

# Domine PHP y MySQL

2<sup>a</sup> EDICIÓN



José López Quijado

www..es

Desde [www.ra-ma.es](http://www.ra-ma.es)  
podrá descargarse  
material adicional.



Ra-Ma®

# **Domine PHP y MySQL**

## **2<sup>a</sup> Edición**

## **Descarga de Material Adicional**

Este E-book tiene disponible un material adicional que complementa el contenido del mismo.

Este material se encuentra disponible en nuestra página Web [www.ra-ma.com](http://www.ra-ma.com).

Para descargarlo debe dirigirse a la ficha del libro de papel que se corresponde con el libro electrónico que Ud. ha adquirido. Para localizar la ficha del libro de papel puede utilizar el buscador de la Web.

Una vez en la ficha del libro encontrará un enlace con un texto similar a este:

*"Descarga del material adicional del libro"*

Pulsando sobre este enlace, el fichero comenzará a descargarse.

Una vez concluida la descarga dispondrá de un archivo comprimido. Debe utilizar un software descompresor adecuado para completar la operación. En el proceso de descompresión se le solicitará una contraseña, dicha contraseña coincide con los 13 dígitos del ISBN del libro de papel (incluidos los guiones).

Encontrará este dato en la misma ficha del libro donde descargó el material adicional.

Si tiene cualquier pregunta no dude en ponerse en contacto con nosotros en la siguiente dirección de correo: [ebooks@ra-ma.com](mailto:ebooks@ra-ma.com)

# **Domine PHP y MySQL**

## **2<sup>a</sup> Edición**

*José López Quijado*





La ley prohíbe  
copiar o imprimir este libro

## DOMINE PHP Y MySQL, 2<sup>a</sup> EDICIÓN

© José López Quijado.

© De la Edición Original en papel publicada por Editorial RA-MA

ISBN de Edición en Papel: 978-84-9964-008-2

Todos los derechos reservados © RA-MA, S.A. Editorial y Publicaciones, Madrid, España.

**MARCAS COMERCIALES.** Las designaciones utilizadas por las empresas para distinguir sus productos (hardware, software, sistemas operativos, etc.) suelen ser marcas registradas. RA-MA ha intentado a lo largo de este libro distinguir las marcas comerciales de los términos descriptivos, siguiendo el estilo que utiliza el fabricante, sin intención de infringir la marca y solo en beneficio del propietario de la misma. Los datos de los ejemplos y pantallas son ficticios a no ser que se especifique lo contrario.

RA-MA es una marca comercial registrada.

Se ha puesto el máximo empeño en ofrecer al lector una información completa y precisa. Sin embargo, RA-MA Editorial no asume ninguna responsabilidad derivada de su uso ni tampoco de cualquier violación de patentes ni otros derechos de terceras partes que pudieran ocurrir. Esta publicación tiene por objeto proporcionar unos conocimientos precisos y acreditados sobre el tema tratado. Su venta no supone para el editor ninguna forma de asistencia legal, administrativa o de ningún otro tipo. En caso de precisarse asesoría legal u otra forma de ayuda experta, deben buscarse los servicios de un profesional competente.

Reservados todos los derechos de publicación en cualquier idioma.

Según lo dispuesto en el Código Penal vigente ninguna parte de este libro puede ser reproducida, grabada en sistema de almacenamiento o transmitida en forma alguna ni por cualquier procedimiento, ya sea electrónico, mecánico, reprográfico, magnético o cualquier otro sin autorización previa y por escrito de RA-MA; su contenido está protegido por la Ley vigente que establece penas de prisión y/o multas a quienes, intencionadamente, reprodujeren o plagiaren, en todo o en parte, una obra literaria, artística o científica.

Editado por:

RA-MA, S.A. Editorial y Publicaciones  
Calle Jarama, 33, Polígono Industrial IGARSA  
28860 PARACUELLOS DE JARAMA, Madrid  
Teléfono: 91 658 42 80  
Fax: 91 662 81 39  
Correo electrónico: [editorial@ra-ma.com](mailto:editorial@ra-ma.com)  
Internet: [www.ra-ma.es](http://www.ra-ma.es) y [www.ra-ma.com](http://www.ra-ma.com)

Maquetación: Gustavo San Román Borruco  
Diseño Portada: Antonio García Tomé

ISBN: 978-84-9964-377-9

E-Book desarrollado en España en septiembre de 2014.

*Quiero dedicar este libro a todos los que les  
gusta la programación tanto como a mí.  
Gracias por vuestro interés.*

# ÍNDICE

---

---

<b>INTRODUCCIÓN .....</b>	<b>15</b>
<b>CAPÍTULO 1. LA ARQUITECTURA CLIENTE-SERVIDOR.....</b>	<b>19</b>
1.1 LAS DIRECCIONES IP Y EL SERVICIO DNS.....	20
1.2 LOS PROTOCOLOS TCP/IP .....	23
1.2.1 Los paquetes.....	24
1.2.2 Los puertos.....	25
1.2.3 Sockets .....	26
1.3 SITIOS DINÁMICOS.....	27
<b>CAPÍTULO 2. MONTANDO LOS SERVIDORES .....</b>	<b>29</b>
2.1 LA PLATAFORMA SERVIDORA .....	30
2.1.1 El servidor Apache .....	30
2.1.2 El intérprete de PHP 5.....	30
2.1.3 La base de datos MySQL.....	31
2.1.4 El servidor de correo .....	31
2.1.5 El servidor FTP .....	31
2.2 EL APPSERV .....	32
2.3 CONFIGURANDO EL SERVIDOR APACHE .....	35
2.4 COMPROBANDO EL FUNCIONAMIENTO.....	37
2.5 EL SERVIDOR DE CORREO.....	39
2.5.1 Configurando el servidor de correo.....	41
2.6 EL SERVIDOR FTP .....	45
2.6.1 Configurando el servidor FTP.....	50
2.7 CONSIDERACIONES FINALES.....	52

<b>CAPÍTULO 3. PRIMEROS PASOS EN PHP .....</b>	<b>53</b>
3.1 MI PRIMERA PÁGINA DINÁMICA .....	53
3.2 INICIACIÓN A LAS VARIABLES .....	58
3.2.1 Los nombres de las variables .....	61
3.3 GESTIÓN BÁSICA DE DATOS.....	63
3.3.1 Conocer y cambiar el tipo de un dato.....	67
3.4 EXPANSIÓN DE VARIABLES .....	71
3.5 OTRAS FUNCIONES DE MANEJO DE VARIABLES .....	74
3.6 VARIABLES DE VARIABLES.....	76
3.7 CONSTANTES .....	77
3.8 MATRICES .....	78
3.8.1 Matrices indexadas .....	79
3.8.2 Matrices asociativas .....	82
3.8.3 Matrices mixtas .....	83
3.8.4 Determinar el tamaño de una matriz .....	85
3.8.5 Ordenar una matriz.....	86
3.9 OTRAS BASES DE NUMERACIÓN .....	87
3.10 COMENTARIOS.....	88
3.11 RASTREO DE VARIABLES .....	89
<b>CAPÍTULO 4. CONDICIONALES, BUCLES Y FUNCIONES .....</b>	<b>91</b>
4.1 CONDICIONALES.....	91
4.2 BUCLES .....	97
4.2.1 Bucles mediante condición numérica.....	98
4.2.2 Bucles mediante condición no numérica .....	100
4.2.3 El bucle foreach .....	102
4.2.4 Interrupciones y reiteraciones .....	104
4.3 FUNCIONES .....	105
4.3.1 Pasando argumentos.....	107
4.3.2 Retorno desde una función.....	112
4.3.3 Ámbito de las variables.....	113
4.3.4 Variables estáticas .....	117
4.3.5 Recursividad.....	118
4.4 OPERADORES A NIVEL DE BIT .....	119
<b>CAPÍTULO 5. USO DE FORMULARIOS .....</b>	<b>123</b>
5.1 ENVÍO DE DATOS DESDE UN FORMULARIO .....	123
5.2 MÉTODOS DE ENVÍO .....	129

5.3 LAS VARIABLES DEL INTÉRPRETE .....	130
5.4 ENVIANDO ARCHIVOS .....	133
5.4.1 Limitando el tamaño del archivo.....	137
5.4.2 Enviando múltiples archivos.....	138
5.5 PROCESANDO LOS ARCHIVOS ENVIADOS .....	141
5.6 ERRORES IMPREVISTOS .....	148
5.7 PÁGINAS AUTO PROCESADAS .....	151
<b>CAPÍTULO 6. FUNCIONES PARA EL MANEJO DE DATOS.....</b>	<b>153</b>
6.1 MANEJO BÁSICO DE CADENAS .....	153
6.2 LA CODIFICACIÓN URL.....	166
6.3 TRATAMIENTO DE CADENAS PARA HTML.....	176
6.4 LAS CADENAS COMO MATRICES .....	180
6.5 ENCRIPCIÓN DE CADENAS .....	183
6.6 FUNCIONES NUMÉRICAS.....	184
6.7 FUNCIONES DE FECHA .....	194
<b>CAPÍTULO 7. EXPRESIONES REGULARES .....</b>	<b>203</b>
7.1 QUÉ SON LAS EXPRESIONES REGULARES .....	204
7.2 INTRODUCCIÓN A LOS PATRONES .....	204
7.2.1 Patrones de secuencia o fijación .....	204
7.2.2 Patrones multiplicadores .....	205
7.2.3 Patrones de alternativa .....	207
7.2.4 Los paréntesis.....	207
7.2.5 Escapado de metacaracteres .....	207
7.2.6 Precedencia de patrones .....	208
7.3 COMPROBACIÓN DE CADENAS .....	209
7.4 EL ESTÁNDAR POSIX .....	210
7.4.1 Patrones de Posix .....	211
7.4.2 Gestión de expresiones regulares Posix .....	212
7.5 EL ESTÁNDAR PERL .....	215
7.5.1 Los patrones de Perl .....	216
7.5.2 Gestión de expresiones regulares Perl.....	218
<b>CAPÍTULO 8. FICHEROS.....</b>	<b>221</b>
8.1 EJECUTANDO OTROS SCRIPTS .....	221
8.1.1 Consideraciones sobre scripts externos.....	226
8.2 LOS FICHEROS .....	227
8.2.1 Abrir ficheros .....	227

8.2.2 Cerrar ficheros.....	229
8.2.3 Cómo leer en los ficheros.....	230
8.2.4 La escritura en ficheros .....	242
8.2.5 Eliminar ficheros .....	243
8.2.6 Copiando ficheros .....	244
8.2.7 Renombrado de ficheros .....	245
8.2.8 Las propiedades de los ficheros .....	245
8.2.9 Los permisos .....	248
8.3 DIRECTORIOS .....	260
8.3.1 Manejo básico de directorios .....	261
<b>CAPÍTULO 9. COOKIES Y SESIONES .....</b>	<b>267</b>
9.1 COOKIES .....	268
9.2 SESIONES .....	275
<b>CAPÍTULO 10. LA COMUNICACIÓN WEB .....</b>	<b>283</b>
10.1 LAS CABECERAS.....	283
10.1.1 Las cabeceras de la solicitud .....	284
10.1.2 Las cabeceras de la respuesta .....	288
10.2 AUTENTICACIÓN.....	291
10.3 SOCKETS.....	294
<b>CAPÍTULO 11. ORIENTACIÓN A OBJETOS.....</b>	<b>299</b>
11.1 CARACTERÍSTICAS DE LA POO .....	299
11.2 CONCEPTOS BÁSICOS .....	300
11.3 CREACIÓN Y USO.....	301
11.4 HERENCIA .....	306
<b>CAPÍTULO 12. IMÁGENES.....</b>	<b>309</b>
12.1 LO QUE NECESITAMOS .....	309
12.2 LO QUE PODEMOS HACER .....	310
12.3 EMPEZANDO A TRABAJAR .....	310
12.4 EL COLOR .....	318
12.5 CREAR IMÁGENES .....	324
12.6 COPIA DE IMÁGENES .....	326
12.7 FIGURAS PREDEFINIDAS .....	329
12.8 FILTROS .....	333
12.9 TEXTO EN LAS IMÁGENES .....	336

<b>CAPÍTULO 13. CORREO ELECTRÓNICO .....</b>	<b>341</b>
13.1 CORREO SENCILLO .....	341
13.2 CORREO COMPLEJO .....	343
<b>CAPÍTULO 14. MISCELÁNEA .....</b>	<b>349</b>
14.1 ERRORES .....	349
14.2 EVALUAR EXPRESIONES .....	351
14.3 FTP .....	352
14.4 PDF .....	355
14.5 LA DIRECCIÓN IP .....	366
<b>CAPÍTULO 15. BASES DE DATOS Y SQL .....</b>	<b>369</b>
15.1 CÓMO ES UNA BASE DE DATOS .....	370
15.2 EL LENGUAJE SQL .....	372
15.2.1 Consultas estructurales .....	373
15.2.2 Consultas de datos .....	381
<b>CAPÍTULO 16. BASES DE DATOS ODBC .....</b>	<b>391</b>
16.1 ESTABLECIENDO LA CONEXIÓN ODBC .....	391
16.2 USO BÁSICO DE ODBC .....	394
16.3 AMPLIANDO CONSULTAS .....	399
<b>CAPÍTULO 17. MYSQL .....</b>	<b>407</b>
17.1 INTRODUCCIÓN A MYSQL .....	408
17.2 UN CASO PRÁCTICO .....	425
17.2.1 La página principal .....	428
17.2.2 Agregar citas .....	430
17.2.3 Borrar una cita .....	431
17.2.4 Modificar una cita .....	431
17.3 HERRAMIENTA DE GESTIÓN VISUAL .....	432
<b>CAPÍTULO 18. SCRIPTS ÚTILES .....</b>	<b>439</b>
18.1 EL ORIGEN DE UNA VISITA .....	439
18.2 EVITANDO LOS BOTS .....	449
18.3 FORMULARIOS EN DOCUMENTOS .....	456
18.4 ACTUALIZACIONES AUTOMÁTICAS .....	460
<b>CAPÍTULO 19. FOROS EN INTERNET .....</b>	<b>463</b>
19.1 OBTENIENDO PHPBB3 .....	464
19.2 INSTALANDO EL FORO .....	465
19.2.1 Primeros pasos .....	465

19.2.2 Configurando el foro .....	471
19.3 USANDO EL FORO .....	474
19.3.1 El registro .....	475
19.4 VUELVE EL ADMINISTRADOR .....	476
19.4.1 Categorías y foros .....	476
19.4.2 Las copias de seguridad .....	478
19.5 CONCLUSIONES .....	478
<b>CAPÍTULO 20. FLASH EN PHP.....</b>	<b>481</b>
20.1 LO QUE NECESITAMOS .....	482
20.2 LA LIBRERÍA MING .....	482
20.2.1 La clase SWFAction.....	483
20.2.2 La clase SWFBitmap.....	484
20.2.3 La clase SWFButton.....	484
20.2.4 La clase SWFDisplayItem.....	485
20.2.5 La clase SWFFill.....	485
20.2.6 La clase SWFFont .....	486
20.2.7 La clase SWFGradient .....	486
20.2.8 La clase SWFMorph.....	486
20.2.9 La clase SWFMovie .....	487
20.2.10 La clase SWFShape.....	487
20.2.11 La clase SWFSprite .....	488
20.2.12 La clase SWFText .....	489
20.2.13 La clase SWFTextField .....	489
20.2.14 Las medidas.....	490
20.2.15 Comprobando la librería .....	490
20.3 NUESTRO PRIMER EJEMPLO .....	491
20.4 AÑADIENDO FORMAS .....	494
20.4.1 Creando líneas rectas.....	494
20.4.2 Creando curvas.....	499
20.4.3 Figuras cerradas.....	502
20.5 RELLENANDO FIGURAS.....	505
20.5.1 Rellenos de color.....	505
20.5.2 Rellenos con gradientes.....	511
20.5.3 Rellenos con imágenes .....	514
20.6 TEXTO .....	518
20.6.1 El texto más simple .....	518
20.6.2 Algunas mejoras .....	520

20.6.3 Colocando el texto.....	522
20.7 ANIMACIONES .....	524
20.7.1 Una animación simple.....	524
20.7.2 Técnicas profesionales .....	526
20.7.3 Escalados.....	534
20.7.4 Eliminación de objetos.....	535
20.7.5 Grabar la película .....	537
20.7.6 Interpolaciones de forma .....	537
20.8 IMÁGENES.....	540
20.9 ACCIONES .....	541
20.10 BOTONES .....	543
20.11 TEXTO DINÁMICO .....	549
20.12 CONSIDERACIONES FINALES.....	553
<b>CAPÍTULO 21. DEPURANDO NUESTRO TRABAJO .....</b>	<b>555</b>
21.1 EL SERVIDOR WAMP .....	555
21.1.1 La instalación de WampServer .....	556
21.1.2 Configurando WampServer .....	561
21.1.3 Probando WampServer .....	563
21.2 EL ZEND DEBUGGER.....	564
21.3 ECLIPSE + PDT .....	566
21.4 DEPURANDO PHP.....	572
<b>APÉNDICE A. LA CONFIGURACIÓN DEL INTÉRPRETE.....</b>	<b>581</b>
A.1 LAS EXTENSIONES .....	581
A.2 LAS DIRECTIVAS .....	583
<b>APÉNDICE A. PALABRAS RESERVADAS DE PHP.....</b>	<b>589</b>
<b>APÉNDICE B. LAS VARIABLES DEL INTÉRPRETE.....</b>	<b>599</b>
<b>APÉNDICE C. DIRECCIONES ÚTILES DE INTERNET .....</b>	<b>603</b>
<b>APÉNDICE D. EL CÓDIGO ASCII.....</b>	<b>607</b>
<b>APÉNDICE E. EL CONTENIDO DEL CD .....</b>	<b>613</b>
<b>APÉNDICE F. LA VERSIÓN 5.3 DE PHP .....</b>	<b>615</b>
<b>ÍNDICE ALFABÉTICO.....</b>	<b>617</b>

## **INTRODUCCIÓN**

---

---

Este libro complementa los trabajos que he dedicado a la programación de sitios web. En los dos anteriores (*"Domine XHTML 1.0 y CSS 2"* y *"Domine JavaScript, 2<sup>a</sup>. Edición"*, publicados por esta misma editorial) me expliqué sobre la programación en el lado del cliente. Ahora que usted, lector, ha llegado a dominar esa línea de trabajo es el momento de dar un gran salto en la creación de sitios web: la programación en el lado del servidor.

A lo largo de la segunda edición de esta obra aprenderá todo lo que necesita para acometer los proyectos más ambiciosos que pueblan la Red de redes. Atrás en el tiempo quedan los días oscuros en que las páginas de Internet se limitaban a cargarse y ejecutarse en el lado del cliente, con unos comportamientos inamovibles y una rigidez que hoy es inaceptable. Aprenderá a crear proyectos como una tienda virtual, foros, libros de visitas, tablones de anuncios, etc. Las posibilidades que abre esta vía de trabajo son ilimitadas.

En los dos primeros capítulos aprenderemos los conceptos básicos de esta forma de trabajar; desde el significado de “lado del cliente” y “lado del servidor” hasta la forma de comprender la arquitectura cliente-servidor, los conceptos básicos sobre redes, el software necesario para trabajar con un servidor, etc.

En los capítulos del 3 al 14 aprenderemos los conceptos básicos de la programación en el lado del servidor. Conceptos que, para el lector que ya conozca las técnicas de programación en el lado del cliente (JavaScript) son familiares, tales como variables, matrices, objetos, etc., serán elevados ahora a su máxima expresión. Para ello emplearemos PHP. Se ha escogido este lenguaje para la programación dinámica de sitios web por varias razones, entre las que cabe destacar las siguientes:

- Es un lenguaje que combina potencia, versatilidad y sencillez de aprendizaje.
- Permite gestionar eficientemente cualquier elemento de una página web.
- Se comunica perfectamente con HTML (XHTML) y JavaScript, permitiendo alcanzar un elevado nivel de integración con el navegador.
- Gestiona con eficacia bases de datos, tan necesarias para gran cantidad de proyectos modernos.
- Es un lenguaje en constante evolución. En el momento de escribir este libro se halla disponible la versión 5 (PHP 5), que presenta grandes mejoras con respecto a las anteriores.
- Es un lenguaje de programación totalmente libre, lo que en el ámbito informático se conoce como *open source*. Esto quiere decir que usted podrá disponer de la última versión completamente gratis. Pero no sólo eso. Además, puede disponer libremente del código fuente original del lenguaje, por si decide modificarlo para adaptarlo a sus necesidades específicas.
- Ningún otro lenguaje de programación está tan extendido en Internet. Los sitios de mayor éxito están basados en el uso de este lenguaje. Por algo será.

Del capítulo 15 al 17 acometeremos el conocimiento de MySQL, el gestor de bases de datos en Internet por excelencia. Entre otras, cuenta con las siguientes ventajas:

- Al igual que PHP, se trata de una herramienta open source. Aunque existen licencias de pago, destinadas a grandes usuarios que necesitan un soporte técnico especializado, éste no será su caso. En este libro encontrará toda la ayuda necesaria para manejar MySQL sin necesidad de soporte técnico. No obstante, existe en Internet gran cantidad de documentación, tanto en inglés como en español.
- Es un gestor de bases de datos sensiblemente más rápido y eficiente que la mayoría de los que hay en el mercado, muchos de los cuales, además, son de pago.
- Permite una gran versatilidad a la hora de almacenar y gestionar todo tipo de datos.
- Se gestiona mediante un lenguaje específico conocido como SQL (cuyo estudio también abordaremos, lógicamente, en el libro) que se integra perfectamente con PHP.

Desde el capítulo 18 seguiremos profundizando en la programación de scripts en el lado del servidor, usando los conceptos que hayamos aprendido en las

partes anteriores, y descubriendo otros nuevos. Además, abordaremos el desarrollo y la puesta en marcha completa de una aplicación web muy extendida hoy en día: un foro virtual. También aprenderemos algo que no todos los programadores de Internet conocen: el desarrollo de películas de Flash... SIN FLASH. Usando, exclusivamente PHP, podremos desarrollar las películas que necesitemos sin comprar ni usar ninguna herramienta adicional, y de un modo muy fácil.

Por último, cierra la obra una serie de apéndices en los que el lector encontrará información muy útil: desde una relación de páginas web de visita obligada hasta la descripción de herramientas complementarias que facilitan enormemente el trabajo del webmaster.

Además, en el CD adjunto encontrará los códigos de todos los ejercicios desarrollados en el texto, así como material complementario necesario para su trabajo.

Deseo que usted encuentre estas disciplinas tan fascinantes como yo. Con este objetivo he redactado este libro, poniendo a su servicio toda mi experiencia, tanto como webmaster como a nivel docente. Si usted disfruta tanto leyendo este libro como yo he disfrutado escribiéndolo, y si le resulta tan útil como deseo, me sentiré satisfecho.

Para poder sacarle partido al presente texto es necesario que conozca perfectamente las técnicas de programación de sitios web en el lado del cliente (XHTML y JavaScript). Si no, no podrá seguir adelante. Si no las conoce, le recomiendo que lea previamente los dos títulos anteriores de esta trilogía.

Cuando complete el estudio de este volumen habrá alcanzado el nivel necesario para empezar a desarrollar sitios auténticamente profesionales por usted mismo. El desarrollo real de sitios dinámicos le dará luego la experiencia necesaria para crear su propio estilo de trabajo.



## LA ARQUITECTURA CLIENTE-SERVIDOR

---

---

Cuando se habla de Internet el ciudadano común sabe que enciende su ordenador, abre su navegador y tiene una barra de direcciones donde teclea el nombre de la página que quiere visitar. Si todo va bien, se accede a la página solicitada y, si no, aparece un mensaje informando de que se ha producido un error. Lo que hay "detrás de la pared" (al otro lado del hilo telefónico o de la conexión ADSL, cable-módem, etc.) no se sabe lo que es. El usuario común no tiene por qué preocuparse de ello. Pero nosotros sí. En algún lugar del mundo (en realidad en distintos lugares) existen unos ordenadores conocidos genéricamente como *servidores*. Se les llama así porque son los que tienen almacenadas las páginas que visita el usuario, y se las sirven cuando éste las solicita. Por extensión, los ordenadores de los usuarios se conocen, genéricamente, como *clientes*. Así pues, cuando nos conectamos a Internet y pedimos una página tenemos dos ordenadores en juego: el nuestro propio (desde el que nos conectamos) que es el cliente y el que nos envía por cable (u otro medio) aquello que hemos pedido (el servidor).

En realidad, la arquitectura de Internet es bastante más compleja que esto, pero lo que hemos mencionado aquí constituye la espina dorsal de todo el sistema. En este capítulo vamos a analizar esta arquitectura en la medida necesaria para comprender qué es lo que vamos a aprender a lo largo del libro. Los conceptos que vamos a exponer a continuación les parecerán muy elementales a las personas con experiencia previa en redes, pero son necesarios como introducción para aquellos lectores que no se hayan asomado nunca a este aspecto de la informática. Ruego a los conocedores del tema que me disculpen por la simplificación que hago de estos conceptos, pero no pretendo que éste sea un tratado acerca de redes sino, simplemente, una introducción necesaria para luego poder ahondar en la programación en el lado del servidor.

## 1.1 LAS DIRECCIONES IP Y EL SERVICIO DNS

En realidad, cuando un usuario se conecta a Internet y escribe en la barra de direcciones de su navegador el nombre de la página que desea visitar tiende a pensar que es mediante este nombre como se localiza dicha página en La Red. Pero esto no es así. Cuando un ordenador está conectado a Internet (y un servidor es un ordenador) tiene asignado un número de identificación que permite localizarlo de forma inequívoca: es la *dirección IP*. Este número es un grupo de cuatro valores, cada uno de los cuales está en un rango entre 0 y 255. Así pues, una dirección IP tiene un aspecto como 192.168.1.1.

Las direcciones IP se dividen, atendiendo a su ámbito, en dos categorías: *internas* y *externas*. Las direcciones internas se emplean para identificar y localizar ordenadores dentro de nuestra propia red local (en el caso de que tengamos una). Las direcciones externas se emplean para acceder a Internet, es decir, para localizar e identificar ordenadores remotos. Dicho de otra manera: la diferencia entre una red local e Internet es el ámbito de trabajo. Mientras que en una red local nos conectamos con ordenadores que están en el mismo edificio que el nuestro (en ocasiones incluso en la misma habitación), en Internet nos conectamos a equipos que pueden estar a 300 metros o en el otro lado del mundo. Los rangos de direcciones IP internas aparecen en la tabla de la figura 1.1.

RANGO
De 10.0.0.0 a 10.255.255.255
De 172.16.0.0 a 172.31.255.255
De 192.168.0.0 a 192.168.255.255

Figura 1.1

Cuando usted se conecta a un ordenador de su red local (en casa o en la oficina) realmente se está conectando a un equipo que tiene asignada un IP de las que aparecen en la tabla. Usted no tiene por qué saber esto. Sus ordenadores, o los de sus compañeros de trabajo, tienen un nombre y un ícono que los identifica para usted en el escritorio. Sin embargo, cada vez que usted “llama” a uno de esos ordenadores (a través del nombre o el ícono), el sistema operativo de su ordenador “sabe” a qué IP tiene que dirigirse.

Existe una dirección IP especial que usaremos mucho en los ejercicios de este libro: la 127.0.0.1. Esta dirección se conoce como *de bucle local* o *localhost*. Identifica siempre al ordenador con el que estamos trabajando. Cualquier plataforma del mundo (cualquier sistema operativo) “sabe” que, si solicitamos un documento, página web, o cualquier otro recurso a esta dirección, en realidad lo

estamos solicitando a nuestro propio ordenador, que lo busca en una carpeta determinada (aquella que se ha definido como localhost). No se preocupe ahora de qué carpeta es esa, cómo se define, ni cómo funciona todo esto en detalle. En el próximo capítulo hablaremos en profundidad de esto.

Cuando un usuario se conecta a Internet y solicita una página web a través de su barra de direcciones, lo que hace es establecer contacto con el ordenador que tiene almacenada esa página web. En este ordenador está funcionando un programa que se conoce, genéricamente, como **servidor web**. Este programa busca la página solicitada en el disco duro del servidor y se la envía al cliente. Así pues, podemos decir que un servidor es un ordenador que almacena páginas web y también un programa que se encarga de entregar dichos documentos a los clientes que lo soliciten. Por analogía, el cliente es un ordenador (aquel desde el que nos conectamos a Internet) y un programa cliente (el navegador con el que nos conectamos y donde se muestra la página solicitada). Como vemos, cada elemento de una arquitectura cliente-servidor se compone, por lo tanto, de un hardware y, por supuesto, un software.

En este momento, el lector avisado me dice: "Pero cuando yo tecleo una dirección de Internet, escribo el nombre de la página que quiero, no el número que se llama dirección IP del que usted hablaba hace un momento". Pues sí. Tiene usted razón. Los creadores de Internet comprendieron muy pronto que para los usuarios es mucho más fácil recordar un nombre con un significado específico que un número de IP. Además, las páginas de Internet pueden cambiar de servidor físico, con lo que cambiaría su dirección IP. Ahí es donde entra en juego el servicio **DNS** (*Domain Name Server*, Servidor de Nombre de Dominio) que completa Internet de una forma muy útil. Cuando usted se conecta a Internet lo hace a través de un proveedor de servicios (ISP, *Internet Service Provider*, Proveedor de Servicios de Internet). Este ISP le asigna dos direcciones DNS que usted especifica en la configuración de su conexión a Internet. Si su ISP le entrega un CD de instalación, es muy posible que el programa instalador establezca estas dos direcciones en la configuración de su navegador; si no, su ISP le dice cuáles son estas direcciones y cómo debe configurar su conexión.

Cuando usted abre su navegador y solicita una página web, su ordenador no se conecta directamente al servidor de dicha página. Se conecta a uno de los servidores DNS establecidos. El servidor DNS contiene un directorio con todas las páginas web del mundo y las IPs de los correspondientes servidores. Así pues, el DNS busca en la lista la página que usted ha solicitado y el número de IP correspondiente. Esto es lo que se llama **resolver el nombre**. El DNS le devuelve el número de IP a su navegador y éste se conecta entonces al servidor web que corresponde a esa IP y que contiene la página solicitada. En resumen, la conexión a una página web se establece así:

- El navegador envía el nombre de la página web deseada al DNS.
- El DNS resuelve el nombre y le devuelve al cliente la IP del servidor que contiene la página solicitada.
- El navegador se conecta al servidor web correspondiente mediante la IP recibida del DNS y solicita la página web.
- El servidor web entrega la página al cliente.

El esquema de funcionamiento aparece en la figura 1.2. Como usted puede ver, se aprecian gráficamente los cuatro pasos mencionados. Estos pasos son siempre los mismos, salvo que usted teclee en la barra de direcciones la IP de la página que desea ver. Pero claro, ningún usuario se aprende o se anota las IPs de las páginas. Todos trabajamos mejor con nombres. Para eso están los DNS, que se actualizan cada vez que aparece una nueva página en Internet.



Figura 1.2

¿Se va haciendo una idea de la estructura de la Red de Redes? En realidad esto es un poco más complejo que lo que hemos visto hasta ahora. Por ejemplo, los ISPs suelen “intercalar” unos equipos entre su ordenador e Internet. Estos equipos se conocen con el nombre de *proxys*. Un proxy es un ordenador que almacena una copia de las páginas que usted visita con más frecuencia. De este modo, cuando

usted solicita de nuevo esas páginas, es el proxy quien se las sirve, sin que se busquen de nuevo en Internet. Este sistema tiene la ventaja de que le sirve las páginas más rápido que si hubiera que acceder a Internet, resolver el nombre y descargarlas desde el servidor correspondiente. Como contrapartida, puede que la página que le sirve el proxy no esté actualizada. En ese caso conviene que el navegador ignore lo que el proxy le ofrece y se conecte a Internet a buscar la página. Usted ya conoce los recursos que emplean los programadores en HTML para que la página se cargue desde el servidor, de modo que el cliente tenga la última versión disponible. Programando en PHP también existen recursos al efecto que usted aprenderá. Lo cierto es que con la implementación masiva de líneas de banda ancha, y los ISP compitiendo diariamente por ofrecer mayor velocidad de conexión, el uso de servidores proxy está dejando de tener sentido para el fin que hemos descrito. Sin embargo, los ISP los siguen empleando a fin de optimizar otros aspectos de la conexión entre el cliente e Internet.

## 1.2 LOS PROTOCOLOS TCP/IP

Conectar entre sí dos o más ordenadores para que puedan compartir información y recursos no es una tarea simple. Entre otras cosas, es necesario que la forma en que transmiten y reciben información los distintos equipos esté normalizada. Para este fin existen unos *protocolos* de comunicación. El concepto de protocolo implica que la transferencia de información se hace en la misma forma en cada ordenador, de tal modo que la información que uno envía, el otro la puede interpretar cuando la recibe. Esto es lógico. Si yo sólo hablo español y usted sólo habla inglés no vamos a entendernos nunca. Hace falta una normalización para que nos entendamos. El conjunto de normas establecidas para la transferencia de información es un protocolo. Todos los ordenadores actuales se comunican entre sí siguiendo lo que se conoce como *pila de protocolos TCP/IP*. Esta pila abarca diversos protocolos que los equipos informáticos emplean según el tipo de información que transfieran entre ellos. Por ejemplo, no se emplea el mismo protocolo para transferir una página web que para transferir un fichero que el usuario quiere descargarse, o para transmitir o recibir un mensaje de correo electrónico. Cada uno de los distintos servicios que uno puede encontrar en una Red funciona mediante un protocolo específico para dicho servicio.

A fin de que todos los ordenadores del mundo “conozcan” e implementen correctamente dichos protocolos se creó lo que hoy día se conoce como *el modelo OSI de la ISO*. La ISO (*International Standard Organization*, Organización Internacional de Normalizaciones) es una agencia dedicada a normalizar todo aquello que pueda normalizarse. Desde el tamaño de las clavijas de los enchufes eléctricos hasta, lógicamente, los protocolos de comunicaciones. El modelo OSI

(*Open Standard Interconnectivity*, Interconexión Normalizada Abierta) reúne todos los protocolos de los distintos servicios de que se puede disponer en una red de ordenadores. En realidad incluye más normalizaciones como, por ejemplo, los conectores de las tarjetas de red y otros aspectos de la arquitectura física de una red. La explicación que damos aquí está, por fuerza, muy simplificada, ya que para obtener un conocimiento profundo de redes necesitaríamos varios volúmenes y, desde luego, es algo que está fuera del objetivo de este libro. Lo que sí tenemos que tener claro es que cualquier ordenador del mundo, independientemente del sistema operativo que monte, respeta estas normalizaciones. Los fabricantes de hardware y de sistemas operativos ya se ocupan de que esto sea así. De hecho, un ordenador o un sistema operativo que no cumpliera estos protocolos no podría conectarse a ninguna red y no tendría mucho uso. Esta pila de protocolos es, por tanto, universal. Su sistema operativo la tiene implementada y gracias a ello puede usted conectarse a Internet. En posteriores capítulos veremos algunos aspectos de estos protocolos especialmente interesantes, a efectos de programación de sitios web.

### 1.2.1 Los paquetes

Como he dicho, no nos vamos a meter en detalle en el funcionamiento de la pila de protocolos TCP/IP ni del modelo OSI. Sin embargo, si necesitamos conocer algunas líneas generales. Por ejemplo, debemos saber que la información que se transmite entre un servidor y un cliente no viaja “de golpe”, sino que antes de ser enviada es dividida en pequeños fragmentos que se conocen con el nombre de **paquetes**. Cuando estos paquetes llegan al ordenador de destino, son reagrupados para reconstruir la información original. Esto funciona así para cualquier transferencia de información de cualquier tipo. Este modo de transferir información tiene una doble razón de ser:

- Por una parte, dada la compleja estructura de nodos de La Red, cada paquete puede viajar por un camino diferente, con lo cual puede haber varios paquetes viajando del servidor al cliente simultáneamente.
- Por otra parte, si algún paquete se pierde o deteriora durante el trayecto, no es necesario reenviar toda la información, sino que basta con volver a mandar ese paquete.

Los paquetes están constituidos, básicamente, por dos partes: la **cabecera** y el **cuerpo**. En el cuerpo es donde se encuentra el fragmento de información que se transmite. En la cabecera encontramos algunas informaciones interesantes:

- El protocolo empleado, dependiendo del servicio del que se trate (un documento web, una descarga de archivos, etc).

- Tamaño del cuerpo del paquete y *checksum*. El checksum es un método de comprobación que se emplea para verificar que el paquete haya llegado correctamente. Si no es así, lo pide de nuevo.
- Las direcciones IP de los dos ordenadores, el de origen y el de destino. El lenguaje PHP permite identificar la IP del cliente. Esto se emplea en sitios de comercio electrónico o donde el cliente puede enviar o solicitar información sensible. Esto nos lleva de nuevo al uso de proxys por parte de los ISP. Algunos proxys interponen su propia IP entre la del cliente y la del servidor, con lo que la IP del cliente queda "oculta". PHP 5 implementa un sistema muy efectivo de identificación de la IP del cliente.
- El puerto de comunicaciones empleado para la transferencia de información. En el próximo apartado comentaremos algo más al respecto.
- Información adicional, tal como la relativa al camino que sigue el paquete para llegar desde el ordenador de origen al de destino.

Como ve, la información contenida en la cabecera es de una importancia tan vital como la que hay en el cuerpo. Volveremos sobre esto en su momento.

### 1.2.2 Los puertos

Cada uno de los servicios que se pueden usar en Internet implica el uso de uno o más **puertos**. Los puertos son canales numerados que se usan para transferir los distintos tipos de información. Los sistemas operativos modernos reconocen más de 65.000 puertos diferentes. De éstos, los más significativos son los primeros 1.024, que se conocen con el nombre genérico de **puertos bien conocidos**. Éstos se emplean para comunicaciones de distintos tipos.

En la tabla de la figura 1.3 se citan los más habituales. Los puertos del 1025 en adelante no tienen, a priori, un uso tipificado, por lo que pueden usarse para lo que se quiera. Este elevado número de puertos tiene una ventaja. Se pueden diseñar sistemas que se comuniquen por cualquiera de los puertos altos para transferencias específicas que un programador implemente en una aplicación. Pero también hay un problema: cuando un puerto no se usa, el Administrador del Sistema debería asegurarse de que está *cerrado*, es decir, que no puede entrar ni salir información a través de él. Pero este estado ideal de cosas sólo se ve en los sueños. En la realidad, cualquier sistema comercial tiene muchos puertos abiertos, aunque no se usen. Esto implica un riesgo de intrusión por parte de personas malintencionadas, que pueden modificar nuestro equipo, robarnos información, etc. Afortunadamente existen en el mercado unos programas conocidos como *cortafuegos (firewalls)* que palián bastante esta situación.

Nº DE PUERTO	SERVICIO	DESCRIPCIÓN
20 y 21	FTP	Transferencia de archivos (descarga o envío de archivos al servidor)
25	SMTP	Envío de correo electrónico
80	HTTP	Transferencia de páginas web
110	POP	Recepción de correo electrónico
43	HTTPS	Transferencia de documentos web bajo una capa segura
8080	HTTP	Transferencia de páginas web (puerto alternativo empleado por algunos sistemas)

*Figura 1.3*

Estos puertos son los más habituales, aunque, como ya se ha dicho, existen muchos más puertos bien conocidos y otros de uso programable. Cuando un ordenador está enviando datos por un puerto se dice que **habla** por ese puerto. Cuando está recibiendo datos, o pendiente a la espera de recibirlos, por un puerto, se dice que **escucha** por ese puerto. Para que se produzca transferencia de datos, debe haber un ordenador hablando y otro escuchando.

Una buena técnica de seguridad, cuando use programas de transferencia de datos, es cambiar los puertos “normales” por otros más altos, por encima de 1025, en los dos equipos participantes en la transferencia.

### 1.2.3 Sockets

Cuando nos conectamos a una página web podemos teclear su dirección IP. Por ejemplo, así: **http://127.0.0.1**. Esto sería una llamada a la página principal (index.htm o index.php) del sitio web alojado en la máquina que tiene la dirección IP indicada. Como lo que buscamos es una página web (servicio HTTP, *Hyper Text Transfer Protocol*, Protocolo de Transferencia de Hiper Texto), el sistema asume, por defecto, que el servidor escucha por el puerto 80. Si el servidor está configurado de tal modo que escucha por otro puerto distinto, es necesario

especificarlo al teclear la dirección. Por ejemplo: si el servidor escucha por el puerto 8080, la dirección anterior quedaría así: **http://127.0.0.1:8080**. El puerto se separa de la IP mediante el signo de dos puntos. Esta notación completa (IP:puerto) se conoce con el nombre de *socket*.

## 1.3 SITIOS DINÁMICOS

Cuando tenemos una página escrita que utiliza únicamente las tecnologías del lado del cliente (HTML y JavaScript) se dice que es estática. Esto significa que sus contenidos son siempre los mismos. Si queremos que cambien, debemos cambiarlos nosotros mismos, editando la página, para que los usuarios vean los nuevos contenidos. A menudo esto es imposible, debido a que hay muchas ocasiones en las que los contenidos deben ser dependientes de una solicitud por parte del usuario. Suponga que tiene usted que escribir una página con un listado de empresas de su ciudad. Las habrá de todos los gremios. Ahora un usuario que consulta su página quiere ver una lista de empresas de, digamos, floristería. El usuario sólo quiere ver las floristerías, que es lo que necesita. No quiere ver una página enorme, con todas las empresas, y tener que andar buscando las floristerías. Quiere un resultado claro y útil. Las páginas dinámicas emplean lo que se conoce como programación en el lado del servidor. Desde el punto de vista técnico, la principal diferencia es que una página estática, cuando es llamada por un cliente, baja al navegador tal como está, y se muestra como es. Cuando el cliente llama a una página dinámica, existe un proceso en el lado del servidor que crea en ese momento la página, incorporando los contenidos que el cliente ha solicitado, y sirve la página recién creada al cliente. Otro cliente que solicite la misma página con otros contenidos recibirá un resultado diferente. Por eso se habla de páginas dinámicas, porque su contenido puede variar y se genera en el servidor en el momento de la solicitud.

## Capítulo 2

# MONTANDO LOS SERVIDORES

---

---

Ya sabemos que las páginas dinámicas se programan empleando tecnologías de servidor. Para probar y analizar los ejercicios de este libro, así como para que desarrolle usted sus propias páginas, necesita tener un servidor de red que le proporcione los servicios necesarios. Por supuesto puede usted buscar, si lo desea, un servidor en Internet. Pero los que hay gratuitos no suelen tener las prestaciones que vamos a necesitar en este libro. Por otra parte, si cada nuevo código que vamos a probar debe ser transferido a un servidor remoto, va usted a perder un tiempo precioso. Por estas y otras razones lo ideal es que tenga usted el servidor en su casa. Esto tiene la ventaja añadida de que usted podrá implementar los servicios que necesite y configurarlos como le convenga en cada caso.

Como la mayoría de los usuarios no tienen una red local instalada en su domicilio, lo que vamos a hacer es usar un solo ordenador como cliente y servidor. Esto no es nada extraño. Muchos desarrolladores, incluido yo mismo, usamos este recurso para tener un entorno cómodo y práctico en el que crear nuestros proyectos. Así pues, nos encontramos con que en nuestro ordenador ya tenemos los programas clientes más habituales: el navegador de Internet (Microsoft Internet Explorer, Netscape Navigator, Ópera o cualquier otro) y el cliente de correo electrónico (p.e. Outlook Express). Si, además, montamos los programas servidores, ya tendremos una red local en nuestro ordenador. La única diferencia es que, al llamar a las páginas así creadas, en lugar de usar una dirección de Internet usaremos la dirección de nuestra propia máquina local (ya veremos cómo). Luego, una vez creado y probado un sitio, lo subiremos al servidor remoto que nos interese y allí estará disponible para todo el mundo. Lo que vamos a hacer en este capítulo es montar nuestra máquina para que constituya, por si misma, una red y configurarla adecuadamente. Es un proceso muy simple y que sólo habrá que llevar a cabo una

vez. Algunos de los programas que vamos a montar ahora no los emplearemos hasta más adelante y, en su momento, los configuraremos adecuadamente.

## 2.1 LA PLATAFORMA SERVIDORA

Los programas que necesitaremos para seguir esta obra y para la mayoría de nuestros trabajos para Internet o para una Intranet serán los siguientes:

- Servidor de páginas web.
- Intérprete de PHP 5.
- Servidor de bases de datos.
- Servidor de correo electrónico.
- Servidor de FTP.

En este apartado vamos a hablar, aunque sea brevemente, sobre estas aplicaciones. Los lectores que ya me conocen saben que soy partidario de usar, siempre que sea posible, programas *open source*, es decir, de tecnología abierta, frente a programas propietarios. Esto es así por varias razones. Principalmente, porque el software open source es gratis, y no es cuestión de que para estudiar el funcionamiento de un programa o un lenguaje haya que pagar por él. Por otra parte los programas de tecnología abierta son tan eficientes y seguros como los programas de pago y, en muchos casos, más. Aquí no podremos usar open source siempre que lo deseemos, pero lo usaremos donde podamos. Si no, recurriremos a versiones trial (uso de 30 días) de determinadas aplicaciones comerciales.

### 2.1.1 El servidor Apache

El programa que vamos a emplear como servidor de páginas web es el Apache (<http://www.apache.org>). Es el servidor web más extendido en Internet. La mayoría de los sitios que usted visita a diario están corriendo sobre este servidor. Las razones son varias. Por una parte, es gratuito. Usted no tiene que pagar nada para emplearlo. Por otra parte, es uno de los servidores más robustos que existen, muy seguro ante los ataques en la Red. Además, es extremadamente fácil de configurar. Los pocos datos de configuración que necesitaremos retocar están almacenados en un archivo de texto plano.

### 2.1.2 El intérprete de PHP 5

El lenguaje de programación objeto de este libro funciona mediante el uso de un intérprete. Cuando el usuario solicita una página desarrollada mediante esta tecnología, el intérprete de PHP (<http://www.php.net>), instalado en el equipo servidor, lee el código de dicha página y lo ejecuta. A continuación genera una

página HTML como resultado de la ejecución, y se la entrega al servidor web, quien se la envía al cliente. PHP es un lenguaje de scripts interpretados. En ese sentido el concepto es similar a JavaScript, dejando de lado el hecho de que este último se interpreta y ejecuta en el lado del cliente, y PHP lo hace en el lado del servidor. Por lo demás, la filosofía de trabajo es similar, aunque un lenguaje de servidor siempre ofrece más recursos y posibilidades que uno de cliente.

### 2.1.3 La base de datos MySQL

Dentro de las múltiples posibilidades que los sitios dinámicos ofrecen al usuario está la obtención de información almacenada en una base de datos en el servidor. Suponga que quiere escribir una página desde la cual el usuario pueda obtener los resultados de los eventos deportivos que se celebran en su ciudad. Usted tendrá una base de datos donde almacenará los resultados de encuentros de fútbol, partidos de tenis, combates de boxeo, carreras de caballos, etc. Su página dinámica recibirá la solicitud del cliente sobre un determinado deporte, buscará esta información en la base de datos y se la servirá al cliente. En el mercado existen diversos sistemas de bases de datos. Nosotros hemos elegido para este libro MySQL, por varias razones. Por una parte, es gratuito (aunque también existen licencias comerciales). Por otra parte, los datos se almacenan en archivos que tienen un tamaño mucho menor que otras bases de datos. Además, el motor de datos de MySQL es mucho más rápido, tanto grabando datos como localizándolos y recuperándolos, que el de otras bases de datos. Eso sin contar con que MySQL ofrece una gran seguridad sobre la integridad de los datos almacenados.

### 2.1.4 El servidor de correo

Es muy posible que sus páginas dinámicas incluyan la posibilidad de remitirle algún mensaje de correo electrónico a los usuarios. Por ejemplo, si usted desarrolla algún foro, o tablón de anuncios, en el que el usuario pueda incluir mensajes y otros usuarios puedan responderle con comentarios u otros mensajes. En ese caso, la página deberá poder mandarle al usuario un aviso, por correo electrónico, de que tiene nuevas respuestas a sus posts. Del mismo modo que para poder conectarnos a una página web es necesario un servidor web, para que se le manden al usuario mensajes es necesario un servidor específico. En este caso montaremos el Argo Mail Server.

### 2.1.5 El servidor FTP

El lenguaje PHP permite escribir código que gestione la transferencia de información, archivos, documentos, etc., mediante el protocolo FTP. Para ello hace falta que la plataforma servidora incluya un servidor de FTP, lo mismo que para

consultar páginas web hace falta un servidor HTTP. Nosotros usaremos el Cesar FTP ([www.acologic.com](http://www.acologic.com)), que es gratuito.

## 2.2 EL APPSERV

En primer lugar vamos a montar el servidor web, el intérprete de PHP y el motor de bases de datos de MySQL. Para ello vamos a contar con una aplicación muy interesante que se llama AppServ. Este programa está disponible en Internet de forma gratuita, en la dirección <http://www.appservnetwork.com>. Como alternativa, también puede usted descargar el programa de la página web de open source de sourceforge, en <http://prdownloads.sourceforge.net/appserv/appserv-win32-2.5.4a.exe?download>. Una vez descargada la ejecutamos y nos aparece la primera pantalla de bienvenida, con el típico mensaje de copyright. Pulsamos sobre el botón y nos encontramos una pantalla en la que se nos pide que seleccionemos el directorio en el que queremos instalar la aplicación. Por defecto aparece C:/Appserv. Deje este directorio y pulse . En la siguiente pantalla se nos pregunta qué tipo de instalación queremos hacer. Seleccione la opción "Custom" (personalizada), y pulse . Esto lo hacemos para asegurarnos de que vamos a instalar todas las aplicaciones que incluye el AppServ. Las vemos en la siguiente pantalla, como se aprecia en la figura 2.1.



Figura 2.1

Observe que en la lista aparecen las tres aplicaciones que hemos decidido instalar, y una cuarta, llamada phpMyAdmin. Más adelante veremos de qué se trata; nos será muy útil para nuestro trabajo con bases de datos MySQL. Asegúrese de que las cuatro casillas están marcadas antes de pulsar .

La siguiente pantalla es importante. Observe la figura 2.2. Aquí tenemos que definir algunos datos de configuración del servidor. Para empezar, estableceremos el nombre del servidor local que vamos a emplear. Ponga localhost, como en el ejemplo.

Como he mencionado anteriormente, este servidor corresponde a la dirección de bucle local (127.0.0.1). Esto es lo que nos va a permitir montar el servidor en nuestro propio ordenador. En el siguiente campo, donde se le pide la dirección de e-mail del Administrador (*Administrator's Email Address*), escriba webmaster@localhost.com, como en la imagen. Por último, en la casilla correspondiente al puerto deje el valor por defecto (80). Como dijimos anteriormente, éste es el puerto por defecto para las transmisiones en protocolo HTTP. Podría poner otro puerto, pero eso nos complicaría innecesariamente el uso diario de nuestro servidor. Más tarde hablaremos de esto.

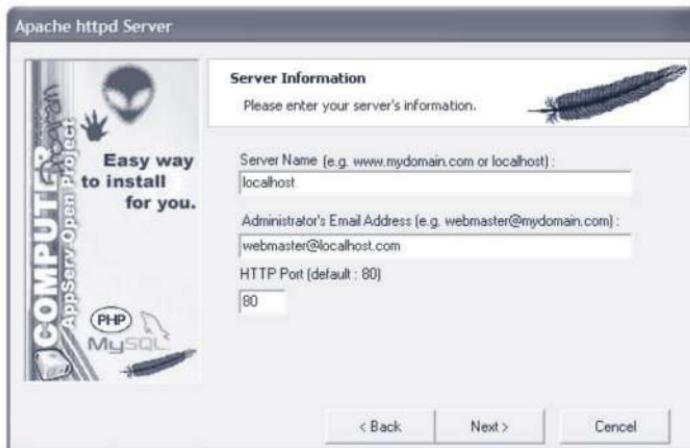
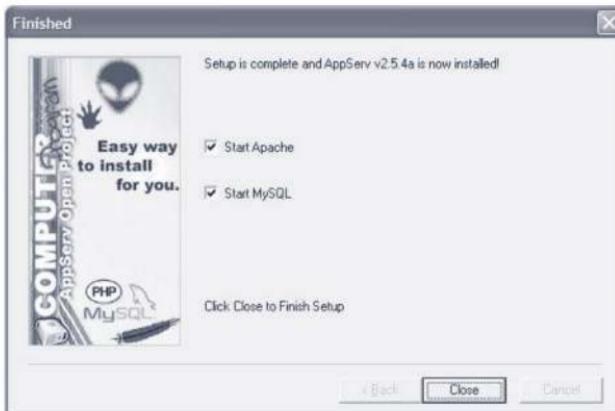


Figura 2.2

La siguiente pantalla corresponde a la instalación del motor de bases de datos de MySQL, tal como aparece en la figura 2.3.

*Figura 2.3*

Deje los campos tal como aparecen en la imagen. No se preocupe por no poner contraseña de acceso. De momento usaremos MySQL para aprendizaje. Más adelante, si lo desea, podrá ponerla. Cuando pulse **Next >** aparecerá una pantalla con una barra de progreso mientras se instalan las aplicaciones. A continuación debe aparecerle una ventana de MS-DOS negra, sin texto. Ciérrala y le aparecerá una pantalla como la que se muestra en la figura 2.4.

*Figura 2.4*

Deje las dos casillas marcadas. Son para que se inicie el servidor web Apache y el servidor de bases de datos MySQL. Al pulsar  le aparecerá una ventana MS-DOS como la de la figura 2.5.



```
C:\windows\System32\cmd.exe

#####
# Installing Apache as an Service #####
#####
# To remove Service please use : apache_serviceuninstall.bat #####
#####

#####
# Now Starting Apache... #####
#####

El servicio de Apache2 está iniciándose.
El servicio de Apache2 se ha iniciado con éxito.

Presione una tecla para continuar . . .
```

Figura 2.5

Si, en lugar de ello, le aparece una ventana negra, sin nada escrito, eso significa que el servidor Apache no se ha iniciado. No se preocupe. Esto puede ocurrir por razones de instalación. En ese caso tendremos que iniciararlo manualmente. Abra el ícono MiPC del escritorio. Entre en el disco duro local (normalmente es C:) y abra la carpeta AppServ, donde se ha realizado la instalación. Dentro de esta carpeta encontrará otras tres: una de ellas aloja el servidor Apache, otra el intérprete de PHP y la tercera contiene MySQL. Entre la carpeta del servidor Apache y encontrará un ícono como el de la figura 2.6.

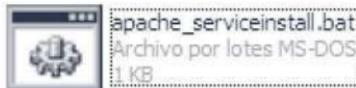


Figura 2.6

Haga doble clic sobre él. Con esto habrá iniciado el servidor Apache y le habrá quedado instalado como **servicio del sistema**, es decir, que se iniciará cada vez que ponga en marcha su equipo.

## 2.3 CONFIGURANDO EL SERVIDOR APACHE

En el servidor Apache es necesario establecer un par de parámetros de configuración, antes de empezar a trabajar. En primer lugar, todos los sitios

dinámicos que usted realice (tanto los ejercicios propuestos en este libro, como los sitios que usted cree por su cuenta) deberán estar alojados en una carpeta específicamente creada al efecto, que actuará como servidor. A su vez, cada sitio estará en su propia carpeta, dentro de la carpeta-servidor. Esto es lógico. Es una cuestión fundamental de organización. No vamos a dedicarnos a crear sitios web buscando una profesionalidad... y teniendo cada archivo perdido en cualquier parte del disco. En mi caso he creado una carpeta, dentro de "Mis documentos" llamada "Mis webs dinamicas". Dado que trabajo con Windows XP, la ruta completa de acceso queda así: "C:\Documents and Settings\Jose\Mis documentos\Mis webs dinamicas".

Una vez que haya usted determinado cuál va a ser su carpeta-servidor, abra de nuevo la carpeta de Apache, dentro de la carpeta AppServ. Dentro de la carpeta de Apache encontrará otra carpeta llamada "conf". Ábrala. Dentro hay un archivo llamado http.conf. Ábralo con cualquier editor de texto plano. No use un editor RTF tipo Word o similar. Emplee, por ejemplo, el bloc de notas de Windows. Una vez abierto, busque una línea que empieza con **DocumentRoot**. Hay una línea que empieza con #DocumentRoot. Esa no nos interesa. Las líneas que empiezan con # son comentarios. Buscamos la que empieza sin #. Una vez que la encuentre, incluya la ruta de la carpeta-servidor que usted ha elegido. De este modo, el servidor Apache sabrá dónde buscar las páginas dinámicas. En mi caso, la línea me queda así: **DocumentRoot "C:/Documents and Settings/Jose/Mis documentos/Mis webs dinamicas"**. Aunque aquí, en el texto, no cabe en una sola línea, usted en su archivo póngalo en una, no en dos.

A continuación busque, un poco más abajo, una línea que empieza por **<Directory**, y que incluye el directorio por defecto. Sustituya este directorio por la ruta de su carpeta-servidor como hizo en la otra línea. En mi caso, me queda tal y como se ve a continuación: **<Directory "C:/Documents and Settings/Jose/Mis documentos/Mis webs dinamicas">**.

Confirme que en las dos líneas ha separado los nombres de las carpetas mediante la barra / y no con \.

Entre las dos líneas aparece, aparte de algunos comentarios, el siguiente bloque de texto:

```
<Directory />
    Options FollowSymLinks ExecCGI Indexes
    AllowOverride None
</Directory>
```

Asegúrese de no modificar en absoluto este bloque. Ahora busque una línea que empieza con **ServerName**. Debe quedar como **ServerName localhost** para que todo vaya bien.

Grabe los cambios efectuados y cierre el editor de texto. Ahora reinicie el ordenador. De esta forma, Apache se iniciará por primera vez como servicio, con la nueva configuración.

## 2.4 COMPROBANDO EL FUNCIONAMIENTO

Ha llegado el momento de comprobar que todo vaya bien. Vamos a comprobar, en primer lugar, el funcionamiento del servidor Apache. Para ello abra la carpeta que ha creado como carpeta-servidor. En ella vamos a crear una página muy sencilla, cuyo nombre es **pruebaApache.htm**. El contenido es el siguiente:

```
<html>
  <body>
    Apache funcionando.
  </body>
</html>
```

Para abrir esta página, no lo haga con doble clic sobre su ícono. Esta página está en lo que será su servidor de pruebas. Abra el navegador. En la barra de direcciones teclee **http://localhost/pruebaApache.htm** y pulse Intro. A modo de aclaración, localhost es el nombre de su servidor local, tal como lo indicamos en el fichero de configuración de Apache, y se refiere a la carpeta-servidor. Esta notación obliga a que el navegador haga una llamada al servidor Apache y le solicite la página. Si todo va bien, su navegador debería mostrar un aspecto similar al de la figura 2.7.



Figura 2.7

Fíjese en que la llamada la hemos hecho por http, como cuando nos conectamos a cualquier página de Internet. Si, en lugar de ver la página que hemos

hecho, nos sale un aviso de que no se puede mostrar la página, se ha producido un error y Apache no ha podido servirle la página al cliente Internet Explorer. Los errores más comunes se producen por una mala configuración. Si a usted se le ha producido algún error, revise el fichero de configuración de Apache como lo detallé en el apartado anterior y reinicie su equipo. Asegúrese también de que la ruta de la carpeta-servidor sea la correcta.

Una vez que Apache está funcionando, es el momento de comprobar el funcionamiento del intérprete de PHP 5. Para ello, vuelva a abrir su carpeta-servidor. Dentro de ella crearemos una página, llamada **pruebaPHP.php**. El código es el siguiente:

```
<?php  
    phpinfo();  
?>
```

De momento no se preocupe de qué es “eso” que ha puesto ahí. Sólo asegúrese de que las tres líneas estén como aparecen en el texto, y de que la extensión del archivo sea php, y no htm como veníamos haciendo anteriormente. La función **phpinfo()** devuelve una gran cantidad de información referida al intérprete de PHP instalado. Ejecute la página tecleando, en la barra de direcciones de su navegador, lo siguiente: **http://localhost/pruebaPHP.php** y pulsando Intro. Si todo va bien, su navegador debe tener un aspecto similar a la figura 2.8. Recuerde que todas las páginas que escriba a partir de ahora que incluyan código PHP deberán ser almacenadas con la extensión .php, y no con htm.



Figura 2.8

No se preocupe ahora de qué significa todo lo que aparece en pantalla. Lo que importa es ver que el intérprete de PHP está funcionando correctamente. Sin embargo, si quiero llamar su atención sobre una cosa. Seleccione el menú *Ver* de la barra de menú de su navegador y elija *Código fuente*. Se le abrirá el editor de texto por defecto (normalmente el bloc de notas) con el código fuente de la página que se está visualizando. ¿De dónde ha salido todo ese código? El intérprete de PHP ha tomado el código de la página, tal como usted lo tecleó, lo ha interpretado y ejecutado, generando una página HTML, y se la ha “entregado” al servidor Apache. Éste, a su vez, ha servido la página al cliente Internet Explorer. Tal como le decía en el capítulo anterior, las páginas dinámicas se generan en el momento de la solicitud y, como usted acaba de ver, el código recibido por el cliente no tiene por qué parecerse al código original de la página.

Este modo de funcionamiento tiene una ventaja añadida. Cuando se ejecuten sus páginas dinámicas no se podrá deducir información clave de su código original a través del código que recibe el cliente, dándole un medio de ocultación de información delicada, tal como nombres de variables, claves de acceso a datos, etc. Esto proporciona mayor seguridad a la hora de evitar ataques a nuestra web por parte de personas malintencionadas. Como usted empieza a apreciar, el salto de páginas estáticas a dinámicas está lleno de ventajas. A lo largo del libro se irá dando cuenta de la potencia de este modo de trabajo.

## 2.5 EL SERVIDOR DE CORREO

Por último, vamos a montar un servidor de correo electrónico. Lo usaremos para enviar e-mails desde nuestras páginas dinámicas. PHP permite el envío de correos electrónicos pero, al igual que para mostrar una página es necesario un servidor que la dispense (el Apache, por ejemplo), para enviar un mensaje es necesario un servidor de correo. Para que todo vaya bien vamos a usar uno de los mejores servidores de correo que existen, además de ser gratuito: el Argo Mail Server. Lo primero que hacemos es descargarnos el instalador desde la web del fabricante (<http://www.agoisoft.com/files/apps/agsmtp.exe>). Una vez descargado, la instalación es muy sencilla: sólo es necesario seguir el asistente. Al activar el ícono vemos la primera pantalla de instalación, tal como en la figura 2.9.



Figura 2.9

Pulsamos el botón y se inicia el proceso de descompresión e instalación, con la ventana que vemos en la figura 2.10.

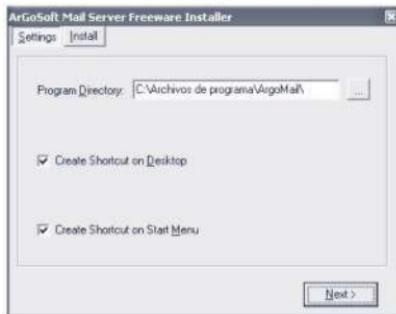


Figura 2.10

Con el botón elija el directorio en el que va a instalar la aplicación. También puede, si lo desea, teclear la ruta en el correspondiente campo de texto. Ponga la misma que aparece en la figura 2.11. Las casillas Create Shortcut on Desktop (Crear un acceso directo en el escritorio) y Create Shortcut on Start Menu (Crear un acceso directo en el menú de inicio) aparecen marcadas por defecto. Déjelas así. Pulse el botón y pasará a la siguiente pantalla de instalación, como se ve en la figura 2.11.

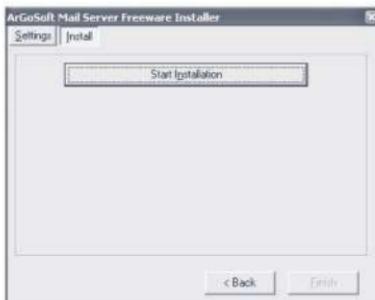


Figura 2.11

Pulse el botón y la instalación empezará automáticamente. En la pantalla le aparecerán los nombres de los ficheros que se

van copiando en su disco duro. Esta parte es automática y muy rápida. Tras unos segundos verá la ventana de la figura 2.12.



Figura 2.12

Pulse el botón **OK** y, a continuación, en la ventana principal de la instalación, pulse **Finish**. Con esto ha concluido la instalación del servidor de correo. Como ve, ha sido extremadamente simple y rápida. Sin embargo, para que este software sea operativo, aún hemos de configurarlo.

### 2.5.1 Configurando el servidor de correo

El servidor de correo es, como usted ya sabe, el programa que, obedeciendo las instrucciones que le demos desde PHP (o desde otro lenguaje, si a eso vamos) enviará mensajes de correo electrónico. La forma en la que opera es la siguiente. Cuando se ejecuta una instrucción de PHP que implica el envío de un correo, el servidor recopila todos los datos: destinatario, remitente, asunto y mensaje. Estos datos son almacenados en el servidor de destino. Por ejemplo, si enviamos un mensaje a **usuario@servidor.com** nuestro servidor de correo buscará en La Red el recurso **servidor.com** y enviará allí nuestro mensaje, a nombre de **usuario**. Como nosotros vamos a usar nuestro equipo único como servidor y cliente, el mensaje se "enviará" a nuestro propio ordenador. Por supuesto, para que todo funcione, la cuenta de correo debe existir. En nuestro servidor vamos a crear una cuenta que usaremos para nuestras pruebas. Se llamará, por supuesto, **pruebas@localhost.com**. Lo primero que hacemos es doble clic sobre el ícono que nos ha aparecido en el escritorio del servidor Argo. Es posible que, en ese momento, le aparezca una ventana de error como la de la figura 2.13.



Figura 2.13

No se preocupe por ella. Pulse el botón **OK**. La ventana de error se cerrará y, en la bandeja de iconos, junto al reloj, en la parte inferior derecha de su pantalla, le aparecerá un ícono similar al del programa, pero más pequeño. Al apoyar el ratón sobre él verá la leyenda **ArgoSoft Mail Server Freeware**. Haga doble clic sobre dicho ícono y obtendrá la pantalla principal del programa, que aparece en la figura 2.14.

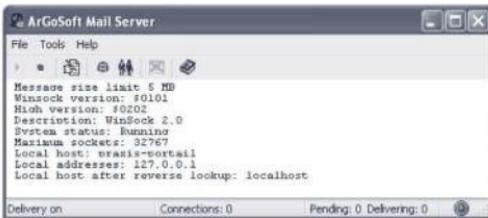


Figura 2.14

No se preocupe si el contenido de texto difiere un poco del que aparece en la imagen. Yo tengo mi propia configuración en mi equipo y usted la suya, que no tiene por qué coincidir.

Pulse el botón **Users**, o bien la opción **Tools** de la barra de menús y, en el menú que se despliega, elija **Users** **Ctrl+U**. También puede usar la combinación de teclas **ctrl.+U**. En cualquier caso se abrirá una ventana donde se pueden gestionar los usuarios (cuentas de correo). Esta ventana se llama **User Setup**. En principio está vacía, como se ve en la figura 2.15.

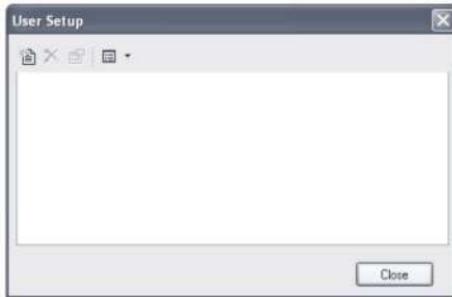


Figura 2.15

Pulse el botón  en la parte superior izquierda de esta ventana. Este botón se usa para crear una nueva cuenta. Se le abrirá la ventana **Add New User**, que aparece en la figura 2.16.



Figura 2.16

En el campo de texto **User Name:** teclee pruebas. En el campo **Real Name:** teclee lo que desee; yo he puesto Cuenta de prueba. En el campo **Password:** escriba la contraseña que desee; yo he grabado la palabra email. El campo **Confirm Password:** es para que escriba de nuevo la contraseña. Deje en blanco los campos Forward Address: y Return Address: y, a continuación, pulse el botón . La ventana **User Setup** queda con el aspecto de la figura 2.17, donde se aprecia el nuevo usuario que hemos creado.

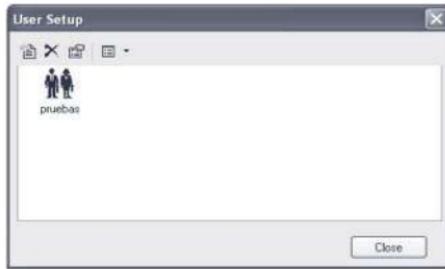


Figura 2.17

Pulse el botón . A continuación, vamos a establecer algunas opciones del dominio. Para ello pulsamos, en la ventana principal del programa que aparece en la figura 2.15, el botón . También podemos desplegar el menú **Tools** y seleccionar la opción **Options** . Además, podemos acceder a las opciones con la combinación de teclas **ctrl.+O**. En cualquier caso, obtendremos la ventana de la figura 2.18 (asegúrese de que está activa la pestaña **General**).



*Figura 2.18*

En el campo **Local Host**: debe teclear la dirección de bucle local (127.0.0.1) tal como aparece en la imagen. Las casillas de verificación también deben quedar como las está viendo.

A continuación seleccione la pestaña **Local Domains**. Verá una ventana parecida a la de la figura 2.20, aunque con todas las casillas en blanco. Escriba localhost.com en la casilla inferior, tal como aparece en la imagen, y pulse el botón . El nombre de servidor que acaba de teclear ha pasado a la casilla superior. Pulse el botón , con lo que esta ventana se cerrará. Es posible que le salga entonces una ventana de aviso como la de la figura 2.19.



*Figura 2.19*

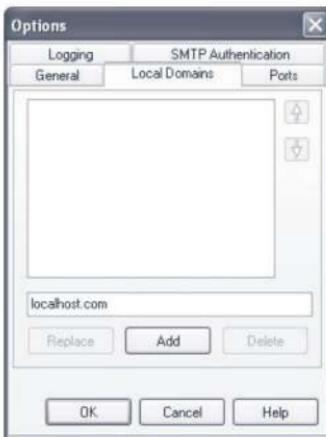


Figura 2.20

No le dé importancia. Ciérrala pulsando el botón . A continuación, cierre la ventana principal del servidor y reinicie su equipo.

Usted ya tiene su servidor de correo funcionando, y ha creado la cuenta `pruebas@localhost.com`, con la contraseña `email`. Si necesita crear más cuentas vuelva a la ventana **User Setup**. La ventana **Options** ya no necesita tocarla. Lo que sí necesita ahora es configurar su cliente de correo electrónico para poder recibir los mensajes que envíe durante las pruebas. Dada la gran variedad de clientes de correo que existen no podemos abarcarnos aquí todos. Recuerde configurar una cuenta con los datos de la que acaba de crear y ponga como servidor POP3, y como servidor SMTP, la dirección IP de bucle local (127.0.0.1). En el capítulo 13 del libro aprenderemos a enviar correos electrónicos desde PHP.

## 2.6 EL SERVIDOR FTP

El servidor FTP permite la transferencia de archivos mediante este popular protocolo. En el capítulo 14 del libro aprenderemos a escribir código PHP para llevar a cabo este tipo de transferencias. De momento vamos a instalar el software necesario. Para trabajar aquí vamos a usar el CesarFTP, que es freeware. Lo podemos descargar de la página del fabricante (<http://www.acologic.com>) o usar el instalador que hay en el CD. Al hacerle doble clic vemos que va a empezar la instalación. Pulsamos el botón  y vemos la pantalla de la figura 2.21.



Figura 2.21

Como ve, se trata de una instalación típica de una aplicación en entorno Windows. Pulse el botón  y pasará a la siguiente pantalla, que aparece representada en la figura 2.22.



Figura 2.22

De nuevo, pulse el botón **Next >** y verá la pantalla que aparece en la figura 2.23.



Figura 2.23

Esta pantalla es importante, porque contiene el acuerdo de licencia de uso del programa. Recuerde que, cada vez que usted instala una aplicación en su equipo, está aceptando, implícitamente, los términos de la licencia establecidos por el fabricante del mismo. En este caso, el acuerdo nos es favorable porque se trata de una licencia de libre uso y distribución.

Para continuar la instalación pulse el botón **Yes** para indicar que acepta la licencia y pasará a la siguiente pantalla, que aparece en la figura 2.24.

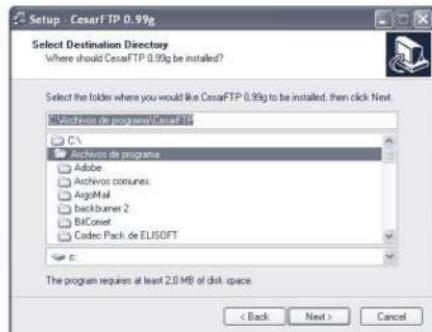


Figura 2.24

Aquí se le ofrece la posibilidad de cambiar el directorio de la instalación. Deje el que el programa le ofrece por defecto y pulse **Next >** para continuar. La siguiente pantalla aparece en la figura 2.25.



Figura 2.25

Aquí tiene la opción de establecer el grupo de programas donde aparecerá luego el ícono de CesarFTP en el menú de inicio. Asegúrese de que la casilla que aparece con el rótulo  **Don't create a Start Menu folder** no está marcada y pulse el botón **Next >** para continuar. Pasará a la pantalla de la figura 2.26.

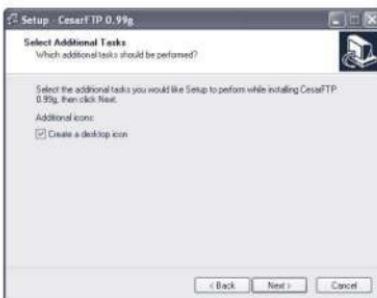


Figura 2.26

Asegúrese de que la casilla  **Create a desktop icon** está activada, como aparece en la imagen. Esto creará un acceso directo al programa en el escritorio. Pulse el botón **Next >** y pasará a la siguiente pantalla, como ve en la figura 2.27.

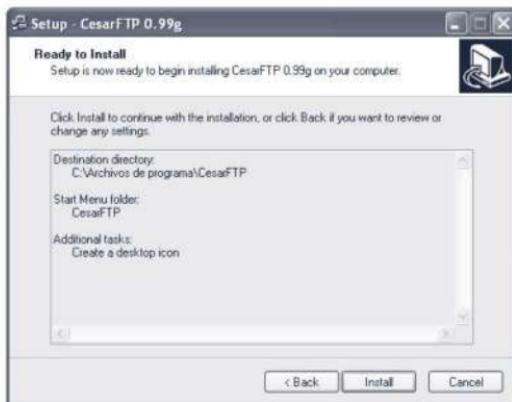


Figura 2.27

Aquí se le informa de que el instalador ya tiene toda la información necesaria para proceder. Pulse el botón **Install** y pasará a la pantalla de instalación, como se ve en la figura 2.28.



Figura 2.28

Una vez concluida la instalación veremos la pantalla de la figura 2.29.



Figura 2.29

Deje marcada la casilla  Launch CesarFTP y pulse el botón **Finish**. Con esto habrá dejado el programa instalado y ejecutándose.

### 2.6.1 Configurando el servidor FTP

Tan importante como instalar el servidor es configurarlo adecuadamente. Tal como hicimos con el servidor de correo vamos a ver aquí cómo ponerlo a punto. En este momento debería tener a la vista la pantalla principal del programa. Si no es así, haga doble clic en el ícono que se le ha creado en el escritorio. La pantalla ofrece el aspecto de la figura 2.30.

Vamos a configurar el servidor para que esté disponible en modo local (localhost), a fin de poder probar los ejercicios que veremos en el capítulo 14. Para conectarse por FTP a un servidor de este protocolo es necesario ser un usuario registrado. Nosotros vamos a crear ahora un usuario, con su nombre y contraseña. Estos datos nos serán necesarios después, así que vamos a introducirlos ahora. Para ello tenemos tres posibilidades:

- Podemos pulsar el botón de la barra superior de herramientas.
- Podemos acceder al menú **Settings** y elegir la segunda opción, cuyo aspecto es el siguiente: **Edit Users & Groups** **Ctrl+U**.
- Por último, podemos pulsar la combinación de teclas **Ctrl+U**.

Da lo mismo como lo hagamos, cualquiera de los tres métodos expuestos nos conducirá a la pantalla adecuada.

La pantalla de Edición de Usuarios tiene, inicialmente (cuando todavía no se ha creado ningún usuario), el aspecto de la figura 2.31.

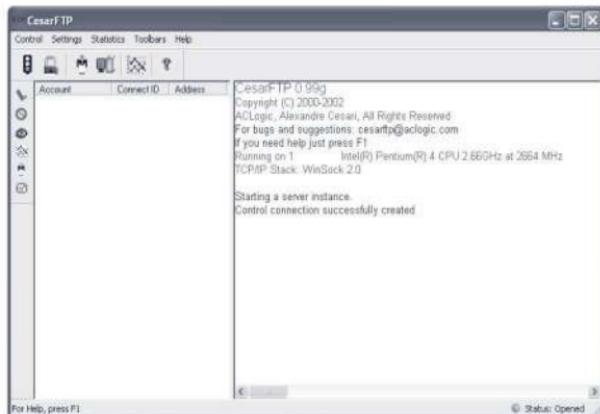


Figura 2.30

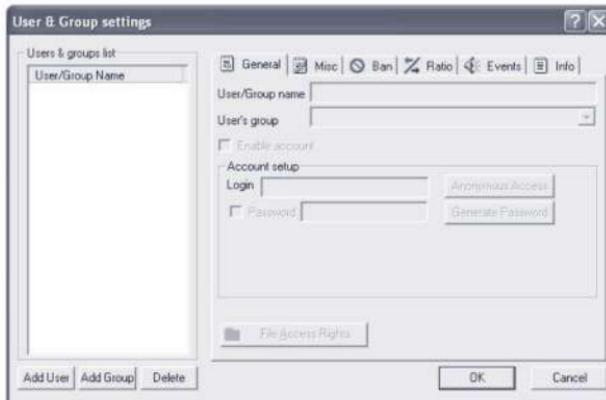


Figura 2.31

Pulse el botón **Add User** que aparece en la esquina inferior izquierda. En la zona blanca que tiene a la izquierda le aparecerá la leyenda **New User**. Además, las casillas **Login** y **Password**, que estaban inhabilitadas aparecen ahora listas para su uso (compruebe que la casilla de verificación a la izquierda de la etiqueta **Password** esté activada. Si no lo está, activela). Teelee un nombre de usuario y una contraseña de acceso. Yo he puesto los datos que ve en la figura 2.32.



Figura 2.32

Usted ponga otros datos cualesquiera y anótelos, para que no se le olviden. No obstante, siempre podrá volver a esta pantalla si lo necesita. Una vez grabados los datos, pulse el botón **OK**, y la creación de un nuevo usuario habrá concluido. La pantalla de Edición de usuarios se cerrará y le quedará la pantalla principal del programa. Pulse el botón “Minimizar”, con lo que el servidor pasará a trabajar en segundo plano. Parece que se ha cerrado, porque no tiene un ícono en la barra de tareas, pero usted sabrá que está activo porque el ícono de CesarFTP aparece en la bandeja de iconos, junto al reloj de Windows, en la parte inferior derecha de su pantalla.

## 2.7 CONSIDERACIONES FINALES

Ya tiene todos los servicios necesarios operando en su ordenador. Ahora ya puede probar todos los ejercicios del libro, según vaya avanzando en su lectura. También puede desarrollar y probar todas las páginas web que usted mismo quiera crear. Cuando usted lleve a cabo un sitio profesional deberá, lógicamente, publicarlo en un servidor que implemente todas las tecnologías que hemos visto en este capítulo. Desde mi experiencia personal le sugiero que contrate sus nombres de dominio y los alojamientos con AMEN (<http://www.amen.es>). No sólo por la excelente relación calidad-precio que ofrecen, sino también por la inmejorable asistencia técnica de que disponen para sus clientes.

## PRIMEROS PASOS EN PHP

---

---

En este capítulo vamos a afrontar los conceptos básicos de la programación en el lado del servidor mediante PHP. Aprenderemos lo más básico para el manejo de datos, pares nombre-valor, manejo de matrices y otros fundamentos.

### 3.1 MI PRIMERA PÁGINA DINÁMICA

Vamos a crear nuestra primera página dinámica. Para ello usaremos un editor de texto plano, como el bloc de notas. No emplee ningún editor de texto RTF (tipo Word de Microsoft), ya que estos editores añaden al texto los códigos de formato y no son adecuados para la programación. Más adelante, cuando usted ya conozca la programación en PHP podrá usar otras herramientas más avanzadas que le facilitarán su trabajo. Abra el editor y copie el siguiente código (lo tiene en el CD adjunto, en la carpeta **capítulo\_03**, con el nombre **primeraPagina.php**).

```
<html>
  <body>
    <?php
      echo ("Mi primera página en PHP.");
    ?>
  </body>
</html>
```

Grabe este código en una carpeta dentro de la carpeta de su disco duro que tiene definida como raíz de documentos en la configuración del servidor Apache (vea, en el capítulo anterior, el apartado 2.3 que habla de la configuración del servidor Apache). En mi ordenador yo tengo la siguiente línea:

DocumentRoot "C:/Documents and Settings/Jose/Mis documentos/Mis webs dinamicas".

Así pues, copio la carpeta **capítulo\_03** dentro de la carpeta **Mis webs dinamicas**.

Fíjese en el código. Está acotado entre las etiquetas <html> y </html>. Dado que, después de todo, es un documento web, se va a mostrar en un navegador. Y los navegadores interpretan código HTML, no PHP. Ahora mire que el código PHP para el servidor está comprendido entre <?php y ?>. Estas *etiquetas* se usan para delimitar el comienzo y el final de las instrucciones que se deben ejecutar en el lado del servidor.

Vamos a ejecutar la página. Para ello, abrimos el navegador y tecleamos, en la barra de direcciones, lo siguiente:

**http://localhost/capítulo\_03/primeraPagina.php**

Esta forma de abrir una página alojada en su equipo local quizás le resulte un poco chocante. Si usted ha estado trabajando hasta ahora con páginas estáticas está acostumbrado a abrirlas haciendo doble clic sobre el ícono. El navegador se abre y carga la página solicitada. Esto ya no nos sirve. Recuerde que una página con código dinámico se interpreta en el servidor y es éste el que envía el resultado al navegador (enseguida veremos ese proceso). Así pues, usted lo que tiene es una *red* (aunque esté toda ella alojada en su equipo local). Le ruego que me disculpe por insistir en este concepto, pero es de vital importancia para comprender la naturaleza de lo que estamos haciendo.

Vamos a analizar, por tanto, lo que hemos tecleado en la barra de direcciones. En primer lugar hemos tecleado **http://**, tal como ocurre cuando nos conectamos a Internet y llamamos a una página que está situada en un servidor remoto. Esta secuencia le está diciendo al navegador que nos vamos a conectar a una página que tiene que transmitirse, desde el servidor correspondiente, mediante el protocolo de transferencia de hiper-texto (**Hyper Text Transfer Protocol**). Si recuerda lo que vimos en el capítulo 1, éste es parte de la pila de protocolos TCP/IP y es el protocolo normalizado para la transferencia de documentos web entre el servidor (en este caso, Apache) y el cliente (su navegador web).

A continuación encontramos la palabra **localhost**. Cuando usted se conecta a Internet, en este punto teclea **www.**, lo que le indica al navegador que el documento que quiere cargar está en la Red de Redes (**World Wide Web**). Entonces el navegador ya sabe que tendrá que conectarse a los DNS para obtener la IP del documento solicitado. Al teclear localhost, le estamos diciendo al navegador

que debe conectarse al servidor que tiene instalado en modo local (el Apache que instalamos en el capítulo 2). El navegador no necesita, por tanto, recurrir a los DNS; ya “sabe” que la IP de localhost es 127.0.0.1. Realmente quien “sabe” esto es el sistema operativo de su ordenador, pero este último detalle no cambia nada respecto al funcionamiento. Además, como la carpeta donde tenemos nuestras páginas web dinámicas está especificada en el parámetro DocumentRoot del archivo de configuración de Apache, el servidor ya sabe dónde tendrá que buscar lo que le solicite el cliente.

A continuación, en la barra de direcciones tecleamos **/capítulo\_03**. Esta es la subcarpeta (dentro de la carpeta que actúa como localhost) donde se encuentra el documento que nos interesa.

Si no se le especifica ningún nombre de documento, el servidor buscará, por defecto, **index.html** o **index.php**. Si en la barra de direcciones sólo tecleamos **http://localhost/capítulo\_03**, el servidor buscará los documentos por defecto. En este caso no existe ningún documento en la carpeta referenciada que se llame index, así que tenemos que especificar el nombre del documento que queremos cargar, en este caso es **/primeraPagina.php** (la barra inclinada, conocida como **slash**, se usa para separar los nombres de las carpetas, y el nombre del documento del nombre de la última carpeta). A continuación pulse la tecla INTRO o pulse el botón . Su navegador quedará con el aspecto que ve en la figura 3.1.



Figura 3.1

Haga clic en **Ver** y, a continuación en **Código fuente** para ver el código fuente de la página. Lo que se obtiene al abrir el editor de texto predeterminado por el navegador es parecido al código que usted tecleó, pero no exactamente igual:

```
<html>
  <body>
    Mi primera página en PHP.
  </body>
</html>
```

Veamos qué ha ocurrido. Cuando el servidor lee la página y se encuentra con etiquetas HTML, tales como <html> o <body>, no las procesa de ningún modo, sino que se limita a enviarlas, tal como están, al navegador. Así mismo, tampoco efectúa ningún proceso específico con el contenido de estas etiquetas. De igual modo no lleva a cabo ninguna acción con un código de script que deba ser interpretado y ejecutado en el lado del cliente como, por ejemplo, JavaScript. Pero cuando encuentra código delimitado por <?php y ?> la cosa cambia. El servidor "sabe" que ese es un código que debe pasarle al intérprete de PHP. Éste lo recibe, lo interpreta, obtiene un resultado y se lo devuelve al servidor quien, a su vez, lo pasa al navegador. Al principio puede que este proceso le parezca un poco complejo, pero es más sencillo de lo que parece, sobre todo porque es totalmente transparente al usuario. Cuando solicitamos al navegador una página dinámica éste nos la muestra, y ya está. El usuario no tiene por qué saber lo que ocurre "al otro lado de los cables".

En nuestro ejemplo tenemos una única instrucción de PHP:

```
echo ("Mi primera página en PHP.");
```

La función *echo ()* se emplea para mostrar información en pantalla. La información que queramos mostrar debe figurar como argumento de la función (entre los paréntesis, que son preceptivos). La instrucción, como la mayoría de las instrucciones en PHP, acaba en *punto y coma*. Dentro de un momento volveremos sobre este particular. El argumento de la función es una cadena de texto que debe ser volcada en pantalla tal cual, por lo que la hemos encerrado entre comillas. Las comillas delimitan la cadena, pero no aparecerán en pantalla, porque no forman parte de ella.

El intérprete PHP toma la instrucción y genera un código HTML que muestra el texto en pantalla. Cuando, en HTML, queremos que aparezca un texto en la página, simplemente lo incluimos dentro de la misma. Pues eso es, exactamente, lo que ha hecho el intérprete de PHP.

Por último, la página HTML generada es enviada, por el servidor, al navegador, y eso es lo que obtenemos al ver el código fuente: un documento HTML creado a partir del código PHP. Pienselo un momento. Empleando esta técnica tendremos la ventaja de que los usuarios no tendrán acceso al código fuente de nuestras páginas, sino sólo al resultado final del mismo. No es, desde luego, el mejor modo de ocultar nuestro trabajo, pero es un comienzo.

Decía hace un momento que las instrucciones de PHP deben acabar en punto y coma (;). En el caso del ejemplo que hemos visto no tiene demasiada importancia, porque sólo hay una instrucción PHP. Cuando el intérprete ve el final

del código, sabe que la instrucción ha terminado y no pasa nada. Pero ahora suponga el listado **error.php** que aparece a continuación. Fíjese en que he “olvidado” (esta vez a propósito) el punto y coma al final de la primera instrucción.

```
<html>
    <body>
        <?php
            echo ("Mi primera página en PHP.");
            echo ("En realidad es la segunda.");
        ?>
    </body>
</html>
```

Al tratar de ejecutar esta página obtenemos el resultado de la figura 3.2.

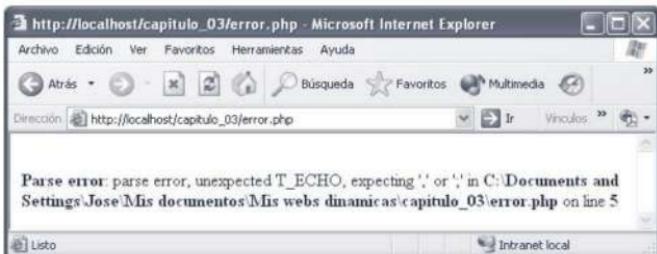


Figura 3.2

Como ve, no se ejecuta la página, sino que se desencadena un error. Este es el más común de los errores que se cometen al transcribir código PHP. Afortunadamente es fácil de localizar y solventar.

A la hora de incluir código PHP en nuestras páginas dinámicas podemos hacerlo de tres maneras diferentes. Yo lo he incluido entre las etiquetas **<?php** y **?>**. Éste es el modo elegido por la mayoría de los webmasters. Es el que yo siempre uso y el que usaremos a lo largo de todo el libro. Existe un modo abreviado, usando las etiquetas **<?** y **?>**. Sin embargo, para que este sistema funcione es necesario activar una directiva en el fichero de configuración del intérprete. Para saber lo que es este fichero y conocer las principales directivas puede consultar el apéndice A, al final del libro. Por último, existen las etiquetas de script, para delimitar el código PHP, colocándolo entre **<script language="php">** y **</script>**. No me gustan, porque al revisar el código pueden inducir a error, pues se parecen a las etiquetas de JavaScript.

## 3.2 INICIACIÓN A LAS VARIABLES

Las variables, también llamadas *pares nombre-valor*, desempeñan en PHP un papel similar al que tienen en otros lenguajes de programación, tales como JavaScript: el almacenamiento temporal, en memoria, de datos que, en principio, se usarán para efectuar cálculos o para ser devueltos al usuario como resultados. Las variables en PHP no necesitan, a diferencia de otros lenguajes, ser declaradas específicamente. Cuando se inicializa una variable, es decir, cuando se le asigna su primer valor, se inicializa de forma automática. Esto quiere decir que el intérprete determina qué cantidad de memoria será necesaria para dicha variable, según el valor que se le haya asignado, y la reserva. En PHP los nombres de las variables van precedidos por el símbolo del dólar (\$). Así pues, la forma correcta de asignarle a un nombre de variable un valor es escribir el signo del dólar, el nombre de la variable, el *operador de asignación*, el valor que se le asigna a esa variable en ese momento y el punto y coma, así:

```
$nombreDeVariable = valor;
```

Por ejemplo, las siguientes asignaciones serían perfectamente válidas:

```
$edad = 38;
$nombre = "José";
$clave = "miClaveDeAcceso";
```

Observe que a la izquierda del operador de asignación aparece el nombre de la variable, precedido, como es preceptivo, por el guarismo \$. A la derecha del operador aparece el valor que se le asigna a la variable. En el primer ejemplo se trata de un valor numérico, que aparece tal cual. En los otros dos ejemplos es una cadena alfanumérica, que debe teclearse entre comillas. Las comillas se usan para delimitar cadenas alfanuméricas, pero no forman parte de estas, de modo que, al recuperar el valor de las dos últimas variables, las cadenas aparecerán sin las comillas. Para recuperar el valor de una variable podemos incluir su nombre como argumento de la función echo () que vimos en el apartado anterior. Observe el listado **mostrarDatos.php**:

```
<html>
<body>
<?php

$edad = 38;
$nombre = "José";
$clave = "miClaveDeAcceso";
echo ($edad);
```

```
?>
<br />
<?php
    echo ($nombre);
?>
<br />
<?php
    echo ($clave);
?>
</body>
</html>
```

El resultado de cargar esta página desde el navegador es el que aparece en la figura 3.3.



Figura 3.3

Observando una vez más el código que se ha cargado en el navegador, encontramos lo siguiente:

```
<html>
    <body>
        38          <br />
        José        <br />
        miClaveDeAcceso
    </body>
</html>
```

Así vemos, de nuevo, que lo que el navegador recibe, en última instancia, no es más que código HTML.

Nuestro código funciona como esperábamos. Crea unas variables, les asigna unos valores y, mediante la función echo (), devuelve esos valores para que sean mostrados en la página. Pero observe que es un código bastante engorroso. En

el listado de la página hemos incluido tres bloques de PHP para realizar algo muy sencillo. Si esto es así con esta página, uno podría pensar que para un resultado más sofisticado y útil necesitaríamos muchísimos bloques de PHP. Y todo porque entre un resultado y otro hemos necesitado intercalar el tag <br /> de HTML para que los datos nos aparezcan en tres líneas en la página. Parece que eso de intercalar código HTML en una página dinámica es un engorro. Nos obliga a estar abriendo y cerrando el código PHP a cada momento.

Los diseñadores de PHP captaron esta cuestión enseguida. Después de todo, si PHP se creó para programar páginas web, y el navegador sólo entiende los lenguajes de cliente, como son HTML y JavaScript, debía de haber un buen nivel de integración entre éstos y aquél. En realidad, dicho nivel es mucho más alto de lo que vamos a ver aquí, tal como usted comprobará según avance en el estudio de la programación dinámica. De momento, baste saber que PHP permite incluir etiquetas de HTML como si fueran cadenas. Al crear el código que se va a enviar al navegador, el intérprete inserta estas etiquetas de la forma apropiada. Así pues, el ejemplo anterior podría quedar como se ve en el listado **mostrarBR.php** a continuación:

```
<html>
  <body>
    <?php
      $edad = 38;
      $nombre = "José";
      $clave = "miClaveDeAcceso";
      echo ($edad);
      echo ("<br />");
      echo ($nombre);
      echo ("<br />");
      echo ($clave);
    ?>
  </body>
</html>
```

El resultado es el mismo, pero con menos líneas de código. Y, ya que estamos con eso de *optimizar el código* (es decir, obtener el resultado deseado con un código lo más breve y claro posible), hablemos del *operador de concatenación*. A la hora de asignar o mostrar resultados podemos usar un punto (.) para que PHP ponga un resultado a continuación de otro. Lo verá más claro en el ejemplo del script **encadenaBR.php**, que aparece a continuación:

```
<html>
  <body>
    <?php
```

```
$edad = 38;  
$nombre = "José";  
$clave = "miClaveDeAcceso";  
echo ($edad."<br />");  
echo ($nombre."<br />");  
echo ($clave);  
?  
</body>  
</html>
```

Ejecute esta página y vea que el resultado es el mismo. Si a eso vamos, podríamos haber puesto las tres salidas en una sola línea, así:

```
echo ($edad."<br />".$nombre."<br />".$clave);
```

Como ve, mediante el operador de concatenación se puede optimizar bastante el código. En realidad el límite de cuántos valores podemos concatenar viene dado por la claridad del código. Si formamos una serie de valores demasiado larga el código resultará menos legible.

### 3.2.1 Los nombres de las variables

Hemos visto que para crear una variable es necesario asignarle un nombre por el que luego será identificada cuando la necesitemos (en breve empezaremos a ver qué podemos hacer con las variables). Este nombre debe ir siempre precedido del signo \$. Es un error muy frecuente olvidar este signo, sobre todo si usted está habituado a otros lenguajes de script, como JavaScript, que no lo emplean. Preste atención a este signo siempre.

Además, es necesario tener otros detalles en cuenta. Por ejemplo, los nombres de las variables son sensibles al uso de mayúsculas y minúsculas. Observe el script **mayusVariables.php**, que le muestra cómo dos variables, cuyo nombre sólo difiere en el tipo de letra, pueden almacenar dos valores distintos. El uso indiscriminado de mayúsculas y minúsculas siempre acaba dando problemas.

```
<html>  
  <body>  
    <?php  
      $nombre = "José";  
      $Nombre = "Laura";  
      echo ($nombre. "<br />");  
      echo ($Nombre);  
    ?>  
  </body>  
</html>
```

Al ejecutarlo verá que cada una de las variables almacena y conserva un valor diferente. Sin embargo, no es buena práctica crear nombres de diversas variables que sean iguales excepto por la capitalización de las letras empleadas, ya que induce a confusión.

Otro aspecto que debe tener en cuenta es que no se deben usar palabras reservadas (nombres de instrucciones, funciones, etc.) como nombres de variables. En realidad esto no es ningún problema si es usted hispano parlante, ya que los nombres que les dé a sus variables serán palabras en español, o adaptaciones de éstas, y las palabras reservadas son todas del idioma inglés. Cabe suponer que para un anglo parlante esto debe ser más complicado. No obstante, en el apéndice B tiene una lista de todas las palabras que no puede emplear como nombres de variables.

NOMBRE VARIABLE	DE	EVALUACIÓN
\$miNombre		<b>CORRECTO.</b>
\$mi nombre		<b>INCORRECTO.</b> Incluye un espacio en el nombre.
miNombre		<b>INCORRECTO.</b> No comienza con el signo \$.
\$nombre_de_4_usuarios		<b>CORRECTO.</b>
\$4usuarios		<b>INCORRECTO.</b> Empieza con un número.
\$los.nombres.de.todos		<b>INCORRECTO.</b> Incluye signos de puntuación.
\$nombreDeMiNiña		<b>INCORRECTO.</b> Incluye la letra ñ.
\$echo		<b>INCORRECTO.</b> Es una palabra reservada.
\$variable_1		<b>CORRECTO DESDE EL PUNTO DE VISTA SINTÁCTICO, PERO DESACONSEJABLE</b> puesto que no da idea del uso que tendrá la variable.

Figura 3.4

A la hora de crear una variable no tiene mucho sentido asignarle un nombre como \$variable\_1, o similar. A menos que posea usted una memoria prodigiosa, es casi preceptivo asignarles nombres lógicos que nos den una referencia acerca de la utilización que se les va a dar a esas variables en el código. Si usted observa los listados que hemos visto en este capítulo, y los que veremos en todo el libro, comprobará que cada variable tiene asignado un nombre coherente con su uso. Cuando los nombres de las variables estén formados por dos o más palabras, se suele poner la primera en minúsculas y la segunda y siguientes con la primera letra mayúscula, así: \$miNombre, \$nombreDelUsuarioPrincipal, etc. Ésta es una práctica muy extendida en todo el mundo y es la que yo empleo siempre. Como alternativa, algunos programadores separan las palabras mediante un guion

bajo ( ) y las ponen todas en minúsculas, así: \$mi\_nombre, \$nombre\_del\_usuario, etc. Elegir un sistema u otro es una cuestión de criterio personal de cada uno. En todo caso, no separe las palabras mediante espacios en blanco, *que no deben aparecer nunca* en el nombre de una variable.

En los nombres de variables se pueden usar, indistintamente, letras y números, *siempre que se comience el nombre con, al menos, una letra a continuación del signo \$*, ya sea ésta mayúscula o minúscula. Tampoco debe incluir en los nombres de variables ningún signo de puntuación, ni letras acentuadas, ni letras de alfabetos locales, tales como la ñ o la ç. Sólo letras del alfabeto inglés.

La tabla de la figura 3.4 recoge distintos nombres de variables, a modo de ejemplo, y aclara cuáles de ellos son incorrectos y por qué.

### 3.3 GESTIÓN BÁSICA DE DATOS

Dentro de las variables se almacenan datos. Éstos pueden ser de varios tipos. En principio, vamos a manejar datos numéricos, que se pueden emplear para llevar a cabo operaciones aritméticas y/o lógicas, y datos alfanuméricos, o de cadena. Además, podemos emplear matrices y objetos (hablaremos de éstos en posteriores capítulos).

Cuando a una variable se le asigna un valor numérico, éste se pone, tal cual, a la derecha del operador de asignación, tal como se ve en los ejemplos anteriores y en la línea siguiente:

```
$edad=38;
```

Con las variables que contienen valores numéricos se pueden hacer operaciones aritméticas, desde las más simples a las más complejas. El script **operacionesBasicas.php** nos muestra unos breves ejemplos sencillos:

```
<?php
$operando1=8;
$operando2=3;
$suma=$operando1+$operando2;
$resta=$operando1+$operando2;
$producto=$operando1*$operando2;
$cociente=$operando1/$operando2;
$modulo=$operando1%$operando2;
echo ("Valor 1: ".$operando1."<br />");
echo ("Valor 2: ".$operando2."<br />");
```

```

echo ("Suma: ".$suma."<br />");  

echo ("Resta: ".$resta."<br />");  

echo ("Producto: ".$producto."<br />");  

echo ("Cociente: ".$cociente."<br />");  

echo ("Resto de la división: ".$modulo."<br />");  

echo ("Incremento de operando1:  

".++$operando1."<br />");  

echo ("Decremento de operando2: ".--$operando2);  

?>

```

Observe los resultados en la figura 3.5.

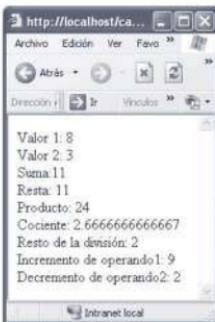


Figura 3.5

Con la mayoría de las operaciones que hemos realizado en este código estará usted familiarizado, ya que son operaciones que realizamos, a nivel doméstico, con una calculadora de bolsillo, todos los días (sumas, restas, divisiones, etc.). Si quiero puntualizar un par de detalles. El operador de la multiplicación es el asterisco, y el de la división es la barra inclinada a la derecha, conocida como *slash*.

Hay, no obstante, un par de operadores con los que, quizás, no esté usted familiarizado. Observe las dos últimas líneas del código y del resultado. Se trata de los operadores de **incremento** (++) y de **decremento** (--). Estos operadores se aplican sobre una variable para incrementarla o decrementarla, respectivamente, en una unidad. A su vez cada uno de estos operadores puede aplicarse antes del valor original o después del mismo, conociéndose como operador de **pre-incremento** o **pre-decremento** (si es aplicado antes) o de **post-incremento** o **post-decremento** (si es aplicado después). El comportamiento es diferente en ambos casos. Vamos a aclarar esto. Suponga un fragmento de código como el siguiente:

```
$valorInicial = 7;  
$valorFinal = ++$valorInicial;
```

La variable \$valorInicial es declarada con un valor de 7 en la primera línea. En la segunda, se realiza un pre-incremento. Esto quiere decir que primero se incrementa \$valorInicial en una unidad, con lo que pasará a valer 8. Después se realiza la asignación a \$valorFinal, con lo que esta valdrá, también, 8. Ahora vea estas líneas:

```
$valorInicial = 7;  
$valorFinal = $valorInicial++;
```

La primera línea funciona igual que en el caso anterior, declarando una variable y su valor. La diferencia está en la segunda línea. En este caso hemos empleado el operador de post-incremento. Lo que ocurre es que primero se realiza la asignación, con lo que \$valorFinal pasará a contener el valor 7, y después se lleva a cabo el incremento, con lo que \$valorInicial pasará a valer 8, pero *ya no se modifica \$valorFinal*, que sigue conservando el 7. Con los operadores de decrecimiento el funcionamiento es idéntico, sólo que, en lugar de añadir una unidad, se resta. Como ve, estos operadores son muy cómodos de usar, cuando queremos añadir o restar una unidad a un valor. Pero veamos cómo hacerlo para añadir o restar mayores cantidades. Por ejemplo, suponga que usted quiere incrementar un valor en tres unidades. Puede hacer lo siguiente:

```
$variable=$variable+3;
```

Desde el punto de vista matemático esta sentencia puede parecer un despropósito, pero no lo es en informática. No olvide que el operador = es de asignación. Así pues, esta sentencia dice: “*Toma el valor almacenado en \$variable, incrementale tres unidades y coloca el nuevo valor en \$variable, de forma que, tras ejecutar la sentencia, valga tres unidades más de lo que valía antes*”. Esto mismo puede hacerse de forma simplificada, así:

```
$variable += 3;
```

Siempre que vaya a incrementar una variable en varias unidades puede usar la **notación simplificada** que acabamos de exponer. Esta notación también puede emplearse para otras operaciones comunes, tal como se muestra en los siguientes ejemplos:

```
$variable -= 3;  
$variable *= 5;  
$variable /= 4;  
$variable %= 2;
```

Además, los valores numéricos permiten otras operaciones de las que hablaremos en el capítulo 6. Con las cadenas podemos llevar a cabo varias acciones, tales como extraer subcadenas, cambiar la capitalización, etc. De momento quiero ampliar un concepto al que ya nos hemos asomado: la concatenación mediante el operador punto (.). Se pueden concatenar cadenas tanto en la asignación de variables, como en la recuperación de los valores por pantalla. Es decir, los siguientes fragmentos de código funcionan de igual forma:

```
$nombre="José ";
$apellido="López";
echo ($nombre.$apellido);
```

El resultado es el mismo que si escribimos:

```
$nombre="José ";
$apellido="López";
$nombreCompleto=$nombre.$apellido;
echo ($nombreCompleto);
```

Quiero insistir (por última vez) en que los valores que son cadenas alfanuméricas deben ir limitados por comillas. Estas comillas no saldrán luego al recuperar el valor. Usted puede usar comillas simples o dobles, a su elección, siempre que la comilla del principio de la cadena sea del mismo tipo que la comilla del final de la misma. Sin embargo, le aconsejo que se acostumbre a acotar las cadenas con comillas dobles. Más adelante entenderá por qué. De momento, sólo hágame caso.

Los tipos de datos que contempla PHP son los siguientes:

- **integer.** Son datos de tipo numérico entero.
- **double.** Son datos numéricos en coma flotante (manejo de decimales).
- **string.** Son cadenas alfanuméricas.
- **boolean.** Son valores de tipo Verdadero o Falso. Se obtienen al evaluar una condición, el estado de una variable, etc.
- **array.** Son matrices (hablaremos de ellas enseguida).
- **object.** Los objetos son estructuras complejas de las que hablaremos más adelante.
- **class.** Las clases son los "moldes" con los que se crean los objetos. Hablaremos de ello más adelante.
- **unknown type.** Tipo desconocido.
- **NULL.** Es el tipo de una variable a la que aún no se le ha asignado valor alguno.

En este apartado hemos visto que hay distintos tipos de valores que se pueden asignar a las variables. Algunos ya los hemos conocido, aunque sea por encima, como es el caso de las cadenas y los valores numéricos. Otros los iremos conociendo según necesitemos de su uso para continuar aprendiendo. Lo que sí necesitamos saber ahora es cómo manejar los tipos de datos para adaptarlos a nuestras necesidades en cada caso. Más adelante ya veremos para qué nos puede ser útil esto.

Antes de continuar con este tema quiero hacer un inciso para comentar la *precedencia de operadores*. Cuando PHP encuentra una operación aritmética en la que intervengan varios operadores, ejecuta primero los productos y las divisiones y luego las sumas y restas. Así pues, suponga que escribimos la siguiente sentencia:

```
$valor = 3+2*5;
```

El resultado será 13. Primero se calcula el producto de  $2*5$ , que es 10, y luego se le suma 3. Usted puede romper la precedencia natural de los operadores mediante el uso de paréntesis. Por ejemplo, suponga lo siguiente:

```
$valor = (3+2)*5;
```

En este caso, primero se efectúa la suma de  $3+2$ , lo que da 5. Luego se realiza el producto, con lo que se obtiene 25.

### 3.3.1 Conocer y cambiar el tipo de un dato

PHP nos permite conocer el tipo de dato que hay almacenado en una variable, mediante la función `gettype()`. Observe el script `obtenerTipo.php`, que aparece a continuación:

```
<html>
  <body>
    <?php
      echo ("El tipo de la variable_1 es:
".gettype($variable_1)."<br />");
      $variable_2=3;

      echo ("La variable_2 vale:
".$variable_2." y su tipo es: ".gettype($variable_2)."<br
/>");
      $variable_3=359483.0928498038925;
      echo ("La variable_3 vale:
".$variable_3." y su tipo es: ".gettype($variable_3)."<br
/>");
```

```

$variable_4="Esto es una cadena";
echo ("La variable_4 vale:
<b>".$variable_4."</b> y su tipo es:
".gettype($variable_4)."<br />");
?>
</body>
</html>

```

Observe el resultado en la figura 3.6.

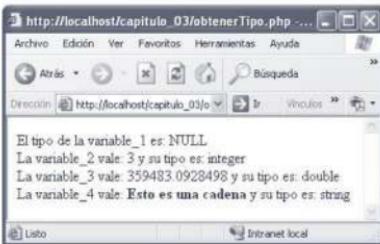


Figura 3.6

Como ve, creamos algunas variables, les asignamos valores y luego, mediante la función `gettype()` determinamos qué tipo de datos contienen. Observe que esta función no muestra el valor que tiene la variable (eso hemos tenido que hacerlo aparte), sino el tipo de dato que es. Mención especial merece la primera línea del resultado. Si examina el código verá que aplicamos la función `gettype()` a una variable que no ha sido declarada. Por esta razón no contiene ningún valor, con lo que el tipo de dato es `NULL`.

PHP nos proporciona otra función relacionada con el tipo de dato que contiene una variable. La función `settype()` permite modificar este tipo de dato, con ciertas limitaciones, como veremos a continuación. Observe el script `cambiarTipo.php` (he omitido las etiquetas HTML del principio y del final para facilitarle la lectura del código).

```

<?php
$variable_1=3594873.0928498038925;
echo ("La variable_1 vale: ".$variable_1." y su
tipo es: ".gettype($variable_1)."<br />"); 
settype ($variable_1,"integer");
echo ("Ahora la variable_1 vale: ".$variable_1."
y su nuevo tipo es: ".gettype($variable_1)."<br />"); 
?>

```

El resultado aparece en la figura 3.7.

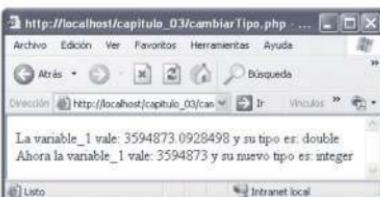


Figura 3.7

La función `settype()` recibe dos argumentos separados por una coma. El primero es la variable cuyo valor queremos cambiar de tipo. El segundo es el nombre del tipo de dato que le asignaremos, encerrado entre comillas. El nombre de los tipos de datos se corresponde con los que aparecen en la lista que he escrito anteriormente. Observe que tenemos una variable de tipo `double` y, al aplicarle la función `settype()`, como hemos hecho en el código, la hemos convertido a `integer`. Es decir: un número decimal lo hemos convertido a entero, lo que, naturalmente, significa que se ha perdido la parte fraccionaria. Sin embargo, éste no es el único efecto de este tipo de conversión. Observe el script `cambioAEntero.php`:

```
<?php
$variable_1=359425634535554334543873.092849803892
5;
echo ("La variable _1 vale: ".$variable_1." y su
tipo es: ".gettype($variable_1)."  
");
settype ($variable_1,"integer");
echo ("Ahora la variable _1 vale: ".$variable_1."
y su nuevo tipo es: ".gettype($variable_1)."  
");
?>
```

El resultado es el que aparece en la figura 3.8.

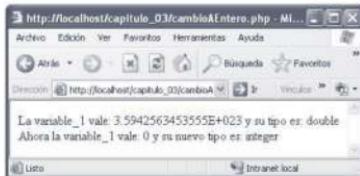


Figura 3.8

Observe dos cosas: en primer lugar, al ser el valor de la variable muy grande aparece como notación exponencial. Por otra parte, al convertir el valor a integer el resultado es 0. Esto es porque los datos de tipo integer no pueden almacenar valores tan grandes como los de tipo double, ni siquiera eliminando la parte fraccionaria.

Podemos convertir un valor numérico a una cadena. Observe el script **cambioACadena.php**.

```
<?php
    $variable_1=359425634535554334543873.092849803892
5;
    echo ("La variable_1 vale: ".$variable_1." y su
tipo es: ".gettype($variable_1)."  
");
    settype ($variable_1,"string");
    echo ("Ahora la variable_1 vale: ".$variable_1."
y su nuevo tipo es: ".gettype($variable_1)."  
");
?>
```

Cuando ejecute esta página verá que el valor que se muestra en la segunda línea es, aparentemente, el mismo que en la primera. Sin embargo, después de la conversión a tipo string la variable ya no contiene un valor numérico, sino una cadena, con lo que se puede tratar como tal a efectos de, por ejemplo, las funciones que estudiaremos en los capítulos 6 y 7.



Figura 3.9

Lo que también podemos hacer es convertir una variable de tipo cadena a tipo numérico. Observe el script **cambioADoble.php**.

```
<?php
    $variable_1="0345";
    echo ("La variable_1 vale: ".$variable_1." y su
tipo es: ".gettype($variable_1)."  
");
    settype ($variable_1,"double");
```

```
echo ("Ahora la variable _1 vale: ".$variable_1."  
y su nuevo tipo es: ".gettype($variable_1)."<br />");  
?>
```

El resultado aparece en la figura 3.9.

También puede cambiarse el tipo de dato de una variable mediante lo que se conoce como **casting**. Consiste en anteponer el tipo deseado, entre paréntesis, antes del nombre de la variable. Por ejemplo, mire el listado **casting.php**, que aparece a continuación:

```
<?php  
$numero=3402.34;  
echo ("La variable vale: ".$numero.", y es de  
tipo: ".gettype ($numero)."<br />");  
  
$numero=(int)($numero);  
  
echo ("Ahora vale: $numero, y es de tipo:  
.gettype ($numero)."<br />");  
?>
```

Fíjese en la línea resaltada, donde se emplea esta técnica. El resultado de ejecutar esta página será el siguiente:

```
La variable vale: 3402.34, y es de tipo: double  
Ahora vale: 3402, y es de tipo: integer
```

Como ve, se ha cambiado el tipo de la variable a integer, truncándose, como es lógico, la parte fraccionaria.

### 3.4 EXPANSIÓN DE VARIABLES

PHP tiene una cualidad específica en el manejo de las variables. Cuando, en una cadena, se incluye el nombre de una variable (con el signo dólar incluido) el intérprete incluye el valor de dicha variable en lugar de su nombre. Observe el script **expansion.php**.

```
<?php  
  
$variable_1="HOLA";  
echo ("La variable_1 vale: $variable_1");  
?>
```

Vea que el nombre de la variable está dentro de la cadena, por lo que cabría pensar que PHP va a reproducir ese nombre tal cual lo hemos escrito. Al menos así actúan otros lenguajes de script que circulan por el mundo, como es el caso de JavaScript, VBScript, etc., así como otros lenguajes de programación, como Java, Visual Basic, etc. Sin embargo, PHP interpreta la variable y lo que incluye en la cadena es el valor correspondiente, tal como se ve en la figura 3.10.



Figura 3.10

Esta característica de PHP se conoce con el nombre de *expansión de variables*. En ese caso, ¿cómo podemos hacer que el nombre de una variable, incluyendo el signo dólar, se muestre tal cual, como parte de la cadena? Bien. Podemos delimitar la cadena completa mediante comillas simples, en lugar de comillas dobles, tal como se ve en el script **comillaSimple.php**.

```
<?php  
    $variable_1="HOLA";  
    echo ('La variable_1 vale: $variable_1');  
?>
```

Otra alternativa para lograr lo mismo es lo que se conoce como *escapar un carácter*. Consiste en añadirle a un carácter determinado un *contraslash*, o barra inclinada inversa, que anula el efecto de dicho carácter. En este caso se lo añadiríamos al carácter dólar del nombre de la variable, como aparece en el script **escaparDolar.php**.

```
<?php  
    $variable_1="HOLA";  
    echo ("La variable_1 vale: \$variable_1");  
?>
```

Quizás la solución más empleada cuando se necesita obviar la expansión de variables sea ésta última, ya que el uso de comillas simples solemos reservarlo para otros fines que estudiaremos más adelante. En ambos casos el resultado de la ejecución del script es el mismo y aparece en la figura 3.11.



Figura 3.11

Escapar un carácter ofrece más posibilidades. Vea en la figura 3.12 una tabla con la lista de los caracteres escapables más comunes.

Para ver un ejemplo suponga que usted quiere incluir unas comillas dobles dentro de su cadena alfanumérica. Tal vez usted pueda pensar, a priori, en algo como lo siguiente:

```
$cadena="Aquí hay alguien que dice "Hola".";
```

Sin embargo esto no funcionaría, ya que al encontrar el intérprete la segunda comilla doble (la que está delante de la H de Hola) se considera que la cadena termina ahí. El resto de lo que usted ha escrito no es interpretable por PHP y genera un error. Sin embargo, usted puede solucionarlo escapando las dos comillas dobles que hay dentro de la cadena (las que delimitan la palabra Hola), así:

```
$cadena="Aquí hay alguien que dice \"Hola\".;
```

Como ve, he añadido un contraslash delante de cada uno de los caracteres que deseo escapar. Otra alternativa es el llamado *anidamiento de comillas*. Consiste en usar comillas simples en el interior de la cadena, así:

```
$cadena="Aquí hay alguien que dice 'Hola'.";
```

Si lo que desea es que aparezcan comillas dobles dentro de la cadena puede delimitar esta con comillas simples, para poder hacer anidamiento, así:

```
$cadena='Aquí hay alguien que dice "Hola".';
```

Sin embargo, permítame desaconsejar esto último. Acostúmbrase a delimitar las cadenas con comillas dobles, como norma. Si quiere incluir otras comillas dobles en la cadena, escápelas.

CARÁCTER ESCAPADO	SIGNIFICADO
\n	Nueva línea
\r	Retorno de carro
\t	Tabulación horizontal
\\$	Signo del dólar
'	Comilla simple
''	Comilla doble
\\"	Contraslash

Figura 3.12

### 3.5 OTRAS FUNCIONES DE MANEJO DE VARIABLES

PHP incluye otras funciones para el manejo básico de variables. En primer lugar, contamos con la función *isset()*, que nos permite determinar si una variable concreta tiene ya un valor asignado o no. Esto es especialmente útil en procesos de depuración de scripts. Durante la corrección de los mismos se puede comprobar si hay contenido en una variable para rastrear posibles errores. La función *isset()* recibe como argumento el nombre de una variable y devuelve un valor de tipo booleano, es decir, *true (verdadero)* o *false (falso)*. Cuando hablemos de condicionales veremos la utilidad de este tipo de valores.



Figura 3.13

Para ver en la práctica el comportamiento de *isset()* observe el script `variablesEstablecidas.php`.

```

<?php
    echo ("El resultado de isset() con \$variable_1
es ".isset ($variable_1)."<br />");
```

```
$variable_1="HOLA";
echo ("Ahora el resultado de isset () con
\$variable_1 es ".isset ($variable_1)."<br />");
?>
```

El resultado aparece en la figura 3.13.

Observe una cosa curiosa. Los valores de tipo false no son mostrados en la página y los valores true aparecen como un 1. Esto puede resultar un poco confuso al principio, pero PHP los evalúa y maneja perfectamente, tal como veremos más adelante.

También contamos con la función *unset ()*, destinada a eliminar el contenido de una variable. Observe el script **variablesEliminadas.php**.

```
<?php
$variable_1="HOLA";
echo ("El tipo de \$variable_1 es ".gettype
($variable_1)."<br />");

unset ($variable_1);
echo ("Ahora el tipo de \$variable_1 es ".gettype
($variable_1)."<br />");

?>
```

Al ejecutarlo obtenemos el resultado de la figura 3.14.



Figura 3.14

Como ve, al mostrarse la segunda línea de resultados en la página nos informa que el tipo de variable es NULL, es decir, *unset ()* ha eliminado el contenido de la misma.

La función *empty ()* tiene un efecto similar a *isset ()*, pero con el resultado opuesto, es decir, devuelve un true booleano si la variable no ha recibido todavía ningún valor. En caso contrario devuelve un resultado false. Observe el script **variablesVacias.php**.

```
<?php
    echo ("El resultado de empty () con \$variable_1
es ".empty ($variable_1). "<br />");

    $variable_1="HOLA";

    echo ("Ahora el resultado de empty () con
\$variable_1 es ".empty ($variable_1). "<br />");
?>
```

Las funciones *is\_integer()*, *is\_double()* e *is\_string()* reciben como argumento el nombre de una variable, y nos devuelven un valor booleano (true o false) dependiendo de que la variable sea un número entero, un número en coma flotante o una cadena, respectivamente. Sin embargo, para determinar el tipo de dato que contiene una variable podemos usar, en la mayoría de los casos, la función *gettype*, vista anteriormente.

### 3.6 VARIABLES DE VARIABLES

PHP puede reconocer y manejar variables que se refieran a otras variables. Se trata de variables cuyo contenido es el nombre de otras variables. Esto resulta especialmente útil cuando se quiere crear código dinámico, basado en ciertas condiciones establecidas por los usuarios de nuestras páginas. Por ejemplo, suponga que usted quiere crear un mensaje de saludo que pueda aparecer en español o en inglés (podríamos ponerlo en más idiomas, por supuesto, pero se trata de un ejemplo sencillo).



Figura 3.15

Usted puede almacenar los dos saludos en variables de memoria y luego mostrar el que convenga, como muestra el script **variablesVariables.php**:

```
<?php
    $saludo="Hola, amigo";
    $saludoParaMostrar="saludo";
    echo ($$saludoParaMostrar."<br />");
```

```
$saludo="Hi, fellow";
echo ($$saludoParaMostrar."<br />");
?>
```

Observe el resultado en la figura 3.15.

Veamos qué ha ocurrido. En primer lugar fíjese en que tenemos una variable llamada \$saludo, cuyo contenido varía durante la ejecución del script y otra, llamada \$\$saludoParaMostrar, cuyo contenido es el nombre de la anterior, sin el signo \$. Cuando queremos recuperar el contenido de \$saludoParaMostrar debemos anteponer un doble signo de dólar, tal como se ve en las dos líneas resaltadas del código. De este modo, el intérprete recupera el contenido de la variable \$saludo. Así pues, \$saludoParaMostrar está actuando como un puntero a \$saludo, y se actualiza en tiempo real. Por supuesto, en este ejemplo no es muy útil. Está diseñado, únicamente, para ilustrar el funcionamiento de esta técnica. Sin embargo, cuando se manejan cookies, bases de datos, ficheros de disco, etc., resulta muy práctica para simplificar el código del script. Yo he escrito scripts en PHP que ahorran hasta un 15% de código mediante el empleo de esta técnica.

### 3.7 CONSTANTES

En ocasiones es necesario crear y utilizar pares nombre-valor cuyo contenido va a permanecer inmutable durante toda la ejecución del script. Por ejemplo, suponga que necesita almacenar el valor de Pi (3.1415927) para efectuar distintos cálculos aritméticos. Este valor no cambia nunca. Para esta finalidad empleamos **constants**, que se crean mediante el uso de la función **define()**. Esta recibe dos argumentos separados con una coma. El primero es el nombre de la constante y el segundo es su valor. Observe el script **constantes.php**.

```
<?php
    define ("Pi", 3.1415927);
    echo ("El valor de Pi es: ".Pi);
?>
```

Al ejecutarlo verá que en la página le aparece el valor de la constante definida. Cuando se define una constante no se puede modificar su valor en ningún punto de la ejecución del script. Por eso se les llama constantes. Si intentamos redefinirla obtendremos un error, por lo que deberemos tomar buena nota de las constantes ya definidas.

Al igual que ocurre con las variables, el nombre de las constantes es sensible a las mayúsculas y minúsculas, por lo que, por ejemplo, Pi no es la misma constante que pi.

Si queremos determinar si una constante está definida en algún punto concreto del script (por ejemplo, a efectos de depuración) podemos usar la función **defined()**, que recibe como argumento el nombre de la constante (entre comillas) y nos devuelve un valor booleano (true o false). Podríamos usar algo así:

```
echo (defined ("Pi"));
```

El uso de constantes puede simplificar algunas acciones que usted quiera realizar en sus páginas. Por ejemplo, suponga que quiere establecer un color de letra determinado que se aplicará a ciertas palabras o frases de su texto para destacarlas y llamar la atención del usuario de su sitio. Puede definir el color en una constante, como en el siguiente ejemplo:

```
define ("colorDestacado", "#FF0000");
```

Como ya sabemos que desde PHP se pueden incluir etiquetas de HTML (en muchos de los ejemplos anteriores hemos incluido la etiqueta <br /> en varias ocasiones) cuando, en otra parte del código, necesitemos usar el color que destaque el texto incluiremos en la página lo siguiente:

```
<?php  
    echo ("<font color=".colorDestacado.">");  
?>
```

Esto tiene la ventaja de que la constante sólo es necesario definirla una vez en todo el script, con independencia de cuántas veces se utilice. Lo lógico es definirla al principio del script. Así, si en algún momento usted decide que no le gusta el color que ha empleado en las palabras destacadas y quiere emplear otro, sólo debe cambiar una línea de código (aquella en la que se define la constante). Quiero que se fije en algo que, seguramente, le habrá llamado la atención. Al usar una constante no se antepone, en ningún caso, el signo \$, como hacemos con las variables. Las constantes y las variables constituyen dos categorías completamente diferentes de pares nombre-valor, de modo que PHP permite que exista una variable y una constante con el mismo nombre en el mismo script y las reconoce como dos datos diferentes. Sin embargo, le desaconsejo esta práctica, por razones de claridad. Es muy fácil cometer errores con esto.

## 3.8 MATRICES

PHP permite almacenar varias variables diferentes, que pueden almacenar valores distintos, bajo el mismo nombre, identificándolas mediante el uso de uno o más índices. Son las **matrices**. Podemos imaginar las matrices como las casillas de

apartados postales en una oficina de correos. Todas pueden ser identificadas mediante, por ejemplo, el nombre de la oficina seguido del número de casilla, y cada una tendrá un contenido distinto. El formato genérico es así:

```
nombreMatriz[índice 1][índice 2]...[índice N] = valor
```

Normalmente se emplean matrices con un solo índice o, como mucho, con dos, aunque se pueden emplear más, si usted lo necesita en algún caso (a mí, hasta la fecha, nunca se me ha presentado la necesidad de más de dos índices).

Las matrices se clasifican, según la naturaleza del índice empleado, en *indexadas* y *asociativas*. A continuación vamos a estudiar el uso y características de ambos tipos. En realidad se parecen bastante: el concepto fundamental es el mismo, como verá enseguida.

### 3.8.1 Matrices indexadas

Son aquéllas en las que el índice es un valor numérico. Para empezar, veamos cómo creariamos una matriz con nombres de personas (aunque, por supuesto, podríamos usar otros valores):

```
$nombres = array ("Pedro", "Ana", "Carmen", "Alfredo",  
"Eva");
```

Con esto se habría creado una matriz de cinco elementos. Fijese en que usamos la función *array()*. Esta función recibe, como argumentos, los elementos que queremos almacenar en la matriz, y la crea. A cada uno de ellos se le ha asignado un índice numérico secuencial, según el orden en que se hallan dispuestos en la declaración, empezando por cero. Para recuperar alguno de los elementos de la matriz usaremos la notación genérica siguiente:

```
$matriz [$n]
```

En esta notación, \$n entre corchetes es un valor numérico referido al índice del elemento de la matriz que nos interesa. Como índice podemos usar un número o una constante o variable que lo represente.

Observe el script **crearMatriz.php**, que incorpora la línea en la que se crea la matriz y que luego muestra los valores almacenados.

```
<?php  
$nombres = array ("Pedro", "Ana", "Carmen",  
"Alfredo", "Eva");
```

```
echo ($nombres[0]."<br />");  
echo ($nombres[1]."<br />");  
echo ($nombres[2]."<br />");  
echo ($nombres[3]."<br />");  
echo ($nombres[4]."<br />");  
?>
```

Con este código se verán los elementos de la matriz en la página. Dese cuenta de que el primer elemento, que contiene Pedro es el que tiene el índice 0. El último elemento, que contiene Eva, es el que tiene asignado el índice 4. Algo que debe saber es que los elementos de una matriz no tienen que ser, necesariamente, del mismo tipo. En una matriz se pueden almacenar cadenas alfanuméricas, valores numéricos, valores booleanos, etc. Observe el script **matrizVariosTipos.php**, similar al anterior pero con contenidos de distintos tipos.

```
<?php  
$nombres = array ("Pedro", "Ana", 34, True);  
echo ($nombres[0]."<br />");  
echo ($nombres[1]."<br />");  
echo ($nombres[2]."<br />");  
echo ($nombres[3]."<br />");  
?>
```

Al ejecutarlo verá que se muestran los valores almacenados en la matriz, como hacia el ejemplo anterior. Quizás lo que le llame la atención es el último valor. Es un true lógico y, como hemos visto en anteriores ejemplos, se muestra en la página como un 1.

Las matrices no tienen un número inamovible de elementos. Si, una vez creada la matriz, usted necesita añadir más elementos, puede hacerlo simplemente declarando el elemento nuevo y su valor. Observe el script **nuevoElemento.php**:

```
<?php  
$nombres = array ("Pedro", "Ana", "Carmen",  
"Alfredo", "Eva");  
echo ($nombres[0]."<br />");  
echo ($nombres[1]."<br />");  
echo ($nombres[2]."<br />");  
echo ($nombres[3]."<br />");  
echo ($nombres[4]."<br />");  
$nombres[5] = "Susana";  
echo ($nombres[5]."<br />");  
?>
```

Quiero llamar su atención sobre la línea que aparece resaltada. En ella ve cómo se crea un nuevo elemento que no aparece en la declaración inicial de la matriz. Al ejecutar la página verá que funciona correctamente. Le he asignado al nuevo elemento el índice 5 porque sé que es el siguiente de la lista, ya que iban de 0 a 4. Esto no es necesario. La línea resaltada en el código podría quedar, perfectamente, así:

```
$nombres [] = "Susana";
```

Al asignar un contenido a un elemento de la matriz sin especificar el índice de dicho elemento el intérprete de PHP crea, de modo automático, el siguiente índice al último empleado. Como, en nuestro ejemplo, el último índice creado es el 4, PHP creará el elemento 5 para almacenar el nuevo contenido.

A la hora de referirnos a uno de los elementos de una matriz podemos escribir el índice, o una variable que contenga el índice, tal como se muestra en el script **situarIndice.php**. Observe en la línea resaltada que he usado una variable donde he almacenado un valor numérico para referirme al índice de la matriz.

```
<?php
    $nombres = array ("Pedro", "Ana", "Carmen",
"Alfredo", "Eva");
    $indice=2;
    echo ($nombres[$indice]."<br />");
?
>
```

En otro orden de cosas, hemos dicho que los índices empiezan a crearse desde 0 y se siguen creando en secuencia incremental. Éste es el comportamiento por defecto de las matrices. Sin embargo, esto no tiene que ser así siempre. Usted puede crear los índices según le convenga en cada caso. Por ejemplo, vea el script **cambiarIndice.php**.

```
<?php
    $nombres = array (1=>"Pedro", "Ana", "Carmen",
"Alfredo", "Eva");
    echo ($nombres[1]."<br />");
    echo ($nombres[2]."<br />");
    echo ($nombres[3]."<br />");
    echo ($nombres[4]."<br />");
    echo ($nombres[5]."<br />");
?
>
```

Observe la linea donde se hace la declaración. Al primer elemento se le asigna, de forma forzada, el índice 1, mediante el operador =>. Como a los demás

elementos no se les asigna un índice específico PHP los numerará secuencialmente a partir del 2. Así pues, en este caso los elementos de la matriz están numerados de 1 a 5, en lugar de estarlo de 0 a 4. El elemento 0 no existe en este caso. Usted puede forzar la creación de los elementos según le convenga en cada caso. Por ejemplo, usted podría crear una matriz con la siguiente declaración:

```
$nombres = array (1=>"Pedro", "Ana", 50=>"Carmen",
"Alfredo", "Eva");
```

En este caso la matriz contendrá los elementos 1, 2, 50, 51 y 52. Cuando se crea una matriz indexada de este modo PHP no reserva un espacio en memoria para los elementos no declarados, con lo que la matriz no ocupa más memoria que si hubiera creado los elementos de 0 a 4. Y, naturalmente, usted puede crear nuevos elementos, incluso con índices intermedios, cuando lo necesite. Por ejemplo, en otro punto del script se podría incluir una línea como la siguiente:

```
$nombres[34] = "Sonia";
```

### 3.8.2 Matrices asociativas

Usar índices numéricos en las matrices es muy práctico en ocasiones. A veces para referirse a un elemento de una matriz resulta interesante calcular su índice mediante operaciones aritméticas basadas en datos proporcionados por el usuario. Sin embargo, hay ocasiones en que el uso de índices numéricos es engorroso por la misma razón que lo es el nombrar a las variables de un script como \$variable\_1, \$variable\_2, etc. Simplemente, los índices numéricos no proporcionan ninguna referencia clara acerca del contenido de un elemento de la matriz. Además, existen algunas matrices específicas que, por su propia naturaleza, exigen otro manejo diferente. Para solucionar esta cuestión PHP implementa las matrices asociativas, en las que los índices son cadenas de texto que podemos definir como nos convenga. Suponga que va a crear una matriz con los datos personales de un amigo suyo. Usted quiere almacenar en la matriz el nombre, la dirección y el teléfono de esta persona. Observe el script **crearAsociativa.php**:

```
<?php
$amigo = array ("nombre"=>"Pedro Torres",
"direccion"=>"CL Mayor, 37", "telefono"=>123456789);
echo ($amigo["nombre"]."<br />");
echo ($amigo["direccion"]."<br />");
echo ($amigo["telefono"]."<br />");
?>
```

Observe la línea resaltada, que corresponde a la declaración de la matriz. Hemos creado tres elementos que contienen los valores que deseábamos almacenar. Cada uno de estos elementos tiene un índice formado por una cadena alfanumérica. Los elementos, a su vez, pueden tener el tipo de valor que necesitemos, tal como ocurre con las matrices indexadas. Como es lógico, en una matriz asociativa también podemos poner, en lugar del índice, una variable que contenga dicho índice. Observe el script **indiceDeAsociativa.php**:

```
<?php
    $amigo = array ("nombre"=>"Pedro Torres",
"direccion"=>"CL Mayor, 37", "telefono"=>123456789);
    $indice="nombre";
    echo ($amigo[$indice]."<br />");
    $indice="direccion";
    echo ($amigo[$indice]."<br />");
    $indice="telefono";
    echo ($amigo[$indice]."<br />");
?
>
```

Observe las líneas resaltadas del código y ejecútelo para comprobar su funcionamiento.

### 3.8.3 Matrices mixtas

Visto lo anterior, parece que falta algo. Por ejemplo, suponga que usted quiere almacenar en una matriz los datos personales de más amigos. La solución puede llegar de la mano de matrices con dos índices: uno numérico para referirse a cada amigo y otro asociativo para almacenar los datos de cada uno. Sería como un casillero en el que cada fila correspondería a un amigo y cada columna a un dato. Observe el script **indexadaYAsociativa.php**. Ponga especial atención a la forma de declarar la matriz, ya que, en este caso, es un poco más compleja que en los anteriores.

```
<?php
    $amigos = array (array("nombre"=>"Pedro Torres",
"direccion"=>"CL Mayor, 37", "telefono"=>123456789),
array("nombre"=>"Carlos Gómez", "direccion"=>"CL Alfareros,
12", "telefono"=>567891234), array("nombre"=>"Susana Casas",
"direccion"=>"CL Sierra Grande, 2", "telefono"=>987654321),
array("nombre"=>"Carmen Pérez", "direccion"=>"CL Himalaya,
189", "telefono"=>502983948));
    echo ("<table border='2' cellpadding='2'
cellspacing='0'>");
    echo ("<tr>");
```

```
echo ("<th>Número</th>");
echo ("<th>Nombre</th>");
echo ("<th>Dirección</th>");
echo ("<th>Teléfono</th>");
echo ("</tr>");
echo ("<tr>");
echo ("<td>0</td>");
echo ("<td>".$amigos[0]["nombre"]."</td>");
echo ("<td>".$amigos[0]["direccion"]."</td>");
echo ("<td>".$amigos[0]["telefono"]."</td>");
echo ("</tr>");
echo ("<tr>");
echo ("<td>1</td>");
echo ("<td>".$amigos[1]["nombre"]."</td>");
echo ("<td>".$amigos[1]["direccion"]."</td>");
echo ("<td>".$amigos[1]["telefono"]."</td>");
echo ("</tr>");
echo ("<tr>");
echo ("<td>2</td>");
echo ("<td>".$amigos[2]["nombre"]."</td>");
echo ("<td>".$amigos[2]["direccion"]."</td>");
echo ("<td>".$amigos[2]["telefono"]."</td>");
echo ("</tr>");
echo ("<tr>");
echo ("<td>3</td>");
echo ("<td>".$amigos[3]["nombre"]."</td>");
echo ("<td>".$amigos[3]["direccion"]."</td>");
echo ("<td>".$amigos[3]["telefono"]."</td>");
echo ("</tr>");
echo ("</table>");

?>
```

En primer lugar, ejecute la página en su navegador para ver cómo funciona. Observe la declaración de la matriz (resaltada en el código). Es una sola línea, aunque por las limitaciones del papel parezcan varias. Lo que hacemos es declarar una matriz indexada en la que cada elemento es, a su vez, una matriz asociativa. Por esa razón dentro de cada elemento de la matriz indexada aparece de nuevo la función array () .

Ahora repare en la forma de acceder a un elemento de una matriz con dos índices. El modo genérico es el siguiente:

```
$matriz [$i1] [$i2]
```

Por supuesto, en una matriz de dos índices puede usar uno numérico y otro asociativo, los dos numéricos o los dos asociativos, según convenga en cada caso.

Una matriz con dos índices no tiene por qué tener el mismo número de elementos en cada fila. Suponga que, en su lista de amigos, le faltan algunos datos. Por ejemplo, puede ser que no conozca la dirección o el teléfono de algunos de sus amigos. La declaración de la matriz podría quedar así:

```
$amigos = array (array("nombre"=>"Pedro Torres",
"direccion"=>"CL Mayor, 37", "telefono"=>123456789),
array("nombre"=>"Carlos Gómez", "telefono"=>567891234),
array("nombre"=>"Susana Casas", "direccion"=>"CL Sierra
Grande, 2", "telefono"=>987654321), array("nombre"=>"Carmen
Pérez", "direccion"=>"CL Himalaya, 189"));
```

Como ve, falta la dirección de Carlos Gómez y el teléfono de Carmen Pérez. Esta forma de declarar una matriz es perfectamente válida en PHP. Otros lenguajes también admiten la creación y uso de estas matrices irregulares. De hecho, muchas de las matrices de dos índices que use serán así.

### 3.8.4 Determinar el tamaño de una matriz

La función *count()* nos permite determinar el número de elementos de una matriz. Observe el script **contarElementos.php**:

```
<?php
$nombres      =      array      (1=>"Pedro",      "Ana",
50=>"Carmen",      "Alfredo",      "Eva");
$personas = count ($nombres);
echo ("Número de personas: ".$personas."<br />");

$amigo       =      array      ("nombre"=>"Pedro      Torres",
"direccion"=>"CL Mayor, 37", "telefono"=>123456789);
$datosDeAmigo = count ($amigo);
echo ("Número de datos de mi amigo:
".$datosDeAmigo."<br />");

$amigos = array (array("nombre"=>"Pedro Torres",
"direccion"=>"CL Mayor, 37", "telefono"=>123456789),
array("nombre"=>"Carlos Gómez", "direccion"=>"CL Alfareros,
12", "telefono"=>567891234), array("nombre"=>"Susana Casas",
"direccion"=>"CL Sierra Grande, 2", "telefono"=>987654321),
array("nombre"=>"Carmen Pérez", "direccion"=>"CL Himalaya,
189", "telefono"=>502983948));
$totalDeAmigos = count ($amigos);
echo ("Total de amigos: ". $totalDeAmigos);

?>
```

La función `count()` recibe como argumento el nombre de la matriz de la cual queremos saber qué número de elementos tiene, y devuelve dicho número de elementos. En el código hay tres matrices que ya hemos usado antes. La función `count()` opera con matrices indexadas o asociativas indistintamente.

Observe que, en el caso de la primera matriz, la función `count()` nos informa de que hay cinco elementos, como debe ser. El hecho de que los índices no empiecen por cero o no sean consecutivos no influye para nada.

Fíjese en el caso particular de la tercera matriz, que tiene dos índices. En este caso, la función `count()` sólo cuenta el índice principal (el de las filas, para entendernos). Si usted quiere determinar el número de elementos de una de las filas, puede hacerlo como se muestra en el script **otraCuenta.php**.

```
<?php  
    $amigos = array (array("nombre"=>"Pedro Torres",  
"direccion"=>"CL Mayor, 37", "telefono"=>123456789),  
array ("nombre"=>"Carlos Gómez", "direccion"=>"CL Alfareros,  
12", "telefono"=>567891234), array ("nombre"=>"Susana Casas",  
"direccion"=>"CL Sierra Grande, 2", "telefono"=>987654321),  
array ("nombre"=>"Carmen Pérez", "direccion"=>"CL Himalaya,  
189", "telefono"=>502983948));  
  
    $totalDeDatos = count ($amigos[0]);  
    echo ("Nº de datos de mi amigo: ".  
$totalDeDatos);  
?>
```

Observe que la función `count()`, en este caso, recibe como argumento una de las filas de la matriz.

