

Los problemas en los datos introducidos

1. La característica "magic quotes"

Antes de la versión 5.4, PHP ofrecía una característica denominada "magic quotes", cuyo principal objetivo era resolver un problema potencial relacionado con el registro de datos en una base de datos realizando una codificación en los datos introducidos en un formulario.

Si la directiva de configuración `magic_quotes_gpc` estaba en `on`, todos los datos que llegan por un método `GET` o `POST` (o por una cookie) se codificaban automáticamente con una barra invertida (`\`) delante de los caracteres apóstrofo (`'`), comillas (`"`) y por supuesto la barra invertida (`\`).

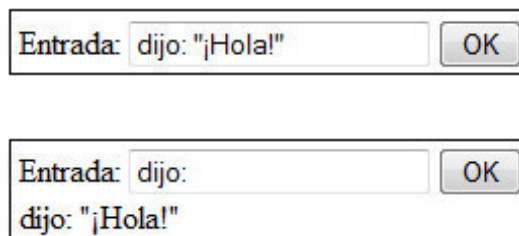
Esta característica era realmente interesante, sobre todo el apóstrofo, si los datos estaban destinados a ser almacenados en una base de datos SQL (*Structured Query Language*) donde el delimitador de cadena de caracteres es el apóstrofo (ver capítulo Acceder a las bases de datos).

En la práctica, esta característica "magic quotes" planteaba dos problemas: necesidad de eliminar o añadir la barra invertida según el contexto y el uso de los datos (visualización, registro en una base de datos, escritura de un código independiente de la configuración de PHP, etc.). Esta característica se declaró obsoleta en la versión 5.3 y se eliminó definitivamente en la versión 5.4: la directiva `magic_quotes_gpc` ya no existe y los datos introducidos no se codifican de ninguna manera. Si los datos están destinados a ser visualizados, no se requiere ningún tratamiento, y si están destinados a ser registrados en una base de datos, conviene efectuar la codificación adaptada (ver el capítulo Acceder a las bases de datos).

2. Otros problemas en los datos introducidos

Puede producirse otro problema de presentación en un campo de formulario si el texto que se muestra contiene un signo de comillas (`"`).

Ejemplo



Código fuente de la página en el navegador (extracto)

```
<form action="entrada.php" method="post">
<div>
  Entrada: <input type="text" name="entrada"
    value="dijo: ";Hola!" />
  <input type="submit" name="ok" value="OK" />
  <br />dijo: ";Hola!"  </div>
</form>
```

En HTML, en los atributos de las etiquetas (`value`, `name`...), el delimitador de cadena es el signo de comillas. En el atributo `value`, la secuencia `"dijo: "` se considera como el valor del atributo y el resto de la cadena se pasa por alto. El problema se produce incluso si las comillas se escapan por medio del carácter `\` porque este último no es un carácter de escape en HTML.

Otro problema de presentación se produce en la página si los datos mostrados contienen etiquetas HTML.

Ejemplo

Entrada:

Olivier **Heurtel**

El fragmento `Olivier Heurtel` da la palabra **Heurtel** en negrita cuando se visualiza en la página HTML. La secuencia `` introducida por el usuario se encuentra tal cual en el código fuente de la página y, por lo tanto, se interpreta por el navegador como la etiqueta de la negrita.

Por último, podemos encontrarnos con un tercer problema al escribir en un campo de comentario.

Ejemplo

Entrada:

Primera línea.
Segunda línea.
Tercera línea.

Primera línea. Segunda línea. Tercera línea.

Un texto de varias líneas en el campo `<textarea>` se vuelve a mostrar tal cual en el campo, pero se muestra sin los saltos de línea en la página HTML. El texto está presente con saltos de línea en el código fuente de la página, pero el salto de línea, fuera de un campo `<textarea>` no es interpretado por el navegador: Es necesario incorporar una etiqueta `
`.

Por tanto, vemos que aparecen tres problemas relativos a la presentación en la página HTML de los datos introducidos por el usuario:

- La presencia del carácter de comillas que puede suponer un problema cuando se usan los datos en un formulario (atributo `value`).
- La presencia de etiquetas HTML válidas que se interpretan como tal por el navegador.
- El no tener en cuenta los saltos de línea en un texto.

El segundo "problema" puede ser interesante si desea ofrecer la posibilidad a un usuario avanzado de introducir texto con algún formato para mostrarlo más tarde en una página.

Para resolver estos tres problemas, PHP dispone de cuatro funciones: `htmlspecialchars`, `htmlentities`, `nl2br` y `strip_tags`.

La función `htmlspecialchars` toma una cadena de caracteres y la devuelve reemplazando ciertos caracteres por su equivalente HTML:

Sintaxis

```
cadena htmlspecialchars(cadena texto [, entero opción [, cadena juego
[, booleano doble_codificación]])
```

texto	Cadena que se va a procesar.	Los
opción	Funcionamiento para los caracteres de comillas (") y el apóstrofo (').	
juego	Juego de caracteres utilizado para la conversión.	
doble_codificación	Indica si es necesario (TRUE, valor predeterminado) o no (FALSE) codificar las entidades HTML ya codificadas.	

Se incluyó en la versión 5.2.3.

caracteres convertidos son los siguientes:

Entrada	Salida
&	&
"	"
'	'
<	<
>	>

El

segundo parámetro permite especificar el funcionamiento para los caracteres de comillas (") y el apóstrofo ('):

Valor	Comportamiento
ENT_COMPAT	La conversión de las comillas, pero no de los apóstrofes (por defecto).
ENT_NOQUOTES	Ninguna conversión.
ENT_QUOTES	Conversión de dos caracteres.
ENT_IGNORE	Ignora las secuencias de código no válidas (en Unicode) en lugar de devolver una cadena vacía (vista previa en la versión 5.3, pero desaconsejada por razones de seguridad).
ENT_SUBSTITUTE	Sustituye las secuencias de código no válido (en Unicode) por un carácter de sustitución (U+FFFD).
ENT_DISALLOWED	Sustituye las secuencias de código no válido (en Unicode) para el tipo de documento especificado por un carácter de sustitución (U+FFFD).
ENT_HTML401	Gestiona el código como si fuese HTML 4.01.
ENT_XML1	Gestiona el código como si fuese XML 1.
ENT_XHTML	Gestiona el código como si fuese XHTML.
ENT_HTML5	Gestiona el código como si fuese HTML 5.

Se
pueden

combinar varios indicadores con el operador « o lógico » (|). El valor predeterminado es ENT_COMPAT | ENT_HTML401.

El tercer parámetro permite definir el juego de caracteres utilizado para la conversión: ISO-8859-1, ISO-8859-15, UTF-8, etc. El valor predeterminado es ISO-8859-1 antes de la versión 5.4 y UTF-8 después. Aunque este parámetro es opcional, se recomienda encarecidamente especificar un valor correcto adaptado a su código.

Desde la versión 5.1.0, existe una función `htmlspecialchars_decode` que realiza la conversión inversa de la función `htmlspecialchars`.

Ejemplo

```
<?php
$texto = 'Olivier & Co. ha declarado: "It\'s raining!";
echo htmlspecialchars($texto, ENT_QUOTES, UTF-8);
?>
```

Resultado en el código fuente de la página (los elementos pertinentes están en negrita)

Olivier & Co. ha declarado: **It's raining!**

La función `htmlentities` presenta un comportamiento idéntico al de `htmlspecialchars` pero para todos los caracteres que tienen un equivalente en HTML (caracteres acentuados, especialmente).

Sintaxis

`cadena htmlentities (cadena texto [, entero opción [, cadena juego [, booleano doble_codificación]])`

texto	Cadena que se va a procesar.	El
opción	Funcionamiento para los caracteres de comillas (") y el apóstrofo (').	
juego	Juego de caracteres utilizado para la conversión.	
doble_codificación	Indica si es necesario (TRUE, valor predeterminado) o no (FALSE) codificar las entidades HTML ya codificadas. Se incluyó en la versión 5.2.3.	

segundo y tercer parámetros tienen el mismo significado que para la función `htmlspecialchars`.

Ejemplo

```
<?php
$texto = 'Olivier & Co. ha declarado : "It\'s raining!";
echo htmlentities($texto, ENT_QUOTES, 'UTF-8');
?>
```

Resultado en el código fuente de la página (los elementos pertinentes están en negrita)

Olivier & Co. ha declarado : **It's raining!**

La función `nl2br` toma una cadena y devuelve esta cadena después de haber insertado una etiqueta HTML de salto de línea delante de cada salto de línea: la etiqueta es `
` (compatibilidad XHTML) a partir de la versión 4.0.5 de PHP y `
` anteriores.

Sintaxis

`cadena nl2br (cadena texto)`

texto	Cadena que se va a procesar.
-------	------------------------------

Ejemplo

```
<?php
$texto = "Primera línea.\nSegunda línea.";
echo nl2br($texto);
?>
```

Resultado en el código fuente de la página (los elementos pertinentes están en negrita)

Primera línea.**
**
Segunda línea.

Por último, la función `strip_tags` toma una cadena de caracteres y la devuelve después de haber eliminado todas las etiquetas HTML que contiene.

Sintaxis

`cadena strip_tags (cadena texto[, cadena etiquetas_permitidas])`

texto	Cadena que se va a procesar.	El
-------	------------------------------	----

etiquetas_permitidas

Lista de etiquetas que se conservarán en la cadena.

segundo parámetro permite especificar las etiquetas que se van a conservar.

Ejemplo

```
<?php
$texto = "<b>Olivier</b> <i>Heurtel</i>";
echo $texto, '<br/>';
echo strip_tags($texto), '<br/>';
echo strip_tags($texto, '<b>') , '<br/>';
?>
```

Resultado

```
Olivier Heurtel
Olivier Heurtel
Olivier Heurtel
```

Estas funciones permitirán ayudar a manejar los diferentes problemas mencionados anteriormente.

Para evitar problemas de presentación en el navegador, es aconsejable realizar una transformación de los datos antes o en la instrucción echo.

Se pueden desarrollar funciones personales para realizar esta operación.

Ejemplo

```
<?php
// Función que permite mostrar datos en un formulario.
// Cifrar todos los caracteres HTML especiales.
function hacia_formulario($valor) {
    return htmlentities($valor, ENT_QUOTES, 'UTF-8');
}
// Función que permite mostrar datos en una página.
// Cifrar todos los caracteres HTML especiales.
// Convertir los saltos de línea en <br />.
function hacia_página($valor) {
    return nl2br(htmlentities($valor, ENT_QUOTES, 'UTF-8'));
}
?>
```

Estas funciones se pueden utilizar en un script de procesamiento de un formulario.

Ejemplo


```
<?php
// Inclusión del archivo que contiene las definiciones de nuestras
// funciones generales.
include('funciones.inc');
// Probar si la página se llama después de la validación del formulario
if (isset($_POST['ok'])) {
    // Recuperación del valor introducido en el formulario
    $apellido = isset($_POST['apellido'])?$_POST['apellido']:'';
    // El valor introducido se vuelve a mostrar en el formulario y
    // en la página ...
}
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
    <head><title>Entrada</title></head>
```

```

<body>
  <form action="entrada.php" method="post">
    <div>
      Apellidos:
      <input type="text" name="apellido"
        value="<?php echo hacia_formulario($apellido); ?>" />
      <input type="submit" name="ok" value="OK" /><br />
      <?php echo hacia_página($apellido); ?>
    </div>
  </form>
</body>
</html>

```

Resultado



Con este enfoque, la entrada se presenta tal cual: si el usuario ha introducido una etiqueta, se la vuelve a encontrar (no se interpreta por el navegador). Si es necesario, las funciones se pueden modificar para eliminar las etiquetas con la ayuda de la función `strip_tags`.

- La función `nl2br` debe llamarse después de las funciones `htmlentities` o `htmlspecialchars`. De lo contrario, la etiqueta `
` agregada por `nl2br` se codifica (en `
`) antes de insertarse en el código fuente de la página y, por lo tanto, no se interpreta por el navegador (el texto `
` aparece en la página). La función `nl2br` no debe llamarse para texto destinado a un campo `<textarea>` (una vez más, es necesario incluir `
` en el campo).

Utilización de filtros

1. Principios

Desde la versión 5.2.0, una nueva extensión permite filtrar y validar los datos, incluidos los introducidos por los usuarios.

Cada filtro está definido por un número (identificador), un nombre y posibles opciones y los indicadores que definen el comportamiento del filtro. Cada opción está definida por un nombre que se utiliza como clave en una matriz asociativa. Cada indicador se define por una constante; para especificar varios indicadores, basta con sumar las constantes correspondientes.

Algunos ejemplos de los filtros (ver la documentación para la descripción de todos los filtros):

Identificador (constante predefinida)	Descripción
<code>FILTER_VALIDATE_INT</code>	Valida un valor como entero. Las opciones <code>min_range</code> y <code>max_range</code> permiten definir un intervalo de validez.
<code>FILTER_VALIDATE_FLOAT</code>	Valida un valor como número de punto flotante.
<code>FILTER_VALIDATE_REGEXP</code>	Valida un valor utilizando una expresión regular compatible con PERL. La expresión regular que se va a utilizar se especifica con la opción <code>regexp</code> .
<code>FILTER_VALIDATE_EMAIL</code>	Valida un valor como dirección de correo electrónico.
<code>FILTER_SANITIZE_STRING</code>	Elimina las etiquetas contenidas en una cadena y codifica los caracteres ' y ". Hay disponibles varios indicadores para eliminar o codificar caracteres adicionales (véase más adelante).
<code>FILTER_SANITIZE_SPECIAL_CHARS</code>	Codifica en HTML los caracteres ', ", <, > y & así como todos los caracteres de código ASCII inferior a 32. Hay disponibles varios indicadores para eliminar o codificar caracteres adicionales (véase más adelante).

Los siguientes indicadores se pueden utilizar con los filtros `FILTER_SANITIZE_STRING` y `FILTER_SANITIZE_SPECIAL_CHARS`:

<code>FILTER_FLAG_STRIP_LOW</code>	Elimina los caracteres de código ASCII por debajo de 32.
<code>FILTER_FLAG_STRIP_HIGH</code>	Elimina los caracteres de código ASCII por encima de 127.
<code>FILTER_FLAG_ENCODE_HIGH</code>	Codifica en HTML los caracteres de código ASCII por encima de 127.

Además, los siguientes indicadores se pueden utilizar con el filtro `FILTER_SANITIZE_STRING`:

<code>FILTER_FLAG_NO_ENCODE_QUOTES</code>	No codifica los caracteres ' y ".
<code>FILTER_FLAG_ENCODE_AMP</code>	Codifica en HTML el carácter &.
<code>FILTER_FLAG_ENCODE_LOW</code>	Codifica en HTML los caracteres de código ASCII por debajo de 32.

Por otra parte, el indicador `FILTER_NULL_ON_FAILURE` se puede utilizar con todos los filtros para que las funciones devuelvan el valor `NULL` en lugar del valor `FALSE` caso de fallo.

Estos filtros se pueden utilizar en las funciones `filter_var`, `filter_var_array`, `filter_input` y `filter_input_array`.

Funciones `filter_var` y `filter_var_array`

La función `filter_var` permite filtrar datos.

Sintaxis

```
mixto filter_var(mixto datos[, entero filtro[, mixto opciones_indicadores]])
```

<code>datos</code>	Datos a filtrar.
<code>filtro</code>	Identificador del filtro a aplicar (<code>FILTER_DEFAULT</code> por defecto).
<code>opciones_indicadores</code>	Opciones y/o posibles indicadores del filtro (ver más abajo).

Esta función devuelve los datos filtrados, o `FALSE` si el filtro falla (o `NULL` si se utiliza el indicador `FILTER_NULL_ON_FAILURE`).

En el caso más general, el parámetro `opciones_indicadores` se especifica en forma de una matriz asociativa que contiene una o dos líneas, con las siguientes claves: `flags` para los indicadores y `opciones` para las opciones. Como se mencionó anteriormente, algunos indicadores se pueden proporcionar mediante la suma de las constantes correspondientes. El valor de las opciones también se define como una matriz asociativa en la que se utiliza el nombre de la opción como clave.

En el caso de que el parámetro `opciones_indicadores` sólo define los indicadores, el valor correspondiente se puede pasar directamente como parámetro, sin necesidad de utilizar una matriz.

Ejemplo 1

```
<?php
echo "<b>Filtrar un número entero</b><br />";
$valores = array('123','abc','1.2',NULL);
foreach ($valores as $x) {
    echo "$x => ";
    var_export(filter_var($x,FILTER_VALIDATE_INT));
    echo "<br />";
}
echo "<b>+ NULL en lugar de FALSE en caso de error</b><br />";
$x = 'abc';
// indicador pasado en opción directamente
$opciones = FILTER_NULL_ON_FAILURE;
echo "$x => ";
var_export(filter_var($x,FILTER_VALIDATE_INT,$opciones));
echo "<br />";
echo "<b>Filtrar un número entero (0-100)</b><br />";
// opciones del filtro definidas a través de una matriz asociativa
```

```

$opciones =
    array
    (
        'opciones' => array('min_range' => 0,'max_range' => 100)
    );
$valores = array('0','100','101');
foreach ($valores as $x) {
    echo "$x => ";
    var_export(filter_var($x,FILTER_VALIDATE_INT,$opciones));
    echo "<br />";
}
echo "<b>+ NULL en lugar de FALSE en caso de error</b><br />";
// Indicador agregado en la matriz de las opciones
$opciones =
    array
    (
        'opciones' => array('min_range' => 0,'max_range' => 100),
        'flags' => FILTER_NULL_ON_FAILURE
    );
$x = '101';
echo "$x => ";
var_export(filter_var($x,FILTER_VALIDATE_INT,$opciones));
echo "<br />";
echo "<b>Filtrar con una expresión regular</b><br />";
$regexp = '<^[0-9]{2}/[0-9]{2}/[0-9]{4}$>';
$opciones =
    array
    (
        'opciones' => array('regexp' => $regexp)
    );
$valores = array('01/01/2007','01/01/07');
foreach ($valores as $x) {
    echo "$x => ";
    var_export(filter_var($x,FILTER_VALIDATE_REGEXP,$opciones));
    echo "<br />";
}
?>

```

Resultado

Filtrar un número entero

```

123 => 123
abc => false
1.2 => false

```

+ NULL en lugar de FALSE en caso de error

```

abc => NULL

```

Filtrar un número entero (0-100)

```

0 => 0
100 => 100
101 => false

```

+ NULL en lugar de FALSE en caso de error

```

101 => NULL

```

Filtrar con una expresión regular

```

01/01/2007 => '01/01/2007'
01/01/07 => false

```

Ejemplo 2

```

<?php

```

```

$texto = "<b>It's raining</b>";
echo "// texto mostrado sin precaución<br />\n";
echo $texto,"<br />\n";
echo "// FILTER_SANITIZE_SPECIAL_CHARS<br />\n";
echo
    filter_var
    (
        $texto,
        FILTER_SANITIZE_SPECIAL_CHARS
    ),
    "<br />\n";
echo "// + opción FILTER_FLAG_ENCODE_HIGH<br />\n";
echo
    filter_var
    (
        $texto,
        FILTER_SANITIZE_SPECIAL_CHARS,
        FILTER_FLAG_ENCODE_HIGH
    ),
    "<br />\n";
echo "// FILTER_SANITIZE_STRING<br />\n";
echo filter_var($texto,FILTER_SANITIZE_STRING, "<br />\n";
echo "// + opción FILTER_FLAG_ENCODE_HIGH<br />\n";
echo
    filter_var
    ($texto,
        FILTER_SANITIZE_STRING,
        FILTER_FLAG_ENCODE_HIGH
    ),
    "<br />\n";
?>

```

Resultado en el navegador

```

// Texto mostrado sin precaución
It's raining
// FILTER_SANITIZE_SPECIAL_CHARS
<b>It's raining</b>
// + opción FILTER_FLAG_ENCODE_HIGH
<b>It's raining</b>
// FILTER_SANITIZE_STRING
It's raining
// + option FILTER_FLAG_ENCODE_HIGH
It's raining

```

Resultado de la fuente de la página

```

// Texto mostrado sin precaución<br />
<b>It's raining</b><br />
// FILTER_SANITIZE_SPECIAL_CHARS<br />
&#60;b&#62;It&#39;s raining&#60;/b&#62;<br />
// + option FILTER_FLAG_ENCODE_HIGH<br />
&#60;b&#62;It&#39;s raining&#60;/b&#62;<br />
// FILTER_SANITIZE_STRING<br />
It&#39;s raining<br />
// + option FILTER_FLAG_ENCODE_HIGH<br />
It&#39;s raining<br />

```

La función `filter_var_array` permite filtrar una matriz de datos.

Sintaxis

```
mixto filter_var_array(matriz datos[, mixto filtros])
[,booleano añadir vacío])
```

datos	Matriz asociativa que contiene los datos a filtrar. Las claves son cadenas de caracteres que identifican a cada dato a validar.
filtros	Definición de filtros a aplicar a cada uno de los datos de la matriz de datos.
añadir_vacío	Indica si se deben añadir líneas en el resultado para los datos que no existen en la matriz de origen (TRUE por defecto).

La función devuelve la matriz de datos filtrados; las líneas para las cuales el filtro ha fallado están como FALSE (o NULL si se utiliza el indicador FILTER_NULL_ON_FAILURE) o aquellas para las que los datos no existen están como NULL, salvo si el parámetro añadir_vacío está como FALSE, en cuyo caso no se añadirán las líneas en el resultado.

En el caso más general, el parámetro filtros se especifica en la forma de una matriz asociativa que contiene las claves de la matriz de datos (la correspondencia entre las matrices datos y filtros se efectúa por medio de la clave). Cada valor de la matriz filtros puede ser un identificador simple de filtro o una matriz asociativa que da una descripción más completa del filtro a aplicar. En este caso, las claves de la matriz filtros son filter para el identificador del filtro, flags para los indicadores que se van a aplicar al filtro y options para las opciones que se van a aplicar al filtro; los valores asociados a las claves flags y options se definen como para la función filter_var.

En el caso de que el mismo filtro se deba aplicar a todos los datos, sin indicador ni opción, el parámetro filtros puede ser un simple número entero igual al identificador del filtro.

Ejemplo

```
<?php
echo '<b>Filtrar una matriz de números enteros</b><br />';
$valores = array('123','abc');
var_export($valores);
echo '<br />=> ';
// Mismo filtro a aplicar a todos los datos,
// sin indicador ni opción.
var_export
    (filter_var_array($valores,FILTER_VALIDATE_INT));
echo '<br />';
echo '<b>Filtrar una matriz de datos diferentes (1)</b><br />';
$valores = array
    (
        'edad' => 123,
        'tamaño' => 'abc',
        'correo' => 'contact@olivier-heurtel.fr'
    );
// Filtro diferente, pero "simple" (sin indicador
// ni opción) a aplicar a los datos.
$filtros = array
    (
        'edad' => FILTER_VALIDATE_INT,
        'tamaño' => FILTER_VALIDATE_INT,
        'correo' => FILTER_VALIDATE_MAIL
    );
var_export($valores);
echo '<br />=> ';
```

```

var_export(filter_var_array($valores,$filtros));
echo '<br />';
echo '<b>Filtrar una matriz de datos diferentes (2)</b><br />';
$valores = array
(
    'edad' => 123,
    'tamaño' => 'abc',
    'correo' => 'contact@olivier-heurtel.fr'
);
// Filtro con opciones e indicador a aplicar a uno de los datos.
$filtro_edad = array
(
    'filter' => FILTER_VALIDATE_INT,
    'options' => array('min_range' => 0,'max_range' => 100),
    'flags' => FILTER_NULL_ON_FAILURE
);
// Observar la mención de un filtro para un dato
// que no existe.
$filtros = array
(
    'edad' => $filtro_edad,
    'tamaño' => FILTER_VALIDATE_INT,
    'peso' => FILTER_VALIDATE_INT, // no existe
    'correo' => FILTER_VALIDATE_MAIL
);
var_export($valores);
echo '<br />';
var_export(filter_var_array($valores,$filtros));
// Desactivar la acción de añadir elementos vacíos
echo '<br />';
echo '<b>Lo mismo que al desactivar la acción de añadir elementos vacíos</b>';
echo '<br /> => ';
var_export(filter_var_array($valores,$filtros,FALSE));
?>

```

Resultado

Filtrar una matriz de números enteros

```

array ( 0 => '123', 1 => 'abc', )
=> array ( 0 => 123, 1 => false, )

```

Filtrar una matriz de datos diferentes (1)

```

array ( 'edad' => 123, 'tamaño' => 'abc', 'correo' =>
'contact@olivier-heurtel.fr', )
=> array ( 'edad' => 123, 'tamaño' => false, 'correo' =>
'contact@olivier-heurtel.fr', )

```

Filtrar una matriz de datos diferentes (2)

```

array ( 'edad' => 123, 'tamaño' => 'abc', 'correo' =>
'contact@olivier-heurtel.fr', )
=> array ( 'edad' => NULL, 'tamaño' => false, 'peso' => NULL, 'correo' =>
'contact@olivier-heurtel.fr', )

```

Lo mismo que al desactivar la acción de añadir elementos vacíos

```

=> array ( 'edad' => NULL, 'tamaño' => false, 'correo' =>
'contact@olivier-heurtel.fr', )

```

En el último ejemplo, observe el valor NULL, que se ha asociado al dato peso (mencionado en el filtro, pero sin datos) y la posibilidad que nos ofrece de desactivar la acción de añadir estos elementos vacíos.

Funciones filter_input y filter_input_array

Las funciones `filter_input` y `filter_input_array` son similares a las funciones `filter_var` y `filter_var_array`, pero se aplican a datos externos a PHP (datos de un formulario, por ejemplo) y no a variables del script.

Sintaxis

```
mixto filter_input(entero origen, cadena nombre_variable[,  
entero filtro[, mixto opciones_indicadores]])
```

origen	Origen de los datos. Una de las constantes <code>INPUT_GET</code> (datos pasados por el método GET), <code>INPUT_POST</code> (datos pasados por el método POST), <code>INPUT_COOKIE</code> (datos pasados por una cookie).
nombre_variable	Nombre de la variable a procesar.
filtro	Identificador del filtro a aplicar (<code>FILTER_DEFAULT</code> por defecto).
opciones_indicadores	Opciones y/o posibles indicadores del filtro (idéntico a la función <code>filter_var</code>).

Esta función devuelve el dato filtrado si todo va bien, `FALSE` si el filtro ha fallado o `NULL` si la variable no está definida. Si se utiliza el indicador `FILTER_NULL_ON_FAILURE`, la función devuelve `NULL` si el filtro ha fallado o `FALSE` si la variable no está definida.

Sintaxis

```
mixto filter_input_array(entero origen[, mixto filtros[, booleano añadir_vacío]])
```

origen	Origen de los datos (idéntica a la función <code>filter_input</code>).
filtros	Definición de los filtros que se van a aplicar a cada uno de los datos de la matriz de datos (idéntico a la función <code>filter_var_input</code>).
añadir_vacío	Indica si se deben añadir líneas en el resultado para los datos que no existen en la matriz de origen (<code>TRUE</code> por defecto).

Esta función es equivalente a una llamada a la función `filter_var_array` efectuada en la matriz `$_GET`, `$_POST` o `$_COOKIE` correspondiente al origen (`INPUT_GET`, `INPUT_POST` o `INPUT_COOKIE`).

La función devuelve la matriz de los datos filtrados o `NULL` si el origen no contiene ningún dato; las líneas para las cuales el filtro ha fallado están como `FALSE` (o `NULL` si se utiliza el indicador `FILTER_NULL_ON_FAILURE`) y aquellas para las que la variable no existe están como `NULL`, salvo si el parámetro `añadir_vacío` está como `FALSE`, en cuyo caso no se añadirán las líneas en el resultado.

En la siguiente sección se darán ejemplos de uso de estas funciones.

2. Aplicación a los formularios

Los filtros se pueden utilizar para implementar todo o parte de los procesamientoos relativos a la gestión de formularios:

- Recuperación de los datos introducidos (funciones `filter_input` y `filter_input_array`).
- Verificación de los datos introducidos (filtros `FILTER_VALIDATE_*`).
- Procesamiento de problemas relativos a los datos introducidos (filtros `FILTER_SANITIZE_*`).

Cabe señalar que los filtros `FILTER_SANITIZE_*` efectúan transformaciones similares a las de las funciones `htmlspecialchars`, `html_entities` y `strip_tags`, presentadas en este capítulo en la sección Los problemas en los datos introducidos, y pueden reemplazarlas en nuestros ejemplos.


Ejemplo

```
<?php
// Inclusión del archivo que contiene las definiciones de nuestras
// funciones generales.
include('funciones.inc');
// Comprobar si la página se llama después de la validación del formulario
if (filter_has_var(INPUT_POST, 'ok')) {
    // Definir los filtros para los datos introducidos.
    $filtros =
        array
        (
            'apellido' => array('filter'=> FILTER_SANITIZE_STRING,
                               'flags' => FILTER_FLAG_ENCODE_HIGH +
                                   FILTER_FLAG_ENCODE_LOW)
        );
    // Recuperar los datos introducidos filtrados.
    $entrada = filter_input_array(INPUT_POST, $filtros);
    $apellido = $entrada['apellido'];
    // El valor introducido se vuelve a mostrar en el formulario y
    // en la página ...
}
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head><title>Entrada</title></head>
<body>
    <form action="entrada.php" method="post">
    <div>
        Apellidos:
        <input type="text" name="apellido"
            value="<?php echo $apellido; ?>" />
        <input type="submit" name="ok" value="OK" /><br />
        <?php echo $apellido; ?>
    </div>
    </form>
</body>
</html>
```

Entrada



Resultado



Nombre:

dijo: \"It's raining\"

En este ejemplo, el filtro `FILTER_SANITIZE_STRING` se utiliza para recuperar los datos introducidos, eliminar cualquier etiqueta y codificar los caracteres que pueden ser un problema en caso de presentación (en la página o en el formulario).

Ir a otra página al final del procesamiento

Al final del procesamiento del formulario, puede ser necesario mostrar otra página, la situación puede variar en función de si el formulario se procesa por el script que lo muestra o por un script independiente.

Variantes posibles

		Formulario procesado por	
		El script de presentación	Otro script
Resultado del procesamiento	OK	Ir a otra página	- Página ya correcta - Ir a otra página
	Problema	- Volver a mostrar el formulario con un mensaje - Ir a una página de error específica	- Volver a mostrar el formulario con un mensaje - Ir a otra página - Mostrar el error en la página actual

Es posible redirigir al usuario a otra página desde el script utilizando la función `header`.

La función `header` permite especificar una cadena que se envía como encabezado HTTP con la página HTML.

Vamos a utilizar el encabezado `location` que redirige la solitud a otra dirección.

Sintaxis de la directiva `location`

`location: URL absoluta o relativa`

Sintaxis con la función `header`

`header('location: URL absoluta o relativa')`

Ejemplos

```
// Redirección a un script PHP situado al mismo nivel.
header('location: error.php');
// Redirección hacia una página HTML situada en un subnivel.
header('location: ./error/entrada.htm');
// Redirección hacia otro sitio.
header('location: http://www.olivier-heurtel.fr');
```

El protocolo HTTP 1.1 requiere una URL absoluta en la directiva `location`. Para ello, puede utilizar las variables globales `$_SERVER['HTTP_HOST']` y `$_SERVER['PHP_SELF']` (véase capítulo Anexo - sección Variables PHP predefinidas).

Ejemplo

```
<?php
$url_relativa = 'error.php';
echo '$url_relativa = ', $url_relativa, '<br />';
echo '$_SERVER[\'HTTP_HOST\'] = ',
    $_SERVER['HTTP_HOST'], '<br />';
```

```

echo '$_SERVER[\'PHP_SELF\'] = ',
$_SERVER['PHP_SELF'],'<br />';
echo 'dirname($_SERVER[\'PHP_SELF\']) = ',
dirname($_SERVER['PHP_SELF'],'<br />';
$url_absoluta = 'http://' . $_SERVER['HTTP_HOST'] .
    rtrim(dirname($_SERVER['PHP_SELF']), '/\\') .
    '/' . $url_relativa;
echo '$url_absoluta = ', $url_absoluta, '<br />';
?>

```

Resultado

```

$url_relativa = error.php
$_SERVER['HTTP_HOST'] = xampp
$_SERVER['PHP_SELF'] = /eni/index.php
dirname($_SERVER['PHP_SELF']) = /eni
$url_absoluta = http://xampp/eni/error.php

```

Ejemplo simple de utilización de un script info.php

```

<?php
// Asignar un valor a $apellido si un número elegido aleatoriamente
// entre 0 y 1 es igual a 1.
$apellido = (rand(0,1)==1)?'Olivier':'';
// Comprobar si $apellido está en blanco.
if ($apellido == '') {
    // La variable $apellido está vacía, esto no es normal:
    // => redirigir al usuario a una página de error.
    header('location: error.htm');
    // Interrumpir la ejecución de este script.
    exit;
}
// La variable $apellido no está vacía, dejar seguir el script.
$mensaje = "¡Hola $nombre!"; // preparar un mensaje
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
    <head><title>Entrada</title></head>
    <body>
        <p><?php echo $mensaje; ?></p>
    </body>
</html>

```

Página error.htm

```

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
    <head><title>Error</title></head>
    <body>
        <div>
            El sitio no está disponible en este momento. Por favor, inténtelo más tarde.<br />
            Gracias por su comprensión.<br />
            <!-- Enlace para volver a intentarlo -->
            <a href="info.php">Intentar de nuevo</a>
        </div>
    </body>
</html>

```

Estadísticamente, la mitad del tiempo, la llamada del script `info.php` da el siguiente resultado:

```
¡Hola Olivier!
```

Y, por lo tanto, la mitad del tiempo, lo siguiente:

```
El sitio no está disponible en este momento. Por favor, inténtelo más tarde.  
Gracias por su comprensión.  
Intentar de nuevo
```

➤ Salvo en casos especiales, la función `header` debe llamarse antes de cualquier instrucción (PHP o HTML) que tenga el efecto de iniciar la construcción de la página HTML (es de alguna manera demasiado tarde para el encabezado). Un simple espacio, en el script, o en un script incluido (`require` o `include`) provoca el mismo error.

➤ Si la característica de almacenamiento en búfer de la página está activada con la directiva de configuración `output_buffering`, el resultado del script no se envía progresivamente, sino que se coloca en un búfer y luego se envía de golpe al final (o en porciones si el búfer es de tamaño limitado). En este caso, es posible utilizar la función `header` sin causar un error cuando el script empiece a construir la página.

PHP ofrece otras funciones relativas a los encabezados:

- `headers_list`: lista de encabezados de la respuesta (versión 5).
- `header_sent` permite comprobar si los encabezados ya han sido enviados.
- `get_headers`: lista de los encabezados reenviados por un servidor, para una URL determinada (versión 5).

Primer ejemplo de lógica de flujo de texto

Un script `entrada.php` proporciona la presentación inicial del formulario y su procesamiento. En caso de error, el formulario se propone de nuevo para su corrección (con los valores introducidos), acompañado de un mensaje de error. En el caso de éxito del procesamiento, se llama a otra página.

```
<?php  
// Incluir el archivo que contiene las definiciones de nuestras  
// funciones generales.  
include('funciones.inc');  
// Probar cómo se llama al script.  
if (isset($_POST['ok'])) {  
    // Procesamiento del formulario.  
    // Recuperar los datos introducidos en el formulario.  
    $apellido = trim($_POST['apellido']);  
    // Controlar los valores introducidos.  
    if ($apellido == '')  
    { $mensaje .= "El apellido es obligatorio.\n"; }  
    if (strlen($apellido) > 10)  
    { $mensaje .= "El apellido debe tener como máximo diez caracteres.\n"; }  
    // Comprobar si hay errores.  
    if ($mensaje == '') {  
        // Ningún error.  
        // Redirigir al usuario a otra página y detener  
        // la ejecución del script.
```

```

    header('location: inicio.php');
    exit;
} else {
    // Error.
    // Preparar el mensaje para mostrarlo.
    $mensaje = hacia_página($mensaje);
}
} else {
    // Presentación inicial.
    // En este sencillo ejemplo, nada que hacer.
}
// En el código HTML siguiente, inclusión de dos pequeñas porciones de
// código PHP para mostrar respectivamente el valor de los campos de
// entrada y el mensaje.
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
    <head><title>Entrada</title></head>
    <body>
        <form action="entrada.php" method="post">
            <div>
                Apellidos:
                <input type="text" name="apellido"
                    value="<?php echo hacia_formulario($apellido); ?>" />
                <input type="submit" name="ok" value="OK" /><br />
                <?php echo $mensaje; ?>
            </div>
        </form>
    </body>
</html>

```

Segundo ejemplo de lógica de flujo de texto

Un script `entrada.php` asegura la presentación inicial del formulario y un `scriptprocesamiento.php` el procesamiento. En caso de fallo, se muestra un mensaje de error y se pide al usuario que vuelva atrás. En caso de éxito del procesamiento, se muestra otra página.

Archivo `entrada.htm`

```

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
    <head><title>Entrada</title></head>
    <body>
        <form action="procesamiento.php" method="post">
            <div>
                Nombre:
                <input type="text" name="nombre" value="" />
                <input type="submit" name="ok" value="OK" />
            </div>
        </form>
    </body>
</html>

```

Script `procesamiento.php`

```

<?php
// Incluir el archivo que contiene las definiciones de nuestras

```

```
// funciones generales.
include('funciones.inc');
// Probar cómo se llama al script.
if (isset($_POST['ok'])) {
    // Procesamiento del formulario.
    // Recuperar los datos introducidos en el formulario.
    $apellido = trim($_POST['apellido']);
    // Controlar los valores introducidos.
    if ($apellido == '')
    { $mensaje .= "El apellido es obligatorio.\n"; }
    if (strlen($apellido) > 10)
    { $mensaje .= "El apellido debe tener como máximo diez caracteres.\n"; }
    // Comprobar si hay errores.
    if ($mensaje == '') {
        // Ningún error.
        // Redirigir al usuario a otra página y detener
        // la ejecución del script.
        header('location: inicio.php');
        exit;
    } else {
        //Error.
        //Preparar el mensaje para mostrarlo.
        $mensaje = hacia_pagina($mensaje);
    }
}

// En el código HTML siguiente, inclusión de una pequeña porción de
// código PHP para mostrar el mensaje.
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
    <head><title>Error</title></head>
    <body>
        <!-- Pequeño formulario que contiene un botón que permite
        ---- volver atrás (con JavaScript) para realizar correcciones.
        -->
        <form>
            <div>
                <?php echo $mensaje; ?><br />
                <input type="button" value="Corregir"
                    onClick="self.history.back()">
            </div>
        </form>
    </body>
</html>
```

Pueden existir otras lógicas de flujo de texto, la función `header` permite considerar varios escenarios (véase la documentación de PHP).

El resultado del procesamiento también se puede mostrar en otra ventana con el atributo `target` de la etiqueta `<form>`.

Ejemplo

```
<form action="procesamiento.php" method="post" target="procesamiento">
```

Si la ventana no existe, será creada por el navegador.

Introducción

1. Información general

La utilización de una base de datos SQL es a menudo esencial para implementar un sitio Web dinámico. De hecho, se trata de una forma estándar de almacenamiento de datos útiles para el sitio Web:

- lista de usuarios con sus preferencias
- catálogo de productos
- seguimiento de las transacciones realizadas

PHP ofrece soporte nativo para muchas bases de datos, como MySQL, Oracle, Microsoft SQL Server, Informix, Sybase. Además, PHP es compatible con ODBC (*Open DataBase Connectivity*) y, por tanto, puede acceder a cualquier base de datos compatible con ODBC.

Además, desde la versión 5 PHP viene con SQLite, una biblioteca que implementa un motor de base de datos SQL. SQLite se puede utilizar para almacenar datos en una base de datos SQL, sin tener que implementar la parte del servidor de la base de datos (como es el caso de MySQL, Oracle, etc.).

En este capítulo estudiaremos MySQL, Oracle y SQLite.

Normalmente, cuando se utiliza una base de datos, el script PHP necesita llevar a cabo una o varias de las siguientes tareas:

- conectarse y desconectarse.
- leer los datos (una o varias líneas).
- actualizar los datos (adición, modificación o supresión).

Estas diferentes tareas se tratan en este capítulo.



Se requieren conocimientos mínimos de SQL para abordar este capítulo.

Para los ejemplos que aquí se presentan, suponemos la existencia de una base de datos que contiene una tabla **ARTÍCULO** que presenta la siguiente estructura:

Columna	Contenido
identificador	Identificador del artículo (suministrado automáticamente por el servidor)
texto	Texto del artículo
precio	Precio del artículo

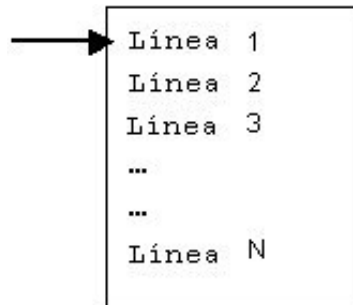
El contenido de esta tabla **ARTÍCULO** es el siguiente:

identificador	texto	precio
1	Albaricoques	35.5
2	Cerezas	48.9
3	Fresas	29.95

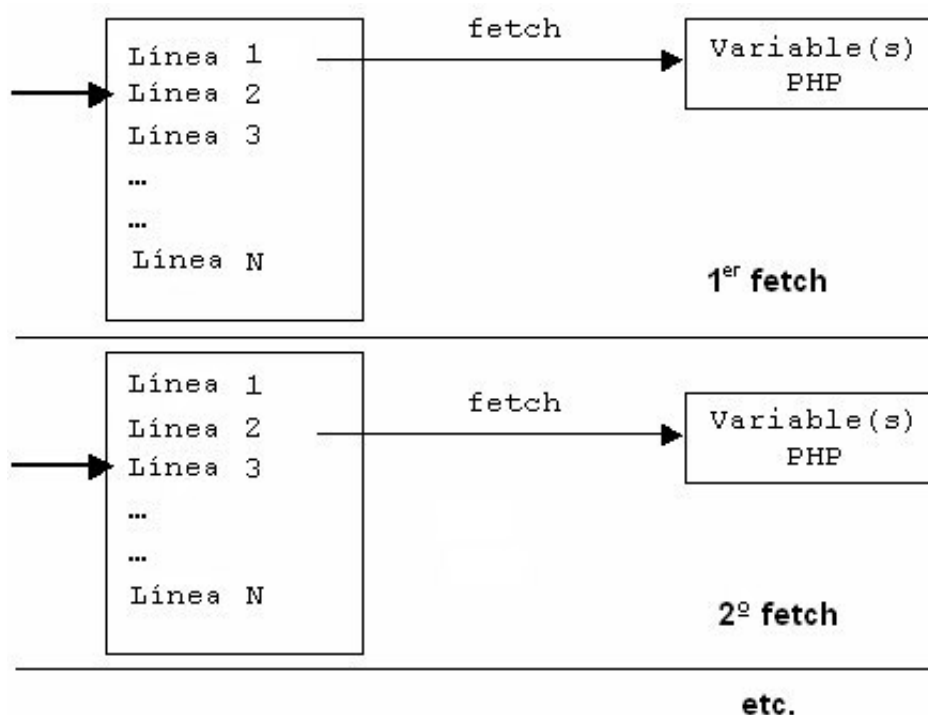
2. El concepto de fetch

Independientemente de la base de datos utilizada, la instrucción de ejecución de una consulta `SELECT` (para la lectura de datos) simplemente ejecuta la consulta; no se devuelven datos. Después de ejecutar la consulta, es necesario extraer las líneas del resultado: el concepto de "fetch".

Para resumir el funcionamiento, la instrucción de ejecución de una consulta `SELECT` identifica un resultado y coloca un puntero interno en la primera línea de este resultado:



La instrucción `FETCH` permite leer la línea actual del resultado, devolver los valores a las variables PHP y mover el puntero a la línea siguiente:



Utilización de MySQL

1. Preámbulo

Desde la versión 5, PHP ofrece dos extensiones para acceder a una base de datos MySQL:

- MySQL (prefijo `mysql_`)
- MySQLi (prefijo `mysqli_`)

La extensión MySQL es la extensión antigua, presente en versiones anteriores de PHP. Esta extensión se puede utilizar para acceder a cualquier versión de MySQL, pero no es compatible con las nuevas características de la versión 4.1 de MySQL.

Para utilizar las nuevas características de la versión 4.1 de MySQL, debe utilizar la extensión MySQLi, introducida en PHP 5. Esta extensión también se puede utilizar para acceder a una versión anterior de MySQL, siempre que no utilice características aparecidas en la versión 4.1 de MySQL.

Aparte de las nuevas características de la versión 4.1 de MySQL, ambas extensiones son muy similares en términos de funcionalidad. Muy a menudo ofrecen las mismas funciones, con una sintaxis idéntica o compatible. Salvo excepción, para pasar de usar MySQL a MySQLi, basta con reemplazar el prefijo `mysql_` en nombre de la función por el prefijo `mysqli_`.

No se recomienda utilizar la antigua extensión MySQL para nuevos desarrollos; en su lugar, se recomienda utilizar la extensión MySQLi.

Comenzaremos con la presentación en detalle del uso de la extensión MySQL. A continuación, estableceremos una correspondencia entre la extensión MySQLi y la extensión MySQL, centrándonos en las diferencias más importantes.

La extensión MySQLi se puede usar ya sea como procedimiento, o bien en forma de objeto.

En su forma orientada a objeto, la extensión MySQLi ofrece tres clases:

<code>mysqli</code>	Conexión entre PHP y MySQL.	Estas diferentes clases ofrecen métodos que permiten efectuar las diferentes acciones (ejecución de una consulta, recuperación del resultado, etc.).
<code>mysqli_stmt</code>	Consulta preparada.	
<code>mysqli_result</code>	Resultado de la ejecución de una consulta.	

En su forma de procedimiento, la extensión MySQLi ofrece funciones que permiten efectuar las mismas acciones. De forma transparente, varias de estas funciones devuelven o aceptan como parámetros objetos de tipo `mysqli` o `mysqli_result`.

En esta obra, presentamos únicamente la forma de procedimiento de la extensión MySQLi.

2. Conexión y desconexión

La función `mysqli_connect` permite establecer una conexión con una base de datos MySQL.

Sintaxis

```
objeto mysqli_connect([cadena host [, cadena usuario ],
cadena contraseña] [, cadena nombre base [,entero puerto]]])
```

Con

<code>host</code>	Nombre (o dirección IP) del host al que debe conectarse (máquina local por defecto).
<code>usuario</code>	Nombre del usuario que se utiliza para establecer la conexión. Valor predeterminado: el propietario del proceso del servidor Web.
<code>contraseña</code>	Contraseña que se utilizará para establecer la conexión. Valor predeterminado: cadena vacía (sin contraseña).
<code>nombre_base</code>	Base de datos MySQL seleccionada por defecto (ninguna por defecto).
<code>puerto</code>	Número del puerto para la conexión al servidor MySQL (puerto estándar 3306 por defecto).

La función `mysqli_connect` devuelve un identificador de conexión (objeto `mysqli`), o en caso de error, el valor `FALSE` acompañado de un aviso enviado a la pantalla (véase el capítulo Administrar los errores en un script PHP para manejar correctamente esta situación).

Varias directivas de configuración permiten definir los valores por defecto para los diferentes parámetros de la función `mysqli_connect` (ver la documentación).

