CAPÍTULO 7

PHP Práctico

En los capítulos anteriores hemos repasado los elementos del lenguaje PHP. En este capítulo, teniendo en cuenta tus nuevas habilidades de programación, te enseño cómo realizar algunas de las más habituales pero importantes tareas prácticas. Aprenderás los mejores procedimientos para tratar con cadenas y lograr un código claro y conciso que se muestre en los navegadores web exactamente como deseas que lo haga, incluida la gestión avanzada de fecha y hora. También encontrarás información sobre cómo crear o bien modificar archivos, incluidos aquellos que suben los usuarios.

Uso de printf

Ya has visto las funciones `print` y `echo`, que lo único que hacen es enviar el texto al navegador. Pero una función mucho más capaz, `printf`, controla el formato de la salida al permitirte añadir caracteres especiales de formato en una cadena. Para cada carácter de formato, `printf` espera que pases un argumento, y se producirá la presentación usando ese formato. Por ejemplo, el siguiente ejemplo utiliza el especificador de conversión `%d` para mostrar el valor `3` en decimal:

```
printf("There are %d items in your basket", 3);
```

Si sustituyes `%d` por `%b`, el valor 3 se mostrará en binario (11). La Tabla 7-1 muestra los especificadores de conversión admitidos.

Tabla 7-1. Especificadores de conversión de printf

Especificador	Acción de conversión sobre arg	Ejemplo (arg de 123)
`%`	Muestra el carácter `%` (no se requiere `arg`)	`%`
`b`	Muestra `arg` como un entero binario	`1111011`
`c`	Muestra el carácter ASCII para `arg`	{
`d`	Muestra `arg` como un entero decimal	`123`
`e`	Muestra `arg` en notación científica	`1.23000e+2`
`f`	Muestra `arg` como punto flotante	`123.000000`
`o`	Muestra `arg` como un entero octal	`173`
`s`	Muestra `arg` como una cadena	`123`
`u`	Muestra `arg` como un decimal sin signo	`123`
`x`	Muestra `arg` en hexadecimal, en minúsculas	`7b`
`X`	Muestra `arg` en hexadecimal, en mayúsculas	`7B`

Puedes tener tantos especificadores como desees en una función `printf`, siempre que pases el correspondiente número de argumentos y siempre que cada especificador esté precedido por el signo `%`. Por lo tanto, el siguiente código es válido, y dará como resultado `"My name is Simon. I'm 33 years old, which is 21 in hexadecimal"`:

```
printf("My name is %s. I'm %d years old, which is %X in
  hexadecimal", 'Simon', 33, 33);
```

Si omites cualquier argumento, recibirás un error de análisis que te informará de que se ha encontrado inesperadamente el paréntesis derecho,).

Un ejemplo más práctico de `printf` define los colores en HTML usando valores decimales. Por ejemplo, supongamos que quieres un color que tiene el valor formado por la tríada de 65 rojo, 127 verde y 245 azul, pero quieres que convertirlo automáticamente en hexadecimal. He aquí una solución fácil:

```
printf("<span style='color:#%X%X%X'>Hello</span>", 65, 127, 245);
```

Comprueba cuidadosamente el formato de la especificación de color contenida entre los apóstrofes (`''`). Primero viene el signo de libra, o de hash, (#) esperado por la especificación de color. Entonces vendrán tres especificadores con formato `%X`, uno para cada uno de los números. La salida resultante de este comando es la siguiente:

```
<span style='color:#417FF5'>Hello</span>
```

Generalmente, será conveniente usar variables o expresiones como argumentos para imprimir. Por ejemplo, si has almacenado valores para tus colores en las tres variables `$r`, `$g` y `$b`, podrías crear un color más oscuro con este cambio:

```
printf("<span style='color:#%X%X%X'>Hello</span>", $r-20, $g-20, $b-20);
```

Ajustes de la precisión

No solo se puede especificar un tipo de conversión, sino que también se puede establecer la precisión del resultado mostrado. Por ejemplo, los importes de moneda se visualizan normalmente solo con dos dígitos de precisión. Sin embargo, después de un cálculo, un valor puede tener una precisión mayor de dos dígitos, como 123.42 / 12, cuyo resultado es 10.285. Para garantizar que dichos valores se almacenan internamente de forma correcta, pero se visualizan con solo dos dígitos de precisión, puedes insertar la cadena `".2"` entre el signo `%` y el especificador de conversión:

```
printf("The result is: $%.2f", 123.42 / 12);
```

La salida de este comando es la siguiente:

```
The result is $10.29
```

Pero en realidad tienes aún mucho más control, porque también puedes especificar si deseas rellenar la salida con ceros o espacios anteponiendo al especificador determinados valores. El Ejemplo 7-1 muestra cuatro combinaciones posibles.

Ejemplo 7-1. Ajuste de la precisión

```
<?php
  echo "<pre>"; // Enables viewing of the spaces

  // Pad to 15 spaces
  printf("The result is $%15f\n", 123.42 / 12);

  // Pad to 15 spaces, fill with zeros
  printf("The result is $%015f\n", 123.42 / 12);

  // Pad to 15 spaces, 2 decimal places precision
  printf("The result is $%15.2f\n", 123.42 / 12);

  // Pad to 15 spaces, 2 decimal places precision, fill with zeros
  printf("The result is $%015.2f\n", 123.42 / 12);

  // Pad to 15 spaces, 2 decimal places precision, fill with #
symbol
  printf("The result is $%'#15.2f\n", 123.42 / 12);
?>
```

El resultado de este ejemplo tiene el siguiente aspecto:

```
The result is $      10.285000
The result is $00000010.285000
The result is $          10.29
The result is $000000000010.29
The result is $###########10.29
```

El funcionamiento es sencillo si se analiza de derecha a izquierda (ver la Tabla 7-2). Fíjate en esto:

- El carácter más a la derecha es el especificador de conversión: en este caso, `f` para el punto flotante.
- Inmediatamente antes del especificador de conversión, si hay un punto y un número juntos, entonces la precisión de la salida la especifica el valor del número.
- Independientemente de si hay o no un prescriptor de precisión, si hay un número, entonces este representa el número de caracteres a los que se debe rellenar la salida. En en el ejemplo anterior, son 15 caracteres. Si la salida ya es igual a o mayor que la longitud del relleno, entonces este argumento se ignora.
- El parámetro más a la izquierda después del símbolo `%` que se permite es un `0`, que se ignora a menos que se haya establecido un valor de relleno, en cuyo caso la salida se rellena con ceros en lugar de espacios. Si se requiere un carácter de relleno distinto de cero o del espacio, puedes utilizar cualquiera que elijas siempre que lo precedas con una sola comilla, como esta: `'#`.
- A la izquierda está el signo `%`, que inicia la conversión.