### 3.8.5 Ordenar una matriz

La gestión de matrices aporta un elemento del que será necesario disponer en ocasiones: la capacidad de reordenación. Efectivamente, dada una matriz usted puede escribir un script que necesite disponer de los datos en cierto orden. En una matriz indexada la reordenación se basa en el valor de cada elemento. Una matriz asociativa puede ordenarse en función de la clave o del valor. Para llevar esto a efecto contamos con varias funciones. La función `sort()` recibe, como argumento, el nombre de una matriz indexada y la ordena según los contenidos de sus elementos, de menor a mayor. Si los contenidos son numéricos está claro cómo se hace la ordenación. Si son alfabéticos, se considera un elemento mayor que otro en base al código ASCII del primer carácter. Si éste es idéntico en ambos elementos, se compara el segundo carácter, y así sucesivamente. Si quiere consultar el código

ASCII lo tiene a su disposición en el Apéndice E. La sintaxis genérica de esta función es la siguiente:

```
sort ($matriz);
```

Al ejecutarla, la matriz no cambia de nombre. Simplemente se ordenan sus elementos como se ha indicado.

Como alternativa, tenemos la función *rsort()*, cuya operativa es similar a la anterior, solo que efectuando el orden en sentido descendente, de mayor a menor. Estas funciones no deben ser empleadas sobre matrices asociativas, ya que reordenan los elementos por valor, pero no tienen en cuenta las claves, con lo cual se pierde la paridad clave-valor que usted haya establecido en cada elemento de la matriz. Para ordenar una matriz asociativa emplee la función *asort()* para ordenar por el valor de cada elemento, de menor a mayor. También puede usar *arsort()*, para ordenar por valor de mayor a menor. Si quiere usar la clave para la ordenación, emplee *ksort()*, para ordenar en sentido ascendente, o *krsort()*, para ordenar en sentido descendente. Estas cuatro últimas funciones sí mantienen cada clave asociada a su valor.

## 3.9 OTRAS BASES DE NUMERACIÓN

Como usted sabe, en informática se manejan, en ocasiones, distintas bases de numeración. PHP nos proporciona una función que permite convertir un número a decimal, cuando este se haya creado en otra base.

Por ejemplo, suponga el número FF en hexadecimal. Todos sabemos que es el equivalente del 255 decimal. Después de todo, seguro que usted ha manejado éste y otros valores de dos dígitos hexadecimales estableciendo el color de fondo o de texto de alguna página. Lo que vamos a ver es cómo PHP efectúa la conversión adecuada desde el valor hexadecimal al decimal. Para ello se emplea la función *intval()*. Recibe dos argumentos, separados por una coma. El primero es el valor, en una base distinta de decimal, y el segundo es la base en la que está dicho valor. Veamos un ejemplo en el script **numeroHexa.php**:

```
<?php  
$numeroHexadecimal="ff";  
$numeroDecimal=intval($numeroHexadecimal,16);  
echo ($numeroDecimal);  
?>
```

Al cargar el script en el navegador nos mostrará el valor decimal correspondiente, que es 255, como decía hace un momento. La función *intval()*

puede recibir un valor en cualquier base. Lo único a tener en cuenta es que el valor sólo emplee dígitos de la base con la que queremos trabajar, es decir, una aplicación de esta función que no sería válida es la siguiente:

```
$numeroDecimal=intval("ff",2);
```

Esto es porque le estamos diciendo que el valor está en binario (base 2) y en esta base sólo existen los dígitos 0 y 1, no el dígito f (por cierto, en valores hexadecimales los dígitos de "a" a "f" pueden estar, indistintamente, en mayúsculas o minúsculas).

Además, PHP proporciona las funciones *doubleval()* y *strval()* para cambiar el tipo de dato de un valor, pero para eso ya tenemos *settype()*.

## 3.10 COMENTARIOS

Cuando escribimos un script de PHP es normal que, al cabo del tiempo tengamos que modificarlo o ampliarlo. Y es muy difícil que podamos recordar lo que hicimos en su momento y por qué. Y no digamos si nuestro script debe ser mantenido por otra persona. Es imperativo que, a lo largo del código, intercalemos comentarios que aclaren el funcionamiento general del script, el uso que se da a cada variable, etc. PHP nos proporciona dos maneras de incluir comentarios aclaratorios. La primera es el doble slash. Dos barras inclinadas hacia la derecha hacen que el intérprete ignore todo lo que hay desde ese punto al final de la línea. La otra manera de incluir un comentario es encerrándolo entre /\* y \*/. Todo lo que haya entre estos dos juegos de guarismos será ignorado por el intérprete, con independencia del número de líneas que ocupe. Además, el poder poner una línea cualquiera como comentario, simplemente añadiéndole las dos barras al principio, nos permite anular temporalmente una instrucción para poder depurar un código cuando no funciona a la primera (lo que ocurre con más frecuencia de la que uno se imagina). Vea el script **comentarios.php**.

```
<?php
    //Esto es un comentario. Puedo poner lo que
    quiera.
        echo ("Arriba hay un comentario, que no sale en
    la página. El intérprete lo ignora.");
        /* Aquí empieza un comentario de varias líneas
    Puedo poner cualquier número de líneas.
        El intérprete las ignora y no sobrecarga la
    página.
            En esta línea termina el comentario. */
?
```

En este código puede ver el uso de los comentarios. Acostúmbrese a documentar sus scripts, para poder repasarlos o depurarlos.

### 3.11 RASTREO DE VARIABLES

En ocasiones es necesario, durante la depuración de un script, conocer el valor que tiene una variable o un elemento de una matriz en un punto determinado de la ejecución. Para ello contamos en PHP con la función `var_dump()`. Ésta recibe tantos argumentos como queramos pasárle, separados por comas. Éstos serán variables o matrices del script. La función envía al navegador el tipo y valor de cada una de dichas variables. En el caso de las matrices, envía la clave, el tipo y el valor de cada elemento. Observe el script `var_dump.php`:

```
<?php
$cadena="Esto es una cadena";
$entero=45;
$flotante=32.76;
$booleano=true;
$matriz=array("Alfa", "Beta", 68, false);

var_dump ($cadena, $entero, $flotante, $booleano,
$matriz);
?>
```

El resultado es el siguiente:

```
string(18) "Esto es una cadena" int(45) float(32.76) bool(true) array(4)
{ [0]=> string(4) "Alfa" [1]=> string(4) "Beta" [2]=> int(68) [3]=> bool(false) }
```

Como ve, se obtiene la información requerida. Sin embargo, dado que aparece todo en una línea, yo aconsejo usar esta función para cada variable o matriz que necesitemos rastrear, con saltos de línea entre una y otra.



## CONDICIONALES, BUCLES Y FUNCIONES

---

---

En este capítulo vamos a conocer tres conceptos fundamentales de cualquier lenguaje de programación actual. Se trata de los condicionales, los bucles y las funciones definidas por el usuario. Si usted ya conoce JavaScript (y, a estas alturas, es de suponer que así sea) estas estructuras le resultarán sumamente familiares. De hecho su utilidad y sintaxis son muy similares a las que usted pueda conocer de otros lenguajes de script o de Programación Orientada a Objetos. Sin embargo, el uso que de estas estructuras se hace en PHP presenta los matices propios de un lenguaje que corre en el lado del servidor, en lugar de hacerlo en el lado del cliente. Estas diferencias de matiz se apreciarán mejor a lo largo de los códigos que iremos viendo en distintos capítulos del libro.

### 4.1 CONDICIONALES

Los condicionales se emplean para determinar si una determinada sentencia o grupo de sentencias se ejecutarán o no, en base al cumplimiento de determinadas condiciones. La estructura condicional más simple obedece al siguiente esquema:

```
if (condición) {  
    sentencias;  
}
```

La palabra reservada *if ()* evalúa la condición que aparece entre paréntesis. Si ésta resulta ser cierta, se ejecuta el conjunto de sentencias que aparecen entre { y }

}. Si la condición no es cierta, estas sentencias no se ejecutan. Mire el script **condicionalSimple.php** en la carpeta correspondiente a este capítulo.

```
<?php
$edad=34;
if ($edad > 18) {
    echo ("El usuario es mayor de edad.");
}
if ($edad > 65) {
    echo ("El usuario está jubilado.");
}
?>
```

Fíjese en que, en primer lugar, declaramos una variable asignándole un valor. En realidad, esta variable podría proceder de cualquier otro proceso en este script o en cualquier otro, pero para los efectos didácticos, así nos vale. A continuación, tenemos dos condicionales. En el primero se evalúa si el valor de la variable es mayor que 18 (observe que se usa el operador comparativo `>`-mayor que-). Como la condición resulta ser cierta, la sentencia que muestra el texto correspondiente se ejecuta. En el segundo condicional se comprueba si el valor es mayor que 65. Como no lo es, la sentencia comprendida en el condicional no se ejecuta.

Las condiciones que se evalúan en estas estructuras son comparaciones entre valores que se representan mediante los **operadores de comparación** recopilados en la tabla de la figura 4.1.

OPERADOR	SIGNIFICADO
<code>==</code>	<b>IGUALDAD.</b> Para que la condición se cumpla, el valor que aparece a la izquierda del operador debe ser igual que el que aparece a la derecha. No confundir con el operador de asignación, que es un solo signo <code>=</code> en lugar de dos.
<code>&lt;</code>	<b>MENOR QUE.</b> Para que la condición se cumpla, el valor a la izquierda del signo debe ser menor que el de la derecha.
<code>&gt;</code>	<b>MAYOR QUE.</b> Para que la condición se cumpla, el valor a la izquierda del signo debe ser mayor que el de la derecha.
<code>&lt;=</code>	<b>MENOR O IGUAL QUE.</b> Para que la condición se cumpla, el valor a la izquierda del signo debe ser menor o igual que el de la derecha.
<code>&gt;=</code>	<b>MAYOR O IGUAL QUE.</b> Para que la condición se cumpla, el valor a la izquierda del signo debe ser mayor o igual que el de la derecha.

OPERADOR (cont.)	SIGNIFICADO (cont.)
<code>!=</code>	<b>DISTINTO DE.</b> Para que la condición se cumpla, el valor a la izquierda del operador debe ser diferente del que hay a la derecha.
<code>!</code>	<b>NEGACIÓN.</b> Para que el condicional se evalúe como cierto, la condición que aparece a continuación debe ser falsa.

Figura 4.1

Cuando el grupo de sentencias cuya ejecución depende de un condicional está formado por una única sentencia, ésta se puede poner, sin llaves, en la misma línea del condicional. Así pues, el ejemplo anterior se podría haber acortado del siguiente modo:

```
<?php
$edad=34;
if ($edad > 18) echo ("El usuario es mayor de
edad.");
if ($edad > 65) echo ("El usuario está jubilado.");
?>
```

La sentencia condicional if proporciona una salida alternativa para el caso de que la comparación resulte no ser cierta, mediante la palabra reservada **else**. Observe el script **condicionalElse.php**:

```
<?php
$edad=17;
if ($edad > 18) {
    echo ("El usuario es mayor de edad.");
} else {
    echo ("El usuario NO es mayor de edad.");
}
?>
```

Cuando se evalúa la condición, si resulta ser cierta, se ejecuta el bloque de sentencias que aparece inmediatamente debajo de dicha evaluación. Si no es cierta, se ejecuta el bloque de sentencias comprendidas en la estructura else.

Una condición no tiene por qué ser tan simple como las que aparecen en los ejemplos que acabamos de ver. PHP contempla el uso de **condiciones múltiples** unidas mediante **operadores lógicos**. Éstos aparecen recogidos en la tabla que aparece en la figura 4.2.

Operador	Significado
&&	<b>AND.</b> Evalúa que se cumplan las dos condiciones que une.
	<b>OR.</b> Evalúa que se cumpla, al menos, una de las dos condiciones que une.
xor	<b>XOR.</b> Evalúa que se cumpla una, y sólo una, de las dos condiciones que une.

*Figura 4.2*

Si unimos dos condiciones mediante, digamos, el operador &&, se formará una condición compuesta que resultará ser cierta si las dos condiciones simples que la forman lo son a su vez. Veamos un ejemplo en el script **condicionalAnd.php**:

```
<?php
$edad=32;
if ($edad>18 && $edad<66) {
    echo ("El usuario está en edad laboral.");
} else {
    echo ("El usuario NO está en edad laboral.");
}
?>
```

Observe la condición múltiple en la línea resaltada del código. Se evalúa que el valor de la variable \$edad sea mayor que 18 y menor que 66. Si alguna de ambas condiciones resultara no ser cierta, la condición completa sería falsa. Pruebe a modificar el código estableciendo el valor de la variable en 15, por ejemplo. Resulta que la segunda condición se cumple (es menor que 66), pero la primera no. Así pues, el mensaje que aparece en la página es el que informa de que el usuario no está en edad de trabajar.

Cuando se emplea el operador || (OR) basta con que se cumpla una de las dos o más condiciones simples que forman la condición compuesta, para que ésta resulte ser cierta. Por ejemplo, suponga el siguiente condicional:

```
if ($edad>18 || $edad==16)
```

Esta condición resultará ser cierta si la variable es mayor que 18 o, si aún siendo menor, contiene el valor 16.

Por supuesto, podemos combinar varias condiciones y operadores lógicos para obtener una condición compuesta tan estricta como sea necesario. Por ejemplo, si queremos saber si una persona, en virtud de su edad, puede ocupar un

puesto de trabajo. En España, con la vigente legislación laboral, una persona puede trabajar a partir de los 18 años, o a partir de los 16 siendo necesario, en este caso, un permiso paterno. El condicional para determinar si un candidato cumple estos requisitos sería el siguiente:

```
if ($edad>17 || ($edad>15 & $permisoPaterno=true))
```

Como ve, se pueden combinar distintas condiciones, para obtener la evaluación necesaria en cada caso.

Otra posibilidad es unir dos condiciones simples mediante el operador *o exclusivo (xor)*. Esto hace que la condición compuesta sea cierta si una de las condiciones simples es cierta y la otra no. Si ninguna es cierta, o lo son las dos, la condición compuesta se evalúa como falsa.

Este tipo de condicionales admite una variante cuando hay que evaluar varias condiciones seguidas, mediante la palabra reservada *elseif*. Suponga que tiene que evaluar a qué rango de edades pertenece el usuario. Observe el script **usoDeElseif.php**:

```
<?php
$edad=32;
if ($edad<=10) {
    echo ("El usuario tiene de 0 a 10 años.");
} elseif ($edad<=20) {
    echo ("El usuario tiene de 11 a 20 años.");
} elseif ($edad<=30) {
    echo ("El usuario tiene de 21 a 30 años.");
} elseif ($edad<=40) {
    echo ("El usuario tiene de 31 a 40 años.");
} elseif ($edad<=50) {
    echo ("El usuario tiene de 41 a 50 años.");
} elseif ($edad<=60) {
    echo ("El usuario tiene de 51 a 60 años.");
} elseif ($edad<=70) {
    echo ("El usuario tiene de 61 a 70 años.");
} elseif ($edad<=80) {
    echo ("El usuario tiene de 71 a 80 años.");
} elseif ($edad<=90) {
    echo ("El usuario tiene de 81 a 90 años.");
} else {
    echo ("El usuario tiene de más de 90
años.");
}
?>
```

Cargue esta página en el navegador para comprobar su funcionamiento. Cambie el valor asignado a la variable \$edad por otros valores y recargue la página para comprobar cómo funciona. Verá que, según el valor que le asigne a la variable, le sale un resultado u otro. Tomemos un momento para aclarar el funcionamiento de un condicional que emplea la cláusula elseif. Como ve, ésta va siempre seguida de una condición. Las condiciones se evalúan en forma secuencial, de modo que, si la primera es falsa, se evalúa la segunda. Si ésta es falsa, se evalúa la tercera. Cuando una condición resulta ser verdadera, no se evalúan las siguientes. Si ninguna es verdadera, se ejecuta el último bloque de sentencias, aquél que está comprendido en la cláusula else.

Existe otro modo de hacer este tipo de evaluaciones secuenciales, mediante la estructura *switch* *O...case*. Observe el script **conmutador.php**, a continuación:

```
<?php
$pais="ITALIA";
switch ($pais) {
    case "INGLATERRA":
        echo ("El usuario es inglés.");
        break;
    case "ALEMANIA":
        echo ("El usuario es alemán.");
        break;
    case "ITALIA":
        echo ("El usuario es italiano.");
        break;
    case "ESPAÑA":
        echo ("El usuario es español.");
        break;
    case "FRANCIA":
        echo ("El usuario es francés.");
        break;

    default:
        echo ("El usuario es de una
nacionalidad no especificada en la lista.");
        break;
}
?>
```

En primer lugar, cargue el script en el navegador para ver qué ocurre. La sentencia switch evalúa una variable. Cada cláusula case evalúa un posible valor de dicha variable. La limitación es significativa. No podemos evaluar si la variable tiene un contenido mayor o menor que un valor determinado. Sólo podemos evaluar valores concretos. Fíjese en que cada cláusula case finaliza con el signo de

dos puntos (:). Además, el bloque de sentencias comprendido en cada cláusula case debe terminar siempre con la sentencia **break**. Esto evita que se sigan evaluando posibilidades cuando se ha encontrado una coincidencia. Por último, quiero llamar su atención sobre el caso específico **default**. Aquí colocamos un bloque de sentencias que se ejecutarán si no se ha dado ninguno de los casos evaluados anteriormente. La cláusula default y el correspondiente bloque de instrucciones son opcionales. Quiero que conozca, también, un detalle importante en los condicionales. Cuando la comparación se establece entre valores alfanuméricos, el intérprete los compara literalmente, teniendo en cuenta las mayúsculas y las minúsculas. Así pues, el valor "Italia" no es lo mismo que "ITALIA". Más adelante aprenderemos a obviar esta situación cuando lo necesitemos.

Para terminar con los condicionales vamos a conocer el operador ?:, llamado, también, **operador ternario**. Permite establecer uno de dos posibles valores en una variable según sea cierta o no una condición. La sintaxis genérica de este condicional es la siguiente:

```
$variable=(condición)?verdadero:falso;
```

El operador evalúa la condición. Si se cumple, le asigna a \$variable el valor verdadero. Si no se cumple, le asigna el valor falso. Una vez más, vea un ejemplo de uso en el script **operadorTernario.php**:

```
<?php  
$edad=17;  
$usuario=($edad>=18)?"El usuario es mayor de  
edad":"El usuario NO es mayor de edad";  
echo ($usuario);  
?>
```

## 4.2 BUCLES

Los bucles son unas estructuras de control que permiten que una secuencia determinada de instrucciones, conocida genéricamente como **cuerpo del bucle**, se ejecute más de una vez. Estas estructuras son muy útiles para repasar el contenido de un fichero, una matriz, una base de datos, etc. Dentro de los bucles se consideran dos tipos principales: aquéllos que se ejecutan atendiendo a una condición numérica y aquéllos que se ejecutan en función de una condición no necesariamente numérica.

A continuación vemos los dos tipos detalladamente.

### 4.2.1 Bucles mediante condición numérica

Cuando necesitamos que una secuencia de instrucciones se ejecute n veces recurrimos a la estructura *for ()*. Está definida mediante una *variable de control*, que es la encargada de determinar el número de iteraciones. La estructura es similar a la empleada en otros lenguajes de script, y obedece al siguiente patrón:

```
for (inicio; condición; paso) {  
    cuerpo del bucle;  
}
```

En este esquema, inicio es el valor inicial que se le asigna a la variable de control, condición establece el valor que se debe alcanzar para que se deje de ejecutar el bucle, y paso es la forma en que se modifica la variable de control. Dicho así quizás suene un poco críptico, así que veamos el funcionamiento mediante un código que lo aclara. Es el script **bucleFor.php**:

```
<?php  
    for ($varCon=1;$varCon <=10;$varCon++) {  
        echo ($varCon."<br />");  
    }  
?>
```

Cuando cargue este script en el navegador verá que le aparece en la página una cuenta de 1 a 10. Como la sentencia que muestra “algo” en la página es la que compone el cuerpo del bucle, inferimos que ésta se ha repetido diez veces, y de ello se ha encargado la estructura for. Veamos cómo opera. En primer lugar, vemos que, siguiendo el patrón que hemos definido anteriormente, tenemos una variable de control a la que hemos llamado \$varCon. En el primer término de la estructura establecemos que el valor inicial de la variable es 1. A continuación, tras el signo de punto y coma, indicamos que el bucle debe iterar mientras que la variable de control tenga un valor menor o igual a 10. Por último, en la tercera parte de la estructura, establecemos que, tras cada iteración, el valor de la variable de control debe ser incrementado en una unidad. Como ve, mediante esta estructura podemos definir un bucle como nos convenga en cada caso.

Llegados a este punto suelo plantearles a mis alumnos la tarea de crear un bucle como el que acabamos de hacer, pero que cuente al revés, de 10 a 1. Pues bien. Conociendo el funcionamiento de esta estructura, la modificaremos de acuerdo a nuestras necesidades. En primer lugar, si hay que contar de 10 a 1, el valor inicial de la variable de control será 10. Como condición tenemos que establecer que el bucle itere mientras el valor sea mayor o igual a 1. Por último indicamos que, tras cada iteración, se decremente la variable de control en una unidad. El bucle nos quedará como aparece en el script **descendenteFor.php**.

```
<?php
    for ($varCon=10;$varCon>=1;$varCon--) {
        echo ($varCon."<br />");
    }
?>
```

Ahora suponga que no queremos que la variable de control se modifique en una unidad en cada iteración, sino en dos. Por ejemplo, podemos necesitar la cuenta atrás que acabamos de crear sólo con los números pares. En ese caso, deberemos modificar el tercer término de nuestra estructura for, tal como aparece en el script **cambioDePaso.php**:

```
<?php
    for ($varCon=10;$varCon>=1;$varCon-=2) {
        echo ($varCon."<br />");
    }
?>
```

Como ve, la construcción de este tipo de bucles es muy simple de adaptar a las necesidades que tengamos en cada caso. Suponga, por ejemplo, que tiene una matriz que desea mostrar en la página mediante un bucle. Observe el script **matrizConFor.php**:

```
<?php
//Se crea la matriz.
$nombrer=array ("Pedro", "Ismael", "Sonia",
"Clara", "Susana", "Alfonso", "Teresa");
//Se determina el número de elementos de la matriz.
$elementos=count ($nombrer);
//Un bucle lleva la cuenta del paso por los elementos.
for ($varCon=0;$varCon<$elementos;$varCon++) {
    echo ($nombrer[$varCon]."<br />");
}
?>
```

Una vez más, fíjese en cómo se ha definido la estructura del bucle. En primer lugar, el valor inicial de la variable de control se ha establecido en cero. Esto es lógico, puesto que vamos a trabajar con una matriz y el primer elemento tiene el índice 0. Después se ha programado el bucle para que itere mientras la variable de control sea menor que el número total de elementos de la matriz. Como tiene siete nombres, el bucle se ejecutará mientras la variable de control tenga un valor comprendido entre 0 y 6.

Dentro del cuerpo del bucle mostramos un elemento de la matriz usando la variable de control como índice de la misma.

Una operación muy común y muy útil en ocasiones es el **anidamiento de bucles**. Consiste en meter un bucle dentro de otro, de forma que, para cada iteración del bucle externo, el bucle interno se ejecuta en todas sus iteraciones.

Suponga que tiene que hacer un script que simule el recorrido de un edificio de cinco plantas. En cada planta hay cuatro viviendas. Nuestro script debe recorrer cada una de las cuatro viviendas de la primera planta. Entonces pasará a la segunda y recorrerá también todas las viviendas, y así sucesivamente. El código aparece en el script **buclesAnidados.php**. Como ve, tenemos dos bucles. Cada uno usa su propia variable de control. La del bucle exterior la hemos llamado \$piso y se refiere a cada una de las plantas del edificio. La del bucle interior la hemos llamado \$puerta y alude a cada una de las viviendas de cada planta. El bucle interior constituye el cuerpo del bucle exterior.

```
<?php
    for ($piso=1;$piso<=5;$piso++) {
        for ($puerta=1;$puerta<=4;$puerta++) {
            echo ("Piso: $piso. - Puerta: $puerta.<br />");
        }
    }
?>
```

Cargue el script en el navegador para comprobar el funcionamiento.

#### 4.2.2 Bucles mediante condición no numérica

Los bucles también pueden crearse de modo que obedezcan a condiciones no necesariamente numéricas. Por ejemplo, podemos necesitar crear un bucle que recorra una matriz buscando un determinado contenido y se detenga si lo encuentra. Este tipo de iteraciones se basan en la estructura **while ()**, que permite que un bucle itere mientras se cumpla una condición. El formato genérico es el siguiente:

```
while (condición) {
    // cuerpo del bucle;
}
```

Para entender el funcionamiento observe el script que aparece listado a continuación, llamado **bucleCondicionado.php**:

```
<?php
    $ciudades=array("Madrid", "Barcelona", "Londres",
    "New York", "Los Angeles", "Chicago");
    $ciudad="";
```

```
$indice=0;
while ($ciudad!="Londres") {
    $ciudad=$ciudades[$indice];
    echo ($ciudad."<br />");
    $indice++;
}
echo ("Final de la ejecución.");
?>
```

Observe detenidamente el script. Tenemos una matriz con nombres de ciudades del mundo, un variable que actuará como índice de dicha matriz y una variable adicional donde se almacenará cada elemento, sustituyendo al anterior valor, en cada iteración del bucle. Y llegamos al bucle. Vemos, en la condición establecida, que el bucle se ejecutará mientras que la variable \$ciudad no tenga el contenido "Londres". Ejecute el script para ver cómo funciona. En la página le aparecen, uno debajo de otro, los nombres de Madrid, Barcelona y Londres. Pero, ¿cómo es eso? ¿No hemos dicho que el bucle se ejecuta mientras no se cumpla la condición de que \$ciudad sea Londres? Pues en el momento en que se encuentre este valor, el bucle debería interrumpirse, no mostrar también esta ciudad. ¿No es así? No exactamente. Para entender qué ha ocurrido, debemos analizar detalladamente el comportamiento del script, teniendo en cuenta que *la condición se evalúa al principio de cada iteración*. Veamos qué ocurre. En primer lugar, el valor de \$indice es 0 y el de \$ciudad es una cadena vacía. Llegamos a la estructura while y vemos que la condición establece que se ejecutará mientras \$ciudad no contenga "Londres". Se evalúa la condición. Como resulta ser cierta, la ejecución entra dentro del bucle. Se busca en la matriz el elemento que corresponde a \$indice (todavía es 0). Es el primer elemento de la matriz, cuyo valor es "Madrid". Este valor se almacena en \$ciudad. A continuación, esta variable se muestra en la página. La siguiente línea incrementa \$indice, con lo que pasa a valer 1. Se acaba el cuerpo del bucle, pasando a la siguiente iteración. De nuevo se evalúa la condición. Como resulta ser cierta, se entra de nuevo en el cuerpo del bucle. En la variable \$ciudad se almacena el elemento que corresponde. Como \$indice vale 1 (de la anterior iteración) el valor es Barcelona. En la siguiente línea se muestra esta ciudad en la página. A continuación, se incrementa \$indice, pasando a valer 2. En este punto termina el cuerpo del bucle, dando paso a la siguiente iteración. La variable \$ciudad todavía contiene "Barcelona" cuando se evalúa la condición, así que, una vez más, se entra en el cuerpo del bucle. En él se busca el elemento que corresponde al índice 2 de la matriz. Es Londres. Pero ya estamos dentro del cuerpo del bucle. Ya se ha evaluado la condición, que resultó ser cierta, así que esta ciudad también se muestra. A continuación se incrementa \$indice, con lo que pasa a valer 3. En la siguiente iteración se evalúa, una vez más, la condición. Esta vez, resulta ser falsa, por lo que se da por terminado el bucle y se sigue con la ejecución de la siguiente instrucción que hay por debajo del mismo.

Una de las características más llamativa de este tipo de bucles es que se puede dar el caso de que no llegue a ejecutarse ni una sola iteración. Suponga un código como `otroCondicionado.php`.

```
<?php
    $ciudades=array("Madrid", "Barcelona", "Londres",
"New York", "Los Angeles", "Chicago");
    $indice=0;
    $ciudad=$ciudades[$indice];
    while ($ciudad!="Madrid") {
        $ciudad=$ciudades[$indice];
        echo ($ciudad."<br />");
        $indice++;
    }
    echo ("Final de la ejecución.");
?>
```

Como ve, es muy parecido al anterior, sólo que la condición resulta ser falsa en la primera evaluación, con lo que no se entra dentro del cuerpo del bucle. Al ejecutarlo verá que no le aparece ninguna ciudad en la lista. Si usted quiere asegurarse de que su bucle ejecute, al menos, una iteración, deberá evaluar la condición *después* del cuerpo del bucle, en lugar de hacerlo *antes*. Para esto, PHP nos proporciona la estructura `do...while`, cuyo formato genérico es el siguiente:

```
do {
    cuerpo del bucle;
} while (condición);
```

Como ve es similar al anterior, pero cambiando el punto donde se evalúa la condición.

### 4.2.3 El bucle `foreach`

Desde la versión 4 de PHP se incluyó una instrucción específica para crear un bucle que permita recorrer una matriz completamente de una forma más fácil. Se trata de `foreach`. En los ejemplos anteriores de bucles en los que hemos manejado matrices hemos visto que nos hace falta gestionar, casi artesanalmente, una variable que actúe como índice para recorrer cada elemento de la matriz. Esta nueva estructura, que se mantiene vigente en la versión 5, nos evita esa necesidad. Además, con las estructuras que hemos estudiado no podemos hacer ese recorrido cuando se trata de una matriz asociativa. En cambio `foreach` nos resuelve esto también. Este bucle responde al siguiente formato genérico:

```
foreach ($matriz as $clave => $valor){  
    cuerpo del bucle;  
}
```

Este bucle se repite para cada elemento de la matriz (en inglés las palabras “for each” significan “para cada uno”). Veamos cómo funciona. Observe el script **recorreMatriz.php**.

```
<?php  
    $ciudades=array("Madrid", "Barcelona", "Londres",  
"New York", "Los Angeles", "Chicago");  
    foreach ($ciudades as $clave => $valor){  
        echo ("La ciudad con el índice $clave tiene  
el nombre $valor. <br />");  
    }  
    echo ("Final de la ejecución.");  
?>
```

Al cargar este script en el navegador veremos cómo aparecen listados todos los elementos de la matriz. Como ve, no hemos tenido que preocuparnos de incrementar el índice. El bucle hace un recorrido secuencial sin intervención de una variable específica. Además, en cada iteración se almacenan, en las variables \$clave y \$valor el índice de la matriz y el contenido del correspondiente elemento. Por supuesto, esto funciona también aunque la matriz sea asociativa, tal como hemos dicho antes. Observe el script **recorreAsociativa.php**:

```
<?php  
    $ciudades=array("md"=>"Madrid",  
"ba"=>"Barcelona", "lo"=>"Londres", "ny"=>"New York",  
"la"=>"Los Angeles", "ch"=>"Chicago");  
    foreach ($ciudades as $clave => $valor){  
        echo ("La ciudad con el índice $clave tiene  
el nombre $valor. <br />");  
    }  
    echo ("Final de la ejecución.");  
?>
```

Como ve el uso de foreach para recorrer una matriz es muy cómodo. Además, PHP nos permite anidar bucles foreach para poder recorrer matrices con más de un índice. Para ver un ejemplo, suponga que usted tiene una matriz con los datos de tres amigos suyos, que viven en distintas ciudades. Los datos de cada amigo (nombre, edad y teléfono) constituyen una matriz, cada una de esas matrices es, a su vez, un elemento de otra matriz, identificado por el nombre de cada ciudad. Desde luego, éste no es el sistema más eficiente para gestionar su agenda. PHP le ofrece mejores opciones, como iremos viendo más adelante, pero centrándonos en

el valor didáctico, este ejemplo nos muestra cómo recorrer completamente una matriz de dos índices de una forma útil y práctica, como verá en su momento. Preste especial atención al uso que hemos hecho de los bucles en el script **recorrerAnidados.php**:

```
<?php
    $amigos=array("Madrid"=>array("nombre"=>"Pedro",
    "edad"=>32, "telefono"=>"91-999.99.99"),
    "Barcelona"=>array("nombre"=>"Susana", "edad"=>34,
    "telefono"=>"93-000.00.00"),
    "Toledo"=>array("nombre"=>"Sonia", "edad"=>42,
    "telefono"=>"925-09.09.09"));
    foreach ($amigos as $ciudad => $cadaUno) {
        echo ("En $ciudad tiene el amigo:<br />");
        foreach ($cadaUno as $clave => $valor) {
            echo ("$clave." : ".$valor."<br />"); }
    }
    echo ("Final de la ejecución.");
?>
```

Empiece por cargar el script en el navegador, para comprobar su funcionamiento. Verá el listado de sus amigos en cada ciudad. Fíjese en el bucle `foreach` externo. Se recorre la matriz principal, llamada `$amigos`, recuperando el índice de cada elemento (que es el nombre de cada ciudad) en la variable `$ciudad`. El contenido de cada elemento, que es una matriz con los datos del amigo que vive en esa ciudad, se recupera en `$cadaUno`. Así pues, en cada iteración del bucle externo, `$cadaUno` contiene una matriz con los datos de una persona. En el bucle interno se recorre esta matriz, recuperando cada dato individual.

#### 4.2.4 Interrupciones y reiteraciones

Existen dos sentencias específicas para alterar la ejecución de un bucle. Se trata de `break` y `continue`. Ya vimos un uso de `break` cuando conocimos la estructura condicional `switch`. Aquí, en los bucles, tiene un uso similar. Interrumpe la ejecución del bucle aunque éste no se haya completado. La sentencia `continue`, por el contrario, interrumpe la ejecución de la iteración actual, pasando a la siguiente.

Veamos un par de ejemplos para comprender el funcionamiento de estas sentencias. En primer lugar, suponga que quiere recorrer la matriz del ejercicio anterior, mostrando cada elemento, hasta que encuentre la ciudad Londres. En ese momento, desea salir del bucle, siguiendo la ejecución en la siguiente línea. Observe el script **romperBucle.php**, que hace, exactamente, eso.

```
<?php  
    $ciudades=array("md"=>"Madrid",  
    "ba"=>"Barcelona", "lo"=>"Londres", "ny"=>"New York",  
    "la"=>"Los Angeles");  
  
        foreach ($ciudades as $clave => $valor){  
            if ($valor=="Londres") break;  
            echo ("La ciudad con el índice $clave tiene  
el nombre $valor. <br />");  
        }  
        echo ("Final de la ejecución.");  
?>
```

Cárguelo en el navegador para comprobar su funcionamiento. Observe que, en cada iteración, se comprueba si el contenido de \$valor es “Londres”. Cuando se cumple la condición, se ejecuta la sentencia break, encargada de interrumpir el bucle y se pasa la ejecución a la siguiente línea inmediatamente por debajo de éste.

Ahora suponga que quiere mostrar todos los elementos de la matriz, excepto el que corresponde a Londres. Observe el script **continuarBucle.php**:

```
<?php  
    $ciudades=array("md"=>"Madrid", "ba"=>"Barcelona",  
    "lo"=>"Londres", "ny"=>"New York", "la"=>"Los Angeles");  
        foreach ($ciudades as $clave => $valor){  
            if ($valor=="Londres") continue;  
            echo ("La ciudad con el índice $clave tiene el  
nombre $valor. <br />");  
        }  
        echo ("Final de la ejecución.");  
?>
```

Observe la línea resaltada. Cuando \$valor contiene “Londres” se interrumpe la ejecución de la iteración actual, para pasar a la siguiente, con lo que, después de “Barcelona”, la siguiente ciudad que se muestra es “New York”. Ejecute el script para comprobar su funcionamiento.

## 4.3 FUNCIONES

A lo largo de lo que hemos estudiado hasta el momento, hemos visto que PHP incorpora una serie de funciones. Son instrucciones que aparecen seguidas de un paréntesis en el que pueden recibir o no algún argumento, y que pueden devolver o no algún valor. En el capítulo anterior conocimos algunas como, por

ejemplo, settype ()). Lo cierto es que PHP incorpora muchísimas funciones para distintos cometidos. En los próximos capítulos iremos descubriendo que son más de lo que ahora imagina, y que resuelven muchas situaciones. Pero, en ocasiones, se nos presenta la necesidad de recurrir a funciones que no están implementadas en el lenguaje. Dada la naturaleza open source de PHP, usted se puede descargar de Internet (<http://www.php.net>) el código fuente del intérprete, modificarlo para adaptarlo a sus necesidades y volver a compilarlo... si es usted un avezado programador con cientos de horas de experiencia, claro. La mayoría de nosotros, sin embargo, no nos atrevemos a meter las manos en un pastel de esa envergadura. Afortunadamente PHP nos proporciona una solución válida para esas situaciones. Si necesitamos una función que no forme parte del lenguaje podemos definirla en nuestro script, con los conocimientos que ya tenemos y los que vamos a adquirir a lo largo de este libro. Para ello, usamos la instrucción **function ()**, que permite definir una función con el código que nosotros deseemos y ejecutarla, cuando sea necesario, con una simple llamada. Dentro de la función pondremos las instrucciones que sean necesarias para que haga lo que nosotros queremos. Estas instrucciones constituyen el **cuerpo de la función**. La definición de una función responde al siguiente esquema genérico:

```
function nombreDeLaFuncion (arg1, arg2,..., argn) {  
    //cuerpo de la función  
}
```

Al definir la función, se establece su nombre y, a través de él, se la podrá invocar en cualquier parte del script que sea necesaria. Los argumentos arg1, arg2, etc., son opcionales. Se emplearán o no, dependiendo de que la función los necesite para hacer aquello para lo que haya sido creada. Veamos un código de ejemplo muy simple para ir asentando ideas. Es el script **funcionNoEjecutada.php**:

```
<?php  
    function mostrarUnNumero(){  
        echo ("Número 2");  
    }  
?>
```

Cargue el script en el navegador. Verá que, simplemente, no aparece nada en la página. La línea que tenía que mostrar "Número 2" no lo muestra. Esto es porque, cuando se define una función, esta se carga en memoria, pero su contenido no se ejecuta hasta que dicha función es invocada por su nombre, como si ese nombre fuera una instrucción más de PHP. Modifiquemos el código para invocar la función, tal como se ve en el script **funcionEjecutada.php**:

```
<?php  
    function mostrarUnNumero(){
```

```
    echo ("Número 2");
}
mostrarUnNúmero();
?>
```

Observe cómo hemos invocado la función, en la línea que aparece resaltada. Cuando cargue este script verá como sí se ejecuta el cuerpo de la función. Un detalle curioso acerca de las funciones es que, excepcionalmente, el nombre de éstas no es sensible a mayúsculas y minúsculas, de forma que el listado anterior habría funcionado igual si la función la invocamos así: mostrarunnumero();. De todas formas esta es una licencia que le desaconsejo, pues es muy posible que en futuras versiones del intérprete esto cambie.

### 4.3.1 Pasando argumentos

A las funciones se les pueden pasar argumentos para adaptar su operativa a cada caso. En realidad esto no es del todo nuevo. Ya lo hemos visto hacer con algunas de las funciones propias del lenguaje. Cuando, por ejemplo, invocamos a gettype (\$dato) para recuperar el tipo de valor contenido en una variable, el nombre de esa variable, colocado entre los paréntesis, constituye el **argumento**, o **parámetro** que le pasamos a la función. Cuando se trata de nuestras propias funciones, el sistema es el mismo. Debemos invocar a la función pasándole el argumento (o los argumentos, puede requerir más de uno) entre paréntesis. Pero además, al definir la función, es necesario especificar qué argumentos debe recibir. Vea, en la página siguiente, el script **funcionConArgumento.php**:

```
<?php
function mostrarUnNúmero($mostrar) {
    echo ("Número $mostrar");
}
$numero=3;
mostrarUnNúmero($numero);
?>
```

Ejecute el script. Vea que le muestra en la página el número 3. Observe en las líneas resaltadas cómo hemos definido la función y cómo la hemos invocado. En la invocación hay un argumento que es la variable \$numero. Cuando se efectúa esta llamada a la función esta variable se aloja, dentro de la función, en la variable \$mostrar, que aparece en la definición. Observe, dentro del cuerpo de la función cómo tomamos la variable mostrar, que ha recibido el valor de \$numero.

Siempre debe haber el mismo número de argumentos en la definición de una función que en la invocación a la misma, y éstos aparecerán separados por comas. Observe el script **funcionDosArgumentos.php**:

```
<?php
    function producto($valor1, $valor2){
        $resultado=$valor1*$valor2;
        echo ("Resultado: $resultado");
    }
    $numero1=3;
    $numero2=4;
    producto($numero1, $numero2);
?>
```

Observe las líneas resaltadas para comprobar cómo se definen e invocan funciones que usan más de un argumento.

Sin embargo, desde la versión 4 de PHP es posible pasarle a una función un número mayor de argumentos de los que está diseñada para recibir. Los argumentos, en este caso, se gestionarán mediante tres funciones que PHP implementa al efecto. Son *func\_num\_args()*, *func\_get\_arg()* y *func\_get\_args()*. La primera devuelve el número total de argumentos pasados a nuestra función. La segunda recibe un número y devuelve ese argumento, y la tercera devuelve todos los argumentos en una matriz indexada. Para comprender la operativa de estas funciones veamos el script **manejoDeArgumentos.php**:

```
<?php
    function comprobarArgumentos(){
//Uso de func_num_args().
        $numeroDeArgumentos=func_num_args();
        echo ("<u>Uso de func_num_args().</u><br
/>");
        echo ("El número de argumentos es
$numeroDeArgumentos<br />");
        echo ("<br />");

//Uso de func_get_arg().
        echo ("<u>Uso de func_get_arg().</u><br
/>");
        for ($contador=0,
$contador<$numeroDeArgumentos; $contador++){
            $argumento=func_get_arg($contador);
            echo ("El argumento $contador tiene
el valor $argumento<br />");
        }
        echo ("<br />");

//Uso de func_get_args()
        echo ("<u>Uso de func_get_args().</u><br
/>");
```

```
$matrizDeArgumentos=func_get_args();
for ($contador=0;
$contador<$numeroDeArgumentos; $contador++){
    $argumento=$matrizDeArgumentos[$contador];
    echo ("El argumento $contador tiene
el valor $argumento<br />");
}
}

$variable1=1;
$variable2="HOLA";
$variable3="ADIÓS";

comprobarArgumentos($variable1, $variable2,
$variable3);
?>
```

Ejecute el script y vea que el resultado se corresponde con el uso de cada una de estas funciones. De este modo, podemos usar estos argumentos aunque no se hayan declarado en la definición de la función. Es importante aclarar que estas tres funciones han sido creadas para ser usadas dentro de las funciones que defina el usuario. Si intenta usarlas en el cuerpo general del script, fuera de una función, se producirá un error con un mensaje como el siguiente:

**Warning:** func\_num\_args(): Called from the global scope - no function context in C:\Documents and Settings\Jose\Mis documentos\Mis webs dinamicas\capítulo\_04\listado.php on line 23

Cuando se le pasan argumentos a una función, éstos se pueden pasar *por valor* o *por referencia*. Veamos qué es esto y cómo funciona. Para ello fíjese en el script *pasoPorValor.php*:

```
<?php
function cuadrado($numero){
    $numero*=$numero;
    echo ("En la función el número se eleva al
cuadrado, obteniendo el resultado: $numero<br />");
}
$numero=3;
echo ("El número, antes de invocar la función
vale: $numero<br />");

cuadrado($numero);
echo ("El número, después de invocar la función
vale: $numero");
?>
```

Al ejecutarlo obtiene el resultado de la figura 4.3.

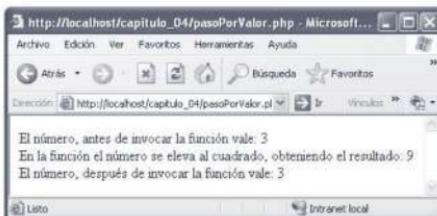


Figura 4.3

Fíjese en que tenemos una variable llamada \$numero que se declara fuera de la función, asignándole el valor 3. Esta variable se pasará como argumento a la función y, en la definición de la misma, aparece también la variable \$numero. Sin embargo, aunque tiene el mismo nombre, no es la misma variable. La que aparece en la definición de la función pertenece, *exclusivamente*, a la función, mientras que la que aparece *fuera* de la función pertenece a todo el script. Para entender esto observe lo que ocurre cuando usted carga el script en el navegador:

En primer lugar se ejecuta la línea:

```
$numero=3;
```

Como ya sabemos, la función se ha cargado en memoria, pero no será ejecutada mientras no se invoque. A continuación, se ejecuta la siguiente línea:

```
echo ("El número, antes de invocar la función vale:  
$numero<br />");
```

Esto nos muestra en la página el valor de la variable, en la primera línea de resultado que ve en la figura 4.3.

Ahora se invoca la función, mediante la linea:

```
cuadrado($numero);
```

Dentro de la función, el valor de \$numero se eleva al cuadrado (se multiplica por sí mismo). Esto ocurre en la linea:

```
$numero*=$numero;
```

Entonces se muestra el valor que adquiere dentro de la función, así:

```
echo ("En la función el número se eleva al cuadrado,  
obteniendo el resultado: $numero<br />");
```

Corresponde a la segunda línea de resultados que aparece en la página.

Cuando termina la función, continúa la ejecución del script, con la línea:

```
echo ("El número, después de invocar la función vale:  
$numero");
```

Entonces se muestra la tercera línea de resultados en la página.

De lo que quiero que se dé cuenta es de que el valor de la variable, que al inicio del script es de 3, sigue inmutable al final del script, independientemente de lo que haya ocurrido en la función. Esto es así porque, cuando se invoca la función no se le pasa la variable del argumento, sino sólo su valor. Es decir, al invocar la función tal como hacemos aquí, no le pasamos como argumento la variable \$numero, sino el valor 3 (o el que tuviera en ese momento). Por esta razón, la función no opera con la variable \$numero del script, sino con el valor 3. Esto es lo que se llama **paso de argumentos por valor**.

Como contraparte está el **paso de argumentos por referencia**, en el que sí se pasa la variable original y se opera con ella, de modo que todo lo que hagamos en la función afectará a dicha variable original. Para indicar que un argumento se va a pasar por referencia se le antepone el signo & en la definición de la función. Vea un ejemplo en el script **pasoPorReferencia.php**:

```
<?php  
function cuadrado(&$numero){  
    $numero*=$numero;  
    echo ("En la función el número se eleva al  
cuadrado, obteniendo el resultado: $numero<br />");  
}  
  
$numero=3;  
echo ("El número, antes de invocar la función  
vale: $numero<br />");  
  
cuadrado($numero);  
echo ("El número, después de invocar la función  
vale: $numero");  
?>
```

Fíjese en la línea resaltada para ver cómo se hace que un argumento pase por referencia. Ahora observe el resultado en la figura 4.4.

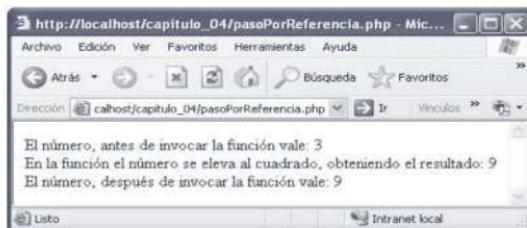


Figura 4.4

La diferencia está en la tercera linea de resultados que aparece en la página. Como ve, esta vez la variable, vista desde fuera de la función, sí ha sido alterada por lo que ha ocurrido dentro de la función.

Cada vez que usted invoque la función que ha sido definida de este modo, el argumento se pasará por referencia. También puede ocurrir que usted necesite, llegado el caso, crear una función en la que los argumentos sean pasados por valor, pero en alguna invocación concreta del script deba pasarlos por referencia. En ese caso, defina la función normalmente, así:

```
function miFuncion ($arg1, $arg2, $arg3){
    cuerpo de la function;
}
```

Cuando necesite pasar argumentos por valor (la mayoría de las veces es así) invóquela normalmente, como se ve a continuación:

```
miFuncion ($arg1, $arg2, $arg3);
```

Pero ahora suponga que necesita hacer una llamada pasando los argumentos \$arg2 y \$arg3 por referencia. No es necesario que redefina la función. Bastará con que la llame así:

```
miFuncion ($arg1, &$arg2, &$arg3);
```

### 4.3.2 Retorno desde una función

En muchas ocasiones las necesidades operativas de nuestros scripts requieren que una función devuelva un valor. Esto se consigue mediante la

instrucción **return** seguida de la variable que contiene dicho valor. Para entender cómo opera esto veamos un ejemplo en el script **funcionConResultado.php**:

```
<?php
    function cuadrado($numero) {
        $numero*=$numero;
        return $numero;
    }
    $numero=3;
    $cuadrado=cuadrado($numero);
    echo ("El cuadrado del número $numero es
$cuadrado.");
?>
```

Ejecute el script para ver cómo funciona y examine el código. Fíjese en la línea resaltada. Es la que devuelve un dato a la instrucción que invocó a la función. En este caso, esta instrucción es una asignación, como puede ver. La instrucción **return** sólo permite “sacar” un valor de la función. Es decir, esta instrucción no puede ir seguida por más de un nombre de variable. Si usted, en algún momento, necesita que la función devuelva al script más de un valor, lo que hará será crear una matriz, dentro del cuerpo de la función, con los valores que necesite, y hacer que la instrucción **return** vaya seguida por el nombre de esa matriz. También puede poner la instrucción **return** sin ninguna variable a continuación. Esto ocasiona que se interrumpa la ejecución de la función y se continúe la ejecución del script. Sin embargo, ésta es una solución que pocas veces necesitará. Normalmente, este uso de **return** está supeditado a un condicional, de tal modo que si se cumple una condición determinada deba abandonarse la función.

### 4.3.3 Ámbito de las variables

Cuando una variable se crea dentro del cuerpo de una función sólo existe dentro de esa función, no en el cuerpo principal del programa.

Vamos a ver ahora cómo funciona esto, en **datosEnFunciones.php**.

```
<?php
    function cuadrado($numero) {
        $cuadrado=$numero*$numero;
        echo ("El cuadrado de $numero es
$cuadrado.<br />");
    }
    $numero=3;
    cuadrado($numero);
```

```

echo ("La variable \$numero vale $numero y es de
tipo ".gettype ($numero)."<br />"); 
echo ("La variable \$cuadrado vale $cuadrado y es
de tipo ".gettype ($cuadrado)."<br />"); 
?>

```

Cuando ejecute este script verá el resultado de la figura 4.5.

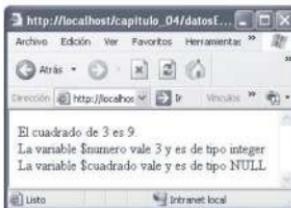


Figura 4.5

Lo que nos interesa es lo siguiente: dentro de la función se ha declarado la variable \$cuadrado. Esta variable almacena un resultado que se muestra mediante la instrucción echo () que forma parte del cuerpo de la función. Cuando se termina la ejecución de la función se continúa con el script. Vemos que hay una línea destinada a mostrar el valor de \$cuadrado y su tipo. Y no nos muestra ningún valor, y como tipo nos muestra NULL. Es decir, la variable no existe. Y, sin embargo, ha sido declarada y utilizada dentro del cuerpo de la función, que acaba de ejecutarse. Lo que vemos con esto es que una variable que se declara dentro de una función no existe fuera de la misma. Es una **variable local**, es decir, de ámbito privado de esa función. Por el contrario, una **variable global**, declarada en el cuerpo principal del script, está disponible para todo el script... excepto para las funciones, tal como muestra el script **ambito.php**:

```

<?php
function cuadrado($valor){
    $cuadrado=$valor*$valor;
    echo ("El cuadrado de $valor es
$cuadrado.<br />"); 
    echo ("La variable \$numero dentro de la
función vale $numero y es de tipo ".gettype ($numero)."<br
/>"); 
}
$numero=3;
cuadrado($numero);

```

```
echo ("La variable \$numero fuera de la función  
vale $numero y es de tipo ".gettype ($numero)."  
echo ("La variable \$cuadrado fuera de la función  
vale $cuadrado y es de tipo ".gettype ($cuadrado)."  
?>
```

Cuando lo ejecute verá un resultado como el que aparece en la figura 4.6.

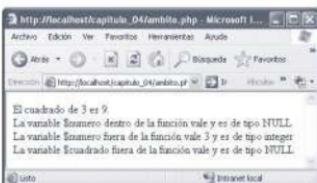


Figura 4.6

Como puede ver, la variable creada en el cuerpo del script no es visible desde dentro de la función. Por esto es que, cuando necesitamos pasar datos del script a la función, usamos los argumentos y, cuando necesitamos pasar un resultado desde la función al cuerpo del script, usamos la instrucción return.

Si queremos usar una variable global dentro del cuerpo de una función, la declararemos con la sentencia **global**, dentro de la función. De este modo, el intérprete "sabe" que esta variable también está disponible dentro de la función. Observe el script **variablesGlobales.php**, que es una variante del anterior.

```
<?php  
  
function cuadrado($valor){  
    global $numero;  
    $cuadrado=$valor*$valor;  
    echo ("El cuadrado de $valor es  
$cuadrado.<br />");  
    echo ("La variable \$numero dentro de la  
función vale $numero y es de tipo ".gettype ($numero)."  
?> ;  
}  
  
$numero=3;  
cuadrado($numero);  
echo ("La variable \$numero fuera de la función  
vale $numero y es de tipo ".gettype ($numero)."  
?> ;
```

```
echo ("La variable \$cuadrado fuera de la función  

vale $cuadrado y es de tipo ".gettype ($cuadrado)."<br />");  

?>
```

Observe la línea resaltada. Vea el resultado de este script en la figura 4.7.

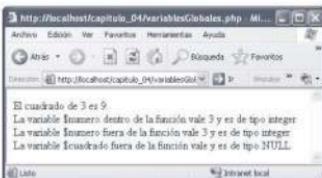


Figura 4.7

Como ve, la variable \$numero ha sido declarada como global dentro del cuerpo de la función. Eso hace que esté disponible tanto para dicha función como para el resto del código del script. Cuando una variable es de ámbito global puede ser llamada mediante una matriz específica de PHP. Se trata de **\$GLOBALS**. Esta matriz es propia del intérprete, es decir, no es necesario declararla, y almacena todas las variables globales que se usan en el script. Cada vez que usted ejecuta un script, esta matriz se crea automáticamente. Se trata de una matriz asociativa en la que se crea un elemento por cada variable global que usa el script. El índice de cada elemento es el nombre de la variable y el contenido es el valor de esa variable. Usted puede usar esta matriz, como ve en el script **matrizDeGlobales.php**:

```
<?php  

function cuadrado($valor){  

    $cuadrado=$valor*$valor;  

    echo ("El cuadrado de $valor es $cuadrado.<br />");  

    echo ("La variable global numero dentro de la  

función vale ".$GLOBALS ["numero"]." y es de tipo ".gettype  

($GLOBALS ["numero"])."<br />");  

}  

$numero=3;  

cuadrado($numero);  

echo ("La variable \$numero fuera de la función vale  

$numero y es de tipo ".gettype ($numero)."<br />");  

echo ("La variable \$cuadrado fuera de la función  

vale $cuadrado y es de tipo ".gettype ($cuadrado)."<br />");  

?>
```

Observe en la línea resaltada cómo hemos usado la matriz \$GLOBALS.

#### 4.3.4 Variables estáticas

Las variables empleadas en las funciones tienen una característica. Si la función es invocada más de una vez durante la ejecución del script, las variables locales de la función no conservan su valor de una llamada a otra. Para ver claro a qué nos referimos observe el script **estaticas.php**:

```
<?php
    function estaticas(){
        $variableNormal+=5;
        echo ("El valor de \$variableNormal al
comienzo de la función es $variableNormal.<br />");
        $variableNormal*=2;
        echo ("Al duplicar \$variableNormal su
valor es $variableNormal.<br />");
    }

    echo ("<u>Primera ejecución de la función.</u><br
/>");
    estaticas();

    echo ("<u>Segunda ejecución de la función.</u><br
/>");
    estaticas();
?>
```

Observe el cuerpo de la función. En la primera línea, se declara una variable y se le añaden cinco unidades. Como dicha variable no existe previamente, y estamos haciendo con ella una operación aritmética, PHP sobreentiende que es una variable numérica y le asigna, inicialmente, el valor cero. Al incrementarla en cinco unidades pasa a tener el valor cinco, tal como se muestra en la segunda línea. Después se multiplica por dos, con lo que pasa a valer diez, tal como se muestra en la cuarta línea. Fíjese en que se invoca dos veces a la función y en ambas ocasiones el resultado es el mismo. Esto quiere decir que el valor de la variable local \$variableNormal se pierde al terminar la ejecución de la función. No se conserva de una ejecución a otra. Este es el funcionamiento normal, y realmente será el que necesitemos en la mayoría de los casos. Pero también puede ser que necesitemos que una variable local conserve su contenido de una ejecución de la función a otra. Para ello usamos la instrucción **static**. Observe el script **usoDeEstaticas.php**:

```
<?php
    function estaticas(){
        static $variableNormal;
        $variableNormal+=5;
```

```
        echo ("El valor de \$variableNormal al  
comienzo de la función es $variableNormal.<br />");  
        $variableNormal*=2;  
        echo ("Al duplicar \$variableNormal su  
valor es $variableNormal.<br />");  
    }  
    echo ("<u>Primera ejecución de la función.</u><br  
/>");  
    estaticas();  
    echo ("<u>Segunda ejecución de la función.</u><br  
/>");  
    estaticas();  
?>
```

Ahora observe el resultado. La primera vez que se llama a la función, como \$variableNormal no existe, PHP la crea y, cuando se le dice que la incremente en cinco unidades, le queda el valor cinco. Al multiplicarla por dos, le queda el valor diez. Como la variable ha sido declarada como estática, este valor se conserva, de modo que, al llamar a la función por segunda vez, cuando se incrementa en cinco unidades, queda con el valor quince. Al multiplicarla por dos, se le asigna el valor treinta.

### 4.3.5 Recursividad

Un uso especial de las funciones definidas por el usuario es el que se conoce como **recursividad**, o **uso recursivo**. Consiste en que dentro del cuerpo de la función se vuelve a invocar a dicha función, con lo que se inicia otra ejecución de la misma, y así sucesivamente. La verdad es que es un recurso poco utilizado pero, por ejemplo, nos va a venir bien para calcular el factorial de un número. Como ya sabe, esta operación consiste en multiplicar un número por todos los que hay menores que él, excepto el 1 y el 0. El factorial de 5 será  $5 \cdot 4 \cdot 3 \cdot 2 = 120$ . Observe el script **recursivas.php**:

```
<?php  
  
function factorial($num) {  
    if ($num==0) {  
        return 1;  
    } else {  
        $valor = $num * factorial ($num-1);  
        return $valor;  
    }  
}  
  
$original=6;
```

```
$valor=factorial($original);
echo ("El factorial de $original es $valor.<br>
/>");  
?>
```

Ejecute el script para ver su funcionamiento. Observe, en la linea resaltada, como la función se llama a sí misma.

## 4.4 OPERADORES A NIVEL DE BIT

Cuando vimos los condicionales, al principio de este capítulo, hablamos de los operadores lógicos **&&** (and), **||** (or), y **xor** (or exclusivo). Existen unas variantes de estos operadores llamadas **operadores a nivel de bit**. He dejado este tema para el final del capítulo a fin de tenerlo convenientemente aislado, desde un punto de vista meramente didáctico.

Usted sabe que cualquier valor es manejado internamente por el ordenador como una secuencia binaria (ceros y unos). Cada uno de estos dígitos binarios es un bit, es decir, la menor unidad de información que puede manejar el ordenador. Nosotros podemos trabajar, desde PHP, con dos (o más) valores a nivel de bit. Para ello contamos con tres operadores que son los siguientes:

- Operador **&**. Comprueba cada bit de un valor con el correspondiente bit de otro. Si ambos son 1, el resultado de ese bit es 1. En caso contrario, el resultado es 0.
- Operador **|**. Comprueba cada bit de un valor con el correspondiente bit de otro. Si uno de los dos es uno, o lo son ambos, el resultado de ese bit es 1. En caso contrario, el resultado es 0.
- Operador **^**. Comprueba cada bit de un valor con el correspondiente bit de otro. Si uno de los dos es 1 y el otro es 0, el resultado de ese bit es 1. En caso contrario, el resultado es 0. Es decir, el resultado es uno si los dos bits son diferentes y 0, si son iguales.

Veamos qué significa esto. Consideremos dos valores: 19 y 13. Su representación en binario es la siguiente:

$$\begin{aligned}19_{(2)} &= 10011 \\13_{(2)} &= 01101\end{aligned}$$

Ahora vamos a realizar las tres operaciones lógicas bit a bit. Esto significa que se hará cada operación del primer bit de la derecha de un valor con el primer bit de la derecha del otro, el segundo bit de un valor con el segundo bit del otro y,

asi, sucesivamente. Veamos los resultados, tras cada una de las operaciones propuestas, como se muestra a continuación:

$19_{(2)} =$	<b>10011</b>
$13_{(2)} =$	<b>01101</b>
<b>Operación &amp;:</b>	<b>00001 que, en base 10 es 1.</b>
<b>Operación  :</b>	<b>11111 que, en base 10 es 31.</b>
<b>Operación ^:</b>	<b>11110 que, en base 10 es 30.</b>

Para comprobar que todo esto funciona contamos con el script **logicosDeBit.php**, cuyo listado es el siguiente:

```
<?php
// Se definen los valores con los que trabajar.
$valor_1=19;
$valor_2=13;
// Se realizan las operaciones lógicas de bit.
$resultadoAnd=$valor_1 & $valor_2;
$resultadoOr=$valor_1 | $valor_2;
$resultadoXor=$valor_1 ^ $valor_2;
// Se muestran los valores y los resultados.
echo ("El valor 1 es: ". $valor_1."<br />");
echo ("El valor 2 es: ". $valor_2."<br />");
echo ("El resultado del valor 1 & el valor 2 es: ".
$resultadoAnd."<br />");
echo ("El resultado del valor 1 | el valor 2 es: ".
$resultadoOr."<br />");
echo ("El resultado del valor 1 ^ el valor 2 es: ".
$resultadoXor."<br />");?
>
```

Ejecútelo en el navegador y verá el resultado de la figura 4.8 en su página:

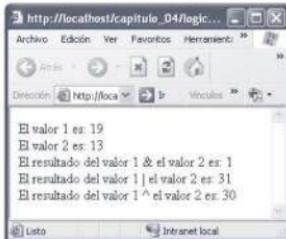


Figura 4.8

Teniendo en cuenta que PHP interpreta, en condiciones lógicas, un valor 0 como false y cualquier otro como true, podemos usar los operadores a nivel de bit en evaluaciones condicionales. Por ejemplo, suponga una línea de código como la siguiente:

```
if ($valor_1 & $valor_2)
```

Si el resultado de la operación es 1 o superior, la condición se evalúa como true. En caso de que el resultado sea 0, la condición se evalúa como false.

## CAPÍTULO 5

# USO DE FORMULARIOS

---

---

A lo largo de los capítulos anteriores hemos empezado a vislumbrar una pequeña parte de la flexibilidad que PHP nos ofrece para manejo de datos. Sin embargo, todavía no sabemos cómo proporcionarle a nuestro script los valores con los que debe trabajar. Hasta ahora hemos almacenado esos valores muy pobemente como parte del código. Sin embargo, es lógico que el usuario deba pasarse a un script algunos datos a través de un formulario.

### 5.1 ENVÍO DE DATOS DESDE UN FORMULARIO

Hemos mencionado anteriormente el caso de que alguien pueda querer ver un listado de empresas de un determinado sector en su ciudad. Así pues, el usuario deberá poder decir qué sector es el que le interesa. Esto se puede hacer mediante un formulario que contenga un campo de tipo lista donde se pueda elegir una opción. Esa opción se pasa al script PHP para que, a su vez, la procese como sea necesario (de esto último ya nos ocuparemos más adelante). Observe el script **elegirSector.htm**, que contiene un formulario con una lista de posibles sectores. En realidad he abreviado la lista para que no ocupe mucho, pero para el ejemplo nos sirve perfectamente.

```
<html>
<head>
</head>
<body>
<form name="f_prof" id="f_prof"
action="elegirSector.php" method="post">
    Elija un sector:
```

```

        <select name="sector" id="sector">
            <option value="0">Electricistas</option>
            <option value="1">Fontaneros</option>
            <option value="2">Transportistas</option>
            <option value="3">Aseguradores</option>
        </select>
        <input type="submit" value="ENVIAR" name="ok"
id="ok">
    </form>
</body>
</html>

```

Cuando empleamos un formulario donde el usuario debe escribir algo en campos de texto, o seleccionar opciones de una lista, o marcar casillas de verificación, etc., dicho formulario tiene, como atributo action, el nombre del script que recopilará los datos del usuario y los procesará en el servidor.

Para cargar esta página en el navegador use la notación que ya conoce: teclee, en la barra de direcciones, [http://localhost/capítulo\\_05/elegirSector.htm](http://localhost/capítulo_05/elegirSector.htm) y pulse Intro. El aspecto de la página es similar al que se ve en la figura 5.1.



Figura 5.1

Vea que tiene una lista desplegable en la que puede elegir una opción. Fíjese en el listado. La lista se llama *sector*. Tiene cuatro opciones cuyos valores van de cero a cuatro. Elija una opción y pulse el botón ENVIAR. En este momento, el formulario llama al script cuyo nombre tiene en el atributo action, y le pasa una variable con el nombre del campo select y el valor de la opción seleccionada. Si el formulario tuviera más campos, pasaría una variable por cada campo.

El script está en **elegirSector.php**, que aparece a continuación:

```
<?php
    $sectores=array("Electricistas","Fontaneros","Tra
nsportistas","Aseguradores");
    echo ("La opción elegida es: $sector<br />");
    echo ("La profesión correspondiente es:
$sectores[$sector].");
?>
```

En la primera linea defino una matriz que contiene las profesiones. Fíjese en que están dispuestas de tal modo que cada una tiene un índice que coincide con el atributo value de las option de la página anterior.

En la segunda línea muestro el valor de la variable \$sector. Esta variable no está declarada en ninguna parte del script, puesto que ha sido recibida desde el formulario de la página elegirSector.htm. Cuando un script es llamado por un formulario, dicho script se inicia, de modo automático, con las variables que corresponden a los campos de dicho formulario. Cada variable tiene el mismo nombre que tenga el campo correspondiente, y tiene el valor que el usuario haya tecleado en dicho campo.

Por último, en la tercera línea muestro el elemento de la matriz que corresponde a la opción elegida. Pruebe esto varias veces, seleccionando distintas opciones, para ver que funciona correctamente.

Vamos a ver otro ejemplo, para afianzar mejor la comprensión del funcionamiento de este mecanismo. Observe el script **formularioSimple.htm**:

```
<html>
    <head>
    </head>
    <body>
        <form name="f_prof" id="f_prof"
action="formularioSimple.php" method="post">
            Teclee su nombre:
            <input type="text" name="nombre" id="nombre">
            <br />
            ¿Es usted soltero?
            <input type="checkbox" name="soltero"
id="soltero" value="SI">

            <br />
            Seleccione su edad:
            <br />
            de 0 a 20 años:
            <input name="edad" type="radio" value="1">
```

```
<br />
de 21 a 40 años:
<input name="edad" type="radio" value="2">
<br />
más de 40:
<input name="edad" type="radio" value="3">
<br />
<input name="datoOculto" type="hidden"
id="datoOculto" value="Este es un campo oculto">
<p>
    <input type="submit" value="ENVIAR" name="ok">
<id="ok">
</p>
</form>
</body>
</html>
```

Ejecute la página. Verá que es un formulario que pide una serie de datos. El aspecto es similar al que se ve en la figura 5.2.

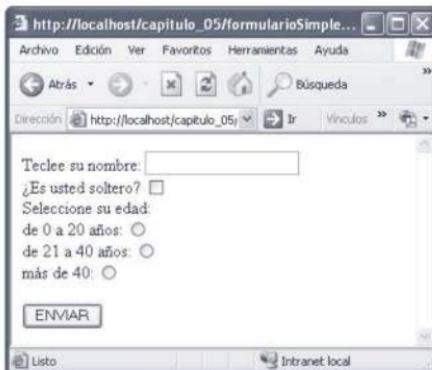


Figura 5.2

Quizás el aspecto no es muy elaborado. Podríamos haber usado una tabla para colocar cada cosa en su sitio, CSS para darle estilos al texto y los campos, etc. Da igual. Lo que nos interesa ahora es ver cómo funciona todo esto. Teclee su nombre donde se le pide, marque la casilla de "soltero" (no importa ahora si usted es soltero o no) y seleccione el botón de radio correspondiente a su edad. A continuación pulse en **ENVIAR** y vea lo que ocurre cuando se llama a **formularioSimple.php**, cuyo código es el siguiente:

```
<?php
    echo ("El nombre tecleado es: '$nombre'<br />");
    echo ("El estado soltero es: '$soltero'<br />");
    echo ("Su rango de edad es: '$edad'<br />");
    echo ("El campo oculto vale: '$datoOculto'<br
/>"); ;
?>
```

El resultado, según los datos que usted haya tecleado, se parecerá al que refleja la figura 5.3.

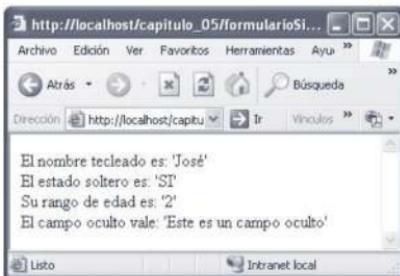


Figura 5.3

Quiero que observe que he puesto, en el código, unas comillas simples acotando cada dato, y así aparecen en el resultado. He hecho esto por una razón: vuelva a ejecutar formularioSimple.htm y, en esta ocasión, NO marque la casilla de verificación que corresponde a *soltero* ni ninguno de los botones de edad. Pulse en **ENVIAR** y vea el resultado, similar a la figura 5.4.

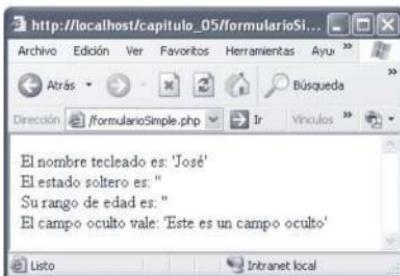


Figura 5.4

Cuando se envían campos de un formulario a un script PHP hay que tener en cuenta este tipo de cosas. Un campo de casilla de verificación o de botones de radio que no ha sido seleccionado envía un valor nulo. Si el campo ha sido seleccionado, envía como valor aquél que tenga en el atributo *value*.

En realidad no es una buena práctica enviar una variable desde un formulario a un script del servidor con un valor nulo, así que tendremos que hacer algo para que esto no suceda. Lo ideal es que el valor del campo de casilla sea SÍ, si el usuario la ha seleccionado, o NO, si la ha dejado sin marcar. Desafortunadamente, HTML permite establecer el valor de la casilla seleccionada, pero no el valor que tendrá si no lo está, así que tenemos que recurrir a un código JavaScript, en la página que contiene el formulario para solventar esto. Observe el script **establecerOculto.htm**:

```
<html>
    <head>
        <script language="javascript"
type="text/javascript">
            function comprobarFormulario() {
                if
(!document.getElementById("soltero").checked)  {
                    document.getElementById("ocultoSoltero").value="NO";
                } else {
                    document.getElementById("ocultoSoltero").value="SI";
                }
            }
        </script>
    </head>
    <body>
        <form name="f_prof" id="f_prof"
action="establecerOculto.php" method="post"
onSubmit="javascript:comprobarFormulario();">
            ¿Es usted soltero?
            <input type="checkbox" name="soltero"
id="soltero" value="SI">
            <br />

            <input type="hidden" name="ocultoSoltero"
id="ocultoSoltero" value="">
            <p>
                <input type="submit" value="ENVIAR" name="ok"
id="ok">
            </p>
        </form>
    </body>
</html>
```

Este listado cuenta, en el formulario, con un campo oculto al que, a priori, no se le asigna ningún valor. Fíjese en el manejador de evento onSubmit que acompaña al formulario. Usted ya sabe cómo actúa: se dispara cuando se pulsa el botón definido como de tipo submit y ejecuta la función indicada **antes** de mandar el formulario. En este caso, se trata de una función de JavaScript que comprueba si la casilla de verificación está marcada o no y, en base a esto, le asigna un valor u otro al campo oculto. Cuando el formulario es enviado, el script PHP que lo recibe tiene en cuenta el valor de este campo oculto, y no el que corresponde a la casilla de verificación, tal como se ve en el script **establecerOculto.php**:

```
<?php  
    echo ("El valor del campo oculto es:  
'$_ocultoSoltero'<br />");  
?>
```

Ejecute el establecerOculto.htm y pruebe el resultado marcando la casilla y sin marcarla, para comprobar el funcionamiento.

## 5.2 MÉTODOS DE ENVÍO

Como usted sabe, a la hora de enviar un formulario al servidor, podemos establecer el valor post o el valor get en el atributo method. En los ejemplos anteriores hemos usado el método post. Observe un detalle de la página donde se carga el script PHP cuando se pulsa el botón **ENVIAR**. Vea la figura 5.5.



Figura 5.5

Fíjese en que en la barra de dirección aparece la llamada al script, y nada más. Ahora ejecute el script **formularioGet.htm**, que es similar, pero envía el formulario mediante el método get. Observe la forma en que se carga el script PHP en la barra de direcciones. Mire la figura 5.6.



Figura 5.6

A pesar de la limitación impuesta por el papel se aprecia claramente la diferencia. Cuando un formulario se manda por el método get las variables que contiene aparecen en la barra de direcciones. En este ejemplo concreto lo que aparece en la barra de direcciones es lo siguiente:

`http://localhost/capitulo_05/establecerOculto.php?nombre=Jos%E9&&oltero=SI&edad=2&datoOculto=Este+es+un+campo+oculto&ok=ENVIAR`

Si se fija en su navegador, verá algunas secuencias de caracteres entre los nombres de las variables y sus valores que le desconcertarán un poco. Es la *codificación URL*. No se preocupe ahora por ello. Hablaremos de este tema más adelante. Lo único que vamos a comentar ahora es que lo que aparece son las variables que corresponden a los campos del formulario y los valores que contienen. Aunque aún no entremos en detalle acerca de la forma de interpretar la codificación URL, si quiero que se fije en la última parte de la línea, concretamente donde aparece **ok=ENVIAR**. Es una variable que corresponde al botón submit del formulario. Como ve, se envía como un campo más.

En realidad no es que vaya a usar muy a menudo esta variable en un script PHP, pero el formulario la envía de todos modos. Por supuesto, también se envía esta variable si el formulario es enviado mediante post, aunque, en ese caso, no la veamos en la url. Lo que quiero con este ejemplo es que repare en las diferencias de los dos métodos de envío. Yo, personalmente, casi siempre empleo el método post, ya que deja una barra de direcciones más limpia, y eso el usuario lo percibe. Por lo demás, las variables y sus valores se envían correctamente en ambos casos.

## 5.3 LAS VARIABLES DEL INTÉRPRETE

El intérprete de PHP maneja algunas variables y matrices que tiene reservadas, de forma que no pueden ser definidas por el usuario. Muchas de éstas comienzan por `$HTTP_` por lo que no debe acostumbrarse a usar este comienzo para sus nombres de variables. Como norma general, y dado que sabemos que PHP distingue las mayúsculas de las minúsculas en los nombres de las variables, cree siempre sus nombres usando las normas que le sugerí en el capítulo 3. Usted ya ha conocido una de las matrices propias de PHP en el capítulo 4, en concreto la matriz `$GLOBALS`. Ahora conocerá algunas más.

Lo primero que debe saber es que, cuando envía datos de un formulario a un script PHP, los valores no están sólo disponibles a través de las variables del mismo nombre, sino también en matrices específicas. Así, por ejemplo, si su formulario es enviado mediante post, los datos se almacenan en una matriz a la que se puede acceder como `$HTTP_POST_VARS`, o `$_POST`. Cuando el formulario es enviado mediante get, los pares nombre-valor quedan almacenados en una matriz accesible mediante `$HTTP_GET_VARS` o, simplemente, `$_GET`. Estas matrices son asociativas. El índice es el nombre de la correspondiente variable. Observe el código del listado **formularioPostVars.htm**, que aparece reproducido a continuación

```
<html>
  <head>
  </head>
  <body>
    <form name="f_prof" id="f_prof"
action="formularioPostVars.php" method="post">

      Teclee su nombre:
      <input type="text" name="nombre" id="nombre">
      <br />
      Teclee su ciudad:
      <input type="text" name="ciudad" id="ciudad">
      <br />
      Teclee su edad:
      <input type="text" name="edad" id="edad">

      <p>
        <input type="submit" value="ENVIAR" name="ok"
id="ok">
      </p>
    </form>
  </body>
</html>
```

Ahora observe el listado de **formularioPostVars.php**, que es el script PHP que obtiene las variables.

```
<?php
  echo ("El valor del campo nombre es: ".
$HTTP_POST_VARS["nombre"]."<br />");
  echo ("El valor del campo ciudad es: ".
$HTTP_POST_VARS["ciudad"]."<br />");
  echo ("El valor del campo edad es: ".
$HTTP_POST_VARS["edad"]."<br />");
?>
```

Ejecute la página que contiene el formulario, teclee unos datos cualesquiera y envíelo para comprobar cómo esos datos se reflejan en el script PHP.

A continuación, observe el script **formularioGetVars.htm**. Es similar a formularioPostVars.htm, pero el formulario se envía mediante get, en lugar de post.

```
<html>
  <head>
  </head>
```

```
<body>
    <form name="f_prof" id="f_prof"
action="formularioGetVars.php" method="get">

        Teclee su nombre:
        <input type="text" name="nombre" id="nombre">
        <br />
        Teclee su ciudad:
        <input type="text" name="ciudad" id="ciudad">
        <br />
        Teclee su edad:
        <input type="text" name="edad" id="edad">
        <p>
            <input type="submit" value="ENVIAR" name="ok"
id="ok">
        </p>
    </form>
</body>
</html>
```

El script **formularioGetVars.php** obtiene los correspondientes datos:

```
<?php
    echo ("El valor del campo nombre es: ".
$HTTP_GET_VARS["nombre"]."<br />"); 
    echo ("El valor del campo ciudad es: ".
$HTTP_GET_VARS["ciudad"]."<br />"); 
    echo ("El valor del campo edad es: ".
$HTTP_GET_VARS["edad"]."<br />"); 
?>
```

Concretando. Cuando un formulario envía un campo a un script PHP, éste encuentra el valor simplemente con referenciar su nombre. Es decir, si el formulario tiene un campo llamado ciudad, al cargarse el correspondiente script existe una variable cuyo nombre es \$ciudad<sup>(1)</sup> y cuyo valor es lo que haya tecleado el usuario. Éste es el sistema que hemos usado en los ejemplos del capítulo 3. Ahora sabemos que también existe una matriz, cuyo nombre depende del método por el que se ha enviado el formulario, y en la que existe un elemento con el valor que haya introducido o elegido el usuario. Dicho elemento se identifica por un índice asociativo cuyo nombre es el nombre del campo.

(1) En realidad, para que un campo de un formulario sea reconocido en el script directamente por el nombre de la variable, debe tener activada la directriz register\_globals, en el fichero de configuración de PHP. Todo lo que necesita saber acerca de este fichero (tanto de ésta como de otras directivas) se encuentra en el Apéndice A: "La configuración del intérprete de PHP".

¿Qué notación debemos usar en nuestros scripts? ¿Llamaremos a las variables simplemente por sus nombres o mediante la correspondiente matriz asociativa? En esto mi postura es clara. El sentido común nos dice que si empleamos las matrices que acabamos de conocer para referirnos a aquellas variables que proceden de campos de un formulario, las distinguiremos enseguida de otras variables que hayan sido creadas en el propio script. Si usted escribe un código sencillo de 20 ó 30 líneas, quizás esto no le parezca muy significativo, pero cuando su script maneja bases de datos, ficheros, etc., y tiene 400 ó 500 líneas, todo lo que sea claridad en la codificación es vital. El intérprete de PHP maneja algunas variables y matrices que tiene reservadas, de forma que no pueden ser definidas por el usuario.

## 5.4 ENVIANDO ARCHIVOS

Ya sabemos cómo enviar datos que el usuario escribe en un formulario y cómo recibirlos en un script en el servidor (lo que luego se pueda hacer con esos datos es tema para otros capítulos). Pero hay un campo especial en algunos formularios. Se trata de los campos de tipo **file** que, como usted ya sabe, permiten el envío de archivos. ¿Cómo se reciben y procesan esos campos? PHP cuenta con dos matrices específicas llamadas **\$HTTP\_POST\_FILES** y **\$FILES** (esta última fue implementada en la versión 4.3). Ambas matrices cumplen la misma función, y el uso de una u otra es indiferente en esta versión de PHP.

Sin embargo, es posible que en futuras versiones **\$HTTP\_POST\_FILES** deje de ser reconocida por el intérprete, por lo que le recomiendo el uso de **\$FILES**. Llegado este punto quiero comentarle una cosa: *PHP no reconoce ni puede procesar ficheros que hayan sido enviados mediante get. Cuando use un formulario desde el cual el usuario pueda enviar ficheros al servidor use SIEMPRE el método post.* Las matrices que gestionan los ficheros son asociativas de dos índices. El primero (el de las filas, para entenderlos) contiene el nombre que se ha dado en el formulario a cada campo del fichero (puede haber más de uno en un mismo formulario). Dentro de cada fila encontramos otro índice con las propiedades del fichero, tales como nombre, tamaño, tipo, etc. Las propiedades que definen un archivo enviado al servidor aparecen recopiladas en la tabla reproducida en la figura 5.7.

PROPIEDAD	CONTENIDO
error	Indica si se ha producido un error en el envío del fichero. Si todo ha ido correctamente, esta propiedad almacena el valor 0. Si se ha producido un error, el valor es 1.
name	El nombre del archivo, tal como lo tiene almacenado el usuario en su ordenador.
tmp_name	Es un nombre temporal que usa PHP para la gestión provisional del archivo, hasta que lo almacene en el disco del servidor, lo envíe por correo electrónico a donde corresponda, o cualquier otra acción que le hayamos programado al script.
type	Almacena el tipo de fichero que se ha enviado. Puede ser una imagen, audio, video, texto plano, etc.
size	El peso del archivo en bytes.

*Figura 5.7*

En la carpeta `ficherosParaSubir`, dentro, a su vez, de la que corresponde a este capítulo, tiene varios archivos de muestra que vamos a emplear para comprobar y aprender todo lo necesario acerca de cómo gestionar con PHP la subida de ficheros a un servidor mediante un formulario. Esta carpeta es donde, en nuestros ejemplos, tendría el usuario los archivos que quiere mandar. Además, encontrará usted la carpeta `ficherosSubidos`, que es donde se irán alojando los ficheros que, supuestamente, mandaría el usuario.

Para empezar, vea el script `enviarFichero.htm`:

```

<html>
  <head>
  </head>
  <body>
    <form action="enviarFichero.php" method="post"
name="f_prof" id="f_prof" enctype="multipart/form-data">
      Fichero:
      <input type="file" name="fichero" id="fichero">
      <input type="submit" value="ENVIAR" name="ok">
    </form>
  </body>
</html>

```

Observe que en el formulario hay un campo de tipo fichero al que hemos llamado, precisamente, `fichero`. En la etiqueta que abre el formulario hemos establecido, como valor del atributo `action`, el nombre del script al que queremos

que el usuario envíe un archivo. Además, hemos establecido el atributo **enctype="multipart/form-data"**, imprescindible, como sabe, en formularios que tienen campos de tipo archivo. Ejecute este listado. Cuando vea la página cargada, pulse el botón **Examinar...** y seleccione un archivo de la carpeta **ficherosParaSubir**. En esta primera prueba seleccione el archivo de imagen llamado **imagen.jpg**. A continuación, cuando el nombre del archivo (con su ruta completa) le aparezca en el campo correspondiente, pulse el botón **ENVIAR**. Con esto enviaremos el archivo seleccionado al script cuyo código aparece en **enviarFichero.php**:

```
<?php
    foreach ($_FILES["fichero"] as $clave => $valor){
        echo ("Propiedad: $clave ---- Valor:
$valor<br />");
    }
?>
```

Este script emplea un bucle **foreach** para recorrer cada uno de los elementos de una matriz que, a su vez, es el elemento “fichero” de la matriz **\$\_FILES**. El nombre de este elemento viene dado por el nombre del campo de archivo en el formulario. Todos los elementos de esta matriz (las propiedades que PHP reconoce del archivo enviado) se listan, obteniendo un resultado similar al de la figura 5.8.

En este ejemplo, las propiedades y sus valores serán los que aquí aparecen excepto, seguramente, el nombre temporal del archivo, que es elegido por PHP de forma aleatoria.

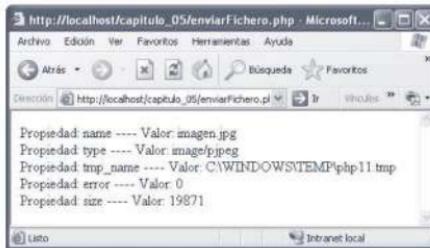


Figura 5.8

En primer lugar, vemos la propiedad **name** que, como se aprecia, corresponde al nombre con el que está grabado el archivo en el disco del cliente. El

que el usuario envíe un archivo. Además, hemos establecido el atributo **enctype="multipart/form-data"**, imprescindible, como sabe, en formularios que tienen campos de tipo archivo. Ejecute este listado. Cuando vea la página cargada, pulse el botón **Examinar...** y seleccione un archivo de la carpeta **ficherosParaSubir**. En esta primera prueba seleccione el archivo de imagen llamado **imagen.jpg**. A continuación, cuando el nombre del archivo (con su ruta completa) le aparezca en el campo correspondiente, pulse el botón **ENVIAR**. Con esto enviaremos el archivo seleccionado al script cuyo código aparece en **enviarFichero.php**:

```
<?php
    foreach ($_FILES["fichero"] as $clave => $valor){
        echo ("Propiedad: $clave ---- Valor:
$valor<br />");
    }
?>
```

Este script emplea un bucle **foreach** para recorrer cada uno de los elementos de una matriz que, a su vez, es el elemento “fichero” de la matriz **\$\_FILES**. El nombre de este elemento viene dado por el nombre del campo de archivo en el formulario. Todos los elementos de esta matriz (las propiedades que PHP reconoce del archivo enviado) se listan, obteniendo un resultado similar al de la figura 5.8.

En este ejemplo, las propiedades y sus valores serán los que aquí aparecen excepto, seguramente, el nombre temporal del archivo, que es elegido por PHP de forma aleatoria.

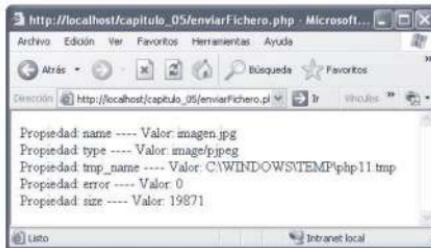


Figura 5.8

En primer lugar, vemos la propiedad **name** que, como se aprecia, corresponde al nombre con el que está grabado el archivo en el disco del cliente. El

nombre, sin más, sin ruta ni nada adicional. Es lógico que la ruta no se transmita, puesto que el archivo no se va a grabar en la misma ruta en el servidor que aquella que el usuario tenga en su equipo cliente.

La propiedad **type** contiene el valor **image/jpeg**. Este valor nos indica que el archivo es una imagen de tipo jpg. Cuando se envía una imagen, el valor de type siempre empieza por **image/** y añade, detrás del slash, una referencia acerca del formato de la imagen.

La propiedad **tmp-name** puede parecer intrascendente, sobre todo si uno piensa que ha sido asignada por el propio intérprete de PHP. Sin embargo es, quizás, una de las propiedades más interesantes, como iremos viendo en su momento.

La propiedad **error** contiene el valor 0, lo que nos indica que la transferencia se ha hecho sin problemas.

Por último, la propiedad **size** almacena el peso del archivo en bytes. Teniendo en cuenta que un Kb son 1024 bytes, en nuestro ejemplo este archivo aparece con un peso de casi 19.5 Kb. Si necesita calcular el peso de un archivo en Mb, recuerde que 1Mb equivale a 1024 Kb.

Los tipos más comunes de archivos que se suelen enviar a través de un formulario establecen unos valores en la propiedad type que aparecen listados en la tabla de la figura 5.9. Hay más tipos de archivos, pero éstos son los más comunes. Además, usted ya sabe cómo visualizar las propiedades de un archivo concreto con el que pueda necesitar trabajar en el futuro.

TYPE	TIPO DE ARCHIVO
<b>image/jpeg</b>	Imagen en formato jpg.
<b>image/gif</b>	Imagen en formato gif.
<b>image/bmp</b>	Imagen en formato bmp.
<b>image/x-png</b>	Imagen en formato png.
<b>audio/mpeg</b>	Audio en formato mpeg (normalmente mp3).
<b>video/mpeg</b>	Vídeos en formato mpeg.
<b>application/x-rar-compressed</b>	Cuando el tipo de un archivo empieza por application/ requiere ser abierto con algún programa específico. En este ejemplo, se trata de un archivo en formato rar, que puede ser abierto con determinadas utilidades de compresión-descompresión.

TYPE (cont.)	TIPO DE ARCHIVO (cont.)
<b>text/plain</b>	Texto plano.
<b>text/richtext</b>	Texto enriquecido.
<b>application/pdf</b>	Archivo para ser abierto con un visor o editor capaz de reconocer el formato pdf.

Figura 5.9

#### 5.4.1 Limitando el tamaño del archivo

Éste es un aspecto muy importante. Cuando incluimos en un formulario la posibilidad de que el usuario envíe archivos al servidor debemos, de algún modo, limitar el tamaño de dichos archivos. Suponga que usted quiere almacenar las imágenes que le mandan para un uso posterior. Y ahora suponga que le mandan una foto del ayuntamiento de su ciudad, a tamaño natural y en formato bmp. Ya tiene su servidor colapsado hasta el fin de los tiempos. Bromas aparte, es importante establecer un límite. Para ello incluimos en el campo de archivo del formulario un campo oculto cuyo nombre sea “**MAX\_FILE\_SIZE**” y cuyo valor sea el peso en bytes que usted quiere establecer como límite. Vea un ejemplo de uso en **ficheroMaximo.htm**:

```
<html>
  <head>
  </head>
  <body>
    <form action="enviarFichero.php" method="post"
name="f_prof" id="f_prof" enctype="multipart/form-data">
      Fichero:
      <input type="hidden" name="MAX_FILE_SIZE"
value="10240">
      <input type="file" name="fichero" id="fichero">
      <input type="submit" value="ENVIAR" name="ok"
id="ok">
    </form>
  </body>
</html>
```

Observe la linea resaltada. Cuando se usa esta técnica para limitar el peso de un archivo, el campo oculto debe ir siempre *antes* del campo de archivo. Lo ideal es ponerlo *inmediatamente antes*, ya que así nos permite, al revisar el código, reconocer ese campo oculto como asociado al campo de archivo. Si el usuario

intenta mandar un archivo que supere el tamaño máximo permitido, dicho archivo no será aceptado por PHP.

Esto significa que las propiedades type y tmp\_name no tendrán contenido, la propiedad size tendrá el valor 0 y la propiedad error tendrá el valor 2. El hecho de que la propiedad tmp\_name no tenga valor asignado imposibilita, en la práctica, el manejo del archivo por parte del script. En este mismo capítulo entenderá usted por qué. De todos modos, este sistema no es seguro. Cualquier persona con un mínimo de práctica puede anular este control de tamaño mediante lo que se conoce como HTML Injection. Es una técnica de hacking muy simple para la manipulación de formularios web. No vamos a entrar aquí a detallar cómo funciona, ya que éste no es un tratado sobre ese tema. Lo único sobre lo que quiero llamar su atención es que, si realmente desea limitar el peso de los archivos que su script deba poder procesar, es mejor recurrir a sistemas más eficaces. Por ejemplo, puede usar la propiedad size, de modo que el script no procese el archivo si el valor de esta propiedad supera un límite pre-establecido. Recuerde esto: siempre que sea posible elija, para llevar a cabo cualquier tarea, la ejecución en el lado del servidor. Esto aleja el funcionamiento de su código de los usuarios. Ellos sólo necesitan obtener resultados rápidos y seguros, no saber cómo los obtienen.

#### 5.4.2 Envío de múltiples archivos

Usted puede crear un formulario que permita al usuario enviar más de un archivo y tratarlos todos como elementos de una matriz. Por ejemplo, suponga que su página le da al cliente la opción de enviar hasta cuatro fotografías al servidor. Veamos cómo hacerlo en el código **multiplesArchivos.htm**, que hemos listado a continuación:

```
<html>
  <body>
    <form action="multiplesArchivos.php" method="post"
enctype="multipart/form-data">
      <input name="archivos[]" type="file">
      <input name="archivos[]" type="file"><br />
      <input name="archivos[]" type="file">
      <input name="archivos[]" type="file"><br />
      <input type="submit" value="Enviar">
    </form>
  </body>
</html>
```

Quiero que se fije en el atributo *name* de los campos de archivo. Lo he resaltado en el código para llamar su atención. *Todos los campos tienen el mismo valor en el atributo name*. Esto no parece tener ningún sentido. Desde que

aprendimos HTML, sabemos que cada campo de un formulario debe tener un nombre único para que se pueda trabajar con él de forma inequívoca. Esto también es válido cuando trabajamos con páginas dinámicas, ya que cada campo se envía como un par nombre-valor al servidor. Sin embargo, éste es un caso especial. Repare en que los nombres de los campos llevan corchetes, indicando que pertenecen a una matriz. Esto hará que los campos sean enviados al servidor como tal matriz. No se les ha atribuido un índice, de modo que el primer campo tomará el índice 0, el segundo el índice 1, y así sucesivamente. Al cargar este listado en el navegador verá el formulario que aparece en la figura 5.10.

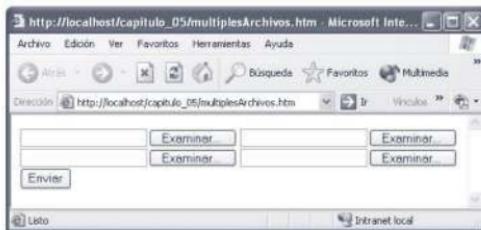


Figura 5.10

Seleccione cuatro archivos de la carpeta `ficherosParaSubir` que está incluida en la correspondiente a este capítulo y pulse el botón `Enviar`. En ese momento los cuatro archivos son enviados, como una matriz, al script que aparece en el atributo `action` del formulario, cuyo nombre es `multiplesArchivos.php`, y cuyo listado aparece a continuación:

```
<?php
// Se define el salto de línea.
define ("salto","<br />\n");
/* Se contabiliza el número total de elementos de la
matriz "archivos", cuyo nombre se corresponde con los campos
enviados desde el formulario. Dado que, dicho formulario
tiene cuatro campos de archivo pertenecientes a la misma
matriz, ese es el número de elementos que esta contiene. Eso
es independiente de que en los campos se haya introducido un
valor o no. Los campos existen, aunque no tengan contenido.*/
$totalDeArchivos=count($_HTTP_POST_FILES["archivos
"]["tmp_name"]);
echo ("El total de campos de archivo es de:
".$totalDeArchivos.$salto);
echo ("Sin embargo, puede que no todos contengan
un archivo.".salto);
```

```
        echo (salto.salto);
    /* Se recorren los cuatro campos, comprobando su
propiedad name. Si no existe quiere decir que el campo no
tiene contenido, es decir, que no se ha subido ningún archivo
en ese campo.*/
        echo ("A continuación se muestra la propiedad
\"name\" (nombre)".salto);
        echo ("de cada uno de los archivos
subidos.".salto);

    for ($contador=0; $contador<$totalDeArchivos;
$contador++) {
        echo ("<b>ARCHIVO $contador:</b> ");
        if
($HTTP_POST_FILES["archivos"]["name"][$contador]== ""){
            echo ("NO EXISTE".salto);
        } else {
            echo
("\"".$HTTP_POST_FILES["archivos"]["name"][$contador]."\").sa
lto);
        }
    }
?>
```

Vamos a ver cómo funciona este código. En primer lugar, se determina el número de elementos que componen la matriz de ficheros enviados al servidor. Nosotros sabemos que son cuatro elementos, ya que en el formulario hay cuatro campos de archivo con el nombre de esta matriz. Cuando se envían múltiples archivos al servidor usando el sistema de matriz, ésta se crea, siempre, con tantos elementos como campos contiene el formulario, con independencia de que el usuario haya seleccionado un archivo para todos los campos, para algunos, o para ninguno. Fíjese en la línea que cuenta los elementos de la matriz, reproducida a continuación:

```
$totalDeArchivos=count($HTTP_POST_FILES["archivos"]["tm
p_name"]);
```

En este caso, la matriz \$HTTP\_POST\_FILES tiene tres índices. El primero es el que corresponde al nombre de la matriz de campos: en este caso, “archivos”. Recuerde que así es como hemos llamado a los campos en el formulario. El segundo índice corresponde a las propiedades de cada uno de los elementos de la matriz “archivos”. El tercer índice es el número de elemento de la matriz “archivos” al que queremos referirnos. Para efectuar la cuenta de elementos de la matriz “archivos” hemos usado la propiedad tmp\_name, pero podríamos haber usado cualquier otra. Eso no va a cambiar el resultado.

A continuación, fíjese en que entramos en un bucle que itera tantas veces como elementos tenga la matriz “archivos”. En este caso son cuatro iteraciones. Dentro del bucle vamos a determinar el valor de la propiedad name de cada uno de los elementos, es decir, de cada uno de los archivos enviados al servidor. Si en alguno de los campos el usuario no ha seleccionado un archivo, esta propiedad no tendrá contenido. Para determinar esto empleamos un condicional, así:

```
if  
($HTTP_POST_FILES["archivos"]["name"][$contador]== "") {
```

Ahora cargue en el navegador, si no lo ha hecho antes, la página multiplesArchivos.htm, y seleccione un archivo para el primer campo, y otro para el tercero. Envíe el formulario, que irá contra el script que acabamos de ver. Éste le mostrará los nombres de los archivos que se han subido. En los correspondientes a los campos segundo y tercero, donde no se seleccionó ningún fichero, no muestra nada.

## 5.5 PROCESANDO LOS ARCHIVOS ENVIADOS

Uno de los usos más habituales de los archivos que son enviados al servidor es su almacenaje en el mismo para ponerlos a disposición de otros usuarios. Consideremos, por ejemplo, algo tan en boga como los portales de contactos, donde cada persona se anuncia para conocer a otras con las que pueda congeniar o hacer amistad. Lo normal es que el anunciante tenga opción a enviar una o más fotografías que quedarán almacenadas en el servidor para que otros usuarios las vean. Vamos a aprender a grabar, en el disco duro del servidor, los archivos que nos envíe el usuario. Para ello, contamos con la función *move\_uploaded\_file()*.

La sintaxis normalizada implica que recibe dos argumentos, separados por una coma. El primero corresponde a la propiedad tmp-name del archivo subido. El segundo corresponde al nombre con el que se grabará dicho archivo en el servidor, incluyendo la ruta. Vamos a empezar considerando el listado **grabarFichero.htm**.

```
<html>  
  <head>  
  </head>  
  <body>  
    <form action="grabarFichero.php" method="post"  
      name="f_prof" id="f_prof" enctype="multipart/form-data">  
      Fichero:  
      <input type="file" name="fichero" id="fichero">  
      <input type="submit" value="ENVIAR" name="ok"  
      id="ok">
```

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

Como ve, no tiene nada de particular. Selecciona un fichero y lo envía al script que hay en **fichero\_080.php**, cuyo código es el siguiente:

```
<?php
    $archivoRecibido=$_FILES["fichero"]["tmp_name"];
    $destino="ficherosSubidos/fotoDelUsuario.jpg";
    move_uploaded_file ($archivoRecibido, $destino);
    echo ("Fichero grabado");
?>
```

En primer lugar compruebe que en la carpeta **ficherosSubidos** no tiene nada. A continuación ejecute **grabarFichero.htm**. Cargue el fichero **imagen.jpg** en el campo de archivo. Este fichero está en la carpeta **ficherosParaSubir**. Para este ejercicio consideramos esta carpeta como perteneciente al cliente y la carpeta **ficherosSubidos** como perteneciente al servidor. Pulse el botón **ENMAR** y pasará el archivo seleccionado al script. Dado que no está trabajando a través de Internet, sino en su propio ordenador, la respuesta será prácticamente inmediata. El script le informará, mediante un mensaje en la página, de que el archivo ha sido grabado en el servidor. Cierre el navegador y abra la carpeta **ficherosSubidos**. Allí encontrará un archivo de imagen con el nombre **fotoDelUsuario.jpg**, que es copia de **imagen.jpg**. Salga de esa carpeta y vuelva a entrar en **ficherosParaSubir**. Verá que el archivo original no ha desaparecido ni sufrido alteración alguna. Es lógico. El usuario no va a subirnos un fichero al servidor a costa de perderlo en su equipo. Veamos qué es lo que hemos hecho. En primer lugar, hemos usado la propiedad **tmp\_name** del archivo subido para referirnos a él, así:

```
$archivoRecibido=$_FILES["fichero"]["tmp_name"];
```

A continuación, hemos creado una variable con el nombre con el que grabaremos el archivo en el servidor, incluyendo la ruta relativa, tal como se muestra a continuación:

```
$destino="ficherosSubidos/fotoDelUsuario.jpg";
```

Por último, hemos grabado el archivo mediante el uso de la función que estamos estudiando, así:

```
move_uploaded_file ($archivoRecibido, $destino);
```

Por supuesto, podríamos haberlo hecho todo en una sola línea de código, ahorrándonos, además, el uso de dos variables, tal como se ve a continuación:

```
move_uploaded_file ($_FILES["fichero"]["tmp_name"],  
"ficherosSubidos/fotoDelUsuario.jpg");
```

Sin embargo, tal como lo hemos hecho queda mucho más legible y claro.

En principio, este código funciona. Sin embargo, adolece de ciertas deficiencias que tenemos que subsanar. Por ejemplo, el usuario puede equivocarse y mandarnos un archivo de audio en lugar de una imagen. O quizás mande una imagen, pero ésta se encuentre en formato bmp, que no queremos usar en nuestra página. Para evitar esto vamos a usar la propiedad type del archivo subido. Lo que haremos será poner un condicional que ofrezca una "salida honrosa" si el archivo no es del tipo adecuado. Puede ser algo como lo siguiente:

```
if ($_FILES["fichero"]["type"] != "image/jpeg") {  
    die ("El fichero no tiene el formato adecuado.");  
}
```

Lo que hacemos es comprobar si el fichero está en formato jpg. Esto no tiene nada que ver con que el nombre del archivo tenga la extensión adecuada. Usted puede grabar un fichero cualquiera en su disco duro y ponerle al nombre la extensión jpg, pero no por eso será una imagen en ese formato. La propiedad type no comprueba la extensión del nombre del archivo, sino el tipo real de formato en el que está grabado. Observe que, si el formato no es el adecuado, se ejecuta la función *die()*. Esta función, de nombre tan dramático ("die", en inglés, significa, literalmente, "muere"), termina la ejecución del script mostrando el texto que recibe como argumento. Así pues, si el formato no es el adecuado, la ejecución del script termina en ese punto. Ya no se comprueba nada más, ni se graba nada, ni se hace nada. Sólo se muestra el mensaje y se acabó.

Esto nos proporciona, además, una posibilidad de limitar el peso de los archivos que el usuario puede enviar al servidor. En el apartado anterior hemos visto una forma de hacer esto, pero hemos comentado que no es segura. Ahora podemos hacerlo de un modo más eficiente, así:

```
if ($_FILES["fichero"]["size"] > 200000) {  
    die ("El fichero es demasiado grande."); }
```

Como ve, esta vez hacemos uso de la propiedad size del archivo transmitido. Si el valor de esta propiedad supera el tope que hemos establecido (en este caso 200000 bytes) la ejecución se interrumpe. Y, ya puestos, podemos hacer otra comprobación, relativa a que se produzca algún error general durante la transferencia del archivo, así:

```
if ($_FILES["fichero"]["error"] != 0) {
    die ("Se ha producido un error.");
}
```

Además, aunque todo haya ido bien hasta este punto, puede ser que, por alguna dificultad técnica, el archivo no pueda grabarse en el servidor, a pesar de todo. La función move\_uploaded\_file () devuelve un valor lógico true si ha podido grabar el archivo adecuadamente en el servidor, y un valor false si no se ha podido grabar. Sabiendo esto podemos usarla en la siguiente manera:

```
if (move_uploaded_file ($_FILES["fichero"]["tmp_name"],
    "ficherosSubidos/fotoDelUsuario.jpg")) {
    echo ("Fichero grabado");
} else {
    echo ("El fichero no se ha podido grabar.");
}
```

Como ve, si se ha podido grabar el archivo en el servidor, se muestra el mensaje que informa de esto. En caso contrario, se informa de que la grabación ha fallado. Y ya tenemos elementos para montar un script bastante más eficiente que el anterior. Lo hemos llamado **comprobarFichero.php**:

```
<?php
// Se comprueba si el archivo tiene el formato
adecuado.
if ($_FILES["fichero"]["type"]!="image/jpeg") {
    die ("El fichero no tiene el formato
adecuado.");
}
// Se comprueba que tenga un peso adecuado.
if ($_FILES["fichero"]["size"] > 200000) {
    die ("El fichero es demasiado grande.");
}
// Se comprueba que no se hayan producido errores.
if ($_FILES["fichero"]["error"] != 0) {
    die ("Se ha producido un error.");
}
// Por último, se intenta grabar y se comprueba si se
graba bien.
$archivoRecibido=$_FILES["fichero"]["tmp_name"];
$destino="ficherosSubidos/fotoDelUsuario.jpg";
if (move_uploaded_file ($archivoRecibido,
$destino)) {
    echo ("Fichero grabado");
} else {
```

```
echo ("El fichero no se ha podido  
grabar.");  
}  
?>
```

Para acceder a este script tenemos **comprobarFichero.htm**, que es similar a los que ya conocemos. Ejecútelo con archivos de distintos tipos y pesos para comprobar cómo funciona.

Sin embargo, aún esto es mejorable. Ya sabemos que podemos incluir etiquetas de HTML en nuestros scripts PHP. Lo hemos hecho varias veces a lo largo de lo que llevamos de libro, insertando dichas etiquetas como argumentos de la función echo (). Lo cierto es que también podemos incluir de este modo código JavaScript. Así pues, podemos hacer que el script regrese a la página del formulario si hay algún error, de modo muy sencillo. Observe el script **comprobarYVolver.php**:

```
<?php  
    $error=false;  
    // Se comprueba si el archivo tiene el formato  
    adecuado.  
    if ($_FILES["fichero"]["type"]!="image/jpeg") {  
        $error=true;  
    }  
  
    // Se comprueba que tenga un peso adecuado.  
    if ($_FILES["fichero"]["size"] > 200000) {  
        $error=true;  
    }  
  
    // Se comprueba que no se hayan producido errores.  
    if ($_FILES["fichero"]["error"] != 0) {  
        $error=true;  
    }  
  
    // Si se ha producido algún error se vuelve al  
    formulario inicial.  
    if ($error){  
        echo ("<script language='javascript'  
type='text/javascript'>");  
        echo  
("location.href='comprobarYVolver.htm';");  
        echo ("</script>");  
    }  
    // Por último, se intenta grabar y se comprueba si se  
    graba bien.
```

```
$archivoRecibido=$_FILES["fichero"]["tmp_name"];
$destino="ficherosSubidos/fotoDelUsuario.jpg";
if (move_uploaded_file ($archivoRecibido,
$destino)) {
    echo ("Fichero grabado");
} else {
    echo ("El fichero no se ha podido
grabar.");
}
?>
```

Observe lo que hacemos. Comprobamos si las propiedades type, size y error del fichero enviado están dentro de los valores establecidos como válidos, si no es así, ponemos el valor true en una variable lógica que, inicialmente, se declara como false. A continuación, comprobamos si se ha producido algún error (si la variable lógica empleada tiene el valor true). Si es así, se ejecuta un código JavaScript (ver en la parte resaltada), que devuelve al usuario a la página donde se pide el fichero. Para comprobar el funcionamiento de este script cargue en su navegador el listado **comprobarYVolver.htm**.

Y sigamos pensando en mejorar nuestro script. Ya tenemos un código que hace todo lo que podemos necesitar. Pero ahora póngase en el papel del usuario. Si se le pide un fichero y, por error, selecciona uno inadecuado, el script hace que se le vuelva a pedir, sin darle ninguna indicación de lo que ha ocurrido. Es fácil que el usuario piense que nuestro sitio funciona mal y lo abandone sin más. Mala perspectiva. Es necesario hacer algo para informar al usuario de la causa por la que no se acepta su fichero antes de volver a pedírselo. Para ver cómo podemos resolver esto observe el script **volverPorFuncion.php**:

```
<html>
<head>
    <script language='javascript'
type='text/javascript'>
        function volver() {
            location.href='volverPorFuncion.htm';
        }
    </script>
</head>
<body>

<?php
$error=false;

// Se comprueba si el archivo tiene el formato
adecuado.
```

```
if ($_FILES["fichero"]["type"] != "image/jpeg") {
    echo ("El archivo no está en el formato
adecuado.<br />");
    $error=true;
}
// Se comprueba que tenga un peso adecuado.
if ($_FILES["fichero"]["size"] > 200000) {
    echo ("El archivo es demasiado grande.<br />");
    $error=true;
}
// Se comprueba que no se hayan producido errores.
if ($_FILES["fichero"]["error"] != 0) {
    echo ("Hay un error en el archivo.<br />");
    $error=true;
}
// Si se ha producido algún error se vuelve al
formulario inicial.
if ($error){
    echo ("<input type='button' name='retorno'
id='retorno' value='VOLVER'
onClick='javascript:volver();';");
} else {
    // Por último, se intenta grabar y se comprueba si se
graba bien.
    $archivoRecibido = $_FILES["fichero"]["tmp_name"];
    $destino="ficherosSubidos/fotoDelUsuario.jpg";
    if (move_uploaded_file($archivoRecibido, $destino))
{
        echo ("Fichero grabado");
    } else {
        echo ("El fichero no se ha podido grabar.");
    }
}
?>
</body>
</html>
```

Veamos cómo opera este script. En primer lugar, vea que tiene una función de JavaScript destinada a devolverle a la página que contiene el formulario donde el usuario decide qué fichero enviar. Esta función, de momento, no se ejecuta. Veamos el código PHP, que es lo que nos interesa. La parte donde se comprueba si hay algún error es igual que en el caso anterior, solo que mostrando, en cada circunstancia, el tipo de error.

A continuación, se comprueba si la variable lógica que usamos como detector de haberse producido un error contiene el valor true. Si es así, se le

muestra al usuario un botón que activa la función JavaScript para volver a la página anterior. Si la variable de error contiene un valor false, no se muestra el botón. En su lugar se almacena el archivo en la carpeta destinada a este fin en el servidor. Como ve, con este código el usuario ya tiene una idea muy clara de lo que ocurre cuando envía un archivo y éste no es correcto.

Un error durante el envío de un archivo puede producirse por múltiples causas. Seguro que usted ha visitado páginas de Internet cuyo código es correcto y, sin embargo, se ha encontrado con alguna dificultad al enviar ficheros. Esto puede deberse a problemas en las líneas telefónicas u otros factores fortuitos. No hay gran cosa que un webmaster pueda hacer para evitar estos errores, pero si puede detectarlos a tiempo, tal como se describe en el siguiente apartado.

## 5.6 ERRORES IMPREVISTOS

Cuando se escribe para Internet, hay que pensar que nuestro sitio lo visitarán usuarios muy diversos, con distintos equipos cliente, con distintos navegadores y con muy diversos comportamientos. Es conveniente tratar de prever el máximo número posible de situaciones en las que pueda producirse un error durante la ejecución de nuestros scripts. Otras veces, esos errores estarán, incluso, causados por nosotros mismos, ya sea por un descuido durante la programación, un archivo que no está donde debe, etc. Se supone que los webmasters somos personas cuidadosas, detallistas, exigentes con nuestro propio trabajo y que probamos cada script hasta la saciedad. Sin embargo, también somos humanos y cometemos equivocaciones. De hecho, el que no se equivoca nunca es que tampoco está haciendo nada que valga la pena. En una ocasión tuve un error garrafal en una de mis clases. No voy a contarle cuál fue, porque todavía me avergüenzo de ello. El caso es que uno de mis alumnos me dijo: "Vaya. Si tú también te equivocas.", a lo que le respondí: "Por supuesto. Me equivoco constantemente. Así es como aprendo". Fue la salida más honrosa que se me ocurrió en ese momento. Anécdotas aparte, el caso es que, en ocasiones, se producen errores que no tenemos previstos. Por ejemplo, suponga que, al querer grabar un fichero con la función `move_uploaded_file()`, escribimos, como ruta para almacenarlo en el servidor, el nombre de una carpeta que no existe. Imagine que, en los códigos anteriores, a la hora de grabar el archivo tenemos algo como lo siguiente:

```
$archivoRecibido=$_FILES["fichero"]["tmp_name"];
$destino="ficherosEnviados/fotoDelUsuario.jpg";
if (move_uploaded_file ($archivoRecibido, $destino)) {
    echo ("Fichero grabado");
} else {echo ("El fichero no se ha podido grabar.");}
```

Fíjese en la línea resaltada. Como ve, se va a intentar grabar el archivo en una ruta inexistente. Uno puede pensar que como no se podrá grabar, el script mostrará el mensaje que hemos previsto para este caso, aquél que dice:

**El fichero no se ha podido grabar.**

Pero no es así. PHP *intenta realmente* ejecutar la grabación del archivo, y al no encontrar la carpeta adecuada nos da un error como el siguiente:

```
Warning: move_uploaded_file(ficherosEnviados/fotoDelUsuario.jpg)
[function.move-uploaded-file]: failed to open stream: No such file or directory in
C:\Documents and Settings\Jose\Mis documentos\Mis webs
dinamicas\capítulo_05\operadorAntiError.php on line 37
Warning: move_uploaded_file() [function.move-uploaded-file]: Unable to move
'C:\WINDOWS\TEMP\php14.tmp' to 'ficherosEnviados/fotoDelUsuario.jpg' in
C:\Documents and Settings\Jose\Mis documentos\Mis webs
dinamicas\capítulo_05\operadorAntiError.php on line 37
El fichero no se ha podido grabar.
```

Como ve, es una salida bastante desagradable a la vista del usuario. Además, en determinado tipo de errores puede llegar a detenerse la ejecución del script.

Cuando una instrucción de PHP pueda (o pensemos que puede) dar un error, lo que hacemos es anteponerle el signo *arroba (@)*, de forma que no se muestren los mensajes de error de PHP y se pueda continuar procesando el script. En este caso, pondremos este *operador de control de errores* justo antes de move\_uploaded\_file (), así:

```
if (@move_uploaded_file ($archivoRecibido, $destino)) {
```

Observe la parte resaltada. Al anteponer este operador, lo que le estamos diciendo al intérprete es que si esta función da un error, no lo muestre y continúe la ejecución del script.

En el script **operadorAntiError.php** tiene usted un ejemplo de uso de este operador. La función move\_uploaded\_file() genera un error que hace imposible la grabación, ya que se ha puesto, a propósito, una ruta equivocada. Cargue la página **operadorAntiError.htm** en su navegador para ver cómo funciona.

El código completo de operadorAntiError.php queda como aparece a continuación:

```
<html>
    <head>
        <script language='javascript'
type='text/javascript'>
            function volver() {
                location.href='operadorAntiError.htm';
            }
        </script>
    </head>
    <body>
<?php
    $error=false;
    // Se comprueba si el archivo tiene el formato
    adecuado.
    if ($_FILES["fichero"]["type"]!="image/pjpeg") {
        echo ("El formato del archivo no es adecuado.<br
/>");
        $error=true;
    }
    // Se comprueba que tenga un peso adecuado.
    if ($_FILES["fichero"]["size"] > 200000) {
        echo ("El archivo es demasiado grande.<br />");
        $error=true;
    }
    // Se comprueba que no se hayan producido errores.
    if ($_FILES["fichero"]["error"] != 0) {
        echo ("Hay un error en el archivo.<br />");
        $error=true;
    }
    // Si se ha producido algún error se vuelve atrás.
    if ($error){
        echo ("<input type='button' name='retorno'
id='retorno' value='VOLVER'
onClick='javascript:volver();';");
    } else {
        // Por último, se intenta grabar.
        $archivoRecibido=$_FILES["fichero"]["tmp_name"];
        $destino="ficherosEnviados/fotoDelUsuario.jpg";
        if (@move_uploaded_file($archivoRecibido,$destino))
{
            echo ("Fichero grabado");
        } else {
            echo ("El fichero no se ha podido grabar.");
        }
    }
?>
</body>
</html>
```

## 5.7 PÁGINAS AUTO PROCESADAS

Cuando se trata de enviar un formulario a un script del servidor, puede ocurrir que queramos tener el código del script en el mismo listado que el formulario. Es lo que se conoce con el nombre de **páginas auto procesadas**. La principal característica de este tipo de páginas, desde el punto de vista de la programación, es que el atributo action del formulario que contienen no recibe ningún valor, de modo que cuando se pulsa el botón destinado a enviar el formulario, este se carga contra la propia página. Esto implica varias cosas. Para empezar, la página tiene que "comprobar" si acaba de ser llamada por el navegador cliente o si ha sido llamada por el envío de su propio formulario, ya que, en el primer caso tiene que mostrarle al usuario la parte HTML (el mencionado formulario y los posibles contenidos visuales) y en el segundo caso tiene que procesar los datos y mostrar los resultados. Veamos un ejemplo de esto en **autoprocesada.php**.

```
<html>
<body>
<?php
//Creamos una constante para los saltos de línea
define ("salto", "<br />\n");
/*Comprobamos si una variable, que sabemos que se genera en
el formulario, ha sido establecida (declarada y/o asignada) o
no. Es la forma de verificar si la página acaba de ser
cargada (en cuyo caso la variable no existe) o ha sido "auto
llamada" (en cuyo caso la variable ha sido generada con el
campo del formulario. */
if (isset($_HTTP_POST_VARS["nombre"])) {
// Si la variable existe, se muestra en la página.
echo ("El valor del campo \"nombre\""
es:".$_HTTP_POST_VARS["nombre"].salto);
} else {
// Si la variable no existe, se muestra el formulario.
?>
<form action="" name="f1" id="f1" method="post">
    Nombre:
    <input type="text" name="nombre" id="nombre">
    <input type="submit" value="Mandar"
name="enviar" id="enviar">
</form>
<?php
}
?>
</body>
</html>
```

Al cargar la página en el navegador sólo verá un campo de texto, con la etiqueta **Nombre:** para que el usuario introduzca el nombre. Al pulsar el botón **Mandar** se carga la misma página (observe el atributo action del formulario en la línea resaltada) y muestra el valor de la variable.

Yo no soy especialmente partidario del uso de esta técnica, salvo en casos muy específicos. Cuando maneje formularios complejos, que cargan contra scripts complejos, verá que usar páginas auto procesadas es, por decir lo mínimo, una de las mejores formas de meterse en problemas. De todos modos, con el tiempo y la experiencia, su instinto y su conocimiento le dirán cuándo puede ser interesante considerar esta técnica.

## CAPÍTULO 6

# FUNCIONES PARA EL MANEJO DE DATOS

---

---

A la hora de realizar una gestión de información para ofrecerle al usuario unos resultados acordes con su petición, a menudo no es suficiente con disponer de datos “en bruto”, sino que se hace necesario elaborarlos de algún modo. Por ejemplo, puede ser que su sistema almacene fechas en formato aaaa-mm-dd y usted quiera ofrecerlas en formato dd/mm/aaaa, si sus usuarios son europeos, o mm-dd-aaaa, si son anglosajones. O puede ser que, dado un texto determinado, usted sólo quiera mostrar una pequeña parte del mismo como enlace a una página donde se mostrará completo. En este capítulo vamos a conocer las principales funciones que proporciona PHP para el manejo de cadenas alfanuméricas, valores aritméticos y fechas. Este capítulo es, desde mi punto de vista, uno de los más áridos para el aprendizaje. Mi consejo es que lo lea por encima para tener una visión de conjunto de las posibilidades que le ofrece PHP para el manejo de datos, y lo use como guía de consulta para utilizar cada función concreta cuando necesite llevar a cabo tal o cual acción. No intente aprenderse de memoria todas las funciones y sus sintaxis.

### 6.1 MANEJO BÁSICO DE CADENAS

Una de las operaciones más habituales consiste en extraer un fragmento de una cadena de texto. Esto lo podemos llevar a cabo con la función *substr()*, que recibe tres argumentos separados por comas. El primero es la cadena de la que queremos extraer un fragmento. El segundo es el carácter a partir del cual queremos el fragmento. Hay que tener en cuenta que los caracteres empiezan a numerarse desde cero por la izquierda, así que si, por ejemplo, ponemos un 3, nos estaremos refiriendo al cuarto carácter de la cadena, en el sentido normal de lectura. Estos dos argumentos son preceptivos. El tercer argumento es opcional y se

refiere a la longitud del fragmento, expresado en número de caracteres. Si no establecemos este argumento, el fragmento se obtendrá desde el carácter indicado en el segundo argumento hasta el final de la cadena original. En el script **subcadena.php**, dentro de la carpeta correspondiente a este capítulo, vemos un ejemplo de funcionamiento de `substr()`. El código es el que aparece a continuación:

```
<?php
$cadenaDeTexto="Esto es una cadena de texto";
$fragmento=substr($cadenaDeTexto, 8,3);
echo ("La cadena original es '$cadenaDeTexto'.<br
/>");
echo ("Aplicamos 'substr(\$cadenaDeTexto,
8,3)'.<br />");
echo ("El fragmento resultante es
'$fragmento'.<br />");
?>
```

Al cargar este script en el navegador observará el resultado de la figura 6.1.



Figura 6.1

Observe que tomamos la cadena original y extraemos un fragmento a partir del carácter 8, es decir, el noveno carácter de la cadena. A este efecto recuerde que cada espacio en blanco, signo de puntuación, número, letra, etc., cuenta como un carácter y se empiezan a contar desde el cero por la izquierda, tal como hemos mencionado. El carácter 8 corresponde a la “u” de “una”. El segundo argumento le indica a la función que queremos un fragmento de tres caracteres. He acotado la cadena y el fragmento entre comillas simples para que se aprecie mejor el principio y el final.

Otra operación muy habitual con cadenas es eliminar los espacios en blanco que quedan al principio y/o al final de una cadena. Para eliminar los espacios al principio de la cadena, usamos la función `ltrim()`. Para eliminar los que pudiera haber al final, usamos la función `chop()` y para eliminar tanto los del

principio como los del final, usamos la función ***trim()***. Estas funciones reciben un único argumento: la cadena a la que hay que eliminarle los espacios. Por espacios, a efectos de estas tres funciones, entendemos los espacios en blanco, las tabulaciones y los saltos de linea. Vemos un ejemplo en el script **recortes.php**:

```
<?php
$cadena=" Esto es una cadena de texto. ";
echo ("La cadena original es '$cadena'<br />");
echo ("El resultado de ltrim (\$cadena) es
'" .ltrim($cadena). "<br />");
echo ("El resultado de chop (\$cadena) es
'" .chop($cadena). "<br />";
echo ("El resultado de trim (\$cadena) es
'" .trim($cadena). "<br />");
?>
```

El resultado lo ve claramente en la figura 6.2.

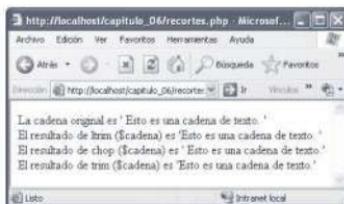


Figura 6.2

A menudo es necesario obtener el código ASCII de un carácter, o bien obtener el carácter que corresponde a un determinado código ASCII. Para ello empleamos las funciones ***ord()*** y ***chr()***, respectivamente. La primera recibe como argumento un carácter y devuelve su código ASCII. La segunda recibe un valor numérico comprendido entre 1 y 255 y devuelve el carácter correspondiente. En el script **ascii.php** vemos un ejemplo de uso de estas funciones.

```
<?php
$caracter="A";
$valor=97;
echo ("El carácter '$caracter' (mayúscula) tiene
el código ASCII ".ord($caracter). "<br />"); 
echo ("El valor $valor corresponde al carácter
'" .chr($valor). "<br />"); 
?>
```

En ocasiones tenemos que conocer el número de caracteres que forman una cadena. Para ello, empleamos la función ***strlen()***, que recibe como argumento la cadena y nos devuelve su longitud, es decir, la cantidad de caracteres que la forman, incluyendo los espacios, signos de puntuación, etc. Podemos ver un ejemplo de su uso en el script **longitud.php**:

```
<?php
$cadena="Mi libro de PHP";
$longitud=strlen($cadena);
echo ("La cadena '$cadena' tiene $longitud
caracteres.<br />");
?>
```

Cargue este script en el navegador para ver su funcionamiento.

A la hora de mostrar resultados o valores en la página, existen varias funciones que debemos tener en cuenta. En algunos textos, y también en scripts ya escritos, usted puede encontrar ***print()***, en lugar de ***echo()***. El uso es el mismo. Utilizar una u otra función para mostrar un valor es una cuestión de criterio personal. Sin embargo, hay una función muy interesante a la hora de mostrar ciertos resultados en la página: se trata de ***printf()***. Ésta recibe dos argumentos, separados por una coma. El primero es una cadena de formato, y el segundo es la cadena que queremos mostrar, de forma que nos la devuelve con el formato especificado en el primer argumento. Veamos un ejemplo de uso en el script **imprimirFormato.php**:

```
<?php
$altura="45";
echo ("Altura: ");
printf("%2.2f",$altura);
echo (" metros.");
?>
```

El resultado aparece en la figura 6.3.



Figura 6.3

La función *sprintf()* actúa como la anterior pero sin poder mostrar su resultado en la página, por lo que éste debe ser almacenado en otra variable. Tiene un ejemplo de su uso en el script **valorFormatead.php**:

```
<?php
    $precio1 = 68.75;
    $precio2 = 54.35;
    $precio = $precio1 + $precio2;
    echo ("El valor de \$precio es $precio<br />");
    $formatead = sprintf ("% .2f €", $precio);
    echo ("El mismo valor formateado es
$formatead<br />");

?>
```

El resultado de la ejecución aparece en la figura 6.4.

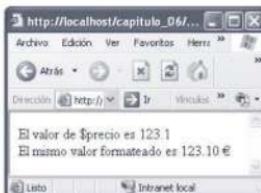


Figura 6.4

Los formatos que contemplan estas dos funciones están recopilados en la tabla de la figura 6.5.

FORMATO	SIGNIFICADO
d	Número entero en base decimal.
b	Número entero en base binaria.
o	Número entero en base octal.
x	Número entero en base hexadecimal (letras minúsculas).
X	Número entero en base hexadecimal (letras mayúsculas).
c	Carácter ASCII correspondiente al valor del argumento.
f	Números fraccionarios (usando signo decimal).
E	Números fraccionarios (usando notación exponencial).
S	Cadenas.

Figura 6.5

Veamos un ejemplo de uso de printf () que aclarará un poco más la operativa de los formatos. Vamos a suponer el caso de que usted quisiera mostrar en una tabla el consumo de combustible de un motor a lo largo de quince días, teniendo en cuenta que gasta, digamos, diecisiete litros diarios. Observe el script **tablaFormatead.php**:

```
<?php
    echo ("<u>TABLA DE GASTO DE COMBUSTIBLE</u><br
/>");
    $formato = "%' _7d * %' _3d = %' _3d<br />";
    echo ("L/diarios - Dias - Total<br />");
    for ($dia=1;$dia<=15;$dia++) {
        printf($formato, 17, $dia, $dia*17);
    }
?>
```

El resultado de la ejecución de este script aparece en la figura 6.6.



Figura 6.6

Como ve, los datos de consumo del motor aparecen alineados de modo que queda un resultado legible a primera vista. Veamos cómo hemos obtenido esta colocación de los datos. En primer lugar fíjese en el uso de printf ():

```
printf($formato, 17, $dia, $dia*17);
```

Como ve, esta función puede recibir más de dos argumentos. Los argumentos segundo y siguientes serán tenidos en cuenta como datos a mostrar adaptándose al formato especificado en el primer argumento. Respecto a este formato encontramos varios usos. Empecemos por '%\_7d'. Cuando en una definición de formato aparece el signo del porcentaje, seguido de la comilla simple, estamos indicando que el siguiente carácter deberá ser usado como de relleno para conseguir una alineación de datos. En este caso es el guion bajo. A continuación vemos el número 7. Esto quiere decir que el dato que se muestre deberá ocupar siete posiciones, completando las que faltan con el carácter de relleno. Como el dato que se muestra aquí (el número 17) tiene dos caracteres, se añaden cinco caracteres de relleno. Por último, la letra d especifica que el valor que se va a mostrar debe aparecer como un número en base 10 (el sistema habitual de numeración).

A continuación, siguiendo con la cadena de formato, encontramos un asterisco. Como éste no es un carácter específico de formato, PHP lo determina como directamente imprimible.

Además, un carácter imprimible determina que el siguiente formato se aplicará al tercer argumento de la función que, en este caso, es la variable \$dia. En general, los formatos se estructuran de la siguiente manera:

- Carácter de relleno. Conviene especificar uno diferente del espacio en blanco dado que, cuando hay varios espacios consecutivos, HTML los ignora.
- Alineamiento. Por defecto, la función alinea por la derecha, poniendo los caracteres de relleno a la izquierda. Si queremos cambiar esto, añadiremos el signo menos (-).
- Valor numérico. Determina el mínimo número de caracteres que se deben mostrar, incluyendo los del valor a mostrar y los de relleno.
- Número de decimales. Se precede por un punto (.) y especifica cuántos decimales se le deben conceder al valor a mostrar, tal como vimos en valorFormateado.php.
- Tipo de datos. Se determinan según la tabla de la figura 6.5.

De todos modos, y esto es una opinión personal, cuando necesite formatear datos para su presentación considere el uso de tablas. Sin embargo, para un formateado rápido de cadenas cuente también con la función `str_pad()`, que permite establecer caracteres de relleno a una cadena de un modo más fácil que `printf()`. Esta función recibe cuatro argumentos. El primero es la cadena que tenemos que completar con caracteres de relleno. El segundo es la longitud total que deberá tener la cadena, incluyendo dichos caracteres. El tercero es el carácter

que se usará como relleno, para completar la longitud especificada. Este argumento es opcional y por defecto, si no se especifica otra cosa, es el espacio en blanco. El cuarto parámetro también es opcional y corresponde a la alineación que se establecerá para el relleno. Por defecto, los caracteres de relleno se colocan a la derecha de la cadena. Los tres posibles valores para este parámetro son los que aparecen a continuación:

- **STR\_PAD\_LEFT**, si queremos que el relleno se sitúe a la izquierda de la cadena.
- **STR\_PAD\_RIGHT** o ninguno, si queremos que el relleno se sitúe a la derecha de la cadena.
- **STR\_PAD\_BOTH**, si queremos que los caracteres de relleno se distribuyan a ambos lados de la cadena.

Para ver un ejemplo de funcionamiento observe el listado del script **rellenaCadena.php**, donde hemos omitido, a propósito, el argumento relativo a la alineación del relleno.

```
<?php  
$cadena="Esto es una cadena";  
$cadena=str_pad($cadena, 25, "+");  
echo ($cadena);  
?>
```

El resultado será que la cadena se mostrará tal como aparece a continuación:

```
Esto es una cadena++++++
```

También contamos con una función específica para repetir una cadena. Se trata de **str\_repeat()**. Esta función recibe dos argumentos. El primero es la cadena en cuestión y el segundo, el número de veces que se debe repetir dicha cadena. Veamos un ejemplo en el script **repiteCadena.php**:

```
<?php  
$cadena="*CADENA*";  
$cadena=str_repeat($cadena, 4);  
echo ($cadena);  
?>
```

En la página le aparecerá lo siguiente:

```
*CADENA***CADENA***CADENA***CADENA*
```

Normalmente, por la naturaleza del uso de esta función, la cadena que constituye el primer argumento suele estar formada por un solo carácter, para crear una línea, un relleno, cosas así.

Con mucha frecuencia es necesario poner todas las letras en mayúsculas o en minúsculas en una cadena antes de mostrarla. Para ello, contamos con las funciones *strtoupper()* y *strtolower()*. Estas funciones reciben como argumento la cadena que queremos modificar. Tiene un ejemplo de funcionamiento en **mayusMinus.php**:

```
<?php
$cadena="Esto Es Una Cadena.";
$mayus=strtoupper($cadena);
$minus=strtolower($cadena);
echo ("La cadena original es $cadena<br />");
echo ("Todo en mayúsculas es $mayus<br />");
echo ("Todo en minúsculas es $minus<br />");
?>
```

Cargue el script en el navegador para ver su funcionamiento.

Existen otras dos funciones que podríamos considerar complementarias de las anteriores: se trata de *ucfirst()* y *ucwords()*. Ambas reciben como argumento una cadena o el nombre de una variable que contiene una cadena. La primera función convierte a mayúscula el primer carácter de la cadena (si es una letra, claro). La segunda convierte a mayúscula la primera letra de cada palabra.

Es importante puntualizar que *estas funciones no convierten ninguna letra en minúscula, es decir, si la cadena ya está en mayúsculas, no resulta modificada*. Por esta razón es conveniente aplicarle a la cadena la función *strtolower* previamente al uso de alguna de estas dos últimas. Vea un ejemplo de uso en el script **capitaliza.php**:

```
<?php
$cadena="esto es una cadena.";
$mayus1=ucfirst($cadena);
$mayus2=ucwords($cadena);

echo ("La cadena original es $cadena<br />");
echo ("El efecto de ucfirst es $mayus1<br />");
echo ("El efecto de ucwords es $mayus2<br />");
?>
```

En la página le aparecerá lo siguiente:

```
La cadena original es esto es una cadena.
El efecto de ucfirst es Esto es una cadena.
El efecto de ucwords es Esto Es Una Cadena.
```

PHP nos proporciona la posibilidad de cambiar determinados caracteres por otros dentro de una cadena. Para ello, podemos usar la función `str_replace()`, que sustituye un fragmento de una cadena por otro, dejando inalterado el resto de la cadena original. Esta función recibe tres argumentos, separados por comas. El primero es el fragmento que deseamos sustituir; el segundo, el fragmento que deseamos que quede en la cadena y el tercero es la propia cadena sobre la que vamos a trabajar. Partimos de la sintaxis genérica de esta función:

```
str_replace ($frag1, $frag2, $original)
```

Esto hará que la función busque, en \$original, todas las apariciones de \$frag1 y las sustituya por \$frag2. Como ya es habitual, veamos un ejemplo de funcionamiento, en el script **cambiaTexto.php**:

```
<?php
$cadenaOriginal="Mi mesa es blanca y tu mesa es
negra.<br />";
echo ($cadenaOriginal);
$cadenaModificada=str_replace("mesa", "silla",
$cadenaOriginal);
echo ($cadenaModificada);
?>
```

Cuando cargue este script, su página será como la de la figura 6.7.



Figura 6.7

Ponga especial cuidado en la sintaxis de esta función. En versiones anteriores de PHP, el orden de los dos primeros argumentos estaba cambiado. Como usted está utilizando PHP 5 no tiene ningún problema.

Tenga en cuenta que esta función es sensible a mayúsculas y minúsculas, de modo que, si no escribe correctamente el fragmento que debe ser remplazado, PHP no podrá localizarlo y no efectuará las sustituciones previstas.

Otra función de sustitución con la que contamos en PHP es `strtr()`. Ésta se utiliza para sustituir todas las apariciones de fragmentos o caracteres de una cadena por otros. Dicho así suena muy parecido a lo que acabamos de ver con `str_replace()`, pero hay una diferencia. La función `strtr()` establece correspondencia entre los caracteres originales y los de sustitución, uno a uno. Suponga que tiene que hacer un trabajo que implique la presentación de un texto sin acentos, y el texto original incluye letras acentuadas. En este caso escribiríamos algo así:

```
$nuevaCadena = strtr ($cadena, "ÁÉÍÓÚáéíóú",
"AEIOUaeiou");
```

En este ejemplo, la variable `cadena` contiene el texto original, con algunas letras acentuadas. En la variable `$nuevaCadena` se almacenará una copia del texto en la que las “Á” habrán sido sustituidas por “A”, las “É” se habrán remplazado con “E”, y, así, sucesivamente. Es decir, cada aparición de un carácter del segundo argumento, será sustituido por el correspondiente del tercero. Veamos un ejemplo en el script `cambiaCaracter.php`:

```
<?php
$cadenaOriginal="INÉS SÁNCHEZ es Técnica en
Electrónica.<br />";
echo ($cadenaOriginal);
$cadenaModificada=strtr($cadenaOriginal,
"ÁÉÍÓÚáéíóú", "AEIOUaeiou");
echo ($cadenaModificada);
?>
```

El resultado corresponde a la figura 6.8.



Figura 6.8

Observe que para una situación como ésta difícilmente podríamos usar str\_replace(), ya que tendríamos que incluir esta función diez veces en el script para contemplar todos los posibles casos de vocales acentuadas, y eso solamente con el español. Imagine ahora si el texto está en francés, alemán o polaco, que tienen más variedad de tildes que nuestro idioma.

A menudo es necesario determinar si una cadena contiene una determinada subcadena. Para esto contamos con la función *strstr()*, que recibe dos argumentos separados por una coma. El primero es la cadena en la que hay que buscar. El segundo es la subcadena que deseamos encontrar. Vea el listado del script **compruebaSubcadena.php**:

```
<?php
    $cadena="Esta es la cadena en la que se buscará
una subcadena.";

    echo ("La cadena es: $cadena<br />");
    $subcadena="la cadena";
    echo ("La subcadena a buscar es: $subcadena<br
/>");

    if (strstr ($cadena,$subcadena)){
        echo ("La subcadena SI existe dentro de la
cadena.<br />");
    } else {
        echo ("La subcadena NO existe dentro de la
cadena.<br />");

    }
?>
```

Al ejecutar este script la función *strstr()* determina que la subcadena está dentro de la cadena. Sin embargo, no nos dice en qué posición. Y esto es algo que, a menudo, vamos a necesitar saber. Para ello contamos con la función *strpos()*. Con la misma sintaxis que la anterior, nos devuelve el punto donde la subcadena se encuentra en la cadena. A este efecto recuerde que los caracteres de una cadena se empiezan a contar por la izquierda y por el cero, de modo que el primer carácter es el 0, el segundo el 1, y así sucesivamente. Observe el script **posicion.php**:

```
<?php
    $cadena="Esta es la cadena en la que se buscará
una subcadena.";
    echo ("La cadena es: $cadena<br />");
    $subcadena="la cadena";
    echo ("La subcadena a buscar es: $subcadena<br
/>");

    $posicion=strpos ($cadena, $subcadena);
```

```
if ($posicion) {
    echo ("La subcadena se inicia dentro de la
cadena en la posición: $posicion.<br />");
} else {
    echo ("La subcadena no forma parte de la
cadena.<br />");}
?>
```

Observe en la primera línea resaltada la sintaxis de la función. En la segunda línea resaltada fíjese en la forma en que hemos usado el condicional. Esto es así porque si la subcadena existe dentro de la cadena, strpos () nos devuelve su posición, pero si no existe nos devuelve un valor lógico false. Para comprender la operativa de este condicional ejecute primero el script, tal como está. Deberá obtener un resultado similar al de la figura 6.9.

A continuación, edite el código para cambiar la línea que define la variable \$subcadena, de modo que quede, por ejemplo, así:

```
$subcadena="XXXX";
```

Ejecute de nuevo el script. El resultado esta vez se parecerá a la figura 6.10. Esto es la forma habitual de trabajar del intérprete de PHP. Las funciones, cuando no pueden obtener un resultado coherente con su objetivo, devuelven un valor lógico false. Esto resulta muy útil para poder determinar mediante condicionales, como hemos hecho aquí, si se ha obtenido o no un resultado. En muchos códigos verá cómo empleamos este recurso con cierta asiduidad.



Figura 6.9

Verifique en el código cómo el valor true o false devuelto por la función puede usarse en condicionales que, de este modo, se hallan supeditados a la ejecución de una función.

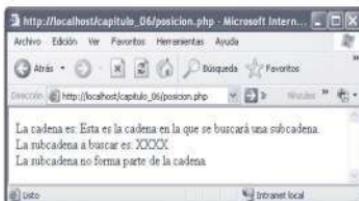


Figura 6.10

Después de hacer esta comprobación, vuelva a editar el script para dejarlo como estaba al principio.