Las conexiones abiertas en un script se cierran automáticamente al final del script, salvo por desconexión explícita con la función `mysqli_close` (ver más abajo).

Anteponer el nombre o la dirección IP del host por `p`: permite obtener un funcionamiento diferente y establecer una conexión "permanente" que no se cerrará al final del script y podrá reutilizarse en este script o en otro posterior.

La directiva de configuración `mysqli.allow_persistent` debe estar en `on` para poder abrir conexiones permanentes. Por otra parte, las directivas `mysqli.max_persistent` y `mysqli.max_links` permiten respectivamente limitar el número de conexiones permanentes

y el número total de conexiones.

La función `mysqli_close` permite cerrar una conexión en curso de script.

Sintaxis

```
booleano mysqli_close(objeto conexión)
```

conexión Identificador de conexión devuelto por la La
función `mysqli_connect`.

función `mysqli_close` devuelve TRUE en caso de éxito y FALSE en caso de error (acompañado de una alerta).

Ejemplo

```
<?php
// Conexión (utilización de los valores por defecto).
mysqli_$conexión =mysqli_connect();
If ($conexión) {
    Echo 'Conexión con éxito: ',
        'versión del servidor =', $conexión->server_info, '<br />';
} else {
    Echo 'Error en la conexión.<br />';
} ;
// Desconexión.
$ok = mysqli_close($conexión);
echo '!Desconexión: '$ok = ', ($ok)?"TRUE":"FALSE", "<br />\n";
?>
```

Resultado

```
Conexión con éxito: versión del servidor = 5.0.77
Desconexión: $ok = TRUE
```

Este ejemplo muestra claramente que el identificador de conexión devuelto por `mysqli_connect` es un objeto.

3. Leer los datos

Leer datos en una base de datos requiere tres operaciones:

- Seleccionar la base de datos a la que enviar la consulta (un servidor MySQL puede contener varias bases de datos independientes).
- Enviar, para su ejecución, una consulta (por ejemplo, `SELECT`) al servidor.
- Recuperar la fila o filas correspondientes a los resultados de la consulta.

Seleccionar una base de datos

La función `mysqli_connect` presentada anteriormente permite seleccionar una base de datos desde la conexión (cuarto parámetro de la función).

La función `mysqli_select_db` permite seleccionar o modificar la base de datos que se va a utilizar para una conexión determinada.

Sintaxis

```
booleano mysqli_select_db (objeto_conexión, cadena nombre_base)
```

Con

conexión Identificador de conexión devuelto por la La
función `mysqli_connect`.

nombre_base Nombre de la base de datos.

función `mysqli_select_db` devuelve TRUE en caso de éxito y FALSE en caso de error. La función `mysqli_select_db` no genera ninguna alerta en caso de error.

Ejemplo

```
<?php
// Conexión (utilización de los valores por defecto).
$conexión = mysqli_connect();
if (!$conexión) {
    exit('Error de conexión.');
```

```

$ok = mysqli_select_db($conexión, 'diane');
if ($ok) {
    echo 'Base de datos seleccionada.<br />';
} else {
    echo 'No se pudo seleccionar la base de datos.';
}
// Desconexión.
$ok = mysqli_close($conexión);
if ($ok) {
    echo 'Desconexión con éxito.';
} else {
    echo 'Error de desconexión.';
}
?>

```

Resultado

Conectado con éxito.
Base de datos seleccionada.
Desconexión con éxito.

Ejecutar una consulta

La función `mysqli_query` permite ejecutar una consulta en una base de datos.

Sintaxis

objeto `mysqli_query(objeto conexión, cadena consulta, [,entero modo])`

Con

conexión	Identificador de conexión devuelto por la función <code>mysqli_connect</code> .
consulta	Texto de la consulta que se va a ejecutar.
modo	Indica si el resultado debe ponerse en el búfer (constante <code>MYSQLI_STORE_RESULT</code> , valor predeterminado) o no (constante <code>MYSQLI_USE_RESULT</code>).

En el caso de una consulta SQL que devuelve un resultado (como `SELECT`, `SHOW` o `DESCRIBE`), la función `mysqli_query` devuelve un identificador de resultado de consulta, en caso de éxito, y `FALSE` en caso de error. Para el resto de sentencias SQL, esta función devuelve `TRUE` en caso de éxito

y `FALSE` en caso de error. La función `mysqli_query` no genera ninguna alerta en caso de error.

Ejemplo (con una consulta SELECT)

```

<?php
// Conexión (utilización de los valores por defecto).
$conexión = mysqli_connect();
if (! $conexión {
    exit('Error de conexión.');
```

Resultado

Ejecución con éxito.
Error de ejecución de la consulta.

En el caso de una consulta SQL que devuelve un resultado, la función `mysqli_query` ejecuta la consulta, indica si la consulta se ha

ejecutado con éxito, pero no devuelve datos. Es necesario extraer las líneas del resultado.

El tercer parámetro de la función `mysqli_query` indica si el resultado de la consulta se almacena en el búfer (constante `MYSQLI_STORE_RESULT`, valor predeterminado) o no (`MYSQLI_USE_RESULT`). Para consultas grandes, el uso de `MYSQLI_USE_RESULT` consume mucha menos memoria y permite extraer la primera línea mucho más rápido: no hay necesidad de esperar a que todo el resultado se almacene en el búfer. Sin embargo, después de ejecutar la consulta con `MYSQLI_USE_RESULT`, no es posible conocer el número de líneas del resultado, ni ejecutar otra consulta antes de haber extraído todas las líneas o haber liberado el resultado (función `mysqli_free_result`).

Conocer el número de líneas del resultado

La función `mysqli_num_rows` permite conocer el número de líneas del resultado.

Sintaxis

```
entero mysqli_num_rows(recurso resultado)
```

Con

resultado Identificador de resultado de consulta devuelto por la función `mysqli_query`.

Esta función se utiliza sólo para las consultas que devuelven un resultado (como `SELECT`, `SHOW` o `DESCRIBE`) y para las cuales el resultado se ha

almacenado en búfer (ver la función `mysqli_query`).

Ejemplo

```
<?php
// Conexión (utilización de los valores por defecto).
$conexión = mysqli_connect();
if (! $conexión) {
    exit('Error de conexión.');
```

```
}
// Selección de la base de datos.
$ok = mysqli_select_db($conexión, 'diane');
if (! $ok) {
    exit('No se pudo seleccionar la base de datos.');
```

```
}
// Ejecución de una consulta SELECT.
$consulta = mysqli_query($conexión, 'SELECT * FROM articulos');
if ($consulta === FALSE) {
    echo 'Error de ejecución de la consulta.','<br />';
} else {
    // Visualización del número de líneas del resultado.
    echo 'Número de artículos: ',mysqli_num_rows($consulta) ,','<br />';
}
```

```
// Ejecución de otra consulta SELECT.
$consulta =
    mysqli_query($conexión,'SELECT * FROM articulos WHERE precio > 40');
if ($consulta === FALSE) {
    echo 'Error de ejecución de la consulta.','<br />';
} else {
    // Visualización del número de líneas del resultado.
    echo
        'Número de artículos cuyo precio es superior a 40: ',
        mysqli_num_rows($consulta),
        '<br />';
}
```

```
// Desconexión.
$ok = mysqli_close($conexión);
?>
```

Resultado

```
Número de artículos: 4
Número de artículos cuyo precio es superior a 40: 1
```

Leer el resultado de la consulta

El resultado de la ejecución de una consulta que devuelve un resultado (como `SELECT`, `SHOW` o `DESCRIBE`) se puede leer por las funciones `mysqli_fetch_array`, `mysqli_fetch_assoc`, `mysqli_fetch_object` o `mysqli_fetch_row` y `mysqli_fetch_all`.

Las funciones `mysqli_fetch_array`, `mysqli_fetch_assoc`, `mysqli_fetch_object` y `mysqli_fetch_row` leen la línea actual del resultado y avanzan el puntero a la línea siguiente; estas funciones se diferencian en el tipo de datos utilizado para devolver el resultado.

Sintaxis

```
matriz mysqli_fetch_array(objeto resultado [, entero tipo])
```

```
matriz mysqli_fetch_assoc(objeto resultado)
objeto mysqli_fetch_object(objeto resultado)
matriz mysqli_fetch_row(objeto resultado)
```

Con

resultado Identificador de resultado de consulta devuelto por la función `mysqli_query`. Las

tipo Tipo de resultado idéntico a una de las siguientes constantes: `MYSQLI_ASSOC`, `MYSQLI_NUM`, `MYSQLI_BOTH` (valor predeterminado).

funciones `mysqli_fetch_array`, `mysqli_fetch_assoc` y `mysqli_fetch_row` devuelven la línea actual del resultado en la forma de una matriz, cada línea de la matriz corresponde a una columna del resultado. La función `mysqli_fetch_object` devuelve la línea actual en la forma de un objeto.

Si no hay ninguna línea que leer en el resultado, estas funciones devuelven `NULL`.

Para la función `mysqli_fetch_assoc`, la matriz es una matriz asociativa cuya clave es el nombre de la columna. Para la función `mysqli_fetch_row`, se trata de una matriz con índices enteros, el índice 0 corresponde a la primera columna, el índice 1 a la segunda, etc. Por último, para la función `mysqli_fetch_array`, el tipo de la matriz depende del segundo parámetro:

<code>MYSQLI_NUM</code>	Matriz con índices enteros (como la función <code>mysqli_fetch_row</code>).	<u>Ejemplo</u>
<code>MYSQLI_ASSOC</code>	Matriz asociativa (como la función <code>mysqli_fetch_assoc</code>).	<u>Resultado de un fetch (lectura con diferentes funciones)</u>
<code>MYSQLI_BOTH</code> (valor predeterminado)	Cada columna está presente dos veces, una vez con un índice entero correspondiente a su posición y una vez con una clave correspondiente a su nombre.	La

Consulta	SELECT * FROM artículos		
Columnas	identificador	texto	precio
1a línea del resultado	1	Albaricoques	35.5

mysqli_fetch_row o mysqli_fetch_array (..., MYSQL_NUM)		mysqli_fetch_assoc o mysqli_fetch_array (... MYSQL_ASSOC)		mysqli_fetch_array (... MYSQL_BOTH)	
Clave	Valor	Clave	Valor	Clave	Valor
0	1	identificador	1	identificador	1
1	Albaricoques	texto	Albaricoques	0	1
2	35.5	precio	35.5	texto	Albaricoques
				1	Albaricoques
				precio	35.5
				2	35.5

función `mysqli_fetch_object` devuelve un objeto, con un atributo por columna, el nombre del atributo corresponde al nombre de la columna. Esta función ofrece parámetros adicionales que permiten precisar el nombre de la clase que se va a instanciar y pasar parámetros al constructor de la clase (ver la documentación).

Ejemplo

Atributo	Valor
identificador	1
texto	Albaricoques
precio	35.5

En caso de utilización de un alias de columna en la consulta `SELECT` (ejemplo `SELECT AVG(precio-sin-iva) precio_medio FROM colección`), es el alias de columna el que se utiliza como clave o nombre de atributo.

Ejemplo

```
<?php
// Inclusión del archivo que contiene la definición de
// la función 'mostrar_matriz'.
require('funciones.inc');
// Conexión (utilización de los valores por defecto).
$conexión = mysqli_connect();
if (! $conexión) {
```

```

    exit('Error de conexión.');
```

Resultado

mysqli_fetch_row

```

0 = 1
1 = Albaricoques
2 = 35.5
```

mysqli_fetch_assoc

```

identificador = 2
texto = Cerezas
precio = 48.9
```

mysqli_fetch_array

```

0 = 3
identificador = 3
1 = Fresas
texto = Fresas
2 = 29.95
precio = 29.95
```

mysqli_fetch_object

```

$línea->identificador = 4
$línea->texto = Melocotones
$línea->precio = 37.2
```

Quinto fetch: nada más

En este ejemplo permite observar:

- Los diferentes modos de recuperación de una línea de resultado.
- El hecho de que a cada *fetch*, el puntero interno avance, y que el *fetch* siguiente devuelva, por tanto, la siguiente línea, hasta haber pasado por todas las líneas.

La función `mysqli_fetch_all` lee todas las líneas del resultado en una matriz.

Sintaxis

```
matriz mysqli_fetch_all(objeto resultado[, entero tipo])
```

Con

resultado Identificador de resultado de la consulta devuelto por la función `mysqli_query`.

La

tipo Tipo de resultado igual a una de las siguientes constantes:MYSQLI_ASSOC, MYSQLI_NUM (valor predeterminado),MYSQLI_BOTH.

función `mysqli_fetch_all` devuelve una matriz multidimensional. La matriz principal es una matriz con índices enteros que contiene una línea para cada línea del resultado. La matriz secundaria contiene los valores de las columnas; el tipo de esta matriz depende del segundo parámetro:

MYSQLI_NUM	Matriz con índices enteros.
MYSQLI_ASSOC	Matriz asociativa.
MYSQLI_BOTH	Cada columna se presenta dos veces, una vez con un índice entero correspondiente a su posición y una vez con una clave correspondiente a su nombre.

Ejemplo

```
if (!$conexión) {
    exit('Error de conexión.');
```

```
}
// Selección de la base de datos.
$ok = mysqli_select_db($conexión,'diane');
if (!$ok) {
    exit('No se pudo seleccionar la base de datos.');
```

```
}
// Ejecución de una consulta.
$sql = 'SELECT * FROM artículos';
$consulta = mysqli_query($conexión,$sql);
// Fetch de todas las líneas:
// - parámetros predeterminados = MYSQLI_NUM
$resultado = mysqli_fetch_all($consulta);
mostrar_matriz($resultado,'mysqli_fetch_all($consulta)');
```

```
// Determinación del número de líneas leídas.
$numero = mysqli_num_rows($consulta);
echo "Número líneas en el resultado";
// Nueva ejecución de la consulta.
$consulta = mysqli_query($conexión,$sql);
// Fetch de todas las líneas:
// - MYSQLI_ASSOC
$resultado = mysqli_fetch_all($consulta,MYSQLI_ASSOC);
mostrar_matriz($resultado,'mysqli_fetch_all($consulta,MYSQLI_ASSOC)');
```

```
// Determinación del número de líneas leídas.
$numero = mysqli_num_rows($consulta);
echo "Número líneas en el resultado";
// Nueva ejecución de la consulta.
$consulta = mysqli_query($conexión,$sql);
// Fetch de todas las líneas:
// - MYSQLI_BOTH
$resultado = mysqli_fetch_all($consulta,MYSQLI_BOTH);
mostrar_matriz($resultado,'mysqli_fetch_all($consulta,MYSQLI_BOTH)');
```

```
// Determinación del número de líneas leídas.
$numero = mysqli_num_rows($consulta);
echo "Número líneas en el resultado";
// Desconexión.
$ok = mysqli_close($conexión);
?>
```

```
<?php
// Inclusión del archivo que contiene
// la función 'mostrar_matriz'.
require('funciones.inc');
// Conexión (utilización de los valor
$conexión = mysqli_connect();
```

Resultado

```
mysqli_fetch_all($consulta)
0 =
  0 = 1
  1 = Albaricoques
  2 = 35.5
1 =
  0 = 2
  1 = Cerezas
  2 = 48.9
2 =
  0 = 3
  1 = Fresas
  2 = 29.95
3 =
  0 = 4
  1 = Melocotones
  2 = 37.2
4 líneas en el resultado

mysqli_fetch_all($consulta,MYSQLI_ASSOC)
0 =
  identificador = 1
```

```

    texto = Albaricoques
    precio = 35.5
1 =
    identificador = 2
    texto = Cerezas
    precio = 48.9
2 =
    identificador = 3
    texto = Fresas
    precio = 29.95
3 =
    identificador = 4
    texto = Melocotones
    precio = 37.2
4 líneas en el resultado

mysqli_fetch_all($consulta,MYSQLI_BOTH)
0 =
    0 = 1
    identificador = 1
    1 = Albaricoques
    texto = Albaricoques
    2 = 35.5
    precio = 35.5
1 =
    0 = 2
    identificador = 2
    1 = Cerezas
    texto = Cerezas
    2 = 48.9
    precio = 48.9
2 =
    0 = 3
    identificador = 3
    1 = Fresas
    texto = Fresas
    2 = 29.95
    precio = 29.95
3 =
    0 = 4
    identificador = 4
    1 = Melocotones
    texto = Melocotones
    2 = 37.2
    precio = 37.2
4 líneas en el resultado

```

En caso de utilización de un identificador de resultado no válido, las diferentes funciones `mysqli_fetch_*` devuelven NULL y muestran una alerta del tipo:

```

Warning: mysqli_fetch_assoc() expects parameter 1 to be
mysqli_result, integer given in /app/scripts/index.php
on line 5

```

¿Qué método utilizar?

Para leer una sola línea, las funciones `mysqli_fetch_array`, `mysqli_fetch_assoc` y `mysqli_fetch_row` son válidas desde el punto de vista del rendimiento. Las funciones `mysqli_fetch_assoc` y `mysqli_fetch_object` permiten utilizar el nombre de las columnas de la consulta y hacer el código más legible.

Para leer todas las líneas, se puede utilizar la función `mysqli_fetch_all` si desea pasar por una matriz y la estructura de la matriz le sirve. Pero si no es el caso, puede utilizar una de las otras funciones `mysqli_fetch_*` y un bucle.

Ejemplo de código para la lectura de una línea

Un primer tipo de lectura consiste a menudo en leer una sola línea de información en una matriz o varias matrices con uniones: obtención de información sobre el usuario que acaba de conectarse, ficha de descripción de un artículo...

Ejemplo

```

<?php
// Inclusión del archivo que contiene la definición de
// la función 'mostrar_matriz'.
require('funciones.inc');
// Inicializar las variables.
$identificador = '';
$texto = '';
$precio = '';
$mensaje = '';

```

```
// Probar cómo se llama al script.
if (! empty($_POST)) {
    // Procesamiento del formulario ...
    // Recuerde: cuando un formulario sólo tiene un campo de texto
    // y el usuario pulsa intro o return, no se considera que
    // se ha hecho clic en el botón de validación. Para saber
    // si el formulario se ha enviado, es necesario comprobar si
    // $_POST (o $_GET) está vacía o no.
    // Recuperar los valores introducidos.
    $identificador = $_POST['identificador'];
    // Controlar los valores introducidos.
    if ($identificador == '') {
        $mensaje .= "El identificador es obligatorio.\n";
    }
    if (preg_match('/^[0-9]+$/', $identificador) == 0) {
        $mensaje .= "El identificador debe ser un número.\n";
    }
    // Comprobar si hay errores en este punto.
    if ($mensaje == '') {
        // Ningún error.
        // Conexión y selección de la base.
        $conexión = mysqli_connect();
        $ok = mysqli_select_db($conexión, 'diane');
        // Ejecutar la consulta de selección.
        $consulta = "SELECT * FROM articulos " .
            "WHERE identificador = $identificador";
        $resultado = mysqli_query($conexión, $consulta);
        // Fetch si la consulta se ha ejecutado correctamente.
        if ($resultado) {
            $articulo = mysqli_fetch_assoc($resultado);
        }
        // Comprobar el resultado.
        if (! $resultado) { // consulta no OK
            $mensaje .= "Error.\n";
        } elseif (! $articulo) { // resultado vacío
            $mensaje .= "Ningún artículo para este identificador.\n";
        } else { // OK
            // Recuperar la información que se va a mostrar.
            $texto = $articulo['texto'];
            $precio = $articulo['precio'];
            // Formato.
            $texto = hacia_página($texto);&nbs

```

Índice

Información

[Título, autor...](#)

Introducción

[Objetivo del libro](#)[Breve historia de PHP](#)[¿Dónde conseguir PHP?](#)[Convenciones de escritura](#)

Información general sobre PHP

[Variables, constantes, tipos y matrices](#)[Operadores](#)[Estructuras de control](#)[Funciones y clases](#)[Gestión de formularios](#)[Acceder a las bases de datos](#)[Administrar las sesiones](#)[Enviar un correo electrónico](#)[Gestión de archivos](#)[Administrar los errores en un script PHP](#)[Anexo](#)

¿Dónde conseguir PHP?

Existen muchas páginas Web dedicadas al lenguaje PHP. Permiten descargar el lenguaje, consultar ejemplos de secuencias de comandos (scripts) o participar en las discusiones de sus foros:

Dirección	Contenido
www.php.net	Página Web oficial de PHP que permite descargar las fuentes y un manual de referencia en línea muy práctico. Puede escribir <code>www.php.net/nombre_funcion</code> para acceder directamente a la ayuda en línea para una función de PHP.
www.php-hispano.net	Página Web española dedicada al lenguaje PHP. Contiene enlaces para cursos, ejemplos, artículos, tutoriales, software... En resumen, una página muy completa que debe ir directamente a los Favoritos.
www.easypHP.org	Página Web disponible en inglés y francés que ofrece gratuitamente un producto instalable (Easy PHP) para la plataforma Windows. Este producto incluye: un servidor Apache, PHP y MySQL. Una vez descargado el producto, haga doble clic en el archivo ejecutable para instalar los diversos elementos. Cinco minutos más tarde, su entorno PHP ya está operativo. Esta página Web es indispensable para aquellos que deseen montar una configuración rápida plenamente operativa en Windows.
www.apachefriends.org/en/xampp.html	Otro sitio que ofrece un producto instalable (XAMPP) en diferentes plataformas (Linux, Windows, Solaris, Mac OS X). Este producto incluye, entre otras cosas, un servidor Apache, PHP y MySQL. Además, su instalación es muy sencilla y muy rápida.
www.zend.com	Página Web oficial del motor de script Zend que también ofrece la descarga de temas clásicos, ejemplos, foro...
Página Información	<i>Página Web de Ediciones ENI donde se pueden descargar los ejemplos descritos en este libro.</i>

Evidentemente, esta lista no es exhaustiva, pero todos los sitios de la misma proporcionan enlaces a otros sitios. ¡No lo dude y siga navegando!

Utilización de Oracle

1. Preámbulo

La extensión "Oracle OCI8", al contrario de lo que su nombre pueda hacernos pensar, se puede utilizar para acceder a cualquier versión de Oracle (8i, 9i, 10g y 11g). Esta extensión es muy completa y muy potente; permite utilizar los tipos LOB y ROWID y establecer vínculos entre las variables PHP y las variables en la consulta SQL (proceso de "bind variable").

- En la versión 5, muchas de las funciones de esta biblioteca han cambiado de nombre. Excepto el prefijo, las funciones tienen ahora nombres idénticos a funciones similares de otras bases de datos. Los antiguos nombres todavía se pueden utilizar por razones de compatibilidad con versiones anteriores, aunque estén obsoletos. Este libro presenta las funciones con sus nombres en la versión 5, pero recuerda el nombre antiguo.

2. Entorno NLS

El entorno NLS (*National Language Support*), utilizado en la ejecución de las consultas, se define por medio de las variables de entorno habituales colocadas en el servidor que ejecuta PHP:

- NLS_LANG
- NLS_DATE_FORMAT...

Si estas variables de entorno no se han establecido, se toma el valor predeterminado de Oracle Server.

Esto puede dar lugar a diferencias de funcionamiento de un mismo programa entre dos entornos. Por ejemplo, si recupera en una consulta una columna de tipo DATE, se realiza una conversión en cadena automáticamente (ya que PHP no admite el tipo DATE como tal) según el parámetro NLS_DATE_FORMAT activo; dependiendo de la configuración y el entorno, la cadena recuperada puede tener diferentes formatos (DD/MM/AAAA, DD-MON-YY...).

Problemas similares ocurren con los datos numéricos que también se convierten en una cadena con un separador decimal y un separador de grupo que puede variar en función del entorno.

Si no controla el entorno, es posible actuar al nivel de código PHP para obtener un código portátil de un entorno a otro:

- bien efectuando sistemáticamente conversiones explícitas en las instrucciones SQL (TO_CHAR(SYSDATE,'DD/MM/YYYY'), TO_DATE('31/08/2001','DD/MM/YYYY'), TO_CHAR(precio,'9999.99'), TO_NUMBER('123.45','999.99') por ejemplo);
- bien ejecutando, después de cada apertura de sesión, consultas ALTER SESSION (ALTER SESSION SET NLS_DATE_FORMAT = 'DD/MM/YYYY' y/o ALTER SESSION SET NLS_NUMERIC_CHARACTERS = ', ' por ejemplo).

3. Conexión y desconexión

La función oci_connect (antiguamente ociLogon) permite abrir una conexión con una base de datos de Oracle.

Sintaxis simplificada

```
recurso oci_connect(cadena usuario, cadena contraseña,
[, cadena nombre_de_servicio [, cadena juego_caracteres]])
```

Con

usuario	Nombre del usuario que se utiliza para establecer la conexión.
contraseña	Contraseña que se utiliza para establecer la conexión.
nombre_de_servicio	Nombre de un servicio de red válido, definido en el archivo tnsnames.ora, sobre el que establecer la conexión. El método llamado Easy Connect, que apareció en la versión 10 de Oracle, también se puede utilizar ([//]host[:puerto] [/nombre_servicio]). Si no se especifica, la función oci_connect utiliza la variable de entorno ORACLE_SID para la conexión a una instancia local, o la variable TWO_TASK (linux) o LOCAL (Windows), para la conexión a una instancia remota.
juego_caracteres	Juego de caracteres utilizado para la sesión (por defecto determinado desde la variable de entorno NLS_LANG).

función oci_new_connect es la misma que la de la función oci_connect.

Ejemplo

```
<?php
// Conexión con un nombre de servicio de red.
$id1 = oci_connect('demeter','demeter','diane');
echo "\$id1 = \$id1<br />\n";
// Nueva conexión con los mismos parámetros que 1.
$id2 = oci_connect('demeter','demeter','diane');
echo "\$id2 = \$id2 (<b> \$id1</b>)<br />\n";
// Nueva conexión con los mismos parámetros que 1,
```

La función oci_connect devuelve un identificador de conexión en caso de éxito o, en caso de error, el valor FALSE acompañado de un mensaje de alerta enviado a la presentación (véase capítulo Administrar los errores en un script PHP).

En el mismo script, llamar dos veces a la función oci_connect con los mismos parámetros no abre dos conexiones diferentes: en la segunda llamada, se devuelve el identificador de la conexión ya abierta (lo que no era el caso en la versión 5.0 y podría llevar a confusión). Esto es importante, ya que las instrucciones emitidas en la primera conexión y las emitidas en la segunda se realizan realmente en la misma transacción: un COMMIT o ROLLBACK en una de las dos conexiones tendrá efecto en ambas.

Si es necesario, la función oci_new_connect antiguamente ocilogon se puede utilizar para abrir dos conexiones independientes en el mismo script con los mismos parámetros. La sintaxis de la

```
// pero utilizando oci_new_connect.
$id3 = oci_new_connect('demeter','demeter','diane');
echo "\$id3 = $id3 (<b>< \ $id1</b><br />\n";
?>
```

Resultado

```
$id1 = Resource id #3
$id2 = Resource id #3 (= $id1)
$id3 = Resource id #4 (<> $id1)
```

Las conexiones abiertas en un script se cierran automáticamente al final del script, excepto por desconexión explícita antes con la función `oci_close` (véase más adelante).

La función `oci_pconnect` (antiguamente `ocipLogon`) permite obtener una acción diferente y establecer una conexión "permanente" que no se cerrará al final del script y se podrá reutilizar en este script o en otro posteriormente.

La sintaxis es idéntica a la sintaxis de la función `oci_connect`.

En la llamada a la función `oci_pconnect` en un script, PHP comprueba si ya se ha abierto una conexión permanente con los mismos parámetros (usuario, contraseña y nombre de servicio): en caso afirmativo, se devuelve el identificador de esta conexión; de lo contrario, se establece una nueva conexión (y esta conexión no se cerrará al final del script, lo que permite su reutilización posterior).

La función `oci_close` (antiguamente `ocilogoff`) permite cerrar una conexión en curso de script.

Sintaxis

```
booleano oci_close(recurso conexión)
```

Con

conexión Identificador de conexión devuelto por las
funciones `oci_connect` o `oci_new_connect`.

En teoría, la función `oci_close` devuelve TRUE en caso de éxito y FALSE (de hecho NULL) en caso de error (acompañado de una alerta).

La función `oci_close` no tiene efecto sobre una conexión permanente.

➤ Todas las conexiones no permanentes se cierran automáticamente al final del script.

Ejemplo

```
<?<?php
// Conexión.
$conexión = oci_connect('demeter','demeter','diane');
echo "\$conexión = $conexión<br />\n";
// Desconexión.
$ok = oci_close($conexión);
echo ' $ok = ',var_dump($ok);
?> >
```

Resultado

```
$conexión = Resource id #2
$ok = bool(true)
```

4. Leer los datos

Leer datos en una base de datos requiere tres operaciones:

- Enviar una consulta (en cso de una instrucción SELECT) al servidor para su análisis (paso de "parse").
- Ejecutar la consulta.
- Recuperar la o las líneas correspondiente al resultado de la consulta.

Analizar una consulta

La función `oci_parse` (antiguamente `ociparse`) permite enviar una consulta para su análisis en el servidor.

Sintaxis

```
recurso oci_parse(recurso conexión, cadena consulta)
```

Con

conexión Identificador de conexión devuelto por la
función `oci_connect` (`oci_new_connect` o `oci_pconnect`).

consulta Texto de la consulta que se va a analizar.

La función `oci_parse` devuelve un identificador de cursor en caso de éxito o FALSE en caso de error. La función devuelve un identificador de cursor incluso aunque el cursor no sea válido, a causa de una consulta incorrecta; un cursor no válido no se detecta hasta el momento de la ejecución (véase más adelante).

Ejemplo

```
<?php
// Conexión.
$conexión = oci_connect('demeter','demeter','diane');
// Definición de la consulta.
```

```
$consulta = 'SELECT * FROM artículos';
// Análisis de la consulta.
$cursor1 = oci_parse($conexión,$consulta);
echo "\$cursor1 = $cursor1<br />\n";
// Definición de una consulta incorrecta (tabla inexistente).
$consulta = 'SELECT * FROM artículo';
// Análisis de la consulta.
$cursor2 = oci_parse($conexión,$consulta);
echo "\$cursor2 = $cursor2<br />\n";
?>
```

Resultado

```
$cursor1 = Resource id #3
$cursor2 = Resource id #4
```

Ejecutar una consulta

La función `oci_execute` (antiguamente `ociexecute`) permite ejecutar una consulta anteriormente enviada para su análisis en el servidor.

Sintaxis

```
booleano oci_execute(recurso cursor[, entero modo])
```

cursor Identificador del cursor que se va a ejecutar.

modo Indica si se debe ejecutar un COMMIT automático o no (sin objeto para una consulta SELECT (véase sección Utilización de Oracle - Actualizar los datos)).

La función `oci_execute` devuelve TRUE en caso de éxito y FALSE en caso de error (acompañado de una alerta).

Ejemplo

```
<?php
// Conexión.
```

```
$conexión = oci_connect('demeter','demeter','diane');
// Definición de la consulta.
$consulta = 'SELECT * FROM artículos';
// Análisis de la consulta.
$cursor1 = oci_parse($conexión,$consulta);
echo "\$cursor1 = $cursor1<br />\n";
// Ejecución de la consulta.
$ok1 = oci_execute($cursor1);
echo '$ok1 = ', ($ok1)?"TRUE":"FALSE", "<br />\n";
// Definición de una consulta incorrecta (tabla inexistente).
$consulta = 'SELECT * FROM artículo';
// Análisis de la consulta.
$cursor2 = oci_parse($conexión,$consulta);
echo "\$cursor2 = $cursor2<br />\n";
// Ejecución de la consulta.
$ok2 = oci_execute($cursor2);
echo '$ok2 = ', ($ok2)?"TRUE":"FALSE", "<br />\n";
?>
```

Resultado

```
$cursor1 = Resource id #5
$ok1 = TRUE
$cursor2 = Resource id #6
Warning: oci_execute(): ORA-00942:
table or view does not exist in /app/scripts/index.php on
line 18
$ok2 = FALSE
```

En el siguiente punto veremos cómo recuperar, en el script, un código o un mensaje de error.

Como hemos indicado en la introducción, la función `oci_execute` ejecuta la consulta, indica si la consulta se ha ejecutado con éxito, pero no reenvía ningún dato. Es necesario extraer las líneas del resultado.

Conocer el número de líneas del resultado

A diferencia de MySQL, no hay manera, antes de la lectura, de conocer el número de líneas del resultado.

Por contra, en el transcurso de la lectura (ver sección siguiente), la función `oci_num_rows` (antiguamente `ociRowCount`) permite conocer el número total de líneas leídas en este punto.

Sintaxis

```
entero oci_num_rows(recurso cursor)
```

cursor Identificador del cursor en el que se ha ejecutado la consulta.

Si la ejecución de la consulta ha fallado, la función `oci_num_rows` devuelve cero.

Leer el resultado de la consulta

Le resultado de la ejecución (con éxito) de una consulta SELECT puede ser leído por las funciones `oci_fetch_array`, `oci_fetch_assoc`, `oci_fetch_object`, `oci_fetch_row` y `oci_fetch_all`.

Las funciones `oci_fetch_array`, `oci_fetch_assoc`, `oci_fetch_object` y `oci_fetch_row` leen una línea de resultado y hacen avanzar el puntero a la línea siguiente; estas funciones se diferencian en el tipo de datos utilizado para devolver el resultado. Aparecieron en la versión 5 y tienen una sintaxis muy similar a la de las funciones equivalentes de MySQL. En las versiones anteriores, la función `OCIFetchInto` se puede utilizar en su lugar; esta última todavía existe por razones de compatibilidad con versiones anteriores, pero queda obsoleta.

Sintaxis

```
matriz oci_fetch_array(recurso cursor [, entero tipo])
matriz oci_fetch_assoc(recurso cursor)
objeto oci_fetch_object(recurso cursor)
matriz oci_fetch_row(recurso cursor)
```

Con

cursor Identificador del cursor anteriormente ejecutado.

tipo Tipo de resultado definido por una combinación de las siguientes constantes:

- OCI_BOTH (valor por defecto)
- OCI_ASSOC
- OCI_NUM
- OCI_RETURN_NULLS
- OCI_RETURN_LOBS

El valor por defecto es OCI_BOTH+OCI_RETURN_NULLS.

Las

funciones `oci_fetch_array`, `oci_fetch_assoc` y `oci_fetch_row` devuelven la línea actual del resultado en forme de una matriz, cada línea de la matriz corresponde a una columna del resultado. La función `oci_fetch_object` devuelve la línea actual en forme de un objeto.

Si no hay ninguna línea que leer en el resultado, estas funciones devuelven FALSE.

Para la función `oci_fetch_assoc`, se trata de una matriz asociativa cuya clave es el nombre de la columna (en mayúsculas). Para la función `oci_fetch_row`, se trata de una matriz con índices enteros, el índice 0 corresponde a la primera columna, el índice 1 a la segunda, etc. Por último, para la función `oci_fetch_array`, el tipo del matriz depende del segundo parámetro:

OCI_NUM Matriz con índices enteros (como la función `oci_fetch_row`).

OCI_ASSOC Matriz asociativa (como la función `oci_fetch_assoc`).

OCI_BOTH Las dos a la vez (valor por defecto). Cada columna está presente dos veces. Equivalente a OCI_NUM + OCI_ASSOC.

Por defecto, las cuatro funciones hacen figurar en el resultado las columnas que tienen un valor NULL. Este comportamiento se puede modificar en el caso de la utilización de la función `oci_fetch_array` no implementando la constante OCI_RETURN_NULLS en el valor pasado en el parámetro `tipo`. En este caso, las columnas con un valor NULL no figuran en el resultado (pero los índices se

conservan en el caso en el que se utilice una matriz con índices enteros).

Las constantes se pueden añadir para precisar varios comportamientos. Par ejemplo, OCI_ASSOC+OCI_RETURN_NULLS permite obtener una matriz asociativa con conservación de los valores NULL.

Ejemplo

Consulta	SELECT * FROM articulos		
Columnas	identificador	texto	precio
1a línea del resultado	1	Albaricoques	

Resultado de una llamada a `oci_fetch_array` con:

OCI_NUM + OCI_RETURN_NULLS		OCI_ASSOC + OCI_RETURN_NULLS		OCI_BOTH + OCI_RETURN_NULLS	
Índice/Clave	Valor	Índice/Clave	Valor	Índice/Clave	Valor
0	1	IDENTIFICADOR	1	0	1
1	Albaricoques	TEXTO	Albaricoques	IDENTIFICADOR	1
2		PRECIO		1	Albaricoques
				TEXTO	Albaricoques
				2	
				PRECIO	



Utilizar OCI_RETURN_NULLS solo vuelve a utilizar OCI_BOTH + OCI_RETURN_NULLS.

En caso de utilización de un alias de columna en la consulta SELECT (ejemplo SELECT AVG(precio) **precio_medio** FROM artículos), es el alias de columna (en mayúsculas, excepto si se especifica entre comillas, en cuyo caso se respetan las mayúsculas y minúsculas) el que se utiliza como clave o nombre de atributo.

Ejemplo

```
<?php
// Inclusión del archivo que contiene la definición de
// mostrar_matriz.
include('funciones.inc');
// Conexión.
$conexión = oci_connect('demeter','demeter','diane');
// Definición de la consulta.
$consulta = 'SELECT * FROM articulos';
// Análisis de la consulta.
$cursor = oci_parse($conexión,$consulta);
// Ejecución de la consulta.
$ok = oci_execute($cursor);
// Primer fetch con oci_fetch_row.
$línea = oci_fetch_row($cursor);
mostrar_matriz($línea,'oci_fetch_row');
// Determinación del número de líneas leídas en este punto.
$numero = oci_num_rows($cursor);
echo "<br />$numero línea leída en este punto";
// Segundo fetch con oci_fetch_assoc.
$línea = oci_fetch_assoc($cursor);
mostrar_matriz($línea,'oci_fetch_assoc');
// Determinación del número de líneas leídas en este punto.
$numero = oci_num_rows($cursor);
```



```

echo "<br />$Número líneas leídas en este punto";
// Tercer fetch con oci_fetch_array:
// - sin segundo parámetro = OCI_BOTH.
$línea = oci_fetch_array($cursor);
mostrar_matriz($línea,'oci_fetch_array');
// Determinación del número de líneas leídas en este punto.
$Número = oci_num_rows($cursor);
echo "<br />$Número líneas leídas en este punto";
// Cuarto fetch con oci_fetch_object.
$línea = oci_fetch_object($cursor);
echo "<p><b>oci_fetch_object</b><br />";
echo "\$línea->IDENTIFICADOR = $línea->IDENTIFICADOR<br />";
echo "\$línea->TEXTO = $línea->TEXTO<br />";
echo "\$línea->PRECIO = $línea->PRECIO<br />";
// Determinación del número de líneas leídas en este punto.
$Número = oci_num_rows($cursor);
echo "<br />$Número líneas leídas en este punto";
// Quinto fetch de nuevo sin parámetros:
// - normalmente, ninguna línea.
$línea = oci_fetch_array($cursor);
if (! $línea) {
    echo "<p><b>Quinto fetch: nada más</b><br />";
}
// Determinación del número de líneas leídas en este punto.
$Número = oci_num_rows($cursor);
echo "<br />$Número líneas leídas en este punto";
?>

```

Resultado (suponiendo que la columna precio esté siempre rellenada)

```

oci_fetch_row
0 = 1
1 = Albaricoques
2 = 35,5
1 línea leída en este punto
oci_fetch_assoc
IDENTIFICADOR = 2
TEXTO = Cerezas
PRECIO = 48,9
2 líneas leídas en este punto
oci_fetch_array
0 = 3
IDENTIFICADOR = 3
1 = Fresas
TEXTO = Fresas
2 = 29,95
PRECIO = 29,95
3 líneas leídas en este punto
oci_fetch_object
$línea->IDENTIFICADOR = 4
$línea->TEXTO = Melocotones
$línea->PRECIO = 37,2
4 líneas leídas en este punto
Quinto fetch: nada más
4 líneas leídas en este punto

```

Este ejemplo permite ver:

- Los diferentes modos de recuperación de una línea de resultado.
- El hecho de que a cada fetch, el puntero interno avance, y que el fetch siguiente devuelva, por tanto, la siguiente línea, hasta haber pasado por todas las líneas.

Las funciones devuelven FALSE y muestran una alerta en los siguientes casos:

- Cursor no analizado (sin llamada a la función `oci_parse`).

Warning: oci_fetch_array() expects parameter 1 to be resource, null given in **/app/scripts/index.php** on line **13**

- Cursor no ejecutado (sin llamada a la función `oci_execute`).

Warning: oci_fetch_array(): ORA-24374: define not done before fetch or execute and fetch in **/app/scripts/index.php** on line **13**

- Cursor ejecutado con error.

Warning: oci_fetch_array(): ORA-24374: define not done before fetch or execute and fetch in **/app/scripts/index.php** on line **13**

La función `oci_fetch_all` (antiguamente `ocifetchstatement`) lee todas las líneas del resultado en una matriz.

Sintaxis

```

entero oci_fetch_all(recurso cursor, matriz resultado, [entero ignorar[,
entero número[, entero tipo]])

```

Con

cursor	Identificador del cursor anteriormente ejecutado.
resultado	Variable que contendrá el resultado de retorno. El contenido inicial de la variable se sobrescribe.
ignorar	Número de líneas del inicio del resultado que no se devuelven. 0 por defecto: el resultado comienza en la primera línea.

La función `oci_fetch_all` devuelve el número de líneas del resultado o FALSE si el resultado no presenta ninguna línea (o en caso de error).

número	Número de líneas a devolver, a partir de la primera línea devuelta. -1 por defecto: se devuelven todas las líneas.
tipo	Tipo de resultado definido por una combinación de las siguientes constantes: OCI_FETCHSTATEMENT_BY_ROW OCI_FETCHSTATEMENT_BY_COLUMN (valor predeterminado) OCI_ASSOC (valor predeterminado) OCI_NUM

⚠ ¡Atención! Las líneas ignoradas, incluso si no están presentes en el resultado, se extraen igualmente (fetch). Sin embargo, la función `oci_num_rows` devuelve el número total de líneas extraídas. En consecuencia, cuando se ignoran las líneas en `oci_fetch_all`, el resultado devuelto por `oci_fetch_all` y `oci_num_rows` es diferente (la diferencia es igual al número de líneas ignoradas).

La matriz devuelta en resultado por la función `oci_fetch_all` es una matriz multidimensional. Por defecto, la matriz principal es una matriz asociativa que contiene una línea para cada columna de la consulta: la clave es igual al nombre de la columna, en mayúsculas, y el valor es una matriz con índices enteros que contiene los valores de la columna para el conjunto de líneas del resultado (índice 0 para la 1a línea, 1 para la 2a, etc.).

La estructura de la matriz se puede cambiar especificando una o varias de las siguientes constantes por el parámetro tipo:

OCI_FETCHSTATEMENT_BY_ROW	La matriz principal contiene una línea para cada línea de la consulta; esta matriz es obligatoriamente con índices enteros. La matriz secundaria contiene una línea para cada columna de la consulta.	No es posible combinar entre ellas las
OCI_FETCHSTATEMENT_BY_COLUMN (valor predeterminado)	La matriz principal contiene una línea para cada columna de la consulta. La matriz secundaria contiene una línea para cada línea de la consulta; esta matriz es obligatoriamente con índices enteros.	
OCI_ASSOC (valor predeterminado)	La matriz de las columnas es una matriz asociativa.	
OCI_NUM	La matriz de las columnas es una matriz con índices enteros.	

constantes OCI_FETCHSTATEMENT_BY_ROW y OCI_FETCHSTATEMENT_BY_COLUMN, ni OCI_ASSOC y OCI_NUM.

La función `oci_fetch_all` devuelve los valores NULL.

En caso de utilización de un alias de columna en la consulta SELECT (ejemplo `SELECT AVG(precio) precio_medio FROM artículos`), es el alias de columna (en mayúsculas, excepto si se especifica entre comillas, en cuyo caso se respetan las mayúsculas y minúsculas) el que se utiliza como clave.

Ejemplo

```
<?php
// Inclusión del archivo que contiene la definición de
// mostrar_matriz.
include('funciones.inc');
// Conexión.
$conexión = oci_connect('demeter','demeter','diane');
// Definición de la consulta.
$query = 'SELECT * FROM artículos';
// Análisis de la consulta.
$cursor = oci_parse($conexión,$query);
// Ejecución de la consulta.
$ok = oci_execute($cursor);
// Fetch de todas las líneas:
// - parámetros por defecto.
$numéro = oci_fetch_all($cursor,$resultado);
mostrar_matriz($resultado,
    'oci_fetch_all($cursor,$resultado)');
echo ($numéro)?"$numéro líneas en el resultado":"FALSE";
// Otra determinación del número de líneas leídas.
$numéro = oci_num_rows($cursor);
echo "<br />$numéro líneas en el resultado";
// Ejecución de la consulta.
$ok = oci_execute($cursor);
// Fetch de todas las líneas:
// - resultado parcial: ignorar la 1a línea
// - dos líneas en total.
$numéro = oci_fetch_all($cursor,$resultado,1,2);
mostrar_matriz($resultado,
    'oci_fetch_all($cursor,$resultado,1,2)');
echo ($numéro)?"$numéro líneas en el resultado":"FALSE";
// Otra determinación del número de líneas leídas.
$numéro = oci_num_rows($cursor);
echo "<br />$numéro líneas en el resultado";
// Ejecución de la consulta.
$ok = oci_execute($cursor);
// Fetch de todas las líneas:
// - resultado parcial: dos líneas en total;
// - presentación por línea.
$numéro = oci_fetch_all($cursor,$resultado,
    0,2,OCI_FETCHSTATEMENT_BY_ROW);
mostrar_matriz($resultado,
    'oci_fetch_all($cursor,$resultado,
    '0,2,OCI_FETCHSTATEMENT_BY_ROW)');
```

```

echo ($numero)?"Número líneas en el resultado":'FALSE';
// Otra determinación del número de líneas leídas.
$numero = oci_num_rows($cursor);
echo "<br />$numero líneas en el resultado";
// Ejecución de la consulta.
$ok = oci_execute($cursor);
// Fetch de todas las líneas:
// - resultado parcial: dos líneas en total;
// - presentación por línea;
// - matriz numérica para las columnas.
$numero = oci_fetch_all($cursor,$resultado,
    0,2,OCI_FETCHSTATEMENT_BY_ROW+OCI_NUM);
mostrar_matriz($resultado,
    'oci_fetch_all($cursor,$resultado,'.
        '0,2,OCI_FETCHSTATEMENT_BY_ROW+OCI_NUM)');
echo ($numero)?"Número líneas en el resultado":'FALSE';
// Otra determinación del número de líneas leídas.
$numero = oci_num_rows($cursor);
echo "<br />$numero líneas en el resultado";
// Definición de una consulta que no devuelve ninguna línea.
$consulta = "SELECT * FROM artículos WHERE 0=1";
// Análisis de la consulta.
$cursor = oci_parse($conexion,$consulta);
// Ejecución de la consulta.
$ok = oci_execute($cursor);
// Fetch de todas las líneas:
// - parámetros por defecto.
$numero = oci_fetch_all($cursor,$resultado);
mostrar_matriz($resultado,'Ningún resultado: por columna');
echo ($numero)?"Número líneas en el resultado":'FALSE';
// Otra determinación del número de líneas leídas.
$numero = ociRowCount($cursor);
echo "<br />$numero línea en el resultado";
// Ejecución de la consulta.
$ok = oci_execute($cursor);
// Fetch de todas las líneas:
// - presentación por línea.
$numero = oci_fetch_all($cursor,$resultado,0,-
1,OCI_FETCHSTATEMENT_BY_ROW);
mostrar_matriz($resultado,'Ningún resultado: por línea');
echo ($numero)?"Número líneas en el resultado":'FALSE';
// Otra determinación del número de líneas leídas.
$numero = ociRowCount($cursor);
echo "<br />$numero línea en el resultado";
?>

```

Resultado

```

oci_fetch_all($cursor,$resultado)
IDENTIFICADOR =
    0 = 1
    1 = 2
    2 = 3
    3 = 4
TEXT0 =
    0 = Albaricoques
    1 = Cerezas
    2 = Fresas
    3 = Meloc

```

Utilización de SQLite

1. Preámbulo

SQLite es una biblioteca que implementa un motor de base de datos SQL.