Tabla 7-2. Componentes de los especificadores de conversión

Inicio	Cáracter relleno	Número	Precisión	Especificador	Ejemplo
%		15		f	10.285000
%	0	15	.2	f	000000000010.29
%	'#	15	.4	f	########10.2850

Relleno de cadenas

También podemos rellenar cadenas con la longitud deseada (como podemos hacer con los números), seleccionar diferentes caracteres de relleno, e incluso elegir entre la justificación izquierda y derecha. El Ejemplo 7-2 muestra un compendio de varios ejemplos.

Ejemplo 7-2. Relleno de cadenas

```
<?php
  echo "<pre>"; // Enables viewing of the spaces

  $h = 'Rasmus';

  printf("[%s]\n",       $h);  // Standard string output
  printf("[%12s]\n",     $h);  // Right justify with spaces to width 12
  printf("[%-12s]\n",    $h);  // Left justify with spaces
  printf("[%012s]\n",    $h);  // Pad with zeros
  printf("[%'#12s]\n\n", $h);  // Use the custom padding character '#'

  $d = 'Rasmus Lerdorf';       // The original creator of PHP

  printf("[%12.8s]\n",   $d);  // Right justify, cutoff of 8
characters
  printf("[%-12.12s]\n", $d);  // Left justify, cutoff of 12
characters
  printf("[%-'@12.10s]\n", $d); //Left justify, pad with '@',
                                  cutoff 10 chars
?>
```

Observa que para propósitos de maquetación en una página web, he usado la etiqueta `<pre>` de HTML para preservar todos los espacios y el carácter `\n` de salto de línea después de cada una de las líneas a mostrar. El resultado de este ejemplo es el siguiente:

```
[Rasmus]
[      Rasmus]
[Rasmus      ]
[000000Rasmus]
[######Rasmus]

[    Rasmus L]
[Rasmus Lerdo]
[Rasmus Ler@@]
```

Cuando especificas un valor de relleno, se ignorarán las cadenas de una longitud igual o superior a ese valor, *a menos* que se indique un valor de corte que reduzca las cadenas a un valor inferior al valor de relleno.

La Tabla 7-3 muestra los componentes disponibles para los especificadores de conversión de cadenas.

Tabla 7-3. Componentes del especificador de conversión de cadenas

Inicia	Justificación	Carácter	Número	Umbral	Especificador	Ejemplo
%					s	[Rasmus]
%	-		10		s	[Rasmus]
%		'#	8	.4	s	[####Rasm]

Uso de sprintf

A menudo, no deseas obtener el resultado de una conversión, pero necesitas utilizarlo en otra parte del código. Aquí es donde entra en juego la función `sprintf`. Con ella, puedes enviar la salida a otra variable en lugar de hacerlo al navegador.

Puedes usarla para hacer una conversión, como en el ejemplo siguiente, que devuelve el valor de cadena hexadecimal para el grupo de color RGB 65, 127, 245 en `$hexstring`:

```
$hexstring = sprintf("%X%X%X", 65, 127, 245);
```

O bien, puedes querer almacenar el mensaje para visualizarlo más tarde:

```
$out = sprintf("The result is: $%.2f", 123.42 / 12);
echo $out;
```

Funciones de fecha y hora

Para llevar un registro de la fecha y hora, PHP usa marcas de tiempo estándar de Unix, que son simplemente el número de segundos desde el comienzo del 1 de enero de 1970. Para determinar la fecha y hora actual, puedes utilizar la función `time`:

```
echo time();
```

Debido a que el valor se almacena en segundos, para obtener la indicación de fecha y hora para este instante la próxima semana, utilizarías lo siguiente, que añade 7 días × 24 horas × 60 minutos × 60 segundos al valor devuelto:

```
echo time() + 7 * 24 * 60 * 60;
```

Si deseas crear una indicación de fecha y hora para una fecha determinada, puedes utilizar la función `mktime`. Su salida es la indicación de fecha y hora `946684800` para el primer segundo del primer minuto de la primera hora del primer día del año 2000:

```
echo mktime(0, 0, 0, 1, 1, 2000);
```

Los parámetros a pasar son, en orden de izquierda a derecha:

- El número correspondiente a la hora (0–23)
- El número correspondiente al minuto (0–59)
- El número correspondiente al segundo (0–59)
- El número correspondiente al mes (1–12)
- El número correspondiente al día (1–31)
- El año (1970–2038, o 1901–2038 con PHP 5.1.0+ en sistemas de 32 bits con signo)

Te preguntarás por qué está limitado a los años 1970 a 2038. Bueno, es porque los desarrolladores originales de Unix pensaron en una fecha base desde la que ningún programador necesitara remitirse a fechas anteriores, y eligieron el comienzo del año 1970.

Afortunadamente, a partir de la versión 5.1.0, PHP es compatible con los sistemas que usan enteros de 32 bits con signo para la indicación de fecha y hora, y admiten las fechas de 1901 a 2038. Sin embargo, eso introduce un problema aún peor que el original, porque los diseñadores de Unix también decidieron que nadie iba a seguir usando Unix después de unos 70 años más o menos y, por lo tanto, creían que podían salirse con la suya almacenando la indicación de fecha y hora como un valor de 32 bits, pero este recurso se agotará el 19 de enero de 2038.

Esto dará lugar a lo que se conoce como el error del Y2K38, (muy parecido al error del milenio, causado por almacenar la parte de la fecha correspondiente a los años como valores de dos dígitos, y que también se debía corregir). PHP ha introducido la clase `DateTime` en la versión 5.2 para superar este problema, pero funcionará solo en la arquitectura de 64 bits, que son la mayoría de los ordenadores actualmente (pero compruébalo antes de usarlo).