## 6.2 LA CODIFICACIÓN URL

Cuando hablábamos del envío de formularios en el capítulo 5 dijimos que, al enviar un formulario al servidor mediante el método GET, los campos, y los valores introducidos por el usuario, se transmiten a través de la URL en un formato específico que se conoce como *codificación URL*. Considere las dos cadenas de texto siguientes:

**Esta es una cadena de texto que será codificada como URL. ¿Cómo se mostrará? ;¡VEÁMOSLO!!**

y

**Esta%20es%20una%20cadena%20de%20texto%20que%20ser%E1%20codificada%20como%20URL.%20%BFC%F3mo%20se%20mostrar%E1%3F%20%A1%A1VE%C1MOSLO%21%21**

La primera cadena es una frase normal, completamente legible. La segunda cadena es, a priori, ilegible. Si la examinamos someramente por encima podemos intuir que, en realidad, es la misma que la primera, codificada de algún modo. El tipo de cifrado corresponde a las estipulaciones de los RFC 1630 y 1738.

**Los RFC (Request For Comments) son documentos, universalmente aceptados, que regulan el funcionamiento y modo de uso de todos los protocolos de comunicaciones en Internet, métodos de cifrado y transmisión de datos, formatos de cabeceras de páginas, etc. Constituyen una riquísima fuente de información y documentación para programadores que se enfrenten a trabajos con redes de ordenadores y, en general, para cualquier persona que sienta curiosidad por estos temas. Para saber más acerca de los RFC consulte el Apéndice D, donde encontrará referencias a sitios especializados de Internet. (N del A).**

Básicamente se trata de lo siguiente. Existen determinados caracteres conocidos como "inseguros". Esto se refiere al hecho de que la transmisión de dichos caracteres en una URL "abre" lo que se conoce como "agujeros de seguridad" o "bugs". Son "puertas traseras" que utilizan los hackers para acceder a zonas de los servidores que normalmente son reservadas, o para abrir ficheros para los que hace falta algún tipo de autorización especial. Por razones de seguridad, estos caracteres no deben ser transmitidos en una URL. Y, sin embargo, son necesarios, así que hay que transmitirlos. Lo que se hace es codificarlos antes de la transmisión y decodificarlos en la recepción, siguiendo un patrón muy eficiente: se toma el código ASCII del carácter que queremos cifrar, se expresa con dos dígitos en hexadecimal y se le antepone el símbolo %. Así pues consideremos, por ejemplo, el más obvio de los caracteres inseguros: el espacio en blanco. Su código ASCII es el 32. En hexadecimal se expresa como 20. Por lo tanto, el espacio en blanco, cifrado según la codificación URL, queda como %20. Los caracteres inseguros son el espacio en blanco, las letras acentuadas (tanto mayúsculas como minúsculas), los signos de puntuación, excepto el punto y el guion (-), y los saltos de línea. Dicho de otro modo, los caracteres no sujetos a codificación son los dígitos del 0 al 9, las letras del alfabeto inglés, el punto y el guion.

PHP permite cifrar textos según la codificación URL. Para ello recurrimos a la función *rawurlencode()*, que recibe un texto "normal" y lo devuelve cifrado. Como es lógico, también existe una función que permite recuperar el texto original. Se trata de *rawurldecode()*. Observe el listado **codificarUrl.htm**:

```
<html>
  <body>
    <p>
      Teclee un texto para su codificaci&acute;n URL.
    </p>
    <form id="formulario1" name="formulario1"
method="post" action="codificarUrl.php">
      <p>
        <textarea name="textoOriginal" cols="50"
rows="4" id="textoOriginal">
          </textarea>
        </p>
        <p>
          <input name="mandar" type="submit" id="mandar"
value="ENVIAR">
          <input name="borrar" type="reset" id="borrar"
value="BORRAR">
        </p>
      </form>
    </body>
  </html>
```

Como ve es una página que contiene un formulario. En dicho formulario hay una caja de texto multilínea y los botones típicos de tipo submit y reset. Escriba un texto con caracteres diversos en la caja de texto, como en la figura 6.11.

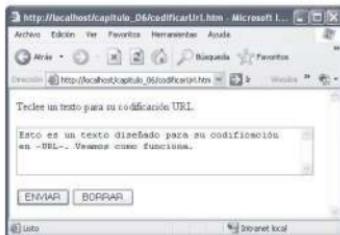


Figura 6.11

Al enviar el formulario se manda el contenido de la caja de texto al script **codificarUrl.php**, cuyo código aparece a continuación:

```
<?php  
    $textoCodificado=rawurlencode($textoOriginal);  
    $textoRecuperado=rawurldecode($textoCodificado);  
?  
<html>  
    <body>  
        <table width="505" border="1">  
            <tr>  
                <td width="505">  
                    <table align="left" width="500"  
cellpadding="4" border="2">  
                        <tr>  
                            <th>TEXTO ORIGINAL</th>  
                        </tr>  
                        <tr>  
                            <td bgcolor="#C0C0C0">  
                                <?php echo ($textoOriginal); ?>  
                            </td>  
                        </tr>  
                    </table>  
                </td>  
            </tr>  
        <tr>  
            <td width="505">  
                <table align="left" width="500"  
cellpadding="4" border="2">
```

```

<tr>
    <th>TEXTO CODIFICADO</th>
</tr>
<tr>
    <td bgcolor="#C0C0C0">
        <?php echo ($textoCodificado); ?>
    </td>
</tr>
</table>
</td>
</tr>
<tr>
    <td width="505">
        <table width="500" cellpadding="4"
border="2">
            <tr>
                <th>TEXTO RECUPERADO</th>
            </tr>
            <tr>
                <td bgcolor="#C0C0C0">
                    <?php echo ($textoRecuperado); ?>
                </td>
            </tr>
            </table>
        </td>
    </tr>
</table>
</body>
</html>

```

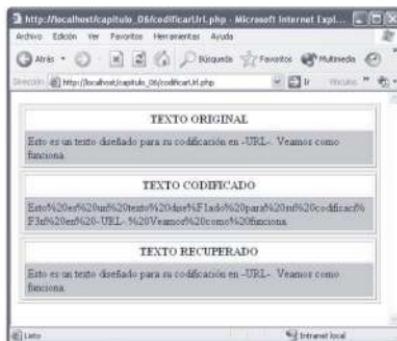


Figura 6.12

Cargue en el navegador la página codificarUrl.htm, escriba en la caja de texto lo mismo que aparece en la figura 6.11 y pulse el botón **ENVIAR**. Al cargarse el script codificarUrl.php obtendrá un resultado como el de la figura 6.12.

En las líneas resaltadas en el código se ve cómo usamos las dos funciones, para cifrar el texto y para descifrarlo. Estas funciones tienen muchísima utilidad en varias situaciones. Una de las más corrientes es cuando el texto que el usuario escribe está destinado a almacenarse en una base de datos o en un archivo de disco. También es importante realizar esta codificación cuando el texto va a ser posteriormente mostrado formando parte de una página. Lógicamente, cuando llegue el momento de mostrarlo, será necesario descifrarlo previamente con la función rawurldecode(). De lo contrario, se mostraría al usuario un contenido codificado e ilegible.

Quiero hacer un pequeño inciso para llamar su atención sobre un detalle en el script. Fíjese en la forma de introducir el texto, que está en variables de PHP, dentro de tablas HTML. Por ejemplo, así:

```
<td bgcolor="#C0C0C0">  
    <?php echo ($textoRecuperado); ?>  
</td>
```

Este modo de integrar código PHP con la salida en HTML es perfectamente válido y muy práctico en infinidad de ocasiones. Fíjese en que, dentro de la celda de una tabla se inicia PHP, se muestra el contenido de una variable y se finaliza PHP para volver a HTML. Dentro de sus páginas podrá iniciar y detener el intérprete de PHP tantas veces como lo necesite. Más adelante aprenderemos también a incluir código del intérprete PHP dentro de JavaScript.

Llegados a este punto usted estará, seguramente, pensando: “Bueno, sí. Pero el texto se transmite tal como el usuario lo ha tecleado. Cuando el formulario manda su contenido al script de PHP todavía no se ha hecho ningún cifrado sobre el texto, sino que éste se hace en el servidor, una vez que ya se ha iniciado el script PHP”. Pues sí, tiene usted razón. Pero yo sólo quería mostrarle estas funciones, para empezar. Ahora vamos a ver cómo se puede llevar a cabo “algo” para que el texto se envíe ya codificado.

Para eso tenemos que recurrir a un pequeño truco, muy usado en ciertos entornos, que consiste en incluir dos formularios, en lugar de uno solo, en la página donde el usuario escribe el texto. Además, necesitamos contar con la ayuda de nuestro viejo amigo JavaScript. Observe el script **urlConJs.htm**:

```
<html>
    <head>
        <script language="javascript"
type="text/javascript">
            function comprobarFormulario () {
                document.getElementById
("ocultoTexto") .value=escape(document.getElementById("textoOr
iginal") .value);
                return true;
            }
        </script>
    </head>
    <body>
        <p>
            Teclee un texto para su codificaci&on URL.
        </p>
        <form id="formulario1" name="formulario1">
            <p>
                <textarea name="textoOriginal" cols="50"
rows="4" id="textoOriginal"></textarea>
            </p>
            </form>
            <form id="formulario2" name="formulario2"
method="post" action="urlConJs.php" onSubmit="javascript:
return comprobarFormulario();">
            <p>
                <input type="hidden" name="ocultoTexto"
id="ocultoTexto" value="">
                <input name="mandar" type="submit" id="mandar"
value="ENVIAR">
                <input name="borrar" type="reset" id="borrar"
value="BORRAR">
            </p>
        </form>
    </body>
</html>
```

Como ve, en el código he resaltado tres bloques. El primero corresponde a una función JavaScript. Enseguida hablaremos de ella. Fijese en el segundo bloque, el que corresponde al primer formulario. Sólo tiene la caja de texto donde el usuario escribirá. Repare en que a este formulario se le han quitado los botones de submit y reset y ni siquiera tiene ya el atributo action. Esto es debido a que los campos de este formulario (el único campo que posee, en este caso) ya no van a ser enviados a ninguna parte. Se quedan en esta página. El texto original que teclee el usuario no va a ir al servidor ni a ningún otro sitio.

Ahora mire el segundo formulario. El parámetro action si está establecido, para enviar los campos que contiene al script que los procesará en el lado del servidor. Sin embargo, vemos que sólo contiene, aparte de los botones, un campo oculto. Además, fíjese en que hemos establecido el evento onSubmit, que llama a una función de JavaScript cuando el usuario pulsa el botón de envío.

Ejecute este script. Verá que el usuario no percibe ninguna diferencia con respecto al anterior. No se ve que haya dos formularios, ni se aprecia nada inusual. Cuando el usuario escribe algo en la caja de texto y pulsa el botón de envío, se ejecuta la función de JavaScript que, en este ejemplo, hemos llamado comprobarFormulario () .

Este pequeño script se encarga de recuperar el texto que el usuario ha escrito en el textarea, le aplica la función escape () , que, como usted sabe, es el equivalente JavaScript de la función rawurlencode de PHP, y almacena el texto cifrado en el campo oculto del segundo formulario. Entonces es cuando se produce el envío. Pero no es el texto original el que se envía, sino el texto cifrado que está almacenado en el campo oculto del segundo formulario.

En la figura 6.13 se ve un ejemplo especialmente interesante del uso de esta página. Enseguida le explico lo de “especialmente interesante”.

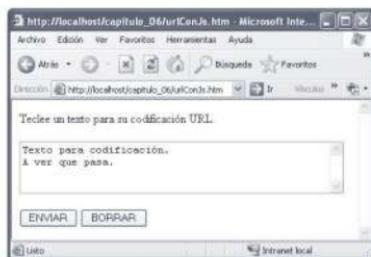


Figura 6.13

Al pulsar el botón **ENVIAR** se carga el script **urlConJs.php**, cuyo código es el siguiente:

```
<?php
    $textoRecuperado=rawurldecode($ocultoTexto);
?>
<html>
<body>
```

```
<table width="505" border="1">
<tr>
    <td width="505">
        <table align="left" width="500"
cellpadding="4" border="2">
            <tr>
                <th>TEXTO ORIGINAL (YA CODIFICADO)</th>
            </tr>
            <tr>
                <td bgcolor="#C0C0C0">
                    <?php echo ($ocultoTexto); ?>
                </td>
            </tr>
            </table>
        </td>
    </tr>
    <tr>
        <td width="505">
            <table align="left" width="500"
cellpadding="4" border="2">
                <tr>
                    <th>TEXTO RECUPERADO</th>
                </tr>
                <tr>
                    <td bgcolor="#C0C0C0">
                        <?php echo ($textoRecuperado); ?>
                    </td>
                </tr>
                </table>
            </td>
        </tr>
    </table>
</body>
</html>
```

El resultado en este ejemplo corresponde al que vemos en la figura 6.14.

Como ve, hemos logrado enviar y recibir el texto ya cifrado. Y ahora viene la explicación de por qué antes le decía que este ejemplo es interesante. Fíjese de nuevo en la figura 6.13. Vea que el usuario ha tecleado el texto en dos líneas. Sin embargo, al recuperarlo para mostrarlo en la siguiente página y decodificarlo, lo recuperamos en una sola línea. En muchos casos puede que esto sea válido, pero recientemente yo tuve que hacer un trabajo en el que el texto debía recuperarse, *exactamente*, como lo había tecleado el usuario en origen. Esto sucede porque el carácter de salto de línea que maneja PHP no corresponde con la etiqueta `<br />` de HTML, así que este último se ve incapaz de mostrarlo en la página. Lo que

tenemos que hacer es buscar la secuencia del texto codificado que corresponde a un salto de línea y sustituirla por una etiqueta `<br />` *antes* de hacer la decodificación. Y, para esto, nos va a ayudar la función `str_replace()` que hemos conocido en este mismo capítulo.

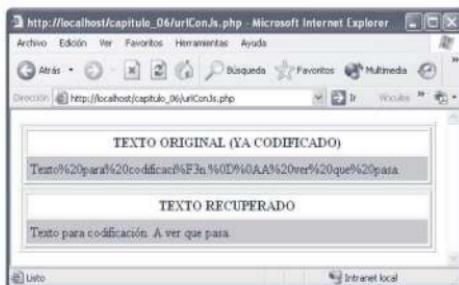


Figura 6.14

Cargue en el navegador la página `urlCheck.htm`. Es la misma que `urlConJs.htm`, pero enviando los datos a otro script: el script `urlCheck.php`, que hace el uso adecuado de esta función. Observe la modificación en la línea que decodifica el texto:

```
$textoRecuperado=rawurldecode(str_replace("%0D%0A",
"<br />",$ocultoTexto));
```

El resultado será el que aparece en la figura 6.15.

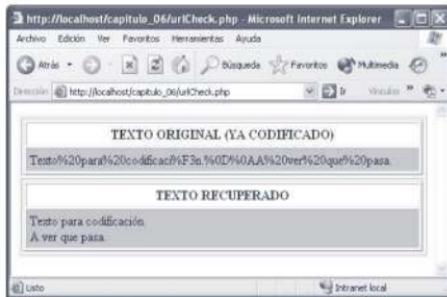


Figura 6.15

Lo que hemos hecho ha sido identificar la secuencia que corresponde al salto de línea (%0D%0A), y sustituirla por <br /> antes de decodificar. *Cuando dos o más funciones de PHP están anidadas una dentro de otra, como en este caso, el intérprete las resuelve desde dentro hacia fuera, de forma que primero se resuelve la más interior, y la siguiente hacia fuera opera con el resultado de la anterior.* En este caso, primero se ejecuta str\_replace (), y rawurldecode () opera con el resultado obtenido.

De este modo, ya sabe usted cómo codificar texto para su envío por La Red, y su posterior almacenamiento en ficheros o bases de datos en el lado del servidor (el tema del almacenamiento será tratado en posteriores capítulos). También sabe cómo recuperar los textos y mostrarlos en la página igual que los tecleó el usuario.

Para cifrar y descifrar textos puede usar también las funciones *urlencode ()* y *urldecode ()*. Son similares a las que ya conoce. La diferencia más evidente que usted va a apreciar es que los espacios en blanco son sustituidos por un signo + en lugar de por la secuencia %20. Esto se emplea cuando el cifrado debe adaptarse a un formato MIME estándar. Si en alguna ocasión se le presenta el caso, use estas dos funciones. Si no, puede usar el juego que quiera. Yo, personalmente, suelo inclinarme por las dos que hemos conocido en primer lugar.

En **encodeDecode.php** tiene un ejemplo de uso de estas dos últimas funciones. Ejecútelo en el navegador y vea el resultado. Como puede comprobar, los espacios en blanco son sustituidos por signos +. Puede hacer algunas pruebas cambiando el contenido de la variable \$cadena, introduciendo distintos caracteres, para ver el comportamiento del script.

```
<?php
$cadena="Esta es la cadena que usaremos para su
codificación";
echo ("La cadena es: $cadena<br />");
$cadenaCodificada=urlencode($cadena);
echo ("La cadena codificada es:
$cadenaCodificada<br />");
$cadenaRecuperada=urldecode($cadenaCodificada);
echo ("La cadena recuperada es:
$cadenaRecuperada<br />");?
>
```

Antes de seguir adelante vamos a recordarle lo que hemos hecho en el urlConJs.htm. El usar dos formularios para una página, uno para que el usuario introduzca los datos y otro donde son almacenados en campos ocultos para su envío, es una solución muy cómoda en muchos casos. Si usted usa en su formulario

cajas de texto, casillas de verificación, botones de opción, etc., encontrará esta técnica sumamente útil. Más adelante, en posteriores capítulos, verá más ejemplos de su uso.

## 6.3 TRATAMIENTO DE CADENAS PARA HTML

En este apartado vamos a conocer algunas funciones que PHP incluye para el tratamiento de cadenas específico para su integración con HTML, así como para gestionar algunos aspectos del entorno web, aunque más adelante conoceremos detalles más avanzados.

Es muy común que el contenido de una variable alfanumérica deba ser mostrado en la página. Y también lo es que contenga determinados caracteres que no correspondan al alfabeto internacional. Como usted sabe, en HTML se contempla que determinados caracteres deben ser sustituidos por las entidades. Por ejemplo, la letra ñ se escribe en una página como &ntilde; para que todos los navegadores del mundo puedan mostrarla sin problemas. Sin embargo, PHP no traduce estos caracteres directamente. Suponga un script como el siguiente:

```
<html>
  <body>
    <?php
      $cadena="España";
      echo ($cadena);
    ?>
  </body>
</html>
```

Si usted carga este script en su navegador y trata de ver el código fuente de la página se encontrará con algo parecido a esto:

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

Fíjese en la línea resaltada. En realidad, para obtener una correcta programación en HTML, esta línea debería ser Espa&ntilde;a. PHP cuenta con una solución para estos casos. Se trata de la función **htmlentities()**. Ésta recibe como argumento una cadena y sustituye todos los caracteres no pertenecientes al alfabeto internacional por las entidades HTML oportunas, devolviendo la cadena lista para su presentación en la página. Observe el script **entidades.php**:

```
<html>
  <body>
    <?php
      $cadena="España";
      echo (htmlentities($cadena));
    ?>
  </body>
</html>
```

Cuando quiera ver el código fuente de la página en su navegador, obtendrá lo siguiente:

```
<html>
  <body>
    Espa&ntilde;a  </body>
</html>
```

Como ve, la letra ñ ha sido convertida a su correspondiente entidad.

Otra función que también genera las entidades HTML de una cadena es *htmlspecialchars()*. Sin embargo, ésta es más antigua y no contempla todas las posibles entidades.

En PHP contamos con la función *get\_html\_translation\_table()*, que genera una matriz con todas las entidades que contempla HTML. Ésta es una matriz asociativa, en la que la clave de cada elemento es el carácter y el valor es la entidad HTML. Observe el script *listaEntidades.php*:

```
<html>
  <body>
    <?php
      $matrizDeEntidades=get_html_translation_table(
HTML_ENTITIES);
      echo ("<table border='2' cellpadding='2'
cellspacing='4'>\n");
      echo ("<tr>\n");
      echo ("<th colspan='16'>LISTADO DE ENTIDADES
HTML</th>\n");
      echo ("</tr>\n<tr>\n");
      for ($contador=0; $contador<8; $contador++){
        echo
      ("<td>Car.</td>\n<td>Entidad</td>\n");
      }
      echo ("</tr>\n");
      echo ("<tr>\n");
      $contador=0;
```

```

foreach ($matrizDeEntidades as $clave=>$valor){
    $valor=substr($valor,1);
    echo
("" $clave</td>\n<td>&#$valor</td>\n");     $contador++;     if ($contador>7){         echo ("</tr>\n<tr>\n");         $contador=0;     } } echo ("</tr>\n"); echo ("</table>"); ?> </body> </html> |
```

El resultado es la tabla que aparece, parcialmente, en la figura 6.16.

LISTADO DE ENTIDADES HTML											
Car.	Entidad	Car.	Entidad	Car.	Entidad	Car.	Entidad	Car.	Entidad	Car.	Entidad
	&nbspl;	i	&iacutel;	#	&cent;	£	&pound;	¤	&curren;	¥	&yen;
-	&suml;	©	&copy;	±	&ordf;	«	&laquo;	¬	&not;	&shy;	®
*	&deg;	±	&plusmn;	²	&sup2;	³	&sup3;	°	&acute;	μ	&micro;
:	&cedil;		&sup1;	º	&ordm;	»	&quo;	%	&frac14;	½	&frac12;

Figura 6.16

Como ve, se genera una tabla con todas las entidades HTML de caracteres no reconocidos en el alfabeto internacional. Observe en el listado del script cómo hemos empleado la función `get_html_translation_table(0)`. Como argumento le hemos pasado **HTML\_ENTITIES**, una constante de PHP que hace referencia a la lista de entidades. La alternativa sería haberle pasado **HTML\_SPECIALCHARS**. Sin embargo, el resultado es mucho más pobre, como se ve en la figura 6.17.

LISTADO DE ENTIDADES HTML											
Car.	Entidad	Car.	Entidad	Car.	Entidad	Car.	Entidad	Car.	Entidad	Car.	Entidad
"	&quot;	<	&lt;	>	&gt;	&	&amp;				

Figura 6.17

Otra función bastante útil en ocasiones es `parse_url(0)`. Esta recibe como argumento la dirección de un documento en Internet o en una Intranet, y devuelve

una matriz con todos los elementos que forman dicha dirección. Observe el script **datosUrl.php**:

```
<html>
<body>
<?php
    $datosDeURL=parse_url("http://www.todoligues.com"
);
    echo ("<table border='2' cellpadding='2'
cellspacing='2'>\n");
    echo ("<tr>\n<th colspan='2'>DATOS DE
URL</th>\n");
    echo ("<tr>\n<th>Dato</th>\n<th>Valor</th>\n");
    echo ("</tr>\n");
    foreach ($datosDeURL as $clave=>$valor) {
        echo
    ("<td>$clave</td>\n<td>$valor</td>\n</tr>\n<tr>\n");
    }
    echo ("</tr>\n");
    echo ("</table>\n");
?
</body>
</html>
```

Observe en la linea resaltada el uso de esta función. El resultado corresponde a la figura 6.18.

DATOS DE URL	
Dato	Valor
scheme	http
host	www.todoligues.com

Figura 6.18

Realmente ésta es una dirección muy simple. Esta función puede sacar más datos, tal como se ve en la tabla de la figura 6.19.

DATO	CONTENIDO
scheme	El protocolo de comunicación. Normalmente suele ser http. Sin embargo, puede ser ftp, file, etc. dependiendo de la comunicación establecida con el recurso.
host	La dirección IP o el nombre del servidor.

DATO	CONTENIDO
port	El número de puerto en el que se espera que el servidor escuche.
user	El login del usuario que accede al recurso.
password	La clave de acceso del usuario especificado.
path	La ruta, dentro del host, donde está el recurso al que se pretende acceder.
query	Los datos que se le pasen al recurso, como es el caso de campos de un formulario enviados mediante GET.

Figura 6.19

En la mayoría de los casos, una URL no tiene especificados todos los posibles valores, como es el ejemplo que hemos visto en el script datosUrl.php.

## 6.4 LAS CADENAS COMO MATRICES

PHP (y algunos otros lenguajes) contempla la posibilidad de convertir una cadena alfanumérica en una matriz, y viceversa: tomar una matriz y colocar todos sus elementos, uno a continuación de otro, formando una cadena que, lógicamente, puede ser almacenada en una variable.

Para convertir una cadena en una matriz de datos debemos seguir dos pasos. En primer lugar, contaremos con la función **chunk\_split()**. Ésta recibe como parámetro obligatorio la cadena que hay que dividir (o la variable que la contiene) y genera otra cadena en la que, cada 76 caracteres, introduce una secuencia “\r\n” (retorno de carro y nueva línea). Como parámetros opcionales podemos incluir el número de caracteres que se contarán en cada parte de la cadena resultante, en lugar de 76, así como la secuencia que actúa como separador, en lugar de “\r\n”. Por ejemplo, suponga que quiere introducir una secuencia “\*-\*” cada 10 caracteres. La sintaxis a emplear sería la siguiente:

```
$cadenaNueva=chunk_split($cadenaOriginal,10,"*-*");
```

Es importante que comprenda que **chunk\_split()** *NO* modifica la cadena que recibe como argumento, sino que genera otra variable con la cadena modificada, así que la original permanece inalterada.

Una vez que tenemos la cadena modificada en otra variable llega el momento de usar la función **explode()**, para convertir dicha cadena en una matriz. Esta función recibe dos parámetros obligatorios. El primero es la secuencia que debe buscar como separador y que es la que establecimos en **chunk\_split()**. El

segundo es el nombre de la cadena modificada, generada por chunk\_split(). La función explode() devuelve una matriz en la que creará un nuevo elemento cada vez que encuentre el separador. Así pues, la matriz tendrá tantos elementos como separadores haya, más uno. Para aclarar conceptos, veamos el uso combinado de estas dos funciones en el listado del script **cadena2matriz.php**:

```
<?php
$cadOriginal="0123456789abcdefghijklmnopqrstuvwxyz";
zABCDEFGHIJKLMNPQRSTUVWXYZ";
//Se usa la funcion chunk_split para generar una cadena
// en la que,cada 10 caracteres, se introduce la
secuencia "*.*".
$cadenaModificada=chunk_split($cadOriginal,10,"*.*");
//Se muestran las cadenas en la página para ver que
//la cadena original no se ha alterado.
echo ("La cadena original es: $ cadOriginal <br
/>"); 
echo ("La cadena modificada es:
$cadenaModificada<br /><br /><br />");
//Se genera una matriz a partir de la cadena
modificada.
$matriz=explode ("*.*",$cadenaModificada);
/*Cada elemento de la matriz contendrá un fragmento de
la cadena original. PHP usa la secuencia separadora ("*.*",
en este ejemplo) para saber donde debe empezar y acabar cada
elemento de la matriz. La matriz se genera, de forma
automática con un último elemento vacío. No olvide tener esto
en cuenta.*/
$elementosDeLaMatriz=count($matriz);
echo ("La matriz tiene $elementosDeLaMatriz
elementos.<br />"); 
for
($contador=0;$contador<$elementosDeLaMatriz;$contador++){
    echo ("El elemento $contador contiene
'$matriz[$contador]'<br />"); 
}
?>
```

Observe en las líneas resaltadas el uso de las dos funciones que acabamos de conocer. Además de lo que ya sabemos, la función explode() admite un tercer parámetro numérico opcional, que hace referencia al número máximo de elementos que deberá tener la matriz. Si establecemos un número de elementos inferior a lo que la matriz necesitaría de por sí, el último elemento contendrá el final de la cadena, incluyendo los separadores como si fueran parte del elemento. De todos modos es muy raro que usted llegue a necesitar de este tercer parámetro. Cuando ejecute este script verá el resultado de la figura 6.20.

En la imagen puede comprobar cómo la función `chunk_split()` añade a la cadena los caracteres especificados como separador ("\*.\*") cada diez caracteres, tal como hemos indicado, y cómo la función `explode` genera la matriz.

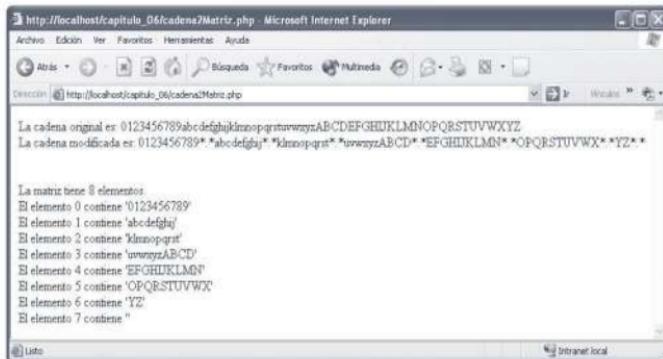


Figura 6.20

La función `implode()` lleva a cabo la misión contraria a `explode()`, es decir, parte de una matriz y une todos sus elementos formando una cadena. Como argumento recibe el nombre de la matriz cuyos elementos deberán constituir la cadena que nos interesa. Para comprobar su funcionamiento he preparado el script `explotaMatriz.php`, que es similar al anterior, aunque le he añadido el siguiente fragmento de código:

```
//Recuperamos la cadena original a partir de la matriz
con implode().
    $nuevaCadena=implode($matriz);
    echo ("La cadena recuperada es: $nuevaCadena<br
/>");
```

Al ejecutarlo verá una página similar a la de la figura 6.20, sólo que ahora aparece, al final, una nueva línea de resultado:

```
La cadena recuperada es:
0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ
Z
```

La función `join()` es idéntica en funcionamiento y uso a `implode()`. Usted puede usar la que desee. Es lo que se conoce como un *Alias*.

El convertir una cadena a una matriz y viceversa puede parecer una operación fútil, pero viene muy bien para localizar y, en su caso, sustituir determinados fragmentos por otros y, si bien las funciones que hemos conocido en este apartado no son el único modo de hacer localizaciones y/o sustituciones, no está de más su conocimiento, ya que las encontrará en varios scripts.

## 6.5 ENCRIPCIÓN DE CADENAS

En ocasiones se encontrará usted con una situación muy típica. Tiene que enviar un valor de una página a otra. En el capítulo anterior le expliqué que puede hacerlo mediante el uso de campos ocultos en un formulario. Sin embargo, los campos ocultos no están, realmente, demasiado ocultos. Evidentemente el usuario convencional no los ve. Fueron diseñados con el fin de pasar datos sin que resulten una molestia innecesaria a los ojos de quien usa nuestra página. Pero no han sido concebidos como un modo efectivo de esconder datos. Cualquiera puede abrir el código fuente de una página desde el navegador y ver los campos que constituyen el formulario y los valores que almacenan. En determinadas ocasiones no es conveniente que el usuario pueda ver estos datos. También nos encontraremos con casos en que determinada información deba almacenarse encriptada en archivos de disco o campos de una base de datos, a fin de que no esté al alcance de cualquiera que sienta curiosidad.

PHP nos proporciona la función *base64\_encode()* para encriptar una cadena. Esta función recibe como argumento una cadena y la devuelve encriptada. Si queremos mayor seguridad, podemos usarla combinada con *strrev()*, que invierte la posición de todos los caracteres de la cadena que recibe como argumento y genera una cadena inversa. Para recuperar una cadena encriptada con *base64\_encode()* contamos con la función *base64\_decode()*, que recibe como argumento la cadena encriptada y devuelve la cadena original, para poder seguir trabajando con ella. El uso de estas tres funciones está ilustrado en el script *cripta64.php*:

```
<?php
$cadenaOriginal="Esto es una cadena.";
echo ("La cadena original es: $cadenaOriginal<br
/>");

// Se encripta la cadena.
$cadenaCodificada=base64_encode($cadenaOriginal);
echo ("La cadena codificada es: $cadenaCodificada<br
/>");

//Se invierten los caracteres.
$cadenaInvertida=strrev($cadenaCodificada);
```

```

echo ("La cadena codificada revertida es:  

$cadenaInvertida<br />");
//Se vuelven a invertir.
$cadenaReinvertida=strrev($cadenaInvertida);
//Se descripta la cadena.
$cadenaRecuperada=base64_decode($cadenaReinvertida);
echo ("La cadena recuperada es: $cadenaRecuperada<br
/>");  

?>

```

Observe en las líneas resaltadas la sintaxis de estas funciones. En cada paso hemos ido mostrando los resultados que aparecen en la figura 6.21.

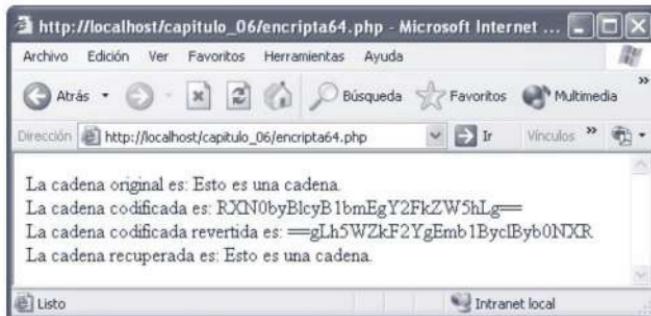


Figura 6.21

Use este sistema para el envío de cadenas en campos ocultos cuando no quiera que estén a la vista de cualquiera, y no se olvide de usar la función de desencriptado en el script receptor.

## 6.6 FUNCIONES NUMÉRICAS

PHP proporciona una buena librería de funciones para la gestión de valores numéricos. En este apartado vamos a conocer las más importantes por su utilidad. Quizás una de las capacidades matemáticas que más llamativa resulte a quienes estudian un nuevo lenguaje es la de generar números aleatorios. PHP resuelve esto muy bien, mediante la función ***rand ()***. En su uso más básico no recibe ningún argumento y genera un número aleatorio entero. El uso más básico responde a la siguiente sintaxis:

```
$aleatorio = rand ();
```

El número aleatorio está comprendido entre 0 y ***MAX\_RAND***, que es un parámetro propio de PHP que determina el valor máximo que podrá tener un número aleatorio. Para obtener el valor de MAX\_RAND usaremos la función ***getmaxrand()***, sin ningún argumento. También puede darse el caso de que queramos que el número aleatorio a obtener se encuentre entre dos límites, *x* e *y*. En ese caso le pasaremos los límites como argumentos a la función rand. La sintaxis sería la siguiente:

```
$aleatorio = rand ($inferior, $superior);
```

Observe el script **aleatorio.php**:

```
<?php
    echo ("Valor de MAX_RAND: ".getrandmax()."<br"
/>");
    $aleatorio=rand();
    echo ("Número aleatorio sin parámetros:
$aleatorio<br />");
    $aleatorio=rand(0,10);
    echo ("Número aleatorio entre 0 y 10:
$aleatorio<br />");
?>
```

Observe en las líneas resaltadas los dos usos posibles de rand(). Este script podría dar un resultado como el siguiente:

```
Valor de MAX_RAND: 32767
Número aleatorio sin parámetros: 11667
Número aleatorio entre 0 y 10: 8
```

Otra función muy empleada es ***sqrt()***, cuyo uso es obtener la raíz cuadrada del valor que recibe como argumento. La sintaxis general es la siguiente:

```
$raiz=sqrt ($valor);
```

Para el cálculo trigonométrico, PHP cuenta con varias funciones. La función ***sin()*** recibe un argumento que representa a un ángulo y devuelve el seno del mismo. La función ***cos()*** recibe un ángulo y devuelve su coseno. La función ***tan()*** recibe un ángulo y devuelve su tangente. La función ***asin()*** recibe un ángulo y devuelve su arco seno. La función ***acos()*** recibe un ángulo y devuelve su arco coseno. Por último, la función ***atan()*** recibe un ángulo y devuelve su arco tangente. Estas funciones son similares a las de otros lenguajes, pero los ángulos que reciben como argumento deben ir expresados en radianes. A tal fin, PHP cuenta con la función ***deg2rad()***, que recibe un ángulo expresado en grados y lo

devuelve en radianes. Naturalmente, también contamos con la función inversa, *rad2deg()*, que recibe un ángulo expresado en radianes y lo devuelve en grados. Observe el script **trigonometricas.php**:

```
<?php
$anguloRectoEnGrados=45;
$anguloRectoEnRadianes=deg2rad($anguloRectoEnGrados);
$seno=sin($anguloRectoEnRadianes);
$coseno=cos($anguloRectoEnRadianes);
$tangente=tan($anguloRectoEnRadianes);
$arcoSeno=asin($anguloRectoEnRadianes);
$arcoCoseno=acos($anguloRectoEnRadianes);
$arcoTangente=atan($anguloRectoEnRadianes);

?>
<html>
<body>

<table width="300" border="1" cellpadding="2">
<tr>
    <th colspan="2">FUNCIONES TRIGONOMÉTRICAS</th>
</tr>

<tr>
    <td colspan="2">
        &Aacute;ngulo en grados:
        <?php echo ($anguloRectoEnGrados); ?>
        <br />
        &Aacute;ngulo en radianes:
        <?php echo ($anguloRectoEnRadianes); ?>
        <br />
    </td>
</tr>
<tr>
    <th>FUNCIÓN</th>
    <th>VALOR</th>
</tr>
<tr>
    <td>Seno</td>
    <td><?php echo($seno); ?></td>
</tr>
<tr>
    <td>Coseno</td>
    <td><?php echo($coseno); ?></td>
</tr>
<tr>
    <td>Tangente</td>
    <td><?php echo($tangente); ?></td>
```

```
</tr>
<tr>
    <td>Arco seno</td>
    <td><?php echo($arcoSeno); ?></td>
</tr>
<tr>
    <td>Arco coseno</td>
    <td><?php echo($arcoCoseno); ?></td>
</tr>
<tr>
    <td>Arco tangente</td>
    <td><?php echo($arcoTangente); ?></td>
</tr>
</table>

</body>
</html>
```

Fíjese en el uso que hacemos de las funciones explicadas en la zona del código que aparece resaltada. El resto sólo es una tabla de HTML para mostrar los resultados, integrando las variables de PHP como ya sabemos hacerlo de ejercicios anteriores. El resultado final puede verse en la figura 6.22.

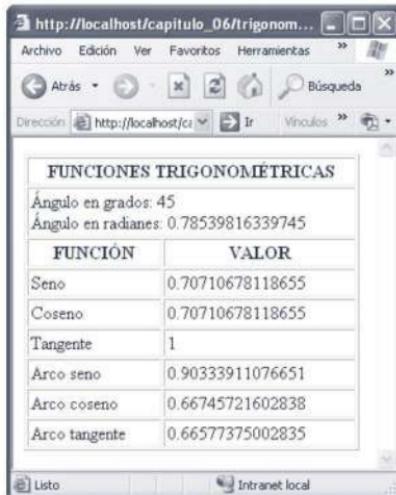


Figura 6.22

Otro grupo de funciones que nos interesa conocer son aquellas destinadas a manejar logaritmos. La función *log ()* devuelve el logaritmo natural del valor que recibe como argumento. La función *log10 ()* devuelve el logaritmo en base 10 del valor que recibe como argumento. La función *exp ()* recibe un valor como argumento y devuelve el número e (base de los logaritmos neperianos, cuyo valor es 2.718281828459) elevado a dicho argumento. Vea el script reproducido a continuación, llamado **logaritmos.php**:

```
<?php
    $valor=4;
    $logaritmo=log($valor);
    $logaritmo10=log10($valor);
    $neperiano=exp($valor);

    echo ("Partimos del valor: $valor<br />");
    echo ("El logaritmo natural es: $logaritmo<br
/>");
    echo ("El logaritmo en base 10 es:
$logaritmo10<br />");
    echo ("El número \"e\" elevado a $valor es:
$neperiano<br />");

?>
```

El resultado que obtenemos en la página es el siguiente:

```
Partimos del valor: 4
El logaritmo natural es: 1.3862943611199
El logaritmo en base 10 es: 0.60205999132796
El número "e" elevado a 4 es: 54.598150033144
```

Un aspecto importante de la gestión de valores numéricos es el uso de distintas bases de numeración. Las más importantes, dentro del área de la informática, aparte de la decimal (base 10), son la hexadecimal (base 16), la octal (base 8) y la binaria (base 2). En ocasiones es preciso convertir un número en una determinada base a otra, para un uso concreto. PHP nos proporciona algunas funciones útiles en este menester.

La función *bindec ()* recibe como argumento una cadena compuesta por ceros y unos que representa a un valor binario y devuelve el correspondiente valor en decimal.

La función *hexdec ()* recibe una cadena que representa un valor hexadecimal (compuesta, por tanto, por los dígitos del 0 al 9 y las letras de la A a la F). Devuelve el correspondiente valor decimal.

La función ***octdec ()*** recibe como argumento una cadena que representa un valor octal (compuesta por dígitos del 0 al 7) y devuelve el correspondiente valor decimal. Por supuesto, también contamos con las funciones inversas.

La función ***decbin ()*** recibe un número decimal y devuelve una cadena formada por ceros y unos que representa al argumento en binario.

También tenemos la función ***dechex ()***, que recibe un número decimal y devuelve una cadena que representa a dicho número en hexadecimal.

Por último, la función ***decoct ()*** recibe un número en decimal y devuelve una cadena que representa a dicho número en octal. Tenemos un ejemplo del uso de estas funciones en el script **bases.php**:

```
<?php

// Se establecen los valores iniciales
$valorDecimal=9058341;
$valorBinario="10100011101011";
$valorOctal="74521";
$valorHexa="FF56C1";

// Se convierte el decimal a otras bases
$decimalBinario=decbin($valorDecimal);
$decimalOctal=decoct($valorDecimal);
$decimalHexa=dechex($valorDecimal);

// Se convierte de otras bases a decimal
$binarioDecimal=bindec($valorBinario);
$octalDecimal=octdec($valorOctal);
$hexaDecimal=hexdec($valorHexa);

?>
<html>
<body>

VER TABLA COMPLETA EN EL ARCHIVO EN DISCO

</body>
</html>
```

Observe las líneas resaltadas en el código, en las que se ve el uso de las funciones que nos ocupan. La parte HTML es sólo una tabla para mostrar los resultados en la página y ha sido suprimida del listado impreso para facilitar la legibilidad. No obstante, está completa en el correspondiente archivo del disco. Al cargar este script en el navegador el resultado es el de la figura 6.23.

The screenshot shows a Microsoft Internet Explorer window with the title bar "http://localhost/capítulo\_04/bases.php - Microsoft Internet Explorer". The menu bar includes "Archivo", "Edición", "Ver", "Favoritos", "Herramientas", and "Ayuda". Below the menu is a toolbar with icons for Back, Forward, Stop, Refresh, Home, Favorites, Multimedia, and others. The address bar shows the URL "http://localhost/capítulo\_04/bases.php". The main content area displays a table titled "TABLA DE CONVERSIÓN". The table has four columns: "VALOR", "ESTA EN", "CONVERTIR A", and "RESULTADO". The data rows are:

VALOR	ESTA EN	CONVERTIR A	RESULTADO
9058341	DECIMAL	BINARIO	100010100011100000100101
		OCTAL	42434045
		HEXA	8a3825
10100011101011	BINARIO		10475
74521	OCTAL	DECIMAL	31057
FF56C1	HEXA		16733889

Figura 6.23

Quiero llamar su atención respecto a un punto concreto del uso de estas funciones: exceptuando los valores en decimal, que son numéricos, todos los valores en otras bases están representados mediante cadenas alfanuméricas. Así pues, un valor binario podrá ser "1101001101", no 1101001101. También podemos convertir un número directamente entre dos bases cualesquiera mediante el uso de la función *base\_convert()*. Esta función recibe tres argumentos separados mediante comas. El primero es una cadena alfanumérica que representa el valor original que queremos convertir, el segundo argumento es un número que indica la base en la que está dicho valor y el tercer parámetro es otro número, para indicar la base a la que queremos convertir dicho valor. Esta función puede manejar cualquier base entre 2 (binario) y 36. Por ejemplo, si quisieramos convertir el valor "A4D2FF" hexadecimal a binario usaríamos una sentencia similar a la siguiente:

```
$valorBinario = base_convert ("A4D2FF", 16, 2);
```

Uno de los valores más manejados cuando de geometría se trata es PI que, como usted sabe, representa la relación entre la longitud de una circunferencia y su diámetro, y es constante. Su valor aproximado es 3.1415926535898. PHP nos proporciona una constante que contiene ese valor, para que no tengamos que teclearlo cuando lo necesitemos. Se trata de *M\_PI*. También podemos obtener este valor mediante el uso de la función *pi()*, que no requiere de argumentos. Observe el script *valorPi.php*:

```
<?php
    echo ("El valor de pi es ".M_PI."<br />");
    echo ("Otra vez. El valor de pi es ".pi()."<br
/>");?
?
```

Como ve, es un script muy sencillito, para ver el valor de PI de dos maneras diferentes. El resultado será el siguiente:

```
El valor de pi es 3.1415926535898
Otra vez. El valor de pi es 3.1415926535898
```

A menudo nos encontramos con la necesidad de redondear un número fraccionario. PHP nos ofrece tres funciones que cumplen esta finalidad, cada una de una manera diferente. En primer lugar, contamos con la función *ceil()*, que recibe un número fraccionario y lo redondea al entero inmediato superior. También contamos con *floor()*, que recibe como argumento un número fraccionario y lo redondea al inmediato entero inferior. Por último, contamos con *round()*. Esta función recibe un número fraccionario. Si la parte fraccionaria es inferior a 5, el redondeo se hace al inmediato entero inferior. Si la parte fraccionaria es 5 o mayor, el redondeo se hace al inmediato entero superior. Vea el script **redondeos.php** para comprobar el uso de estas funciones:

```
<?php

$fraccionario_1=3.35;
$fraccionario_2=3.75;
$usoDeCeil_1=ceil($fraccionario_1);
$usoDeCeil_2=ceil($fraccionario_2);
$usoDeFloor_1=floor($fraccionario_1);
$usoDeFloor_2=floor($fraccionario_2);
$usoDeRound_1=round($fraccionario_1);
$usoDeRound_2=round($fraccionario_2);

?>

<html>
  <body>

    VER TABLA COMPLETA EN EL ARCHIVO DE DISCO
    .....
    .....
    .....
    </body>
  </html>
```

Las líneas resaltadas son las que nos interesan, pues muestran el uso de las funciones de redondeo. La parte de HTML suprimida, como ya hicimos en un ejemplo anterior, sólo contiene una tabla para mostrar los resultados en la página. En el archivo en disco está el código completo. Observe el resultado en la figura 6.24.

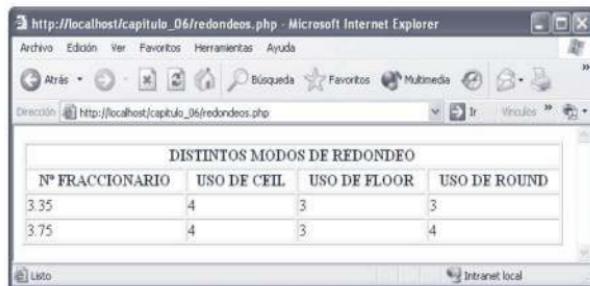


Figura 6.24

Compruebe cómo los resultados se ajustan a la descripción que hemos dado de las tres funciones.

PHP cuenta con dos funciones muy útiles para determinar cuál es el mayor y cuál el menor de un conjunto de valores. Las funciones ***max ()*** y ***min ()*** han sido implementadas a estos efectos. Pueden recibir un número ilimitado de argumentos, separados por comas. Por ejemplo, suponga el siguiente fragmento de código:

```
$valor_1=14;
$valor_2=20;
$valor_3=30;

$mayor=max ($valor_1, $valor_2, $valor_3);
$menor=min ($valor_1, $valor_2, $valor_3);
```

Esto hará que en la variable \$mayor se almacene el valor 30 y en la variable \$menor se almacene el valor 14.

Además, cualquiera de los argumentos puede ser una matriz que contenga, a su vez, varios valores. En ese caso, la función max () considerará en el lugar de ese argumento el mayor elemento de la matriz. La función min (), por el contrario, considerará ese argumento como el elemento de menor valor de la matriz.

En PHP contamos también con una función destinada a la potenciación de valores numéricos. Se trata de ***pow ()***, que recibe dos argumentos numéricos separados por una coma. El primero es la base y el segundo es el exponente al que queremos elevar la base. Así, por ejemplo, el número 84, al cubo, será:

```
$resultado = pow (84, 3);
```

Esta sentencia almacenará, en la variable \$resultado, el valor 592704, que se obtiene al realizar la operación aritmética indicada.

No quiero terminar este apartado sin tocar un tema que, a menudo, necesitará usted. Para dar un formato específico a valores numéricos al mostrarlos en la página podemos usar la función **number\_format()**. Esta función recibe cuatro argumentos, de los cuales sólo el primero es obligatorio. Los demás son opcionales. Éstos son:

- El valor que es necesario presentar formateado (o el nombre de la variable que contiene dicho valor).
- El número de cifras decimales que queremos mostrar (por defecto, ninguno).
- El carácter que vamos a emplear como separador de los decimales (por defecto, el punto).
- El carácter que vamos a emplear como separador de miles (por defecto, la coma).

Veamos un ejemplo sencillo en el script **formatearNumero.php**:

```
<?php
    $valor=123456789.987654321;
    /* A continuación se hace el redondeo en la misma
línea de la impresión. */
    echo (number_format($valor));
?>
```

El resultado que vemos en la página es:

123,456,790

Observe que, al no haber establecido ninguna posición decimal, se ha producido un efecto de redondeo. Como la parte fraccionaria es superior a 5 décimas, el susodicho redondeo se ha producido hacia arriba. Ahora hagamos una prueba, sustituyendo el separador de miles por un punto, poniendo un coma para separar la parte entera de la fraccionaria y estableciendo dos cifras decimales. Lo tiene en el script **formatearNumeroTotal.php**:

```
<?php
    $valor=123456789.987654321;
    echo (number_format($valor, 2, ",", "."));
?>
```

El resultado que vemos en la página es:

123.456.789,99

Observe el efecto de redondeo que se ha producido en los decimales. Al establecer sólo dos decimales, mientras que el valor original tiene más, se redondea.

## 6.7 FUNCIONES DE FECHA

Uno de los datos que más a menudo podemos llegar a manejar en una aplicación cliente-servidor es la fecha y hora del sistema. Piense en lo importante que puede llegar a ser este tipo de información, a fin de determinar cuándo se dio de alta un abonado en su sitio web, o la fecha de una compra si usted monta una tienda virtual, o cuándo fue la última visita de un usuario a fin de enviarle un e-mail con un saludo comercial, etc. En este apartado vamos a conocer algunas funciones destinadas a recuperar la fecha y hora del sistema y extraer determinadas partes de esa información. Por ejemplo, es probable que usted sólo necesite, para un proceso determinado, el día y el mes, pero no el año, la hora ni los minutos. En el momento de trabajar con la fecha y hora del sistema hay dos cosas que usted no debe olvidar: en primer lugar recuerde siempre que usted trabajará con la fecha y hora del servidor, no de su equipo local.

De momento, es la misma, porque está trabajando sobre una plataforma servidora montada en su propio equipo. Pero, en su momento, usted subirá su sitio a un servidor de acceso público para que esté disponible para todo el mundo. Ése es el objetivo de Internet. También puede ser que coloque las páginas de su sitio en un servidor privado para la red local de una empresa, por ejemplo, si desarrolla una aplicación cliente servidor específica para dicha organización. En todo caso los scripts de PHP que usted use se ejecutarán en el lado del servidor y, por lo tanto, cuando hagan uso de la fecha y hora del sistema, será la del servidor, no la del usuario que se conecta a su aplicación. Esto tiene una ventaja importante: da lo mismo que el ordenador cliente no tenga la fecha correcta. Cuando el script toma la fecha es la del servidor, por lo que, si ésta está correctamente ajustada, la aplicación funcionará con datos válidos. En cambio, si su aplicación necesita, por cualquier razón, la fecha y hora del ordenador cliente tendrá que obtener estos datos mediante JavaScript. En posteriores capítulos verá cómo transferir información entre estos dos lenguajes.

Otro detalle importante es que la fecha de un ordenador no es absoluta. Aunque usted la vea como tal en su entorno de trabajo, la fecha es lo que se conoce como *marca de tiempo* o *timestamp*. La timestamp es la diferencia, en segundos, entre el momento actual y lo que se conoce como comienzo de la era Unix: la hora 00:00:00 GMT (es decir, del meridiano 0 o de Greenwich) del día 1 de Enero de

1970. Es el momento cero para la informática, tal como la conocemos hoy. Como es lógico, la timestamp no nos sirve de gran cosa, por mucha importancia que, como efemérides, le demos a la hora cero. Sin embargo, PHP cuenta con funciones que nos pueden dar la fecha y hora reales.

En primer lugar, vamos a conocer la forma de recuperar la timestamp actual. Para ello, contamos con la función *time()*, cuyo uso vemos en el listado del script **marcaDeTiempo.php**:

```
<?php  
    $marca=time();  
    echo ($marca);  
?>
```

Cuando he ejecutado este script en mi navegador he obtenido lo siguiente:

1138924580

Como es lógico, cuando usted lo ejecute obtendrá un resultado mayor, ya que mayor será la timestamp.

Sin embargo, a nosotros nos interesa poder trabajar con las fechas y horas de una manera más práctica, lo que quiere decir más parecida al modo habitual de manejo de estos datos en la vida real. Para ello contamos con la función *getdate()*, que nos devuelve una matriz asociativa con todos los datos que podamos necesitar sobre la fecha y hora del sistema. Observe el script **datosDeFecha.php**:

```
<?php  
    $fecha=getdate();  
?>  
<html>  
    <body>  
        <table border="1" cellpadding="2">  
            <tr>  
                <th colspan="2">DATOS DE LA FECHA</th>  
            </tr>  
            <tr>  
                <th>CLAVE</th>  
                <th>VALOR</th>  
            </tr>  
        <?php  
            foreach ($fecha as $clave=>$valor){  
                echo ("<tr>\n");  
                echo ("<td>$clave</td>\n");  
                echo ("<td>$valor</td>\n");  
            }  
        </table>  
    </body>  
</html>
```

```

        echo "</tr>\n";
    }
?>
</table>
</body>
</html>

```

Cuando ejecute este script verá que obtiene una relación de pares clave-valor muy amplia, que contempla todos los datos que podamos necesitar obtener de una fecha. Incluso contamos con el día de la semana en nombre y en número, el número de días transcurridos del año actual, etc. De aquí podemos obtener cualquier dato que necesitemos acerca de la fecha y hora del servidor en el momento de la ejecución. Incluso contamos con una clave, de nombre "0", que contiene la marca de tiempo.

Observe la figura 6.25, donde se aprecia la lista de claves y valores que contiene la matriz obtenida mediante `getdate()`. Lógicamente sus datos, al ejecutar el script, serán los de la fecha y hora de su máquina servidora.



The screenshot shows a web browser window with the URL `http://localhost/c...` in the address bar. The page title is "DATOS DE LA FECHA". Below it is a table with two columns: "CLAVE" and "VALOR". The table contains the following data:

CLAVE	VALOR
<code>seconds</code>	38
<code>minutes</code>	45
<code>hours</code>	5
<code>mday</code>	22
<code>wday</code>	3
<code>mon</code>	2
<code>year</code>	2006
<code>yday</code>	52
<code>weekday</code>	Wednesday
<code>month</code>	February
0	1140583538

Figura 6.25

Usted puede obtener, mediante `getdate()` los datos de la fecha correspondiente a una determinada marca de tiempo, pasando dicha marca como argumento a la función.

Existe un modo más práctico de obtener la fecha del sistema en un momento dado. La función ***date()*** permite obtener la fecha y/o la hora en un formato que convenga a nuestros intereses. Para ello le tenemos que pasar, como argumento, una cadena con dicho formato expresado mediante unas claves predefinidas. Veamos un ejemplo de su uso, antes de entrar en detalles acerca del formato, en el script **fechaHoy.php**:

```
<?php
$fecha=date("d/m/Y");
echo ("La fecha de hoy es: $fecha<br />");
```

?>

Al ejecutar este script en el navegador obtenemos el resultado siguiente:

La fecha de hoy es: 03/02/2006

Podemos configurar la cadena de formato para obtener la fecha y/o la hora tal como convenga a nuestros intereses, según la tabla de la figura 6.26.

CLAVE	FORMATO
d	Día del mes, con dos dígitos, es decir, de “01” a “31”.
j	El día del mes, con un dígito o dos, según proceda (de “1” a “31”).
M	El nombre del mes en curso, abreviado con tres letras, a partir del nombre en inglés. Por ejemplo, en agosto, aparecerá “Aug”.
m	El mes en curso, en formato de “01” a “12”.
n	El mes en curso, en formato de “1” a “12”.
F	Nombre completo del mes, en inglés.
Y	El año en curso con cuatro cifras. Por ejemplo, “2006”.
y	El año en curso, representado con las dos últimas cifras. Por ejemplo, “06” si estamos en el año 2006.
D	Abreviatura con tres letras del nombre del día de la semana, en inglés. Si, por ejemplo, es viernes, aparecerá “Fri”.
I (l min)	El nombre del día de la semana completo, en inglés.
L	Un “0” si el año no es bisiesto y un “1” si lo es.
t	Número de días del mes en curso en formato de “28” a “31”.
z	Día del año. De “0” a “364” (de “0” a “365”, si es bisiesto).
w	Número del día de la semana. Tenga en cuenta que empieza por el domingo, con el valor “0” y acaba el sábado con el valor “6”.
h	La hora en formato de “01” a “12”.
H	La hora en formato de “00” a “23”.
g	La hora en formato de “1” a “12”.

CLAVE	FORMATO (cont.)
G	La hora en formato de “0” a “23”.
i	Los minutos en formato de “00” a “59”.
s	Los segundos en formato de “00” a “59”.
A	En la hora aparecerá la secuencia “A.M.” o “P.M”, según sea por la mañana o por la tarde.
a	En la hora aparecerá la secuencia “a.m.” o “p.m”, según sea por la mañana o por la tarde.
S	Cadena de dos caracteres con el sufijo ordinal en inglés. Por ejemplo, “st”, “nd”, “rd” o “th”.
Z	La diferencia, en segundos, con respecto a la hora GMT.
U	La marca de tiempo.

Figura 6.26

La lista de posibles formatos está ordenada empezando por los formatos relativos a fechas, siguiendo por los que se refieren a las horas y, para finalizar, tres formatos especiales (S, Z y U) que permiten extraer información adicional de la función date(). Además de los formatos especificados, usted puede añadir caracteres especiales para que aparezcan en el resultado de la función. Observe la línea resaltada en formatearNúmeroTotal.php y vea que contiene unas barras inclinadas para separar el día del mes, y éste del año. Fíjese en que esas barras aparecen en el resultado. Puede añadir otros caracteres, si lo necesita para su diseño. Por ejemplo, si quisiera obtener la hora usando el signo de dos puntos (:) para separar las horas de los minutos y éstos de los segundos, podría usar el siguiente formato: “H:i:s”.

También contamos con la función **gdate()**, que funciona de modo similar a date(), pero devolviendo la hora del meridiano de Greenwich (meridiano cero), es decir, con independencia de la hora que tenga el sistema servidor.

Estas funciones son bastante útiles y, si usted se dedica al desarrollo de aplicaciones web, las empleará mucho en su vida profesional. Sin embargo, adolecen de una pequeña limitación: las referencias a los nombres de los meses y de los días de la semana están en inglés. Si bien esto no debe suponer un problema para quien trabaja en programación, ya que este idioma es universal para la informática, nuestras aplicaciones web estarán diseñadas para usuarios que no necesariamente se tienen por qué sentir cómodos con el uso de terminología anglosajona. Lo que pretendemos es recuperar los nombres de meses y días de la semana en el idioma de los usuarios a los que vaya destinado potencialmente nuestro trabajo. Para ello contamos con dos funciones que debemos conocer. La

primera de ellas es ***setlocale*** 0, que permite establecer en el servidor la configuración relativa a un idioma determinado. Esta función recibe dos argumentos separados por una coma. El primero estipula respecto a qué factores queremos establecer una configuración local y el segundo determina respecto a qué idioma queremos establecer dicha configuración. El primer argumento es una constante de PHP, a elegir de entre las cuatro siguientes:

- **LC\_CTYPE** que se refiere a la conversión de cadenas.
- **LC\_NUMERIC** que se refiere a la conversión de valores numéricos.
- **LC\_TIME** que se refiere a la conversión de fechas y horas.
- **LC\_ALL** que se refiere a todas las anteriores (en general, use ésta siempre).

El segundo parámetro es una cadena que representa el idioma que queremos para la configuración local de nuestro servidor. Por ejemplo, pondremos “SP” para español, “EN” para inglés, “FR” para francés, “GE” para alemán, “IT” para italiano, etc. En general, pondremos las dos primeras letras del nombre en inglés del idioma que queremos. Para fijar la configuración en español usaremos:

```
setlocale (LC_ALL, "SP");
```

CLAVE	FORMATO
%a	Abreviatura del nombre del dia de la semana en el idioma establecido.
%A	Nombre completo del día de la semana en el idioma establecido.
%b	Abreviatura del nombre del mes en el idioma establecido.
%B	Nombre del mes en el idioma establecido.
%c	Fecha y hora presentadas según las convenciones del idioma establecido.
%d	Día del mes en formato de “01” a “31”.
%H	Hora en formato de “00” a “23”.
%I	Hora en formato de “01” a “12”.
%j	Día del año en formato de “001” a “366”.
%m	Mes en formato de “01” a “12”.
%M	Minutos en formato de “00” a “59”.
%p	Las cadenas “a.m.” o “p.m.” (o sus equivalentes en el idioma establecido).
%S	Segundos, de “00” a “59”.
%U	Número de la semana del año. Tenga en cuenta que la primera semana del año en curso se cuenta a partir del primer domingo.
%w	Día de la semana en número, 0 para el domingo, 6 para el sábado.

CLAVE	FORMATO (Cont.)
%x	La fecha, sin la hora.
%X	La hora, sin la fecha.
%y	Los dos últimos dígitos del año en curso.
%Y	El año en curso, con cuatro dígitos.
%z	Nombre o abreviatura de la zona horaria.

*Figura 6.27*

Una vez establecida la configuración local usaremos otra función para obtener la fecha y/u hora del sistema. Se trata de *strftime()*, que recibe un argumento con el formato en que queremos recuperar los datos necesarios. La mecánica es similar a la función *date()*, pero las cadenas relativas a los formatos son diferentes. Aparecen en la tabla de la figura 6.27.

Como ve, todos los posibles formatos van precedidos por el signo %. Veamos un ejemplo del uso de estas dos funciones en el script **ajusteLocal.php**:

```
<?php
    setlocale(LC_ALL, "SP");
    echo (strftime("%A, %x"));
?>
```

Vea cómo se ha aplicado la sintaxis de *setlocale()* para establecer el sistema en español y cómo se hace uso de *strftime()* para obtener la fecha con el nombre del día de la semana. El resultado en el navegador es el siguiente:

**miércoles, 08/02/2006**

Vea que se incluye en el resultado la coma y el espacio de separación que hemos puesto en nuestra cadena de formato. Además de obtener la fecha y hora del servidor también podemos “crear” una fecha y hora concreta para determinados usos. Por ejemplo, a la hora de establecer la caducidad de las cookies (ya hablaremos de ellas más adelante). Para ello utilizaremos la función *mkttime()*. Ésta recibe seis argumentos separados por comas. Son seis valores numéricos que representan la hora, los minutos, los segundos, el mes, el día y el año de una fecha concreta, y nos devuelve la marca de tiempo que corresponde al momento establecido. Vea un ejemplo de su uso en el script **marcaDeFecha.php**:

```
<?php
$marcaDeTiempo=mkttime (20,0,0,10,24,2010);
```

```
echo ("La marca de tiempo para las 20:00:00 horas  
del día 24 de Octubre de 2.020 es $marcaDeTiempo");  
?>
```

El resultado en la página es el siguiente:

```
La marca de tiempo para las 20:00:00 horas del día 24  
de Octubre de 2.020 es 1287943200
```

Observe en la línea resaltada la sintaxis correcta de esta función.

Para terminar este capítulo vamos a conocer la función *checkdate ()*, que se emplea para determinar si una fecha es correcta o no. Esto viene al caso cuando la fecha la introduce un usuario en un formulario. Imagine que el usuario, por error, introduce 30 de febrero. Lo normal es que, a partir de que nuestro script empiece a trabajar con esta fecha, se produzcan errores o la aplicación no devuelva los resultados esperados. Veamos cómo esta función puede ayudarnos a evitar problemas. Le pasaremos tres parámetros numéricos, que representarán el mes, el día y el año de la fecha que queremos comprobar. Si es correcta, la función nos devolverá un valor lógico true. Si no lo es, nos devolverá un false. Veamos un ejemplo de uso en el script **fechaCorrecta.php**:

```
<?php  
    $dia=30;  
    $mes=2;  
    $anio=2003;  
    echo ("La fecha $dia/$mes/$anio ");  
    if (checkdate($mes, $dia, $anio)){  
        echo ("es correcta.<br />");  
    } else {  
        echo ("NO es correcta.<br />");  
    }  
    $dia=31;  
    $mes=3;  
    $anio=2006;  
    echo ("La fecha $dia/$mes/$anio ");  
    if (checkdate($mes, $dia, $anio)){  
        echo ("es correcta.<br />");  
    } else {  
        echo ("NO es correcta.<br />");  
    }  
?>
```

Observe, en ambos ejemplos, el uso de la función *checkdate ()*. El resultado que vemos en la página es el siguiente:

**La fecha 30/2/2003 NO es correcta.  
La fecha 31/3/2006 es correcta.**

Esta función será de mucha utilidad cuando use las fechas que pueda introducir el usuario para trabajar con bases de datos, hacer cálculos de edades, etc.

## CAPÍTULO 7

### EXPRESIONES REGULARES

---

---

Las expresiones regulares constituyen una de las herramientas más flexibles y potentes a la hora de procesar cadenas alfanuméricas en cualquier lenguaje de programación moderno. Tanto es así que, a pesar de haber hablado en el capítulo anterior del tratamiento de cadenas, he reservado a este tema un capítulo propio, no tanto por la extensión del tema en si, como por su trascendencia.

A lo largo de su vida profesional usted desarrollará proyectos que requieran la entrada de datos del usuario a través de campos de texto. En ellos el usuario escribe "libremente" lo que desea. Y es normal que, en ocasiones, cometiera errores. Todos nos equivocamos de tecla alguna vez al mecanografiar.

El uso de expresiones regulares permite filtrar, hasta cierto punto, algunos errores que podrían causar unos resultados incorrectos o un mal funcionamiento de nuestra aplicación. A este efecto me vienen a la memoria las palabras del que fue mi primer profesor de informática: "Es posible diseñar aplicaciones a prueba de errores, pero es imposible diseñar una aplicación a prueba de usuarios". Bromas aparte, suponga que en su sitio web el usuario debe teclear una dirección de correo electrónico donde, por ejemplo, se le mandarán las respuestas a sus consultas, la información que solicite en un momento dado, etc. Y suponga que, por un descuido, el usuario no teclea el signo de la arroba que separa el nombre del dominio. Cuando la aplicación intente enviarle un correo electrónico (y ya veremos cómo se hace eso), se producirá un error.

El uso de expresiones regulares nos permitirá detectar que hay un error en la dirección de e-mail, y remitir al usuario a una página que le informe de dicho error y le dé opción a introducir correctamente su dirección.

A lo largo de este capítulo veremos qué son, exactamente, las expresiones regulares, cómo funcionan y cómo nos pueden ayudar a mejorar la eficiencia de nuestra programación.

## 7.1 QUÉ SON LAS EXPRESIONES REGULARES

Las expresiones regulares son, básicamente, patrones con los que comparar una cadena, obteniendo un resultado lógico de tipo true si la cadena se ajusta al patrón y un false en caso contrario. Así pues, podremos crear un patrón que contenga la arquitectura típica de una dirección de correo electrónico y después comparar las cadenas que introduzcan los usuarios con dicho patrón. Cuando el resultado de esa comparación sea true, la ejecución continuará en la siguiente página del sitio. Si el resultado es false, la ejecución saltará a una página en la que se le pida al usuario que vuelva a introducir el dato.

Los patrones se construyen a base de lo que se conoce como *metacaracteres*, que son unos caracteres que, dentro de este entorno, tienen un significado específico. Además, se pueden incluir en las expresiones regulares caracteres que sólo se representan a sí mismos.

Las expresiones regulares en PHP se pueden usar siguiendo dos estándares que el lenguaje acepta: el estándar *Posix* y el estándar *Perl*. En este capítulo veremos los dos.

## 7.2 INTRODUCCIÓN A LOS PATRONES

En esta sección vamos a ver cómo se crearán los patrones que luego se usarán para filtrar cadenas. Los conceptos que vamos a ver ahora son generales de las expresiones regulares y la mayoría de ellos se ajustarán a los dos estándares (Posix y Perl). Más adelante entraremos en detalles de cada uno de ellos.

Los patrones que se usan para construir expresiones regulares se pueden considerar clasificados en cuatro categorías: *de secuencia o fijación, multiplicadores, de alternativa y paréntesis*.

### 7.2.1 Patrones de secuencia o fijación

El patrón más simple está compuesto por un solo carácter. Por ejemplo “a”. Con esta expresión coincidirán todas las cadenas que contengan la letra a (minúscula). Así pues, coincidirán “Sonia”, “Eva” y “Jaime” pero no coincidirán “Pedro”, “coche” ni “globo”, por ejemplo. Éste sería un ejemplo de una expresión

regular formada por un carácter que se representa a sí mismo. La expresión más simple, usando un metacarácter es “.” (el *punto*), que representa a un carácter cualquiera. Por supuesto, un patrón así sería demasiado ambiguo: con él coincidiría cualquier cadena que tenga, al menos, un carácter cualquiera, es decir, que no sea una cadena vacía.

A partir de este concepto ya podemos pensar en expresiones regulares más complejas. Por ejemplo “a.t”. Con esta expresión coincidirán todas las cadenas que contengan la letra a, a continuación un carácter cualquiera y la letra t. Por ejemplo, “masticar”, “aptitud” y “antiguo”. Como ve, estas palabras cumplen la condición especificada.

Los metacaracteres “[” y “]” permiten referirse a un *rango* de caracteres que definamos para buscar coincidencias. Por ejemplo, supongamos el patrón “[abcde]”. Con él coincidirá cualquier cadena que contenga alguna de las letras especificadas, como “Coco”, “sábado” o “Septiembre”. No coincidirán “tifus”, “sino” ni “divino”.

Cuando los caracteres que forman parte de un rango tienen valores ASCII consecutivos se puede expresar el rango mediante el primer carácter (el de código ASCII más bajo), seguido de un guion y el último carácter (el de código ASCII más alto). El ejemplo del patrón anterior quedaría así: “[a-e]”. Si tiene alguna duda acerca del código ASCII, puede mirarlo en el Apéndice E.

En ocasiones queremos usar las expresiones regulares para identificar cadenas que no contengan caracteres de un determinado rango. Para ello usamos el metacarácter “^”, anteponiéndolo al primer carácter del rango. Suponga que quiere identificar cadenas que NO contengan las letras de la “a” a la “e” (minúsculas). El patrón adecuado será “[^a-e]”.

A menudo debemos crear una expresión destinada a identificar un carácter al principio o al final de una cadena. Si creamos un patrón como “^F”, coincidirán con él todas las cadenas que empiecen con la F (mayúscula). No confunda este uso del metacarácter ^ con el de negar un rango. Vea la diferencia de sintaxis. Para determinar que una cadena debe acabar con un carácter concreto pondremos el metacarácter \$ a continuación de aquel que queremos establecer como final. Si fijamos el patrón “a\$”, coincidirán con él todas las cadenas que terminen con la a minúscula.

## 7.2.2 Patrones multiplicadores

Son aquellos que se usan para repetir un carácter en la expresión regular. Suponga que quiere una expresión tal que coincidan con ella todas las cadenas que

tengan tres letras “a” (minúscula) seguidas. Podría crear un patrón como “aaa”, pero es más adecuado “a{3}”. Al expresar entre llaves un número a continuación de un carácter indicamos que, para coincidir con la expresión regular, la cadena debe incluir ese carácter repetido tantas veces como indica el número (por supuesto, seguidas). Con el patrón especificado en el ejemplo coincidirán las cadenas “lkjfsaalkd\$”, 094r0aaaaa0f04w3i” y “lsrfkjpoigkjeaaa”. Fíjese en que da igual si hay más de los tres caracteres (como es el caso de la segunda cadena): sólo se busca la existencia de, al menos, tres.

Para comprobar si un carácter se repite entre n y m veces estableceremos los dos límites, entre llaves, a continuación de dicho carácter. Suponga el patrón “a{2,5}”. Con él concordarán todas las cadenas en las que se encuentre el carácter “a” entre dos y cinco veces seguidas. Si aparece menos de dos veces (una o ninguna) o más de cinco, la cadena no coincidirá con la expresión.

Existe otro multiplicador, en el que se pone el límite inferior de veces que queremos que aparezca el carácter en la cadena, a fin de determinar si concuerda o no. Este límite se sigue de una coma, sin límite superior, así: “a{2,}”. Coincidirán todas las cadenas que tengan, al menos, dos veces seguidas la “a” (minúscula; en este ejemplo no se contarian las mayúsculas).

El multiplicador “\*\*” a continuación de un carácter determina que, para que la cadena coincida con la expresión, dicho carácter debe repetirse cualquier número de veces (incluso ninguna). Si se pone a pensarla, verá que este multiplicador carece de uso práctico.

El multiplicador “+”, a continuación de un carácter determina que, para que haya coincidencia, dicho carácter debe estar presente, al menos, una vez en la cadena.

El multiplicador “?” a continuación de un carácter indica que, para que la cadena coincida, dicho carácter deberá estar presente una vez, o ninguna. No habrá coincidencia si aparece más de una vez.

Cuando veamos la construcción de expresiones regulares a partir de estos patrones elementales, verá que formar una expresión con sólo un multiplicador, sin especificar más caracteres, hace que muchas veces los multiplicadores tengan efectos curiosos. Por ejemplo, suponga el caso de una expresión como la siguiente: “z{2,4}”. En teoría no deben coincidir las cadenas que tengan más de una “z”. Por ejemplo, la cadena “zzzzzz” no debería coincidir. Y sin embargo, coincide. Esto es debido a que PHP no examina la cadena globalmente, sino secuencialmente. Así pues, se encuentra una coincidencia con la primera secuencia de “zz”, con lo que el lenguaje ya da la expresión por coincidente.

### 7.2.3 Patrones de alternativa

Suponga una expresión como la siguiente: “**a|b|c**”. El carácter que separa la primera letra de la segunda, y ésta de la tercera, es la barra que se emplea en los operadores OR lógicos (usados en condicionales), aunque en este caso sólo una vez (**|**) y no dos (**||**). Con este patrón coinciden todas las cadenas que incluyan la letra “**a**”, o bien la “**b**”, o la “**c**”. Como es lógico, la cadena también coincidirá si incluye dos de las tres, o las tres.

Si se pone a pensarlo, verá que, en el ejemplo concreto que he puesto aquí, también sería equivalente un rango como el siguiente: “[abc]”, o bien “[a-c]”. Sin embargo, mediante el metacarácter de alternativa, al contrario de lo que sucede con los rangos, los caracteres no tienen por qué ser consecutivos según el código ASCII. Así pues, podemos crear una expresión como “**x|j**”, con la que coincidirán las cadenas que incluyan la “**x**” o la “**j**” (o ambas).

### 7.2.4 Los paréntesis

Los paréntesis tienen una triple finalidad. Por una parte, se emplean para agrupar patrones simples, como los que hemos visto en los ejemplos anteriores, formando patrones más complejos y, por lo tanto, más selectivos a la hora de que una cadena coincida con ellos. Por otra parte, nos permitirán memorizar (ya veremos cómo hacerlo, más adelante) partes de una cadena que coincidan con la expresión. Además, tienen un uso adicional que veremos en el apartado 7.2.6.

El uso de paréntesis permite, como hemos dicho, agrupar patrones simples. Por ejemplo, suponga que quiere crear un multiplicador que busque en la cadena a comparar la secuencia “**j2**” tres veces seguidas. Podría crear el patrón “**j2j2j2**”, pero mejor solución es la siguiente: “[**j2**]{3}”

### 7.2.5 Escapado de metacaracteres

Seguramente a estas alturas usted se habrá preguntado cómo hacer que una expresión regular busque en la cadena, por ejemplo, un punto. Es decir, si usted crea el patrón “**.**”, se dará la coincidencia con cualquier carácter, ya que el punto no es un carácter que se represente a sí mismo, sino un metacarácter. Para ello recurrimos al escapado de metacaracteres, mediante un contraslash.

En el patrón “**\.**”, el punto se ha convertido en un carácter que se representa a sí mismo, con lo que, para que haya coincidencia en la cadena deberá aparecer un punto. Si usted quiere escapar más de un metacarácter debe hacerlo con cada uno, por ejemplo: “**\.\{\\\l\}**”.

## 7.2.6 Precedencia de patrones

Como ya he comentado recientemente, los patrones se pueden agrupar para formar expresiones regulares complejas que permitan hacer una comprobación selectiva de cadenas. Suponga un patrón como el siguiente: “j | x\$”. Hay que tener cuidado al establecer un patrón así. Según lo leemos, puede significar que buscamos una cadena que acabe en “x” o en “j”. O puede significar que buscamos una cadena que contenga una “j” o que acabe en “x”. ¿Cuál es la interpretación correcta? Tal como está planteado el patrón, la segunda. Existe un orden establecido entre los metacaracteres, conocido como precedencia. El metacarácter de fijación “\$” tiene mayor precedencia que el de alternativa (“|”). Por esta razón se ejecuta antes. Y, ¿cómo podemos hacer que la expresión tenga la primera interpretación? Aquí es donde entran en juego de nuevo los paréntesis. Nos permiten alterar la precedencia de los metacaracteres. En este ejemplo escribiremos un patrón como el siguiente: “(j | x) \$”. Esto hará que coincidan con él todas las cadenas que acaben en “j” o en “x”. La precedencia natural de los metacaracteres es la siguiente:

- Paréntesis.
- Multiplicadores.
- Secuencia o fijación.
- Alternativa.

Para ver cómo podemos combinar varios patrones para formar una expresión regular, vamos a crear una que permita identificar si una cadena es correcta como dirección de correo electrónico. La solución es la siguiente:

```
^( [a-z] | [0-9] ) + ( \. ? ( [a-z] | [0-9] ) + ) ? @ ( [a-z] | [0-9] ) {2,} ( \. [a-z] {2,3} ) ( \. [a-z] {2} ) ? $
```

Como ve, la expresión resultante es bastante más compleja de lo que uno puede imaginar al principio, pero si la analizamos veremos que es un uso de los patrones que ya conocemos. En primer lugar, fíjese en que hay un signo @, obligatorio en toda dirección de correo electrónico. Debe aparecer uno, y sólo uno, separando el nombre del usuario del nombre del servidor.

Empecemos por analizar la parte de la izquierda, destinada al nombre del usuario. Vemos que la cadena que concuerde con esta expresión debe comenzar con, al menos, una letra o dígito, tal como indica el patrón “^( [a-z] | [0-9] ) +”. A continuación, puede haber un carácter punto (“\.”) y una o más letras o dígitos. Esta última parte puede darse una vez o ninguna, tal como indica el signo (“?”) que aparece al final. Esto es porque los nombres de usuario de las direcciones de correo electrónico pueden, como usted sabe, en determinados servidores, estar

formadas por dos partes, separadas entre sí por un punto. En este caso, no hemos tenido en cuenta el hecho de que el nombre de usuario de una dirección válida puede incluir el guión bajo, para simplificar un poco el ejemplo, ya que es el primero que vemos tan, digamos, complejo.

Después de la comprobación del signo @, encontramos la parte de la expresión destinada a comprobar el nombre del servidor. El nombre puede ser tan sencillo como "servidor.es" o "servidor.com" o puede estar formado por tres partes, tal como "servidor.com.ar" o "servidor.co.uk". En cualquier caso, debe haber una primera parte del nombre del servidor que contenga, al menos, dos letras o dígitos. Es el patrón "( [a-z] | [0-9] ) {2, }" el encargado de comprobar que así sea. A continuación, puede haber un punto y dos o tres caracteres. El patrón "( \. [a-z] {2,3} )" comprueba que esta primera terminación sea correcta. Opcionalmente puede haber otra terminación formada por un punto y dos letras, tal como nos dice el patrón "( \. [a-z] {2} )". Vea que esta última terminación puede darse una vez, o ninguna, según especifica el signo "?". Por último, está el "\$", encargado de establecer que la cadena debe terminar ahí. No puede haber nada más a continuación.

Una observación final. La cadena con una dirección de correo sólo concuerda con la expresión regular si todas sus letras son minúsculas. En este caso eso no es un problema, ya que así es como se escriben las direcciones de e-mail. Más adelante veremos cómo evitar esta limitación si es necesario.

### 7.3 COMPROBACIÓN DE CADENAS

Ya es hora de que usted pueda comprobar personalmente qué cadenas concuerdan con qué expresiones regulares y cuáles no lo hacen. A tal fin, usaremos el listado **compruebaExpresion.htm**, en la carpeta de este capítulo. No tiene nada especial. Es sólo un formulario en el que usted introduce una expresión regular y una cadena que quiere cotejar con la expresión. Cuando envíe el formulario, el script **compruebaExpresion.php** que lo recibe comprobará si la cadena concuerda con la expresión o no. De momento no voy a reproducir aquí los listados, ya que aún no hemos explicado la mecánica de las funciones destinadas a comprobar si una cadena concuerda con una expresión regular o no. Por ahora sólo veamos cómo opera. Al cargar **compruebaExpresion.htm** en el navegador usted verá la página de la figura 7.1.

En el campo **Expresión regular** introduzca la expresión que hemos visto anteriormente para la comprobación de direcciones de correo electrónico. En el campo **cadena** introduzca su propia dirección de e-mail y pulse **COMPROBAR**. El

formulario será enviado al script correspondiente, que le mostrará una página como la que se ve en la figura 7.2.



Figura 7.1

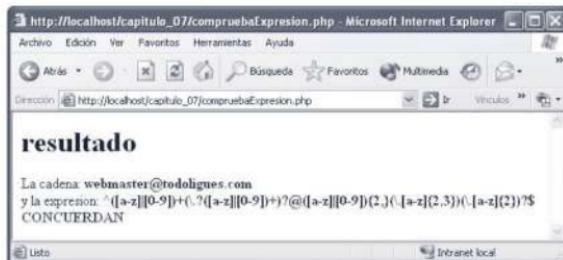


Figura 7.2

Como le he dicho, por ahora no intente “bucear” en el listado. De momento sólo sepa que cuenta con esta herramienta para comprobar cadenas y expresiones regulares.

## 7.4 EL ESTÁNDAR POSIX

El estándar Posix, que implementa PHP, añade nuevos patrones a los que ya conocemos para formar expresiones regulares. También cuenta con las funciones necesarias para comprobar si una cadena coincide con una expresión regular, extraer las coincidencias a una matriz y sustituir partes de una cadena. En esta sección nos ocuparemos de todos estos asuntos.

### 7.4.1 Patrones de Posix

El estándar Posix añade los patrones para la construcción de expresiones regulares que aparecen en la tabla de la figura 7.3. Si se para a analizar los patrones que aparecen verá que, por sí solos, no pueden constituir la base necesaria para la construcción de expresiones regulares complejas y flexibles, pero combinados con los caracteres que se representan a sí mismos, y con los metacaracteres que ya conocemos, obtenemos una mayor flexibilidad y comodidad. Por ejemplo, suponga un patrón como el siguiente: “[a-z] | [A-Z] | [0-9]”. En su lugar podemos escribir “[**:alnum:**]”. El funcionamiento será el mismo. De igual modo, el estándar Posix nos simplifica otras anotaciones. Por ejemplo, suponga que quiere crear un patrón que coincida con signos de puntuación empleados en el español. Posix le proporciona un patrón para identificar signos de puntuación del inglés. Sólo tenemos que añadir aquéllos que son propios del español y lo tendremos resuelto. Será lo siguiente: “[**:punct:**] | ; | ,”. Como ve, he unido nuestros propios signos de puntuación a los del inglés con el metacarácter de la alternativa.

PATRONES POSIX PARA EXPRESIONES REGULARES	
PATRÓN	COINCIDE CON
[ <b>:alnum:</b> ]	Cualquier dígito del 0 al 9 o cualquier letra mayúscula o minúscula, excepto las acentuadas, la “Ñ”, la “ñ”, la “Ç” y la “ç”.
[ <b>:alpha:</b> ]	Cualquier letra, mayúscula o minúscula, excepto las acentuadas, la “Ñ”, la “ñ”, la “Ç” y la “ç”.
[ <b>:blank:</b> ]	El espacio en blanco (“ ”) o la tabulación (“\t”).
[ <b>:cntrl:</b> ]	Cualquier carácter cuyo código ASCII sea inferior a 32.
[ <b>:digit:</b> ]	Cualquier dígito del 0 al 9.
[ <b>:lower:</b> ]	Cualquier letra minúscula, excepto las acentuadas, la “ñ” y la “ç”.
[ <b>:space:</b> ]	El espacio en blanco (“ ”), la tabulación (“\t”), el retorno de carro (“\r”) y el salto de línea (“\n”).
[ <b>:upper:</b> ]	Cualquier letra mayúscula, excepto las acentuadas, la “Ñ” y la “Ç”.
[ <b>:punct:</b> ]	Cualquier carácter de puntuación del inglés.
[ <b>:print:</b> ]	Cualquier carácter imprimible.
[ <b>:xdigit:</b> ]	Cualquier carácter de la base de numeración hexadecimal.
[ <b>:&lt;:</b> ]	Cadena vacía al principio de una palabra.
[ <b>:&gt;:</b> ]	Cadena vacía al final de una palabra.

Figura 7.3

Y, ¿qué ocurre si queremos un patrón que busque cualquier letra, mayúscula o minúscula, acentuada o no, del alfabeto español? Pues, una vez más

combinaremos los patrones propios de Posix con los metacaracteres que ya conocemos, así: “[**:alpha:**] | [ÁÉÍÓÚÑÇÁÉÍÓÚÑç]”.

Quiero llamar su atención sobre los dos últimos patrones de la tabla: “[[:<:]]” y “[[:>:]]”. Éstos determinan que se producirá coincidencia si hay un espacio en blanco, un signo de puntuación o nada al principio o al final, respectivamente, de una palabra, pero no si hay una letra o dígito. La mejor forma de comprender esto es comprobar unos ejemplos. Suponga el patrón “[[:<:]]hacer”. Coincidirán con él las cadenas “Quiero hacer”, “hacerte” o “tu”, pero no, por ejemplo, la cadena “deshacer”.

Por lo demás, en el estándar Posix son válidas, también, las reglas de precedencia de metacaracteres.

#### 7.4.2 Gestión de expresiones regulares Posix

El uso principal de las expresiones regulares es poder cotejar una cadena para determinar si coincide o no con una determinada expresión. Para ello, contamos con la función *ereg()* que, en su uso más básico, recibe dos argumentos. La expresión regular, en primer lugar, y la cadena a comprobar en segundo. Si la cadena concuerda con la expresión, la función devuelve el valor lógico *true*. En caso contrario, devuelve *false*. El uso básico coincidirá con el siguiente esquema:

```
$concuerda = ereg ($expresión, $cadena);
if ($concuerda) {
    echo ("La cadena concuerda con la expresión.");
} else {
    echo ("La cadena NO concuerda con la
expresión.");
}
```

Además, podemos añadirle a esta función, como tercer parámetro opcional, el nombre de una matriz, en la que se guardarán aquellos fragmentos de la cadena que concuerden con la expresión. Cada vez que uno de estos fragmentos es encontrado, se almacena en un nuevo elemento de la matriz, que luego podemos usar para recuperarlos de forma aislada.

El uso de esta capacidad de la función *ereg()* corresponde al siguiente esquema:

```
$concuerda = ereg ($expresión, $cadena, $matriz);
if ($concuerda) {
    echo ("La cadena concuerda con la expresión.");
```

```
    } else {
        echo ("La cadena NO concuerda con la
expresión.");
    }
    echo ("<br />");

$elementos=count($matriz);
for ($contador=0; $contador<$elementos; $contador++) {
    echo ("El elemento $contador contiene ");
    echo ($matriz[$contador]."<br>");
}
```

Atención a una cosa. Si queremos que un fragmento de la cadena que coincide con determinado patrón de la expresión regular quede almacenado en la matriz es necesario que el patrón esté rodeado de paréntesis. Por ejemplo, si tenemos la cadena “cadena” y el patrón “de”, se producirá una coincidencia, pero no se almacenará el fragmento coincidente en la matriz. En cambio, sí se almacenará si el patrón es “(de)”.

De todo esto veremos un ejemplo claro en esta misma sección. Pero antes permítame hacerle una puntualización. Esta función busca las coincidencias de la cadena con la expresión regular teniendo en cuenta el uso de mayúsculas y minúsculas, de modo que, por ejemplo, la cadena “A” NO coincide con el patrón “a”. Si es necesario que no se tenga en cuenta la capitalización de las letras, deberemos usar la función *ereg\_i()*, cuya operativa es la misma que la de *ereg()*, pero ignorando si las letras son mayúsculas o minúsculas. Así pues, con el uso de esta última función, el patrón “a” coincidirá con la cadena “a” y también con la cadena “A”. Si alguna vez no le coincide una cadena que usted piensa que debería coincidir, tenga esto en cuenta.

Una función especialmente útil es *ereg\_replace()*. Recibe tres parámetros, separados por comas. El primero es un patrón o expresión regular; el segundo, una cadena de sustitución y el tercero, la cadena que vamos a confrontar con el patrón. Lo que hace esta función es buscar coincidencias en la cadena a examinar. Cuando encuentra un fragmento de la cadena que coincide con el patrón, lo reemplaza por la cadena de sustitución. Suponga el siguiente fragmento de código:

```
$patron="def";
$sustitución="zzz";
$cadena="abcdefghijkl";
$nCadena=ereg_replace ($patron, $sustitución, $cadena);
echo ($nCadena);
```

Nos mostrará en la página el valor de \$nCadena, así:

abczzzghi

Una vez más, esta función es sensible a mayúsculas y minúsculas, con lo que si el patrón es, siguiendo el ejemplo que acabamos de poner, “def” y en la cadena aparece la secuencia “Def”, o bien “DeF”, o cualquier otra combinación que implemente mayúsculas, no se dará como coincidente con el patrón. Y, una vez más, PHP nos proporciona una “solución honorable”: la función *ereg\_replace()*, que actúa como la anterior, pero sin tener en cuenta la capitalización de las letras.

Y ahora que ya conocemos el uso de las funciones destinadas a trabajar con expresiones regulares, volveremos sobre la aplicación que usamos para confrontar una expresión con una cadena. Sin embargo, vamos a trabajar con una versión más completa que la anterior. Cargue en su navegador el listado **compruebaPosix.htm**. Éste no lo vamos a analizar, ya que no tiene nada de particular. Sólo es un formulario que obtiene una expresión regular y una cadena y las envía al script que aparece en el script **compruebaPosix.php**, cuyo código es el siguiente:

```
<html>
<body>
<h1><center>Resultado</center></h1>
<?php
    define("salto", "<br>\n");
    if (!isset($_HTTP_POST_VARS["cadena"]) || 
!isset($_HTTP_POST_VARS["expresion"])){
        echo "La cadena y la expresión regular deben tener
contenido." . salto;
        die ();
    }
    // Se eliminan los "\" introducidos al recuperar las
variables.
    $_HTTP_POST_VARS["expresion"] =
ereg_replace("\\\\\\\\\\\\", "\\", $_HTTP_POST_VARS["expresion"]);
    $_HTTP_POST_VARS["cadena"] = ereg_replace("\\\\\\\\\\\\",
"\\", $_HTTP_POST_VARS["cadena"]);
    // Se comprueba si la cadena coincide con la expresión
regular, almacenando,
    //además, las coincidencias en una matriz, siempre que
los patrones coincidentes
    //estén entre paréntesis, tal como se explica en el
texto.
    $correcto = ereg($_HTTP_POST_VARS["expresion"],
$_HTTP_POST_VARS["cadena"], $matriz);
    //Se muestran la expresión y la cadena
    echo ("La cadena:
<b>$HTTP_POST_VARS[cadena]</b>" . salto);
```

```
echo ("y la expresion:  
<b>$HTTP_POST_VARS[expresion]</b>".salto);  
    //Se verifica si ha habido coincidencias.  
    if ($correcto) {  
        // Si las ha habido, se muestra la concordancia y los  
        elementos almacenados en la matriz.  
        echo ("<font  
color=blue><b>CONCUERDAN</b></font>".salto);  
        $elementos=count($matriz);  
        for ($contador=1; $contador<$elementos;  
$contador++){  
            echo "Elemento $contador:  
<b>$matriz[$contador]</b>" . salto;  
        }  
    } else {  
        //Si no ha habido concordancia, se avisa de ello  
        echo ("<font color=red><b>NO  
CONCUERDAN</b></font>".salto);  
    }  
?  
</body>  
</html>
```

He resaltado las líneas sobre las que quiero llamar su atención. En primer lugar, hay dos usos consecutivos de la función `ereg_replace()` destinados a eliminar los contraslash “sobrantes” de las variables que almacenan la expresión regular y la cadena a cotejar. Éstos son signos de escape que se añaden durante la recuperación del contenido de una variable cuando su contenido ya incluye un contraslash.

Por otro lado, he resaltado el uso de la función `ereg()`, con los tres parámetros, a fin de que vea cómo se construye la matriz. Una observación al respecto. Cuanto más estricta y compleja resulte la expresión regular (sobre todo en el uso de los multiplicadores), menos fragmentos de la cadena se recuperarán para la matriz. Sin embargo, una expresión regular demasiado “flexible” podría dar lugar a que coincidan cadenas que realmente no deseamos. ¿Cuál es la solución adecuada? Depende de cada caso, claro; de lo que pretendamos hacer con el resultado de `ereg()` o `eregi()`.

## 7.5 EL ESTÁNDAR PERL

Se trata de la alternativa a Posix. Se llama así por haber sido implementado, inicialmente, en el lenguaje del mismo nombre, aunque hoy es plenamente soportado por PHP. Al igual que hemos hecho con Posix en la sección

anterior, en ésta veremos las particularidades de los patrones de este estándar, así como las funciones de PHP destinadas a gestionarlo.

### 7.5.1 Los patrones de Perl

Lo primero que debemos saber es que una expresión regular según el estándar Perl debe ir acotada por limitadores. Éstos son dos caracteres idénticos, no alfanuméricos, que se ponen al principio y al final de la expresión regular. Normalmente, se suele emplear el slash (""). Por ejemplo, la siguiente sería una expresión Perl válida:

```
/[a-f]/
```

Sin embargo, si nuestra expresión regular incluye uno o más slashes en su interior deberemos utilizar otro delimitador. Suponga una expresión como la siguiente:

```
# [0-9] / [a-z] #
```

En este caso, hemos usado la almohadilla o “cucarachita” como delimitador, lo que es perfectamente válido. Otra alternativa, si queremos seguir usando los slashes como delimitadores, es escapar los que aparecen dentro de la expresión. Así, el siguiente patrón es equivalente al anterior:

```
/ [0-9] \ / [a-z] /
```

Además de los metacaracteres estándar que conocimos en la sección 7.2, el estándar Perl implementa su propia relación de metacaracteres, aunque no es tan amplia como la del estándar Posix. La relación aparece en la tabla representada en la figura 7.4.

PATRONES PERL PARA EXPRESIONES REGULARES	
PATRÓN	COINCIDE CON
\b	Un límite de palabra o de cadena. Si precede a un carácter, éste deberá ser el primero de una palabra o de la cadena para que se dé la coincidencia. Si se pone después de un carácter, éste deberá ser el último de la palabra o de la cadena para que se dé la coincidencia.
\B	Cualquier carácter que no sea principio o final de palabra o cadena.
\d	Cualquier dígito del 0 al 9 (numeración decimal).
\D	Ningún dígito decimal.
\w	Cualquier letra, mayúscula o minúscula, acentuada o no, incluyendo la “Ñ”, la “ñ”, la “C”, la “ç” y el guion bajo (“ ”).

PATRONES PERL PARA EXPRESIONES REGULARES (Cont.)	
PATRÓN	COINCIDE CON
\w	Ninguna letra, ni el guión bajo.
\s	Cualquier carácter de espaciación: el espacio en blanco (" "), la tabulación ("t"), la nueva línea ("n") o el retorno de carro ("r").
\S	Ningún carácter de espaciación.

Figura 7.4

Las expresiones regulares en Perl pueden cambiar su comportamiento añadiéndole determinados modificadores después del delimitador de cierre (el que va al final de la expresión regular). Éstos aparecen en la tabla de la figura 7.5.

MODIFICADORES PARA LAS EXPRESIONES REGULARES DE PERL	
Modif.	Significa que
<b>A</b>	Equivale al metacarácter de fijación "^". Es decir, indica que es el principio de la cadena lo que tiene que coincidir con la expresión regular.
<b>i</b>	Evita que se hagan distinciones entre mayúsculas y minúsculas. Evidentemente, este modificador no tiene sentido si usamos el metacarácter "\w" de Perl. Sólo tiene razón de usarse si usamos caracteres que se representan a sí mismos o metacaracteres estándar.
<b>m</b>	Se usa para tratar cadenas formadas por más de una línea, de forma que los metacaracteres "\$" y "^" se aplicarán a cada línea.
<b>s</b>	Hace que el metacarácter punto (".") coincida también con un salto de línea, como si éste fuera un carácter más.
<b>U</b>	Altera el comportamiento de los modificadores de multiplicación. Normalmente éstos "consumen" el máximo posible de la cadena. Es decir, si tenemos una expresión regular como "/x+/", cuya coincidencia se produce cuando el carácter "x" se encuentra en la cadena una o más veces, y la confrontamos con la cadena "jjjxxxxzzz", la coincidencia se produce con toda la secuencia de las cuatro "x". Si usamos este modificador, la coincidencia sólo se produce con una "x". Es útil tener esto en cuenta si usamos paréntesis para poder almacenar las coincidencias en una matriz. Si usted tiene una expresión como "/(x+)/" y la confronta con la cadena de este ejemplo, en la matriz se almacenará "xxxx". Sin embargo, si su expresión es "/(x+)/U" y la confronta, en la matriz se almacenará "x".
<b>x</b>	Permite añadir el símbolo # y un comentario al final de la expresión regular.

MODIFICADORES PARA EXPRESIONES REGULARES PERL (Cont.)	
Modif.	Significa que
S	Hace que PHP optimice la evaluación de la expresión regular para que la comparación se efectúe más rápido.
X	Causa un error cuando en la expresión aparece el contraslash de escapados ("\\") seguido de un carácter que se representa a sí mismo. Es decir, el contraslash sólo puede ir seguido de un metacarácter que necesitemos escapar.

Figura 7.5

Las expresiones regulares de Perl contemplan una funcionalidad adicional conocida como **condicionales**. Esto quiere decir que se buscará la coincidencia con un patrón de la expresión supeditada a la coincidencia de otro anterior o posterior. Los condicionales aparecen en la tabla de la figura 7.6. Esto flexibiliza mucho las posibilidades de estas expresiones regulares.

CONDICIONALES DE LAS EXPRESIONES REGULARES PERL	
CONDICIONAL	SIGNIFICADO
Patrón1(?=patrón2)	La coincidencia con “patrón1” sólo se produce si a continuación encuentra una coincidencia con “patrón2”.
Patrón1(?!=patrón2)	La coincidencia con “patrón1” sólo se produce si a continuación NO encuentra una coincidencia con “patrón2”.
(?<=patrón1)patrón2	La coincidencia con “patrón2” sólo se produce si va precedida de una coincidencia con “patrón1”.
(?<!patrón1)patrón2	La coincidencia con “patrón2” sólo se produce si NO va precedida de una coincidencia con “patrón1”.

Figura 7.6

## 7.5.2 Gestión de expresiones regulares Perl

PHP implementa las funciones necesarias para la adecuada confrontación de cadenas alfanuméricas con expresiones regulares según el estándar Perl. La primera que vamos a conocer es `preg_match()`. La operativa de esta función es equivalente a la de `ereg()`, que vimos en la sección 7.4.2. Al igual que ésta, recibe tres argumentos separados por comas. El primero es la expresión regular, el

segundo es la cadena que se va a comparar y el tercero es el nombre de la matriz donde almacenaremos las coincidencias si procede. Este último parámetro es opcional.

En este caso, no existe una función específica para ejecutar comparaciones ignorando las mayúsculas y las minúsculas ya que, por la propia naturaleza del estándar Perl, es en la propia expresión regular donde se indica si queremos ignorar la capitalización de las letras. A tal efecto, repase la tabla de la figura 7.5; concretamente, el modificador i.

La función *preg\_replace()* actúa de modo equivalente a *ereg\_replace()*. Recibe tres argumentos. El primero es la expresión regular, el segundo es una cadena de reemplazo y el tercero es la cadena a comparar con la expresión. Donde se produzca una coincidencia, el fragmento coincidente de la cadena será sustituido por la cadena de reemplazo.

Para comprobar la operatividad del estándar Perl utilice el listado **compruebaPerl.htm**. Como en los casos anteriores, este listado no tiene nada de especial. Sólo es un formulario donde se obtiene una expresión regular y una cadena y se envían a un script que está en **compruebaPerl.php**, cuyo código es el siguiente:

```
<html>
<body>
<h1><center>Resultado</center></h1>
<?php
    define("salto", "<br>\n");
    if (!isset($_HTTP_POST_VARS["cadena"]) || 
!isset($_HTTP_POST_VARS["expresion"])) {
        echo "La cadena y la expresión regular deben tener
contenido." . salto;
        die ();
    }
    // Se eliminan los "\" introducidos al recuperar las
variables.
    $_HTTP_POST_VARS["expresion"] =
ereg_replace("\\\\\\\\\\\\", "\\", $_HTTP_POST_VARS["expresion"]);
    $_HTTP_POST_VARS["cadena"] = ereg_replace("\\\\\\\\\\\\",
"\\", $_HTTP_POST_VARS["cadena"]);
    // Se comprueba si la cadena coincide con la expresión
regular, almacenando,
    //además, las coincidencias en una matriz, siempre que
los patrones coincidentes
    //estén entre paréntesis, tal como se explica en el
texto.
```

```

$correcto = preg_match($HTTP_POST_VARS["expresion"],
$HTTP_POST_VARS["cadena"], $matriz);
//Se muestran la expresión y la cadena
echo ("La cadena:
<b>$HTTP_POST_VARS[cadena]</b>".salto);
echo ("y la expresion:
<b>$HTTP_POST_VARS[expresion]</b>".salto);
//Se verifica si ha habido coincidencias.
if ($correcto) {
    // Si las ha habido, se muestra la concordancia y los
    elementos almacenados en la matriz.
    echo "<font
color=blue><b>CONCUERDAN</b></font>".salto);
    $elementos=count($matriz);
    for ($contador=1; $contador<$elementos;
$contador++){
        echo "Elemento $contador:
<b>$matriz[$contador]</b>" . salto;
    }
} else {
    //Si no ha habido concordancia, se avisa de ello
    echo ("<font color=red><b>NO
CONCUERDAN</b></font>".salto);
}
?>
</body>
</html>

```

Observe, en la linea resaltada, cómo se confronta una cadena con una expresión regular Perl. Pruebe esta herramienta de comparación de cadenas. A fin de que se familiarice un poco con el estándar Perl, le muestro algunas expresiones regulares que puede usar, en la tabla de la figura 7.7.

EXPRESIONES REGULARES PERL DE EJEMPLO	
EXPRESIÓN	USO
/(\d{6,8})-([A-Z])/	Comprobación del NIF (el documento español de identidad) cuyo formato es un número de entre 6 y 8 dígitos, seguido de un guión y una letra mayúscula.
/(\d{1,2})\/(\d{1,2})\/(\d{4})/	Fecha en formato dd/mm/aaaa.

Figura 7.7

## CAPÍTULO 8

# FICHEROS

---

---

En este capítulo vamos a conocer las posibilidades que PHP incorpora para la gestión de ficheros. Este tema lo vamos a considerar en dos grandes líneas. Por un lado, vamos a tratar los propios scripts de PHP como ficheros (lo que, de hecho, son). Desde este punto de vista veremos cómo un script de PHP puede “llamar” a otro si es necesario. Por otro lado, veremos cómo un script de PHP puede almacenar, recuperar y/o eliminar cierta información general en archivos de texto plano que se manejarán en el lado del servidor.

### 8.1 EJECUTANDO OTROS SCRIPTS

Cuando se escribe mucho código nos encontramos con que, a menudo, determinadas funciones o fragmentos del código están presentes, sin cambios, en distintos scripts. Por ejemplo, suponga que va a montar una tienda virtual. El acceso a la base de datos de los productos deberá estar incluido en, prácticamente, la totalidad de las páginas. Cuando se presenta una situación así, aparece un concepto conocido como *reutilización de código*. En realidad este concepto es mucho más amplio, ya que implica la posibilidad de que el código que hoy escribimos para, por ejemplo, crear una tienda, nos pueda servir, con muy pocos cambios, para que, dentro de dos meses, podamos crear otra diferente. Pero, por lo que a este capítulo respecta, vamos a ver cómo podemos usar un fragmento de código en varios scripts diferentes, sin necesidad de re-escribirlo cada vez. La solución pasa por colocar el fragmento que será común en otro script, al que invocaremos cuando sea necesario. Para ello, PHP nos proporciona cuatro instrucciones que debemos conocer: *include ()*, *require ()*, *include\_once ()* y *require\_once ()*. Aunque el objetivo de las cuatro es similar, presentan algunas

diferencias que debemos conocer, para aplicar la más adecuada en cada caso. A lo largo de las siguientes páginas, vamos a describir el proceso de incorporar un script dentro de otro con todas las posibilidades.

Vamos a suponer un script que contiene la definición de una función para el cálculo de amortización de un préstamo. Esta función recibirá tres argumentos, que serán el capital del préstamo, el tipo de interés aplicable y el plazo de amortización, en meses. A partir de ahí calculará el importe de las cuotas (considerando un interés simple) que se deben pagar y devolverá este valor. Es un script, que he llamado **calculointereses.php**, muy sencillo, cuyo código es el que aparece reproducido a continuación:

```
<?php
// Se define una función que calcula la cuota mensual
// para unos valores de capital, interes, y plazo.
function calculaCuota ($capital, $interes,
$meses) {
    $capital+=($capital*$interes)/100;
    $cuota=$capital/$meses;
    // Se formatea la cuota para una salida "presentable".
    // (ver capítulo 6 si hay dudas).
    $cuotaFormato=sprintf("%.2f", $cuota);
    // La función devuelve la cuota mensual.
    return ($cuotaFormato);
}
?>
```

Ahora suponga que necesita disponer de esta función en un script para que esté disponible para el usuario. No tiene que copiar el código de la función en el script. Basta con que incluya una línea como la siguiente:

```
require ("calculointereses.php");
```

Veamos un ejemplo de uso. Para ello cargue en el navegador el código de **formularioIntereses.htm**. No voy a reproducir aquí el listado, puesto que sólo es un formulario HTML, sin mayor interés. El caso es que los datos que introduzca el usuario se envían a **calculointereses.php**, que es el script encargado de procesarlos. Su código es el siguiente:

```
<?php
define ("salto", "<br>\n");
//Se carga el script externo.
require ("calculointereses.php");
```

```
echo ("El capital solicitado es:  
".$HTTP_POST_VARS["importe"]." euros.".salto);  
echo ("El tipo de interés es:  
".$HTTP_POST_VARS["tipo"]." %.".salto);  
echo ("El plazo de amortización es de:  
".$HTTP_POST_VARS["plazo"]." meses.".salto.salto);  
$pagoMensual=calculaCuota($HTTP_POST_VARS["im-  
porte"], $HTTP_POST_VARS["tipo"],  
$HTTP_POST_VARS["plazo"]);  
echo ("El importe de las cuotas es de:  
$pagoMensual euros mensuales".salto);  
?>
```

Fíjese en la primera linea resaltada, que hace uso de require () para incorporar el código de calculoIntereses.php dentro de formularioIntereses.php. A partir de este momento, la función calculaCuota (), que se ha definido en el primer script, está disponible en el segundo, y podemos llamarla, tal como se hace en la segunda línea que aparece resaltada. El uso, por tanto, de require () está muy claro: cuando tengamos un fragmento de código que vaya a ser incorporado a varios scripts de nuestro sitio web, lo grabaremos en un script externo y lo llamaremos cuando sea necesario, en lugar de copiar dicho fragmento en todas las páginas. En este ejemplo, el cuerpo de la función calculaCuota, que está definida en un script externo, puede ser incorporado a cualquier script del sitio, con sólo hacer una llamada con require (), tal como hemos visto. Esto tiene una ventaja adicional, muy importante. Si usted necesita cambiar el código que aparece en el script externo, sólo tiene que cambiarlo en dicho script. A partir de ahí, cuando se le requiera se cargará el código modificado.

Hasta la versión 4 de PHP, la sentencia require () presentaba ciertas limitaciones en su funcionamiento con condicionales. Sin embargo, éstas han sido subsanadas en la versión 5.

Existe una sentencia equivalente que es **include ()**. Su uso y sintaxis son idénticos a los que acabamos de ver. Sin embargo, existe una diferencia de matiz en su operativa ante un error. Suponga que llamamos a un script externo que no existe (por ejemplo, por haber escrito mal el nombre). Veamos el comportamiento de ambas instrucciones ante esta eventualidad. Observe el script **errorNoInclude.php**, a continuación:

```
<?php  
include("ficheroNoExistente.php");  
echo ("EJECUCIÓN FINALIZADA");  
?>
```

Como ve en la línea resaltada se llama a un script que no existe. Al ejecutarlo veremos en la página el siguiente resultado:

```
Warning: include(ficheroNoExistente.php) [function.include]: failed to open stream: No such file or directory in C:\Documents and Settings\Jose\Mis documentos\Mis webs dinamicas\capítulo_08\errorNoInclude.php on line 2
Warning: include() [function.include]: Failed opening 'ficheroNoExistente.php' for inclusion (include_path='.;C:\php5\pear') in C:\Documents and Settings\Jose\Mis documentos\Mis webs dinamicas\capítulo_08\errorNoInclude.php on line 2
EJECUCIÓN FINALIZADA
```

Como puede comprobar, se lanzan un par de avisos (Warning) de que algo va mal, pero la ejecución continúa, mostrándose el contenido de la siguiente línea del script, que exhibe un texto en la página.

Ahora observe el listado de **errorNoRequire.php**, que es el siguiente:

```
<?php
    require("ficheroNoExistente.php");
    echo ("EJECUCIÓN FINALIZADA");
?>
```

Como ve, es el mismo que el anterior, sólo que la invocación a un script externo no existente se hace con require(), en lugar de con include(). Al ejecutarlo obtenemos el siguiente resultado en la página:

```
Warning: require(ficheroNoExistente.php) [function.require]: failed to open stream: No such file or directory in C:\Documents and Settings\Jose\Mis documentos\Mis webs dinamicas\capítulo_08\errorNoRequire.php on line 2
Fatal error: require() [function.require]: Failed opening required
'ficheroNoExistente.php' (include_path='.;C:\php5\pear') in C:\Documents and Settings\Jose\Mis documentos\Mis webs dinamicas\capítulo_08\errorNoRequire.php on line 2
```

Como ve, obtenemos un aviso (Warning) y un error fatal (Fatal error) que detiene la ejecución, no mostrándose en la página la siguiente línea del script.

¿Cuándo usar require() y cuándo usar include()? Es, principalmente, una cuestión de criterio personal. Yo me he acostumbrado a emplear include(), pero, como le digo, ése es mi criterio que no tiene, necesariamente, que coincidir con el suyo. En versiones anteriores de PHP tenía importancia con el empleo de funciones de usuario, pero eso es ya meramente anecdótico. En la actualidad, ambas funciones operan igual.

Con el uso de estas instrucciones puede surgir un problema si el script externo contiene la definición de una función y la invocación forma parte, por ejemplo, de un bucle. Suponga un script externo como **cuadradoExterno.php**:

```
<?php
    function calculaCuadrado ($numero) {
        $cuadrado=$numero*$numero;
        echo ("El cuadrado de $numero es
$cuadrado.".salto);
    }
?>
```

Como ve, es una función muy sencillita, que ni siquiera devuelve resultado alguno, y que sólo recibe como argumento un número, lo eleva al cuadrado y lo muestra en la página. Ahora vamos a crear un script que llame a esta función para calcular el cuadrado de los números del 1 al 10. El código se llama **mostrarCuadrados.php**:

```
<?php
    define ("salto","<br>\n");
    for ($contador=1; $contador<=10; $contador
++) {
        include ("cuadradoExterno.php");
        calculaCuadrado ($contador);
    }
?>
```

Cuando lo ejecute verá que se le muestra en la página el cuadrado del primer valor del bucle (1) y, a continuación, se produce el siguiente error:

**Fatal error:** Cannot redeclare calculacuadrado() (previously declared in C:\Documents and Settings\Jose\Mis documentos\Mis webs dinamicas\capitulo\_08\cuadradoExterno.php:2) in C:\Documents and Settings\Jose\Mis documentos\Mis webs dinamicas\capitulo\_08\cuadradoExterno.php on line 5

Este error se produce porque, al estar la invocación en un bucle, se llama al script externo en cada iteración de dicho bucle. Sin embargo, PHP no permite cargar en un script la definición de una función más de una vez. Se lo repito, resaltado en negrita, porque esto da lugar a muchos errores.

*En ningún script de PHP se puede definir una función más de una vez, bajo ningún concepto y de ninguna manera.*

La solución a este problema puede ser, en este caso, colocar la sentencia `include ()` antes del bucle. Sin embargo, si por el diseño de su script, no le queda más remedio que colocarla dentro del bucle, utilice, en su lugar, `include_once ()`. Observe el script **cuadradosCorrectos.php**:

```
<?php
    define ("salto","<br />\n");
    for ($contador=1; $contador<=10; $contador
++) {
        include_once ("cuadradoExterno.php");
        calculaCuadrado ($contador);
    }
?>
```

Observe la línea resaltada en el código. Esta función comprueba si ya se ha incluido previamente el script externo. Si es el caso, ya no efectúa la inclusión. De este modo, se evita el error de redeclarar una función.

PHP cuenta también con la correspondiente instrucción `require_once ()`, equivalente a `require ()`, pero con comprobación previa de haberse cargado ya el script externo a que se refiere.

Recuerde. Siempre que su script externo incluya la definición de una función, cárguelo con `include_once ()` o con `require_once ()`. De este modo, evitará errores. Sin embargo, esto tiene una contrapartida. Si su script externo declara variables que son usadas por el script principal, sólo se cargarán una vez. Si necesita recargarlas, con los valores originales, mediante otra llamada al script externo, con `require_once ()` e `include_once ()` no podrá hacerlo, ya que estas dos instrucciones se ejecutan, siempre, previa comprobación de no haber sido ejecutadas anteriormente en el mismo script.

### 8.1.1 Consideraciones sobre scripts externos

Es importante que tenga en cuenta un hecho. Hasta ahora todos los scripts externos que hemos cargado están en la misma ruta que aquéllos en los que se efectúa la invocación. Si los scripts externos se encuentran en otra carpeta o directorio es necesario especificar la ruta de acceso en la llamada. Por ejemplo. Suponga que, dentro de su carpeta principal de trabajo tiene dos carpetas. La primera, llamada **usoGeneral**, contiene todos los scripts de sus páginas; la segunda, llamada **archivosComunes**, contiene los scripts externos que deberán ser cargados desde los principales. Para invocar un script externo desde una de las páginas deberá usar algo parecido a lo siguiente:

```
include ("../archivosComunes/scriptExterno.php");
```

Esto, si lo piensa, no es nada nuevo. Ocurre lo mismo que cuando se usa un enlace en HTML o se llama a un archivo en cualquier lenguaje. Usted ya está lo suficientemente familiarizado con HTML y JavaScript como para que el asunto de las rutas no le suponga ningún problema.

Lo llamativo de este modo de incluir scripts externos es que usted puede invocar, incluso, a scripts que se encuentren en otro servidor, accediendo a ellos de forma remota. Por ejemplo, su código podría tener una línea como la siguiente:

```
include ("http://otrosScripts.com/scriptExterno.php");
```

Sin embargo, para que esto funcione es necesario que tenga activada la directiva `allow_url_fopen` en el fichero de configuración de PHP (`php.ini`). Vea el Apéndice A “La configuración del intérprete de PHP” para aprender lo necesario al respecto.

Naturalmente, todo lo que hemos dicho en este breve apartado es válido, también, para las sentencias `require()`, `include_once()` y `require_once()`, cuyo funcionamiento, como ha podido ver, es muy similar.

## 8.2 LOS FICHEROS

PHP permite la gestión de ficheros de texto plano, es decir, sin un formato específico, que se almacenan en el lado del servidor. Este tipo de ficheros pueden ser usados para almacenar ciertos datos acerca del tráfico en la red, o de los usuarios que se han conectado, etc. Además, permite la gestión de otros ficheros adicionales (sesiones, cookies del cliente, etc.) pero eso es tema para otros capítulos. Aquí nos vamos a ocupar de ficheros que usted puede grabar y/o leer en la forma que sea necesario. Cuando se trabaja con un fichero para almacenar o recuperar cierto tipo de información se llevan a cabo, de modo genérico, tres operaciones básicas: la apertura del fichero, la lectura o escritura en el mismo y su cierre. En ese sentido la gestión de ficheros desde PHP es similar al manejo de ficheros tradicionales de papel en un archivador de oficina de los de toda la vida. En este capítulo aprenderemos a realizar todas estas operaciones.

### 8.2.1 Abrir ficheros

Para abrir un fichero recurrimos a la función `fopen()`, que recibe dos argumentos. El primero es el nombre del fichero que es necesario abrir, con su ruta, si ésta no es la misma que aquella donde se encuentra grabado el script. El segundo

argumento es el modo en que queremos abrir el fichero. Éste dependerá de lo que vayamos a hacer con él mismo. Existen siete modos posibles para abrir un fichero, que aparecen recogidos en la tabla de la figura 8.1.

MODO DE APERTURA DE FICHEROS	
MODO	SIGNIFICADO
a	Abre el fichero para añadir datos al final del mismo. No permite la lectura.
a+	Igual que el anterior, pero permitiendo la lectura.
r	Abre el fichero sólo para lectura. No permite la escritura de datos.
r+	Abre el fichero para lectura, permitiendo la escritura de datos que se grabarán al principio del contenido actual.
w	Abre un fichero para escribir datos que sustituirán a los que ya pueda haber. No permite la lectura.
w+	Igual que el anterior, pero permitiendo la lectura.
b	Abre un fichero binario.

Figura 8.1

En los modos “a”, “a+”, “w” y “w+” se intenta abrir el fichero. Si no existe, se crea en ese momento. En los modos “r” y “r+” se intenta abrir el fichero. Si no existe, se produce un mensaje de error como el siguiente:

**Warning:** fopen(ficheroR.txt) [function.fopen]: failed to open stream: No such file or directory in C:\Documents and Settings\Jose\Mis documentos\Mis webs dinamicas\capítulo\_08\abrirModoRPlus.php on line 2

Este mensaje nos informa del intento de abrir un fichero que no existe. Fíjese en la parte *failed to open stream: No such file or directory*, que le informa de ello.

El modificador “b” se emplea, adicionalmente a alguno de los anteriores, para abrir ficheros binarios. Por ejemplo, un archivo que, en lugar de texto, contiene una imagen. Cuando se trabaja en una plataforma Unix (o Linux, Solares, etc.) no existe diferencia entre abrir un fichero de texto o uno binario. Windows, en cambio, sí reconoce esa diferencia. Tenga en cuenta que, aunque usted esté trabajando en su equipo en entorno Windows, la mayoría de los servidores profesionales corren bajo Linux. Las razones para esto son varias. En primer lugar, no hay que pagar una licencia. Además Linux es abierto, se puede modificar y

recompilar según las necesidades del servicio. Microsoft, en cambio, vende sus licencias de uso de Windows "tal como es". Si no cubre una necesidad específica, no se puede hacer nada. Cuando usted escriba su sitio para Internet, y una vez lo haya probado y le funcione correctamente, seguro que contratará el alojamiento en un servidor profesional. Si bien la mayoría de las compañías ofrecen plataformas Linux y Windows, estas últimas suelen ser más caras, ya que al coste propio del servicio hay que añadirle un prorrato por la licencia que se le paga a Microsoft. Ésto no debe preocuparle. Como el trabajo con sus scripts es llevado a cabo por el intérprete PHP, éstos funcionarán bien en cualquier servidor local o remoto.

Volviendo al tema de la apertura de ficheros, esta es, como le digo, la primera operación que debe llevar a cabo su script cuando quiera leer o escribir en un fichero. La sintaxis general de fopen () es la siguiente:

```
$manejador=fopen($nombreFichero, $modo);
```

Fíjese en que esta función devuelve un *identificador*, o *manejador* (como prefiera), que debe ser asignado a una variable. Este manejador será el que posteriormente se emplee para leer o escribir en el fichero, así como para cerrarlo.

Llegados a este punto quiero hacerle un inciso del que seguramente no oirá hablar mucho. Esta función permite abrir ficheros remotos, es decir, situados en otros servidores. Sin embargo, para ello es necesario que esté activada la directriz allow\_url\_fopen en el fichero de configuración. Vea el Apéndice A para saber cómo activar o desactivar esta directriz. Cuando quiera acceder a ficheros situados en servidores remotos, deberá contar con los permisos adecuados. Evidentemente, por razones de seguridad, uno no puede acceder por las buenas a un fichero que está en otro servidor. Esto es lógico. Si usted pone un fichero en su servidor no querrá que todo el mundo pueda leerlo y, lo que puede ser más grave, escribir en él a su antojo. Más adelante, en este mismo capítulo, hablaremos sobre el tema de los permisos.

## 8.2.2 Cerrar ficheros

Cuando hayamos llevado a cabo cualquier operación con un fichero, tanto de lectura como de escritura, es necesario cerrarlo. Para ello recurrimos a la función *fclose()*, que recibe, como argumento, el manejador que se creó al abrir el fichero. De este modo, podemos tener abiertos varios ficheros en un script, y cerrarlos según vayamos dejando de necesitarlos. Recuerde una cosa. Siempre que abra un fichero deberá cerrarlo cuando termine de trabajar con él. El hecho de dejar un fichero abierto puede acarrear problemas en posteriores accesos e, incluso, la pérdida de información. *NUNCA, bajo ningún concepto, olvide cerrar los ficheros*

que haya abierto. Quizás, si usted prueba a usar ficheros y dejarlos sin cerrar, no le ocurría nada en una ocasión, o en diez, o en cien mil, pero es un problema que está dejando al azar, y no se hacen planes para la suerte.

La sintaxis genérica de esta función es la siguiente:

```
fclose ($manejador);
```

El manejador es, como le he dicho antes, el que se creó al abrir el fichero.

### 8.2.3 Cómo leer en los ficheros

Usted ya sabe cómo abrir y cerrar un fichero. Ahora veamos qué podemos hacer desde que se abre hasta que se cierra. Para empezar vamos a considerar la lectura de un fichero. En la carpeta correspondiente a este capítulo tiene un fichero de texto plano, cuyo nombre es *lecturaSimple.txt*. En su interior sólo hay una frase grabada. Abralo con su editor de texto, para comprobar su contenido, pero no lo modifique en modo alguno. Para leer todo el contenido de un fichero, de modo indiscriminado, podemos usar la función *fpassthru ()*. Ésta recibe, como argumento, el manejador con el que se ha abierto el fichero, y vuelca en la salida estándar (la pantalla de su navegador) el contenido del mismo. Vea el script *lecturaSimpleR.php*, listado a continuación:

```
<?php  
    $manejadorFicherol=fopen  
("lecturaSimple.txt","r");  
    fpassthru ($manejadorFicherol);  
    fclose ($manejadorFicherol);  
?>
```

Cargue el script en el navegador para comprobar su resultado. En la página le aparece el contenido del fichero de texto. Observe que el script tiene tres líneas de código, que realizan las tres operaciones básicas que conocemos. La primera línea abre el fichero, la segunda lo lee y la tercera lo cierra. Quiero llamar su atención sobre algo que hemos comentado, pero que aquí puede ver claramente por primera vez. La primera línea abre el fichero en modo lectura ("r") y crea un manejador de fichero, al que hemos llamado \$manejadorFicherol. Este manejador es el que emplean la segunda y tercera linea para leer y cerrar el fichero respectivamente.

La función fpassthru () tiene un uso adicional. Si le asignamos la salida a una variable, nos devuelve el número de caracteres que contiene el fichero. Observe el script *listadoConCuentaR.php*, que aparece a continuación:

```
<?php
    // Se define el salto de línea.
    define ("salto","<br>\n");
    // Se abre el fichero para lectura.
    $manejadorFicherol=fopen
("lecturaSimple.txt","r");
    echo ("El contenido del fichero es: <b>");
    /* Se muestra el contenido del fichero y se
    asigna el total de caracteres a una variable. */
    $caracteres=fpassthru ($manejadorFicherol);
    echo ("</b>".salto."El total de caracteres es de:
<b>");
    // Se muestra el total de caracteres.
    echo ($caracteres."</b>".salto);
    fclose ($manejadorFicherol);
?>
```

El resultado que aparece en la página corresponde a la figura 8.2.

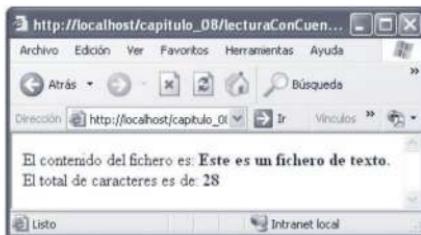


Figura 8.2

PHP nos ofrece otras funciones destinadas a la lectura de un fichero. La función *fread()* permite leer un número determinado de caracteres. Esta función recibe dos argumentos. El primero es, cómo no, el manejador del fichero. El segundo es el número de caracteres que deseamos leer. La función devuelve una parte del fichero, de tantos bytes de longitud como hayamos expresado en el segundo argumento. Este resultado de la función debe ser asignado a una variable para luego poder mostrarlo en la página, o procesarlo del modo que nos resulte conveniente.

Esta función trabaja con un *puntero* que se posiciona en el fichero, de modo que, al abrirlo, apunta al principio del mismo. Al ejecutarse la función, el puntero se “desplaza” tantos caracteres como se hayan leído. Así pues, si el segundo argumento de la función es, digamos, 10, el puntero quedará posicionado

en el carácter 11. Si se vuelve a ejecutar la función, se leerá a partir de ahí, no desde el comienzo, como la primera vez. También puede ocurrir que se llegue al final del fichero. Todos los ficheros tienen una marca llamada **EOF (End Of File)**, que señala esta circunstancia. Si la función `fread()` debe leer, digamos, 20 caracteres, pero encuentra antes la marca EOF, leerá hasta ese punto.

Esta función se complementa muy bien con **`rewind()`**, que recibe, como argumento, el manejador del fichero sobre el que queremos trabajar, y posiciona el puntero de lectura en el primer carácter.

Observe el código **leerConReadR.php**, listado a continuación:

```
<?php
// Se define el salto de línea.
define ("salto","<br>\n");
// Se abre el fichero para lectura.
$manejadorFicherol=fopen
("lecturaSimple.txt","r");
/* Se leen doce caracteres. El puntero de lectura queda
posicionado para leer a partir del carácter 13. */
$bloque=fread ($manejadorFicherol, 12);
echo ("Los doce primeros caracteres del fichero
son: <b>$bloque");

/* Se leen otros doce caracteres. El puntero de lectura
queda posicionado para leer a partir del carácter 25.
*/
$bloque=fread ($manejadorFicherol, 12);
echo ("</b>".salto."El siguiente bloque de doce
caracteres es: <b>$bloque");

/* Se intentará leer otros doce caracteres, pero se
alcanza la marca EOF, con lo que se detiene la lectura
antes.*/
$bloque=fread ($manejadorFicherol, 12);
echo ("</b>".salto."El siguiente bloque es:
<b>$bloque");

echo ("</b>".salto."Si queremos volver a leer hay
que \"rebobinar\".");
/* Se rebobina el puntero de lectura, que vuelve a
quedarse apuntando al primer carácter. */
rewind ($manejadorFicherol);
/* Se vuelve a leer el primer bloque de caracteres para
confirmar que el puntero ha quedado al principio. */
$bloque=fread ($manejadorFicherol, 12);
echo ("</b>".salto."Volvemos a leer el primer
bloque de doce caracteres: <b>$bloque");
echo ("</b>".salto);
```

```
// Se cierra el fichero
fclose ($manejadorFicherol);
?>
```

Al ejecutar el código verá el resultado de la figura 8.3.

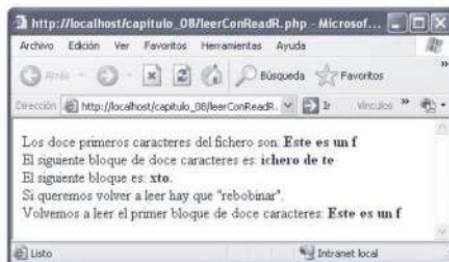


Figura 8.3

Observe en las líneas resaltadas cómo hemos hecho uso de las dos funciones: fread () y rewind () .

Podemos determinar el tamaño de un fichero mediante la función *filesize()*, que recibe como argumento el nombre del fichero cuyo tamaño queremos averiguar. Preste atención a esto. A esta función no se le pasa el manejador con el que abrimos el fichero, ya que es operativa, incluso, si no se ha abierto ningún fichero. Se le pasa el nombre (y, en su caso, la ruta) del fichero que nos interesa. Mediante el uso de esta función podríamos leer un fichero completo usando fread(). Vea el código del script leerConSizeR.php:

```
<?php
// Se define el salto de línea.
define ("salto","<br>\n");
// Se determina el tamaño del fichero.
$caracteres=filesize ("lecturaSimple.txt");
// Se abre el fichero para lectura.
$manejadorFicherol=fopen
("lecturaSimple.txt","r");
// Se lee todo el fichero de una vez.
$bloque=fread ($manejadorFicherol, $caracteres);
echo ("El contenido del fichero es:
<b>$bloque</b>" . $salto);
// Se muestra el tamaño del fichero.
```

```
echo ("El total de caracteres es de:  
<b>$caracteres</b>");  
    // Se cierra el fichero  
    fclose ($manejadorFicherol);  
?>
```

Vea, en las líneas resaltadas, cómo hemos usado la función `filesize()` para determinar el tamaño del fichero y leerlo en un solo bloque. El resultado final coincide con la figura 8.2.

En PHP contamos con otra función de lectura que permite recuperar el contenido de un fichero en bloques de un solo carácter; se trata de `fgetc()`, que recibe, como argumento, el manejador del fichero del cual queremos leer. Observe el código del script `leerCaracterR.php`, donde se ha usado esta función, en combinación con `filesize()` para leer todo el contenido de un fichero, carácter a carácter.

```
<?php  
// Se define el salto de línea.  
define ("salto","<br>\n");  
  
// Se determina el tamaño del fichero.  
$caracteres=filesize ("lecturaSimple.txt");  
  
// Se abre el fichero para lectura.  
$manejadorFicherol=fopen ("lecturaSimple.txt","r");  
  
// Se lee el fichero carácter a carácter  
$contenido="";  
for ($contador=1; $contador<=$caracteres;  
$contador++){  
    $contenido.=fgetc($manejadorFicherol);  
}  
  
// Se muestra todo el contenido.  
echo ("El contenido del fichero es:  
<b>$contenido</b>".salto);  
// Se muestra el tamaño del fichero.  
echo ("El total de caracteres es de:  
<b>$caracteres</b>");  
    // Se cierra el fichero  
    fclose ($manejadorFicherol);  
?>
```

Al ejecutar este código se encuentra con una página que se corresponde, también, con la figura 8.2. Observe, en el bloque resaltado, cómo hemos hecho uso

de un bucle, basado en el tamaño del fichero, para recuperar su contenido, carácter a carácter.

Cuando se lee un fichero, éste no tiene que tener, necesariamente, la extensión `.txt`, como hemos venido haciendo hasta ahora. Usted puede programar su script para que lea, por ejemplo, el listado de una página HTML, que también es un fichero de texto, pero que tiene la extensión `.htm`. En ese caso, se cargará la página leída en el navegador como si la hubiéramos llamado directamente. Puede ver un ejemplo de esto en `leerHTML.php`, cuyo listado aparece a continuación:

```
<?php  
    $manejadorFicherol=fopen ("ficheroHTML.htm","r");  
    fpassthru ($manejadorFicherol);  
    fclose ($manejadorFicherol);  
?>
```

Como ve, se trata de leer una página HTML. Ejecútelo y verá un resultado como el de la figura 8.4.



Figura 8.4

Si pulsa en `Ver` y, a continuación en `Código fuente` verá el listado de la página que hay cargada en el navegador, tal como muestra la figura 8.5.

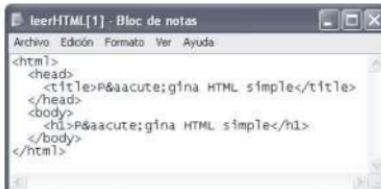


Figura 8.5

En PHP contamos con otra función de lectura de ficheros destinada a leer cadenas completas, llamada *fgets ()*. Ésta recibe dos argumentos separados por comas. El primero es, lógicamente, el manejador con el que se ha abierto el fichero. El segundo es el número de bytes que queremos leer, más uno. Así, si queremos leer 10 caracteres, como segundo parámetro pondremos 11. Dicho así, suena muy similar a *fread ()*. Sin embargo, hay algunas diferencias. El segundo argumento es opcional. Si no se incluye se considerará 1.024 por defecto. Además, si antes de leer los caracteres especificados aparece un salto de línea, la lectura termina en ese punto. Esta función trabaja muy bien en combinación con *feof ()*, que se emplea para detectar cuándo se ha llegado al final del fichero. Ésta recibe como argumento el manejador del fichero y devuelve un valor lógico, true o false, dependiendo de que se haya alcanzado o no el final. Puede ver el uso de estas dos funciones en el script **leerCadenas.php**:

```
<?php
// Se abre el fichero
$manejadorFicherol=fopen ("ficheroHTML.htm","r");
/* Se inicia un bucle que se ejecutará mientras
no se alcance el final del fichero.*/
while (!feof($manejadorFicherol)) {
/* Se lee una cadena, hasta el salto de línea.*/
$contenido=fgets ($manejadorFicherol);
echo ($contenido);
}
// Se cierra el fichero.
fclose ($manejadorFicherol);
?>
```

Al ejecutar este código se obtiene el resultado de la figura 8.4. Vea, en las líneas resaltadas, cómo hemos empleado las dos funciones que acabamos de conocer.

Existe otra función, similar a *fgets ()*, que es *fgetss ()*. Su sintaxis es la misma que la de la anterior, pero elimina TODAS las etiquetas de HTML, PHP y JavaScript. Observe el listado **leerSinEtiquetas.php**:

```
<?php
// Se abre el fichero
$manejadorFicherol=fopen ("ficheroHTML.htm","r");
/* Se inicia un bucle que se ejecutará mientras
no se alcance el final del fichero.*/
while (!feof($manejadorFicherol)) {
/* Se lee una cadena, hasta el salto de línea.
No se leen las etiquetas HTML. */
$contenido=fgetss ($manejadorFicherol);
```

```
        echo ($contenido);
    }
// Se cierra el fichero.
fclose ($manejadorFicherol);
?>
```

La única diferencia con el código anterior está en la línea resaltada. Al cargar este script verá su página con el aspecto de la figura 8.6.

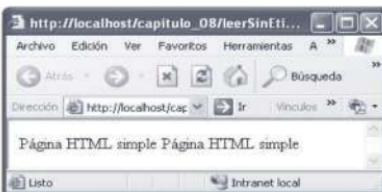


Figura 8.6

Ahora haga clic en **Ver** y, a continuación en **Código fuente**. Verá el listado de la página que hay cargada en el navegador, tal como ve en la figura 8.7.

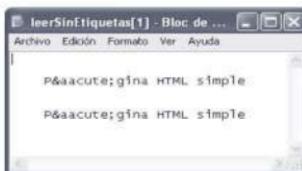


Figura 8.7

Como ve, se han suprimido todas las etiquetas HTML, quedando sólo las cadenas de texto que hay en el fichero, sin más.

Además de las funciones ya vistas existe una función de lectura de ficheros que resulta muy práctica. Se trata de *file()*. Como argumento recibe un nombre de fichero que no tiene por qué estar abierto previamente, y devuelve una matriz indexada en la que cada elemento tiene una línea del fichero. Observe el listado *leerParaMatriz.php*:

```
<?php
define ("salto","<br>\n");
```

```

// Se lee un fichero a una matriz.
$matriz = file("multiLinea.txt");
// Se recorre la matriz, mostrando cada elemento.
foreach ($matriz as $elemento => $contenido) {
    echo ("Elemento nº <b>$elemento</b>
Contiene: <b>$contenido</b>".salto);
}
?>
```

Al ejecutarlo verá en la página las líneas de texto del fichero multiLinea.txt como elementos de una matriz. Posteriormente puede tratar la susodicha matriz y/o sus elementos como cualquier otra, con las funciones que ya conoce.

Cuando se lleva a cabo la lectura de un fichero es necesario, muy a menudo, detectar la posición del puntero de lectura. Ya aprendimos, al ver la función fread() lo que es este puntero. Y también hemos aprendido a "rebobinarlo" al principio del fichero con rewind(). Además, el puntero es capaz de indicarnos si ha alcanzado el final del fichero mediante la función feof(). Existe una función específica para obtener la posición del puntero en cualquier momento de la lectura. Se trata de *fstell()*, que recibe como argumento el manejador con el que se abrió el fichero. En el siguiente código, llamado **verElPuntero.php** vemos cómo funciona:

```

<?php
// Se define el salto de línea
define ("salto","<br>\n");
// Se abre el archivo para su lectura.
$manejador=fopen("lecturaSimple.txt","r");
// Mientras no se alcance el final del fichero
while(!feof($manejador)){
// Se lee un carácter.
    $caracter=fgetc($manejador);
// Se obtiene la posición del puntero.
    $posicion=f.tell($manejador);
/* Se comprueba si se ha leído un carácter.
Esto se hace para no tratar la marca de final
de fichero como un carácter.*/
    if ($caracter){
// Se muestra el carácter y la posición que ocupa.
        echo ("El carácter en la posición ");
        echo("<b>$posicion</b> es <b>\"");
        echo("$caracter</b>\"".salto);
    }
}
// Se cierra el fichero.
fclose ($manejador);
?>
```

Quiero llamar su atención sobre la línea que aparece resaltada, donde se usa la función `ftell()` para determinar la posición del puntero de lectura. Cuando ejecute este script verá cómo se muestra en la página el recorrido del puntero, junto con el carácter que tiene el fichero en cada posición.

Quiero que se fije en una cosa. En este código hemos hecho la identificación de la posición del puntero *después* de leer un carácter, dentro del bucle. Fíjese en la disposición de las líneas de código, en cada iteración del bucle, *primero* leemos un carácter (con lo que se produce un desplazamiento del puntero), *y luego* vemos su posición. Al ejecutar el código, la primera posición que aparece corresponde al número 1. Esto es así porque, en realidad, el primer carácter es el 0. Si el uso de `ftell()` estuviera *antes* de la lectura de cada carácter, las posiciones aparecerían numeradas de 0 a 27, en lugar de hacerlo de 1 a 28.

Observe una vista parcial del resultado de este script en la figura 8.8.

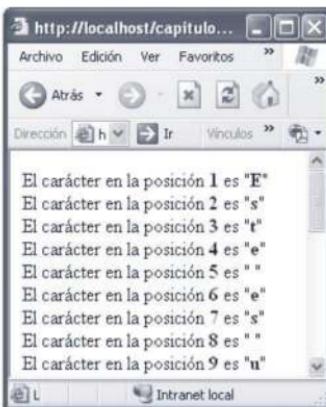


Figura 8.8

Y, ahora que ya sabe determinar la posición del puntero, veamos de qué modo lo llevaremos a donde nos pueda interesar. Para desplazar el puntero usted ya conoce la función `rewind()`, que lo posiciona al principio del fichero. Sin embargo, en ocasiones querremos llevarlo a otra posición diferente. Para ello, contamos con la función `fseek()`. Ésta recibe dos argumentos. El primero es el manejador del fichero sobre el que estamos trabajando, y el segundo es la posición a la que queremos desplazar el puntero. Esta posición se toma con respecto al inicio del fichero. Observe el script `moverPuntero.php`, listado a continuación:

```

<?php
// Se define el salto de línea.
define ("salto","<br>\n");
// Se abre el fichero para lectura.
$manejadorFicherol=fopen
("lecturaSimple.txt","r");
// Se lee la mitad del fichero
$contenido=fread ($manejadorFicherol,14);
// Se muestra la mitad leída.
echo ("La primera mitad del fichero es:
<b>$contenido</b>".salto);
echo ("Nos vamos a desplazar al primer carácter
del fichero.".salto);
// Desplazamos el puntero al cuarto carácter.

fseek ($manejadorFicherol,3);

// leemos el cuarto carácter y lo mostramos
echo ("El carácter correspondiente a la posición
actual es <b>".fgetc($manejadorFicherol)."</b>".salto);
// Se cierra el fichero
fclose ($manejadorFicherol);
?>

```

Vea en la linea resaltada que desplazamos el puntero hasta el cuarto carácter del fichero, que es el número tres ya que, como hemos visto anteriormente, los caracteres empiezan a numerarse desde 0.

El resultado de la ejecución de este script aparece en la figura 8.9.