SQLite se puede utilizar para almacenar datos en una base de datos SQL, sin tener que implementar la parte del servidor de la base de datos (como es el caso de MySQL, Oracle, etc.).

SQLite lee y escribe directamente en los archivos de la base de datos.

Para obtener más información sobre SQLite, visite su página Web: <http://sqlite.org/>

PHP ofrece dos extensiones para trabajar con una base de datos SQLite:

- SQLite para la versión 2 de SQLite
- SQLite3 para la versión 3 de SQLite

En la versión 5.4 de PHP, la extensión SQLite3 está activada por defecto y la extensión SQLite ya sólo está disponible mediante PECL (*PHP Extension Community Library*).

En este libro, presentaremos únicamente la extensión SQLite3.

SQLite3 es una extensión orientada a objetos que ofrece 3 clases:

- SQLite3: conexión entre PHP y la base de datos
- SQLite3Stmt: gestión de consultas preparadas
- SQLite3Result: gestión de resultados

2. Abrir y cerrar una base de datos

La apertura de una base de datos SQLite se efectúa durante la instanciación de un objeto de la clase SQLite3.

Sintaxis del método constructor de la clase SQLite3

```
SQLite3::__construct (cadena archivo[, entero modo[, cadena cifrado]])
```

Con

archivo	Ruta al archivo de la base de datos.
modo	Modo de apertura de la base de datos: SQLITE3_OPEN_READONLY : apertura en sólo lectura. SQLITE3_OPEN_READWRITE : apertura en lectura/escritura. SQLITE3_OPEN_CREATE : creación de la base de datos si no existe. El valor por defecto es: SQLITE3_OPEN_READWRITE SQLITE3_OPEN_CREATE.
cifrado	Clave opcional utilizada para el cifrado y descifrado de los datos.

En caso de error, se eleva una excepción.

Ejemplo

```
<?php
// Apertura en lectura/escritura de una base de datos existente.
$dbase = new SQLite3('/app/sqlite/diane.dbf');
var_dump($base);
?>
```

Resultado

```
object(SQLite3)#1 (0) { }
```

Las bases de datos abiertas en un script se cierran automáticamente al final del script, salvo por desconexión explícita con el método `SQLite3::close`.

El método `SQLite3::close` permite cerrar una base de datos.


Sintaxis

```
SQLite3::close()
```

El método `SQLite3::close` devuelve `TRUE` en caso de éxito y `FALSE` en caso de error.

Ejemplo

```
<?php
// Apertura en lectura/escritura de una base de datos que ya existe.
$dbase = new SQLite3('/app/sqlite/diane.dbf');
// Cierre de la base de datos.
$ok = $base->close();
?>
```

 Existe también un método `open()` que permite abrir una base de datos SQLite en un objeto ya instanciado: este método ofrece la misma sintaxis que el método constructor de la clase.

3. Leer los datos

Para leer los datos de una base de datos es necesario realizar dos operaciones:

- Ejecutar una consulta (en este caso `SELECT`) en la base de datos.
- Recuperar la fila o filas correspondientes a los resultados de la consulta.

Ejecutar una consulta

El método `SQLite3::query` permite ejecutar una consulta en una base de datos.

Sintaxis

```
SQLite3Result SQLite3::query(cadena consulta)
```

Con

`consulta` Texto de la consulta que se va a ejecutar.

En el caso de una sentencia `SELECT`, el método `SQLite3::query` devuelve un objeto de la

clase `SQLite3Result` (resultado de consulta) en caso de éxito y `FALSE` en caso de error (y se muestra un mensaje de advertencia); para el resto de sentencias SQL, este método debe devolver `TRUE` en caso de éxito, pero también devuelve un objeto de la clase `SQLite3Result`.

Ejemplo

```
<?php
// Apertura en lectura/escritura de una base de datos que ya existe.
$dbase = new SQLite3('/app/sqlite/diane.dbf');
// Definición de la consulta.
$query = 'SELECT * FROM artículos';
// Ejecución de la consulta.
$resultado1 = $dbase->query($query);
echo '$resultado1 = ',
    var_dump($resultado1,"<br />\n");
// Definición de una consulta incorrecta.
$query = 'SELECT * FROM artículo';
// Ejecución de la consulta.
$resultado2 = $dbase->query($query);
echo '$resultado2 = ',
    var_dump($resultado2,"<br />\n");
?>
```

Resultado

```
$resultado1 = object(SQLite3Result)#3 (0) { }
Warning: SQLite3::query(): Unable to prepare statement: 1,
no such table: article
in /app/scripts/index.php on line 13
$resultado2 = bool(false)
```

Conocer el número de líneas del resultado

A diferencia de MySQL, no hay ningún medio, antes de la lectura, de conocer el número de líneas del resultado. Para conocer el número de líneas en el resultado, es necesario extraer las líneas y contarlas.

Leer el resultado de la consulta

El resultado de la ejecución (icon éxito!) de una consulta puede ser leído por el método `SQLite3Result::fetchArray`.

Sintaxis

```
matriz SQLite3Result::fetchArray(entero tipo)
```

Con

tipo Tipo de resultado idéntico a una de las siguientes constantes: `SQLITE_ASSOC`, `SQLITE_NUM`, `SQLITE_BOTH` (valor por defecto).

El método `SQLite3Result::fetchArray` devuelve la línea actual del resultado en forma de una matriz; cada línea de la matriz corresponde a una columna del resultado.

Si no hay ninguna línea que leer en el resultado, la función devuelve `FALSE`.

El tipo de matriz (numérica o asociativa) depende del segundo parámetro:

SQLITE3_NUM	Matriz con índices enteros.
SQLITE3_ASSOC	Matriz asociativa.
SQLITE3_BOTH (valor predeterminado)	Las dos a la vez. Cada columna está presente dos veces.

Ejemplo

Consulta	SELECT * FROM artículos		
Columnas	identificador	texto	precio
1a línea del resultado	1	Albaricoques	35.5

Resultado de un fetch con:

fetch_Array (...,SQLITE3_NUM)		fetch_Array (...,SQLITE3_ASSOC)		fetch_Array (...,SQLITE3_BOTH)	
Índice/Clave	Valor	Índice/Clave	Valor	Índice/Clave	Valor
0	1	identificador	1	identificador	1
1	Albaricoques	texto	Albaricoques	0	1
2	35.5	precio	35.5	texto	Albaricoques
				1	Albaricoques
				precio	35.5
				2	35.5

En caso de utilización de un alias de columna en la consulta SELECT (por ejemplo SELECT AVG(precio) **precio_medio** FROM artículos), es el alias de columna el que se utiliza como clave o nombre de atributo.

Ejemplo

```
<?php
// Inclusión del archivo que contiene la definición de
// mostrar_matriz.
include('funciones.inc');
// Apertura.
$dbase = new SQLite3('/app/sqlite/diane.dbf');
// Definición de la consulta.
$query = 'SELECT * FROM artículos';
// Ejecución de la consulta.
$resultado = $dbase->query($query);
// Primer fetch sin segundo parámetro.
$linea = $resultado->fetchArray();
mostrar_matriz($linea,'sin parámetro');
// Segundo fetch con SQLITE3_BOTH.
$linea = $resultado->fetchArray(SQLITE3_BOTH);
mostrar_matriz($linea,'SQLITE3_BOTH');
// Tercer fetch con SQLITE3_ASSOC.
$linea = $resultado->fetchArray(SQLITE3_ASSOC);
mostrar_matriz($linea,'SQLITE3_ASSOC');
// Cuarto fetch con SQLITE3_NUM.
$linea = $resultado->fetchArray(SQLITE3_NUM);
mostrar_matriz($linea,'SQLITE3_NUM');
```

```
// Quinto fetch de nuevo sin parámetro:
// - normalmente, ninguna línea.
$línea = $resultado->fetchArray();
if (! $línea) {
    echo "<p><b>Quinto fetch: nada más</b>";
}
?>
```

Resultado

sin parámetro

```
0 = 1
identificador = 1
1 = Albaricoques
texto = Albaricoques
2 = 35.5
precio = 35.5
```

SQLITE3_BOTH

```
0 = 2
identificador = 2
1 = Cerezas
texto = Cerezas
2 = 48.9
precio = 48.9
```

SQLITE3_ASSOC

```
identificador = 3
texto = Fresas
precio = 29.95
```

SQLITE3_NUM

```
0 = 4
1 = Melocotones
2 = 37.2
```

Quinto fetch: nada más

Este ejemplo permite ver:

- Los diferentes modos de recuperación de una línea de resultado.
- En cada fetch, el puntero interno avanza, y el fetch siguiente devuelve, por tanto, la siguiente línea, hasta haber pasado por todas las líneas.

Ejemplo de código para la lectura de una línea

Un primer tipo de lectura consiste a menudo en leer una sola línea de información en una matriz (o varias matrices con uniones): información sobre el usuario que acaba de conectarse, ficha de información de un artículo...

Ejemplo

```
<?php
// Incluir un archivo que contiene las diferentes funciones
// generales.
require('funciones.inc');
// Inicializar las variables.
$identificador = '';
$texto = '';
$precio = '';
$mensaje = '';
// Probar cómo se llama al script.
```



```

if (! empty($_POST)) {
    // Procesamiento del formulario ...
    // Recuerde: cuando un formulario sólo tiene un campo de texto
    // y el usuario pulsa intro o return, no se considera que
    // se ha hecho clic en el botón de validación. Para saber
    // si el formulario se ha enviado, es necesario comprobar si
    // $_POST (o $_GET) está vacía o no.
    // Recuperar los valores introducidos.
    $identificador = $_POST['identificador'];
    // Controlar los valores introducidos.
    if ($identificador == '') {
        $mensaje .= "El identificador es obligatorio.\n";
    }
    if (preg_match('/^[0-9]+$/', $identificador) == 0) {
        $mensaje .=
            "El identificador debe ser un número.\n";
    }
    // Comprobar si hay errores en este punto.
    if ($mensaje == '') {
        // Ningún error.
        // Apertura de la base.
        $base = new SQLite3('/app/sqlite/diane.dbf');
        // Ejecutar la consulta de selección.
        $consulta = "SELECT * FROM artículos
            WHERE identificador = $identificador";
        $resultado = $base->query($consulta);
        // Fetch si la consulta se ha ejecutado correctamente.
        if ($resultado) {
            $artículo = ($resultado->fetchArray());
        }
        // Comprobar el resultado.
        if (! $resultado) { // consulta no OK
            $mensaje .= "Error.\n";
        } elseif (! $artículo) { // resultado vacío
            $mensaje .=
                "Ningún artículo para este identificador.\n";
        } else { // OK
            // Recuperar la información que se va a mostrar.
            $texto = $artículo['texto'];
            $precio = $artículo['precio'];
            // Formato.
            $texto = hacia_página($texto);
            $precio = number_format($precio,2,',', ' ');
            $precio = hacia_página($precio);
        }
    }
    // Comprobar si hay errores en este punto.
    if ($mensaje != '') {
        // Error.
        // Preparar el mensaje para mostrarlo.
        $mensaje = hacia_página($mensaje);
        // Asegurarse de que la información que se va a
        // mostrar está vacía.
        $texto = '';
        $precio = '';
    }
}
// Presentación de la página ...
?>
<?php echo '<?xml version="1.0" encoding="UTF-8"?>','\n'; ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"

```

```

"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head><title>Consulta de un artículo</title></head>
<body>
  <!-- Formulario de búsqueda (;muy simple!) -->
  <form action="consulta-articulo.php" method="post">
    <div>Identificador del artículo:
    <input type="text" name="identificador"
      value="<?php echo hacia_formulario($identificador) ?>" />
    <input type="submit" name="ok" value="OK" /></div>
  </form>
  <!-- Resultado de la búsqueda -->
  <table border="0" cellpadding="4">
    <tr><td><u>Texto:</u></td>
      <td><?php echo $texto; ?></td></tr>
    <tr><td><u>Precio:</u></td>
      <td><?php echo $precio; ?></td></tr>
  </table>
  <div><?php echo $mensaje; ?></div>
</body>
</html>

```

Resultado (después de introducir la información y hacer clic en el botón OK)

Identificador del artículo: 1

Texto: Albaricoques

Precio: 35.50

OK

Es relativamente fácil escribir una función genérica que permite leer una línea.

Ejemplo

```

<?php
function db_leer_línea($consulta) {
  // La variable $ok se utiliza para saber
  // si todo se transmite correctamente.
  // Abrir la base de datos.
  $ok = (bool) ($base = new SQLite3('/app/sqlite/diane.dbf'));
  // Ejecutar la consulta y comprobar el resultado para
  // asignar la variable $ok.
  if ($ok) {
    $ok = ( ($resultado = $base->query($consulta)) != FALSE );
  }
  // Leer la línea.
  if ($ok) {
    $línea = $resultado->fetchArray();
  }
  // Devolver $línea o FALSE en caso de error.
  return ($ok)?$línea:FALSE;
}
?>

```

Es posible (deseable) mejorar la gestión de errores (ver sección Utilización de SQLite - Gestión de errores).

Ejemplo de utilización

```

<?php
// Incluir el archivo que contiene las diferentes funciones
//generales.
require('funciones.inc');
// Definir la consulta.
$consulta = 'SELECT * FROM artículos WHERE identificador = 1';
// Leer el resultado.
$línea = db_leer_línea($consulta);
if ($línea) {
    echo "{$línea['texto']} - {$línea['precio']}";
}
?>

```

Resultado (excepto error)

Albaricoques - 35.5

Ejemplo de código para la lectura de todas las líneas

Un segundo tipo de lectura consiste a menudo en mostrar una lista de elementos extraídos de la base (lista de usuarios, lista de artículos...).

Para permitir al usuario actuar en la lista, es posible integrar un formulario o enlaces con la tabla HTML.

Ejemplo

```

<?php
// Incluir un archivo que contiene las diferentes funciones
// generales.
require('funciones.inc');
// Inicializar la variable de mensaje.
$mensaje = '';
// Abrir la base de datos.
$ok = (bool) ($base = new SQLite3('/app/sqlite/diane.dbf'));
// Ejecutar la consulta de selección:
$consulta = 'SELECT * FROM artículos';
$resultado = $base->query($consulta);
// Comprobar el resultado.
if (! $resultado) { // consulta no OK
    // Mensaje de error.
    $mensaje .= "Error.\n";
    // Preparar el mensaje para su presentación.
    $mensaje = hacia_página($mensaje);
}
// Presentación de la página ...
?>
<?php echo '<?xml version="1.0" encoding="UTF-8"?>','\n"; ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
    <head><title>Lista de artículos</title></head>
    <body>
        <!-- construcción de una tabla HTML en el interior
        ++++ de un formulario -->
        <form action="lista-articulos.php" method="post">
        <table border="1" cellpadding="4" cellspacing="0">
        <!-- línea de título -->
        <tr align="center">
        <th>Texto</th><th>Precio</th><th>Caso</th><th>Enlace</th>

```

```

</tr>
<?php
// Código PHP para las líneas de la matriz.
if ($resultado) { // Si hay un resultado que mostrar.
    // Bucle de fetch.
    $numero_líneas = 0;
    while ($artículo = $resultado->fetchArray()) {
        $numero_líneas++;
        // Formato de los datos.
        $artículo['texto'] = hacia_página($artículo ['texto']);
        $artículo['precio'] = hacia_página(
            number_format($artículo['precio'],2,',', ' '));
        // Generación de la línea de la tabla HTML:
        // - una casilla de verificación en una columna;
        // - un enlace en otra columna.
        printf(
            '<tr><td>%s</td><td>%s</td><td>%s</td><td>%s</td></tr>',
            $artículo['texto'],
            $artículo['precio'],
            "<input type=\"checkbox\" name=\"elección[]\"
              value=\"{$artículo['identificador']}\" />",
            "<a href=\"javascript:alert({$artículo['identificador']})\">
              action<a>");
    } // while
    // Si el resultado está vacío, preparar un mensaje.
    if ($numero_líneas == 0) {
        $mensaje = hacia_página('Ningún artículo en la base.');
```

Resultado


- Presentación inicial de la página:

Texto	Precio	Casilla	Enlace
Albaricoques	35,50	<input type="checkbox"/>	acción
Cerezas	48,90	<input type="checkbox"/>	acción
Fresas	29,95	<input type="checkbox"/>	acción
Melocotones	37,20	<input type="checkbox"/>	acción

- Después de hacer clic en un enlace:

Texto	Precio	Casilla	Enlace
Albaricoques	35,50	<input checked="" type="checkbox"/>	acción
Cerezas	48,90	<input type="checkbox"/>	acción
Fresas	29,95	<input checked="" type="checkbox"/>	acción
Melocotones	37,20	<input type="checkbox"/>	acción

Mensaje de página ...

 2

- Después de hacer clic en el botón **Acción**:

Texto	Precio	Casilla	Enlace
Albaricoques	35,50	<input type="checkbox"/>	acción
Cerezas	48,90	<input type="checkbox"/>	acción
Fresas	29,95	<input type="checkbox"/>	acción
Melocotones	37,20	<input type="checkbox"/>	acción

Identificador(es) marcado(s): 1+3

En el enlace, en lugar de código JavaScript, es posible poner una URL real y encadenar con otra página (véase el capítulo Administrar las sesiones para comprender cómo colocar un parámetro en la URL para poder transmitir información, en este caso el identificador elegido, a otra página).

En cuanto al procesamiento del formulario, todo es posible.

Una vez más, para la lectura de varias líneas, se puede utilizar una función genérica.

Ejemplo

```
<?php
function db_leer_líneas_en_matriz($consulta) {
    // La variable $ok se utiliza para saber
    // si todo se transmite correctamente.
    // Abrir la base de datos.
    $ok = (bool) ($base = new SQLite3('/app/sqlite/diane.dbf'));
    // Ejecutar la consulta y probar el resultado para mostrar la variable $ok.
    if ($ok) {
        $ok = ( ($resultado = $base->query($consulta)) != FALSE );
    }
    // Leer el resultado y almacenarlo en una matriz.
    if ($ok) {
        while ($línea = $resultado->fetchArray()) {
            $matriz[] = $línea;
        }
    }
    // Devolver $matriz o FALSE en caso de error.
    return ($ok)?$matriz:FALSE;
}
?>
```

Utilización

```
<?php
// Incluir el archivo que contiene las diferentes funciones generales.
require('funciones.inc');
// Definir la consulta.
$consulta = 'SELECT * FROM artículos';
// Leer el resultado en una matriz.
$artículos = db_leer_líneas_en_matriz($consulta);
// Mostrar el primer artículo leído.
if ($artículos) {
    echo count($artículos), ' artículos<br />';
    echo 'Primer artículo: ';
    echo $artículos[0]['texto'], ' - ', $artículos[0]['precio'];
}
?>
```

Resultado (excepto error)

```
4 artículos
Primer artículo: Albaricoques - 35.5
```

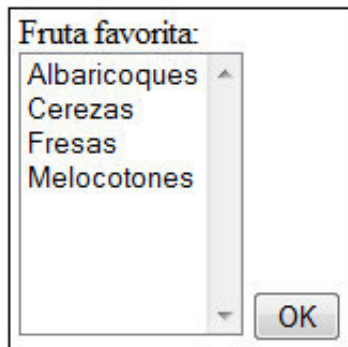
Esta función genérica se puede utilizar, por ejemplo, para construir una lista <select> en un formulario.

Ejemplo

```
<?php
// Incluir el archivo que contiene las diferentes funciones
// generales
require('funciones.inc');
// Carga de la lista de frutas
$consulta =
```

```
'SELECT identificador,texto FROM artículos ORDER BY texto';
$frutas_del_mercado = db_leer_líneas_en_matriz($consulta);
?>
<?php echo '<?xml version="1.0" encoding="UTF-8"?>','\n"; ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head><title>Lista de artículos</title></head>
<body>
<!-- construcción del formulario -->
<form action="lista-seleccionar.php" method="post">
<div>
Fruta preferida:<br />
<select name="frutas[]" multiple size="8">
<?php
// Código PHP que genera la parte dinámica del formulario.
// Examinar la lista que se va a mostrar.
foreach($frutas_del_mercado as $fruta) {
    // Generar la etiqueta 'opción' con la variable
    // el identificador para el atributo 'value' y el
    // texto para el texto mostrado en la lista.
    echo "<option value=\"{$fruta['identificador']}\">
        {$fruta['texto']}\n";
}
?>
</select>
<input type="submit" name="ok" value="OK" /><br />
</div>
</form>
</body>
</html>
```

Resultado



4. Actualizar los datos

Actualizar los datos consiste en ejecutar las consultas INSERT (creación), UPDATE (modificación) o DELETE (eliminación).

La ejecución de este tipo de consulta se puede efectuar con el método `SQLite3::query`, como para una consulta `SELECT`. Sin embargo, existe otro método `SQLite3::exec`, especialmente previsto para ejecutar una consulta que no devuelve ningún resultado y cuyo uso se recomienda.

Sintaxis

```
Booleano SQLite3::exec(cadena consulta)
```

Con

consulta Texto de la consulta que se va a ejecutar.

El método `SQLite3::query` devuelve `TRUE` en caso de éxito y `FALSE` en caso de error (mostrando un mensaje de alerta).

Además, dos métodos interesantes son: `SQLite3::changes` y `SQLite3::lastInsertRowId`

El método `SQLite3::changes` permite conocer el número de líneas afectadas (insertadas, modificadas o eliminadas) por la última consulta `INSERT`, `UPDATE` o `DELETE` ejecutada en una base de datos.

Sintaxis

```
entero SQLite3::changes()
```

Si la última consulta ha fallado, el método `SQLite3::changes` devuelve 0.

El método `SQLite3::lastInsertRowId` devuelve el valor del último identificador generado por una consulta `INSERT` en una base de datos, para una columna declarada en `INTEGER PRIMARY KEY`.

Sintaxis

```
entero SQLite3::lastInsertRowId()
```

Si no se ha generado ningún identificador automáticamente, el método `SQLite3::lastInsertRowId` devuelve 0.



La consulta `INSERT` no tiene por qué ser la última consulta ejecutada en la base de datos.

Ejemplo

```
<?php
// Inclusión del archivo que contiene la definición de las
// funciones generales.
require('funciones.inc');
// Definición de una pequeña función para mostrar la lista
// de artículos.
function MostrarArticulos() {
    $consulta = 'SELECT * FROM artículos';
    $articulos = db_leer_líneas_en_matriz($consulta);
    if ($articulos) {
        echo '<b>Lista de artículos:</b><br />';
        foreach($articulos as $a) {
            echo "$a[identificador] - $a[texto] - $a[precio]<br />";
        }
    } else {
        echo "<b>Más artículos.</b><br />";
    }
}
// Abrir la base.
$dbase = new SQLite3('/app/sqlite/diane.dbf');
// Visualización de control.
```



```

MostrarArticulos();
// Consulta INSERT.
$consulta = "INSERT INTO articulos(texto,precio)" .
    "VALUES('Peras',29.9)";
$resultado = $base->exec($consulta);
$identificador =
    $base->lastInsertRowID(); // recuperar el identificador
echo "Identificador del nuevo artículo = $identificador.<br />";
// Consulta UPDATE.
$consulta = 'UPDATE articulos SET precio = precio * 1.1' .
    'WHERE precio < 40';
$resultado = $base->exec($consulta);
$numero = $base->changes();
echo "$numero artículo(s) añadido(s).<br />";
// Consulta DELETE.
$consulta = 'DELETE FROM articulos WHERE precio > 40';
$resultado = $base->exec($consulta);
$numero = sqlite_changes($base);
echo "$numero artículo(s) eliminado(s).<br />";
// Visualización de control.
MostrarArticulos();
?>

```

Resultado

Lista de artículos:

```

1 - Albaricoques - 35.5
2 - Cerezas - 48.9
3 - Fresas - 29.95
4 - Melocotones - 37.2
Identificador del nuevo artículo = 5.
4 artículo(s) añadido(s).
2 artículo(s) eliminado(s).

```

Lista de artículos:

```

1 - Albaricoques - 39.05
3 - Fresas - 32.945
5 - Peras - 32.89

```

Crear formularios que permitan actualizar los datos es muy simple. Por ejemplo, vamos a construir un formulario que permite realizar una entrada en lista.

Presentación del formulario

Identificador	Texto	Precio	Eliminar
1	Albaricoques	35,50	<input type="checkbox"/>
2	Cerezas	48,90	<input type="checkbox"/>
3	Fresas	29,95	<input type="checkbox"/>
4	Melocotones	37,20	<input type="checkbox"/>
5	Plátanos	15,35	<input type="checkbox"/>
6	Manzanas	24,50	<input type="checkbox"/>
7	Naranjas	30,00	<input type="checkbox"/>
12	Peras	29,90	<input type="checkbox"/>
	<input type="text"/>	<input type="text"/>	
	<input type="text"/>	<input type="text"/>	
	<input type="text"/>	<input type="text"/>	
	<input type="text"/>	<input type="text"/>	
	<input type="text"/>	<input type="text"/>	

Registrar

El formulario ofrece el contenido actual de la tabla que se puede cambiar (entrada directa en los campos) o eliminar (a través de casillas de verificación), además de cinco líneas en blanco que permiten introducir nuevos valores. En ninguno de estos casos se puede introducir el identificador (es el motor SQLite quien debe asignarlo).

Cada línea de la matriz contiene 4 cuadros de formulario que se denominan (atributo name de la etiqueta `<input>`) de la siguiente manera:

Columna	Nombre
Identificador	<code>entrada[i][modificar]</code>
Texto	<code>entrada[i][texto]</code>
Precio	<code>entrada[i][precio]</code>
Eliminar	<code>entrada[i][eliminar]</code>

El índice `i` es el identificador del artículo para las líneas existentes, y un número entre -1 y -5 para las líneas en blanco. El campo de la columna **identificador** es un campo oculto (`tipo="hidden"`), que se utiliza para identificar las líneas en las que el usuario ha realizado alguna modificación.

Con este proceso de denominación, todos los datos introducidos se recuperan en el script PHP en

forma de una matriz multidimensional llamada \$líneas. Cada línea de la matriz corresponde a una línea del formulario y la clave equivale al identificador (o -1 a -5 para las líneas nuevas) y el valor equivale a una matriz asociativa que proporciona los elementos introducidos.

Para identificar las líneas cambiadas por el usuario, los campos de entrada del texto y del precio de las líneas existentes contienen la pequeña porción de código JavaScript siguiente:

```
onChange="documento.formulario[$n].value=1"
```

Este código JavaScript tiene el efecto, cada vez que se cambia el campo en cuestión, de colocar un 1 en el campo oculto asociado a la línea. El formulario se llama formulario (<form name = "formulario"...), la expresión documento.formulario[n] designa el enésimo campo del formulario formulario del documento actual, el primer campo del formulario tiene el número 0. En la fuente, la variable \$n se calcula para cada línea \$i del formulario, por la fórmula $\$n = 4 * (\$i - 1)$: el campo oculto de la línea 1 tiene el número 0 (es el primero del formulario), el de la línea 2 el número 4 y así sucesivamente.

Este ejemplo puede (debe) mejorarse:

- Para controlar los datos introducidos por el usuario.
- Para gestionar los errores.

Fuente

```
<?php
// Inclusión del archivo que contiene las funciones
// generales.
require('funciones.inc');
// Abrir la base de datos.
$dbase = new SQLite3('/app/sqlite/diane.dbf');
// Procesamiento del formulario.
if (isset($_POST['ok'])) {
    // Recuperar la matriz que contiene los datos introducidos.
    $líneas = $_POST['entrada'];
    foreach($líneas as $identificador => $línea) {
        // Limpieza de los datos introducidos.
        $texto = trim($línea['texto']);
        // Para el precio, reemplazar la coma por un punto
        // y eliminar los espacios.
        $precio = str_replace(',', '.', $línea['precio']);
        $precio = str_replace(' ', '', $precio);
        // En este punto, debe comprobar los datos introducidos ...
        // Definición de la consulta que se va a ejecutar.
        // Para cada acción, utilizaremos una consulta
        // diferente.
        $consulta = '';
        if ($identificador < 0 and $texto.$precio != '') {
            // Identificador negativo y algo introducido
            // = creación = INSERT.
```

PHP Data Objects (PDO)

PHP Data Objects (PDO) es una extensión introducida en la versión 5.1, que define una interfaz uniforme para acceder a las bases de datos en PHP. El acceso a una base de datos a través de PDO se efectúa por medio de un controlador que expone las características de la base de datos.

Cabe señalar que PDO no proporciona una capa de abstracción de la base de datos, sino una capa de abstracción del acceso a las bases de datos. Las consultas se escriben y deben respetar la sintaxis de la base de datos que utiliza; PDO no reescribe consultas SQL y no emula las características que faltan (a excepción de las consultas con parámetros).

Muchas bases de datos disponen de un controlador PDO como MySQL, Oracle, Microsoft SQL Server y SQLite. Tenga en cuenta que en la versión 5.4, los controladores de PDO para Oracle y SQL Server están todavía en fase experimental.

PDO es una extensión orientada a objetos compuesta de tres clases:

- PDO: conexión entre PHP y la base de datos,
- PDOStatement: consulta preparada, y, después de la ejecución, resultado asociado,
- PDOException: excepción planteada por PDO.

En este capítulo, presentaremos esta extensión utilizando un sencillo ejemplo comentado:

```
<?php
// Definición de los parámetros de conexión.
// La sintaxis de la fuente (Data Source Name o DSN)
// es específica a cada controlador.
// Cambiar el valor de la variable $prueba para comprobar
// diferentes bases de datos.
$prueba = 3;
switch ($prueba) {
    caso 1: // MySQL
        $fuente = 'mysql:host=localhost;dbname=diane';
        $usuario = 'root';
        $contraseña = '';
        break;
    caso 2: // Oracle
        $fuente = 'oci:dbname=diane';
        $usuario = 'demeter';
        $contraseña = 'demeter';
        break;
    caso 3: // SQLite (versión 3.x)
        $fuente = 'sqlite:/app/sqlite/diane.dbf';
        $usuario = '';
        $contraseña = '';
        break;
}
// Definición de dos consultas de prueba.
// Tener en cuenta que la consulta de inserción tiene parámetros ('es
// de hecho la única característica que emula PDO,
// si no se admite de manera nativa por la base de
// datos).
$sql_select = 'SELECT * FROM artículos ORDER BY identificador';
$sql_insert = 'INSERT INTO artículos(texto,precio) VALUES(:p1,:p2)';
// Todas las operaciones se efectúan en un bloque
// 'try' para recuperar las excepciones planteadas por PDO.
try {
    // Conexión a la base de datos.
```

```

$db = new PDO($fuente, $usuario, $contraseña);
// Modificación de los parámetros de la conexión para
// solicitar que se planteen excepciones en caso
// de error.
$db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
// Preparar una consulta para la inserción.
$stmt = $db->prepare($sql_insert);
// Enlazar los parámetros.
$stmt->bindParam(':p1', $nom);
$stmt->bindParam(':p2', $precio);
// Asignar un valor a las variables.
$nombre = 'Manzanas';
$precio = 30.5;
// Ejecutar la consulta.
// Para las bases de datos que admiten las transacciones,
// se pueden utilizar los métodos beginTransaction(), commit() y rollback()
// de los objetos PDO.
$stmt->execute();
// Preparar una consulta para la selección.
$stmt = $db->prepare($sql_select);
// Ejecutar la consulta.
$stmt->execute();
// Recuperar el resultado.
// Hay disponibles varios métodos para recuperar
// el resultado: fetch(), fetchObject(), fetchAll().
// El método fetch() dispone de un parámetro que permite
// de especificar el tipo de resultado (matriz, objeto, etc.).
while ($línea = $stmt->fetch()) {
    echo "$línea[1] - $línea[2]<br />\n";
}
// Liberar los recursos.
$stmt = null;
$db = null;
} catch (PDOException $e) {
    // Gestionar las excepciones.
    echo 'Error!: ', $e->getMessage(), '<br />';
    die();
}
?>

```

Resultado

```

Albaricoques - 35.5
Cerezas - 48.9
Fresas - 29.95
Melocotones - 37.2
Manzanas - 30.5

```

"Magic quotes": el regreso

1. Preámbulo

En el capítulo Gestionar los formularios hemos visto que las versiones anteriores de PHP ofrecían una característica denominada "magic quotes", cuyo principal objetivo era resolver un problema relacionado con el registro de datos en una base de datos realizando una codificación en los datos introducidos en un formulario.

Ejemplo (inserción en la base de un dato que contiene un apóstrofo)

```
<?<?php
// Dato que supone un problema (puede introducirse sin querer
// en un formulario).
$texto = "Pomme d'api";
$precio = 10;
// Consulta.
$consulta = "INSERT INTO artículos(texto,precio)" .
            "VALUES(' $texto', $precio)";
echo "$consulta<br />";
// Ejecución con MySQL.
echo "<p><b>MySQL</b><br />";
$conexión = mysqli_connect();
$ok = mysqli_select_db($conexión, 'diane');
$resultado = mysqli_query($conexión, $consulta);
echo mysqli_error($conexión), '<br />'; // MySQL no genera ninguna alerta
// Ejecución con Oracle.
echo "<p><b>Oracle</b><br />";
$conexión = oci_connect('demeter', 'demeter', 'diane');
$resultado = oci_execute(oci_parse($conexión, $consulta));
// Ejecución con SQLite.
echo "<p><b>SQLite</b><br />";
$base = new SQLite3('/app/sqlite/diane.dbf');
$resultado = $base->query($consulta);
?> >
```

Resultado

```
INSERT INTO artículos(texto,precio) VALUES('Pomme d'api',10)
```

MySQL

You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near 'api',10)' at line 1

Oracle

Warning: oci_parse(): ORA-01756: quoted string not properly terminated in **/app/scripts/index.php** on line 19
Warning: oci_execute() expects parameter 1 to be resource, boolean given in **/app/scripts/index.php** on line 19

SQLite

Warning: SQLite3::query(): Unable to prepare statement: 1, near "api": syntax error in **/app/scripts/index.php** on line 23

En SQL, el delimitador de cadena de caracteres es el apóstrofo: si una consulta envía la cadena 'Pomme

d'api' a la base de datos, esta última interpretará 'Pomme d' como una cadena y no sabrá qué hacer con el resto (api').

Para resolver este problema, se debe indicar a la base de datos que los apóstrofes dentro de la cadena no son delimitadores, generalmente colocando antes del apóstrofo un carácter "mágico" ("de escape"): el carácter de barra invertida (\) para MySQL o el apóstrofo (') para MySQL y otras bases de datos como Oracle, Sybase y Microsoft SQL Server o SQLite.

Ejemplo

```
<?<?php
// Dato corregido (válido para todas las bases de datos).
$texto = "Pomme d'api";
$precio = 10;
// Consulta.
$consulta = "INSERT INTO artículos(texto,precio)" .
            "VALUES ('$texto', $precio)";
echo "$consulta<br />";
// Ejecución con MySQL.
echo "<p><b>MySQL</b><br />";
$conexión = mysqli_connect();
$ok = mysqli_select_db($conexión, 'diane');
$resultado = mysqli_query($conexión, $consulta);
echo mysqli_error($conexión), '<br />'; // MySQL no genera ninguna alerta
// Ejecución con Oracle.
echo "<p><b>Oracle</b><br />";
$conexión = oci_connect('demeter', 'demeter', 'diane');
$resultado = oci_execute(oci_parse($conexión, $consulta));
// Ejecución con SQLite.
echo "<p><b>SQLite</b><br />";
$base = new SQLite3('/app/sqlite/diane.dbf');
$resultado = $base->query($consulta);
?> >
```

Resultado

```
INSERT INTO artículos(texto,precio) VALUES('Pomme d'api',10)
```

MySQL

Oracle

SQLite

En cuanto a MySQL, la codificación con el carácter barra invertida (\) también funciona (pero no con otras bases de datos).

```
$texto = "Pomme d\api";
```

En las versiones anteriores de PHP, la característica "magic quotes" de codificación automática respondía a este problema: si estaba activada (directiva de configuración `magic_quotes_gpc = on`), todos los datos procedentes de un formulario (métodos GET o POST), de una URL (método GET) o de una cookie se codificaban automáticamente con el carácter de barra invertida (\), o apóstrofo (') si la directiva de configuración `magic_quotes_sybase` estaba en `on`.

En la versión 5.4, la función "magic quotes" se ha eliminado definitivamente ya que planteaba otros problemas por otra parte.

En consecuencia, deberemos efectuar el cifrado necesario en el momento del registro en la base de datos. Esto lo veremos en la siguiente sección.

2. Carga de datos procedentes de una base de datos

Antes de la versión 5.3, era posible realizar una codificación "magic quotes" automática en los datos leídos en una base de datos, pero sólo para MySQL y SQL Server.

Para ello, bastaba con asignar el valor `on` a la directiva de configuración `magic_quotes_runtime`.

Desde la versión 5.3, esta característica quedó obsoleta y se eliminó definitivamente en la versión 5.4.

En consecuencia, un dato leído de una base de datos no toma ningún cifrado "magic quotes" automático, lo cual es bueno para poder manipular este dato y especialmente su presentación. Si el dato leído está destinado a ser registrado de nuevo en la base de datos, será necesario "escapar" correctamente los apóstrofes (ver el punto siguiente).

Ejemplo

```
<?php
// Consulta.
$consulta = 'SELECT texto FROM artículos " .
            "WHERE texto LIKE '%api%'";
// Ejecución con MySQL.
$conexión = mysqli_connect();
$ok = mysqli_select_db($conexión, 'diane');
$resultado = mysqli_query($conexión, $consulta);
$artículo = mysqli_fetch_array($resultado);
echo "$artículo[0]<br />";
?>
```

Resultado

Pomme d'api

3. Actualización de los datos de la base de datos

Para actualizar los datos de la base de datos, asegúrese de que todos los datos de tipo "texto" tienen el carácter de escape adaptado (`\` o `'` para MySQL, `'` para las bases de datos de otro tipo) antes de cada apóstrofo.

Se pueden utilizar las funciones `addslashes` y `mysqli_real_escape_string` para ello.

Sintaxis

```
cadena addslashes(cadena valor)
cadena mysqli_real_escape_string(objeto conexión,
cadena valor)  cadena SQLite3::escapeString(cadena valor)
```


Con

conexión	Identificador de conexión devuelto por la función <code>mysqli_connect</code> .
valor	Cadena de caracteres a escapar.

La función `mysqli_real_escape_string` agrega una barra invertida (`\`) delante de todos los caracteres apóstrofo (`'`), comillas (`"`) y barra invertida (`\`), NUL (ASCII 0), retorno de línea (`\n`) y retorno de carro (`\r`) encontrados en la cadena `valor`.

La función `addslashes` agrega una barra invertida (`\`) delante de todos los caracteres apóstrofo (`'`), comillas (`"`), barra invertida (`\`) y NUL (ASCII 0) encontrados en la cadena `valor`.

El método `SQLite3::escapeString` añade simplemente un apóstrofo delante de todos los caracteres apóstrofo (`'`) que se encuentren en la cadena `valor`.

 El parámetro `valor` ya no necesita ser escapado.

Ejemplo

```
<?php
$valor = " ' \ \" "; // es para probar...
// addslashes
echo addslashes($valor), '<br />';
// mysqli_real_escape_string
$conexión = mysqli_connect();
echo mysqli_real_escape_string($conexión, $valor), '<br />';
//SQLite3::escapeString
$dbase = new SQLite3('/app/sqlite/diane.dbf');
echo $dbase->escapeString($valor), '<br />';
?>
```

Resultado

```
\ ' \ \ \"
\' \' \' \'
\' \' \' \'
\' \' \ "
```

Si trabaja específicamente con una base de datos MySQL o SQLite, se recomienda utilizar la función o el método específico de esta base de datos.

Si desea tener un código que funcione con todas las bases de datos y, en particular Oracle, la solución es escribir su propia función.

Ejemplo

```
<?php
function hacia_base($valor) {
    // El único carácter que supone verdaderamente un problema es el apóstrofo (');
    // por tanto, es el único que se escapa por esta función.
    // Una solución válida para todas las bases de datos consiste en
    // escaparlos por sí mismo => reemplazando ' por ''.
    return str_replace("'", "", $valor);
}
$valor = "Pomme d'api";
echo hacia_base($valor);
?>
```

Resultado

```
Pomme d''api
```

Esta función se puede llamar durante la construcción de una consulta.

Ejemplo

```
<?php
```

```
function hacia_base($valor) {
    // El único carácter que supone verdaderamente un problema es el apóstrofo (');
    // por tanto, es el único que se escapa por esta función.
    // Una solución válida para todas las bases de datos consiste en
    // escaparlos por sí mismo => reemplazando ' por ''.
    return str_replace("'", "", $valor);
}
$texto = "Pomme d'api";
$precio = 10;
// La utilización del sprintf hace más legible la construcción
// de la consulta.
$consulta = sprintf(
    "INSERT INTO artículos(texto,precio) VALUES('%s',%s)",
    hacia_base($texto),
    $precio);
echo $consulta;
?>
```

Resultado

```
INSERT INTO artículos(texto,precio) VALUES('Pomme d''api',10)
```

También es posible escribir una función genérica en el mismo espíritu que la función `sprintf`, marcando la ubicación de los parámetros por una secuencia `%n`, `n` donde 1 es para el primer parámetro, 2 para el segundo... Esta función acepta un número variable de parámetros, el primero es la estructura de la consulta y los siguientes los valores de los parámetros en orden de numeración (ver capítulo Variables, constantes, tipos y matrices - sección Algunas funciones útiles sobre las cadenas de caracteres, las fechas y los números).

Ejemplo

```
<?php
function construir_consulta($sql) {
    // Recuperar el número de parámetro.
    $numero_param = func_num_args();
    // Hacer bucle para todos los parámetros a partir del segundo
    // (el primero contiene la consulta de base).
    for($i=1;$i<$numero_param;$i++) {
        // Recuperar el valor del parámetro.
        $valor = func_get_arg($i);
        // Si es una cadena, escaparla.
        if (is_string($valor)) {
            $valor = str_replace("'", "", $valor);
        }
        // Colocar el valor en su ubicación %n (n = $i).
        $sql = str_replace("%$i", $valor, $sql);
    }
    // Devolver la consulta.
    return $sql;
}
// Las variables contienen valores que provienen de alguna parte ...
$texto = "Pomme d'api";
$precio = 10;
// Construcción de la consulta.
$consulta = construir_consulta(
    "INSERT INTO artículos(texto,precio) VALUES('%1',%2)",
    $texto,
    $precio);
echo $consulta;
?>
```

Resultado

```
INSERT INTO artículos(texto,precio) VALUES('Pomme d'api',10)
```



No hay tal problema utilizando consultas con parámetros en Oracle o MySQL.

Ejemplo con Oracle

```
<?php
// Conexión.
$conexión = oci_connect('demeter','demeter','diane');
// consulta INSERT (con parámetros).
$consulta = 'INSERT INTO artículos(texto,precio)
            VALUES(:p1,:p2)';
// Análisis.
$cursor = oci_parse($conexión,$consulta);
// Asociación entre las variables y los parámetros.
oci_bind_by_name($cursor,':p1',$texto,50);
oci_bind_by_name($cursor,':p2',$precio,32);
// Ejecución de la consulta.
$texto = "Pomme d'api"; // ningún problema con d'api
$precio = 10;
$ok = oci_execute($cursor); // COMMIT automático
$numéro = oci_num_rows($cursor);
echo "$numéro artículo insertado.";
?>
```

Resultado

1 artículo insertado.

Descripción del problema

El protocolo HTTP (*HyperText Transfer Protocol*) es un protocolo "sin estado". Es decir, no hay nada que permita identificar que el mismo usuario que ha estado previamente en la página A ahora está accediendo a la página B.

En cuanto a PHP, ahora sabemos que una variable tiene un ámbito de aplicación igual al script en el que está definida, y que existe sólo durante el tiempo de ejecución del script.

Ejemplo

- Página HTML de "conexión":

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head><title>Entrada</title></head>
  <body>
    <form action="inicio.php" method="post">
      <div>
        Nombre: <input type="text" Name="nombre" value= "" /><br />
        <input type="submit" name="ok" value="OK" />
      </div>
    </form>
  </body>
</html>
```

- Script PHP para el procesamiento de la página:

```
<?php
// Procesamiento del formulario.
if (! empty($_POST)) {
  // Recuperar la información introducida.
  $nombre = $_POST['nombre'];
}
// Visualización de la página de inicio.
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head><title>Inicio</title></head>
  <body>
    <div>
      ¡Hola <?php echo isset($nombre)?$nombre:''; ?>!<br />
      <!-- enlace hacia otra página -->
      <a href="acción.php">Acción</a>
    </body>
  </html>
```

- Script PHP llamado por el enlace:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head><title>Acción</title></head>
  <body>
    <div>
      <!-- visualización del nombre del usuario -->
      ¡Hola <?php echo isset($nombre)?$nombre:''; ?>!<br />
      Acción ...
    </div>
  </body>
</html>
```

```
</body>
</html>
```

Resultado

- Presentación inicial y entrada:



Nombre:

- Resultado al hacer clic en el botón **OK**:

¡Hola Olivier!
Acción

- Resultado al hacer clic en el enlace

¡Hola!
Acción...

La información introducida en la primera página se define en la segunda página, a la que se llama para procesar los datos introducidos en el formulario. La variable \$nombre, definida en este script, no existe en el script acción.php (problema de ámbito de aplicación).

El problema es el mismo si el script inicio.php se llama de nuevo por el enlace.

Ejemplo

```
<?php
// Procesamiento del formulario.
if (! empty($_POST)) {
    // Recuperar la información introducida.
    $nombre = $_POST['nombre'];
}
// Visualización de la página de inicio.
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
    <head><title>Inicio</title></head>
    <body>
        <div>
            ¡Hola <?php echo isset($nombre)?$nombre:''; ?>!<br />
            <!-- enlace hacia otra página -->
            <a href="inicio.php">Inicio</a>
        </div>
    </body>
</html>
```

Resultado

- Presentación en la página de inicio después de la identificación:

¡Hola Olivier!

Inicio

- Resultado al hacer clic en el enlace:

¡Hola!

Inicio

El resultado es el mismo: el valor de la variable al final de la primera ejecución del script no se conserva al final del script (el problema del tiempo de vida).

Sin embargo, un sitio interactivo que no se limita a mostrar las páginas una tras otra, a menudo necesita el punto de vista de la lógica de aplicación para identificar a un usuario de una página a otra y mantener información relativa a este usuario de una página a otra (por lo general, un carro de la compra realizada por el usuario en una página siempre debe estar definido en la página de pago).