Para visualizar la fecha, utiliza la función `date`, que admite una gran cantidad de opciones de formato que te permiten visualizar la fecha de la forma que desees. El formato es el siguiente:

```
date($format, $timestamp);
```

El parámetro `$format` debe ser una cadena que contenga especificadores de formato como se detalla en la Tabla 7-4, y `$timestamp` debe ser una indicación Unix de fecha y hora. Para ver la lista completa de especificadores, consulta la documentación (*http://php.net/manual/es/function.date.php)*. El siguiente comando mostrará la fecha y hora en curso en el formato `"Thursday July 6th, 2017 - 1:38pm"`:

```
echo date("l F jS, Y - g:ia", time());
```

Tabla 7-4. Principales especificadores de formato de la función date

Formato	Descripción	Valor devuelto
Especificadores del día		
d	Día del mes, dos dígitos, con ceros a la izquierda	01 al 31
D	Día de la semana, tres letras	Mon a Sun
j	Día del mes, sin ceros a la izquierda	1 al 31
l	Día de la semana, nombres completos	Sunday a Saturday
N	Día de la semana, numérico, Monday a Sunday	1 al 7
S	Sufijo para el día del mes (útil con especific.)	st, nd, rd, or th
w	Día de la semana, numérico, Sunday a Saturday	0 al 6
z	Día del año	0 al 365
Especificador de la semana		
W	Número de la semana del año	01 al 52
Especificadores del mes		
F	Nombre del mes	January a December
m	Nombre del mes con ceros a la izquierda	01 al 12
M	Nombre del mes, tres letras	Jan a Dec
n	Número del mes, sin ceros a la izquierda	1 al 12
t	Número de días en un mes dado	28 al 31
Especificadores del año		
L	Año bisiesto	1 = Yes, 0 = No
y	Año, 2 dígitos	00 al 99
Y	Año, 4 dígitos	0000 al 9999
Especificadore de tiempo		
a	Antes o después del mediodía, minúsculas	am o pm
A	Antes o después del mediodía, mayúsculas	AM o PM
g	Hora del día, formato 12 horas, sin ceros a la izq.	1 a 12
G	Hora del día, formato 24 horas, sin ceros a la izq.	0 a 23
h	Hora del día, formato 12 horas, con ceros a la izq.	01 t a 12
H	Hora del día, formato 24 horas, con ceros a la izq.	00 a 23
i	Minutos, con ceros a la izquierda	00 a 59
s	Segundos, con ceros a la izquierda	00 a 59

Constantes de fecha

Hay un número de constantes útiles que puedes usar con el comando `date` para devolver la fecha en formatos específicos. Por ejemplo, `date(DATE_RSS)` devuelve la fecha y hora en el formato válido para un feed RSS. Algunos de los más utilizados son los siguientes:

`DATE_ATOM`
: Este es el formato de las fuentes Atom. El formato PHP es `"Y-m-d\TH:i:sP"` y la salida del ejemplo es `"2022-10-22T12:00:00+00:00"`.

`DATE_COOKIE`
: Este es el formato de las cookies establecidas desde un servidor web o JavaScript. El formato PHP es `"l, d-M-y H:i:s T"` y la salida del ejemplo es `"Wednesday, 26-Oct-22 12:00:00 UTC"`.

`DATE_RSS`
: Este es el formato de los canales RSS. El formato PHP es `"D, d M Y H:i:s O"` y la salida del ejemplo es `"Wed, 26 Oct 2022 12:00:00 UTC"`.

`DATE_W3C`
: Este es el formato del Consorcio World Wide Web. El formato PHP es `"Y- m-d\TH:i:sP"` y la salida de ejemplo es `"2022-10-26T12:00:00+00:00"`.

La lista completa se puede consultar en la documentación (*http://php.net/manual/es/class.datetime.php)*.

Uso de la verificación de fecha

Has visto cómo presentar una fecha válida en varios formatos. ¿Pero cómo puedes comprobar si un usuario ha enviado una fecha válida al programa? La respuesta es pasar el mes, día y año a la función `checkdate`, que devuelve el valor `TRUE` si la función fecha es válida o `FALSE` si no lo es.

Por ejemplo, si se introduce el 31 de septiembre de cualquier año, siempre será una fecha no válida.

El Ejemplo 7-3 muestra el código que puedes usar para la verificación. Tal como aparece, encontrará la fecha no válida.

Ejemplo 7-3. Verificación de la validez de una fecha

```
<?php
  $month = 9;      // September (only has 30 days)
  $day   = 31;     // 31st
  $year  = 2022;   // 2022

  if (checkdate($month, $day, $year)) echo "Date is valid";
  else echo "Date is invalid";
?>
```

Manejo de archivos

Por potente que sea, MySQL no es la única (o necesariamente la mejor) manera de almacenar los datos en un servidor web. A veces puede ser más rápido y más conveniente acceder directamente a los archivos en el disco duro. Los casos en los que puede ser necesario hacer esto son cuando se modifican imágenes tales como avatares de usuario cargados o con archivos de registro que desees procesar.

Primero, sin embargo, una nota sobre los nombres de archivos: si escribes código que se puede usar en varias instalaciones de PHP, no hay manera de saber si estos sistemas son sensibles a mayúsculas y minúsculas. Por ejemplo, los nombres de archivo de Windows y macOS no distinguen entre mayúsculas y minúsculas, pero los de Linux y Unix sí. Por lo tanto, debes suponer siempre que el sistema distingue entre mayúsculas y minúsculas, y seguir una convención como por ejemplo los nombres de archivo en minúsculas.

Verificación de la existencia de un archivo

Para determinar si un fichero existe, puedes utilizar la función `file_exists`, que devuelve `TRUE` o `FALSE`, y se utiliza así:

```
if (file_exists("testfile.txt")) echo "File exists";
```

Creación de archivos

En este momento, *testfile.txt* no existe, así que vamos a crearlo y a escribir unas cuantas líneas. Escribe el Ejemplo 7-4 y guárdalo como *testfile.php*.

Ejemplo 7-4. Creación de un sencillo archivo de texto

```
<?php // testfile.php
  $fh = fopen("testfile.txt", 'w') or die("Failed to create file");

  $text = <<<_END Line 1
Line 2
Line 3
_END;

  fwrite($fh, $text) or die("Could not write to file");
  fclose($fh);
  echo "File 'testfile.txt' written successfully";
?>
```

Si un programa llama a la función `die`, el archivo abierto se cerrará automáticamente como parte de la terminación del programa.