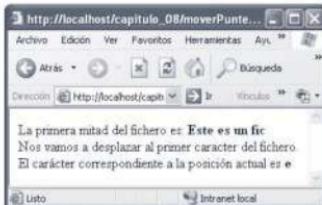


Figura 8.9

En la actualidad esta función puede recibir un tercer argumento, opcional, que indica desde dónde va a hacerse el desplazamiento. Como hemos visto, por defecto se hace desde el principio del fichero. Este argumento puede ser una de las tres constantes siguientes:

- **SEEK\_SET** – Efectúa el desplazamiento desde el inicio del fichero. Es el valor que se asume por defecto, por lo que se puede omitir.
- **SEEK\_CUR** – Efectúa el desplazamiento desde la posición actual del puntero. Si el segundo argumento es positivo, desplaza el puntero hacia delante. Si es negativo, lo desplaza hacia atrás.
- **SEEK\_END** – Efectúa el desplazamiento desde el final del fichero hacia atrás. En este caso el segundo argumento debe ser negativo.

Para ver un ejemplo del uso de este tercer parámetro mire el listado del script  **fseek3argumentos.php**, que aparece a continuación:

```
<?php
// Se define el salto de línea.
define ("salto","<br>\n");
// Se abre el fichero para lectura.
$manejadorFicherol=fopen
("lecturaSimple.txt","r");
// Se lee la mitad del fichero
$contenido=fread ($manejadorFicherol,14);
// Se muestra la mitad leída.
echo ("La primera mitad del fichero es:
<b>$contenido</b>".salto);
echo ("Nos vamos a desplazar al primer carácter
del fichero.".salto);
/* Desplazamos el puntero cinco caracteres hacia
atrás desde la posición actual. */
fseek ($manejadorFicherol,-5,SEEK_CUR);
// leemos el correspondiente carácter y lo mostramos
echo ("El carácter correspondiente a la posición
actual es <b>".fgetc($manejadorFicherol)."</b>".salto);
fclose ($manejadorFicherol); // Cierra el fichero
?>
```

En la linea resaltada ve el uso completo de esta función. Al ejecutar el script obtendrá el resultado de la figura 8.10.

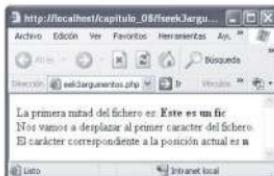


Figura 8.10

### 8.2.4 La escritura en ficheros

El proceso de escribir en un fichero es, paradójicamente, más simple que el de lectura, desde el punto de vista de las funciones destinadas a ello. Lo primero que debemos tener en cuenta cuando vayamos a escribir algo en un fichero es que debemos abrirlo usando el modo "a" o el modo "w", en lugar del modo "r". Si, además, prevemos que pueda ser necesario efectuar alguna lectura en el fichero, usaremos los modos "a+" o "w+", que permiten realizar ambas operaciones sin necesidad de cerrar y volver a abrir.

Para escribir en un fichero, empleamos la función *fwrite()*, que recibe tres argumentos. El primero es el manejador con el que se ha abierto el fichero. El segundo es el contenido que queremos escribir. El tercer parámetro es opcional. Se trata de un límite (un valor numérico) referente a la cantidad máxima de caracteres que se escribirán en la operación en curso, de modo que, si el segundo argumento tiene mayor longitud que la permitida por el tercero, sólo se grabará parte del nuevo contenido. Para comprobar la operativa de esta función vamos a empezar con un código que la emplea con los dos primeros parámetros, sin incluir el tercero. Se llama **escrituraCompleta.php**, y aparece listado a continuación:

```
<?php
    // Se define el salto de línea
    define ("salto","<br>\n");
    /* Se abre el archivo para su escritura. Recuerde que,
    cuando se intenta abrir un fichero para escritura, si este no
    existe, PHP se encarga de crearlo.*/
    $manejador=fopen("escrituraCompleta.txt","a");
    // Se define una cadena.
    $cadena="Esto es una cadena de texto.";
    /* Se intenta escribir la cadena y se comprueba si ha
    sido posible.*/
    if (@fwrite($manejador,$cadena)){
        echo ("La cadena se ha escrito en el
        archivo.");
    } else {
        echo ("NO SE HA PODIDO ESCRIBIR.");
    }
    // Se cierra el fichero.
    fclose ($manejador);
?>
```

Observe, en la línea resaltada, el uso de *fwrite()*. En este ejemplo hemos aprovechado la cualidad que tienen muchas funciones de PHP de devolver un valor true o false para comprobar si se ha podido llevar a cabo la operación en curso. Como el fichero donde se escribe la cadena, llamado **escrituraCompleta.txt**, no

existe, por defecto, en la carpeta en la que estamos trabajando, el intérprete del lenguaje se encarga de crearlo. Recuerde que, si estuviéramos trabajando en un servidor remoto (como se trabaja en Internet) el archivo de texto se crearía en dicho servidor, no en el lado del cliente.

Una vez haya ejecutado este script comprobará cómo se ha creado el archivo de texto. Ábralo con cualquier editor y verá que su contenido es la cadena que hemos establecido como segundo parámetro de la función.

Ahora ejecute el código **escrituraLimitada.php**. Es muy similar al anterior. La única diferencia (aparte del nombre del fichero de texto) es el uso de `fwrite()` con el tercer argumento, limitando la operación de escritura a seis caracteres, así:

```
if (@fwrite($manejador,$cadena,6)) {
```

Observe el uso de este parámetro. Si ahora abre con un editor de texto el fichero **escrituraLimitada.txt** verá que el contenido tiene sólo los seis primeros caracteres de la cadena que se quiso grabar, incluyendo el espacio en blanco.

Existe otra función para escritura en ficheros. Su nombre es **fputs()** y es lo que se conoce como un **alias** de `fwrite()`. Esto significa que tiene la misma sintaxis y funciona exactamente igual, por lo que puede usar cualquiera de las dos funciones indistintamente.

## 8.2.5 Eliminar ficheros

En ocasiones es necesario que se borren determinados ficheros en el servidor. Probablemente sean archivos que ya no van a utilizarse más, y que están ocupando un espacio que podemos necesitar para otro uso. En ese caso, empleamos la función **unlink()**, que recibe como argumento el nombre del archivo a eliminar. Para ejecutar esta función es necesario que el fichero no esté abierto. Si lo está, no se podrá llevar a cabo la eliminación. Si el fichero no existe, tampoco se podrá llevar a cabo su borrado, como es lógico. En la carpeta de este capítulo hay un fichero llamado **ficheroParaEliminar.txt**. No tiene ningún contenido. Sólo lo vamos a usar para ver cómo opera la función que estamos estudiando. Observe el código del script **eliminarFichero.php**, listado a continuación:

```
<?php  
// Se define el salto de línea  
define ("salto","<br>\n");  
/*Se intenta eliminar un fichero y se informa del  
resultado.*/
```

```

        if (@unlink("ficheroParaEliminar.txt")){
            echo ("Se ha eliminado el fichero.".salto);
        } else {
            echo ("NO se pudo eliminar el
fichero.".salto);
        }
    ?>

```

En la linea resaltada vemos cómo se usa unlink (). Cuando cargue este script en el navegador, verá que se elimina el fichero. Un mensaje le informa de ello en la página, tal como se ve en el código. Si vuelve a cargarlo, le saldrá el otro mensaje: el que le informa de que no se ha podido eliminar el fichero. Lógico, ya se eliminó en la primera ejecución, así que ya no existe y, por lo tanto, no puede eliminarse de nuevo. Recuerde que cuando se elimina un fichero se pierde de forma definitiva e irreversible. En el servidor no existe papelera de reciclaje, ni nada así. Ya no podrá recuperarlo de ningún modo.

## 8.2.6 Copiando ficheros

Un fichero se puede copiar, en la misma ruta en la que está o en otra diferente, desde un script PHP mediante el uso de la función *copy ()*. Esta función recibe dos argumentos. El primero es el nombre del fichero que queremos copiar y el segundo es el nombre que tendrá la copia. Si la ruta de alguno de ellos es diferente de aquélla en la que se ejecuta el script, también deberemos especificarla, como es lógico. Observe el listado **copiaFichero.php**, a continuación:

```

<?php
// Se define el salto de línea
define ("salto","<br>\n");
// Se establece el nombre que tendrá el fichero
original, y el de la copia.
$original="multilinea.txt";
$copia="copiado.txt";
// Se intenta copiar y se muestra el resultado.
if(@copy($original, $copia)){
    echo ("El fichero ha sido copiado.".salto);
} else {
    echo ("El fichero NO pudo ser
copiado.".salto);
}
?>

```

Este script está diseñado para hacer una copia de uno de los ficheros de texto empleados en las pruebas de lectura. Si todo ha ido correctamente, la página debe informarle de que el archivo se ha copiado, y en la carpeta de este capítulo

debe aparecer ahora el archivo de la copia, con el nombre especificado, y con el mismo contenido que el archivo original. Como es lógico, para que esto funcione el archivo original debe existir en la ruta especificada.

### 8.2.7 Renombrado de ficheros

PHP le permite cambiar el nombre de un archivo desde un script. Para ello recurrimos a la función *rename()*. Ésta debe recibir dos argumentos. El primero es el nombre que tiene el fichero antes de renombrarlo. El segundo parámetro es el nuevo nombre que le queremos dar. Esta función se puede usar, también, para cambiar la ubicación de un archivo (lo que se conoce como *mover un archivo*), simplemente especificando una ruta diferente en el nuevo nombre. En las anteriores versiones de PHP no se podía usar para mover un archivo, en entornos Unix, si el destino no pertenecía al mismo disco que el origen, pero esto ha sido subsanado en la actual versión.

Suponga el siguiente fragmento de código:

```
$origen="fichero.txt";
$destino="nuevoFichero.txt";
@rename ($origen, $destino);
```

Esto cambiaria el nombre del archivo “fichero.txt” por “nuevoFichero.txt”, dejándolo en la misma ruta. Ahora observe lo siguiente:

```
$origen="fichero.txt";
$destino="otroDirectorio/fichero.txt";
@rename ($origen, $destino);
```

Este fragmento movería el archivo “fichero.txt” a un directorio llamado “otroDirectorio” que se encuentra dentro del actual, conservando el nombre original del archivo.

Por supuesto, para que estas operaciones se puedan llevar a cabo es preciso que el archivo exista con el nombre original en la ruta especificada.

### 8.2.8 Las propiedades de los ficheros

No siempre se puede leer un fichero. A veces éste no existe, o puede que no tengamos los permisos adecuados (hablaremos de permisos más adelante, en este mismo capítulo). Lo mismo puede ocurrir con las operaciones de escritura o de eliminación. Antes de intentar llevar a cabo alguna de estas maniobras es interesante saber si se podrá efectuar. PHP nos proporciona funciones muy útiles.

Empezaremos por conocer la función *file\_exists ()*. Como argumento recibe un nombre de fichero (incluyendo la ruta, si procede) y devuelve un valor lógico, que será true si el fichero existe en la ruta especificada, o false en caso contrario.

La función *is\_readable ()* recibe como argumento un nombre de fichero (y la ruta, en su caso) y comprueba si el fichero se puede leer. En ese caso devuelve un valor true, y un valor false si el fichero no existe o no es legible.

Otra función a la que deberemos recurrir es *is\_writeable ()*. Recibe un nombre de fichero (con su ruta, si procede) y devuelve un valor true si el fichero nos permite escribir en él. Si el fichero no existe o no es escribible, obtendremos un valor false. Un fichero puede no ser escribible si está almacenado en un soporte que no admite la re-escritura, como un CD o similar.

La función *is\_executable ()* recibe un nombre de fichero (y su ruta) y devuelve un valor true si el fichero es ejecutable. Si el fichero no existe o no es ejecutable, devuelve un valor false. Esta función tradicionalmente disponible para usuarios de plataformas Unix, se ha incorporado para Windows en la actual versión de PHP.

La función *is\_file ()* determina si el nombre pasado como argumento es un fichero. En ese caso devuelve un valor true. Si no existe o no es un fichero, devuelve el valor false.

La función *is\_dir ()* determina si el nombre pasado como argumento es un directorio. En ese caso devuelve un valor true. Si no existe o no es un directorio, devuelve el valor false.

La función *is\_link ()* determina si el nombre pasado como argumento es un enlace simbólico. En ese caso devuelve un valor true. Si no existe o no es un enlace simbólico devuelve el valor false. Esta función, y el concepto de enlace simbólico, pertenecen a los entornos Unix. En entornos Windows no existen enlaces simbólicos. El concepto equivalente serían los accesos directos, pero PHP no los reconoce como enlaces, ni como nada. Por tanto, en entornos Windows esta función carece de sentido.

Para ver cómo operan estas funciones vamos a emplear el listado **comprobarPropiedades.htm**, que aparece a continuación:

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

```
<form name="propiedades" id="propiedades"
method="post" action="comprobarPropiedades.php">
    <input type="file" name="elemento" id="elemento">
    <input type="submit" name="mandar" id="mandar"
value="Enviar">
</form>
</body>
</html>
```

Aparentemente no tiene nada de especial. Sólo es un formulario con un campo de archivo en el que el usuario selecciona un fichero de su disco duro. Sin embargo, fíjese en la línea que aparece resaltada, donde se abre el formulario. No hemos incluido el atributo enctype="multipart/form-data", preceptivo en los formularios que incluyen campos de archivos. Como sabe, este atributo permite que, al mandar el formulario, se envíe el archivo seleccionado. Al no haberlo incluido, sólo se envía el nombre (con la ruta) de dicho archivo, que es, precisamente, lo que necesitamos en este caso. Este nombre se envía al script **comprobarPropiedades.php**, listado a continuación:

```
<?php
define ("salto","<br>\n");
// Se comprueba si existe el fichero.
if (file_exists($_POST_VARS["elemento"])) {
    echo ("El elemento existe.".salto);
} else {
    echo ("El elemento NO existe en la ruta
especificada ".salto);
}
// Se comprueba si es un fichero.
if (is_file($_POST_VARS["elemento"])) {
    echo ("El elemento es un fichero.".salto);
} else {
    echo ("El elemento NO es un fichero.".salto);
}
// Se comprueba si es un directorio.
if (is_dir($_POST_VARS["elemento"])) {
    echo ("El elemento es un directorio.".salto);
} else {
    echo ("El elemento NO es un directorio.".salto);
}
// Se comprueba si es un enlace simbólico.
if (is_link($_POST_VARS["elemento"])) {
    echo ("El elemento es un enlace simbólico.".salto);
} else {
    echo ("El elemento NO es un enlace
simbólico.".salto);
```

```
        }
        // Se comprueba si se puede leer.
        if (is_readable($HTTP_POST_VARS["elemento"])) {
            echo ("El elemento es legible.".salto);
        } else {
            echo ("El elemento NO es legible.".salto);
        }
        // Se comprueba si se puede escribir.
        if (is_writeable($HTTP_POST_VARS["elemento"])) {
            echo ("El elemento es escribible.".salto);
        } else {
            echo ("El elemento NO es escribible.".salto);
        }
        // Se comprueba si se puede ejecutar.
        if (is_executable($HTTP_POST_VARS["elemento"])) {
            echo ("El elemento es ejecutable.".salto);
        } else {
            echo ("El elemento NO es ejecutable.".salto);
        }
    ?>
```

Observe que lo único que hacemos es comprobar cada una de las propiedades que hemos descrito antes y mostrar en la página el mensaje adecuado en cada caso. Usted puede probar estas páginas con distintos archivos y ver los resultados.

### 8.2.9 Los permisos

Cuando se intenta trabajar con archivos es necesario tener permiso para hacerlo. Dicho así suena como una tontería, de tan evidente que parece, pero pensemos un poco en ello. Suponga que usted tiene, en un servidor remoto, unos archivos de imagen (fotografías, gráficos, etc.), archivos de texto, de vídeo, de sonido o, incluso, sus propios scripts de PHP. Todos estos archivos pueden ser leídos, re-escritos, eliminados, cambiados de nombre... si se cuenta con los permisos para ello. Por ejemplo, usted tiene publicado su sitio en un servidor de Internet. En una de las páginas se le da al cliente la opción de que le envie una fotografía, mediante un campo de archivo. Esta fotografía, tras pasar los controles adecuados, será almacenada por su script en un directorio del servidor, tal como aprendimos a hacerlo en el capítulo 5. Este archivo (la imagen) tiene un propietario, que es, en este caso, el intérprete de PHP, ya que se ha grabado mediante un script escrito en este lenguaje. Todos los archivos alojados en el servidor pueden ser gestionados, según los permisos otorgados, por tres *niveles de usuarios*: el propietario, el grupo y todos los demás. Éste es un concepto con el que los usuarios de Linux están muy familiarizados, aunque a los de Windows les

resultará, probablemente, desconocido. Volviendo al ejemplo de la fotografía, si usted ahora se conecta mediante un cliente de FTP a su servidor remoto, accede con su login y contraseña y trata de descargarse la fotografía que subió el usuario para tener una copia en su ordenador, seguramente no pueda hacerlo. Esto se debe a que el programa cliente de FTP no es el propietario de ese archivo.

Cada nivel de usuario puede tener *tres tipos de permisos* sobre los archivos: *de lectura, de escritura y de ejecución*. También puede ocurrir que un determinado nivel de usuarios no tenga ningún permiso sobre ciertos archivos. Como norma general, el propietario suele tener todos los permisos sobre sus propios archivos. Cuando usted alquila un alojamiento en un servidor remoto, el segundo nivel de usuarios, el grupo, suele estar formado por un solo usuario, que es el mismo que el propietario. El tercer nivel está formado por todos los demás y lo normal, en un servidor de Internet, es que este grupo no tenga ningún permiso sobre sus archivos.

La cuestión es que usted puede modificar los permisos de un archivo en concreto, para que, por ejemplo, todos los demás usuarios (el tercer nivel) puedan leerlo, pero no puedan modificarlo ni ejecutarlo. De este modo, usted ya podría conectarse a su servidor mediante un cliente FTP y descargarse la fotografía que antes se le negaba.

Los permisos se identifican mediante un sistema de numeración octal, basado en la numeración binaria. El número 4 corresponde al permiso de lectura, el 2 al de escritura y el 1 al de ejecución. Si usted, como propietario de un archivo, quiere establecer permiso de lectura y escritura, pero no de ejecución, usará el valor 6, que es la suma del valor 4 (lectura) y el 2 (escritura). Para establecer los permisos de un archivo se suman los valores básicos de cada uno de los tipos de permiso que se puedan otorgar. Los niveles de usuarios y los *niveles de permisos* aparecen en la tabla de la figura 8.11.

TABLA DE PERMISOS PARA FICHEROS			
Usuarios	Propietario	Grupo	Otros
Permisos			
Lectura	4	4	4
Escritura	2	2	2
Ejecución	1	1	1

Figura 8.11

Cuando se otorgan los permisos de un archivo es necesario establecer, en la misma instrucción, los de los tres niveles de usuarios. Suponga que quiere darle a un archivo todos los permisos para el propietario, permiso de lectura para el grupo y ningún permiso para todos los demás usuarios. Sumando los valores que acabamos de ver en la tabla, vemos que el propietario recibe el valor 7 ( $4+2+1$ ), el grupo recibe el valor 4 y el resto de los usuarios recibe el 0. Así pues, la secuencia adecuada es 740. Cuando se cambian los permisos desde PHP, esta secuencia debe ir siempre precedida por un 0, para asegurarnos de que el intérprete entiende que este número es octal. Por lo tanto, la secuencia definitiva queda así: 0740.

Para cambiar los permisos de un archivo recurrimos a la función **chmod()**, que recibe dos argumentos. El primero es el nombre del archivo, con la ruta, si es necesario. El segundo es la secuencia de permisos que queremos establecer. También contamos con la función **fileperms()**, que recibe como argumento el nombre de un fichero (y la ruta, si procede), y nos devuelve un valor numérico que indica los permisos actuales de ese fichero. Y aquí se presenta una pequeña dificultad. El valor devuelto por esta función no está en el formato octal habitual de los permisos. Debemos transformarlo a octal y mostrar, solamente, los cuatro últimos caracteres. Esto nos dará la secuencia real de permisos del fichero en cuestión, incluyendo el 0 preliminar que denota un valor octal. Veamos cómo en **mostrarPermisos.php**:

```
<?php
    echo (substr(sprintf('%o',
        fileperms("multiLinea.txt")), -4));
?>
```

Como ve, es un código muy simple. Usamos la función **fileperms()** para recuperar los permisos de uno de los ficheros contenidos en la carpeta de este capítulo. La secuencia obtenida la formateamos en octal mediante la función **printf()** que vimos en el capítulo 6, y extraemos los cuatro últimos caracteres de dicha secuencia. Tome nota. Si la función **substr()** recibe un número negativo para la cuenta de caracteres empieza a contarlos desde el final de la cadena. Al ejecutar este script verá en la página la secuencia 0666, salvo que su fichero tenga otros permisos asignados (por ejemplo, si está marcado como de "Sólo lectura"). El valor 6 significa permiso de lectura (4) y escritura (2), para los tres niveles de usuarios.

Ahora vamos a hacer un trabajo un poco más completo. Crearemos una página que nos permita elegir un fichero, ver sus permisos actuales y cambiarlos. El sistema constará de tres páginas. La primera es un formulario totalmente basado en tecnologías de cliente (HTML y JavaScript), llamado **permisos.htm**. En él se selecciona un fichero en un campo de archivo. En un marco flotante se cargará un pequeño script PHP, parecido al que acabamos de ver, llamado **leerPermisos.php**,

que permitirá leer los permisos del archivo seleccionado en tiempo real. Es decir, si usted cambia el archivo, se leerán los permisos de la nueva selección. El formulario contará también con nueve casillas para poder establecer los distintos permisos para los tres niveles de usuarios. Al enviar el formulario, los datos llegarán al script **modificarPermisos.php**, que se encargará de otorgar los permisos seleccionados al archivo elegido. El código de **permisos.htm** es el siguiente:

```
<html>
  <head>
    <title>Gesti&on de permisos de
ficheros</title>
    <script language="javascript"
type="text/javascript">
      /* La siguiente función se ocupa de cargar el script
"leerPermisos.php" en el marco flotante. Además, le pasa,
como variable GET (en la URL), el nombre del fichero cuyos
permisos se quieren leer. Dado que este script se carga en un
marco, la llamada no aparece en la barra de direcciones del
navegador, donde sigue cargada la página principal (esta
misma). Esta función se llama cada vez que cambia el valor en
el campo de archivo.*/
      function actualizarPermisoVigente (){
        document.getElementById("marcoPermisos").src
=
"leerPermisos.php?fichero="+document.getElementById("ficheroE
legido").value;
      }

      /* La siguiente función se ocupa de actualizar el nodo
de texto con la secuencia de permisos que desea establecer el
usuario. Esta función es invocada cada vez que se activa o
desactiva alguna de las checkbox relativas a los permisos que
hay en el formulario principal, y recalcula la secuencia de
permisos, mostrándola en el correspondiente nodo de texto.*/
      function actualizarNuevoPermiso (){
        // Se recalcula el valor de los permisos para el
propietario.
        permisosPropietario=0;
        for (contador=1;contador<4;contador++){
          if (document.forms["fPermisos"].elements
[contador].checked) permisosPropietario +=
parseInt(document.forms["fPermisos"].elements[contador].value
);
        }
        // Se recalcula el valor de los permisos para el grupo.
        permisosGrupo=0;
```

```
        for (contador=4; contador<7; contador++) {
            if (document.forms["fPermisos"].elements[contador].checked) permisosGrupo += parseInt(document.forms["fPermisos"].elements[contador].value);
        }
        // Se recalcula el valor de los permisos para el resto de usuarios.
        permisosOtros=0;
        for (contador=7; contador<10; contador++) {
            if (document.forms["fPermisos"].elements[contador].checked) permisosOtros+=parseInt(document.forms["fPermisos"].elements[contador].value);
        }
permisosParaOtorgar="0"+permisosPropietario+permisosGrupo+permisosOtros;
        // Se re-escribe el nodo de texto de los permisos.
        document.getElementById("nuevoPermiso").firstChild.nodeValue = permisosParaOtorgar;
    }
    /* La siguiente función es invocada cuando se pulsa el botón de "RESET" del formulario.
    Se encarga de limpiar los nodos de texto.*/
    function borrarPermisosActuales (){
        document.getElementById("permisoVigente").firstChild.nodeValue = "0000";
        actualizarNuevoPermiso();
        document.getElementById("nuevoPermiso").firstChild.nodeValue = "0000";
    }
    /* La siguiente función es invocada cuando se pulsa el botón destinado a enviar el formulario y cambiar los permisos. Lo que hace es cargar el nombre del fichero y el valor de permisos elegidos en los campos ocultos del formulario secundario, que es el que se enviará al script encargado de modificar los permisos. El formulario principal no es enviado a ninguna parte. Sólo es una interfaz para el usuario.*/
    function enviarFormulario (){
        document.getElementById("ocultoNombre").value=document.getElementById("ficheroElegido").value;
        document.getElementById("ocultoPermisos").value=document.getElementById ("nuevoPermiso").firstChild.nodeValue;
```

```
        document.fEnviar.submit();
    }
</script>
</head>

<body>
<h3>GESTIÓN DE PERMISOS DE FICHEROS<br>
    Empleamos las funciones fileperms() y chmod()</h3>
<!-- Aquí empieza el que podemos llamar "Formulario Principal". Contiene la interfaz del usuario. Sus datos no serán nunca enviados a ninguna parte (observe que no tiene valor alguno en el atributo action). Los datos que nos interesa enviar se cargarán luego en campos ocultos de otro formulario, que es el que será enviado.-->
<form action="" method="post" name="fPermisos"
id="fPermisos"
onReset="javascript:borrarPermisosActuales();"
    <table width="600" border="0" cellspacing="0"
cellpadding="0">
        <tr>
            <td align="right">Seleccione el
fichero:&ampnbsp</td>
            <td><input name="ficheroElegido" type="file"
id="ficheroElegido" size="50"
onChange="javascript:actualizarPermisoVigente ();"></td>
        </tr>
    </table>
    <table width="600" border="0" cellspacing="0"
cellpadding="0">
        <tr align="center" valign="middle">
            <td height="44" colspan="4"><h4>Establezca
los permisos que desea otorgar al fichero.</h4></td>
        </tr>
        <tr>
            <td width="173">&ampnbsp</td>
            <td width="130">Lectura</td>
            <td width="120">Escritura</td>
            <td width="167">Ejecución</td>
        </tr>
        <tr>
            <td>Propietario</td>
            <td><input name="lProp" type="checkbox"
id="lProp2" value="4"
onClick="javascript:actualizarNuevoPermiso ();"></td>
            <td><input name="eProp" type="checkbox"
id="eProp" value="2"
onClick="javascript:actualizarNuevoPermiso ();"></td>
```

```
<td><input name="xProp" type="checkbox"
id="xProp2" value="1"
onClick="javascript:actualizarNuevoPermiso () ;"></td>
</tr>
<tr>
    <td>Grupo</td>
    <td><input name="lGrupo" type="checkbox"
id="lGrupo2" value="4"
onClick="javascript:actualizarNuevoPermiso () ;"></td>
        <td><input name="eGrupo" type="checkbox"
id="eGrupo" value="2"
onClick="javascript:actualizarNuevoPermiso () ;"></td>
            <td><input name="xGrupo" type="checkbox"
id="xGrupo5" value="1"
onClick="javascript:actualizarNuevoPermiso () ;"></td>
                </tr>
                <tr>
                    <td>Otros</td>
                    <td><input name="lOtros" type="checkbox"
id="lOtros4" value="4"
onClick="javascript:actualizarNuevoPermiso () ;"></td>
                        <td><input name="eOtros" type="checkbox"
id="eOtros3" value="2"
onClick="javascript:actualizarNuevoPermiso () ;"></td>
                            <td><input name="xOtros" type="checkbox"
id="xOtros2" value="1"
onClick="javascript:actualizarNuevoPermiso () ;"></td>
                                </tr>
                                </table>
                                <table width="600" border="0" cellspacing="0"
cellpadding="0">
                                    <tr align="center">
                                        <td width="297" height="55"><input
name="aceptar" type="button" id="aceptar" value="Cambiar
permisos" onClick="javascript:enviarFormulario(); "></td>
                                            <td width="297"><input name="borrar"
type="reset" id="borrar" value="Limpiar formulario"></td>
                                        </tr>
                                    </table>
                                </form>
                                <table width="600" border="0" cellspacing="0"
cellpadding="0">
                                    <tr>
                                        <td width="297">&nbsp;</td>
                                        <td width="297">&nbsp;</td>
                                    </tr>
                                    <tr>
```

```
<td align="right">Permisos actuales del
fichero:&nbsp;</td>
<!-- Se crea el nodo de texto con los permisos actuales
del fichero. -->
<td><div id="permisoVigente">0000</div></td>
</tr>
<tr>
    <td align="right">Nuevos permisos:&nbsp;</td>
<!-- Se crea el nodo de texto donde se reflejan los
nuevos permisos del fichero. -->
    <td><div id="nuevoPermiso">0000</div></td>
</tr>
</table>
<!-- El siguiente formulario es el que contiene dos
campos ocultos que se actualizarán con el nombre del fichero
y los permisos que deseamos darle. Estos dos campos serán
enviados al script que se encarga de modificar los permisos
del fichero elegido.-->
<form action="modificarPermisos.php" method="post"
name="fEnviar" id="fEnviar">
    <input name="ocultoNombre" type="hidden"
id="ocultoNombre">
    <input name="ocultoPermisos" type="hidden"
id="ocultoPermisos">
</form>
<!-- En el marco flotante, y de forma invisible al
usuario, se carga el script que lee los permisos actuales del
archivo y los muestra en el correspondiente nodo de texto de
la página principal.-->
    <iframe name="marcoPermisos" id="marcoPermisos"
src="" width = "1" height = "1" frameborder="0">
    </iframe>
</body>
</html>
```

Si analiza el código, verá que no tiene complicación alguna. Se trata sólo de una página HTML, con la colaboración de algo de JavaScript para complementar su funcionalidad. Toda la página es tecnología de cliente. Los comentarios incorporados le facilitarán el análisis del código, si desea ver cómo funciona. El motivo de haber incluido el listado aquí es doble: por una parte, quiero que vea un nuevo ejemplo (ya hemos visto alguno anteriormente) del uso de una página que emplea dos formularios. Uno de ellos (al que nos referimos como formulario principal en los comentarios) contiene la interfaz del usuario. Este formulario no se envía a ninguna parte. El otro (que el usuario no ve y del que no sabe nada) contiene campos ocultos que son actualizados con los datos adecuados antes del envío. Es éste formulario el que se envía contra un script del servidor. El

otro motivo por el que he copiado aquí el listado HTML es el marco flotante que aparece al final. Como ve, sólo tiene un píxel de lado y no tiene bordes, con lo cual ni el propio marco ni su contenido serán nunca visibles para el usuario. En este marco no hay, en principio, ninguna página cargada. Es desde una función de JavaScript que se carga un script en dicho marco. Este script trabaja “por debajo”, es decir, tal como hemos dicho, de forma invisible al usuario. Luego enviará sus resultados a la página principal, de un modo que veremos en seguida. Esta técnica es muy empleada en páginas dinámicas. Esto es así porque el código PHP se interpreta en el servidor, generando, como usted sabe, un código HTML que es enviado al cliente. Si en cada llamada a un script PHP se “recrea” la página, el usuario lo percibiría y resulta un efecto bastante molesto a la vista el que la página principal se recargue cuando se está trabajando con ella. Recargándola en un marco invisible le evitamos molestias al usuario. Es una cuestión de estilo y de fidelización de visitantes. La persona que navegue por nuestro sitio debe sentirse cómoda.

La página que acabamos de ver tiene el aspecto de la figura 8.12.

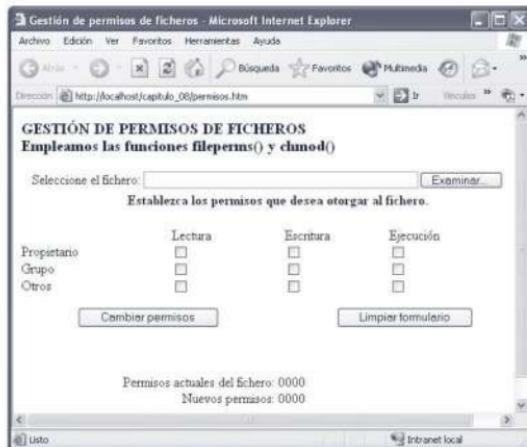


Figura 8.12

Observe debajo del formulario los dos nodos de texto con los permisos actuales del fichero y los nuevos permisos que deseamos otorgarle. Cada vez que se selecciona un archivo, el código JavaScript hace que se cargue en el marco invisible un script PHP encargado de leer los permisos de dicho archivo y, en

función de éstos, modificar el correspondiente nodo de texto y las casillas, activando las que corresponden. El código encargado de esto, **leerPermisos.php**, es el siguiente:

```
<?php
/* Se eliminan las barras sobrantes que se han
producido durante la transmisión del nombre del fichero (ver
capítulo 7).*/
$HTTP_GET_VARS["fichero"] = ereg_replace("\\\\\\\\", 
"/", $HTTP_GET_VARS["fichero"]);
/* Se obtienen los permisos del archivo especificado. El
nombre es pasado mediante GET, pero no aparece en la barra de
direcciones porque esta página se carga en un marco de la
página principal. Además, el nombre del fichero no se
transmite desde un formulario enviado con GET, sino que es
transmitido como una variable desde una línea de JavaScript
(ver el código de la página principal).*/
$permisosActuales=substr(sprintf('%o',
fileperms($HTTP_GET_VARS["fichero"])), -4);
?>
<html>
<body>
<script language="javascript" type="text/javascript">
/* Se obtienen los permisos del fichero especificado,
que se han recuperado con PHP, en una variable de
JavaScript.*/
permisos=<?php echo ($permisosActuales);?>;
/* Se muestran los permisos en un nodo de texto de la
página principal.*/
this.parent.document.getElementById
("permisoVigente").firstChild.nodeValue = permisos;
this.parent.document.getElementById
("nuevoPermiso").firstChild.nodeValue = permisos;
/* Se obtienen los permisos de los tres niveles de
usuarios como variables numéricas.*/
permisosPropietario=parseInt(permisos.substr(1,1));
permisosGrupo=parseInt(permisos.substr(2,1));
permisosOtros=parseInt(permisos.substr(3));
/* Se marcan o desmarcan las casillas correspondientes
a los permisos del propietario en la página principal,
evaluando los tres posibles niveles de permisos, de forma que
queden marcadas las casillas que corresponden a los permisos
actuales.*/
this.parent.document.getElementById
("lProp").checked=eval(permisosPropietario>3);
if (permisosPropietario>3) permisosPropietario-=4;
```

```
this.parent.document.getElementById  
("eProp").checked=eval(permisosPropietario>1);  
if (permisosPropietario>1) permisosPropietario-=2;  
this.parent.document.getElementById  
("xProp").checked=eval(permisosPropietario>0);  
  
/* Se marcan o desmarcan las casillas correspondientes  
a los permisos del grupo en la página principal, evaluando  
los tres posibles niveles de permisos, de forma que queden  
marcadas las casillas que corresponden a los permisos  
actuales.*/  
this.parent.document.getElementById  
("lGrupo").checked=eval(permisosGrupo>3);  
if (permisosGrupo>3) permisosGrupo-=4;  
this.parent.document.getElementById  
("eGrupo").checked=eval(permisosGrupo>1);  
if (permisosGrupo>1) permisosGrupo-=2;  
this.parent.document.getElementById  
("xGrupo").checked=eval(permisosGrupo>0);  
  
/* Se marcan o desmarcan las casillas correspondientes  
a los permisos del nivel "Otros" en la página principal,  
evaluando los tres posibles niveles de permisos, de forma que  
queden marcadas las casillas que corresponden a los permisos  
actuales.*/  
this.parent.document.getElementById  
("lOtros").checked=eval(permisosOtros>3);  
if (permisosOtros>3) permisosOtros-=4;  
this.parent.document.getElementById  
("eOtros").checked=eval(permisosOtros>1);  
if (permisosOtros>1) permisosOtros-=2;  
this.parent.document.getElementById  
("xOtros").checked=eval(permisosOtros>0);  
</script>  
</body>  
</html>
```

El script está comentado para que pueda seguirlo con comodidad. Como ve se distinguen dos partes claramente diferenciadas. La primera es código PHP que filtra, en primer lugar, el nombre del archivo, para eliminar las barras sobrantes que se han añadido durante la carga, tal como aprendimos en el capítulo anterior. A continuación, se hace uso de la función fileperms (), para obtener los permisos del archivo seleccionado. La segunda parte es un JavaScript muy simple para enviar los permisos de los distintos niveles de usuario a la página principal, y actualizar el valor de los nodos de texto y el estado de las casillas de verificación. Como este script se recarga cada vez que se cambia el fichero elegido en la página principal,

estos datos siempre se actualizan en tiempo real. Quiero llamar su atención, especialmente, sobre la línea que he resaltado, para que vea cómo podemos *leer desde JavaScript una variable de PHP*.

Por último tenemos el script **modificarPermisos.php**, cuyo listado aparece a continuación:

```
<?php
    define ("salto","<br>\n");
    /* Se eliminan las barras sobrantes que se han
    producido durante la transmisión del nombre del fichero (ver
    capítulo 7).*/
    $HTTP_POST_VARS["ocultoNombre"] =
ereg_replace("\\\\\\\\", "/", $HTTP_POST_VARS["ocultoNombre"]);
    // Los niveles de permisos no pueden ser una cadena.
    $permisosOtorgados=octdec($HTTP_POST_VARS["oculto
Permisos"]);
    // Se intentan cambiar los permisos y se verifica si ha
    sido posible.
    if (@chmod($HTTP_POST_VARS["ocultoNombre"],
$permisosOtorgados)){
        echo ("Los permisos han sido cambiados
satisfactoriamente.".salto);
        echo ("El archivo
".$HTTP_POST_VARS["ocultoNombre"]);
        echo (" tiene ahora los permisos
".$HTTP_POST_VARS["ocultoPermisos"].salto);
    } else {
        echo ("LOS PERMISOS NO PUDIERON
CAMBIARSE.".salto);
    }
?>
<html>
<body>

    <button onClick="location.href='permisos.htm';">
    REGRESAR
    </button>
</body>
</html>
```

Como ve, se recibe el nombre de un fichero y los permisos que se desea darle. Estos datos proceden de los campos ocultos del formulario en la página permisos.htm. En este script se hace uso de la función chmod () para llevar a cabo el cambio de permisos. Quiero que se fije en la parte resaltada. La función chmod

() no permite que el parámetro relativo a los permisos sea una cadena alfanumérica. Sin embargo, es precisamente en ese formato como se transmiten los datos desde un formulario, así que hay que hacer una conversión antes de aplicar los permisos. La conversión que hemos hecho aquí es la aconsejada por la documentación oficial de PHP ([www.php.net](http://www.php.net)), y funciona correctamente.

Un comentario final importante. Cuando su servidor es una máquina corriendo bajo Windows existe una limitación. Esta plataforma no reconoce los tres niveles de usuarios, de modo que los permisos que usted fije para el propietario del archivo quedarán también para el grupo y el resto de usuarios. Si por ejemplo, usted da permisos de lectura y escritura al propietario, y sólo de lectura al grupo y al resto, todos tendrán permisos de lectura y escritura. Esta limitación no existe si su servidor está corriendo en una máquina bajo Linux o cualquier otra plataforma Unix. Esto se debe a que la función chmod () hace uso de las atribuciones de gestión de permisos del propio sistema operativo. Muchos servidores profesionales son Linux, pero si alguna vez trabaja para un servidor Windows recuerde esta circunstancia. Sólo cuenta el propietario de los archivos.

### 8.3 DIRECTORIOS

Como usted sabe, los ficheros en su disco duro se organizan en una estructura compartimentada, dentro de unos contenedores adecuados. En entornos Windows estos contenedores se llaman carpetas. En entornos Unix, Linux, etc., se llaman directorios. El nombre cambia, pero el concepto es similar (salvando, lógicamente, las diferencias de ambos sistemas de archivos).

Por supuesto, cuando usted trabaja de modo remoto contra un servidor, el sistema de organización persiste. Por ejemplo, si su sitio incluye un formulario para que el usuario le envíe fotografías que luego se publicarán en alguna de sus páginas, lo normal es que éstas se almacenen en un directorio específico. Los archivos de código (HTML, JS, PHP, etc.) también tendrán su propio directorio, y así, sucesivamente. Los directorios tienen cierta funcionalidad implícita, es decir, se pueden crear nuevos directorios, se puede cambiar el directorio por defecto y se pueden eliminar directorios. PHP nos proporciona las funciones adecuadas para ello, que vamos a conocer en esta sección.

Cuando trabajamos con un directorio, en él se pueden encontrar una serie de ficheros y otros directorios, dando lugar a la típica estructura en árbol de cualquier sistema de almacenamiento actual. Entre los elementos que hay en un directorio encontraremos dos específicos: el punto ("."), que se refiere al directorio actual y el punto-punto (".."), que se refiere al directorio padre del actual. Por supuesto, este último no existe en el directorio raíz de la partición.

Usted ya sabe referirse a los directorios de una partición. El concepto de rutas absolutas y rutas relativas lo conoce desde que empezó a manejar enlaces en HTML. Pero hay un punto sobre el que, llegado este momento, quiero llamar su atención, a pesar de que ya lo hemos usado de pasada. Usted sabe que en un entorno Windows puede usar el slash (“/”) o el contraslash (“\”) indistintamente para separar los nombres de los directorios en una ruta. Sin embargo, en entornos Unix-Linux sólo se reconoce el slash, por lo que nosotros siempre emplearemos este guarismo para garantizar la portabilidad de aplicaciones.

### 8.3.1 Manejo básico de directorios

Existen tres operaciones básicas que podemos realizar con los directorios: crear un nuevo directorio, cambiar el directorio actual (por defecto) y eliminar un directorio. PHP nos proporciona tres funciones de muy sencillo uso para esto. La primera es ***mkdir()***, que permite crear un nuevo directorio. Esta función recibe dos argumentos, separados por comas. El primero es el nombre del directorio que queremos crear y el segundo representa los permisos con los que queremos crear dicho directorio. Respecto a los permisos de los directorios, es válido todo lo que hemos aprendido sobre permisos en la parte de ficheros, incluyendo el uso de las funciones que hemos estudiado. Suponga que queremos crear, dentro del directorio actual, uno con el nombre “carpetaDeImagenes” y con el máximo de permisos para los tres niveles de usuarios. Usaremos la función ***mkdir()*** como sigue:

```
mkdir ("carpetaDeImagenes", 0777);
```

Ahora suponga que quiere crear un directorio llamado “otraCarpeta”, directamente en el directorio raíz de la partición activa (normalmente, C:/, en entornos Windows). Usaremos la siguiente instrucción:

```
mkdir ("C:/otraCarpeta", 0777);
```

Para cambiar el directorio actual usamos la función ***chdir()***. Ésta recibe un argumento único, que es el nombre de la carpeta a la que queremos referirnos. Suponga que queremos acceder al directorio llamado “carpetaDeImagenes” que hemos creado anteriormente. Usaremos la siguiente instrucción:

```
chdir ("carpetaDeImagenes");
```

Ahora suponga que queremos acceder a un directorio llamado “carpetaDeSonidos”, que se encuentra en una rama paralela a la ruta por defecto, es decir, “cuelga” del padre del directorio actual. Usaremos lo siguiente:

```
chdir ("../carpetaDeSonidos");
```

Si queremos eliminar un directorio recurrimos a la función **rmdir()**, que recibe como argumento el nombre del directorio que deseamos que desaparezca. Para borrar un directorio es necesario que esté vacío (sin ficheros u otras carpetas en su interior) y que tengamos permiso de escritura sobre él. Suponga que, habiendo vuelto al directorio de nuestros ejemplos anteriores, desea eliminar "carpetaDeImagenes". Usaría una instrucción como la siguiente:

```
rmdir ("carpetaDeImagenes");
```

Respecto a la eliminación de un directorio le digo lo mismo que respecto a un fichero: es definitiva e irreversible, así que pienselo dos veces.

Cada una de las tres funciones que acabamos de ver devuelve un valor true si ha podido llevarse a cabo correctamente, y un valor false en caso contrario.

Existe una función adicional, menos conocida, que nos permite recuperar el nombre del directorio actual. Se llama **getcwd()**. Esta función no recibe ningún argumento.

Y ahora que ya sabemos crear directorios, cambiar el directorio activo y eliminar aquéllos que ya no necesitemos, tendremos que poder hacer "algo", con los directorios. Por supuesto, siempre podremos grabar en ellos los ficheros que lleguen a través de formularios, mediante la función **move\_uploaded\_file()**, que conocimos en el capítulo 5. O podremos copiar, renombrar o eliminar archivos, o incluso crearlos y leerlos, mediante las funciones que hemos aprendido en este mismo capítulo.

Sin embargo, hay cuatro operaciones básicas que aún no conocemos, destinadas a "ver" el contenido de un directorio. En efecto, en ocasiones es necesario determinar qué ficheros y sub-directorios contiene una carpeta. Para ello, debemos empezar por "abrirla", mediante la función **opendir()**. El argumento que recibe es el nombre del directorio cuyo contenido queremos leer. Esta función devuelve un **manejador de directorio**, de modo similar a lo que ocurre cuando abrimos un fichero.

Una vez abierto el directorio recurrimos a la función **readdir()** para leer su contenido. Como argumento le pasamos el manejador que obtuvimos al abrirlo y nos muestra (en seguida veremos cómo) el contenido del directorio.

Cuando se lee el contenido de un directorio mediante la función anterior, se está moviendo internamente un puntero, de modo análogo a como ocurre con los ficheros. Si queremos situarlo al principio del directorio, usaremos la función **rewinddir()**, que recibe, como argumento, el manejador del directorio.

Por último, cuando hayamos terminado de leer el contenido de un directorio, lo cerraremos con *closedir()*. Esta función recibe, como argumento, el manejador del directorio.

Suponga que desea leer el contenido de la carpeta principal en la partición activa. Para este ejemplo supondremos que estamos trabajando en un entorno Windows y que nuestra partición se llama C:. El siguiente script, de nombre **mostrarC.php**, mostraría su contenido:

```
<?php
    // Se define el salto de línea
    define ("salto","<br>\n");
    // Se "abre" el directorio principal de la partición.
    $manejador=opendir("C:/");
    echo ("Contenido del directorio principal:".salto);

    // Se "rebobina" el directorio para asegurarnos de
    posicionarnos al principio.
    rewinddir($manejador);

    // Mientras haya elementos (directorios o ficheros)
    para leer.
    while ($contenido=readdir($manejador)){
        echo ($contenido.salto);
    }
    // Se cierra el directorio.
    closedir($manejador);
?>
```

Cuando ejecute este código verá en su página una lista de los directorios y archivos que tiene en su disco duro. Sin embargo, es una forma bastante limitada de explorar el disco. Podemos mejorarla mucho. Por ejemplo, un usuario siempre agradece una interfaz gráfica cómoda y manejable. Observe el listado del script que aparece a continuación, y que hemos llamado **gestionarDirectorios.php**:

```
<html>
    <head>
        <title>Navegar por los directorios</title>
        <script language="javascript" type="text/javascript">

            /*La siguiente función es invocada cada vez que se hace
            click en uno de los enlaces que se crean dinámicamente más
            adelante. Como argumento recibe la ruta del disco a la que
            debemos ir a leer. Este valor lo pasa a un campo oculto de un
            formulario y envía dicho formulario. Como es un autoprocesado
```

```
(no tiene valor alguno en el atributo action) se recarga esta
misma página.*/
function recargar(direccion){

document.getElementById("rutaParaLeer").value=direccion;
    document.fRutas.submit();
}
</script>
</head>

<body>
    <h3>Listado de ficheros y directorios</h3>
    <form name="fRutas" id="fRutas" method="post"
action="">
        <input type="hidden" value="" name="rutaParaLeer"
id="rutaParaLeer">
        </form>
        <table width="300" border="0" cellpadding="4"
cellspacing="0">
            <?php
// Se define el salto de línea
define ("salto","\n");

/* Se comprueba si la variable de POST llamada
"rutaParaLeer" tiene contenido. Si no lo tiene, es que se
acaba de cargar el script en el navegador y se le asigna la
ruta raíz de la partición activa (válido solo para entornos
Windows). Si se ha recargado el script por haber pulsado uno
de los enlaces esta variable tiene el contenido que se haya
asignado al campo oculto del formulario. Dicho campo tiene,
como es lógico, el mismo nombre que esta variable.*/
if ($HTTP_POST_VARS["rutaParaLeer"]==""){
    $HTTP_POST_VARS["rutaParaLeer"]="C:/";
}
// Se abre el directorio especificado.

$manejador=opendir($HTTP_POST_VARS["rutaParaLeer"]);
// Se "rebobina" el directorio para asegurarnos de
posicionarnos al principio.
rewinddir($manejador);
// Mientras haya elementos (directorios o ficheros)
para leer.
while ($contenido=readdir($manejador)){
    echo ("<tr>".salto);
    echo ("<td>");
/*
Cuando se lee cada elemento se establece como nombre
completo de dicho elemento el mismo, precedido por la ruta
actual, y se almacena en la variable $nuevaRuta. */

```

```
$nuevaRuta=$HTTP_POST_VARS["rutaParaLeer"].$contenido."/";  
/* Se comprueba si el nombre completo de la ruta  
corresponde a un directorio. Si no es así, asumiremos que,  
genéricamente, es un fichero. En un entorno Windows, esto nos  
vale. En un entorno Linux quedaría un poco pobre y tendríamos  
que "afinar" más comprobando, por ejemplo, si el elemento es  
un enlace simbólico.*/
    if (is_dir($nuevaRuta)){
        /* Si es un directorio mostramos el icono  
correspondiente, creando, además, un enlace para poder  
acceder a dicho directorio mediante el formulario y la  
función JavaScript que hemos visto anteriormente en el  
código.*/
        echo ("<a"
href='javascript:recargar(\"".$nuevaRuta."\")'>");
            echo ("<img src='iconos/directorio.gif'
alt='Directorio' border=0");
            echo ("</a>");
        } else {
            /* Si no es un directorio mostramos el icono que hemos  
puesto para ficheros (elemento genérico) y no creamos enlace  
alguno.*/
            echo ("<img src='iconos/fichero.gif'
alt='Fichero'");
        }
        echo ("</td>".salto."<td>");
        /* Mostramos el nombre del elemento.*/
        echo ("<b>$contenido</b>");
        echo ("</td>".salto."</tr>".salto);
    }
// Se cierra el directorio.
closedir($manejador);
?>
</table>
</body>
</html>
```

Al ejecutar este código obtenemos como resultado una página similar a la que aparece en la figura 8.13.

Lógicamente, el resultado que le aparezca a usted será diferente, dado que no tiene los mismos directorios y ficheros que yo. Observe los dos tipos de icono diferentes que aparecen a la izquierda de cada elemento y que indican si el mismo es una carpeta u otro elemento (genéricamente, un fichero). Los iconos que se refieren a un directorio son, además, un enlace para abrirla. Pulse sobre alguno de

ellos y verá cómo la página se recarga, mostrando el contenido del mismo. Si, una vez dentro de una carpeta pulsa sobre el elemento .. verá que vuelve al directorio padre correspondiente. En los ficheros no he montado enlaces para su apertura por dos razones. La primera, y más evidente, es que sobre los ficheros ya hemos hablado y se escapa del objetivo de este ejercicio en concreto. La segunda es que abrir indiscriminadamente ficheros del disco duro, sin saber cuáles y de qué modo, puede acarrearnos problemas.



Figura 8.13

Estudie el código, que emplea funciones que ya conocemos perfectamente, así como recursos de HTML y JavaScript. Preste especial atención a los comentarios que he incluido al efecto, ya que le aportan gran valor didáctico.

Un comentario acerca de la lectura de directorios. PHP le permite, al igual que ocurre con los ficheros, la apertura remota, es decir, en otro equipo. Desde la versión 5 se ha incorporado la envoltura URL `ftp://`. Esto significa que podemos abrir directorios remotos para trabajar con ellos mediante el protocolo de transferencia de archivos. Hablaremos de la gestión remota de recursos más adelante, en este mismo libro.

## CAPÍTULO 9

### COOKIES Y SESIONES

---

---

Cuando nos conectamos a un sitio web remoto solemos hacerlo mediante el protocolo http://. Una conexión típica a Internet tiene la forma http://www.todoligues.com. También podemos usar, en determinadas páginas, el protocolo https:// (las llamadas páginas seguras). Ambos son lo que se conoce como *protocolos sin estado*. Esto significa que ni el servidor ni el cliente conservan memoria de los pares nombre-valor empleados en una página, al pasar a otra, a menos que pasemos dichos datos específicamente, mediante un formulario. Para solventar esto aparecen las *cookies* y las *sesiones*. Ambas son, en su concepto básico, formas de almacenar provisionalmente unos datos que otra página recopilará para seguir trabajando con ellos. La diferencia fundamental es el lugar de almacenamiento de dichos datos. Las cookies se usan para guardarlos en el cliente y las sesiones para guardarlos en el servidor. Ambos sistemas tienen sus ventajas y sus limitaciones. Por ejemplo, si recurrimos a las sesiones, y nuestro sitio cuenta con muchas visitas simultáneas, el servidor se puede sobrecargar. Si preferimos usar cookies y el cliente las tiene desactivadas, se presentará un fallo (solucionable, en última instancia). Cuando necesitamos almacenar datos para pasarlos de una página a otra, el empleo de sesiones o de cookies es, en algunos casos, una cuestión de criterio personal. En otras situaciones, debido al alcance que deban tener los datos, se presenta como más adecuado un sistema u otro. En este capítulo aprenderemos a usar ambas técnicas y a decidir cuál elegir en cada caso.

Como medida preliminar, olvide los mitos que haya podido oír respecto a estas técnicas. Ni las cookies son peligrosas para el ordenador del cliente, ni las sesiones son una tecnología complicada, accesible sólo a los elegidos. Son, simplemente, modos de almacenamiento de pares nombre-valor, así como de matrices si procede, que resultan muy útiles.

## 9.1 COOKIES

Una cookie (“galletita”, en inglés) es un archivo de texto que contiene el nombre de una variable y su valor. Este archivo se almacena en el ordenador cliente durante la navegación por Internet y puede estar disponible para recuperar el valor de la variable en posteriores navegaciones. Recuerde esto: una cookie solo puede contener texto plano, no programas ejecutables ni nada potencialmente peligroso. Cuando el usuario se conecta a Internet y entra en nuestro sitio, el código en el servidor es el encargado de almacenar la cookie en el cliente.

Y, ¿para qué queríamos almacenar una variable en el ordenador del cliente? Puede haber tantas respuestas a esta pregunta como usted sea capaz de imaginar. Quizás uno de los usos más populares es el del idioma del propio usuario. Usted crea un sitio web destinado a visitantes de diversas nacionalidades. Para ello crea una página y luego hace una copia en la que cambia los textos a otro idioma. Tiene, por tanto, dos copias de la página. Por ejemplo, una en español y otra en inglés. Si un usuario se conecta por primera vez a su sitio, tiene opción a elegir en qué idioma quiere ver los contenidos. El nombre del idioma elegido queda almacenado en una variable en el propio ordenador del cliente (una cookie). Ahora suponga que el internauta se vuelve a conectar a su sitio, digamos, al día siguiente. El servidor encuentra la cookie, lee el idioma que eligió el usuario y le remite, directamente, a la página en dicho idioma. Esto crea un efecto de personalización que contribuye a que la persona se sienta cómoda visitando el sitio.

Veamos cómo usar este mecanismo. En primer lugar necesitamos un script que determine si el usuario se conecta a nuestro sitio por primera vez, o si ya se ha conectado antes, es decir, si en su ordenador no existe ninguna cookie nuestra que almacene su idioma o si ya está grabada. En caso de que no exista la cookie, deberá cargarse una página donde se le pida al usuario su idioma y deberá crearse la cookie. En el caso de que ésta ya exista, deberá leerse y redireccionalar al usuario a la página en su propio idioma. El mecanismo responde al esquema de la figura 9.1.



Figura 9.1

Analice el esquema. Vea que, en el acceso al sitio por la página principal existe un script que verifica si el cliente que se ha conectado tiene grabada una cookie con una variable relativa al idioma del usuario. Si no se encuentra esta variable, se llega a una página donde se le pide que elija su idioma. Desde ahí, se salta a otra página donde se graba la variable *y*, por fin, a la página de contenidos elegida. Si existe la cookie, se lee la variable *y*, en función de su valor, se salta a la página elegida.

Cuando un script debe leer una variable que está grabada en una cookie se recurre a una matriz propia de PHP, llamada `$_COOKIE[]`. Se trata de una matriz asociativa cuyo índice es el nombre de la variable que esperamos poder leer desde la cookie del cliente. En el ejemplo de las páginas con varios idiomas, el idioma del cliente podría, por ejemplo, almacenarse en una variable llamada `$idiomaUsuario`, con lo que para leer dicha variable desde una cookie usaríamos una llamada como la siguiente: `$_COOKIE["idiomaUsuario"]`.

ARGUMENTO	DATO	USO
Nombre de la variable.	Cadena	Este parámetro es obligatorio y se usará como índice de la matriz asociativa <code>\$_COOKIE[]</code> para recuperar el contenido.
Valor de la variable.	Cadena	El valor de la variable. Si no se pone, la función elimina la cookie en el equipo del cliente.
Fecha de expiración.	Entero	Se usa para determinar el tiempo durante el que la cookie es válida. Si se intenta leer la cookie después del plazo establecido, no se puede y hay que grabarla de nuevo. Si no se incluye este parámetro, la cookie expira al cerrar el navegador.
Dominio	Cadena	El dominio desde el que se leerá la cookie. Si no se incluye, se puede leer desde cualquier dominio.
Ruta de acceso.	Cadena	La carpeta, dentro del servidor, desde la que se leerá la cookie. Si no se incluye, se puede leer desde cualquier carpeta.
Página segura.	Booleana	Si esta variable vale true, sólo se puede leer la cookie desde una página segura ( <code>https</code> ).

Figura 9.2

Para grabar una cookie en el ordenador del cliente usamos la función *setcookie()*. Esta puede recibir hasta seis argumentos, aunque sólo el primero es obligatorio, siendo los otros cuatro opcionales. Los seis parámetros que se le pueden pasar a esta función aparecen en la tabla de la figura 9.2.

Para aclarar cómo funciona todo esto vamos a ver unos listados que responden al esquema de la figura 9.1. No cargue todavía ninguno de estos scripts en su navegador. Primero analicemos su funcionamiento. Empezaremos por el que corresponde a la página principal, donde se comprueba si ya existe la cookie. En este ejercicio lo hemos llamado **usoDeCookies.php**, para darle el sentido propio del ejemplo. En un sitio real este código estaría incluido en index.php, o en default.php (dependiendo del servidor) que es el nombre que tiene que tener la página cero de cualquier sitio en Internet.

```
<?php
// Se comprueba si existe la cookie.
if (!$COOKIE["idiomaUsuario"]){
    // Si no existe, se determina como página la destinada
    a elegir el idioma.
    $pagina="pedirIdioma.htm";
} elseif ($_COOKIE["idiomaUsuario"]=="sp"){
    // Si existe la cookie y el valor de la variable es
    "sp" se irá a la página en español.
    $pagina="spanish.htm";
} else {
    // Si el valor no es "sp" se irá a la página en inglés.
    $pagina="english.htm";
}
?>
<html>
    <head>
        <script language="javascript"
type="text/javascript">
            // Se redirige a la página seleccionada.
            location.href=<?php echo($pagina); ?>;
        </script>
    </head>
</html>
```

Observe, en las líneas resaltadas, cómo se intenta leer el valor de la variable `$_COOKIE["idiomaUsuario"]`. Si la cookie no existe, se fija el nombre de la página a la que saltaremos como la destinada a preguntarle al usuario su idioma. Si existe la cookie y el valor de la variable es "sp", se irá a la página en español. Si no, se irá a la página en inglés. He diseñado este ejercicio de modo que sólo incluya dos posibilidades de idioma, para simplificarlo, pero podría haber más.

El listado de la página que le pregunta el idioma al usuario es muy simple. Lo he llamado **pedirIdioma.htm**:

```
<html>
  <head>
    <title>Pedir el idioma del usuario</title>
    <script language="javascript" type="text/javascript">
      function enviarIdioma (idioma){
        location.href = "grabarCookie.php?idiomaUsuario
= "+idioma;
      }
    </script>
  </head>
  <body>
    <a href="javascript:enviarIdioma('sp');">Ver la
versión del sitio en español</a>
    <br>
    <a href="javascript:enviarIdioma('en');">View the
english version of the site</a>
  </body>
</html>
```

Como ve, sólo se trata de dos enlaces convencionales. Ambos llaman a la misma función JavaScript, cada uno con un argumento que será el valor que represente al idioma elegido. La función sólo toma ese argumento y se lo entrega al script encargado de grabar la cookie en el equipo cliente, cuyo nombre es **grabarCookie.php** y cuyo listado aparece a continuación:

```
<?php
// Se graba una cookie con una validez de 24 horas.

setcookie("idiomaUsuario",$idiomaUsuario,time()+86400);
?>
<html>
  <body>
    <script language="javascript"
type="text/javascript">
      // Se regresa a la página principal.
      location.href="usoDeCookies.php";
    </script>
  </body>
</html>
```

Observe la línea resaltada. Esta es la que graba la cookie. El primer parámetro establece el nombre de la variable. El segundo establece el valor que tendrá la variable. Este valor se obtiene desde la URL, tal como es invocado este

script desde pedirIdioma.htm. El tercer valor es la fecha de caducidad de la cookie. Se ha tomado el momento actual (cuando se ejecuta el script) y se le han sumado 24 horas (86.400 segundos). Si el usuario vuelve a visitar el sitio antes de que hayan transcurrido las veinticuatro horas, se le redireccionará a la página en su idioma. Si vuelve a entrar pasado el plazo establecido (o desde otro ordenador), se le volverá a preguntar su idioma. Normalmente se establece un plazo pensando en la posible frecuencia de visitas. Si no se hubiese especificado el plazo de caducidad, la cookie expiraría al cerrar el navegador, con lo que no tendría mucho sentido en este ejemplo.

Y ahora sí. Cargue en su navegador el script **usoDeCookies.php**. Como es la primera vez que lo carga y su cookie aún no existe, se le pasará a la página donde se le pregunta su idioma, que aparece en la figura 9.3.

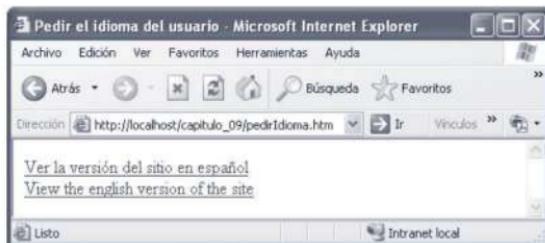


Figura 9.3

Pulse, por ejemplo, en el enlace correspondiente a la versión en español. Eso le llevará a la página encargada de grabar la cookie. Desde ahí saltará a la página principal y, como ya existe la cookie, irá a la página en español, tal como se ve en la figura 9.4.



Figura 9.4

Si ahora cierra su navegador y, a continuación, lo vuelve a abrir cargando el script **usoDeCookies.php** le reconducirá, directamente, a la figura 9.4. Durante 24 horas será así. Pasado el plazo, volverá a ver la figura 9.3. Si quiere eliminar la cookie antes de que se cumpla el plazo, deberá ejecutar el script **borrarCookie.php**, cuyo listado aparece a continuación:

```
<?php  
    setcookie("idiomaUsuario");  
?>
```

Como ve, es muy simple. En la página no verá usted nada (queda totalmente en blanco), ya que no hay ningún contenido visualizable. Lo que hacemos es “establecer” la cookie con el parámetro obligatorio del nombre de la variable... y sin ningún otro. Esto elimina la cookie en el cliente. Si ejecuta este script y, a continuación vuelve a ejecutar **usoDeCookies.php** se encontrará, de nuevo, con la página de la figura 9.3. Puede visitar el portal de contactos <http://www.todoligues.com> que hace uso de esta técnica para almacenar el idioma del usuario.

Sin embargo, aún podemos mejorar esto. Cuando se graba una cookie, como en este caso, para que expire a las 24 horas, eso es, exactamente, lo que sucede. Si el usuario entra de nuevo en el plazo establecido, la cookie funciona, pero pasado ese plazo, la cookie expira indefectiblemente, ya que no se renueva de modo automático. Tendríamos que contar con un mecanismo que, al entrar dentro del plazo establecido, renovara la cookie por otro lapso similar. Observe el listado del siguiente script, llamado **usoDeCookiesRenovables.php**:

```
<?php  
    // Se comprueba si existe la cookie.  
    if (!$_COOKIE["idiomaUsuario"]){  
        // Si no existe, se determina como página la destinada  
        // a elegir el idioma.  
        $pagina="pedirIdioma.htm";  
        } elseif ($_COOKIE["idiomaUsuario"]=="sp"){  
            /* Si existe la cookie y el valor de la variable es  
            "sp" se irá a la página en español. Además, se vuelve a  
            grabar la cookie por otras 24 horas. */  
            setcookie ("idiomaUsuario","sp",time()+86400);  
            $pagina="spanish.htm";  
        } else {  
            /* Si el valor no es "sp" se irá a la página en inglés.  
            Además, se vuelve a grabar la cookie por otras 24 horas. */  
            setcookie ("idiomaUsuario","en",time()+86400);  
            $pagina="english.htm";  
        }
```

```
?>
<html>
  <head>
    <script language="javascript"
type="text/javascript">
      // Se redirige a la página seleccionada.
      location.href=<?php echo($pagina); ?>;
    </script>
  </head>
</html>
```

Observe, en los resaltes, que si la cookie ya existe vuelve a ser grabada de nuevo, prolongando, así, su periodo de validez.

Hasta ahora sólo hemos hablado de los tres primeros argumentos. Si establecemos el cuarto, que corresponde al nombre de dominio, la cookie sólo podrá ser leída desde el dominio que se especifique en este parámetro. Esto no es aconsejable si piensa que puede cambiar el nombre de su dominio en el futuro.

Si establece un directorio en el quinto parámetro, el script que lee la cookie deberá estar en dicho directorio en el servidor, o no se podrá leer la cookie.

Por último, si establece el sexto argumento con el valor true, la cookie sólo podrá ser leída por un script al que se acceda mediante una capa segura (protocolo https://). Esto es importante si la cookie almacena algún valor "sensible", tal como un número de tarjeta de crédito o similar (en una tienda virtual, por ejemplo). Grabar datos potencialmente delicados en cookies es muy desaconsejable.

También podemos grabar cookies que contengan una matriz. Por ejemplo, suponga el siguiente fragmento de código:

```
$matriz[0] = "Primer elemento";
$matriz[1] = "Segundo elemento";
$matriz[2] = "Tercer elemento";

setcookie ("matriz", $matriz, time() + 86400);
```

Con esto habremos grabado una matriz en una cookie. Para recuperar un elemento dado al leer la cookie usaremos lo siguiente:

```
$primerValor = $_COOKIE["matriz"][0];
```

Una cosa importante. Cuando utilice la función setcookie (), debe colocarla en su script *antes de que se produzca ninguna salida hacia la página*. Es decir,

antes de cualquier sentencia echo (), print () , printf () o cualquier dato visualizable. Además, debe ir *antes de ninguna etiqueta HTML*. Esto se debe a que esta función trabaja a nivel de cabeceras, un tema sobre el que hablaremos más adelante. La última versión de PHP 5 emplea determinadas técnicas que permiten usar setcookie () después de llevar a cabo algunas salidas, pero no se acostumbre a ello, pues esto se obtiene a costa de sacrificar rendimiento. Establezca sus cookies en el momento adecuado: al principio del script. La solución óptima pasa por usar un script específico para la grabación de las cookies, como hemos visto aquí.

## 9.2 SESIONES

La gestión de sesiones es un sistema para almacenar pares nombre-valor en el lado del servidor. El uso de sesiones en lugar del de cookies está más definido cuando las variables deben permanecer a disposición de los scripts que emplea el usuario durante la navegación por distintas páginas del mismo sitio. Suponga que navega por un sitio en el que todas las páginas (o muchas de ellas) deben estar personalizadas para cada usuario en concreto. Quizás el ejemplo más clásico y relevante sea una tienda en Internet. El usuario navega por las páginas de distintos productos, viendo cuáles le interesan y seleccionando uno aquí y otro allí para su compra. Los productos elegidos deben recordarse de alguna manera de una página a otra, para que, cuando el usuario cierre la cesta le podamos mostrar una relación y el importe total.

Dado que HTTP es un protocolo sin estados, es decir, no conserva memoria de una página a otra, como ya sabemos, cada compra individual debe almacenarse en un archivo. Sin embargo, es lógico suponer que el usuario no necesita que el recuerdo de esa compra permanezca disponible para usos sucesivos del sitio, así que no es necesario almacenar una cookie en el ordenador del cliente. También podemos considerar otro ejemplo muy clásico: los típicos portales de contactos, donde cada página aparece personalizada con el nombre de usuario de la persona que entra. Así, cuando visita la ficha de otro usuario o le envía un e-mail, queda constancia de quién ha hecho esa visita o quién firma el mensaje. Puede ver un ejemplo de este uso en el portal <http://www.todoligues.com>, donde usted entra con su nombre de usuario y su contraseña (tras haberse registrado, desde luego), y dicho nombre de usuario encabeza cada una de las páginas que visita.

La forma de almacenar los datos necesarios es mediante unos archivos de servidor que se reconocen porque su nombre empieza con `sess_` seguido del *número de identificación de la sesión*, también llamado *número de sesión* o *session id*. El session id se genera de forma aleatoria para cada conexión de un usuario al sitio. Es una cadena formada por un elevado número de dígitos alfanuméricos, de modo que cada usuario, al conectarse, crea un nuevo archivo de

sesión, único para él, para esa sesión de trabajo. Cuando abandona el sitio (dirigiéndose a otro, o cerrando el navegador), el archivo ya es innecesario. Si el usuario se vuelve a conectar a nuestro sitio se creará otra sesión.

Para que un script pueda trabajar con sesiones es necesario que incluya, antes de usar ningún dato relativo a la sesión, la función `session_start()`. Esta función no recibe argumentos y devuelve un valor booleano true, salvo que se haya producido algún error. Tiene dos misiones. Si es la primera vez que se ejecuta en el sitio para una visita en concreto, crea un nuevo archivo de sesión, con un session id exclusivo para ese uso en concreto. Si ya está creada la sesión, la “abre”, es decir, facilita el acceso a las variables que haya registradas.

Para registrar una variable se emplea la función `session_register()`, que recibe tantos argumentos como sean necesarios. Cada uno de ellos es el nombre de una variable o matriz que deseamos que esté disponible, a través de la sesión, para otras páginas del sitio.

Las variables de sesión están disponibles a través de la matriz asociativa de PHP `$_SESSION[]`, que recibe como índice el nombre de la variable que nos interesa recuperar. También podemos usar la matriz `$HTTP_SESSION_VARS[]`, pero esta requiere que esté activada la directriz `register_globals` en el fichero de configuración `php.ini` (vea el Apéndice A para saber más sobre este tema). Además, esta última matriz es más antigua que aquella. En la actualidad se mantiene por razones de compatibilidad con scripts de PHP 4, pero no se garantiza que siga vigente en futuras versiones del lenguaje. Utilice `$_SESSION[]`. De este modo se asegurará de que sus scripts funcionen en el futuro.

Una cosa importante. Como ya hemos mencionado, la sesión en curso se identifica mediante el session id. Este es un valor que se almacena en una constante del sistema que, por defecto, se llama `PHPSESSID`. El nombre puede ser cambiado en el archivo de configuración de PHP, aunque mi recomendación es que no lo cambie. La función `session_name()`, que no recibe argumentos, nos devuelve el nombre de esta constante. Por otra parte tenemos la función `session_id()`, también sin parámetros, que nos devuelve el valor de la constante de identificación, es decir, el session id de la sesión en curso. Estas dos funciones son muy necesarias ya que, para poder leer las variables almacenadas en un fichero de sesión es necesario que esta constante, con su valor, pase de una página a otra. Para ello, podemos usar un campo oculto de un formulario, que se puede enviar mediante GET o POST, o podemos incluir esta información en la barra de direcciones.

Aunque hay más funciones que podemos emplear para optimizar la gestión de sesiones, con las que ya tenemos podemos hacer un uso básico, aunque muy eficiente, de las sesiones. Para comprender su uso hemos creado un sitio ficticio,

muy simple, pero que ilustra perfectamente el uso de esta tecnología. El sitio en cuestión responde al esquema de la figura 9.5.

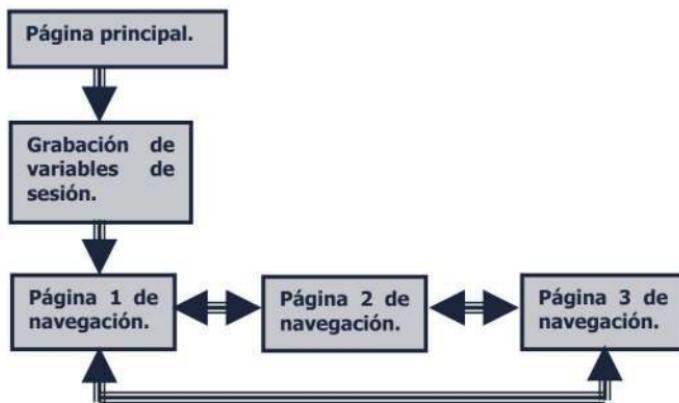


Figura 9.5

En primer lugar, tenemos una página principal, donde le vamos a pedir al usuario dos datos, a modo de ejemplo, que deberán estar disponibles durante toda la navegación por el sitio. He elegido un nombre de usuario y un color para texto. Podríamos haber elegido otros datos. Sólo es un ejemplo. A continuación, tenemos una página donde dichos datos se graban en las correspondientes variables de sesión. Esta página es transparente al usuario, de forma que no tiene por qué percibir nada. Por último, tenemos tres páginas que, en un sitio real, serían de contenidos. En ellas hemos hecho que se muestre el nombre del usuario y, en el color elegido, la página en la que está navegando en cada momento. Cada una de estas páginas tiene sendos enlaces que permiten el acceso a las otras páginas del sitio.

A continuación, vamos a reproducir los listados de las páginas de este sitio. No las cargue en el navegador todavía. Primero veamos los códigos y luego los probaremos. El script usado en la página principal se llama, en este ejemplo, **usoDeSesionesInicio.php**, y es el siguiente:

```
<?php  
/* Se inicia la sesión. Como no hay ninguna sesión  
activa, se crea en este momento, asignándole, como  
session_id, una cadena alfanumérica formada por guarismos
```

```
aleatorios. El nombre del fichero de sesión en el servidor
será sess_, seguido de dicha cadena. */
    session_start ();
    /* Se indica que variables van a formar parte del
    archivo de la sesión, aunque, de momento, no tienen valor
    alguno asignado.*/
    session_register ($nombreDeUsuario,
$colorElegido);
?
<html>
    <head>
        <title>Uso de sesiones</title>
    </head>
    <body>
        <!-- Se crea un formulario donde se van a grabar dos
        datos: el nombre del usuario y el color que elige de una
        lista. Los campos que recogen estos datos tienen los mismos
        nombres que las variables de sesión. Este formulario se envía
        al script que tiene la misión de almacenar los valores de
        estos campos en las variables de sesión. Este script es
        transparente al usuario.-->
        <form name="fPrincipal" id="fPrincipal"
method="post" action="paginaCeroDeSesiones.php">
            Nombre de usuario:
            <input type="text" name="nombreDeUsuario"
id="nombreDeUsuario" value="">
            <br>
            Color:
            <select name="colorElegido" id="colorElegido">
                <option value="#FF0000">Rojo</option>
                <option value="#00FF00">Verde</option>
                <option value="#0000FF">Azul</option>
            </select>
            <br>
        <!-- Además, hay un campo oculto que se usará para
        enviar la constante de session id -->
            <input type="hidden" name="<?php echo
(session_name()); ?>" value="<?php echo (session_id()); ?>">
            <input type="submit" name="mandar"
value="ACCEDER">
        </form>
    </body>
</html>
```

Vea el uso de las funciones `session_start()` y `session_register()` que hemos comentado anteriormente, y que aparecen en las dos primeras líneas resaltadas. Lea los comentarios adjuntos que le explican el por qué de estas líneas. Observe, en la

tercera línea resaltada, cómo pasamos la constante de identificación de la sesión en un campo oculto dentro del formulario. Éste carga contra el script cuyo código aparece a continuación y cuyo nombre es **paginaCeroDeSesiones.php**:

```
<?php
// Se define el salto de línea
define ("salto","<br>\n");
/* Como esta página recibe, en este caso a través de un
campo oculto, una constante que identifica una sesión, la
función session_start ya no crea una sesión, si no que abre
la sesión vigente.*/
session_start();
/* Los dos campos del formulario son asignados a las
variables de sesión. */
$_SESSION["nombreDeUsuario"]=$_POST["nombreDeUsua
rio"];
$_SESSION["colorElegido"]=$_POST["colorElegido"];
?>
<html>
<head>
<script language="javascript" type="text/javascript">
/* La siguiente función, que se ejecuta a la carga de
la página, llama a la primera página de contenidos,
pasándole, como campo oculto, la constante de id. de la
sesión. */
function mandar(){
    document.f0.submit();
}
</script>
<head>
<!-- Hacemos que, a la carga de la página, se envíe un
campo oculto con la constante PHPSESSID -->
<body onLoad="javascript:mandar();">

<!-- El formulario contiene la constante PHPSESSID en
un campo oculto. -->
<form name="f0" id="f0" method="post"
action="paginaUno.php">
<input type="hidden" name="<?php echo
(session_name()); ?>" value="<?php echo (session_id()); ?>">
</form>
</body>
</html>
```

Como ve, este script sólo toma los datos procedentes del formulario anterior y los asigna a las correspondientes variables de sesión. A continuación envía un formulario, de modo automático mediante un JavaScript que se ejecuta a

la carga de la página, contra la primera página de navegación, con el identificador de sesión en un campo oculto. Este identificador deberá pasar siempre entre las páginas en las que deseemos poder disponer de las variables de sesión. El listado de la primera página de contenidos, cuyo nombre es **paginaUno.php** es el siguiente:

```
<html>
<head>
<title>Uso de sesiones</title>
</head>
<body>
<?php
// Se define el salto de línea.
define ("salto","<br>\n");
/* Se abre la sesión cuyo id. se ha recibido. */
session_start();
/* Se muestra el nombre del usuario y, en el color
elegido, la página en la que estamos.*/
echo ("El nombre de usuario es:
".$_SESSION["nombreDeUsuario"].salto);
echo ("El color elegido es: <h1><font
color='".$._SESSION["colorElegido"].">PÁGINA
UNO</font></h1>".salto);
?>
<!-- A continuación se proveen dos enlaces, para
acceder a las restantes páginas de contenidos pasando, en la
URL, la constante de identificación de sesión. --&gt;
&lt;a href="paginaDos.php?&lt;?php echo (session_name());
?&gt;=&lt;?php echo (session_id()); ?&gt;">Ir a PÁGINA 2</a>
<br>
<a href="paginaTres.php?<?php echo (session_name());
?>=<?php echo (session_id()); ?>">Ir a PÁGINA 3</a>
</body>
</html>
```

Observe que se reabre la sesión, con `session_start()` y, a partir de ahí, las dos variables que hemos definido como variables de sesión están disponibles. Los listados de **paginaDos.php** y **paginaTres.php** no los reproduczo aquí por su similitud con el que tiene a la vista.

Veamos cómo funciona este conjunto de scripts. En primer lugar, cargue en el navegador **usoDeSesionesInicio.php**. Verá un formulario con dos campos, en el que se le pide su nombre de usuario y que elija un color. Al enviar el formulario mediante el botón habilitado al efecto, estos dos datos, junto con el campo oculto que contiene el session id, viajan al servidor, al script **paginaCeroDeSesiones.php**.

En este script se asignan las variables del formulario a las correspondientes variables de sesión, tras haber iniciado la sesión con `session_start()`. A partir de ese momento, estas variables quedan disponibles para todas las páginas, mientras dure la visita. El script nos redirecciona automáticamente a `páginaUno.php`.

Los scripts de `páginaUno.php`, `páginaDos.php` y `páginaTres.php` muestran el nombre del usuario y, en el color elegido, muestran el nombre de la página en la que nos encontramos. Como ve, estos datos se obtienen de las correspondientes variables de sesión. Además, cada página posee dos enlaces que le permiten acceder a las demás. Estos enlaces pasan la constante `PHPSESSID` en la URL. De este modo, usted puede ver que, tanto si esta constante pasa por la URL, como si pasa mediante un campo de formulario, al ejecutarse `session_start()` se cargan en memoria las variables de la sesión en curso.

El identificativo de sesión puede ser almacenado en una cookie en el equipo del cliente. En ese caso no es necesario transferirlo de página a página. Sin embargo, mi recomendación es que, ya que usa los recursos del servidor, no emplee este sistema, ya que si el navegador del cliente tiene inhabilitadas las cookies, no le funcionará la gestión de la sesión. No obstante, PHP cuenta con un mecanismo llamado ***enable-trans-sid***. Desde la versión 4.2.0 este mecanismo está activado en la compilación del intérprete, por lo que no es necesario transmitir expresamente el `PHPSESSID`. Si en su servidor no está activado en la compilación, puede usar la directiva `session.use-trans-id` del archivo de configuración (vea el Apéndice A, para saber lo que necesita sobre este archivo). De todos modos, si en una instalación en particular no es posible usar sesiones con transferencia automática de session id (que es lo que significa trans-id), siempre puede usar el `PHPSESSID` como hemos visto en el ejemplo.

PHP cuenta con algunas funciones más para la gestión de sesiones. Quizás una de las más interesantes es `session_is_registered()`. Esta función recibe, como argumento, el nombre de una variable de sesión, sin el signo \$ y entre comillas. Si la variable está realmente registrada como tal variable de sesión, devuelve el valor true. En caso contrario, devuelve false. Esta función es muy útil para asegurarnos de que un usuario acceda al sitio por la página principal. Lo que hacemos es registrar las variables de sesión en dicha página. En las demás, después de `session_start()`, comprobamos si una variable de sesión está registrada. En caso de no estarlo, redirigimos al usuario a la página principal. Siguiendo con los scripts del ejemplo que hemos usado aquí, vemos que en el archivo principal (`usoDeSesionesInicio.php`) se registran las variables de sesión. Esto es un buen hábito. Siempre deben registrarse en el archivo principal de un sitio todas las variables de sesión, incluso aunque no vayan a usarse en algunas de las páginas. En los demás archivos, justo después de abrir la sesión, incluiremos el siguiente fragmento:

```
if (! Session_is_registered ("nombreDeUsuario")) {  
    ?>  
    <script language="javascript"  
type="text/javascript">  
        location.href="usoDeSesionesInicio.php";  
    </script>  
    <?php  
}
```

Como ve, se comprueba si la variable de sesión está registrada. En caso de no estarlo, se abandona el modo PHP para que JavaScript envíe al usuario a la página principal del sitio.

La función *session\_unregister ()* recibe el nombre de una variable de sesión y la borra del fichero de la sesión actual. A partir de ese momento, esta variable ya no existe como tal, y no puede ser usada.

La función *session\_destroy ()* elimina el archivo de sesión. No recibe argumentos. Esta función puede ser colocada en un script que deba ejecutarse cuando el usuario haga clic en un enlace de salida del sitio. Sin embargo, teniendo en cuenta que la mayoría de los usuarios salen de un sitio simplemente cerrando el navegador o, incluso, apagando el ordenador sin más, es difícil que este script llegara a ejecutarse nunca. Yo, personalmente, nunca la he usado.

La función *session\_unset ()* borra el contenido de todas las variables de sesión. Esta función no recibe argumentos. De todos modos, dado que usamos la matriz *\$\_SESSION* la forma correcta de eliminar el contenido de una variable de sesión es *unset (\$\_SESSION ["variableParaLimpiar"]);*. El uso de *session\_unset ()* cuando se emplea esta matriz puede producir errores.

## CAPÍTULO 10

# LA COMUNICACIÓN WEB

---

---

En este capítulo vamos a conocer algunas prestaciones de PHP que, por su propia naturaleza, son especialmente aptas para su integración en comunicaciones web (HTTP, FTP, etc.). En realidad, no es ninguna sorpresa, ya que PHP es un lenguaje de script para trabajar en una arquitectura cliente-servidor. Sin embargo, los conceptos que vamos a conocer aquí, aún con ser muy simples, nos resultarán extremadamente útiles para integrarlos en algunas de las páginas que desarrollemos en nuestra trayectoria profesional. En este capítulo conocerá más variables y matrices propias de PHP, y entenderá el cómo y el por qué de algunos conceptos que hemos avanzado en anteriores capítulos, sin entrar en mayores explicaciones.

### 10.1 LAS CABECERAS

Cuando se navega por Internet (así como cuando se usa el correo electrónico, o se suben programas a un servidor, etc.) existe una doble comunicación. En primer lugar está la **solicitud**, cursada por el cliente al servidor. Así, cuando usted se conecta a cualquier página de Internet, su navegador envía un requerimiento al servidor para que le proporcione los contenidos de dicha página. El servidor escucha su petición y, si es posible, le remite los contenidos solicitados. Es la segunda comunicación, llamada **respuesta**. Siempre es de ese modo. Primero se produce una solicitud por parte del cliente, seguida de una respuesta por parte del servidor. A continuación, se cierra la comunicación. Es decir, el servidor, cuando ha enviado el recurso solicitado al cliente, cierra la comunicación, hasta que el cliente haga otra solicitud. *Por eso HTTP es un protocolo sin estados.* Ésta es la principal diferencia operacional entre una aplicación web y un programa de usuario. Y por esta razón son necesarias las técnicas que vimos en el capítulo

anterior cuando hay que pasar valores de una página a otra. En cada comunicación (solicitud y respuesta) existen dos partes claramente diferenciadas: la **cabecera** y el **cuerpo o contenido** del mensaje. Ambas se separan con una línea en blanco, razón por la cual no puede haber líneas sin contenido en la cabecera. De las cabeceras nos vamos a ocupar en este apartado.

Antes de seguir adelante vamos a aclarar el concepto de cabecera. En las solicitudes, la cabecera contiene datos acerca del navegador del cliente, el sistema operativo, etc. El cuerpo del mensaje es la solicitud en sí que el cliente ha efectuado. El cuerpo de la solicitud es lo que se conoce como un **URI (Universal Resource Identifier, Identificador Universal de Recursos)**. El URI puede estar formado por la URL del sitio al que queremos acceder y, en su caso, por la ruta. Además, puede haber pares nombre-valor a continuación. Por ejemplo, suponga que usted accede a una página desde un formulario enviado mediante GET. En la barra de direcciones de su navegador podría aparecer algo como lo siguiente:

**http://www.misitio.com/main.php?nombre=Juan&apellido=Garrido**

Todo lo que hay a continuación del signo "?" se conoce como **Query String**, o **Cadena de la Consulta**. Recuerde que una solicitud se genera cada vez que se le pide algo al servidor. Esto significa no sólo que usted escriba algo en la barra de direcciones. También es una solicitud cuando usted envía un formulario o pulsa un enlace. En las respuestas del servidor (cuando se le envía la página solicitada), el término cabecera no se refiere a la sección <head></head> que hay en cualquier página web. Esto forma parte del contenido del mensaje. La cabecera son unos datos previos a la propia página en sí, que informan al navegador de si se ha podido cursar la petición, la fecha, el tipo de documento, etc.

### 10.1.1 Las cabeceras de la solicitud

Vamos a empezar viendo las cabeceras de una solicitud para familiarizarnos con algunos conceptos. Para ello usaremos la función **getallheaders()**, que no recibe ningún argumento. Esta función recupera las cabeceras de la solicitud en una matriz asociativa. En cada elemento de esta matriz el nombre es el de una de las líneas de contenido de la cabecera y el valor es el que tiene asignada dicha linea. Para comprender mejor esto veamos el listado **cabecerasSolicitud.php**:

```
<html>
<head>
<title>Cabeceras de la solicitud</title>
</head>
<body>
```

```
<table width="400" border="4" cellpadding="4"
cellspacing="0">
    <tr><th colspan="2">
        <h2>Listado de cabeceras</h2>
    </th></tr>
    <tr><th
width="120">Cabecera</th><th>Valor</th></tr>
    <?php
    /* Se obtienen las cabeceras de la solicitud en una
matriz asociativa.*/
        $cabeceras=getallheaders ();
    /* Se muestra cada elemento de la matriz.*/
        foreach ($cabeceras as $nombre=>$valor) {
            ?
            <tr><td>
                <?php echo ($nombre); ?>
            </td><td>
                <?php echo ($valor); ?>
            </td></tr>
            <?php
        }
    ?
    </table>
    </body>
</html>
```

Observe, en la línea resaltada, el uso de la función que estamos estudiando. Al cargar este script en su navegador obtendrá el resultado de la figura 10.1.

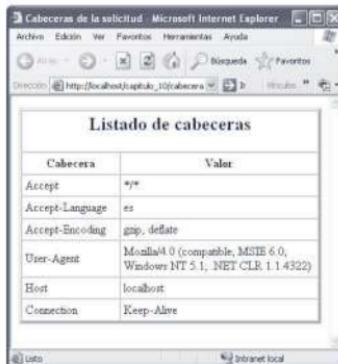


Figura 10.1

En la tabla que obtenemos vemos las cabeceras que tiene esa solicitud en concreto. Los nombres de las mismas, así como su significado aparecen en la tabla de la figura 10.2.

CABECERA	SIGNIFICADO
<b>Accept</b>	Muestra una lista con los tipos de recursos que el navegador es capaz de gestionar y mostrar al usuario, es decir, texto, imágenes, clips de vídeo y/o audio, animaciones, etc. En el ejemplo obtenemos <code>*/*</code> . Eso quiere decir que el navegador que estoy usando tiene todos los plugins necesarios para gestionar cualquier contenido de hipermedia.
<b>Accept-Language</b>	Muestra el idioma en el que está configurado el navegador.
<b>Accept-Encoding</b>	Muestra los tipos de codificación soportados.
<b>User-Agent</b>	Muestra información acerca del navegador y el sistema operativo.
<b>Host</b>	El nombre del servidor al que nos conectamos.
<b>Connection</b>	El estado de la conexión. Se mantiene abierta (Keep-Alive) porque en el momento de generarse esta matriz, la página todavía no ha sido enviada al navegador. Cuando lo sea, será el servidor quien cierre la conexión.

Figura 10.2

Éstas son sólo algunas de las cabeceras que pueden preceder a la solicitud. Si recuerda el capítulo 2, vimos una función de PHP que da información muy completa acerca de la instalación del intérprete en nuestro servidor. Se trata de `phpinfo()`. En aquel momento no entrámos en detalles acerca de qué era lo que estábamos viendo. Cargue el script <http://localhost/pruebaPHP.php> (recuerde que debe tenerlo en la carpeta que ha definido como servidor local. Revise el capítulo 2 si duda). Una vez cargado busque el apartado **HTTP Headers Information** y verá una doble tabla con una lista de las cabeceras que hay.

Si sólo desea recuperar una o dos cabeceras concretas, puede usar algunas de las variables propias del intérprete para ello, en lugar de la función `getallheaders()`. Observe el listado de `algunasCabeceras.php`:

```
<html>
  <head>
    <title>Algunas cabeceras de la solicitud</title>
```

```
</head>
<body>

<?php
    define ("salto", "<BR>\n");
    echo ("ALGUNAS CABECERAS".salto);
    echo ("El valor de <b>\$HTTP_USER_AGENT</b> es
<b>$HTTP_USER_AGENT</b>".salto);
        echo ("El valor de <b>\$HTTP_ACCEPT_LANGUAGE</b>
es <b>$HTTP_ACCEPT_LANGUAGE</b>".salto);
    ?>

</body>
</html>
```

Vea, en las líneas resaltadas, cómo hemos hecho uso de dos de las variables del lenguaje para recuperar sendas líneas de contenido de la cabecera de petición. El resultado es el de la figura 10.3.

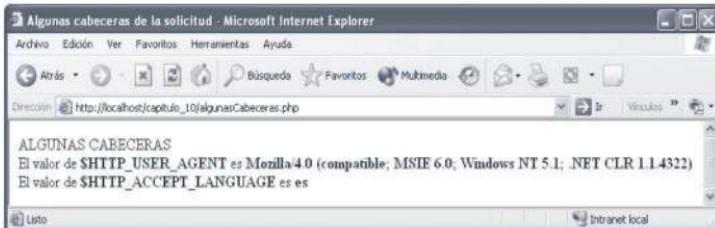


Figura 10.3

En general, si desea recuperar alguna cabecera mediante este sistema, el nombre de la variable es `$HTTP_` seguido del nombre de la cabecera, en mayúsculas, y sustituyendo los guiones normales (si los hay) por guiones bajos.

Si ejecuta de nuevo el script `pruebaPHP.php` que tiene en la carpeta raíz de su servidor, busque bajo el encabezamiento **Apache Environment** la lista de variables que le proporciona PHP a este respecto. También puede encontrarlas como elementos de la matriz `$_SERVER[]` (equivalente al antiguo `$HTTP_SERVER_VARS[]`), bajo el encabezamiento **PHP Variables**.

Esta información es usada por los webmasters para determinar datos acerca de la plataforma desde la que accede cada cliente a fin de personalizar los contenidos para cada cliente específico.

```
</head>
<body>

<?php
    define ("salto", "<BR>\n");
    echo ("ALGUNAS CABECERAS".salto);
    echo ("El valor de <b>\$HTTP_USER_AGENT</b> es
<b>$HTTP_USER_AGENT</b>".salto);
        echo ("El valor de <b>\$HTTP_ACCEPT_LANGUAGE</b>
es <b>$HTTP_ACCEPT_LANGUAGE</b>".salto);
    ?>

</body>
</html>
```

Vea, en las líneas resaltadas, cómo hemos hecho uso de dos de las variables del lenguaje para recuperar sendas líneas de contenido de la cabecera de petición. El resultado es el de la figura 10.3.

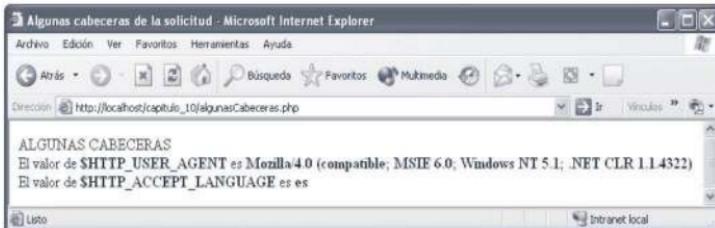


Figura 10.3

En general, si desea recuperar alguna cabecera mediante este sistema, el nombre de la variable es `$HTTP_` seguido del nombre de la cabecera, en mayúsculas, y sustituyendo los guiones normales (si los hay) por guiones bajos.

Si ejecuta de nuevo el script `pruebaPHP.php` que tiene en la carpeta raíz de su servidor, busque bajo el encabezamiento **Apache Environment** la lista de variables que le proporciona PHP a este respecto. También puede encontrarlas como elementos de la matriz `$_SERVER[]` (equivalente al antiguo `$HTTP_SERVER_VARS[]`), bajo el encabezamiento **PHP Variables**.

Esta información es usada por los webmasters para determinar datos acerca de la plataforma desde la que accede cada cliente a fin de personalizar los contenidos para cada cliente específico.

## 10.1.2 Las cabeceras de la respuesta

Las cabeceras de la respuesta son enviadas por el servidor al navegador interpretadas por este. Así pues, lo primero que intuimos es que nos ofrecen ciertas posibilidades de manipular el comportamiento del navegador. No se asuste. Esto no quiere decir que usted (ni nadie) pueda usar las cabeceras de la respuesta con fines maliciosos, tales como anular el antivirus, espesar contenidos del disco del cliente, ni nada así. Son manipulaciones totalmente inofensivas, destinadas a mejorar, o no, el comportamiento del navegador. Por ejemplo, tengamos en cuenta la primera línea de cabecera que se envía con la respuesta. Tiene el siguiente formato genérico:

**Protocolo CódigoDeEstado Descripción**

Por ejemplo, una cabecera típica es la siguiente:

**HTTP/1.1 200 OK**

La primera parte muestra que la comunicación es en HTTP 1.1, que es la versión actual del protocolo. A continuación, se recibe el código de estado, que es un número de tres cifras y que se refiere al resultado de la respuesta. El código 200 le indica al navegador que la respuesta ha sido correctamente enviada por el servidor. Esto lo vemos en la descripción del código (OK). Cuando el navegador recibe este código de estado (el 200), "sabe" que el documento ha sido correctamente enviado y lo muestra.



Figura 10.4

Para trabajar con las cabeceras de respuesta desde PHP contamos con la función **header()**, que según el argumento que reciba, enviará una determinada

cabecera al navegador. Por ejemplo, examine el listado del script **noEncontrado.php**:

```
<?php  
    header ("HTTP/1.1 404 No se encuentra");  
?>
```

Esta cabecera envía el famoso (y temido) código 404, que se da cuando solicitamos una página que no existe en el servidor. Al cargar el script obtendrá un resultado que seguro que ha visto más de una vez en sus navegaciones por Internet, tal como le muestra la figura 10.4.

La descripción del código de error que hemos puesto en castellano no tiene mayor trascendencia, como ve. Lo que el navegador interpreta es el código en si.

Recuerde que la función `header()` debe ser llamada antes que cualquier salida, ya sea mediante etiquetas HTML normales, líneas en blanco de un archivo, o desde PHP. Es un error muy común insertar código externo con `include()` o `require()` y terminar con espacios o líneas en blanco que son impresas antes de una llamada a `header()`. El mismo problema existe cuando se usa un archivo PHP/HTML único. Sin embargo, la cabecera que hemos enviado en el script **noEncontrado.php** es un caso excepcional. Una cabecera con un código de estado, como la que aparece en este script, es la primera en ser enviada al cliente, con independencia de su situación real. No obstante, acostúmbrase a poner la función `header()` al principio del script.

Los códigos de estado HTTP determinan, como acaba de comprobar, el comportamiento inicial del navegador. Estos códigos tienen siempre tres cifras y se agrupan dependiendo de la primera, según la tabla de la figura 10.5.

#### CÓDIGOS DE RESPUESTA HTTP

Código	Significado
<i>Ixx</i>	Informativo.
101	Se ha producido un cambio de protocolo.
<i>2xx</i>	Envío realizado con éxito.
200	Envío completado sin errores (es el código que acompaña a un resultado exitoso habitual).
204	Sin contenido.
<i>3xx</i>	Redireccionamientos.
302	Redirección a otro documento.

### CÓDIGOS DE RESPUESTA HTTP (cont.)

Código	Significado
<i>4xx</i>	<i>Error causado por el cliente.</i>
401	No está autorizado a ver esta página.
403	Contenido prohibido.
404	No se ha encontrado la página.
407	Se requiere un proxy.
408	Tiempo de espera agotado.
414	URI de solicitud demasiado largo.
<i>5xx</i>	<i>Error causado por el servidor.</i>
500	Error Interno.
503	Servicio no disponible.
505	Versión de HTTP no soportada.

Figura 10.5

Existen algunos otros códigos, pero los que aparecen aquí son los que más habitualmente se generan.

La función header () proporciona más posibilidades que enviar un determinado código de estado del protocolo al navegador. Por ejemplo, suponga que ha mudado su sitio a otro dominio. Antes su sitio estaba bajo el nombre, digamos, www.sitioantiguo.com, y ahora está en www.sitionuevo.com. Así pues, usted desea que los usuarios que no sepan del cambio y pretendan llegar a sus contenidos a través del dominio antiguo, sean redirigidos, automáticamente, al nuevo sitio. Para ello el fichero index.php del alojamiento del sitio antiguo será sustituido por el siguiente script:

```
<?php
    header("Location: http://www.sitionuevo.com/");
?>
```

Existen un par de cabeceras con las que es muy interesante contar cuando programamos páginas dinámicas. En estas páginas es habitual que los contenidos se actualicen con frecuencia. Eso puede suponer una interferencia con el caché de los navegadores, que pueden estar mostrándole al usuario una información caduca. Esto lo solucionamos con las siguientes líneas de cabecera al principio del script.

```
header ("Cache-Control: no-cache, must-revalidate");
header ("Expires: Mon, 26 Jul 1997 05:00:00 GMT");
```

La primera linea le dice al navegador que no atienda al caché y cargue la página directamente desde el servidor. La segunda establece una fecha, que debe ser anterior a la actual, como momento de expiración de la página. De este modo, se recoge todo el contenido procedente del servidor.

## 10.2 AUTENTICACIÓN

A la hora de pedirle a un usuario que se identifique, mediante su nombre y contraseña podemos usar un formulario para que introduzca estos dos datos. Sin embargo, si queremos darle a nuestro sitio un acceso más curioso, con aspecto de mayor profesionalidad, podemos recurrir a un uso muy particular de header(). Observe el listado del script **pedirDatos.php**, que aparece a continuación:

```
<?php  
    header('WWW-Authenticate: Basic realm="Solo  
clientes"');  
    header('HTTP/1.0 401 Restricted area');  
?>
```



Figura 10.6

La primera línea hace que el navegador muestre un cuadro en el que se le solicita al usuario su nombre y contraseña, tal como aparece en la figura 10.6. Seguro que usted ha visto este mecanismo funcionando en varios sitios y se ha preguntado cómo programarlo.

Observe, en el cuadro de diálogo que se muestra, cómo aparece la inscripción “Sólo clientes”, que tenemos escrita como texto en la primera línea del script.

Ya tenemos un mecanismo que permite pedirle al usuario su nombre y contraseña. Ahora, cuando el usuario escriba estos datos, tenemos que poder recuperarlos en el script para poder comprobar si son correctos. Esto se hace a través de dos variables del propio lenguaje: son **\$PHP\_AUTH\_USER** y **\$PHP\_AUTH\_PW**. Como habrá podido deducir de sus nombres, en la primera se almacena el login del usuario y en la segunda, su contraseña. Vea el script **mostrarDatos.php**:

```
<?php
    // Se comprueba si existe la variable de nombre de
    usuario
    if (!isset($PHP_AUTH_USER)){
        // Si no existe se activa el mecanismo que la solicita.
        header('WWW-Authenticate: Basic realm="Solo
    clientes"');
        header('HTTP/1.0 401 Restricted area');
    }
    define ("salto","<br>\n");
    // Se muestran los datos tecleados.
    echo ("El login tecleado es:
<b>$PHP_AUTH_USER</b>".salto);
    echo ("La contraseña tecleada es:
<b>$PHP_AUTH_PW</b>".salto);
?>
```

Al ejecutarlo, teclee, como nombre, “Mi nombre” y como contraseña “Mi clave” (sin las comillas). Pulse **Aceptar** y vea cómo se le muestran los datos tecleados en la página, gracias a las dos líneas que aparecen resaltadas en el listado.

Y ya sólo nos falta algún mecanismo para poder comparar los datos tecleados por el cliente con unas entradas predefinidas, de modo que podamos determinar si quien se ha conectado es un usuario legítimo o no. Si lo es, le daremos acceso a los contenidos. Si no está registrado, se lo indicaremos mediante un mensaje. La relación de usuarios registrados, así como sus contraseñas, pueden proceder de varias fuentes. Una base de datos es el sistema más potente y versátil,

pero también pueden estar en un fichero de disco, una matriz de memoria, etc. En la cuarta parte del libro aprenderemos a manejar bases de datos. De momento, veamos cómo autenticar a un usuario a partir de la información de una matriz. Observe el listado del script que aparece a continuación, llamado **autenticarUsuario.php**:

```
<?php
    // Se comprueba si existe la variable de nombre de
    usuario
    if (!isset($PHP_AUTH_USER)){
        // Si no existe se activa el mecanismo que la solicita.
        header('WWW-Authenticate: Basic realm="Solo
        clientes"');
        header('HTTP/1.0 401 Restricted area');
    }
    // Creamos el salto de línea.
    define ("salto","<br>\n");
    /* Establecemos la matriz de usuarios legítimos. En una
    aplicación real, estos datos procederían de una base de
    datos.*/
$matrizUsuarios=array("usuario_1"=>"clave_1","usuario_2"=>"cl
ave_2", "usuario_3"=>"clave_3");
    /* Se crea una variable con el valor false, que se
    usará para comprobar si se produce alguna coincidencia con la
    lista de usuarios registrados.*/
$autorizado=false;
    /* Se comprueban los datos introducidos con cada
    elemento de la matriz. Si hay alguna coincidencia se pone la
    variable autorizado en true y se abandona el bucle.*/
foreach ($matrizUsuarios as $login=>$pw){
    if ($login==$PHP_AUTH_USER || $pw==$PHP_AUTH_PW){
        $autorizado=true;
        break;
    }
}
/* Se comprueba si el usuario está autorizado.*/
if ($autorizado){
    echo ("Bienvenido a los contenidos de la página.");
} else {
    echo ("No estás autorizado a visitar este sitio.");
}
?>
```

Tenemos una matriz asociativa en el propio script que contiene los datos de tres usuarios legítimos. Lo que hacemos (vea los comentarios del código) es cotejar

la identificación que teclea el cliente con cada uno de los elementos de la matriz, hasta hallar una coincidencia o terminar el recorrido. En base a esta comprobación se decide si mostrar los contenidos o no.

Echemos un vistazo a la cabecera de autenticación:

```
header('WWW-Authenticate: Basic realm="Solo  
clientes"');
```

Fíjese en la parte resaltada. La palabra **Basic** puede ser sustituida por **Digest**. En ese caso se envían los datos de autenticación del usuario con un mayor nivel de encriptación, pero en determinadas instalaciones de la plataforma servidora puede no funcionar. Si desea usar esta encriptación, pruebe el sistema antes de abrirlo al público.

### 10.3 SOCKETS

En el primer capítulo del libro supimos, a grandes rasgos, lo que es un socket, desde el punto de vista de la pila de protocolos TCP/IP. Ahora vamos a saber cómo podemos manejar un socket desde nuestra página. La idea es conectarnos a un equipo que tiene asignada una dirección IP determinada, a través de un puerto concreto, y poder leer su contenido, o escribir en él. La forma de abrir, usar y cerrar un socket es muy similar a como hacíamos con los ficheros del disco (vea el capítulo 8). La principal diferencia está en que, para abrir un socket, usamos la función **fsockopen()**, en lugar de **fopen()**. Esta función recibe dos argumentos, separados por una coma. El primero es el nombre del recurso al que queremos acceder y el segundo es el puerto por el que esperamos que el servidor esté escuchando. La función devuelve un manejador que luego usaremos como hacemos con los ficheros. Eso, suponiendo que el socket se abra correctamente. Si no es así, la función devuelve un valor false. Por ejemplo, si queremos acceder a localhost, emplearemos la siguiente sintaxis:

```
$manejador=fsockopen("127.0.0.1", 80);
```

Esto nos permitirá conectarnos al servidor local por el puerto 80, que es el que tenemos configurado para escuchar solicitudes HTTP.

Otra alternativa es indicar el nombre de dominio del recurso al que nos queremos conectar. Observe el siguiente ejemplo:

```
$manejador=fsockopen("www.todoligues.com", 80);
```

Este método presenta un pequeño inconveniente. Antes de establecer la conexión, el script debe resolver el nombre, es decir, conectarse a su servidor DNS y determinar cuál es la dirección IP del dominio indicado (vea el capítulo 1). Por esta razón es, técnicamente, más lento aunque eso es algo que, con las máquinas y líneas actuales, usted no va a notar.

Las demás funciones, tanto de lectura como de escritura y cierre del socket, son las mismas que se usan para los ficheros.

Vamos a ver un ejemplo interesante. A lo largo de este capítulo usted ha aprendido a ver el contenido de las cabeceras de la solicitud en la página, con la función `getallheaders()` y a "forzar" algunas cabeceras de respuesta, con `header()`. Pero no hemos visto estas cabeceras. Vamos a usar un socket para lograr recuperar las cabeceras de respuesta de una página, ignorando el contenido. Observe el listado del script `cabecerasRespuesta.php`:

```
<?php
// Se abre el socket.
$manejador=fsockopen("127.0.0.1",80);
// Se manda una solicitud.
fputs ($manejador,"GET / HTTP/1.0\r\n\r\n");
$cadena="";
/* Se recorre el fichero remoto que se recibe (la
respuesta) hasta que se encuentre un carácter "<", lo que
indicará una etiqueta HTML o PHP. Como lo que nos interesa
son las cabeceras, en cuanto se encuentre una de estas
etiquetas sabemos que ya estamos en el cuerpo de la respuesta
y por lo tanto ya no es lo que buscamos, así que, cuando se
encuentre el primer "<" se interrumpe el recorrido.*/
while (!feof($manejador)){
    $guarismo=fgetc($manejador);
    if ($guarismo=="<"){
        break;
    }
    /* Cada carácter se va añadiendo a la cadena de las
cabeceras para mostrarla luego. Si se encuentra el carácter
correspondiente al ASCII 13 (salto de línea), lo que se añade
es un salto de línea HTML para poder mostrarlo en la
página.*/
    if (ord($guarismo)==13){
        $cadena.= "<br>\n";
    } else {
        $cadena.=$guarismo;
    }
}
echo ($cadena);
```

```
// Se cierra el socket.  
fclose ($manejador);  
?>
```

Quiero que preste especial atención a la línea que abre el socket, y que aparece resaltada en el listado. Fíjese en que, tal como hemos dicho, en la función `fsockopen()` aparece, en primer lugar, el nombre del recurso que queremos abrir. Como, en este caso, se trata del servidor local y ya conocemos su IP, la tecleamos directamente. En segundo lugar aparece el puerto por el que queremos establecer la comunicación. Como vamos a trabajar con el protocolo HTTP, indicamos el valor 80. Si fuéramos a trabajar con el protocolo SMTP, por ejemplo, pondríamos el valor 25. Cada protocolo trabaja en un puerto determinado. Vea el capítulo 1 al respecto.

A continuación vemos una línea que escribe una solicitud, mediante `fputs`. Como ya he comentado, los sockets, una vez abiertos, se tratan como si fueran ficheros. Como nuestro objetivo es obtener las cabeceras de la respuesta, es necesario que primero se presente una solicitud.

En ese momento se empieza a leer la respuesta. Vea los comentarios incluidos en el código para entender cómo la procesamos. El resultado lo vemos en la figura 10.7.

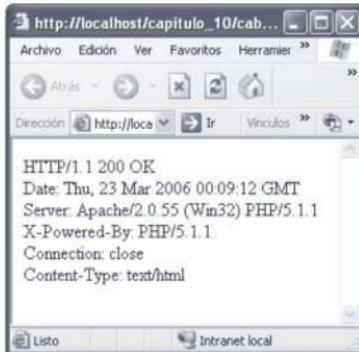


Figura 10.7

Veamos qué hemos obtenido. La primera línea nos informa de que la transmisión por HTTP ha sido correcta (código de estado 200). Vea, en la figura 10.5 los códigos de estado HTTP.

La segunda línea informa de la fecha y hora del servidor, no del cliente. No olvide que estamos recuperando las cabeceras de respuesta.

La tercera línea informa de cuál es el software del servidor, la plataforma sobre la que estamos trabajando y la versión del intérprete de PHP.

La cuarta línea es un poco redundante, puesto que nos da, una vez más, la versión del intérprete.

La quinta línea es la que nos informa de que la conexión está cerrada. Recuerde lo que le he comentado anteriormente, acerca de que, una vez enviada la respuesta, el servidor corta la conexión, hasta que se produce la siguiente solicitud.

Por último, la sexta línea informa del tipo de contenido de la página.

## CAPÍTULO 11

# ORIENTACIÓN A OBJETOS

---

---

Empezamos aquí con un capítulo tan breve en cuanto a contenido, como interesante desde el punto de vista de nuestras futuras aplicaciones prácticas, tal como veremos en seguida.

La Programación Orientada a Objetos (POO, en adelante) es la tendencia actual de todos los lenguajes de alto nivel, dadas las posibilidades que abre. Básicamente se trata de considerar analogías entre las situaciones que tenemos que resolver mediante la programación y los objetos del mundo real a los que estamos acostumbrados. En este capítulo aprenderemos a entender esa consideración y a utilizarla en nuestro favor.

### 11.1 CARACTERÍSTICAS DE LA POO

La POO basa las ventajas que nos ofrece en tres características principales:

- Abstracción.
- Reutilización.
- Actualización.

Para entender estas características, vamos a considerar la analogía con un objeto del mundo real: un automóvil. Seguramente usted conduce uno todos los días, desde su casa a su lugar de trabajo o a cualquier otra parte. Sin embargo, a menos que usted sea mecánico, no tiene por qué saber cómo funcionan los engranajes de la caja de cambios, o las bielas, los diferenciales, la bomba de inyección, etc. Quizás nunca se haya parado a pensarlo, pero en eso consiste la

abstracción. Usted usa un objeto, le saca todo el rendimiento que puede, pero no sabe cómo funciona internamente y, en realidad, no le hace ninguna falta. Después de todo, si su coche sufre una avería, aunque supiera arreglarla no tendría ni tiempo, ni las herramientas adecuadas. Así pues, ¿para qué va a saber de mecánica? Con la POO pasa lo mismo. Usted podrá incluir en sus scripts determinados módulos de código que hagan lo que usted necesita, sin necesidad de saber realmente cómo funcionan "por dentro".

La reutilización es una de las posibilidades más interesantes de la POO. Volviendo a la analogía de nuestro ejemplo, usted usa su coche para ir a trabajar pero, cuando llega el domingo, lo usa para salir al cine. Seguramente, lo emplee también para ir de compras, o para desplazamientos más largos durante sus vacaciones. Con los objetos de software ocurre lo mismo. Usted tiene un objeto destinado, por ejemplo, a realizar un proceso determinado con la información procedente de una base de datos. Ese mismo objeto puede hacer lo mismo con otra información procedente de cualquier otra fuente. Usted simplemente incluye el objeto en los scripts que crea y lo usa de acuerdo a sus necesidades, con pocas o ninguna modificación.

Por último, considere la actualización. Al igual que usted puede instalarle a su coche un nuevo radio CD, o una batería más potente, también puede actualizar, de la forma adecuada, los objetos de software para dotar de nuevas funcionalidades a sus scripts, o bien mejorar las actuales.

## 11.2 CONCEPTOS BÁSICOS

La POO se basa en los siguientes conceptos fundamentales:

- Clases
- Objetos
- Herencia
- Propiedades
- Métodos
- Eventos (en algunos entornos de programación, no en PHP)

Para crear un objeto necesitamos tener lo que se llama una *clase*. Podemos considerar las clases como moldes, a partir de los que se crean los objetos. Volviendo a las analogías con el mundo real, su coche está formado por una carrocería, que ha sido fabricada en unos moldes de estampación. Otros elementos, como el bloque del motor o el volante, han sido fabricados en unos moldes de fusión. Estos moldes existen también en el mundo de la programación. A partir de estos moldes se crean los objetos.

Su coche tiene unas características que lo definen. Usa tal o cual combustible, es de un determinado color, tiene un tamaño concreto, etc. Los objetos de software también tienen unas características, en forma de variables, que se llaman **propiedades**.

El coche implementa determinadas funcionalidades. Usted puede poner el motor en marcha, acelerar, frenar, girar las ruedas, etc. Los objetos hacen uso de unas funciones para lograr su objetivo. Éstas se llaman **métodos**.

Cuando se fabricó su coche, las piezas de la carrocería, de la mecánica y del interior salieron con la misma forma que tenían los respectivos moldes. Esto se conoce, en programación, como **herencia**. Los objetos tienen las mismas propiedades y métodos que las clases a partir de las que han sido creados. Además, a partir de una clase, se pueden crear otras (en seguida entraremos en detalles). Estas clases, creadas a partir de otras, también heredan sus propiedades y métodos.

### 11.3 CREACIÓN Y USO

Con la teoría y las analogías que hemos visto ya es suficiente. Vamos a pasar a crear nuestras propias clases y objetos, así como a implementar propiedades y métodos.

Antes de crear un objeto necesitamos disponer de la clase a partir de la que se implementará dicho objeto. Para crear una clase usamos la sentencia **class**, seguida del nombre de la clase que vamos a crear y del cuerpo de la clase, que es donde se definen las propiedades y métodos que luego tendrán los objetos de esa clase. La sintaxis genérica obedece al siguiente esquema:

```
class nombreDeLaClase {  
    propiedades  
    métodos  
}
```

A partir de que la clase haya sido definida podemos crear todos los objetos que necesitemos de dicha clase. Para ello usamos el operador **new**, tal como se ve en la siguiente sintaxis genérica:

```
$objeto = new nombreDeLaClase ();
```

Para empezar a familiarizarnos con esta forma de trabajo nada mejor que ver un ejemplo de su uso. Para ello vamos a ver un ejemplo muy sencillo, en el que se define una clase destinada a sumar dos números. Vea el script **claseSimple.php**:

```

<?php
// Aquí empieza la definición de la clase.
class sumador {
    function sumador (){
    }
    function sumar($sum_1, $sum_2){
        $total=$sum_1+$sum_2;
        return $total;
    }
}
// Terminamos la definición de la clase.
// =====
// Creamos un nuevo objeto de la clase que tenemos en
el script.
$hacerSumas=new sumador ();
// Llamamos al método sumar del objeto que hemos
creado.
$cinco = $hacerSumas->sumar (2,3);
// Mostramos el resultado.
echo ("La suma de 2 + 3 da $cinco.");
?>

```

Al cargar el script en el navegador vemos el resultado de la figura 11.1.



Figura 11.1

Si está usted pensando que para obtener “eso” no hacía falta tanto código, tiene razón. Sin embargo, el listado de este script nos va a permitir anclar conceptos necesarios, y eso es lo que importa.

Vamos a empezar viendo cómo definimos la clase que vamos a emplear. Fíjese en la primera parte del código:

```

// Aquí empieza la definición de la clase.
class sumador {
    function sumador (){

```

```
    }
    function sumar($sum_1, $sum_2){
        $total=$sum_1+$sum_2;
        return $total;
    }
} // Terminamos la definición de la clase.
```

Observe que, dentro del cuerpo de la clase, hay una función (un método) que tiene el mismo nombre que dicha clase. Esto es lo que se llama un *constructor*. Sirve para poder crear objetos de dicha clase y que éstos hereden sus propiedades. En este ejemplo concreto, el constructor no sirve para nada. De hecho puede suprimirlo y el script le funcionará igualmente. Sin embargo, he querido incluirlo como una primera toma de contacto con esta idea. En el próximo ejemplo veremos un uso real del mismo.

Observe la función **sumar ()**. Es la definición de un método. Cuando se crea una clase (familiarícese ya con la terminología adecuada), las funciones se llaman métodos. Las variables propias de la clase (en este primer script no hay ninguna) se llaman propiedades. Quiero llamar su atención acerca de lo que son "variables propias de la clase": son aquéllas que se definen en el cuerpo de la clase (no dentro de un método) y serán heredadas por los objetos que implementemos a partir de dicha clase.

Siguiendo con el script, vemos cómo creamos un objeto, así:

```
$hacerSumas=new sumador();
```

A continuación hacemos una llamada al método **sumar** del objeto que hemos creado, tal como ve en la siguiente línea:

```
$cinco = $hacerSumas->sumar (2,3);
```

Fíjese en el operador **->** que usamos para referirnos a una propiedad o un método de un objeto. La sintaxis genérica es la siguiente:

```
$objeto->propiedad
$objeto->método (argumentos, si proceden);
```

Quiero llamar su atención respecto a un punto de la sintaxis. Fíjese en el ejemplo genérico que acabamos de citar que la propiedad no lleva el signo \$, aún tratándose de una variable. Esto es así porque este signo ya precede al nombre del objeto. Cuando se usa esta notación, el nombre de la propiedad se escribe sin \$. Lo contrario producirá que la propiedad no sea accesible y, por lo tanto, no se le puede

asignar un valor, ni leerlo. Sin embargo, si quiere declarar o inicializar la propiedad fuera del cuerpo de un método, para lo cual no usará la referencia `$this`, sí debe anteponerle el signo `$`.

Antes de pasar al siguiente ejemplo debe saber que en la POO existe la referencia `$this`, que se usa como comodín para referirse al objeto que estamos manejando sin especificar su nombre. Este concepto, así como el de propiedades, está presente en el siguiente ejemplo, llamado `claseFuncional.php`:

```
<?php

// Empieza la definición de la clase.
class coche {
    function coche($marca, $modelo, $color){
        $this->marca=$marca;
        $this->modelo=$modelo;
        $this->color=$color;
    }
    function mostrar (){
        echo ("<table width='200' border='2'
cellpadding='2' cellspacing='0'>");
        echo ("<tr><td width='50%'>MARCA</td><td
width='50%'>".$this->marca."</td></tr>");
        echo ("<tr><td>MODELO</td><td>".$this-
>modelo."</td></tr>");
        echo ("<tr><td>COLOR</td><td>".$this-
>color."</td></tr>");
        echo ("</table>");
    }
}
// Termina la definición de la clase.
// =====

// Se crean tres objetos formando una matriz
$movil[0]=new coche("Peugeot", "405", "Gris");
$movil[1]=new coche("Renault", "Master", "Blanco");
$movil[2]=new coche("Cadillac", "Seville", "Azul");

/* Mediante un bucle se ejecuta el método mostrar de
cada uno de los objetos.*/
for ($contador=0;$contador<3; $contador++)
$movil[$contador]->mostrar();
?>
```

Al ejecutar este script verá en su navegador el resultado que aparece en la figura 11.2.



Figura 11.2

Como en el caso anterior, vamos a ir despiezando el código, para ver cómo funciona. En primer lugar tenemos la definición de la clase coche, tal como aparece a continuación:

```
class coche {  
    function coche($marca, $modelo, $color){  
        $this->marca=$marca;  
        $this->modelo=$modelo;  
        $this->color=$color;  
    }  
    function mostrar (){  
        echo ("<table width='200' border='2'  
cellpadding='2' cellspacing='0'>");  
        echo ("<tr><td width='50%'>MARCA</td><td  
width='50%'>".$this->marca."</td></tr>");  
        echo ("<tr><td>MODELO</td><td>".$this-  
>modelo."</td></tr>");  
        echo ("<tr><td>COLOR</td><td>".$this-  
>color."</td></tr>");  
        echo ("</table>");  
    }  
}
```

Observe el método constructor, que aparece resaltado. Recibe tres argumentos que aparecen como variables. Dentro de este método se asigna cada argumento a una de las propiedades del objeto. ¿De qué objeto? Del que se está creando en ese momento. Recuerde: es un constructor. Este método se ejecuta cada vez que se crea un objeto, y sólo entonces. Observe que nos referimos al objeto en proceso, mediante la referencia \$this. Para crear un objeto en nuestro ejemplo usamos una sentencia como la siguiente:

```
$movil[0]=new coche("Peugeot", "405", "Gris");
```

El nombre del objeto es un elemento de una matriz. Los objetos pueden agruparse en matrices, igual que las variables. Pero eso no importa ahora. Lo que nos interesa es conocer el proceso. Cuando creamos un objeto como en este ejemplo, el operador new invoca al constructor de la clase. Vea que la invocación se hace pasando tres cadenas como parámetros. Estas se alojan en las variables que hay en la definición del método. No confunda estas variables con las propiedades del objeto que se crean dentro del cuerpo del constructor. A pesar de tener el mismo nombre, las propiedades (variables de la clase) no tienen nada que ver con las variables que recibe el constructor.

Lo último que vemos es un bucle que recorre cada elemento de la matriz de objetos, llamando al método mostrar de cada objeto. Este método se encarga de crear una tabla y mostrar el valor de las propiedades. A partir de que el objeto ha sido creado, podemos hacer con él lo que queramos.

Como he mencionado anteriormente, una de las características de la POO es la abstracción. De hecho, cuando se usa esta técnica lo normal es que la definición de la clase se encuentre en un fichero aparte, y que sea llamada desde el script mediante include () o require (). Y, de hecho, usted lo único que necesita saber acerca de una clase es qué propiedades y métodos tienen los objetos que se implementen a partir de la misma. La operativa de esos métodos “por dentro” ya no nos interesa.

Por supuesto, siempre es bueno conocer las clases que usamos, ya que, de este modo, podremos actualizarlas o modificarlas si lo necesitamos.

## 11.4 HERENCIA

Como ha visto en el apartado anterior, los objetos heredan las propiedades y los métodos de la clase a la que pertenecen. Pero el concepto de herencia es más amplio. Usted puede crear una clase a partir de otra. La clase original se conoce, genéricamente, con el nombre de *superclase* y las clases que se crean a partir de

ella se conocen, también genéricamente, como *clases derivadas* o *subclases*. Para crear una subclase a partir de una superclase empleamos la palabra clave *extends*. Vea un ejemplo general:

```
class A {  
    propiedad_A1  
    propiedad_A2  
    metodo_A3  
    metodo_A4  
}  
  
class B extends A {  
    propiedad_B1  
    metodo_B2  
}
```

Fíjese en que hemos declarado la clase B como derivada de la clase A. Por esta razón, la clase B tiene todas las propiedades y métodos de la clase A, además de los suyos propios. Si ahora creamos un objeto de la clase A, éste tendrá las propiedades y métodos de su clase. Si creamos un objeto de la clase B, tendrá los métodos y las propiedades de ambas clases.

En el capítulo 13, dedicado al correo electrónico, verá una clase real, que se emplea en scripts profesionales, y será un ejemplo muy útil de POO.



## CAPÍTULO 12

# IMÁGENES

---

---

En este capítulo vamos a aprender a gestionar las imágenes que necesitemos presentar en nuestras páginas. Por supuesto, usted ya sabe que HTML le permite insertar imágenes en un documento web. Sin embargo, PHP le permite actuar sobre las imágenes antes de que sean enviadas al navegador, implementando ciertas funcionalidades que conoceremos en este capítulo.

### 12.1 LO QUE NECESITAMOS

Para trabajar con imágenes, PHP implementa una librería con funcionalidades específicas, conocida como **GD**. Desde la versión 5 del lenguaje se implementa la versión 2 de la librería, conocida como **GD2**. Si usted instaló PHP a partir del paquete AppServ que le comenté en el capítulo 2 debe tener esta librería disponible. Para asegurarse, abra el fichero php.ini, que deberá estar en el directorio C:/Windows/ (presupongo que usted está trabajando con esta plataforma). En el fichero debe haber una línea como la siguiente:

```
extension=php_gd2.dll
```

Asegúrese de que esa línea aparece tal cual, sin nada más. En concreto, *NO* debe llevar nada precediéndola.

Además, dentro del directorio C:/appserv/php/ext/ debe haber un archivo llamado php\_gd2.dll. Si es así, ya tiene todo lo que necesita para empezar a trabajar con imágenes. Si ha instalado la última versión de AppServ, esta librería existe por defecto, con lo que no se tiene que preocupar de ella.

## 12.2 LO QUE PODEMOS HACER

PHP maneja imágenes, principalmente, para Internet. A través de las funciones que incorpora la librería GD2 podremos crear imágenes a partir de ficheros ya almacenados en el servidor, así como otras completamente nuevas.

PHP maneja imágenes en formatos GIF, JPEG, PNG y BMP. Este último formato no nos interesa especialmente, dado el peso de los archivos en que se almacenan las imágenes. El formato GIF fue suprimido en la librería GD 1.8 que empleaba PHP 4, pero en la actual versión también está disponible.

A lo largo de este capítulo veremos cómo podemos crear una imagen, copiarla, redimensionarla, cambiar un determinado color por otro, determinar la paleta de colores que forman una imagen e, incluso, modificarla píxel a píxel. También aprenderemos a crear gráficos a partir de datos numéricos, lo que nos será muy útil para representar evoluciones, bien de tipo climatológico, financiero, etc.

PHP proporciona funciones para llevar a cabo todas estas operaciones. De este modo, podemos implementar imágenes muy adecuadas en nuestras páginas, sin necesidad de recurrir a programas de retoque fotográfico, ni de edición de imagen. Por supuesto, no conseguiremos las mismas funcionalidades desde PHP que desde un programa profesional al efecto, pero sí lograremos unos resultados bastante buenos para dinamizar nuestras páginas. Y, lo que es más importante: la edición que llevemos a cabo desde scripts de PHP puede hacerse en base a datos que procedan de otras fuentes de una forma muy cómoda.

## 12.3 EMPEZANDO A TRABAJAR

Lo primero que vamos a hacer es comprobar que nuestro intérprete de PHP (y, más concretamente, la librería GD2 que tenemos instalada) soporta los distintos tipos de imágenes. Para determinadas operaciones con cada imagen existen unas funciones llamadas *imagegif()*, *imagejpeg()*, *imagepng()* e *imagewbmp()*. De momento no nos importa lo que podemos hacer con ellas. Nos ocuparemos de eso muy pronto. Lo que sí sabemos, porque así nos lo dicen las especificaciones de PHP, es que, si alguna de estas funciones no existe (si no es reconocida por el intérprete) es que el tipo de imagen al que se refiere la función no es soportado. La función *function\_exists()* recibe, como argumento, el nombre de una función cualquiera del lenguaje, y nos devuelve un valor true o false, según esa función exista o no. Esto es válido, no sólo para las funciones de imagen, sino también para comprobar cualquier otra función del lenguaje. Si alguna vez tiene dudas acerca de si su intérprete soporta tal o cual función, utilice *function\_exists()* para determinarlo. Observe el listado *tiposDeImagen.php*:

```
<?php
    define ("salto", "<br>\n");
    if (function_exists("imagegif")) echo ("Manejo
imágenes GIF.".salto);
    if (function_exists("imagejpeg")) echo ("Manejo
imágenes JPG.".salto);
    if (function_exists("imagepng")) echo ("Manejo
imágenes PNG.".salto);
    if (function_exists("imagewbmp")) echo ("Manejo
imágenes BMP.".salto);
?>
```

Como ve, se comprueba la existencia de las cuatro funciones que hemos mencionado, cuyo uso, como digo, se comentará más adelante. Lo que importa es el resultado obtenido, que vemos a continuación:

**Manejo imágenes GIF.**  
**Manejo imágenes JPG.**  
**Manejo imágenes PNG.**  
**Manejo imágenes BMP.**

Si usted obtiene otro resultado en la página, en el que falte alguna linea, deberá actualizar su versión del intérprete. No obstante, si ha instalado el paquete AppServ, tal como le describí en el capítulo 2, no debe tener ningún problema.

Cuando se trabaja con un script que va a enviar una imagen al navegador es necesario incluir, al principio de dicho script, una cabecera (ver la función header () en el capítulo 10) indicando que el contenido incluye una imagen, y de qué tipo es, tal como aparece en la siguiente línea:

```
header ("Content-type: image/jpeg");
```

Las alternativas a image/jpeg son image/gif, image/png e image/wbmp, dependiendo del tipo de imagen que maneje el script. Yo tengo que anotar que algunos scripts funcionan perfectamente sin necesidad de incluir la cabecera. No obstante, dado que la necesidad de la misma aparece en las especificaciones oficiales del lenguaje, y que no siempre se puede omitir, en lo sucesivo la incluiremos.

Las imágenes se tratan en PHP bajo la misma filosofía de trabajo que usábamos para los ficheros de texto (ver capítulo 8): primero se “abre” la imagen, que podrá existir como un archivo en el servidor, o en otra parte, pero podrá no existir, si lo que vamos a hacer es crear una nueva imagen. Esta operación genera un **manejador**, que emplearemos para todas las operaciones que queramos llevar a

cabo sobre la imagen. Por último, el equivalente a cerrar un fichero, será destruir el manejador que hemos usado, a fin de liberar recursos en la memoria.

Para crear un manejador, a través del cual trabajaremos con una imagen usaremos las siguientes funciones: *imagecreatefromjpeg()*, *imagecreatefromgif()*, *imagecreatefrompng()* e *imagecreatefromwbmp()*. Elegiremos una u otra según el tipo de imagen con el que vamos a trabajar. La función recibe, como argumento, el nombre y, en su caso, la ruta del fichero que contiene la imagen con la que queremos trabajar, y devuelve el manejador correspondiente. Si no se puede acceder a la imagen especificada, la función devuelve un valor false. Por si se da ese caso es interesante predecir la función del operador de errores (@). Por lo tanto, podemos decir que estas funciones sirven para crear un manejador para una imagen ya existente en un archivo.

A continuación, una vez creado el manejador, haremos “algo” con la imagen. De momento, vamos, simplemente, a mostrarla en el navegador. Para ello emplearemos alguna de las cuatro funciones que habíamos mencionado en tiposDeImagen.php, y que eran las siguientes: *imagegif()*, *imagejpeg()*, *imagepng()* e *imagewbmp()*. Emplearemos una u otra, según sea el tipo de imagen al que se refiere el manejador. La función elegida recibirá, como argumento, la variable que contenga dicho manejador.

Por último, para “cerrar” la imagen (destruir el manejador) y liberar así recursos de la memoria emplearemos la función *imagedestroy()*, que recibe el manejador que queremos cerrar como parámetro.

Para ver la operativa de estas tres funciones observe el listado del script **mostrarRotulo.php**:

```
<?php
    header("Content-type: image/jpeg");
    // Se crea un manejador para una imagen.
    $imagenJPG=imagecreatefromjpeg
    ("imagenes/lenguaje.jpg");
    // Se envía la imagen al navegador.
    imagejpeg($imagenJPG);
    // Se liberan recursos.
    imagedestroy($imagenJPG);
?>
```

Este script toma un archivo de imagen que está en una carpeta, llamada “ímagenes”, dentro de la carpeta correspondiente a este capítulo en el CD adjunto. Como puede ver, lo primero que hacemos es enviar la correspondiente cabecera al navegador, siguiendo, como le comenté anteriormente, las especificaciones del

lenguaje. A continuación, creamos un manejador que será la referencia al archivo de imagen y que nos permitirá trabajar con ella. Después, enviamos la imagen al navegador y, por último, eliminamos el manejador, a fin de liberar recursos en la memoria del sistema. Es importante que no olvide esto último, ya que, si su script envía muchas imágenes y usted no libera recursos, podría llegar a colapsar el servidor, sobre todo en páginas con muchos visitantes diarios.

El resultado del script que acabamos de ver aparece en la figura 12.1.

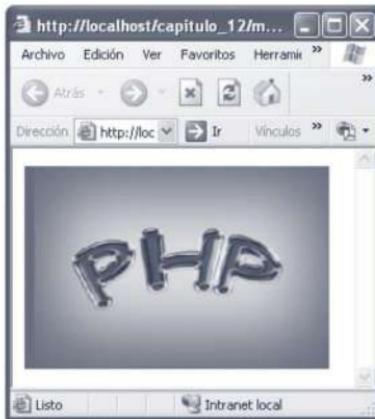


Figura 12.1

La función `imagejpeg()`, así como sus homólogas para los otros tipos de imágenes, soporta un segundo parámetro opcional. Suponga la siguiente línea en el script:

```
imagejpeg($imagenJPG, "imagenes/copiaDeImagen.jpg");
```

Esto copiará el archivo original en otro, cuyo nombre y ruta se especifica en el segundo argumento.

Además, puede incluir un tercer parámetro (también opcional), indicando la calidad con la que desea mostrar la imagen, o grabarla en el nuevo fichero. Si lo que desea es lo primero, podrá usar la siguiente línea:

```
imagejpeg($imagenJPG, "", 40);
```

Como ve, en el lugar del segundo parámetro ponemos una cadena vacía. De este modo no se grabará la imagen en un fichero, sino que será enviada al navegador. El tercer parámetro es un valor numérico. Cuanto más alto sea, con más calidad se mostrará la imagen, pero también tardará más en descargarse. En el caso de que lo que deseemos sea grabar la imagen en otro archivo, especificando la calidad (y, por tanto, también, el peso del nuevo fichero), usaremos la siguiente línea:

```
imagejpeg($imagenJPEG, "imagenes/copiaDeImagen.jpg",  
40);
```

Observe, en la figura 12.2 la misma imagen que habíamos visto anteriormente, con un valor para la calidad de sólo 1. Como puede comprobar, el resultado es desastroso, por decir lo mínimo.

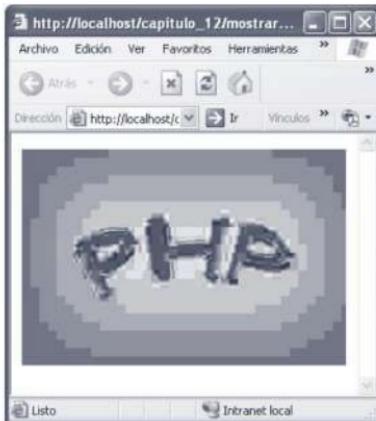


Figura 12.2

Cuando un script envía una imagen al navegador, su nombre puede ser incluido como atributo src en la etiqueta img de una página HTML. Observe el listado de **cargaExterna.htm**, que aparece a continuación:

```
<html>  
  <head>  
    <title>Cargando imagen desde un script</title>  
  </head>  
  <body>
```

```

</body>
</html>
```

Vea la línea resaltada. Cuando cargue esta página en el navegador obtendrá, al igual que antes, el resultado de la figura 12.1.

Otra operación básica que podemos llevar a cabo es obtener la anchura y altura, en pixeles, de una imagen. Para obtener la anchura podemos usar la función *imagesx()*. La altura nos la proporciona la función *imagesy()*. Estas dos funciones reciben, como argumento, el manejador de una imagen. Es útil obtener las dimensiones físicas de una imagen, ya que, de este modo, se puede reservar sitio en una celda de una tabla, o decidir una mayor o menor calidad de representación. A mayor tamaño de imagen, menor calidad (dentro de unos límites razonables, por supuesto), para acelerar el proceso de envío de la imagen al navegador. Para ver cómo operan estas propiedades mire el script **dimensiones.php**:

```
<?php
    define("salto","<br>\n");
    // Se crea un manejador para una imagen.
    $ImagenJPG=imagecreatefromjpeg
    ("imagenes/lenguaje.jpg");

    // Se muestran la anchura y la altura.
    echo (salto."Anchura de la imagen:
    ".imagesx($ImagenJPG));
    echo (salto."Altura de la imagen:
    ".imagesy($ImagenJPG).salto);

    // Se liberan recursos.
    imagedestroy($ImagenJPG);
?>
```

Este código crea un manejador a partir de la imagen que hemos visto anteriormente y nos muestra la anchura y altura, en pixeles, de la misma. El resultado es el siguiente:

**Anchura de la imagen: 300**  
**Altura de la imagen: 200**

Puede darse el caso de que usted necesite la imagen y el texto en la misma página. Lo mejor, en esos casos, es tener una página HTML en la que se incluyan referencias a pequeños scripts de PHP, como hemos visto en la página anterior. Observe el listado **mostrarDimensiones.php**:

```
<html>
```

```
<head>
    <title>Cargando imagen desde un script</title>
</head>

<body>
    
    <?php
        include ("dimensiones.php");
    ?>
</body>
</html>
```

Como ve, se combinan dos scripts externos para mostrar ambos tipos de resultados, tanto el gráfico como el textual. Este tipo de uso es muy habitual, ya que permite que, cada vez que se carga la página, se obtengan resultados procedentes de otros scripts que, a su vez, pueden estar actualizándose desde otras fuentes, tales como bases de datos (tema del que hablaremos en los capítulos 15, 16 y 17 del libro). El resultado de este último script es el que aparece en la figura 12.3.



Figura 12.3

Para finalizar este apartado conoceremos la función *getimagesize()*, que nos proporciona ambos parámetros, además del tipo de imagen y algún dato adicional. Esta función recibe, como argumento, el nombre y, en su caso, la ruta, de un archivo de imagen. No recibe un manejador, como otras funciones, sino el nombre del archivo, directamente. La función *getimagesize()* devuelve una matriz

de siete elementos. Para ver cómo operan estas funciones observe el script **propiedades.php**:

```
<html>
<head>
    <title>Propiedades de una imagen</title>
</head>
<body>
    <h1>PROPIEDADES DE UNA IMAGEN</h1>
    <table width="300" border="2" cellpadding="2"
cellspacing="0">
        <tr><th colspan="2">Lista de
propiedades</th></tr>
        <?php
            $matriz=getimagesize("imagenes/lenguaje.jpg");
            foreach ($matriz as $elemento=>$valor) {
                echo ("<tr><td>$elemento</td><td>
$valor</td></tr>");
            }
        ?>
    </table>
</body>
</html>
```

Observe, en la línea resaltada, cómo usamos la función que nos interesa. El resultado de este script es el que aparece en la figura 12.4.

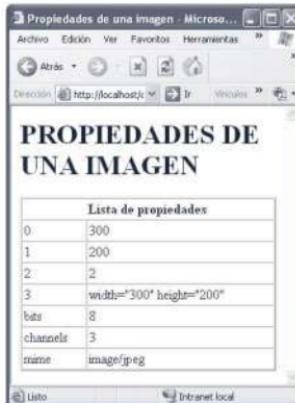


Figura 12.4

de siete elementos. Para ver cómo operan estas funciones observe el script **propiedades.php**:

```
<html>
<head>
    <title>Propiedades de una imagen</title>
</head>
<body>
    <h1>PROPIEDADES DE UNA IMAGEN</h1>
    <table width="300" border="2" cellpadding="2"
cellspacing="0">
        <tr><th colspan="2">Lista de
propiedades</th></tr>
        <?php
            $matriz=getimagesize("imagenes/lenguaje.jpg");
            foreach ($matriz as $elemento=>$valor) {
                echo ("<tr><td>$elemento</td><td>
$valor</td></tr>");
            }
        ?>
    </table>
</body>
</html>
```

Observe, en la línea resaltada, cómo usamos la función que nos interesa. El resultado de este script es el que aparece en la figura 12.4.