El término "sesión" designa el período de tiempo correspondiente a la navegación continua de un usuario en un sitio. "Administrar las sesiones" significa, por lo tanto, ser capaz de identificar el momento en que un nuevo usuario accede a una página del sitio y conservar la información relativa a ese usuario hasta que sale del sitio. El usuario no tiene por qué ser un usuario autenticado por medio de un nombre de usuario y contraseña, puede ser perfectamente un usuario "anónimo", no referenciado por el sitio, que realiza una compra. Cada vez más, los sitios interactivos ofrecen características de identificación (miembro, suscriptor...), ya que permite mantener información sobre el usuario de una visita a otra (por ejemplo, las preferencias). Esta posibilidad también se estudia en este capítulo, pero desde la perspectiva del concepto de sesión, la visita del usuario el viernes corresponderá a una sesión diferente de su visita del lunes, aunque parte de la información introducida el lunes es probable que se retome el viernes.

Este capítulo tiene como objetivo presentar las diferentes técnicas que permitan por una parte identificar a un usuario y, por otra, "seguir" a ese usuario y los datos asociados a él, de una página a otra.

En primer lugar, veremos cómo autenticar a un usuario y crear un identificador único.

A continuación, abordaremos la administración de sesiones, los métodos "artesanales" y las nuevas características disponibles en PHP desde la versión 4.

Por último, concluiremos este capítulo, evocando las técnicas que permiten conservar información de una visita a otra

Autenticación

1. Información general

Algunos sitios necesitan autenticar a los usuarios que acceden al sitio para comprobar si están registrados.

Esta autenticación por lo general consta de dos pasos:

- Introducción de las credenciales de usuario, por lo general un nombre y una contraseña.
- Verificación de que las credenciales introducidas corresponden a un usuario registrado.

2. Introducción de las credenciales de identificación

Las credenciales de identificación se pueden introducir de dos formas:

- A través de un formulario previsto a tal efecto.
- A través de las funciones de autenticación HTTP.

Formulario

Es muy sencillo crear un pequeño formulario que permita al usuario introducir un nombre y una contraseña.

Ejemplo de script PHP (login.php) que muestra este formulario (función de verificación de que el usuario existe por el momento sin definir)

```
<?php
// Inclusión del archivo que contiene las funciones generales.
include('funciones.inc');
// Función que verifica que las credenciales de identificación
// introducidas son correctas.
function usuario_existe($identificador,$contraseña) {
    // Aleatoria, esperando algo mejor ...
    return (bool) rand(0,1);
}
// Inicialización de las variables.
$identificador = '';
$contraseña = '';
$mensaje = '';
// Procesamiento del formulario.
if (isset($_POST['conexión'])) {
    // Recuperar la información introducida.
    $identificador = $_POST['identificador'];
    $contraseña = $_POST['contraseña'];
    // Verificar que el usuario existe.
    if (usuario_existe($identificador,$contraseña)) {
        // El usuario existe ...
        // Ir a otra página y detener
        // el script.
        header('location: inicio.php');
        exit;
    } else {
        // El usuario no existe ...
    }
}
```

```

// Mostrar un mensaje y proponer de
// nuevo la identificación.
$mensaje = 'Identificación incorrecta. ';
$mensaje .= 'Vuelva a intentarlo.';
// Dejar que el formulario se muestre de nuevo ...
}
}
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head><title>MiSitio.com</title></head>
<body>
<form action="login.php" method="post">
<table border="0">
<tr>
<td align="right">Identificador:</td>
<td><input type="text" Name="identificador" value=
"<?php echo hacia_formulario($identificador); ?>" /></td>
</tr>
<tr>
<td align="right">Contraseña:</td>
<td><input type="password" Name="contraseña" value=
"<?php echo hacia_formulario($contraseña); ?>" /></td>
</tr>
<tr>
<td></td>
<td align="right"><input type="submit" name="conexión"
value="Conexión" /></td>
</tr>
</table>
<?php echo $mensaje; ?>
</form>
</body>
</html>

```

Resultado

- Presentación inicial:

- Entrada:

- Resultado si la identificación es errónea:

Identificador:

Contraseña:

Identificación incorrecta. Vuelva a intentarlo.

La utilización de un campo de tipo password permite ocultar los datos introducidos de la contraseña.

Si los datos introducidos no son correctos, la página se vuelven a proponer. De lo contrario, se muestra una página de inicio.

Autenticación HTTP

Mediante la función `header`, es posible hacer que el navegador muestre un cuadro de diálogo que invite al usuario a que introduzca un nombre y una contraseña. El mensaje de encabezado a enviar es:

```
WWW-Authenticate: Basic realm="xxxxx"
```

xxxxxx = nombre que se muestra (nombre de la organización, por ejemplo)

Si el usuario hace clic en el botón **OK**, el script se llama de nuevo con los valores introducidos disponibles en las variables predefinidas: `$PHP_AUTH_USER` y `$PHP_AUTH_PW` accesibles en la matriz asociativa `$_SERVER`.

Ejemplo (script `login.php` con esta técnica)

```
<?php
// Inclusión del archivo que contiene las funciones generales.
include('funciones.inc');
// Función que verifica que las credenciales de identificación introducidas
// son correctas.
function usuario_existe($identificador,$contraseña) {
    // Aleatoria, esperando algo mejor ...
    return (bool) rand(0,1);
}
// Función que muestra la autenticación HTTP.
function autenticación($mensaje) {
    header("WWW-Authenticate: Basic realm=\"\$mensaje\"");
    // Si el usuario hace clic en el botón cancelar,
    // se ejecutan las líneas siguientes (de lo contrario, el script se
    // llama de nuevo, pero con $PHP_AUTH_USER rellenado
    // y el script no pasará ya por aquí).
    // Mostrar un mensaje y proponer al usuario
    // volver a intentarlo.
    echo 'Debe introducir un nombre y una contraseña ',
        'para acceder al sitio.<br />';
    echo '<a href="login.php">Volver a intentarlo</a>';
    exit;
}
if (! isset($_SERVER['PHP_AUTH_USER'])) {
    // Ninguna variable $PHP_AUTH_USER = primera llamada del script.
    // Solicitud de identificación.
    autenticación('MiSitio.com');
} else {
    // Variable $PHP_AUTH_USER existe = llamada después de entrada.
    // Recuperar la información introducida.
```

```
$identificador = $_SERVER['PHP_AUTH_USER'];  
$contraseña = $_SERVER['PHP_AUTH_PW'];  
// Verificar que el usuario existe.  
if (usuario_existe($identificador,$contraseña)) {  
    // El usuario existe ...  
    // Ir a otra página y detener el script.  
    header('location: inicio.php');  
    exit;  
} else {  
    // El usuario no existe ...  
    // Intentar de nuevo.  
    autenticación('MiSitio.com: identificación incorrecta');  
}  
}  
?>
```

Resultado

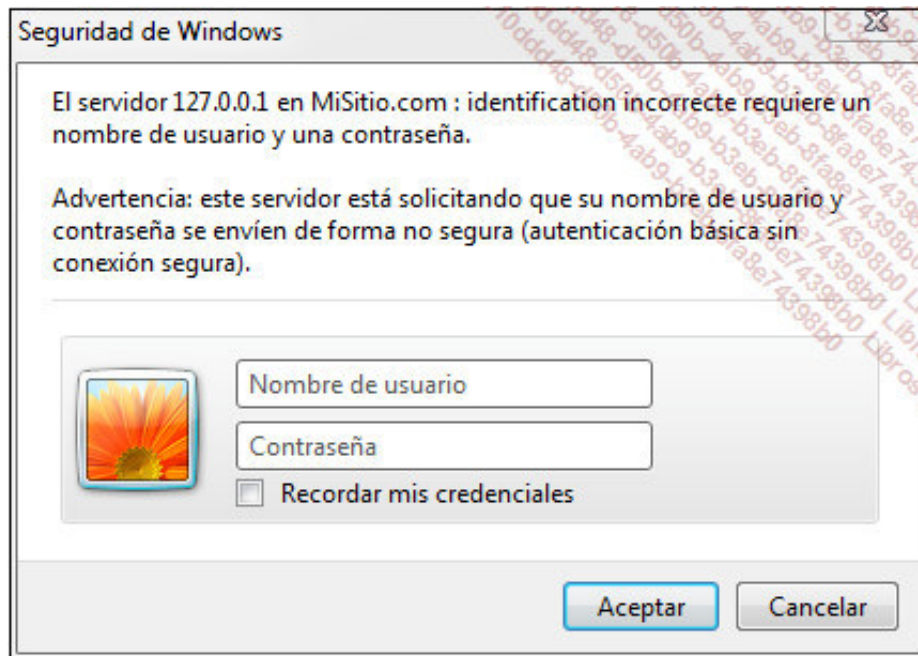
- Presentación inicial (Internet Explorer versión 9):



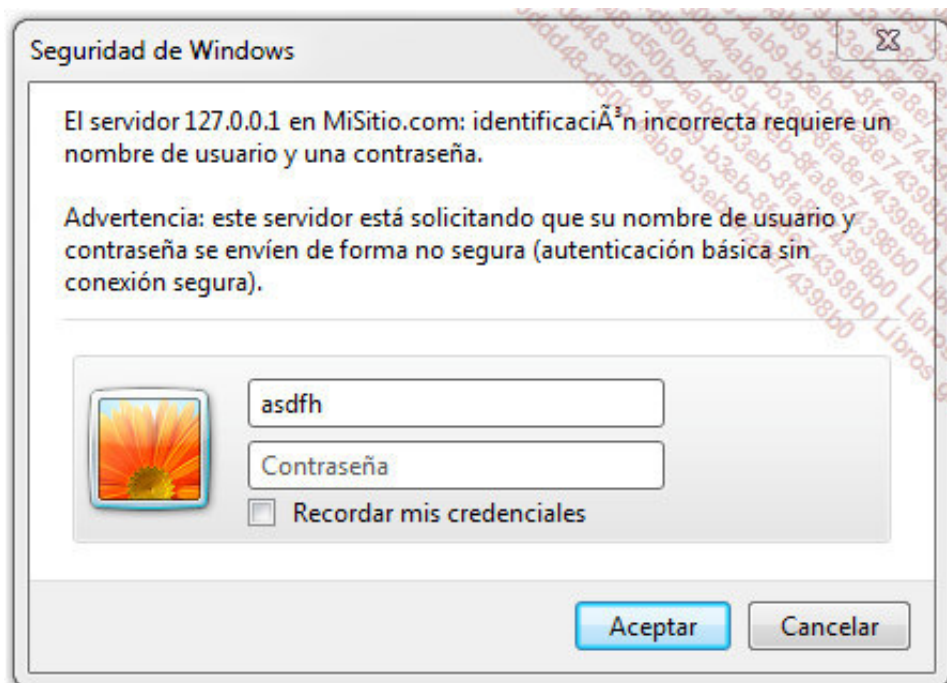
- Haga clic en el botón **Cancelar**:

Debe introducir un nombre y una contraseña para acceder al sitio.
[Volver a intentarlo](#)

- Nueva presentación (por el enlace) y entrada:



- Nueva presentación si la identificación es incorrecta:



Si la identificación es correcta, se muestra una página de inicio.

- Por ahora, en los dos ejemplos, el acceso a la página de inicio no está protegido: un usuario que solicita acceso a esta página puede acceder sin problemas.

3. Verificar las credenciales de identificación introducidas

Sea cual sea el método de identificación utilizado en el punto anterior, a continuación, debe verificar que la información introducida se corresponden con un usuario "conocido".

Normalmente, este control se realiza utilizando una base de datos que contiene la lista de usuarios y probablemente otra información.

Vamos a tomar como hipótesis de trabajo para el futuro que utilizamos una base de datos MySQL y en esta base, la existencia de una tabla `usuarios` con dos columnas, `identificador` y `contraseña`.

Ejemplo

```
<?php
// Función que verifica que las credenciales de identificación introducidas
// son correctas.
function usuario_existe($identificador,$contraseña) {
    // Conexión y selección de la base de datos.
    $conexión = mysqli_connect();
    mysqli_select_db($conexión,'diane');
    // Definición y ejecución de una consulta preparada.
    $sql = 'SELECT 1 FROM usuarios ';
    $sql .= 'WHERE identificador = ? AND contraseña = ?';
    $consulta = mysqli_stmt_init($conexión);
    $ok = mysqli_stmt_prepare($consulta,$sql);
    $ok = mysqli_stmt_bind_param
        ($consulta,'ss',$identificador,$contraseña);
    $ok = mysqli_stmt_execute($consulta);
    mysqli_stmt_bind_result($consulta,$existe);
    $ok = mysqli_stmt_fetch($consulta);
    mysqli_stmt_free_result($consulta);
    // La identificación tiene éxito si la consulta devuelve
    // una línea (el usuario existe y la contraseña
    // es correcta).
    // Si este es el caso $existe contiene 1, de lo contrario está
    // vacía. Basta con devolverla como un valor booleano.
    return (bool) $existe;
}
?>
```

Es posible utilizar otros métodos de autenticación que no dependen de una base de datos (un simple archivo, por ejemplo).

Crear un identificador único

En muchas situaciones, incluidas las relativas a la gestión de sesiones, es necesario generar identificadores únicos.

De hecho, en el contexto de la gestión de sesiones, este identificador es a menudo útil, pues permite, como su nombre indica, identificar las sesiones y, por lo tanto, poder diferenciarlas.

PHP ofrece la función `uniqid` para generar identificadores únicos.

Sintaxis

```
cadena uniqid() ([cadena prefijo [, booleano plus_unique]])
```

prefijo Prefijo que se agregará al identificador. Colocar una cadena vacía o nada si no desea un prefijo.

plus_unique Si este parámetro se establece como `TRUE`, se agregan datos adicionales al final del valor devuelto para obtener un identificador más largo y más difícil de identificar.

La función `uniqid` devuelve una cadena de trece caracteres, o veintitrés si el parámetro `plus_unique` está en `TRUE` (sin incluir el prefijo) calculado a partir de la hora actual en microsegundos.

Ejemplo

```
<?php
echo uniqid(), '<br />';
echo uniqid(), '<br />';
echo uniqid('abc'), '<br />';
echo uniqid('', TRUE), '<br />';
?>
```

Resultado

```
46a441974c4b7
46a441974c4c3
abc46a441974c4c8
46a441974c4cb2.69830597
```

Este ejemplo muestra que el identificador generado es único, incluso si la diferencia entre dos llamadas sucesivas es baja. Por contra, el identificador generado puede considerarse insuficientemente aleatorio y un poco demasiado fácil de determinar.

Una técnica clásica consiste en cifrar el identificador generado. La función `md5` permite hacerlo muy fácilmente, utilizando un método MD5.

Sintaxis

```
cadena md5(cadena valor)
```

Valor Cadena que se va a cifrar.

La función `md5` devuelve la cadena cifrada.

Ejemplo

```
<?php
echo md5('olivier');
?>
```

Resultado

d3ca5dde60f88db606021eeba2499c02

La combinación de las funciones `uniqid` y `md5` da el siguiente código:

```
<?php
echo md5(uniqid()),'<br />';
echo md5(uniqid()),'<br />';
?>
```

Resultado

d4ca38f2b9a349dd1138497046ceb15e
952afa792bfcbb08c94cd6dee357e590

El nuevo identificador tiene 32 caracteres; ahora es más aleatorio y menos fácil de determinar.

Para los paranoicos de la seguridad, es posible ir más lejos utilizando, además, un prefijo aleatorio.

Ejemplo

```
<?php
echo md5(uniqid(rand())),'<br />';
echo md5(uniqid(rand())),'<br />';
?>
```

Resultado

afa8a8704dd18d9f61b726aceeb0a80b
06f7a56aeba14bc4ce70b7b5d66d4aa0

Se puede escribir una función genérica para definir un identificador único.

Ejemplo

```
<?php
function identificador_único() {
    // generación del identificador
    return md5(uniqid(rand()));
}
?>
```

Pasar la información a través de la URL

1. Principio

La URL (*Uniform Resource Locator*) se pueden utilizar para pasar información de una página a otra.

Sintaxis

```
url_clasica?nombre=valor[&...]
```

El signo de interrogación (?) introduce la lista de parámetros de URL separados por el carácter &, cada parámetro se compone de un par nombre/valor en la forma `nombre=valor`.

Ejemplos

```
www.misitio.com/info/inicio.php?nombre=Olivier
buscar.php?nombre=Olivier&apellido=HEURTEL
```

En PHP, los parámetros pasados a la URL corresponden a un script, están disponibles en este script en forma de variables sin necesidad de realizar ningún análisis de la URL.

Los principios de recuperación de las variables son los mismos que para los valores introducidos en un formulario (véase el capítulo Gestión de formularios).

Todos los parámetros de la URL se almacenan automáticamente en la matriz asociativa `$_GET`: la clave de la matriz es igual al nombre del parámetro.

Ejemplo

- Script `pagina1.php`

```
<?php
// Inicialización de una variable.
$nombre='Olivier';
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head><title>Página 1</title></head>
<body>
<div>
<!-- enlace hacia la página 2 pasando el valor de $nombre
en la URL -->
<a href="pagina2.php?nombre=<?php echo $nombre; ?>">Página 2</a>
</div>
</body>
</html>
```

- Código fuente de la página en el navegador

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head><title>Página 1</title></head>
<body>
<div>
<!-- enlace hacia la página 2 pasando el valor de $nombre
```

```

    en la URL -->
    <a href="pagina2.php?nombre=Olivier">Página 2</a>
  </div>
</body>
</html>

```

- Script pagina2.php

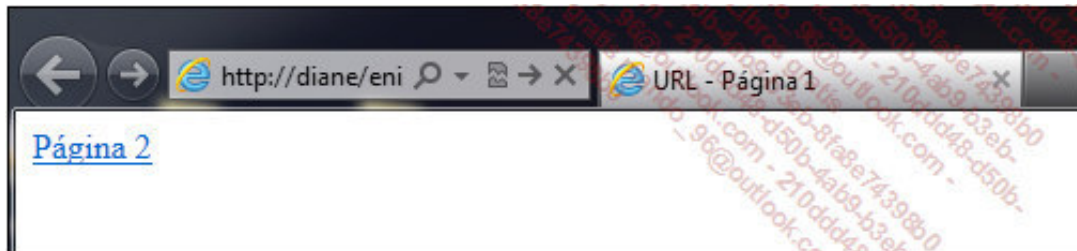
```

<?php
$nombre = $_GET['nombre'];
echo $nombre;
?>

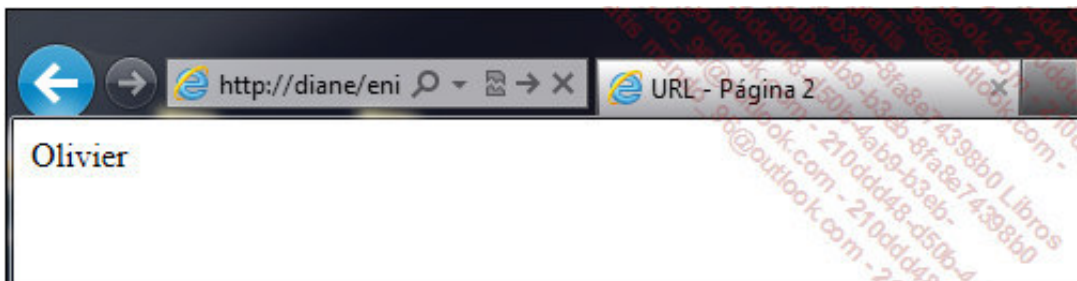
```

Resultado

- Presentación de la página 1:



- Resultado al hacer clic en el enlace:



Por lo tanto, las variables definidas en un script se pueden transmitir a otro script. Se transmite el valor, no la propia variable; nada impide en el script de destino recuperar el valor para colocarlo en una variable con otro nombre.

Si el valor que se transmite no contiene caracteres especiales (espacio, signo &, signo de interrogación (?), etc.), puede colocarse directamente en la URL como se indica anteriormente. De lo contrario, es necesario codificarla para evitar la interpretación incorrecta de estos caracteres especiales.

En el ejemplo anterior, si la variable contiene "Oliver & Xavier", sólo "Oliver" se recuperará en la variable `$nombre` a su llegada ya que el & se interpreta como el separador de parámetros.

Esta codificación se puede realizar muy fácilmente con las funciones `urlencode` o `rawurlencode`.

Sintaxis

```

cadena urlencode(cadena valor)
cadena rawurlencode(cadena valor)

```

valor Cadena que se va a codificar.

Estas dos funciones devuelven la cadena después de la codificación. La codificación consiste en sustituir todos los caracteres no alfanuméricos por una secuencia %xy, xy siendo un número hexadecimal igual al código ASCII del carácter. La diferencia entre las dos funciones es sutil y sólo implica el carácter de espacio: la función `urlencode` reemplaza los espacios con el carácter "más" (+), el verdadero carácter "más" se codifica, mientras que la función `rawurlencode` reemplaza los espacios por la secuencia %20 (código ASCII 32 en hexadecimal). La función `urlencode` es coherente con el tipo MIME `application/x-www-form-urlencoded` (tipo usado para transmitir los valores de los formularios), mientras que la función `rawurlencode` cumple con RFC1738; a priori, debemos utilizar la función `rawurlencode`.

Ejemplo

```
<?php
// Inicialización de una variable.
$nombre='Olivier & Xavier';
echo urlencode($nombre),'<br />';
echo rawurlencode($nombre),'<br />';
?>
```

Resultado

```
Olivier+%26+Xavier
Olivier%20%26%20Xavier
```

- El script `pagina1.php` puede modificarse de la siguiente manera:

```
<?php
// Inicialización de una variable.
$nombre='Olivier & Xavier';
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head><title>Página 1</title></head>
<body>
<div>
<!-- enlace hacia la página 2 pasando el valor de $nombre
en la URL -->
<a href="pagina2.php?nombre=<?php echo rawurlencode($nombre); ?>">
Página 2</a>
</div>
</body>
</html>
```

Código fuente de la página en el navegador

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head><title>Página 1</title></head>
<body>
<div>
<!-- enlace hacia la página 2 pasando el valor de $nombre
en la URL -->
<a href="pagina2.php?nombre=Olivier%20%26%20Xavier">
Página 2</a>
</div>
</body>
</html>
```

Resultado mostrado a la llegada

Olivier & Xavier

La cadena de la consulta también puede construirse mediante la función `http_build_query` que apareció en la versión 5.

Sintaxis simplificada

```
cadena http_build_query(matriz datos [, cadena prefijo])
```

Con

datos	Matriz que contiene los datos que se utilizarán para construir la cadena de la consulta. El índice o la clave de la matriz se utiliza como nombre del parámetro para el valor asociado.
prefijo	Prefijo que se va a utilizar para el nombre de los parámetros, cuando sea un índice numérico. Permite tener a la llegada un nombre que se puede utilizar como nombre de variable (un nombre de variable PHP no puede comenzar con un dígito).

Esta función construye, a continuación, una cadena de consulta con el formato `clave1=valor1&clave2=valor2&...` utilizando las claves (o índices) y los valores encontrados en la matriz de datos. Si se introduce, el parámetro `prefijo` se añadirá delante de los índices numéricos.

Ejemplo

```
<?php
// Inicialización de la matriz que contiene los datos.
$datos=array('nombre' => 'Olivier & Xavier','David + Thomas');
// Construcción de la cadena de la consulta:
// - sin prefijo;
echo http_build_query($datos),'<br />';
// - con prefijo.
echo http_build_query($datos,'v_'),'<br />';
?>
```

Resultado

```
nombre=Olivier+%26+Xavier&0=David+%2B+Thomas
nombre=Olivier+%26+Xavier&v_0=David+%2B+Thomas
```

➤ Existen dos funciones, `urldecode` y `rawurldecode`, que permite decodificar una cadena ya codificada, respectivamente por `urlencode` o `rawurlencode`. Estas funciones de decodificación no necesitan ser llamadas cuando se transmiten datos codificados a través de la URL. En efecto, estos datos se decodifican automáticamente a su llegada.

2. "magic quotes": el regreso

Como explicamos en el capítulo Gestionar los formularios, en las versiones anteriores de PHP, el valor recuperado en el script de llegada podía someterse a la codificación "magic quotes" si la directiva de

configuración `magic_quotes_gpc` estaba en on.

En la versión 5.4, esta función se eliminó definitivamente y ya no se realiza ningún tratamiento sobre los datos para eliminar el posible cifrado.

3. Aplicación a la gestión de sesiones

Esta técnica de transmisión de datos a través de la URL se puede utilizar para administrar las sesiones, transmitiendo la información de la sesión en la URL.

Los principios son los siguientes:

- Cada sesión está marcada con un identificador único.
- Este identificador de sesión se integra sistemáticamente en las URL que permiten navegar entre las diferentes páginas del sitio.
- En cada script (página) concerniente a la gestión de sesiones, empiece por comprobar si se ha llamado al script con una URL que contiene un identificador de sesión; si este no es el caso, el usuario no tiene todavía una sesión abierta (es la primera página que visita) y debe iniciar una sesión. Este inicio de sesión se puede realizar simplemente mediante la asignación de un identificador único o redirigiendo a una página de identificación, si el sitio no acepta usuarios anónimos.

Ejemplo (con dos páginas)

- Script `paginal.php`:

```
<?php
// Inclusión del archivo que contiene las
// funciones generales.
include('funciones.inc');
// Comprobar si la sesión está abierta, es decir si se
// ha transmitido una variable "sesión" mediante la URL.
if (! isset($_GET['sesión']) ) {
    // Variable "sesión" vacía = sin sesión.
    // => iniciar sesión.
    // Para este ejemplo:
    // - identificador de sesión;
    $sesión = identificador_único();
    // - fecha/hora de inicio de sesión;
    $fecha = date('\l\ e d/m/Y a las H:i:s');
    // - mensaje.
    $mensaje = "Nueva sesión: $sesión - $fecha";
} else {
    // Variable "sesión" no vacía = sesión abierta.
    // => recuperar la información de la URL.
    $sesión = $_GET['sesión'];
    $fecha = $_GET['fecha'];
    // Construir un mensaje.
    $mensaje = "Sesión ya iniciada: $sesión - $fecha";
}
// Construcción de los parámetros de la URL: $sesión + $fecha.
// $sesión no necesita codificarse.
$url = "?sesión=$sesión&fecha=".rawurlencode($fecha);
// Determinación de la fecha y de la hora actual (no la
// del inicio de sesión.
$actual = 'Hoy es el día '.date('d/m/Y').
    ' ; son las '.date('H:i:s');
```

```
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head><title>Página 1</title></head>
  <body>
    <div>
      <b>Página 1 - <?php echo<>$actual; ?></b><br />
      <?php echo $mensaje; ?><br />
      <!-- enlace hacia la página 2 -->
      <a href="pagina2.php<?php echo $url; ?>">Página 2</a>
    </div>
  </body>
</html>
```

- Script `pagina2.php` (idéntico a la excepción de la parte HTML):

```
<?php
...
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head><title>Página 2</title></head>
  <body>
    <div>
      <b>Página 2 - <?php echo $actual; ?></b><br />
      <?php echo $mensaje; ?><br />
      <!-- enlace hacia la página 1 -->
      <a href="pagina1.php<?php echo $url; ?>">Página 1</a>
    </div>
  </body>
</html>
```

Resultado

- Primera llamada de la URL `http://.../pagina1.php`:

Página 1 - Hoy es el día 27/06/2013; son las 10:46:18

Nueva sesión: 4d1f27549106035ef491dcce84ee1940 - el 27/06/2013 a las 10:46:18

[Página 2](#)

- Resultado al hacer clic en el enlace a la Página 2:

Página 2 - Hoy es el día 27/06/2013; son las 10:46:54

Sesión ya iniciada: 4d1f27549106035ef491dcce84ee1940 - el 27/06/2013 a las 10:46:18

[Página 1](#)

- Resultado al hacer clic en el enlace a la Página 1:

Página 1 - Hoy es el día 27/06/2013; son las 10:47:38

Sesión ya iniciada: 4d1f27549106035ef491dcce84ee1940 - el 27/06/2013 a las 10:46:18

[Página 2](#)

- Resultado al hacer clic en el botón **Atrás** del navegador:

Página 2 - Hoy es el día 27/06/2013; son las 10:46:54

Sesión ya iniciada: 4d1f27549106035ef491dcce84ee1940 - el 27/06/2013 a las 10:46:18

[Página 1](#)

La página no se ha actualizado ya que está en la caché del navegador. Una solución a este problema se propone en la sección Pasar información por la URL - Notas y conclusión.

El uso de esta técnica se puede ilustrar en una gestión de sesiones con autenticación de usuarios.

Ejemplo

- Script login.php para la autenticación:

```
<?php
// Inclusión del archivo que contiene las
// funciones generales.
include('funciones.inc');
function usuario_existe($identificador,$contraseña) {
    // Conexión y selección de la base de datos.
    $conexión = mysqli_connect();
    mysqli_select_db($conexión,'diane');
    // Definición y ejecución de una consulta preparada.
    $sql = 'SELECT 1 FROM usuarios ';
    $sql .= 'WHERE identificador = ? AND contraseña = ?';
    $consulta = mysqli_stmt_init($conexión);
    $ok = mysqli_stmt_prepare($consulta,$sql);
    $ok = mysqli_stmt_bind_param
        ($consulta,'ss',$identificador,$contraseña);
    $ok = mysqli_stmt_execute($consulta);
    mysqli_stmt_bind_result($consulta,$existe);
    $ok = mysqli_stmt_fetch($consulta);
    mysqli_stmt_free_result($consulta);
    // La identificación tiene éxito si la consulta devuelve
    // una línea (el usuario existe y la contraseña
    // es correcta).
    // Si este es el caso $existe contiene 1, de lo contrario está
    // vacía. Basta con devolverla como un valor booleano.
    return (bool) $existe;
}
// Inicialización de las variables.
$identificador = '';
$contraseña = '';
$mensaje = '';
// Procesamiento del formulario.
if (isset($_POST['conexión'])) {
    // Recuperar la información introducida.
    $identificador = $_POST['identificador'];
    $contraseña = $_POST['contraseña'];
    // Verificar que el usuario existe.
    if (usuario_existe($identificador,$contraseña)) {
        // El usuario existe ...
        // => iniciar la sesión
        $sesión = identificador_único();
        $fecha = date('\e\l d/m/Y a las H:i:s');
        // A continuación, redirigirlo a la página proporcionando la
        // información en la URL.
        $url = "?sesión=$sesión&fecha=".rawurlencode($fecha).
            "&identificador=".rawurlencode($identificador);
        header("location: paginal.php$url");
        exit;
    } else {
        // El usuario no existe ...
        // Normalmente, mostrar un mensaje y proponer de
        // nuevo la identificación.
        $mensaje = 'Identificación incorrecta. Vuelva a intentarlo.';
    }
}
```

```
// Dejar que el formulario se muestre de nuevo ...
}
}
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head><title>Inicio de sesión</title></head>
<body>
<form action="login.php" method="post">
<table border="0">
<tr>
<td align="right">Identificador:</td>
<td><input type="text" Name="identificador" value=
"=php echo hacia_formulario($identificador); ?" /></td>
</tr>
<tr>
<td align="right">Contraseña:</td>
<td><input type="password" Name="contraseña" value=
"=php echo hacia_formulario($contraseña); ?" /></td>
</tr>
<tr>
<td></td>
<td align="right"><input type="submit" name="conexión"
value="Conexión" /></td>
</tr>
</table>
<?php echo $mensaje; ?>
</form>
</body>
</html>
```

- Script `paginal.php` para una página del sitio:

```
<?php
// Inclusión del archivo que contiene las
// funciones generales.
include('funciones.inc');
// Comprobar si la sesión está abierta, es decir si se
// ha transmitido una variable "sesión" mediante la URL.
if (! isset($_GET['sesión']) ) {
// Variable "sesión" vacía = sin sesión.
// => redirigir al usuario a una página de inicio de sesión.
header('location: login.php');
exit;
} else {
// Variable "sesión" no vacía = sesión iniciada.
// => recuperar el resto de información de la URL.
// Para este ejemplo:
// - identificador de sesión;
$sesión = $_GET['sesión'];
// - fecha/hora de inicio de sesión;
$fecha = $_GET['fecha'];
// - identificador del usuario.
$ididentificador = $_GET['identificador'];
// Construir un mensaje.
$mensaje = "Sesión: $sesión - $ididentificador - $fecha";
}
// Construcción de los parámetros de la URL:
// $sesión + $fecha + $ididentificador.
// $sesión no necesita codificarse.
```

```

$url = "?sesión=$sesión&fecha=".rawurlencode($fecha) .
    "&identificador=".rawurlencode($identificador);
// Determinación de la fecha y de la hora actual (no la
// del inicio de sesión).
$actual = 'Hoy es el día '.date('d/m/Y') .
    '; son las '.date('H:i:s');
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
    <head><title>Página 1</title></head>
    <body>
        <div>
            <b>Página 1 - <?php echo $actual; ?></b><br />
            <?php echo $mensaje; ?><br />
            <!-- enlace hacia la página 2 -->
            <a href="pagina2.php<?php echo $url; ?>">Página 2</a>
        </div>
    </body>
</html>

```

La primera llamada de la URL `http://.../pagina1.php` provoca la redirección del usuario a la página de identificación. Después identificarse con éxito, el usuario vuelve a la página 1 que muestra la siguiente información:

Página 1 - Hoy es el día 27/06/2013; son las 10:55:18

Sesión: 5bca7a5262a907c23b3bda2be215d4ad - heurtel - el 27/06/2013 a las 10:55:18

[Página 2](#)

El acceso a las páginas del sitio se encuentra protegido, o más bien, parece protegido (más información en la sección siguiente).



El script `login.php` muestra la posibilidad de colocar parámetros detrás de la URL pasada a la función `header`.

4. Notas y conclusión

Los usuarios maliciosos y/o malintencionados

Para ambos ejemplos, un usuario (malicioso y/o malintencionado) que llama a una URL `http://.../pagina1.php?sesión=abc` accede sin problemas a la página solicitada saltándose el mecanismo de inicio de sesión y, especialmente, el paso por la página de identificación en el segundo ejemplo:

Página 1 - Hoy es el día 27/06/2013; son las 11:00:58

Sesión: abc - -

[Página 2](#)

De hecho, actualmente, el script sólo comprueba la existencia de un valor para el parámetro de sesión en la URL, sin comprobar si el valor en cuestión corresponde a una sesión real. Para ello, debemos conservar, del lado del servidor, en un archivo o una base de datos, el seguimiento de las sesiones realmente abiertas, es decir, el rastro de los identificadores de sesión realmente asignados por la aplicación. Además, debe haber un mecanismo de tiempo de vida (30 minutos, 1 hora, 6 horas...) que garantice que un usuario no utiliza un identificador de sesión asignado previamente.

No transmitirlo todo en la URL

El mecanismo utilizado para conservar, del lado del servidor, el rastro de sesiones abiertas también se puede utilizar para almacenar información adicional sobre la sesión y no estar obligado así a pasar todos los valores por la URL: sólo el identificador de sesión puede serlo. En ese momento, al comienzo de cada script, basta con utilizar el identificador de sesión para recuperar información adicional de la sesión y, al final del script, volver a registrar, del lado del servidor, la información de sesión que ha cambiado.

En el caso de procesamiento de un formulario

Al procesar un formulario, el script no se llama mediante un enlace `<a href...>` como acabamos de ver, sino por el atributo de `action` de la etiqueta `<form>`.

La técnica de transmisión de información de la sesión a través la URL se puede utilizar en la URL del atributo `action`.

Ejemplo

```
<?php
...
$url = "?sesión=$sesión";
...
?>
...
<form action="pagina1.php<?php echo $url; ?>" method="post">
...
```

El script llamado (`pagina1.php` en nuestro ejemplo) puede recuperar el identificador de la sesión en la matriz `$_GET` y los valores del formulario en la matriz `$_POST`.

Veremos en la sección Pasar información por un campo de formulario oculto, que la información de sesión también puede, en este caso, transmitirse en el formulario.

Forzar la actualización de una página

Para resolver el problema de las páginas que no se actualizan debido a que siguen presentes en la memoria caché del navegador, se deben enviar encabezados adicionales en la página HTML con la función `header`.

Ejemplo

```
// HTTP 1.0
header("Pragma: no-cache");
// HTTP 1.1
header("Cache-Control: no-store, no-cache, must-revalidate");
```

Conclusión

Utilizar la URL para transmitir información de sesión es posible, pero requiere una gran cantidad de código para construir algo sólido.

No indicamos voluntariamente ningún ejemplo más completo para la gestión de sesiones con esta técnica, ya que desde PHP 4, las características de administración de sesiones se introdujeron para simplificar el desarrollo y ofrecer soluciones simples a los problemas ya mencionados: por tanto hay que utilizarlas (ver sección Utilizar la gestión de sesiones de PHP).

La técnica presentada en esta sección debe tomarse como un método para proporcionar información simple de una página a otra, sin buscar una seguridad importante.

Pasar información a través de un campo de formulario oculto

1. Principio

En el capítulo Gestionar los formularios vimos que la información introducida en un formulario se transmitía al script encargado del procesamiento y podía mostrarse en una nueva página.

Este método se puede utilizar para la transmisión de otro tipo de información no introducida por el usuario, por lo general, colocándola en un campo de formulario oculto.

Ejemplo

- Script paginal.php

```
<?php
// Inicialización de una variable.
$nombre='Olivier';
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head><title>Página 1</title></head>
<body>
<!-- enlace hacia la página 2 con un botón de formulario -->
<form action="pagina2.php" method="post">
<div>
<!-- la información transmitida está oculta -->
<input type="hidden" Name="nombre" value= "<?php echo $nombre; ?>" />
<br /><input type="submit" name="pagina2" value="Página 2" />
</div>
</form>
</body>
</html>
```

- Código fuente de la página en el navegador

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head><title>Página 1</title></head>
<body>
<!-- enlace hacia la página 2 con un botón de formulario -->
<form action="pagina2.php" method="post">
<div>
<!-- la información transmitida está oculta -->
<input type="hidden" Name="nombre" value= "Olivier" />
<br /><input type="submit" name="pagina2" value="Página 2" />
</div>
</form>
</body>
</html>
```

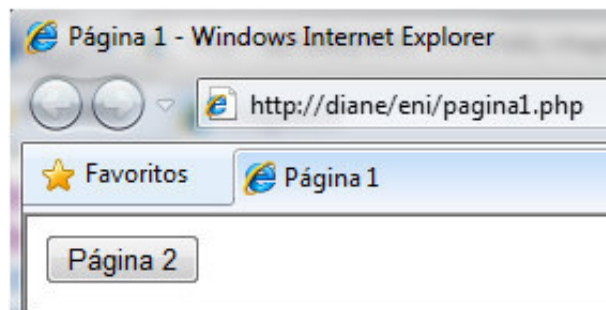
- Script pagina2.php

```
<?php
$nombre = $_POST['nombre'];
```

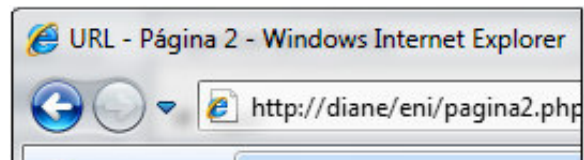
```
echo $nombre;  
?>
```

Resultado

- Presentación de la página 1:



- Resultado al hacer clic en el botón:



Con el formulario, no hay ningún problema de codificación (se hace automáticamente).

2. Aplicación a la gestión de sesiones

Esta técnica de transmisión de datos a través de un formulario se puede utilizar para la administración de sesiones, transmitiendo la información de la sesión en los campos ocultos de un formulario.

Los principios aplicados son los mismos que con la URL.

Ejemplo (con dos páginas)

- Script `pagina1.php`:

```
<?php  
//Inclusión del archivo que contiene las  
//funciones generales.  
include('funciones.inc');  
//Comprobar si la sesión está abierta, es decir si se  
//ha transmitido una variable "sesión" mediante la URL.  
if(! isset($_POST['sesión']) ) {  
    // Variable "sesión" vacía = sin sesión.  
    // => iniciar sesión.  
    // Para este ejemplo:  
    // - identificador de sesión;  
    $sesión = identificador_único();  
    // - fecha/hora de inicio de sesión;  
    $fecha = date('\l\e d/m/Y a las H:i:s');
```

```
// - mensaje.
$mensaje = "Nueva sesión: $sesión - $fecha";
} else {
//Variable "sesión" no vacía = sesión abierta.
//=> recuperar la información de la URL.
$sesión = $_POST['sesión'];
$fecha = $_POST['fecha'];
//Construir un mensaje.
$mensaje = "Sesión ya iniciada: $sesión - $fecha";
}
//Determinación de la fecha y de la hora actual (no la
//del inicio de sesión).
$actual = 'Hoy es el día '.date('d/m/Y').
        ' ; son las '.date('H:i:s');
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head><title>Página 1</title></head>
<body>
<div>
<b>Página 1 - <?php echo $actual; ?></b><br />
<?php echo $mensaje; ?><br />
</div>
<!-- enlace hacia otras páginas con un formulario -->
<form action="pagina2.php" method="post">
<div>
<input type="hidden" Name="sesión"
value= "<?php echo $sesión; ?>" />
<input type="hidden" Name="fecha"
value= "<?php echo $fecha; ?>" />
<input type="submit" name="pagina2" value="Página 2" />
</div>
</form>
</body>
</html>
```

- Script pagina2.php (idéntico a la excepción de la parte HTML):

```
<?php
...
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head><title>Página 2</title></head>
<body>
<div>
<b>Página 2 - <?php echo $actual; ?></b><br />
<?php echo $mensaje; ?><br />
</div>
<!-- enlace hacia otras páginas con un formulario -->
<form action="pagina1.php" method="post">
<div>
<input type="hidden" Name="sesión"
value= "<?php echo $sesión; ?>" />
<input type="hidden" Name="fecha"
value= "<?php echo $fecha; ?>" />
<input type="submit" name="pagina1" value="Página 1" />
</div>
```

```
</form>
</body>
</html>
```

Resultado

- Primera llamada de la URL `http://.../pagina1.php`:

Página 1 - Hoy es el día 27/06/2013; son las 11:06:24

Nueva sesión: 3d810c655ebc95f343babbbalfe100f7 - el 27/06/2013 a las 11:06:24

- Resultado al hacer clic en el botón **Página 2**:

Página 2 - Hoy es el día 27/06/2013; son las 11:07:05

Sesión ya iniciada: 3d810c655ebc95f343babbbalfe100f7 - el 27/06/2013 a las 11:06:24

- Resultado al hacer clic en el botón **Página 1**:

Página 1 - Hoy es el día 27/06/2013; son las 11:07:27

Sesión ya iniciada: 3d810c655ebc95f343babbbalfe100f7 - el 27/06/2013 a las 11:06:24

- Resultado al hacer clic en el botón **Atrás** del navegador:

Página 2 - Hoy es el día 27/06/2013; son las 11:07:05

Sesión ya iniciada: 3d810c655ebc95f343babbbalfe100f7 - el 27/06/2013 a las 11:06:24

La página no se ha actualizado ya que está en la caché del navegador. Una solución a este problema se propone en la siguiente sección Notas y conclusión.

El inicio de sesión, pasando por una página de identificación se puede realizar siguiendo el mismo principio que en la sección Pasar información a través de la URL. El único problema con la redirección de la página de identificación a otra página del sitio, después de una conexión exitosa es que el uso de la función `header` no permite transmitir información a través del método del formulario. La forma más sencilla consiste entonces en invertir ligeramente la lógica y un hacer que el script de la página 1 procese el formulario de identificación; este último es responsable de reenviar al usuario a la página de identificación en caso de fallo de conexión. En la práctica, este enfoque no es muy elegante desde el punto de vista de la organización del código.

3. Notas y conclusión

Los usuarios maliciosos y/o malintencionados

Un usuario (malicioso y/o malintencionado) puede crear una página con su propio formulario, incluyendo un campo llamado "sesión", que contenga un valor cualquiera y que llame al script `pagina1.php`; en este caso tendría acceso sin problemas a la página 1, saltándose el mecanismo de inicio de sesión y, especialmente, el paso por la página de identificación del segundo ejemplo.

Como en la sección Pasar información a través de la URL, la solución consiste en hacer un seguimiento, del lado del servidor, de las sesiones realmente abiertas.

No transmitirlo todo en el formulario

Como en la sección Pasar información a través de la URL, el mecanismo utilizado para mantener, del lado del servidor, el rastro de las sesiones abiertas, puede utilizarse también para conservar información adicional en la sesión y, de este modo, no estar obligado a pasar todos los valores en el formulario.

Caso de la navegación a través de la etiqueta <a href...>

La técnica propuesta en esta sección no es fácil de combinar con la navegación a través de la etiqueta `<a href...>`.

Hay varias soluciones posibles:

- Establecer una navegación a base de botones; no es muy elegante y, sobre todo, no respeta el espíritu de navegación por la Web.
- Utilizar ambas técnicas; lo que puede provocar que la programación sea demasiado pesada, excepto si se utiliza sistemáticamente el método `GET` para el procesamiento de los formularios: independientemente de si datos se transmiten a través de la URL o de formulario, se pueden recuperar en `$_GET`.

Conclusión

Es posible utilizar los formulario para transmitir información de la sesión, pero esto requiere una gran cantidad de código para construir algo sólido y puede llevar a una interfaz de navegación que no esté en armonía con el espíritu de la Web.

Como para la administración de sesiones a través de la URL, no iremos más lejos en los ejemplos: por tanto, se deben utilizar las características de administración de sesiones de PHP (ver sección Utilizar la gestión de sesiones de PHP).

La técnica presentada en esta sección debe considerarse como una técnica que permite la transmisión puntual de información adicional en un formulario ya presente. Por contra, crear un formulario con un botón y algunos campos ocultos, especialmente para transmitir información, no es muy elegante en términos de organización del código.

Utilizar cookies

1. Principio

Una cookie es un pequeño archivo depositado por un sitio en el equipo del usuario y puede contener información.

Las cookies se reenvían automáticamente al servidor Web por el navegador cuando el usuario navega por las páginas del sitio.

PHP permite recuperar muy fácilmente, en las variables, los datos almacenados en la cookie.

La función `setcookie` permite depositar una cookie en el equipo del usuario.

Sintaxis