Cuando ejecutas lo anterior en un navegador, si todo está bien, recibirás el mensaje `File 'testfile.txt' written successfully`. Si recibes un mensaje de error, es posible que el disco duro esté lleno o, lo que es más probable, que no tengas permiso

para crear el archivo o escribir en él, en cuyo caso deberás modificar los atributos de la carpeta de destino de acuerdo con tu sistema operativo. De lo contrario, el archivo *testfile.txt* debería residir en la misma carpeta en la que guardaste el programa *testfile.php*. Si abres el archivo en un editor de texto o de programa; el contenido se verá así:

```
Line 1
Line 2
Line 3
```

Este sencillo ejemplo muestra la secuencia que sigue el manejo de archivos:

1. Comienza siempre abriendo el archivo. Lo haces mediante una llamada a `fopen`.
2. A continuación puedes llamar a otras funciones. Aquí escribimos en el archivo (`fwrite`), pero puedes también leer de un archivo existente (`fread` o `fgets`) y hacer otras cosas.
3. Termina por cerrar el archivo (`fclose`). Aunque el programa realiza esta acción automáticamente cuando termina, deberías proceder a hacer una limpieza y cerrar el archivo cuando hayas terminado.

Cada archivo abierto requiere un recurso de archivo para que PHP pueda acceder a él y administrarlo. El ejemplo anterior establece la variable `$fh` (que elegí para representar al identificador de archivos, *file handle*) en el valor que devuelve la función `fopen`. A partir de entonces, cada función de manejo de archivos que accede al archivo abierto, como `fwrite` o `fclose`, debe pasar `$fh` como parámetro para identificar el archivo al que se está accediendo. No te preocupes por el contenido de la variable `$fh`; es un número que PHP usa para referirse a información interna sobre el archivo, tú simplemente pasas la variable a otras funciones.

En caso de fallo, `fopen` devuelve `FALSE`. El ejemplo anterior muestra una sencilla forma de capturar y responder al fallo: llama a la función `die` para finalizar el programa y presentar al usuario un mensaje de error. Una aplicación web nunca abortaría de esta forma abrupta (en su lugar crearíamos una página web con un mensaje de error), pero es aceptable para nuestros propósitos de prueba.

Observa el segundo parámetro de la llamada a `fopen`. Se trata sencillamente del carácter `w`, que le dice a la función que abra el archivo para escribir. La función crea el archivo si todavía no existe. Ten cuidado cuando juegues con estas funciones: si el archivo ya tiene el parámetro de modo `w`, hace que la llamada a `fopen` borre los contenidos antiguos (¡incluso si no escribes nada nuevo!).

Hay varios parámetros de modo que se pueden utilizar, como se detalla en la Tabla 7-5. Los modos que incluyen un signo `+` se explican con más detalle en la sección "Actualización de archivos", en la página 147.

Tabla 7-5. Modos compatibles con fopen

Modo	Acción	Descripción
`'r'`	Lee desde el principio del archivo	Lo abre solo para lectura; coloca el puntero al principio del archivo. Devuelve `FALSE` si el archivo no existe todavía.
`'r+'`	Lee desde el principio	Lo abre para lectura y escritura; coloca el puntero al del y permite escribir principio del archivo. Devuelve `FALSE` si el archivo no existe todavía.
`'w'`	Escribe desde el principio del archivo y lo trunca	Lo abre solo para escritura; coloca el puntero al principio del archivo y trunca la longitud del archivo a cero. Si el archivo no existe, intenta crearlo.
`'w+'`	Escribe desde el principio del archivo, trunca el archivo y permite la lectura	Lo abre para lectura y escritura; coloca el puntero al principio del archivo y trunca la longitud del archivo a cero. Si el archivo no existe, intenta crearlo.
`'a'`	Añade al final del archivo	Lo abre solo para escritura; coloca el puntero al final del archivo. Si el archivo no existe, intenta crearlo.
`'a+'`	Añade al final del archivo y permite la lectura	Lo abre para lectura y escritura; coloca el puntero al principio del archivo. Si el archivo no existe, intenta crearlo.

Lectura de archivos

La manera más fácil de leer un archivo de texto es extraer una línea completa con `fgets` (piensa que la `s` final representa a *string*), como en el Ejemplo 7-5.

Ejemplo 7-5. Lectura de un archivo con fgets

```
<?php
  $fh = fopen("testfile.txt", 'r') or
    die("File does not exist or you lack permission to open it");

  $line = fgets($fh);
  fclose($fh);
  echo $line;
?>
```

Si has creado el archivo como se muestra en el Ejemplo 7-4, obtendrás la primera línea:

```
Line 1
```

Puedes recuperar varias líneas o porciones de líneas mediante la función `fread`, como en el Ejemplo 7-6.

Ejemplo 7-6. Lectura de un archivo con fread

```
<?php
  $fh = fopen("testfile.txt", 'r') or
    die("File does not exist or you lack permission to open it");

  $text = fread($fh, 3); fclose($fh);
```

```
  echo $text;
?>
```

He solicitado tres caracteres en la llamada `fread`, así que el programa muestra esto:

```
Lin
```

La función `fread` se utiliza normalmente con datos binarios. Si la utilizas con datos de texto que ocupan más de una línea, recuerda contar los caracteres del salto de línea.

Copia de archivos

Probemos la función `copy` de PHP para crear un clon de *testfile.txt*. Escribe el Ejemplo 7-7, guárdalo como *copyfile.php* y luego llama al programa en tu navegador.

Ejemplo 7-7. Copia de un archivo

```
<?php // copyfile.php
  copy('testfile.txt', 'testfile2.txt') or die("Could not copy file");
  echo "File successfully copied to 'testfile2.txt'";
?>
```

Si vuelves a revisar tu carpeta, verás que ahora tienes el nuevo archivo *testfile2.txt*. Por cierto, si no deseas que tus programas incurran en un intento de copia fallido, puedes probar la sintaxis alternativa del Ejemplo 7-8. Para ello se utiliza el operador `!` (`NOT`) como una abreviatura rápida y fácil. Colocado delante de una expresión, aplica el operador `NOT` a la misma, por lo que la declaración equivalente aquí en español comenzaría "Si no puedes copiar...".

Ejemplo 7-8. Sintaxis alternativa para copiar un archivo

```
<?php // copyfile2.php
  if (!copy('testfile.txt', 'testfile2.txt'))
    echo "Could not copy file";
  else echo "File successfully copied to 'testfile2.txt'";
?>
```

Movimiento de archivos

Para mover un archivo, lo renombras con la función `rename`, como en el Ejemplo 7-9.