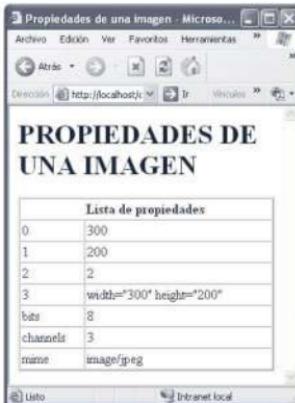


Figura 12.4

Los resultados obtenidos se interpretan de la siguiente manera:

- Clave **0**. Contiene la anchura de la imagen, en pixeles.
- Clave **1**. Contiene la altura de la imagen en pixeles.
- Clave **2**. El contenido se refiere al formato de la imagen (tipo de imagen), según la siguiente lista: 1=GIF, 2=JPG, 3=PNG, 4=SWF, 5=PSD, 6=BMP, 7=TIFF (orden de bytes intel), 8=TIFF (orden de bytes motorola), 9=JPC, 10=JP2, 11=JPX, 12=JB2, 13=SWC, 14=IFF, 15=WBMP, 16=XBM. Como ve, PHP puede reconocer muchos tipos de imagen diferentes, aunque, para entornos web, sólo usaremos algunos de ellos, como usted sabe. En concreto, nos interesan los cuatro primeros. El cuarto corresponde a animaciones de Flash.
- Clave **3**. Contiene una cadena con la altura y la anchura, que puede ser usada, directamente, en etiquetas <img> de HTML.
- Clave **bits**. El número de bits por color, es decir, lo que se conoce como profundidad de color.
- Clave **channels**. El número de canales de color. Es 3 cuando el color de la imagen es RGB y 4 cuando es CMYK (imágenes para impresión).

Clave **mime**. La codificación que debemos especificar en el Content\_type de la cabecera.

## 12.4 EL COLOR

Es necesario comprender cómo son los colores en una imagen, ya que PHP nos proporciona algunas funciones para manejarlos, pero no podremos usarlas adecuadamente si no comprendemos su naturaleza. Al final del apartado anterior hemos hecho alguna mención al color. Ahora aprenderemos todo lo que necesitamos al respecto.

Las imágenes que se muestran en la pantalla de un ordenador se forman a partir de tres colores básicos: el rojo, el verde y el azul. De ahí que a este modo de trabajar con el color se le llame, en términos informáticos, RGB (de los nombres en inglés: Red, Green y Blue). Seguramente, si usted recuerda sus tiempos de estudiante, sus profesores le enseñaron que, por ejemplo, el verde es un color compuesto, que se forma a partir del azul y el amarillo. Esto es cierto para las mezclas de tintas para imágenes impresas, lo que se conoce como *colores-tinta*. Pero las imágenes en una pantalla se forman a partir de otras combinaciones, conocidas como *colores-luz*. En este entorno, siguiendo con el ejemplo mencionado, el amarillo es un color compuesto, formado a partir de rojo y verde. Debido a estas diferencias a la hora de formar el color, las imágenes en una página web pueden no ser iguales, en lo que a tonalidades se refiere, a las mismas imágenes cuando son volcadas a la impresora. Cuando queremos que una imagen,

en la pantalla, muestre un aspecto más similar al que tendrá una vez impresa, es necesario cambiar su formato de colores al conocido como CMYK (los equipos actuales hacen esto de modo automático, casi siempre).

Realmente, lo que nos interesa aquí es el formato de colores RGB, ya que estamos trabajando para la creación de páginas web que serán visualizadas en un monitor. Cualquier color que podamos montar en una imagen se forma a partir de una combinación de dos de los colores básicos, o de los tres. Cada uno de estos colores puede adoptar uno de 256 posibles valores (de 0 a 255). Esto nos ofrece un total de más de diecisésis millones de posibles combinaciones, lo que satisface completamente las necesidades de cualquier imagen. La ausencia total de los tres colores básicos (valor 0 para cada uno) da, como resultado, el negro. La presencia total de los tres (valor 255 para cada uno) muestra el blanco. Distintas combinaciones de valores dan como resultado diferentes colores.

Algo que debemos saber es que cada imagen lleva asociada su propia paleta de colores. Ésta es una referencia de todas las tonalidades que forman dicha imagen. Cada tonalidad está compuesta por unos determinados valores de rojo, verde y azul. Estas combinaciones se almacenan en posiciones numeradas de la paleta, lo que permite acceder a cada color individualmente y manipularlo, si fuera necesario. A continuación vamos a conocer un par de funciones destinadas a descubrir cada uno de los tonos de la paleta correspondiente a una imagen. La primera de ellas nos devuelve el número total de colores que forman la paleta. Se trata de ***imagecolorstotal()***, que recibe, como argumento, el manejador con el que se ha abierto la imagen. Observe el script ***totalDeColores.php***, que hace uso de esta función:

```
<?php
// Se crea un manejador para una imagen.
$imagen=imagecreatefromgif ("imagenes/lenguaje.gif");
// Se obtiene el total de colores de la imagen.
$coloresTotales=imagecolorstotal($imagen);

// Se muestra el resultado.
echo ("La imagen imagenes/lenguaje.gif tiene un total
de $coloresTotales colores.");

// Se liberan recursos.
imagedestroy($imagen);
?>
```

Cuando ejecute este script obtendrá el siguiente resultado:

**La imagen imagenes/lenguaje.gif tiene un total de 255 colores.**

Esto significa que en la paleta de la imagen hay un total de 255 tonalidades diferentes.

Como complemento de esta función contamos con *imagecolorsforindex()*. Ésta recibe dos argumentos, separados por una coma. El primero es el manejador de la imagen con la que estamos trabajando. El segundo es un número que corresponde a un índice de la paleta de la imagen. Por ejemplo, para el resultado del script anterior, el segundo argumento de esta función deberá estar entre 0 y 254. La función devuelve una matriz con los valores de rojo, verde y azul del índice especificado. Además, la matriz contiene un cuarto valor que corresponde a la transparencia del color en la imagen (lo que se conoce como *factor alfa*). Para ver cómo opera esto, vamos a crear una página que nos muestre todos los colores de la paleta, con sus correspondientes valores. Es decir, que obtendremos 255 matrices, cada una de ellas con cuatro elementos. Observe el siguiente script, que hace uso de esta función. Se llama **paleta.php**:

```
<?php
define ("salto", "<br>\n");
// Se crea un manejador para una imagen.
$imagen=imagecreatefromgif ("imagenes/lenguaje.gif");
// Se obtiene el total de colores de la imagen.
$coloresTotales=imagecolorstotal($imagen);
// Se muestra el total de colores de la imagen.
echo ("La imagen imagenes/lenguaje.gif tiene un total
de $coloresTotales colores.".salto);
// Se inicia la tabla de colores.
echo ("<table width='480' border='2' cellpadding='0'
cellspacing='0'>");
echo
("<tr><td>Indice</td><td>Rojo</td><td>Verde</td>");
echo
("<td>Azul</td><td>Alfa</td><td>Color</td></tr>");
// Un bucle recorre cada uno de los colores de la
paleta
for ($cuenta=0; $cuenta<$coloresTotales; $cuenta++){
    //Se muestra el número de índice en la paleta.
    echo ("<tr align='center'><td>".$cuenta."</td>");
    // Se obtiene la matriz con los cuatro valores del
    color del índice actual.
    $matriz=imagecolorsforindex ($imagen,$cuenta);

    // Se va a calcular la expresión hexadecimal del color
    actual.
    $valorHexa="#";
    // Se recorre toda la matriz del color actual.
```

```
foreach ($matriz as $clave=>$valor){  
    // Se muestra el valor del componente que estamos  
    viendo.  
    echo ("<td>".$valor."</td>");  
    // Si es el componente rojo, verde o azul se añade el  
    valor hexadecimal.  
    if ($clave=="red" || $clave=="green" ||  
$clave=="blue") {  
        if ($valor<16){  
            // Si tiene menos de dos dígitos hexa, se antepone un  
            cero.  
            $valorHexa.=("0".dechex($valor));  
        } else {  
            $valorHexa.=dechex($valor);  
        }  
    }  
    // Se muestra una celda con el color actual de fondo.  
    echo ("<td bgcolor=\"$valorHexa\">&nbsp;</td>");  
    echo ("</tr>");  
}  
echo ("</table>");  
// Se liberan recursos.  
imagedestroy($imagen);  
?>
```

Lea los comentarios. Lo que hacemos es que, una vez determinado el número total de colores de la paleta, los recorremos todos mediante un bucle. En cada iteración obtenemos la matriz con los tres componentes básicos del color que estamos analizando (Rojo, Verde y Azul) y el factor alfa del mismo. En el caso de esta imagen, todos los colores tienen un factor alfa de 0, es decir, son totalmente opacos, sin transparencias. Pero eso no importa a los efectos que buscamos. Una vez obtenidos los componentes de cada color, mostramos sus valores en una tabla. Además, formamos una cadena hexadecimal con los valores del rojo, verde y azul, y le añadimos a cada fila de la tabla una celda con el correspondiente color de fondo. Ejecute el script para ver el resultado. Yo no lo he reproducido aquí porque una tabla con 255 filas no es para incluirla en estas páginas, debido a su tamaño. Además, observará que muchas de las tonalidades son parecidas entre sí, a simple vista, diferenciándose muy poco en sus valores. Esto se debe a que la imagen contiene varios degradados suaves, que generan muchos colores similares.

Ya conocemos cómo determinar cuántos colores tiene una imagen y sabemos cómo extraer toda la información posible de la paleta asociada. Sabemos que cada uno de los colores está identificado por un índice y tiene una composición RGB. Lo mejor de esto es que, a partir de ahí, podemos modificar cada uno de los

tonos que forman la paleta. Para ello usamos la función *imagecolorset()*. Esta función recibe, nada menos, que cinco argumentos, que son los siguientes:

- El manejador con el que se ha abierto la imagen.
- El número de índice que tiene en la paleta el color que queremos modificar.
- El valor de rojo que queremos darle al nuevo color.
- El valor de verde que queremos darle al nuevo color.
- El valor de azul que queremos darle al nuevo color.

Para ver cómo funciona vamos a escribir un script que muestre el negativo de una imagen. Lo que haremos es recorrer cada tonalidad de la paleta, e invertir los valores de rojo, verde y azul. Para ello, tomaremos el valor actual de cada componente y lo restaremos de 255, que es el máximo que puede tener ese componente. Así pues, suponga que tiene un tono rojo puro (255,0,0). Restaremos los valores actuales de 255 para cada componente individual, como hemos dicho, obteniendo 0,255,255, es decir, la mezcla de verde y azul, que da como resultado el color cyan. Entonces asignamos el nuevo color obtenido al correspondiente índice en la paleta. Por último, mostramos la imagen para comprobar el resultado. El script que hace esto es **cambiarColores.php**:

```
<?php
    header("Content-type: image/gif");

    // Se crea un manejador para una imagen.
    $imagen=imagecreatefromgif ("imagenes/lenguaje.gif");

    // Se obtiene el total de colores de la imagen.
    $coloresTotales=imagecolorstotal($imagen);

    // Un bucle recorre cada uno de los colores de la
    paleta
    for ($cuenta=0; $cuenta<$coloresTotales; $cuenta++){

        // Se obtiene una matriz con los colores del índice
        actual
        $matriz=imagecolorsforindex ($imagen, $cuenta);

        // Se recalculan los valores de los componentes para
        lograr el tono inverso.
        $nuevoRojo=255-$matriz["red"];
        $nuevoVerde=255-$matriz["green"];
        $nuevoAzul=255-$matriz["blue"];

        // Se asignan los nuevos valores al índice en curso.
        imagecolorset($imagen, $nuevoRojo, $nuevoVerde, $nuevoAzul);
    }

    // Se muestra la imagen
    imagegif($imagen);
    imagedestroy($imagen);
}
```

```
imagecolorset ($imagen, $cuenta, $nuevoRojo,  
$nuevoVerde, $nuevoAzul);  
}  
// Se muestra el resultado y se liberan recursos.  
imagegif ($imagen);  
imagedestroy ($imagen);  
?>
```

Al ejecutar el script se obtiene el resultado de la figura 12.5.

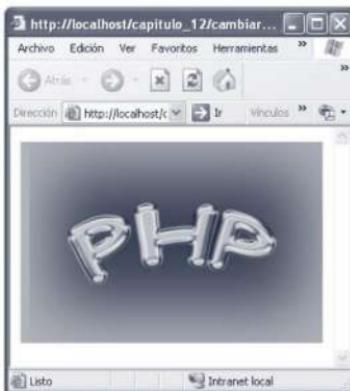


Figura 12.5

La paleta de colores puede tener un máximo de 256 colores. Si hay menos, se pueden añadir nuevos colores a la paleta con la función *imagecolorallocate()*. Esta recibe cuatro argumentos, según la siguiente relación:

- El manejador con el que se ha abierto la imagen.
- El valor de rojo que queremos darle al nuevo color.
- El valor de verde que queremos darle al nuevo color.
- El valor de azul que queremos darle al nuevo color.

El nuevo color se coloca en la primera posición que queda libre. La función devuelve el número de índice correspondiente a esa posición. Si no se ha podido ejecutar correctamente, se obtiene como resultado el valor -1.

Esta función se complementa con *imagecolordeallocate()*, cuya finalidad es eliminar un color de la paleta actual. Esta función recibe dos argumentos: el

manejador de la imagen y el número de índice que corresponde al color que queremos eliminar de la tabla. Pero recuerde. Esta función sólo puede eliminar un color previamente insertado con `imagecolorallocate()`. No se puede usar para eliminar los colores que forman parte de la paleta original.

La función `imagecolorallocate()` no permite establecer la transparencia de un color (el factor alfa). Para ello, tenemos la función `imagecolorallocatealpha()`. Ésta funciona igual que la anterior, pero recibiendo un quinto argumento, que es el nivel de transparencia que le queremos asignar al color. Dicho color podrá ser también eliminado con `imagecolordeallocate()`.

## 12.5 CREAR IMÁGENES

Hasta el momento hemos visto algunas posibilidades que nos ofrece PHP para manejar y mostrar imágenes existentes. Pero esto es sólo parte de lo que podemos hacer. El lenguaje nos permite crear imágenes nuevas, que no existen en el disco. Para ello usaremos la función `imagecreatetruecolor()`. Antes se usaba `imagecreate()`, que sigue existiendo por razones de compatibilidad. Pero la librería GD2 nos permite usar la nueva función, que nos ofrece la posibilidad de crear imágenes con *color verdadero* (8 bits por canal más alfa). Esta función recibe dos argumentos numéricos que representan la anchura y la altura de la nueva imagen, expresadas en píxeles. La función crea una nueva imagen sin contenido y devuelve un manejador para trabajar con ella. En algunos casos, la nueva función no da el resultado deseado. Si lo ocurre eso, use la función antigua que opera perfectamente.

Una vez creada la nueva imagen (que, aún, no muestra nada) deberemos crear una paleta de colores para ella. Esto lo haremos con la función `imagecolorallocate()`, que hemos comentado al final del apartado anterior. *El primer color que se asigne a la paleta será el color de fondo de la imagen.*

PHP nos permite trabajar con el color de los píxeles de una imagen de modo individual. Para ello necesitamos conocer las coordenadas x e y del pixel que nos interesa. Tenga en cuenta que las coordenadas x e y empiezan en 0, en la esquina superior izquierda de la imagen, contando hacia la derecha y hacia abajo. Para darle un determinado color a un píxel usaremos la función `imagesetpixel()`. ¿Se hace una idea de los argumentos que recibe esta función? Son los siguientes:

- El manejador de la imagen, obtenido con `imagecreatetruecolor()`.
- La coordenada x del píxel que nos interesa.
- La coordenada y del píxel que nos interesa.
- El índice que ocupa el color deseado en la paleta, y que ha sido obtenido mediante `imagecolorallocate()`.

Por último, una vez creada la imagen, podemos enviarla al navegador, o almacenarla en un archivo de disco, tal como aprendimos a hacerlo en el apartado 12.3 de este capítulo.

Para comprender la operativa conjunta de estas funciones vamos a crear una imagen nueva mediante un script. Se llama **crearNuevaImagen.php**, y su listado es el siguiente:

```
<?php
    header("Content-type: image/gif");
    // Se crea un manejador para una imagen nueva, de 400
    pixeles de ancho y 100 de alto.
    $imagen=imagecreatetruecolor (400, 100);
    // Se definen los colores que usaremos.
    $color[0]=imagecolorallocate ($imagen, 0, 0, 0);
    //Color negro
    $color[1]=imagecolorallocate ($imagen, 255, 255,
255); //Color blanco
    $color[2]=imagecolorallocate ($imagen, 255, 0, 0);
    //Color rojo
    $color[3]=imagecolorallocate ($imagen, 0, 255, 0);
    //Color verde
    $color[4]=imagecolorallocate ($imagen, 0, 0, 255);
    //Color azul
    $color[5]=imagecolorallocate ($imagen, 255, 255, 0);
    //Color amarillo
    $color[6]=imagecolorallocate ($imagen, 255, 0, 255);
    //Color morado
    $color[7]=imagecolorallocate ($imagen, 0, 255, 255);
    //Color cyan
    $color[8]=imagecolorallocate ($imagen, 127, 127,
127); //Color gris
    $color[9]=imagecolorallocate ($imagen, 255, 100,
255); //Color rosa
    // Ya están los colores en la paleta.
    /////////////////////////////////
    // Se recorren todos los pixeles de la nueva imagen,
    asignándoles colores de la paleta.
    for ($y=0; $y<100; $y++){
        for ($x=0; $x<400; $x++){
            $indiceDeColor=floor($y/10);
            imagesetpixel ($imagen, $x, $y,
$color[$indiceDeColor]);
        }
    }
    // Se graba la imagen en un archivo de disco.
    imagegif ($imagen, "imagenes/bandera.gif");
```

```
// Se muestra el resultado y se liberan recursos.  
imagegif ($imagen);  
imagedestroy ($imagen);  
?>
```

Como ve, este script hace uso de las funciones que hemos descrito, en el orden adecuado, para generar una imagen. Una vez creada ésta, primero se graba en el disco y luego es enviada al navegador. La parte de grabarla en el disco se podría haber omitido, si no fuera necesario conservar una copia de la imagen para usos posteriores. Por ejemplo, si creamos imágenes que representen estadísticas en tiempo real y no es necesario conservarlas, se pueden mostrar en el navegador, sin más.

Observe una cosa. Al crear la paleta de colores he usado una matriz para referirme a cada una de las tonalidades de la misma, a fin de facilitar luego el acceso en el bucle de una manera cómoda. El resultado de este script se ve en la figura 12.6.

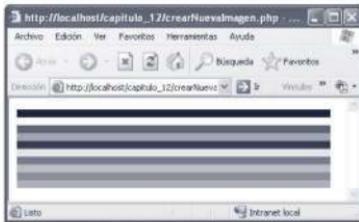


Figura 12.6

Sabiendo movernos por una imagen en términos de coordenadas x e y podemos determinar el color de un píxel de la imagen que nos interese. Para ello usaremos la función *imagecolorat()*, que recibe tres argumentos: el primero es el manejador de la imagen, y los otros dos corresponden a las coordenadas x e y del píxel que nos interesa. La función devuelve el índice que tiene ese color en la paleta empleada.

## 12.6 COPIA DE IMÁGENES

Podemos copiar una imagen, o un fragmento de ella, en otra imagen para conseguir resultados curiosos. Para ello usaremos la función *imagecopy()*. Esta función recibe, nada menos, que ocho parámetros, según se detalla en la lista que aparece a continuación:

- El manejador de la imagen de destino.
- El manejador de la imagen de origen.
- La coordenada x de la imagen de destino donde se iniciará la copia.
- La coordenada y de la imagen de destino donde se iniciará la copia.
- La coordenada x de la imagen de origen desde donde se empezará a copiar.
- La coordenada y de la imagen de origen desde donde se empezará a copiar.
- La anchura del fragmento de la imagen de origen que deseamos copiar.
- La altura del fragmento de la imagen de origen que deseamos copiar.

Para entender la mecánica de esta función nada mejor que crear un ejemplo que nos lo aclare. Vamos a partir de una de las imágenes que tenemos en el disco. Tomaremos un fragmento de la misma y lo copiaremos en otra imagen, nueva, que no será guardada, sino, solamente, enviada al navegador. Además, ese fragmento se copiará tres veces en la nueva imagen, en distintas posiciones (coordenadas), para que veamos cómo funciona.

Queremos el fragmento que aparece resaltado en la figura 12.7.



Figura 12.7

Queremos la parte central, la que aparece enmarcada en rojo. Así que, lo primero que tenemos que hacer es determinar sus coordenadas con respecto a la imagen. La esquina superior izquierda del fragmento que deseamos, con respecto a la esquina superior izquierda de la imagen tiene las coordenadas  $x=30$  y  $y=45$ . La anchura del fragmento es de 240 pixeles, y su altura es de 110 pixeles. El resultado final que queremos obtener aparece en la figura 12.8.



Figura 12.8

Observe el script **copiarFragmento.php**, listado a continuación:

```
<?php
    header("Content-type: image/gif");
    // Se crea el manejador de la imagen original.
    $imagenOriginal=imagecreatefromgif
("imagenes/lenguaje.gif");
    // Se crea un manejador para una imagen nueva, de 720
pixeles de ancho y 110 de alto.
    $imagenNueva=imagecreatetruecolor (721, 110);

    // Se copia el fragmento en tres posiciones diferentes
de la nueva imagen.
    imagecopy ($imagenNueva, $imagenOriginal, 0, 0, 30,
45, 240, 110);
    imagecopy ($imagenNueva, $imagenOriginal, 240, 0,
30, 45, 240, 110);
    imagecopy ($imagenNueva, $imagenOriginal, 480, 0,
30, 45, 240, 110);

    // Se muestra el resultado.
    imagegif ($imagenNueva);
    // Se liberan recursos.
    imagedestroy($imagenOriginal);
    imagedestroy($imagenNueva);
?>
```

Ejecute el script para ver el resultado. Observe, en las líneas resaltadas, cómo se ha copiado el fragmento elegido en tres posiciones diferentes de la nueva imagen, que es la que se muestra al final. Fíjese también que, al terminar el script, destruimos los dos manejadores que hemos usado, para liberar correctamente los recursos empleados.

Existe una variante de esta función que, además de permitir copiar un fragmento (o la totalidad) de una imagen, nos permite redimensionarla. Esto es especialmente útil cuando se cuenta con una galería de imágenes y hay que montar una página de miniaturas. Se trata de *imagecopyresized()*. La lista de parámetros que recibe empieza a parecerse a la de desempleados en España: nada menos que 10 (para una función, es bastante). Éstos son los siguientes:

- El manejador de la imagen de destino.
- El manejador de la imagen de origen.
- La coordenada x de la imagen de destino donde se iniciará la copia.
- La coordenada y de la imagen de destino donde se iniciará la copia.
- La coordenada x de la imagen de origen desde donde se copiará.
- La coordenada y de la imagen de origen desde donde se copiará.
- Anchura del fragmento copiado en la imagen de destino.

- Altura del fragmento copiado en la imagen de destino.
- La anchura del fragmento de la imagen de origen que deseamos copiar.
- La altura del fragmento de la imagen de origen que deseamos copiar.

Para comprobar cómo opera este mecanismo, vamos a escribir un script que toma la imagen que hemos usado antes, y crea una miniatura de la misma. El código, llamado **miniatura.php**, nos mostrará la miniatura en la página para que comprobemos el resultado.

```
<?php
    header("Content-type: image/gif");
    // Se crea el manejador de la imagen original.
    $imagenOriginal=imagecreatefromgif
("imagenes/lenguaje.gif");
    // Se crea un manejador para una imagen nueva, de 150
    píxeles de ancho y 100 de alto.
    $imagenNueva=imagecreatetruecolor (150, 100);
    // Se copia la imagen, miaturizada, en la nueva imagen.
    imagecopyresized ($imagenNueva, $imagenOriginal, 0,
0, 0, 0, 150, 100, 300, 200);
    // Se muestra el resultado.
    imagegif ($imagenNueva);
    // Se liberan recursos.
    imagedestroy($imagenOriginal);
    imagedestroy($imagenNueva);
?>
```

Observe, en la línea resaltada, cómo hemos hecho uso de esta función. Como le he comentado antes, esta función es imprescindible, cuando hay que crear miniaturas de imágenes, para no tener que hacerlo a mano. Lo suyo es crear la miniatura y guardarla, en otra carpeta, con el nombre de la imagen original.

Un comentario acerca de la copia de imágenes. El intérprete trata de copiar la paleta de la imagen original en la nueva imagen. Esto funciona casi siempre. Si, en algún caso concreto, no le funciona, cree una copia de la paleta usted mismo. Puede escribir un script que lo haga, leyendo cada uno de los colores de la paleta original y luego incluyéndolo en la nueva paleta mediante el uso de la función `imagecolorallocate()`, que ya conoce.

## 12.7 FIGURAS PREDEFINIDAS

Hasta ahora hemos aprendido a establecer el color de píxeles individuales. Sin embargo, las prestaciones de PHP para trabajar con imágenes van más allá de esto. El intérprete (y, más concretamente, la librería GD2) ofrece la posibilidad de

crear formas geométricas predefinidas mediante determinadas funciones que vamos a conocer en este apartado. Con la combinación de las distintas formas básicas se podrán lograr resultados muy interesantes, como veremos en este mismo capítulo.

Quizás la forma más simple, después de un pixel aislado, desde luego, es una linea recta. Para dibujarla contamos con la función *imagineLine()*, que recibe los siguientes parámetros:

- Manejador de la imagen en la que vamos a dibujar.
- Coordenada x del origen de la línea.
- Coordenada y del origen de la línea.
- Coordenada x del final de la línea.
- Coordenada y del final de la línea.
- Índice del color deseado en la paleta de colores.

El ancho de la línea es de un pixel. Si desea que su línea sea más gruesa, puede trazar varias líneas paralelas. También hay otra manera que veremos enseguida, en este mismo apartado.

Esta función tiene una alternativa llamada *imageDashedLine()*, que recibe los mismos argumentos y opera de modo similar. La diferencia es que ésta última traza una línea discontinua, mientras que la primera dibuja líneas continuas.

Con PHP podemos dibujar rectángulos. Para ello contamos con la función *imagerectangle()*. A esta función hay que darle las coordenadas de la esquina superior izquierda e inferior derecha del rectángulo. La forma geométrica obtenida es hueca, es decir, sólo obtenemos el borde, de un pixel de ancho, sin relleno. Aunque, como veremos, también hay alternativas para esto. Los argumentos que recibe esta función son los siguientes:

- El manejador de la imagen sobre la que vamos a trabajar.
- La coordenada x de la esquina superior izquierda del rectángulo.
- La coordenada y de la esquina superior izquierda del rectángulo.
- La coordenada x de la esquina inferior derecha del rectángulo.
- La coordenada y de la esquina inferior derecha del rectángulo.
- El índice del color deseado dentro de la paleta de la imagen.

No siempre querremos lineas rectas. PHP nos permite también dibujar circunferencias, elipses, o arcos; en definitiva, lineas curvas. Para ello, contamos con la función *imagearc()*. Ésta es muy configurable, según los parámetros que le pasemos. Así, podremos determinar si queremos una circunferencia, una ellipse vertical u horizontal o si queremos una linea curva cerrada o sólo un arco de la misma. Los argumentos que recibe esta función son los siguientes:

- El manejador de la imagen en la que vamos a dibujar.
- La coordenada x del centro de la circunferencia o elipse.
- La coordenada y del centro de la circunferencia o elipse.
- La anchura. Es el radio horizontal de la circunferencia o elipse. En el caso de una circunferencia, ambos parámetros son iguales.
- El ángulo en el que se empieza a dibujar, expresado en grados sexagesimales. El ángulo 0 es el punto más a la derecha de la circunferencia o elipse.
- El ángulo en el que se termina de dibujar. Los grados crecen en sentido de las agujas del reloj: si usted crea un arco, con el ángulo inicial a 0 y el final a 180, estará dibujando la mitad inferior de la circunferencia o la elipse.
- El índice de color en la paleta de la imagen.

Si quiere dibujar una linea curva cerrada puede usar, alternativamente, la función *imageellipse()*. Ésta recibe los siguientes argumentos:

- El manejador de la imagen en la que vamos a dibujar.
- La coordenada x del centro de la circunferencia o elipse.
- La coordenada y del centro de la circunferencia o elipse.
- La anchura. Es el radio horizontal de la circunferencia o elipse.
- La altura. Es el radio vertical de la circunferencia o elipse. En el caso de una circunferencia, ambos parámetros son iguales.
- El índice de color en la paleta de la imagen.

Como ve, esta función no requiere ángulo de inicio ni de final, ya que dibuja una figura completa, no sólo un arco, así que, en muchos casos, es más cómoda de usar. Esta función se incorporó a PHP a partir de la versión 4.0.6, aunque en los manuales antiguos de PHP 4 no suele aparecer.

También podemos crear formas poligonales. Para ello usamos la función *imagepolygon()*. Ésta recibe los siguientes argumentos:

- El manejador de la imagen en la que vamos a dibujar.
- Una matriz indexada con las coordenadas de todos los vértices del polígono. Los dos primeros elementos contendrán las coordenadas x e y del primer vértice. Los elementos tercero y cuarto contendrán las coordenadas x e y del segundo vértice y, así, sucesivamente.
- El número total de vértices. PHP se encarga de trazar las líneas entre los mismos, y una linea final uniendo el último con el primero, cerrando la línea.
- El índice del color deseado dentro de la paleta de la imagen.

Las formas creadas hasta ahora son huecas, como ya he mencionado anteriormente. Están formadas por un borde de un solo píxel de ancho. Si alguna forma queremos rellenarla de color usaremos la función *imagefill()*. La operativa es la siguiente. Se le asigna a la función un punto de la imagen y, a partir de ahí, rellena, en el color establecido, hasta que encuentre unos bordes que delimitan la zona de relleno. Es decir, se usa para llenar figuras cerradas. Si la figura no está cerrada se rellena con el color indicado toda la imagen.

Esta función recibe los siguientes argumentos:

- El manejador de la imagen con la que vamos a trabajar.
- La coordenada x del punto donde se inicia el relleno.
- La coordenada y del punto donde se inicia el relleno.
- El índice del color deseado dentro de la paleta de la imagen.

Si quiere dibujar rectángulos o polígonos llenados del mismo color que el borde que los delimita, puede usar las funciones *imagefilledrectangle()*, *imagefilledpolygon()* o *imagefilledellipse()*, que reciben los mismos argumentos que *imagerectangle()*, *imagepolygon()* e *imageellipse()*, respectivamente.

*Una consideración que debe tener siempre en cuenta es que todas las coordenadas que reciben estas funciones se toman siempre con respecto a la esquina superior izquierda de la imagen (0, 0).* La mejor forma de conocer realmente estas funciones es, desde luego, crear un script que las use y nos muestre el resultado. El script se llama **formas.php**, y su listado es el siguiente:

```
<?php
    header("Content-type: image/gif");
    // Se crea un manejador para una imagen nueva, de 200
    pixeles de ancho y 200 de alto.
    $imagen=imagecreate (200, 200);
    // Se definen los colores que usaremos.
    $blanco=imagecolorallocate ($imagen, 255, 255, 255);
//Color de fondo
    $negro=imagecolorallocate ($imagen, 0, 0, 0);
    $rojo=imagecolorallocate ($imagen, 255, 0, 0);
    $azul=imagecolorallocate ($imagen, 0, 0, 255);
// Ya están los colores en la paleta.
///////////////////////////////
// Dibujamos una línea vertical continua.
    imageline ($imagen, 10, 10, 10, 190, $negro);
// Dibujamos una línea vertical discontinua.
    imagedashedline ($imagen, 20, 10, 20, 190, $negro);
// Dibujamos un rectángulo azul hueco.
    imagerectangle ($imagen, 30, 10, 80, 50, $azul);
```

```
// Dibujamos un rectángulo azul hueco y lo rellenamos
de rojo.
    imagerectangle ($imagen, 30, 70, 80, 110, $azul);
    imagefill ($imagen, 35, 80, $rojo);
// Dibujamos un rectángulo azul relleno.
    imagefilledrectangle ($imagen, 30, 130, 80, 170,
$azul);
// Dibujamos una circunferencia negra rellena.
    imagefilledellipse ($imagen, 150, 50, 40, 40,
$negro);
// Dibujamos un triángulo azul relleno.
// Empezamos por crear la matiz donde se almacenan los
tres vértices.
    $vertices[0]=150;
    $vertices[1]=100;
    $vertices[2]=180;
    $vertices[3]=180;
    $vertices[4]=120;
    $vertices[5]=180;
// Ahora se dibuja el triángulo.
    imagepolygon ($imagen, $vertices, 3, $azul);
// Se muestra el resultado.
    imagejpeg ($imagen);
// Se liberan recursos.
    imagedestroy($imagen);
?>
```

Como ve, hemos creado una imagen nueva. Aunque todas las funciones que hemos visto pueden dibujar sobre imágenes ya existentes, de este modo veremos los resultados más claros. Pruebe el script, para ver los resultados. No hemos usado todas las posibles formas geométricas, para no extendernos mucho. Sólo algunas, para que usted pueda comprobar su operatividad. Experimente usted con éstas y con las restantes. Por cierto. Antes le mencioné que existía una forma de crear líneas de más de un pixel de ancho. Se puede usar la función `imagefilledrectangle()` para crear un rectángulo largo y estrecho que aparecerá como una línea gruesa de trazado ancho. Naturalmente, esto sólo vale si la línea es horizontal o vertical.

## 12.8 FILTROS

A una imagen se le pueden aplicar filtros para conseguir determinados efectos, similares a los que se logran con algunas aplicaciones de retoque de imágenes. Para ello contamos con la función `imagefilter()`, que recibe dos argumentos. El primero de ellos es, cómo no, el manejador de la imagen sobre la que vamos a trabajar. El segundo es una constante que determina el tipo de filtro

que queremos aplicar. Algunos de estos filtros requieren, para su funcionamiento, hasta tres parámetros adicionales.

Vamos a ver, a modo de ejemplo, unos scripts muy sencillitos. Luego comentaremos los posibles filtros que ofrece la librería GD2. Empezaremos por uno que convierte una imagen en color a escala de grises. El script se llama **grises.php**, y su listado es el siguiente.

```
<?php
    header("Content-type: image/gif");
    // Se crea un manejador para una imagen.
    $imagen =
imagecreatefromjpeg("imagenes/lenguaje.jpg");

    // Se aplica el filtro de escala de grises.
    imagefilter($imagen, IMG_FILTER_GRAYSCALE);
    // Se muestra la imagen.
    imagejpeg ($imagen);
    // Se liberan recursos.
    imagedestroy($imagen);
?>
```

Observe, en la línea resaltada, cómo se ha usado la función, con la constante que establece que se transforme la imagen a escala de grises. Pruebe el script para verificar su correcto funcionamiento. Vamos a ver otro ejemplo. El script **coloriado.php** toma una imagen y la cubre de un filtro de color verde, con lo que se tiñen con este tono los colores originales.

```
<?php
    header("Content-type: image/gif");
    // Se crea un manejador para una imagen.
    $imagen =
imagecreatefromjpeg("imagenes/lenguaje.jpg");

    // Se aplica el filtro de escala de grises.
    imagefilter($imagen, IMG_FILTER_COLORIZE, 0, 255, 0);
    // Se muestra la imagen.
    imagejpeg ($imagen);
    // Se liberan recursos.
    imagedestroy($imagen);
?>
```

Pruébelo para ver el resultado. Vea, en la línea resaltada, cómo se usan los argumentos numéricos cuando son necesarios. En el caso de este filtro, son tres, pero en otros sólo será uno. Vea las descripciones más abajo.

Como último ejemplo, usaremos un filtro que elimina el suavizado de los detalles, dando un efecto de boceto. El listado de **sketchy.php** es similar a los anteriores, sólo que la línea de filtro es la siguiente:

```
imagefilter($imagen, IMG_FILTER_MEAN_REMOVAL);
```

Los filtros que podemos emplear aparecen en la tabla de la figura 12.9.

FILTROS PARA IMÁGENES	
FILTRO	EFFECTO Y COMENTARIOS
<i>IMG_FILTER_NEGATE</i>	Crea el negativo de una imagen.
<i>IMG_FILTER_GRAYSCALE</i>	Convierte una imagen a escala de grises.
<i>IMG_FILTER_BRIGHTNESS</i>	Cambia el brillo de una imagen. Utilice un argumento numérico, a continuación de la constante del filtro, para establecer el nivel de brillo.
<i>IMG_FILTER_CONTRAST</i>	Cambia el contraste de una imagen. Utilice un argumento numérico, a continuación de la constante del filtro, para establecer el nivel de contraste.
<i>IMG_FILTER_COLORIZE</i>	Crea un filtro de color para la imagen. Utilice tres argumentos numéricos para establecer los valores de rojo, verde y azul del filtro.
<i>IMG_FILTER_EDGEDETECT</i>	Resalta los bordes de la imagen. Llamamos bordes a aquellas zonas donde se producen cambios de color.
<i>IMG_FILTER_EMBOSS</i>	Crea un efecto de relieve.
<i>IMG_FILTER_GAUSSIAN_BLUR</i>	Desenfoque gausiano de la imagen.
<i>IMG_FILTER_SELECTIVE_BLUR</i>	Desenfoque de la imagen.
<i>IMG_FILTER_MEAN_REMOVAL</i>	Efecto boceto.
<i>IMG_FILTER_SMOOTH</i>	Suaviza la imagen. Utilice un argumento numérico para establecer el nivel de suavizado.

Figura 12.9

Pruebe los distintos filtros, modificando alguno de los scripts que tiene en el CD. Use distintos argumentos numéricos en aquellos filtros que lo admitan, para ver los resultados.

## 12.9 TEXTO EN LAS IMÁGENES

Otro efecto interesante es incorporar texto en las imágenes, a modo de notas aclaratorias, o para cualquier fin publicitario, etc. PHP nos proporciona algunas funciones muy interesantes.

La función *imagestring()* permite insertar una cadena de texto en la posición deseada en una imagen. Los argumentos que recibe son los siguientes:

- El manejador de la imagen a la que le vamos a añadir el texto.
- El tamaño de la fuente a emplear, de 1 a 5, siendo el 1 el más pequeño y el 5 el más grande.
- La coordenada x donde se inicia el texto.
- La coordenada y donde se inicia el texto.
- La cadena de texto que queremos incluir en la imagen.
- El índice de la paleta de colores de la imagen que corresponde al color deseado.

Como complemento a esta función contamos con *imagestringup()*, que coloca una cadena de texto en vertical, en lugar de horizontalmente. La sintaxis es la misma que la de la anterior.

Para ver cómo podemos usar estas funciones, hemos creado un script que muestra la media mensual de temperaturas de un año en una ciudad cualquiera, que podría ser la suya. Se llama **grafica.php**, y su listado es el siguiente:

```
<?php
    header("Content-type: image/gif");
    // Se crea un manejador para una imagen nueva, de 320
    // pixeles de ancho y 220 de alto.
    $imagen=imagecreate (320, 220);
    // Se definen los colores que usaremos.
    $blanco=imagecolorallocate ($imagen, 255, 255, 255);
    //Color de fondo
    $negro=imagecolorallocate ($imagen, 0, 0, 0);
    $rojo=imagecolorallocate ($imagen, 255, 0, 0);
    // Ya están los colores en la paleta.
    /////////////////////////////////
    // Se dibuja el rótulo vertical de TEMPERATURAS.
    imagestringup ($imagen, 4, 10, 135, "TEMPERATURAS",
    $negro);
    // Se dibujan las graduaciones de las temperaturas.
    imagestring ($imagen, 3, 40, 10, "50", $negro);
    imagestring ($imagen, 3, 40, 30, "40", $negro);
```

```
imagestring ($imagen, 3, 40, 50, "30", $negro);
imagestring ($imagen, 3, 40, 70, "20", $negro);
imagestring ($imagen, 3, 40, 90, "10", $negro);
imagestring ($imagen, 3, 47, 110, "0", $negro);
imagestring ($imagen, 3, 33, 130, "-10", $negro);
imagestring ($imagen, 3, 33, 150, "-20", $negro);

// Se dibuja el rótulo horizontal de MESES.
imagestring ($imagen, 4, 160, 190, "MESES", $negro);
// Se dibujan las iniciales de los meses.
imagestring ($imagen, 3, 70, 170, "E", $negro);
imagestring ($imagen, 3, 90, 170, "F", $negro);
imagestring ($imagen, 3, 110, 170, "M", $negro);
imagestring ($imagen, 3, 130, 170, "A", $negro);
imagestring ($imagen, 3, 150, 170, "M", $negro);
imagestring ($imagen, 3, 170, 170, "J", $negro);
imagestring ($imagen, 3, 190, 170, "J", $negro);
imagestring ($imagen, 3, 210, 170, "A", $negro);
imagestring ($imagen, 3, 230, 170, "S", $negro);
imagestring ($imagen, 3, 250, 170, "O", $negro);
imagestring ($imagen, 3, 270, 170, "N", $negro);
imagestring ($imagen, 3, 290, 170, "D", $negro);

// Se dibujan los once segmentos de línea que forman la
gráfica.
imageline ($imagen, 70, 140, 90, 110,$rojo);
imageline ($imagen, 90, 110, 110, 90,$rojo);
imageline ($imagen, 110, 90, 130, 80,$rojo);
imageline ($imagen, 130, 80, 150, 60,$rojo);
imageline ($imagen, 150, 60, 170, 60,$rojo);
imageline ($imagen, 170, 60, 190, 50,$rojo);
imageline ($imagen, 190, 50, 210, 30,$rojo);
imageline ($imagen, 210, 30, 230, 50,$rojo);
imageline ($imagen, 230, 50, 250, 60,$rojo);
imageline ($imagen, 250, 60, 270, 90,$rojo);
imageline ($imagen, 270, 90, 290, 110,$rojo);
// Se muestra el resultado.
imagejpeg ($imagen);
// Se liberan recursos.
imagedestroy($imagen);
?>
```

Por supuesto, el script es, conceptualmente, muy simple. Lo que quiero decir es que, si esto fuera real, las coordenadas con las que se han trazado las líneas de la gráfica procederían de otras fuentes, tales como una base de datos, tema que veremos en los capítulos 15, 16 y 17 del libro. También podríamos haber optimizado el dibujo de las temperaturas mediante un bucle. Pero esto no tiene

relevancia ahora. He preferido que quedara muy claro lo que hemos hecho, con fines didácticos. Con el tiempo, cuando usted tenga soltura en escribir scripts para sus páginas, irá descubriendo la forma de optimizar el código para lograr los mismos o mejores resultados con menos líneas, y de obtener los datos de forma adecuada.

Vea, en las líneas resaltadas, cómo hemos usado estas funciones. Al ejecutar el script obtendrá el resultado de la figura 12.10.

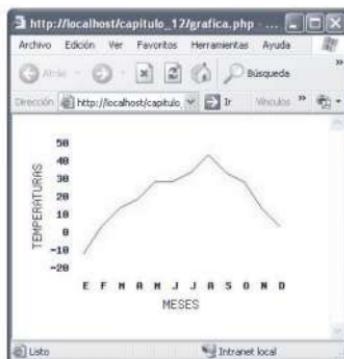


Figura 12.10

PHP también permite incluir tipografías True Type (ttf) en las imágenes. Esto es lógico. La fuente que emplea por defecto el intérprete para escribir en las imágenes, y que hemos usado en el ejemplo anterior es bastante pobre para efectos publicitarios o decorativos. Para incluir otras tipografías en una imagen contamos con la función *imagefttext()*. Los argumentos que necesita para trabajar son los siguientes:

- El manejador de la imagen en la que se va a incluir un rótulo.
- El tamaño de la fuente, en puntos.
- El ángulo con el que deberá quedar el rótulo respecto a la horizontal. Si el ángulo es 0, el rótulo quedará totalmente horizontal. Valores positivos de este parámetro rotan el texto en sentido contrario a las agujas del reloj.
- La coordenada x donde empieza el texto.
- La coordenada y donde empieza el texto.
- El índice de la paleta de colores de la imagen que corresponde al color en el que queremos el texto.

- Una cadena con el nombre y, en su caso, la ruta del archivo que contiene la tipografía. Los archivos de tipografías deben ser accesibles para el script. Lo normal es que se coloquen en el servidor, junto con las páginas, o en una carpeta específica.
- La cadena de texto que queremos incluir en la imagen.

Fíjese en la descripción de los argumentos. Es necesario que usted disponga de los archivos ttf con las fuentes que desee incorporar, pero no es preciso que el usuario que se conecte a su página los tenga, para visualizar correctamente la imagen. Basta con que estén en el servidor donde se alojan sus páginas. Sin embargo, aquí se presenta un pequeño escollo. Muchos archivos ttf tienen derechos de autor. Usted deberá incluir tipografías libres de derechos, o bien contar con el permiso para usar las que incluya.

Suponga un script con una línea como la siguiente:

```
imagettfttext ($imagen, 30, 15, 50, 100, $rojo,  
"fuentes/suave.ttf", "Texto en una imagen");
```

Si usted ha creado una imagen en blanco, y ha definido los colores adecuados en la paleta, como ya hemos hecho en otros ejemplos, podría obtener un resultado como el de la figura 12.11.



Figura 12.11

Naturalmente, tendrá que tener la fuente en la ruta especificada. Y, por si se lo está preguntando, puede incluir estos rótulos en cualquier imagen que ya tenga, no sólo en las que usted cree nuevas.

## CORREO ELECTRÓNICO

---

---

PHP permite el envío de correos electrónicos, como no podía ser menos en un lenguaje diseñado para trabajar en el lado del servidor. Esto es extremadamente útil. Si usted, por ejemplo, crea un foro, es normal que los usuarios puedan intercambiarse correos, o que el propio script les envíe avisos por e-mail cuando haya respuestas a los posts. En otros tipos de aplicaciones, como sitios de comercio electrónico o portales de contactos, también es necesario escribir a los usuarios.

### 13.1 CORREO SENCILLO

En PHP contamos con la función *mail()* para enviar correos electrónicos. Ésta recibe cinco parámetros, separados por comas, de los que tres son obligatorios y los otros dos son opcionales. Los tres primeros son: la dirección del destinatario, el asunto del mensaje y el texto del mismo, por ese orden. Así pues, la sintaxis general de la función será la siguiente:

```
mail ($destinatario, $asunto, $mensaje);
```

Si el envío se ha realizado correctamente la función devuelve un valor true, en caso de que no se pueda efectuar, devolverá un valor false.

Vamos a ver un ejemplo de uso de esta función. Para que todo vaya correctamente usted tiene que tener configurado su servidor de correo electrónico como le expliqué en el capítulo 2. Además, deberá tener su cliente de correo con la cuenta configurada como le indiqué al final de dicho capítulo. El primer script que vamos a ver aquí es **mailSimple.php**, listado a continuación:

```
<?php
    // Se declaran los tres parámetros que se usarán en el
    correo.
    $destinatario="pruebas@localhost.com";
    $asunto="Correo de prueba.";
    $mensaje="Esto es un correo para probar la función
mail() de PHP.";
    // Se intenta enviar el correo y se muestra un mensaje
en la página con el resultado.
    if (mail ($destinatario, $asunto, $mensaje)){
        echo ("MENSAJE ENVIADO");
    } else {
        echo ("ENVIO FALLIDO");
    }
?>
```

Observe, en la línea resaltada, el uso de la función mail. Los argumentos son tres variables que hemos definido anteriormente. Cuando cargue el script, si todo ha ido correctamente, en la página le aparecerá el texto **MENSAJE ENVIADO**. Al abrir su cliente de correo deberá encontrarse con un mensaje con el asunto y el contenido que hemos creado. El remitente es **me@localhost.com**. Esta es una cuenta que crea por defecto el servidor de correo para hacer los envíos.

El cuarto parámetro, opcional, está reservado a cabeceras adicionales, como una dirección de remitente, una dirección de respuesta, una dirección para copias del mensaje (CC), etc. Vea el script **mailConCabeceras.php**:

```
<?php
    // Se declaran los tres parámetros que se usarán en el
    correo.
    $destinatario="pruebas@localhost.com";
    $asunto="Correo de prueba.";
    $mensaje="Esto es un correo para probar la función
mail() de PHP.";
    $cabeceras="From:
usuario@localhost.com."\r\n."Reply-To:
webmaster@localhost.com";
    // Se intenta enviar el correo y se muestra un mensaje
en la página con el resultado.
    if (mail ($destinatario, $asunto, $mensaje,
$cabeceras)){
        echo ("MENSAJE ENVIADO");
    } else {
        echo ("ENVIO FALLIDO");
    }
?>
```

En la primera linea resaltada se ha declarado una variable con unas cabeceras adicionales. Fíjese en cómo está construida. La sintaxis es muy simple. Separamos el nombre de cada cabecera de su valor con el signo dos puntos (:), y cada cabecera de la siguiente con la secuencia de salto de linea. Cuando ejecute este script recibirá un mensaje en el que le aparecerá el remitente que ha indicado y, si pulsa el botón de respuesta de su cliente de correo, verá el destinatario de la respuesta tal como ha establecido en la cabecera.

El quinto parámetro, opcional, está reservado para comandos de Unix (suponiendo que ésta sea su plataforma servidora). Aquí no vamos a hablar de ello porque los comandos de Unix están fuera de las pretensiones de este libro.

Ya podemos incorporar esta funcionalidad a nuestras páginas. Usted puede crear un formulario que recopile los datos necesarios del cliente y los envíe a un script que, a su vez, use la función mail () para mandar un correo al destinatario.

## 13.2 CORREO COMPLEJO

La función mail () está bastante bien concebida, para la época en que fue diseñada. Después de todo, forma parte de PHP desde sus primeras versiones. Pero no nos engañemos. Para un uso práctico hoy día es un poco pobre. ¿Ha visto usted algunos de los e-mails que recibe cada día en su buzón? Tienen un formato HTML, con imágenes, texto con distintas apariencias y, en ocasiones, implementan funcionalidades como un formulario o enlaces. Enviar un e-mail así no es posible con la función mail (). Pero hay otras soluciones. En el capítulo 11 le prometí que, al hablar de correo electrónico, veríamos la utilidad de las clases y la POO. Y, como lo prometido es deuda, vamos a conocer una clase muy interesante para el envío de correos electrónicos: se trata de **phpMailer**, de Brent R. Matzelle, que permite enviar correos sofisticados con mucha comodidad. La clase está formada por dos scripts en PHP: **class.phpmailer.php** y **class.smtp.php**. Aunque estas clases están incluidas en el CD adjunto al libro, podemos visitar la web <http://easynews.dl.sourceforge.net/sourceforge/phpmailer/phpmailer-1.73.zip>, en donde bajaremos las dos clases en un comprimido. Estas clases se distribuyen bajo licencia GNU. El texto original de la licencia se encuentra también en el CD. En resumen, la licencia GNU para software libre dice que:

- Usted puede usar el producto (en este caso las dos clases que estamos estudiando) como mejor convenga a sus necesidades.
- Puede modificarlo, si lo desea. El código fuente está a su disposición.
- Si efectúa alguna modificación o mejora que usted estime interesante, puede, si lo desea, hacerla pública.

- Puede transmitir el producto a otros usuarios, bien sea de forma onerosa o gratuita.
- La única obligación que usted contrae es que, en caso de transmitir el producto a terceros, debe darles a conocer los términos de la licencia GNU e informarles de que ellos disfrutan, a su vez, de los mismos derechos que usted.

Nosotros incluiremos en nuestros scripts, con include () o require (), el primero de los archivos (class.phpmailer.php) ya que, éste, a su vez, incluye el segundo. En el CD que acompaña a este libro le incluyo la traducción completa al español del manual original de la clase. Aquí vamos a conocer su uso mediante un ejemplo práctico.

Lo primero que debemos hacer es diseñar el mensaje que queremos que le llegue al usuario. Aquí es donde más esfuerzo tendrá que hacer. No olvide que está diseñando un mensaje para enviar por correo electrónico. A ningún usuario le gusta recibir un mail promocional, informativo, publicitario ni de ninguna otra índole que tarda una hora en descargarse y que le satura el buzón. Así pues, nuestro e-mail deberá usar los recursos que HTML ofrece, pero sin recargar innecesariamente el resultado final. Observe la figura 13.1.



Figura 13.1

Este será el e-mail que les enviemos a nuestros usuarios. Lo primero que hacemos es escribir el código HTML, como si fuera una página web. Además, tendremos que disponer de las imágenes necesarias. En la carpeta del capítulo 13 hay un directorio, llamado **imagen**, que contiene el logotipo que vamos a usar.

A continuación, abriremos el código HTML de la página y sustituiremos todas las comillas dobles que hay por comillas simples. A efectos de HTML esto no tiene ninguna trascendencia, pero para luego incluir el mensaje en un listado PHP si es importante. Una vez hecho esto, llevaremos a cabo una pequeña

modificación donde haya que insertar una imagen. En concreto, en nuestro ejemplo la línea HTML siguiente será modificada:

```
<img src='imagen/cabeceraCorreo.gif' width='220'  
height='85'>
```

Sustituiremos esta linea por:

```
<img src='cid:logotipo' width='220' height='85'>
```

Lo que hacemos es cambiar el origen de las imágenes por **cid**: y un alias para la imagen. Esto es por el funcionamiento propio de la clase que vamos a usar. Enseguida verá usted de dónde sale ese alias. Si el mensaje tiene diferentes imágenes, cada una debe ser referida por su propio alias.

Ahora vamos a poner las etiquetas HTML, el texto, y todo lo que forme parte del futuro mensaje como contenido de una variable en un archivo PHP. En la carpeta lo he grabado como **mensaje.php** y su listado es el siguiente:

```
<?php  
$contenidoDelMensaje = "<html><head><title>SALUDO</title>  
</head>";  
$contenidoDelMensaje .= "<body><table width='400'  
border='0' cellspacing='0' cellpadding='4'>";  
$contenidoDelMensaje .= "<tr><td  
width='164'><strong><font face='Arial, Verdana, Tahoma'>";  
$contenidoDelMensaje .= "Bienvenid@ al primer portal de  
contactos totalmente gratuito.</font></strong></td>";  
$contenidoDelMensaje .= "<td width='220'><img  
src='cid:logotipo' width='220' height='85'></td>";  
$contenidoDelMensaje .= "</tr><tr><td>&nbsp;</td><td>&nbs  
p;</td></tr><tr>";  
$contenidoDelMensaje .= "<td colspan='2'><strong><font  
face='Arial, Verdana, Tahoma'>Para conocernos mejor haz  
click:</font></strong>";  
$contenidoDelMensaje .= "&nbsp;<a  
href='http://www.todoligues.com'>aqu&iacute;</a>.</font></str  
ong></td></tr>";  
$contenidoDelMensaje .= "</table></body></html>";  
?>
```

Vea que tenemos una variable para incluir todo el HTML que forma el mensaje. Fíjese en la línea resaltada, donde se incluye la imagen. De momento guardamos y cerramos este archivo. A continuación, vamos a crear un script que lo envíe por correo electrónico. Éste se llamará **mailHTML.php**.

```
<?php
/* Incluimos el fichero que contiene el mensaje que le queremos enviar al usuario.*/
include ("mensaje.php");
/* Incluimos el archivon de phpMailer.*/
include ("class.phpmailer.php");
/* Implementamos un nuevo objeto de la clase phpMailer.
*/
$correo = new PHPMailer();
/* A continuación se establecen llos valores de algunas propiedades del objeto, y se invoca a alguno de sus métodos, para preparar el mensaje antes de enviarlo al usuario.*/
/* En primer lugar le indicamos al objeto que el mensaje es HTML, mediante el método al efecto.*/
$correo->IsHTML(true);
/* Indicamos cual va a ser el servidor SMTP.*/
$correo->host = "localhost.com";
/* Indicamos el nombre del remitente.*/
$correo->From = "Remitente del correo";
/* Indicamos la dirección de correo del destinatario.
*/
$correo->AddAddress("pruebas@localhost.com");
/* Aquí incluimos la imagen que aparecerá en el mensaje, con su alias, separando ambos parámetros con una coma. Si nuestro mensaje lleva más de una imagen (no se lo aconsejo), invoque al método AddEmbeddedImage tantas veces como sea necesario, y dele a cada imagen un alias único.*/
$correo-
>AddEmbeddedImage("imagen/cabeceraCorreo.gif",logotipo);
/* El asunto del mensaje.*/
$correo->Subject = "MENSAJE PUBLICITARIO";
/* La dirección de correo del remitente.*/
$correo->FromName = "webmaster@todoligues.com";
/* El cuerpo del mensaje, que es la variable que hemos definido en el archivo mensaje.php.*/
$correo->Body = $contenidoDelMensaje;
/* El máximo de caracteres por línea.*/
$correo->WordWrap = 50;
/* Se intenta enviar el mensaje con el método Send del objeto y se informa al usuario del resultado.*/
if ($correo->Send()){
    echo ("MENSAJE ENVIADO CORRECTAMENTE.");
} else {
    echo ("EL ENVÍO HA FALLADO.");
}
?>
```

Cargue este script en el navegador y, a continuación, abra su cliente de correo, para ver cómo ha recibido el mensaje. Como ve, los resultados son mucho más satisfactorios que el envío de un correo de texto plano con mail(). En la figura 13.2 puede ver un ejemplo de cómo recibe el correo el usuario con el cliente Outlook Express, de Microsoft. Si analiza el script verá que lo que hacemos es implementar un objeto de la clase phpMailer, establecer algunas propiedades, ejecutar determinados métodos y, una vez preparado el objeto, lo enviamos con el método Send(). Lea los comentarios que he incluido en el script, y la traducción del manual de la clase, que está en el CD, para conocer lo necesario sobre esta clase. Use el ejemplo que le he puesto para efectuar distintas pruebas. Por supuesto, con phpMailer puede enviar correos ASCII, pero sería infructuoso utilizar la clase.

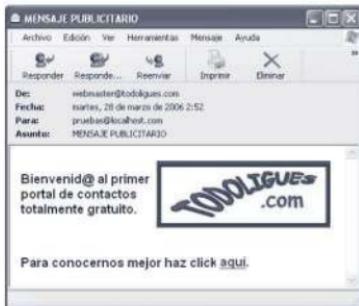


Figura 13.2

En el mundo PHP existen muchas clases que, como ésta, pueden ser usadas libremente y nos facilitan el trabajo diario de modo increíble. En el Apéndice D encontrará algunas direcciones de Internet donde podrá recopilar material muy útil.

Déjeme hacerle una observación. Cuando suba a Internet sus páginas, el servidor donde las aloje deberá tener implementado un servicio de SMTP. Normalmente, en plataformas Linux se emplea SendMail. En plataformas basadas en un sistema operativo propietario (Windows, para entendernos) se suelen emplear aplicaciones comerciales que, no vamos a engañarnos, encarecen el producto final. En cualquier caso, deberá estar implementado este servicio para que sus scripts puedan enviar correos electrónicos.

Antes de pasar al siguiente capítulo, lea el manual de la clase phpMailer en el CD e imprímalo para guardarla para futuras consultas.

## CAPÍTULO 14

### MISCELÁNEA

---

---

He reservado para este capítulo aquellos conceptos que, por una u otra razón, no tienen un capítulo propio, pero que son importantes y, además, no son fáciles de encontrar en la mayoría de la bibliografía existente. Estoy seguro de que los contenidos que hay aquí le resultarán muy útiles a lo largo de su camino como programador para la web.

#### 14.1 ERRORES

Cuando se escribe un script, a menudo se producen errores: una división en la que el divisor es cero, una llamada a un método o propiedad inexistente en un objeto, etc. Muchos son fáciles de detectar y reparar, como errores sintácticos, (palabras clave mal escritas), pero otros no lo son tanto. A veces, los errores generan mensajes de aviso (Warning) o de error (Fatal error). En el caso de los primeros, la ejecución del script, normalmente, continúa, aunque los resultados pueden no ser correctos. En el segundo caso, se suele interrumpir la ejecución. No obstante, el peor escenario posible, pesadilla de todos los programadores, es aquél en el que no se produce, por defecto, ningún mensaje de aviso, pero, no obstante, hay un error en la lógica de tratamiento de nuestros datos. Y, claro, los resultados pueden no ser los esperados. En ocasiones, la diferencia entre los resultados que deberíamos obtener y los que realmente obtenemos es muy sutil. Detectar y localizar ese tipo de errores es, usando una expresión popular, “tarea de chinos”. PHP nos facilita un poco la tarea de capturar esos fallos que pueden hacernos perder horas. Vamos a conocer la función *error\_reporting()*. Esta función recibe un argumento formado por una o más constantes, según el tipo de errores que queramos detectar. Estas constantes aparecen en la tabla de la figura 14.1.

CONSTANTES DE ERROR		
VALOR	CONSTANTE	ERRORES DETECTADOS
1	E_ERROR	Errores fatales en tiempo de ejecución. Se detiene la ejecución del script.
2	E_WARNING	Avisos en tiempo de ejecución. La ejecución del script no se detiene.
4	E_PARSE	Errores como la falta de un punto y coma, paréntesis que no se cierran, etc.
8	E_NOTICE	Notificaciones en tiempo de ejecución.
16	E_CORE_ERROR	Errores fatales que ocurren durante el arranque inicial de PHP.
32	E_CORE_WARNING	Advertencias (errores no-fatales) que ocurren durante el arranque inicial de PHP.
64	E_COMPILE_ERROR	Errores fatales en tiempo de compilación. Es como un E_ERROR, excepto que es generado por el Motor de Scripting de Zend. <sup>(1)</sup>
128	E_COMPILE_WARNING	Advertencias en tiempo de compilación (errores no fatales). Es como un E_WARNING, excepto que es generado por el Motor de Scripting de Zend. <sup>(1)</sup>
256	E_USER_ERROR	Mensaje de error generado por el usuario.
512	E_USER_WARNING	Mensaje de advertencia generado por el usuario.
1024	E_USER_NOTICE	Anotación generada por el usuario.
2047	E_ALL	Todos los errores y advertencias excepto los correspondientes a la constante E_STRICT.
2048	E_STRICT	Noticias de tiempo de ejecución. Habilite este valor para hacer que PHP sugiera cambios en su código que velarán por el mejor uso y la compatibilidad de su código.

<sup>(1)</sup> La tecnología Zend se emplea para compilar scripts de PHP para usos específicos y queda fuera del alcance de este volumen.

Figura 14.1

La mejor forma de detectar cualquier posible error es usar las dos constantes de mayor valor. Para incluir dos o más constantes como argumento de la función éstas deben ir separadas por una barra vertical. Por ejemplo, suponga que quiere fijar los niveles de detección E\_WARNING, E\_PARSE y E\_NOTICE. Observe la siguiente línea:

```
error_reporting (E_WARNING | E_PARSE | E_NOTICE);
```

La más completa comprobación de cualquier posible incidencia, no obstante, se obtiene con la siguiente instrucción:

```
error_reporting (E_STRICT);
```

Empiece con ella sus scripts, hasta que los pruebe y funcionen perfectamente. Cuando haya logrado el resultado deseado borre la línea, ya que, con esos niveles de detección de error, lo normal es que durante la ejecución se produzcan mensajes de aviso que los usuarios no deben ver.

## 14.2 EVALUAR EXPRESIONES

PHP nos permite evaluar expresiones que contengan instrucciones válidas del lenguaje mediante *eval ()*. Por ejemplo, suponga que en un script usted tiene una línea como la siguiente:

```
eval ("echo (\\"Hola\\");") ;
```

Esto equivaldría a:

```
echo ("Hola");
```

Si lo prueba en un script verá que funciona correctamente. Pero claro. Al utilizar eval () hemos tecleado más código y la línea resulta más enrevesada. Así pues, ¿para qué podríamos querer esto? La cuestión es la siguiente: como ha podido comprobar, como argumentos de eval () incluimos una expresión alfanumérica, entre comillas, que representa una instrucción del intérprete. El asunto entonces pasa a ser otro: ¿de dónde procede esa expresión? Si logramos que se forme una expresión con la instrucción adecuada en base a circunstancias que pueden variar, podremos lograr que nuestro script se adapte a dichas circunstancias funcionando correctamente en cada caso.

Probablemente todo esto le resulte un poco críptico ahora. Sin embargo, tenga en cuenta lo que supone tener determinadas instrucciones almacenadas, como

cadenas de texto en, digamos, una base de datos (tema que se estudiará en los tres capítulos siguientes) y poder evaluarlas según se vayan leyendo.

## 14.3 FTP

El protocolo FTP (*File Transfer Protocol*) ha sido diseñado y creado para la transferencia de archivos entre diferentes equipos. El intérprete de PHP implementa funciones destinadas a gestionar este protocolo de modo que los usuarios de sus scripts puedan transferir ficheros, siempre y cuando, naturalmente, tengamos los datos de acceso al servidor.

Para trabajar vía FTP con un servidor son necesarios tres pasos básicos:

- Establecer la conexión y acceder al servidor.
- Efectuar la transferencia de archivos, ya sea de subida (*upload*) o de bajada (*download*).
- Cerrar la conexión.

En primer lugar nos vamos a centrar en el establecimiento de la conexión. Para empezar, tenemos que crear una conexión con el servidor. Para ello, PHP nos proporciona la función *ftp\_connect()*, que recibe, como argumento obligatorio, el nombre o dirección IP del servidor al que nos vamos a conectar. Además, puede recibir un segundo parámetro opcional, para indicar el número del puerto por el que se va a hacer la conexión. Si no se especifica, PHP asume el puerto por defecto para el protocolo que, como usted sabe, es el 21. También podemos añadir otro parámetro opcional. Una cifra que expresa el tiempo de espera en segundos para cualquier operación de red subsiguiente. Por defecto, este tiempo es de 90 segundos. Si se excede en alguna operación, PHP la interrumpirá y emitirá un mensaje de error. La función crea un manejador para la conexión, a través del cual intentaremos el acceso FTP al servidor. Si se ha producido un error, la función devuelve un valor false. Para conectar con nuestro servidor local de FTP, que fue instalado en el capítulo 2, usaremos la siguiente sintaxis de la función:

```
$conexionFTP=ftp_connect("localhost");
```

Una vez efectuada la conexión, debemos intentar el acceso al servidor FTP, autenticándonos con nuestro nombre de usuario y contraseña, que deberán coincidir con los que establecimos en la configuración del servidor en el capítulo 2. Para ello contamos con la función *ftp\_login()*, que recibe tres argumentos. El primero es el manejador que hemos obtenido al establecer la conexión, el segundo es el nombre de usuario, y el tercero es la contraseña de acceso. Por ejemplo, para acceder con nuestro nombre y contraseña, usaremos la sentencia siguiente:

```
$acceso=ftp_login($conexionFTP,"usuario","clave");
```

Si el acceso se logra correctamente, la función devuelve un valor true. En caso contrario obtendremos un valor false. Este resultado se puede usar luego en una evaluación condicional, por ejemplo.

Sea lo que sea que hagamos con nuestra conexión FTP, al final del script tendremos que cerrarla. Dejar una conexión abierta es un despropósito que puede dar lugar a muchos problemas. Para cerrar una conexión usamos la función *ftp\_close()*, que recibe, como argumento, el manejador que obtuvimos al abrir dicha conexión (cada conexión genera un manejador).

Para familiarizarnos un poco con estos conceptos, vea el script **conexionFTP.php**, listado a continuación:

```
<?php
    define ("salto","<br>\n");

    // Se intenta establecer una conexión FTP.
    $conexionFTP=ftp_connect("localhost");
    /* Se comprueba si se ha podido establecer la conexión.
    Si no se ha logrado, se interrumpe la ejecución.*/
    if ($conexionFTP){
        echo ("CONEXIÓN ESTABLECIDA.".salto);
    } else {
        die ("NO HAY CONEXIÓN.");
    }

    // Se intenta la autenticación del usuario.
    $acceso=ftp_login($conexionFTP,"usuario","clave");

    /* Se comprueba si se ha logrado el acceso. Si no, se
    interrumpe la ejecución.*/
    if ($acceso){
        echo ("ACCESO AUTORIZADO");
    } else {
        echo ("ACCESO DENEGADO");
    }
    echo (salto);
    ftp_close ($conexionFTP);
    echo ("CONEXIÓN CERRADA.")

?>
```

Vea, en las líneas resaltadas, el uso de las funciones que hemos conocido hasta ahora. Si todo va bien, la página que obtenga será como la de la figura 14.2.

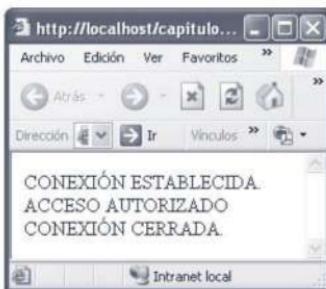


Figura 14.2

Si se le ha producido un error hay dos puntos que debe comprobar:

- En primer lugar, asegúrese de que el servidor FTP se está ejecutando. El icono de CesarFTP debe estar en la bandeja de iconos, junto al reloj de Windows, en la parte inferior derecha de la pantalla.
- Revise el nombre de usuario y contraseña de la función `ftp_login()`. Estos datos deben coincidir con los del usuario que usted creó en el capítulo 2.

Y llega el momento de hacer algo con la conexión que ya sabemos abrir. Hay algunas operaciones básicas: bajar un archivo, subirlo, ver el directorio actual del servidor y cambiar ese directorio por otro.

Para bajar un archivo usamos la función `ftp_get()`, que recibe los siguientes parámetros:

- El manejador de la conexión, obtenido con `ftp_connect()`.
- El nombre (y, en su caso, la ruta) con el que queremos colocar el archivo en la máquina local.
- El nombre (y, en su caso, la ruta) con el que se encuentra el archivo en la máquina remota.
- El modo de transferencia. Éste puede ser una de dos constantes posibles: `FTP_ASCII`, usado para transferir archivos de texto plano, o `FTP_BINARY`, válido para cualquier tipo archivo.

La sintaxis general obedece al siguiente patrón:

```
ftp_get ($manejador, $local, $remoto, $modo);
```

Esta función devuelve un valor lógico true si se ha podido ejecutar correctamente, o un valor false, en caso contrario.

Para subir un archivo al servidor usamos la función *ftp\_put()*. Ésta recibe los mismos argumentos que la anterior, pero en un orden ligeramente distinto. Así:

```
ftp_put ($manejador, $remoto, $local, $modo);
```

Para conocer el directorio del servidor en el que estamos trabajando en un momento dado usamos la función *ftp\_pwd()*. Como argumento recibe, únicamente, el manejador de la conexión, y devuelve la ruta remota.

Si queremos cambiar el directorio remoto de trabajo usaremos la función *ftp\_chdir()*. A ésta habremos de pasarle dos argumentos: en primer lugar, como es lógico, el manejador de la conexión. En segundo lugar, una cadena con la nueva ruta de trabajo. Como es lógico, para acceder a un directorio, usted debe tener los permisos adecuados.

Además de estas cuatro funciones básicas, hay otras que debe conocer. En primer lugar está *ftp\_pasv()*, que permite activar o desactivar el modo de FTP pasivo. Los argumentos que recibe son el manejador de la conexión y un valor de tipo lógico. Si este último es true, se habilita el modo pasivo. En caso contrario, se deshabilita.

La función *ftp\_exec()* es usada para iniciar la ejecución de un archivo de comandos (un programa) en el servidor. Como argumentos recibe el manejador de la conexión y el nombre (y, en su caso, la ruta) del archivo de comandos. Lógicamente usted tiene que tener permiso de ejecución para el archivo de comandos.

Las funciones de FTP que hemos visto sólo operan sin problemas trabajando con un cliente local y un servidor remoto. Usando el servidor localhost puede encontrar fallos de funcionamiento.

## 14.4 PDF

Seguro que usted sabe lo que son documentos en formato PDF (*Portable Document Format*). Se trata de documentos de texto, que pueden incluir imágenes, hipervínculos, gráficos, etc. Los textos pueden estar en diferentes tipografías. Su aspecto es similar al de cualquier documento en formato de texto enriquecido. La diferencia es que están creados de tal modo que mantienen su aspecto bajo cualquier sistema de impresión. Estos documentos se crean mediante aplicaciones

profesionales que suelen tener un coste elevado para el usuario personal. Normalmente, el uso de estos programas se limita a empresas y profesionales. En el mercado existen otras aplicaciones creadas para la visualización de documentos PDF, normalmente de uso gratuito, pero que no permiten generar ni editar dichos documentos. Sin duda el visualizador de PDF más conocido es el Acrobat Reader, de la firma Adobe, que puede ser descargado, gratis, del sitio web del fabricante (<http://www.adobe.com>). Si no tiene usted esta aplicación, descárguesela ahora.

Lo que la mayoría de la gente ignora es que PHP permite generar documentos PDF sin necesidad de comprar costosas aplicaciones comerciales. Usted podrá crear este tipo de documentos mediante los scripts adecuados. Y aquí aparece un problema. Para poder hacerlo, PHP necesita la librería PDFlib. Pero ésta es también de pago. Sin embargo, esta vez tenemos una carta bajo la manga: se trata de **FPDF** (Free PDF). ¿Recuerda cuando, en el capítulo anterior, aprendimos a usar una clase para el envío de e-mails? FPDF es una clase destinada a la creación de documentos en este formato. El archivo de la clase, y todos los archivos adicionales y de documentación, se encuentran en la carpeta fpdf, dentro del directorio correspondiente a este capítulo, en el CD adjunto al libro. Además, puede visitar la web donde encontrará todo lo necesario (<http://www.fpdf.org>).

En este apartado vamos a ver un ejemplo de uso de la clase para comenzar a familiarizarnos con ella. Empezaremos creando un script, al que vamos a llamar **básicoPDF.php**. Por supuesto, lo primero que haremos será requerir el archivo que contiene la clase, así:

```
require ("fpdf/fpdf.php");
```

A continuación, instanciaremos un objeto de la clase, para, mediante sus métodos y propiedades, lograr nuestro objetivo, así:

```
$objetoPDF=new FPDF();
```

El constructor de la clase puede recibir tres argumentosopcionales:

- El primero es el formato del papel, que puede ser alargado (el formato normal), apaisado, Carta, Legal, etc. El formato por defecto es alargado, y se representa como "P" (la letra P, entre comillas). El apaisado se representa como "L". También podría ponerse "Carta", "Legal", etc.
- A continuación se expresa la unidad de medida, también entre comillas. Por defecto se usa el milímetro ("mm"). Otras posibilidades son el punto ("pt"), el centímetro ("cm"), o la pulgada ("in").
- Por último, se puede elegir el tamaño del papel. Por defecto es "A4".

Por lo tanto, el constructor, tal como lo hemos usado aquí, sería equivalente a lo siguiente:

```
$objetoPDF=new FPDF("P", "mm", "A4");
```

A continuación, debemos crear una página. Los documentos se forman a partir de páginas, y cada una debe ser creada específicamente. Si no, no podremos escribir nada en el documento. Para ello, empleamos el método *AddPage()*, así:

```
$objetoPDF->AddPage();
```

Lo siguiente que haremos es definir la tipografía que emplearemos en el documento con el método *SetFont()*. Distintas partes de un documento pueden escribirse, como es lógico, con diferentes tipografías. En ese caso, habrá que usar este método cada vez que necesitemos un cambio. Observe la siguiente línea:

```
$objetoPDF->SetFont("Arial","B",16);
```

El primer argumento que aparece es el nombre de la tipografía. Le sigue el tipo de letra (B, para negrita, I, para cursiva, y/o U, para subrayado). Aquí puede usar combinaciones, si lo desea, como, por ejemplo, "BI", para negrita cursiva. Si no desea ningún atributo para la fuente, ponga una cadena vacía.

El último argumento es el tamaño de la letra, expresado en puntos. Para el tamaño de fuente no se admite otra unidad de medida.

Ahora vamos a crear una celda. Las celdas son contenedores que se alojan en las páginas y donde, a su vez, se aloja el contenido del documento. Es decir, el texto que incluyamos no va colocado, en general, directamente, en las páginas, sino en celdas. Para crear una celda usamos el método *Cell()*, tal como se ve en la siguiente línea:

```
$objetoPDF->Cell(190,10,$cadena);
```

Los dos primeros argumentos son la anchura y la altura de la celda. Como hemos usado los milímetros de unidad de medida (valor por defecto al crear el objeto), en esta unidad se entienden las dimensiones. El tercer argumento es una variable que contiene la cadena de texto que deseamos incluir en el documento. Por defecto la celda tiene un borde invisible, aunque eso podemos cambiarlo. Ya lo veremos. Por último, con todo ya preparado, generamos el documento mediante el uso del método *Output()*, tal como se ve a continuación:

```
$objetoPDF->Output();
```

El resultado de este script es el de la figura 14.3.

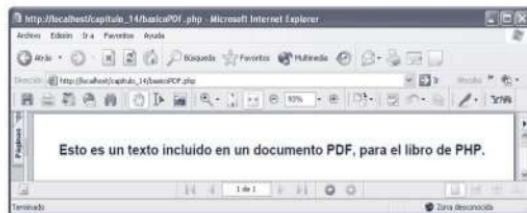


Figura 14.3

El listado completo del script es el siguiente:

```
<?php
    require("fpdf/fpdf.php");
    $objetoPDF=new FPDF();
    $cadena="Esto es un texto incluido en un documento
    PDF, para el libro de PHP.";
    $objetoPDF->AddPage();
    $objetoPDF->SetFont("Arial","B",16);
    $objetoPDF->Cell(190,10,$cadena);
    $objetoPDF->Output();
?>
```



Figura 14.4

A continuación, vamos a ver otro ejemplo, un poco más completo, para aprender algunos conceptos adicionales. El script se llama **completoPDF.php**. Al cargarlo en el navegador, obtendrá un documento PDF como el de la figura 14.4.

La filosofía de trabajo es, prácticamente, la misma que en el ejemplo anterior. Sin embargo, aquí ya podemos apreciar que se puede generar un documento completo, con las características que necesitemos. Vamos a analizar el código y su comportamiento para entender cómo operan algunos de los métodos de esta clase. Por supuesto, el principio del script es similar al anterior:

```
/* Se incorpora la clase FPDF al script. */
require("fpdf/fpdf.php");
/* Se crea un objeto de la clase FPDF. */
$objetoPDF=new FPDF();
/* Se genera una página del documento. */
$objetoPDF->AddPage();
```

Como ve, hasta aquí no hay nada especial. Pero ahora viene el momento de fijar la primera celda de contenidos de la página. Tenemos que establecer la cadena de texto que aparecerá, pero también el borde de color rojo y el fondo de tono turquesa que hemos elegido. Además, estableceremos también el grosor del borde y la tipografía. Por último, con todos estos valores, montaremos la celda en la página, tal como aparece en la figura 14.5.

## Este es el título del documento.

Figura 14.5

El código necesario para este encabezamiento es el siguiente:

```
/* Se establecen los parámetros de la tipografía y se
determina la misma. */
$fuente="Arial";
$estilo="B";
$talla=30;
$objetoPDF->SetFont($fuente, $estilo, $talla);
/* Se establece el color de línea para rodear una celda
de contenidos.*/
$rojo=255;
$verde=0;
$azul=0;
$objetoPDF->SetDrawColor ($rojo, $verde, $azul);
```

```
/* Se establece el ancho de línea para rodear la celda
de contenidos. */
$anchoDeBorde=2;
$objetoPDF->SetLineWidth ($anchoDeBorde);
/* Se establece un color de relleno para una celda de
contenidos. */
$rojo=0;
$verde=200;
$azul=200;
$objetoPDF->SetFillColor ($rojo, $verde, $azul);
/* Se fijan los valores para crear una celda de
contenidos y se genera esta. */
$ancho=190;
$alto=20;
$cadena="Esto es el título del documento.";
$borde=1;
$posicion=0;
$alineacion="C";
$relleno=1;
$objetoPDF->Cell($ancho, $alto, $cadena, $borde,
$posicion, $alineacion, $relleno);
```

Observe el código y los comentarios que se incluyen. Vea como hemos establecido, en primer lugar, la tipografía. Después el color de línea que rodeará la celda de contenidos, y el ancho de dicha línea. Luego hemos fijado el color de relleno que se usará para la celda de contenidos. Por último, se genera la celda para su inclusión en el documento.

Preste especial atención a la forma en que hemos ido estableciendo los valores necesarios en variables, de modo que, al invocar los métodos del objeto de la clase FPDF le pasamos dichas variables como argumento. El hacerlo de ese modo es para facilitar la comprensión de la sintaxis de cada método, y el que usted pueda comparar las líneas de código con las especificaciones que aparecen en los tutoriales oficiales de la clase (están en la web y también en el CD adjunto).

Por lo demás, esta parte es similar a la celda de contenidos del ejemplo anterior, salvo por el hecho del color de la linea y del fondo. Siempre que vayamos a usar el método Cell () y queramos una linea y un fondo, las especificaciones de éstos deben establecerse previamente con los métodos *SetDrawColor ()*, *SetLineWidth ()* y *SetFillColor ()*.

A continuación incluimos un salto de línea, tal como se ve en el siguiente fragmento:

```
/* Se incluye un salto de línea*/
```

```
$objetoPDF->Ln(30);
```

El argumento del método *Ln ()* se refiere a la altura del salto de linea. Si no se incluye, se toma la altura de la última celda de contenidos que se haya incluido en el documento.

La parte que viene a continuación es más interesante. Observe el fragmento de documento que aparece en la figura 14.6.

Esto es un texto para mostrar cómo crear un documento de tipo PDF sin contar con aplicaciones específicas para la creación de este tipo de archivos. Sin embargo si es necesario tener una aplicación que permita visualizarlos. El Adobe Acrobat Reader lo permite y puede descargarse, gratuitamente de su página: <http://www.acrobat.com>.

Figura 14.6

Cuando se emplea el método *Cell ()* se incluye en el documento una celda de contenidos con la posibilidad de incorporar a la misma una línea de texto. La alternativa es *MultiCell ()*, que permite incluir un texto de varias líneas. En el texto puede haber secuencias \n para forzar un salto de linea dentro de la multicelda, si lo deseamos.

El fragmento de documento que ve se ha llevado a cabo en dos pasos principales. Por una parte, se ha creado una multicelda con el texto que parece en negro. Por la otra, se ha creado el enlace que vemos en azul.

Empecemos por la creación de la multicelda y el texto en negro. El código es el siguiente:

```
/* Se establecen los parámetros de la tipografía y se
determina la misma. */
$fuente="Arial";
$estilo="";
$talla=12;
$objetoPDF->SetFont($fuente, $estilo, $talla);
/* Se establecen los parámetros de una celda de
contenidos multilínea. */
$ancho=0; // Todo el ancho útil en la página.
$alto=6; // Altura por línea.
$cadena="Esto es un texto para mostrar como crear un
documento de tipo PDF sin contar con ";
$cadena.="aplicaciones específicas para la creación
de este tipo de archivos. Sin embargo";
$cadena.=" si es necesario tener una aplicación que
permite visualizarlos. El Adobe Acrobat ";
```

```
$cadena.="Reader lo permite y puede descargarse,  
gratuitamente de su página: ";  
$borde=0; //No se verá el borde.  
$alineacion="J"; // Alineación justificada.  
/* Se imprime la celda multilínea con los parámetros  
establecidos. */  
$objetoPDF->MultiCell($ancho, $alto, $cadena, $borde,  
$alineacion);
```

Vea que empezamos por establecer la tipografía, como en el caso de las celdas de una sola línea. A continuación, fijamos los parámetros de la multicelda. Tal como hicimos antes, los establecemos mediante variables, para facilitar la comprensión del código. Preste especial atención a la anchura de la multicelda (variable **\$ancho**). Al darle el valor 0, se ocupará todo el ancho disponible en la página (exceptuando, como es lógico, los márgenes de la misma). Observe la sintaxis del método MultiCell () y compárela con la que tiene en el tutorial.

Ahora viene el momento de crear el enlace azul a la página de Adobe. El código necesario es el siguiente:

```
/* Se establece el color de fuente en azul, para un  
vínculo. */  
$rojo=0;  
$verde=0;  
$azul=255;  
$objetoPDF->SetTextColor ($rojo, $verde, $azul);  
  
/* Se incluye el texto del vínculo con el método Text.  
Al aparecer una dirección web  
este método crea, automáticamente, el enlace a dicha  
dirección. */  
$x=65;  
$y=62;  
$cadena="http://www.acrobat.com";  
$objetoPDF->Text($x, $y, $cadena);  
/* Se reestablece la tipografía en negro. */  
$rojo=0;  
$verde=0;  
$azul=0;  
$objetoPDF->SetTextColor ($rojo, $verde, $azul);  
/* Se escribe un punto, para terminar la frase que  
aparece. */  
$x=110;  
$y=62;  
$cadena=". ";  
$objetoPDF->Text($x, $y, $cadena);
```

El método *Text ()* permite colocar un texto flotante en la página, indicando las coordenadas cartesianas donde se inicia dicho texto. Pero tiene otra particularidad: observe la línea que aparece resaltada. En este caso la cadena es una dirección web (vea el contenido de la variable en la línea inmediatamente anterior). Cuando eso sucede, el texto insertado con este método se convierte, automáticamente, en un vínculo a la página especificada. No obstante, sigue siendo necesario establecer el color de la fuente en azul, como hacemos previamente en esta parte del código. Tras haber colocado el texto del vínculo, volvemos a establecer el color negro para la tipografía y usamos de nuevo el método *Text ()* para escribir un punto que cierre la frase, tal como se aprecia en la figura 14.6.

Si seguimos viendo el documento nos encontramos, tras un salto de linea, con la sección central, reflejada en la figura 14.7.

Como ve, con la clase FPDF pueden generarse documentos que incluyan enlaces. También puede incluir imágenes, como se ve a continuación:

El uso de esta clase es muy potente. Permite generar cualquier documento. Además, los textos a incluir pueden proceder de cualquier fuente. Así, si usted desarrolla un sitio de comercio electrónico, puede usar los datos del cliente o del pedido para generar facturas, albaranes, o resúmenes de compras. Dado que la salida del documento se puede almacenar en un archivo en disco, El cliente puede copiar todas las notas de transacciones en su propio ordenador.



Figura 14.7

Aquí tenemos tres partes. La primera, formada por dos líneas de texto, se ha constituido mediante una multicelda, como ya sabemos hacerlo. Su código, muy simple, es el siguiente:

```
/* Se establecen los parámetros de una celda de
contenidos multilínea. */
$ancho=0; // Todo el ancho útil en la página.
$alto=6; // Altura por línea.
$cadena="Como ve, con la clase FPDF pueden generarse
documentos que incluyan enlaces. ";
$cadena.="También puede incluir imágenes, como se ve
a continuación:";
$borde=0; //No se verá el borde.
$alineacion="J"; // Alineación justificada.
/* Se imprime la celda multilínea con los parámetros
establecidos. */
```

```
$objetoPDF->MultiCell($ancho, $alto, $cadena, $borde,
$alineacion);
```

Como ve, en esta parte no hay nada nuevo. Pero fíjese en el fragmento de código que imprime el texto siguiente:

```
/* Se establecen los parámetros de una celda de
contenidos multilínea. */
$ancho=100; // Solo una parte del ancho útil de la
página.
$alto=6; // Altura por línea.
$cadena="El uso de esta clase es muy potente. Permite
generar cualquier documento. ";
$cadena.="Además, los textos a incluir pueden
proceder de cualquier fuente. Así, si ";
$cadena.="usted desarrolla un sitio de comercio
electrónico, puede usar los datos del ";
$cadena.="cliente o del pedido para generar facturas,
albaranes, o resúmenes de compras. ";
$cadena.="Dado que la salida del documento se puede
almacenar en un archivo en disco, ";
$cadena.="El cliente puede copiar todas las notas de
transacciones en su propio ordenador.";
$borde=0; //No se verá el borde.
$alineacion="J"; // Alineación justificada.
/* Se imprime la celda multilínea con los parámetros
establecidos. */
$objetoPDF->MultiCell($ancho, $alto, $cadena, $borde,
$alineacion);
```

Observe la línea resaltada. Esta vez se especifica un valor diferente de 0 para el ancho de la multicelda. Eso significa que esta vez no se ocupará todo el ancho útil en la página, sino solamente la anchura indicada, en la unidad que se estableció cuando usamos el constructor de la clase. Como, en este script, no hemos especificado dicha unidad de medida, se usan los milímetros, que es la unidad por defecto. Así pues, nuestra multicelda tendrá 100 milímetros de ancho y el texto se distribuirá en ese espacio, en tantas líneas como sean necesarias.

Esto deja un espacio en blanco en la parte derecha de la página. Ahí vamos a colocar una imagen. Mire el siguiente fragmento de código:

```
/* Se incluye una imagen en el documento, con todos los
parámetros necesarios en este caso. */
$archivo="imagenes/aranaroja.jpg";
$x=120;
$y=82;
```

```
$anchura=70;
$altura=55;
$objetoPDF->Image ($archivo, $x, $y, $anchura,
$altura);
```

Como ve, se trata de una operación extremadamente simple. Existe un método llamado **Image ()**. Éste recibe el nombre del archivo que contiene la imagen que deseamos mostrar en el documento. Además, recibe las coordenadas y el tamaño, expresados en la unidad de medida que estemos empleando. Este método es similar a **Text ()** en el sentido de que coloca el objeto de forma flotante, mediante coordenadas cartesianas.

Nuestro documento de ejemplo termina con un vulgar texto, como se ve en la figura 14.8.

Vea los tutoriales incluidos con la clase. Como han sido escritos por los autores de la misma, le proporcionan toda la información necesaria para usarla, además de múltiples ejemplos sumamente interesantes.

Figura 14.8

Esta parte no tiene ya ningún misterio. Su código es el siguiente:

```
/* Se establecen los parámetros de una celda de
contenidos multilínea. */
$ancho=0; // Todo el ancho útil en la página.
$alto=6; // Altura por línea.
$cadena="Vea los tutoriales incluidos con la clase.
Como han sido escritos por los autores de ";
$cadena.= "la misma, le proporcionan toda la
información necesaria para usarla, además de ";
$cadena.= "múltiples ejemplos sumamente
interesantes.";
$borde=0; //No se verá el borde.
$alineacion="J"; // Alineación justificada.
/* Se imprime la celda multilínea con los parámetros
establecidos. */
$objetoPDF->MultiCell($ancho, $alto, $cadena, $borde,
$alineacion);
```

Por último, damos la salida al navegador con el método **Output ()**.

Como puede ver, la clase FPDF supone una enorme ayuda si sus scripts tienen que generar informes, facturas u otros documentos para el usuario. Esta clase se puede descargar de la página oficial, que como he mencionado antes es

[www.fpdf.org](http://www.fpdf.org). Aunque yo se la he incluido en el CD, no deje de visitar la página, ya que, en cualquier momento, pueden aparecer actualizaciones o mejoras. Es una de las grandes ventajas del Open Source. Existe una gran comunidad de programadores que hacen sus aportaciones de forma desinteresada.

La clase proporciona una gran cantidad de métodos, además de los aquí expuestos, para establecer todas las posibles características de un documento PDF. Nosotros aquí no vamos a detallarlos todos. No es necesario ya que, con el paquete de la clase, están incluidos los tutoriales del autor, y sería repetir innecesariamente la documentación que ya tiene en el CD. El objetivo de este apartado es darle a conocer esta herramienta, y creo que se ha logrado.

## 14.5 LA DIRECCIÓN IP

Como usted sabe, la dirección IP identifica una conexión en una red. Si se trata de una red local, se tratará de una IP interna. Si se trata de una conexión a Internet, tendremos una IP externa. Pero, en cualquier caso, esa dirección es única para nuestra conexión en un momento dado. El intérprete de PHP permite identificar la dirección IP con la que un cliente se conecta al servidor. Para ello, contamos con una variable de servidor: `$_SERVER["REMOTE_ADDR"]`. Sin embargo, en ocasiones, sobre todo si el cliente se conecta mediante un proxy, debemos contar con otras dos variables alternativas muy útiles en muchos casos: `$_SERVER["CLIENT_IP"]` y `$_SERVER["HTTP_X_FORWARDED_FOR"]`. En general, la manera de determinar lo mejor posible la IP de una conexión a nuestro servidor es la que aparece a continuación:

```
$ip = 0;
/* Si la variable $_SERVER['HTTP_CLIENT_IP'] tiene
algún contenido, se asigna a la variable $ip. */
if (!empty($_SERVER['HTTP_CLIENT_IP'])) $ip =
$_SERVER['HTTP_CLIENT_IP'];
/* Si la variable $_SERVER['HTTP_X_FORWARDED_FOR']
tiene algún contenido, se asigna a la variable $ip. En ese
caso debe separarse la IP pública del cliente de las
obtenidas a través de posibles redes locales. */
if (!empty($_SERVER['HTTP_X_FORWARDED_FOR'])) {
/* Se abre la matriz de IP's obtenidas de
$_SERVER['HTTP_X_FORWARDED_FOR']. */
$ListaDeip = explode (", ",
$_SERVER['HTTP_X_FORWARDED_FOR']);
/* Si ya hay contenido en $ip, se añade a la matriz. */
if ($ip) {
array_unshift($ListaDeip, $ip);
$ip = 0;
```

[www.fpdf.org](http://www.fpdf.org). Aunque yo se la he incluido en el CD, no deje de visitar la página, ya que, en cualquier momento, pueden aparecer actualizaciones o mejoras. Es una de las grandes ventajas del Open Source. Existe una gran comunidad de programadores que hacen sus aportaciones de forma desinteresada.

La clase proporciona una gran cantidad de métodos, además de los aquí expuestos, para establecer todas las posibles características de un documento PDF. Nosotros aquí no vamos a detallarlos todos. No es necesario ya que, con el paquete de la clase, están incluidos los tutoriales del autor, y sería repetir innecesariamente la documentación que ya tiene en el CD. El objetivo de este apartado es darle a conocer esta herramienta, y creo que se ha logrado.

## 14.5 LA DIRECCIÓN IP

Como usted sabe, la dirección IP identifica una conexión en una red. Si se trata de una red local, se tratará de una IP interna. Si se trata de una conexión a Internet, tendremos una IP externa. Pero, en cualquier caso, esa dirección es única para nuestra conexión en un momento dado. El intérprete de PHP permite identificar la dirección IP con la que un cliente se conecta al servidor. Para ello, contamos con una variable de servidor: `$_SERVER["REMOTE_ADDR"]`. Sin embargo, en ocasiones, sobre todo si el cliente se conecta mediante un proxy, debemos contar con otras dos variables alternativas muy útiles en muchos casos: `$_SERVER["CLIENT_IP"]` y `$_SERVER["HTTP_X_FORWARDED_FOR"]`. En general, la manera de determinar lo mejor posible la IP de una conexión a nuestro servidor es la que aparece a continuación:

```
$ip = 0;
/* Si la variable $_SERVER['HTTP_CLIENT_IP'] tiene
algún contenido, se asigna a la variable $ip. */
if (!empty($_SERVER['HTTP_CLIENT_IP'])) $ip =
$_SERVER['HTTP_CLIENT_IP'];
/* Si la variable $_SERVER['HTTP_X_FORWARDED_FOR']
tiene algún contenido, se asigna a la variable $ip. En ese
caso debe separarse la IP pública del cliente de las
obtenidas a través de posibles redes locales. */
if (!empty($_SERVER['HTTP_X_FORWARDED_FOR'])) {
/* Se abre la matriz de IP's obtenidas de
$_SERVER['HTTP_X_FORWARDED_FOR']. */
$ListaDeip = explode (", ",
$_SERVER['HTTP_X_FORWARDED_FOR']);
/* Si ya hay contenido en $ip, se añade a la matriz. */
if ($ip) {
array_unshift($ListaDeip, $ip);
$ip = 0;
```

```
        }
        /* Se eliminan IP's privadas, procedentes de posibles
redes locales, así como la dirección de bucle local.
Cuando se encuentra una IP pública (externa) se
devuelve como resultado de la función. */
foreach ($ListaDeip as $direccion) if
(!eregi("^(192\.168|172\.16|10|224|240|127|0)\.", 
$direccion)) $ip=$direccion;
}
/* Si no había contenido en
$_SERVER['HTTP_X_FORWARDED_FOR'] se devuelve la IP obtenida
mediante $_SERVER['REMOTE_ADDR']. */
$ip ? $ip : $_SERVER['REMOTE_ADDR'];
```

En el fragmento anterior, al terminar la ejecución del mismo, la variable de cadena **\$ip** contiene la IP de la conexión del cliente. Esta cadena se puede almacenar en un fichero de texto o en una base de datos, junto con la fecha y hora de la conexión, por si fuera necesario identificar, posteriormente, al usuario que se ha conectado. También se puede usar para otros fines que veremos más adelante: tenemos un ejemplo de uso en uno de los scripts del capítulo 18.

## CAPÍTULO 15

# BASES DE DATOS Y SQL

---

---

Empezamos en este capítulo el estudio de la creación y gestión de bases de datos. Las bases de datos, (**BBDD**) son estructuras donde se almacena información siguiendo unas pautas de disposición y ordenación para el posterior procesado de los datos. Es una definición tan buena como cualquier otra. Seguramente, si usted lee cincuenta libros al respecto, encuentre otras tantas definiciones distintas, pero todas coincidirán en lo esencial. Como sistema de almacenamiento de datos, las BBDD son mucho más eficientes que los archivos de texto que conocemos en el capítulo 8. Y esto por varias razones pero, principalmente, porque nos permiten un acceso directo al dato que necesitamos sin que sea preciso recorrer todo un fichero para encontrarlo. Además, los modernos sistemas de gestión de BBDD relacionales admiten el almacenamiento de muchos tipos de datos, no sólo texto plano. Hablamos de BBDD relacionales cuando podemos establecer relaciones entre las distintas informaciones que componen una base de datos. Por ejemplo, suponga que usted quiere guardar la lista de clientes de una empresa de servicios. Por otro lado, tiene una lista de los servicios que ofrece dicha entidad. Además conserva una relación histórica de los servicios empleados por cada cliente. A partir de ahí, mediante el uso de un *Sistema de Gestión de Bases de Datos Relacionales (RDBMS, Relation Data Base Management System)* puede sacar estadísticas u otros informes que le ayuden a la planificación u organización de su trabajo diario.

Los motores de BBDD actuales (la parte que se encarga de gestionar la BBDD fuera de la vista del usuario) están basados en el lenguaje **SQL (Structured Query Language, Lenguaje Estructurado de Consultas)**. Éste va a ser el objeto de estudio en este capítulo, de una forma genérica, sin entrar a usar ninguna base de datos real. Esto, que puede parecer, en principio, un poco demasiado teórico, nos va a servir como introducción para el uso de RDBMS en los dos capítulos

siguientes. Además, constituirá una interesante guía de referencia para su uso en el futuro. En los dos capítulos siguientes veremos cómo gestionar BBDD desde PHP, lo que representará la verdadera potencia de esta tecnología.

## 15.1 CÓMO ES UNA BASE DE DATOS

Es imposible hablar del lenguaje SQL sin saber, previamente, cómo son las bases de datos que gestiona. En una BBDD la información se almacena en tablas. Cada tabla está formada por *filas*, llamadas también *registros* o *tuplas*. Cada tupla está dividida en *campos*, que forman *columnas*. Cada campo contiene un dato y todos los campos de una columna tienen la misma estructura, es decir, que en todos ellos se almacena un dato del mismo tipo, tamaño máximo, etc. Por ejemplo, si uno de los datos de la tabla donde se almacenan los servicios es el nombre de cada servicio, ese dato será de tipo cadena alfanumérica y tendrá un máximo de caracteres. Para que nos hagamos una idea, vea la tabla de la figura 15.1.

N. <sup>º</sup>	NOMBRE	CIUDAD	PAÍS
1	Alfonso Gómez	Madrid	España
2	Giancarlo Bennedetti	Roma	Italia
3	Hans Krüger	Berlín	Alemania
4	Mauricio Rodríguez	Lima	Perú
5	John Foster	Londres	Reino Unido
6	Alice Coppertone	Las Vegas, Nevada	Estados Unidos

Figura 15.1

Esta tabla podría ser la lista de gerentes de una multinacional. Vea que cada fila (registro) almacena los datos de una persona, y que cada columna (campo) almacena un dato determinado de todos los registros. Esta es una de las características de una tabla en una BBDD. Se conoce con el nombre de *homogeneidad estructural*. En una BBDD pueden coexistir tantas tablas como sean necesarias. Cada una tendrá su propia estructura (disposición de los datos).

Dentro de una tabla es conveniente (aunque no obligatorio) crear un *campo clave*. Éste es uno de los campos de la tabla que sirve como referencia para la localización de registros. Aunque con los motores de BBDD actuales es posible localizar un registro por cualquiera de sus campos, el hacerlo mediante el campo clave acelera el proceso, sobre todo cuando se gestionan tablas con millones de registros. No obstante, con las tecnologías actuales, localizar un registro, o más de uno, por un campo que no sea clave es perfectamente posible y viable. En el ejemplo de la tabla de la figura 15.1, el campo clave podría ser el número que

identifica a cada registro. Otra razón para el uso de un campo clave es que los motores de BBDD no permiten que éste tenga el mismo contenido en dos registros diferentes. Así pues, volviendo al ejemplo de la tabla de la figura 15.1, podría ser que nuestra multinacional incluyera en su plantilla a una segunda persona llamada John Foster (creo que, en los países anglosajones, es un nombre bastante corriente). Pero sólo uno de ellos podría tener el n.<sup>º</sup> 5, si es que este dato se ha definido como campo clave en la estructura de la tabla.

El campo clave de una tabla se conoce también con el nombre de *clave primaria*. Además, podemos definir otros campos como *claves secundarias*, llamadas también, *foráneas* o *extranjeras*. Una clave secundaria es un campo que, normalmente, coincide con la clave primaria de otra tabla, y se emplea para construir relaciones entre ambas. Siguiendo con el ejemplo de la tabla de la figura 15.1, podríamos tener otra tabla en la que almacenáramos los nombres de diferentes países, y la lista de ciudades, o provincias, de dichos países en los que la empresa tiene sucursales. El nombre del país sería la clave principal en esta tabla, y clave foránea en la de personal.

Para facilitarle la comprensión de los elementos con los que va a trabajar considere el almacenamiento de información en una base de datos según el esquema de la figura 15.2.

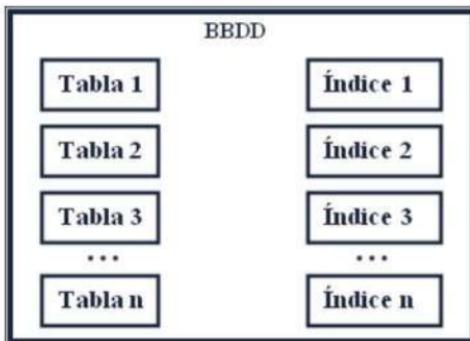


Figura 15.2

Como ve, dentro del contenedor de la BBDD se alojan las tablas, que almacenarán la información necesaria. Puede haber tantas como necesitemos. La información que necesitemos manejar para nuestra aplicación la distribuiremos en estas tablas siguiendo criterios de lógica. Por ejemplo, si usted va a usar una base

de datos para la gestión de una empresa, creará una tabla para los clientes, otra para los empleados, otra para los productos o servicios que ofrece la empresa, otra para las ventas, otra para las compras, otra para el almacén, etc.

De los índices hablaremos en este mismo capítulo. De momento, considérelos como elementos auxiliares para ayudarnos en la gestión de las tablas.

## 15.2 EL LENGUAJE SQL

En los albores de la informática de gestión, allá por los tardíos 70 o primeros 80 del pasado siglo, las aplicaciones comerciales que manejaban ficheros de datos tenían sus propios formatos para almacenar y recuperar la información. Cada nuevo programa que se desarrollaba, normalmente bajo demanda y a medida en aquellos tiempos, grababa y recuperaba los datos en un formato específico, nativo de esa aplicación o del lenguaje en el que había sido escrita. Esto presentaba varios problemas. El usuario de un programa de, digamos, contabilidad que quisiera migrar a otro necesitaba reconvertir todos los datos, en ocasiones, volviendo a grabarlos. Además, en una empresa mediana o grande (las únicas que, en aquellos días, podían permitirse tener sistemas informáticos), los datos de almacén podían no estar grabados en un formato compatible con los de facturación. Esto obligaba a grabar los pedidos en los dos departamentos. Cuando la información se graba por duplicado o triplicado en diferentes sistemas siempre pueden aparecer diferencias. Además, es posible que unos datos se actualicen en un sistema pero no en otro. Y, en muchas ocasiones, cuando había varias versiones de la misma información circulando por distintos departamentos de una misma empresa, no se sabía cuáles eran los datos más actualizados. Esto es lo que se llama ***inconsistencia de la información***. Yo, personalmente, lo llamo un verdadero desastre.

Surgía la necesidad de tratar los archivos de datos de forma que todos presentaran un formato coherente, y que cualquier aplicación pudiera acceder a las mismas fuentes de información. El lenguaje SQL cubría estas necesidades. Por una parte encapsulaba los datos, interponiéndose entre estos y la aplicación, de modo que ésta usaba instrucciones SQL y era él quien manejaba los datos. Así pues, bastaba con que una aplicación implementase SQL para que pudiera acceder a los datos de cualquier otra aplicación. Por otra parte, dado que está orientado a la gestión básica de los datos, SQL tiene, relativamente, pocas instrucciones, y es muy fácil de aprender y manejar con soltura. SQL se integra así con los lenguajes de programación modernos, supeditado a ellos. Por ejemplo, SQL no puede mostrar un registro de una BBDD en una página web. Pero si puede recuperar el contenido del registro y cedérselo al script PHP, que es el que se encargará de darle el formato adecuado y mostrarlo en un documento HTML. Y PHP sí puede

gestionar SQL. En los dos capítulos siguientes verá ejemplos de funcionamiento que reforzarán este concepto.

Las instrucciones de SQL (llamadas, genéricamente, *consultas*) se pueden considerar divididas en dos grupos principales. *Las estructurales*, también llamadas *de definición de datos*, o *DLL* y *las de datos*, también llamadas *de manipulación de datos* o *DML*. Las primeras están destinadas a crear, modificar y eliminar las BBDD y las estructuras de las tablas que las conforman así como los índices. Las segundas se ocupan de incorporar nuevos registros a las tablas, buscar determinados registros según los criterios necesarios, modificar los datos grabados o eliminarlos, si procede. Como es lógico, iniciaremos el estudio de SQL con el primer grupo, ya que antes de trabajar con datos tenemos que tener un sitio donde poder almacenarlos y clasificarlos.

### 15.2.1 Consultas estructurales

Lo primero que tenemos que hacer para gestionar una base de datos es crearla. Esto lo hacemos con la consulta ***CREATE DATABASE***, cuya sintaxis genérica es la siguiente:

```
CREATE DATABASE IF NOT EXISTS baseDeDatos;
```

Como ve, ponemos la consulta (***CREATE DATABASE IF NOT EXISTS***), seguida del nombre con el que queremos almacenar la base de datos (en nuestro ejemplo, *baseDeDatos*) y terminamos con punto y coma. Todas las consultas SQL, al igual que las de PHP, deben terminar con punto y coma. La cláusula ***IF NOT EXISTS*** es opcional, pero si no la incluye y la base de datos ya existe se produce un error. Como norma general, inclúyala siempre, al menos hasta que adquiera cierta soltura.

Quiero llamar su atención sobre una cosa. La base de datos recién creada es sólo un contenedor que almacenará las tablas donde se guardarán los datos. Pero sin las tablas, la base de datos, por sí misma, no permite guardar ninguna información. Antes de pensar en crear tablas (o en realizar cualquier operación, cuando ya estén creadas), es necesario especificar la base de datos con la que vamos a trabajar, ya que en el dispositivo de almacenamiento, sea éste local o remoto, podemos tener más de una BBDD.

Para seleccionar una usaremos la consulta ***USE***, así:

```
USE baseDeDatos;
```

Puede que usted desee eliminar una base de datos completamente. En ese caso utilice La consulta **DROP DATABASE**, tal como ve a continuación:

```
DROP DATABASE IF EXISTS baseDeDatos;
```

Especifique, tal como aparece, el nombre de la BBDD que desea eliminar. Pero recuerde. Cuando elimina una base de datos se borra absolutamente todo lo que contiene. Tablas, índices, datos... No queda nada. Y esta operación es irreversible. Así que, a menos que tenga usted copia de seguridad, más vale que esté completamente seguro de lo que hace. La cláusula IF EXISTS es opcional, pero si no la incluye y la BBDD no existe se producirá un error.

Una vez que se ha creado una BBDD con CREATE DATABASE, y seleccionado con USE, llega el momento de crear las tablas. Cuando se crea una tabla se especifican los campos que formarán cada registro de la misma. A modo de ejemplo, vamos a crear una tabla de empleados de una empresa, donde se podrá almacenar el n.<sup>º</sup> de empleado, el nombre, el salario bruto mensual y la categoría profesional, así como un indicador sobre si el empleado es de sexo masculino ("M") o femenino ("F") y el departamento en que trabaja. Para la creación de tablas contamos con la consulta **CREATE TABLE**, que es muy versátil, según veremos enseguida. A lo largo de los dos capítulos siguientes veremos varios ejemplos de su uso.

```
CREATE TABLE IF NOT EXISTS empleados (
    numero INT NOT NULL,
    nombre VARCHAR (50),
    salario FLOAT (6,2) ZEROFILL,
    categoria CHAR (30),
    sexo ENUM ('M','F'),
    departamento CHAR (2),
    PRIMARY KEY (numero)) TYPE=MyISAM;
```

Con esto se habrá creado la estructura de la tabla en la que, posteriormente, se introducirán los datos. Vamos a analizar un poco lo que acabamos de hacer. Fíjese en que hemos usado la consulta CREATE TABLE, seguida del nombre de la tabla que queremos crear. Entre paréntesis aparece la definición de los campos que tendrá la tabla, separando cada uno del siguiente con una coma y un salto de línea. Lo del salto de línea es opcional (se ve más claro).

Las comas, en cambio, son obligatorias. Cada campo se crea anotando su nombre, seguido del tipo de dato que se almacenará y, opcionalmente, algunos atributos para definir su comportamiento. Los tipos de datos que se pueden manejar desde SQL son los que ve en la tabla de la figura 15.3.

TIPOS DE DATOS SQL	
TIPO	DESCRIPCIÓN
<b>TINYINT</b>	Entero de 0 a 255 sin signo o de -128 a 127 con signo.
<b>SMALLINT</b>	Entero de 0 a 65.535 sin signo, o de -32.768 a 32.767 con signo.
<b>MEDIUMINT</b>	Entero de 0 a 16.777.215 sin signo o de -8.388.608 a 8.388.607 con signo.
<b>INT</b>	Entero de 0 a 4.294.967.295 sin signo o de -2.147.483.648 a 2.147.483.647 con signo.
<b>BIGINT</b>	Entero de 0 a 18.446.744.073.709.551.616 sin signo o desde el negativo -9.223.372.036.854.775.808 a 9.223.372.036.854.775.807 con signo.
<b>FLOAT (M,D)</b>	Número en coma flotante de simple precisión. M es la cantidad máxima de dígitos, sin contar el signo ni el punto decimal. D es el número de dígitos decimales.
<b>DOUBLE (M,D)</b>	Número en coma flotante de doble precisión. M y D como en caso anterior.
<b>DATE</b>	Fecha en formato AAAA-MM-DD o AA-MM-DD.
<b>DATETIME</b>	Fecha y hora en formato AAAA-MM-DD HH:MM:SS.
<b>TIME</b>	Hora en formato HH:MM:SS, o bien HHMMSS o HHMM.
<b>CHAR (N)</b>	Cadena de n caracteres. Se reservan n caracteres, aunque el dato que luego se grabe ocupe menos.
<b>VARCHAR (N)</b>	Cadena de longitud variable. Se reservan n caracteres, pero si el dato ocupa menos, se reduce la longitud.
<b>TINYTEXT</b>	Texto plano con un máximo de 255 caracteres.
<b>TEXT</b>	Texto plano con un máximo de 65.535 caracteres.
<b>MEDIUMTEXT</b>	Texto plano con un máximo de 16.777.215 caracteres.
<b>LONGTEXT</b>	Texto plano con un máximo de 4.294.967.295 caracteres.
<b>TINYBLOB</b>	Archivo binario (puede ser texto RTF, imágenes, etc. con un máximo de 255 bytes).
<b>BLOB</b>	Archivo binario con un máximo de 65.535 bytes.
<b>MEDIUMBLOB</b>	Archivo binario con un máximo de 16.777.215 bytes.
<b>LONGBLOB</b>	Archivo binario con un máximo de 4.294.967.295 bytes.
<b>ENUM ('V1','V2', ETC.)</b>	Campo que acepta cualquiera de los posibles valores enumerados entre paréntesis.
<b>DECIMAL (M,D)</b>	Número guardado como cadena alfanumérica. M y D como en caso anterior.

Figura 15.3

No se deje abrumar por la gran cantidad de tipos de datos que existen. En la práctica sólo usará unos cuantos. Además, enseguida se familiarizará con ellos. En el ejemplo que aparece antes de la tabla vemos algunos de los tipos más habituales. El tipo de dato siempre se escribe con mayúsculas en la declaración de una tabla, por convencionalismo. Algunos motores de BBDD soportan las mayúsculas y las minúsculas, pero acostumbre a escribirlos en mayúsculas por claridad y por convencionalismo (casi todos los programadores seguimos este formato, a fin de compartir información).

Los tipos **CHAR** y **VARCHAR** se usan para almacenar cadenas alfanuméricas y van seguidos de un número entre paréntesis. En el caso de un dato CHAR, ese número indica la cantidad de caracteres que se reservan para la cadena. Si el dato introducido tiene menos caracteres, no importa. El campo seguirá ocupando lo mismo en el disco. Cuando se emplea el tipo VARCHAR, y el dato introducido tiene menos caracteres de los que se reservaron, ese campo, en ese registro concreto, ocupa menos: tantos bytes como caracteres tenga la cadena grabada. Por lo tanto, en muchos casos, es más interesante usar VARCHAR, que CHAR, sobre todo en campos cuyos datos puedan variar mucho.

Los datos de tipo **FLOAT**, **DOUBLE** y **DECIMAL** reciben dos parámetros entre paréntesis, separados por una coma. El primero es el número máximo total de dígitos que podrá tener el valor almacenado. Esto incluye los dígitos de la parte entera y fraccionaria, pero no el punto decimal ni el signo. Por ejemplo si define un dato como FLOAT (6,2) le cabrá el valor 9.384,23 y también -9.384,23.

Los campos de tipo **ENUM** se usan para almacenar un valor de una serie de posibles valores discretos. Por ejemplo, se emplean mucho para almacenar datos de tipo verdadero, falso o similar. Lo que hacemos es incluir entre paréntesis, una lista con cada uno de los posibles valores. El motor de la BBDD se encargará de generar un error si se intenta introducir un valor que no esté en la lista. Los valores se separan con comas y se acotan con comillas simples. Ni éstas ni aquéllas formarán parte del valor. Por ejemplo, si definimos un campo como ENUM ('S', 'N') quiere decir que puede recibir una S o una N, pero ningún otro valor. Estos campos se emplean mucho en aplicaciones que den al usuario opción a elegir uno de varios valores posibles.

Cuando se crea una tabla, al declarar los campos que la constituirán, además del tipo de cada dato, se le pueden asignar algunos atributos, destinados a modificar su comportamiento, o acotarlo.

La lista de posibles atributos para los campos es la que aparece recogida en la tabla de la figura 15.4.

ATRIBUTOS DE DATOS SQL	
ATRIBUTO	DESCRIPCIÓN
<b>NULL</b>	Indica que el campo podrá quedar sin ningún contenido cuando se cree un registro, o cuando se modifique uno ya existente.
<b>NOT NULL</b>	Indica que este campo debe tener asignado un contenido obligatoriamente en cada registro. Si se intenta dejar sin contenido se producirá un error.
<b>DEFAULT valor</b>	Indica el valor que el campo asumirá por defecto. Cuando se cree un nuevo registro de la tabla en el campo se almacenará, de forma automática, el valor establecido por defecto, aunque, por supuesto, si se especifica otro valor, se grabará el especificado. Si el campo es numérico, se escribirá el valor, sin más. Si el campo es alfanumérico se acotará con comillas simples.
<b>ZEROFILL</b>	Este atributo se aplica a campos numéricos cuando queremos que se rellenen con ceros los dígitos de más. Por ejemplo, si se reserva en un campo FLOAT sitio para cuatro dígitos y se almacena el valor 867, quedará como 0867.
<b>UNIQUE</b>	Se utiliza cuando queremos que no se pueda repetir el valor de un campo en otro registro. Por ejemplo, si creamos una tabla de personal y uno de los campos es el DNI de cada persona, es lógico que no pueda haber dos personas con el mismo número de documento. Si se intenta asignar un valor a ese campo, que ya exista en otro registro, se producirá un error. Hay que indicar que MySQL no reconoce este atributo. Lo ignora, sin dar ningún error, por razones de compatibilidad, pero no lo tiene en cuenta.
<b>UNSIGNED</b>	Este atributo, aplicado a campos numéricos, hace que el motor de la base de datos genere un error si se intenta introducir un valor con signo, lo que, en la práctica, impide el uso de valores negativos.
<b>AUTO_INCREMENT</b>	Este atributo se asigna a un campo numérico cuando deseamos que se incremente de forma automática. Es decir, cada nuevo registro que se crea almacenará, en dicho campo, un valor que sea una unidad más que el anterior. Si elimina un registro, el valor que tenía en ese campo no se le volverá a asignar a ningún otro.

ATRIBUTOS DE DATOS SQL (Cont.)	
ATRIBUTO	DESCRIPCIÓN
<b>PRIMARY KEY</b>	Este atributo define un campo como clave primaria. Eso hace que en dicho campo no pueda haber valores repetidos, lo que, en la práctica, sustituye al atributo <b>UNIQUE</b> en aquellos motores de BBDD que no lo utilizan, como es el caso de MySQL.

*Figura 15.4*

Fíjese que, al final de la definición de la estructura de la tabla empleados que hemos visto aparecen dos cláusulas. En primer lugar tenemos:

```
PRIMARY KEY (numero)
```

Para establecer un campo como clave primaria podemos poner el atributo **PRIMARY KEY** en la misma línea que definimos el campo o, como hemos hecho aquí, al final de la lista de campos, como una cláusula. ¿Cuál es la diferencia? En la práctica, ninguna. Es una cuestión de criterio. Yo, personalmente, encuentro más legible la definición de la tabla poniendo esta cláusula al final, pero si usted lo prefiere de otro modo, podría haber escrito, al definir el campo, lo siguiente (el resultado es el mismo):

```
numero INT NOT NULL PRIMARY KEY,
```

El lenguaje SQL le permite el uso de claves foráneas, tal como le mencioné en el apartado anterior. Estas claves, normalmente, se asignan a campos que coinciden con otros que, a su vez, son clave primaria en otra tabla, y se usan para relacionar tablas entre sí. Suponga el ejemplo de la tabla de empleados que hemos creado, con una ligera modificación:

```
CREATE TABLE IF NOT EXISTS empleados (
    numero INT NOT NULL,
    nombre VARCHAR (50),
    salario FLOAT (6,2) ZEROFILL,
    categoria CHAR (30),
    sexo ENUM ('M','F'),
    departamento CHAR (2),
    PRIMARY KEY (numero),
    FOREIGN KEY (departamento) REFERENCES departamentos
    ON UPDATE CASCADE
    ON DELETE SET NULL
) TYPE=MyISAM;
```

La cláusula **REFERENCES** indica en qué tabla es clave principal la que aquí es foránea, siempre trabajando en la misma BBDD.

La cláusula **ON UPDATE** indica lo que debe hacer el motor de BBDD si se modifica el valor del campo en la tabla departamentos.

La cláusula **ON DELETE** indica lo que debe hacer el motor de BBDD si se elimina el valor del campo en la tabla departamentos. Las posibilidades para estas dos últimas cláusulas son:

- **CASCADE.** Actualiza (ON UPDATE) o elimina (ON DELETE) los registros de la tabla de empleados según los de la tabla de departamentos.
- **SET NULL.** Establece el valor NULL en el campo departamento de la tabla de empleados, si el registro asociado es eliminado.
- **RESTRICT.** Impide la modificación o eliminación del campo departamento en la tabla de departamentos.

Y, ahora, veamos la creación de la tabla de departamentos de la empresa:

```
CREATE TABLE IF NOT EXISTS departamentos (
    departamento CHAR (2) NOT NULL,
    descripcion VARCHAR (50),
    PRIMARY KEY (departamento)
) TYPE=MyISAM;
```

No obstante, considere esto una mera curiosidad, ya que MySQL, que es el motor de BBDD que usaremos en este libro, no emplea claves foráneas. Si, en algún momento, usted trabaja con otro motor de BBDD que le admite el uso de claves foráneas, aquí tiene una referencia.

Otra cláusula a la que le tenemos que prestar atención es la siguiente:

**TYPE=MyISAM**

El tipo de tabla se refiere a la forma en que el motor de BBDD va a gestionarla. Esto es transparente al usuario. Las alternativas a **MyISAM** son **ISAM** y **HEAP**. Cuando emplee MySQL use siempre el tipo MyISAM, que es el menos restrictivo y más eficiente.

Como es lógico, también puede, si lo desea, eliminar una tabla de la BBDD con la que está trabajando. Para ello emplearemos la consulta **DROP TABLE**, según la siguiente sintaxis genérica:

```
DROP TABLE IF EXISTS nombreDeLaTabla;
```

Pero tenga cuidado. Al igual que ocurre con las BBDD, la eliminación de una tabla es irreversible. Se pierde todo: estructura y datos.

El lenguaje SQL permite la creación de índices que el motor de BBDD usará para agilizar la búsqueda de registros, mediante la consulta ***CREATE INDEX***. Por ejemplo, suponga que, en la tabla de empleados que hemos creado, prevemos que vamos a necesitar hacer búsquedas por el nombre del empleado. El campo se llama nombre. Podemos crear un índice adecuado con la consulta

```
CREATE INDEX indicePorNombre ON empleados (nombre);
```

Como norma general, la sintaxis para la creación de un índice es la que aparece a continuación:

```
CREATE INDEX índice ON tabla (campo);
```

El índice no es gestionado por ninguna consulta SQL específica, sino que es el propio motor de BBDD el que, de forma transparente, lo mantiene actualizado cuando se crean, modifican o eliminan registros y el que lo usa para la localización de los mismos. La adición, edición y eliminación de datos se tratará en la siguiente sección.

Para eliminar un índice usaremos la consulta ***DROP INDEX***, tal como aparece a continuación:

```
DROP INDEX índice;
```

Esto no presenta mayor problema, ya que, mientras conservemos las tablas, el índice puede volver a crearse cuando sea necesario.

Y una cuestión importante. En ocasiones es necesario modificar la estructura de una tabla, conservando la información que ya contiene. Esto ocurre más a menudo de lo que parece, ya que las necesidades de los usuarios de la aplicación evolucionan y hay que incorporar nuevos datos o eliminar o modificar algunos de los que ya existen. Antes de la aparición de SQL, esta tarea podía llegar a ser realmente engorrosa. La estructura de una tabla puede ser modificada si es necesario mediante la consulta ***ALTER TABLE***, cuya sintaxis genérica es la siguiente:

```
ALTER TABLE tabla  
ADD campo definicionDelCampo,
```

```
CHANGE campoAntiguo campoNuevo definicionDelCampo,  
DROP campo;
```

Lo primero que tenemos que especificar es, por supuesto, el nombre de la tabla sobre la que vamos a efectuar las modificaciones que necesitemos. Por supuesto, dicha tabla deberá existir en la BBDD en uso.

A continuación, podemos añadirle campos nuevos a la tabla, mediante la cláusula **ADD**, cambiar el nombre y la definición de campos ya existentes mediante la cláusula **CHANGE**, o eliminar campos mediante la cláusula **DROP**.

Por supuesto, no es necesario que en una consulta de este tipo aparezcan las tres cláusulas, sino sólo aquellas que necesitemos usar. Además, cada cláusula que incluyamos puede aparecer varias veces en la consulta. Por ejemplo, usted puede tener que añadir dos, tres o más campos a su tabla. Para cada uno de los campos nuevos usará una línea con la cláusula ADD. Si necesita llevar a cabo varios cambios o eliminaciones, podrá usar todas las cláusulas CHANGE o DROP que requiera.

### 15.2.2 Consultas de datos

En la sección anterior hemos aprendido a crear una BBDD y las tablas que la forman. Ahora llega el momento de introducir datos. Vamos a trabajar con la tabla de empleados que definimos anteriormente. Repito aquí la creación de la misma, para que tenga los datos a mano. Copio la última definición de la estructura de la tabla para que podamos verla con todos los datos, con un ligero cambio para mejorar las capacidades didácticas del ejemplo:

```
CREATE TABLE IF NOT EXISTS empleados (  
    numero INT NOT NULL,  
    nombre VARCHAR (50),  
    salario FLOAT (6,2) ZEROFILL,  
    categoria CHAR (30) DEFAULT NULL,  
    sexo ENUM ('M', 'F'),  
    departamento CHAR (2),  
  
    PRIMARY KEY (numero),  
    FOREIGN KEY (departamento) REFERENCES departamentos  
    ON UPDATE CASCADE  
    ON DELETE SET NULL  
) TYPE=MyISAM;
```

Para añadir registros a una tabla usamos la consulta **INSERT INTO**, que inserta un registro en la tabla que se especifique, colocando los valores que se

indican en los campos correspondientes. La sintaxis genérica de esta consulta es la que se muestra a continuación:

```
INSERT INTO tabla (campo_1, campo_2, campo_3, ..., campo_n) VALUES (valor_1, valor_2, valor_3, ..., valor_n);
```

Por ejemplo, para añadir el registro de un empleado a la correspondiente tabla usaríamos la siguiente consulta:

```
INSERT INTO empleados (numero, nombre, salario, categoria, sexo, departamento) VALUES (1, "Pedro García Fernández", 700, "Limpador", "M", "AL");
```

En ocasiones se complementan, como en el ejemplo que acabamos de ver, todos los campos del registro. En ese caso SQL permite omitir los nombres de los campos. La consulta anterior podría, por lo tanto, haber quedado así:

```
INSERT INTO empleados VALUES (1, "Pedro García Fernández", 700, "Limpador", "M", "AL");
```

Es imprescindible, en este caso, que los valores que aparecen estén en el mismo orden que los campos cuando se creó la tabla, ya que SQL respeta este orden y asigna los datos uno a uno.

También puede ser que no vayamos a grabar todos los campos del nuevo registro. Por ejemplo, en la estructura existen campos que admiten el valor NULL, es decir, que se permite crear el registro dejando ese campo sin rellenar si es necesario. Por ejemplo, suponga que aún no sabemos la categoría profesional con la que vamos a contratar a nuestro nuevo empleado. En un caso así, sí es obligatorio especificar la lista de campos que van a cumplimentarse. La consulta quedaría así:

```
INSERT INTO empleados (numero, nombre, salario, sexo, departamento) VALUES (1, "Pedro García Fernández", 700, "M", "AL");
```

Como ve, esta vez no se asigna ningún valor al campo "categoria". Esto no da ningún problema. Si mira la definición de la estructura de la tabla, verá que el campo admite el valor NULL y lo toma por defecto. Pero si es imprescindible que aparezca la relación de campos que vamos a cumplimentar para que SQL asigne cada valor al campo correcto.

Además, es necesario especificar la lista de nombres de campos en esta consulta si alguno de los campos numéricos se creó con el atributo

**AUTO\_INCREMENT**, ya que a un campo así no le asignaremos valor alguno al crear el registro. Es el propio motor de la BBDD el que se encarga de colocar el valor correspondiente en dicho campo. No olvide que estos campos se llenan automáticamente con valores numéricos secuenciales. Como norma, para evitarse problemas, especifique siempre la lista de campos en estas consultas.

Una vez que se hayan grabado registros en una tabla es muy posible que usted quiera localizar los datos de un registro determinado. Para ello, empleamos la consulta **SELECT**, que es muy versátil, lo que resulta interesante a la hora de localizar registros en grandes tablas, pero requiere una sintaxis muy amplia. En su forma más simple es como sigue:

```
SELECT campo_1, campo_2, ..., campo_n FROM tabla WHERE  
condición;
```

Esto recupera los campos especificados de la tabla indicada para aquellos registros que cumplan la condición. Por ejemplo, suponga la consulta siguiente:

```
SELECT nombre, salario FROM empleados WHERE  
departamento="AL";
```

Con esta consulta conseguiremos los nombres y salarios de todos los empleados que trabajen en el departamento especificado, por lo que del resto de los trabajadores no se obtendrán los datos solicitados.

Ahora suponga que lo que pretendemos es recuperar todos los datos de los registros que cumplan la condición especificada. No es, en este caso, necesario teclear la lista de campos. Basta con usar, en su lugar, un asterisco. La consulta quedará así:

```
SELECT * FROM empleados WHERE departamento="AL";
```

Y, si lo que deseamos es recuperar todos los datos, o algunos de ellos, de todos los registros de la tabla, las consultas adecuadas serían las siguientes:

```
SELECT * FROM empleados;
```

o bien:

```
SELECT nombre, salario FROM empleados;
```

Al suprimir la condición que se establece en la cláusula WHERE, la consulta se ejecuta sobre la totalidad de filas.

La cláusula WHERE, que permite establecer condiciones de selección de registros, es especialmente potente y flexible. Hablaremos más adelante de ella en profundidad. De momento, para los ejemplos que aparecen aquí haremos un uso simple de ella, tal como hemos visto hasta ahora. Más adelante, veremos cómo se pueden combinar condiciones mediante operadores y otras técnicas para lograr consultas tan precisas como sean necesarias.

A menudo, sobre todo si se trata de obtener informes a partir de los registros de una tabla, es conveniente poder agrupar éstos mediante ciertos criterios. Para ello, empleamos la cláusula **GROUP BY**. Suponga, por ejemplo, que desea obtener un listado de los nombres de los empleados, y sus salarios, agrupados por el departamento al que pertenecen. La consulta será la siguiente:

```
SELECT nombre, salario FROM empleados GROUP BY
departamento;
```

Con esto obtendremos el nombre y el salario de todos los trabajadores (no hay cláusula WHERE, aunque podría haberla si la necesitáramos), agrupados por departamento. Esto facilita mucho determinadas tareas. Por ejemplo, imagine que desea saber el coste mensual en salarios de cada departamento. SQL le proporciona la cláusula **SUM... AS**, que permite obtener sumas de valores numéricos. Mire la siguiente consulta:

```
SELECT nombre SUM (salario) AS totalSueldos FROM
empleados GROUP BY departamento;
```

Con esto lograremos que los registros de los empleados aparezcan agrupados por departamentos, bajo el titular **totalSueldos**, con un subtotal de los salarios para cada uno de los grupos. De este modo podríamos determinar el coste de la masa salarial de la empresa, por departamentos.

Otra operación a la que tenemos acceso es la posmediación aritmética de valores numéricos. Imagine que, a efectos estadísticos, deseamos calcular cuál es el sueldo medio de los empleados, es decir, sumando el total y dividiendo entre el número de empleados. Para ello, usaremos la cláusula **AVG...AS** (procedente de la palabra inglesa AVERAGE, que significa promedio). Vea la siguiente consulta:

```
SELECT nombre AVG (salario) AS media FROM empleados
GROUP BY departamento;
```

En ocasiones necesitamos obtener los datos de una tabla ordenados según un criterio, pero sin que se produzca una agrupación por bloques. Para ello, contamos con la cláusula **ORDER BY**. A su vez, la ordenación puede ser

ascendente (de menor a mayor), si no le añadimos ningún modificador o (dependiendo del motor de BBDD empleado) si añadimos el modificador **ASC**. Para lograr una ordenación descendente (de mayor a menor) recurriremos al modificador **DESC**. Observe la siguiente consulta:

```
SELECT nombre, salario FROM empleados ORDER BY salario  
DESC;
```

Con esta consulta obtendremos una relación de los nombres y salarios de los empleados, ordenados según el salario, desde el que cobra más al que menos. Como puede comprobar, el hecho de que un campo forme parte de la selección no esóbice para que, además, forme parte de los criterios empleados. Es un rasgo que confunde a muchos principiantes al empezar a diseñar consultas. SQL es muy flexible y, en poco tiempo, usted manejará estos detalles con soltura.

Lógicamente, tanto la cláusula GROUP BY, como ORDER BY pueden admitir más de un criterio. Vea la siguiente consulta:

```
SELECT * FROM empleados ORDER BY departamento ASC,  
salario DESC;
```

Con esto obtendremos todos los datos de todos los registros, ordenados ascendente por el departamento donde trabaja cada empleado. A su vez, dentro de cada departamento, estarán ordenados, descendente, por el salario.

A menudo es interesante saber cuántos registros tiene un grupo, o cuántos cumplen una condición. Para ello, SQL nos proporciona la cláusula **COUNT...AS**, que permite el conteo de registros. Vea el siguiente ejemplo:

```
SELECT COUNT (*) AS total FROM empleados, ORDER BY  
departamento;
```

Podemos establecer una condición basada en similitudes no exactas. Esto quiere decir que podemos crear un criterio sobre patrones. Veamos qué es esto. SQL nos proporciona el operador **LIKE**. Esta palabra, en inglés, es un comparativo que se refiere a la similitud entre dos cosas. Suponga que, en su tabla de empleados, tiene personal en el departamento de contabilidad, cuyo código es "CO" y en control de calidad, cuyo código es "CC". Además, como es lógico, tendrá gente en otros departamentos, pero usted quiere recuperar la lista de los empleados de estos dos que he mencionado. Vea la siguiente consulta:

```
SELECT nombre, salario FROM empleados WHERE  
departamento LIKE "C*";
```

El asterisco que aparece en la cadena que hay a continuación de LIKE significa “cualquier carácter o conjunto de caracteres”. Es decir, se recupera el nombre y el salario de la tabla de empleados para todos aquellos cuyo departamento empiece por la letra “C” y, a continuación, tenga cualquier cosa. Para ver otro ejemplo, suponga que desea ver la lista, con todos los datos, de los empleados cuyo nombre es “José”. La consulta será la siguiente:

```
SELECT * FROM empleados WHERE nombre LIKE "José*";
```

Otra cláusula muy interesante es **LIMIT**, que permite acotar la cantidad de registros obtenidos en una consulta. Por ejemplo, suponga que hacemos una consulta de selección de los empleados que trabajan en el departamento de producción, y resulta que son, digamos, 300.

Manejar tantos datos simultáneamente es, cuanto menos, engorroso e incómodo, así que trabajaremos, de momento, con los diez primeros empleados. Luego, con los diez siguientes y, así, sucesivamente. De este modo el trabajo será más llevadero. Vea la siguiente consulta:

```
SELECT * FROM empleados WHERE departamento="PR" LIMIT  
10;
```

Esto recuperará los diez primeros registros. Para obtener los diez siguientes recurriremos a la consulta que aparece a continuación:

```
SELECT * FROM empleados WHERE departamento="PR" LIMIT  
11, 20;
```

Como ve, la cláusula **LIMIT** debe ir seguida por el primer registro que deseamos recuperar, una coma y el último que queremos. Entonces la consulta obtiene los que están comprendidos entre ambos límites.

Antes hemos mencionado que la cláusula **WHERE** es muy flexible. Operadores como **LIKE** le dan más potencia de la que pueda parecer en principio. Pero aún hay más.

Podemos usar **operadores** para encadenar varias condiciones bajo una sola cláusula **WHERE**. En SQL consideraremos dos tipos de operadores: los **aritméticos** y los **lógicos**. La lista de operadores que podemos usar es la que aparece en la doble tabla de la figura 15.5.

Como ve son pocos pero, convenientemente combinados, tienen mucha potencia para el procesamiento de datos.

El asterisco que aparece en la cadena que hay a continuación de LIKE significa “cualquier carácter o conjunto de caracteres”. Es decir, se recupera el nombre y el salario de la tabla de empleados para todos aquellos cuyo departamento empiece por la letra “C” y, a continuación, tenga cualquier cosa. Para ver otro ejemplo, suponga que desea ver la lista, con todos los datos, de los empleados cuyo nombre es “José”. La consulta será la siguiente:

```
SELECT * FROM empleados WHERE nombre LIKE "José*";
```

Otra cláusula muy interesante es **LIMIT**, que permite acotar la cantidad de registros obtenidos en una consulta. Por ejemplo, suponga que hacemos una consulta de selección de los empleados que trabajan en el departamento de producción, y resulta que son, digamos, 300.

Manejar tantos datos simultáneamente es, cuanto menos, engorroso e incómodo, así que trabajaremos, de momento, con los diez primeros empleados. Luego, con los diez siguientes y, así, sucesivamente. De este modo el trabajo será más llevadero. Vea la siguiente consulta:

```
SELECT * FROM empleados WHERE departamento="PR" LIMIT  
10;
```

Esto recuperará los diez primeros registros. Para obtener los diez siguientes recurriremos a la consulta que aparece a continuación:

```
SELECT * FROM empleados WHERE departamento="PR" LIMIT  
11, 20;
```

Como ve, la cláusula **LIMIT** debe ir seguida por el primer registro que deseamos recuperar, una coma y el último que queremos. Entonces la consulta obtiene los que están comprendidos entre ambos límites.

Antes hemos mencionado que la cláusula **WHERE** es muy flexible. Operadores como **LIKE** le dan más potencia de la que pueda parecer en principio. Pero aún hay más.

Podemos usar **operadores** para encadenar varias condiciones bajo una sola cláusula **WHERE**. En SQL consideraremos dos tipos de operadores: los **aritméticos** y los **lógicos**. La lista de operadores que podemos usar es la que aparece en la doble tabla de la figura 15.5.

Como ve son pocos pero, convenientemente combinados, tienen mucha potencia para el procesamiento de datos.

<b>OPERADORES DE LA CLÁUSULA WHERE (aritméticos)</b>	
<b>OPERADOR</b>	<b>DESCRIPCIÓN</b>
<	Mayor que.
>	Menor que.
=	Igual a.
<=	Menor o igual que.
>=	Mayor o igual que.
<>	Distinto de.
<b>LIKE</b>	Parecido a.
<b>BETWEEN</b>	Comprendido entre dos valores.
<b>IN</b>	Se usa para hacer una consulta en una tabla que está en otra BBDD distinta de la que estamos usando.

<b>OPERADORES DE LA CLÁUSULA WHERE (lógicos)</b>	
<b>OPERADOR</b>	<b>DESCRIPCIÓN</b>
<b>AND</b>	Une dos condiciones. Deben cumplirse ambas.
<b>OR</b>	Une dos condiciones de modo que seleccionará los registros que cumplan, al menos, una de ellas.
<b>NOT</b>	Selecciona los registros que no cumplan la condición.

Figura 15.5

Por ejemplo, suponga que quiere obtener una lista de los empleados cuyo sueldo es mayor que 700 euros, pero menor que 900 y que, además, trabajan en el almacén, cuyo código de departamento es "AL". La consulta será la siguiente:

```
SELECT * FROM empleados WHERE salario>700 AND
salario<900 AND departamento="AL";
```

Ésta sería una forma de hacerlo. Y, en realidad, funciona. Pero también podemos hacerlo de otro modo, mediante el uso del operador BETWEEN. La consulta quedaría así:

```
SELECT * FROM empleados WHERE (salario BETWEEN 700 AND
900) AND departamento="AL";
```

El uso de paréntesis, en este caso, está destinado, simplemente, a facilitar la legibilidad de la consulta por nuestra parte. En otros casos pueden usarse paréntesis para agrupar condiciones.

Por ejemplo, suponga que queremos obtener la lista de los empleados que ganan 800 euros o más y que trabajan en el almacén, o bien que trabajan en

contabilidad (cuyo código de departamento puede ser, por ejemplo, "CO"), ganen lo que ganen. Vea la siguiente consulta:

```
SELECT * FROM empleados WHERE (salario>=800 AND
departamento="AL") OR departamento="CO";
```

Ahora observe la siguiente consulta. Es muy parecida, pero no igual:

```
SELECT * FROM empleados WHERE salario>=800 AND
(departamento="AL" OR departamento="CO");
```

Esta última nos proporciona los datos de todos los empleados cuyo salario sea de 800 euros o más y cuyo departamento sea el almacén o contabilidad, pero no muestra aquellos de contabilidad que ganen menos de 800.

El operador IN permite efectuar una consulta sobre otra base de datos diferente de la que estamos usando en este momento, así:

```
SELECT * FROM tabla WHERE campo="valor" IN "BBDD
alternativa";
```

Personalmente no soy partidario de esto. Trabaje con las tablas de la BBDD que está usando. Si tiene que cambiar de BBDD, hágalo.

Como ve, la cláusula WHERE puede llegar a ser muy potente, permitiendo que se efectúen consultas de selección tan filtradas como necesitemos. Pero, además, esta cláusula está también disponible, con toda su potencia y flexibilidad para las consultas de actualización y eliminación de registros.