```
booleano setcookie(cadena nombre [, cadena valor [, entero vencimiento [,  
cadena ruta [, cadena dominio [, booleano asegurado[,  
booleano http_únicamente]]]]])
```

Con

nombre	Nombre de la cookie.
valor	Valor almacenado en la cookie.
vencimiento	Fecha de caducidad de la cookie (marca de tiempo Unix).
ruta	Ruta de acceso en el servidor donde está disponible la cookie. Colocar / para hacer que la cookie esté disponible en todo el dominio o /rep/ para que la cookie esté disponible en el directorio /rep/ del dominio y todos sus subdirectorios. De forma predeterminada, igual al directorio desde donde se envió la cookie.
dominio	Dominio al que se reenvía la cookie. .miSitio.com (con un punto al principio) permite, por ejemplo, hacer que la cookie esté disponible para todos los subdominios demiSitio.com.
asegurado	Incluir TRUE para indicar que la cookie debe transmitirse únicamente en una conexión segura (FALSE por defecto).
http_únicamente	Incluir TRUE para indicar que la cookie debe transmitirse únicamente por el protocolo HTTP (FALSE por defecto). Apareció con la versión 5.2.0.

Si la función sólo se llama con el parámetro `nombre`, la cookie que lleva este nombre se elimina del equipo del usuario. Si los parámetros `dominio` y `ruta` se han especificado durante el envío de la cookie, debe especificarlos de manera idéntica para eliminar la cookie (poner fecha de caducidad en el pasado).

Si se especifica el parámetro `valor`, se envía una cookie con el nombre `nombre` que contiene el valor `valor` al equipo del usuario; si ya existe una cookie con este nombre, esta última se actualiza con el nuevo valor.

El parámetro `vencimiento` permite determinar la fecha de vencimiento de la cookie (y por lo tanto la fecha de su eliminación del equipo del usuario); si este parámetro no se especifica (o es igual a 0) la

cookie expira al final de la sesión, es decir, cuando el usuario abandona el sitio.

Las cookies se envían en el encabezado de la página. Al igual que la función `header`, la función `setcookie` debe llamarse antes de cualquier instrucción (PHP o HTML), esta última tiene el efecto de iniciar la construcción de la página HTML. En caso de problema, se muestra un mensaje del siguiente tipo:

Warning: Cannot modify header information - headers already sent
in `/app/scripts/test.php` on line 3

La función `setcookie` devuelve `TRUE` si la instrucción se ha podido ejecutar (no se ha transmitido todavía ningún dato) y `FALSE` en caso contrario. Por contra, el código de retorno de la función no da ninguna información sobre el hecho de que la cookie haya podido realmente depositarse en el equipo del usuario: si este último rechaza las cookies, la función `setcookie` devuelve al menos `TRUE` aunque la cookie no se haya depositado.

- Las cookies las gestiona el sitio; dos cookies de sitios distintos pueden tener el mismo nombre. La cookie se deposita en el equipo del usuario por la función `setcookie` y luego se reenvía durante la visita de cualquier página del sitio.

Ejemplo

```
<?php
// Envío de una cookie llamada "nombre" que contiene
// el valor "Olivier" y expira al final de
// la sesión.
$ok = setcookie('nombre','Olivier');
// Idem pero expira en la fecha (time() en segundos)
// más 30 veces 24 veces 3600 segundos (es 30 días).
$ok = setcookie('nombre','Olivier',time()+(30*24*3600));
// Eliminación de la cookie llamada 'nombre'.
$ok = setcookie('nombre');
?>
```

Cuando la cookie se reenvía al servidor Web por el navegador, a petición de una página PHP, el valor de la cookie está disponible en una variable de PHP como un mecanismo idéntico al aplicado para los formularios y URL.

El valor de cada cookie enviada por el navegador se registra automáticamente en la matriz asociativa `$_COOKIE`: la clave de la matriz es igual al nombre de la cookie.

- Las variables de cookie también están disponibles en la matriz asociativa `$_REQUEST`.

Ejemplo

- Script `paginal.php` que deposita dos cookies:

```
<?php
// La primera cookie expira al final de la sesión.
$ok1 = setcookie('nombre','Olivier');
// Segunda cookie expira en 30 días.
$ok2 = setcookie('apellido','HEURTEL',time()+(30*24*3600));
// Resultado.
if ($ok1 and $ok2) {
    $mensaje = 'Cookies depositadas (al menos, a priori)';
}
```



```

} else {
    $mensaje = 'Una de las cookies no se ha podido depositar';
}
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head><title>Página 1</title></head>
<body>
<div>
<?php echo $mensaje;?><br />
<!-- enlace hacia la página 2 -->
<a href="pagina2.php">Página 2</a>
</div>
</body>
</html>

```

- Script `pagina2.php` que muestra el valor de las dos cookies:

```

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head><title>Página 2</title></head>
<body>
<div>
<?php
if ( isset($_COOKIE["nombre"]) ) {
    echo "\$_COOKIE["nombre"] = {$_COOKIE['nombre']}<br>";
} else {
    echo "\$_COOKIE["nombre"] = <br>";
}
if ( isset($_COOKIE["apellido"]) ) {
    echo "\$_COOKIE["apellido"] = {$_COOKIE['apellido']}<br>";
} else {
    echo "\$_COOKIE["apellido"] = <br>";
}
?>
</div>
</body>
</html>

```

Resultado

- Presentación de la página 1:

Cookies depositadas (al menos, a priori)

Página 2

- Resultado al hacer clic en el enlace

```

$_COOKIE["nombre"] = Olivier
$_COOKIE["apellido"] = HEURTEL

```

- Resultado de un regreso, antes de treinta días, a la página 2 del mismo sitio:

```

$_COOKIE["nombre"] =
$_COOKIE["apellido"] = HEURTEL

```

La cookie de duración igual a la sesión ya no existe después de abandonar la sesión y la información se pierde; la información almacenada en la otra cookie sigue estando disponible (dentro de los límites de

su vida útil).

Es posible almacenar cualquier cadena en la cookie sin tener que preocuparse por una posible codificación/decodificación: la codificación y decodificación se realiza automáticamente.

La cookie se deposita en el equipo del usuario mediante la función `setcookie`, a continuación, se reenvía posteriormente al visitar cualquier página del sitio (en función del valor de los parámetros `ruta` y `dominio`); la cookie no está inmediatamente disponible en la página que la envía.

Ejemplo

```
<?php
// Valor de la cookie antes.
$antes = (isset($_COOKIE ['hora']))?$_COOKIE ['hora']:'';
// Depósito de la cookie que expira al final de la sesión.
$ok = setcookie('hora',date('H:i:s'));
// Valor de la cookie después.
$después = (isset($_COOKIE ['hora']))?$_COOKIE ['hora']:'';
// Hora actual.
$sactual = date('H:i:s');
// Visualización.
echo "Actual: $sactual<br />";
echo "Antes: $antes<br />";
echo "Después: $después<br />";
?>
```

Resultado de la primera llamada

```
Actual: 14:53:18
Antes:
Después:
```

En la primera llamada, la cookie no existe antes (esto es normal) y todavía no existe después, ya que simplemente se ha enviado, pero todavía no tiene efecto (formas imaginativas de decirlo).

Resultado de la segunda llamada (en la misma sesión)

```
Actual: 14:53:30
Antes: 14:53:18
Después: 14:53:18
```

En la segunda llamada, el valor de la cookie está disponible desde el comienzo del script (la cookie se devuelve con la solicitud para la página), y tiene un valor que se corresponde con el momento en que se envió; por contra, su valor después no refleja inmediatamente la realidad (mismo principio que para el depósito inicial: la cookie todavía no es "efectiva").

Resultado de la tercera llamada (en la misma sesión)

```
Actual: 14:53:41
Antes: 14:53:30
Después: 14:53:30
```

En la tercera llamada, la cookie tiene un valor que corresponde a su actualización en la segunda llamada.

Una consecuencia de este modo es que no es posible, inmediatamente después de haber depositado la cookie, comprobar si la cookie ha sido aceptada o no por el equipo.

Para determinar si un equipo acepta las cookies, debe depositar una cookie y volver a cargar una

página en la que la presencia o ausencia de la cookie determine si el equipo acepta las cookies.

Ejemplo de script probar_cookie.php que permite realizar esta comprobación

```
<?php
// Comprobar si es la segunda llamada de la página.
if (! isset($_GET['vuelta'])) {
    // No ...
    // Depositar la cookie.
    setcookie('prueba','prueba');
    // Y volver a cargar la página con una información en
    // la URL indicando que es la segunda vez que se pasa.
    header('Location: probar_cookie.php?vuelta=1');
} else {
    // Sí ...
    // Comprobar si la cookie es "efectiva".
    if (isset($_COOKIE['prueba'])) { // sí ...
        echo 'Cookie aceptada';
    } else { // no ...
        echo 'Cookie rechazada';
    }
}
?>
```

Es posible recuperar una matriz como valor de la cookie en reserva de utilizar una notación de tipo matriz en el depósito de la cookie.

Ejemplo (basado en el ejemplo anterior)

```
<?php
// Inclusión del archivo que contiene las funciones generales.
include('funciones.inc');
// Comprobar si es la segunda llamada de la página.
if (! isset($_GET['vuelta'])) {
    // No ...
    // Depositar la cookie.
    setcookie('prueba[0]','cero');
    setcookie('prueba[1]','uno');
    // Y volver a cargar la página con una información en
    // la URL indicando que es la segunda vez que se pasa.
    header('Location: probar_cookie.php?vuelta=1');
} else {
    // Sí ...
    // Comprobar si la cookie es "efectiva".
    if (isset($_COOKIE['prueba'])) { // sí ...
        echo 'Cookie aceptada<br />';
        mostrar_matriz($_COOKIE['prueba']);
    } else { // no ...
        echo 'Cookie rechazada';
    }
}
?>
```

Resultado

```
Cookie aceptada
0 = cero
1 = uno
```

En la práctica, en realidad hay varias cookies depositadas en el ordenador del usuario, pero los valores se recuperan desde el script PHP en forma de una matriz. Para depositar una sola cookie que contenga varios valores, puede proceder por concatenación, o utilizar una función como `implode` (y `explode` al devolver la cookie).

2. "magic quotes": el regreso

Como ya hemos explicado en el capítulo Gestionar los formularios, en las versiones anteriores de PHP, el valor recuperado en el script de llegada podía someterse a la codificación "magic quotes" si la directiva de configuración `magic_quotes_gpc` (*Get/Post/Cookie*) estaba en `on`.

En la versión 5.4, esta función se eliminó definitivamente y ya no se realiza ningún tratamiento sobre los datos para eliminar el posible cifrado.

3. Aplicación a la gestión de sesiones

Las cookies tienen la gran ventaja, sobre las técnicas descritas hasta ahora, de ser totalmente independientes de la etiqueta de navegación `` y de la gestión de formularios: administrar las sesiones utilizando cookies permite superar las desventajas asociadas con los otros dos métodos.

Tienen, por contra, una gran desventaja: pueden ser rechazadas por los internautas.

Los principios de aplicación son los mismos que para la gestión a través de URL.

Vamos a ilustrar estos principios con la ayuda dos páginas (emitiendo la hipótesis de que las cookies se aceptan).

Ejemplo (con dos páginas y partiendo de la hipótesis de que las cookies se aceptan)

- Script `pagina1.php`:

```
<?php
// Inclusión del archivo que contiene las
// funciones generales.
include('funciones.inc');
// Comprobar si la sesión está abierta, es decir si se
// ha transmitido una variable "sesión" mediante la cookie.
if (! isset($_COOKIE['sesión']) ) {
    // Variable "sesión" vacía = sin sesión.
    // => iniciar sesión.
    // Para este ejemplo:
    // - identificador de sesión;
    $sesión = identificador_único();
    // - fecha/hora de inicio de sesión.
    $fecha = date('\e\l d/m/Y a las H:i:s');
    // Depositar dos cookies para almacenar esta información.
    // Tiempo de vida = ;la sesión exactamente!
    setcookie('sesión', $sesión);
    setcookie('fecha', $fecha);
    // - mensaje.
    $mensaje = "Nueva sesión: $sesión - $fecha";
} else {
    // Variable "sesión" no vacía = sesión abierta.
    // => recuperar la información de la URL.
    $sesión = $_COOKIE['sesión'];
```

```

$fecha = $_COOKIE['fecha'];
// Construir un mensaje.
$mensaje = "Sesión ya iniciada: $sesión - $fecha";
}
// Determinación de la fecha y de la hora actual (no la
// del inicio de sesión).
$actual = 'Hoy es el día '.date('d/m/Y').
        ' ; son las '.date('H:i:s');
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head><title>Página 1</title></head>
<body>
<div>
<b>Página 1 - <?php echo $actual; ?></b><br />
<?php echo $mensaje; ?><br />
<!-- enlace hacia la página 2 -->
<a href="pagina2.php<?php echo $url; ?>">Página 2</a>
</div>
</body>
</html>

```

- Script `pagina2.php` (idéntico a la excepción de la parte HTML):

```

<?php
...
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head><title>Página 2</title></head>
<body>
<div>
<b>Página 2 - <?php echo $actual; ?></b><br />
<?php echo $mensaje; ?><br />
<!-- enlace hacia la página 1 -->
<a href="pagina1.php<?php echo $url; ?>">Página 1</a>
</div>
</body>
</html>

```

Resultado

- Primera llamada de la URL `http://.../pagina1.php`:

Página 1 - Hoy es el día 27/06/2013; son las 13:29:25

Nueva sesión: `abcee2d3b8e2f40ca9eafae438cd7bf1` - el 27/06/2013 a las 13:29:25

[Página 2](#)

- Resultado al hacer clic en el enlace a la Página 2:

Página 2 - Hoy es el día 27/06/2013; son las 13:30:05

Sesión ya iniciada: `abcee2d3b8e2f40ca9eafae438cd7bf1` - el 27/06/2013 a las 13:29:25

[Página 1](#)

- Resultado al hacer clic en el enlace a la Página 1:

Página 1 - Hoy es el día 27/06/2013; son las 13:31:02

Si hace clic en el botón **Atrás** de su navegador, la página no se actualiza porque está en la caché del navegador. Una solución a este problema se propone en la siguiente sección Notas y conclusión.

- Con las cookies, es posible navegar entre las páginas y gestionar formularios sin preocuparse de la información de sesión: se transmiten en paralelo.

El uso de esta técnica se puede ilustrar en una gestión de sesiones con autenticación de usuarios.

Ejemplo

- Script login.php para la autenticación:

```
<?php
// Inclusión del archivo que contiene las
// funciones generales.
include('funciones.inc');
function usuario_existe($identificador,$contraseña) {
    // Conexión y selección de la base de datos.
    $conexión = mysqli_connect();
    mysqli_select_db($conexión,'diane');
    // Definición y ejecución de una consulta preparada.
    $sql = 'SELECT 1 FROM usuarios ';
    $sql .= 'WHERE identificador = ? AND contraseña = ?';
    $consulta = mysqli_stmt_init($conexión);
    $ok = mysqli_stmt_prepare($consulta,$sql);
    $ok = mysqli_stmt_bind_param
        ($consulta,'ss',$identificador,$contraseña);
    $ok = mysqli_stmt_execute($consulta);
    mysqli_stmt_bind_result($consulta,$existe);
    $ok = mysqli_stmt_fetch($consulta);
    mysqli_stmt_free_result($consulta);
    // La identificación tiene éxito si la consulta devuelve
    // una línea (el usuario existe y la contraseña es correcta).
    // Si este es el caso $existe contiene 1, de lo contrario está
    // vacía. Basta con devolverla como un valor booleano.
    return (bool) $existe;
}
// Inicialización de las variables.
$identificador = '';
$contraseña = '';
$mensaje = '';
// Procesamiento del formulario.
if (isset($_POST['conexión'])) {
    // Recuperar la información introducida.
    $identificador = $_POST['identificador'];
    $contraseña = $_POST['contraseña'];
    // Verificar que el usuario existe.
    if (usuario_existe($identificador,$contraseña)) {
        // El usuario existe ...
        // => iniciar sesión.
        $sesión = identificador_único();
        $fecha = date('\e\l d/m/Y a las H:i:s');
        // Depositar tres cookies para almacenar esta información.
        // Tiempo de vida = ¡la sesión exactamente!
        setcookie('sesión',$sesión);
        setcookie('fecha',$fecha);
    }
}
```

```

        setcookie('identificador',$identificador);
        // A continuación, redirigirle hacia la página 1.
        header('location: paginal.php');
        exit;
    } else {
        // El usuario no existe ...
        // Normalmente, mostrar un mensaje y proponer de
        // nuevo la identificación.
        $mensaje = 'Identificación incorrecta. Vuelva a intentarlo.';
        // Dejar que el formulario se muestre de nuevo ...
    }
}
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head><title>Inicio de sesión</title></head>
<body>
    <form action="login.php" method="post">
    <table border="0">
    <tr>
        <td align="right">Identificador:</td>
        <td><input type="text" Name="identificador" value=
            "<?php echo hacia_formulario($identificador); ?>" /></td>
    </tr>
    <tr>
        <td align="right">Contraseña:</td>
        <td><input type="password" Name="contraseña" value=
            "<?php echo hacia_formulario($contraseña); ?>" /></td>
    </tr>
    <tr>
        <td></td>
        <td align="right"><input type="submit" name="conexión"
            value="Conexión" /></td>
    </tr>
    </table>
    <?php echo $mensaje; ?>
    </form>
</body>
</html>

```

- Script paginal.php para una página del sitio:

```

<?php
// Inclusión del archivo que contiene las
// funciones generales.
include('funciones.inc');
// Comprobar si la sesión está abierta, es decir si se
// ha transmitido una variable "sesión" mediante la cookie.
if (! isset($_COOKIE['sesión']) ) {
    // Variable "sesión" vacía = sin sesión.
    // => redirigir al usuario a una página de inicio de sesión.
    header('location: login.php');
    exit;
} else {
    // Variable "sesión" no vacía = sesión iniciada.
    // => recuperar la información de las cookies.
    // Para este ejemplo:
    // - identificador de sesión;
    $sesión = $_COOKIE['sesión'];
    // - fecha/hora de inicio de sesión;

```

```

$fecha = $_COOKIE['fecha'];
// - identificador del usuario.
$identificador = $_COOKIE['identificador'];
// Construir un mensaje.
$mensaje = "Sesión: $sesión - $identificador - $fecha";
}
// Determinación de la fecha y de la hora actual (no la
// del inicio de sesión).
$actual = 'Hoy es el día '.date('d/m/Y').
        ' ; son las '.date('H:i:s');

?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head><title>Página 1</title></head>
<body>
<div>
<b>Página 1 - <?php echo $actual; ?></b><br />
<?php echo $mensaje; ?><br />
<!-- enlace hacia la página 2 -->
<a href="pagina2.php">Página 2</a>
</div>
</body>
</html>

```

La primera llamada de la URL `http://.../pagina1.php` provoca la redirección del usuario a la página de identificación, después de haberse identificado con éxito, el usuario vuelve a una página que muestra la siguiente información:

Página 1 - Hoy es el día 27/06/2013; son las 13:34:23

Sesión: 8ca88eb27ec21263d50cca2144a72458 - heurtel - el 27/06/2013 a las 13:34:23

[Página 2](#)

4. Notas y conclusión

Los usuarios maliciosos y/o malintencionados

Aunque esto es un poco más complejo de implementar, un usuario (malicioso y/o malintencionado) puede acceder a una página, saltándose el proceso de inicio de sesión, especialmente el paso por la página de identificación en el segundo ejemplo.

Como en los otros dos casos (URL y formulario), la solución consiste en hacer un seguimiento, del lado del servidor, de las sesiones realmente abiertas.

No transmitirlo todo en las cookies

Las cookies tienen sus limitaciones:

- Los navegadores limitan el número de cookies por dominio.
- El tamaño de una cookie es limitado.

Una vez más, el mecanismo utilizado para conservar, del lado del servidor, el rastro de las sesiones abiertas también se puede utilizar para almacenar información adicional acerca de la sesión. De este modo, no está obligado a pasar todos los valores en las cookies.

Conclusión

Las cookies son muy útiles para la gestión de sesiones, pero no son necesariamente aceptadas por todos los usuarios.

Para un sitio destinado a usuarios identificados (suscritos, registrados), es posible imponer la aceptación de las cookies para ser miembro del sitio, lo importante es informar adecuadamente a los usuarios de que la cookie sólo permanecerá durante el tiempo que dure la sesión.

Para un sitio para el público en general, probablemente no sea aceptable confiar en la gestión de las sesiones a través de las cookies.

De todos los métodos descritos hasta ahora, la técnica "cookie" y la técnica "URL" se pueden considerar como las menos restrictivas. Una gestión de sesiones, en la que sólo el identificador de sesión debe transmitirse de una página a otra (el resto de información se almacena en el servidor) se puede implementar con relativa facilidad mediante el uso de cookies, si el usuario las acepta y utilizando la URL en el caso contrario. Esto es exactamente lo que propone la administración de sesiones de PHP que vamos a ver ahora: por lo tanto, no nos entretendremos en reconstruir lo que ya existe.

También veremos en este capítulo (véase la sección Conservar la información de una visita a otra), que la cookie es una buena herramienta (siempre sujeta a la aceptación del usuario) para almacenar información de un sesión a otra (utilizando una cookie con un tiempo de vida especificado en su creación).

Índice

Información

[Título, autor...](#)

Introducción

[Objetivo del libro](#)[Breve historia de PHP](#)[¿Dónde conseguir PHP?](#)[Convenciones de escritura](#)

Información general sobre PHP

[Variables, constantes, tipos y matrices](#)[Operadores](#)[Estructuras de control](#)[Funciones y clases](#)[Gestión de formularios](#)[Acceder a las bases de datos](#)[Administrar las sesiones](#)[Enviar un correo electrónico](#)[Gestión de archivos](#)[Administrar los errores en un script PHP](#)[Anexo](#)

Convenciones de escritura

La sintaxis de las funciones se describe de la siguiente manera en este libro:

```
tipo_retorno nombre_función(tipo_parámetro nombre_parámetro)
```

tipo_retorno Tipo de valor devuelto (retorno) de la función.

nombre_función Nombre de la función.

tipo_parámetro Tipo de parámetro aceptado por la función.

nombre_parámetro Nombre que se da al parámetro.

Los tipos de datos posibles se presentarán en el capítulo Variables, constantes, tipos y matrices. En el caso de que la función acepte un parámetro de cualquier tipo y/o devuelva un valor de cualquier tipo, se utiliza el término *mixto*.

Si la función no devuelve ningún valor, la información *tipo_retorno* se omite.

Ejemplo

```
nombre_función(tipo_parámetro nombre_parámetro)
```

Si la función no toma ningún parámetro, la información *tipo_parámetro* y *nombre_parámetro* se omiten.

Ejemplo

```
tipo_retorno nombre_función()
```

Los parámetros opcionales se indican entre corchetes ([]).

Ejemplo

```
tipo_retorno nombre_función([tipo_parámetro nombre_parámetro])
```

Si la función acepta varios parámetros, estos últimos se indican, separados por una coma, siguiendo la misma convención.

Ejemplo

```
tipo_retorno nombre_función(tipo_parámetro_1  
nombre_parámetro_1, tipo_parámetro_2 nombre_parámetro_2)
```

Si un parámetro se puede repetir un número indefinido de veces, simplemente sigue la secuencia [, ...].

Ejemplo

```
tipo_retorno nombre_función(tipo_parámetro nombre_parámetro[, ...])
```

Utilizar la gestión de sesiones de PHP

1. Principios

Desde la versión 4, PHP ofrece un conjunto de funciones que facilitan la gestión de sesiones. Los principios son los siguientes:

- Un identificador único se asigna automáticamente a cada sesión.
- Este identificador único se transmite de una página a otra, bien mediante una cookie (si el equipo acepta cookies) o a través de la URL, en caso contrario; en cualquier caso, es PHP quien elige automáticamente el enfoque correcto y garantiza esta transferencia (con algunas reservas en relación con la configuración).
- Las variables para las que desea conservar el valor de una página a otra durante la duración de la sesión se indican a PHP, que se encarga automáticamente de devolver los valores al comienzo del script y guardarlos al final del script. De forma predeterminada, la copia se efectúa en el disco, en los archivos temporales, pero es posible, con un poco de desarrollo, guardar estos valores en una base de datos mediante PHP. Estas variables se denominan "variables de sesión".

En resumen, PHP se encarga de toda la gestión.

2. Implementación

Las principales funciones del módulo de gestión de sesiones son las siguientes:

Nombre	Función
<code>session_start</code>	Abre una nueva sesión o reactiva la sesión actual.
<code>session_id</code>	Devuelve (o modifica) el identificador de la sesión.
<code>session_name</code>	Devuelve (o modifica) el nombre de la variable utilizada para almacenar el identificador de la sesión.
<code>session_unset</code>	Elimina el registro de todas las variables de la sesión.
<code>session_destroy</code>	Elimina la sesión.
<code>session_status</code>	Devuelve el estado actual de una sesión.


session_start

Sintaxis

```
booleano session_start()
```

La función `session_start` preguntará al entorno para detectar si el usuario actual ya ha iniciado una sesión. En caso afirmativo, las variables almacenadas en la sesión se restituyen. De lo contrario, se abre una nueva sesión con la asignación de un identificador.

La función `session_start` devuelve `TRUE` si se ha podido crear la sesión con éxito y `FALSE` en caso contrario.

 Antes de la versión 5.3, esta función siempre devolvía `TRUE`.

Cualquier script que participe en la gestión de sesiones debe llamar a `session_start` para acceder a las variables de sesión.

Si todavía no se ha iniciado la sesión, la función `session_start` tratará de depositar una cookie que contiene el identificador de sesión en el ordenador del usuario: por lo tanto, es importante, como para las funciones `header` y `setcookie` que el principio la página todavía no se haya enviado al navegador. En caso de problema, se muestra un mensaje del siguiente tipo:

```
Warning: session_start(): Cannot send session cookie -  
headers already sent by in /app/scripts/index.php on line 3  
Warning: session_start(): Cannot send session cache limiter -  
headers already sent in /app/scripts/index.php on line 3
```

Ejemplo

```
<?php  
// Abrir/reactivar la sesión.  
session_start();  
?>
```

session_id

Sintaxis

```
cadena session_id([cadena nuevo_valor])
```

`nuevo_valor` Nuevo valor asignado al identificador de sesión.

Llamada sin parámetros, la función `session_id` devuelve el valor del identificador de sesión. Este valor siempre estará vacío si en el script no se llama a `session_start`.

Llamada con un parámetro, la función `session_id` modifica el valor asignado al identificador de sesión. En la práctica, en la mayoría de los casos, esto no es de mucho interés.

Ejemplo

```
<?php  
// Recuperar el valor de session_id antes.  
$antes = session_id();  
// Abrir/reactivar la sesión.  
session_start();  
// Recuperar el valor de session_id después.  
$después = session_id();  
// Visualización.  
$actual = date("H:i:s");  
echo "Hora: $actual<br />";  
echo "Antes: $antes<br />";  
echo "Después: $después<br />";  
?>
```

Resultado

- Primera llamada

Hora: 20:59:50

Antes:

Después: 20khscp2ae6b76cmsksof1tr37

- Segunda llamada sin salir del sitio (misma sesión)

Hora: 21:00:24

Antes:

Después: 20khscp2ae6b76cmsksof1tr37

- Tercera llamada después de salir del sitio (nueva sesión)

Hora: 21:01:39

Antes:

Después: 64pnn1n94k098e109iqoud61b0

Los tres ejemplos muestran que antes de llamar a la función `session_start`, la función `session_id` devuelve una cadena vacía, incluso si ya se ha iniciado la sesión (caso de la segunda llamada).

En la tercera llamada, el identificador de sesión es nuevo porque es una nueva sesión.

session_name

Sintaxis

```
cadena session_name([cadena nuevo_valor])
```

nuevo_valor Nuevo valor asignado al nombre de la variable que almacena el identificador de sesión.

Llamada sin parámetros, la función `session_name` devuelve el nombre de la variable en la que se almacena el identificador de sesión. La función `session_name` devuelve un resultado, incluso si en el script no se ha llamado a la función `session_start`.

Llamada con un parámetro, la función `session_name` cambia el nombre de la variable. En este caso, es necesario llamar a `session_name()` antes que a `session_start()`. En la práctica, en la mayoría de los casos, esto no presenta un gran interés (se restablece a su valor por defecto al final del script).

Ejemplo

```
<?php
// Recuperar el valor de session_name antes.
$antes = session_name();
// Abrir/reactivar la sesión.
session_start();
// Recuperar el valor de session_name después.
$después = session_name();
// Visualización.
echo "Antes: $antes<br />";
echo "Después: $después<br />";
?>
```

Resultado

Antes: PHPSESSID

Después: PHPSESSID

El nombre de la variable se define por la directiva de configuración `session.name`. Puede modificar esta directiva para utilizar otro nombre.

Manipular las variables registradas en la sesión

Después de llamar a la función `session_start`, los datos de sesión se pueden manipular directamente en la matriz asociativa `$_SESSION`. Todas las entradas almacenadas en la matriz `$_SESSION` se guardan automáticamente como datos de sesión.

Para guardar un nuevo dato en la sesión, basta con almacenar estos datos en la matriz `$_SESSION`, con la clave de su elección.

Para leer o editar un datos de sesión guardado previamente, basta con acceder a la matriz `$_SESSION` utilizando la clave correcta.



Recuerde que debe llamar a la función `session_start` para poder manipular los datos de sesión utilizando la matriz `$_SESSION`.

Ejemplo

- Script `pagina1.php` que abre una sesión y guarda los datos en la sesión.


```
<?php
// Abrir/reactivar la sesión.
session_start();
// Guardar dos informaciones en la sesión.
$_SESSION['nombre'] = 'Olivier';
$_SESSION['información'] = // es una matriz ...
    array('nombre'=>'Olivier','apellido'=>'HEURTEL');
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
    <head><title>Página 1</title></head>
    <body>
        <div><a href="pagina2.php">Página 2</a></div>
    </body>
</html>
```

- Script `pagina2.php` que muestra el valor de las variables de sesión:

```
<?php
// Llamada a session_start.
session_start();
// Visualización.
echo '$_SESSION[\'nombre\'] = ',
    isset($_SESSION['nombre'])?$_SESSION['nombre']:'',
    '<br />';
echo '$_SESSION[\'información\'][\'apellido\'] = ',
    isset($_SESSION['información']['apellido'])?
        $_SESSION['información']['apellido']:'',
    '<br />';
?>
```

Resultado en la página 2 después de mostrar la página 1 y hacer clic en el enlace

```
$_SESSION['nombre'] = Olivier
$_SESSION['información']['apellido'] = HEURTEL
```


 No hay codificación sobre los datos almacenados en la sesión.

La matriz `$_SESSION` es una variable "superglobal": está disponible en todo el script, incluso dentro de las funciones, sin necesidad de declararla global (`global $_SESSION` no es necesario).

Como se muestra en el ejemplo anterior, la función `isset` se puede utilizar para comprobar si los datos se almacenan en la sesión.

Si desea eliminar datos de sesión, puede utilizar la función `unset` para eliminar la entrada en la matriz `$_SESSION`: `unset($_SESSION['nombre'])`.

Para eliminar de un solo golpe todos los datos de sesión, puede asignar una matriz vacía (`array()`) a la matriz `$_SESSION`.


 En ambos casos, no elimine la totalidad de la variable `$_SESSION` (`unset($_SESSION)`).

En las versiones anteriores, se podían utilizar tres funciones para manipular los datos registrados en la sesión: `session_register`, `session_is_registered` y `session_unregister`. Estas funciones ya quedaron obsoletas en la versión 5.3 y se han eliminados definitivamente en la versión 5.4. Para manipular los datos de sesión, es necesario utilizar la matriz `$_SESSION`.

session_unset

Sintaxis

```
session_unset()
```

 Esta función se conserva por razones de compatibilidad con versiones anteriores. En su lugar, es aconsejable utilizar la función `unset` para eliminar los valores de la matriz `$_SESSION`, o asignarle una matriz vacía (`array()`).

La función `session_unset` elimina el registro de todas las variables de la sesión.

La función `session_unset` no tiene ningún efecto si se llama antes de la función `session_start`.

session_destroy

Sintaxis

```
booleano session_destroy()
```

Después de una llamada a la función `session_destroy`, la sesión ya no existe; una llamada a la función `session_id` devuelve una cadena vacía y una llamada posterior a la función `session_start` abrirá una nueva sesión.

La función `session_destroy` devuelve `TRUE` en caso de éxito y `FALSE` en caso de error.

La función `session_destroy` devuelve `FALSE` y muestra una alerta, si se llama antes de la

función `session_start`.

La función `session_destroy` no elimina los datos de sesión en el script actual. Para eliminar inmediatamente todos los datos de sesión, puede asignar una matriz vacía a `$_SESSION`.

Del mismo modo, esta función no destruye la cookie de sesión posiblemente utilizada para propagar el identificador de sesión. Para eliminar la cookie de sesión, puede utilizar la función `setcookie`.

Ejemplo

- Script `pagina1.php`

```
<?php
// Abrir/reactivar la sesión.
session_start();
// Guardar una información en la sesión.
$_SESSION['nombre'] = 'Olivier';
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head><title>Página 1</title></head>
<body>
<div>
<b>Página 1</b><br />
<?php
echo 'Hola ', $_SESSION['nombre'], '<br />';
echo 'session_id() = ', session_id(), '<br />';
?>
<a href="pagina2.php">Página 2</a><br />
</div>
</body>
</html>
```

- Script `pagina2.php`

```
<?php
// Abrir/reactivar la sesión.
session_start();
// Destruir la sesión.
session_destroy();
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head><title>Página 2</title></head>
<body>
<div>
<b>Página 2</b><br />
<?php
echo 'Hola ', $_SESSION['nombre'], '<br />';
echo 'session_id() = ', session_id(), '<br />';
?>
<a href="pagina3.php">Página 3</a><br />
</div>
</body>
</html>
```

- Script `pagina3.php`


```

<?php
// Abrir/reactivar la sesión.
session_start();
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head><title>Página 3</title></head>
<body>
<div>
<b>Página 3</b><br />
<?php
echo 'Hola ', $_SESSION['nombre'], '<br />';
echo 'session_id() = ', session_id(), '<br />';
?>
</div>
</body>
</html>

```

Resultado

- Página 1

Página 1

Hola Olivier
 session_id() = 64pnn1n94k098e109iqoud61b0
Página 2

- Página 2

Página 2

Hola Olivier
 session_id() =
Página 3

- Página 3

Página 3

Hola
 session_id() = 64pnn1n94k098e109iqoud61b0

En la segunda página, el dato de sesión todavía no se ha eliminado. Por contra, en la tercera página, el dato está vacío. En realidad, se trata de una nueva sesión, pero se ha reutilizado el mismo identificador ya que la cookie de sesión no se había eliminado.

Es posible modificar el script `pagina2.php` para que destruya completamente la sesión.

Ejemplo

```

<?php
// Abrir/reactivar la sesión.
session_start(); // Eliminar toda la información de sesión. $_SESSION = array();
// Eliminar la cookie de la sesión (si se utiliza).
// La cookie lleva el nombre de la variable que almacena
// el identificador de sesión.
if (isset($_COOKIE[session_name()])) {
    setcookie(session_name(), '', time()-1, '/');
}
// Destruir la sesión.

```

```

session_destroy();
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head><title>Página 2</title></head>
<body>
<div>
<b>Página 2</b><br />
<?php
echo 'Hola ', $_SESSION['nombre'], '<br />';
echo 'session_id() = ', session_id(), '<br />';
?>
<a href="pagina3.php">Página 3</a><br />
</div>
</body>
</html>

```

Resultado

- Página 1

Página 1

Hola Olivier
 session_id() = 89mt1vv683ifg5htc06g635ug1
[Página 2](#)

- Página 2

Página 2

Hola
 session_id() =
[Página 3](#)

- Página 3

Página 3

Hola
 session_id() = m7gttbpcbfbc0emijglf7jhn66

El dato de sesión se elimina correctamente desde la segunda página, y se utiliza un nuevo identificador de sesión en la tercera página.

session_status

Sintaxis

```
entero session_status()
```

La función `session_status` devuelve una constante que da el resultado actual de la sesión:

PHP_SESSION_DISABLED	Las sesiones se desactivan.
PHP_SESSION_NONE	Las sesiones se activan, pero no se inicia ninguna sesión.
PHP_SESSION_ACTIVE	Las sesiones se activan y se inicia una sesión.

Ejemplo

```
<?php
// Función que muestra el nombre de la constante
// de estado a partir de su valor.
function texto_estado($valor) {
    switch ($valor) {
        case PHP_SESSION_DISABLED:
            return 'PHP_SESSION_DISABLED';
        case PHP_SESSION_NONE:
            return 'PHP_SESSION_NONE';
        case PHP_SESSION_ACTIVE:
            return 'PHP_SESSION_ACTIVE';
    }
    return '?';
}
echo 'Antes de session_start(): ',
    texto_estado(session_status()), '<br />';
session_start();
echo 'Después de session_start(): ',
    texto_estado(session_status()), '<br />';
session_destroy();
echo 'Después de session_destroy(): ',
    texto_estado(session_status()), '<br />';
?>
```

Resultado

```
Antes de session_start(): PHP_SESSION_NONE
Después de session_start(): PHP_SESSION_ACTIVE
Después de session_destroy(): PHP_SESSION_NONE
```

3. Autogestión de la transmisión del identificador de sesión

Normalmente, el identificador de sesión se transmite automáticamente por PHP, ya sea por cookies o por URL (si el usuario no acepta las cookies).

Sin embargo, tres directivas de configuración controlan este comportamiento:

- `session.use_cookies`
- `session.use_trans_sid`
- `session.use_only_cookies`

Si la directiva `session.use_cookies` es igual a 0, PHP ni siquiera intenta utilizar cookies para transmitir el identificador de sesión. Por contra, si la directiva es igual a 1 (valor predeterminado), PHP intenta utilizar cookies.

Si la directiva `session.use_trans_sid` es igual a 0 (valor predeterminado), PHP no utiliza la URL para transmitir el identificador de sesión. Por contra, si la directiva es igual a 1 y el identificador de sesión no puede transmitirse a través de una cookie (debido a la configuración o la negativa del usuario), entonces PHP utiliza la URL para transmitir el identificador de sesión.

Si la directiva `session.use_only_cookies` está en 1, sólo se utilizan cookies para transmitir el identificador de sesión. De forma predeterminada, la directiva está en 1 desde la versión 5.3.

La principal consecuencia es la siguiente: si la directiva `session.use_trans_sid` es igual a 0 (valor predeterminado) y si el identificador de sesión no se puede transmitir a través de una cookie (debido a la configuración o la negativa del usuario), la gestión ya no funciona.

En términos de seguridad, no es aconsejable permitir la transmisión del identificador de sesión en la URL. Es preferible el uso de cookies, pero esto requiere que el usuario las acepte. Los valores predeterminados de las directivas de configuración van en esta dirección.

Ejemplo

- Script `pagina1.php`

```
<?php
// Abrir/reactivar la sesión.
session_start();
// Recuperar el identificador de sesión.
$sesión = session_id();
// Registrar una información en la sesión.
$_SESSION['nombre'] = 'Olivier';
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head><title>Página 1</title></head>
<body>
<div>
<b>Página 1</b><br />
<?php
// Mostrar el ID de la sesión.
echo 'session_id() = ',session_id(), '<br />';
// Mostrar el dato de sesión.
echo 'nombre = ',
    isset($_SESSION['nombre'])?$_SESSION['nombre']:'', '<br />';
?>
<a href="pagina2.php">Página 2</a><br />
</div>
</body>
</html>
```

- Script `pagina2.php`

```
<?php
// Abrir/reactivar la sesión.
session_start();?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head><title>Página 2</title></head>
<body>
<div>
<b>Página 2</b><br />
<?php
// Mostrar el ID de la sesión.
echo 'session_id() = ',session_id(), '<br />';
// Mostrar el dato de sesión.
echo 'nombre = ',
    isset($_SESSION['nombre'])?$_SESSION['nombre']:'', '<br />';
?>
</div>
</body>
</html>
```

Resultado (primer caso)

- Presentación inicial de la página 1:

Página 1

```
session_id() = g3jfgkkcjf8nvg3u3utja51n86
nombre = Olivier
Página 2
```

- Presentación de la página 2 después de hacer clic en el enlace Página 2 (si el usuario rechaza las cookies y si **session.use_trans_sid = 0**):

Página 2

```
session_id() = ghu6ctr1kbss9glcc71l2kdgt7
nombre =
```

La información de sesión no se ha transmitido y la llamada a la función `session_start` ha abierto una nueva sesión.

Resultado (segundo caso)

- Presentación inicial de la página 1:

Página 1

```
session_id() = ukb4oquhoc19mq381hjdvd88m14
nombre = Olivier
Página 2
```

- Presentación de la página 1 después de hacer clic en el enlace Página 2 (si el usuario rechaza las cookies, pero **session.use_trans_sid = 1** y **session.use_only_cookies = 0**):

Página 2

```
session_id() = ukb4oquhoc19mq381hjdvd88m14
nombre = Olivier
```

En este caso, la sesión se conserva correctamente.

Observemos la fuente HTML de la primera página para comprender lo que ha sucedido:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head><title>Página 1</title></head>
<body>
<div>
<b>Página 1</b><br />
session_id() = 0283b72fb608b8255874c4e9144f8d47<br />nombre
= Olivier<br />
<a href="pagina2.php?PHPSESSID=ukb4oquhoc19mq381hjdvd88m14">Página
2</a><br />
</div>
</body>
</html>
```

La etiqueta `<a href>` ha sido automáticamente reescrita por PHP para integrar la transmisión del identificador de sesión (llamada `PHPSESSID` por defecto). Si hubiera habido un parámetro en la URL, PHP habría añadido el parámetro `PHPSESSID` en la URL (`&PHPSESSID=...`).

Por contra, se plantea un problema en caso de regreso a la página 1 mediante el botón **Atrás** de su

navegador:

Página 1

```
session_id() = mi0c105o7uv4mdsslqf5vpj142
nombre = Olivier
Página 2
```

Como el identificador de sesión no se transmite cuando se llama a esta página, se abre una nueva sesión.

Del mismo modo, si una página incluye un formulario, PHP agrega automáticamente un campo oculto para transmitir el identificador de sesión al enviar el formulario.

Ejemplo

```
<input type="hidden" name="PHPSESSID"
value="mi0c105o7uv4mdsslqf5vpj142" />
```

Por contra, la información no se transmite al redirigir con la función `header`.

Si el identificador no se transmite automáticamente por PHP, conviene garantizar esta transmisión, lo más simple es efectuarla a través de la URL (como PHP hace cuando la opción está habilitada).

El nombre del identificador de sesión se define por una directiva de configuración y el uso en "bruto" del valor `PHPSESSID` puede conducir a código que no es portátil.

Una primera posibilidad consiste en utilizar las funciones `session_name` y `session_id`, para construir el parámetro que se incluirá en la URL en el formato `nombre_identificador=valor_identificador`.

La segunda posibilidad (más sencilla) consiste en utilizar una constante llamada `SID`, que PHP inicializa automáticamente, con una cadena `nombre_identificador=valor_identificador`.

Ejemplo

```
<?php
//Abrir/reactivar la sesión.
session_start();
//Construir la propia cadena.
echo 'Mi SID: ',session_name(),'=',session_id(),'<br />';
//Mostrar la constante SID.
echo 'SID: ',SID;
?>
```

Resultado

```
Mi SID: PHPSESSID=t4e1re53t1rv1d2a1ki4bgr6a6
SID: PHPSESSID=t4e1re53t1rv1d2a1ki4bgr6a6
```

Si se cambia el nombre de la variable en el archivo `php.ini`, la constante `SID` lo tiene en cuenta.

Esta constante, tal como se define, se puede utilizar directamente para la construcción de una URL, transmitiendo el valor del identificador de sesión.

Ejemplos

- En una etiqueta ``:

```
<a href="pagina2.php?<?php echo SID; ?>">Página 2</a>
```

- En el atributo `action` de la etiqueta `<form>`:

```
<form action="pagina2.php?<?php echo SID; ?>" method="post">
```

- En una redirección con la función `header`:

```
header('Location: pagina2.php?'.SID);
```

Por contra, en un campo oculto de un formulario, es necesario utilizar las funciones `session_name` y `session_id`.

Ejemplo

```
<input type = "hidden"
name="<?php echo session_name(); ?>"
value="<?php echo session_id(); ?>" />
```

➤ La constante `SID` no se rellena si no se acepta la cookie, excepto en el script que acaba de iniciar la sesión (ya que PHP aún no sabe si el cliente acepta o no la cookie). Integrar un `SID` vacío en una URL no es muy elegante, pero no plantea ningún problema (`` es válido). Si integra sistemáticamente el `SID` en la URL y la transmisión automática funciona (ya sea por cookies o por URL), no es muy elegante, pero no plantea ningún problema.

Por tanto, para escribir código portátil y elegante, conviene incorporar el `SID` en la URL cuando sea necesario.

Una función genérica puede perfectamente asumir esta tarea, probando la directiva de configuración y la constante `SID`.

Ejemplo

```
<?php
function url($url) {
//Si la directiva de configuración session.use_trans_sid
//está en 0 (no se realiza ninguna transmisión automática por URL) y
//si SID no está vacío (el equipo ha rechazado la cookie), entonces
//debemos gestionar la transmisión.
if((get_cfg_var('session.use_trans_sid') == 0)
and (SID != '')) {
//Agregar la constante SID detrás de la URL con un ?
//si todavía no hay ningún parámetro, o con un & en
//caso contrario.
$url .= ((strpos($url, '?') === FALSE)?'?' : '&').SID;
}
return $url;
}
//Abrir/reactivar la sesión.
session_start();
//Algunas pruebas.
echo url('pagina2.php'), '<br />';
echo url('pagina3.php?nombre=Olivier'), '<br />';
?>
```

Resultado

- Si `session.trans_id = 0` y la cookie se rechaza:

```
pagina2.php?PHPSESSID=rnug67kreboecedi8r9ohfd414
pagina3.php?nombre=Olivier&PHPSESSID=rnug67kreboecedi8r9ohfd414
```

- Si `session.trans_id = 0` y la cookie se acepta (pero 1a llamada):

```
pagina2.php?PHPSESSID=7hbp60dtuule20d1lda1j45235
pagina3.php?nombre=Olivier&PHPSESSID=7hbp60dtuule20d1lda1j45235
```

- Si `session.trans_id = 0` y la cookie se acepta (pero 2a llamada):

```
pagina2.php
pagina3.php?nombre=Olivier
```

- Si `session.trans_id = 1` y la cookie se rechaza o acepta:

```
pagina2.php
pagina3.php?nombre=Olivier
```

En el último caso, cuando la cookie se rechaza, PHP agrega automáticamente el SID en la URL (ya que la directiva `session.trans_id` está en on).

Esta función puede llamarse allí donde sea necesario construir una URL susceptible de transmitir el identificador de sesión.



Sea cual sea la configuración, esta función se debe utilizar cuando se realice una redirección con la función `header` (ya que PHP no reescribe nunca las URL llamadas por esta función).

4. Algunas directivas de configuración adicionales

Además de las ya mencionadas, es bueno conocer las directivas de configuración siguientes:

<code>session.save_path</code>	Directorio donde se almacenan los archivos temporales que contienen la información de la sesión.
<code>session.auto_start</code>	Si se establece en 1, la función <code>session_start</code> se llama automáticamente al principio de cada script (para un código portátil, es preferible llamar explícitamente a la función <code>session_start</code>).
<code>session.cookie_lifetime</code>	Tiempo de vida de las cookies depositadas en el equipo del usuario. El valor 0 propuesto por defecto se adapta perfectamente para una cookie de sesión.
<code>session.cookie_path</code>	Ruta de acceso en el servidor donde está disponible la cookie de sesión (ver la función <code>setcookie</code>). / por defecto.
<code>session.cookie_domain</code>	Dominio al que se reenvía la cookie (ver la función <code>setcookie</code>). Vacío por defecto.
<code>session.cookie_secure</code>	Indica si la cookie se debe transmitir sólo a través de una conexión segura (ver la función <code>setcookie</code>). Off por defecto.