Ejemplo 7-9. Mover un archivo

```
<?php // movefile.php
  if (!rename('testfile2.txt', 'testfile2.new'))
    echo "Could not rename file";
  else echo "File successfully renamed to 'testfile2.new'";
?>
```

También puedes utilizar la función `rename` en directorios. Para evitar cualquier mensaje de advertencia si el archivo original no existe, puedes llamar antes a la función `file_exists` para verificarlo.

Eliminación de archivos

Borrar un archivo es solo cuestión de usar la función `unlink` para borrarlo del sistema de archivos, como se hace en el Ejemplo 7-10.

Ejemplo 7-10. Eliminación de un archivo

```
<?php // deletefile.php
  if (!unlink('testfile2.new')) echo "Could not delete file";
  else echo "File 'testfile2.new' successfully deleted";
?>
```

Siempre que accedas directamente a los archivos de tu disco duro, también debes tener la seguridad de que es imposible que tu sistema de archivos se vea comprometido. Por ejemplo, si eliminas un archivo de una entrada de usuario, debes estar absolutamente seguro de que se trata de un archivo que se puede eliminar de forma segura y que el usuario puede eliminarlo.

Como en el caso de mover un archivo, se mostrará un mensaje de advertencia si el archivo no existe, advertencia que puedes evitar mediante `file_exists` para comprobar primero su existencia antes de llamar a `unlink`.

Actualización de archivos

A menudo, querrás añadir más datos a un archivo guardado. Esto se puede hacer de muchas maneras. Puedes usar uno de los modos para añadir escritura (ver la Tabla 7-5), o puedes simplemente abrir un archivo para leer y escribir usando uno de los otros modos compatibles con la escritura, y mover el puntero del fichero al lugar correcto dentro del fichero en el que deseas escribir o del que quieres leer.

El *puntero del archivo* es la posición dentro de un archivo en la que tendrá lugar el siguiente acceso al archivo, ya sea una lectura o una escritura. No es lo mismo que el *manejador de archivos* (que está almacenado en la variable `$f` en el Ejemplo 7-4), que contiene detalles sobre el archivo al que se está accediendo.

Puedes ver cómo funciona escribiendo el Ejemplo 7-11 y lo puedes guardar como *update.php*. Después llámalo en tu navegador.

Ejemplo 7-11. Actualización de una archivo

```
<?php // update.php
  $fh   = fopen("testfile.txt", 'r+') or die("Failed to open file");
  $text = fgets($fh);

  fseek($fh, 0, SEEK_END);
  fwrite($fh, "$text") or die("Could not write to file");
  fclose($fh);

  echo "File 'testfile.txt' successfully updated";
?>
```

Este programa abre *testfile.txt* para lectura y escritura configurando el modo con `'r+'`, que pone el puntero al principio del archivo. A continuación, utiliza la función `fgets` para leer una sola línea del archivo (hasta el primer salto de línea). Después se llama a la función `fseek` para mover el puntero del archivo directamente al final del archivo, en cuyo punto la línea del texto que se extrajo del inicio del archivo (almacenada en `$text`) se agrega al final del archivo y este se cierra. El archivo resultante tiene el siguiente aspecto:

```
Line 1
Line 2
Line 3
Line 1
```

La primera línea se ha copiado con éxito y luego se ha añadido al final del archivo.

Tal y como se utiliza aquí, además de la gestión del archivo `$fh`, la función `fseek` ha pasado otros dos parámetros, `0` y `SEEK_END`. `SEEK_END` le dice a la función que mueva el puntero del fichero al final del archivo, y `0` le dice cuántas posiciones se debería mover hacia atrás desde ese punto. En el caso del Ejemplo 7-11, se utiliza el valor `0` porque el puntero debe permanecer al final del archivo.

Hay otras dos opciones de búsqueda disponibles para la función `fseek`: `SEEK_SET` y `SEEK_CUR`. La opción `SEEK_SET` le dice a la función que fije el puntero del archivo en la posición exacta dada por el parámetro anterior. De este modo, el siguiente ejemplo desplaza el puntero a la posición 18:

```
fseek($fh, 18, SEEK_SET);
```

`SEEK_CUR` sitúa el puntero del archivo en la posición actual *más* el valor del desplazamiento indicado. Por lo tanto, si el puntero del archivo se encuentra ahora en la posición `18`, la siguiente llamada lo moverá a la posición `23`:

```
fseek($fh, 5, SEEK_CUR);
```

Aunque esto no es recomendable a menos que tengas razones especiales para hacerlo, es incluso posible utilizar archivos de texto como este (pero con longitudes de línea fijas) como simples bases de datos de archivos planos. Tu programa puede usar `fseek` para moverte hacia adelante y hacia atrás dentro de dicho archivo para recuperar, actualizar y agregar nuevos registros. También puedes borrar registros sobrescribiéndolos con cero caracteres, etc.

Bloqueo de archivos debido a accesos múltiples

A menudo, muchos usuarios llaman a programas web al mismo tiempo. Si más de una persona intenta escribir en un archivo simultáneamente, este puede corromperse. Y si alguien escribe en él mientras otro está leyendo de él, el archivo no se ve afectado, pero la persona que lo está leyendo puede obtener resultados extraños. Para gestionar usuarios

que acceden de forma simultánea, debes utilizar la función `flock` de bloqueo de archivos. Esta función pone en cola todas las demás solicitudes para acceder a un archivo hasta que tu programa libera el bloqueo. Por lo tanto, siempre que tus programas usen acceso de escritura en archivos a los que pueden acceder simultáneamente otros usuarios, también debes agregar el bloqueo de archivos como en el Ejemplo 7-12, que es una versión actualizada del Ejemplo 7-11.

Ejemplo 7-12. Actualización de un archivo con bloqueo del mismo

```
<?php
  $fh   = fopen("testfile.txt", 'r+') or die("Failed to open file");
  $text = fgets($fh);

  if (flock($fh, LOCK_EX))
  {
    fseek($fh, 0, SEEK_END);
    fwrite($fh, "$text") or die("Could not write to file");
    flock($fh, LOCK_UN);
  }

  fclose($fh);
  echo "File 'testfile.txt' successfully updated";
?>
```

El bloqueo de archivos es un truco para preservar el mejor tiempo de respuesta posible a los visitantes de tu sitio web: antes de realizar un cambio en un archivo, hay que bloquearlo directamente y, a continuación, desbloquearlo inmediatamente después. Si se bloquea un archivo durante más tiempo, la aplicación se ralentizará innecesariamente. Esta es la razón por la cual las llamadas a `flock` en el Ejemplo 7-12 se producen directamente antes y después de la llamada a `fwrite`.