A menudo es necesario cambiar el valor de un campo en uno o más registros. Por ejemplo, suponga que a un empleado le cambia de departamento o le sube el sueldo. Para todos los cambios que necesitemos realizar en los datos de nuestra BBDD contamos con **UPDATE**. Observe la siguiente consulta genérica:

```
UPDATE tabla SET campo_1= nuevoValor_1,
campo_2=nuevoValor2, ..., campo_n=nuevoValor_n WHERE
criterio;
```

Por ejemplo, suponga que desea cambiar a un empleado, cuyo nombre conoce, del departamento de contabilidad al de facturación, cuyo valor es, digamos "FA". La consulta será la siguiente:

```
UPDATE empleados SET departamento="FA" WHERE
nombre="Ángel Rodríguez Macías";
```

Como ve, no hay ninguna dificultad en mantener las tablas actualizadas, según sea necesario.

Ahora suponga que decide subir el sueldo un 5% a todos los empleados que, actualmente, cobren menos de 900 euros. Mire la siguiente consulta:

```
UPDATE empleados SET salario=salario*1.05 WHERE  
salario<900;
```

Por cierto, si es usted empresario, ésta es una buena idea. Y si es trabajador por cuenta ajena, le parecerá aún mejor. Bromas aparte, llega el momento de que aprendamos a eliminar registros de una tabla. Un empleado que se jubila, o se marcha de la empresa, por ejemplo, crearía esta necesidad. La consulta **DELETE** permite hacer, precisamente, esto. Su sintaxis genérica es la siguiente:

```
DELETE * FROM tabla WHERE criterio;
```

Por ejemplo, suponga que se marcha un empleado, del que sabemos el nombre. La consulta adecuada sería la siguiente:

```
DELETE * FROM empleados WHERE nombre="Pedro Martín  
Puig-de la Torre";
```

Por lo tanto, las consultas que podemos efectuar sobre los registros de una tabla son, a modo de resumen, las cuatro que aparecen a continuación

- **INSERT INTO**. Para añadir registros.
- **SELECT**. Para localizar registros.
- **UPDATE**. Para actualizar registros.
- **DELETE**. Para eliminar registros.

Y ahora vamos a ampliar un poco nuestro espectro de trabajo. Vamos a volver al caso del empleado que se jubila o se marcha. No debería bastarnos con eliminar el correspondiente registro de la tabla. Deberíamos tener otra tabla, que podría llamarse, por ejemplo, **exEmpleados**. En ella almacenaremos los datos de los empleados que ya no formen parte de la plantilla. Sin embargo, con lo que tenemos hasta ahora, la tarea es, conceptualmente, bastante engorrosa:

- En primer lugar, tenemos que buscar el registro del empleado que va a marcharse, en la tabla “empleados”, y anotar sus datos.
- A continuación, lo añadimos a la tabla “exEmpleados”.
- Por último, borramos el registro en “empleados”.

Afortunadamente, SQL permite manejar, en una misma consulta, campos de diferentes tablas de la misma BBDD. Para ello, basta con que precedamos el nombre de cada campo del de la tabla con la que queremos trabajar, separando ambos por un punto.

```
INSERT INTO exEmpleados (exEmpleados.numero,
exEmpleados.nombre, exEmpleados.salario,
exEmpleados.categoría, exEmpleados.sexo,
exEmpleados.departamento) VALUES (SELECT * FROM empleados
WHERE empleados.nombre= "Ana Márquez Contreras");
```

Con esto hemos copiado el registro de una persona, localizada por su nombre, de una tabla a la otra. Ahora ya sólo hay que eliminar el registro en la tabla original, así:

```
DELETE * FROM empleados WHERE nombre= "Ana Márquez
Contreras";
```

Fíjese en que, dentro de la consulta de inserción, como valores, aparecen los resultados de una consulta de selección. Esto es lo que se conoce como subconsultas. El motor de MySQL, sin embargo, no implementa esta técnica, así que la consideraremos meramente anecdótica. Otros motores de BBDD, basados en SQL, sí la soportan. En general, cada motor de BBDD tiene sus propias peculiaridades.

Recuerde que este capítulo le ha mostrado las principales características de SQL. No trate de aprenderlas de memoria. Úselo a modo de consulta cuando tenga dudas. En los dos próximos capítulos aprenderemos a implementar las consultas de este lenguaje en PHP, y veremos ejercicios prácticos que le ayudarán a comprender el manejo de las BBDD.

## CAPÍTULO 16

# BASES DE DATOS ODBC

---

---

PHP puede conectar con bases de datos de dos formas: mediante el uso directo de funciones propias del intérprete o a través del estándar ODBC (*Open Data Base Connectivity*). En este capítulo vamos a conocer la conectividad de scripts de PHP con motores de BBDD que trabajan bajo ODBC. En realidad, este no es el modo de trabajo más eficiente posible, ya que al ser necesario interponer el ODBC entre el script y la base de datos se reduce el rendimiento del sistema. Sin embargo, es necesario cuando se nos encarga una gestión y el motor de base de datos nos viene impuesto. Hace pocos años tuve que preparar un sistema para la Intranet de una empresa cuyos datos de trabajo estaban, por expreso deseo del cliente, en una base de datos de Microsoft Access. En ese caso es necesario recurrir a ODBC, y en este capítulo aprenderemos a manejarlo.

### 16.1 ESTABLECIENDO LA CONEXIÓN ODBC

En la carpeta correspondiente a este capítulo existe una base de datos de Microsoft Access que usaremos para trabajar. Se llama **comercial.mdb**. Esta BBDD no tiene ninguna tabla todavía. Nosotros crearemos dentro las que necesitemos, y los datos necesarios, desde nuestros scripts de PHP. Sin embargo, para poder trabajar con ella es necesario, previamente, establecer la conexión ODBC. Aquí vamos a detallar cómo se hace esto en un entorno Windows XP Professional, con el Service Pack 2. Si su entorno de trabajo es otro, consulte con el proveedor, o la ayuda en línea, de su Sistema Operativo.

Lo primero que debemos hacer es acceder al Panel de Control. Para ello, pulsamos el botón de Inicio y, en el menú de inicio, elegiremos “Panel de Control”.

Una vez que se abra el panel de control, seleccionaremos la opción **Rendimiento y Mantenimiento**. Esto abre una ventana en la que seleccionaremos **Herramientas Administrativas**. En la siguiente ventana haremos doble clic sobre el ícono **Orígenes de Datos**. Esto abre el gestor de ODBC, cuyo aspecto inicial aparece en la figura 16.1.

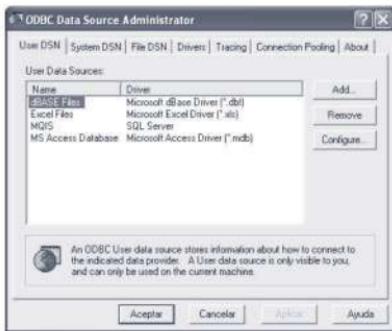


Figura 16.1

Pulse sobre la pestaña **System DSN** y, a continuación, sobre el botón **Add...**, con lo que se abrirá la ventana de la figura 16.2.



Figura 16.2

Seleccione la opción resaltada en la imagen y pulse el botón **Finalizar**. Se abrirá la ventana que aparece en la figura 16.3.



Figura 16.3

En la casilla etiquetada como **Nombre del origen de datos**: teclearemos un nombre para nuestra conexión ODBC. Ponga, por ejemplo, **ODBC\_1**. En la casilla que aparece como **Descripción**: pondremos una descripción suficientemente explícita para saber a qué BBDD se refiere la conexión que estamos creando. Escriba **BBDD para pruebas**. Ahora pulse el botón **Seleccionar...**. Se abre una ventana de selección de archivos. En ella elegiremos la base de datos comercial.mdb que hay en la carpeta correspondiente a este capítulo. La ventana quedará, entonces, con el aspecto de la figura 16.4.



Figura 16.4

Pulse el botón **Aceptar** para cerrar esta ventana. En la ventana de la figura 16.1 se ha añadido la conexión que acabamos de crear, así: **ODBC1 Microsoft Access Driver (\*.mdb)**. Pulse, también aquí, el botón **Aceptar** para terminar con el proceso. Ya hemos creado la conexión ODBC necesaria para trabajar.

## 16.2 USO BÁSICO DE ODBC

Como usted, sin duda, ya habrá imaginado, después de ver las técnicas de trabajo con otros aspectos de PHP, el trabajo con una base de datos se basa en tres etapas fundamentales:

- Establecer contacto con la BBDD.
- Realizar las operaciones necesarias.
- Cerrar los recursos empleados.

Vamos a empezar por el principio: abriendo la BBDD y autenticándonos para que luego podamos trabajar. Además, vamos a ver ahora la forma de cerrar la conexión cuando el trabajo haya terminado. Para lo primero, recurrimos a la función *odbc\_connect()*. Ésta recibe tres argumentos, que son los siguientes:

- El nombre de la conexión ODBC (nombre del origen de datos), tal como lo asignamos en la ventana de la figura 16.4.
- El nombre de usuario.
- La clave de acceso. Estos dos últimos parámetros no los hemos establecido en nuestra conexión para facilitar los ejemplos.

Esta función devuelve un manejador que será el que empleemos en todas las operaciones que necesitemos realizar sobre la BBDD. Si se produce un error y la función no puede abrir la base de datos, devuelve un valor lógico false. Cuando hayamos terminado de trabajar, deberemos liberar los recursos empleados. Esto significa, de momento, que debemos cerrar la BBDD. Para ello, recurrimos a la función *odbc\_close()*. Ésta recibe, como argumento, el manejador que obtuvimos con *odbc\_connect()*.

Veamos un primer paso en el script **abreCierra.php**, listado a continuación:

```
<?php
    define ("salto","<br>\n");
    // Se intenta establecer la conexión con la base de
datos.
    $conexionODBC=odbc_connect ("ODBC1", "", "");
    // Se comprueba si se ha establecido y se informa de
ello.
    if ($conexionODBC) {
        echo ("LA CONEXIÓN ODBC HA SIDO
ESTABLECIDA." .salto);
    } else {
```

```
die ("NO SE PUDO ESTABLECER LA CONEXIÓN  
ODBC.");  
}  
// Se cierra la conexión.  
odbc_close ($conexionODBC);  
echo ("LA CONEXIÓN ODBC HA SIDO CERRADA.".salto);  
?>
```

Observe, en las líneas resaltadas, cómo hemos usado las dos funciones de las que acabamos de hablar. Vea que, al establecer la conexión, no hemos puesto nombre de usuario ni contraseña. Esto es porque, cuando creamos la conexión ODBC, en el apartado anterior, no establecimos estos parámetros, de modo que ahora no los necesitamos. En una plataforma más simple, como Windows ME, por ejemplo, habríamos creado un usuario. Pero con Windows XP Professional no lo hemos hecho para que no importe en qué sesión estamos trabajando. El resultado de este script, si todo va bien, debe ser el siguiente:

**LA CONEXIÓN ODBC HA SIDO ESTABLECIDA.  
LA CONEXIÓN ODBC HA SIDO CERRADA.**

Y ahora, que ya tenemos la base de datos y podemos acceder a ella, vamos a crear las tablas necesarias. Para llevar a cabo las consultas necesarias sobre la BBDD, contamos con la función *odbc\_do()*, que recibe dos parámetros. El primero es el manejador de la conexión obtenido mediante *odbc\_connect()*. El segundo es la consulta que vamos a realizar, en una cadena alfanumérica. Observe el listado del script *crearTablas.php*:

```
<?php  
define ("salto","<br>\n");  
// Se establece la conexión con la base de datos.  
$conexionODBC=odbc_connect ("ODBC1", "", "");  
// Se comprueba si se ha establecido y se informa de  
ello.  
if ($conexionODBC){  
    echo ("LA CONEXIÓN ODBC HA SIDO  
ESTABLECIDA.".salto);  
} else {  
    die ("NO SE PUDO ESTABLECER LA CONEXIÓN ODBC");  
}  
  
// Consulta para crear la tabla de productos  
$consulta="CREATE TABLE productos (codigoProducto  
INTEGER PRIMARY KEY, descripción CHAR(40), pCompra FLOAT,  
pVenta FLOAT, stock INTEGER);";  
// Se ejecuta la consulta.
```

```

    odbc_do ($conexionODBC, $consulta);
    // Consulta para crear la tabla de productos
    $consulta="CREATE TABLE empleados (codigoEmpleado
INTEGER PRIMARY KEY, nombre CHAR(50), salario FLOAT,
departamento CHAR ( 2 ) ;";
    // Se ejecuta la consulta.
    odbc_do ($conexionODBC, $consulta);

    // Se cierra la conexión.
    odbc_close ($conexionODBC);
    echo ("LA CONEXIÓN ODBC HA SIDO CERRADA.".salto);
?>

```

Vea, en el fragmento sombreado, cómo se crean las consultas necesarias en forma de cadenas y cómo se incorporan a la función `odbc_do ()` para ser ejecutadas. Cargue el script en el navegador. Cuando haya terminado abra la base de datos con Microsoft Access para comprobar que, efectivamente, han sido creadas las dos tablas.

Y aquí vamos a hacer un pequeño inciso. Si bien en el capítulo anterior conocimos las posibles consultas que ofrece SQL, cada motor de BBDD presenta ciertas peculiaridades, como ya mencioné en dicho capítulo. El motor de Access, por ejemplo, no permite incluir la cláusula IF NOT EXISTS, al crear una tabla. Si lo intenta, se produce un error de sintaxis. Si ahora intenta ejecutar de nuevo el script, verá que se producen sendos errores al tratar de crear las tablas, porque éstas ya existen. Los errores que aparecen son los siguientes:

**Warning:** `odbc_do()` [function.odbc-do]: SQL error: [Microsoft][Controlador ODBC Microsoft Access] La tabla 'productos' ya existe., SQL state S0001 in SQLExecDirect in C:\Documents and Settings\Jose\Mis documentos\Mis webs dinamicas\capítulo\_16\crearTablas.php on line 15

**Warning:** `odbc_do()` [function.odbc-do]: SQL error: [Microsoft][Controlador ODBC Microsoft Access] La tabla 'empleados' ya existe., SQL state S0001 in SQLExecDirect in C:\Documents and Settings\Jose\Mis documentos\Mis webs dinamicas\capítulo\_16\crearTablas.php on line 20

Una forma de evitar los mensajes de error que aparecen es anteponiendo el operador `@` a la función `odbc_do ()`, y asignando el resultado de la misma a una variable ya que, si no se ha podido ejecutar, devuelve un valor false. El código puede ser como el del script **crearTablasExistentes.php**:

```

<?php
define ("salto","<br>\n");

```

```
// Se intenta establecer la conexión con la base de
datos.
$cexionODBC=odbc_connect ("ODBC1", "", "");
// Se comprueba si se ha establecido y se informa de
ello.
if ($conexionODBC){
    echo ("LA CONEXIÓN ODBC HA SIDO
ESTABLECIDA.".salto);
} else {
    die ("NO SE PUDO ESTABLECER LA CONEXIÓN ODBC");
}
// Consulta para crear la tabla de productos
$consulta="CREATE TABLE productos (codigoProducto
INTEGER PRIMARY KEY, descripcion CHAR(40), pCompra FLOAT,
pVenta FLOAT, stock INTEGER);";
// Se ejecuta la consulta.
$resultadoDeLaConsulta=@odbc_do ($conexionODBC,
$consulta);
// Se comprueba si se ha podido realizar la consulta.
if ($resultadoDeLaConsulta){
    echo ("La tabla de productos se ha creado
satisfactoriamente.");
} else {
    echo ("La tabla de productos no se ha podido
crear.");
}
echo (salto);
// Consulta para crear la tabla de productos
$consulta="CREATE TABLE empleados (codigoEmpleado
INTEGER PRIMARY KEY, nombre CHAR(50), salario FLOAT,
departamento CHAR (2));";
// Se ejecuta la consulta.
$resultadoDeLaConsulta=@odbc_do ($conexionODBC,
$consulta);
// Se comprueba si se ha podido realizar la consulta.
if ($resultadoDeLaConsulta){
    echo ("La tabla de productos se ha creado
satisfactoriamente.");
} else {
    echo ("La tabla de productos no se ha podido
crear.");
}
echo (salto);

// Se cierra la conexión.
odbc_close ($conexionODBC);
echo ("LA CONEXIÓN ODBC HA SIDO CERRADA.".salto);
?>
```

Observe, en los fragmentos resaltados, cómo hemos creado unos filtros para evitar los mensajes de error de PHP e informar al usuario de cómo se ha desarrollado la creación de las tablas. Abra la base de datos con Access y elimine, manualmente, las dos tablas creadas. A continuación, cierre la base de datos y ejecute este último script. Verá que el mensaje es de que se han podido crear las dos tablas, tal como se muestra a continuación:

**LA CONEXIÓN ODBC HA SIDO ESTABLECIDA.**  
**La tabla de productos se ha creado satisfactoriamente.**  
**La tabla de productos se ha creado satisfactoriamente.**  
**LA CONEXIÓN ODBC HA SIDO CERRADA.**

Ahora cierre el navegador y vuelva a ejecutar el script. El resultado es el que cabía esperar:

**LA CONEXIÓN ODBC HA SIDO ESTABLECIDA.**  
**La tabla de productos no se ha podido crear.**  
**La tabla de productos no se ha podido crear.**  
**LA CONEXIÓN ODBC HA SIDO CERRADA.**

Al estar ya creadas las tablas, no se pueden crear de nuevo. Y esto nos lleva a dos funciones muy interesantes en la depuración de scripts. La primera de ellas es *odbc\_error()*. La segunda es *odbc\_errormsg()*. Ambas reciben, como único parámetro, el manejador de la conexión obtenido con *odbc\_connect()*. Si se produce un error, la primera función devuelve el código que tiene ese error. Todos los posibles errores que se puedan llegar a producir en la ejecución de una consulta cualquiera tienen asignado un código. La segunda función devuelve un mensaje de error explicativo. Estas funciones devuelven la información del error producido en la última consulta ejecutada. Vea el listado del script **comprobarErrores.php**. Aquí le reproduczo, únicamente, el fragmento que nos interesa ahora. El resto del script es como los anteriores.

```
// Se ejecuta la consulta.  
$resultadoDeLaConsulta=@odbc_do ($conexionODBC,  
$consulta);  
// Se comprueba si se ha podido realizar la consulta.  
if ($resultadoDeLaConsulta){  
    echo ("La tabla de productos se ha creado  
satisfactoriamente.".salto);  
} else {  
    echo ("La tabla de productos no se ha podido  
crear.".salto);  
    echo ("El número de error es:  
".odbc_error($conexionODBC).salto);
```

```
echo ("La descripción del error es:  
".odbc_errormsg($conexionODBC).salto);  
}
```

Si ve el listado completo en el script que hay en el disco, comprobará que esta técnica se ha implementado a la creación de ambas tablas. Vea, en las líneas resaltadas, cómo se han usado estas funciones. Si ejecuta el script cuando las tablas ya existen, obtendrá, en la página, la información acerca del error, tanto el código del mismo, como el mensaje asociado, así:

**LA CONEXIÓN ODBC HA SIDO ESTABLECIDA.**  
**La tabla de productos no se ha podido crear.**  
**El número de error es: S0001**  
**La descripción del error es: [Microsoft][Controlador ODBC Microsoft Access] La tabla 'productos' ya existe.**  
**La tabla de productos no se ha podido crear.**  
**El número de error es: S0001**  
**La descripción del error es: [Microsoft][Controlador ODBC Microsoft Access] La tabla 'empleados' ya existe.**  
**LA CONEXIÓN ODBC HA SIDO CERRADA.**

Como ve, podemos personalizar la salida de errores en el navegador, para gestionar la base de datos de un modo más elegante.

### 16.3 AMPLIANDO CONSULTAS

Las tablas se crean una vez, y ya quedan ahí para su uso. En este apartado vamos a conocer la forma de efectuar lo que podría ser la gestión diaria de la base de datos. En primer lugar, crearemos un script que nos permita dar de alta nuevos productos. En realidad, serán dos páginas. La primera es una interfaz de usuario con un formulario para registrar el nuevo producto. La segunda es un script que graba el producto en la tabla. Previamente, sin embargo, debe hacer una comprobación de que el código de producto, que es clave primaria, no existe ya. Si existe, debe informar al usuario de que dicho código ya ha sido asignado. Y aquí empieza a ponerse interesante. Para comprobar si un valor de un campo ya existe en una base de datos podemos hacerlo de dos maneras:

- Si el campo es clave primaria (como ocurre en este caso con el código del producto), al tratar de grabar un registro con un valor duplicado se produce un error que, como hemos visto, puede ser capturado y dar lugar a un mensaje en la página.

- Tanto si el campo es clave primaria como si no, podemos buscar ese valor en la tabla y determinar si ha sido encontrado. En nuestro ejemplo nos vamos a centrar en este último sistema por ser más profesional y eficiente.

Para comprobar la existencia de un valor en un campo de la tabla, contamos con la función *odbc\_fetch\_row()*, que devuelve un valor booleano true si hay registros recuperados por una selección. Si no hay registros, esta función devuelve el valor booleano false. Y esto es, justamente, lo que necesitamos. En primer lugar, se hace la consulta, buscando registros cuyo campo **codigoProducto** coincida con el código del nuevo artículo que deseamos registrar. Si existe un registro que cumpla esta condición, se avisa de ello. Si no, se graba el nuevo artículo.

El formulario destinado a recopilar los datos proporcionados por el usuario es el listado **altaProductos.htm**. Este no contiene nada especial. Es sólo código HTML y JavaScript, con un formulario. El código JavaScript es un filtrado previo de los datos del formulario, para asegurarnos de que en los campos que deban ser numéricos, no se hayan tecleado datos alfanuméricos. La única licencia que se permite es la coma para los decimales (tiene que ser coma, no punto, ya que estos datos se grabarán en una tabla de Access que, en la configuración en español, separa los decimales con una coma).

Estos datos son enviados a **altaProductos.php**. Éste es, realmente, el que nos interesa. Su listado es el siguiente:

```
<?php
    define ("salto","<br>\n");
    // Se intenta crear la conexión con la base de datos.
    $conexionODBC=odbc_connect ("ODBC1", "", "");
    if (!$conexionODBC){ // Si no se ha establecido.
        die ("NO SE PUDO ESTABLECER LA CONEXIÓN ODBC");
    }
    // Se comprueba si el código ya existe en la tabla.
    $consulta="SELECT * FROM productos WHERE
codigoProducto=".$_POST["mCodigo"]."";
    $ejecutaConsulta=odbc_do($conexionODBC, $consulta);
    $registroExiste=odbc_fetch_row ($ejecutaConsulta);
    // Si existe, se informa de la duplicidad. Si no, se
graba el registro.
    if ($registroExiste){
        echo ("El código ya existe. No se puede
grabar.".salto);
    } else {
```

```
$consulta="INSERT INTO productos VALUES
('".$_POST["mCodigo"]."','".$_.POST["mDescripcion"]."',
'".$_POST["mPrecioCompra"]."','".$_.POST["mPrecioVenta"]."',
'".$_POST["mStock"]."');";
    odbc_do($conexionODBC, $consulta);
    echo ("REGISTRO GRABADO".salto);
}
odbc_close ($conexionODBC); // Cierro la base de
datos.
?>
```

Observe, en la primera linea resaltada, cómo hemos usado la función `odbc_fetch_row()`, para determinar si, como consecuencia de una consulta de selección, se han obtenido registros o no. Fíjese en que la ejecución de la consulta es asignada a una variable (`$ejecutaConsulta`), que luego sirve de argumento a la función que nos ocupa.

En el fragmento resaltado que aparece más abajo vemos que, si el código no existía previamente, se graba el registro completo. Cargue en el navegador el formulario **altaProducto.htm**, y registre algunos productos para comprobar el funcionamiento.

Del mismo modo que se emplea `odbc_do()` para realizar consultas de inserción o de selección, puede emplearse para llevar a cabo consultas de actualización o de eliminación de registros, creando éstas tal como aprendimos en el capítulo anterior.

En ocasiones, es necesario dirigir el resultado de una consulta de selección a la página que se le envía al usuario. Si no es muy importante el aspecto del resultado final, podemos usar la función `odbc_result_all()`, que genera una tabla HTML con los resultados de la selección. Esta función recibe dos argumentos. El primero es el resultado de ejecutar la consulta de selección. El segundo es una cadena con el formato de la tabla. Vea un ejemplo de uso en el script **verMatrizDeProductos.php**:

```
<?php
define ("salto","<br>\n");

// Se intenta establecer la conexión con la base de
datos.
$conexionODBC=odbc_connect ("ODBC1", "", "");

// Se comprueba si se ha establecido.
if (!$conexionODBC){
    die ("NO SE PUDO ESTABLECER LA CONEXIÓN ODBC");
```

```

        }
        // Se seleccionan todos los registros de la tabla de
        productos.
        $consulta="SELECT * FROM productos;";
        $ejecutaConsulta=odbc_do($conexionODBC, $consulta);

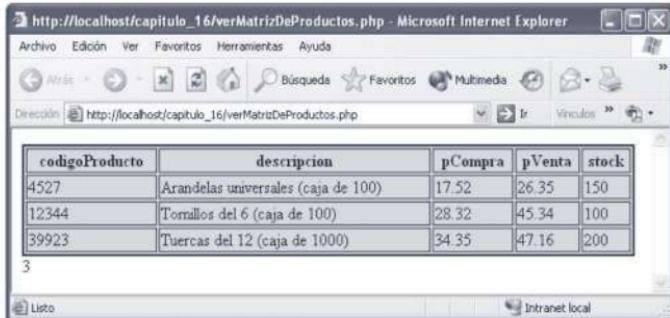
        // Se muestran en una tabla de HTML.
        $formato="width='600', bgcolor='pink', border='2',
        bordercolor='blue'";
        $registros=odbc_result_all ($ejecutaConsulta,
        $formato);
        echo ($registros);

        // Se cierra la base de datos.
        odbc_close ($conexionODBC);
    ?>

```

En la línea que aparece resaltada vemos el uso de esta función. El resultado que se muestra está formado por la tabla HTML con todos los datos. Previamente a la ejecución del script he creado tres productos, a modo de ejemplo.

Vea el resultado en la figura 16.5.



The screenshot shows a Microsoft Internet Explorer window displaying a table of product data. The table has five columns: 'codigoProducto', 'descripcion', 'pCompra', 'pVenta', and 'stock'. The data is as follows:

codigoProducto	descripcion	pCompra	pVenta	stock
4527	Arandelas universales (caja de 100)	17.52	26.35	150
12344	Tornillos del 6 (caja de 100)	28.32	45.34	100
39923	Tuerces del 12 (caja de 1000)	34.35	47.16	200

Below the table, the number '3' is displayed, indicating the total number of rows. At the bottom of the browser window, there are links for 'Lista' and 'Intranet local'.

Figura 16.5

Observe que, a pie de tabla, aparece el número total de registros.

Cuando se ha ejecutado una consulta, podemos obtener bastante información del resultado de la misma. Por ejemplo, contamos con la función *odbc\_num\_fields()*, que devuelve el número de campos que forman la tabla. Esta función recibe, como argumento, el resultado de una consulta de selección.

También contamos con la función *odbc\_num\_rows()*. Como la anterior, recibe, a modo de parámetro, el resultado de una consulta de selección. Devuelve el número de registros obtenidos por dicha consulta. Si no se ha obtenido ningún registro, devuelve 0. Si se ha producido algún error, devuelve -1. Esta función no opera correctamente con bases de datos de Access, devolviendo el resultado -1. Por lo tanto, para contar los registros que se obtienen a partir de una consulta de selección deberemos usar la cláusula COUNT de SQL que conocimos en el capítulo anterior. Vea el script **contarRegistros.php**:

```
<?php
    define ("salto","<br>\n");
    // Se intenta establecer la conexión con la base de
    datos.
    $conexionODBC=odbc_connect ("ODBC1", "", "");
    // Se comprueba si se ha establecido.
    if (!$conexionODBC){
        die ("NO SE PUDO ESTABLECER LA CONEXIÓN ODBC");
    }

    // Se seleccionan todos los registros de la tabla de
    productos con COUNT para contarlos.
    $consulta="SELECT COUNT (*) FROM productos;";
    $ejecutaConsulta=odbc_do($conexionODBC, $consulta);
    // Se obtiene el total de registros.
    $total=odbc_result ($ejecutaConsulta,1);
    echo ("El total de registros es $total." . salto);
    // Se cierra la base de datos.
    odbc_close ($conexionODBC);
?>
```

Observe la línea resaltada. La función *odbc\_result()* recibe dos argumentos. El primero es el resultado de ejecutar una consulta de cuenta. El segundo es un 1, para que seleccione adecuadamente el campo (enseñaremos este parámetro). El resultado de la función es el número total de filas obtenidas por la consulta. Como yo he dado de alta tres productos para las pruebas, en la página obtengo lo siguiente:

**El total de registros es 3.**

Lo más lógico es que, una vez que sabe cómo obtener el resultado de una consulta (que, por cierto, en el entorno de BBDD se conoce con el nombre de *cursor*), quiera hacer algo con los datos que ha recuperado de la tabla. Vea el script **listarRegistros.php**, que nos muestra cómo podemos recorrer un cursor y nos enseña algunos conceptos. Observe el código reproducido a continuación, que maneja un cursor de Base de Datos.

```
<html>

    <head><title>Leyendo registros desde
Access</title></head>

    <body>
        <table border="2" width="400">
            <tr><th>Código</th>
            <th>Descripción</th></tr>
            <?php
                // Se establece la conexión con la BBDD
                $conexionODBC = odbc_connect("ODBC1", "", "");

                // Se crea la consulta de selección.
                $consulta = "SELECT codigoProducto, descripcion
FROM productos;";

                // Se ejecuta la consulta, obteniendo un cursor.
                $ejecutaConsulta = odbc_do($conexionODBC,
$consulta);
                // Se obtiene el número de campos del cursor.
                $totalDeCampos =
odbc_num_fields($ejecutaConsulta);
                // Se recorre el cursor.
                while (odbc_fetch_row($ejecutaConsulta)){
                    echo "<tr>";
                    for ($campo = 1; $campo <= $totalDeCampos;
$campo++){
                        echo "<td align='center'>";
                        if (odbc_result($ejecutaConsulta, $campo) != null)
                            echo odbc_result($ejecutaConsulta,
$campo);
                        echo "</td>";
                    }
                    echo "</tr>";
                }
                // Se liberan recursos y se cierra la conexión.
                odbc_free_result($ejecutaConsulta);
                odbc_close($conexionODBC);
            ?>
        </table>
    </body>
</html>
```

Fíjese en cómo usamos `odbc_fetch_row()` para recorrer el cursor. El resultado de este script aparece en la figura 16.6.



Figura 16.6

Y, llegados a este punto, usted dirá que este resultado se parece mucho al de la figura 16.5, obtenido con el script `verMatrizDeProductos.php`. Y sí. Tiene razón, pero menos. Mientras que con aquél obteníamos toda la tabla HTML en una sola instrucción, ahora podemos acceder a cada dato de forma individual, y usarlo para comparaciones u otras finalidades. Vea, en la línea resaltada, el uso del segundo parámetro de `odbc_result()`. Corresponde al campo cuyo valor queremos recuperar, teniendo en cuenta que los campos se numeran a partir de 1, según están definidos en la estructura de la tabla.

Si ha seguido los procesos que hemos llevado a cabo verá que son bastante rígidos. Aunque PHP proporciona algunas otras funciones para la gestión de BBDD de tipo ODBC (vea el manual oficial en [www.php.net](http://www.php.net)), éstas se implementan a través de esta interfaz; no son funciones nativas del lenguaje. Esto redunda en una menor flexibilidad, además de un mayor tiempo de ejecución de los scripts, lo que, en tablas con muchos registros, puede resultar engoroso en grado sumo. En el próximo capítulo conoceremos la tecnología MySQL, mucho más flexible, potente y eficiente. No obstante, si debe usar alguna vez ODBC, aquí tiene una referencia que puede ayudarle.



## CAPÍTULO 17

### MYSQL

---

---

Este capítulo está dedicado al estudio del gestor MySQL. Éste será el motor de BBDD por excelencia para nuestras aplicaciones para Internet o para redes locales. Si bien el capítulo anterior lo hemos dedicado a BBDD basadas en ODBC, no usaremos, salvo circunstancias muy especiales, ese sistema. Y esto, por las siguientes razones:

- Suponga que debe implementar una BBDD Access. Eso obliga a tener Microsoft Access corriendo en el servidor. Pero Microsoft Access sólo corre sobre sistemas operativos de Microsoft. Ya tenemos el coste de dos licencias (por una parte el S.O. y por otra, el propio Access). Desde luego, existen licencias de pago para MySQL, que ofrecen soporte técnico al profesional, pero usted puede implementar y usar MySQL, de forma totalmente operativa con una licencia gratuita.
- Además, una BBDD Access ocupa más espacio en disco que la equivalente en MySQL. Y si bien Microsoft Access implementa herramientas útiles, éstas son necesarias en el caso de aplicaciones de usuario, pero no son necesarias cuando se trabaja con PHP.
- Para usar Microsoft Access ha sido necesario crear una conexión ODBC, implementando esta tecnología entre nuestros scripts y el motor de BBDD, como hemos visto al principio del capítulo anterior. Utilizando MySQL nos ahorraremos este paso.
- Las funciones de gestión de BBDD por ODBC no son nativas de PHP, sino que están implementadas como una API. Eso redunda en una mayor rigidez y limitaciones.

En este capítulo vamos a conocer el uso de MySQL desde PHP.

Fíjese en que, a diferencia de lo que ocurre en el capítulo anterior, en la carpeta de ejercicios correspondiente a este capítulo no hay ninguna base de datos, ni la habrá. Esto es porque las BBDD de MySQL se crean en otra parte del servidor, no en la carpeta donde se alojan los scripts. ¿Cuál es esa otra parte? Bueno. Si usted ha instalado su servidor local (con el que está haciendo las pruebas) tal como le enseñé en el capítulo 2, tiene una carpeta en su disco duro llamada **AppServ**. Dentro de ella hay, entre otras, una carpeta llamada **mysql**. A su vez, dentro de ella aparece otra, llamada **data**. Dentro de esta última aparece cada base de datos que usted use, como una carpeta. Las dos que hay, **mysql** y **test**, se crean por defecto al instalar MySQL. Pero todo esto, en realidad, no nos importa. Es una mera curiosidad. ¿Cómo? ¿Qué estoy diciendo? ¿Qué no nos importa donde se alojan las BBDD con las que trabajemos? Pues, por chocante que le pueda resultar, así es. Es otra ventaja de MySQL: la abstracción. Nuestros scripts de PHP, a través de las funciones adecuadas, le dan instrucciones al motor de MySQL y es éste el encargado de ir a la BBDD y ejecutar las consultas necesarias. Nosotros no debemos preocuparnos de más. Y así es como debe ser. A lo largo de este capítulo vamos a conocer las técnicas necesarias para llevar a buen puerto nuestro trabajo. En primer lugar conoceremos las principales funciones que implementa PHP para la gestión de BBDD. Después veremos una aplicación completa que nos enseñará todo lo que necesitamos saber al respecto. Finalizaremos el capítulo con el uso de una de las más populares herramientas para el uso de MySQL: el **PHPMyAdmin**.

## 17.1 INTRODUCCIÓN A MYSQL

Para manejar una base de datos MySQL, así como para crearla, crear, modificar o eliminar tablas, o cualquier otra operación que deseemos llevar a cabo tenemos que empezar haciendo que nuestro script se conecte al motor de MySQL. Recuerde lo que le he comentado unas líneas más arriba: el script da las instrucciones al motor de la BBDD y es éste quien se encarga de operar sobre ella.

Para establecer esta conexión desde PHP contamos con la instrucción **mysql\_connect()**, que recibe los siguientes argumentos:

- Dirección del servidor, o el nombre del mismo.
- Puerto. Se refiere al puerto por el que escucha el servidor MySQL. El puerto por defecto es 3306. Si éste es el que usted tiene configurado (y lo es, si instaló AppServ como se describe en el capítulo 2), no es necesario especificarlo.
- Ruta del socket. Este parámetro sólo se usa en servidores Unix o Linux. En servidores Windows no tiene uso. Por defecto es “/tmp/mysql.sock”.

- Nombre del usuario. Es el que usted puso al instalar el AppServ. Revise el capítulo 2. Si siguió mis instrucciones, el nombre de usuario es "mysql".
- Contraseña. Es la que se estableció al instalar el AppServ. En el capítulo 2 le indiqué que dejara este campo en blanco, así que la contraseña es una cadena vacía.

Observe el script **conectarBBDD.php**, listado a continuación:

```
<?php  
$conectado = @mysql_connect ("localhost","mysql","");
if ($conectado){
    echo ("SE HA CONECTADO AL SERVIDOR MySQL.");
} else {
    echo ("NO SE PUDO CONECTAR AL SERVIDOR MySQL.");
}?>
```

Observe la linea resaltada, donde se usa la función destinada a conectar con el motor de MySQL. Vea que esta función se ha precedido con el operador de control de errores para que, si se produce algún problema, no nos salga en la página ninguno de los antiestéticos mensajes que ofrece PHP en tales casos. Como ve, la función devuelve un manejador que será usado más adelante. En este script, el manejador se emplea para poder notificarle si la conexión se ha establecido o no. En caso de que la conexión no haya podido establecerse, las tres causas posibles son las siguientes:

- El servidor no se ha instalado de acuerdo a las instrucciones que le ofrecí en el capítulo 2. En ese caso deberá desinstalar el AppServ e instalarlo de nuevo.
- Por alguna razón, el servidor MySQL se ha detenido. Si siguió las instrucciones del capítulo 2, está instalado de tal modo que usted puede reiniciar su ordenador y el servidor arrancará sin problemas. Si no es así, desinstale el AppServ e instálelo de nuevo siguiendo mis instrucciones.
- Una causa muy común es que el servidor MySQL reconoce, por defecto, un usuario llamado **root**. Para aquel lector familiarizado con entornos Unix-Linux, esta palabra tiene un significado especial. En todo caso, cambie la línea resaltada en el script por la siguiente:

```
$conectado = @mysql_connect ("localhost","root","");

```

Con esto le funcionará. Quiero que comprenda algo sobre este mecanismo. Cuando una línea de un script establece conexión con el servidor MySQL, esta

conexión permanece operativa mientras se está ejecutando el script. Una vez que la ejecución concluye, la conexión se cierra de modo automático. Más adelante, veremos el modo de que no ocurra así, por medio de las llamadas *conexiones persistentes* pero, por ahora, considere este comportamiento por defecto.

Una vez establecida la conexión con el motor de BBDD, lo primero en lo que tenemos que pensar es en crear la base de datos sobre la que, posteriormente, trabajaremos. Para ejecutar una consulta de SQL contamos con la función *mysql\_query ()*, que recibe dos argumentos: la consulta a ejecutar y el manejador de la conexión. Si recuerda los conceptos que vimos en el capítulo 15 (éste es un buen momento para echarle un vistazo) le será muy fácil crear la consulta adecuada. Vea el listado del script llamado *crearBBDD.php*:

```
<?php
    define ("salto","\n<br>");
    // Se intenta conectar con el motor de MySQL
    $conectado = @mysql_connect ("localhost","root","");
    if ($conectado){
        echo ("SE HA CONECTADO AL SERVIDOR MySQL.".salto);
    } else {
        // Si la conexión falla, se avisa de ello y se
        interrumpe la ejecución.
        die ("NO SE PUDO CONECTAR AL SERVIDOR MySQL.");
    }
    // Se forma la consulta para crear la BBDD.
    // Como ve, se almacena en una cadena
    $consulta="CREATE DATABASE IF NOT EXISTS gestion;";
    // Se ejecuta la consulta.
    $hacerConsulta=mysql_query ($consulta, $conectado);
?>
```

Fíjese en lo que hemos hecho en las líneas resaltadas. En primer lugar, hemos creado la consulta que necesitamos y la hemos almacenado en una cadena. A continuación, hemos usado la función que estamos estudiando para que se ejecute la consulta. Quiero que repare en el hecho de que el resultado de *mysql\_query ()* se ha asignado a una variable. En este caso, esto nos lo podríamos haber ahorrado. Sin embargo, en muchas ocasiones (sobre todo en consultas de selección), esta variable es un *cursor*, con el resultado de la consulta, para trabajar posteriormente, con él.

Esta parte es la más importante del capítulo, porque el 95% de todo el trabajo que un webmaster lleva a cabo sobre bases de datos se basa en estas dos sentencias, tal como veremos en seguida. La cuestión es elaborar las consultas adecuadas en cada caso. Esa habilidad se desarrolla con el tiempo y la

experimentación, así como estudiando soluciones similares llevadas a cabo por otros programadores. Vamos a crear un par de tablas para almacenar los productos que se venden en una hipotética empresa y los proveedores de la misma. En primer lugar, diseñaremos (sobre papel) la estructura de datos que necesitamos, a priori. Aunque, como es lógico, esta estructura se podrá modificar si es necesario, cuanto mejor hecho esté el análisis inicial de necesidades del sistema, y menos modificaciones se lleven a cabo después, mejores y más rápidos resultados obtendremos. La tabla de proveedores podría tener una estructura como la siguiente:

- **codigoProveedor.** Campo alfanumérico de tres caracteres. Será clave principal.
- **nombreProveedor.** Campo alfanumérico de longitud variable, con un máximo de cuarenta caracteres.
- **direccionProveedor.** Campo alfanumérico de longitud variable, con un máximo de ochenta caracteres.
- **telefonoProveedor.** Campo alfanumérico de nueve caracteres. A través del script se verificará que sólo se tecleen dígitos.
- **ciudadProveedor.** Campo alfanumérico de longitud variable, con un máximo de cuarenta caracteres.
- **provinciaProveedor.** Campo alfanumérico de longitud variable, con un máximo de veinte caracteres.
- **emailProveedor.** Campo alfanumérico de longitud variable de hasta ochenta caracteres.

Observe una cosa. En los nombres que he destinado a cada campo he repetido la parte “Proveedor” de un modo que puede parecer, en principio, innecesario y rebuscado. No lo es. Cuando se desarrolla un sistema de gestión de BBDD la complejidad y el volumen de los datos a manejar pueden llegar a ser una limitación a la hora de escribir el código y/o depurarlo. Cuanto más claro esté todo, mejor. Otro punto a tener en cuenta es el tipo de dato de cada campo. En concreto, quiero aclararle algo que hace dudar a algunas personas al principio. El determinar cuándo un campo alfanumérico será de longitud fija o variable. En general, cuando el campo vaya a almacenar cadenas de cuatro caracteres o menos, lo pondremos siempre como de longitud fija (tipo CHAR). Cuando se prevea que puede almacenar cadenas de más de cuatro caracteres, y no sepamos exactamente cuántos, o la longitud de la cadena en ese campo pueda variar de un registro a otro, usaremos datos de longitud variable (tipo VARCHAR).

La tabla de productos puede tener una estructura similar a la siguiente:

- **codigoProducto.** Campo alfanumérico de tres caracteres. Será clave principal.

- **descripcionProducto.** Campo alfanumérico de longitud variable de hasta cincuenta caracteres.
- **codigoProveedorProducto.** Campo alfanumérico de tres caracteres.
- **precioCompraProducto.** Campo numérico de tipo flotante.
- **precioVentaProducto.** Campo numérico de tipo flotante.
- **stockProducto.** Campo numérico de tipo entero.

Veamos cómo crear las tablas a partir de estos requerimientos. Mire el listado del script **crearTablas.php**:

```
<?php
    define ("salto","\n<br>");
    // Se intenta conectar con el motor de MySQL
    $conectado = @mysql_connect ("localhost","root","");
    if ($conectado){
        echo ("SE HA CONECTADO AL SERVIDOR
MySQL.".salto.salto);
    } else {
        // Si la conexión falla, se avisa de ello y se
        interrumpe la ejecución.
        die ("NO SE PUDO CONECTAR AL SERVIDOR MySQL.");
    }
    /* Se selecciona la base de datos con la que vamos a
    trabajar. En un mismo servidor podemos tener más de una BBDD.
    Este paso es imprescindible.*/
    mysql_select_db ("gestion", $conectado);
    // Se forma la consulta para crear la tabla de
proveedores.
    $consulta="CREATE TABLE IF NOT EXISTS proveedores ";
    $consulta.=(codigoProveedor CHAR (3), " ;
    $consulta.="nombreProveedor VARCHAR (40), " ;
    $consulta.="direccionProveedor VARCHAR (80), " ;
    $consulta.="telefonoProveedor CHAR (9), " ;
    $consulta.="ciudadProveedor VARCHAR (40), " ;
    $consulta.="provinciaProveedor VARCHAR (20), " ;
    $consulta.="emailProveedor VARCHAR (80), " ;
    $consulta.="PRIMARY KEY (codigoProveedor)) ";
    $consulta.="TYPE=MyISAM;";
    /* Se muestra la consulta en la página. Este paso no es
    necesario para la ejecución, pero es muy útil en la fase de
    aprendizaje.*/
    echo ("<b>- CONSULTA DE CREACIÓN DE LA TABLA DE
PROVEEDORES: </b>");
    echo ($consulta.salto.salto);
    // Se ejecuta la consulta.
    $hacerConsulta=mysql_query ($consulta, $conectado);
```

```
// Se forma la consulta para crear la tabla de
productos.
$consulta="CREATE TABLE IF NOT EXISTS productos ";
$consulta.=(codigoProducto CHAR (3), ";
$consulta.="descripcionProducto VARCHAR (50), ";
$consulta.="codigoProveedorProducto CHAR (3), ";
$consulta.="precioCompraProducto FLOAT, ";
$consulta.="precioVentaProducto FLOAT, ";
$consulta.="stockProducto INT, ";
$consulta.="PRIMARY KEY (codigoProducto)) ";
$consulta.="TYPE=MyISAM;";
/* Se muestra la consulta en la página. Este paso no es
necesario para la ejecución, pero es muy útil en la fase de
aprendizaje.*/
echo ("<b>- CONSULTA DE CREACIÓN DE LA TABLA DE
PRODUCTOS: </b>");
echo ($consulta.salto.salto);
// Se ejecuta la consulta.
$hacerConsulta=mysql_query ($consulta, $conectado);
?>
```

No ejecute aún el script. Primero vamos a analizarlo por partes, para comprender cómo funciona. Quiero empezar llamando su atención sobre la línea que aparece resaltada:

```
mysql_select_db ("gestion", $conectado);
```

La función `mysql_select_db()` es la que selecciona la base de datos con la que vamos a trabajar. En un mismo servidor puede haber más de una BBDD, por lo que resulta imprescindible indicarle al script la que nos interesa. Esto es necesario incluso en el caso de que sólo haya una BBDD creada, ya que el motor de MySQL sabe que podría haber más. La selección de la BBDD se efectúa sobre una base ya existente y que va a ser utilizada, por lo que no se empleará esta función en scripts cuya única finalidad sea la creación o eliminación de bases de datos. Observe que no la pusimos en el script `crearBBDD.php`. Como norma general, se incorporará esta función al script siempre que vaya a llevarse a cabo alguna de las operaciones mencionadas a continuación:

- Trabajo con estructuras de tablas (creación, modificación o eliminación).
- Trabajo con los registros (inserción, consulta, actualización o borrado).

Esta función recibe dos argumentos. El primero es el nombre de la base de datos con la que vamos a trabajar y el segundo, el manejador de la conexión que se ha creado anteriormente.

Es un hecho que el establecimiento de la conexión y la selección de la base de datos son operaciones muy comunes en todos los scripts de un sitio dinámico. Así pues, lo habitual es crear un mini-script con estas dos funciones y luego incluirlo con include () o con require () en los scripts que se vayan a ejecutar. Un ejemplo de este mini-script podría ser el siguiente:

```
<?php
    $servidor="server";
    $usuario="user";
    $clave="password";
    $manejador=mysql_connect ($servidor, $usuario,
$clave);
    $baseDeDatos="base";
    // Se establece la conexión.
    mysql_select_db ($baseDeDatos, $manejador);
?>
```

Dado que en una base de datos podemos usar tantas tablas como sea necesario, es una práctica habitual que los scripts pertenecientes a un mismo sitio empleen una sola BBDD. De este modo, se logra mantener una claridad estructural práctica, a la hora de programar.

Continuando con el análisis del script **crearTablas.php** vemos que lo que aparece a continuación es la consulta de la creación de la tabla de proveedores y su almacenamiento en una cadena, así:

```
$consulta="CREATE TABLE IF NOT EXISTS proveedores ";
$consulta.=(codigoProveedor CHAR (3), " ;
$consulta.=nombreProveedor VARCHAR (40), " ;
$consulta.=direccionProveedor VARCHAR (80), " ;
$consulta.=telefonoProveedor CHAR (9), " ;
$consulta.=ciudadProveedor VARCHAR (40), " ;
$consulta.=provinciaProveedor VARCHAR (20), " ;
$consulta.=emailProveedor VARCHAR (80), " ;
$consulta.=PRIMARY KEY (codigoProveedor)) ";
$consulta.="TYPE=MyISAM;" ;
```

Y, una vez definida la consulta que queremos efectuar, y que sigue la sintaxis propia de SQL, la ejecutamos, así:

```
$hacerConsulta=mysql_query ($consulta, $conectado);
```

También vemos que las consultas son volcadas a la pantalla del navegador. Esto es sólo a efectos didácticos, y no es necesario para que se ejecuten. El resultado de este script aparece en la figura 17.1.

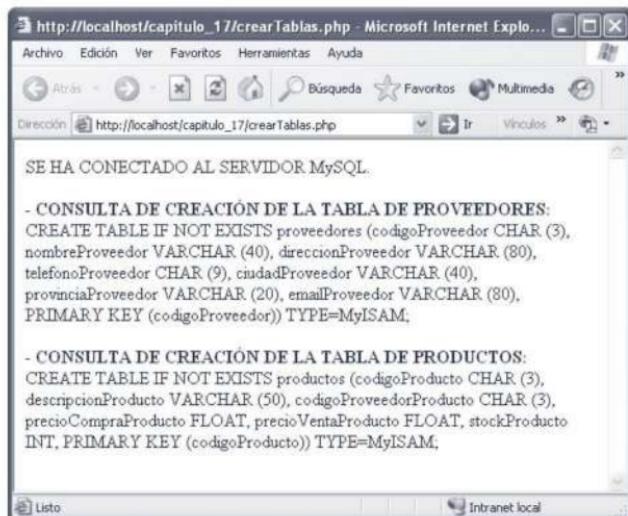


Figura 17.1

Con la ejecución de este script se habrán creado las tablas que deseamos. La creación de tablas sólo se ejecuta una vez, y ya quedan las estructuras almacenadas. Ahora vamos a ver cómo añadir registros. Para ello vamos a usar el script **insertarProveedor.php**, listado a continuación:

```
<?php
// Se incluye el mini-script que establece la conexión
con MySQL y selecciona la BBDD que necesitamos.
require("usarGestion.php");

// Se crea una consulta para insertar un registro.
$consulta="INSERT INTO proveedores (codigoProveedor,
nombreProveedor, direccionProveedor, telefonoProveedor,
ciudadProveedor, provinciaProveedor, emailProveedor) VALUES
('PR1', 'ACEROS DUROS, S.A.', 'CL Piedras Blancas, 67',
'919998877', 'Fuenlabrada', 'MADRID',
'ventas@acerosduros.org');";

// Se vuelca la consulta a la página, a fines
didácticos.
echo ($consulta."<br>");
```

```
// Se ejecuta la consulta.  
$hacerConsulta=mysql_query($consulta, $conectado);  
// Se comprueba si se ha podido grabar el registro.  
if ($hacerConsulta){  
    echo ("SE HA GRABADO EL REGISTRO.");  
} else {  
    echo ("NO SE HA PODIDO GRABAR EL REGISTRO.");  
}  
?>
```

Fíjese en que lo primero que hacemos es incluir el mini-script externo, tal como mencionábamos antes, que establece la conexión y selecciona la base de datos sobre la que trabajaremos. Se llama **usrGestion.php** y su listado es el siguiente:

```
<?php  
$conectado = mysql_connect ("localhost","root","");
mysql_select_db ("gestion", $conectado);
?>
```

Como ve, no tiene nada de especial. Continuando con el script que graba un proveedor, el siguiente paso es crear la consulta adecuada para SQL, así:

```
$consulta="INSERT INTO proveedores (codigoProveedor,
nombreProveedor, direccionProveedor, telefonoProveedor,
ciudadProveedor, provinciaProveedor, emailProveedor) VALUES
('PRI', 'ACEROS DUROS, S.A.', 'CL Piedras Blancas, 67',
'919998877', 'Fuenlabrada', 'MADRID',
'ventas@acerosduros.org');";
```

Si revisa la sintaxis que aparece en el capítulo 15 verá que es la forma correcta de hacer una consulta de inserción.

Acto seguido, ejecutamos la consulta, como ya sabemos, con la función **mysql\_query ()**, tal como ve:

```
$hacerConsulta=mysql_query($consulta, $conectado);
```

Por último, llevamos a cabo una comprobación para verificar si se ha podido ejecutar la consulta. La primera vez que cargue el script en el navegador se ejecutará sin problemas. Sin embargo, si trata de cargarlo de nuevo no se podrá ejecutar. Esto es porque el campo **codigoProveedor** es clave primaria y no se puede insertar otro registro que tenga el mismo valor en dicho campo. Si modifica, en la consulta, el valor para el código podrá insertar otro registro, aunque los demás datos sean los mismos.

Desde luego, éste no es el modo adecuado de crear consultas para SQL. Me refiero al hecho de que los valores se hayan escrito como parte del código del script. Lo normal es que procedan de variables cuyos contenidos se obtengan a partir de, por ejemplo, un formulario que rellene el usuario. En el siguiente apartado veremos un sitio completo, para la gestión de una agenda diaria, que emplea este sistema.

Una de las cosas más importantes que debemos conocer es la *captura de errores*. Durante el desarrollo de un sitio es una herramienta muy útil, para ver la causa de los posibles fallos en el tratamiento de nuestras bases de datos MySQL. Para llevar a cabo una depuración de errores, contamos con dos funciones muy útiles: la primera es *mysql\_errno()* y la segunda, *mysql\_error()*. Ninguna de ellas recibe argumentos. Lo que devuelven es información relativa al error producido durante la última consulta ejecutada, es decir, a la última vez que se ha usado *mysql\_query()*. La primera de estas dos funciones devuelve un código numérico relativo al error producido. La segunda devuelve la descripción del error. Si la consulta se ha ejecutado sin problemas, la primera función devuelve un 0 y la segunda una cadena vacía. Para ver cómo operan estas funciones mire el siguiente script, llamado **error.php**:

```
<?php
    // Se incluye el mini-script que establece la conexión
    // con MySQL y selecciona la BBDD que necesitamos.
    require("usarGestion.php");
    // Se crea una consulta para insertar un registro.
    $consulta="INSERT INTO proveedores (codigoProveedor,
    nombreProveedor, direccionProveedor, telefonoProveedor,
    ciudadProveedor, provinciaProveedor, emailProveedor) VALUES
    ();";
    // Se vuela la consulta a la página, a fines
    didácticos.
    echo ($consulta."<br>");
    // Se ejecuta la consulta.
    $hacerConsulta=mysql_query($consulta, $conectado);

    // Se recupera información sobre el error producido.
    $numeroError=mysql_errno();
    $tipoError=mysql_error();

    // Se muestra lo que tenemos acerca del error.
    echo ("El número del error es:
<b>$numeroError</b><br>");
    echo ("La descripción del error es:
<b>$tipoError</b><br>");
?>
```

Vemos que este script es parecido al que hicimos para insertar un proveedor, pero con un fallo que hemos incluido a propósito. En la consulta de inserción no hemos puesto los valores para los campos. Al ejecutarlo obtenemos el resultado de la figura 17.2.

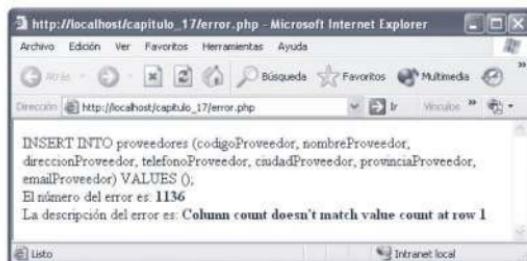


Figura 17.2

Desafortunadamente, no siempre los mensajes de error son tan claros como en este ejemplo, y no siempre el error es tan evidente pero, en cualquier caso, estas dos funciones son una inestimable ayuda a la hora de depurar scripts.

Lo siguiente que vamos a hacer es ejecutar el script **insertarProductos.php**. El listado no se ha reproducido aquí, porque es muy similar a **insertarProveedor.php**. El motivo de ejecutar este script es que tengamos varios registros en una de las tablas, para seguir avanzando en el estudio de las funciones de gestión de MySQL. Así pues, ejecute el script ahora.

Si quiere echarle un vistazo al listado verá que es muy simple y, desde el punto de vista de la programación, poco práctico y extremadamente mejorable, pero nos sirve para insertar tres registros que usaremos en algunas pruebas a continuación. Al igual que el script que insertaba un proveedor, este sólo puede ejecutarse correctamente la primera vez. Si intenta ejecutarlo de nuevo, verá que ya no se insertan los registros. Esto es porque el campo **codigoProducto** es clave primaria y no se puede duplicar su valor ni dejar en blanco.

Una vez que se haya ejecutado el script vamos a efectuar una consulta de selección, de forma que obtengamos, por ejemplo, los productos cuyo precio de costo es inferior a 10 €. En este caso son sólo dos, pero podrían ser muchos más. Es posible que nos interese fraccionar el resultado obtenido, o efectuar consultas más selectivas para obtener unos resultados más acordes con lo que necesitemos en un momento dado. En todo caso, seguramente necesitaremos saber el número de

registros obtenidos como resultado de una consulta de selección. Para ello, contamos con la función `mysql_num_rows()`. Ésta recibe, como argumento, el cursor generado al hacer una consulta y devuelve el número de registros que forman dicho cursor.

En el siguiente script, llamado **contarRegistros.php**, llevamos a cabo una consulta de selección, para contar cuántos productos cumplen la condición mencionada anteriormente. El listado es el siguiente:

```
<?php
    // Se incluye el mini-script que establece la conexión
    // con MySQL y selecciona la BBDD que necesitamos.
    require("usarGestion.php");
    // Se crea una consulta de selección.
    $consulta="SELECT * FROM productos WHERE
    precioCompraProducto<10.00;";
    // Se vuela la consulta a la página, a fines
    didácticos.
    echo ($consulta."<br>");
    // Se ejecuta la consulta.
    $hacerConsulta=mysql_query($consulta, $conectado);
    echo ("<br><hr><br>");
    // Se obtiene el número de registros que forman el
    cursor.
    $numeroDeRegistros=mysql_num_rows ($hacerConsulta);
    echo ("El número de registros recuperados es de:
    $numeroDeRegistros.");
?
>
```

Observe la línea resaltada, donde se ve el uso de la función que nos interesa y que recibe, como parámetro, el cursor obtenido de ejecutar la consulta. El resultado de este script es el que vemos en la figura 17.3.

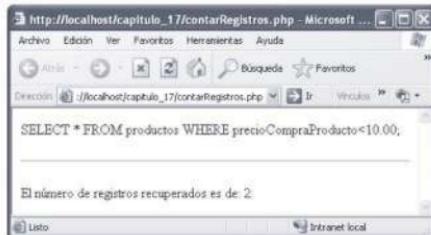


Figura 17.3

Como es lógico, cuando usted hace una consulta de selección para obtener determinados campos de los registros que se ajusten a un criterio, querrá poder acceder a cada uno de los datos recuperados de forma individual, bien para mostrarlos en la página, o para hacer cálculos con ellos, o para cualquier otra finalidad. La función que nos va a ayudar a hacer esto es *mysql\_result()*, que recibe tres argumentos. Éstos son:

- El cursor obtenido al ejecutar la consulta con *mysql\_query()*.
- El número de registro de dicho cursor que deseamos leer. Un cursor puede recuperar varios registros, y éstos quedan numerados a partir del 0.
- El nombre del campo que deseamos leer, en el registro indicado por el parámetro anterior. También se puede indicar el número del campo, teniendo en cuenta que éstos se numeran a partir de 0 y que el cursor sobre el que trabajamos no tiene por qué tener todos los campos de la tabla. Sólo tiene los que fueron especificados en la consulta.

Como siempre, veremos la mecánica operativa de esta función con un ejemplo. Vamos a crear un script que obtenga los datos de los productos con un coste inferior a 10 € y nos los ofrezca en una tabla. Lo llamaremos *matrizDeDatos.php*:

```
<html>
  <head>
    <title>Matriz De Datos</title>
    <?php
      // Se incluye el mini-script que establece la conexión
      // con MySQL y selecciona la BBDD que necesitamos.
      require("usarGestion.php");
      // Se crea una consulta de selección.
      $consulta="SELECT codigoProducto,
      descripciónProducto, precioCompraProducto FROM productos
      WHERE precioCompraProducto<10.00;";
      // Se ejecuta la consulta.
      $hacerConsulta=mysql_query($consulta, $conectado);
      // Se obtiene el número de registros que forman el
      cursor.
      $numeroDeRegistros=mysql_num_rows ($hacerConsulta);
    ?>
  </head>
  <body>
    <table border="2" cellpadding="2" cellspacing="0">
      <tr><th colspan="3">Relaci&ocute;n de productos de
      menos de 10 &euro;</th></tr>
      <tr><th>C&oacute;digo</th>
```

```

<th>Descripción</th>
<th>P. Compra</th></tr>
<?php
/* Se crea un bucle que se repetirá tantas veces como
elementos tenga el cursor obtenido anteriormente. */
for ($contador=0; $contador<$numeroDeRegistros;
$contador++){
    /* Se obtienen los datos almacenados en el elemento del
cursor al que apunta la variable de control del bucle. */
    $codigo=mysql_result ($hacerConsulta,
$contador, "codigoProducto");
    $descripcion=mysql_result ($hacerConsulta,
$contador, "descripcionProducto");
    $precioCompra=mysql_result ($hacerConsulta,
$contador, "precioCompraProducto");
    ?>
    <!-- Se crea una fila de la tabla con los datos
obtenidos -->
    <tr><td><?php echo ($codigo); ?></td>
    <td><?php echo ($descripcion); ?></td>
    <td><?php echo ($precioCompra); ?></td></tr>
    <?php
}
?>
</table>
</body>
</html>

```

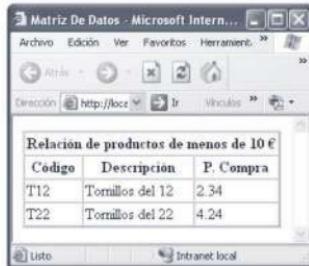


Figura 17.4

Fíjese particularmente en las líneas resaltadas. El mecanismo de este ejemplo es el siguiente: en la primera parte, se efectúa la consulta de selección y se obtiene un cursor, almacenado en la variable \$hacerConsulta. Esto sucede antes de entrar en la sección <body> de la página, pero eso no es importante. Vea que el

número de registros de dicho cursor también se memoriza. A continuación llegamos a un bucle, que se repetirá tantas veces como registros se hayan almacenado en el cursor. En mi ejemplo son dos. Dentro del bucle aparece la función que nos interesa, apuntando a cada uno de esos registros, y extrayendo un dato. Como debemos extraer tres datos de cada registro, la función aparece tres veces. Observe cómo están dispuestos los argumentos de la misma. En particular, vea el tercer argumento. Coincide con el nombre del campo recuperado, que solicitamos en la consulta de selección. El resultado corresponde a la figura 17.4.

También contamos con la función *mysql\_fetch\_array()* para la obtención de los datos recuperados con una consulta de selección. Esta función es muy empleada por su eficiencia. Dispone los datos del cursor en una matriz por cada registro y recibe dos argumentos, descritos a continuación:

- El cursor obtenido como resultado de ejecutar una consulta de selección.
- Una constante para indicar el tipo de matriz que almacenará los datos de cada registro. Si el valor es *MYSQL\_ASSOC*, la matriz de cada registro será asociativa. Cada elemento de la misma se identifica por el nombre del campo que nos interesa. Si el valor es *MYSQL\_NUM*, la matriz será indexada. Cada dato se identifica por un número, empezando desde 0. Si el valor es *MYSQL\_BOTH*, se podrá acceder por el índice o el nombre del campo, indistintamente.

Dicho así puede sonar un poco críptico, cuando uno no tiene experiencia con esta función. Como ya es habitual, veremos un ejemplo que nos aclarará la operativa de la misma. El script se llama **segundaMatrizDeDatos.php**, y su listado es el siguiente:

```
<html>
<head>
    <title>Matriz De Datos</title>
    <?php
        require("usarGestion.php");
        $consulta="SELECT codigoProducto,
descripciónProducto, precioCompraProducto FROM productos
WHERE precioCompraProducto<10.00;";
        $hacerConsulta=mysql_query($consulta, $conectado);
    ?>
</head><body>
    <table border="2" cellpadding="2" cellspacing="0">
        <tr><th colspan="3">Relaci&ocute;n de productos de
menos de 10 &euro;</th></tr><tr><th>C&ocacute;digo</th>
<th>Descripcióacute;n</th><th>P. Compra</th></tr>
```

```
<?php
    // Se recorre el cursor, mientras haya elementos,
    recuperando cada dato.
    while ($fila = mysql_fetch_array($hacerConsulta,
    MYSQL_ASSOC)) {
        echo
        ("<tr><td>".$fila["codigoProducto"]."</td>");
        echo
        ("<td>".$fila["descripcionProducto"]."</td>");
        echo
        ("<td>".$fila["precioCompraProducto"]."</td></tr>");
    }
    ?>
    </table>
</body>
</html>
```

Una vez más, preste especial atención a las líneas que parecen resaltadas. Vea cómo usamos un bucle para recorrer el cursor, mediante el uso de la función que nos ocupa. Cada registro del cursor es almacenado en una matriz de tipo asociativo, tal como se indica en el segundo parámetro de la función. Ésta es la forma más cómoda y eficiente de hacerlo, por una razón. En este caso, sólo recuperamos tres campos (vea la consulta de selección, en el bloque inicial del script). El primer campo tendría el índice 0, el segundo el 1 y el tercero el 2. Sin embargo, cuando hay más campos, y éstos se usan para cálculos u otras finalidades, acordarse del índice que les corresponde, según fueron dispuestos en la consulta, es una forma de buscarse problemas. Es mucho mejor identificar cada campo por su nombre. El resultado de este script es el mismo que el del anterior, tal como aparece en la figura 17.4.

Resumiendo. Cuando se usan bases de datos MySQL (las que usaremos en nuestros scripts siempre que sea posible), la mecánica de trabajo de cada script es siempre la misma:

- Conectamos con el motor de la base de datos y seleccionamos aquélla con la que vamos a trabajar. Estas dos operaciones suelen estar en un mini-script al que se llama cuando es necesario.
- Diseñamos la consulta que necesitemos. En este apartado hemos visto consultas de inserción y de selección, pero igualmente podemos usar consultas de actualización o de eliminación de datos. Para diseñar las consultas necesarias seguiremos la sintaxis de SQL expuesta en el capítulo 15. En los ejemplos vistos en este apartado se han usado datos fijos. Lo normal es que éstos sean introducidos por el usuario mediante

un formulario. En el apartado 17.2 veremos un ejemplo real de uso de bases de datos, que nos enseñará mucho más acerca de las mismas.

- Ejecutamos la consulta. Si ésta es de selección, los resultados se almacenan en un cursor, a partir del cual se pueden recuperar los datos de forma individual.
- Durante la depuración de un script usaremos las funciones de captura de errores que hemos estudiado en este apartado. Una vez que el script funcione correctamente, podremos eliminarlas.

Como comentario anecdótico, debo indicarle que existen dos funciones de PHP, destinadas a la gestión de BBDD de MySQL, que se han quedado obsoletas y han desaparecido. Sin embargo, es posible que, en scripts antiguos, aún las vea. En todo caso, dado que la mayoría de ellas ya no funcionan en la versión 5 de PHP, absténgase de usarlas. Éstas son:

Para la creación de una BBDD, `mysql_create_db()`.

Para la eliminación de una BBDD, `mysql_drop_db()`.

En su lugar use `mysql_query()`, con la consulta adecuada en cada caso.

Anteriormente, al hablar de establecer la conexión al motor de base de datos, le comenté que había un modo de establecer esa conexión sin que se pierda al terminar la ejecución del script: el modo de hacerlo es usando la función `mysql_pconnect()`, en lugar de `mysql_connect()`. La sintaxis y argumentos son idénticos, pero la conexión sigue abierta aunque el script termine su ejecución. Es lo que se conoce como **conexiones persistentes**. Sin embargo, este sistema no me gusta por dos razones:

- En primer lugar, si el servidor que usted contrata para el alojamiento de su sitio tiene instalado el intérprete PHP como un módulo CGI, las conexiones persistentes no funcionan. Cualquier conexión que usted abra con `mysql_pconnect()` funcionará igual con `mysql_connect()`.
- Por otro lado, cada conexión crea un proceso en el servidor. Normalmente, con conexiones que se cierran al terminar el script, el servidor libera esos procesos, para poder atender peticiones de otros clientes. Forma parte de las técnicas que se conocen, globalmente, como **balanceo de carga**, cuyo objetivo es que el servidor atienda al mayor número posible de clientes. Pero el proceso abierto con una conexión persistente no se libera al terminar el script. Si el número de clientes conectados a un servidor MySQL supera el número de procesos que dicho servidor admite, no se podrán conectar más. Aunque los que ya están conectados no necesiten el servidor de BBDD, éste les mantiene el proceso, bloqueando otras solicitudes.

Y, hablando de todo un poco: con conexiones normales (yo le recomiendo que las use siempre) el cierre de la base de datos no es exactamente automático, como le he dado a entender. En principio, la conexión se cierra al terminar el script, pero siempre es aconsejable asegurarnos de que queda cerrada, y todos los recursos usados quedan liberados. Lo primero que tenemos que tener en cuenta es que, cuando se efectúa una consulta de selección, se obtiene un cursor. Este ocupa memoria y debe ser eliminado cuando ya no sea necesario. Para ello emplearemos la función `mysql_free_result()`, cuyo argumento es el identificativo del cursor. No use esta función antes de tiempo, ya que los datos proporcionados por una consulta de selección dejan de estar disponibles en memoria una vez el cursor es destruido. Si después de usar esta función necesitará alguno de los datos que formaban parte del cursor, deberá ejecutar una nueva consulta. Esta función debe ir precedida, como norma general, del operador de supresión de avisos (@). Esto se debe a que, si el cursor está vacío, o la última consulta no ha sido de selección, se produce un aviso (WARNING), que estropea la salida en la página. Además, debemos cerrar la conexión con el servidor de BBDD al terminar el script. Para ello empleamos la función `mysql_close()`, cuyo argumento es el manejador que obtuvimos al abrir la conexión con `mysql_connect()`. Termine sus scripts con estas dos líneas, cuando use BBDD. Hasta ahora, yo no he usado estas dos funciones porque todavía no se las había mencionado y porque los scripts que hemos desarrollado hasta ahora son muy pocos y muy simples y han usado pocos recursos.

## 17.2 UN CASO PRÁCTICO

En el apartado anterior nos hemos asomado a las posibilidades de MySQL, de una forma muy rudimentaria y limitada, pero que nos ha permitido atisbar la potencia y flexibilidad de este sistema de gestión de información. En este apartado completaremos nuestros conocimientos mediante el estudio de un sitio totalmente desarrollado. Aprenderemos más sobre MySQL y las funciones de PHP destinadas a su uso. Así mismo, veremos la forma adecuada de obtener datos para nuestras consultas a partir de formularios, y cómo crear las consultas correctamente. El ejemplo que proponemos es una agenda de tipo dietario, donde anotaremos nuestras citas y, como es lógico, podremos revisarlas, editarlas o borrarlas.

Todos los scripts y archivos necesarios para esta aplicación se encuentran dentro de la carpeta **agenda** que, a su vez, está dentro de la carpeta correspondiente a este capítulo. Lo primero que haremos será cargar en el navegador el script **crearBase.php**, cuyo listado es el siguiente:

```
<?php  
// Se establece la conexión con el motor de BBDD.  
$conectado=mysql_connect ("localhost", "root", "");
```

```
// Se crea una consulta para crear la base de datos, si
esta no existe aún.
$consulta="CREATE DATABASE IF NOT EXISTS dietario;";
$hacerConsulta=mysql_query ($consulta, $conectado);

// Se selecciona la base de datos recién creada.
mysql_select_db ("dietario", $conectado);

// Se elimina la tabla, si esta existiera, para poder
crearla nueva.
$consulta="DROP TABLE IF EXISTS citas;";
$hacerConsulta=mysql_query ($consulta, $conectado);

// Se crea la estructura de la tabla.
$consulta="CREATE TABLE citas (idCita INT PRIMARY KEY
AUTO_INCREMENT, horaCita TIME, diaCita DATE, asuntoCita
VARCHAR(255));";
$hacerConsulta=mysql_query ($consulta, $conectado);

/* Se comprueba si se ha podido completar correctamente
la última operación, lo que, en este caso, implicará que
también se han llevado a cabo, sin problemas, las operaciones
anteriores. El resultado se muestra en la página. */

if ($hacerConsulta){
    echo ("La Base de datos y la tabla han sido
creadas.");
} else {
    echo ("Ha surgido algún problema durante la
creación de la BBDD y/o la tabla.<br>");
    echo ("El problema es el siguiente:<br>");
    echo ("Código: <b>".mysql_errno ()."</b><br>");
    echo ("Descripción:: <b>".mysql_error
() . "</b><br>");
}

// Se liberan recursos y se cierra la base de datos.
@mysql_free_result ($hacerConsulta);
mysql_close ($conectado);
?>
```

Si todo va bien, deberá obtener en la página una línea de texto que le informa de que la base de datos y la tabla han sido creadas correctamente. La estructura de la tabla de citas es la siguiente:

- **idCita.** Campo de tipo numérico. Almacena un identificativo para cada cita. Es de tipo clave primaria y auto incrementable, lo que significa

que su contenido para cada registro se genera de modo automático, incrementando en una unidad el contenido para el registro anterior.

- **horaCita.** Es un campo de tipo time (hora). Hasta este momento no hemos usado este tipo de campos, pero son muy útiles en ocasiones. Pueden almacenar una hora en formato hh:mm:ss.
- **diaCita.** Es un campo de tipo date (fecha). Almacenan fechas en formato yyyy-mm-dd y se usan para muchísimas aplicaciones. En cualquier aplicación medianamente profesional de la web encontrará datos de este tipo. Por ejemplo, si creamos un foro, cada usuario tendrá un registro de una tabla, en el que constará la fecha en la que se dio de alta, la fecha de su última visita, etc. Hasta ahora no habíamos manejado este tipo de datos, pero lo haremos en este ejemplo.
- **asuntoCita.** Se trata de un campo de texto, de longitud variable, donde se almacenará la cita en sí, con un máximo de 255 caracteres.

Este script lo ejecutaremos sólo una vez. Si lo ejecutamos cuando ya hayamos usado la aplicación, las citas que tengamos se perderán ya que, como puede ver en el listado del script, una de las cosas que hace es eliminar la tabla, si existe. Para ejecutarlo teclee, en la barra de direcciones de su navegador, [http://localhost/capítulo\\_17/agenda/crearBase.php](http://localhost/capitulo_17/agenda/crearBase.php) y pulse Enter. Vea que hemos incluido, en la ruta de acceso, el nombre de la carpeta agenda, donde está el script, como es lógico. El script debe ejecutarse antes de acceder a la agenda de citas en sí misma, ya que es imprescindible que exista la base de datos y tenga dentro la tabla adecuada.

Una vez que ya tenga creada la base de datos y su tabla de citas, pasamos a ejecutar la agenda. Para ello teclearemos [http://localhost/capítulo\\_17/agenda](http://localhost/capítulo_17/agenda) en la barra de direcciones. Como la página principal (dentro de la carpeta de la agenda) es **index.php**, que es el archivo que el servidor Apache busca por defecto, no es necesario que tecleemos su nombre completo. Cuando cargue la agenda lo primero que le llamará la atención es su aspecto tosco y poco elaborado. Le pido disculpas por ello, pero me ha parecido lo mejor desde el punto de vista didáctico. He querido prescindir, en la medida de lo posible, de hojas de estilo, código JavaScript (aunque algo hay) e, incluso, de algún código PHP que habría mejorado el rendimiento, como el uso de cookies o de sesiones. La razón de hacerlo así es que los árboles no nos impidan ver el bosque. He pretendido que este ejercicio se centre en el uso de la base de datos, obviando otras cuestiones. No obstante, si la agenda le parece útil (y espero que así sea), puede personalizarla a su gusto, una vez que haya comprendido su funcionamiento, que es de lo que se trata. A continuación, vamos a ver cómo funciona cada uno de los scripts que componen la aplicación, centrándonos en la forma en que se hace uso de la base de datos. No entrará en detalles sobre otros aspectos generales, como manipulación de cadenas, presentaciones en la página, etc.

### 17.2.1 La página principal

Cuando cargue la agenda por primera vez, verá una página como la que se muestra en la figura 17.5.



Figura 17.5

En la página vemos una cuenta de las citas que hay para el día en curso. Por defecto, este día coincide con la fecha del servidor (que, al ser nuestro propio equipo, puesto que estamos trabajando en modo local, coincide con la fecha de nuestro ordenador). Así se muestra en la segunda línea de texto. Si pulsamos el botón **Otro día** podremos cambiar la fecha en curso de la agenda y ver las citas correspondientes a otro día, así como grabar citas para la fecha que nos interese. Pulsando el botón **Agregar cita** podremos añadir una cita en la agenda, con la fecha en curso.

Examinemos el listado del script. Vemos que se incluyen dos mini-scripts: uno para establecer la fecha que se muestra en la agenda y otro para acceder a la base de datos. Éstos no tienen mayor misterio. Sin embargo, si nos interesa la parte que recupera las citas del día y las muestra en la página (si las hay, claro). El script se llama **index.php** y la parte del listado que nos ocupa es la siguiente:

```
/* Se crea una consulta para recuperar todos los datos
de las citas con fecha del día en curso. La consulta de
selección se crea de tal modo que ordene las citas por la
hora. */
$consulta="SELECT * FROM citas WHERE
diaCita='".$fechaEnCurso."' ORDER BY horaCita;";

/* Se ejecuta la consulta de selección.*/
$hacerConsulta=mysql_query($consulta, $conexion);
```

```
/* Se determina el número de registros recuperados por
el cursor, porque si es 0 el diseño de la página (parte HTML)
es diferente que si hay registros. */
$numeroDeCitasDelDia = mysql_num_rows
($hacerConsulta);
echo ("Citas del
día:". $numeroDeCitasDelDia . $salto);
```

En primer lugar vamos a ver cómo hacemos una consulta de selección para obtener las citas que haya para el día en curso. Observe, en la primera línea resaltada, cómo hemos hecho la consulta, estableciendo la fecha como criterio. Los campos de tipo date (fecha) y time (hora) son tratados como cadenas alfanuméricas. Fíjese en el uso de la cláusula ORDER BY. Al ejecutar esta consulta obtenemos un cursor con los registros que nos interesan, ordenados por la hora. En la segunda línea resaltada contabilizamos los registros. Por una parte, usamos esta cuenta para mostrarla en la página. Por otro lado, si el número de registros recuperados es mayor que 0 (es decir, si hay citas para el día en curso) habrá que disponer la posibilidad de que dichas citas puedan ser modificadas o eliminadas.

El cursor obtenido al ejecutar la consulta se trata como aparece a continuación:

```
/* Se comprueba si hay citas en el cursor. Si es así,
se dibujará una tabla en la que se mostrarán las citas y unos
botones de selección.*/
if ($numeroDeCitasDelDia > 0){
    echo ("<table width='500' border='0'
cellspacing='0' cellpadding='0'>");
    while ($cita = mysql_fetch_array($hacerConsulta,
MYSQL_ASSOC)) {
        echo ("<tr><td>".$cita["horaCita"]."</td>");
        echo ("<td>".$cita["asuntoCita"]."</td>");

        /* Cada cita tiene asociado un botón de selección
para si el usuario quiere modificarla o borrarla. El valor
del botón es el identificativo de la cita, de modo que se
usará en las correspondientes consultas de actualización o
eliminación en las páginas que proceda.*/
        echo ("<td><input type='radio'
id='citaSeleccionada' name='citaSeleccionada'
value='".$cita["idCita"].".'>");
        echo ("</td></tr>");
    }
    echo ("</table>");
    /* Si existen citas se mostrarán los botones de borrar
```

```

y modificar. */
echo ("<input name='borrarCita' type='button'
id='borrarCita' value='Eliminar Cita'
onClick='javascript:saltar(\"eliminarCita.php\");'>".salto);
echo ("<input name='cambiarCita' type='button'
id='cambiarCita' value='Modificar cita'
onClick='javascript:saltar(\"cambiarCita.php\");'>".salto);
}

/* En todo caso se mostrarán los botones de agregar
cita y cambiar la fecha en curso. */
echo ("<input name='nuevaCita' type='button'
id='nuevaCita' value='Agregar cita'
onClick='javascript:saltar(\"agregarCita.php\");'>".salto);
echo ("<input name='cambiarFecha' type='button'
id='cambiarFecha' value='Otro d&iacute;a'
onClick='javascript:saltar(\"cambiarFecha.php\");'>".salto);

```

Vea, en el primer bloque resaltado, cómo usamos la función `mysql_fetch_array()` para mostrar en la página la información relativa a cada cita. Además, creamos, para cada una, un botón de tipo radio. A este botón se le asigna, como valor, el identificativo de la cita (campo `idCita`) que, al ser clave primaria, es único para cada registro. De este modo, el usuario podrá seleccionar la cita que deseé para su modificación o eliminación. Observe también, en el segundo bloque que aparece resaltado, que, si el número de registros es mayor que 0, se crean los botones para estas funciones.

### 17.2.2 Agregar citas

La parte destinada a agregar citas está dividida en dos páginas. La primera es un formulario, que hemos llamado **agregarCita.php**, que recopila los datos de la nueva cita que se quiere grabar. La fecha es la que tuviéramos como fecha en curso, en la página principal. La hora y minutos de la cita se establecen a partir de unas listas desplegables. Por último, el asunto de la cita se teclea en un área de texto. En esta página no se actúa sobre la base de datos. La información incluida en el formulario es enviada, al pulsar , al script **grabarNuevaCita.php**. Éste toma las variables creadas en el formulario y las usa para formar una consulta de inserción, tal como se aprecia a continuación:

```

// Se crea la hora, a partir de las horas y minutos
establecidos en el formulario de nueva cita.
$horaDeCita=$_POST["hora"].".". $_POST["minutos"];
// Se monta la consulta para grabar una nueva cita.
$consulta="INSERT INTO citas (diaCita, horaCita,
asuntoCita) VALUES
('".$fechaEnCurso."','".$horaDeCita."','".$_POST["asunto"]."');";

```

```
// Se ejecuta la consulta.  
$hacerConsulta=mysql_query ($consulta,$conexion);
```

Como ve, al campo idCita no se le asigna un valor específico. Cuando se creó la tabla, este campo se declaró con la cláusula AUTO INCREMENT. Esto significa que el valor es asignado, de forma automática, por MySQL.

### 17.2.3 Borrar una cita

Este script, llamado **eliminarCita.php**, es el más simple de todos los que componen esta aplicación. Cuando es llamado desde la página principal, lo primero que hace es “asegurarse” de que se ha seleccionado una cita para eliminar. Esto es muy sencillo. Dado que, en la página principal, cada cita tiene asignado un botón de radio cuyo valor es, precisamente, el identificativo de la cita, comprobaremos si se ha asignado esa variable, así:

```
/* Si se intenta acceder sin haber seleccionado una  
cita, se regresa al index. */  
if (!isset($_POST["citaSeleccionada"]))  
header("Location: index.php");
```

Si se ha pasado este control, significa que hay una cita seleccionada. En ese caso, procedemos a montar y ejecutar una consulta de eliminación:

```
/* Se crea una consulta para eliminar la cita que se  
haya seleccionado en la página principal. La cita se designa  
a través del campo 'idCita', cuyo valor queda asignado a los  
botones de radio de la pagina index.php (ver código).*/  
$consulta="DELETE FROM citas WHERE  
idCita='".$_POST["citaSeleccionada"]."';;  
// Se ejecuta la consulta de eliminación.  
$hacerConsulta=mysql_query($consulta, $conexion);
```

Por último, se liberan los recursos empleados y se cierra la conexión con MySQL.

### 17.2.4 Modificar una cita

Esta función es un poco más sofisticada que la anterior, pero la filosofía es la misma. Las operaciones necesarias se dividen, al igual que para la grabación de nuevas citas, en dos páginas. La primera, llamada **cambiarCita.php** recupera la cita elegida mediante una consulta de selección y muestra sus datos en un formulario. En éste se pueden cambiar esos datos. El segundo script, llamado

**grabarCambios.php**, recopila los datos definitivos y los graba, en lugar de los que hubiera antes, mediante una consulta de actualización, así:

```
// Se toman todos los datos necesarios del formulario  
de modificaciones.  
$nuevaHora=$_POST["hora"].".". $_POST["minutos"];  
$nuevaFecha=$_POST["anno"].".". $_POST["mes"]."."  
".$_POST["dia"];  
// Se monta y ejecuta la consulta de actualización.  
$consulta="UPDATE citas SET  
diaCita='".$nuevaFecha."', horaCita='".$nuevaHora."',  
asuntoCita='".$_POST["asunto"]."' WHERE  
idCita=".$_POST["citaSeleccionada"].";  
$hacerConsulta=mysql_query($consulta, $conexion);
```

Preste atención a la línea resaltada para ver cómo el contenido de las variables se ha incorporado a la consulta.

Como ve, mediante el uso del lenguaje SQL y el motor de BBDD de MySQL, la gestión de bases de datos no encierra dificultad alguna. A lo largo del resto de este libro verá más ejemplos interesantes que le ayudarán a diseñar sus propios scripts.

## 17.3 HERRAMIENTA DE GESTIÓN VISUAL

A menudo, cuando se trabaja con bases de datos MySQL, uno desearía disponer de alguna interfaz gráfica para la gestión de las mismas. En el mercado existen varias herramientas de este tipo. Algunas son de pago y otras gratuitas. La más conocida, sin duda, es **phpMyAdmin** que es de libre uso y distribución y está escrita, íntegramente, en PHP.

Esta herramienta está incorporada en la aplicación AppServ, que usamos en el capítulo 2 para montar el servidor. Sin embargo, nosotros usaremos la última versión disponible, que es más estable. En el CD adjunto está grabada en la carpeta correspondiente a este capítulo. Para ejecutarla, teclee, en la barra de direcciones de su navegador, [http://localhost/capítulo\\_17/gestion/phpMyAdmin](http://localhost/capitulo_17/gestion/phpMyAdmin) y pulse Enter. La herramienta phpMyAdmin (cuya última versión está disponible en <http://www.phpmyadmin.net>) le permite realizar todo tipo de operaciones con bases de datos MySQL, tales como:

- Ver la estructura de las tablas.
- Ver el contenido de determinados campos o registros (o de todos ellos, llegado el caso).

- Diseñar y ejecutar cualquier consulta SQL, viendo su resultado al momento.
- Generar, de modo automático, una consulta SQL para reproducir la tabla o tablas que le interesen. De este modo, si pierde estos datos, ejecuta la consulta SQL obtenida y recupera la información.
- Gestionar los permisos de acceso al motor MySQL.

Además de estas operaciones principales, phpMyAdmin le permite realizar muchas otras, así como obtener información importante acerca de su instalación. Al iniciarla la aplicación su aspecto es el de la figura 17.6.

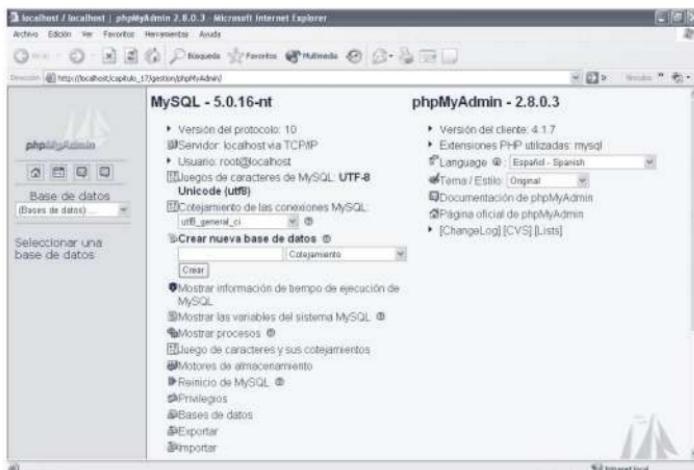


Figura 17.6

Fíjese en el marco izquierdo, donde aparece una lista desplegable. Ésta contiene la relación de todas las bases de datos MySQL que tenga usted en su plataforma servidora (incluyendo, naturalmente, la base de datos de citas que se creó en el ejercicio del apartado anterior). Debajo de la lista nos indica que debemos seleccionar una BBDD con la que trabajaremos. Elegiremos, precisamente, la base “dietario” que usamos para la agenda. Al seleccionar una base de datos, en el marco izquierdo aparece el nombre de la misma, con un número entre paréntesis. Este número (en nuestro caso, el 1) se refiere a la cantidad de tablas que contiene la BBDD elegida. Debajo, aparece el nombre de la tabla. Si hubiera más tablas, aparecerían listadas todas ellas, una debajo de otra.

El marco derecho también ha cambiado. Toda la interfaz gráfica de phpMyAdmin está ahora orientada a trabajar con la BBDD seleccionada, aunque, en el marco izquierdo, se mantiene la lista desplegable, por si queremos seleccionar otra base diferente. Su aspecto es similar al de la figura 17.7.

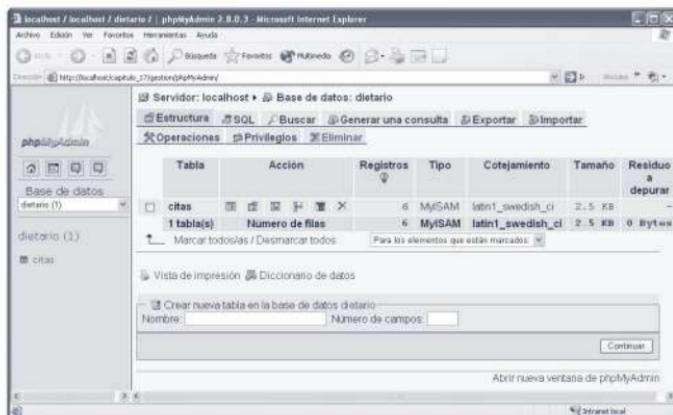


Figura 17.7

Observe la parte central del marco derecho. En una fila, aparece el nombre de la tabla, junto a unos botones. Si hubiera más tablas, aparecerían debajo de ésta. Los botones (colocados bajo el encabezamiento **Acción**) permiten llevar a cabo distintas acciones sobre la tabla. Al arrimar el puntero a cada botón aparece una etiqueta flotante indicando la acción que podemos llevar a cabo. Éstas son:

**Examinar.** Muestra una relación con los registros de la tabla. Si son muchos, los agrupa de 30 en 30.

**Estructura.** Permite ver la estructura de la tabla, con todos los tipos de datos, nombres de campos, tamaños, etc.

**Buscar.** Permite establecer criterios de búsqueda para llevar a cabo una consulta de selección, mostrando los registros que cumplen los criterios especificados. Si son muchos, fracciona el resultado de 30 en 30.

**Insertar.** Permite añadir un registro a la tabla.

**Vaciar.** Permite eliminar el contenido de la tabla, manteniendo la estructura. Si pulsa este botón, antes de eliminar los registros se le pedirá confirmación, ya que esta operación no es reversible.

**Eliminar.** Permite eliminar toda la tabla (datos y estructura). Antes de proceder, se le pedirá confirmación, ya que la operación no es reversible.

Además, tenemos, para gestionar la base de datos, los enlaces que aparecen en la parte superior del marco derecho, reproducidos con detalle en la figura 17.8.



Figura 17.8

Los enlaces **Estructura** y **Eliminar** se corresponden, en su función, con los botones homónimos de los que acabamos de ver. Uno de los enlaces más interesantes de este apartado es **Exportar**. Al pulsarlo obtenemos, en el marco derecho, la imagen de la figura 17.9 (el marco izquierdo se mantiene intacto, aunque no aparezca en esta reproducción).

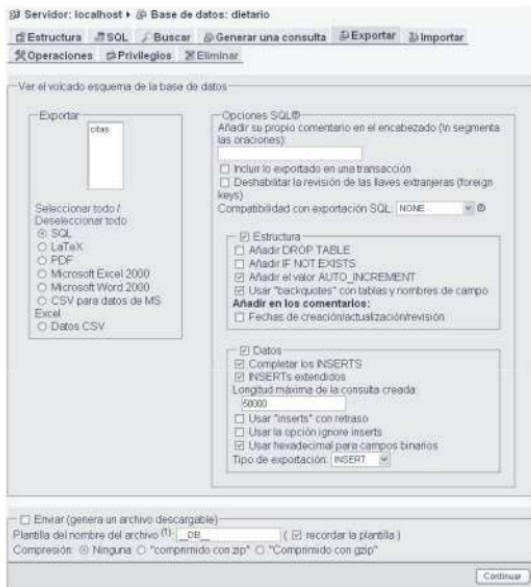


Figura 17.9

En el cuadro de la izquierda seleccionaremos la tabla o tablas que queremos exportar. En este ejemplo sólo tenemos una. La seleccionamos con un clic de ratón, de modo que su nombre quede resaltado (). En las opciones que hay debajo del nombre de la tabla, nos aseguraremos de que esté activada la que corresponde a SQL, tal como aparece en la figura 17.9. Las demás opciones las dejaremos como aparecen por defecto, excepto en el apartado **Estructura**, donde marcaremos las casillas correspondientes a **Añadir DROP TABLE** y **Añadir IF NOT EXISTS**. El aspecto de este apartado deberá coincidir con el de la figura 17.10.



Figura 17.10

Con esto, tenemos listo el servicio de exportación de la BBDD. Pulsaremos el botón , situado en la parte inferior derecha, y nos encontraremos con el resultado de la exportación. Ésta consiste en la generación de un fichero de texto (que se puede seleccionar, copiar al portapapeles y guardar desde cualquier editor de texto plano) con las consultas necesarias para recrear las tablas que hayamos seleccionado, en su estado actual. Si las tablas se corrompen, o sufren algún daño, a partir de las consultas generadas podremos recuperarlas.

Otra de las opciones interesantes de las pestañas de la parte superior (ver figura 17.8) es , que es contraria a la que acabamos de ver. A partir de un archivo de texto, con las consultas adecuadas, genera las tablas necesarias. De este modo recuperaremos una o más tablas que se hayan perdido. Lo primero que debemos hacer es seleccionar el archivo donde tenemos las consultas SQL adecuadas, en el apartado de la figura 17.11.



Figura 17.11

A continuación pulsaremos el botón , situado en la parte inferior derecha, y la importación se llevará a cabo. También le echaremos un vistazo, en la

parte superior, al enlace SQL. Al pulsarlo vemos, en el marco de la derecha, la pantalla de la figura 17.12.



Figura 17.12

En la caja de texto podemos escribir libremente una consulta SQL (desde luego, la palabra “libremente” no significa que no debamos respetar la sintaxis correcta del lenguaje). A continuación, pulsaremos el botón **Continuar** y la consulta se ejecutará, mostrándonos los resultados. Si hubiera un error de sintaxis en la consulta, veremos un mensaje avisándonos de ello.

Si no estamos muy duchos en la sintaxis de SQL y tenemos dudas acerca de cómo escribir una consulta determinada, podemos hacer dos cosas. La primera sería echarle un vistazo al capítulo 15 del libro. La otra alternativa es pulsar el enlace Generar una consulta. Éste nos da paso a una pantalla donde podemos establecer el tipo de consulta, los criterios, los campos que nos interesan, etc. A continuación, pulsando en el botón **Modificar la consulta** obtendremos la sintaxis correcta de la consulta en la ventana que hay al efecto en la parte inferior de esta página.

La herramienta phpMyAdmin es mucho más útil de lo que parece en un principio y, una vez se familiarice con ella, le será difícil prescindir de su uso. Yo aquí le he mostrado las opciones más interesantes y útiles que tiene con el objetivo de darle a conocer esta estupenda aplicación. Cuando visite la página oficial de esta herramienta, en la página <http://www.phpmyadmin.net>, hallará documentación más detallada, aunque con lo que hemos visto en estas pocas páginas tendrá más que suficiente para su día a día. No obstante, recuerde la frase de cierto pensador: “La curiosidad es una virtud. El conformismo es un vicio del espíritu. Sólo los necios lo ven al revés.”.



## CAPÍTULO 18

# SCRIPTS ÚTILES

---

---

Este capítulo contiene una serie de scripts cuyo conocimiento le resultará especialmente útil. Por una parte, puede ponerlos, tal como están o con las modificaciones que considere oportunas, en sus propias páginas. Todo el material que aparece en este capítulo es de uso común y libre de derechos. Por otra parte, estos scripts ofrecen un importante valor didáctico. En ellos usaremos funciones y estructuras que se han visto a lo largo del libro. Además, aprenderemos algunos conceptos adicionales, o complementarios, que nos mostrarán cómo sacarle partido a PHP de forma práctica.

### 18.1 EL ORIGEN DE UNA VISITA

A menudo nuestras páginas son visitadas por internautas de distintos lugares del mundo. Después de todo, esa es la razón de ser de Internet: que los contenidos que haya en la Red estén a disposición de cualquiera que se conecte a nuestro sitio. Una idea muy interesante es determinar cuál es el país desde el que se ha establecido la conexión. Esto nos permitirá varios niveles de personalización del sitio, según desde donde se conecte un usuario. Los más significativos son:

- Redireccionar al usuario a una página con contenidos que le puedan interesar en función de su entorno sociocultural local.
- Mostrar los textos de las páginas en el idioma del país desde el que se conecta el usuario.
- Si nuestro sitio es de comercio electrónico, podremos mostrar los precios de los artículos y servicios que ofrecemos, así como la factura final, en la moneda local del usuario.

Para hacer esto necesitamos dos cosas: la IP desde la que se conecta el usuario y una forma de relacionar esa IP con un país concreto. Lo primero ya sabemos cómo obtenerlo (vea el capítulo 14), aunque, en este ejercicio, ampliaremos el tema. Para lo segundo necesitamos una relación de IP asociadas a países. Dado que Internet es una Red de ámbito mundial y que cada IP es única (no puede haber dos conexiones simultáneas con la misma IP), éstas se hallan asignadas, por cuestiones de unificación, a los distintos países. Así, cada país tiene unos rangos de IP. Por lo tanto, aunque dentro de un país los usuarios cambien su IP (el caso de las IP dinámicas) obtendrán otra dirección dentro de las que tiene asignadas su país. Existe una tabla, con la información necesaria, que se actualiza con bastante frecuencia. Se conoce como **IP to Country** (De la IP al país). Nos la podemos descargar desde <http://ip-to-country.directi.com/>. Se trata de un archivo comprimido. Si lo abrimos, encontraremos un archivo en formato **CSV (Comma Separated Values)**, Valores separados por comas). El formato CSV es muy empleado para el almacenamiento de tablas, ya que resulta muy cómodo de manejar con PHP (y con otros lenguajes), como veremos en seguida. La tabla que nos interesa está en la carpeta country, dentro de la que corresponde a este capítulo. De todos modos, no deje de visitar la dirección que aparece más arriba, si desea obtener la misma tabla con las últimas actualizaciones. En la figura 18.1 tiene un fragmento de la tabla, tal como se la obtiene al descomprimir el archivo que se descarga de Internet. Como ve, es texto plano, con los datos encerrados entre comillas y separados por comas.

1	2	3	4	5
1	"33996344","33996351","GB","GBR","UNITED KINGDOM"			
2	"50331648","69956103","US","USA","UNITED STATES"			
3	"69956104","69956111","EM","BHU","BERMUDA"			
4	"69956112","83886079","US","USA","UNITED STATES"			
5	"94585424","94585439","SE","SWE","SWEDEN"			
6	"100663296","121195295","US","USA","UNITED STATES"			
7	"121195296","121195327","IT","ITA","ITALY"			
8	"121195328","152305663","US","USA","UNITED STATES"			
9	"152305664","152338431","GB","GBR","UNITED KINGDOM"			
10	"152338432","167772159","US","USA","UNITED STATES"			
11	"184549376","201674095","US","USA","UNITED STATES"			
12	"201674096","201674111","CA","CAN","CANADA"			
13	"201674112","202035199","US","USA","UNITED STATES"			
14	"202035200","202035711","IN","IND","INDIA"			
15	"202035712","205500987","US","USA","UNITED STATES"			
16	"205500988","205500991","CA","CAN","CANADA"			
17	"205500992","208605279","US","USA","UNITED STATES"			

Figura 18.1

Cada línea es un rango de IP, con los datos del país al que corresponde. Lo primero que nos llama la atención es que las direcciones IP no están en el formato típico del protocolo IPv4, formado por cuatro valores entre 0 y 255, separados por

puntos. En este fichero, las direcciones IP aparecen codificadas en forma de cadenas que representan valores numéricos de tipo long. De momento acepte esto. En seguida volveremos sobre ese tema.

La tabla contiene también una abreviatura del nombre del país con dos letras, otra con tres letras (para su uso en distintas aplicaciones) y el nombre del país, en inglés.

Esta tabla es muy legible para nosotros, tal como está, en texto plano. Y también es útil para ser manejada por aplicaciones del tipo hoja de cálculo. Sin embargo, crear un script que busque una IP determinada en esta tabla es tedioso, y su ejecución puede ser lenta. Tenga en cuenta que la tabla tiene más de 65.000 líneas. Así, pues, lo primero que se impone es convertir esta tabla al formato de base de datos MySQL para manejarla de un modo más eficiente. Para ello contamos con el script **crearBBDD.php**, listado a continuación:

```
<?php
    // Se establece conexión con el motor de base de datos,
y se verifica.
    $conexion=@mysql_connect ("localhost","root","");
    if (!$conexion) die ("NO SE PUDO CONECTAR CON EL
MOTOR DE BBDD.");
    // Se borra la base de datos si existía previamente.
    $consulta ="DROP DATABASE IF EXISTS iptocountry";
    $hacerConsulta=@mysql_query ($consulta, $conexion);
    // Se crea la base de datos.
    $consulta="CREATE DATABASE iptocountry;";
    $hacerConsulta=@mysql_query($consulta, $conexion);
    // Se selecciona la base de datos creada.
    @mysql_select_db("iptocountry",$conexion);
    // Se crea la tabla de los rangos de IP, y se verifica.
    $consulta="CREATE TABLE ipCountry (ipStart DOUBLE NOT
NULL, ipEnd DOUBLE NOT NULL, countryCode2 CHAR(2) NOT NULL
default '', countryCode3 CHAR(3) NOT NULL default '',
countryName VARCHAR(50) NOT NULL default '', PRIMARY KEY
(ipStart, ipEnd) TYPE=MyISAM;";
    $hacerConsulta=@mysql_query($consulta, $conexion);
    if (!$hacerConsulta) die ("LA TABLA NO PUDO SER
CREADA.");
    // Se abre el fichero csv de los rangos de IP's.
    $manejadorFichero=fopen("ip-to-country.csv","r");
    // Se define una matriz para almacenar los rangos.
    $matriz=array();
    /* Mientras haya datos, se lee cada línea y se almacena
en un elemento de $matriz. Cada elemento es, a su vez, una
matriz con cinco elementos. */
```

```

        while (!feof($manejadorFichero)){
            $matriz []=fgetcsv($manejadorFichero);
        }
        /* Se recorre la matriz y se desglosa cada elemento,
para insertar una nueva fila de la tabla. */
        foreach ($matriz as $rangoIPs){
            /* Se comprueba que el elemento tenga un contenido
real.*/
            if ($rangoIPs[0]!=""){
                $consulta="INSERT INTO ipCountry (ipStart, ipEnd,
countryCode2, countryCode3, countryName) VALUES
(\"$rangoIPs[0]\", \"$rangoIPs[1]\", \"$rangoIPs[2]\",
\"$rangoIPs[3]\", \"$rangoIPs[4]\");";
                $hacerConsulta=mysql_query($consulta, $conexion);
            }
        }
        /* Se cierra el fichero CSV y la BBDD.*/
        fclose ($manejadorFichero);
        mysql_close($conexion);
        /* Se muestra el resultado.*/
        echo ("La BBDD y la tabla han sido creadas.<br>");
    ?>

```

Vamos a analizarlo, ya que veremos conceptos que ya conocemos, y aprenderemos otros nuevos.

En primer lugar, se establece conexión con el motor de MySQL. Si existe la base de datos, se elimina, para poder crearla de nuevo. Una vez creada la BBDD, se genera la consulta para crear la tabla que usaremos en este ejercicio. Observe dicha consulta detenidamente:

```

$consulta="CREATE TABLE ipCountry (ipStart DOUBLE NOT
NULL, ipEnd DOUBLE NOT NULL, countryCode2 CHAR(2) NOT NULL
default '', countryCode3 CHAR(3) NOT NULL default '',
countryName VARCHAR(50) NOT NULL default '', PRIMARY KEY
(ipStart, ipEnd)) TYPE=MyISAM;";

```

En la tabla cada registro corresponderá a un rango de IP, con los datos del país asociado. Los campos que tiene la tabla son los siguientes:

- ipStart
- ipEnd
- countryCode2
- countryCode3
- countryName

Los dos primeros son de tipo double y almacenarán los valores numéricos correspondientes al principio y al final del rango de IP. El tercer campo almacenará la abreviatura del nombre del país con dos letras. El cuarto campo almacenará la abreviatura con tres letras y el quinto y último campo almacenará el nombre del país.

Lo que debe llamarnos la atención es la parte resaltada. Fíjese en que, como clave primaria, se han establecido dos campos (los dos que delimitan el rango de IP), en lugar de uno solo. Esto es perfectamente válido, cuando se trabaja con BBDD de MySQL. En este caso nos facilitará la localización de un rango determinado por los dos campos, acelerando la ejecución del script.

Una vez creada la tabla, vamos a pasar todos los datos del fichero CSV a la misma. Para ello, abrimos el fichero en modo de lectura. Para leer el contenido usamos las siguientes líneas:

```
// Se define una matriz para almacenar los rangos.  
$matriz=array();  
/* Mientras haya datos, se lee cada línea y se almacena  
en un elemento de $matriz. Cada elemento es, a su vez, una  
matriz con cinco elementos. */  
while (!feof($manejadorFichero)){  
    $matriz[] = fgetcsv($manejadorFichero);  
}
```

Fíjese en que hemos creado una matriz. En cada elemento de la misma pretendemos almacenar todos los datos de una línea del fichero, es decir, de uno de los rangos de IP. Para ello, leemos el fichero en un bucle que se estará ejecutando mientras no se encuentre el final (vea, en el capítulo 8, la función `feof()`, que detecta el final de un fichero). Los datos de cada línea generan un nuevo elemento de la matriz. Y aquí aparece una función nueva: `fgetcsv()`, tal como ve en la línea resaltada. Esta función ha sido implementada en PHP específicamente para la lectura de ficheros CSV. Como argumento, recibe el manejador con el que ha sido abierto el fichero. El resultado es la lectura de una línea completa, hasta el salto de línea. A través de las comas, identifica cuál es cada dato, cuyo valor debe aparecer entrecomillado, tal como se ve en la figura 18.1. Con todos los datos, genera una matriz indexada, con un elemento por dato. En este caso, la matriz tendrá cinco elementos. Supongamos que aplicamos la función para leer la primera línea, exclusivamente. La matriz tendrá los siguientes elementos:

```
Array (  
    [0] "33996344"  
    [1] "33996351"
```

```
[2] "GB"  
[3] "GRB"  
[4] "UNITED KINGDOM"  
)
```

En el caso del bucle de nuestro código, esta matriz se almacena en un elemento de la matriz que hemos llamado **\$matriz**. Esto significa que **\$matriz**, al final del bucle, tendrá más de 65.000 elementos (uno por cada línea del fichero CSV) y cada uno será una pequeña matriz con cinco elementos. Así pues, **\$matriz** es, realmente, una matriz de dos dimensiones. Si no recuerda bien cómo operan las matrices, revise el capítulo 3. Observe, en la linea resaltada que a **\$matriz** no se le ha asignado un índice específico, así que, en cada iteración del bucle, se genera un nuevo elemento.

Tras generar **\$matriz**, se usa un bucle `foreach` para recorrerla entera. Cada elemento, que ya sabemos que es, a su vez, una pequeña matriz, contiene cinco datos. Éstos se usan para generar un nuevo registro en la tabla MySQL. Vea que, dentro del bucle `foreach` se comprueba si la IP inicial (elemento [0] de la matriz) tiene contenido. Esto se hace porque el fichero CSV puede contener líneas en blanco al final del mismo, y éstas también generan elementos en la matriz principal, pero no deben almacenarse en la tabla MySQL.

Una vez generada la tabla, cerramos la BBDD y el fichero CSV.

Este script tarda algunos segundos en ejecutarse completamente, dado el gran volumen de datos a tratar. Afortunadamente, sólo debe ejecutarse una vez. La solución alcanzada para pasar los datos del fichero CSV a MySQL no es la óptima, en cuanto a velocidad de ejecución. Sin embargo, este script, a pesar de su aparente torpeza, es fiable y robusto, y no produce errores.

El siguiente paso es diseñar una función que capture la IP de la conexión del cliente. Observe el listado del script **obtenerIP.php**, que aparece a continuación:

```
<?  
function obtenerIPCliente() {  
/* En la variable $ip se almacenará la dirección del  
cliente. */  
    $ip = 0;  
/* Si la variable $_SERVER['HTTP_CLIENT_IP'] tiene  
algún contenido, se asigna a la variable $ip. */  
    if (!empty($_SERVER['HTTP_CLIENT_IP'])) $ip =  
$_SERVER['HTTP_CLIENT_IP'];
```

```
/* Si la variable $_SERVER['HTTP_X_FORWARDED_FOR']
tiene algún contenido, se asigna a la variable $ip. En ese
caso debe separarse la IP pública del cliente de las
obtenidas a través de posibles redes locales. */
if (!empty($_SERVER['HTTP_X_FORWARDED_FOR'])) {
    /* Se abre la matriz de IP's obtenidas de
    $_SERVER['HTTP_X_FORWARDED_FOR']. */
    $ListaDeip = explode (",",
    $_SERVER['HTTP_X_FORWARDED_FOR']);
    /* Si ya hay contenido en $ip, se añade a la matriz. */
    if ($ip) {
        array_unshift($ListaDeip, $ip);
        $ip = 0;
    }
    /* Se eliminan IP's privadas, procedentes de posibles
    redes locales, así como la dirección de bucle local. Cuando
    se encuentra una IP pública (externa) se devuelve como
    resultado de la función. */
    foreach ($ListaDeip as $direccion) if
    (!eregi("(^192\.168|172\.16|10|224|240|127|0)\.", $direccion)) return $direccion;
}
/* Si no había contenido en
$_SERVER['HTTP_X_FORWARDED_FOR'] se devuelve la IP obtenida
mediante $_SERVER['REMOTE_ADDR']. */
return $ip ? $ip : $_SERVER['REMOTE_ADDR'];
}
?>
```

Como ve, el script es bastante sencillo. No obstante, vamos a comentar algunos detalles. En primer lugar, se intenta recuperar la IP del cliente mediante la variable de servidor `$_SERVER['HTTP_CLIENT_IP']`. Como sabe, es una de las que se emplean para esta finalidad. Sin embargo, es posible que, por la propia naturaleza de la conexión, esta variable no contenga la información que buscamos. Por esta razón se revisa el contenido de otra importante variable de servidor: `$_SERVER['HTTP_X_FORWARDED_FOR']`. Suponga que el cliente tiene un ordenador que forma parte de una red local. Esta variable contiene la IP pública y las IP privadas que recorre la conexión en la red local del cliente hasta que sale al exterior. Además, adicionalmente, puede tener otras IP, de proxys que haya entre el cliente y el servidor. Esta variable de servidor contiene una cadena con todas las IP, públicas y privadas, que intervienen en la conexión. La primera IP pública que aparece es la del cliente. Estas IP están separadas unas de otras mediante una coma y un espacio en blanco. Así pues, podemos usar la función `explode()` para crear una matriz en la que cada elemento contendrá una dirección IP, tal como se ve en la siguiente línea:

```
$ListaDeip = explode (" ",  
$_SERVER['HTTP_X_FORWARDED_FOR']);
```

Como puede apreciar, la función `explode ()` recibe dos argumentos. El primero es el separador que delimita cada uno de los datos que deberán constituir elementos independientes en la matriz. El segundo es la cadena que contiene todos los datos. La función devuelve la matriz que necesitamos.

Si la variable `$ip` tenía ya algún valor previamente, obtenido mediante `$_SERVER['HTTP_CLIENT_IP']`, se añade como primer elemento de la matriz, mediante la función `array_unshift ()`. Ésta recibe dos argumentos. El primero es el nombre de una matriz y el segundo es un dato que queremos incorporar como un elemento más de dicha matriz. La función coloca ese dato al principio de la matriz, desplazando los demás elementos. Opcionalmente, esta función puede recibir más datos, separados por comas, que irá colocando en la matriz.

Por último, usamos expresiones regulares para descartar las direcciones IP privadas, que puedan formar parte de la red local del cliente. Para analizar cómo funciona la expresión regular que hemos empleado aquí, observe, en la figura 1.1 del capítulo 1 del libro los rangos de IP privadas que pueden ser usadas en redes locales. La expresión regular que hemos creado elimina también la dirección de localhost (127.0.0.1). Por esta razón, este script no puede ser probado en su máquina local, como ha hecho con todos los del libro, sino que debe ser ejecutado desde un servidor remoto, para poder tener un IP pública real.

La función termina cuando se comprueba si ya se ha obtenido la IP. Si no es así, se recurre a la variable de servidor `$_SERVER ['REMOTE_ADDR']`. En un caso u otro, la función devuelve la IP pública de la conexión del cliente.

Y ya tenemos todo lo que necesitamos. Ahora nos hace falta un script que lea la IP pública (usando la función que acabamos de ver), la convierta en un número long y busque, en la BBDD de MySQL a qué país corresponde. El listado `determinarPais.php` se ocupa de todo ello:

```
<?php  
/* Se incluyen el mini-script que abre la BBDD. */  
include ("abrirDB.php");  
/* Se incluye el archivo que contiene la función para  
obtener la IP del cliente. */  
include ("obtenerIP.php");  
/* Se obtiene la IP del cliente, en el formato típico:  
"xxx.xxx.xxx.xxx". */  
$direccionReal=obtenerIPCliente();
```

```
$consulta="SELECT countryCode2, countryName FROM
ipcountry WHERE ipStart<=$direccionNumerica AND
ipEnd>=$direccionNumerica;";
```

Observe que, en este ejercicio, sólo vamos a recuperar dos datos del registro: la abreviatura del país con dos letras, y el nombre completo del país. Para ello, una vez ejecutada la consulta, extraemos dichos datos del cursor resultante, como se muestra en el siguiente fragmento de código:

```
/* Se recuperan los datos de la BBDD. */
$datosDePais=mysql_fetch_array ($hacerConsulta,
MYSQL_ASSOC);

$codigoDePais=strtolower($datosDePais["countryCode2"]);
$nombreDePais=$datosDePais["countryName"];
```

Con el código del país formaremos una cadena con el nombre de una imagen. En la carpeta imágenes, dentro de la carpeta country, tenemos todas las banderas de los países que podamos necesitar, nombradas con estos códigos. Después, mostramos la IP, el nombre del país y su bandera, tal como se aprecia en la figura 18.2.



Figura 18.2

Yo he hecho las pruebas subiendo los scripts a un servidor y conectándome desde un ordenador situado en España, tal como puede apreciarse. Si usted hace sus pruebas desde otro país, verá el nombre y la bandera del mismo en su navegador.

Y con esto damos por terminado el ejercicio. Ya se ha logrado determinar desde qué país se conecta el usuario, que es de lo que se trataba. A partir de aquí, deberemos usar esa información para personalizar el sitio acorde con cada idioma, nacionalidad, cultura, moneda local, etc., según el contenido de nuestras páginas.

## 18.2 EVITANDO LOS BOTS

Suponga que usted monta un sitio que requiera que los usuarios se registren para poder acceder a determinados servicios. Los datos de los usuarios, como puedan ser el login o la clave de acceso, quedan almacenados en una BBDD. Como cualquier webmaster que se precie, usted querrá tener el máximo de usuarios posible. Pero querrá que sean usuarios reales. ¿Qué quiere decir "usuarios reales"? Existen muchas aplicaciones, conocidas, genéricamente, como *bots*, que se conectan a sitios de Internet y generan, de forma automática, solicitudes de ingreso, logins, contraseñas y todos los datos necesarios. Cuando uno de estos bots localiza una página web destinada a usuarios registrados empieza a generar, automáticamente, gran cantidad de usuarios "fantasma". Éstos no existen en la realidad, pero están "abultando" en la BBDD, saturando el servidor, y dificultando que se preste un servicio eficiente a usuarios legítimos. Estos bots son, por lo tanto, herramientas maliciosas, dañinas, cuyo ataque hay que eludir. Son varias las soluciones que se han creado para que un sitio web detecte los intentos de registro de un bot y lo evite. En este apartado vamos a conocer una de las más simples y eficaces. Tanto es así que sitios con tanto volumen de usuarios como Hotmail (<http://www.hotmail.com>) emplean una solución como la que estudiaremos aquí.

El objetivo es generar una cadena aleatoria de caracteres. Una vez creada, montaremos una imagen que contenga dicha cadena, sobre un fondo determinado. En la imagen, con total comodidad por parte del usuario, se leen los caracteres. Sólo que ya no son caracteres, sino parte de un gráfico. Al usuario se le pide que copie la cadena en una casilla, para poder seguir adelante con el registro. Sólo un usuario humano puede hacer esto, que parece tan simple. De esta forma, ningún bot puede registrarse en el sitio.

La herramienta que vamos a desarrollar aquí, y que luego usted podrá usar en sus sitios web, es muy simple. Consta de cuatro scripts, cuya función se resume a continuación:

- El script **index.php**. Es la página principal del sitio. Se encarga de generar la cadena de texto aleatoria que luego usaremos en el gráfico. Esta cadena pasará de un script a otro mediante el uso de una sesión.
- El script **crearImagen.php**. Este recibe, por medio de una sesión, la cadena aleatoria y la monta en una imagen de fondo.
- El script **formulario.php**. Es donde se registra el usuario, y donde debe copiar, en una casilla al efecto, el texto que aparece en la imagen.
- El script **comprobarResultado.php**. Es al que va a parar el formulario y donde se comprueba si la cadena tecleada por el usuario coincide con la que se generó aleatoriamente en el primer script y que aparecía en la imagen creada al efecto.

En primer lugar, vamos a ver el script **index.php**, que genera la cadena. Su listado es muy simple, como se puede comprobar:

```
<?php
    /* Se inicia una nueva sesión y se registra la variable
    que pasará la cadena a otras páginas. */
    session_start ();
    session_register ("cadena");
    /* Se establece la longitud de la cadena en un valor
    aleatorio entre cinco y diez caracteres. */
    $longitud=rand(5,10);
    /* La cadena se declara sin contenido. */
    $cadena="";
    /* Un bucle generará la cadena, carácter a carácter. Se
    repite tantas veces como caracteres vaya a tener la cadena.
    */
    for ($contador=1; $contador<=$longitud; $contador++){
        /* Se genera un número aleatorio correspondiente a los
        códigos ASCII de las mayúsculas. */
        $caracter=rand(97,122);
        /* Se añade el carácter a la cadena. */
        $cadena.=chr($caracter);
    }

    /* La cadena aleatoria generada se asigna a la variable
    de sesión para que pase a otras páginas. */
    $_SESSION["cadena"]=$cadena;
?
<html> <head>
    <script language="javascript"
type="text/javascript">
        function irAlFormulario(){
            location.href="formulario.php?"+<?php
echo(SID); ?>";
        }
    </script>
</head>
<body onLoad="javascript:irAlFormulario();">
</body>
</html>
```

Se trata de generar una cadena que tenga un número aleatorio de caracteres, entre cinco y diez. El haber elegido estos límites es arbitrario. Usted puede poner los límites que desee. Sin embargo, una cadena de esta longitud está bien para este tipo de comprobaciones. Fíjese en que empezamos inaugurando una sesión y registrando una variable que luego contendrá la cadena, así:

```
session_start();
session_register ("cadena");
```

A continuación, determinaremos cuántos caracteres tendrá la cadena, mediante el uso de la función rand (), que ya conoce, así:

```
$longitud=rand(5,10);
```

Inicialmente, la cadena se declara como vacía. A continuación, hay un bucle que ejecuta tantas iteraciones como hayamos obtenido para la longitud de la cadena. En cada iteración, se le añade un carácter. Éstos se obtienen mediante el código ASCII de las letras minúsculas (valores del 97 al 122). Podríamos haber ampliado esto a las mayúsculas, pero, para este ejercicio, no lo he estimado necesario. En cada iteración del bucle se genera un valor aleatorio comprendido dentro del rango que nos interesa y, mediante la función ord (), que ya conoce, se obtiene un carácter que se incorpora a la cadena. El bucle, eliminando los comentarios para facilitar su legibilidad, queda así:

```
for ($contador=1; $contador<=$longitud; $contador++){
    $caracter=rand(97,122);
    $cadena.=chr($caracter);
}
```

Como ve, más simple no puede ser. A continuación, tomamos la cadena y la asignamos a la variable de sesión, así:

```
$_SESSION["cadena"]=$cadena;
```

El código HTML y JavaScript que aparece al final da paso al formulario de registro, pasándole el identificativo de la sesión.

En el formulario apenas he cuidado los detalles. Una vez más, he querido evitar código superfluo para no distraer su atención de la funcionalidad PHP que nos interesa en este capítulo. Por ejemplo, podría haber usado estilos para mejorar la apariencia. O podría haber incluido un breve código JavaScript que hiciera una comprobación rutinaria del formulario antes de enviarlo. En este ejercicio, usted puede enviar el formulario aunque no haya tecleado un nombre de usuario y una contraseña. En el registro de un usuario en un sitio real, usted debería incluir un código JavaScript que no permitiera el envío del formulario si faltan datos esenciales, o si las dos contraseñas introducidas no coinciden, etc. Pero esa es una fase de la programación en el lado del cliente que nosotros, usted y yo, ya tenemos superada y que no viene al caso ahora. El formulario tiene un aspecto similar al que se ve en la figura 18.3.



Figura 18.3

Fíjese en la siguiente línea del listado del formulario, que es la que nos interesa aquí:

```

```

En la parte resaltada ve cómo llamamos al script que genera la imagen, como fuente de una imagen incluida desde HTML. En el capítulo 12 aprendimos que esto es posible. Observe que, al llamar al script, le pasamos el identificativo de la sesión, para que recupere, en el servidor, la cadena que se generó en index.php. El script **crearImagen.php** aparece listado a continuación:

```
<?php
/* Se establece la cabecera para envío de imágenes al navegador.*/
header("Content-type: image/jpeg");
/* Se inicia la sesión. */
session_start ();
/* Se asigna la variable de sesión a una variable de trabajo. */
$cadena=$_SESSION["cadena"];
```

```
/* Se genera un número aleatorio para elegir un fondo
para la imagen. */
$numeroDeFondo=rand (1,3);

/* Se crea una matriz con tres colores definidos para
el texto. En cada fondo se empleará un color diferente. */
$coloresDeTexto=array(1=>array("r"=>0, "g"=>0,
"b"=>128), array("r"=>255, "g"=>204, "b"=>255), array("r"=>0,
"g"=>0, "b"=>0));

/* Se calcula la longitud de la cadena menos cinco
caracteres. Como la cadena podrá tener entre cinco y diez
caracteres, la siguiente variable adoptará un valor entre
cero y cinco. Esta variable se usará para calcular el
posicionamiento horizontal de la cadena en la imagen.*/
$longitudCadena=strlen($cadena)-5;

/* Se abre una imagen con el fondo elegido
aleatoriamente. */

$imagenFondo="imagenes/fondos/fondo_". $numeroDeFondo.".jpg";

// Se crea un manejador para una imagen.
$imagenJPG=imagecreatefromjpeg ($imagenFondo);

/* Como ya se sabe el fondo que se pondrá, se elige el
color adecuado para el texto y se incorpora a la paleta. */
$colorDeCadena=imagecolorallocate ($imagenJPG,
$coloresDeTexto[$numeroDeFondo] ["r"],
$coloresDeTexto[$numeroDeFondo] ["g"],
$coloresDeTexto[$numeroDeFondo] ["b"]);

/* Se determinan las propiedades de texto para
incorporar la cadena a la imagen.*/
$talla=30;
$angulo=rand(-18,18);
$x=65-($longitudCadena*12);
$y=60+($angulo*2);
$fuente="fuentes/courier.ttf"; // Letra de ancho
fijo.

/* Se monta la cadena sobre la imagen. */
imagettftext($imagenJPG, $talla, $angulo, $x, $y,
$colorDeCadena, $fuente, $cadena);

/* Se trazan diez líneas aleatorias para "ensuciar" la
imagen. */
for ($contador=1; $contador<=10; $contador++){
$xOrigen=rand(0,249);
```

```
$yOrigen=rand(0,99);
$xFinal=rand(0,249);
$yFinal=rand(0,99);
imageline ($imagenJPG, $xOrigen, $yOrigen, $xFinal,
$yFinal, $colorDeCadena);
}

// Se envía la imagen al navegador.
imagejpeg($imagenJPG);
// Se liberan recursos.
imagedestroy($imagenJPG);
?>
```

El código empieza enviando la cabecera que nos permitirá mandar luego la imagen al navegador, abriendo la sesión y recuperando el valor de la cadena aleatoria de la correspondiente variable de sesión, así:

```
header("Content-type: image/jpeg");
session_start ();
$cadena=$_SESSION["cadena"];
```

Para crear la imagen disponemos de tres fondos. En cada ejecución se elegirá uno de los tres, de forma aleatoria. Los fondos los hemos llamado fondo\_1.jpg, fondo\_2.jpg y fondo\_3.jpg. Así, tomar uno al azar es tan fácil como generar un número aleatorio entre uno y tres y usarlo para construir una cadena con el nombre del archivo que vamos a cargar.

También tendremos que añadir a la imagen un color para el texto. Este color deberá ser diferente, según el fondo que se cargue, para que la cadena destaque lo suficiente como para ser legible. Lo que hacemos es crear una matriz en memoria con los valores de rojo, verde y azul para los tres posibles colores del texto, de la siguiente manera:

```
$coloresDeTexto=array(1=>array("r"=>0, "g"=>0,
"b"=>128), array("r"=>255, "g"=>204, "b"=>255), array("r"=>0,
"g"=>0, "b"=>0));
```

Como ve, se trata de una matriz bidimensional, es decir, una matriz de matrices. Es indexada, empezando a partir de 1, para que cada matriz asociativa corresponda a uno de los posibles fondos. Cada una de las matrices asociativas tiene los tres valores de rojo, verde y azul para obtener el color deseado.

A continuación, elegimos un fondo, según el número aleatorio que obtuvimos anteriormente, y lo abrimos, asignándole un manejador, así:

```
$imagenFondo="imagenes/fondos/fondo_".  
$numeroDeFondo.".jpg";  
$imagenJPG=imagecreatefromjpeg ($imagenFondo);
```

Ahora, a la paleta de colores se le incorpora el color para dibujar la cadena de texto, según el índice de la matriz que corresponda al número elegido para el color de fondo:

```
$colorDeCadena=imagecolorallocate ($imagenJPG,  
$coloresDeTexto[$numeroDeFondo] ["r"],  
$coloresDeTexto[$numeroDeFondo] ["g"],  
$coloresDeTexto[$numeroDeFondo] ["b"]);
```

Para ello empleamos la función imagecolorallocate (), que vimos en el capítulo 12 del libro.

El siguiente paso es dibujar la cadena en la imagen. Podríamos haber usado la función imagestring (), pero sus resultados son muy pobres. En su lugar, emplearemos imagettftext () que, como sabe, es mucho más potente y flexible. Lo primero que debemos hacer es determinar los parámetros que deberemos pasarle a la función, así:

```
$talla=30;  
$angulo=rand(-18,18);  
$x=65-($longitudCadena*12);  
$y=60+($angulo*2);  
$fuente="fuentes/courier.ttf";
```

El tamaño (variable **\$talla**) lo hemos fijado en 30 puntos. El ángulo del texto, respecto de la horizontal, será un valor aleatorio, comprendido entre los límites -18 y 18, a fin de que ninguna parte de la cadena caiga fuera de la imagen. Las coordenadas de posicionamiento de la cadena con respecto a la imagen de fondo (variables **\$x** y **\$y**) se determinan en base a la longitud de la cadena y el ángulo, de forma que la cadena siempre quepa completa en la imagen. Por último, está la fuente (variable **\$fuente**). Hemos elegido la letra courier para que todos los caracteres tengan el mismo ancho. De este modo, es más fácil el posicionamiento de la cadena.

Ahora dibujamos la cadena, con los datos anteriores, sobre la imagen, como se aprecia a continuación:

```
imagettfttext($imagenJPG, $talla, $angulo, $x, $y,  
$colorDeCadena, $fuente, $cadena);
```

Con esto, la imagen ya estaría formada. Sin embargo, los bots más recientes usan técnicas de OCR (*Optical Character Recognizing*, Reconocimiento Óptico de Caracteres) para poder leer las cadenas que aparecen en este tipo de imágenes. Para dificultar el uso de OCR, le hemos dado a la cadena un ángulo, para que no esté totalmente horizontal. Pero podemos hacer algo más: podemos "ensuciar" la imagen con trazos que no le dificulten la lectura a un operador humano, pero si a un sistema automatizado. Para ello, vamos a recurrir a un bucle que genera diez líneas aleatorias, que se incorporan a la imagen, con el mismo color que el texto. Se trata de un bucle que ejecuta diez iteraciones. En cada una de ellas determina unas coordenadas al azar (siempre dentro de los límites de la imagen) y traza una línea mediante el uso de la función `imageline()`, que usted ya conoce. El código de este bucle es el siguiente:

```
for ($contador=1; $contador<=10; $contador++) {
    $xOrigen=rand(0,249);
    $yOrigen=rand(0,99);
    $xFinal=rand(0,249);
    $yFinal=rand(0,99);
    imageline ($imagenJPG, $xOrigen, $yOrigen, $xFinal,
    $yFinal, $colorDeCadena);
}
```

Para finalizar el script, se envía la imagen generada al navegador y se liberan los recursos empleados. Por último, tenemos el script que determina si el usuario ha tecleado la cadena correcta en la casilla correspondiente del formulario. El listado del script, simplificado al máximo, se llama **comprobarResultado.php**.

```
<?php
session_start();
if ($_POST["textoDeImagen"] == $_SESSION["cadena"]){
    echo ("ACCESO CONCEDIDO");
} else {
    echo ("ACCESO DENEGADO");
}
?>
```

Con este sistema (conocido popularmente como **captcha**) estaremos a salvo de los ataques de los bots.

## 18.3 FORMULARIOS EN DOCUMENTOS

En muchos portales de Internet se le solicita al usuario que se registre para poder acceder a los contenidos. Al registrarse el usuario, este espera, si acaso sea a nivel subconsciente, un mínimo de personalización. Una solución muy interesante

es usar un documento editable en formato de texto enriquecido (**RTF, Rich Text Format**) en el que se incluirán los datos personales del usuario, tal como los haya tecleado en el formulario de registro, logrando una carta de presentación personalizada. Ésta podrá ser guardada en el ordenador del cliente. El usuario podrá editarla, imprimirla, etc.

La cuestión es disponer de una carta modelo, que deberá estar archivada en el servidor. En esta carta habrá ciertas partes del texto que serán genéricas y que se reemplazarán, en cada caso, con los datos suministrados por el usuario a través del formulario. De este modo, tendremos una carta que se personalizará para cada usuario. La carta modelo puede responder a la de la figura 18.4.



Figura 18.4

Observe dónde aparecen los óvalos rojos (que no son parte del documento). Hay unas claves que aparecen como una secuencia de tres letras mayúsculas, precedidas y sucedidas por dobles acentos circunflejos. Estas claves, tal como se han dispuesto en el documento, serán sustituidas por el nombre real del usuario (clave ^^NRE^^), su login (clave ^^LOG^^), su contraseña (clave ^^PWD^^) y su dirección de correo electrónico (clave ^^COR^^).

Lo primero que necesitamos es un formulario donde el usuario teclee estos datos, para darse de alta en nuestro portal. Una vez más, hemos omitido las más elementales comprobaciones del formulario, a nivel de JavaScript en el lado del cliente, así como los estilos, para facilitar la legibilidad del código. El listado de la página se llama **formulario.htm**, y es el siguiente:

```
<html>
  <head>
    <title>Formulario para RTF</title>
  </head>
  <body>
    <h2>BIENVENIDO A NUESTRA COMUNIDAD VIRTUAL</h2>
    <h3>RELLENA EL SIGUIENTE FORMULARIO PARA
INSCRIBIRTE</h3>
    <form action="mostrarRTF.php" method="post"
name="fSuscripcion" id="fSuscripcion">
      <table width="500" height="205" border="0"
cellpadding="4" cellspacing="0">
        <tr>
          <td height="15" align="right"><strong>Nombre
real:&nbsp;</strong></td>
          <td><input name="NRE" type="text"
id="NRE"></td>
        </tr>
        <tr>
          <td height="15" align="right"><strong>Nombre
de usuario:&nbsp;</strong></td>
          <td><input name="LOG" type="text"
id="LOG"></td>
        </tr>
        <tr>
          <td height="30"
align="right"><strong>Contraseña:&nbsp;</strong></td>
          <td><input name="PWD" type="text"
id="PWD"></td>
        </tr>
        <tr>
          <td height="30" align="right"><strong>Correo
electr&acute;nico:&nbsp;</strong></td>
```

```
<td><input name="COR" type="text"
id="COR"></td>
</tr>
<tr align="center">
    <td height="85" colspan="2"><input
type="submit" name="Submit" value="Enviar"></td>
    </tr>
</table>
</form>
</body>
</html>
```

Observe las líneas resaltadas. Vea que a los campos se les ha asignado unos nombres que coinciden con las claves en la carta modelo. Este formulario se envía al script **mostrarRTF.php**, cuyo listado es el siguiente:

```
<?php
    function convertir($textoFinal) {
        foreach($_POST as $clave=>$valor) $textoFinal =
str_replace("^{".strtoupper($clave)."}", $valor,
$textoFinal);
        return $textoFinal;
    }
    header("Content-type: application/msword");
    header("Content-Disposition: inline");
    $fichero = "saludos.rtf";
    $manejador = fopen($fichero,"r");
    $textoFinal = fread($manejador,filesize($fichero));
    $textoFinal = convertir($textoFinal);
    echo($textoFinal);
?>
```

Como ve, es muy simple. En primer lugar tenemos dos cabeceras destinadas a informar al navegador que se va a enviar un documento rtf. Éstas son:

```
header("Content-type: application/msword");
header("Content-Disposition: inline");
```

Ahora abrimos el fichero RTF y lo leemos, tal como aprendimos a hacer en el capítulo 8. El hecho de que no se trate de un fichero de texto plano, como los que usábamos en dicho capítulo es, en este caso, totalmente irrelevante. La cuestión es que leemos todo el contenido del fichero de una vez y lo almacenamos en una variable, llamada **\$textoFinal**.

Y ahora viene la parte interesante. Es cuando se llama a la función destinada a sustituir las claves de la carta genérica por los datos del usuario. Como

ve, es muy simple. Buscamos cada uno de los datos que formaban parte del formulario y que, como usted sabe, han sido enviados en la matriz `$_POST`, ya que el formulario se mandó por este método. Como los nombres de los campos coinciden con las claves, lo que hacemos es recorrer toda la matriz con un bucle `foreach` y reemplazar en la cadena cada campo, según se vaya encontrando con su contenido. Los limitadores que aparecen (los dobles circunflejos) antes y después de cada clave se emplean para que la función `str_replace()` no sustituya nada más que las claves por los datos del usuario.

Cargue en el navegador la página que contiene el formulario, tecleando, en la barra de direcciones [http://localhost/capítulo\\_18/formularios/formulario.htm](http://localhost/capitulo_18/formularios/formulario.htm) y rellene unos datos ficticios. Cuando pulse el botón `Enviar` se le abrirá un cuadro de diálogo como el de la figura 18.5.



Figura 18.5

Pulse el botón `Abrir` y se le abrirá una ventana con la carta personalizada con los datos que introdujera en el formulario.

## 18.4 ACTUALIZACIONES AUTOMÁTICAS

Como usted sabe, uno de los problemas que presenta la actualización de los contenidos de su sitio en Internet es que los usuarios pueden no estar viendo la última versión. Si un usuario habitual se conecta a sus sitios con frecuencia, es posible que la caché del navegador le muestre unos contenidos que no están actualizados. Para evitar esto, PHP le ofrece algunas cabeceras que deberá incluir en sus páginas. Son sólo unas pocas líneas, con lo que los ficheros no se sobrecargan, y merece la pena. Éstas son las siguientes:

```
header ("Expires: Mon, 22 Sep 1997 09:00:00 GMT");
header ("Last-Modified: " . gmdate("D, d M Y H:i:s") .
" GMT");
```

```
    header ("Cache-Control: no-store, no-cache, must-
revalidate");
    header ("Cache-Control: post-check=0, pre-check=0",
false);
    header ("Pragma: no-cache");
```

En la carpeta de ejercicios correspondiente a este capítulo tiene un fichero, llamado **actual.php**. Este fichero no es realmente un script de PHP. Sólo contiene estas líneas que acaba de ver, para que las copie y pegue en sus scripts. No cometa el error de intentar incluir este fichero con include () o require (). Sólo copie y pegue las líneas que contiene al principio de cada listado.



## CAPÍTULO 19

### FOROS EN INTERNET

---

---

Una de las aplicaciones más populares que podemos encontrar en Internet es un foro. Sin duda, uno de los mejores y más llamativos (al menos para nosotros, los que vivimos de la informática) es el que podemos encontrar en la web del programador (<http://www.lawebdelprogramador.com>). Sin embargo, en la Red los hay de todos los tipos y para todos los gustos. Existen foros de música, de cine, de encuentros entre amigos, de temas políticos, religiosos, sociales... La lista es interminable. Y también hay foros generalistas, donde se habla de todo.

Usted sabe cómo funcionan estas aplicaciones. En primer lugar, se tiene que inscribir el usuario. Esto no siempre es así. Existen algunos foros abiertos, donde no es necesario inscribirse, pero, en la mayoría de ellos, sí lo es. Después, recibe un e-mail, solicitándole confirmación y, a partir de ahí, ya puede postear sus notas, como en un tablón de anuncios virtual. Los demás usuarios las verán y podrán publicar respuestas a sus preguntas o comentarios. Además, usted también puede publicar respuestas a los posts de otros usuarios.

En realidad, un foro es, conceptualmente, muy simple. Con los conocimientos que hemos adquirido en este libro usted ya podría ponerse a la tarea de desarrollar una aplicación de este tipo y publicarla. Pero, seguramente, le vendría muy bien una experiencia adicional: ese conocimiento no escrito que sólo se obtiene con el tiempo y el trabajo.

Sin embargo, hay un modo muy sencillo de publicar un foro totalmente profesional, probado y que funciona, muy completo y eficiente. Y encima, es totalmente gratis. Estoy hablando de una plantilla configurable que usted puede usar para diseñar su propio foro, con muy poco trabajo. Una vez diseñado, sólo

debe colocar los archivos en un servidor que haya contratado y estará funcionando en pocos minutos. La plantilla de la que hablo es phpBB3 (PHP Bulletin Board), que podemos descargar libremente de su página oficial (<http://www.phpbb.com>). En el momento de redactar estas líneas la última versión disponible es la 2.0.20, que se encuentra en la carpeta correspondiente a este capítulo.

## 19.1 OBTENIENDO PHPBB3

A estas alturas del libro, le supongo lo bastante interesado como para querer tener la última versión disponible, de forma completamente gratuita: conéctese a la página [www.phpbb.com](http://www.phpbb.com), cuyo aspecto ve en la figura 19.1.



Figura 19.1

Pulse el enlace **DOWNLOAD LATEST RELEASE**, situado a la derecha debajo de la cabecera de página, para pasar a la página de descargas. Posteriormente, podrá seguir explorando el sitio si desea más información sobre este producto. Uno de los enlaces más interesantes es **SUPPORT**, en la barra gris que forma parte de la cabecera, donde se accede a gran cantidad de documentación sobre PHPBB. Lamentablemente, está en inglés. Si usted no se halla familiarizado con este idioma, quizás sea el momento de plantearse su estudio. No obstante, en este capítulo encontrará todo lo necesario para poner en marcha su propio foro.