`session.cache_limiter`

Determina el comportamiento en relación con el caché de todas las páginas que participan en la gestión de las sesiones enviadas al navegador. Valores posibles: `nocache`, `private`, `public`. Esta información se transmite en el encabezado de respuesta del servidor Web. `nocache` por defecto.

`session.cache_expire`

Tiempo de vida en minutos de las páginas en la caché. 180 por defecto. No tiene efecto si el valor `nocache` se especifica en `session.cache_limiter`.

5. Aplicación a la gestión de sesiones

Los principios establecidos con las otras técnicas se aplicarán de manera similar cuando se utiliza la funcionalidad de gestión de sesiones de PHP.

El principio fundamental es llamar sistemáticamente a la función `session_start` al principio de cada script, ya sea para abrir una sesión en una primera visita o para reactivar un contexto de sesión en la continuación de la navegación.

Ejemplo (con dos páginas)

- Script `pagina1.php`:

```
<?php
//Inclusión del archivo que contiene las
//funciones generales.
include('funciones.inc');
//Abrir/reactivar la sesión.
session_start();
//Comprobar si la sesión es nueva (es decir, si está abierta por
//la llamada a session_start() a continuación) o es antigua (es decir,
//si fue abierta
//por una llamada anterior a session_start()).
//Lo mejor es comprobar si una de nuestras variables de sesión
//ya está registrada.
if(! isset($_SESSION['fecha'])) {
    //Variable "fecha" todavía no registrada.
    //=> nueva sesión.
    //=> iniciar la sesión a nivel de aplicación.
    //Para este ejemplo:
    // - determinar la fecha/hora de apertura de la sesión;
    $fecha = date('\e\l d/m/Y a las H:i:s');
    // - guardar la fecha/hora de inicio de la sesión;
    $_SESSION['fecha'] = $fecha;
    // - recuperar el identificador de la sesión (para info);
    $sesión = session_id();
    // - preparar un mensaje.
    $mensaje = "Nueva sesión: $sesión - $fecha";
} else {
    //Variable "fecha" ya registrada.
    //=> antigua sesión.
    //=> recuperar las variables de sesión utilizadas
    //en este script.
    //Para este ejemplo:
    // - fecha/hora de apertura de la sesión;
    $fecha = $_SESSION['fecha'];
```

```
// - recuperar el identificador de la sesión (para info);
$sesión = session_id();
// - preparar un mensaje.
$mensaje = "Sesión ya iniciada: $sesión - $fecha";
}
//Determinación de la fecha y de la hora actual (no la
//del inicio de sesión).
$actual = 'Hoy es el día '.date('d/m/Y').
        ' son las '.date('H:i:s');
?>
<!DOCTYPE html PUBL
```

Conservar la información de una visita a otra

Si desea conservar la información sobre un usuario de una visita a otra (posiblemente muy distantes en el tiempo), hay dos soluciones dominantes:

- Depositar una cookie en su equipo (si es posible con su previo consentimiento).
- Almacenar la información del lado del servidor (la opción más práctica es utilizar una base de datos) y asociar esa información con una identificación (por lo general un nombre y una contraseña) que el usuario debe introducir en cada visita.

Algunos sitios utilizan una solución intermedia, elegante y fácil de usar; esta solución consiste en enofrecer al usuario depositar en su ordenador una cookie que contiene sólo uno o dos datos que permiten el acceso automático al sitio (sin introducir un nombre y una contraseña), la información adicional se recupera en una base de datos.

Vamos a ilustrar esta solución con la ayuda de dos páginas:

- Una página de personalización (script `personalizar.php`) que permite al usuario activar o desactivar la conexión automática.
- Una página de identificación (script `login.php`) que, según el caso, efectúa la conexión de forma automática o pide al usuario que se conecte.

Cada conexión del usuario es una sesión.

Fuente

- Script `personalizar.php`:

```
<?php
// Inclusión del archivo que contiene las funciones generales.
include('funciones.inc');
// Abrir/reactivar la sesión.
session_start();
// Inicialización de las variables.
$mensaje = '';
// ¿La sesión se ha iniciado al nivel de la aplicación?
if (isset($_SESSION['identificador'])) { // sí
    // Recuperar la información de sesión.
    $identificador = $_SESSION['identificador'];
    $contraseña = $_SESSION['contraseña'];
    // ¿Se llama al script en el procesamiento del formulario?
    if (isset($_POST['activar'])) { // sí
        // Activar la conexión automática.
        // Depositar dos cookies de un tiempo de vida de 30 días,
        // una para el identificador del usuario y una para su
        // contraseña.
        $vencimiento = time() + (30 * 24 * 3600);
        setcookie('identificador', $identificador, $vencimiento);
        setcookie('contraseña', $contraseña, $vencimiento);
        // Preparar un mensaje.
        $mensaje = 'Conexión automática activada';
    } elseif (isset($_POST['desactivar'])) { // sí
        // Desactivar la conexión automática.
        // Eliminar las dos cookies.
        setcookie('identificador');
        setcookie('contraseña');
        // Preparar un mensaje.
        $mensaje = 'Conexión automática desactivada';
    }
}
```

```

    }
} else { // Sesión no abierta a nivel de aplicación.
    // Redirigir al usuario a una página de inicio de sesión.
    header('Location: login.php');
    exit;
}
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
    <head><title>MiSitio.com - Personalizar</title></head>
    <body>
        <form action="personalizar.php" method="post">
            <div>
                <input type="submit" name="activar"
                    value="Activar la conexión automática" /><br />
                <input type="submit" name="desactivar"
                    value="Desactivar la conexión automática" /><br />
                <?php echo $mensaje; ?><br />
            </div>
        </form>
    </body>
</html>

```

- Script login.php (variante simple de la versión anterior):

```

<?php
// Inclusión del archivo que contiene las funciones generales.
include('funciones.inc');
// Función que verifica que las credenciales de identificación
// introducidas son correctas.
function usuario_existe($identificador,$contraseña) {
    // Conexión y selección de la base de datos.
    $conexión = mysqli_connect();
    mysqli_select_db($conexión,'diane');
    // Definición y ejecución de una consulta preparada.
    $sql = 'SELECT 1 FROM usuarios ';
    $sql .= 'WHERE identificador = ? AND contraseña = ?';
    $consulta = mysqli_stmt_init($conexión);
    $ok = mysqli_stmt_prepare($consulta,$sql);
    $ok = mysqli_stmt_bind_param
        ($consulta,'ss',$identificador,$contraseña);
    $ok = mysqli_stmt_execute($consulta);
    mysqli_stmt_bind_result($consulta,$existe);
    $ok = mysqli_stmt_fetch($consulta);
    mysqli_stmt_free_result($consulta);
    // La identificación tiene éxito si la consulta devuelve
    // una línea (el usuario existe y la contraseña
    // es correcta).
    // Si este es el caso $existe contiene 1, de lo contrario está
    // vacía. Basta con devolverla como un valor booleano.
    return (bool) $existe;
}
// Inicialización de las variables.
$identificador = '';
$contraseña = '';
$mensaje = '';
$acción = '';
// ¿Se llama al script en la validación del formulario?
if (isset($_POST['conexión'])) { // sí
    // => conexión manual.

```

```

// Recuperar la información introducida.
$identificador = ($_POST['identificador']);
$contraseña = ($_POST['contraseña']);
// Indicar la acción a realizar a continuación.
$acción = 'conexión';
// Preparar un mensaje en caso de problema.
$mensaje = 'Identificación incorrecta. '.
    "Vuelva a intentarlo".
// De lo contrario, ¿hay una cookie de "identificador"?
} elseif (isset($_COOKIE['identificador'])) { // sí
    // => conexión automática.
    // Recuperar la información de las cookies.
    $identificador = $_COOKIE['identificador'];
    $contraseña = $_COOKIE['contraseña'];
    // Indicar la acción a realizar a continuación.
    $acción = 'conexión';
    // Preparar un mensaje en caso de problema.
    $mensaje = 'Identificación automática incorrecta. '.
        'Inténtelo de forma manual.';
}
// Finalmente, ¿qué hacemos?
if ($acción == 'conexión') { // intentar una conexión
    // Verificar que el usuario existe.
    if (usuario_existe($identificador,$contraseña)) {
        // El usuario existe ...
        // => iniciar la sesión a nivel de aplicación.
        session_start();
        session_regenerate_id(); // en el caso en que ...
        $_SESSION['identificador'] = $identificador;
        $_SESSION['contraseña'] = $contraseña;
        // Redirigir al usuario a otra página del sitio
        // (¡sólo hay una!).
        header('location: '.url('personalizar.php'));
        exit;
    } // usuario_existe
} // $acción == 'conexión'
// Si es la primera llamada, o si la conexión manual
// o automática ha fallado, dejar que se muestre el formulario.
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head><title>MiSitio.com - Identificación</title></head>
<body>
    <form action="login.php" method="post">
    <table border="0">
    <tr>
        <td align="right">Identificador:</td>
        <td><input type="text" Name="identificador" value=
            "<?php echo hacia_formulario($identificador); ?>" /></td>
    </tr>
    <tr>
        <td align="right">Contraseña:</td>
        <td><input type="password" Name="contraseña" value=
            "<?php echo hacia_formulario($contraseña); ?>" /></td>
    </tr>
    <tr>
        <td></td>
        <td align="right"><input type="submit" name="conexión"
            value="Conexión" /></td>
    </tr>

```

```
</table>
<?php echo $mensaje; ?>
</form>
</body>
</html>
```

Resultado

- Primera llamada:

Formulario de conexión manual con campos para 'Identificador' y 'Contraseña', y un botón 'Conexión'.

- Error en la conexión manual:

Formulario de conexión manual con el identificador 'heurtel' y la contraseña oculta por puntos. El botón 'Conexión' está deshabilitado. Se muestra el mensaje: 'Identificación incorrecta. Vuelva a intentarlo.'

- Después de una conexión exitosa:

Botones para 'Activar la conexión automática' y 'Desactivar la conexión automática'.

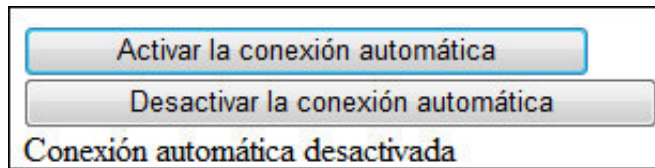
- Haga clic en el botón **Activar**:

Formulario después de activar la conexión automática. El botón 'Activar la conexión automática' está deshabilitado, el botón 'Desactivar la conexión automática' está habilitado, y se muestra el mensaje: 'Conexión automática activada'.

- Salir y volver a una de las dos páginas: llegada directa a la página de personalización (se ha realizado una conexión automática):

Botones para 'Activar la conexión automática' y 'Desactivar la conexión automática'.

- Haga clic en el botón **Desactivar**:

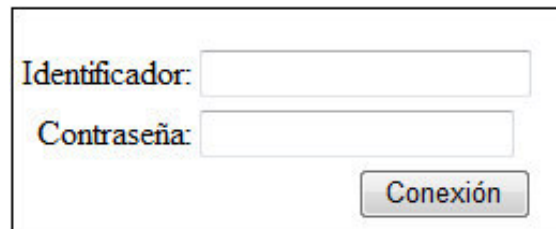


Activar la conexión automática

Desactivar la conexión automática

Conexión automática desactivada

- Salir y volver a una de las dos páginas: se muestra la página de identificación (más conexión automática):

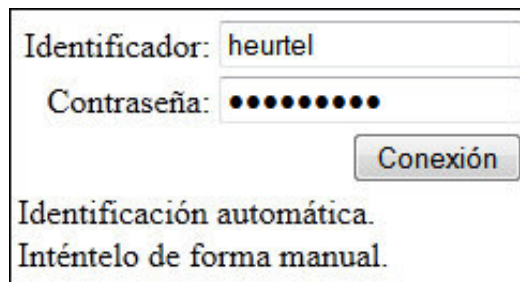


Identificador:

Contraseña:

Conexión

- Repita la secuencia de activación y luego vuelva a una de las dos páginas, después de haber eliminado la cuenta de usuario (la conexión automática falla):



Identificador: heurtel

Contraseña: ●●●●●●●●

Conexión

Identificación automática.
Inténtelo de forma manual.

- Es preferible almacenar la contraseña del usuario en forma cifrada o utilizar un identificador de conexión automático asociado al usuario. De esta manera, en ambos casos, la contraseña del usuario no estará expuesta a miradas indiscretas.

Breve resumen de las variables Get/Post/Cookie/Session

Desde el comienzo de este libro, nos hemos encontrado con variables "especiales", relacionadas con datos del formulario, con datos transmitidos a través de una URL, con datos de una cookie o con datos de sesiones.

Hemos visto que estas variables funcionan bajo el mismo principio: son accesibles por medio de matrices asociativas `$_GET`, `$_POST`, `$_COOKIE` y `$_SESSION`. Además, la matriz asociativa `$_REQUEST` reagrupa el contenido de las matrices `$_GET`, `$_POST` y `$_COOKIE`.

La matriz `$_REQUEST` debe utilizarse con precaución, ya que contiene información proporcionada al script a través de varios mecanismos; no estamos necesariamente seguros de que la información leída llega por los medios esperados.

El hecho de que las matrices `$_GET`, `$_POST` y `$_COOKIE` se creen realmente depende de la directiva de configuración `variables_order`.

Esta directiva es una cadena compuesta por los caracteres G, P y C que corresponden a los tipos ya mencionados, y de otros dos caracteres, E, correspondiente a las variables de entorno, y S correspondiente a las variables del servidor HTTP. De forma predeterminada, la directiva `variables_order` es igual a EGPCS.

Las variables de entorno del sistema operativo y las variables del servidor HTTP están disponibles en el entorno PHP a través de las matrices asociativas `$_ENV` y `$_SERVER`.

Desde la versión 5.3, el orden en el que las variables *Get*, *Post* y *Cookie* se definen en la matriz `$_REQUEST` está condicionado por la directiva de configuración `request_order`. Esta directiva es una cadena compuesta por los caracteres G, P y C correspondientes a los tipos ya mencionados. Si esta directiva no se especifica, el contenido de la matriz `$_REQUEST` se ve afectada por la directiva `variables_order` (como en las versiones anteriores).

Ejemplo

- Script `pagina1.php`:

```
<?php
// Abrir una sesión y registrar una información de
// sesión llamada "x" de valor "SESSION".
session_start();
$_SESSION['x'] = 'SESSION';
// Depositar una cookie llamada "x" de valor "COOKIE".
setcookie('x','COOKIE');
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head><title>Página 1</title></head>
  <body>
    <!-- En un formulario, poner una variable dada
    llamada "x" de valor "GET" en la URL del
    atributo "action" -->
    <form action="pagina2.php?x=GET" method="post">
    <!-- Incluir también un campo llamado "x" de
    valor "POST" -->
    <input type="hidden" name="x" value="POST" />
    <!-- Más un botón para ir a la página 2 -->
    <input type="submit" name="ok" value="Página 2">
  </form>
</body>
```


</html>

- Script pagina2.php:

```
<?php
// Reactivar la sesión.
session_start();
// Mostrar los valores de 'x' a partir de las matrices.
echo '$_GET[\x] = ',
    isset($_GET['x'])?$_GET['x']:'',<br />;
echo '$_POST[\x] = ',
    isset($_POST['x'])?$_POST['x']:'',<br />;
echo '$_COOKIE[\x] = ',
    isset($_COOKIE['x'])?$_COOKIE['x']:'',<br />;
echo '$_SESSION[\x] = ',
    isset($_SESSION['x'])?$_SESSION['x']:'',<br />;
echo '$_REQUEST[\x] = ',
    isset($_REQUEST['x'])?$_REQUEST['x']:'',<br />;
?>
```

En el primer script, varios datos diferentes están asociados con el mismo identificador "x".

Resultado de la presentación de la página 1 después de hacer clic en el botón de Página 2

```
$_GET['x'] = GET
$_POST['x'] = POST
$_COOKIE['x'] = COOKIE
$_SESSION['x'] = SESSION
$_REQUEST['x'] = POST
```

En este ejemplo, la matriz `$_REQUEST` contiene el valor "POST", ya que la directiva `request_order` contiene el valor GP. La letra P aparece después de la letra G y la letra C se omite: las variables *Post*, por lo tanto, tienen prioridad sobre las variables *Get* y las variables *Cookie* no se insertan en la matriz.

Si la directiva `request_order` se modifica en CPG, obtenemos el siguiente resultado:

```
$_GET['x'] = GET
$_POST['x'] = POST
$_COOKIE['x'] = COOKIE
$_SESSION['x'] = SESSION
$_REQUEST['x'] = GET
```

La tabla `$_REQUEST` contiene de ahora en adelante el valor "GET".

Por otra parte, si la directiva `request_order` se modifica en CP, obtenemos el siguiente resultado:

```
$_GET['x'] =
$_POST['x'] = POST
$_COOKIE['x'] = COOKIE
$_SESSION['x'] = SESSION
$_REQUEST['x'] = POST
```

La matriz `$_GET` ya no se alimenta y la matriz `$_REQUEST` contiene de ahora en adelante el valor "POST".



Utilizar el mismo nombre para información distinta no es una buena práctica de programación; un desarrollador sensible no se encontrará nunca con este problema. Aparte de esto, debemos ser

conscientes de que un usuario (malicioso y/o malintencionado) puede proporcionar fácilmente un valor por un medio determinado (GPC, por ejemplo) a una variable cuyo origen piensa controlar. Escribir el código cuyo funcionamiento está vinculado a un valor determinado de la `directivavariables_order` tampoco es probablemente una buena idea desde el punto de vista de la portabilidad y la facilidad de mantenimiento. Lo que recomienda el equipo de desarrollo de PHP es lo siguiente: recupere los valores "EGPCS" para sus matrices asociativas respectivas para evitar problemas.

Índice

Utilización de SQLite
PHP Data Objects (PDO)
"Magic quotes": el regreso

Administrar las sesiones
Descripción del problema
Autenticación
Crear un identificador único
Pasar la información a través de la URL
Pasar información a través de un campo de formulario oculto
Utilizar cookies
Utilizar la gestión de sesiones de PHP
Conservar la información de una visita a otra
Breve resumen de las variables
Get/Post/Cookie/Session

Enviar un correo electrónico
Información general
Enviar un mensaje de texto sin archivos adjuntos
Enviar un mensaje en formato MIME

Gestión de archivos
Administrar los errores en un script PHP
Anexo

Información general

Un sitio interactivo a menudo necesita enviar correos electrónicos a los usuarios, por ejemplo, para confirmar una compra, un registro o enviar un boletín de noticias.

La función `mail`, propuesta por PHP, permite responder de manera simple a este tipo de necesidad. En esta sección se explica cómo utilizar esta función, en primer lugar, para enviar mensajes de texto (sin datos adjuntos) y, a continuación, para enviar mensajes en formato MIME (*Multipurpose Internet Mail Extensions*).

Además, PHP ofrece una biblioteca potente, pero más compleja de usar, para gestionar mensajes mediante el protocolo IMAP (*Internet Message Access Protocol*). Esta biblioteca no se tratará en este libro, ya que no es esencial para satisfacer las necesidades descritas anteriormente.

Subir

Condiciones generales de uso

Copyright - ©Editions ENI



Enviar un mensaje de texto sin archivos adjuntos

La función `mail` permite enviar un mensaje de correo electrónico.

Sintaxis

```
booleano mail(cadena destinatario, cadena objeto, cadena mensaje
[, cadena encabezado])
```

destinatario	Dirección de correo electrónico del destinatario. Es posible especificar varios destinatarios separados con comas.
objeto	Asunto del mensaje.
mensaje	Texto del mensaje.
encabezado	Encabezados adicionales.

La función `mail` envía el mensaje caracterizado por los diferentes parámetros a un servidor de correo definido por las directivas de configuración siguientes:

Win32	SMTP	Dirección del servidor SMTP (<i>Simple Mail Transfer Protocol</i>) al cual enviar el mensaje. Ejemplo: <code>smtp.wanadoo.es</code>
	<code>sendmail_from</code>	Dirección de correo electrónico del remitente. Ejemplo: <code>webmaster@misitio.com</code> Esta directiva debe estar presente, incluso aunque esté vacía.
Unix	<code>sendmail_path</code>	Ruta de acceso al ejecutable del servidor de correo (puede incluir parámetros). Ejemplo: <code>sendmail -t -i</code>

La función `mail` devuelve `TRUE` si el mensaje se ha podido enviar al servidor (lo cual no garantiza que éste último haya podido enviarlo con éxito) y `FALSE` en caso contrario. No hay manera de saber si el mensaje ha sido enviado con éxito; esta verificación debe llevarse a cabo fuera de PHP.

El cuarto parámetro permite especificar información adicional que se envía en el encabezado del mensaje; para incluir varios datos, deben ir separados por la secuencia `\r\n`.



La secuencia a utilizar como separador de encabezado es a menudo una fuente de problemas (y de debate en foros de discusión). Muy a menudo, sólo funciona la secuencia `\n`; a veces, la secuencia `\r\n` no funciona.

Ejemplo de mensaje simple

```
<?php
// Destinatario.
$destinatario = 'contact@olivier-heurtel.fr';
// Asunto.
$asunto = 'Registro a miSitio.com';
// Mensaje.
$mensaje =
'Señor Heurtel,
Gracias por su registro en nuestro sitio'
```

```

miSitio.com.
Esperamos que este sitio cumpla con sus expectativas.
El webmaster.';
// Envío.
$ok = mail($destinatario,$asunto,$mensaje);
?>

```

Este ejemplo muestra que el mensaje se puede escribir directamente en varias líneas; también es posible construirlo por concatenación.

Ejemplo de mensaje más complejo

```

<?php
// Dos destinatarios separados por una coma.
$destinatarios = 'olivier@diane.com,xavier@zeus.fr';
// Asunto.
$asunto = 'Registro en la maratón';
// Mensaje.
$mensaje .= "Olivier, Xavier,\n";
$mensaje .= "Confirmamos su registro al\n";
$mensaje .= "maratón del domingo 11 de abril.\n";
$mensaje .= "La salida tendrá lugar a las 8:45.\n\n";
$mensaje .= "La organización.\n";
// Encabezados adicionales.
$encabezados.= "From: \"Registro\" <contact@maraton.es>\n";
$encabezados.= "Reply-To: \"Registro\" <contact@maraton.es>\n";
$encabezados.= "X-Priority: 1\n";
// Envío.
$ok = mail($destinatarios,$asunto,$mensaje,$encabezados);
?>

```

Los encabezados adicionales se especifican con el formato palabra clave: valor.

Los encabezados más habituales son los siguientes:

From: Origen del mensaje con el formato ["nombre limpio"] <dirección de correo electrónico>.

Ejemplos:

```

"Olivier HEURTEL" <contact@olivier-heurtel.fr>
<contact@olivier-heurtel.fr>

```

To: Destinatario(s) (mismo formato que el encabezado From).

Reply-To: Dirección de respuesta (mismo formato que el encabezado From).

X-Priority: Prioridad del mensaje.
 1 = prioridad más alta
 2 = prioridad alta
 3 = prioridad normal
 4 = prioridad baja



En el parámetro destinatario, no es posible especificar una dirección con el formato ["nombre limpio"] <dirección de correo electrónico>. Por contra, es posible indicar las direcciones de esta forma si se envían en el encabezado To:, duplicando el parámetro destinatario.

Enviar un mensaje en formato MIME

1. Preámbulo

En esta sección vamos a discutir cómo enviar mensajes con formato MIME, o más generalmente en formato Multipart MIME.

El formato MIME permite enviar un mensaje con un formato diferente al de texto: HTML, imágenes...

El formato Multipart MIME permite enviar un mensaje compuesto de varias partes, cada una con un formato diferente (texto más una imagen, por ejemplo) y una de las "partes" puede ser un archivo adjunto.

El objetivo de este punto, sin entrar en detalles sobre el formato MIME (se puede encontrar más información en los numerosos RFC que tratan este tema), es mostrar concretamente cómo proceder en dos casos típicos, el envío de un mensaje en formato HTML y el envío de un mensaje con datos adjuntos.

2. Mensaje en formato HTML

El caso del envío de un mensaje en formato HTML permite ilustrar el uso del formato MIME simple.

Ejemplo (fuente de un mensaje MIME en formato HTML)

```
From: "Olivier" <olivier@diane.com>
To: "Xavier" <xavier@zeus.fr>
Subject: ¡Hola!
Date: Mon, 10 Sep 2001 09:24:13 -0100
Message-ID: <3b9c6a403d9f000b@hermes.diane.com>
MIME-Version: 1.0
Content-Type: text/html; charset=iso-8859-1
Content-Transfer-Encoding: 8bit
```

```
<html>
<head><title>¡Hola!</title></head>
<body>
<font color="green">¡Hola!</font>
</body>
</html>
```

Un mensaje MIME simple incluye los encabezados estándar de un mensaje, a continuación, tres líneas de encabezados adicionales (en negrita) que indican que el mensaje está en formato MIME y, finalmente, el cuerpo del mensaje propiamente dicho.

Las tres líneas de encabezados adicionales son las siguientes:

MIME-Version	Indica que el mensaje está en formato MIME y especifica la versión.
Content-Type	Indica el tipo MIME del contenido.
Content-Transfer-Encoding	Indica el tipo de cifrado.

Algunos tipos MIME habituales:

<code>text/plain</code>	Texto simple. El juego de caracteres utilizado se puede especificar con la opción <code>charset</code> (por ejemplo, <code>iso-8859-1</code>).
<code>text/html</code>	Documento en formato HTML. El juego de caracteres utilizado se puede especificar con la opción <code>charset</code> (por ejemplo, <code>iso-8859-1</code>).
<code>image/jpeg</code>	Imagen en formato JPEG.
<code>application/octet-stream</code>	Datos binarios.

Algunos tipos de cifrado habituales:

<code>7bit</code>	Cifrado del texto en 7 bits.
<code>8bit</code>	Cifrado del texto en 8 bits (se utiliza para mantener los acentos).
<code>base64</code>	Cifrado para los datos binarios (ver la función <code>base64_encode</code>).

Convencionalmente, para los tipos MIME `text/*`, se utiliza un tipo de cifrado `7bit` u `8bit` y el cuerpo del mensaje contiene el texto limpio. También es posible utilizar un cifrado `base64` y colocar en el cuerpo del mensaje datos cifrados en consecuencia (ver la función `base64_encode` más abajo).

Para los tipos MIME correspondientes a datos binarios (imagen, sonido, documento PDF...), se utiliza un cifrado `base64` y el cuerpo del mensaje contiene los datos cifrados en consecuencia (ver la función `base64_encode` más abajo).

Enviar un mensaje en formato HTML con la función `mail` es relativamente simple.

Es necesario:

- colocar las líneas correspondientes en el encabezado.
- colocar los datos HTML en el cuerpo del mensaje.

Ejemplo

```
<?php
// Destinatarios.
$destinatarios = 'xavier@zeus.fr';
// Asunto.
$asunto = '¡Hola!';
// Encabezados adicionales.
$encabezados = "From: \"Olivier\" <olivier@diane.com>\n";
$encabezados .= "MIME-Version: 1.0\n";
$encabezados .= "Content-Type: text/html; charset=iso-8859-1\n";
$encabezados .= "Content-Transfer-Encoding: 8bit\n";
// Mensaje (HTML).
$message .= "<html>\n";
$message .= "<head><title>¡Hola!</title></head>\n";
$message .= "<body>\n";
$message .= "<font color=\"green\">¡Hola!</font>\n";
$message .= "</body>\n";
$message .= "</html>\n";
// Envío.
$ok = mail($destinatarios,$asunto,$message,$encabezados);
?>
```

Si utiliza un cifrado `base64`, puede llamar a la función `base64_encode` para realizar el cifrado de

datos.

Sintaxis

```
cadena base64_encode(cadena datos)
```

datos Datos que se van a cifrar.

Esta función devuelve los datos cifrados.

Además, para cumplir con las especificaciones, debe cortar los datos cifrados en base64 en bloques de 76 bytes separados por una secuencia `\r\n`.

Esta operación puede realizarse de manera muy sencilla utilizando la función `chunk_split`.

Sintaxis

```
cadena chunk_split (cadena datos, entero longitud, cadena
separador)
```

datos Datos que se van a cortar.

longitud Longitud de los fragmentos (por defecto, 76).

separador Separador de bloques (`\r\n` por defecto).

La función `chunk_split` devuelve la cadena cortada.

Ejemplo de uso para el envío de un mensaje

```
<?php
// Destinatarios.
$destinatarios = 'xavier@zeus.fr';
// Asunto.
$asunto = '¡Hola!';
// Encabezados adicionales.
$encabezados = '';
$encabezados .= "From: \"Olivier\" <olivier@diane.com>\r\n";
$encabezados .= "MIME-Version: 1.0\r\n";
$encabezados .= "Content-Type: text/html; charset=iso-8859-1\r\n";
$encabezados .= "Content-Transfer-Encoding: base64\r\n";
// Mensaje (HTML).
$mensaje .= "<html>\n";
$mensaje .= "<head><title>¡Hola!</title></head>\n";
$mensaje .= "<body>\n";
$mensaje .= "<font color=\"green\">¡Hola!</font>\n";
$mensaje .= "</body>\n";
$mensaje .= "</html>\n";
// Cifrado de corte.
$mensaje = chunk_split(base64_encode($mensaje));
// Envío.
$ok = mail($destinatarios,$asunto,$mensaje,$encabezados);
?>
```

3. Mensaje con archivo adjunto

El caso del envío de un mensaje con archivo adjunto permite ilustrar el uso del formato Multipart MIME.

Ejemplo (fuente de un mensaje con un archivo adjunto en formato Multipart MIME)

```

From: "Olivier" <olivier@diane.com>
To: "Xavier" <xavier@zeus.fr>
Subject: ¡Hola!
Date: Mon, 10 Sep 2001 09:24:13 -0100
Message-ID: <3b9c6a403d9f000b@hermes.diane.com>
MIME-Version: 1.0
Content-Type: multipart/mixed; boundary="=O=L=I=V=I=E=R="

--=O=L=I=V=I=E=R=
Content-Type: text/plain; charset=iso-8859-1 Content-Transfer-Encoding: 8bit

Ver archivo adjunto.

--=O=L=I=V=I=E=R=
Content-Type: application/octet-stream; name="PJ.DOC"
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename="PJ.DOC"

0M8R4KGxGuEAAA...
AAAAAAAAAAAAAAAAAAAAA==

--=O=L=I=V=I=E=R=

```

Un mensaje Multipart MIME contiene los encabezados estándar de un mensaje, a continuación, dos líneas de encabezados adicionales (en negrita) que indican que el mensaje está en formato Multipart MIME y, finalmente, las diferentes partes del mensaje; cada parte tiene su propio encabezado que indica su formato (tipo MIME y cifrado).

En el encabezado del mensaje, encontramos las dos líneas `MIME-Version` y `Content-Type`. Para un mensaje Multipart MIME, `Content-Type` es igual a `multipart/mixed` seguido de una opción `boundary` que da la cadena utilizada para marcar el comienzo de las diferentes partes del mensaje (`=O=L=I=V=I=E=R=` en nuestro ejemplo).

Cada parte del mensaje comienza con dos signos "menos" seguidos de la cadena dada por `boundary` (`--=O=L=I=V=I=E=R=`, en nuestro ejemplo).

En el encabezado de cada parte, encontramos las dos líneas `Content-Type` y `Content-Transfer-Encoding` que especifican el tipo MIME y el cifrado de la parte.

Los datos de la parte vienen a continuación.

El final del mensaje está marcado por dos signos "menos", seguidos de la cadena dada por `boundary` seguida de dos signos "menos" (`--=O=L=I=V=I=E=R=`, en nuestro ejemplo).

La cadena dada por `boundary` debe elegirse con cuidado para evitar que se confunda con los datos, lo cual provocaría una interpretación incorrecta del mensaje.

En el encabezado de cada parte, es posible indicar una línea de encabezado adicional (`Content-Disposition`) que sugiere al cliente de correo una manera de presentar los datos correspondientes al usuario: los dos valores posibles son `inline` (los datos se presentan directamente en el mensaje) o `attachment` (los datos se presentan como un archivo adjunto); en este caso, se puede sugerir un nombre de archivo mediante la opción `filename`.

Ejemplo

```
Content-Disposition: attachment; filename="PJ.DOC"
```

Para enviar un mensaje Multipart MIME con la función `mail`, debe construir las diferentes partes del mensaje en el parámetro `message`, respetando la estructura indicada anteriormente; el parámetro encabezado de la función `mail` contiene los encabezados estándar, incluye el encabezado indicando que se trata de un mensaje en formato Multipart MIME.

- Las diferentes partes del mensaje debe estar separadas por una línea en blanco (`\r\n`). Dentro de cada parte, ocurre lo mismo entre el encabezado de la parte y los datos. Vea la nota en la sección Enviar un mensaje de texto sin datos adjuntos relativa a la secuencia `\r\n`.

Ejemplo de envío de un mensaje con archivo adjunto

```
<?php
// Destinatarios.
$destinatarios = 'xavier@zeus.fr';
// Asunto.
$asunto = '¡Hola!';
// Encabezados adicionales.
$encabezados = '';
// -> origen del mensaje
$encabezados .= "From: \"Olivier\" <olivier@diane.com>\r\n";
// -> mensaje en formato Multipart MIME
$encabezados .= "MIME-Version: 1.0\r\n";
$encabezados .= "Content-Type: multipart/mixed; ";
$encabezados .= "boundary=\"=O=L=I=V=I=E=R=\"\r\n";
// Mensaje.
$message = "";
// -> primera parte del mensaje (texto propiamente dicho)
// -> encabezado de la parte
$message .= "---=O=L=I=V=I=E=R=\r\n";
$message .= "Content-Type: text/plain; ";
$message .= "charset=iso-8859-1\r\n ";
$message .= "Content-Transfer-Encoding: 8bit\r\n";
$message .= "\r\n"; // Línea en blanco
// -> datos de la parte
$message .= "Ver el archivo adjunto.\r\n";
$message .= "\r\n"; // línea en blanco
// -> segunda parte del mensaje (archivo adjunto)
// -> encabezado de la parte
$message .= "---=O=L=I=V=I=E=R=\r\n";
$message .= "Content-Type: application/octet-stream; ";
$message .= "name=\"PJ.DOC\"\r\n";
$message .= "Content-Transfer-Encoding: base64\r\n";
$message .= "Content-Disposition: attachment; ";
$message .= "filename=\"pj.doc\"\r\n";
$message .= "\r\n"; // línea en blanco
// -> lectura del archivo correspondiente al archivo adjunto
// (asegurarse de que no hay cifrado magic quotes)
set_magic_quotes_runtime(0);
$datos = file_get_contents('pj.doc');
// -> cifrado y corte de los datos
$datos = chunk_split(base64_encode($datos));
// -> datos de la parte (integración en el mensaje)
$message .= "$datos\r\n";
$message .= "\r\n"; // línea en blanco
// Delimitador de fin del mensaje.
$message .= "---=O=L=I=V=I=E=R=--\r\n";
// Envío del mensaje.
$ok = mail($destinatarios,$asunto,$message,$encabezados);
```

?>

Se pueden enviar varios datos adjuntos mediante la repetición, tantas veces como sea necesario, de las líneas de código correspondientes.

- En teoría, todo el mensaje podría construirse en el encabezado (parámetro `mensaje` equivale a una cadena vacía). En la práctica y como regla general, esto funciona en la plataforma Linux, pero no en la plataforma Windows.

Manipular los archivos en el servidor

PHP ofrece un gran número de funciones que permiten manipular los archivos en el servidor.

Las funciones utilizadas con mayor frecuencia son las siguientes:

Nombre	Función
<code>fopen</code>	Abrir un archivo
<code>fclose</code>	Cerrar un archivo
<code>fread</code>	Leer el contenido de un archivo (en una cadena)
<code>file</code>	Leer el contenido de un archivo (en una matriz)
<code>fwrite</code>	Escribir en un archivo
<code>file_get_contents</code>	Abrir, leer y cerrar un archivo
<code>file_put_contents</code>	Abrir, escribir y cerrar en un archivo
<code>copy</code>	Copiar un archivo
<code>unlink</code>	Eliminar un archivo
<code>rename</code>	Cambiar el nombre de un archivo
<code>file_exists</code>	Probar la existencia de un archivo
<code>filesize</code>	Obtener el tamaño de un archivo
<code>chdir</code>	Cambiar el directorio actual
<code>opendir</code>	Abrir un directorio
<code>closedir</code>	Cerrar un directorio
<code>readdir</code>	Leer el contenido de un directorio
<code>Scandir</code>	Lista el contenido de un directorio (en una matriz)

Algunas de estas funciones tomarán como parámetro un nombre de archivo o de directorio. En una plataforma de Windows, para especificar una ruta de acceso en una cadena de caracteres delimitada por comillas, debe escapar la barra invertida (con una barra invertida = `\\`) o puede usar una notación tipo "Unix", con barras (`/`).

Por ejemplo, la ruta `c:\temp\info.txt` se puede escribir como `"c:\\temp\\info.txt"` o `"c:/temp/info.txt"`. Si no se ha especificado ninguna ruta, se utiliza el directorio actual. Se pueden especificar nombres relativos utilizando el carácter `.` (punto) para designar el directorio actual, y `..` (dos puntos) para designar el directorio superior.

La constante predefinida `DIRECTORY_SEPARATOR` da el carácter separador utilizado en los nombres de directorio para la plataforma en la que está instalado PHP. La constante predefinida `PHP_EOL` da la secuencia de caracteres utilizada por la plataforma para representar una nueva línea.

Además, varias funciones tienen un parámetro (llamado `utilizar_inclusión` en las sintaxis de este libro) que permite buscar el archivo en los directorios especificado por la directiva de configuración `include_path`.

fopen

La función `fopen` permite abrir un archivo.

Sintaxis

```
recurso fopen(cadena nombre_archivo, cadena mode
[, entero utilizar_inclusión])
```

nombre_archivo	Ruta de acceso del archivo a abrir.
modo	<p>Modo de apertura del archivo</p> <p>r: sólo lectura</p> <p>r+: lectura y escritura</p> <p>w: sólo escritura (vacía el archivo si existe, lo crea si no existe)</p> <p>w+: lectura y escritura (vacía el archivo si existe, lo crea si no existe)</p> <p>a: sólo escritura (además - crea el archivo si no existe)</p> <p>a+: lectura y escritura (además - crea el archivo si no existe)</p> <p>x: sólo escritura con la creación previa del archivo (genera un error si el archivo ya existe)</p> <p>x+: lectura y escritura con la creación previa del archivo (genera un error si el archivo ya existe)</p> <p>c: sólo escritura (si el archivo existe, coloca el puntero al principio del archivo sin vaciarlo ni generar error; crea el archivo si no existe)</p> <p>c+: lectura y escritura (si el archivo existe, coloca el puntero al principio del archivo sin vaciarlo ni generar error; crea el archivo si no existe)</p> <p>En Windows, agregar un b para manipular los archivos binarios (esta opción, si está presente, se ignora en Unix); utilizar un t en lugar de b especifica un modo de texto (las secuencias <code>\n</code> se reemplazarán automáticamente por un <code>\r\n</code>).</p>
utilizar_inclusión	<p>Poner 1 (o TRUE) para buscar el archivo en los directorios especificados por la directiva de configuración <code>include_path</code>.</p>

La función `fopen` devuelve un puntero de archivo en caso de éxito y `FALSE` en caso de fallo.



Por razones de portabilidad, se recomienda utilizar siempre la opción `b` cuando abra archivos con `fopen`.

fclose

La función `fclose` permite cerrar un archivo.

Sintaxis

```
booleano fclose(recurso archivo)
```

archivo	Puntero de archivo previamente abierto.
---------	---

La función `fclose` devuelve `TRUE` en caso de éxito y `FALSE` en caso de fallo.

fread

La función `fread` permite leer el contenido de un archivo (en una cadena).

Sintaxis

```
cadena fread(recurso archivo, entero longitud)
```

`archivo` Puntero de archivo previamente abierto.

`longitud` Número de bytes a leer en el archivo.

La función `fread` devuelve los datos leídos o `FALSE` en caso de fallo.

file

La función `file` permite leer el contenido de un archivo (en una matriz).

Sintaxis

```
matriz file(cadena nombre_archivo [, entero indicador])
```

`nombre_archivo` Ruta de acceso del archivo a leer.

`indicador` Uno o varios agregándolos a las constantes siguientes:
`FILE_USE_INCLUDE_PATH`: buscar el archivo en los directorios especificados por la directiva de configuración `include_path`.
`FILE_IGNORE_NEW_LINES`: no agregar la secuencia de nueva línea al final de cada línea de la matriz.
`FILE_SKIP_EMPTY_LINES`: ignorar las líneas vacías.

La función `file` devuelve los datos leídos o `FALSE` en caso de error.

fwrite

La función `fwrite` permite escribir en un archivo.

Sintaxis

```
entero fwrite(recurso archivo, cadena datos[, entero longitud])
```

`archivo` Puntero de archivo previamente abierto.

`datos` Datos a escribir en el archivo.

`longitud` Si se especifica, indica el número de bytes a escribir (toda la cadena `datos` por defecto).

La función `fwrite` devuelve el número de bytes escritos o `FALSE` en caso de fallo.

file_get_contents

La función `file_get_contents` permite leer el contenido de un archivo (en una cadena).

Sintaxis

```
cadena file_get_contents(cadena nombre_archivo [,
entero utilizar_inclusión])
```

Con

nombre_archivo	Ruta de acceso del archivo a leer.
utilizar_inclusión	Poner 1 (o TRUE) para buscar el archivo en los directorios especificados por la directiva de configuración <code>include_path</code> .

La función `file_get_contents` abre el archivo cuyo nombre se pasa como parámetro y devuelve su contenido en una cadena. La función devuelve `FALSE` en caso de error.

Utilizar esta función es más eficiente que abrir el archivo (`fopen`), leerlo (`fread`) y volverlo a cerrar (`fclose`).

file_put_contents

La función `file_put_contents` permite escribir el contenido de una cadena dentro de un archivo. Esta función apareció en la versión 5.

Sintaxis

```
entero file_put_contents(cadena nombre_archivo, cadena datos [, entero mode])
```

Con

nombre_archivo	Ruta de acceso del archivo a leer.
datos	Datos a escribir en el archivo.
modo	<p>Constante <code>FILE_USE_INCLUDE_PATH</code> para escribir en el primer directorio especificado en la directiva de configuración <code>include_path</code>.</p> <p>Constante <code>FILE_APPEND</code> para agregar los datos al final del archivo, si éste ya existe.</p> <p>Constante <code>LOCK_EX</code> para obtener un bloqueo exclusivo en el archivo durante la escritura.</p> <p>Especificar la suma de las dos constantes de combinar las dos opciones.</p>

La función `file_put_contents` abre el archivo cuyo nombre se pasa como parámetro, escribe el contenido de la cadena que se pasa como parámetro y luego cierra el archivo.

Si el archivo no existe, lo crea. Si el archivo existe, sustituye su contenido, a menos que la constante `FILE_APPEND` se haya especificado en el tercer parámetro, en cuyo caso los datos se agregan al final del archivo.

La función devuelve el número de bytes escritos en el archivo, o `FALSE` en caso de error.

copy

La función `copy` permite copiar un archivo.

Sintaxis

```
booleano copy(cadena origen, cadena destino)
```

origen Ubicación del archivo de origen.

destino Ubicación del archivo de destino.

La función `copy` realiza la copia y devuelve `TRUE` en caso de éxito y `FALSE` en caso de fallo.

unlink

La función `unlink` permite eliminar un archivo.

Sintaxis

```
booleano unlink(cadena nombre_archivo)
```

nombre_archivo Ubicación del archivo que desea eliminar.

La función `unlink` elimina el archivo, devuelve `TRUE` en caso de éxito y `FALSE` en caso de fallo.

rename

La función `rename` permite cambiar el nombre de un archivo.

Sintaxis

```
booleano rename(cadena anterior_nombre, cadena nuevo_nombre)
```

anterior_nombre Ubicación del archivo al que desea cambiar el nombre.

nuevo_nombre Nuevo nombre del archivo.

La función `rename` cambia el nombre del archivo y devuelve `TRUE` en caso de éxito y `FALSE` en caso de fallo.

file_exists

La función `file_exists` permite probar la existencia de un archivo.

Sintaxis

```
booleano file_exists(cadena nombre_archivo)
```

nombre_archivo Ubicación del archivo que desea probar.

La función `file_exists` devuelve `TRUE` si el archivo existe y `FALSE` en caso contrario.

filesize

La función `filesize` permite obtener el tamaño de un archivo.

Sintaxis

```
entero filesize(cadena nombre_archivo)
```

nombre_archivo Ubicación del archivo.

La función `filesize` devuelve el tamaño del archivo o `FALSE` en caso de fallo.

chdir

La función `chdir` permite cambiar el directorio actual.

Sintaxis

```
booleano chdir(cadena ruta)
```

ruta Directorio deseado.

La función `chdir` devuelve `TRUE` en caso de éxito y `FALSE` en caso de fallo.

opendir

La función `opendir` permite abrir un directorio.

Sintaxis

```
recurso opendir(cadena ruta)
```

ruta Directorio que desea abrir.

La función `opendir` abre un directorio para leer su contenido y devuelve un puntero a la carpeta, o `FALSE` en caso de fallo.

closedir

La función `closedir` permite cerrar un directorio.

Sintaxis

```
closedir(recurso directorio)
```

directorio Puntero de directorio previamente abierto.

La función `closedir` cierra un puntero de directorio previamente abierto.

readdir

La función `readdir` permite leer el contenido de un directorio.

Sintaxis

```
cadena readdir(recurso directorio)
```

directorio Puntero de directorio previamente abierto.

La función `readdir` devuelve el nombre del siguiente archivo en el directorio (en ningún orden en particular), o `FALSE` si ya no hay más nombres de archivos que leer en el directorio.

scandir

La función `scandir` lista el contenido de un directorio en una matriz.

Sintaxis

```
matriz scandir(cadena ruta[, entero orden])
```

ruta	Directorio correspondiente.
orden	Orden deseado: SCANDIR_SORT_NONE: ninguno. SCANDIR_SORT_ASCENDING: orden alfabético (predeterminado). SCANDIR_SORT_DESCENDING: orden alfabético inverso.

La función `scandir` devuelve una matriz que lista el contenido del directorio correspondiente o `FALSE` en caso de error.

Ejemplos de uso

Algunas operaciones básicas con archivos

```
<?php
// Abrir un archivo en escritura.
$f = fopen('info.txt','wb');
// Escribir en el archivo.
fwrite($f, 'Olivier HEURTEL');
// Cerrar el archivo.
fclose($f);
// Abrir un archivo en lectura.
$f = fopen('info.txt','rb');
// Leer en el archivo.
$texto = fread($f, filesize('info.txt'));
// Cerrar el archivo.
fclose($f);
// Mostrar la información leída
echo $texto,'<br />';
// Más simple: utilizar file_get_contents.
$texto = file_get_contents('info.txt');
// Mostrar la información leída.
echo $texto;
?>
```

Resultado

```
Olivier HEURTEL
Olivier HEURTEL
```

Visualización del contenido de un directorio

```
<?php
// Primer método: opendir + readdir + closedir.
echo '<b>1) opendir + readdir + closedir</b><br />';
// Abrir el directorio.
$dir = opendir('../documentos');
// Examinar el directorio enumerando el nombre de un archivo
// en cada iteración.
while($archivo = readdir($dir)) {
echo $archivo,'<br />';
}
// Cerrar el directorio.
```

```
closedir($dir);  
// Segundo método: scandir.  
echo '<br /><b>2) scandir</b><br />';  
// Listar el contenido del directorio en una matriz  
$archivos = scandir('../documentos');  
// Examinar el resultado.  
foreach ($archivos as $archivo) {  
    echo $archivo, '<br />';  
}  
?>
```

Resultado

1) Opendir + readdir + closedir

```
.  
..  
cv.pdf  
info.txt  
logo.jpg  
Foto.gif  
pj.doc
```

2) scandir

```
.  
..  
cv.pdf  
info.txt  
logo.jpg  
Foto.gif  
pj.doc
```

Las funciones `readdir` y `scandir` permiten recuperar el nombre de los archivos y de los directorios contenidos en un directorio. El directorio actual y el directorio superior también se devuelven con los símbolos `.` (punto) y `..` (dos puntos).