La primera llamada a `flock` establece un bloqueo de archivo exclusivo al archivo al que se refiere `$fh` mediante el parámetro `LOCK_EX`:

```
flock($fh, LOCK_EX);
```

A partir de este momento, ningún otro proceso puede escribir en el archivo (o incluso leerlo) hasta que se libere el bloqueo con el parámetro `LOCK_UN`, así:

```
flock($fh, LOCK_UN);
```

En cuanto se libera el bloqueo, se vuelve a permitir el acceso de otros procesos al fichero. Esta es una de las razones por las que debes volver a buscar el punto en el que deseas acceder a un archivo cada vez que necesites leer o escribir datos; otro proceso podría haber cambiado el archivo desde el último acceso.

Sin embargo, ¿te has dado cuenta de que la llamada para solicitar un bloqueo exclusivo está anidada como parte de una declaración `if`? Esto se debe a que `flock` no es compatible con todos los sistemas; por lo tanto, es aconsejable comprobar si se ha hecho efectivo el bloqueo, por si acaso no se ha podido conseguir.

Otra cosa que debes considerar es que `flock` es lo que se conoce como una candado *de aviso*. Esto significa que bloquea solo otros procesos que llaman a la función. Si tienes cualquier código que va directamente y modifica los archivos sin implementar el bloqueo de archivos con `flock`, siempre anulará el bloqueo y podría causar estragos en tus archivos.

Por cierto, implementar el bloqueo de archivos y luego accidentalmente dejarlo fuera en una sección de código puede llevar a un error extremadamente difícil de localizar.

`flock` no funcionará en NFS ni en muchos otros sistemas de archivos en red. Además, al usar un servidor multiproceso como ISAPI, no puedes confiar en `flock` para proteger los archivos contra otros scripts PHP que se ejecutan en hilos paralelos de la misma instancia del servidor. Además, `flock` no es compatible con ningún sistema que utilice el antiguo FAT, como son las versiones anteriores de Windows.

En caso de duda, puedes intentar hacer un bloqueo rápido en un archivo de prueba al inicio de un programa y comprobar que puedes bloquear el archivo. No olvides desbloquearlo (y tal vez borrarlo si no es necesario) después de comprobarlo.

Recuerda también que cualquier llamada a la función `die` realiza automáticamente un desbloqueo del archivo y lo cierra como parte de la finalización del programa.

Lectura de archivos completos

Una función útil para leer un archivo completo sin tener que usar gestores de archivos es `file_get_contents`. Es muy fácil de usar, como se puede ver en el Ejemplo 7-13.

Ejemplo 7-13. Uso de file_get_contents

```
<?php
  echo "<pre>"; // Enables display of line feeds
  echo file_get_contents("testfile.txt");
  echo "</pre>"; // Terminates <pre> tag
?>
```

Pero la función tiene en realidad mucha más utilidad, porque también se puede utilizar para obtener un archivo de un servidor a través de Internet, como en el Ejemplo 7-14, que solicita el HTML de la página de inicio de Marcombo, y luego lo muestra como si el usuario hubiera navegado a la propia página. El resultado será similar al de la Figura 7-1.

Ejemplo 7-14. Captura de la página de inicio de Marcombo

```
<?php
  echo file_get_contents("http://marcombo.com");
?>
```

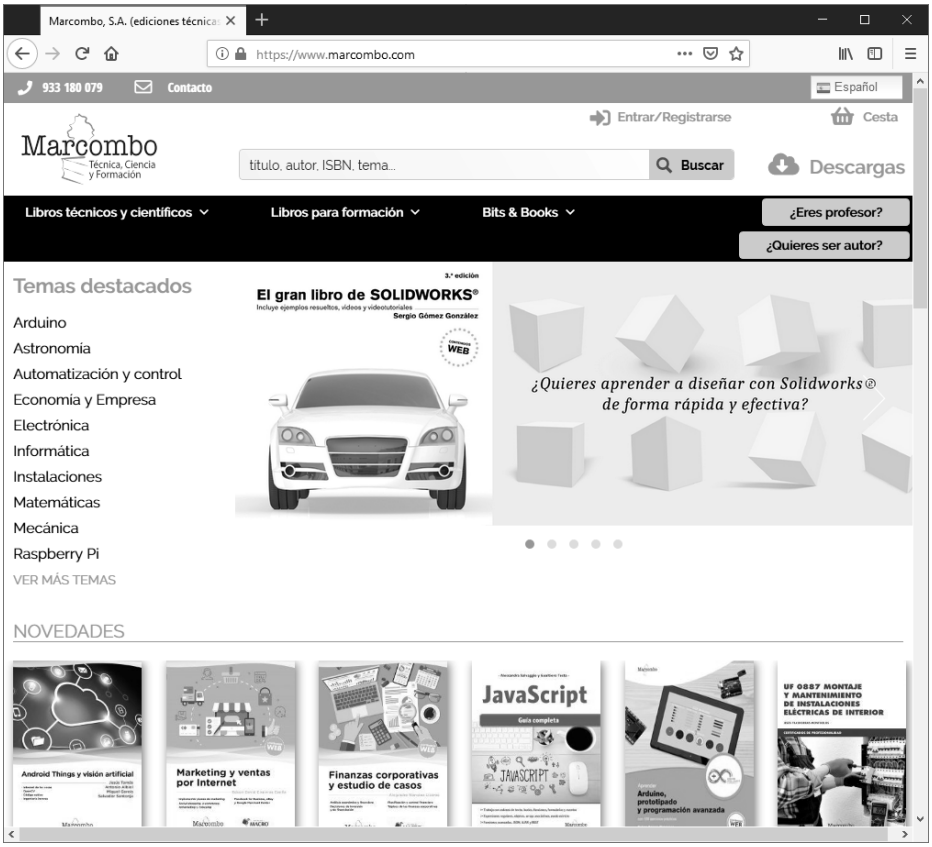

Figura 7-1. Página de inicio de Marcombo capturada con file_get_contents

Carga de archivos

Cargar archivos a un servidor web es un tema que puede parecer desalentador a muchas personas, pero en realidad no se puede hacer que sea mucho más fácil. Todo lo que necesitas hacer para subir un archivo desde un formulario es elegir un tipo especial de codificación llamado `multipart/form-data`, y tu navegador se encargará del resto. Para ver cómo funciona, escribe el programa del Ejemplo 7-15 y guárdalo como *upload.php*. Cuando lo ejecutes, verás un formulario en tu navegador que te permite subir el archivo que hayas elegido.