Una vez en la página de descargas, busque el apartado que se refiere al paquete completo (Full package). Si su ordenador trabaja bajo Windows, pulse el enlace [.zip](#); si su equipo es Linux, haga clic en [.bz2](#). Ambos enlaces acceden al mismo contenido. El caso es que, dicho contenido está comprimido y empaquetado para que sea más rápida y cómoda la descarga. El formato de compresión zip es más habitual en entornos Windows, mientras que bz2 es más propio de plataformas

Linux. En el caso de su ordenador personal es muy fácil que sea una plataforma Windows, mayoritaria entre los usuarios domésticos; en el caso de un servidor web profesional es más fácil que se trate de una plataforma Linux, por varias razones, entre las que se cuentan fiabilidad, estabilidad y un coste más asequible.

Una vez descargado el paquete, pulse el enlace **TRANSLATIONS**, en la página de descargas. Busque el idioma español, donde podrá descargar los paquetes para la aplicación y para las dos plantillas principales (subsilver 2 y prosilver), en los enlaces que ve en la figura 19.2.

Español Spanish							<a href="#">View changelog</a>
Last Updated: Wed Mar 23, 2010 10:46 pm Maintainer: JuanMane							

Figura 19.2

En la carpeta correspondiente a este capítulo, los paquetes ya han sido descomprimidos, y tenemos a la vista la carpeta **phpBB3**, que contiene todos los scripts, imágenes y archivos complementarios para instalar el foro, en inglés o español. Con esto ya tenemos todo el material necesario para empezar a trabajar. Vamos a hacer una instalación previa del foro, en modo local.

## 19.2 INSTALANDO EL FORO

Instalar el foro consiste en establecer algunos parámetros, para que luego, al tenerlo a disposición del público, satisfaga nuestras expectativas y las de nuestros usuarios.

### 19.2.1 Primeros pasos

El primer paso es colocarlo en un servidor público de Internet. A tal fin, primero debe tener un nombre de dominio y un alojamiento contratado. Éste deberá estar en un servidor que soporte PHP y MySQL. Hoy día, este tipo de servicios se contrata a costes muy reducidos. Yo le sugiero que visite el sitio de AMEN (<http://www.amen.es>) donde puede contratar varios packs. Si se pone en contacto por e-mail con los administradores del sitio, le asesorarán acerca del que más le interesa. Tal vez le convenga contratar su nombre de dominio y alquilar un pack Servidor Privado Linux. Su coste no es alto y las prestaciones que ofrece son estupendas. De hecho es uno de los mejores proveedores que yo conozco. Además, una vez contratado, le prestan todo el soporte post-venta que pueda necesitar, de una forma rápida y eficiente. No vamos a entrar aquí en detalles acerca de la

contratación y uso del alojamiento, ya que no es el objetivo de este libro. Si tiene alguna duda específica en cuanto al uso del servidor, la empresa con la que haya contratado le prestará asistencia técnica. En mi experiencia con AMEN, ésta es inmejorable.

Cuando haya contratado el dominio y el alojamiento, transfiera, mediante un programa de FTP o mediante la propia interfaz de AMEN (un sistema visual muy intuitivo y cómodo de utilizar), todo el contenido de la carpeta phpBB2. Fíjese en que no le digo que transfiera esta carpeta con su contenido, sino, directamente dicho contenido, que deberá quedar en la carpeta raíz de su sitio.

Como alternativa, puede instalarlo en su ordenador, en modo local, hasta que llegue el momento de publicar un foro "real". Recuerde que en el capítulo 2 aprendimos a montar un servidor local (localhost).

Ahora vamos a fijarnos en el script **crearBBDD.php**, que le he puesto en la carpeta correspondiente a este capítulo. Su listado es el siguiente:

```
<?php  
$conexion = mysql_connect("localhost","root","");
$consulta = "CREATE DATABASE IF NOT EXISTS foros";
mysql_query($consulta, $conexion);
mysql_close ($conexion);
?>
```

Su objetivo es crear la base de datos necesaria para los foros. Preste especial atención a la línea que establece conexión con el motor de base de datos. Deberá modificar los parámetros de conexión de acuerdo con los que le proporciona la empresa con la que ha contratado el servidor. Una vez modificado el script, súbalo también a la carpeta raíz del alojamiento que haya contratado, junto con los demás archivos y carpetas que subió anteriormente.

Vamos a suponer que su sitio se llame <http://www.misforos.com>. Tenga presente el nombre. Nosotros, en este capítulo, haremos la configuración en el servidor local. Usted puede hacerla así la primera vez, para familiarizarse con el procedimiento. Pero lo normal es que configure el foro que tenga subido en Internet. En los pasos siguientes sustituya [http://localhost/capítulo\\_19/phpBB3/](http://localhost/capitulo_19/phpBB3/) por <http://www.misforos.com/> (o el nombre de dominio que haya contratado; aquí supondremos que su dominio es el que le he puesto de ejemplo).

Empiece por ejecutar el script **crearBBDD.php**. Si está trabajando en modo local, teclee, en la barra de direcciones de su navegador, o en la equivalente en la barra de tareas, [http://localhost/capítulo\\_19/crearBBDD.php](http://localhost/capítulo_19/crearBBDD.php). Si ya ha

subido los ficheros al servidor remoto que vaya a usar para la aplicación, teclee <http://www.misforos.com/crearBBDD.php>. Si no se produce ningún error, el resultado es una página en blanco.

Ahora que ya está creada la base de datos, llamaremos a la página de instalación de los foros. Si está en modo local, la dirección es [http://localhost/capítulo\\_19/phpBB3](http://localhost/capítulo_19/phpBB3). Para el servidor remoto sería <http://www.misforos.com>. Con esto accederá a la página principal de instalación, cuyo aspecto es el de la figura 19.3.



Figura 19.3

Puede que la página no quepa completa en su pantalla. En ese caso, deberá desplazarla hacia abajo para verla completa. Se trata de una breve introducción a la aplicación. A la izquierda tiene unos enlaces para ver el texto de la licencia GNU/GPL que avala a este software y acceder al soporte. Pulse la pestaña **INSTALAR** para dar paso a la instalación. La pestaña **CONVERTIR** sólo tiene uso si existe una versión previa (**phpBB2**, por ejemplo).

Esta versión del programa soporta múltiples Bases de Datos, tal como vemos en la figura 19.4.

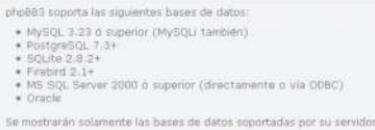


Figura 19.4

A continuación pulsaremos el botón Proceder al siguiente paso, situado en la parte inferior de la página y pasamos a ver los requerimientos del sistema, que deberán aparecer en verde, tal como se ve en la figura 19.5.

<b>versión y parámetros PHP:</b>	
<b>Obligatorio:</b> Necesita al menos la versión 4.3.3 de PHP con el fin de instalar phpBB3. Si este modo se muestra debajo de su instalación PHP será funcionando en ese modo. Esta verificación limitará las en la administración remota o en implementaciones similares.	
versión PHP >= 4.3.3:	Sí
Parámetro PHP register_globals deshabilitado:	Sí
Todos las funciones que si están activadas están inhabilitadas para que no se ejecuten. Sin embargo, para que el script funcione bien, el parámetro register_globals debe deshabilitado en su instalación PHP por razones de seguridad.	
Parámetro PHP allow_url_fopen está habilitado:	Sí
Obligatorio para que el script funcione. El script usa algunas funciones de phpinfo como ayudas para dar información sobre las funciones adecuadamente en el.	
La función PHP getmetype() está disponible:	Sí
<b>Obligatorio:</b> Para que phpBB3 funcione correctamente, la función getmetype() debe estar disponible.	
Soporte PCRE (PCRE):	Sí
soporta las funciones si su instalación PHP no está compilada con soporte para PCRE (o es la extensión PCRE).	

Figura 19.5

Si alguno de estos requerimientos no se cumple deberemos revisar nuestra instalación de PHP. Más abajo, en la misma página vemos una relación de motores de bases de datos instalados en nuestro equipo. En realidad, para la mayoría de las aplicaciones con tener MySQL es suficiente.

A continuación las secciones correspondientes a los módulos opcionales y a las carpetas y archivos de trabajo. Pulsaremos el botón **Comenzar instalación** para empezar con el proceso que nos ocupa.

Lo siguiente que vemos es un breve formulario donde se configura el acceso a la base de datos. En la figura 19.6 vemos lo que podría ser una instalación típica en modo local. En el caso de usar un servidor público, deberíamos especificar un nombre de usuario y una contraseña que serán los que nos haya proporcionado la empresa suministradora del hosting.

<b>Configuración de base de datos:</b>	
Tipo de base de datos:	MySQL con Extensiones PHP5.0
NOMBRE del servidor de la base de datos u ODBC:	localhost
puerto:	3306
socket:	
Nombre en el servidor de la base de datos:	foro
Nombre de usuario:	root
Contraseña:	root
Clave de base de datos:	root
Prefijo para tablas en la base de datos:	foro_
<b>Proceder al siguiente paso</b>	

Figura 19.6

En la siguiente fase, se hace una prueba de conexión con la Base de Datos. Si todo va bien, el resultado debe ser el de la figura 19.7.



Figura 19.7

En caso de producirse un error, éste suele deberse a haber tecleado mal el nombre de usuario o la contraseña del motor de base de datos. En ese caso, deberá teclear de nuevo los datos anteriores. Si todo ha ido bien hasta aquí, pulse el botón Proceder al siguiente paso, que le llevará al formulario que permite establecer los datos básicos del administrador. Aquí decide usted cual será su nombre de usuario y su contraseña. En este ejemplo he usado **admin\_foros** como ambos datos aunque, en una instalación real, deberíamos buscar una contraseña más segura. El formulario típico podría tener el aspecto de la figura 19.8.

The screenshot shows a configuration form for an administrator. It includes fields for the site's default language (set to Spanish), the administrator's name (set to admin\_foros), and the administrator's password (represented by a series of asterisks). There are also fields for confirming the password, the contact email (set to pruebas@localhost.com), and confirming the contact email (also set to pruebas@localhost.com). At the bottom is a button labeled "Proceder al siguiente paso".

Figura 19.8

Ponga especial cuidado en no olvidar el nombre de administrador y la contraseña que elija aquí, ya que no podría acceder posteriormente al panel de administración de los foros. Tenga especial cuidado en el uso de mayúsculas y minúsculas. Esto suele ser fuente de problemas. Dependiendo de cómo tenga configurado su sistema operativo, podría recibir indicaciones específicas en

pantalla cuando tenga activado el modo mayúsculas. Una vez decidido su nombre y contraseña, anótelos en un papel, para no olvidarlos.

Recuerde que en una instalación real, deberá indicar correctamente su correo electrónico, ya que la aplicación lo usará para ponerse en contacto con usted si llega el momento de enviarle algún aviso.

El siguiente paso es una prueba de conectividad y, a continuación, la confirmación de haber llegado correctamente hasta aquí. A continuación aparece un formulario donde se establecen algunos datos relativos al SMTP (el protocolo de envío de correos electrónicos) y al propio servidor de alojamiento. Normalmente no deberá tocar nada en una instalación local. En el caso de una instalación remota en un servidor público, si debe modificar alguno de los valores que aparecen por defecto su proveedor de servicios de Internet (ISP) le proporcionará los datos adecuados.

Tras esto, se nos informa de que se han creado las tablas de la base de datos, y algunos datos básicos para el funcionamiento inicial (por ejemplo, los datos de administrador).

Después de esto, se muestra el mensaje de la figura 19.9.



Figura 19.9

Nos interesa especialmente el último párrafo, que nos dice que debemos eliminar la carpeta **Install** antes de continuar. Hágalo ahora. Esto es un mecanismo de seguridad para evitar reinstalar la aplicación borrando datos existentes.

Pulse el botón Identificarse y accederá al panel de administración de su aplicación. La instalación ha concluido.

## 19.2.2 Configurando el foro

Llegados a este punto podríamos sentirnos un poco abrumados por la cantidad de opciones y enlaces que vemos en la página. Después de todo, somos el Administrador Principal del sitio y tenemos que poder configurar absolutamente todos los detalles de funcionamiento del mismo. A la izquierda vemos más de veinte enlaces a distintas opciones, y eso sólo en la pestaña General, que aparece activada por defecto, aunque, en la parte superior hay otras siete pestañas, cada una con sus diferentes opciones. La parte superior de la página tiene el aspecto de la figura 19.10.

ESTADÍSTICA	VALOR	ESTADÍSTICA	VALOR
Número de mensajes:	1	Pensiones por día:	1
Número de temas:	1	Tareas por día:	1
Número de usuarios:	1	Último por día:	
Número de adjuntos:	0	Adjuntos por día:	0.00

Figura 19.10

En la página de phpBB existe una detalladísima documentación aunque, lamentablemente, sólo está disponible en inglés. Aquí no vamos a ver todas las opciones posibles, ya que excede con mucho el objetivo de este capítulo. Sin embargo si vamos a hacer una útil visita a vista de pájaro de los detalles que más nos interese conocer para gestionar un foro en Internet.

En la página que hemos visto, en la pestaña **GENERAL**, aparece, un poco más abajo, un historial de las cinco últimas operaciones que realicemos como administradores, tal como se ve en la figura 19.11.

USUARIO	IP DEL USUARIO	HORA	ACIÓN
admin_foros	127.0.0.1	Sab, 10 Abr 2010, 01:18	Jabber: parámetros cambiados
admin_foros	127.0.0.1	Sab, 10 Abr 2010, 01:00	Instalado phpBB 3.0.7-PL1

Figura 19.11

Si queremos ver más de cinco movimientos podremos pulsar en el enlace **Ver log de administradores**. Esto es especialmente útil cuando nuestro sitio contiene muchos foros, posts y usuarios y realizamos con frecuencia operaciones de mantenimiento, de modo que podemos no recordar cuándo hicimos tal o cual cosa, o si la hemos hecho siquiera.

Un apartado importante, dentro de la pestaña GENERAL, es el que aparece como Configuración del sitio. Pulse el enlace situado a la izquierda de la página, en la sección del mismo nombre, y accederá al formulario que se ve en la figura 19.12, donde configuraremos los datos más fundamentales del foro.

Configuración del Sitio	
Nombre del Sitio:	Mis foros
Descripción del Sitio:	Foros de prueba en mi servidor local
Deshabilitar Sitio:	<input type="radio"/> Sí <input checked="" type="radio"/> No Este hace al Sitio inaccesible a los usuarios. Si quiere, también puede introducir un breve mensaje (255 caracteres) para mostrar.
Idioma por defecto:	Español <input type="radio"/>
Formato de fecha:	Personalizar... D, d M Y, H:i
Zona horaria del sistema:	[UTC] Horario Islas Canarias, Horario Europa Oc...
Habilitar horario de ahorro de energía:	<input type="radio"/> Sí <input checked="" type="radio"/> No
Estilo por defecto:	prosilver <input type="radio"/>
Sustituir estilo del usuario:	<input type="radio"/> Sí <input checked="" type="radio"/> No Reemplaza el estilo del usuario por el estilo por defecto.

Figura 19.12

En una instalación de prueba en modo local, los datos que aparecen en la imagen nos valen perfectamente. En caso de ser un sitio público, elija bien el nombre y la descripción del mismo, ya que esta información influirá en el posicionamiento del sitio cuando éste sea localizado por los buscadores.

La opción **Deshabilitar Sitio** permite bloquear el acceso a los usuarios, con carácter temporal, si se tienen que hacer cambios importantes en la instalación o contenidos de las tablas. En ese caso se puede incluir un mensaje de tipo *Sitio en reparación*, o similar, para que lo vean los usuarios que intenten acceder.

También podemos cambiar, por ejemplo, el formato en que se manejarán las fechas, o la zona horaria.

Especial relevancia podría tener la opción de Estilo por defecto, que nos permite fijar el aspecto o skin del foro, si tenemos más de un tema instalado. En este caso sólo tenemos disponible prosilver.

Para ver los estilos o temas con los que contamos pulse la pestaña **ESTILOS** de la parte superior y verá una página como la de la figura 19.13. Observe que tiene disponible, aunque desinstalado, el estilo **subsilver 2**. Pulse en el enlace **Instalar**, para que se instale este estilo en su foro. En el formulario que aparece no cambie nada. Simplemente pulse el botón Enviar, al final de dicho formulario.

Más adelante volveremos sobre la cuestión de la instalación de temas.

HOMBRE DEL ESTILO	USADO POR (USUARIOS ACTIVOS)	Opciones	Acciones
<b>Estilos instalados</b>			
<b>prosilver</b> *	52	Detalles   Desactivar   Exportar   Borrar   Vista previa	
<b>Estilos desinstalados</b>			
<b>subsilver2</b>			Instalar

Copyright: © 2008 phpBB Group

Crear nuevo estilo:  Opcional  Enviar

Figura 19.13

En el formulario de la figura 19.12 merece especial atención la última opción, que permite forzar a que los usuarios que visiten el foro lo encuentren con el tema elegido por el administrador. Si no usamos esta opción, cada usuario podrá elegir un tema de los que tengamos instalados. En principio parece que deberíamos dar libertad a los usuarios para que seleccionaran su aspecto favorito y, en la mayoría de los casos, esto será lo adecuado. No obstante, en algunos casos determinados temas podrían no funcionar siempre bien y, en ese caso, lo mejor es que el administrador elija el más adecuado. Como siempre, en situaciones así, lo mejor es probar los temas que nos gusten y ver qué resultado dan.

Dentro de la pestaña **GENERAL**, no debe pasar por alto la opción **Características del sitio**, que es una de las más versátiles e interesantes a la hora de la configuración. De los valores que se pueden establecer en esta página, preste especial atención a los siguientes:

- **Mensajes privados.** Determina si los usuarios podrán enviarse, entre sí, mensajes privados, es decir, que no estarán visibles a los demás usuarios del foro. Por supuesto, si usted habilita esta opción, cada usuario tendrá la posibilidad de que uno o todos los demás usuarios no puedan enviarle mensajes privados. Dicho de otro modo: habilitar esta opción permite a los usuarios recibir mensajes privados, pero no les obliga a ello.

- **Permitir la suscripción a temas.** Permite que los usuarios se suscriban a uno o más hilos del foro, de modo que se les notifique por e-mail cuando se produzcan nuevos posts en los hilos de su interés.
- **Permitir adjuntos y Permitir adjuntos en mensajes privados.** Estas dos opciones deberán estar deshabilitadas en la mayoría de los casos, ya que podrían permitir, a un usuario malintencionado, el envío de software malicioso. Con el tiempo, uno se vuelve paranoico respecto a la seguridad en Internet, pero es mejor pecar por prudencia que por riesgo.
- **Permitir BBCode, emoticonos y firmas.** ¿A quién no le gusta personalizar sus posts o sus mensajes privados? Salvo que sus foros sean muy formalistas o dirigidos a un público muy reservado, estas opciones deberían mantenerse activadas. Sin embargo, permítame un comentario, a tenor de lo mencionado en el punto anterior, respecto a los BBCode. Un hipotético usuario malintencionado podría usar los foros para publicar enlaces a software peligroso, generalmente accesible a través de servidores no controlados, en países de legislación laxa en esta materia.

A continuación pulse en **Configuración de registro de usuarios**, en el menú de la izquierda de la pestaña **GENERAL**. Éste es otro formulario con interesantes opciones, en este caso relativas al registro de los usuarios, entre las que cabe destacar las siguientes:

- **Longitud de nombre de usuario.**
- **Longitud de la contraseña.**
- **Caracteres en nombre de usuario.**
- **Complejidad de la contraseña.**
- **Forzar cambio de contraseña.**
- **Permitir cambios de nombre de usuario.**
- **Permitir reutilización de email.**

El resto de las opciones son, qué duda cabe. Interesantes también. Sin embargo, con esto tenemos una primera configuración del foro válida para echarle un vistazo... de momento.

### 19.3 USANDO EL FORO

Contemplemos, por un momento, el punto de vista del usuario. Cierre el navegador para desconectarse. A continuación, ábralo de nuevo. Cuando un usuario no registrado o no identificado acceda al sitio, verá una página como la de la figura 9.14.



Figura 9.14

Preste especial atención al logo de la cabecera. Por supuesto, éste deberá ser sustituido por el que usted mismo decida para su sitio, acorde con la temática del mismo. Este logo lo encontrará en la carpeta donde haya instalado los foros, con el nombre y ruta siguientes: **styles/prosilver (o el nombre del tema que sea)/imageset/site\_logo.gif**. Sustituya esta imagen por la que usted deseé, teniendo en cuenta que no debe ponerle otro nombre, debe ser de tipo gif, gif 89 o gif 89a y debe tener 139 píxeles de ancho y 52 de alto.

Al margen de este inciso, como nuevo usuario, lo primero que debe hacer es registrarse. Si ya estuviera registrado, debería identificarse con su nombre y contraseña.

### 19.3.1 El registro

El nuevo usuario deberá pulsar sobre el enlace **Registrarse**, que aparece en la parte superior derecha de la página, debajo de la cabecera. Éste da paso a una pantalla con las condiciones de registro. El usuario, para poder seguir adelante, deberá hacer clic sobre el botón rotulado con **Estoy de acuerdo con las reglas**. Con esto accede al formulario de registro. Éste es extremadamente simple. Se le pide al nuevo usuario su nombre, su dirección de e-mail y una contraseña. Además, si el administrador lo habilitó, aparecerá una imagen de las llamadas CAPTCHA, con unos números y letras que el usuario deberá copiar en una casilla de texto. Éste es un recurso muy usado en Internet hoy día para evitar el registro masivo de usuarios "fantasma" por parte de robots. Solo un humano es capaz de leer los dígitos y letras de un CAPTCHA. Ningún programa informático convencional puede hacerlo. En el capítulo 18 usted aprendió cómo programar estos recursos en sus propios sitios.

Un detalle a tener en cuenta es que, funcionando en modo local, puede tener problemas al intentar registrar un nuevo usuario, si no tiene un servidor de correo instalado y conectado a Internet, ya que la aplicación intentará comprobar la validez del dominio de la dirección de e-mail que teclee.

## 19.4 VUELVE EL ADMINISTRADOR

La sección anterior ha sido un inciso, a fin de que compruebe que el foro funciona. En realidad no tiene más misterio, ya que es la parte del usuario y está diseñada como una interfaz funcional, intuitiva y cómoda de usar. Tenga esto en cuenta cuando desarrolle código propio. El usuario no tiene por qué tener conocimientos avanzados de Internet, ni usted debe presuponérselos. Los proyectos que lleve a cabo deberán ser amigables, para evitar que los usuarios elijan otros sitios diferentes de los suyos. Salga del foro y conéctese, de nuevo, como Administrador. Pulse el enlace **Ir al Panel de Administración (ACP)**, situado en la parte más inferior de la página y, tras proporcionar de nuevo su contraseña, como medida de seguridad, volverá a la pantalla de la figura 19.10 que, como recordará, es el punto de arranque para cualquier opción de configuración. Una vez que tenemos una idea de cómo se usa el foro, veamos cómo organizar los contenidos de los usuarios.

### 19.4.1 Categorías y foros

Lo más habitual es que usted quiera que los usuarios que se conecten a su página encuentren una clasificación en los foros que les permita postear sus mensajes en el lugar más adecuado. Por ejemplo, si su foro es relativo al mundo del motor, podrá tener las siguientes categorías:

- Vehículos terrestres.
- Vehículos marítimos.
- Vehículos aéreos.
- Vehículos espaciales.
- Prototipos.

A su vez, cada una de estas categorías podrá tener foros específicos relativos, por ejemplo, a marcas, aspectos comerciales, mecánica, historia, etc. De este modo, un usuario siempre encontrará el foro adecuado para su mensaje, o dónde buscar los mensajes que le puedan interesar de otros usuarios.

En esta versión de phpBB el concepto tradicional de categorías ha desaparecido, tal como nos advierte un texto específico al pulsar sobre la pestaña marcada como **FOROS**, tal como se ve en la figura 19.15.



Figura 19.15

Sin embargo, ésta es una cuestión muy relativa, ya que se pueden crear foros dentro de otros foros, actuando éstos de nivel superior o más global como categorías. De hecho, el foro que se genera por defecto para iniciar el servicio aparece rotulado como **Mi primera categoría**. Así pues, realmente no se produce un cambio funcional relevante, en este sentido, respecto a anteriores versiones del producto, o a otras aplicaciones con este fin.

Y esto nos lleva al primer paso: para renombrar esta categoría (o foro principal, si pretendemos ser estrictos con lo que acabamos de comentar), pulsamos sobre el icono de color verde con forma de rueda dentada que aparece a la derecha del nombre. Esto nos lleva a un formulario donde podemos establecer los aspectos de la categoría o foro, de los que los principales serán los siguientes:

- **Tipo de foro.** Aquí podemos elegir Categoría, Foro o Enlace. Como ve, lo de la desaparición de las categorías que se mencionaba en la página de la figura 9.15 es algo muy relativo. Es posible que, en futuras versiones, esté previsto eliminar definitivamente este concepto, pero, a pesar del aviso que nos ha dado la aplicación, aquí sigue existiendo.
- **Foro padre.** Si éste fuera un foro o subforo, aquí indicaríamos a cuál pertenece. En el caso de una categoría, ésta puede no pertenecer a ninguna otra, o ser una subcategoría.
- **Nombre del foro.** El nombre con el que se verá y utilizará la categoría o foro que estemos creando o editando.
- **Descripción.** Un comentario explicando cuál será la temática del foro.
- **Imagen del foro.** Si queremos que se use una imagen para identificar el foro, aquí teclearemos su nombre, incluyendo la ruta donde se encuentra con respecto a la carpeta raíz del sitio.
- **Contraseña del foro y Confirmar contraseña.** Si se debe teclear una contraseña para usar el foro, es necesario indicarla en estos campos aunque, como nos recuerda un texto en la página, es preferible usar el sistema de permisos de la aplicación.

Desde la página mostrada en la figura 19.15 podemos crear nuevas categorías o foros, o eliminar las existentes. Esto último lo haremos con el icono rojo que aparece a la derecha de cada categoría o foro.

### 19.4.2 Las copias de seguridad

Es un hecho de la vida que los accidentes ocurren. Y, desde luego, el mundo virtual no es una excepción. Es conveniente que usted saque copias de seguridad de la base de datos. De este modo, si por cualquier causa, ésta se estropea o se borra, podrá recuperar la información relativa a todos los usuarios de su foro, así como los mensajes que tuviera. Para efectuar una copia de la base de datos haga clic en el enlace **Copia de seguridad**, situado bajo el encabezamiento **BASE DE DATOS** en el marco izquierdo de la pestaña **MANTENIMIENTO** del Panel de Administración. Con ello verá una página como la de la figura 19.16.



Figura 19.16

Seleccione, si no lo está, el tipo de copia completo, para descargar, y pulse el enlace **Seleccionar todo**, a fin de que la copia se haga de todas las tablas, con estructura y datos. A continuación pulse el botón **Enviar**, lo que le permitirá descargar la copia en su ordenador, local.

Si, en el campo **Tipo de archivo** selecciona la opción **Text**, podrá reinstalar luego la Base de Datos directamente en su servidor, en caso de que sea necesario.

## 19.5 CONCLUSIONES

Fíjese en que en este capítulo nos hemos salido totalmente de la tónica general del libro. No hemos conocido ni una sola instrucción nueva de PHP, ni de SQL, ni hemos añadido nada que ya no supiera usted sobre programación a nivel

de código. Lo que hemos hecho es buscar un enfoque práctico del uso de PHP. El conocimiento de la sintaxis del lenguaje, de sus capacidades y posibilidades es imprescindible para dedicarse a la programación web. Pero seamos claros: es absurdo que uno pierda semanas o, incluso, meses, desarrollando algo que ya existe y que podemos usar libremente. No interprete mal mis palabras. En ningún caso estoy incitando al "pirateo" ni al robo de aplicaciones. Nada más lejos de mi intención. Pero los autores del foro que hemos conocido en este capítulo lo ponen libremente a nuestra disposición para su uso, modificación (si lo deseamos) y redistribución. En Internet existen muchísimas herramientas, sumamente útiles y bien desarrolladas, que se nos ofrecen en esas condiciones. Desde simples scripts para incluir en nuestro sitio con una finalidad concreta hasta aplicaciones completas como la que aquí hemos descubierto. En el Apéndice D del libro tiene muchas direcciones de Internet que no debería dejar de conocer. En ellas puede obtener mucho material y documentación adicional que le ayudará mucho en su trabajo. Por una parte, le ahorrará muchísimas horas. Y eso, en una sociedad apresurada como la nuestra, es vital. Por otra parte, como los scripts PHP son código abierto, siempre podrá estudiarlos y aprender de ellos, viendo cómo otros webmasters con experiencia han resuelto determinadas situaciones. Sin embargo, permítame el lector darle un consejo: NUNCA, bajo ningún concepto, utilice material, aplicaciones, programas, etc., que se hallen sujetos a derechos de autor, si no tiene permiso para ello. Por una parte va contra la Ley. Y por la otra, es una vulneración de la propia dignidad.



## CAPÍTULO 20

### FLASH EN PHP

---

---

El lenguaje PHP es más potente de lo que parece. En los capítulos anteriores hemos visto una muestra de lo que podemos hacer con esta magnífica herramienta. Para cerrar este libro vamos a hablar de algo que muy poca gente conoce. PHP permite desarrollar clips interactivos en Flash (formato .swf) para incorporarlos a nuestras páginas... ¡¡SIN NECESIDAD DE TENER FLASH!!! Esto es importante, ya que el programa de Macromedia resulta caro para un particular. A menudo, tenemos que renunciar a una buena idea para una página por no poder costear una licencia de esta aplicación. Esto, que para una empresa no es relevante, supone, en ocasiones, la diferencia entre poder plasmar nuestra idea o no poder hacerlo, cuando somos usuarios individuales.

PHP proporciona varias clases, cada una con sus propios métodos, para la creación de clips basados en Flash 4 (incluyendo ActionScript). Es cierto que la versión 4 de Flash está obsoleta hoy día (en el momento de escribir estas líneas, Macromedia ofrece el producto MX Studio 8, que incorpora Flash 8). Sin embargo, gran cantidad de las animaciones Flash que vemos en Internet no usan características que no estuvieran ya disponibles en la versión 4 del producto. Sólo en muy contadas ocasiones echaremos de menos alguna cualidad de Flash MX o versiones posteriores.

Antes de seguir adelante debo hacerle una advertencia. Las clases que PHP implementa para la creación de clips swf son experimentales. En el manual oficial de PHP 5 se encuentra, en las páginas relativas a dichas clases, la siguiente advertencia: *Esta función es EXPERIMENTAL. Esto significa que el comportamiento de esta función, el nombre de esta función y en definitiva TODO lo documentado sobre esta función, puede cambiar en una futura versión*

*de PHP SIN AVISO. La advertencia queda hecha, y utilizar esta extensión queda bajo su propia responsabilidad.*

Lo tenemos en cuenta. Sin embargo, desde mi punto de vista (y esto es sólo una opinión personal) no debemos preocuparnos. Las clases no van a desaparecer. Puede que, en futuras versiones cambie el nombre de alguna de ellas, o de alguno de sus métodos. Pero eso será todo. Y, basándome en la experiencia de anteriores versiones de PHP, si cambia algún nombre, el antiguo se mantendrá, si bien sea como un alias, por razones de compatibilidad. También puede que, en futuras versiones, se modifique ligeramente el funcionamiento de algún método. Si es así, quizás tengamos que retocar ligeramente alguno de nuestros scripts. Sin embargo, son muchos "quizás" para un posible cambio en el futuro que ni siquiera podemos asegurar. Así pues, podemos contar con las clases que PHP nos ofrece, y aquí aprenderemos a sacarles partido.

## 20.1 LO QUE NECESITAMOS

Para que las clases que crean y gestionan películas de Flash estén disponibles, es necesario que nuestra instalación de PHP cumpla unos determinados requisitos. Éstos son los siguientes:

- Librería **MING**. Debe estar disponible. En plataformas Windows tendremos el fichero **php\_ming.dll**. En plataformas Linux, el fichero se llama, en cambio, **php\_ming.so**. Si usted trabaja con una plataforma Windows e instaló la aplicación AppServ tal como le indiqué en el capítulo 2, ya tiene su librería MING grabada en el disco.
- Activación de la librería MING. Debe quitar el comentario en la línea correspondiente en el fichero **php.ini** (vea el Apéndice A). Si su instalación es Linux, PHP debe estar compilado con **--with-MING**.
- Además, necesita tener en su ordenador el reproductor Shockwave Placer. Éste es necesario para poder ver cualquier animación de Flash, tanto si la hace usted, como si se la descarga de Internet o si está embebida en cualquier página web. En realidad lo normal es que ya lo tenga, pero, si no es así, se lo puede descargar gratuitamente de la página del fabricante de Flash (<http://www.macromedia.com>).

## 20.2 LA LIBRERÍA MING

En esta sección vamos a incluir un resumen de las clases y métodos de la librería MING. Posteriormente, en otras secciones subsiguientes, aprenderemos su uso. Lea esta sección por encima y utilícela como recordatorio cuando necesite

refrescar su memoria. No trate de aprenderse su contenido. Puede que una pequeña parte del material que aparece aquí ni siquiera llegue a utilizarlo en su trabajo real. Sólo está a modo de referencia. De todos modos, cuando vaya a desarrollar clips de Flash en PHP póngale un marcador a estas páginas. Al final de esta sección comentaremos algunas generalidades importantes que es bueno conocer acerca de la librería MING antes de empezar a trabajar en serio con ella.

Los nombres de las clases de MING empiezan todos con el prefijo SWF. Son un total de trece y aparecen recogidas en la tabla de la figura 20.1.

CLASES DE LA LIBRERÍA MING	
CLASE	USO
<b>SWFACTION</b>	Permite incluir ActionScript en la película.
<b>SWFBITMAP</b>	Se usa para manejo de ficheros de imagen.
<b>SWFBUTTON</b>	Se usa para crear y definir botones interactivos.
<b>SWFDISPLAYITEM</b>	Gestión de propiedades de algunos objetos.
<b>SWFFILL</b>	Su uso es la gestión de rellenos.
<b>SWFFONT</b>	Se usa para manejo de tipografías.
<b>SWFGRADIENT</b>	Su finalidad es gestionar gradientes (degradados).
<b>SWFMORPH</b>	Interpolaciones de forma.
<b>SWFMOVIE</b>	Permite crear y manejar la película principal.
<b>SWFSHAPE</b>	Su uso radica en la creación y gestión de formas geométricas.
<b>SWFSprite</b>	Permite trabajar con clips que se incorporarán a la película principal.
<b>SWFTEXT</b>	Diseñada para la creación de textos, presenta algunas deficiencias en servidores Windows.
<b>SWFTEXTFIELD</b>	Es casi como la anterior, pero funciona correctamente en cualquier plataforma.

Figura 20.1

A continuación haremos un breve recorrido, a vista de pájaro, por las clases.

### 20.2.1 La clase SWFACTION

Esta clase permite crear objetos con distintas instrucciones de ActionScript. Aparte de su propio constructor no tiene métodos.

## 20.2.2 La clase SWFBitmap

Esta clase ha sido especialmente concebida para la gestión de archivos de mapa de bits, de modo que en nuestra película swf podamos incluir imágenes. Sólo trabaja con archivos de tipo gif, png o jpg progresivo. Sus métodos (dos, solamente) aparecen en la tabla de la figura 20.2.

MÉTODOS DE LA CLASE SWFBitmap	
MÉTODO	FINALIDAD
<code>getHeight</code>	Establece la altura de una imagen.
<code>getWidth</code>	Establece la anchura de una imagen.

Figura 20.2

## 20.2.3 La clase SWFButton

Esta clase se emplea para la creación de botones interactivos que se montarán en la película a fin de permitirle al usuario interactuar con los posibles eventos de la misma. Según el evento al que queremos que responda el botón, además de la definición del mismo, cuenta con los siete métodos que aparecen resumidos en la tabla de la figura 20.3.

MÉTODOS DE LA CLASE SWFButton	
MÉTODO	FINALIDAD
<code>addAction</code>	Establece una acción que desencadenará el botón, así como el evento que dará lugar a la misma.
<code>addShape</code>	Permite definir la forma que tendrá el botón en cada uno de sus posibles estados.
<code>setAction</code>	Establece una acción que desencadenará el botón cuando se haga clic en él (el evento más corriente).
<code>setDown</code>	Establece la forma que tendrá el botón cuando se pulse el botón.
<code>setHit</code>	Establece la forma del botón cuando se retire el puntero del ratón.
<code>setUp</code>	Establece la forma del botón cuando se apoya el puntero del ratón.

Figura 20.3

## 20.2.4 La clase SWFDisplayItem

Permite variar propiedades tales como la posición, el ángulo, el tamaño, etc., de determinados objetos. Resulta muy útil, en combinación con otras clases, a la hora de llevar a cabo algunas interpolaciones (término que se usa en entornos Flash para referirse a las modificaciones en una línea de tiempo). Esta clase cuenta con un total de dieciséis métodos mostrados en la tabla de la figura 20.4 que aparece en la página siguiente.

MÉTODOS DE LA CLASE SWFDisplayItem	
MÉTODO	FINALIDAD
<code>addColor</code>	Añade un color para transformaciones, en formato RGB.
<code>move</code>	Mueve el objeto a partir de su posición actual en la película.
<code>moveTo</code>	Mueve el objeto a una posición concreta.
<code>multColor</code>	Establece intensidades para los colores.
<code>remove</code>	Elimina el objeto.
<code>rotate</code>	Gira el objeto en torno al eje Z un determinado número de grados.
<code>rotateTo</code>	Gira el objeto en torno al eje Z hasta un grado determinado.
<code>scale</code>	Escala el objeto con relación a su tamaño actual.
<code>scaleTo</code>	Escala el objeto hasta un determinado tamaño.
<code>setDepth</code>	Establece la profundidad del objeto con respecto a otros de la misma película.
<code>setName</code>	Establece un nombre para el objeto (a efectos de ActionScript).
<code>setRatio</code>	Establece un índice de modificación en una interpolación de forma.
<code>skewX</code>	Distorsiona el objeto en el eje X.
<code>skewXTo</code>	Distorsiona el objeto en el eje X.
<code>skewY</code>	Distorsiona el objeto en el eje Y.
<code>skewYTo</code>	Distorsiona el objeto en el eje Y.

Figura 20.4

## 20.2.5 La clase SWFFill

Se diseñó para escalar o rotar imágenes o gradientes que serán usados para relleno de otras formas. Esta clase cuenta con los cinco métodos mostrados en la tabla de la figura 20.5.

MÉTODOS DE LA CLASE SWFFill	
MÉTODO	FINALIDAD
<code>moveTo</code>	Mueve el origen del relleno a unas coordenadas globales determinadas.
<code>rotateTo</code>	Ajusta la rotación del relleno en el eje Z.
<code>scaleTo</code>	Escala el motivo del relleno.
<code>skewXTo</code>	Distorsiona en el eje X.
<code>skewYTo</code>	Distorsiona en el eje Y.

*Figura 20.5*

### 20.2.6 La clase SWFFont

Esta clase permite definir un objeto de tipografía para usarlo luego en objetos de texto. Su único método aparece en la tabla de la figura 20.6.

MÉTODO DE LA CLASE SWFFont	
MÉTODO	FINALIDAD
<code>getWidth</code>	Obtiene la anchura de una fuente.

*Figura 20.6*

### 20.2.7 La clase SWFGradient

Se usa para crear objetos que definan un tono para un gradiente. Al igual que la clase anterior, sólo cuenta con un método. Cada tonalidad de un gradiente se puede definir de modo individual, formando luego composiciones a través de rellenos. El método de esta clase aparece en la tabla de la figura 20.7.

MÉTODO DE LA CLASE SWFGradient	
MÉTODO	FINALIDAD
<code>addEntry</code>	Añade un tono a la lista para el gradiente.

*Figura 20.7*

### 20.2.8 La clase SWFMorph

Esta clase se usa para definir interpolaciones de forma, es decir, cambios en la morfología de los objetos. Tiene los métodos listados en la figura 20.8.

MÉTODO DE LA CLASE SWFMorph	
MÉTODO	FINALIDAD
<code>getShape1</code>	Obtiene un manejador de la forma inicial.
<code>getShape2</code>	Obtiene un manejador de la forma final.

*Figura 20.8*

### 20.2.9 La clase SWFMovie

Esta clase permite crear un objeto para almacenar la película principal. Es decir, todos los objetos, textos, formas, imágenes, así como los desplazamientos, rotaciones, morfologías, código ActionScript, etc., se montarán en el objeto creado a partir de esta clase. También incluirá clips secundarios (cada uno con su propia línea de tiempo). Cuenta con los diez métodos que vemos en la tabla reproducida en la figura 20.9.

MÉTODO DE LA CLASE SWFMovie	
MÉTODO	FINALIDAD
<code>add</code>	Añade un objeto a la película actual.
<code>nextFrame</code>	Pasa al siguiente fotograma de la animación.
<code>output</code>	Envía la película al navegador.
<code>remove</code>	Elimina un objeto de la película actual.
<code>save</code>	Graba la película en un fichero de disco.
<code>setBackground</code>	Establece el color de fondo de la película.
<code>setDimensions</code>	Establece el tamaño de la película.
<code> setFrame</code>	Establece el número total de fotogramas en la animación completa.
<code>setRate</code>	Establece la velocidad de reproducción.
<code>streammp3</code>	Determina un fichero de sonido (formato .mp3) como fondo de la película. Este método no es, según algunas fuentes, muy estable.

*Figura 20.9*

### 20.2.10 La clase SWFShape

Esta clase se implementa para crear formas geométricas. A tal fin, cuenta con un total de diez métodos, cada uno de ellos con una funcionalidad específica con las que debe familiarizarse, debido a la importancia de esta clase. Los métodos se listan en la tabla de la figura 20.10.

MÉTODO DE LA CLASE SWFShape	
MÉTODO	FINALIDAD
<code>addFill</code>	Añade un relleno a una forma geométrica, según las condiciones que se establezcan.
<code>drawCircle</code>	Dibuja una circunferencia.
<code>drawCubic</code>	Dibuja una forma cúbica.
<code>drawCurve</code>	Dibuja una curva hasta un punto relativo al actual.
<code>drawCurveTo</code>	Dibuja una curva desde el punto actual hasta unas coordenadas establecidas.
<code>drawLine</code>	Dibuja una línea desde el punto actual hasta un punto relativo al mismo.
<code>drawLineTo</code>	Dibuja una línea desde el punto actual hasta unas coordenadas establecidas.
<code>movePen</code>	Mueve la pluma de dibujo desde el punto actual hasta un punto relativo al mismo.
<code>movePenTo</code>	Mueve la pluma de dibujo a unas coordenadas establecidas.
<code>setLeftFill</code>	Establece el relleno a izquierdas.
<code>setLine</code>	Establece las propiedades de una línea.
<code>setRightFill</code>	Establece el relleno a derechas.

*Figura 20.10*

### 20.2.11 La clase SWFSprite

Esta clase se usa para crear pequeños clips secundarios que se incluirán en la película principal. Son algo así como los famosos sprites (de ahí el nombre de la clase) que se usaban en las primeras animaciones populares de los años ochenta. Tiene cuatro métodos que vemos listados en la figura 20.11.

MÉTODO DE LA CLASE SWFSprite	
MÉTODO	FINALIDAD
<code>add</code>	Añade un objeto al mini clip.
<code>nextFrame</code>	Pasa al siguiente fotograma.
<code>remove</code>	Elimina un objeto del mini clip.
<code>setFrames</code>	Establece el número de fotogramas del mini clip. A más fotogramas, más calidad del sprite, pero también más peso específico.

*Figura 20.11*

## 20.2.12 La clase SWFText

Esta clase se usa para crear objetos de texto para incluir en nuestras películas. Algunas fuentes aseguran que bajo servidores Windows puede no ser totalmente estable, pero que no da problemas bajo servidores Linux. A causa de ello, nosotros no la usaremos aquí. Los códigos que se incluyen en el CD (como todos los scripts del libro) han sido probados en ambas plataformas, funcionando sin problemas. Cuando usted use esta clase, si registra algún tipo de anomalía, sustitúyala por SWFTextField en su código y se evita problemas de la plataforma. Cuenta con los siete métodos que aparecen en la tabla de la figura 20.12.

MÉTODO DE LA CLASE SWFText	
MÉTODO	FINALIDAD
<code>addString</code>	Añade una cadena al objeto de texto.
<code>getWidth</code>	Devuelve el ancho de la cadena de texto.
<code>moveTo</code>	Mueve la pluma (el cursor) a unas coordenadas en el espacio del objeto de texto.
<code>setColor</code>	Establece el color del texto.
<code>setFont</code>	Establece la fuente para el texto.
<code>setHeight</code>	Establece la altura de la fuente.
<code>setSpacing</code>	Establece el espaciado entre letras.

Figura 20.12

## 20.2.13 La clase SWFTextField

Diseñada también para crear objetos de texto, y de más moderna factura que la anterior, implementa un total de doce métodos, como ve en la tabla de la figura 20.13.

MÉTODO DE LA CLASE SWFTextField	
MÉTODO	FINALIDAD
<code>addString</code>	Añade una cadena al objeto de texto.
<code>align</code>	Establece la alineación del texto.
<code>setBounds</code>	Establece la anchura y altura del texto.
<code>setColor</code>	Establece el color del texto.
<code>SetFont</code>	Establece la fuente para el texto.
<code>setHeight</code>	Establece la altura del texto.
<code>setIndentation</code>	Establece el sangrado.
<code>setLeftMargin</code>	Establece el margen izquierdo.

MÉTODO DE LA CLASE SWFTextField (cont.)	
MÉTODO	FINALIDAD
<b>setLineSpacing</b>	Establece el espaciado entre líneas.
<b>setMargins</b>	Establece los márgenes.
<b>setName</b>	Establece un nombre para el objeto de texto, a efectos de su manejo mediante ActionScript.
<b>setRightMargin</b>	Establece el margen derecho.

Figura 20.13

### 20.2.14 Las medidas

La librería MING trabaja con unas unidades de medida determinadas. Para los giros y curvaturas se usan los grados sexagesimales (circunferencia de 360º). Para los tamaños, desplazamientos, etc., se emplean los twips. En su primera versión, implementada en PHP 4.0.5, se empleaban los twips. La equivalencia es de veinte twips por pixel. Es posible que, en algún manual o referencia, usted encuentre, todavía, medidas expresadas en twips. Nosotros, cuando hablemos de medidas radiales usaremos los grados. Cuando nos refiramos a medidas lineales hablaremos, indistintamente, de unidades o de píxeles.

### 20.2.15 Comprobando la librería

Lo primero que vamos a hacer es ejecutar un script que comprueba si la librería MING está disponible. En caso de que no lo esté, dicho script se ocupará de cargarla dinámicamente. El listado, disponible en el disco bajo el nombre **comprobarMING.php**, es el siguiente:

```
<?php
    if(!extension_loaded('ming')){
        echo ("La librería MING no se encuentra
cargada.<br>");
        if(strtoupper(substr(PHP_OS, 0, 3)) == 'WIN'){
            dl('php_ming.dll') or die("No se pudo cargar la
librería MING.");
        } else {
            dl('php_ming.so') or die("No se pudo cargar la
librería MING.");
        }
        echo ("La librería MING ha sido cargada
dinámicamente.");
    } else {
```

```
echo ("La librería MING ya está cargada  
previamente.");  
}?>
```

En primer lugar se emplea la función *extension\_loaded()*. Ésta recibe, como argumento, el nombre de una librería de PHP. Si dicha librería se encuentra cargada, devuelve un valor booleano true. En caso contrario, el resultado es false.

En caso de que la librería no se encuentre cargada, se muestra un mensaje informando de ello. A continuación, se tiene que detectar el sistema operativo del servidor, a fin de determinar el nombre del archivo que contiene la librería que nos interesa. Recuerde que, en instalaciones Windows, este archivo tiene la terminación .dll, mientras que en sistemas Linux el archivo termina en .so. Para averiguar cuál es el sistema operativo de la plataforma servidora, usamos la constante del lenguaje *PHP\_OS*. Si el valor de dicha constante contiene la secuencia **WIN**, estaremos trabajando con alguna versión de Windows. En caso contrario, asumiremos que el sistema es Linux o alguna otra instalación derivada de Unix.

Una vez determinado el sistema operativo (y, en consecuencia, el nombre del archivo de la librería MING), procedemos a su carga dinámica, es decir, en tiempo de ejecución, mediante el uso de la función *dl()*. Ésta recibe, como argumento, el nombre del fichero de librería que necesitamos y, si es posible, lo carga. Digo lo de "si es posible" porque pueden pasar muchas cosas. Un fichero inexistente o dañado haría inviable la carga. Además, si la directiva *safe\_mode* del fichero de configuración tiene el valor On (por defecto el valor es Off), la carga dinámica de la librería no podrá llevarse a cabo. De hecho, activar esta directiva limita muchas de las funcionalidades del lenguaje. En aras de la seguridad, se recorta la operatividad. Si no es estrictamente necesario, no la active.

Si todo ha ido correctamente, el script informa de que se ha cargado la librería. Si ya estaba cargada de antes, también se obtiene un mensaje informando de ello. Este script puede resultar útil para la carga dinámica de una librería en un momento dado. Pero si lo que desea, como en este caso, es tener la librería disponible para trabajar con ella de un modo más o menos regular, mejor cárguela como se menciona en el apartado 20.1. De hecho el uso de *dl()* se considera obsoleto en el actual PHP 5, aunque la función sigue siendo operativa.

## 20.3 NUESTRO PRIMER EJEMPLO

Una vez pasada la fase preliminar de las secciones anteriores, y dando por sentado que usted ya dispone de la librería MING cargada por defecto, vamos a

proceder a crear una primera película de Flash desde PHP. Será un ejemplo muy simple, sin ninguna finalidad práctica, pero con el valor didáctico de empezar a sentar conceptos útiles.

Observe el listado del script **fondoAzul.php**, reproducido a continuación:

```
<?php
/* Creamos un nuevo objeto de la clase SWFMovie. */
$pelicula = new SWFMovie();
/* Establecemos las dimensiones de la película. */
$pelicula->setDimension(300, 150);
/* Establecemos la velocidad de reproducción. */
$pelicula->setRate(15);
/* Establecemos el color de fondo. */
$pelicula->setBackground(0, 0, 255);
/* Le indicamos al navegador el tipo de contenidos. */
header("Content-type:application/x-shockwave-flash");
/* Enviamos la película al navegador. */
$pelicula->output();
?>
```

Lo primero que tenemos que hacer es instanciar la clase **SWFMovie**, es decir, crear un objeto de dicha clase. El objeto se llamará **\$pelicula** y se crea con la siguiente línea:

```
$pelicula = new SWFMovie();
```

Como ve, usamos el operador **new**, tal como hacemos siempre que queremos crear un objeto a partir de una clase. Este objeto será nuestra película de Flash, en formato **.swf**. Siempre que vayamos a hacer una animación de este tipo será necesario crearla de este modo.

A continuación, estableceremos el ancho y el alto de la película, mediante el método **setDimension()**. Éste recibe los argumentos expresados en píxeles: primero la anchura y después la altura, tal como se ve en el código.

El siguiente paso es establecer la velocidad de reproducción de nuestra animación. Las películas de Flash, al igual que las del cine, están formadas por una secuencia de fotogramas que se reproducen uno a continuación de otro. Mediante el método **setRate()**, le indicamos a PHP a qué velocidad se tienen que suceder dichos fotogramas, pasándole, como argumento, a dicho método el número de fotogramas que se deben reproducir por segundo. Una velocidad muy baja hará que la animación se vea a saltos. Un argumento muy alto puede sobrecargar el navegador, alterando la continuidad de la película. En el cine, la velocidad estándar

es de 24 *FPS (Frames Per Second, Fotogramas Por Segundo)*. En animaciones Flash se suele poner una velocidad de entre 12 y 50 fps. En nuestro ejemplo hemos fijado 15. Quiero llamar su atención sobre el hecho de que, en este caso concreto, no tiene mucho sentido, ya que nuestra primera película sólo tiene un fotograma. Sin embargo, así aprendemos a usar el método.

Ahora debemos establecer el color de fondo de la película. Para ello, empleamos el método *setBackground ()*, que recibe tres argumentos que representan los valores de rojo, verde y azul del color deseado (en este orden). En nuestro ejemplo, creamos un fondo de color azul puro.

El método *output ()* no recibe ningún argumento. Se emplea para enviar la película al navegador. Sin embargo, antes de que eso suceda, es necesario enviar una cabecera indicando que se va a mandar una animación de tipo swf. Eso lo haremos con la siguiente línea:

```
header("Content-type:application/x-shockwave-flash");
```

Preste especial atención a esto. En ningún caso se puede enviar nada al navegador antes de esta cabecera. Si se envía cualquier contenido, aunque sea un salto de línea o un espacio en blanco, no se podrá enviar la cabecera y, por tanto, tampoco se podrá visualizar la película. En realidad, la línea de cabecera que acabamos de ver puede colocarse al principio del código del script, aunque sólo sea por clarificar el listado. Funcionalmente nos da lo mismo.

Además, surge otro problema. Cargue el script en el navegador. Verá que la película (el fondo azul) no respeta las dimensiones de 300 píxeles de ancho y 150 de alto que establecimos con el método *setDimension ()*. Esto se debe a que, si no se limita específicamente, el vídeo swf se redimensiona de forma dinámica, ocupando todo el espacio disponible en el navegador. La solución pasa por cargar el script desde una página HTML, insertado como objeto, en lugar de cargarlo directamente. El listado de dicha página, al que hemos llamado **fondoAzul.htm** no tiene ningún secreto, como se ve a continuación:

```
<html>
  <head>
    <title>Página de fondo azul</title>
  </head>

  <body>
    <object classid="clsid:D27CDB6E-AE6D-11cf-96B8-
444553540000"
      codebase="http://download.macromedia.com/pub/shockwave/cabs/f
lash/swflash.cab#version=7,0,0,0" width="300" height="150">
```

```
<param name="movie" value="fondoAzul.php">
<embed src="fondoAzul.php"
pluginspage="http://www.macromedia.com/shockwave/download/ind
ex.cgi?P1_Prod_Version=ShockwaveFlash" type="application/x-
shockwave-flash" width="300" height="150">
</embed>
</object>
</body>
</html>
```

Lo único en lo que debemos reparar es en los atributos resaltados en las etiquetas <param> y <embed>. Vea que, en lugar de cargar directamente una animación swf lo que cargamos es el fichero PHP que genera dinámicamente dicha animación. Esta técnica es similar a la que empleábamos en el capítulo 12 cargando scripts que generaban imágenes. De este modo, además de asegurarnos de que se respetan las dimensiones de la película, podemos incluir, si lo deseamos, otros contenidos en la página.

Y ya tenemos una página en la que se visualiza correctamente una película de Flash creada mediante PHP. A partir de ahora, montaremos nuestros escenarios de ejemplo siguiendo esta técnica: por una parte, tendremos un script que genere una animación swf y por otra, tendremos una página HTML para su visualización.

## 20.4 AÑADIENDO FORMAS

Ya sabemos cómo crear la base de una película. Pero sin añadir elementos a la misma, poco sentido tiene. En esta sección aprenderemos a crear distintas formas geométricas en nuestras animaciones, con distintas propiedades, colores, rellenos, etc.

Todas las formas geométricas que se puedan crear en una película Flash se formarán a partir de dos tipos de líneas: rectas y curvas. Uniendo varias de estas líneas (de uno u otro tipo, o de ambos) lograremos toda clase de formas, tales como polígonos (de cualquier número de lados), cuadrados, etc. Por supuesto, para crear líneas (o cualquier otro elemento que queramos añadir a la película) deberemos contar con la base de la misma, tal como aprendimos a crearla en la sección anterior.

### 20.4.1 Creando líneas rectas

Las líneas se crean a partir de la clase SWFShape, que también forma parte de la librería MING. Para dibujar una línea, debemos empezar por crear un objeto

de esta clase, usando el constructor de la misma, tal como se muestra en la siguiente línea de código:

```
$linea_1 = new SWFShape();
```

Este fragmento de código creará un objeto llamado **\$linea\_1**, que será el que contenga la línea que queremos trazar.

A continuación, debemos definir la linea (la forma en que se visualizará en la película) mediante tres propiedades básicas: su grosor, su color y su transparencia. Esta última se conoce, en los entornos gráficos, como *factor alfa*. Estas tres propiedades se definen mediante el método *setLine()*, que recibe los cinco argumentos siguientes:

- Grosor de la linea, expresado, por supuesto, en píxeles.
- Valor de rojo del color elegido.
- Valor de verde del color elegido.
- Valor de azul del color elegido.
- Factor alfa. Éste será un valor de 0 a 255, siendo 0 una transparencia total (la linea no se verá) y 255 la total opacidad.

Por ejemplo, suponga que deseamos definir el objeto **\$linea\_1** como una linea de 10 píxeles de ancho, amarilla y totalmente opaca. La siguiente instrucción lo hará:

```
$linea_1->setLine(10, 255, 255, 0, 255);
```

Ya sabemos cómo queremos que sea la linea. Ahora viene el momento de especificar el punto en el que se empezará a dibujar. Para ello, recurrimos al método *movePenTo()*, que recibe dos argumentos: las coordenadas x e y del punto inicial de la linea. Estas coordenadas se toman con respecto a la esquina superior izquierda del área de la animación, cuyas coordenadas son 0, 0. Por ejemplo, podríamos tener la siguiente linea de código para fijar el punto inicial de dibujo:

```
$linea_1->movePenTo(150, 50);
```

En este caso, dicho punto estaría a 150 píxeles a la derecha del borde izquierdo de la película y a 50 píxeles por debajo del borde superior de la misma. El siguiente paso es indicar hasta dónde se debe dibujar la linea. Para ello contamos con dos métodos. Elegiremos uno u otro según una diferencia de funcionamiento que comentaremos en seguida. El primero de los métodos de dibujo de rectas es *drawLine()* y el segundo es *drawLineTo()*. Ambos reciben dos argumentos que representan sendas coordenadas x e y. La diferencia que

mencionábamos es la siguiente: mientras el primer método dibuja tantos píxeles en horizontal y en vertical como se indique, el segundo dibuja hasta las coordenadas especificadas. Para entenderlo mejor, suponga que, después de la última línea de código que hemos visto (la que posicionaba el punto inicial de dibujo en 150, 50), escribimos lo siguiente:

```
$linea_1->drawLine(200, 150);
```

Esto hará que *la pluma* (como se llama en entornos gráficos a la herramienta que traza líneas) dibuje durante 200 píxeles en horizontal y 150 en vertical. Teniendo en cuenta las coordenadas de inicio, se dibujará hasta el punto 350, 200. En cambio, piense que usaremos el segundo método de dibujo, así:

```
$linea_1->drawLineTo(200, 150);
```

Esto haría que la pluma dibujara hasta las coordenadas 200, 150. Es decir, se dibujaría desplazando la pluma 50 píxeles en horizontal y 100 en vertical con respecto al punto de origen.

Dicho de otro modo: las coordenadas que recibe como argumentos el método drawLine () son relativas al punto de origen de la líneas, mientras que las que recibe el método drawLineTo () lo son respecto a la esquina superior izquierda de la animación. Por eso, en el primer caso se habla de *coordenadas relativas* y en el segundo, de *coordenadas absolutas*. También se dice que el primer método define una trayectoria y el segundo, un destino.

Con los métodos que acabamos de ver, ya hemos definido completamente una línea recta. Sin embargo, ésta no se visualizará, aún, en la película. Siempre que se defina un elemento (una línea, un texto, un botón, etc.) es necesario añadirlo a la película. Para ello, usamos el método add (), del objeto de la clase SWFMovie. El argumento que recibe este método es el nombre del objeto que contiene el elemento que deseamos añadir; en nuestro caso, la línea. Por lo tanto, en nuestro código incluiremos lo siguiente:

```
$pelicula->add($linea_1);
```

Y, ahora sí, ya podemos enviar la película swf al navegador mediante el uso del método output (). El script **unaLinea.php** dibuja una línea recta en la animación. Su listado es el siguiente:

```
<?php  
/* Se prepara la "base" de la película, como aprendimos  
a hacerlo en el script anterior. */
```

```
$pelicula = new SWFMovie();
$pelicula->setDimension(500, 350);
$pelicula->setRate(15);
$pelicula->setBackground(0, 0, 255);

/* Se crea un nuevo objeto de la clase SWFShape, para
que contenga una línea.*/
$linea_1 = new SWFShape();

/* Se define el grosor, el color y el factor alfa de la
línea.*/
$linea_1->setLine(10, 255, 255, 0, 255);

/* Se define el punto de origen.*/
$linea_1->movePenTo(150, 50);
/* Se define la trayectoria.*/
$linea_1->drawLine(200, 150);

/* Se añade la línea a la película.*/
$pelicula->add($linea_1);

/* Se envía la película al navegador.*/
header("Content-type:application/x-shockwave-flash");
$pelicula->output();
?>
```

Como en el caso anterior, no cargaremos este script directamente en el navegador, sino que lo haremos mediante una página HTML. El listado, llamado **unaLinea.htm** es igual que en el caso anterior, cambiando, únicamente, el nombre del script que le pasamos al objeto incrustado y las dimensiones que reservamos, así:

```
<object classid="clsid:D27CDB6E-AE6D-11cf-96B8-
444553540000" codebase=
"http://download.macromedia.com/pub/shockwave/cabs/flash/swf1
ash.cab#version=7,0,0,0" width="500" height="350">
<param name="movie" value="unaLinea.php">
<embed src="unaLinea.php"
pluginspage="http://www.macromedia.com/shockwave/download/ind
ex.cgi?P1_Prod_Version=ShockwaveFlash" type="application/x-
shockwave-flash" width="500" height="350">
</embed>
</object>
```

Al cargar esta página en el navegador veremos un resultado como el de la figura 20.14.

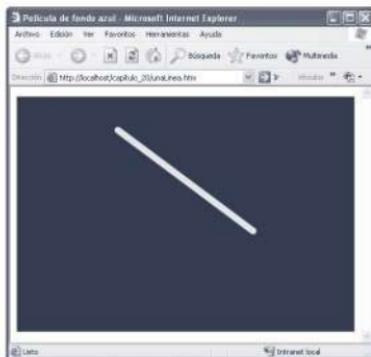


Figura 20.14

En el siguiente ejemplo vamos a poner dos líneas cruzadas en la película, cada una de ellas con un color distinto. A la que cruce por encima le daremos una cierta transparencia, a fin de que pueda comprobar el funcionamiento del factor alfa (quinto parámetro del método setLine () que se usa para definir la línea). El código se llama **dosLineas.php**, y su listado es el siguiente:

```
<?php
/* Se prepara la "base" de la película, como aprendimos
a hacerlo en el script anterior. */
$pelicula = new SWFMovie();
$pelicula->setDimension(500, 350);
$pelicula->setRate(15);
$pelicula->setBackground(0, 0, 255);
/* Se crea una línea, como aprendimos a hacerlo en el
script anterior. */
$linea_1 = new SWFShape();
$linea_1->setLine(50, 255, 255, 0, 255); //Ancho, 50,
para que se vea bien.
$linea_1->movePenTo(150, 50);
$linea_1->drawLine(200, 150);

/* Se crea otra línea. */
$linea_2 = new SWFShape();
$linea_2->setLine(50, 255, 0, 0, 127); //Alfa=127
(semitransparente).
$linea_2->movePenTo(450, 50);
$linea_2->drawLineTo(150, 300); //Uso de drawLineTo()
() .
```

```
/* Se añaden las líneas a la película. */
$pelicula->add($linea_1);
$pelicula->add($linea_2);

/* Se envía la película al navegador. */
header("Content-type:application/x-shockwave-flash");
$pelicula->output();
?>
```

Observe, en la primera línea resaltada, cómo hemos definido un valor alfa de sólo 127 para una línea de color rojo.

Fíjese en el segundo bloque resaltado, donde las líneas definidas anteriormente se añaden a la película, mediante el método add(). La última línea que se añade será la que quede “por encima”. En este caso es la que hemos llamado \$linea\_2 que tiene cierta transparencia. El resultado es que el punto donde se cruzan tiene el aspecto de la figura 20.15.



Figura 20.15

Para verlo, cargue en el navegador la página dosLineas.htm, como ya va siendo habitual, y podrá apreciar el efecto.

#### 20.4.2 Creando curvas

Dibujar curvas en un lienzo es casi tan simple como dibujar rectas. En realidad sólo es necesario definir un punto adicional. Al igual que ocurre en muchos entornos gráficos, una curva se define por un total de tres puntos, que se conocen como *origen, anclaje y control*. En la figura 20.16 puede verlos en un esquema.

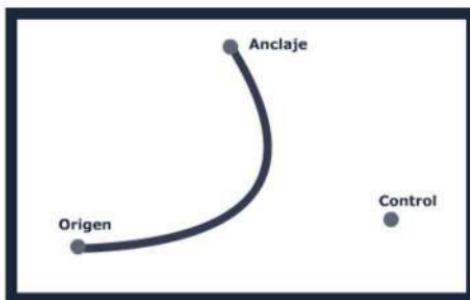


Figura 20.16

Modificando la posición de cualquiera de estos puntos se redefine la curva. Se llama punto de origen al comienzo de la curva. El anclaje es el punto donde finaliza. Por último, el punto de control sirve para determinar la curvatura de la línea. Observe que éste último “tira” de la curva pero siempre queda fuera de ella. Es decir, la *crestas de la curva* nunca alcanza el punto de control.

Para definir una curva, al igual que para definir una recta, necesitamos empezar creando un objeto de la clase SWFShape. A continuación, usaremos el método `setLine()` para establecer el grosor, el color y el factor alfa del trazo. Después, usaremos `movePenTo()` para fijar el punto de origen. Hasta aquí, todo es igual que en el caso de las líneas rectas. Para dibujar una curva, contamos con dos métodos a elegir: se trata de `drawCurve()` y `drawCurveTo()`. Cualquier que sea el método elegido, recibirá cuatro parámetros: los dos primeros corresponden a las coordenadas x e y del punto de control; los dos últimos se refieren a las coordenadas x e y del punto de anclaje. Siempre en este orden: primero se define el control y luego el anclaje.

De forma equivalente a como ocurre con las líneas rectas, si empleamos el método `drawCurve()`, se entiende que las coordenadas son a contar desde el punto que hayamos fijado como origen de la línea con `movePenTo()`. Si empleamos `drawCurveTo()`, las coordenadas se consideran con respecto al punto 0, 0 (esquina superior izquierda) de la base de la película.

A modo de ejemplo, vamos a ver el listado del script **curva.php**. Empezaremos, como siempre, creando el objeto que será la película, con un fondo de unas dimensiones y un color. Después crearemos el objeto de la clase SWFShape que contendrá la curva. Luego añadiremos la curva a la película y, por último, enviaremos ésta al navegador. El listado del script es el siguiente:

```
<?php
/* Se empieza definiendo la base de la película, como
ya es habitual. */
$pelicula = new SWFMovie();
$pelicula->setDimension(500, 350);
$pelicula->setRate(15);
$pelicula->setBackground(0, 0, 255);

/* Se crea un objeto para definir la curva */
$curva = new SWFShape();
$curva ->setLine(10, 255, 0, 0, 255);
$curva ->movePenTo(50, 300);
/* Se define el punto de control y el de anclaje. */
$curva ->drawCurveTo(400, 250, 200, 60);

/* Se añade la curva a la película. */
$pelicula->add($curva);

/* Se envía la película al navegador. */
header("Content-type:application/x-shockwave-flash");
$pelicula->output();
?>
```

El resultado de este script aparece en la figura 20.17.

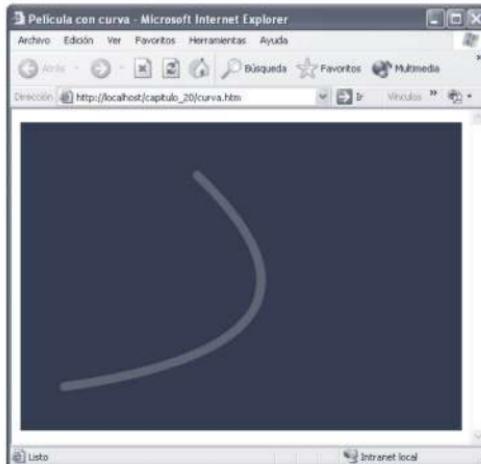


Figura 20.17

Repare en cómo hemos usado el método drawCurveTo () en la línea resaltada del listado.

Antes de seguir adelante debo hacerle una observación. Los métodos movePenTo (), drawLine (), drawLineTo (), drawCurve () y drawCurveTo () aceptan valores negativos como coordenadas. Pero tenga cuidado. Si una parte de la línea, ya sea ésta recta o curva, cae fuera del área de la película, sólo será visible la parte que esté dentro. Ésto es válido no sólo para los métodos ya vistos, sino para cualquier otro que se refiera a posicionamiento de objetos en la película.

### 20.4.3 Figuras cerradas

Ahora que ya sabemos cómo dibujar líneas rectas o curvas en nuestra película, viene el siguiente paso. Crear formas cerradas. Esto es muy fácil. Si queremos, por ejemplo, un rectángulo, dibujaremos las cuatro líneas necesarias para los lados. Si queremos un pentágono, las líneas serán cinco. También podemos combinar líneas rectas y curvas para llevar a cabo diversas figuras.

Empezaremos dibujando un cuadrado. Observe el listado **cuadrado.php**:

```
<?php
    /* Primero se crea la base de la película, como
    siempre. */
    $pelicula = new SWFMovie();
    $pelicula->setDimension(400, 350);
    $pelicula->setRate(15);
    $pelicula->setBackground(0, 0, 255);

    /* Se crea un objeto de línea para dibujar los cuatro
    lados */
    $ladosDeCuadrado = new SWFShape();
    $ladosDeCuadrado->setLine(5, 255, 255, 0, 255);
    /* Se posiciona la pluma en la esquina superior
    izquierda del cuadrado a dibujar. */
    $ladosDeCuadrado->movePenTo(50, 50);
    /* Se dibuja el lado izquierdo. */
    $ladosDeCuadrado->drawLineTo(50, 300);
    /* Se dibuja el lado inferior. */
    $ladosDeCuadrado->drawLineTo(300, 300);
    /* Se dibuja el lado derecho. */
    $ladosDeCuadrado->drawLineTo(300, 50);
    /* Se dibuja el lado superior. */
    $ladosDeCuadrado->drawLineTo(50, 50);
    /* Se añade la línea, con los cuatro lados dibujados, a
    la película. */
```

```
$pelicula->add($ladosDeCuadrado);
/* Se envía la película al navegador. */
header("Content-type:application/x-shockwave-flash");
$pelicula->output();
?>
```

Vea las líneas resaltadas. Lo que hacemos es crear un objeto de la clase SWFShape, como ya es habitual, y hacerle trazar cuatro líneas, dibujando hasta las distintas coordenadas. Al ejecutar **cuadrado.htm** obtenemos el resultado de la figura 20.18.

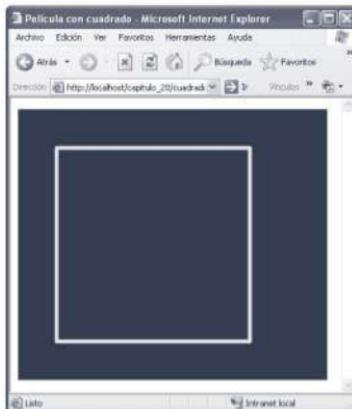


Figura 20.18

Eficiente, ¿verdad? Dése cuenta de una cosa. Cuando se define una línea, empezamos moviendo la pluma, con el método `movePenTo()`, hasta la posición inicial. Después realizamos un trazo (ya sea recto o curvo). En ese momento, la pluma queda situada donde termina dicho trazo, de modo que el siguiente empieza desde ese punto. Ahora vamos a buscar una forma más sofisticada. Estará constituida por una combinación de líneas curvas y rectas. Vea el script **formaMixta.php**:

```
<?php
/* Primero se crea la base de la película, como
siempre. */
$pelicula = new SWFMovie();
$pelicula->setDimension(400, 350);
$pelicula->setRate(15);
```

```
$pelicula->setBackground(0, 0, 255);
/* Se crea un objeto de línea para dibujar los cuatro
lados. */
$ladosDeForma = new SWFShape();
$ladosDeForma->setLine(5, 255, 255, 0, 255);
/* Se posiciona la pluma en la esquina superior
izquierda del cuadrado a dibujar. */
$ladosDeForma->movePenTo(50, 50);
/* Se dibuja el lado izquierdo (una curva). */
$ladosDeForma->drawCurveTo(200, 130, 50, 300);
/* Se dibuja el lado inferior (una recta). */
$ladosDeForma->drawLineTo(300, 300);
/* Se dibuja el lado derecho (una curva). */
$ladosDeForma->drawCurveTo(400, 140, 300, 50);
/* Se dibuja el lado superior (una recta). */
$ladosDeForma->drawLineTo(50, 50);
/* Se añade la línea, con los cuatro lados dibujados, a
la película. */
$pelicula->add($ladosDeForma);
/* Se envía la película al navegador. */
header("Content-type:application/x-shockwave-flash");
$pelicula->output();
?>
```

El resultado corresponde a la figura 20.19.

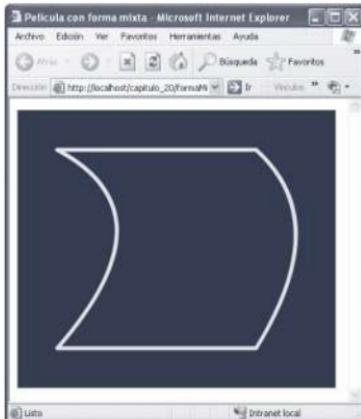


Figura 20.19

Observe en la parte resaltada del código cómo se van dibujando los lados de la forma, cada uno a partir de donde terminó el anterior. Como vé, puede formar cualquier figura que desee, con sólo saber determinar las coordenadas de los puntos que necesita. Más adelante veremos ejemplos más sofisticados.

## 20.5 RELLENANDO FIGURAS

Ahora que ya sabemos cómo crear formas cerradas con el contorno deseado, vamos a dotarlas de un relleno, para poder construir figuras completas. La librería MING nos ofrece tres modos de llenar una figura: con un color sólido, con un gradiente (degradado) entre dos colores o con una imagen. En esta sección conoceremos las tres posibilidades para que usted pueda usar las que considere necesarias.

### 20.5.1 Rellenos de color

Para llenar una figura con un color sólido, debemos dar dos pasos adicionales a la creación de la misma. En primer lugar, definiremos el relleno como una variable creada mediante el método *addFill()* de la clase SWFShape. Este método recibe cuatro argumentos. Los tres primeros representan los valores de rojo, verde y azul del color que deseamos para llenar nuestra figura. El cuarto parámetro es opcional y representa el factor alfa. Si lo omitimos, se toma el valor por defecto (255) que implica opacidad total. En definitiva, la definición del color del relleno podría quedar así:

```
$relleno=$linea->addFill ($rojo, $verde, $azul, $alfa);
```

En este ejemplo genérico, la variable \$linea es el objeto de la clase SWFShape que hemos creado anteriormente para definir una figura cerrada.

Una vez hecho esto, hemos de establecer el relleno en la figura. Para ello contamos con dos métodos: *setLeftFill()* y *setRightFill()*, ambos de la clase SWFShape. Estos métodos (el que elijamos en cada caso) se aplican a partir del objeto de la clase SWFShape que será la linea de contorno de la figura. Cualquiera de estos métodos recibe, como argumento, la variable que obtuvimos anteriormente como resultado del uso de addFill(). Siguiendo con nuestro ejemplo, podríamos tener una línea de código como la siguiente:

```
$linea->setRightFill ($relleno);
```

Y, ¿cuál de los dos métodos usaremos y por qué? La respuesta a esa pregunta depende de cómo se vaya a trazar el contorno de la figura. Suponga que

ésta será un cuadrado. Pues bien. Depende de que los lados los vayamos a trazar en el sentido de las agujas del reloj o en sentido contrario. Si empezamos por trazar el lado superior de izquierda a derecha, luego el derecho de arriba a abajo, después el inferior de derecha a izquierda y, por último, el izquierdo de abajo a arriba, la figura se habrá trazado en sentido de las agujas del reloj, es decir, hacia la derecha. En ese caso, para rellenarla, usaremos el método `setRightFill()`. Sin embargo, si trazamos primero el lado izquierdo de arriba abajo, luego el inferior de izquierda a derecha, después el derecho de abajo a arriba y, para finalizar, el superior de derecha a izquierda, el contorno se habrá dibujado hacia la izquierda y, para rellenarlo, usaremos `setLeftFill()`. Observe la figura 20.20, en la que se representa, esquemáticamente, la diferencia de trazado en ambos casos, con una reseña del método que deberemos emplear en cada caso.

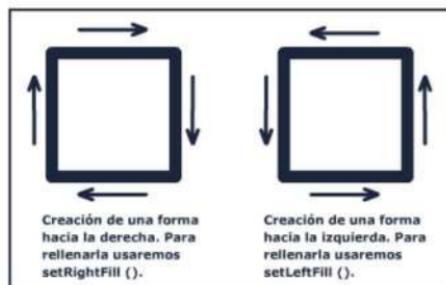


Figura 20.20

En mi experiencia, el uso de un método u otro con colores sólidos no tiene influencia sobre el resultado. Sin embargo, las especificaciones de la librería MING nos dicen que debemos usar los métodos según el criterio expuesto, y así lo haremos.

Vamos, como siempre, a ver un código de ejemplo, para materializar ideas. El listado se llama **rellenoColorSolido.php**:

```
<?php
/* Primero se crea la base de la película. */
$pelicula = new SWFMovie();
$pelicula->setDimension(400, 350);
$pelicula->setRate(15);
$pelicula->setBackground(0, 0, 255);
/* Se crea un objeto de línea para dibujar los cuatro
lados de un cuadrado. */
```

```
$cuadrado = new SWFShape();
$cuadrado->setLine(5, 255, 255, 0, 255);
/* Se define un relleno rojo, sin transparencia. */
$rellenoRojo=$cuadrado->addFill(255, 0, 0, 255);

/* Se asocia el relleno a la línea que definirá el
contorno. */
$cuadrado->setLeftFill ($rellenoRojo);
/* Se posiciona la pluma en la esquina superior
izquierda del cuadrado a dibujar. */
$cuadrado->movePenTo(50, 50);
$cuadrado->drawLineTo(50, 300);
$cuadrado->drawLineTo(300, 300);
$cuadrado->drawLineTo(300, 50);
$cuadrado->drawLineTo(50, 50);
/* Se añade la línea, con los cuatro lados dibujados, a
la película. */
$pelicula->add($cuadrado);

/* Se envía la película al navegador. */
header("Content-type:application/x-shockwave-flash");
$pelicula->output();
?>
```

El resultado de ejecutar la página **rellenoColorSolido.htm**, que carga este script, es el que aparece en la figura 20.21.

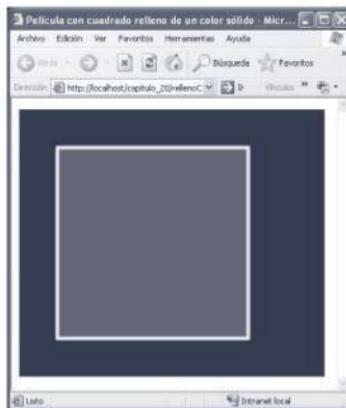


Figura 20.21

Observe el bloque resaltado en el listado del script. Fíjese en que, cuando definimos el relleno, todavía no se ha especificado la trayectoria de la línea, de modo que el intérprete de PHP no "sabe" aún qué forma se va a llevar a cabo con dicha línea. Como ve, el uso de estos métodos es muy flexible. Todas las formas cerradas que se crean con la línea especificada se llenarán con el color elegido. Sin embargo, debe recordar una cosa: atendiendo a lo que hemos mencionado anteriormente, si se crean más formas cerradas deberá dibujar el contorno "a izquierdas". Esto es bastante interesante. Con sólo definir una línea y un relleno, podemos dibujar en nuestro lienzo todas las formas que deseemos. Todas ellas tendrán, por supuesto, el mismo contorno y el mismo relleno. Observe el listado del script **dosFormasRellenas.php**:

```
<?php
/* Primero se crea la base de la película, como
siempre. */
$pelicula = new SWFMovie();
$pelicula->setDimension(400, 350);
$pelicula->setRate(15);
$pelicula->setBackground(0, 0, 255);

/* Se crea un objeto de línea para dibujar los cuatro
lados de un cuadrado. */
$cuadrado = new SWFShape();
$cuadrado->setLine(5, 255, 255, 0, 255);

/* Se define un relleno de color rojo, sin
transparencia. */
$rellenoRojo=$cuadrado->addFill(255, 0, 0, 255);
/* Se asocia el relleno a la línea que definirá el
contorno. */
$cuadrado->setLeftFill ($rellenoRojo);

/* Se posiciona la pluma en la esquina superior
izquierda del primer cuadrado a dibujar. */
$cuadrado->movePenTo(50, 50);
$cuadrado->drawLineTo(50, 150);
$cuadrado->drawLineTo(150, 150);
$cuadrado->drawLineTo(150, 50);
$cuadrado->drawLineTo(50, 50);
/* Se posiciona la pluma en la esquina superior
izquierda del segundo cuadrado a dibujar. */
$cuadrado->movePenTo(200, 200);
$cuadrado->drawLineTo(200, 300);
$cuadrado->drawLineTo(300, 300);
$cuadrado->drawLineTo(300, 200);
$cuadrado->drawLineTo(200, 200);
```

```
/* Se añade la linea, con los dos cuadrados dibujados,
a la película. */
$pelicula->add($cuadrado);
/* Se envía la película al navegador. */
header("Content-type:application/x-shockwave-flash");
$pelicula->output();
?>
```

Vea que, con el mismo objeto (**\$cuadrado**) se definen dos formas, en dos ubicaciones distintas del área de trabajo. El resultado aparece en la figura 20.22.

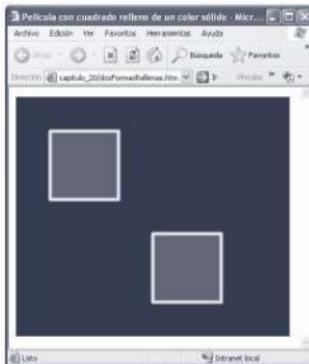


Figura 20.22

Si quiere dibujar otras figuras en el mismo lienzo con diferentes propiedades puede hacerlo creando más objetos de la clase SWFShape, tal como vemos en el listado **diversasFormas.php**:

```
<?php
/* Primero se crea la base de la película, como
siempre. */
$pelicula = new SWFMovie();
$pelicula->setDimension(400, 350);
$pelicula->setRate(15);
$pelicula->setBackground(0, 0, 255);

/* Se crea un objeto de línea para dibujar los cuatro
lados de un cuadrado. */
$cuadrado = new SWFShape();
$cuadrado->setLine(5, 255, 255, 0, 255);
```

```
/* Se crea otro objeto para definir otras formas. */
$triangulo = new SWFShape();
$triangulo->setLine(5, 0, 255, 0, 255);

/* Se define un relleno de color rojo, sin
transparencia. */
$rellenoRojo=$cuadrado->addFill(255, 0, 0, 255);
/* Se asocia el relleno a la línea que definirá el
contorno. */
$cuadrado->setLeftFill ($rellenoRojo);

/* Se posiciona la pluma en la esquina superior
izquierda del primer cuadrado a dibujar. */
$cuadrado->movePenTo(50, 50);
$cuadrado->drawLineTo(50, 150);
$cuadrado->drawLineTo(150, 150);
$cuadrado->drawLineTo(150, 50);
$cuadrado->drawLineTo(50, 50);

/* Se posiciona la pluma en la esquina superior
izquierda del segundo cuadrado a dibujar. */
$cuadrado->movePenTo(200, 200);
$cuadrado->drawLineTo(200, 300);
$cuadrado->drawLineTo(300, 300);
$cuadrado->drawLineTo(300, 200);
$cuadrado->drawLineTo(200, 200);

/* Se posiciona la pluma en el vértice superior del
triángulo. */
$triangulo->movePenTo(75, 200);
$triangulo->drawLineTo(50, 250);
$triangulo->drawLineTo(100, 250);
$triangulo->drawLineTo(75, 200);

/* Se añade la línea, con los dos cuadrados dibujados,
a la película. */
$pelicula->add($cuadrado);
$pelicula->add($triangulo);

/* Se envía la película al navegador. */
header("Content-type:application/x-shockwave-flash");
$pelicula->output();
?>
```

Observe los dos bloques resaltados, donde se define, en primer lugar, un objeto de la clase SWFShape para crear otras formas. En el segundo bloque vemos cómo se define un triángulo. Por último, observe que, además de añadir el objeto

de los dos cuadrados, también añadimos el del triángulo a la película. Y aquí ya aparece una pauta común para el desarrollo de aplicaciones con la librería MING. Esto lo mencionamos brevemente al principio del capítulo, pero éste es el momento de insistir en ello. Siempre empezaremos por crear una base para la película, a partir de la clase **SWFMovie**. Ésta será el lienzo sobre el que dibujemos. Después definiremos los objetos que queremos incorporar a nuestro escenario. Hasta ahora estamos usando para ello la clase **SWFShape**, pero también usaremos objetos de otras clases. Por último, cada uno de los objetos creados se añaden a la película con el método **add()** y se muestra el resultado con **output()**.

Al cargar el listado anterior a través de **diversasFormas.htm** obtendremos el resultado de la figura 20.23.

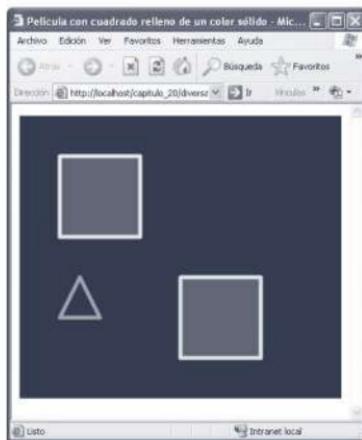


Figura 20.23

### 20.5.2 Rellenos con gradientes

A menudo, el relleno de una forma con un color sólido no es suficiente para obtener el resultado visual que deseamos. La librería MING, siguiendo con su línea de implementar funcionalidades propias de Flash, permite diseñar degradados entre colores, a través de la clase **SWFGradient**. Esta clase implementa un único método (aparte, naturalmente, del constructor). Se trata de **addEntry()**, que se usará para añadir distintos colores al degradado. Un gradiente siempre se forma entre dos o más colores, pasando de uno a otro con suavidad.

Para crear un gradiente necesitamos, en primer lugar, instanciar la clase SWFGradient en un objeto, que será sobre el que luego añadimos las tonalidades. Para ello, usaremos el constructor, sin argumentos, tal como se aprecia en la línea de ejemplo siguiente:

```
$degradado=new SWFGradient();
```

La variable **\$degradado** es el objeto que va a contener el gradiente. Sin embargo, todavía no se ha definido ningún color para el mismo y, como hemos comentado, será necesario definir, al menos, dos. Para ello, usamos el único método que tiene esta clase y que es, como ya hemos comentado, addEntry(). Éste recibe nada menos que cinco argumentos, a saber:

- Ratio. Es un valor entre 0.0 y 1.0 y que indica la zona del gradiente donde se inicia el color que vamos a añadir. Dicho así quizás suene un poco críptico, pero lo aclararemos al ver unos ejemplos concretos (en este mismo apartado).
- Rojo. El valor de rojo del color deseado. Estará entre 0 y 255, como ya es habitual.
- Verde. El valor de verde del color deseado, entre 0 y 255.
- Azul. El valor de azul del color deseado, entre 0 y 255.
- Alfa. El factor alfa (grado de transparencia respecto al fondo) del color deseado, entre 9 y 255.

Por ejemplo, si queremos definir el degradado anterior entre los colores verde y rojo totalmente opacos (sin transparencias), emplearemos las siguientes líneas de código:

```
$degradado->addEntry(0.0, 0, 255, 0, 255);  
$degradado->addEntry(0.5, 255, 0, 0, 255);
```

A continuación, deberemos usar, de nuevo, el método addFill() de la clase SWFShape. Sin embargo, en este caso su sintaxis es un poco diferente. En lugar de recibir como argumentos las cifras que definen un color para el relleno, como hacemos con los tonos sólidos, le pasaremos dos argumentos. El primero será, como seguramente usted ya ha imaginado, el nombre del objeto que hemos creado para el gradiente. El segundo argumento es una constante específica que indica el tipo de gradiente. Ésta puede ser una de las siguientes:

- **SWFFILL\_RADIAL\_GRADIENT**. Se usa para gradientes radiales, es decir, desde un centro hacia el exterior, en círculos.
- **SWFFILL\_LINEAR\_GRADIENT**. Este valor se refiere a gradientes lineales.

Suponga que va a trazar una forma con un objeto de la clase SWFShape, llamado **\$contorno**. Podríamos definir un relleno de gradiente, con el que ya tenemos, del siguiente modo:

```
$relleno = $contorno->addFill ($degradado,  
SWFFILL_RADIAL_GRADIENT);
```

Para terminar, asignaremos este relleno a la línea de contorno con setRightFill () o setLeftFill (), como hacemos con los rellenos sólidos.

Considere el script **rellenoGradienteSimple.php**, cuyo código aparece reproducido a continuación:

```
<?php  
/* Primero se crea la base de la película, como  
siempre. */  
$pelicula = new SWFMovie();  
$pelicula->setDimension(400, 350);  
$pelicula->setRate(15);  
$pelicula->setBackground(0, 0, 255);  
  
/* Se crea un objeto de línea para dibujar el contorno  
de una forma. */  
$forma = new SWFShape();  
$forma->setLine(5, 255, 255, 0, 255);  
  
/* Se crea un objeto de gradiente. */  
$degradado = new SWFGradient();  
/* Se le asignan dos colores al degradado: uno al  
principio del mismo y otro a la mitad. */  
$degradado->addEntry(0.0, 255, 0, 0, 255);  
$degradado->addEntry(0.5, 0, 255, 0, 255);  
  
/* Se define un relleno con el degradado, de forma  
radial. */  
$relleno=$forma->addFill($degradado,  
SWFFILL_RADIAL_GRADIENT);  
/* Se asocia el relleno a la línea que definirá el  
contorno. */  
$forma->setLeftFill ($relleno);  
  
/* Se posiciona la pluma en la esquina superior  
izquierda de la forma a dibujar. */  
$forma->movePenTo(50, 50);  
/* Se dibuja la forma hacia la izquierda. */  
$forma->drawLineTo(50, 300);
```

```

$forma->drawLineTo(300, 300);
$forma->drawLineTo(300, 50);
$forma->drawLineTo(50, 50);

/* Se añade la línea, con los cuatro lados dibujados, a
la película.*/
$pelicula->add($forma);
/* Se envía la película al navegador.*/
header("Content-type:application/x-shockwave-flash");
$pelicula->output();
?>

```

Al cargarlo mediante **rellenoGradienteSimple.htm** obtendremos un resultado similar al que aparece en la figura 20.24.

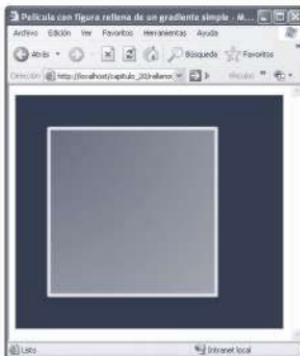


Figura 20.24

Como puede apreciar se ha logrado un degradado muy suave entre ambos tonos. Si modificamos los valores del argumento ratio (el primero) del método addEntry, podemos lograr degradados más marcados. Por ejemplo, sustituya los valores 0.0 y 0.5 por 0.4 y 0.6 respectivamente y observe el resultado.

### 20.5.3 Rellenos con imágenes

Una figura puede rellenarse con una imagen, de forma que podemos lograr un efecto bastante aparente. En la carpeta imágenes, dentro de la que corresponde a este capítulo, hay seis fotografías que podemos usar para experimentar este sistema de relleno. La librería MING incluye la posibilidad de usar imágenes en formato gif, en formato png y en jpg progresivo. El término “progresivo” se refiere a una

variante de jpg que manejan algunos programas y que genera imágenes que se cargan en los navegadores en varias fases, de modo que primero se ve una imagen de muy baja calidad, pero de carga muy rápida y posteriormente se van cargando, progresivamente (de ahí el nombre) partes de la imagen que no se cargaron al principio, mejorando la calidad de presentación. Nosotros, llegados a este punto, usaremos imágenes png, que admiten transparencias (cosa que el formato jpg no admite) y que tienen un peso aceptable para los anchos de banda actuales.

Para contar en nuestro script con una imagen que podamos usar para un relleno o para cualquier otra finalidad, lo primero que debemos hacer es abrir el fichero de imagen y leer su contenido, así:

```
$archivo="imagenes/andromeda.png";
$manejador=fopen ($archivo, "rb");
$datosDeImagen=fread($manejador, filesize($archivo));
```

A partir del contenido leído en el fichero, crearemos un nuevo objeto de la clase *SWFBitmap*, para tener la imagen disponible para su uso con la librería MING, del siguiente modo:

```
$mapaDeBits = new SWFBitmap ($datosDeImagen);
```

El objeto *\$mapaDeBits*, de la clase SWFBitmap, se pasa ahora como argumento al método addFill, de la clase SWFShape, para añadir esta imagen como relleno. En este caso, al igual que ocurre con los gradientes, el método addFill recibe un segundo parámetro, que determinará el modo en que la imagen rellenará el contorno que tracemos, tal como se ve en la siguiente lista:

- ***SWFFILL\_TILED\_BITMAP***. Hace que la imagen, si es más pequeña que el área a llenar, se repita en mosaico hasta cubrir dicho espacio.
- ***SWFFILL\_CLIPPED\_BITMAP***. Coloca la imagen en la figura una sola vez. Utilice este valor, exclusivamente, cuando el tamaño de la imagen coincida con el de la forma en la que se va a alojar. Si no es así, el resultado puede ser bastante chapucero.

Por ejemplo, podríamos escribir una línea de código como la siguiente:

```
$relleno = $forma->addFill ($mapaDeBits,
SWFFILL_TILED_BITMAP);
```

Por último, usaremos de nuevo el método setRightFill () o setLeftFill (), según proceda, para establecer el relleno, así:

```
$forma->setRightFill ($relleno);
```

Como siempre, veremos un código que ilustra esta técnica. El script se llama **rellenoImagen.php**:

```
<?php
/* Primero se crea la base de la película, como
siempre. */
$pelicula = new SWFMovie();
$pelicula->setDimension(400, 350);
$pelicula->setRate(15);
$pelicula->setBackground(0, 0, 255);
/* Se crea un objeto de línea para dibujar el contorno
de una forma. */
$forma = new SWFShape();
$forma->setLine(5, 255, 255, 0, 255);

/* Se abre un fichero de imagen, se lee y se cierra. */
$archivo="imagenes/crepusculo.png";
$manejador=fopen ($archivo, "rb");
$datosDeImagen=fread($manejador, filesize($archivo));
fclose ($manejador);

/* Se crea un objeto SWFBitmap con los datos leidos del
fichero de imagen. */
$mapaDeBits = new SWFBitmap ($datosDeImagen);

/* Se define un relleno con la imagen repetida. */
$relleno=$forma->addFill($mapaDeBits,
SWFFILL_TILED_BITMAP);

/* Se asocia el relleno a la línea que definirá el
contorno. */
$forma->setLeftFill ($relleno);
/* Se posiciona la pluma en la esquina superior
izquierda de la forma a dibujar. */
$forma->movePenTo(50, 50);
/* Se dibuja la forma hacia la izquierda. */
$forma->drawLineTo(50, 300);
$forma->drawLineTo(300, 300);
$forma->drawLineTo(300, 50);
$forma->drawLineTo(50, 50);
/* Se añade la línea, con los cuatro lados dibujados, a
la película. */
$pelicula->add($forma);
/* Se envía la película al navegador. */
header("Content-type:application/x-shockwave-flash");
```

```
$pelicula->output();  
?>
```

Observe, en el bloque resaltado, cómo se usa una imagen para crear un relleno. Vea cómo seguimos los pasos que hemos indicado. En primer lugar, se abre el fichero en modo de lectura de datos binarios ("rb"). Después se lee el contenido total del fichero y, una vez almacenado éste en una variable, se cierra. No olvide nunca cerrar un fichero que haya abierto, cuando ya no lo necesite.

A continuación, se usa la variable donde se almacenó el contenido del fichero para crear un objeto de la clase SWFBitmap y se emplea dicho objeto como argumento del método addFill(). Fíjese en que, de momento, hemos empleado, como segundo valor de dicho método, SWFFILL\_TILED\_BITMAP, para que la imagen se repita, llenando la forma que vamos a crear. El resultado se ve en la figura 20.25.

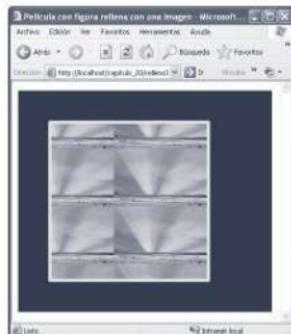


Figura 20.25

La clase SWFBitmap proporciona dos métodos muy útiles para trabajar con imágenes: se trata de `getWidth()` y `getHeight()`, que devuelven la anchura y la altura de la imagen en píxeles. Si queremos obtener estos valores, usaremos la siguiente sintaxis genérica:

```
$anchura = $mapaDeBits->getWidth();  
$altura = $mapaDeBits->getHeight();
```

En este ejemplo la variable `$mapaDeBits` es el objeto que hemos creado de la clase SWFBitmap.

## 20.6 TEXTO

Por supuesto, a nuestras películas de Flash también podemos añadirles texto que aparecerá en las mismas. Para ello, contamos con dos clases de la librería MING: *SWFText* y *SWFTextField*. De la primera dicen algunas fuentes que no funciona correctamente en servidores Windows. En mis experiencias esto es así, razón por la cual usaremos la segunda clase en este capítulo, olvidándonos de la primera. Además, para la implementación de textos en nuestra película usaremos la clase auxiliar *SWFFont*, que permite establecer una tipografía.

### 20.6.1 El texto más simple

La mejor forma de empezar es viendo un ejemplo de uso de texto. Mire el script **textoSimple.php**, donde hemos usado el mínimo de posibilidades, con el fin de iniciarnos de un modo muy fácil. El listado es el siguiente:

```
<?php
    /* Se prepara la "base" de la película, como ya es
    habitual. */
    $pelicula = new SWFMovie();
    $pelicula->setDimension(200, 100);
    $pelicula->setRate(15);
    $pelicula->setBackground(0, 0, 255);
    /* Se crea un objeto de texto (clase SWFTextField). */
    $texto_1 = new SWFTextField();
    /* Creamos un objeto para la tipografía a partir de la
    clase SWFFont. */
    $fuente = new SWFFont("Verdana");
    /* Asociamos la tipografía al texto. */
    $texto_1->setFont ($fuente);

    /* Establecemos el color del texto. */
    $texto_1->setColor (255, 255, 0);
    /* Creamos una cadena para el objeto de texto. */
    $texto_1->addString ("Hola, mundo.");
    /* Se añade el objeto de texto a la película. */
    $pelicula->add($texto_1);
    /* Se envía la película al navegador. */
    header("Content-type:application/x-shockwave-flash");
    $pelicula->output();
?>
```

Observe el bloque resaltado. En primer lugar, creamos un objeto de la clase *SWFTextField*, usando el constructor de la misma, para tener en él el texto que pondremos en la película.

Después, creamos un objeto para contener una tipografía. Estos objetos se crean a partir de la clase auxiliar SWFFont. Al constructor le pasamos, como argumento, el nombre de la tipografía que deseamos usar. No debería tener que decirlo, pero, naturalmente, el archivo de dicha tipografía debe estar disponible en el ordenador.

El siguiente paso es asignarle el objeto de tipografía al objeto de texto. Esto lo hacemos con el método *setFont()* del objeto de la clase SWFTextField. Este método recibe, como argumento, el nombre del objeto de la clase SWFFont que contiene la fuente elegida.

El siguiente paso es establecer el color del texto, mediante el uso del método *setColor()*. Éste recibe tres argumentos que representan los valores de rojo, verde y azul del color elegido.

Lo siguiente que debemos hacer es decidir qué texto se mostrará y asignárselo al objeto. Esto lo hacemos mediante el método *addString()*, que recibe, como argumento, la cadena que deseamos que se incluya en el texto.

Y ya está listo el objeto para ser añadido a la película. Esto lo hacemos con el método *add()* de la clase SWFMovie, tal como hacemos con las formas. Por último, como ya es habitual, enviamos la película al navegador. El resultado aparece en la figura 20.26.



Figura 20.26

## 20.6.2 Algunas mejoras

En el ejemplo de texto anterior hemos visto cómo se puede añadir un rótulo a la película de Flash. Sin embargo, esta inserción es muy pobre. Con el tamaño estándar de la letra, y directamente colocada en la esquina superior izquierda de la película, no dice mucho de las capacidades de la clase SWFTextField. A continuación, haremos uso de tres métodos de esta clase para mejorar un poco la presentación.

En primer lugar, conoceremos el método *setHeight()*, que permite fijar la altura de la letra a tantos pixeles como indiquemos en el argumento. Naturalmente, la anchura de la tipografía queda también modificada, a fin de mantener correctamente las proporciones.

El siguiente método interesante es *setBounds()*. Éste permite crear un marco invisible en el que encasillar el texto. Como argumentos recibe la anchura y la altura de dicho marco. Si el texto, con el tamaño de fuente que hayamos establecido, excede el tamaño del marco, la parte que quede fuera del mismo no será visible, aún cuando caiga dentro del lienzo de la película. Esta característica nos será útil cuando hagamos animaciones. Aunque el texto se puede colocar directamente en la animación sin el uso de este método, para lograr mejores resultados, en más de un sentido, deberemos usarlo.

El último método del que vamos a hablar aquí es *align()*. Permite establecer una alineación del texto, según el argumento que reciba, que será uno de los cuatro siguientes:

- *SWFTEXTFIELD\_ALIGN\_LEFT*. Establece alineación izquierda.
- *SWFTEXTFIELD\_ALIGN\_CENTER*. Establece alineación centrada.
- *SWFTEXTFIELD\_ALIGN\_RIGHT*. Establece alineación derecha.
- *SWFTEXTFIELD\_ALIGN\_JUSTIFY*. Establece alineación justificada (izquierda y derecha, simultáneamente).

Veamos un ejemplo, para reafirmar conceptos. El script se llama **textoPosicionado.php**:

```
<?php
/* Se prepara la "base" de la película, como ya es
habitual. */
$pelicula = new SWFMovie();
$pelicula->setDimension(200, 100);
$pelicula->setRate(15);
$pelicula->setBackground(0, 0, 255);
```

```
/* Se crea un objeto de texto, con una tipografía,  
color y cadena, como en el ejemplo anterior. */  
$texto_1 = new SWFTextField();  
$fuente = new SWFFont("Verdana");  
$texto_1->setFont ($fuente);  
$texto_1->setColor (255, 255, 0);  
$texto_1->addString ("Hola, mundo.");  
  
/* Se establece la altura de la tipografía (La anchura  
se auto ajusta para mantener las proporciones). */  
$texto_1->setHeight (20);  
/* Se establece un marco para contener el texto. */  
$texto_1->setBounds (150, 80);  
/* Se alinea el texto en el marco. */  
$texto_1->align(SWFTEXTFIELD_ALIGN_CENTER);  
  
/* Se añade el objeto de texto a la película. */  
$pelicula->add($texto_1);  
/* Se envía la película al navegador. */  
header("Content-type:application/x-shockwave-flash");  
$pelicula->output();  
?>
```

Vea el bloque resaltado. En él se han usado estos tres métodos para establecer un tamaño de fuente, crear el marco y alinear el texto dentro del marco. Vea el resultado en la figura 20.27.



Figura 20.27

Como ve, el resultado ha mejorado con respecto a la muestra anterior. Sin embargo, hay algo que llama la atención. Donde usamos el método align ()

establecemos una alineación centrada. Sin embargo, el texto aparece, casi, a la izquierda del lienzo. Aclaremos cosas: este método alinea el texto *con respecto al marco creado con setBounds ()*, no *con respecto al lienzo de la película*.

Los tres métodos que hemos visto en este apartado pertenecen a la clase SWFTextField.

### 20.6.3 Colocando el texto

Con los recursos recién comentados podemos lograr, ya, un rudimentario posicionamiento del texto en la película, pero aún se echa de menos poder colocarlo en unas coordenadas determinadas. Si estuviéramos usando para el texto la clase SWFText sería fácil, ya que cuenta con el método moveTo () para posicionar un texto. Sin embargo, ya sabemos que no podemos usar esta clase, ya que nuestro script debe funcionar igual en un servidor Linux que sobre una plataforma Windows. Así pues, debemos buscar otra solución. Vamos a recurrir a una técnica que, la primera vez que se ve, parece un poco rebuscada; sin embargo, veremos que es muy cómoda y la usaremos muy a menudo. En el momento de añadir el objeto de texto a la película, en lugar de hacerlo "por las buenas", como hemos venido haciendo hasta ahora, asignaremos el resultado de esa operación a un manejador. Después, ya podremos usar el método *moveTo ()* para colocarlo en el lienzo. Este método colocará el marco creado con *setBounds ()* que, a su vez, contiene la cadena asignada con *addString ()*. El método recibe las coordenadas x e y donde se colocará la esquina superior izquierda del marco con respecto al lienzo de trabajo.

Veamos un ejemplo en el script **colocarTexto.php**:

```
<?php
/* Se prepara la "base" de la película, como ya es
habitual. */
$pelicula = new SWFMovie();
$pelicula->setDimension(200, 100);
$pelicula->setRate(15);
$pelicula->setBackground(0, 0, 255);

/* Se crea un objeto de texto, con sus características,
como en el ejemplo anterior. */
$texto_1 = new SWFTextField();
$fuente = new SWFFont("Verdana");
$texto_1->setFont ($fuente);
$texto_1->setColor (255, 255, 0);
$texto_1->addString ("Hola, mundo.");
$texto_1->setHeight (20);
```

```
$texto_1->setBounds (150, 80);
$texto_1->align(SWFTEXTFIELD_ALIGN_CENTER);
/* Se añade el objeto de texto a la película,
ALMACENANDO EL RESULTADO EN UN MANEJADOR. */
$manejadorTexto=$pelicula->add($texto_1);
/* Se mueve el objeto a una posición. */
$manejadorTexto->moveTo(50,50);/* Se envía la
película al navegador. */
header("Content-type:application/x-shockwave-flash");
$pelicula->output();
?>
```

Una vez más, repare en la parte resaltada del código. Todo lo anterior ya lo conocemos de los ejemplos que hemos ido viendo.

Vea cómo añadimos el objeto \$texto\_1 a la película con el método *add ()*... y le asignamos el resultado de esta operación a un manejador. A continuación, le aplicamos a éste el método *moveTo ()*, dándole unas coordenadas de posicionamiento definitivo. Como alternativa podríamos haber usado el método *move ()*, que recibe el desplazamiento relativo desde la posición actual, tanto en el eje de x como en el de y. La diferencia entre estos dos métodos es similar a la que ya conocemos entre, por ejemplo, *drawLine ()* y *drawLineTo ()*.

El resultado de cargar este script a través de **colocarTexto.htm**, será el de la figura 20.28.



Figura 20.28

Ésta es una técnica que usaremos muy a menudo. En realidad, el resultado del método *add ()* genera un objeto de la clase ***SWFDisplayItem***, que es el que usamos como manejador. Esta clase está concebida, específicamente, para poder

efectuar desplazamientos, rotaciones, etc. Más adelante veremos, mediante otros ejemplos de scripts, más posibilidades para el texto y para otros objetos, derivadas, precisamente, de la clase `SWFDisplayItem`. Sin embargo, aquí ya le hemos encontrado un uso magnífico.

## 20.7 ANIMACIONES

Hasta ahora, hemos creado scripts que generan perfectamente películas de Flash en formato swf. Sin embargo, todas ellas adolecen de una limitación importante: sólo tienen un fotograma. Casi nada. En lugar de animaciones hemos creado fotos fijas. En esta sección vamos a aprender a añadirle más fotogramas a nuestra película, de modo que, realmente, lo sea. Para ello, la clase `SWFMovie` nos proporciona el método `nextFrame()`. Éste no recibe argumento alguno. Simplemente, genera el siguiente fotograma, en el que podremos cambiar la apariencia de la película. Como ve, ya tenemos clara la idea. En definitiva, una animación es una secuencia de fotos fijas (cada fotograma es una de ellas) que se suceden a una velocidad determinada. Esta velocidad ya la establecimos al crear la base de la película usando el método `setRate()`.

### 20.7.1 Una animación simple

Cuando se trabaja en entornos de Flash, se habla de dos tipos de movimiento en una película. Éstos son los siguientes:

- **Interpolación de forma.** Es cuando un objeto cambia su forma, tamaño o color de un fotograma a otro.
- **Interpolación de movimiento.** Es cuando un objeto cambia su posición en el escenario de un fotograma a otro.

Ruego a aquellos lectores que conocen la herramienta Flash que me perdonen la imprecisión e inexactitud en la definición de las interpolaciones. He tenido que adaptar este concepto al modo de trabajo con PHP que, si bien emula el comportamiento de Flash, a nivel de desarrollo presenta algunas diferencias.

Vamos a empezar por desarrollar una interpolación de movimiento muy simple. Crearemos una película que presentará un cuadrado. Éste se desplazará desde el lado izquierdo del lienzo, al derecho. Para hacer más simple la figura y poder centrarnos en la forma en que se crea el desplazamiento, nuestro cuadrado no tendrá ni siquiera un relleno. Observe el listado de `movimientoSimple.php`:

```
<?php  
/* Primero se crea la base de la película. */
```

```
$pelicula = new SWFMovie();
$pelicula->setDimension(400, 60);
$pelicula->setRate(30);
$pelicula->setBackground(0, 0, 255);
/* Se crea un objeto de línea con cuatro lados */
$ladosDeCuadrado = new SWFShape();
$ladosDeCuadrado->setLine(5, 255, 255, 0, 255);
$ladosDeCuadrado->movePenTo(5, 5);
$ladosDeCuadrado->drawLineTo(5, 55);
$ladosDeCuadrado->drawLineTo(55, 55);
$ladosDeCuadrado->drawLineTo(55, 5);
$ladosDeCuadrado->drawLineTo(5, 5);
/* Se añade la línea, con los cuatro lados dibujados, a
la película. Se crea un manejador de la clase SWFDisplayItem,
para poder mover la figura. */
$manejadorDeCuadrado=$pelicula-
>add($ladosDeCuadrado);
/* El bucle determinará el movimiento. */
for ($posicion=5;$posicion<340;$posicion+=2){
    $pelicula->nextFrame(); // Siguiente fotograma.
    $manejadorDeCuadrado->moveTo ($posicion, 0);
//Movimiento de la figura.
}
/* Se envía la película al navegador. */
header("Content-type:application/x-shockwave-flash");
$pelicula->output();
?>
```

Al cargar la página **movimientoSimple.htm**, en el navegador verá algo similar a la figura 20.29.

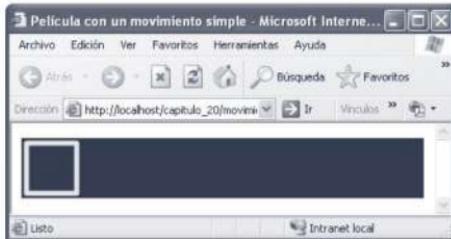


Figura 20.29

Sin embargo, esta vez algo ha cambiado. El cuadradito de color amarillo se mueve desde el lado izquierdo del lienzo, donde se sitúa inicialmente, hasta el lado

derecho del mismo. Cuando alcanza el final, vuelve a iniciar el movimiento desde la izquierda.

Veamos qué hemos hecho. Observe el fragmento resaltado del script. La parte anterior es, simplemente, para crear la base de la película y dibujar el cuadrado. No tienen nada nuevo. Pero, cuando llega el momento de añadir la forma a la película, en lugar de usar el método add () sin más, le hemos asignado el resultado de dicha operación a un manejador que es creado, automáticamente, como un objeto de la clase SWFDisplayItem. Esto es lo mismo que hemos hecho al final de la sección anterior dedicada al texto. La clase SWFDisplayItem, permítame recordárselo, se usa para proveer movimientos, rotaciones, escalados, etc., a los objetos. Cada objeto que vayamos a necesitar “vivo” en una película, debe ser asociado a la clase SWFDisplayItem mediante un manejador en el momento de añadirlo a la animación, tal como hemos visto.

Sin embargo, todavía tenemos sólo un fotograma. Nuestra “animación” es, todavía, una simple foto fija. Y observe el bucle. Vamos a usarlo para incluir, en su interior, el método nextFrame (). En cada iteración del bucle se generará un nuevo fotograma. Ya sólo falta que, en cada nuevo fotograma, el objeto (el cuadradito amarillo) se desplace un poco, situándose en una nueva posición. Para ello, no trabajaremos directamente sobre nuestra figura, sino que lo haremos a través de su manejador. La clase SWFDisplayItem proporciona, como usted ya sabe, el método moveTo (), que recibe dos parámetros: las coordenadas x e y de la posición a la que hay que mover el objeto. La coordenada y es fija, en este ejemplo, ya que no queremos movimiento vertical. Para el movimiento horizontal ponemos, como coordenada x, la variable de control del bucle.

Con esto logramos que la figura se desplace a una nueva posición en cada nueva iteración del bucle (lo que implica un nuevo fotograma). Sin embargo, aún hay algo que no parece estar previsto. Por el límite que hemos impuesto a la variable de control en el bucle, al llegar a la derecha el cuadradito debería detenerse. Sin embargo, la animación se repite indefinidamente, volviendo a empezar desde el comienzo. Éste es el comportamiento por defecto de las películas de Flash, que PHP imita. Si queremos que la animación se ejecute una sola vez tendremos que recurrir a las acciones, un tema del que hablaremos más adelante en otra sección de este mismo capítulo.

## 20.7.2 Técnicas profesionales

Hasta ahora hemos dibujado algunas formas de un modo muy elemental y hemos aplicado “por encima” técnicas que necesitábamos conocer. Llegados a este punto, es el momento de empezar a depurar y mejorar nuestro trabajo para irlo

dotando, cada vez, de mayor profesionalidad y obtener, poco a poco, mejores resultados. En este apartado vamos a conocer un par de detalles que hasta el momento no habíamos tocado, pero que nos serán muy útiles a la hora de crear una determinada película.

En primer lugar, quiero hablarle del sistema de coordenadas cartesianas (x, y). No sé si ha reparado en ello, pero todo lo que hace, cuando trabaja con una figura que ha creado para su película, es relativo a la esquina superior izquierda de la película o de la figura con la que trabaja. Piense, por ejemplo, en los rellenos que hacemos con gradientes (apartado 20.5.2). Observe de nuevo la figura 20.24, en la que se le aplica a una forma un relleno con un gradiente radial. Fíjese en que el gradiente parece ser parte de un círculo (para eso es radial) y da la impresión de que el centro de dicho círculo (el origen del gradiente) estuviera en la esquina superior izquierda del recuadro. Parecida situación tiene lugar cuando, por ejemplo, se hace rotar una figura. Si creamos un cuadrado y lo hacemos girar en la animación, el centro de rotación estará en la esquina superior izquierda del lienzo. Vamos a verlo. Observe el script **rotarEsquina.php**:

```
<?php
    /* Primero se crea la base de la película, como
    siempre. */
    $pelicula = new SWFMovie();
    $pelicula->setDimension(400, 400);
    $pelicula->setRate(15);
    $pelicula->setBackground(0, 0, 255);
    /* Se crea un objeto de línea para dibujar los cuatro
    lados */
    $ladosDeCuadrado = new SWFShape();
    $ladosDeCuadrado->setLine(3, 255, 255, 0, 255);
    $ladosDeCuadrado->movePenTo(180, 180);
    $ladosDeCuadrado->drawLineTo(180, 220);
    $ladosDeCuadrado->drawLineTo(220, 220);
    $ladosDeCuadrado->drawLineTo(220, 180);
    $ladosDeCuadrado->drawLineTo(180, 180);

    /* Se añade la línea, con los cuatro lados dibujados, a
    la película. Se crea un manejador de la clase SWFDisplayItem,
    para poder rotar la figura. */
    $manejadorDeCuadrado=$pelicula-
>add($ladosDeCuadrado);

    /* El bucle determinará la rotación. */
    for ($posicion=0;$posicion<100;$posicion++){
        $pelicula->nextFrame(); // Siguiente fotograma.
```

```
$manejadorDeCuadrado->rotate (10); //Movimiento de
la figura.
}
/* Se envía la película al navegador. */
header("Content-type:application/x-shockwave-flash");
$pelicula->output();
?>
```

Fíjese en la línea resaltada. Es similar a la que usábamos antes para mover el cuadradito de un lado a otro del lienzo, sólo que en este caso estamos usando el método *rotate ()*, de la clase SWFDisplayItem, para provocar un giro. Este método recibe, como argumento, el número de grados que hay que girar. Si el argumento es positivo, el giro es hacia la izquierda; si es negativo, es hacia la derecha. Hecho este pequeño inciso, uno puede esperar que, puesto que hemos dibujado el cuadradito en el centro del lienzo, permanezca allí y gire sobre su propio centro. Pero esto no es así. El giro se produce sobre la esquina superior izquierda de la película. Cargue el script en su navegador a través de la página **rotarEsquina.htm** y lo comprobará.

Si queremos que la rotación se produzca, como es de esperar, sobre el centro geométrico de nuestra figura, la solución pasa por dos fases. En primer lugar, tenemos que aprender a considerar la posibilidad de usar valores negativos en las coordenadas de nuestras figuras. Así pues, si vamos a crear un cuadrado de 40 píxeles de lado, al usar el método *movePenTo ()* para fijar el origen, los argumentos no serán (0, 0), sino (-20, -20). De este modo, el punto (0, 0) no quedará alineado con una esquina, sino en el centro de la figura.

Pero aún hay más. La librería MING cuenta con la clase *SWFSprite*, que no hemos empleado hasta el momento. Ésta tiene unos pocos métodos comunes con la clase SWFMovie. Su finalidad es crear mini-clips, también llamados *sprites*. Éstos son unas sub-películas, con una funcionalidad muy limitada, que luego incorporaremos a la película principal, de modo que se reproducen dentro de ésta con una cierta autonomía. Esto es mucho más potente y útil de lo que pueda parecer en principio. Observe el script **rotacionSimple.php**:

```
<?php
/* Se crea la base de la película. */
$peliculaPrincipal = new SWFMovie();
$peliculaPrincipal->setBackground(0, 100, 255);
$peliculaPrincipal->setDimension(200, 200);
$peliculaPrincipal->setRate(30);

/* Se crea un cuadrado: bordes negros y relleno rojo.
*/
```

```
$cuadrado = new SWFShape();
$cuadrado->setLine(2, 0, 0, 0, 255);
$relleno = $cuadrado->addFill(255, 0, 0);
$cuadrado->setRightFill($relleno);
$cuadrado->movePenTo(-50, -50);
$cuadrado->drawLineTo(50, -50);
$cuadrado->drawLineTo(50, 50);
$cuadrado->drawLineTo(-50, 50);
$cuadrado->drawLineTo(-50, -50);

/* Se crea un miniclip que contendrá el cuadrado. */
$clip = new SWFSprite();
/* Se añade el cuadrado al miniclip, generando un
manejador que es un objeto de la clase SWFDisplayItem. */
$manejadorDeClip = $clip->add($cuadrado);
/* Por medio del manejador se hace que el cuadrado gire
en el miniclip. */
for ($pasos=0; $pasos<9; $pasos++){
    $clip->nextFrame();
    $manejadorDeClip->rotate(-10);
}
/* Se añade el miniclip a la película principal. */
$manejadorEnPeliculaPrincipal = $peliculaPrincipal-
>add($clip);
/* Se sitúa el miniclip en el centro del lienzo. */
$manejadorEnPeliculaPrincipal->moveTo(100, 100);
header('Content-type: application/x-shockwave-
flash');
$peliculaPrincipal->output();
?>
```

En primer lugar, cárguelo en el navegador, a través de la página **rotacionSimple.htm**, para ver cómo funciona. Se encontrará con un clip de película de 200 píxeles de ancho por otros tantos de alto, con una base de color azul. En el centro aparece un cuadrado de borde negro y relleno rojo, de 100 píxeles de altura y otros 100 de anchura. Esta figura aparece rotando sobre su centro (que, además, hemos hecho coincidir con el centro de la película; esto no necesariamente tiene por qué ser así).

La parte del script que crea la base de la película y el cuadrilátero ya no tiene ningún secreto. Lo único sobre lo que quiero llamar su atención a este respecto es el hecho de que, en la creación de la forma, se han usado coordenadas negativas y positivas, de modo que el punto (0, 0) de la misma está en su centro geométrico, y no en una esquina. Ahora observemos la parte resaltada. Aquí vemos materializado lo que hemos estado explicando, de forma teórica, en este apartado. En primer lugar, creamos un mini-clip, de la clase SWFSprite, y le añadimos la

figura, obteniendo, así, un manejador de la clase SWFDisplayItem. Esto lo hacemos con las siguientes líneas:

```
$clip = new SWFSprite();
$manejadorDeClip = $clip->add($cuadrado);
```

Este manejador se usa, en un bucle, para rotar el mini-clip, diez grados a la derecha en cada nuevo fotograma (cada iteración del bucle), según las líneas que vienen a continuación:

```
for ($pasos=0; $pasos<9; $pasos++){
    $clip->nextFrame();
    $manejadorDeClip->rotate(-10);
}
```

El mini-clip ya está operando, de forma autónoma. La figura ya gira. Ahora sólo nos falta añadir esto a la película principal (ya que de otro modo no lo veremos) y posicionarlo en el centro de la misma, así:

```
$manejadorEnPeliculaPrincipal = $peliculaPrincipal-
>add($clip);
$manejadorEnPeliculaPrincipal->moveTo(100, 100);
```

Y ya tenemos nuestro objetivo conseguido. La técnica del uso de sprites es mucho más potente que lo que acabamos de ver aquí. Cuando tenemos una película con varios elementos, es mucho mejor, incluso si no lo consideramos necesario, crear un sprite para cada forma, figura, texto, etc. Así, si cambiamos algo en un elemento, no afectará al resto. No obstante, habrá casos puntuales en que deberá colocar un elemento, directamente, en la película principal.

El tema de las coordenadas negativas para construir una figura podemos usarlo, como decíamos al principio de este apartado, para cambiar el aspecto de un relleno de gradiente. Lo que perseguimos es que el origen del mismo esté en el centro geométrico de la figura. Pero esta vez no vamos, sin embargo, a usar un sprite, ya que no va a haber movimiento en la película. Vea el script **rellenoGradienteCentrado.php**:

```
<?php
/* Primero se crea la base de la película, como hacemos
siempre. */
$pelicula = new SWFMovie();
$pelicula->setDimension(200, 200);
$pelicula->setRate(15);
$pelicula->setBackground(0, 0, 255);
```

```
/* Se crea un objeto de línea para dibujar el contorno
de una forma. */
$forma = new SWFShape();
$forma->setLine(5, 255, 255, 0, 255);

/* Se crea un objeto de gradiente. */
$degradado = new SWFGradient();
/* Se le asignan dos colores al degradado. */
$degradado->addEntry(0.0, 255, 0, 0, 255);
$degradado->addEntry(0.07, 0, 255, 0, 255);

/* Se define un relleno con el degradado, de forma
radial. */
$relleno=$forma->addFill($degradado,
SWFFILL_RADIAL_GRADIENT);
/* Se asocia el relleno a la línea que definirá el
contorno. */
$forma->setLeftFill ($relleno);

/* Se posiciona la pluma en la esquina superior
izquierda de la forma a dibujar. */
$forma->movePenTo(-50, -50);
/* Se dibuja la forma hacia la izquierda. */
$forma->drawLineTo(-50, 50);
$forma->drawLineTo(50, 50);
$forma->drawLineTo(50, -50);
$forma->drawLineTo(-50, -50);

/* Se añade la línea, con los cuatro lados dibujados, a
la película. Se incluye un manejador para poder posicionarla.
*/
$manejadorDeFigura = $pelicula->add($forma);
$manejadorDeFigura->moveTo(100, 100);

/* Se envía la película al navegador. */
header("Content-type:application/x-shockwave-flash");
$pelicula->output();
?>
```

Preste atención a los dos bloques resaltados. En el primero se construye la figura usando un sistema de coordenadas que incluye valores negativos y positivos, de modo que, una vez más, la posición (0, 0) está en el centro y no en una esquina.

En el segundo bloque resaltado se añade, directamente, la forma sobre la película. Aunque aquí podríamos haber usado un sprite y, pese a no ser

estrictamente necesario, hubiera quedado más profesional, he querido omitirlo por claridad.

Cuando cargamos en el navegador este script, a través de la página **rellenoGradienteCentrado.htm** obtenemos el resultado de la figura 20.30.



Figura 20.30

Fíjese en la diferencia con respecto al gradiente que obteníamos en la figura 20.24. Esta vez el círculo del degradado se origina en el centro geométrico de la forma. Ahora vamos a ver una variación sobre este tema, usando un sprite (esta vez, sí), para lograr el mismo resultado. En seguida entenderá por qué. Observe el listado **rellenoGradienteSprite.php**:

```
<?php
/* Primero se crea la base de la película. */
$pelicula = new SWFMovie();
$pelicula->setDimension(200, 200);
$pelicula->setRate(15);
$pelicula->setBackground(0, 0, 255);
/* Se crea un objeto de línea para dibujar el contorno
de una forma. */
$forma = new SWFShape();
$forma->setLine(5, 255, 255, 0, 255);
/* Se crea un objeto de gradiente. */
$degradado = new SWFGradient();
```

```
/* Se le asignan dos colores al degradado. */
$degradado->addEntry(0.0, 255, 0, 0, 255);
$degradado->addEntry(0.1, 0, 255, 0, 255);
/* Se define un relleno con el degradado radial. */
$relleno=$forma->addFill($degradado,
SWFFILL_RADIAL_GRADIENT);
/* Se asocia el relleno al contorno. */
$forma->setLeftFill ($relleno);
/* Se posiciona la pluma en la esquina superior
izquierda de la forma a dibujar. */
$forma->movePenTo(-50, -50);
/* Se dibuja la forma hacia la izquierda. */
$forma->drawLineTo(-50, 50);
$forma->drawLineTo(50, 50);
$forma->drawLineTo(50, -50);
$forma->drawLineTo(-50, -50);

/* Se crea un sprite para alojar la forma. */
$clip = new SWFSprite();
$manejadorDeClip = $clip->add($forma);
/* Se crea el siguiente fotograma del clip.
!!!IMPRESINDIBLE!!!
$clip->nextFrame();

/* Se añade el clip a la película.
Se incluye un manejador para poder posicionarlo. */
$manejadorDeFigura = $pelicula->add($clip);
$manejadorDeFigura->moveTo(100, 100);
/* Se envía la película al navegador. */
header("Content-type:application/x-shockwave-flash");
$pelicula->output();
?>
```

Si cargamos la película en el navegador mediante el código de la página **rellenoGradienteSprite.htm**, verá que el resultado no difiere del anterior. He querido que vea este código sólo por una razón. Observe especialmente el fragmento resaltado, donde la forma se incorpora a un sprite. Vea la línea en la que se usa el método `nextFrame ()`. Este método pertenece a la clase `SWFSprite` (vea que se está aplicando sobre el mini-clip, que es un objeto de dicha clase) y su uso es el mismo que el de su homónimo de la clase `SWFMovie` que ya conocemos. Cuando se crea un sprite y se le asigna una figura para que forme parte de él, como en este caso, el uso de este método es imprescindible, incluso cuando no va a haber realmente una animación, como ocurre aquí. Si no usa este método, aunque añada el clip a la película principal, no será visible. Pruébelo. Elimine la línea que usa el método `nextFrame ()` del mini-clip y cargue de nuevo la página. Verá el fondo azul de la película, pero no el cuadrilátero relleno.

## 20.7.3 Escalados

Quizás sea el cambio de tamaño de un objeto una de las animaciones primarias más vistas en películas de Flash. El efecto es el de que dicho objeto se acerca al observador o se aleja de él. La clase SWFDisplayItem cuenta con dos métodos a este efecto: *scale()* y *scaleTo()*. En este apartado vamos a aprender a realizar escalados sobre un objeto, a través de la clase SWFSprite. Como veremos es bastante simple. Lo único que tenemos que hacer, una vez creado un sprite e incluida en él una forma, es utilizar el método adecuado sobre un manejador. En definitiva, lo mismo que venimos haciendo hasta ahora. Observe el listado **escaladoSimple.php**:

```
<?php
/* Primero se crea la base de la película, como
siempre. */
$pelicula = new SWFMovie();
$pelicula->setDimension(300, 300);
$pelicula->setRate(40);
$pelicula->setBackground(0, 0, 255);

/* Se crea un objeto de línea para dibujar los cuatro
lados de un cuadrado. */
$cuadrado = new SWFShape();
$cuadrado->setLine(5, 255, 255, 0, 255);

/* Se define un relleno de color rojo, sin
transparencia. */
$rellenoRojo=$cuadrado->addFill(255, 0, 0, 255);
/* Se asocia el relleno a la línea que definirá el
contorno. */
$cuadrado->setLeftFill ($rellenoRojo);

/* Se posiciona la pluma en la esquina superior
izquierda de la forma a dibujar. */
$cuadrado->movePenTo(-10, -10);
/* Se dibuja la forma hacia la izquierda. */
$cuadrado->drawLineTo(-10, 10);
$cuadrado->drawLineTo(10, 10);
$cuadrado->drawLineTo(10, -10);
$cuadrado->drawLineTo(-10, -10);

/* Se crea un sprite para alojar la forma. */
$clip = new SWFSprite();
$manejadorDeClip = $clip->add($cuadrado);

/* Se crea un escalado mediante un bucle. */
```

```
for ($cambios=1; $cambios<150; $cambios++){
    $clip->nextFrame();
    $manejadorDeClip->scale (1.02, 1.02);
}

/* Se añade el sprite a la película.*/
$manejadorDePelícula = $película->add($clip);
$manejadorDePelícula->moveTo (150, 150);

/* Se envía la película al navegador.*/
header("Content-type:application/x-shockwave-flash");
$pelicula->output();
?>
```

Si carga este script en el navegador, mediante la página **escaladoSimple.htm**, verá una película de fondo azul. En el centro aparece un pequeño cuadradito de contorno amarillo y relleno rojo, de 10 píxeles de lado. Éste se agranda progresivamente hasta llenar el total del espacio de la película. Luego, la animación se repite indefinidamente (ya aprenderemos a detenerla).

Observe la parte resaltada del código. En primer lugar se crea un mini-clip, usando el constructor de la clase SWFSprite. A éste se le incorpora el cuadrilátero, que ya tenemos diseñado, y se obtiene un manejador (**\$manejadorDeClip**) que, según ya sabemos, pertenece a la clase SWFDisplayItem. A continuación, tenemos un bucle que tendrá 149 iteraciones, según nos dicen los parámetros de la variable de control del mismo. Dentro del bucle se llevan a cabo dos operaciones. Por un lado, se genera un nuevo fotograma para el sprite. Por el otro, y esto es lo nuevo, se aplica el método `scale ()` al manejador del sprite. Este método recibe dos argumentos. El primero representa el cambio de anchura y el segundo el cambio de altura. El valor de estos argumentos (que, en nuestro ejemplo son iguales para mantener la proporción) es un multiplicador relativo a los valores actuales del tamaño del sprite. Así pues, poner 1.02 equivale a multiplicar el parámetro de que se trate (anchura o altura) por la cifra especificada. Dicho de otro modo. En cada iteración del bucle (y, por tanto, en cada nuevo fotograma) la anchura se elevará al 102% de su valor actual; y otro tanto ocurrirá con la altura.

## 20.7.4 Eliminación de objetos

Lo mismo que podemos añadir formas a los sprites o a las películas, también podemos eliminarlos mediante el uso del método `remove ()`, que recibe, como argumento, el nombre de objeto que representa al elemento a eliminar. Las clases SWFSprite y SWFMovie cuentan con este método. Podemos usarlo para eliminar una forma de un sprite o éste de la película principal.

En nuestro próximo ejercicio veremos una forma quieta en la película. También veremos otra que se desplaza hacia ella. Cuando se produce la colisión entre ambas, la que estaba quieta desaparece, y la otra continúa su desplazamiento. El script se llama **eliminacion.php**, y su listado es el siguiente:

```
<?php
/* Primero se crea la base de la película, como
siempre. */
$pelicula = new SWFMovie();
$pelicula->setDimension(300, 30);
$pelicula->setRate(40);
$pelicula->setBackground(0, 0, 255);

/* Se crea un objeto que será un cuadrado de contorno
amarillo sin relleno. */
$cuadrado_1 = new SWFShape();
$cuadrado_1->setLine(3, 255, 255, 0, 255);
$cuadrado_1->movePenTo(-10, -10);
$cuadrado_1->drawLineTo(-10, 10);
$cuadrado_1->drawLineTo(10, 10);
$cuadrado_1->drawLineTo(10, -10);
$cuadrado_1->drawLineTo(-10, -10);

/* Se crea otro objeto: un cuadrado de contorno verde
sin relleno. */
$cuadrado_2 = new SWFShape();
$cuadrado_2->setLine(3, 0, 255, 0, 255);
$cuadrado_2->movePenTo(-10, -10);
$cuadrado_2->drawLineTo(-10, 10);
$cuadrado_2->drawLineTo(10, 10);
$cuadrado_2->drawLineTo(10, -10);
$cuadrado_2->drawLineTo(-10, -10);

/* Se crea un sprite para alojar el cuadrado amarillo.
*/
$clip_1 = new SWFSprite();
$manejadorDeClip_1 = $clip_1->add($cuadrado_1);
$clip_1->nextFrame();
/* Se crea un sprite para alojar el cuadrado verde. */
$clip_2 = new SWFSprite();
$manejadorDeClip_2 = $clip_2->add($cuadrado_2);
$clip_2->nextFrame();

/* Se añaden los sprites a la película y se colocan en
sus posiciones iniciales. */
$manejadorDeForma_2 = $pelicula->add($clip_2);
$manejadorDeForma_1 = $pelicula->add($clip_1);
```

```
$manejadorDeForma_1->moveTo (15, 15);
$manejadorDeForma_2->moveTo (150, 15);

/* Se crea un bucle que desplaza el sprite del cuadrado
amarillo por la película. */
for ($desplazamiento=1; $desplazamiento<276;
$desplazamiento++){
    $pelicula->nextFrame();
    $manejadorDeForma_1->move(1,0);
    /* En el momento de alcanzar la posición del recuadro
verde, este desaparece (es eliminado). */
    if ($desplazamiento==136) $pelicula-
>remove($manejadorDeForma_2);
}
/* Se envía la película al navegador. */
header("Content-type:application/x-shockwave-flash");
$pelicula->output();
?>
```

Cuando cargue este script, a través de la página **eliminacion.htm**, verá una película de 300 píxeles de ancho y 30 de alto, de fondo azul. En el lateral izquierdo hay un cuadradito amarillo y en el centro uno verde. El amarillo se desplaza hacia la derecha. Cuando alcanza al verde, éste desaparece. Vea, en el bloque resaltado del código, cómo hemos usado el método `remove` para eliminar el sprite del cuadradito verde de la película en el momento adecuado.

## 20.7.5 Grabar la película

Puede que, cuando usted realice una película de Flash mediante la librería MING, desee que ésta quede almacenada en el disco duro del servidor en lugar de mostrarse en la página. O, tal vez, además de mostrarse. Para que la película se grabe usamos el método `save()` de la clase SWFMovie. Este método recibe, como argumento, el nombre del archivo con el que queremos grabar la película. Por ejemplo, podríamos tener una línea como la siguiente:

```
$pelicula->save ("miPelícula.swf");
```

Con esto, la película quedará almacenada en el servidor con el nombre especificado.

## 20.7.6 Interpolaciones de forma

Hasta el momento hemos visto rotaciones, desplazamientos, escalados... En definitiva: animaciones que podemos englobar dentro de la concepción de Flash

relativa a las interpolaciones de movimiento. Las interpolaciones de forma son cambios estructurales en la morfología de una figura. Para realizar este tipo de cambios debemos recurrir a la clase **SWFMorph**, especialmente diseñada a este fin. Esta clase es especialmente simple de manejar. En realidad sólo tiene dos métodos (aparte, naturalmente, de su propio constructor), que no reciben argumentos. El primero es **getShape1()** y el segundo es **getShape2()**. Lo que hacemos es diseñar dos formas diferentes. Éstas se asignan a los métodos del objeto de la clase SWFMorph que definen los dos estados límites de la transformación. Luego sólo tenemos que establecer la velocidad de la misma, mediante el método **setRatio()** de la clase SWFDisplayItem. Observe el listado del script **morphing.php**:

```
<?php
/* Primero se crea la base de la película, como
siempre. */
$pelicula = new SWFMovie();
$pelicula->setDimension(300, 300);
$pelicula->setRate(30);
$pelicula->setBackground(0, 0, 255);
/* Se crean dos formas: un cuadrado y un rectángulo. */
$cuadrado = new SWFShape();
$rectangulo = new SWFShape();

/* Se crea un objeto de interpolación de forma. */
$interpolacion = new SWFMorph ();
/* Se asignan las dos shapes a las formas de la
interpolación */
$cuadrado = $interpolacion->getShape1 ();
$rectangulo = $interpolacion->getShape2 ();

/* Se construye el cuadrado. */
$cuadrado->setLine(3, 255, 255, 0, 255);
$cuadrado->movePenTo(-30, -30);
$cuadrado->drawLineTo(30, -30);
$cuadrado->drawLineTo(30, 30);
$cuadrado->drawLineTo(-30, 30);
$cuadrado->drawLineTo(-30, -30);
/* Se construye el rectángulo. */
$rectangulo->setLine(3, 0, 255, 0, 255);
$rectangulo->movePenTo(-80, -15);
$rectangulo->drawLineTo(80, -15);
$rectangulo->drawLineTo(80, 15);
$rectangulo->drawLineTo(-80, 15);
$rectangulo->drawLineTo(-80, -15);

/* Se crea un sprite y se le añade la interpolación */
```

```
$clip= new SWFSprite();
$manejadorDeClip=$clip->add($interpolacion);
$clip->nextFrame();
/* Se crea un bucle que ejecuta la interpolación. */
for ($cambio=0.0; $cambio<=1.0; $cambio+=0.02){
    $manejadorDeClip->setRatio ($cambio);
    $clip->nextFrame ();
}

/* Se añade el sprite a la película y se coloca en el
centro. */
$manejadorDePelicula = $pelicula->add($clip);
$manejadorDePelicula->moveTo (150, 150);
/* Se envía la película al navegador. */
header("Content-type:application/x-shockwave-flash");
$pelicula->output();
?>
```

Cargue el script en el navegador a través de la página **morphing.htm**. Verá un cuadrilátero amarillo que se transforma en un rectángulo verde. Veamos cómo funciona. Observe el primer bloque resaltado. Antes de llegar a él, las dos formas (la que contendrá el cuadrado y la que contendrá el rectángulo) ya han sido creadas como objetos de la clase SWFShape. En el bloque que nos ocupa se crea un objeto nuevo de la clase SWFMorph. Después, a cada una de las formas se le asigna uno de los métodos de dicho objeto, de modo que estamos definiendo la transformación (que, en nuestro script, hemos llamado **Sinterpolacion**), indicando cuáles serán los límites de la misma. El método getShape1 () determina cuál será el estado de inicio de la transformación y getShape2 () determina el estado final de la misma.

Tras esta fase, pasamos a dibujar las dos formas, como ya es habitual. Esta parte no tiene nada nuevo. Ahora veamos el segundo bloque resaltado. Creamos un sprite y le añadimos la transformación, obteniendo un manejador para dicho sprite. Esto lo hacemos mediante las siguientes líneas:

```
$clip= new SWFSprite();
$manejadorDeClip=$clip->add($interpolacion);
```

Y llegamos al momento en que se debe ejecutar la transformación. Como ya es habitual, ésta es gobernada por un bucle, que genera los nuevos fotogramas necesarios. Además, establece un valor cambiante (basado en la variable de control) como argumento para el método setRatio (). Éste es el que se encarga de determinar cuál será el siguiente estado de la transformación. Fíjese en la estructura del bucle. Cuanto más pequeños sean los incrementos (en nuestro caso son de sólo 0.02), más suave resultará el cambio. Si este valor es demasiado alto, el cambio se verá “a saltos”.

Una vez llegados a este punto, sólo nos queda incorporar el sprite (que ya contiene la transformación) a la película principal, posicionarlo en el lienzo de la misma y enviar ésta al navegador.