Índice

Autenticación

Crear un identificador único

Pasar la información a través de la URL

Pasar información a través de un campo de formulario oculto

Utilizar cookies

Utilizar la gestión de sesiones de PHP

Conservar la información de una visita a otra

Breve resumen de las variables

Get/Post/Cookie/Session

Enviar un correo electrónico

Información general

Enviar un mensaje de texto sin archivos adjuntos

Enviar un mensaje en formato MIME

Gestión de archivos

Manipular los archivos en el servidor

magic quotes

Cargar un archivo desde el cliente: "file upload"

Descargar un archivo desde el servidor: "download"

Administrar los errores en un script PHP

Anexo

magic quotes

En las versiones anteriores de PHP, si la directiva de configuración `magic_quotes_runtime` estaba en `on`, los datos leídos en los archivos tomaban un código "magic quotes".

En la función 5.4, la función "magic quotes" se ha eliminado definitivamente.

En consecuencia, un dato vinculado en un archivo no toma ningún código "magic quotes" de forma automática, lo cual es bueno para poder manipular este dato y en particular su visualización. Si el dato leído está destinado a ser registrado en la base de datos, será necesario escapar correctamente los apóstrofes (ver capítulo Acceder a las bases de datos).

Subir

Condiciones generales de uso

Copyright - ©Editions ENI



Cargar un archivo desde el cliente: "file upload"

Algunos sitios pueden ofrecer a los usuarios cargar documentos desde su equipo al servidor Web: enviar un curriculum vitae en un sitio (sitio de búsqueda de empleo), adjuntar un archivo en un mensaje (sitio de mensajería) o, simplemente, guardar el documento en el servidor (sitio de almacenamiento).

En terminología anglosajona, esta característica se denomina "file upload" (en oposición a "download", que permite descargar un documento desde el servidor a su equipo).

Esta característica es muy fácil de implementar en PHP y tiene dos requisitos:

- en un formulario, proporcionar un espacio para que el usuario especifique la ubicación del archivo en su equipo;
- en el script de procesamiento del formulario, recuperar el archivo enviado por el usuario y hacer algo.

En el capítulo Gestionar los formularios , discutimos la posibilidad de disponer en un formulario una zona que permite indicar la ubicación de un archivo en su equipo (`type="file"`):

Colocar una zona de este tipo no es suficiente. Para provocar la transferencia del archivo, basta con añadir el atributo `enctype="multipart/form-data"` en la etiqueta `<form>`:

```
<form action="entrada.php" method="post"
      enctype="multipart/form-data">
```



Esta técnica funciona sólo con los formularios que utilizan el método `POST`.

Además, es posible añadir una zona oculta en el formulario para limitar el tamaño de los archivos que se pueden enviar al servidor. Esta zona oculta, obligatoriamente situada antes de la zona de tipo `file` debe llamarse `MAX_FILE_SIZE` (atributo `name`) y especificar el tamaño máximo en bytes en el atributo `value`:

Ejemplo de zona oculta para limitar el tamaño de los archivos a 10 KB

```
<input type="hidden" name="MAX_FILE_SIZE" value="10240">
```

El valor especificado en esta zona no puede superar el valor de la directiva de configuración `upload_max_filesize` (2 MB por defecto). Si la zona oculta no está presente, se aplica el tamaño especificado en la directiva `upload_max_filesize`.

Por otra parte, la carga sólo es posible si la directiva de configuración `file_uploads` está en `on`.

Ejemplo de formulario completo

```
<form action="entrada.php" method="post"
      enctype="multipart/form-data">
<div>
  Archivo:
  <input type="file" name="archivo" value="" />
  <input type="submit" name="ok" value="OK" />
</div>
</form>
```

Cuando un archivo se envía con un formulario, la información sobre este archivo es accesible en el script PHP gracias a la variable `$_FILES`; y el valor introducido por el usuario ya no está disponible en `$_POST`.

`$_FILES` es una matriz asociativa multidimensional; la primera clave es igual al nombre de la zona de tipo `file` del formulario (`archivo` en nuestro ejemplo) y la matriz asociativa correspondiente presenta cinco líneas:

Clave	Valor
<code>name</code>	Nombre del archivo (sin ruta de acceso)
<code>tipo</code>	Tipo MIME del archivo (suministrado por el navegador)
<code>size</code>	Tamaño del archivo en bytes
<code>tmp_name</code>	Nombre del archivo temporal creado en el servidor (ruta completa)
<code>error</code>	Código de error. Una de las constantes siguientes: <code>UPLOAD_ERR_OK</code> (0): ningún error. <code>UPLOAD_ERR_INI_SIZE</code> (1): tamaño del archivo superior al tamaño definido por la directiva de configuración <code>upload_max_filesize</code> . <code>UPLOAD_ERR_FORM_SIZE</code> (2): tamaño del archivo superior al tamaño definido por la opción <code>MAX_FILE_SIZE</code> del formulario. <code>UPLOAD_ERR_PARTIAL</code> (3): el archivo se ha cargado parcialmente. <code>UPLOAD_ERR_NO_FILE</code> (4): ningún archivo introducido. <code>UPLOAD_ERR_NO_TMP_DIR</code> (6): ningún directorio temporal. Se incluyó en la versión 5.0.3. <code>UPLOAD_ERR_CANT_WRITE</code> (7): error al escribir el archivo en el disco. Se incluyó en la versión 5.1.0. <code>UPLOAD_ERR_EXTENSION</code> (8): transferencia detenida por una extensión. Se incluyó en la versión 5.2.0.



Según la versión, existe un código de error 5, sin nombre de constante, en caso de error en el archivo introducido (nombre, ruta). Este código de error ya no existe desde la versión 5.2.

En el servidor, el archivo transferido es un archivo temporal con un nombre del tipo `php*.tmp`, situado en el directorio definido por la directiva de configuración `upload_tmp_dir`.

La estructura de `$_FILES` permite tener varias zonas de tipo `file` en el formulario y, por lo tanto, autorizar la carga de varios archivos.

Ejemplo

El formulario contiene dos zonas de tipo `file` llamadas `archivo1` y `archivo2`:

archivo1	name	Foto.gif
	tipo	image/gif
	tmp_name	d:\temp\phpB.tmp
	size	2376
archivo2	name	cv.pdf
	tipo	application/pdf
	tmp_name	d:\temp\php12.tmp

	size	52147
--	------	-------

Si el archivo temporal creado en el servidor no se explota (nombrar/copiar/mover) por el script PHP que procesa el formulario, se elimina automáticamente al final del script. En el script PHP, por lo tanto, debe manejar el archivo temporal según las necesidades de la aplicación.

Ejemplo completo

```
<?php
// Inclusión del archivo que contiene las funciones generales.
include('funciones.inc');
// Inicialización de la variable de mensaje.
$mensaje = '';
// Procesamiento del formulario.
if (isset($_POST['ok'])) {
    // Recuperar la información sobre el archivo.
    $información = $_FILES["archivo"];
    // Extrayendo:
    //     - su nombre;
    $nombre = $información['name'];
    //     - su tipo MIME.
    $tipo_mime = $información['type'];
    //     - su tamaño.
    $tamaño = $información['size'];
    //     - la ubicación del archivo temporal.
    $archivo_temporal = $información['tmp_name'];
    //     - el código de error.
    $código_error = $información['error'];
    // Controles y procesamiento.
    switch ($código_error) {
        case UPLOAD_ERR_OK:
            // Archivo recibido.
            // Determinar su destino final.
            $destino = "/app/documentos/$nombre";
            // Copiar el archivo temporal (probar el resultado).
            if (copy($archivo_temporal,$destino)) {
                // Copia OK => mostrar un mensaje de confirmación.
                $mensaje = "Transferencia finalizada - Archivo = $nombre - ";
                $mensaje .= "Tamaño = $tamaño bytes - ";
                $mensaje .= "Tipo MIME = $tipo_mime.";
            } else {
                // Problema al copiar => mostrar un mensaje de error.
                $mensaje = 'Problema al copiar al servidor.';
            }
            break;
        case UPLOAD_ERR_NO_FILE:
            // El archivo introducido no existe.
            $mensaje = 'El archivo introducido no existe.';
            break;
        case UPLOAD_ERR_INI_SIZE:
            // Tamaño archivo > upload_max_filesize.
            $mensaje = "Archivo '$nombre' no transferido ";
            $mensaje .= ' (tamaño > upload_max_filesize).';
            break;
        case UPLOAD_ERR_FORM_SIZE:
            // Tamaño archivo > MAX_FILE_SIZE.
            $mensaje = "Archivo '$nombre' no transferido ";
            $mensaje .= ' (tamaño > MAX_FILE_SIZE).';
            break;
        case UPLOAD_ERR_PARTIAL:
            // Archivo parcialmente transferido.
```

```

    $mensaje = "Archivo '$nombre' no transferido ";
    $mensaje .= ' (se produjo un problema durante la transferencia).';
    break;
case UPLOAD_ERR_NO_TMP_DIR:
    // ningún directorio temporal.
    $mensaje = "Archivo '$nombre' no transferido ";
    $mensaje .= ' (ningún directorio temporal).';
    break;
case UPLOAD_ERR_CANT_WRITE:
    // Error al escribir el archivo en el disco.
    $mensaje = "Archivo '$nombre' no transferido ";
    $mensaje .= ' (error al escribir el archivo en el
disco).';
    break;
case UPLOAD_ERR_EXTENSION:
    // Transferencia detenida por la extensión.
    $mensaje = "Archivo '$nombre' no transferido ";
    $mensaje .= ' (transferencia detenida por la extensión).';
    break;
default:
    // ¡Error inesperado!
    $mensaje = "Archivo no transferido ";
    $mensaje .= " (error desconocido: $código_error ).";
}
}
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head><title>Upload</title></head>
<body>
    <form action="upload.php" method="post"
        enctype="multipart/form-data">

    <div>
        Archivo:
        <input size="100" type="file" name="archivo" value="" />
        <input type="submit" name="ok" value="OK" /><br />
        <?php echo vers_page($mensaje); ?>
    </div>
</form>
</body>
</html>

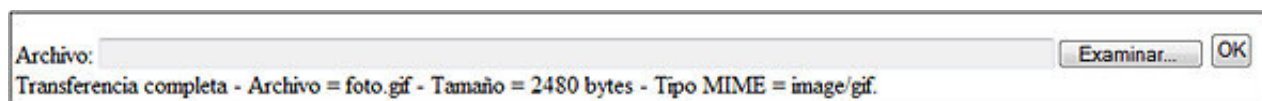
```

Resultado

- Vista inicial del formulario y selección de un archivo (usando el botón **Examinar**):



- Resultado al hacer clic en el botón **OK**:



En este ejemplo, el archivo se copia en un directorio en el servidor.

Descargar un archivo desde el servidor: "download"

Para descargar un archivo desde el servidor, es posible utilizar el método clásico que consiste en utilizar un vínculo (etiqueta <a>).

Ejemplo

```
<a href="cv.pdf">Descargar</a>
```

Esta técnica puede causar problemas:

- Es el navegador el que decide ofrecer al usuario un diálogo para guardar o mostrar el documento directamente, si sabe cómo.
- Los archivos que son interpretados por el servidor (.php por ejemplo) o por el navegador (.htm por ejemplo) no se pueden cargar de esta manera.

Es posible utilizar otra técnica, si desea forzar al navegador a que ofrezca al usuario un diálogo para guardar el archivo y que lo haga para cualquier tipo de documento.

Esta técnica consiste en enviar algunos encabezados específicos usando la función `header`, seguido del documento propiamente dicho.

- No hay garantía de que esta técnica funcione correctamente o de la misma manera con todos los navegadores. En particular, hay problemas conocidos con algunas versiones antiguas de Internet Explorer.

Los encabezados mínimos que hay que enviar son los siguientes:

Content-Disposition: attachment; filename=...	Este encabezado sugiere al navegador que procese el documento como un archivo adjunto (attachment) y proponer al usuario que lo guarde con el nombre definido por filename.
Content-Type: ...	Este encabezado comunica al navegador el tipo MIME del documento.

Algunos tipos MIME comunes:

Tipo MIME	Naturaleza del documento
application/msword	Documento de Microsoft Word
application/octetstream	Genérico
application/pdf	Documento PDF
application/vnd.ms-excel	Documento de Microsoft Excel
application/vnd.ms-powerpoint	Documento de Microsoft PowerPoint
application/zip	Documento Zip
image/bmp	Imagen en formato de mapa de bits.
image/gif	Imagen en formato GIF
image/jpeg	Imagen en formato JPEG

image/tiff	Imagen en formato TIFF
image/png	Imagen en formato PNG
text/html	Documento HTML
text/plain	Documento de texto

Al poner cualquier cosa como tipo MIME (x/y por ejemplo), el navegador normalmente debe arreglárselas con la extensión del documento.

Después de enviar los encabezados, sólo queda enviar el documento "directamente" a la página, bien mediante una lectura (fread) seguido de un echo, o más sencillamente mediante la función `readfile` que lee un archivo y lo envía directamente a la salida.

Sintaxis

```
entero readfile(cadena nombre_archivo [, entero utilizar_inclusión])
```

nombre_archivo	Ruta de acceso del archivo a leer.
utilizar_inclusión	Poner 1 (o TRUE) para buscar el archivo en los directorios especificados por la directiva de configuración <code>include_path</code> .

La función `readfile` devuelve el número de bytes leídos o FALSE en caso de error.

Esta técnica se ilustrará con la ayuda de un script `download.php` que ofrece una lista de documentos para descargar. Presentamos dos ejemplos: un ejemplo que utiliza un formulario y un ejemplo que utiliza enlaces.

En ambos casos, la lista de documentos que se pueden descargar está codificada en el script. En una aplicación real, esta lista de documentos provendría sin duda de una base de datos (véase el capítulo Acceder a las bases de datos).

Utilización de un formulario

En este primer ejemplo, el script genera un formulario que tiene un botón de tipo "image" (`type="image"`) para cada archivo propuesto para su descarga, el nombre del botón es igual al número del documento.

Como vimos en el capítulo Gestionar los formularios, recuperamos en `$_POST` dos variables `n_x` y `n_y` dando respectivamente la posición relativa horizontal y vertical del clic en el interior de la imagen (n siendo el nombre de la imagen donde se hace clic, en nuestro caso un número).

Ejemplo

```
<?php
// Lista de documentos (procedente sin duda de una
// base de datos en una aplicación real).
$documentos = array('cv.pdf','foto.gif');
// Procesamiento del formulario si $_POST no está vacío.
if (! empty($_POST)) {
    // Recuperar el número del documento.
    // Tomar la clave de la primera línea de $_POST
    // (normalmente del tipo n_x, n siendo el número del documento).
    list($numero) = each($_POST);
    // Convertir la cadena en entero => sólo queda el n°.
    $numero = (integer) $numero;
```

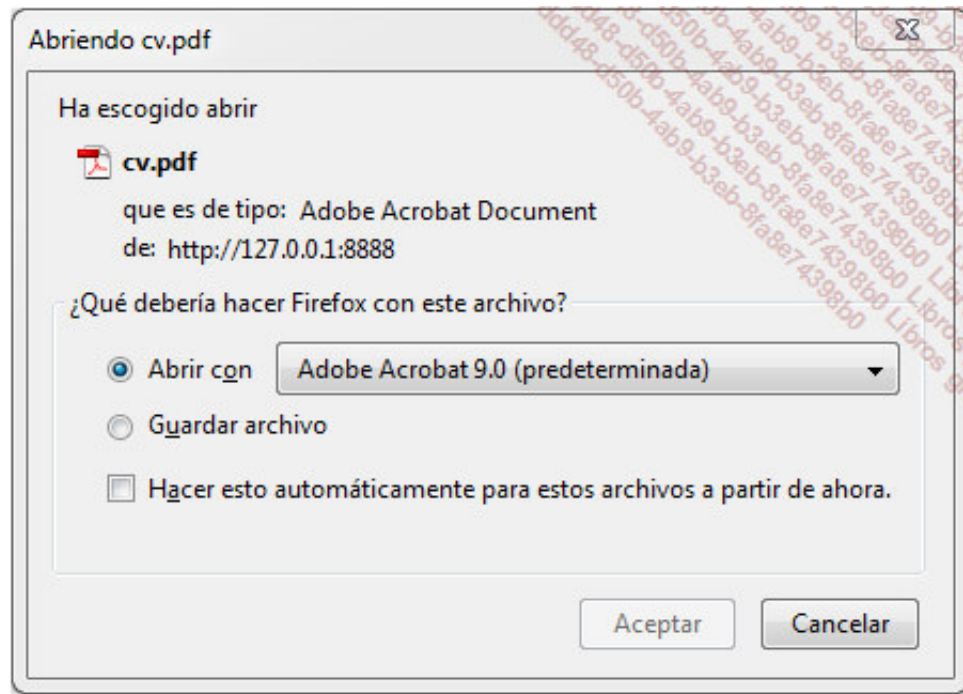
```
// Deduciendo el nombre del documento.
$nombre_archivo = $documentos[$número];
// Enviar el encabezado de adjunto.
$header = "Content-Disposition: attachment; ";
$header .= "filename=$nombre_archivo\n" ;
header($header);
// Enviar el encabezado del tipo MIME (aquí, "desconocido").
header("Content-Type: x/y\n");
// Enviar el documento.
// No hay cifrado de tipo magic quotes antes de leer el archivo.
set_magic_quotes_runtime(0);
readfile($nombre_archivo);
}
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head><title>Download</title></head>
<body>
<form action="Download.php" method="post">
<table border="1" cellpadding="4" cellspacing="0">
<tr align="center">
<th>documento</th><th>descargar</th>
</tr>
<?php
// Una pequeña porción de código PHP para generar las líneas de la
// matriz que presenta la lista de documentos.
// Examinar la lista de documentos y utilizar el nombre
// para la visualización y el número como nombre de la imagen.
foreach($documentos as $número => $documento) {
    echo sprintf
    (
        "<tr><td>%s</td><td align=\"center\">%s</td></tr>\n",
        $documento,
        "<input type=\"image\" name=\"$número\"
        src=\"download.gif\" />"
    );
}
?>
</table>
</form>
</body>
</html>
```

Resultado (en Firefox)

- Vista inicial de la página:

documento	descargar
cv.pdf	
foto.gif	

- Después de hacer clic en la imagen asociada al documento cv.pdf:



Utilización de vínculos

En este segundo ejemplo, el script genera una matriz que incluye un enlace para cada archivo propuesto para su descarga. El enlace llama de nuevo al script pasando el número del documento como un parámetro en la URL.

Ejemplo

```
<?php
// Lista de documentos (procedente sin duda de una
// base de datos en una aplicación real).
$documentos = array('cv.pdf','foto.gif');
// Procesamiento del formulario si $_GET no está vacío.
if (! empty($_GET)) {
    // Recuperar el número del documento.
    $numero = $_GET['no'];
    // Deduciendo el nombre del documento.
    $nombre_archivo = $documentos[$numero];
    // Enviar el encabezado de adjunto.
    $header = "Content-Disposition: attachment; ";
    $header .= "filename=$nombre_archivo\n" ;
    header($header);
    // Enviar el encabezado del tipo MIME (aquí, "desconocido").
    header("Content-Type: x/y\n");
    // Enviar el documento.
    // No hay cifrado de tipo magic quotes antes de leer el archivo.
    set_magic_quotes_runtime(0);
    readfile($nombre_archivo);
}
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head><title>Download</title></head>
<body>
<table border="1" cellpadding="4" cellspacing="0">
```

```

<tr align="center"><th>documento</th></tr>
<?php
// Una pequeña porción de código PHP para generar las líneas de la
// matriz que presenta la lista de documentos.
// Examinar la lista de documentos y utilizar el nombre
// para la visualización y el número en la URL.
foreach($documentos as $numero => $documento) {
    echo sprintf
    (
        "<tr><td>%s</td></tr>\n",
        "<a href=\"download.php?no=$numero\">$documento</a>"
    );
}
?>
</table>
</body>
</html>

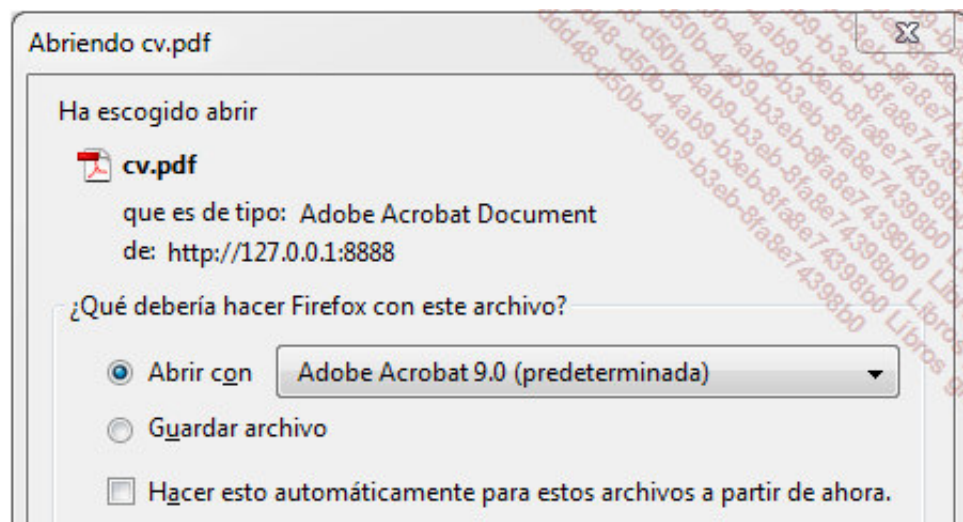
```

Resultado (en Firefox)

- Vista inicial de la página:

documento
cv.pdf
foto.gif

- Después de hacer clic en el enlace asociado al documento cv.pdf:



Subir

Índice

Información

[Título, autor...](#)

Introducción

[Objetivo del libro](#)[Breve historia de PHP](#)[¿Dónde conseguir PHP?](#)[Convenciones de escritura](#)

Información general sobre PHP

[¿Qué es PHP?](#)[Estructura básica de una página PHP](#)[Configuración de PHP](#)[Utilizar PHP desde la línea de comandos](#)

Variables, constantes, tipos y matrices

Operadores

Estructuras de control

Funciones y clases

Gestión de formularios

Acceder a las bases de datos

Administrar las sesiones

Enviar un correo electrónico

Gestión de archivos

Administrar los errores en un script PHP

Anexo

¿Qué es PHP?

PHP es un lenguaje de script que se ejecuta del lado del servidor, el código PHP se incluye en una página HTML normal. Por lo tanto, se puede comparar con otros lenguajes de script que se ejecutan según el mismo principio: ASP (*Active Server Pages*), JSP (*Java Server Pages*) o PL/SQL Server Pages (PSP).

A diferencia de un lenguaje como JavaScript, donde el código se ejecuta del lado del cliente (en el explorador), el código PHP se ejecuta del lado del servidor. El resultado de esta ejecución se incrusta en la página HTML, que se envía al navegador. Este último no tiene conocimiento de la existencia del procesamiento que se ha llevado a cabo en el servidor.

Esta técnica permite realizar páginas Web dinámicas cuyo contenido se puede generar total o parcialmente en el momento de la llamada de la página, gracias a la información que se recopila en un formulario o se extrae de una base de datos.

Ejemplo sencillo de página PHP

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Ejemplo de página PHP</title>
  </head>
  <body>
    <?php
      echo '<p>iHola Olivier!</p>';
    ?>
  </body>
</html>
```

La parte en negrita es el código PHP incluido en la página HTML dentro de las etiquetas `<?php` y `?>`. En este sencillo ejemplo, el código PHP, simplemente muestra un texto estático "¡Hola Olivier!" gracias a la función `echo`. En un programa real de PHP, es probable que este texto se genere de forma dinámica en función de la identificación del usuario.

Para indicar al servidor Web que una página HTML contiene código PHP que debe ejecutarse, basta con dar al archivo una extensión específica: `.php` (excepto en caso de configuración especial del servidor).

El siguiente diagrama explica cómo el servidor Web procesa un archivo PHP.

Índice

Pasar información a través de un campo de formulario oculto

Utilizar cookies

Utilizar la gestión de sesiones de PHP

Conservar la información de una visita a otra

Breve resumen de las variables
Get/Post/Cookie/Session

Enviar un correo electrónico

Información general

Enviar un mensaje de texto sin archivos adjuntos

Enviar un mensaje en formato MIME

Gestión de archivos

Manipular los archivos en el servidor

magic quotes

Cargar un archivo desde el cliente: "file upload"

Descargar un archivo desde el servidor: "download"

Administrar los errores en un script PHP

Información general

Mensajes de error de PHP

Las funciones de gestión errores

Información general

Un error, en un script PHP, se puede manifestar de dos maneras, posiblemente al mismo tiempo:

- por un valor de retorno determinado de la función PHP en la que se encuentra el error;
- por un mensaje enviado directamente a la página.

Ejemplos

Función	Comportamiento en caso de error
require	Si el archivo parametrado no existe, se muestra un mensaje <u>pero</u> la función no devuelve ningún código especial.
mysqli_query	Si el servidor MySQL devuelve un error en la ejecución de la consulta, no aparece ningún mensaje <u>pero</u> La función devuelve FALSE (la naturaleza del error se puede recuperar por otras funciones).
oci_execute	Si el servidor Oracle devuelve un error en la ejecución de la consulta, aparece un mensaje <u>y</u> la función devuelve FALSE (la naturaleza del error se puede recuperar por otras funciones).

Por tanto, gestionar los errores en un script PHP consiste, en general, en establecer un mecanismo para detectar la generación de un error con el fin de que éste muestre un mensaje en lugar del mensaje que muestra PHP directamente.

Subir



Mensajes de error de PHP

Los mensajes de error (o avisos) que muestra PHP tienen un nivel correspondiente a su gravedad:

Valor	Constante asociada	Descripción
1	E_ERROR	Error fatal de ejecución (mensaje "fatal error: ..."). El script no se ejecuta. Ejemplos: llamada a una función que no existe, archivo mencionado en la instrucción <code>require</code> que no existe.
2	E_WARNING	Alerta de ejecución (mensaje "warning: ..."). El script continúa. Ejemplos: intento de apertura (con <code>fopen</code>), de un archivo no existente, apertura de una conexión MySQL sin éxito... En general, si el script continúa provoca otros mensajes del mismo tipo.
4	E_PARSE	Error de compilación ("Parse error: ..."). El script no se ejecuta. Ejemplos: olvido de un punto y coma, un paréntesis de cierre...
8	E_NOTICE	Advertencia durante la ejecución (mensaje "Notice: ..."). Por defecto, PHP está configurado para no mostrar estas advertencias. El script continúa. Ejemplo: utilización de una variable no inicializada.
16	E_CORE_ERROR	Error grave al inicializar PHP.
32	E_CORE_WARNING	Alerta durante la inicialización de PHP.
64	E_COMPILE_ERROR	Error grave durante la compilación.
128	E_COMPILE_WARNING	Alerta durante la compilación.
256	E_USER_ERROR	Error generado por el desarrollador.
512	E_USER_WARNING	Alerta generada por el desarrollador.
1024	E_USER_NOTICE	Advertencia generada por el desarrollador.
2048	E_STRICT	Consejos durante la ejecución. Permite a PHP sugerir cambios para mejorar la portabilidad del código, especialmente para las futuras versiones (con una función en desuso, por ejemplo). Este nivel apareció en la versión 5.
4096	E_RECOVERABLE_ERROR	Error grave recuperable. Si el desarrollador no gestiona el error (ver más abajo), se detiene el script. Se incluyó en la versión 5.2.0.
8192	E_DEPRECATED	Alertas durante la ejecución. Permite mostrar alertas sobre el código que utiliza características obsoletas y que podría no

		funcionar ya en futuras versiones. Se incluyó en la versión 5.3.
16384	E_USER_DEPRECATED	Alerta de depreciación generada por el desarrollador. Se incluyó en la versión 5.3.
32767	E_ALL	Desde la versión 5.4, todos los errores y advertencias (suma de los niveles anteriores). Antes de la versión 5.4, E_ALL no integraba el nivel E_STRICT y por tanto su valor era 30719.

Ejemplo de error grave

```
<?php
$archivo = fope('/tmp/info.txt','r');
$texto = fread($archivo,100);
fclose($archivo);
?>
```

Resultado

Fatal error: Call to undefined function: **fope()** in
/app/scripts/index.php on line 2

Ejemplos de alerta

```
<?php
$archivo = fopen('/tmp/infos.txt','r');
$texto = fread($archivo,100);
fclose($archivo);
?>
```

Resultado

Warning: fopen(/tmp/infos.txt): failed to open stream:
No such file or directory in /app/scripts/index.php on line 2
Warning: fread () expects parameter 1 to be resource, boolean given
in /app/scripts/index.php on line 3
Warning: fclose() expects parameter 1 to be resource, boolean given
in /app/scripts/index.php on line 4

Ejemplo de error de análisis

```
<?php
echo '¡Hola!<br />' // ;falta el punto y coma!
echo '¡Bienvenido!<br />';
?>
```

Resultado

Parse error: syntax error, unexpected T_ECHO, expecting ','
or ';' in /app/scripts/index.php on line 3

Las funciones de gestión errores

PHP ofrece varias funciones que permiten gestionar correctamente los errores en un script:

Nombre	Función
<code>error_reporting</code>	Define los niveles de errores que se muestran por PHP.
<code>error_log</code>	Envía un mensaje de errores a un destino (archivo, por ejemplo).
<code>set_error_handler</code>	Especifica el nombre de una función de usuario que se utiliza como controlador de error.
<code>restore_error_handler</code>	Restaura el controlador de error antiguo.
<code>trigger_error</code> <code>user_error</code>	Genera un error definido por el desarrollador (<code>user_error</code> es un alias de <code>trigger_error</code>).
<code>error_get_last</code>	Devuelve información sobre el último error encontrado en el script.

Además, el operador @, colocado antes del nombre de una función, permite eliminar la visualización de los mensajes generados, en caso de error, en la función.

Ejemplo

```
<?php
```

```
$archivo = @fopen('/tmp/infos.txt','r');
$texto = @fread($archivo,100);
@fclose($archivo);
?>
```

Al ejecutar este script, no aparece ningún mensaje ya que el archivo solicitado no existe.

Si el error provoca una parada del script, el uso del operador @ no cambia nada; la página mostrada está vacía o incompleta y no se muestra ningún mensaje al usuario. Por tanto, es necesario probar el resultado de las funciones, o usar un controlador de error para controlar el flujo del programa y realizar los procesamientos necesarios.

error_reporting

La función `error_reporting` permite definir los niveles de errores para los cuales el programa permite a PHP mostrar los mensajes.

Sintaxis

```
entero error_reporting([entero niveles])
```

niveles Niveles de error mostrados por PHP, expresados en forma de una suma de los valores asignados a cada nivel.

La

función `error_reporting` devuelve el valor anterior. Llamada sin parámetros, esta función sólo devolverá el valor actual, sin cambiar nada.

Para definir los niveles deseados, se recomienda encarecidamente, para asegurar la compatibilidad futura, utilizar las constantes y no poner un los valores duros en el programa.

La constante `E_ALL`, igual a la suma de todas las otras constantes (excepto `E_STRICT`), se puede utilizar para solicitar la visualización de todos los niveles de error.

Por el contrario, un nivel igual a 0 provoca la supresión de la visualización de todos los mensajes; esto es equivalente, para el conjunto del script, al operador @ que se puede utilizar en una función.

Del mismo modo que se definen los valores de niveles de error, especificar varios niveles es muy sencillo por medio de operaciones aritméticas en las constantes.

Ejemplo

```
E_ERROR+E_WARNING
```

Niveles `E_ERROR` y `E_WARNING`

```
E_ALL-E_USER_ERROR-
E_USER_WARNING-E_USER_
NOTICE
```

Todos los niveles excepto E_USER_ERROR, E_USER_WARNING, y E_USER_NOTICE

El valor por defecto (todo excepto E_NOTICE = E_ALL-E_NOTICE) se define por la directiva de configuración error_reporting.

Además, la directiva de configuración display_errors permite autorizar (on) o prohibir (off) la visualización de los mensajes de error; si display_errors es off, dar un valor cualquiera a error_reporting (en el archivo .ini o en un script) no tiene ningún efecto.

En fase de desarrollo, es aconsejable mostrar todos los errores (E_ALL), incluye las advertencias, para escribir el código lo más limpio posible.

Ejemplo

```
<?php
// Valor actual de error_reporting.
echo '<b>error_reporting = ',error_reporting(),'/</b><br />';
// Por defecto igual a todo excepto E_NOTICE = E_ALL - E_NOTICE.
echo '= E_ALL - E_NOTICE = ',(E_ALL-E_NOTICE),'/<br />';
// Visualización de una variable no inicializada.
echo "\$x (no inicializada) = \$x => ningún mensaje <br />";
// Pase de error_reporting a E_ALL (todo).
error_reporting(E_ALL);
echo '<b>error_reporting = E_ALL</b><br />';
// Visualización de una variable no inicializada.
echo "\$x (no inicializada) = \$x => mensaje <br />";
// Lectura de un archivo que no existe.
if (! readfile('/tmp/infos.txt')) {
    echo 'Error en readfile => mensaje<br />';
};
// Pase de error_reporting a 0 (nada).
error_reporting(0);
echo '<b>error_reporting = 0</b><br />';
// Lectura de un archivo que no existe.
if (! readfile('/tmp/infos.txt')) {
    echo 'Error en readfile => más mensaje<br />';
};
?>
```

Resultado

```
error_reporting = 32759
= E_ALL - E_NOTICE = 32759
\$x (no inicializada) = => ningún mensaje
error_reporting = E_ALL
Notice: Undefined variable: x in /app/scripts/index.php on
line 20
\$x (no inicializada) = => mensaje
Warning: readfile(/tmp/infos.txt): failed to open stream:
No such file or directory in /app/scripts/index.php on line 22
Error en readfile => mensaje
error_reporting = 0
Error en readfile => más mensaje
```

En general, una vez en producción, se recomienda desactivar la visualización de mensajes de error (ya sea por una directiva de configuración, o llamando a error_reporting(0) al comienzo de cada script para ser independiente de la configuración). Esta inhibición de mensajes de error, justificada en parte por razones de seguridad (los mensajes de error PHP revelan información sobre el árbol del servidor), permite también mostrar mensajes propios.

error_log

La función `error_log` permite enviar un mensaje de error a un destino determinado.

Sintaxis

```
entero error_log(cadena mensaje, entero tipo_destino,
[cadena destino[, cadena complemento]])
```

La

mensaje	Mensaje a enviar.
tipo_destino	Tipo de destino: 0: historial de PHP 1: dirección de correo electrónico 2: ya no es una opción (antiguamente puesto de depuración) 3: archivo 4: controlador de identificación SAPI
destino	Especifica el destino para los tipos 1 y 3: 1: dirección de correo electrónico 3: nombre del archivo (creado automáticamente si no existe)
complemento	Encabezado(s) complementario(s) a enviar en el mensaje del caso 1 (ver la función <code>mail</code> en el capítulo Enviar un correo electrónico para más información).

función `error_log` devuelve `TRUE` en caso de éxito y `FALSE` en caso contrario.

Por el tipo de destino 0 (historial de PHP), el archivo de salida se define por la directiva de configuración `error_log`.



Si la directiva de configuración `log_errors` está en `on`, los mensajes se escriben siempre en el archivo especificado por la directiva `error_log`, sin que sea necesario llamar a la función `error_log`.

Ejemplo

```
<?php
// No se muestran errores en el script.
error_reporting(0);
// Lectura de un archivo que no existe.
$nombre_archivo = '/tmp/infos.txt';
if (! readfile($nombre_archivo)) {
    // Escritura de un mensaje de error en un archivo de seguimiento
    // especifico a la aplicación.
    error_log("No se puede leer el archivo $nombre_archivo.\n",
        3, '/app/logs/miAplicación.log');
    // Visualización de un mensaje para el usuario.
    echo 'No se puede completar su solicitud; ',
        'vuelva a intentarlo más tarde.';
};
?>
```

Resultado en el archivo `miAplicación.log`

No se puede leer el archivo `/tmp/infos.txt`.

Resultado en el navegador

No se puede completar su solicitud; vuelva a intentarlo más tarde.

La función `error_log` es útil, en la fase de pruebas o de producción, para conservar un seguimiento de un error en alguna parte. Sin embargo, no sustituye a la visualización de un mensaje explícito para el usuario.

set_error_handler

La función `set_error_handler` permite especificar el nombre de una función de usuario que debe llamarse para gestionar los errores de forma centralizada.

Sintaxis

```
cadena set_error_handler(cadena función [, entero niveles])
```

función	Nombre de la función encargada de gestionar los errores.	La
niveles	Niveles de error correspondientes (apareció en la versión 5).	

función `set_error_handler` devuelve el nombre de la anterior función responsable de la gestión de errores (cadena vacía si no había ninguna) o `FALSE` en caso de error. Desde la versión 5.5, el valor `NULL` se puede pasar por el parámetro `function` para reinicializar el controlador a su valor predeterminado.

La función de gestión de errores debe aceptar al menos dos parámetros: el primero, para el nivel del error y el segundo para el mensaje de error. Se pueden especificar tres parámetros adicionales para el nombre del archivo en el que se produjo el error, el número de la línea donde se generó el error y el contexto del error (matriz con todas las variables existentes en el momento del error).

Si la función de gestión de errores devuelve `FALSE`, entonces la gestión de errores normal de PHP continúa. De lo contrario, desde el momento en que se especifica un controlador de error, PHP no muestra ningún mensaje adicional, independientemente del valor de `error_reporting`. Además, de forma predeterminada, se llama al controlador de error para todos los errores, independientemente de su nivel, sea cual sea el valor de `error_reporting`. El parámetro `niveles` permite especificar los niveles adecuados. Los niveles `E_ERROR`, `E_PARSE`, `E_CORE_ERROR`, `E_CORE_WARNING`, `E_COMPILE_ERROR` y `E_COMPILE_WARNING` no pueden gestionarse de esta manera.

Ejemplo

```
<?php
// Definir el controlador de error.
function controlador_errores
    ($nivel,$mensaje,$archivo,$línea) {
    // Mostrar el archivo correspondiente, con el número de línea.
    echo "Archivo = $archivo<br />";
    echo "Línea   = $línea<br />";
    // Mostrar el nivel y el mensaje.
    echo "Nivel = $nivel <br />";
    echo "Mensaje = $mensaje<br />";
    // No ejecutar el controlador interno de PHP
    // (superfluo, como es el caso si la función no devuelve
    // explícitamente FALSE).
    return TRUE;
}
// Especificar el controlador que se va a utilizar.
set_error_handler('controlador_errores');
// Generar un error.
readfile('/tmp/infos.txt');
// Mostrar un mensaje de fin.
echo 'Fin';
?>
```

Resultado

```
Archivo = /app/scripts/index.php
Línea = 19
Nivel = 2
Mensaje = readfile(/tmp/infos.txt):
failed to open stream: No such file or directory
Fin
```

Este ejemplo muestra que si el error no causa la interrupción del script, este último continúa después de la llamada al controlador de error; por lo que es su responsabilidad detener la ejecución del script, utilizando la instrucción `exit` o de la función `die` (véase el capítulo Estructuras de control - sección Interrumpir el script).

Ejemplo

```

<?php
// Definir el controlador de error.
function controlador_errores
    ($nivel,$mensaje,$archivo,$línea) {
    // Mostrar el archivo correspondiente, con el número de línea.
    echo "Archivo = $archivo<br />";
    echo "Línea    = $línea<br />";
    // Mostrar el nivel y el mensaje.
    echo "Nivel = $nivel <br />";
    echo "Mensaje = $mensaje<br />";
    // Interrumpir el script.
    exit;
}
// Especificar el controlador que se va a utilizar.
set_error_handler('controlador_errores');
// Generar un error.
readfile('/tmp/infos.txt');
// Mostrar un mensaje de fin.
echo 'Fin';
?>

```

Resultado

```

Archivo = /app/scripts/index.php
Línea = 17
Nivel = 2
Mensaje = readfile(/tmp/infos.txt):
failed to open stream: No such file or directory

```

En el controlador de error, se puede utilizar la función `header` (véase el capítulo Gestión de formularios - sección Ir a otra página al final del procesamiento) para redirigir al usuario a una página HTML o un script PHP encargada de mostrar los mensajes de error.

Esta técnica es muy útil porque permite centralizar la gestión de errores (la función se puede definir en un archivo incluido) y separar claramente el código encargado del procesamiento normal, del código responsable de gestionar los errores.

restore_error_handler

La función `restore_error_handler` permite restaurar el anterior controlador de error después de un cambio efectuado con `set_error_handler`.

Sintaxis

```
restore_error_handler()
```

Ejemplo

```

<?php
// Definir un primer controlador de errores.
function controlador1 ($numero,$mensaje) {
    // Muestra un mensaje simple.
    echo '=> controlador n° 1<br />';
}
// Definir un segundo controlador de errores.
function controlador2 ($numero,$mensaje) {
    // Muestra un mensaje simple.
    echo '=> controlador n° 2<br />';
}
// Definir una función que genera un error.
function generar_error() {
    // Mostrar un mensaje.
    echo 'Generar un error<br />';
    // Leer un archivo que no existe.
}

```

```

    readfile('/tmp/infos.txt');
}
// Primera secuencia: sin controlador.
echo '<b>Sin controlador</b><br />';
generar_error();
// Segunda secuencia: controlador número 1.
set_error_handler('controlador');
echo '<b>Utilizar el controlador nº 1</b><br />';
generar_error();
// Tercera secuencia: controlador número 2.
set_error_handler('controlador2');
echo '<b>Utilizar el controlador nº 2</b><br />';
generar_error();
// Cuarta secuencia: restaurar el anterior controlador.
restore_error_handler();
echo '<b>Primer restore_error_handler()</b><br />';
generar_error();
// Quinta secuencia: restaurar el anterior controlador.
restore_error_handler();
echo '<b>Segundo restore_error_handler()</b><br />';
generar_error();
?>

```

Resultado

Sin controlador

Generar un error

Warning: readfile(/tmp/infos.txt): failed
to open stream: No such file or directory in
/app/scripts/index.php on line 17

Utilizar el controlador nº 1

Generar un error

=> controlador nº 1

Utilizar el controlador nº 2

Generar un error

=> controlador nº 2

Primer restore_error_handler()

Generar un error

=> controlador nº 1

Segundo restore_error_handler()

Generar un error

Warning: readfile(/tmp/infos.txt): failed
to open stream: No such file or directory in
/app/scripts/index.php on line 17

Este ejemplo ilustra la posibilidad de apilar y desapilar los controladores de errores con las funciones `set_error_handler` y `restore_error_handler`.

Si una parte del script necesita de un controlador diferente del utilizado en el resto del script, basta con llamar a `set_error_handler`, al principio de la sección en cuestión para definir el controlador, a continuación, `restore_error_handler`, al final de la sección, para restaurar el anterior.

Por otra parte, el controlador por defecto, es decir, el de PHP, siempre se sustituye al final de la ejecución de un script; por lo que no es necesario llamar a `restore_error_handler` al final del script. Como hemos visto anteriormente, desde la versión 5.5, para restaurar el controlador por defecto durante el script, es posible llamar a la función `set_error_handler` pasándole el valor `NULL` como nombre de función.

trigger_error (o su alias user_error)

La función `trigger_error` permite provocar un error definido por el desarrollador como si este error hubiese sido provocado por PHP de forma nativa.

Sintaxis

```
trigger_error(cadena mensaje, entero nivel)
```

mensaje Mensaje del error.

nivel Nivel del error
entre E_USER_ERROR, E_USER_WARNING y E_USER_NOTICE (icualquier
otro valor genera un error!).

por `trigger_error` se procesa por el controlador interno de PHP (visualización del mensaje), o el posible controlador definido por `set_error_handler`. En el primer caso, el script se detiene si el nivel del error es igual a `E_USER_ERROR` (de lo contrario, continuará). En el segundo caso, el script continúa sea cual sea el nivel del error (si es preciso, el encargado de interrumpir el script es el controlador).

Ejemplo con el controlador interno

```
<?php
// Generar un error E_USER_NOTICE.
trigger_error('*** mi mensaje ***',E_USER_NOTICE);
// Generar un error E_USER_WARNING.
trigger_error('*** mi mensaje ***',E_USER_WARNING);
// Generar un error E_USER_ERROR.
trigger_error('*** mi mensaje ***',E_USER_ERROR);
// Mostrar un mensaje de fin.
echo 'Fin';

?>
```

Resultado

```
Notice: *** mi mensaje *** in /app/scripts/index.php on line 3
Warning: *** mi mensaje *** in /app/scripts/index.php on line 5
Fatal error: *** mi mensaje *** in /app/scripts/index.php on line 7
```

Este ejemplo muestra que los dos primeros errores (niveles `E_USER_NOTICE` y `E_USER_WARNING`) no provocan que se detenga el script, a diferencia de la tercera (nivel `E_USER_ERROR`).

Ejemplo con un controlador externo

```
<?php
// Definir el controlador de error.
function controlador_errores($nivel,$mensaje) {
    // Mostrar simplemente el nivel y el mensaje.
    echo "Nivel = $nivel <br />";
    echo "Mensaje = $mensaje<br />";
    // No interrumpe el script (exit en comentario).
    // exit;
}
// Especificar el controlador que se va a utilizar.
set_error_handler('controlador_errores');
// generar un error E_USER_NOTICE.
trigger_error('*** mi mensaje ***',E_USER_NOTICE);
// generar un error E_USER_WARNING.
trigger_error('*** mi mensaje ***',E_USER_WARNING);
// generar un error E_USER_ERROR.
trigger_error('*** mi mensaje ***',E_USER_ERROR);
// mostrar un mensaje de fin.
echo 'Fin';

?>
```

Resultado

```
$nivel = 1024
$mensaje = *** mi mensaje ***
$nivel = 512
$mensaje = *** mi mensaje ***
$nivel = 256
```



```
$mensaje = *** mi message ***  
Fin
```

error_get_last

La función `error_get_last` devuelve información sobre el último error encontrado en el script. esta función apareció en la versión 5.2.3.

Sintaxis

```
matriz error_get_last()
```

La función `error_get_last` devuelve una matriz asociativa que contiene las claves `tipo`, `mensaje`, `file` y `line`.

Ejemplo