Ejemplo 7-15. Cargador de imagen upload.php

```
<?php // upload.php
  echo <<<_END
    <html><head><title>PHP Form Upload</title></head><body>
    <form method='post' action='upload.php' enctype='multipart/form-data'>
    Select File: <input type='file' name='filename' size='10'>
    <input type='submit' value='Upload'>
    </form>
_END;

  if ($_FILES)
  {
    $name = $_FILES['filename']['name'];
    move_uploaded_file($_FILES['filename']['tmp_name'], $name);
    echo "Uploaded image '$name'<br><img src='$name'>";
  }

  echo "</body></html>";
?>
```

Examinemos este programa sección por sección. La primera línea de la declaración `echo`, formada por varias líneas, inicia el documento HTML, muestra el título y, a continuación, inicia el cuerpo del documento.

Después llegamos al formulario, que selecciona el método POST de envío de formularios, establece el destino de los datos enviados al programa *upload.php* (el programa mismo) e indica al navegador web que los datos enviados se deben codificar a través del tipo de contenido de `multi part/form-data`.

Con el formulario configurado, las líneas siguientes muestran el aviso `Select File:` (seleccionar archivo) y luego solicitan dos entradas. La primera petición es para un archivo; usa un tipo de entrada de `file`, un nombre de `filename` y un campo de entrada con un ancho de 10 caracteres. La segunda entrada que se requiere es solo un botón de envío al que se le asocia la etiqueta `Upload` (cargar) (que sustituye el texto del botón por defecto de `Submit Query` (enviar consulta). Y luego se cierra el formulario.

Este breve programa muestra una técnica muy común en la programación web, en la que se llama dos veces a un programa: una vez cuando el usuario visita por primera vez una página, y otra vez cuando el usuario pulsa el botón de envío.

El código PHP para recibir los datos cargados es bastante simple, porque todos los archivos cargados se colocan en la matriz asociativa del sistema `$_FILES`. Por lo tanto, es suficiente una rápida comprobación para ver si `$_FILES` contiene algo para determinar si el usuario ha subido un archivo. Esto se hace con la sentencia `if ($_FILES)`.

La primera vez que el usuario visita la página, antes de subir un archivo, `$_FILES` está vacía, por lo que el programa se salta este bloque de código. Cuando el usuario sube un archivo, el programa se ejecuta otra vez y descubre un elemento en la matriz `$_FILES`.

Una vez que el programa se da cuenta de que se ha cargado un archivo, el nombre real, tal y como se lee en el ordenador que lo ha cargado, se recupera y se coloca en la variable `$name`. Ahora todo lo que se necesitas es mover el archivo cargado desde la ubicación temporal en la que lo almacenó PHP a una ubicación permanente. Hacemos esto con la función `move_uploaded_file`, le pasamos el nombre original del archivo y, con ese nombre, se guarda en el directorio en uso.

Finalmente, la imagen cargada se muestra dentro de una etiqueta `IMG`, y el resultado se debería ver como el de la Figura 7-2.

Si ejecutas este programa y recibes un mensaje de advertencia como `Permission denied` para la llamada de la función `move_uploaded_file`, es posible que no tengas los permisos necesarios establecidos para la carpeta en la que se está ejecutando el programa.

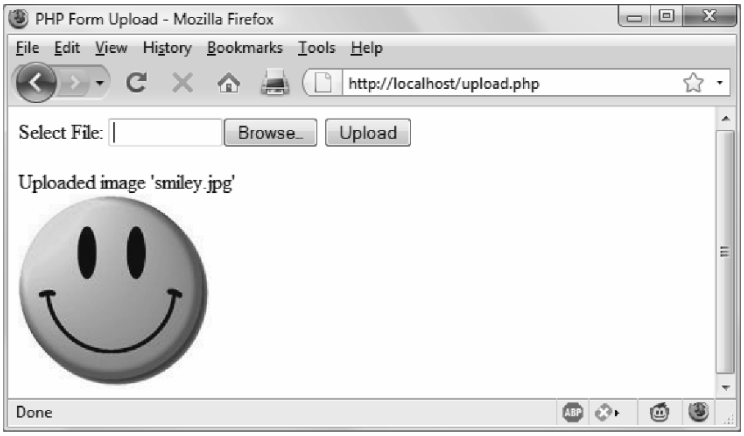

Figura 7-2. Carga de una imagen como datos de formulario

Uso de $_FILES

Cuando se carga un archivo se almacenan cinco cosas en la matriz `$_FILES`, como se muestra en la Tabla 7-6 (donde `file` es el nombre del campo de carga del archivo suministrado por el formulario de envío).

Tabla 7-6. Contenido de la matriz $_FILES

Elemento de la matriz	Contenidos
`$_FILES['file']['name']`	El nombre del archivo cargado (p. ej., *smiley.jpg*)
`$_FILES['file']['type']`	El tipo de contenido del archivo (p. ej., *image/jpeg*)
`$_FILES['file']['size']`	El tamaño del archivo en bytes
`$_FILES['file']['tmp_name']`	El nombre del archivo temporal almacenado en el servidor
`$_FILES['file']['error']`	El código de error resultante de la carga del archivo

A los tipos de contenido se les solía conocer como tipos *MIME* (Multipurpose Internet Mail Extension), pero debido a que más tarde se extendió su uso a todo Internet, ahora se los llama a menudo *Internet media types (tipos de medios de Internet)*. La Tabla 7-7 muestra algunos de los tipos usados con más frecuencia que aparecen en `$_FILES['file']['type']`.

Tabla 7-7. Algunos tipos habituales de Internet media contents (contenido de medios)

application/pdf	image/gif	multipart/form-data	text/xml
application/zip	image/jpeg	text/css	video/mpeg
audio/mpeg	image/png	text/html	video/mp4
audio/x-wav	image/tiff	text/plain	video/quicktime

Validación

Espero que ahora no haga falta decir (aunque lo haré de todos modos) que la validación de datos de formularios es de suma importancia, debido a la posibilidad de que los usuarios intenten piratear tu servidor.