## 20.8 IMÁGENES

Anteriormente hemos aprendido a usar imágenes archivadas en el disco como relleno de una forma. Ahora vamos a aprender más sobre lo que podemos hacer con imágenes en nuestras películas, con el fin de lograr mejores resultados. En este apartado vamos a preocuparnos solamente de un tema: incorporar una imagen a la película sin que tenga que ser el relleno de una figura, sino que sea la propia imagen, por decirlo de algún modo, la protagonista. Observe el listado **bitmaps.php**:

```
<?php
/* Primero se crea la base de la película. */
$pelicula = new SWFMovie();
$pelicula->setDimension(300, 300);
$pelicula->setRate(30);
$pelicula->setBackground(0, 0, 255);

/* Se crea un nuevo objeto SWFBitmap con una imagen. */
$foto = new SWFBitmap
(file_get_contents("imagenes/atmosfera.png"));
/* Se añade el objeto de clase SWFBitmap a un sprite y
se le da movimiento. */
$clip = new SWFSprite ();
$manejadorDeClip = $clip->add ($foto);
$clip->nextFrame ();
for ($cambio=1; $cambio<200; $cambio++){
    $manejadorDeClip->move (1, 1);
    $clip->nextFrame ();
}

/* Se añade el sprite a la película y se posiciona. */
$manejadorDePelicula = $pelicula->add($clip);
$manejadorDePelicula->moveTo (0, 0);

/* Se envía la película al navegador. */
header("Content-type:application/x-shockwave-flash");
$pelicula->output();
?>
```

Las imágenes se incorporan a nuestras películas creando un objeto de la clase SWFBitmap. El constructor de esta clase recibe, como argumento, el

contenido del fichero de la imagen que queremos incorporar. En nuestro script hemos usado una función específica: `file_get_contents()` obtiene el contenido de un fichero de una sola vez, y lo almacena en una variable. En este caso, lo hemos almacenado en nuestro objeto, al ponerlo como argumento del constructor. Ésta es la forma más cómoda de crear un objeto de imagen. Puede ver la mecánica de esta operación en el bloque resaltado del código.

A partir de ahí, ya no hay nada nuevo. Hemos creado un sprite y le hemos dado un movimiento para que pueda apreciar el resultado. Cargue la página **bitmaps.htm** para verlo.

## 20.9 ACCIONES

A lo largo de este capítulo, hemos visto varias animaciones que se repiten indefinidamente, y hemos comentado que encontrariamos el modo de cambiar ese comportamiento. La forma de hacer esto, y muchísimas otras cosas, son las acciones. Éstas son instrucciones de ActionScript (el lenguaje de programación de Flash) que se incorporan a nuestro código PHP mediante la clase **SWFAction**. En este apartado vamos a ver un ejemplo muy simple, para entender cómo funciona esta clase. Con ActionScript incorporado a sus scripts de PHP tendrá toda la potencia de Flash. No obstante, debo advertirle que aquí no estudiaremos el lenguaje ActionScript, ya que queda totalmente fuera de las pretensiones de esta obra. Al margen de eso, es un lenguaje que requiere un libro completo para él solo, ya que es muy amplio. La editorial Ra-Ma cuenta con muy buena bibliografía al respecto. Si usted desea aprender ActionScript, debería consultar en su librería habitual sobre dicho material. Para ver cómo poner una acción en una animación, usaremos la instrucción “`stop()`”, que detiene una línea de tiempo. Veamos una animación que ya tenemos de un ejercicio anterior, en la que un cuadrado se deslizaba por la película y borraba otro al alcanzarlo. Esta vez, al cargar la animación, ésta solo se reproducirá una vez. Al terminar, se detendrá. El script se llama **acciones.php**, y su listado es el siguiente:

```
<?php
/* Primero se crea la base de la película. */
$pelicula = new SWFMovie();
$pelicula->setDimension(300, 30);
$pelicula->setRate(40);
$pelicula->setBackground(0, 0, 255);

/* Se crea un objeto de la clase SWFAction, para
contener una instrucción de ActionScript. */

$instruccion = new SWFAction ("stop();");
```

```
/* Se crea un objeto que será un cuadrado de contorno
amarillo sin relleno. */
$cuadrado_1 = new SWFShape();
$cuadrado_1->setLine(3, 255, 255, 0, 255);
$cuadrado_1->movePenTo(-10, -10);
$cuadrado_1->drawLineTo(-10, 10);
$cuadrado_1->drawLineTo(10, 10);
$cuadrado_1->drawLineTo(10, -10);
$cuadrado_1->drawLineTo(-10, -10);

/* Se crea otro objeto: un cuadrado de contorno verde
sin relleno. */
$cuadrado_2 = new SWFShape();
$cuadrado_2->setLine(3, 0, 255, 0, 255);
$cuadrado_2->movePenTo(-10, -10);
$cuadrado_2->drawLineTo(-10, 10);
$cuadrado_2->drawLineTo(10, 10);
$cuadrado_2->drawLineTo(10, -10);
$cuadrado_2->drawLineTo(-10, -10);

/* Se crea un sprite para alojar el cuadrado amarillo.
*/
$clip_1 = new SWFSprite();
$manejadorDeClip_1 = $clip_1->add($cuadrado_1);
$clip_1->nextFrame();

/* Se crea un sprite para alojar el cuadrado verde. */
$clip_2 = new SWFSprite();
$manejadorDeClip_2 = $clip_2->add($cuadrado_2);
$clip_2->nextFrame();
/* Se añaden los sprites a la película y se colocan en
sus posiciones iniciales. */
$manejadorDeForma_2 = $pelicula->add($clip_2);
$manejadorDeForma_1 = $pelicula->add($clip_1);
$manejadorDeForma_1->moveTo (15, 15);
$manejadorDeForma_2->moveTo (150, 15);

/* Se crea un bucle que desplaza el sprite del cuadrado
amarillo por la película. */
for ($desplazamiento=1; $desplazamiento<276;
$desplazamiento++){
    $manejadorDeForma_1->move(1,0);
    /* En el momento de alcanzar la posición del recuadro
    verde, este desaparece (es eliminado). */
    if ($desplazamiento==136) $pelicula-
>remove($manejadorDeForma_2);
    $pelicula->nextFrame();
}
```

```
/* Se carga la instrucción en la película. */
$pelicula->add($instrucción);
/* Se envía la película al navegador. */
header("Content-type:application/x-shockwave-flash");
$pelicula->output();
?>
```

El código es como el que ya conocemos del ejercicio anterior. Lo he reproducido completo para que pueda ver dónde van colocadas las dos nuevas líneas, que aparecen resaltadas. En la primera de ellas, creamos un objeto de la clase SWFAction, usando, como argumento del constructor, la instrucción de ActionScript que deseamos. En el segundo bloque resaltado, vemos cómo se añade este objeto a la película. Así, para comprobar su funcionamiento, cargue en el navegador la página **acciones.htm** y verá que, esta vez, la película no se reproduce indefinidamente. Cuando conozca ActionScript podrá incluir más instrucciones en sus películas.

## 20.10 BOTONES

Como sabemos, una de las características de una película de Flash es su interactividad. El usuario puede interactuar con la película, de forma que ésta responda a determinados actos de aquél. Y uno de los elementos más significativos en esta interactividad son los botones. Éstos son figuras sobre las que el usuario puede actuar con el ratón, de forma que detecten el paso del mismo, o un clic, y desencadenen una acción como consecuencia.

Para crear un botón, es necesario definir cuatro formas. La primera será el aspecto de dicho botón en reposo, es decir, cuando no se apoya sobre él el puntero del ratón ni tiene lugar ningún evento. La segunda forma será el aspecto del botón cuando el puntero del ratón esté apoyado sobre el mismo, pero no se haya pulsado. La tercera corresponde al aspecto que tendrá cuando se haya pulsado. La cuarta forma es más etérea, en el sentido de que no es visible. Corresponde al área en la que el botón detecta la presencia del ratón, es decir, en la que el botón es susceptible de ser pulsado. Esta forma es como una sombra invisible que cubre el botón y, sin la cual, este no puede detectar el puntero.

Para aclarar esto, nada como un buen ejemplo. El siguiente script, muestra un botón en la película. Se llama **botónSimple.php**, y su listado es el siguiente:

```
<?php
/* Primero se crea la base de la película. */
$pelicula = new SWFMovie();
$pelicula->setDimension(60, 60);
```

```
$pelicula->setRate(15);
$pelicula->setBackground(0, 0, 255);

/* Creamos cuatro formas, que representarán los estados
del botón. */
$reposo = new SWFShape();
$activo = new SWFShape();
$pulsado = new SWFShape();
$area = new SWFShape();

/* Creamos el botón, en base a las cuatro formas. */
$boton = new SWFButton ();
$boton->addShape ($reposo, SWFBUTTON_UP);
$boton->addShape ($activo, SWFBUTTON_OVER);
$boton->addShape ($pulsado, SWFBUTTON_DOWN);
$boton->addShape ($area, SWFBUTTON_HIT);

/* Definimos las cuatro formas, como círculos con
diferente color. */
$reposo->setLine (1,200,100,100,255);
$relleno=$reposo->addFill (200,100,100);
$reposo->setRightFill($relleno);
$reposo->drawCircle (20);
///////////////////////////////
$activo->setLine (1,0,100,100,255);
$relleno=$activo->addFill (0,100,100);
$activo->setRightFill($relleno);
$activo->drawCircle (20);
/////////////////////////////
$pulsado->setLine (1,0,255,255,255);
$relleno=$pulsado->addFill (0,255,255);
$pulsado->setRightFill($relleno);
$pulsado->drawCircle (20);
/////////////////////////////
$area->setLine (1,200,100,100,0);
$relleno=$area->addFill (200,100,100,0);
$area->setRightFill($relleno);
$area->drawCircle (20);
/* Se añade el botón a la película. */
$manejadorDePelicula=$pelicula->add($boton);
$manejadorDePelicula->moveTo (30, 30);
/* Se envía la película al navegador. */
header("Content-type:application/x-shockwave-flash");
$pelicula->output();
?>
```

Al ejecutar este script, mediante la carga de la página **botonSimple.htm**, veremos en el navegador una pequeña película, de sólo 60 pixeles de lado, con un

botón redondito en su centro. El botón tiene tres estados visibles diferentes (cambiando, en este caso, de color). El primero corresponde al botón en reposo. El segundo se ve cuando se apoya el puntero del ratón sobre él y el tercero cuando, teniendo el puntero sobre el botón de la película, se pulsa el botón izquierdo del ratón. Los tres posibles estados aparecen en la figura 20.31.



Figura 20.31

Vamos a estudiar el código del script en el que aprenderemos varias cosas. En primer lugar, observe que el botón tiene forma circular. Ésta es una forma que no habíamos empleado hasta ahora, sumidos en nuestra vorágine de cuadriláteros y rectángulos, y resulta muy fácil de crear. Al contrario de lo que ocurre con otras, en las que hay que dibujar una línea para cada lado, un círculo se traza con una sola instrucción: el método *drawCircle()*, que recibe, como argumento, el radio, en pixeles, de la circunferencia que queremos trazar.

Una vez aclarado esto, vemos cómo se construye el botón, a partir de las cuatro formas creadas. Observe el fragmento resaltado en el script. En él vemos que empezamos creando un nuevo objeto de la clase *SWFButton*, que ha sido diseñada para esta finalidad. A continuación, a dicho objeto se le añaden las cuatro formas, mediante el uso del método *addShape()*. Éste recibe dos argumentos. El nombre del objeto correspondiente a la forma que queremos añadir y el estado al que se asociará dicha forma. Como ya sabemos, el botón se compone de cuatro estados posibles, aunque, al ejecutar la película sólo veamos tres. Éstos son:

- ***SWFBUTTON\_UP***. Es el estado que corresponde al botón en reposo.
- ***SWFBUTTON\_OVER***. Corresponde al momento en que el puntero del ratón está apoyado sobre la figura.
- ***SWFBUTTON\_DOWN***. Es cuando, teniendo el puntero apoyado sobre la figura, pulsamos el botón izquierdo del ratón.
- ***SWFBUTTON\_HIT***. Es el área en que se reconocerá que el puntero del ratón está apoyado sobre la figura. Esta forma es invisible, de modo que nos da igual el color que le pongamos, tanto de línea como de relleno. Esto implica que el botón no reconoce cuando acercamos el

puntero a las formas visibles, sino cuando lo acercamos a esta forma en concreto. En nuestro ejemplo, la geometría y tamaño de esta forma coincide con las de las otras, por lo que el efecto es que el botón completo reacciona cuando le apoyamos el puntero.

Fíjese en una cosa. El botón funciona perfectamente, salvo que no hace nada. Esto es porque aún no le hemos asignado ninguna acción específica. Para eso necesitamos recurrir al método **addAction()**, cuyo objetivo es asignarle acciones a los posibles estados de un botón. Observe el script **botonActivo.php**, listado a continuación:

```
<?php
/* Primero se crea la base de la película. */
$pelicula = new SWFMovie();
$pelicula->setDimension(200, 100);
$pelicula->setRate(30);
$pelicula->setBackground(0, 0, 255);

/* Creamos cuatro formas, que representarán los estados
del botón. */
$reposo = new SWFShape();
$activo = new SWFShape();
$pulsado = new SWFShape();
$area = new SWFShape();

/* Se crea una instrucción, a partir de la clase
SWFAction, para parar la película. El argumento del
constructor es una instrucción de ActionScript aplicada al
objeto "movimiento", con el que se identificará la película
(ver más abajo). */
$instruccion = new SWFAction ("movimiento.stop();");

/* Creamos el botón, en base a las cuatro formas. */
$boton = new SWFButton ();
$boton->addShape ($reposo, SWFBUTTON_UP);
$boton->addShape ($activo, SWFBUTTON_OVER);
$boton->addShape ($pulsado, SWFBUTTON_DOWN);
$boton->addShape ($area, SWFBUTTON_HIT);

/* Le añadimos la instrucción anterior al botón,
asociada al evento de pulsar. */
$boton->addAction ($instruccion,
SWFBUTTON_MOUSEDOWN);

/* Definimos las cuatro formas, como círculos con
diferente color. */
```

```
$repositorio->setLine (1,200,100,100,255);
$relleno=$repositorio->addFill (200,100,100);
$repositorio->setRightFill($relleno);
$repositorio->drawCircle (20);
///////////////////////////////
$activo->setLine (1,0,100,100,255);
$relleno=$activo->addFill (0,100,100);
$activo->setRightFill($relleno);
$activo->drawCircle (20);
///////////////////////////////
$pulsado->setLine (1,0,255,255,255);
$relleno=$pulsado->addFill (0,255,255);
$pulsado->setRightFill($relleno);
$pulsado->drawCircle (20);
///////////////////////////////
$area->setLine (1,200,100,100,0);
$relleno=$area->addFill (200,100,100,0);
$area->setRightFill($relleno);
$area->drawCircle (20);

/* Se crea un objeto para que se mueva. */
$cuadradito = new SWFShape ();
$cuadradito->setLine (2,255,255,0,255);
$relleno=$cuadradito->addFill (255,255,0);
$cuadradito->setRightFill ($relleno);
$cuadradito->movePenTo (-15, 15);
$cuadradito->drawLineTo (15, 15);
$cuadradito->drawLineTo (15, -15);
$cuadradito->drawLineTo (-15, -15);
$cuadradito->drawLineTo (-15, 15);
/* Se añade el botón a la película. */
$manejadorDePelícula=$película->add($botón);
$manejadorDePelícula->moveTo (30, 30);
/* Se añade el cuadradito a la película. */
$manejadorDeClip=$película->add ($cuadradito);
$manejadorDeClip->moveTo (20,80);
/* Se le asigna un identificador al cuadradito en la
película, para poder asignarle luego la instrucción de
ActionScript que se creó anteriormente. */
$manejadorDeClip->setName("movimiento");
/* El movimiento del cuadradito, a través de una
trayectoria horizontal. */
for ($contador=0; $contador<160; $contador++){
    $película->nextFrame();
    $manejadorDeClip->move (1,0);
}

/* Se envía la película al navegador. */
```

```
header("Content-type:application/x-shockwave-flash");
$pelicula->output();
?>
```

Cargue el script en el navegador a través, como siempre, de la página **botonActivo.htm**. Verá dos elementos en la película. En la parte superior de la misma, hay un botón como el del ejemplo anterior. En la parte inferior, hay un cuadradito que se desliza de izquierda a derecha. Pulse el botón y verá que el cuadradito se detiene. Es un ejemplo muy sencillo, pero que ilustra perfectamente el mecanismo que tratamos de ver.

Para entender cómo funciona esto vamos a reparar en los tres bloques del listado que aparecen resaltados. En primer lugar, se crea un objeto de la clase SWFAction como aprendimos a hacerlo en el apartado anterior. Como usted ya sabe, al crear un objeto de esta clase le pasamos al constructor una instrucción de ActionScript como argumento. La que vamos a usar en este caso es stop(), que se usa para detener una película. Sin embargo, esta vez hay algo diferente. En la cadena que compone el argumento aparece algo más. El nombre "movimiento" se asigna al cuadradito colocado en la película, tal como se ve, en el tercer bloque resaltado del código, mediante el uso del método **setName()**. Cuando queremos poder ejecutar código de ActionScript sobre un elemento de la película, es necesario asignarle un identificador, con este método, al manejador. Este método pertenece a la clase SWFDisplayItem, así que sólo puede usarse con un manejador.

Volviendo a la instrucción, vea que en la cadena que la compone aparece el identificativo, seguido de un punto y la propia instrucción. Así, cuando llegue el momento de ejecutarla, PHP "sabrá" que tiene que actuar sobre el objeto que tiene este identificativo. El punto que separa dicho identificativo de la instrucción en sí corresponde a la notación propia de ActionScript y no vamos a entrar aquí en detalles al respecto.

Ahora veamos el segundo bloque resaltado. El método **addAction()**, de la clase SWFButton permite asociar una instrucción de ActionScript a un evento del botón. Este método recibe dos argumentos. El primero es el objeto SWFAction que contiene la instrucción que nos interesa. El segundo es el evento que tiene que tener lugar para que dicha instrucción se ejecute. Los botones de Flash, a través de su área activa (la forma que corresponde a SWFBUTTON\_HIT) pueden detectar todos los eventos necesarios para gestionar cualquier acción, tales como la presencia del puntero del ratón, pulsar o soltar el botón del ratón, etc. Los posibles eventos que pueden desencadenar una instrucción son los siguientes:

- **SWFBUTTON\_MOUSEOVER.** Detecta el paso del puntero sobre el botón.

- **SWFBUTTON\_MOUSEOUT.** Detecta el momento en que el puntero sale del botón.
- **SWFBUTTON\_MOUSEUP.** Detecta cuándo se suelta el botón izquierdo del ratón con el puntero sobre el botón de Flash.
- **SWFBUTTON\_MOUSEUPOUTSIDE.** Detecta cuándo se libera el botón izquierdo del ratón con el puntero fuera del botón de Flash, siempre que la pulsación se produjera con dicho puntero sobre el botón de Flash.
- **SWFBUTTON\_MOUSEDOWN.** Detecta cuándo se pulsa el botón izquierdo del ratón con el puntero sobre el botón de Flash.
- **SWFBUTTON\_DRAGOUT.** Detecta cuándo se pulsa el botón izquierdo del ratón con el puntero sobre el área activa del botón de Flash y se arrastra dicho puntero fuera sin soltar el botón del ratón.
- **SWFBUTTON\_DRAGOVER.** Detecta cuándo se arrastra el puntero sobre el área activa del botón de Flash, con el botón izquierdo del ratón previamente pulsado.

Una vez asociada correctamente una instrucción de ActionScript a un evento de un botón, la instrucción se desencadena en el momento de que ocurra el evento especificado.

## 20.11 TEXTO DINÁMICO

Anteriormente, en el apartado 20.6, tuvimos un encuentro con la posibilidad de incorporar texto a nuestras películas. Aunque entonces no lo mencioné, porque no era el momento, la librería MING nos permite crear campos de texto dinámicos, es decir, que no tengan un texto predefinido, sino que su contenido dependa de valores que se obtienen de la ejecución del script, el entorno de la película, eventos de usuario, etc. Si usted está familiarizado con Flash, sabrá que este programa también ofrece este tipo de campos de texto, para ser manejados desde ActionScript. Esto es igualmente válido cuando se usa la librería MING de PHP para emular a Flash.

Vamos a ver un ejemplo muy simple, porque no es nuestra intención divagar incluyendo aquí, como ya he mencionado anteriormente, ActionScript. Así pues, nuestro ejemplo sólo usará una instrucción de este lenguaje, que resulta imprescindible. Vea el listado de **textoDinamico.php**:

```
<?php
/* Empezamos creando la base de la película. */
$peliculaPrincipal = new SWFMovie();
$peliculaPrincipal->setDimension(80, 100);
```

```
$peliculaPrincipal->setRate(15);
$peliculaPrincipal->setBackground(0, 0, 255);
/* Creamos una forma con un cuadrilátero.*/
$cuadrilatero = new SWFShape();
$cuadrilatero->setLine(2, 255, 255, 0, 255);
$cuadrilatero->movePenTo(-25, 25);
$cuadrilatero->drawLineTo(25, 25);
$cuadrilatero->drawLineTo(25, -25);
$cuadrilatero->drawLineTo(-25, -25);
$cuadrilatero->drawLineTo(-25, 25);

/* Creamos un sprite y le añadimos el cuadrilátero que
acabamos de dibujar.*/
$forma = new SWFSprite();
$forma->add($cuadrilatero);
$forma->nextFrame();

/* Creamos un campo de texto dinámico y establecemos
sus atributos.*/
$campoDeTextoDinamico = new
SWFTextField(SWFTEXTFIELD_DRAWBOX | SWFTEXTFIELD_NOSELECT |
SWFTEXTFIELD_NOEDIT);
$campoDeTextoDinamico->setBounds(50, 15);
$campoDeTextoDinamico->setFont(new SWFFont("_sans"));
$campoDeTextoDinamico->setColor(0, 0, 0);
$campoDeTextoDinamico->setName("campoDeTexto"); //Identificativo para ActionScript.

/* Creamos un segundo sprite y le añadimos el campo de
texto.*/
$clipDeTexto = new SWFSprite();
$clipDeTexto->add($campoDeTextoDinamico);
$clipDeTexto->nextFrame();

/* Añadir añadimos el sprite con el cuadrilátero a la
película.*/
$manejadorDeForma = $peliculaPrincipal->add($forma);
$manejadorDeForma->moveTo(40, 35);
$manejadorDeForma->setName("forma");

/* Añadimos el sprite con el campo de texto a la
película.*/
$manejadorDeClipDeTexto = $peliculaPrincipal-
>add($clipDeTexto);
$manejadorDeClipDeTexto->moveTo(15, 70);
$manejadorDeClipDeTexto->setName("cajaDeTexto");
/* Creamos un objeto SWFAction, con el ActionScript
Necesario, y lo incorporamos a la película.*/

```

```
$peliculaPrincipal->add(new  
SWFAction("_root.cajaDeTexto.campoDeTexto = _root.forma._x +  
' , ' + _root.forma._y;"));  
/* Enviamos la película al navegador. */  
header("Content-type:application/x-shockwave-flash");  
$peliculaPrincipal->output();  
?>
```

Cuando carguemos este script en el navegador como ya sabemos, a través de la página **textoDinamico.htm**, verá una película como la de la figura 20.32.

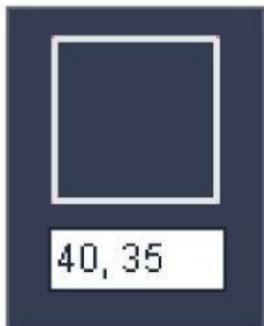


Figura 20.32

En esta imagen se aprecian dos elementos. Por un lado, en la parte superior, tenemos una forma geométrica. Por el otro, tenemos un campo de texto en el que aparecen las coordenadas en las que se encuentra posicionada la figura.

La forma en que se han obtenido esas coordenadas y se han enviado al campo de texto no se puede explicar aquí sin conocer ActionScript. Toda la operativa depende de la siguiente línea:

```
$peliculaPrincipal->add(new  
SWFAction("_root.cajaDeTexto.campoDeTexto = _root.forma._x +  
' , ' + _root.forma._y;"));
```

En ella ve cómo se ha cargado una instrucción ActionScript en un objeto SWFAction. Sin embargo, aún sin entender este código, vemos que hace referencia a tres elementos de la película: **cajaDeTexto**, **campoDeTexto** y **forma**. Se trata de objetos que han recibido estos identificativos mediante el uso del método `setName()`, de la clase `SWFDisplayItem`, como ya aprendimos anteriormente. Vea las líneas

resaltadas en el listado del script. Por otra parte, tenemos un elemento llamado **\_root**, que es un identificativo genérico que representa a la película principal, a la que, en definitiva, pertenecen todos los demás elementos. Las etiquetas **\_x** e **\_y** (con el guion bajo delante) se usan en ActionScript para referirse a coordenadas de posicionamiento.

Si bien, como ya he mencionado, la sintaxis y prestaciones de ActionScript quedan fuera del objetivo de este libro, sí quiero, en cambio, hablarle del campo de texto dinámico en sí mismo, con independencia del origen de su contenido. Observe la forma en que lo hemos creado. El constructor recibe unos argumentos que no habíamos usado anteriormente. Éstos determinan el tipo de campo de texto y su comportamiento. Recordemos cómo hemos creado el campo de texto que usamos en este ejemplo:

```
$campoDeTextoDinamico = new  
SWFTextField(SWFTEXTFIELD_DRAWBOX | SWFTEXTFIELD_NOSELECT |  
SWFTEXTFIELD_NOEDIT);
```

Las constantes que podemos añadir a un campo de texto en el momento de su creación son las siguientes:

- **SWFTEXTFIELD\_DRAWBOX**. Dibuja una línea alrededor del campo de texto.
- **SWFTEXTFIELD\_HTML**. Permite usar texto formateado al estilo de HTML.
- **SWFTEXTFIELD\_MULTILINE**. Permite escribir varias líneas de texto en el campo.
- **SWFTEXTFIELD\_NOEDIT**. Impide que el usuario pueda editar el contenido del campo.
- **SWFTEXTFIELD\_NOSELECT**. Impide que el usuario pueda seleccionar el contenido del campo.
- **SWFTEXTFIELD\_PASSWORD**. Oculta el texto que el usuario teclee, al estilo de las contraseñas.
- **SWFTEXTFIELD\_WORDWRAP**. Permite el salto de línea automático cuando se alcanza el final de la misma.

Si necesitamos incluir en el constructor dos o más de estas constantes las uniremos, como aparece en el listado del script, mediante el operador OR de bit (|).

Estas constantes sólo están disponibles para los campos de texto creados a partir de la clase **SWFTextField**, no los que se creen a partir de **SWFText**. No obstante, ya hemos mencionado anteriormente que esta última clase no la usaremos por los fallos que presenta sobre servidores Windows.

## 20.12 CONSIDERACIONES FINALES

A lo largo de este capítulo hemos aprendido a usar la librería MING para crear películas de Flash. Hemos conocido todas las principales prestaciones que PHP nos ofrece en ese sentido. Sin embargo, este conocimiento adolece de una importante precariedad sin ActionScript, como ya he mencionado en un par de ocasiones. Desde aquí, quiero alentar encarecidamente al lector a que estudie este tema, ya que, una vez que conozca este lenguaje, podrá llevar a cabo interesantes proyectos por sí mismo. También quiero aconsejarle que no deje de visitar los contenidos de un sitio web muy bien realizado. Lo localizará en la dirección <http://www16.brinkster.com/gazb/ming/index.html>. Esta página contiene una interesante librería de películas Flash, realizadas con MING... y los códigos fuente de todas ellas, de modo que puede ver las que le resulten interesantes en un momento dado y adaptar el listado a sus propios diseños. Además, estos contenidos se pueden visitar, copiar y utilizar de forma totalmente gratuita.

Espero sinceramente que este capítulo, al igual que el resto del libro, le haya resultado tan interesante de leer como a mí de escribir, y que le resulte útil en su vida profesional como webmaster.

## CAPÍTULO 21

# DEPURANDO NUESTRO TRABAJO

---

---

Con este capítulo se inicia una línea de trabajo completamente nueva, en la que vamos a perseguir tener un entorno de desarrollo cómodo y práctico, que nos facilite el día a día con PHP. Ésta es, o debe ser siempre, la filosofía de trabajo que deberemos emplear en todas nuestras singladuras informáticas. No me malinterprete. No se trata, en modo alguno, de aquello de “la ley del mínimo esfuerzo” ni de “haz lo menos posible y lo que haya que hacer, que lo haga otro”. Ni mucho menos. Bromas aparte, cuanto más cómodo y eficiente sea el entorno de trabajo, más podremos dedicar nuestros recursos intelectuales, de tiempo y de todo tipo al verdadero desarrollo, y a la concepción e implementación de las soluciones necesarias en cada caso. Veamos, pues, de qué herramientas podemos valernos para facilitar nuestro trabajo, no sólo de desarrollo sino, también, de depuración y puesta a punto.

### 21.1 EL SERVIDOR WAMP

En el capítulo 2 del libro aprendimos a montar las herramientas necesarias de trabajo. Para instalar el servidor Apache, el motor de MySQL y el intérprete de PHP recurrimos a la aplicación AppServ, todo un clásico en esta materia. En este capítulo vamos a conocer una aplicación integrada que monta estos tres programas de una manera mucho más cómoda en entornos Windows. Se trata del servidor WAMP (siglas de Windows, Apache, MySQL y PHP). Lo primero que deberíamos hacer es descargarlo de la página oficial ([www.wampserver.com](http://www.wampserver.com)). Sin embargo, allí solo encontraremos la última versión. En este caso, vamos a montar una más antigua. La versión actual, en el momento de escribir estas líneas, es la 2.0i, aunque nosotros trabajaremos con la 2.0a. Esto se debe a que nos instalará una revisión de

la versión 5.2 de PHP, mientras que la actual es la 5.3. Aunque parece lo más adecuado disponer siempre de las últimas versiones, para nuestro aprendizaje vamos a recurrir a la que le ofrecemos en el CD del libro, por razones que entenderá cuando lea el Apéndice G, que incluye comentarios específicos acerca de la versión 5.3 de PHP.

### 21.1.1 La instalación de WampServer

El instalador de Wamp es un único fichero incluido, como le hemos comentado, en el CD, ya que se trata de un producto de libre distribución. Ejecute el fichero con un doble clic y verá la ventana de la figura 21.1.

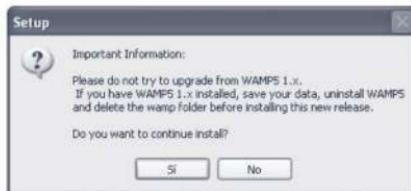


Figura 21.1

Se trata de un aviso para que no intentemos actualizar desde la versión 1.5 del programa. Si ésta se hallara instalada, deberíamos desinstalarla previamente. Es muy importante que lo hagamos así. Como no es nuestro caso, pulsamos el botón Sí, para proceder a la instalación, y vemos la ventana de la figura 21.2.



Figura 21.2

Pulse el botón **Next >** para pasar a la siguiente fase, tal como aparece en la figura 21.3.



Figura 21.3

Como ve, se trata de los términos de la licencia de uso de la aplicación. Al tratarse de una licencia GNU – GPL, no nos impone, realmente, restricciones de uso, modificación y/o redistribución. Acepte los términos y pase a la siguiente fase, mostrada en la figura 21.4.

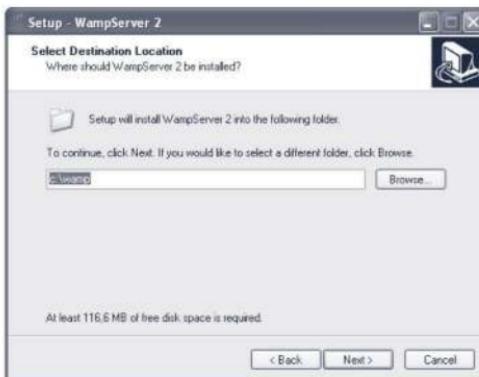


Figura 21.4.

Lo que aquí se le pide que indique es el directorio de su disco duro donde se instalará WampServer. Usted puede cambiarlo si lo desea, pero mi experiencia es que nos conviene dejar el que aparece por defecto. En caso contrario, la aplicación podría no funcionar adecuadamente. Evite pues, la tentación de seleccionar otro directorio y pase a la siguiente fase, mostrada en la figura 21.5.



Figura 21.5

Acorde al criterio de comodidad mencionado al inicio de este capítulo, marque las dos casillas, para crear un acceso directo en el escritorio, y un ícono de inicio en la barra de tareas. Después, pase a la fase mostrada en la figura 21.6.



Figura 21.6

Aquí simplemente aparecen los datos de instalación que hemos elegido, en cuanto a directorio e iconos se refiere, en las fases anteriores. Si no estamos de acuerdo con alguno de estos datos, podemos usar el botón < Back para volver. En este caso, procedemos a la instalación mediante el botón **Install**, llegando a la ventana de la figura 21.7.

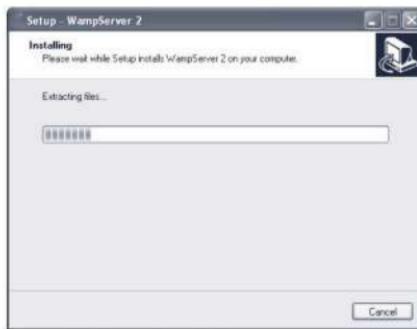


Figura 21.7

Mediante una barra de progreso, se aprecia cómo se va realizando la instalación. Ésta es, en realidad, muy rápida, quedando lista en pocos segundos. Lo siguiente que nos pregunta el instalador es acerca de cuál será nuestro navegador de Internet predeterminado. Le recomiendo encarecidamente que use la última versión de Firefox. Lo encontrará en [www.mozilla-europe.org/es/firefox/](http://www.mozilla-europe.org/es/firefox/). En la figura 21.8 puede ver esta elección.

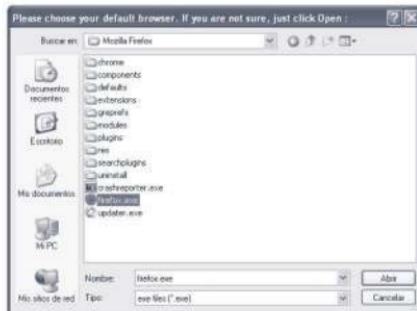


Figura 21.8

Durante unos segundos, verá de nuevo la barra de progreso anterior, mientras terminan de copiarse los ficheros. Después, se le piden los datos de correo a efectos de la función mail () de PHP. En la casilla SMTP deje el valor por defecto (**localhost**). En la casilla Email, ponga la dirección que desee emplear. Como yo voy a trabajar en modo local, he tecleado **pruebas@localhost.com**. Deberé crear esa dirección en mi servidor de correo (vea el capítulo 2 del libro al respecto), para poder usarla. En conjunto, esta fase queda como aparece en la figura 21.9.



Figura 21.9

Prácticamente hemos terminado. En el escritorio de su ordenador ya tiene un ícono de acceso directo al programa recién instalado. Pulse el botón **Next >** para acceder a la última ventana. Mostrada en la figura 21.10.



Figura 21.10

Asegúrese de desamarra la casilla **Launch WampServer 2 now** ya que, en caso contrario, se iniciará el servidor al terminar la instalación. Antes de iniciar el servidor, y esto es muy importante, debe detener otros servidores que pudieran influir en el funcionamiento de WampServer. Entre éstos, se incluyen, sin limitarnos a ellos, servidores de correo electrónico y de FTP. Si necesita tener estas aplicaciones funcionando junto a su WampServer, inicie primero este último (haciendo, por ejemplo, doble clic sobre el ícono del escritorio) y luego inicie el resto de los servidores, empezando por el de correo. No lo haga al revés o podría encontrarse con que no le funciona correctamente la aplicación.

Una vez desmarcada la casilla, pulse el botón **Finish**, con lo que se cerrará el programa de instalación. Detenga, como le he comentado, cualquier otro servidor, y arranque el WampServer.

En la barra de tareas verá un ícono nuevo con el aspecto de un reloj semicircular, como se aprecia en el recorte reproducido en la figura 21.11.



Figura 21.11

Este ícono pasa por tres fases. En la primera, se muestra con una zona en rojo; en la segunda, esta zona se amplía y aparece en amarillo; por último, queda completamente en blanco. Esto indica que todos los servicios (Apache, PHP y MySQL) se han iniciado correctamente. Si el ícono permanece en rojo o amarillo, es señal de que alguno de los servicios no se ha iniciado. Esto ocurre cuando, por ejemplo, no hemos tenido en cuenta lo que le comentaba acerca de los servidores de correo y FTP.

### 21.1.2 Configurando WampServer

Lo primero que vamos a ver, una vez puesto en marcha el WampServer, es cómo establecer el idioma. No tiene sentido usar la aplicación en inglés, si nos ofrece la oportunidad de usarla en nuestra lengua vernácula. Para ello, hacemos clic con el botón secundario del ratón (normalmente, el derecho) sobre el ícono de la barra de tareas reproducido en la figura 21.11. Esto nos abre un pequeño menú contextual, donde elegiremos la opción **Language** y, en la lista que se despliega, haremos clic con el botón principal sobre la opción **spanish**. Si ahora vuelve a abrir el menú contextual verá que este aparece en español. Éste será ahora el idioma para seguir trabajando con WampServer en lo sucesivo.

Ahora tenemos que decidir cuál será la carpeta de nuestro disco duro que emplearemos como localhost, o servidor local, para nuestro trabajo con PHP. Por defecto la aplicación crea la carpeta **/www** dentro de la propia carpeta de instalación del programa. En nuestro ejemplo, se ha creado **c:/wamp/www**. Sin embargo, es muy posible que usted tenga ya proyectos en otra carpeta que ha usado hasta ahora como localhost, y desee seguir usando la suya. En mi caso, tengo la carpeta **/Mis webs**, dentro de **Mis documentos** (perdón por la falta de originalidad), quedando, en mi ordenador, la siguiente ruta: **C:/Documents and Settings/Jose/Mis documentos/Mis webs**.

**Atención:** Asegúrese de separar los términos de una ruta empleando SIEMPRE las barras convencionales de notación de directorios (/) en lugar de las barras invertidas que le ofrece Windows por defecto (\). Recuerde que Windows admite los dos sistemas, pero las plataformas \*nix podrían tener problemas con las barras invertidas, y nosotros trabajamos para Internet, con lo que debemos buscar, siempre que sea posible, la universalización en todo lo que hacemos, como norma general.

Pues bien. Debemos decirle, al servidor Apache, dónde vamos a colocar nuestras páginas dinámicas. Para ello haremos clic con el botón primario del ratón sobre el ícono de la figura 21.11, lo que nos abrirá el menú principal de WampServer, mostrado en la figura 21.12.



Figura 21.12

En la opción **Apache** seleccione, en la lista desplegable que aparece, la opción **httpd.conf**. Esto le dará acceso al archivo de configuración de Apache, que aparece como un texto plano, en el editor por defecto del sistema, que suele ser el bloc de notas.

Busque la siguiente línea:

```
DocumentRoot "c:/wamp/www/"
```

y sustitúyala por:

```
DocumentRoot "c:/Documents and Settings/Jose/Mis  
documentos/Mis webs/"
```

A continuación, busque la siguiente línea:

```
<Directory "c:/wamp/www/">
```

y sustitúyala por:

```
<Directory "c:/Documents and Settings/Jose/Mis  
documentos/Mis webs/">
```

Después grabe el archivo y cierre la ventana del editor.

Vuelva a abrir el menú de la figura 21.12 y seleccione la opción **Reiniciar los Servicios**. Esto es necesario cada vez que modifique alguna configuración de Apache, PHP o MySQL. Las especificaciones de WampServer dicen que, cuando hacemos una modificación en la configuración, la aplicación reinicia los servicios automáticamente, pero no cuesta ningún trabajo asegurarnos de ello haciendo un reiniciado manual.

Durante el reiniciado, veremos cómo el ícono muestra el color amarillo y luego el rojo, según se van deteniendo los servicios. Tras unos segundos, pasa de nuevo al amarillo y, finalmente, al blanco, indicando que se han reiniciado correctamente los servicios del sistema.

### 21.1.3 Probando WampServer

Para probar el funcionamiento de WampServer, lo más fácil es cargar directamente una página dinámica con, por ejemplo, la famosa función `phpinfo()` de PHP. Grabe un archivo con esta función como único código, como el que aparece a continuación, con el nombre **info.php** en la carpeta que ha decidido usar como localhost.

```
<?php  
    phpinfo();  
?>
```

Cargue en su navegador la dirección <http://localhost/info.php>. El resultado deberá ser el de la figura 21.13.

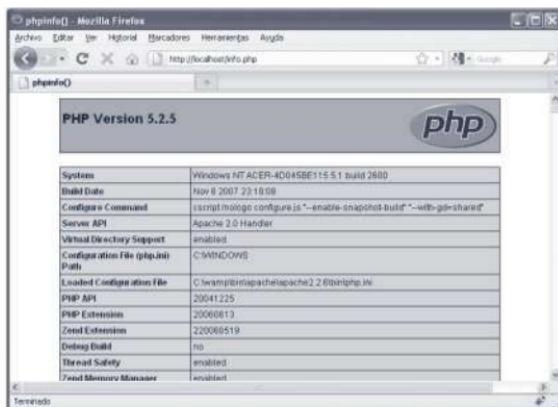


Figura 21.13

Aunque la tabla de datos que se genera no cabe completa en la pantalla, como se aprecia en la imagen, debiendo usar la barra de scroll, al menos vemos que el WampServer está funcionando correctamente.

## 21.2 EL ZEND DEBUGGER

En este apartado vamos a instalar una extensión de PHP. Una extensión es un “añadido” que mejora el funcionamiento de PHP o le añade nuevas funcionalidades. En Internet existen múltiples extensiones, tanto de la propia empresa desarrolladora de PHP (Zend Technologies) como de terceros. La extensión que vamos a instalar en este caso es una ayuda para la depuración (debugging) de nuestros proyectos en PHP. En este apartado, sin embargo, vamos a centrarnos, exclusivamente, en la instalación. Será más adelante, en este mismo capítulo, cuando veamos la forma de usarla.

Lo primero que debemos hacer es descargar de Internet los archivos de la extensión, que se encuentran en la página <http://downloads.zend.com/pdt/server-debugger/>. En este caso pulsaremos sobre el enlace **ZendDebugger-5.2.15-cygwin\_nt-i386.zip**, que corresponde a la versión para Windows de la extensión, válida para la versión 5.2 de PHP, que es la que nos ha instalado WampServer.

Una vez descargada la extensión, nos encontraremos con un archivo comprimido. Lo abrimos y vemos una carpeta, de nombre **ZendDebugger-5.2.15RC1-egwin\_nt-i386**. La extraeremos en nuestro ordenador y la abriremos. Usted la tiene en el CD del libro. No necesita descargarla de Internet.

Abra la carpeta y encontrará varias subcarpetas, con versiones de la extensión para distintas versiones de PHP. Abra la carpeta llamada **5\_2\_x\_comp**, que es la que vamos a usar aquí. Contiene un archivo, llamado **ZendDebugger.dll**, que es la extensión en sí mismo. Cópielo al portapapeles para pegarlo en el directorio de extensiones de PHP. Este directorio, si ha instalado WampServer como le he detallado en este capítulo, es **c:/wamp/bin/php/php5.2.5/ext/**. No obstante, si tiene otra instalación, o si desea confirmarlo por usted mismo, cargue la página **info.php** y busque la clave **extension\_dir**, como ve en la figura 21.14.

<code>expose_php</code>	On	
<code>extension_dir</code>	<code>c:/wamp/bin/php/php5.2.5/ext/</code>	<code>c:/wamp/bin/php/php5.2.5/ext/</code>
<code>upload_max_filesize</code>	On	On

Figura 21.14

Abra la carpeta indicada y pegue el archivo **ZendDebugger.dll** desde el portapapeles de Windows. Ahora modificamos el archivo de configuración de PHP, llamado **php.ini**. Para ello vamos a abrir, de nuevo, el menú de opciones de WampServer que veíamos en la figura 21.12. Seleccionaremos **PHP** y, en la lista que aparece, haremos clic sobre **php.ini**, con lo que este archivo, de texto plano, se abre en el editor por defecto. Al final del fichero añada el siguiente párrafo:

```
[Zend]
zend_extension_ts=c:/wamp/bin/php/php5.2.5/ext/ZendDebugger.dll
zend_debugger.allow_hosts=127.0.0.1
zend_debugger.expose_remotely=always
```

Fíjese en la dirección IP de su servidor local (127.0.0.1) que le indica a PHP qué ordenador puede hacer uso de la extensión que estamos instalando. Grabe los cambios y cierre el editor de texto. Después, reinicie WampServer y recargue la página **info.php**. Busque el copyright de Zend, como se ve en la figura 21.15.

This program makes use of the Zend Scripting Language Engine:  
 Zend Engine v2.2.0, Copyright (c) 1998-2007 Zend Technologies  
 with Zend Debugger v5.2.15, Copyright (c) 1999-2008, by Zend Technologies



Figura 21.15

Preste especial atención a la reseña **with Zend Debugger v5.2.15** que debe aparecer en el recuadro. Esto nos indica que ya está correctamente instalada la extensión que nos ocupa aunque, como ya hemos mencionado, aún debemos ver, más adelante, cómo hacer uso de ella.

## 21.3 ECLIPSE + PDT

Lo siguiente que vamos a hacer es obtener un IDE adecuado para trabajar. Un IDE (*Integrated Development Environment*, Entorno Integrado de Desarrollo) es un programa para facilitar el trabajo con determinadas herramientas (en este caso, con PHP). Como la mayoría de las aplicaciones, los hay gratuitos y comerciales. Nosotros vamos a usar un IDE gratuito que, además, es de los mejores que hay en el mercado. Se trata de Eclipse con PDT (*PHP Development Tools*, Herramientas de Desarrollo en PHP). Para instalarlo debemos contar, previamente, con un entorno de Java JRE en nuestra plataforma Windows (para usuarios de Linux, todas las distribuciones actuales cuentan con Java instalado). Lo descargamos en la web [http://java.com/es/download/windows\\_xpi.jsp?locale=es&host=java.com](http://java.com/es/download/windows_xpi.jsp?locale=es&host=java.com). Ejecute el fichero que descargue, y el entorno JRE quedará instalado en su ordenador. A continuación debemos descargar el IDE Eclipse, de la siguiente URL: <http://www.eclipse.org/downloads/download.php?file=/technology/epp/downloads/release/galileo/SR2/eclipse-php-galileo-SR2-win32.zip>. A fin de facilitar la descarga, vaya a la dirección <http://www.eclipse.org> y pulse el enlace **Downloads**. En la página de descargas, pulse sobre el enlace **Eclipse for PHP Developers**, que aparece reproducido en la figura 21.16.



Figura 21.16

Esto le lleva, directamente, a la página de descarga. Pulse sobre el enlace reproducido en la figura 21.17 (el mirror referenciado en el enlace puede ser diferente cuando cargue la página).

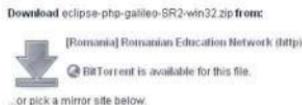


Figura 21.17

Una vez descargado el archivo, descomprímalo en, por ejemplo, su carpeta **Archivos de programa, Program files** o similar. Al extraer los contenidos del archivo comprimido, se encontrará con una carpeta llamada **eclipse**. Dentro de ésta, existe un archivo llamado **eclipse.exe**. Saque un acceso directo al escritorio, para poder ejecutar Eclipse desde allí. Como ve, este producto no necesita instalación previa. Simplemente se descarga y se usa.

**Atención:** Para poder usar Eclipse para PHP (Eclipse + PDT) es necesario tener instalado un entorno de ejecución de Java (JRE, *Java Runtime Environment*) o una máquina virtual de Java (JVM, *Java Virtual Machine*), tal como se menciona al principio de esta sección. En caso contrario, Eclipse dará un error y no se podrá ejecutar.

Al poner en marcha Eclipse, lo primero que encontramos, tras la carátula de presentación, es una ventana como la de la figura 21.18.

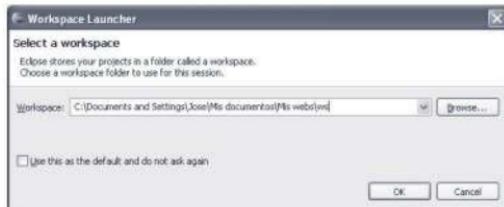


Figura 21.18

Esto es muy importante. Tanto que, hasta que no confirmemos este paso, Eclipse no puede ponerse en funcionamiento. Un **espacio de trabajo (Work Space)** es un contenedor para almacenar nuestros proyectos, nuestras páginas y todo lo que desarrollemos desde Eclipse. Cada proyecto debe estar en un espacio de trabajo (aunque, en un mismo espacio de trabajo puedan coexistir más proyectos, lo que, desde luego, es desaconsejable a efectos de organización).

Aunque podemos colocar nuestro espacio de trabajo en el lugar que deseemos, lo lógico es que se encuentre dentro de la carpeta que hemos definido como localhost. A su vez, dentro de esta ruta, definimos el nombre de nuestro espacio de trabajo. En el ejemplo de la figura 21.18, lo hemos llamado **ws** (la imaginación al poder).

La casilla **Use this as the default and do not ask again** no la vamos a marcar por el momento. Si lo hicieramos, Eclipse tomaría el espacio de trabajo que

estamos creando como defectivo, y no nos volvería a preguntar al respecto en sucesivos usos. Esto puede tener utilidad si trabajamos en algún proyecto grande y laborioso que nos vaya a tener ocupados durante muchas sesiones. Si andamos trabajando en varios proyectos, lo más normal es que usemos diferentes espacios de trabajo en diferentes sesiones. No obstante, si seleccionamos esta casilla, luego podremos cambiar de espacio de trabajo mediante el menú **File**, opción **Switch Workspace**.

Tras pulsar el botón **OK** accedemos a la pantalla de bienvenida de Eclipse, mostrada en la figura 21.19.



Figura 21.19

Esta pantalla sólo aparece cada vez que se crea un nuevo espacio de trabajo, no si usamos uno ya existente. Los cinco iconos que aparecen en la pantalla, distribuidos de forma aparentemente caprichosa, tienen un significado que muestran al pasar el puntero del ratón por encima de cada uno de ellos. Éstos significados son los siguientes:

- **Overview.** Muestra una vista global de las características del producto.
- **What's new.** Informa de las novedades habidas con respecto a anteriores versiones.
- **Samples.** Da paso a algunos ejemplos de desarrollo de aplicaciones.
- **Tutorials.** Muestra tutoriales de funcionamiento de Eclipse (en inglés, lamentablemente).
- **Workbench.** La palabra en cuestión se traduce por “Banco de trabajo” y da paso al entorno de desarrollo como tal. Esta es, pues, la opción que vamos a ver a continuación.

Al acceder al banco de trabajo desde el correspondiente ícono (o, directamente, si hemos optado por un espacio de trabajo ya existente), nos encontramos con la interfaz principal del programa, mostrada en la figura 21.20. Como ve, está dividida en varias zonas.

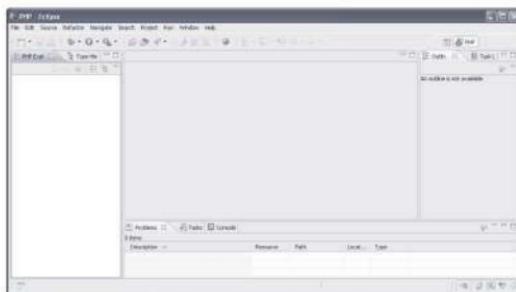


Figura 21.20

En la parte superior encontramos la barra de menús y, debajo de ésta, la barra de botones: lo típico de cualquier aplicación en entorno Windows.

Debajo de estas barras, a la izquierda tenemos el explorador del espacio de trabajo. Aquí aparecerán iconos de los proyectos que desarrollemos, así como de los ficheros, scripts, etc., que formen parte de cada proyecto.

En la zona central tenemos la ventana donde se mostrará (y se podrá editar) el script de PHP en el que estemos trabajando en un momento dado.

A la derecha encontramos otra ventana donde se mostrarán las variables, matrices y otros datos que use nuestro script.

Debajo de la ventana central y derecha vemos una ventana apaisada donde aparecerán, entre otras cosas, detalles acerca de posibles errores de funcionamiento del script en el que estemos trabajando.

**Atención:** Si tiene curiosidad por ello (y, si ha llegado a este punto doy por sentado que así es), entre, desde el explorador de Windows, en la carpeta que usa como localhost. Verá que, dentro de ella, se ha creado una carpeta con el nombre del espacio de trabajo que hemos definido. En el interior de esta carpeta, existen otras carpetas y algunos ficheros que Eclipse usa, internamente, para trabajar, aunque no formarán parte de nuestros proyectos.

Llegado a este punto, vamos a crear un nuevo proyecto. Tenga en cuenta que, tanto por razones técnicas, como lógicas a efectos de organización, todos los scripts con los que trabajemos deben formar parte de un proyecto. Para crear un nuevo proyecto de PHP, pulsamos el menú **File** y, dentro de este, la opción **New**. A su vez, elegimos **PHP Project** y se nos abrirá la ventana de la figura 21.21.



Figura 21.21

En esta ventana se nos pide un nuevo nombre para el proyecto. Éste es necesario. En nuestro ejemplo lo vamos a llamar **Proyecto 1**. Escriba este nombre en la casilla de la parte superior, etiquetada como **Project name**, y pulse el botón **Finish**, dejando el resto de las opciones como aparecen por defecto. En la ventana del explorador del espacio de trabajo vemos ahora un ícono que representa el proyecto que acabamos de crear, tal como se ve en la figura 21.22.



Figura 21.22

**Atención:** Observe que en la carpeta del espacio de trabajo se ha creado otra carpeta relativa al proyecto, con los datos básicos que Eclipse necesita para funcionar.

Haga doble clic sobre el nombre del proyecto, tal como aparece resaltado, y verá que su aspecto cambia al de la figura 21.23.



Figura 21.23

En realidad, lo que aquí vemos es que no vemos nada. Me explico: no vemos nada relevante, a excepción de los datos que el propio Eclipse manejará. Aún no existe ningún script en el proyecto. Apoye el puntero del ratón sobre el ícono del proyecto y haga clic con el botón secundario. En el menú que se despliega, elija la opción **New > y**, dentro de ésta, seleccione **PHP File**. Se abrirá la ventana de la figura 21.24.

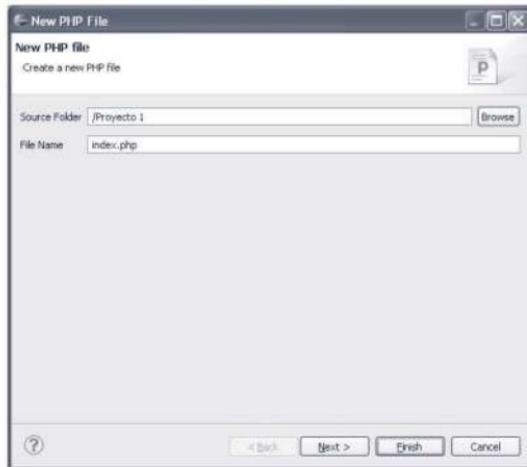


Figura 21.24

No cambie el contenido de la casilla **Source Folder**, que se refiere a la carpeta donde se almacenará el nuevo script. Por defecto le aparece el nombre de la carpeta relativa al proyecto en el que estamos trabajando. Ésta es la que nos interesa ahora.

En la casilla **File Name**, teclee **index.php**, como se ve en la imagen. A continuación, pulse el botón **Finish**. El nombre del script aparece en la ventana de la izquierda, “colgando” del nombre del proyecto. En la ventana central aparece el título del script (en la pestaña superior) y ya puede editar directamente el código, tal como se muestra en la figura 21.25.



Figura 21.25

A partir de aquí, podemos crear un script escribiendo el código, y pasar a depurarlo en busca de los errores más molestos y difíciles de detectar que se pueden dar en un script. Los que tienen lugar cuando se usan variables o datos con nombres distintos a los que tienen.

## 21.4 DEPURANDO PHP

En la ventana central vamos a escribir un primer script muy simple, como el mostrado a continuación:

```
<?php
    $miCadena = "Hola, Mundo.";

    function saludar($aQuien){
        echo $aQuien."<br />";
        echo "Hoy es ".date("d-m-Y");
    }

    saludar($miCadena);
?>
```

Como ve, es un código muy sencillito, que no tiene nada especial, pero que nos va a resultar muy útil en este momento. Lo grabamos con el menú **File**, opción **Save** y pulsamos la flechita negra que aparece junto a un botón verde en la barra de herramientas, tal como se ve en detalle ampliado en la figura 21.26.



Figura 21.26

Al pulsar la flechita negra, se abre un menú en el que seleccionamos la opción **Run configurations...**. Aquí lo que más nos importa es la última parte de la pestaña **Server**, rotulada como **URL**. Debemos desmarcar, si aparece marcada, la casilla de verificación etiquetada como **Auto Generate**. En la casilla de texto editable que aparece a la derecha de la etiqueta **URL**: debemos asegurarnos de que esté escrita toda la ruta de acceso al script que estamos editando, incluyendo el nombre de la carpeta del espacio de trabajo y el de la carpeta del proyecto. En conjunto, la sección **URL** de la ventana **Run Configurations** debe quedar como se ve en el corte en detalle de la figura 21.27.



Figura 21.27

Pulse el botón Run y verá cómo en la ventana central se abre una pestaña de ejecución donde vemos la página que se genera con el script que hemos escrito tal como se ve en la figura 21.28.



Figura 21.28

Pulse sobre la pestaña con el nombre del archivo que aparece en la parte superior izquierda de esta venta, para volver al modo de edición. Observe la ventana de la parte derecha del banco de trabajo de eclipse, en la pestaña rotulada como **Outline**. Observe que le aparece la lista de todas las variables y funciones que incluye el script. Como es lógico, esto es muy útil a la hora de rastrear el funcionamiento de una página. Tiene un corte en detalle en la figura 21.29.



Figura 21.29

Vamos a crear un nuevo proyecto, tal como ya sabemos hacerlo, dentro del mismo espacio de trabajo, y lo vamos a llamar **Proyecto 2**. A su vez, dentro de este proyecto, vamos a crear un script llamado **sintaxis.php**. La estructura de proyectos de la ventana izquierda queda como se ve en la figura 21.30.

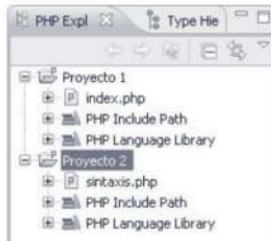


Figura 21.30

El listado del script **sintaxis.php** es el que aparece a continuación:

```
<?php
    variable $miVariable = "Primer error";
    cosa miFuncionErronea(){
        declaremos otroError = array();
    }
?>
```

Como ve, es un cúmulo de errores de sintaxis. En realidad sería difícil acumular tantos despropósitos en tan corto script, pero se trata de “forzar” un poco

la situación, con fines meramente didácticos. Cuando grabamos el script, la ventana de la parte inferior muestra, en la pestaña **Problems**, los errores de sintaxis detectados, como se ve en la figura 21.31.

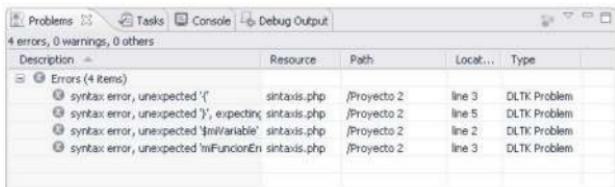


Figura 21.31

Éste es el primer paso de depuración de un script: asegurarnos de que no tenga errores de sintaxis. También es el más cómodo de ejecutar: con sólo grabar el script, se muestran los errores que haya. Además, aparece una indicación en la ventana de edición, al lado de cada línea que tiene un error, tal como se ve en la figura 21.32.



Figura 21.32

Además, cuando un script presenta errores de sintaxis, aparece una marca adicional en el nombre del proyecto correspondiente, en la ventana de proyectos, como se ve en la figura 21.33.

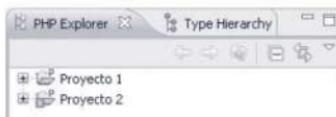


Figura 21.33

La siguiente fase de depuración de un script es (o puede llegar a ser, en muchos scripts) mucho más compleja y, a la vez, más relevante. La mayoría de los fallos de ejecución de un script (cuando éste está bien concebido y la sintaxis de las líneas es la correcta) tienen lugar a causa de declarar una variable (o matriz, si a eso vamos) con un nombre y pretender usarla, más adelante, con otro nombre diferente. Llegados a este punto, sería utilísimo poder obtener una lista de las variables, matrices y funciones que emplea nuestro script y ver su estado en cada momento de la ejecución, comprobando los valores que contienen y cómo afectan a otros datos. El IDE Eclipse, en combinación con la extensión Zend Debugger que hemos añadido a PHP nos permite, exactamente, eso.

Para ver como se usa esta funcionalidad, vamos a crear un tercer proyecto, al que llamaremos **Proyecto 3** y, dentro de éste, un script al que llamaremos **index.php**. Su listado es el siguiente:

```
<?php

$meses = Array(Array("mes"=>"Enero",
" dias"=>"31"), Array("mes"=>"Febrero", "dias"=>"28"),
Array("mes"=>"Marzo", "dias"=>"31"), Array("mes"=>"Abril",
" dias"=>"30"), Array("mes"=>"Mayo", "dias"=>"31"),
Array("mes"=>"Junio", "dias"=>"30"), Array("mes"=>"Julio",
" dias"=>"31"), Array("mes"=>"Agosto", "dias"=>"31"),
Array("mes"=>"Septiembre", "dias"=>"30"),
Array("mes"=>"Octubre", "dias"=>"31"),
Array("mes"=>"Noviembre", "dias"=>"30"),
Array("mes"=>"Diciembre", "dias"=>"31"));

$numeroDeMeses = count($meses);

for($contador = 0; $contador<$numeroDeMeses;
$contador ++){
    echo "El mes de ";
    echo $meses[$contador] ["mes"];
    echo " tiene ";
    echo $meses[$contador] ["dias"];
    echo " días.<br/>";
}

?>
```

Como ve, se trata de crear una matriz bidimensional, muy sencillita, indexada, en la que cada elemento es, a su vez, una matriz asociativa, con los nombres de los doce meses del año y los días que tiene cada mes. A continuación, un bucle recorre la matriz y muestra los datos.

La cuestión es poder ver, no sólo la lista de variables, sino el estado de cada una de ellas en un momento dado. Para ello, una vez grabado el script, vamos a poner el IDE en el modo de vista de depuración. Esto podemos hacerlo pulsando el menú **Window**. Dentro de éste seleccionaremos **Open Perspective y**, a su vez, **PHP Debug**. El aspecto general del IDE cambia al del la figura 21.33.

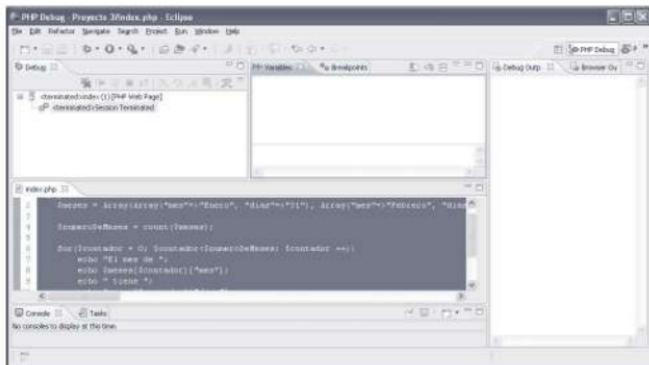


Figura 21.33

En primer lugar, vaya a la ventana donde aparece el listado. Ponga el puntero en la columna que hay a la izquierda, donde están los números de línea, y haga doble clic sobre el número de la línea 11. Verá que aparece un topito de color azul al lado del número de línea, tal como se aprecia en la figura 21.34.

A screenshot of the Eclipse code editor showing the file 'index.php'. The code defines an array of months and prints them out. Line 11 is highlighted with a small blue dot, indicating it is the current line of execution. The code is as follows:

```
1 <?php
2     $meses = Array(Array("mes"
3
4     $numeroDeMeses = count($me
5
6     for($contador = 0; $contad
7         echo "El mes de ";
8         echo $meses[$contador]
9         echo " tiene ";
10        echo $meses[$contador]
11        echo " días: ".date("d").".<br>
12    }
13
14 ?>
```

Figura 21.34

Este topito es lo que se conoce como un **punto de ruptura**. Los puntos de ruptura (o **breakpoints**) son interrupciones forzadas que creamos para poder ejecutar el script paso a paso, a fin de comprobar el estado de las variables en cada momento. Usted puede establecer en su listado tantos puntos de ruptura como desee, haciendo doble clic sobre los números de línea. Si hace doble clic sobre un número de linea que ya tenga un punto de ruptura, éste desaparece. Como hemos incluido el punto de ruptura en una linea que está en un bucle, la ejecución de depuración se detendrá en cada iteración del bucle.

Para activar la ejecución en modo de depuración pulse el botón **Debug**, situado en la barra de herramientas del IDE, cuyo aspecto recuerda a un pequeño insecto (bug), tal como se ve en el detalle de la figura 21.35.



Figura 21.35

Con esto, el script se ejecuta hasta que se alcanza el primer punto de ruptura. En ese momento, podemos ver las variables y matrices que existen, junto con sus valores, en la ventana variables, como vemos en la muestra de la figura 21.36.

Name	Value
\$_POST	Array [0]
\$_GET	Array [9]
\$_COOKIE	Array [2]
\$_FILES	Array [0]
\$meses	Array [12]
\$numeroDeMeses	(int) 12
\$contador	(int) 1

Arreglo [0]

Figura 21.36

Observe también la ventana **Debug**, situada en la parte superior izquierda, y reproducida en la figura 21.37.

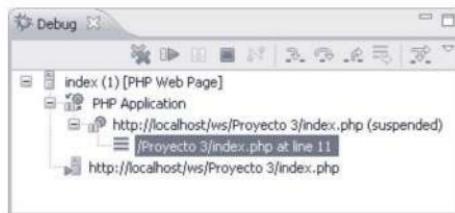


Figura 21.37

Esta ventana contiene detalles acerca del script que se está depurando. Además tiene un botón, en la parte superior, en forma triangular, reproducido en el detalle de la figura 21.38. Éste se usa para continuar la ejecución después de cada ruptura, a fin de comprobar cómo va cambiando el estado de las variables.



Figura 21.38

El programa tiene muchas más opciones y herramientas complementarias, por lo que le aconsejo que experimente con él. No se lo he incluido en el CD porque, pese a ser una herramienta gratuita, los autores no permiten la libre distribución, así que descárguelo de Internet.

En la carpeta correspondiente a este capítulo, en el CD adjunto, tiene el espacio de trabajo completo que hemos usado en los ejercicios del texto.



## APÉNDICE A

# LA CONFIGURACIÓN DEL INTÉRPRETE

---

---

Cuando usted instala un servidor de documentos web, tanto si éste es una máquina remota como si se trata de su propio equipo local, y además instala el intérprete de PHP, éste funciona de una manera predeterminada, con unas prestaciones que tiene a priori. Sin embargo, es posible configurarlo de modo que cambie esas prestaciones. A lo mejor, usted se encuentra con el hecho de que, determinadas funciones que hemos estudiado en este libro no son reconocidas por el intérprete, generándose un error. Eso no es, necesariamente, un fallo de PHP, ni un error del libro. Es cuestión de tener las *extensiones* adecuadas funcionando. Por ejemplo, en el capítulo 12, donde se habla de imágenes, se menciona que hay que trabajar con la librería GD2; en el capítulo 17, se trabaja con bases de datos MySQL, que necesitan, también, una extensión adecuada.

Para que todo funcione adecuadamente, la instalación de PHP debe contar con las extensiones adecuadas, y tener establecidas determinadas directivas.

### A.1 LAS EXTENSIONES

Si usted instaló PHP en su equipo a partir de la aplicación AppServ, tal como se describe en el capítulo 2, tiene una carpeta llamada **ext**, donde se almacenan las librerías dinámicas de las extensiones (archivos .dll para Windows). Esta carpeta está en la siguiente ruta:

**C:/AppServ/php/ext**

Dentro de esta carpeta debe tener los siguientes archivos:

- `php_bz2.dll`
- `php_curl.dll`
- `php_dba.dll`
- `php_dbase.dll`
- `php_exif.dll`
- `php_fdf.dll`
- `php_filepro.dll`
- `php_gd2.dll`
- `php_gettext.dll`
- `php_ifx.dll`
- `php_imap.dll`
- `php_interbase.dll`
- `php_ldap.dll`
- `php_mbstring.dll`
- `php_mcrypt.dll`
- `php_mhash.dll`
- `php_mime_magic.dll`
- `php_ming.dll`
- `php_msql.dll`
- `php_mssql.dll`
- `php_mysql.dll`
- `php_mysqli.dll`
- `php_oci8.dll`
- `php_openssl.dll`
- `php_pdo.dll`
- `php_pdo_firebird.dll`
- `php_pdo_mssql.dll`
- `php_pdo_mysql.dll`
- `php_pdo_oci.dll`
- `php_pdo_ocis8.dll`
- `php_pdo_odbc.dll`
- `php_pdo_pgsql.dll`
- `php_pdo_sqlite.dll`
- `php_pgsql.dll`
- `php_psell.dll`
- `php_shmop.dll`
- `php_snmp.dll`
- `php_soap.dll`
- `php_sockets.dll`
- `php_sqlite.dll`
- `php_sybase_ct.dll`
- `php_tidy.dll`
- `php_xmlreader.dll`

- `php_xmlrpc.dll`
- `php_xsl.dll`

La lista anterior se compone de un total de 45 archivos. Dado que aparecen en orden alfabético, le será muy fácil comprobar si están todos. Si le faltase alguno (cosa que no es habitual) desinstale y vuelva a instalar AppServ.

## A.2 LAS DIRECTIVAS

Además de los archivos antes mencionados hay uno al que le debe prestar especial atención. Se llama `php.ini` y se encuentra, en sistemas Windows, en el directorio **C:/Windows**, en el disco duro principal. En distribuciones Linux consulte la documentación de su distribución específica.

Abra este archivo con su editor de texto plano. No se asuste por el tamaño que tiene, de más de mil cien líneas. La mayoría de ellas empiezan con un punto y coma (;). Esto indica que esa línea es un comentario, con lo que el intérprete no la tiene en cuenta. Usted podría borrar todas las líneas que presentan esta característica y el intérprete seguiría funcionando igual que antes. Sin embargo, le desaconsejo esto, ya que muchos comentarios son muy interesantes para nosotros.

Ahora vamos a fijarnos en las líneas que no son comentarios, las que contienen las directivas que determinan muchos aspectos del comportamiento del intérprete. Éstas están divididas en bloques, en base a los distintos aspectos del intérprete que controlan. Cada bloque se inicia con el nombre del mismo encerrado entre corchetes. Además, dentro de los bloques, las directivas están clasificadas siguiendo criterios lógicos, en sub-bloques, que se inician con unas líneas de comentarios. Empecemos por buscar lo siguiente:

```
;;;;;;;;;;;  
; Dynamic Extensions ;  
;;;;;;;;;;;
```

Este apartado determina qué extensiones se cargarán al iniciarse el servidor. Por ejemplo, mire la extensión relativa a la librería gráfica GD2, unas líneas más abajo:

```
extension=php_gd2.dll
```

Si la línea aparece como un comentario (precedida por punto y coma), la extensión no se cargará. Muchas de estas líneas se corresponden con algunos de los

archivos .dll que hay en la carpeta de extensiones vista en el apartado anterior, aunque no necesariamente todos los archivos están mencionados en estas líneas. Volviendo al ejemplo de la librería gráfica, si la línea que le he mostrado está comentada es como si no existiera el archivo php\_gd2.dll. No podrá usar en sus páginas las funciones gráficas. Mi consejo es que, en esta parte donde se definen las extensiones, quite los signos de punto y coma situados al principio en cada una de las líneas que tienen el aspecto **extensión=php\_archivo.dll**, cuyos nombres de archivo coincidan con alguno de la carpeta de extensiones. No es mucho trabajo y merece la pena. Esta parte del archivo de configuración quedará con un aspecto similar al siguiente:

```
extension=php_mbstring.dll
extension=php_bz2.dll
;extension=php_curl.dll
extension=php_dba.dll
extension=php_dbase.dll
extension=php_exif.dll
;extension=php_fdf.dll
extension=php_filepro.dll
extension=php_gd2.dll
extension=php_gettext.dll
extension=php_ifx.dll
extension=php_imap.dll
;extension=php_interbase.dll
extension=php_ldap.dll
;extension=php_mcrypt.dll
;extension=php_mhash.dll
extension=php_mime_magic.dll
extension=php_ming.dll
;extension=php_mssql.dll
;extension=php_msqli.dll
extension=php_mysql.dll
;extension=php_oci8.dll
;extension=php_openssl.dll
;extension=php_oracle.dll
extension=php_pgsql.dll
extension=php_shmop.dll
extension=php_snmp.dll
extension=php_sockets.dll
;extension=php_sqlite.dll
;extension=php_sybase_ct.dll
extension=php_tidy.dll
extension=php_xmlrpc.dll
extension=php_xsl.dll
```

Además, es necesario que el intérprete “sepa” en qué ruta debe buscar las extensiones dinámicas. Busque una línea que empieza por **extension\_dir**. Mediante esta directiva, se establece la ruta donde buscar los archivos .dll adecuados. En mi caso, habiendo instalado el intérprete como se describe en el capítulo 2, la linea que nos ocupa tiene el siguiente aspecto:

```
extension_dir = "C:/AppServ/php/ext/"
```

Tenga en cuenta una cosa. Si activa la linea que se refiere a una extensión dinámica, ésta deberá estar disponible en el directorio especificado. Muchas extensiones no son parte de la instalación por defecto de PHP, sino que deben ser obtenidas aparte. Algunas son libres y otras de pago. Si usted activa en el fichero php.ini la línea que corresponde a una extensión (quitándole el punto y coma del principio), al reiniciar el sistema, se intentará cargar la extensión correspondiente, si ésta no existe, verá en el monitor de su ordenador un mensaje como el de la figura A.1.



Figura A.1

Si eso le sucede al reiniciar su ordenador, tome nota de la extensión mencionada (en el ejemplo es `php_curl.dll`). Si dispone del fichero, instálelo en la carpeta indicada; si no, anule la línea, en el archivo de configuración, precediéndola, como ya sabe, con un punto y coma. Aparte de la configuración de las extensiones dinámicas, el fichero `php.ini` cuenta con otras directivas. Algunas de ellas debemos conocerlas, por ser de uso muy común o muy específico. Aparecen en la tabla de la figura A.2, con su nombre, el valor por defecto y su uso.

### DIRECTIVAS DEL FICHERO `php.ini`

Directiva	Valor por defecto	Uso
<code>engine</code>	<code>On</code>	Cuando está activada, permite que el lenguaje PHP corra bajo un servidor Apache.
<code>asp_tags</code>	<code>Off</code>	Cuando está activada permite delimitar el código PHP con las etiquetas <code>&lt;%</code> y <code>%&gt;</code> , en lugar de <code>&lt;?php</code> y <code>?&gt;</code> .

<b>DIRECTIVAS DEL FICHERO php.ini (cont.)</b>		
<b>Directiva</b>	<b>Valor por defecto</b>	<b>Uso</b>
<b>precision</b>	<b>12</b>	El máximo número de dígitos decimales para valores en coma flotante.
<b>output_buffering</b>	<b>Off</b>	Cuando está activada, permite enviar cabeceras incluso después de haber enviado contenidos al navegador. Se recomienda no usar esta característica, ya que sobrecarga de trabajo el servidor.
<b>register_globals</b>	<b>Off</b>	Cuando está activada, permite que las variables de formulario, cookies o sesiones generen, automáticamente, variables de memoria homónimas. Así pues, si existe en un script la variable \$_POST["variable"], se puede acceder a su contenido mediante \$variable.
<b>file_uploads</b>	<b>On</b>	Cuando está activada, permite el envío al servidor de ficheros del cliente a través de formularios, tal como se describe en el capítulo 5.
<b>short_open_tag</b>	<b>On</b>	Permite abrir código PHP usando <? en lugar de <?php.
<b>allow_url_fopen</b>	<b>On</b>	Permite el acceso a ficheros remotos.
<b>post_max_size</b>	<b>8M</b>	Establece el tamaño máximo de datos que se pueden enviar desde un formulario mediante post.

Figura A.2

Aunque la lista de las directivas de PHP es sensiblemente mayor que la aquí expuesta, éstas son las que, con mayor frecuencia, necesitará conocer y/o modificar.

Cuando cambie el valor de una directiva necesitará reiniciar el servidor Apache para que los nuevos valores sean operativos. Para ello, puede situarse en la carpeta C:\AppServ\Apache y hacer doble clic sobre los siguientes iconos:

- apache\_serviceuninstall.bat
- apache\_stop.bat

- apache\_start.bat
- apache\_serviceinstall.bat

Como alternativa simple, puede reiniciar su ordenador. Al arrancar de nuevo y cargarse el servidor Apache, los nuevos valores de las directivas de php.ini se asumirán automáticamente.

Una reseña final. Si usted usa una distribución Linux en lugar de Windows, las extensiones terminan con .so en lugar de hacerlo con .dll.



## APÉNDICE B

### PALABRAS RESERVADAS DE PHP

---

---

Este apéndice contiene una relación de las palabras reservadas de PHP. Éstas NO deben ser usadas NUNCA como nombres de variables, matrices, objetos, clases, funciones de usuario, constantes, ni nada que se pueda definir por programación. Hacerlo dará lugar a errores que, en ocasiones, son, irónicamente, muy difíciles de localizar por lo obvio de los mismos.

No todas las palabras que aparecen en esta lista son habituales de la programación en PHP, por las siguientes razones:

- Algunas proceden de versiones anteriores del lenguaje (se han quedado obsoletas).
- Otras no se usan, pero están reservadas para versiones posteriores.
- También hay muchísimas que no son palabras clave de PHP, propiamente dicho, sino que pertenecen a extensiones que se pueden instalar aparte.
- Otras son alias de funciones actuales.

En todo caso, nos abstendremos de usarlas para los fines descritos en el párrafo anterior.

#### A

**abstract**  
**add\_root**

**addaction**  
**addcolor**  
**addfill**  
**addshape**  
**addstring**

align  
and  
array  
as  
attributes

**B**

Break

**C**

case  
catch  
children  
chop  
class  
clone  
close  
closedir  
com\_get  
com\_propput  
com\_propget  
com\_propset  
com\_set  
const  
continue  
count  
current  
cv\_add  
cv\_auth  
cv\_command  
cv\_count  
cv\_delete  
cv\_done  
cv\_init  
cv\_lookup  
cv\_new  
cv\_report  
cv\_return  
cv\_reverse  
cv\_sale  
cv\_void

cv\_status  
cv\_textvalue  
ccvs\_add  
ccvs\_auth  
ccvs\_command  
ccvs\_count  
ccvs\_delete  
ccvs\_done  
ccvs\_init  
ccvs\_lookup  
ccvs\_new  
ccvs\_report  
ccvs\_return  
ccvs\_reverse  
ccvs\_sale  
ccvs\_status  
ccvs\_textvalue  
ccvs\_void

**D**

declare  
default  
die  
dir  
diskfreespace  
disk\_free\_space  
do  
domxml\_add\_root  
domxml\_attributes  
domxml\_attrname  
domxml\_children  
domxml\_dumpmem  
domxml\_getattr  
domxml\_get\_attribute  
domxml\_intdtd  
domxml\_last\_child  
domxml\_new\_child  
domxml\_new\_xmldoc  
domxml\_node  
domxml\_parent  
domxml\_root  
domxml\_set\_attribute  
domxml\_set\_content  
domxml\_set\_attribute  
domxml\_setattr

domxml\_unlink\_node  
doubleval  
drawarc  
drawcircle  
drawcubic  
drawcubicto  
drawcurve  
drawcurveto  
drawglyph  
drawline  
drawlineto  
dtd  
dumpmem

**E**

echo  
else  
elseif  
empty  
enddeclare  
endfor  
endforeach  
endif  
endswitch  
endwhile  
eval  
exit  
exception  
extends

**F**

fbsql  
fbsql\_db\_query  
final  
floatval  
for  
foreach  
fputs  
function  
fwrite

get\_attribute  
getascent  
getascent  
getattr  
getdescent  
getdir  
getheight  
getleading  
getleading  
getshape1  
getshape2  
gettext  
getwidth  
getwidth  
getwidth  
global  
gzputs  
gzwrite

**G**

highlight\_file

**H**

if  
i18n\_convert  
i18n\_discover\_encoding  
i18n\_http\_input  
i18n\_http\_output  
i18n\_internal\_encoding  
i18n\_ja\_jp\_hantozan  
i18n\_mime\_header\_decode  
i18n\_mime\_header\_encode  
imap\_body  
imap\_create  
imap\_createmailbox  
imap\_fetchtext  
imap\_getmailboxes  
imap\_getsubscribed

**I**

imap\_header  
 imap\_headerinfo  
 imap\_list  
 imap\_list\_full  
 imap\_listmailbox  
 imap\_listscan  
 imap\_scanmailbox  
 imap\_listsubscribed  
 imap\_lsub  
 imap\_lsub\_full  
 imap\_rename  
 imap\_renamemailbox  
 imap\_scan  
 implements  
 implode  
 include  
 include\_once  
 ini\_alter  
 ini\_set  
 interface  
 is\_double  
 is\_float  
 is\_integer  
 is\_int  
 is\_long  
 is\_int  
 is\_real  
 is\_writeable  
 is\_writable  
 isset

**J**

join

**L**

labelframe  
 labelframe  
 last\_child  
 lastchild  
 ldap\_close  
 ldap\_unbind

list

**M**

magic\_quotes\_runtime  
 mb\_convert\_encoding  
 mb\_convert\_kana  
 mb\_decode\_mimeheader  
 mb\_detect\_encoding  
 mb\_http\_input  
 mb\_http\_output  
 mb\_internal\_encoding  
 mb\_strcut  
 mb\_strlen  
 mbstrpos  
 mb\_stripos  
 mb\_substr  
 mbstrlen  
 mbstrpos  
 mbstrripos  
 mbsubstr  
 ming\_SetCubicThreshold  
 ming\_SetCubicThreshold  
 ming\_setscale  
 ming\_setScale  
 move  
 moveopen  
 movepento  
 moveto  
 moveto  
 moveto  
 msql  
 msql\_create\_db  
 msql\_createdb  
 msql\_db\_query  
 msql\_dbname  
 msql\_drop\_db  
 msql\_dropdb  
 msql\_field\_flags  
 msql\_field\_len  
 msql\_field\_name  
 msql\_field\_table  
 msql\_field\_type  
 msql\_fieldflags  
 msql\_fieldlen

mysql_fieldname	mysql_field_flags
mysql_fieldtable	mysql_field_len
mysql_fieldtype	mysql_field_name
mysql_free_result	mysql_field_table
mysql_freeresult	mysql_field_type
mysql_list_dbs	mysql_fieldflags
mysql_list_fields	mysql_fieldlen
mysql_list_tables	mysql_fieldname
mysql_listdbs	mysql_fieldtable
mysql_listfields	mysql_fieldtype
mysql_listtables	mysql_free_result
mysql_num_fields	mysql_freeresult
mysql_num_rows	mysql_list_dbs
mysql_numfields	mysql_listdbs
mysql_numrows	mysql_list_fields
mysql_regcase	mysql_list_tables
mysql_result	mysql_listfields
mysql_select_db	mysql_listtables
mysql_selectdb	mysql_num_fields
mysql_tablename	mysql_numfields
mssql_affected_rows	mysql_num_rows
mssql_close	mysql_numrows
mssql_connect	mysql_result
mssql_data_seek	mysql_select_db
mssql_fetch_array	mysql_selectdb
mssql_fetch_field	mysql_tablename
mssql_fetch_object	
mssql_fetch_row	
mssql_field_seek	
mssql_free_result	
mssql_get_last_message	
mssql_min_client_severity	name
mssql_min_message_severity	new
mssql_min_server_severity	new_child
mssql_num_fields	new_xmldoc
mssql_num_rows	nextframe
mssql_pconnect	nextframe
mssql_query	node
mssql_result	
mssql_select_db	
mysql	
mysql_create_db	oci8append
mysql_createdb	oci8assign
mysql_db_name	oci8assignelem
mysql_db_query	oci8close
mysql_dbname	oci8free
mysql_drop_db	oci8getelem
mysql_dropdb	

**N**

name  
new  
new\_child  
new\_xmldoc  
nextframe  
nextframe  
node

**O**

oci8append  
oci8assign  
oci8assignelem  
oci8close  
oci8free  
oci8getelem

oci8load	public
oci8max	
oci8ocifreecursor	
oci8save	
oci8savefile	
oci8size	
oci8trim	recode
oci8writetemporary	recode_string
oci8writetofile	remove
ocicloselob	require
ocicollappend	require_once
ocicollassign	return
ocicollassignelem	rewind
ocicollgetelem	rewaddir
ocicollmax	root
ocicollsize	rotate
ocicoltrim	rotateto
ocifreecoll	rtrim
ocifreedesc	
ocifreestatement	
ocieloadlob	save
ocisavelob	savetofile
ocisavelobfile	scale
ociwritetemporarylob	scaleto
ociwritelobtofile	set
odbc_do	set_attribute
odbc_exec	set_content
odbc_field_precision	set_magic_quotes_runtime
odbc_field_len	setaction
or	setattr
output	setbackground
	setbounds
	setcolor
	setdepth
	setdimension
<b>P</b>	setdown
parent	setFont
pdf_add_outline	setframes
pdf_add_bookmark	setheight
pg_clientencoding	sethit
pg_client_encoding	setindentation
pg_setclientencoding	setleftfill
pg_set_client_encoding	setleftmargin
php_user_filter	setline
pos	
print	
private	
protected	

**R****S**

setlinespacing  
setmargins  
setmatrix  
setname  
setover  
setrate  
setratio  
setrightfill  
setrightmargin  
setspaceing  
setup  
show\_source  
sizeof  
skewx  
skewxto  
skewxto  
skewy  
skewyto  
skewyto  
snmpwalkoid  
snmprealwalk  
static  
strchr  
strstr  
streammp3  
swfaction  
swfaction\_init  
swfbitmap  
swfbitmap\_getHeight  
swfbitmap\_getWidth  
swfbitmap\_init  
swfbutton  
swfbutton\_addAction  
swfbutton\_addShape  
swfbutton\_init  
swfbutton\_setAction  
swfbutton\_setDown  
swfbutton\_setHit  
swfbutton\_setOver  
swfbuttonSetUp  
swfdisplayitem\_addColor  
swfdisplayitem\_move  
swfdisplayitem.moveTo  
swfdisplayitem\_multColor  
swfdisplayitem\_rotate  
swfdisplayitem\_rotateTo  
swfdisplayitem\_scale  
swfdisplayitem\_scaleTo  
swfdisplayitem\_setDepth  
swfdisplayitem\_setMatrix  
swfdisplayitem\_setName  
swfdisplayitem\_setRatio  
swfdisplayitem\_skewX  
swfdisplayitem\_skewXTo  
swfdisplayitem\_skewY  
swfdisplayitem\_skewYTo  
swffill  
swffill\_init  
swffill\_moveTo  
swffill\_rotateTo  
swffill\_scaleTo  
swffill\_skewXTo  
swffill\_skewYTo  
swffont  
swffont\_getAscent  
swffont\_getDescent  
swffont\_getLeading  
swffont\_getWidth  
swffont\_init  
swfgradient  
swfgradient\_init  
swfgradient\_addEntry  
swfmorph  
swfmorph\_getShape1  
swfmorph\_getShape2  
swfmorph\_init  
swfmovie  
swfmovie\_add  
swfmovie\_init  
swfmovie\_labelFrame  
swfmovie\_nextFrame  
swfmovie\_output  
swfmovie\_remove  
swfmovie\_save  
swfmovie\_saveToFile  
swfmovie\_setBackground  
swfmovie\_setDimension  
swfmovie\_setFrames  
swfmovie\_setRate  
swfmovie\_streamMp3  
swfshape  
swfshape\_addfill  
swfshape\_drawarc  
swfshape\_drawcircle  
swfshape\_drawcubic  
swfshape\_drawcubicto

swfshape_drawcurve	sybase_fetch_field
swfshape_drawcurveto	sybase_fetch_object
swfshape_drawglyph	sybase_fetch_row
swfshape_drawline	sybase_field_seek
swfshape_drawlineto	sybase_free_result
swfshape_init	sybase_get_last_message
swfshape_movepen	sybase_min_client_severity
swfshape_movenpto	sybase_min_error_severity
swfshape_setleftfill	sybase_min_message_severity
swfshape_setline	sybase_min_server_severity
swfshape_setrightfill	sybase_num_fields
swfsprite	sybase_num_rows
swfsprite_add	sybase_pconnect
swfsprite_init	sybase_query
swfsprite_labelFrame	sybase_result
swfsprite_nextFrame	sybase_select_db
swfsprite_remove	
swfsprite_setFrames	
swftext	
swftext_addString	throw
swftext_getLeading	try
swftext_getWidth	
swftext_init	
swftext_moveTo	
swftextSetColor	
swftextSetFont	
swftext_setHeight	
swftext_setSpacing	
swftextfield	unlink
swftextfield_addString	unset
swftextfield_align	use
swftextfield_init	
swftextfield_setBounds	
swftextfieldSetColor	
swftextfieldSetFont	
swftextfield_setHeight	
swftextfield_setIndentation	var
swftextfield_setLeftMargin	
swftextfield_setLineSpacing	
swftextfield_setMargins	
swftextfield_setName	
swftextfield_setRightMargin	
switch	while
sybase_affected_rows	
sybase_close	
sybase_connect	
sybase_data_seek	
sybase_fetch_array	

**T****U****V****W**

**X****xor****xpath\_new\_context**  
**xptr\_new\_context**

## APÉNDICE C

### LAS VARIABLES DEL INTÉRPRETE

En este apéndice se incluye una relación de las variables predefinidas, propias de PHP, que podemos usar en nuestros scripts. Dentro de este tipo de variables encontramos dos categorías: aquéllas que se refieren al servidor y la conexión y aquéllas que se refieren al propio intérprete, a datos que éste maneja y al entorno de trabajo. En este apéndice aparecen recogidas ambas, en las tablas que aparecen a continuación.

VARIABLES RELATIVAS AL SERVIDOR	
VARIABLE	USO Y COMENTARIOS
HTTP_ACCEPT	Tipos de ficheros cuya transmisión se permite del servidor al cliente.
HTTP_ACCEPT_LANGUAGE	Idioma por defecto del navegador.
HTTP_ACCEPT_ENCODING	Tipo de codificación para datos comprimidos.
HTTP_USER_AGENT	Navegador y versión del mismo.
HTTP_HOST	Nombre del servidor.
HTTP_CONNECTION	Estado de la conexión.
PATH	Rutas por defecto en el sistema.
SERVER_SIGNATURE	Versión del servidor y el intérprete y puerto por el que escucha el servidor.
SERVER_SOFTWARE	Versión del servidor y el intérprete.

<b>VARIABLES RELATIVAS AL SERVIDOR (Cont.)</b>	
<b>VARIABLE</b>	<b>USO Y COMENTARIOS</b>
<b>SERVER_NAME</b>	Nombre del servidor.
<b>SERVER_ADDR</b>	Dirección IP del servidor.
<b>SERVER_PORT</b>	Puerto por el que escucha el servidor.
<b>REMOTE_ADDR</b>	Dirección IP del cliente. Si la conexión es a través de un proxy no transparente, esta variable podría contener la IP del mismo.
<b>REMOTE_PORT</b>	Puerto por el que ha establecido la conexión el cliente.
<b>DOCUMENT_ROOT</b>	Ruta absoluta que se usa como carpeta cero del servidor.
<b>SERVER_PROTOCOL</b>	Protocolo de comunicación con el servidor.
<b>REQUEST_METHOD</b>	Método de solicitud de la conexión.
<b>QUERY_STRING</b>	Cadena de consulta (si existe).
<b>REQUEST_URI</b>	Dirección del script.
<b>SCRIPT_NAME</b>	Nombre del script.

<b>VARIABLES RELATIVAS A PHP</b>	
<b>VARIABLE</b>	<b>USO Y COMENTARIOS</b>
<b>\$_SERVER["HTTP_ACCEPT"]</b>	Variable de servidor (*)
<b>\$_SERVER["HTTP_ACCEPT_LANGUAGE"]</b>	Variable de servidor (*)
<b>\$_SERVER["HTTP_ACCEPT_ENCODING"]</b>	Variable de servidor (*)
<b>\$_SERVER["HTTP_USER_AGENT"]</b>	Variable de servidor (*)
<b>\$_SERVER["HTTP_HOST"]</b>	Variable de servidor (*)
<b>\$_SERVER["HTTP_CONNECTION"]</b>	Variable de servidor (*)
<b>\$_SERVER["PATH"]</b>	Variable de servidor (*)
<b>\$_SERVER["SERVER_NAME"]</b>	Variable de servidor (*)
<b>\$_SERVER["SERVER_ADDR"]</b>	Variable de servidor (*)
<b>\$_SERVER["SERVER_PORT"]</b>	Variable de servidor (*)

<b>VARIABLES RELATIVAS A PHP (Cont.)</b>	
<b>VARIABLE</b>	<b>USO Y COMENTARIOS</b>
<code>_SERVER["DOCUMENT_ROOT"]</code>	Variable de servidor (*).
<code>_SERVER["SERVER_PROTOCOL"]</code>	Variable de servidor (*).
<code>_SERVER["REQUEST_METHOD"]</code>	Variable de servidor (*).
<code>_SERVER["QUERY_STRING"]</code>	Variable de servidor (*).
<code>_SERVER["REQUEST_URI"]</code>	Variable de servidor (*).
<code>_SERVER["SCRIPT_NAME"]</code>	Variable de servidor (*).
<code>_SERVER["PHP_SELF"]</code>	Variable de servidor (*).
<code>_ENV["ALLUSERSPROFILE"]</code>	Perfil general de usuarios.
<code>_ENV["CommonProgramFiles"]</code>	Ruta de aplicaciones.
<code>_ENV["COMPUTERNAME"]</code>	Nombre del equipo de trabajo.
<code>_ENV["NUMBER_OF_PROCESSORS"]</code>	Número de procesadores.
<code>_ENV["OS"]</code>	Sistema operativo.
<code>_ENV["PROCESSOR_ARCHITECTURE"]</code>	Tipo de procesador.
<code>_ENV["PROCESSOR_IDENTIFIER"]</code>	Identificador del procesador.
<code>_ENV["PROCESSOR_LEVEL"]</code>	Nivel (familia) del procesador.
<code>_ENV["PROCESSOR_REVISION"]</code>	Versión del procesador.
<code>_ENV["ProgramFiles"]</code>	Ruta de las aplicaciones.
<code>_ENV["SystemDrive"]</code>	Dispositivo de almacenamiento.
<code>_ENV["SystemRoot"]</code>	Ruta del sistema.
<code>_ENV["TEMP"]</code>	Almacenamiento de archivos temporales.
<code>_ENV["TMP"]</code>	Almacenamiento de archivos temporales.
<code>_ENV["USERPROFILE"]</code>	Ruta local del usuario.
<code>_ENV["windir"]</code>	Almacenamiento de Windows (en plataformas con este sistema). En Linux esta variable no procede.

VARIABLES RELATIVAS A PHP (Cont.)	
VARIABLE	USO Y COMENTARIOS
<code>_POST["variable"]</code>	Es una variable enviada desde un formulario mediante POST.
<code>_GET["variable"]</code>	Es una variable enviada desde un formulario mediante GET.
<code>_FILES["fichero"]</code>	Es un fichero enviado desde un formulario (exclusivamente por método POST).
<code>_SESSION["variable"]</code>	Variable de sesión.
<code>_COOKIE["variable"]</code>	Variable de cookie.
<code>PHP_SELF</code>	El nombre del script en ejecución.
<code>_SERVER["REMOTE_ADDR"]</code>	La dirección IP del cliente.
<code>_SERVER["REMOTE_PORT"]</code>	El puerto por el que establece la conexión el cliente.
<code>SSERVER["HTTP_CLIENT_IP"]</code>	La dirección IP del cliente (cuando no funciona REMOTE_ADDR).
<code>SSERVER["HTTP_X_FORWARDED_FOR"]</code>	Cuando la conexión se realiza a través de uno o más proxys no anónimos, esta variable contiene una cadena con la ruta de proxys.

(\*) Las variables de PHP (segunda tabla), cuyo nombre empieza por `_SERVER` contienen los datos del servidor que aparecen en la primera tabla.

Estas tablas constituyen un cuadro resumen. No aparecen todas las variables del servidor, pero muchas de ellas no las usarás nunca. He querido reflejar aquí las que son más significativas. Si tiene curiosidad por ver las demás, ejecute el script `pruebaPHP.php`, en la carpeta raíz de su servidor local, o cualquier otro script que contenga la función `phpinfo()`. A pesar de que le pueda parecer mucha información, normalmente usará las diez últimas variables, que aparecen con un sombreado más claro.

## APÉNDICE D

### DIRECCIONES ÚTILES DE INTERNET

---

---

En este apéndice se listan direcciones de Internet que usted siempre debería tener a mano e, incluso, conocer de memoria, ya que le serán de una inestimable ayuda en su trabajo diario.

**<http://www.php.net>.**

Es la página oficial del lenguaje. En ella encontrará siempre las últimas versiones disponibles. Además, contiene gran cantidad de documentación en varios idiomas y, por supuesto, el manual oficial de PHP, que puede consultar en línea o descargarse en formato .chm. También encontrará aquí el código fuente del lenguaje. Si es usted un apasionado de la programación de alto nivel, puede “bucear” en el código e, incluso, modificarlo para tener su propia versión de PHP. Otra posibilidad que le ofrece el sitio es la colaboración para el desarrollo de PHP: usted puede aportar sus ideas y compartirlas con la comunidad de programadores, o ver las ideas de los demás y aprender de ellas.

**<http://www.apache.org>**

Sitio oficial del proyecto Apache (el servidor web). Aquí puede descargarse la última versión de esta estupenda herramienta, así como acceder a gran cantidad de información, sobre programación, configuración para distintas plataformas, etc.

**<http://www.appservnetwork.com>**

Sitio de obligada visita, donde puede obtener, sin cargo alguno, la última versión disponible del paquete AppServ, que utilizamos en el capítulo 2 para su instalación. Además, podrá conseguir algunos añadidos (“add-ons”) que le permitirán obtener más rendimiento de esta aplicación.

<http://www.aclogic.com>

Si desea actualizar su servidor CesarFTP a la última versión, así como obtener documentación e información adicional, no deje de pasar por aquí.

<http://www.argosoft.com>

Página oficial del fabricante del servidor de correo Argo. Como usted habrá podido comprobar, su instalación y configuración es extremadamente simple, pero su funcionamiento es muy eficiente y, además, es una aplicación libre.

<http://www.fpdf.org>

En esta página puede descargarse la última versión disponible de la clase FPDF, así como toda la documentación que hay respecto a la misma. Si recuerda el texto del libro, en el capítulo 14 aprendimos a usar esta clase para generar documentos .pdf a partir de PHP. Todo el material de este sitio es open source.

<http://www.lawebdelprogramador.com>

Estupendo punto de encuentro para cualquier aficionado o profesional de la informática. Tienen los mejores foros de consulta de la Red, tanto para PHP, Apache, MySQL o cualquier otro tema relacionado con la informática. Además, cuentan con una bolsa de trabajo en constante actualización. También puede encontrar artículos, noticias, código para descargar (en varios lenguajes), documentación, etc. No deje de visitarla.

<http://www.hotscripts.com>

Estupendo sitio donde se pueden obtener gran cantidad de scripts (en PHP y otros lenguajes), que podemos usar para nuestro trabajo diario. Los scripts aquí obtenidos pueden usarse directamente o modificarse como deseé para adaptarlos a su proyecto. Son de uso libre y gratuito. Y, si usted desea colaborar con la comunidad de usuarios, puede enviar sus propios scripts.

<http://www.phpbb.com>

El phpbb es una herramienta para el desarrollo y puesta en marcha de foros de Internet. Es completamente configurable, muy potente y extremadamente fácil de usar. Si alguna vez tiene que desarrollar un foro totalmente profesional, use phpbb que, además, es gratis (aunque el foro que usted haga no tiene por qué serlo, desde luego). En la página hay gran cantidad de documentación. Además, se puede bajar el proyecto (totalmente desarrollado en PHP) y empezar a trabajar con él inmediatamente.

**<http://www.jabber.org>**

Sitio oficial del proyecto Jabber, que permite implementar servicios de mensajería instantánea entre usuarios, de forma totalmente gratuita. Se basa en una aplicación de servidor, de arquitectura distribuida, y la correspondiente aplicación de cliente. Ambas son de libre uso.

**<http://sourceforge.net>**

Este sitio contiene una infinidad de proyectos relacionados con PHP, así como servidores “espejo” para descargar aplicaciones de diversos fabricantes. También ofrece grandes cantidades de documentación. La variedad e interés del material que podemos encontrar aquí es tal que resultaría una tarea titánica referenciarla toda aquí. No deje de visitar el sitio.

**<http://www.elhacker.net>**

Sitio dedicado a la seguridad informática. Ofrecen gran cantidad, y calidad, de documentación al respecto. Tradicionalmente era de descargas gratuitas. Ahora, algunas de ellas implican un coste. Sin embargo, todavía se puede encontrar material libre muy interesante.

**<http://www.adobe.com>**

Página principal de la firma de software Adobe. Desde allí se puede descargar, de forma gratuita, la aplicación Adobe Acrobat Reader, que permite leer documentos en formato .pdf.

**<http://ip-to-country.directi.com/>**

Página desde la que se puede descargar la última versión actualizada de la relación IP to country (ver capítulo 18).

**<http://www.amen.es>**

La empresa Amen proporciona una excelente relación calidad-precio en servicios de Internet (registro de dominios, alojamientos Windows y Linux, etc). Por sus prestaciones, ofertas y servicio post-venta es uno de los mejores sitios de esta naturaleza que se pueden encontrar, actualmente, en el mercado.

<http://www.rfc-es.org/>

Esta página contiene documentos RFC traducidos al español. Los RFC (*Request For Comments*) son las especificaciones acerca del funcionamiento de los distintos protocolos que operan en Internet. Desde los populares HTTP (navegación web), SMTP y POP3 (correo electrónico) o FTP, hasta los más específicos. El problema es que la mayoría de los documentos RFC sólo se encuentran en inglés. Si usted está familiarizado con este idioma puede, incluso, colaborar con esta página traduciendo nuevos textos al español.

<http://www.devshe.com/>

Sitio con gran cantidad de artículos y scripts de PHP. También se encuentra material útil sobre SQL y MySQL, así como sobre otras tecnologías. El único handicap es que está en inglés, pero dada la variedad y calidad del material que ofrecen, de modo gratuito, merece la pena visitarlo. Y algunos de sus artículos están traducidos al español.

<http://www16.brinkster.com/gazb/ming/index.html>

En esta página encontraremos gran cantidad de ejemplos de películas Flash creadas con la librería Ming de PHP. De cada una de ellas podemos obtener, de forma libre y gratuita, el código, que podemos usar para aprender de él o para incluirlo, total o parcialmente, en nuestros propios trabajos.

En este apéndice le he ofrecido una lista de sitios breve, pero sumamente seleccionada. El material de trabajo que encontrará aquí le facilitará sus desarrollos para Internet de una forma que usted no puede suponer aún. No obstante, en la Red aparecen con gran frecuencia nuevos sitios que merece la pena conocer. Si usted descubre alguno que no figure en esta lista, el autor le quedará muy agradecido si le informa de ello.

## APÉNDICE E

### EL CÓDIGO ASCII

Este apéndice muestra la tabla completa del código ASCII, a fin de que usted la tenga como referencia. Como sabe, este código es una relación de 256 caracteres (del 0 al 255) referenciados mediante un código numérico, ya que, en última instancia, el ordenador sólo reconoce números. Cada letra, signo de puntuación, etc., está representado mediante uno de los códigos numéricos de esta lista, reconocida universalmente. En la actualidad, el código ASCII forma parte de una lista mucho más amplia, llamada **Unicode**. Este código tiene 65.536 caracteres y ha sido diseñado para incluir también los signos de escritura de otros idiomas como el árabe, el chino, el ruso, el griego, etc., aunque es poco factible que usted llegue a necesitar estos caracteres. No obstante, si está interesado, puede visitar <http://www.unicode.org>, donde encontrará más información. Dado que ésta es una obra eminentemente práctica, aquí sólo voy a reproducir el código ASCII, con el que usted tendrá todo lo que vaya a necesitar de momento.

CÓDIGO ASCII	CARÁCTER	CÓDIGO ASCII	CARÁCTER
0	Nulo	128	ç
1	SOH	129	ü
2	STX	130	é
3	ETX	131	à
4	EOT	132	ä
5	ENQ	133	à
6	ACK	134	à
7	Beep	135	ç
8	Retroceso	136	ê
9	Tabulación	137	ë

CÓDIGO ASCII	CARÁCTER	CÓDIGO ASCII	CARÁCTER
10	Salto de línea	138	è
11	VT	139	ï
12	Salto de página	140	î
13	<ENTER>	141	ì
14	SO	142	Ã
15	SI	143	À
16	DLE	144	É
17	DC1	145	æ
18	DC2	146	È
19	DC3	147	ô
20	DC4	148	ö
21	NAK	149	ð
22	SYN	150	û
23	ETB	151	ù
24	CAN	152	ÿ
25	EN	153	ò
26	SUB	154	ü
27	<ESC>	155	ø
28	FS	156	£
29	GS	157	Ø
30	RS	158	×
31	US	159	f
32	Espacio en blanco	160	â
33	!	161	í
34	"	162	ó
35	#	163	ú
36	\$	164	ñ
37	%	165	Ñ
38	&	166	¤
39	,	167	º
40	(	168	¿
41	)	169	®
42	*	170	¬
43	+	171	½
44	,	172	¼
45	-	173	í
46	.	174	»
47	/	175	»
48	0	176	—
49	1	177	—
50	2	178	—
51	3	179	—

CÓDIGO ASCII	CARÁCTER	CÓDIGO ASCII	CARÁCTER
52	4	180	¡
53	5	181	À
54	6	182	Ã
55	7	183	Ã
56	8	184	®
57	9	185	¡
58	:	186	¡
59	;	187	+
60	<	188	+
61	=	189	¢
62	>	190	¥
63	?	191	+
64	@	192	+
65	À	193	-
66	À	194	-
67	Ç	195	+
68	Ð	196	-
69	È	197	+
70	È	198	ã
71	Ã	199	Ã
72	Ñ	200	+
73	Ñ	201	+
74	Ñ	202	-
75	Ñ	203	-
76	Ñ	204	¡
77	Ñ	205	-
78	Ñ	206	+
79	Ñ	207	□
80	Ñ	208	Ñ
81	Ñ	209	Ð
82	Ñ	210	È
83	Ñ	211	È
84	Ñ	212	È
85	Ñ	213	í
86	Ñ	214	í
87	Ñ	215	í
88	Ñ	216	í
89	Ñ	217	+
90	Ñ	218	+
91	[	219	-
92	\	220	-
93	]	221	¡
94	^	222	í

CÓDIGO ASCII	CARÁCTER	CÓDIGO ASCII	CARÁCTER
95	~	223	ó
96	á	224	ó
97	é	225	ß
98	í	226	ô
99	ç	227	õ
100	đ	228	õ
101	è	229	ñ
102	é	230	µ
103	é	231	þ
104	é	232	þ
105	í	233	ú
106	í	234	ú
107	í	235	ú
108	í	236	ý
109	í	237	ý
110	ñ	238	-
111	ñ	239	-
112	ñ	240	-
113	ñ	241	±
114	ñ	242	-
115	ñ	243	¾
116	ñ	244	¶
117	ñ	245	§
118	ñ	246	÷
119	ñ	247	,
120	ñ	248	º
121	ñ	249	"
122	ñ	250	*
123	{	251	¹
124		252	³
125	}	253	²
126	~	254	—
127		255	

Los códigos del 0 al 31 son lo que se conoce como códigos de control. La mayor parte de éstos se emplean en programación de aplicaciones con lenguajes como Visual Basic, Java, C++, etc., por lo que el estudio de su significado queda fuera del propósito de este libro. Sin embargo, hay algunos, dentro de este grupo, que sí es interesante conocer.

- El código 7 representa un pitido del altavoz del ordenador.
- El código 8 representa la tecla de retroceso situada, normalmente, en la parte superior derecha del bloque principal del teclado.

- El código 9 representa la tecla de tabulación.
- El código 10 representa un salto de línea en los listados impresos.
- El código 12 también se usa para la impresora y representa un salto de página.
- El código 13 representa un retorno de carro (tecla <ENTER>).
- El código 27 representa la tecla <ESC> (escape).

Los demás códigos de este primer bloque (del 0 al 31) se suelen usar en programación de redes locales y protocolos de comunicación. Como he dicho, a nosotros no nos afectan para nada a efectos de JavaScript. No obstante, a modo de referencia-curiosidad, los resumo a continuación brevemente.

- NUL – Nulo
- SOH – Start of heading (Inicio de cabecera)
- STX – Start of text (Inicio de texto)
- ETX – End of text (Final de texto)
- EOT – End of transmission (Final de transmisión)
- ENQ – Enquiry (Requerimiento)
- ACK – Acknowledge (Reconocimiento)
- VT – Vertical tabulation (Tabulado vertical)
- SO – Shift out (Desactivación del modo de mayúsculas)
- SI – Shift in (Activación del modo de mayúsculas)
- DLE – Data link escape (Escape del enlace de datos)
- DC1 – Device control 1 (Control de dispositivo 1)
- DC2 – Device control 2 (Control de dispositivo 2)
- DC3 – Device control 3 (Control de dispositivo 3)
- DC4 – Device control 4 (Control de dispositivo 4)
- NAK – Negative Acknowledge (Reconocimiento negativo)
- SYN – Synchronous Idle (Sincronización)
- ETB – End of transmission block (Final de bloque de transmisión)
- CAN – Cancel (Cancelar)
- EM – End of medium (Final de soporte)
- SUB – Substitute (Reemplazar)
- FS – File separator (Separador de archivos)
- GS – Group separator (Separador de grupos)
- RS – Record separator (Separador de registros)
- US – Unit separator (Separador de unidades)

## APÉNDICE F

### EL CONTENIDO DEL CD

---

---

En este apéndice se incluye información que debe conocer acerca del material adicional asociado a este libro, antes de querer instalarlo en su ordenador. Dicho material adicional contiene dos carpetas principales: **ejemplos** y **software**. En la primera encontrará todos los scripts que se han creado con fines didácticos, organizados en distintas carpetas, una para cada capítulo (salvo los capítulos 1, 2 y 15, que no necesitan ningún script). Además, hay un par de ficheros que no se han incluido en la carpeta específica de ninguno de los capítulos, llamados **pruebaPHP.php** y **pruebaApache.htm**. Copie el contenido de la carpeta **ejemplosDelLibro** en aquella carpeta de su disco duro que vaya a usar como servidor local. Consulte el capítulo 2 para saber lo necesario sobre este punto. Recuerde: no copie la carpeta **ejemplos** completa, sino sólo su contenido.

En la carpeta **software** encontrará algunos programas, códigos y documentos adicionales, de libre distribución, necesarios para trabajar con el material didáctico incluido en el libro. El único programa que echará en falta es el AppServ (capítulo 2). Esto se debe a que éste no es de libre distribución, por lo que deberá obtenerlo desde el sitio web del fabricante (vea el Apéndice D). No obstante, la descarga y uso del programa son gratuitos.

El programa **WampServer** se emplea en el capítulo 21 del libro, como alternativa compacta y de libre distribución al AppServ. A fin de que pruebe otros programas similares de libre uso y distribución, si desea experimentar, también encontrará **XAMPP** en el material adicional.

Así mismo, encontrará un servidor de correo, llamado **agsmail.exe** (Argo) y un servidor de FTP (**CesarFTP.exe**).

En la sub-carpetas **notepadplusplus** encontrará un editor de texto plano, que le vendrá muy bien para crear sus propios scripts, ya que, al contrario que el Bloc de Notas o el VIM, resalta la sintaxis de PHP, permitiendo detectar palabras clave mal escritas. Ejecute el instalador y experimente con el editor.

En la sub-carpetas **clases y paquetes PHP** encontrará códigos de PHP, de los que se habla en el texto del libro, que le resultarán sumamente útiles en su estudio de este lenguaje, así como en su vida profesional. En concreto, encontrará:

- La clase FPDF, de libre uso y distribución, para generar documentos PDF sin ninguna herramienta adicional.
- La clase phpMailer, para enviar documentos por correo electrónico.
- El paquete phpMyAdmin, para facilitar el trabajo con bases de datos MySQL.
- El paquete phpBB3, para la construcción de foros.

Conserve todo este material, que le resultará muy útil.

## APÉNDICE G

### LA VERSIÓN 5.3 DE PHP

---

---

En el capítulo 21 del libro se ha insistido en la instalación del programa WampServer en una versión ligeramente más antigua que la última publicada por el autor ([www.wampserver.com](http://www.wampserver.com)). Esto se debe a que la que le ofrecemos instala la versión 5.2 de PHP, mientras que la actual instala la versión 5.3. Y ¿cuál es la razón para preferir una versión más antigua de PHP? Esta pregunta podría admitir varias respuestas. Las dos más relevantes son las siguientes:

En primer lugar, y en lo que se refiere a las técnicas de depuración descritas en el capítulo 21, la extensión Zend Debugger que estudiamos no existe, en el momento de escribir estas líneas, para PHP 5.3, sino solamente para PHP 5.2. Dada la utilidad de dicha extensión, y hasta que los autores publiquen una variante adecuada, necesitará disponer de PHP 5.2.

Por otra parte, las técnicas de sesiones descritas en el capítulo 9 cambian ligeramente. Por no entrar en detalles, las funciones propias de sesión, excepción hecha de `session_start()`, han sido deprecadas y, al tratar de usarlas, generan un error. Las más importantes son las siguientes:

```
session_register()  
session_unregister()  
session_is_registered()
```

Con el fin de compatibilizar código de PHP para diferentes versiones, le sugiero que preceda cada una de estas funciones con el operador `@` que, como sabe, omite los errores que se producen si no se puede ejecutar una función.



# ÍNDICE ALFABÉTICO

---

---

## \$

\$ COOKIE .....	269, 270, 273, 274
\$ FILES.....	133, 135, 142, 143, 144, 145, 146, 147, 148, 150
\$ SERVER .....	287, 366, 367, 444, 445, 446
\$ SERVER ['REMOTE_ADDR'].....	446
\$HTTP_POST_FILES .....	133
SPHP_AUTH_PW .....	292, 293
SPHP_AUTH_USER .....	292, 293

## A

Abstracción .....	299
Actualización .....	299
AddEntry .....	511
Align .....	520
Alineamiento .....	159
Ámbito de las variables .....	113
Anclaje .....	499
Anidamiento de bucles .....	100
Apache.....	30, 35, 36, 37, 38, 39, 53, 54, 55, 287, 555, 561, 562, 563, 586, 603, 604
Archivos ttf .....	339
ArgoSoft.....	42
Argumento .....	107
Asociativas. <i>Véase</i> matrices	
Atributo action.....	124, 134, 139, 151, 152, 171, 253, 264
Atributo AUTO_INCREMENT .....	383
Atributo enctype.....	135, 247

Tributo method.....	129
Atributo PRIMARY KEY.....	378
AUTENTICACIÓN .....	291

## B

Balanceo de carga .....	424
Barra de direcciones.....	284
Basic realm .....	291, 292, 293, 294
Booleano .....	74
Botón Comenzar instalación .....	468
Bots .....	449
BOTS .....	449
Break.....	104
Bucle local. <i>Véase</i> localhost	

## C

Cabecera.....	24, 284
Cabeceras adicionales .....	342, 343
Cabeceras de la respuesta.....	288
Cache-Control .....	290
Cadena de la Consulta. <i>Véase</i> Query String	
Campo clave .....	370
Campos de tipo ENUM .....	376
Campos de un formulario.....	128, 133, 180
Captcha .....	456
Captura de errores .....	417
Carácter de relleno .....	159
CASCADE.....	379

Checksum.....	25
Chr.....	155
Clase.....	300
Clase SWFAction.....	483, 541, 543, 546, 548
Clase SWFBitmap.....	484
Celase SWFButton.....	545, 548
Clase SWFDisplayItem.....	485
Clase SWFFill.....	485
Clase SWFFont.....	486
Clase SWFGradient.....	486, 511
Clase SWFMorph.....	486, 538, 539
Clase SWFMovie.....	487, 492
Clase SWFShape.....	487
Clase SWFSprite.....	488
Clase SWFText.....	489
Clase SWFTextField.....	489
Clases derivadas.....	307
Cláusula ADD.....	381
Cláusula AVG...AS.....	384
Cláusula CHANGE.....	381
Cláusula COUNT...AS.....	385
Cláusula DROP.....	381
Cláusula GROUP BY.....	384, 385
Cláusula IF NOT EXISTS.....	373, 396
Cláusula ON DELETE.....	379
Cláusula ON UPDATE.....	379
Cláusula ORDER BY.....	384, 429
Cláusula REFERENCES.....	379
Cláusula SUM... AS.....	384
Cláusula WHERE.....	383, 384, 386, 388
Clave bits.....	318
Clave channels.....	318
Clave mime.....	318
Clave primaria.....	371
Claves secundarias.....	371
Clientes.....	19, 21, 29, 45, 52, 291, 292, 293, 294, 369, 424
Closedir.....	263
CMYK.....	319
Codificación URL.....	130, 166, 167
Colocando el texto.....	522
Color verdadero.....	324
Colores RGB.....	319
Colores-luz.....	318
Colores-tinta.....	318
Comma Separated Values.....	440
Compilado con --with-MING.....	482
Condicionales.....	74, 91, 92, 95, 97, 119, 121, 165, 207, 218, 223
Condiciones múltiples.....	93
Conexiones persistentes.....	410, 424
Configurando el foro.....	471
Constantes.....	77, 78, 240, 349, 351, 354, 552, 589
Constructor.....	303
Consulta CREATE INDEX.....	380
Consulta CREATE TABLE.....	374
Consulta DROP DATABASE.....	374
Consulta DROP INDEX.....	380
Consulta INSERT INTO.....	381
Consulta SELECT.....	383
Consultas de datos.....	373
Consultas de definición de datos. <i>Véase</i> consultas estructurales	
Consultas de manipulación de datos. <i>Véase</i> consultas de datos	
Consultas estructurales.....	373
Contenido.....	284
Content-type.....	311
Continue.....	104
Contrashash. <i>Véase</i> escapar un carácter	
Control.....	499
Cookie.....	268, 269, 270, 271, 272, 273, 274, 275, 281, 602
Cookies.....	267
Coordenadas absolutas.....	496
Coordenadas relativas.....	496
Copias de seguridad.....	478
Cortafuegos. <i>Véase</i> firewall	
Creando curvas.....	499
CREATE DATABASE.....	373
Cresta de la curva.....	500
CSV. <i>Véase</i> Comma Separated Values	
Cuerpo.....	24, 284
Cuerpo del bucle.....	97, 98, 99, 100, 101, 102, 103
Cursor.....	403, 410

**D**

Datos de tipo FLOAT, DOUBLE y DECIMAL.....	376
De alternativa.....	204, 207, 208
De ejecución.....	249
De lectura.....	249
De secuencia o fijación.....	204
Decremento.....	64
Depurando PHP.....	572
Dirección IP.....	20, 21, 26, 45, 179, 294, 295, 352, 366, 565, 602
Directiva safe_mode.....	491
Directiva session.use-trans-id.....	281

DLL. <i>Véase</i> consultas estructurales	
DML. <i>Véase</i> consultas de datos	
DNS.....	20, 21, 22, 54, 295
DocumentRoot .....	36, 54, 55, 563
DrawCurve .....	500
DrawCurveTo.....	500
DrawLine .....	495
DrawLineTo .....	495

**E**

Eclipse + PDT .....	566
El apartado Estructura.....	436
El lenguaje SQL .....	372
El origen de una visita.....	439
El servidor Wamp .....	555
Enable-trans-sid.....	281
Envoltura URL ftp:// .....	266
EOF .....	232
Ereg_replace.....	213
Eregi_replace .....	214
Escalados.....	534
Escapado de metacaracteres .....	207
Escapar un carácter .....	72
Estándar ODBC.....	391
Estándar PERL .....	215
Estándar Posix.....	204, 210, 211, 212, 216
Estilo subsilver 2 .....	473
Estructura while .....	100, 101
Eval .....	351
Eventos.....	300
Expansión de variables.....	72
Expires .....	290
Expresiones regulares.....	203, 204, 205, 206, 208, 209, 210, 211, 212, 214, 217, 218, 220, 446
Extensiones .....	581

**F**

Facilitar la legibilidad .....	189, 387, 458
Factor alfa .....	320, 495
Fatal error.....	224, 225, 349
Feof .....	236
Fgete.....	234
Fgetesv .....	443
Fgets.....	236
Fgetss .....	236
Figuras cerradas .....	502
Filas.....	370
File .....	237

File_get_contents .....	541
Filtro .....	334
Firewalls.....	25
Flash.....	481
Foreach.....	102
Formas .....	494
Formato de texto enriquecido .....	457
FPDF .....	356
Fputs .....	243
Frames Per Second.....	493
Ftell .....	238
FTP.....	26, 30, 31, 45, 50, 249, 283, 352, 353, 354, 355, 466, 561, 606, 613
Ftp_pasv .....	355
Función array_unshift .....	446
Función asin .....	185
Función asort .....	87
Función atan .....	185
Función base_convert .....	190
Función base64_decode .....	183
Función base64_encode .....	183
Función bindec .....	188
Función checkdate .....	201
Función chmod .....	250, 259, 260
Función chop .....	154
Función chunk_split .....	180, 182
Función copy .....	244
Función count .....	85, 86
Función date .....	197, 198, 200
Función debin .....	189
Función dechex .....	189
Función decoct .....	189
Función define .....	77
Función defined .....	78
Función deg2rad .....	185
Función dir .....	143
Función empty .....	75
Función error_reporting .....	349
Función exp .....	188
Función explode .....	180, 181, 182, 445, 446
Función extension_loaded .....	491
Función fclose .....	229
Función file_exists .....	246
Función fileperms .....	250, 258
Función filesize .....	233
Función fopen .....	227
Función fpassthru .....	230
Función fseek .....	239
Función fsockopen .....	294, 296
Función ftp_chdir .....	355
Función ftp_close .....	353
Función ftp_connect .....	352

Función ftp_exec.....	355	Función mysql_fetch_array.....	422, 430
Función ftp_get.....	354	Función mysql_free_result.....	425
Función ftp_login.....	352, 354	Función mysql_num_rows.....	419
Función ftp_put.....	355	Función mysql_pconnect.....	424
Función ftp_pwd.....	355	Función mysql_query.....	410, 416
Función function_exists .....	310	Función mysql_select_db.....	413
Función fwrite .....	242	Función number_format.....	193
Función get_html_translation_table .....	177, 178	Función octdec .....	189
Función getallheaders.....	284, 286, 295	Función odbc_close .....	394
Función getdate.....	195	Función odbc_connect .....	394
Función getimagesize.....	316	Función odbc_do.....	395, 396
Función filetype.....	67, 68, 76	Función odbc_fetch_row.....	400, 401
Función gmdate.....	198	Función odbc_num_fields.....	402
Función header .....	288, 289, 290, 311	Función odbc_num_rows.....	403
Función hexdec .....	188	Función odbc_result.....	401, 403
Función htmlentities.....	176	Función odbc_result_all .....	401
Función imagearc .....	330	Función opendir .....	262
Función imagecolorallocate.....	323, 324, 329, 455	Función preg_replace .....	219
Función imagecolorallocatealpha.....	324	Función rand .....	184, 185, 451
Función imagecolorset .....	326	Función rawurlencode .....	167, 172
Función imagecolorset .....	322	Función readdir .....	262
Función imagecopy .....	326	Función rename .....	245
Función imagecreatetruecolor .....	324	Función rewaddir .....	262
Función imagedestroy .....	312	Función rsort .....	87
Función imageellipse .....	331	Función session_destroy .....	282
Función imagefill .....	332	Función session_id .....	276
Función imagefilledrectangle .....	333	Función session_name .....	276
Función imagefilter .....	333	Función session_register .....	276
Función imageline .....	330, 456	Función session_start .....	276, 279
Función imagepolygon .....	331	Función session_unregister .....	282
Función imagerectangle .....	330	Función session_unset .....	282
Función imagesetpixel .....	324	Función setcookie .....	270, 274
Función imagestring .....	336, 455	Función settype .....	68, 69
Función imagesx .....	315	Función sin .....	185
Función imagesy .....	315	Función sort .....	86
Función imagettfttext .....	338	Función sprintf .....	157
Función ip2long .....	447	Función str_pad .....	159
Función is_dir .....	246	Función strlen .....	156
Función is_executable .....	246	Función strstr .....	164
Función is_file .....	246	Función substr .....	153, 250
Función is_link .....	246	Función time .....	195
Función is_readable .....	246	Función trim .....	155
Función isset .....	74	Función unlink .....	243
Función join .....	182	Función var_dump .....	89
Función log .....	188		
Función long2ip .....	447		
Función ltrim .....	154		
Función mail .....	341, 342, 343, 560		
Función move_uploaded_file.....	141, 144, 148, 149, 262		
Función mysql_close.....	425		
		<b>G</b>	
		GD.....	309
		GD2.....	309
		Gestión de ficheros .....	221, 227
		Get, en lugar de post .....	131
		Getcwd.....	262

GetShape1 .....	538
GetShape2 .....	538
GetWidth .....	517
Grabar la película .....	537

**H**

Headers Information.....	286
Herencia .....	300, 301, 306
Homogeneidad estructural .....	370
Htmlspecialchars .....	177
HTTP.....	26, 32, 33, 130, 131, 132, 133, 139, 140, 141, 151, 214, 215, 219, 220, 223, 247, 248, 257, 259, 264, 265, 275, 276, 283, 286, 287, 288, 289, 290, 291, 292, 293, 294, 295, 296, 366, 367, 444, 445, 446, 599, 600, 602, 606
HTTPS .....	26
Hyper Text Transfer Protocol .....	26, 54

**I**

Identificador. Véase manejador .....	
Identificador Universal de Recursos. Véase URI .....	
Image .....	365
Imagecolorallocate .....	323
Imagecolortotal .....	319
Imagecopyresized .....	328
Imagecreat .....	324
Imagecreatefromgif .....	312
Imagecreatefromjpeg .....	312
Imagecreatefrompng .....	312
Imagecreatefromwbmp .....	312
Imagedashedline .....	330
Imagefilledellipse .....	332
Imagefilledpolygon .....	332
Imagefilledrectangle .....	332
Imagegif .....	310
Imagejpeg .....	310
Imágenes .....	514
Imagepng .....	310
Imagestringup .....	336
Imagewbmp .....	310
Include .....	221, 223
Include_once .....	221
Inconsistencia de la información .....	372
Incremento .....	64
Indexadas. Véase matrices .....	
Instrucción mysql_connect .....	408
Interpolación de forma .....	524

Interpolación de movimiento .....	524
Interpolaciones de forma .....	537
Intérprete de PHP .....	30
IP to Country .....	440
Is_double .....	76
Is_integer .....	76
Is_string .....	76
Is_writeable .....	246

**K**

Ksort .....	87
-------------	----

**L**

La clase SWFButton .....	484
La conexión está cerrada .....	297
La función cos .....	185
La función echo() .....	56
La pluma .....	496
Lado del cliente .....	15, 17, 27, 31, 56, 91, 243, 451, 458
Lado del servidor .....	15, 16, 19, 27, 31, 53, 54, 91, 138, 172, 175, 194, 221, 227, 275, 341
Las directivas .....	583
Las extensiones .....	581
La web del programador .....	463
LC_ALL .....	199
LC_CTYPE .....	199
LC_NUMERIC .....	199
LC_TIME .....	199
Lenguaje Estructurado de Consultas. Véase SQL .....	
Librería l .....	84, 309, 310, 324, 329, 334, 356, 482, 490, 491, 494, 505, 511, 514, 515, 518, 528, 537, 541, 553, 581, 583, 584, 606
Librería MING .....	482
LIMIT .....	386
Lineas rectas .....	494
Localhost .....	20, 33, 37, 38, 41, 44, 45, 50, 54, 55, 124, 130, 286, 294, 342, 346, 352, 353, 355, 409, 410, 412, 416, 425, 441, 446, 460, 466, 560, 562, 563, 564, 567, 569
Location .....	290
Logotipo .....	344
Los tipos CHAR y VARCHAR .....	376

**M**

Manejador.....	129, 229, 230, 231, 232, 233, 234, 236, 238, 239, 242, 243, 262, 263, 264, 265, 294, 295, 296, 311, 312, 313, 315, 316, 319, 320, 322, 323, 324, 325, 326, 327, 328, 329, 330, 331, 332, 333, 334, 336, 338, 352, 353, 354, 355, 394, 395, 398, 409, 410, 413, 414, 425, 443, 453, 454, 459, 487, 515, 516, 522, 523, 525, 526, 527, 529, 530, 531, 533, 534, 535, 539, 548
Marca de tiempo.....	194, 196, 198, 200, 201
Matrices .....	15, 53, 63, 66, 78, 79, 80, 81, 82, 83, 85, 86, 87, 89, 102, 103, 130, 133, 267, 283, 306, 320, 444, 454, 569, 576, 578, 589
Matriz \$_POST .....	460
Matriz \$_SESSION .....	282
Max .....	192
MAX_FILE_SIZE .....	137
MAX_RAND .....	185
Método add .....	496, 499, 511, 519, 523, 526
Método addAction .....	546, 548
Método addFill .....	505
Método AddPage .....	357
Método addShape .....	545
Método Cell .....	357, 360, 361
Método drawCircle .....	545
Método Ln .....	361
Método move .....	523
Método movePenTo .....	495, 503, 528
Método moveTo .....	522, 523, 526
Método nextFrame .....	524, 526, 533
Método output .....	493, 496
Método Output .....	357, 365
Método remove .....	535, 537
Método rotate .....	528
Método save .....	537
Método setBackground .....	493
Método setDimension .....	492, 493
Método SetFont .....	357
Método setHeight .....	520
Método setLine .....	495, 498, 500
Método setRate .....	492, 524
Método Text .....	362, 363
Métodos.....	301
Métodos.....	300
Min .....	192
Mkdir .....	261
Modelo OSI de la ISO .....	23
Modificador ASC .....	385

Modificador DESC .....	385
Mover un archivo .....	245
Multicelda .....	361
MultiCell .....	361
Multiplicadores .....	204, 205, 206, 215
MyISAM .....	379
MySQL.....	16, 31, 32, 33, 34, 35, 377, 378, 379, 390, 405, 407, 408, 409, 410, 412, 413, 415, 417, 418, 419, 420, 423, 424, 425, 431, 432, 433, 441, 442, 443, 444, 446, 447, 465, 468, 555, 561, 563, 581, 604, 606, 614
MySQL_ASSOC .....	422
MySQL_BOTH .....	422
Mysql_create_db .....	424
Mysql_drop_db .....	424
Mysql_errno .....	417
Mysql_error .....	417
MYSQL_NUM .....	422
Mysql_result .....	420

**N**

Niveles de permisos .....	249, 257, 258, 259
Niveles de usuarios.....	248, 249, 250, 251, 257, 260, 261
Nombre de variable .....	62
Notación simplificada .....	65
Número de identificación de la sesión .....	275
Número de sesión .....	275

**O**

Objetos .....	300
Odbc_error .....	398
Odbc_errormsg .....	398
Odbc_fetch_row () para recorrer el cursor .....	404
Opción Run configurations .....	573
Open Data Base Connectivity .....	391
Open source .....	16, 30, 106, 604
Operador & .....	119
Operador ^ .....	119
Operador   .....	119
Operador de asignación .....	58, 63, 92
Operador de concatenación .....	60, 61
Operador de control de errores .....	149, 409
Operador de errores .....	312
Operador or exclusivo .....	95
Operador ternario .....	97
Operadores a nivel de bit .....	119, 121
Operadores de comparación .....	92

Operadores lógicos.....	93, 94, 119
Operadores para encadenar varias condiciones.....	386
Optimizar el código.....	60, 338
Ord .....	155
Origen .....	499
Overview .....	568

**P**

Páginas auto procesadas .....	151, 152
Palabra reservada elseif.....	95
Palabra reservada if.....	91
Paquetes .....	24
Parámetro .....	107
Paréntesis.....	56, 67, 71, 91, 105, 107, 204, 207, 208, 213, 214, 217, 219, 350, 375, 387
Pares nombre-valor.....	53, 58, 77, 78, 130, 267, 275, 284
Parse_url.....	178
PDF .....	355
Pestaña CONVERTIR.....	467
Php.ini .....	583
Php_gd2.dll .....	309
Php_ming.dll .....	482
Php_ming.so .....	482
PHP_OS .....	491
Phppbb3 .....	464
Phpinfo.....	38, 286, 563, 602
PhpMailer .....	343
PhpMyAdmin.....	33, 432, 433, 434, 437, 614
PHPSESSID .....	276
Pila de protocolos. <i>Véase</i> protocolos	
Pixelles .....	490
POO.....	299, 300, 304, 306, 307, 343
POP .....	26
Post-decremento.....	64, 65
Post-incremento.....	64
Precedencia de operadores .....	67
Precedencia de patrones .....	208
Pre-decremento .....	64
Preg_match.....	218
Pre-incremento .....	64, 65
Print .....	156
Printf .....	156
Probando WampServer .....	563
Procesar ficheros .....	133
Programación Orientada a Objetos.....	91, 299
Propiedad error.....	136, 138
Propiedad name.....	135, 140, 141
Propiedad size .....	136, 138, 143

Propiedad tmp-name .....	136, 141
Propiedad type .....	136, 143
Propiedades .....	301
Propiedades .....	300
Protocolo sin estados.....	275, 283
Protocolos.....	23, 24, 54, 166, 267, 294, 606, 611
Proxys .....	22, 25, 445, 602
Puertos .....	25
Puertos bien conocidos .....	25, 26
Puntero .....	231
Punto y coma ..	56, 58, 98, 350, 373, 583, 585

**Q**

Query String.....	284
-------------------	-----

**R**

Rad2deg .....	186
RDBMS. <i>Véase</i> Sistema de Gestión de Bases de Datos Relacionales	
Recursividad .....	118
Red local .....	20, 29, 194, 366, 445, 446
Redondeo .....	191, 193, 194
Referencia \$this .....	304, 306
Referirse a un rango .....	205
Registros .....	370
Relation Data Base Management System. <i>Véase</i> Sistema de Gestión de Bases de Datos Relacionales	
Relleno .....	505
Require .....	221
Require_once .....	221, 226
Resolver el nombre .....	21, 23, 295
Respuesta .....	283
RESTRICT .....	379
Resuelve .....	175
Reutilización .....	299
Reutilización de código .....	221
Rewind .....	232
Rich Text Format. <i>Véase</i> formato de texto enriquecido	
Root, usuario .....	409
RTF. <i>Véase</i> formato de texto enriquecido	

**S**

Samples .....	568
Scale .....	534
ScaleTo. <i>Véase</i> scale	
SEEK_CUR .....	241

SEEK_SET.....	241
Sentencia break.....	97, 105
Sentencia class.....	301
ServerName.....	37
Servicio del sistema.....	35
Servidor Apache.....	30, 35, 36, 37, 39, 53, 427, 555, 562, 585, 586, 587
Servidor de correo.....	31, 39, 41, 45, 50, 341, 342, 476, 560, 604, 613
Servidor FTP. <i>Véase</i> FTP	
Servidor web.....	21, 22, 31, 32, 35, 465, 603
Servidores.....	19, 21, 23, 29, 30, 39, 167, 208, 228, 229, 260, 408, 474, 483, 489, 518, 552, 561, 605
Sesiones.....	267, 275
Sess.....	275
Session id.....	275
Session_is_registered.....	281
SET NULL.....	379
SetBounds.....	520
SetDrawColor.....	360
SetFillColor.....	360
SetLeftFill.....	505
SetLineWidth.....	360
Setlocale.....	199
SetRightFill.....	505
Signo arroba. <i>Véase</i> operador de control de errores	
Sistema de Gestión de Bases de Datos Relacionales.....	369
Slash.....	55, 64, 88, 136, 216, 261
SMTP.....	26, 45, 296, 346, 347, 470, 560, 606
Sockets.....	26
Solicitud.....	283
Source Folder.....	572
Sprites.....	528
SQL.....	369
STR_PAD_BOTH.....	160
STR_PAD_LEFT.....	160
STR_PAD_RIGHT.....	160
Str_repeat.....	160
strftime.....	200
Strrev.....	183
Strtolower.....	161
Strtoupper.....	161
Strtr.....	163
Structured Query Language. <i>Véase</i> SQL	
Subclases. <i>Véase</i> clases derivadas	
Superclase.....	306
SWFBUTTON_DOWN.....	545
SWFBUTTON_DRAGOUT.....	549
SWFBUTTON_DRAGOVER.....	549

SWFBUTTON_HIT.....	545, 548
SWFBUTTON_MOUSEDOWN.....	549
SWFBUTTON_MOUSEOUT.....	549
SWFBUTTON_MOUSEOVER.....	548
SWFBUTTON_MOUSEUP.....	549
SWFBUTTON_MOUSEUPOUTSIDE.....	549
SWFBUTTON_OVER.....	545
SWFBUTTON_UP.....	545
SWFFILL_CLIPPED_BITMAP.....	515
SWFFILL_LINEAR_GRADIENT.....	512
SWFFILL_RADIAL_GRADIENT.....	512
SWFFILL_TILED_BITMAP.....	515, 517
SWFTEXTFIELD_ALIGN_CENTER.....	520
SWFTEXTFIELD_ALIGN_JUSTIFY.....	520
SWFTEXTFIELD_ALIGN_LEFT.....	520
SWFTEXTFIELD_ALIGN_RIGHT.....	520
SWFTEXTFIELD_DRAWBOX.....	552
SWFTEXTFIELD_HTML.....	552
SWFTEXTFIELD_MULTILINE.....	552
SWFTEXTFIELD_NOEDIT.....	552
SWFTEXTFIELD_NOSELECT.....	552
SWFTEXTFIELD_PASSWORD.....	552
SWFTEXTFIELD_WORDWRAP.....	552

**T**

TCP/IP.....	23, 24, 54, 294
Texto Dinámico.....	549
Timestamp.....	194, 195
Tipo de error.....	147
Tipos de permisos.....	249
Tuplas.....	370
Tutorials.....	568

**U**

Universal Resource Identifier ..... <i>Véase</i> URI	
UPDATE.....	388
URI.....	284
Urldecode.....	175
Urlencode.....	175
Uso recursivo. <i>Véase</i> recursividad	

**V**

Valor false .....	144, 148, 246, 262, 293, 294, 312, 341, 352, 353, 355, 396
Valor NULL.....	382
Variable de control.....	98, 99, 100, 421, 526, 535, 539
Variables estáticas.....	117

Ver log de administradores ..... 472

Workbench ..... 568

WampServer ..... 556, 613

WWW-Authenticate ..... 291

Warning ..... 109, 149, 224, 228, 349, 396

ZEND DEBUGGER ..... 564

What's new ..... 568

ZendDebugger.dll ..... 565

**W****Z**

# Domine PHP y MySQL

2ª EDICIÓN

Este libro está diseñado y escrito para aquellas personas que, conociendo XHTML y JavaScript, desean dar un salto adelante en la creación de sitios web, con la programación dinámica en el lado del servidor. Atrás quedan los días oscuros en que las páginas de Internet eran meros documentos de texto e imágenes formateados de un modo rígido, y con unos contenidos inamovibles. Hoy los usuarios saben que pueden interactuar con páginas web, obteniendo los resultados que desean... y quieren disponer de esa prerrogativa. El autor ha reflejado aquí unos conocimientos prácticos y actualizados, para que usted pueda crear sitios realmente dinámicos y atractivos. Este libro no es, en modo alguno, una guía exhaustiva de todas las funciones de PHP. El manual oficial del lenguaje ya contiene toda esa información, muy bien clasificada. En lugar de ello, se ha buscado dar un enfoque práctico al aprendizaje de PHP 5 y MySQL, la práctica y la experiencia harán el resto. A través de las páginas de este texto, usted conocerá las técnicas necesarias para desarrollar e implementar sitios web realmente prácticos, útiles y eficientes.

Pero esta obra va más allá. El autor ha conseguido hacerle llegar algunos conocimientos que no todos los programadores de PHP poseen, y que resultan de gran utilidad. Entre otras cosas, usted aprenderá que con PHP puede:

- Generar documentos en formatos RTF y PDF.
- Identificar la IP de los clientes que se conecten a sus páginas.
- Enviar correos electrónicos formateados en HTML (incluso, con JavaScript), en lugar de simple texto plano.
- Montar un foro de Internet en una hora de trabajo (o menos).
- Crear películas de Flash e integrarlas en sus páginas web... SIN NECESIDAD DE FLASH.
- Depurar con eficiencia sus códigos.

Estas y muchas otras prestaciones estarán a su disposición, sin necesidad de invertir ningún dinero en herramientas ni técnicas adicionales, y por supuesto, cuando haya completado la lectura de este libro, tendrá los conocimientos necesarios para afrontar cualquier reto profesional que se le presente.

Deseo que los conocimientos recopilados en este volumen le resulten tan fascinantes y útiles como me han resultado a mí. Si al concluir la lectura se ve usted capaz de llevar a cabo proyectos profesionales, me daré por satisfecho.



9 788499 640082

WWW  
Ra-Ma

Desde [www.ra-ma.es](http://www.ra-ma.es)  
podrá descargarse  
material adicional.

ra-ma.es

 Ra-Ma®