```
<?php  
// Generar un error (sin mostrarlo: @...).  
@readfile('/tmp/infos.txt');  
// Continuar el script.  
echo 'suite ...<br />';  
// Mostrar la información sobre el último error.  
foreach (error_get_last() as $clave => $valor) {  
    echo "$clave => $valor<br />";  
}  
?>
```

Resultado

```
más ...  
tipo => 2  
mensaje => readfile(/tmp/infos.txt):  
failed to open stream: No such file or directory  
file => /app/scripts/index.php  
line => 3
```

La variable \$php_errormsg

Si la directiva de configuración `track_errors` está en `on`, el último mensaje del último error está disponible en la variable `$php_errormsg`.

Si se utiliza un controlador de error personalizado (ver la función `set_error_handler`), `$php_errormsg` se define sólo si la función de control de error devuelve `FALSE`.

Ejemplo

```
<?php  
// Generar un error (sin mostrarlo: @...).  
@readfile('/tmp/infos.txt');  
// Mostrar $php_errormsg.  
echo '$php_errormsg = ', $php_errormsg;  
?>
```

Resultado (si track_errors = on)

```
$php_errormsg = readfile(/tmp/infos.txt):  
failed to open stream: No such file or directory
```

La utilización de esta variable directamente en un script, es un posible sustituto para el establecimiento de un controlador de errores.



Existen numerosas directivas de configuración relacionadas con los errores, además de las presentadas en este capítulo. Para obtener más información, consulte la documentación de PHP.

Variables PHP predefinidas

PHP predefine un gran número de variables relativas a su funcionamiento. Para acceder a esta información, es necesario utilizar matrices asociativas propuestas por PHP.

Las matrices asociativas son las siguientes:

Nombre	Contenido
<code>\$GLOBALS</code>	Matriz asociativa de todas las variables disponibles en el ámbito del script (ver capítulo Funciones y clases)
<code>\$_COOKIE</code>	Matriz asociativa de variables pasadas al script por las cookies (ver capítulo Administrar las sesiones)
<code>\$_GET</code>	Matriz asociativa de variables pasadas al script por un método GET (ver capítulos Gestión de formularios y Administrar las sesiones)
<code>\$_POST</code>	Matriz asociativa de variables pasadas al script por un método POST (ver capítulos Gestión de formularios y Administrar las sesiones)
<code>\$_FILES</code>	Matriz asociativa que contiene información acerca de los archivos cargados desde el ordenador del usuario al servidor Web (ver capítulo Gestión de archivos)
<code>\$_ENV</code>	Matriz asociativa de variables de entorno del sistema operativo pasadas al script (ver capítulo Administrar las sesiones)
<code>\$_SERVER</code>	Matriz asociativa de variables del servidor pasadas al script (variable de entorno y variables del servidor HTTP especialmente)
<code>\$_REQUEST</code>	Matriz asociativa que combina las matrices <code>\$_GET</code> , <code>\$_POST</code> y <code>\$_COOKIE</code> (ver capítulos Gestión de formularios y Administrar las sesiones)
<code>\$_SESSION</code>	Matriz asociativa de datos de sesión accesible en el script (ver capítulo Administrar las sesiones)

Estas matrices asociativas son variables "súper globales": están disponibles en todo el script, incluso dentro de las funciones sin necesidad de declararlas globales (`global $...` no sirve para nada).

Ejemplo: visualización del contenido de `$_SERVER`

```

HTTP_HOST = athena
HTTP_USER_AGENT = Mozilla/5.0 (Macintosh; Intel Mac OS X 10.7; rv:6.0)
Gecko/20100101 Firefox/6.0
HTTP_ACCEPT = text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
HTTP_ACCEPT_LANGUAGE = es-es,en-us;q=0.8,es;q=0.5,en;q=0.3
HTTP_ACCEPT_ENCODING = gzip,deflate
HTTP_ACCEPT_CHARSET = ISO-8859-1,utf-8;q=0.7,*;q=0.7
HTTP_CONNECTION = keep-alive
PATH = /usr/kerberos/sbin:/usr/kerberos/bin:/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin:/root/bin
SERVER_SIGNATURE =
SERVER_SOFTWARE = Apache/2.2.21 (Unix) PHP/5.5.0
SERVER_NAME = athena
SERVER_ADDR = 192.168.76.200
SERVER_PORT = 80

```

```
REMOTE_ADDR = 192.168.76.1
DOCUMENT_ROOT = /usr/local/apache2/htdocs
SERVER_ADMIN = you@example.com
SCRIPT_FILENAME = /app/script/eni/index.php
REMOTE_PORT = 49341
GATEWAY_INTERFACE = CGI/1.1
SERVER_PROTOCOL = HTTP/1.1
REQUEST_METHOD = GET
QUERY_STRING =
REQUEST_URI = /eni/index.php
SCRIPT_NAME = /eni/index.php
PHP_SELF = /eni/index.php
REQUEST_TIME = 1328790616.472
```

Algunas informaciones interesantes aparecen en negrita.

La función `phpinfo` permite mostrar esta información (entre otra).

Constantes PHP predefinidas

PHP predefine un gran número de constantes que incluyen:

Nombre	Contenido
<code>__FILE__</code> *	Nombre del archivo en ejecución. Da el nombre del archivo incluido si se utiliza en un archivo incluido.
<code>__LINE__</code> *	Número de línea en ejecución. Da el número de la línea del archivo incluido si se utiliza en un archivo incluido.
<code>__DIR__</code> *	Carpeta del archivo en ejecución. Equivalente a <code>dirname(__FILE__)</code> . Se incluyó en la versión 5.3.
<code>__NAMESPACE__</code>	Nombre del espacio de nombres actual. Se incluyó en la versión 5.3.
<code>PHP_VERSION</code>	Versión de PHP.
<code>PHP_OS</code>	Sistema operativo del servidor PHP. Ejemplos (no exhaustivos): AIX, Linux, SunOS, WINNT.
<code>TRUE</code> *	Valor booleano verdadero (TRUE).
<code>FALSE</code> *	Valor booleano falso (FALSE).
<code>E_*</code>	Códigos de error (ver capítulo Administrar los errores en un script PHP).
<code>DIRECTORY_SEPARATOR</code>	Carácter separador utilizado en los nombres de directorio para la plataforma en la que está instalado PHP.
<code>PHP_EOL</code>	Secuencia de caracteres utilizada por la plataforma para representar una nueva línea. Se incluyó en la versión 5.0.2.
<code>PHP_INT_MAX</code>	Valor del entero mayor. Se incluyó en la versión 5.0.5.
<code>PHP_INT_SIZE</code>	Tamaño de los enteros (número de bytes). Se incluyó en la versión 5.0.5.

Las constantes seguidas por un asterisco son sensibles a mayúsculas y minúsculas (se pueden utilizar indistintamente en mayúsculas o minúsculas).

Ejemplos adicionales

1. Introducción

La potencia del lenguaje PHP se ve reforzada por la existencia de un gran número de bibliotecas que amplían las características del lenguaje: corrección ortográfica, generación de documentos PDF (*Portable Document Format*), manipulación de documentos XML (*eXtensible Markup Language*), acceso a servidores FTP (*File Transfer Protocol*), acceso a servidores IMAP (*Internet Message Access Protocol*), acceso a directorios LDAP (*Lightweight Directory Access Protocol*), cifrado, generación de documentos de Shockwave Flash, gestión del protocolo SNMP (*Simple Network Management Protocol*), compresión, etc. Algunas bibliotecas requieren bibliotecas adicionales. Estas bibliotecas dan respuesta a diferentes necesidades especiales y no se describen en detalle en este libro.

En esta parte del anexo, vamos a dar tres ejemplos comentados de uso de estas bibliotecas, correspondientes a tres necesidades frecuentes:

- Leer un documento XML.
- Generar un documento PDF.
- Generar una imagen.

Para obtener más información sobre las bibliotecas disponibles o sobre una función, consulte la ayuda en línea disponible en la página Web oficial de PHP (www.php.net/manual/es/).

2. Leer un documento XML

Este ejemplo ilustra las posibilidades de la extensión SimpleXML, incluida en la versión 5.

Para este ejemplo, suponemos que hay una lista de artículos almacenada en un archivo llamado `articulos.xml`:

```
<?xml version='1.0' encoding='UTF-8'?>
<articulos>
  <articulo código="A1" color="amarillo">
    <identificador>1</identificador>
    <texto>Albaricoques</texto>
    <precio>35.5</precio>
  </articulo>
  <articulo código="A2" color="rojo">
    <identificador>2</identificador>
    <texto>Cerezas</texto>
    <precio>48.9</precio>
  </articulo>
  <articulo código="A3" color="rojo">
    <identificador>3</identificador>
    <texto>Fresas</texto>
    <precio>29.95</precio>
  </articulo>
  <articulo código="A4" color="amarillo">
    <identificador>4</identificador>
    <texto>Melocotones</texto>
    <precio>37.2</precio>
  </articulo>
</articulos>
```

Código

```
<?php echo '<?xml version="1.0" encoding="UTF-8"?>','\n'; ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Leer un documento XML</title>
  </head>
  <body>
    <div>

    <?php
    // Cargar el documento XML = simplexml_load_file() :
    // - devuelve un objeto de la clase simplexml_element
    //   o FALSE en caso de error (documento XML mal formado, por ejemplo).
    $xml = simplexml_load_file('articulos.xml');
    if (! $xml) { exit; }

    // El objeto $xml tiene una estructura que corresponde a la
    // estructura de nuestro documento:
    // - artículo (matriz de objetos):
    //   - identificador ;
    //   - texto ;
    //   - precio.
    // Ruta del nudo artículo (matriz).
    echo "<b>Ruta de \$xml->artículo</b><br />\n";
    foreach ($xml->artículo as $artículo) {
      printf("%s,%s,%s,%s,%s<br />\n",
        $artículo->identificador,
        $artículo->texto,
        $artículo->precio,
        $artículo['código'],
        $artículo['color']);
    }
  </div>
</body>
</html>
```

```
// Acceso a una información específica.
echo "<b>Acceso a una información específica</b><br />\n";
printf("Antes - Precio de %s = %s (código = %s)<br />\n",
    $xml->artículo[2]->texto,
    $xml->artículo[2]->precio,
    $xml->artículo[2]['código']);
$xml->artículo[2]->precio = 123;
$xml->artículo[2]['código'] .= '+';
printf("Después - Precio de %s = %s (código = %s)<br />\n",
    $xml->artículo[2]->texto,
    $xml->artículo[2]->precio,
    $xml->artículo[2]['código']);

// Extracción de los atributos de un nodo = método attributes() :
// - devuelve un objeto de la clase simplexml_element ;
// - en nuestro ejemplo, recuperación de los atributos del 1er artículo.
$atributos = $xml->artículo[0]->attributes();

// Ruta de los atributos así recuperados.
echo "<b>Atributos del primer artículo</b><br />\n";
foreach($atributos as $nombre => $valor) {
    printf("%s = %s<br />\n", $nombre, $valor);
}

// Extraer los hijos de un nodo = método children():
// - devuelve un objeto de la clase simplexml_element.
echo "<b>Ruta del árbol</b><br />\n";
echo "raíz<br />\n";
foreach ($xml->children() as $nombrel => $nivel1) {
    printf("----%s (%s,%s)<br />\n",
        $nombrel, $nivel1['código'], $nivel1['color']);
    foreach ($nivel1->children() as $nombred => $nivel2) {
        printf("-----%s = %s<br />\n", $nombred, $nivel2);
    }
}

// Efectuar una búsqueda Xpath = método xpath():
// - devuelve una matriz de objetos de la clase simplexml_element.
echo "<b>Búsqueda Xpath: /articulos/artículo</b><br />\n";
$resultado = $xml->xpath("/articulos/artículo");
foreach ($resultado as $valor) {
    printf("%s,%s<br />\n", $valor->identificador, $valor->texto);
}
echo "<b>Búsqueda Xpath: artículo/texto</b><br />\n";
$resultado = $xml->xpath("artículo/texto");
foreach ($resultado as $valor) {
    printf("%s<br />\n", $valor);
}
echo "<b>Búsqueda Xpath: //precio</b><br />\n";
$resultado = $xml->xpath("//precio");
foreach ($resultado as $valor) {
    printf("%s<br />\n", $valor);
}

// Generar una cadena XML = método asXML().
echo "<b>Cadena XML</b><br />\n";
file_put_contents ('los_articulos.xml', $xml->asXML());
file_put_contents ('un_articulo.xml', $xml->artículo[0]->asXML());
echo "Ver los archivos 'los_articulos.xml' y 'un_articulo.xml'<br />\n";

?>

</div>
</body>
</html>
```

Resultado en pantalla

```
Ruta de $xml->artículo
1,Albaricoques,35.5,A1,amarillo
2,Cerezas,48.9,A2,rojo
3,Fresas,29.95,A3,rojo
4,Melocotones,37.2,A4,amarillo
Acceso a una información específica
Antes - precio de Fresas = 29.95 (código = A3)
Después - precio de Fresas = 123 (código = A3+)
Atributos del primer artículo
código = A1
color = amarillo
Ruta del árbol
raíz
----artículo (A1,amarillo)
-----identificador = 1
-----texto = Albaricoques
-----precio = 35.5
----artículo (A2,rojo)
-----identificador = 2
-----texto = Cerezas
-----precio = 48.9
----artículo (A3+,rojo)
-----identificador = 3
```

```

-----texto = Fresas
-----precio = 123
----artículo (A4,amarillo)
-----identificador = 4
-----texto = Melocotones
-----precio = 37.2
Búsqueda Xpath: /articulos/articulo
1,Albaricoques
2,Cerezas
3,Fresas
4,Melocotones
Búsqueda Xpath: articulo/texto
Albaricoques
Cerezas
Fresas
Melocotones
Búsqueda Xpath: //precio
35.5
48.9
123
37.2
Cadena XML
Ver los archivos 'los_articulos.xml' y 'un_articulo.xml'

```

Contenido del archivo los_articulos.xml

```

<?xml version="1.0" encoding="UTF-8"?>
<articulos>
  <articulo código="A1" color="amarillo">
    <identificador>1</identificador>
    <texto>Albaricoques</texto>
    <precio>35.5</precio>
  </articulo>
  <articulo código="A2" color="rojo">
    <identificador>2</identificador>
    <texto>Cerezas</texto>
    <precio>48.9</precio>
  </articulo>
  <articulo código="A3" color="rojo">
    <identificador>3</identificador>
    <texto>Fresas</texto>
    <precio>123</precio>
  </articulo>
  <articulo código="A4" color="amarillo">
    <identificador>4</identificador>
    <texto>Melocotones</texto>
    <precio>37.2</precio>
  </articulo>
</articulos>

```

Contenido del archivo un_artículo.xml

```

<articulo código="A1" color="amarillo">
  <identificador>1</identificador>
  <texto>Albaricoques</texto>
  <precio>35.5</precio>
</articulo>

```

3. Generar un documento PDF

La extensión PDF de PHP permite generar documentos PDF. Esta extensión utiliza la biblioteca PDFlib que requiere una licencia (<http://www.pdflib.com/products/pdflib-family/>).

Existen varias alternativas gratuitas para generar documentos PDF, como la biblioteca FPDF (<http://www.fpdf.org/>). Esta biblioteca no se instala por defecto con PHP, pero está presente en el paquete XAMPP. Esta es la biblioteca que usaremos en este ejemplo.

El código siguiente genera el siguiente documento PDF:

Lista de artículos	
Texto	Precio
Albaricoques	35,50
Cerezas	48,90
Fresas	29,95
Melocotones	37,20

La lista de artículos se lee en el archivo XML `articulos.xml` utilizado en la sección Leer un documento XML.

Código

```
<?php

// Inclusión de la biblioteca.
include ('fpdf.php');

// Cargar la lista de artículos desde el documento XML.
$xml = simplexml_load_file('articulos.xml');
if (! $xml) {
    exit('Error al cargar la lista de artículos.');
```

```
}
// Crear un nuevo documento PDF = new FPDF() :
// - primer parámetro = orientación
//   > P = retrato - L = Paisaje
// - segundo parámetro = unidad de medida
//   > pt = punto - mm = milímetro - cm = centímetro
// - tercer parámetro = formato (A3, A4, etc.)
// Todos los parámetros son opcionales. Predeterminado = P, mm, A4.
$pdf = new FPDF('P','mm','A4');
```

```
// Definir los saltos de página automáticos = SetAutoPageBreak():
// - primer parámetro = automático (true/false)
//   > P = retrato - L = Paisaje
// - segundo parámetro = margen.
//   > distancia en relación al final de la página que desencadena
//     el salto (2 cm predeterminado, si activo)
$pdf->SetAutoPageBreak(false);
```

```
// Crear una nueva página en el documento = AddPage():
// - primer parámetro = orientación.
//   > P = retrato - L = Paisaje
// Por defecto, el del documento.
$pdf->AddPage();
```

```
// Definir la información de resumen del documento = SetTitle(),
// SetAuthor(), SetSubject().
$pdf->SetTitle('Lista de artículos');
$pdf->SetAuthor('Olivier HEURTEL');
$pdf->SetSubject('Frutas');
```

```
// Definir la fuente utilizada = SetFont():
// - primer parámetro = familia;
//   > nombre de una familia estándar (Courier, Helvetica o Arial,
```

```
// Times, Symbol, ZapfDingbats) o de un nombre definido por
// AddFont().
// - segundo parámetro (opcional) = estilo;
// > combinación de: B = negrita - I = cursiva - U = subrayado
// - tercer parámetro (opcional) = tamaño en puntos.
// Ver también el método SetFontSize() para modificar el tamaño.
$pdf->SetFont('Arial','B',16);

// Escribir texto desde la posición actual = Write():
// - primer parámetro = altura de la línea;
// - segundo parámetro = texto a escribir.
// Utiliza las características actuales de fuente, colores, etc.
// El retorno de línea es automático cuando se llega al final del
// margen derecho (o cuando se encuentra el carácter \n).
$pdf->Write(5, 'Lista de artículos');

// Efectuar un salto de línea = Ln():
// - primer parámetro (opcional) = altura de la línea.
// La abscisa toma el valor del margen izquierdo.
$pdf->Ln(10);

// Cambiar el tamaño de fuente = SetFontSize() :
// - primer parámetro = fuente en puntos.
$pdf->SetFontSize(12);

// Definir el color que se utilizará para el texto = SetTextColor():
// - si un solo parámetro = nivel de gris (entre 0 y 255);
// - si 3 parámetros = componentes RGB (entre 0 y 255).
$pdf->SetTextColor(255,0,0); // rojo

// Definir el color que se utilizará para el fondo = SetFillColor():
// - si un solo parámetro = nivel de gris (entre 0 y 255);
// - si 3 parámetros = componentes RGB (entre 0 y 255).
$pdf->SetFillColor(255,255,140); // amarillo claro

// Escribir una celda = Cell():
// - primer parámetro = longitud (0 = hasta el margen derecho) ;
// - segundo parámetro = altura;
// - tercer parámetro = texto a escribir;
// - cuarto parámetro = borde;
// > es un número: 0 = ningún borde - 1 = marco
// > es una cadena: combinación de L (izquierda), T (alto),
// R (derecha), B (bajo)
// - quinto parámetro = posición final;
// > 0 = a la derecha - 1 = inicio de la línea siguiente - 2 = debajo
// - sexto parámetro = alineación;
// > L o cadena vacía = a la izquierda - C = centrado - R = a la derecha
// - séptimo parámetro = relleno.
// > 0 = no - 1 = sí
// solo el primer parámetro es obligatorio.
$pdf->Cell(80,7,'Texto',1,0,'C',1); // título de la columna
$pdf->Cell(40,7,'Precio',1,1,'C',1); // título de la columna

// Cambiar el color y la fuente.
$pdf->SetFont('Arial','',12); // '' = normal
$pdf->SetTextColor(0,0,0); // negro

// En un bucle, escribir los datos de la matriz.
foreach ($xml->artículo as $artículo) { // ruta del documento XML
    $precio = number_format((float)$artículo->precio,2,',',' ');
    $pdf->Cell(80,7,$artículo->texto,1);
    $pdf->Cell(40,7,$precio,1,1,'R'); // línea siguiente + a la derecha
}

// Colocarse en un lugar específico de la página = SetXY():
// - primer parámetro = abscisa (x);
// - segundo parámetro = ordenada (y).
// El origen es la esquina superior izquierda.
// Si los valores son negativos, el origen es la esquina
// inferior derecha.
// Ver también SetX() y SetY().
$pdf->SetXY(10,-10); // 1 cm a la izquierda, 1 cm del final

// Mostrar el número de página = PageNo().
$pdf->SetFontSize(10);
$pdf->Cell(0,0,'Página '. $pdf->PageNo(),0,0,'R');

// Mostrar una imagen = Image():
// - primer parámetro = nombre del archivo;
// - segundo parámetro = abscisa de la esquina superior izquierda;
// - tercer parámetro = ordenada de la esquina superior izquierda;
// - cuarto parámetro (opcional) = longitud de la imagen;
// > 0 o nada = calculado automáticamente
// - quinto parámetro (opcional) = altura de la imagen;
// > 0 o nada = calculado automáticamente
// - sexto parámetro (opcional) = tipo.
// > JPG o JPEG o PNG
// > Deducido de la extensión si ausente
$pdf->Image('../imágenes/logo.jpg',10,285,20);
```

```
// Definir el color a utilizar para el dibujo = SetDrawColor():
// - si un solo parámetro = nivel de gris (entre 0 y 255);
// - si 3 parámetros = componentes RGB (entre 0 y 255).
$pdf->SetDrawColor(128); // nivel de gris
$pdf->line(10,15,200,15); // línea horizontal superior
$pdf->line(10,285-2,200,285-2); // línea horizontal inferior

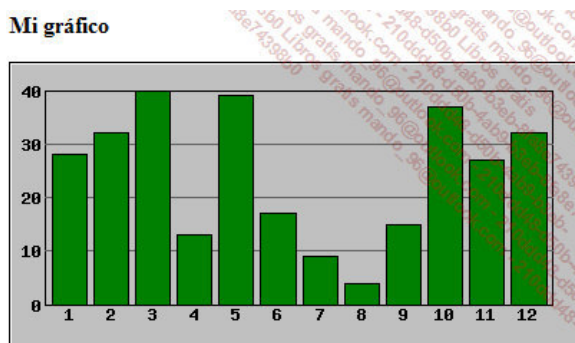
// Enviar el documento a un destino = Output():
// - primer parámetro (opcional) = nombre del archivo;
// - segundo parámetro (opcional) = tipo de destino.
//   > F = archivo en el servidor
//   I = navegador (en línea)
//   D = navegador (descarga)
// Si no hay parámetro: destino = I.
// Si se especifica un nombre: destino predeterminado = F.
$pdf->Output(); // navegador (en línea)

?>
```

4. Generar una imagen

La extensión GD de PHP permite crear y manipular imágenes.

El código siguiente permite generar la siguiente página:



La generación del gráfico se realiza de manera dinámica en la llamada a la página; para este ejemplo, los datos del gráfico se calculan de manera aleatoria.

Código

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Mi gráfico</title>
  </head>
  <body>
    <div>
      <b>Mi gráfico</b>
      <p>
        
      </p>
    </div>
  </body>
</html>
```

Observe que la fuente de la imagen es un script PHP que crea la imagen de forma dinámica cuando se muestra la página. Esta técnica también puede utilizarse para mostrar en una página una imagen que está almacenada en una base de datos.

Script mi-grafico.php

```
<?php

// Definir un encabezado que indique que se trata de una imagen (aquí PNG).
header('Content-type: image/png');

/*

Las funciones definidas en el script utilizan dos variables globales:
- $imagen = recurso de imagen en curso de creación;
- $altura_imagen = altura de la imagen.

Para las coordenadas, el origen es la esquina superior izquierda.
Para el dibujo de nuestro gráfico, utilizamos un origen en la esquina inferior izquierda (más práctico). Las funciones efectúan la conversión.

*/

// Función de dibujo de un rectángulo
// - $x1,$y1 = punto 1
// - $x2,$y2 = punto 2
// - $borde, $fondo = colores del borde y del fondo
```

```
//
function rectángulo($x1,$y1,$x2,$y2,$borde,$fondo) {

    global $imagen;
    global $altura_imagen;

    // Conversión del sistema de coordenadas para el eje y.
    $y1 = $altura_imagen - 1 - $y1;
    $y2 = $altura_imagen - 1 - $y2;

    // Dibujar el borde de un rectángulo = imagenrectángulo.
    imagenrectángulo($imageb,$x1,$y1,$x2,$y2,$borde);

    // Relleno (si es necesario, por ejemplo $fondo informado).
    if ( ( $fondo != NULL) and ($x1 != $x2) and ($y1 != $y2) ) {

        // Rellenar un rectángulo = imagenhijaderectángulo.
        imagenhijaderectángulo($imageb,$x1+1,$y1-1,$x2-1,$y2+1,$fondo);
    }
}

// Función de dibujo de una línea:
// - $x1,$y1 = punto 1;
// - $x2,$y2 = punto 2;
// - $color = color.
//
function línea($x1,$y1,$x2,$y2,$color) {

    global $imagen;
    global $altura_imagen;

    // Conversión del sistema de coordenadas para el eje y.
    $y1 = $altura_imagen - 1 - $y1;
    $y2 = $altura_imagen - 1 - $y2;

    // Dibujar una línea = imageline.
    imageline($imagen,$x1,$y1,$x2,$y2,$color);
}

// Función de dibujo de un texto:
// - $fuente = código de la fuente predefinida (1 a 5)
// - $x,$y = punto de referencia
// - $texto = texto
// - $color = color
// - $horizontal = alineación horizontal con respecto al punto
// de referencia
// > D = alineado a la derecha, C = centrado, G = alineado a la izquierda
// - $vertical = alineación vertical con respecto al punto de referencia
// > H = texto superior, C = centrado, B = texto inferior
//
function texto($fuente,$x,$y,$texto,$color,$horizontal,$vertical) {

    global $imagen;
    global $altura_imagen;

    // Conversión del sistema de coordenadas para el eje y.
    $y = $altura_imagen - 1 - $y;

    // Calcular la longitud de un carácter en una fuente =
    // imagefontwidth.
    $longitud = imagefontwidth($fuente) * strlen($texto);

    // Calcular la altura de un carácter en una fuente =
    // imagefontheight.
    $altura = imagefontheight($fuente);

    // Cálculo de coordenadas en función de la alineación.
    switch ($horizontal) {
        case 'D':
            $x = $x - $longitud;
            break;
        case 'C':
            $x = $x - floor($longitud/2);
            break;
        case 'G':
            break;
    }
    switch ($vertical) {
        case 'H':
            $y = $y - $altura;
            break;
        case 'C':
            $y = $y - floor($altura/2);
            break;
        case 'B':
            break;
    }
}
```

```

// Dibujar un texto = imagestring.
imagestring($imagen,$fuente,$x,$y,$texto,$color);
}

// Para este ejemplo, los datos del gráfico se calculan de
// forma aleatoria:
// - unidad, valor mínimo y valor máximo para el eje y;
$eje_y_unidad = 10;
$eje_y_min = 0;
$eje_y_max = 40;
// - número de barras: entre 5 y 15;
$num_barras = rand(5,15);
// - poner los datos en la matriz.
$datos = array();
for ($i = 1 ; $i <= $num_barras ; $i++) {
    $datos[$i] = rand($eje_y_min,$eje_y_max);
}

// Dimensiones del dibujo (píxeles).
$longitud_imagen = 400; // longitud de la imagen
$altura_imagen = 200; // altura de la imagen
$margen_blanco = 2; // margen blanco
$margen_izquierdo = 25; // margen izquierdo con la zona de trazado
$margen_derecho = 20; // margen derecho con la zona de trazado
$margen_superior = 20; // margen superior con la zona de trazado
$margen_inferior = 30; // margen inferior con la zona de trazado
$separación_barras = 5; // separación entre las barras

// Deducir la longitud y la altura de la zona de trazado.
$longitud_trazado = $longitud_imagen - $margen_derecho - $margen_izquierdo;
$altura_trazado = $altura_imagen - $margen_superior - $margen_inferior;

// Deducir la longitud de las barras y la escala del eje y.
$longitud_barra = ($longitud_trazado-$separación_barras)/$num_barras;
$separación_barras;
$escala_eje_y = ($altura_trazado-1) / $eje_y_max;

// Crear la imagen = imagecreatetruecolor.
$imagen = imagecreatetruecolor($longitud_imagen,$altura_imagen);

// Definir los colores en RGB = imagecolorallocate.
$blanco = imagecolorallocate($imagen, 255, 255, 255);
$negro = imagecolorallocate($imagen, 0, 0, 0);
$gris_claro = imagecolorallocate($imagen, 192, 192, 192);
$gris_oscura = imagecolorallocate($imagen, 100, 100, 100);
$verde = imagecolorallocate($imagen, 0, 128, 0);

// Coordenadas de la imagen.
$ox1 = 0; $oy1 = 0;
$ox2 = $longitud_imagen - 1; $oy2 = $altura_imagen - 1;
// Dibujar el marco exterior.
rectángulo($ox1,$oy1,$ox2,$oy2,$negro,$blanco);

// Coordenadas de la imagen menos el borde blanco.
$mx1 = $ox1 + $margen_blanco; $my1 = $oy1 + $margen_blanco;
$mx2 = $ox2 - $margen_blanco; $my2 = $oy2 - $margen_blanco;
// Dibujar el fondo gris claro.
rectángulo($mx1,$my1,$mx2,$my2,$gris_claro,$gris_claro);

// Coordenadas de la zona de trazado.
$tx1 = $ox1 + $margen_izquierdo; $ty1 = $oy1 + $margen_inferior;
$tx2 = $ox2 - $margen_derecho; $ty2 = $oy2 - $margen_superior;
// Dibujar el marco de la zona de trazado.
rectángulo($tx1,$ty1,$tx2,$ty2,$negro,NULL);

// Dibujo de líneas horizontales con su etiqueta.
$x1 = $tx1;
$x2 = $tx2;
for ($eje = $eje_y_min ; $eje <= $eje_y_max ; $eje += $eje_y_unidad) {
    $y1 = $ty1 + $eje * $escala_eje_y;
    $y2 = $y1;
    // No dibujar la línea abajo y arriba.
    if ( ( $eje > $eje_y_min ) and ( $eje < $eje_y_max ) ) {
        línea($x1,$y1,$x2,$y2,$gris_oscura);
    }
    texto(3,$x1-2,$y1,$eje,$negro,"D","C");
}

// Dibujo de las barras.
$i = 0;
foreach($datos as $clave => $valor) {
    $i++;
    $x1 = $tx1 + $separación_barras + ($i-1)*($separación_barras+$longitud_barra);
    $x2 = $x1 + $longitud_barra;
    $y1 = $ty1;
    $y2 = $y1 + $valor * $escala_eje_y;
    rectángulo($x1,$y1,$x2,$y2,$negro,$verde);
    texto(3,($x1+$x2)/2,$y1,$clave,$negro,"C","B");
}

// Generar la imagen en formato PNG = imagepng:

```

```
// - ya sea en la pantalla (sin segundo parámetro);  
// - ya sea en un archivo (segundo parámetro = nombre del archivo).  
// Existen funciones similares para otros formatos.  
imagepng($imagen /*, 'imagen.png' */);  
  
// Eliminar la imagen = imagedestroy.  
imagedestroy($imagen);  
  
?>
```

Estructura básica de una página PHP

1. Las etiquetas PHP

Como hemos visto anteriormente, el código PHP se incluye en una página HTML dentro de las etiquetas (también conocidas como por su término en inglés, "tags").

PHP acepta cuatro sintaxis para las etiquetas:

- `<?php ... ?>`
- `<script language="php"> ... </script>`
- `<? ... ?>`
- `<% ... %>`

La primera es la sintaxis habitual y la más recomendada.

La segunda sintaxis, más pesada, utiliza la etiqueta estándar `script`; puede ser útil si su editor de HTML interpreta de manera incorrecta el resto de sintaxis.

La sintaxis de la tercera sólo es posible si está permitida en el archivo de configuración de PHP (`php.ini`) poniendo el parámetro `short_open_tag` en `on`. No es aconsejable utilizar esta sintaxis si el código debe desplegarse en un servidor cuya configuración no puede modificar y que no es compatible con esta sintaxis.

La cuarta sintaxis permite utilizar la etiqueta ASP, pero sólo es factible si está permitido en el archivo de configuración de PHP estableciendo el parámetro `asp_tags` en `on`.

2. La función echo

La función `echo` es la función básica de cualquier página PHP. Permite mostrar una o varias cadenas y, por tanto, incluir texto en la página HTML que se envía al explorador.

Sintaxis

```
echo(cadena de texto)  
echo cadena de texto[, ...]
```

`texto`: texto que se mostrará.

La primera sintaxis únicamente acepta un parámetro, mientras que la segunda acepta varios.

Ejemplo

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"  
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">  
<html xmlns="http://www.w3.org/1999/xhtml">  
  <head>  
    <title>Ejemplo de página PHP</title>  
  </head>  
  <body>  
    <p>  
      <?php  
        echo('¡Hola Olivier!');  
    </p>  
  </body>  
</html>
```

```
?>
<br />
<?php
    echo '¡Hola ', 'Valeria', '!';
?>
</p>
</body>
</html>
```

Resultado

```
¡Hola Olivier!
¡Hola Valeria!
```

No hay salto de línea automático en el resultado de la ejecución del código PHP. Si es necesario, se debe insertar la etiqueta HTML `
` que provoca un salto de línea en la página HTML final (véase el ejemplo anterior).

El texto parametrado en la función `echo` se puede escribir en varias líneas en la fuente, pero se muestra sólo en una en el resultado.

Ejemplo

```
<?php
echo '¡Hola
Olivier', '!';
?>
```

Resultado

```
¡Hola Olivier!
```

3. El separador de instrucciones

En PHP, todas las instrucciones deben terminar con un punto y coma.

Ejemplo

```
<?php
echo '¡Hola ';
echo 'Olivier!';
?>
```

Resultado

```
¡Hola Olivier!
```

Si se omite el punto y coma, se genera un error.

La única excepción es la instrucción que precede a la etiqueta de cierre para el que se puede omitir el punto y coma.

Es posible escribir varias instrucciones en la misma línea siempre y cuando estén separadas por un punto y coma. Sin embargo, a veces esta escritura dificulta la legibilidad del código.

Ejemplo


```
<?php
echo '¡Hola ' ; echo 'Olivier!';
?>
```

Resultado

```
¡Hola Olivier!
```

4. El comentario

PHP ofrece dos sintaxis:


- // o # para insertar comentarios en una línea "dedicada" o después de una instrucción.
- /* ... */ para insertar comentarios en varias líneas.

Ejemplo

```
<html>
<?php
// comentario en una sola línea
# comentario en una sola línea
/* comentario en
varias líneas */
echo '¡Hola'; // comentario hasta el final de la línea
echo 'Olivier!'; # comentario hasta el final de la línea
?>
</html>
```

Resultado

```
¡Hola Olivier!
```

 Los comentarios /*... */ no deben ser anidados.

5. Mezclar PHP y HTML

Existen muchos enfoques para mezclar PHP y HTML.

Sin embargo, estos enfoques se basan en dos principios simples:

- La página puede contener una o varias inclusiones de código PHP.
- El código PHP genera el "texto" que se integra en la página HTML que se envía al explorador. Por lo tanto, cualquier "texto" comprensible por el navegador puede generarse por código PHP: texto simple, código HTML, código JavaScript...

Los siguientes ejemplos utilizan variables y funciones de PHP (recuperación de la fecha y la hora). Estos conceptos se discuten en mayor detalle en el capítulo Variables, constantes, tipos y matrices.

Ejemplo de página que contiene código PHP en varios lugares

```
<?php
// Declaración de variables que se utilizarán más adelante.
```

```
// Esta sección de código PHP no genera una salida en la
// página HTML (no hay ninguna llamada a echo).
$nombre = 'Olivier'; // nombre del usuario
$titulo_pagina = 'Ediciones ENI presenta...'; // título de la página
$hoy = date("d/m/Y"); // fecha del día
$hora = date("H:i:s"); // hora
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>
<?php /* muestra el título */ echo $titulo_pagina; ?>
</title>
</head>
<body>
<p>    <?php
/* Muestra el nombre del usuario.
** Las etiquetas de negrita del nombre (<b>) y del salto de
** línea (<br />) están incluidas en la cadena enviada
** por echo.
*/
echo "¡Hola <b>$nombre</b>!<br />";
// Muestra la fecha y la hora.
echo "Hoy estamos a $hoy; son las $hora.";
?>
</p>
</body>
</html>
```

Resultado

```
¡Hola Olivier!
Hoy estamos a 24/06/2013; son las 08:17:48.
```

Fuente de la página en el navegador (los elementos generados por PHP están en negrita)

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>
Ediciones ENI presenta... </title>
</head>
<body>
<p>
    ¡Hola <b>Olivier</b>!<br />Hoy estamos a 24/06/2013;
son las 08:17:48.
</p>
</body>
</html>
```

Ejemplo de página generada en su totalidad por código PHP (siguiendo el principio CGI)

```
<?php
// Declaración de variables que se utilizarán más adelante.
$nombre = 'Olivier'; // nombre del usuario
$titulo_pagina = 'Ediciones ENI presenta...'; // título de la página
```

```

$hoy = date("d/m/Y"); // fecha del día
$hora = date("H:i:s"); // hora
// Generación de las etiquetas de apertura del documento HTML.
echo '<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" ',
    '"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">';
echo '<html xmlns="http://www.w3.org/1999/xhtml">';
echo '<head>';
echo "<title>$titulo_pagina</title>";
echo '</head>';
echo '<body>';
echo '<p>';
/* Muestra el nombre del usuario.
** Las etiquetas de negrita del nombre (<b>) y del salto de línea
** (<br />) están incluidas en la cadena enviada por echo.
*/
echo ";Hola <b>$nombre</b>!<br />";
// Muestra la fecha y la hora.
echo "Hoy estamos a $hoy; son las $hora.";
echo '</p>';
echo '</body>';
echo '</html>';
?>

```

Resultado

```

;Hola Olivier!
Hoy estamos a 24/06/2013; son las 08:19:42.

```

Código fuente de la página en el navegador (todo está en una línea)

```

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd"><html
xmlns="http://www.w3.org/1999/xhtml"><head><title>
Ediciones ENI presenta...</title></head><body><p>;Hola
<b>Olivier</b>!<br />Hoy estamos a 24/06/2013; son las
08:19:42.</p></body></html>

```

No existe ninguna regla para mezclar PHP y HTML. Un enfoque comúnmente utilizado por los desarrolladores es utilizar PHP únicamente para generar la parte verdaderamente dinámica de la página; el resto se escribe directamente en HTML en el archivo. Esta técnica hace que el código sea menos pesado y permite ver inmediatamente dónde se encuentra la lógica de la aplicación.

El documento HTML que se envía al navegador debe ser válido y, si es posible, debe ser conforme a los estándares HTML o XHTML del W3C (*World Wide Web Consortium*). Si es necesario, utilice el servicio de validación del W3C (<http://validator.w3.org/>).

En este libro, los ejemplos respetan al máximo la recomendación XHTML 1.0. Sin embargo, por razones de espacio, los ejemplos no integran sistemáticamente las declaraciones iniciales:

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
...

```

6. Normas de denominación

Cualquier entidad PHP con nombre (variable, constante, función...) debe tener un nombre que respete las siguientes reglas:

- empezar con una letra o guión bajo (_);
- a continuación, debe contener letras, números o guiones bajos.

En esta definición, una letra representa cualquier letra mayúscula o minúscula entre A y Z (a-z y A-Z) y todos los caracteres ASCII entre 127 y 255. Por lo tanto, los caracteres acentuados están permitidos, pero no los caracteres del tipo # \$ % & que tienen un significado especial en el lenguaje PHP.

Configuración de PHP

1. El archivo de configuración php.ini

A lo largo de este libro nos encontramos con varias directivas de configuración que se utilizan para modificar el comportamiento de PHP.

Estas directivas de configuración se introducen en el archivo de configuración de PHP (`php.ini`).

PHP dispone de dos archivos `php.ini` de muestra: `php.ini-development` y `php.ini-production`.

El archivo `php.ini-development` es un ejemplo de un archivo de configuración especialmente destinado a su uso en un entorno de desarrollo. Por el contrario, el archivo `php.ini-production` está especialmente destinado a su uso en un entorno operativo y contiene la configuración de PHP más segura y/o de mejor rendimiento.

Ambos archivos presentan multitud de comentarios que explican el papel de cada directiva y ofrecen consejos sobre su uso.

Para utilizar uno de estos archivos, cópielo a la ubicación apropiada de su plataforma y cámbiele el nombre a `php.ini`. El archivo `php.ini` se encuentra especialmente en los siguientes lugares (en este orden):

- una ubicación específica en el servidor Web (por ejemplo, la directiva `PHPIniDir` de Apache 2).
- un lugar definido por la variable de entorno `PHPRC`.
- la carpeta `/usr/local/lib` en Linux/Unix y `c:\windows` o `c:\winnt` en Windows.

Con Apache, también es posible definir directivas de configuración de PHP en el archivo de configuración de Apache (por ejemplo `httpd.conf`) o en un archivo `htaccess`.

Por otra parte, desde la versión 5.3, PHP permite definir directivas de configuración por directorio, pero sólo en modo CGI o FastCGI. Para ello hay disponible dos métodos:

- Crear un archivo `.user.ini` en el directorio con los valores deseados.
- Definir una o varias secciones `[PATH=/ruta/al/directorio]` en el archivo `php.ini`.

Para obtener más información sobre la aplicación y las limitaciones de estas características, consulte la documentación de PHP.

En este libro, y a menos que se indique lo contrario, asumiremos que se han colocado dos directivas relativas a la gestión de errores de la siguiente manera:

```
display_errors = on
```

Se mostrarán los errores.

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

Se mostrarán todos los errores, a excepción de los errores de nivel `E_NOTICE` (informaciones simples, por ejemplo cuando se utiliza una variable no inicializada).

```
date.timezone = "Europe/Paris"
```

Huso horario por defecto.

2. Información sobre la configuración

PHP dispone de dos funciones particularmente útiles para obtener información sobre la configuración: `phpversion` y `phpinfo`.

La función `phpversion` devuelve el número de versión de PHP y la función `phpinfo` muestra una gran cantidad de información sobre la configuración de PHP y su entorno. El número de versión también está disponible a través de la constante predefinida `PHP_VERSION`.

Sintaxis

```
cadena phpversion([cadena extensión])  
booleano phpinfo([entero qué])
```

Con

extensión

Si se especifica este parámetro, la función devuelve la versión de la extensión con este nombre (o `FALSE` si se desconoce la versión o si la extensión no está disponible).

qué

Naturaleza de la información deseada. Utilizar una o varias (total) de las siguientes constantes:

`INFO_GENERAL` (1) : información general (versión, ubicación del archivo `php.ini`, sistema operativo, etc.).

`INFO_CREDITS` (2) : información sobre los autores.

`INFO_CONFIGURATION` (4) : información sobre la configuración (valores de directivas).

`INFO_MODULES` (8) : información sobre los módulos cargados con su configuración respectiva.

`INFO_ENVIRONMENT` (16) : información sobre el entorno (ver la variable `$_ENV` en anexo).

`INFO_VARIABLES` (32) : valores de todas las variables predefinidas (ver anexo).

`INFO_LICENSE` (64) : información de licencia.

`INFO_ALL` (-1) : toda la información (valor predeterminado).

`phpinfo` devuelve `TRUE` en caso de éxito y `FALSE` en caso de error.

Ejemplo 1

```
<?php  
echo phpversion();
```

?>

Resultado

5.5.0

Ejemplo 2

```
<?php
// información general e información
// de licencia
phpinfo(INFO_GENERAL+INFO_LICENCIA);
?>
```

Resultado

PHP Version 5.5.0



System	Linux athena.local 2.6.32-100.26.2.el5 #1 SMP Tue Jan 18 20:11:49 EST 2011 x86_64
Build Date	Jun 21 2013 05:54:56
Configure Command	'./configure' '--with-apxs2=/usr/local/apache2/bin/apxs' '--with-iconv' '--with-gd' '--with-mysqli=mysqlnd' '--with-mysql=mysqlnd' '--with-pdo-mysql=mysqlnd' '--with-oci8=shared,instantclient,/opt/oracle/instantclient_11_2' '--with-pdo-oci=instantclient,/opt/oracle/instantclient_11_2,11.2'
Server API	Apache 2.0 Handler
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/usr/local/lib
Loaded Configuration File	/etc/php.ini
Scan this dir for additional .ini files	(none)
Additional .ini files parsed	(none)
PHP API	20121113
PHP Extension	20121212
Zend Extension	220121212
Zend Extension Build	API220121212,NTS
PHP Extension Build	API20121212,NTS
Debug Build	no
Thread Safety	disabled
Zend Signal Handling	disabled
Zend Memory Manager	enabled
Zend Multibyte Support	disabled
IPv6 Support	enabled
DTrace Support	disabled
Registered PHP Streams	php, file, glob, data, http, ftp, phar
Registered Stream Socket Transports	tcp, udp, unix, udg
Registered Stream Filters	convert.iconv.*, string.rot13, string.toupper, string.tolower, string.strip_tags, convert.*, consumed, dechunk

This program makes use of the Zend Scripting Language Engine:
 Zend Engine v2.5.0-dev, Copyright (c) 1998-2013 Zend Technologies

Powered By



Ejemplo 3


```
<?php  
// toda la información  
phpinfo();
```

Subir

[Condiciones generales de uso](#)

Copyright - ©Editions ENI



Buscar

[Índice](#) [Notas y marca páginas](#) [Favorito](#)

Índice

Información

[Título, autor...](#)

Introducción

[Objetivo del libro](#)[Breve historia de PHP](#)[¿Dónde conseguir PHP?](#)[Convenciones de escritura](#)

Información general sobre PHP

[¿Qué es PHP?](#)[Estructura básica de una página PHP](#)[Configuración de PHP](#)[Utilizar PHP desde la línea de comandos](#)

Variables, constantes, tipos y matrices

Operadores

Estructuras de control

Funciones y clases

Gestión de formularios

Acceder a las bases de datos

Administrar las sesiones

Enviar un correo electrónico

Gestión de archivos

Administrar los errores en un script PHP

Anexo

Sintaxis simplificada`php [opciones] [script]`Con`opciones`Opciones en la línea de comandos (por ejemplo, `-h` para obtener la ayuda, `-v` para la versión, etc.).`script`

El archivo que contiene el código PHP que se va a ejecutar.

Ejemplo

```
# php -v
PHP 5.5.0 (cli) (built: Jun 21 2013 05:56:07)
Copyright (c) 1997-2013 The PHP Group
Zend Engine v2.3.0, Copyright (c) 1998-2013 Zend Technologies
```

```
## php script.php Olivier
¡Hola Olivier!
```

Contenido del script `script.php`:

```
<?php
// mostrar un mensaje simple
// utilizando el parámetro
// pasado en la línea de comandos
echo "¡Hola $argv[1]!\n";
?>
```

Desde la versión 5.4, PHP en línea de comandos ofrece también un pequeño servidor Web que se puede utilizar para el desarrollo. Este servidor Web integrado se puede iniciar de la siguiente manera:

`php -S servidor:puerto [-t ruta]``servidor`

Nombre o dirección IP del servidor.

`puerto`

Puerto de escucha.

`ruta`

Directorio raíz de los documentos (directorio actual por defecto).

[Subir](#)