Además de los datos de entrada creados de forma maliciosa, otra cosa que también tienes que hacer es comprobar si se ha recibido realmente un archivo y, en caso afirmativo, si se ha enviado el tipo correcto de datos.

Si tenemos todas estas cosas en cuenta, el contenido del Ejemplo 7-16, *upload2.php*, es una forma más segura de reescritura de *upload.php*.

Ejemplo 7-16. Una versión más segura de upload.php

```
<?php // upload2.php
  echo <<<_END
    <html><head><title>PHP Form Upload</title></head><body>
    <form method='post' action='upload2.php' enctype='multipart/form-data'>
    Select a JPG, GIF, PNG or TIF File:
    <input type='file' name='filename' size='10'>
    <input type='submit' value='Upload'></form>
_END;

  if ($_FILES)
  {
    $name = $_FILES['filename']['name'];

    switch($_FILES['filename']['type'])
    {
      case 'image/jpeg': $ext = 'jpg'; break;
      case 'image/gif':  $ext = 'gif'; break;
      case 'image/png':  $ext = 'png'; break;
      case 'image/tiff': $ext = 'tif'; break;
      default:           $ext = ''; break;
    }
    if ($ext)
    {
      $n = "image.$ext";
      move_uploaded_file($_FILES['filename']['tmp_name'], $n);
```

```
      echo "Uploaded image '$name' as '$n':<br>";
      echo "<img src='$n'>";
    }
    else echo "'$name' is not an accepted image file";
  }
  else echo "No image has been uploaded";

  echo "</body></html>";
?>
```

La sección de código que no es HTML se ha expandido desde la media docena de líneas del Ejemplo 7-15 a más de 20 líneas, y comienza en `if ($_FILES)`.

Al igual que en la versión anterior, esta línea `if` verifica si se ha contabilizado realmente algún dato, pero ahora hay una parte `else` cerca de la parte inferior del programa que envía un mensaje a la pantalla cuando no se ha cargado nada.

Dentro de la sentencia `if`, a la variable `$name` se le asigna el valor del nombre del archivo recuperado del ordenador que lo subió (igual que antes), pero esta vez no confiaremos en el usuario que nos ha enviado datos válidos. En su lugar, una declaración `switch` verifica el tipo de contenido subido frente a los cuatro tipos de imagen que soporta este programa. Si se produce una coincidencia, a la variable `$ext` se le asigna la extensión de archivo de tres letras para ese tipo. Si no existe ninguna coincidencia, el archivo cargado no es uno de los tipos aceptados y la variable `$ext` es una cadena vacía `""`.

La siguiente sección de código verifica entonces la variable `$ext` para ver si contiene una cadena de caracteres y, si es así, crea un nuevo nombre de archivo llamado `$n` con el nombre de base *image* y la extensión almacenada en `$ext`. Esto significa que el programa tiene un control total sobre el nombre del archivo a crear, ya que solo puede ser uno de *image.jpg*, *image.gif*, *image.png* o *image.tif*.

Una vez que tenemos la seguridad de que el programa no se ha visto comprometido, el resto del código PHP es muy parecido al de la versión anterior. Mueve temporalmente la imagen cargada a su nueva ubicación y luego la muestra, a la vez que muestra los nombres de las imágenes antiguas y nuevas.

No te preocupes por tener que borrar el archivo temporal que PHP crea durante el proceso de carga, porque si el archivo no se ha movido o no se ha renombrado, se eliminará automáticamente cuando el programa se cierra.

Después de la declaración `if`, está el correspondiente `else`, que se ejecuta solo si se ha cargado un tipo de imagen que no es compatible (en cuyo caso muestra el correspondiente mensaje de error).

Cuando escribas tus propias rutinas de carga de archivos, te recomiendo encarecidamente que utilices una rutina similar y que elijas nombres y ubicaciones predefinidos para los archivos subidos. De esta forma, aseguras que no tendrá éxito ningún intento de añadir

nombres de ruta y otros datos maliciosos a las variables que utilizas. Si esto significa que más de un usuario podría terminar cargando un archivo con el mismo nombre, podrías prefijar dichos archivos con los nombres de los usuarios o guardarlos en carpetas creadas individualmente para cada usuario.

Pero si debes usar el nombre de archivo suministrado, debes desinfectarlo permitiendo solo caracteres alfanuméricos y el punto; esto lo puedes hacer con el siguiente comando, usando una expresión regular (ver Capítulo 17) para realizar una búsqueda, y reemplazarlo en `$name`:

```
$name = preg_replace("/[^A-Za-z0-9.]/", "", $name);
```

Esto permite solo los caracteres A-Z, a-z, 0-9, y puntos en la cadena de caracteres `$name`, y elimina todo lo demás.

Mejor aún, para tener la seguridad de que tu programa funcionará en todos los sistemas, independientemente de si distinguen entre mayúsculas y minúsculas, probablemente deberías utilizar el siguiente comando, que cambia todos los caracteres en mayúsculas a minúsculas simultáneamente.

```
$name = strtolower(ereg_replace("[^A-Za-z0-9.]", "", $name));
```

A veces puedes encontrar el media type (tipo de medio) de `image/pjpeg`, que indica un JPEG progresivo, pero puedes agregarlo con seguridad a tu código como un alias de `image/jpeg`, así:

```
case 'image/pjpeg':
case 'image/jpeg': $ext = 'jpg'; break;
```

Llamadas al sistema

A veces PHP no tendrá la función que necesitas para realizar una cierta acción, pero el sistema operativo en el que se está ejecutando PHP puede llevarla a cabo. En tales casos, puedes utilizar el sistema `exec` para hacer el trabajo.

Por ejemplo, para ver rápidamente el contenido del directorio de trabajo, puedes utilizar un programa como el del Ejemplo 7-17. Si tienes un sistema Windows, se ejecutará de la misma forma que si usaras el comando `dir` de Windows. En Linux, Unix o macOS, haz un comentario o elimina la primera línea y no hagas comentarios sobre la segunda para usar el comando `ls` del sistema. Es posible que quieras escribir este programa, guárdalo como *exec.php* y llámalo en tu navegador.

Ejemplo 7-17. Ejecución de un comando del sistema

```
<?php // exec.php
  $cmd = "dir";     // Windows
  // $cmd = "ls";   // Linux, Unix & Mac

  exec(escapeshellcmd($cmd), $output, $status);
```

```
  if ($status) echo "Exec command failed";
  else
  {
    echo "<pre>";
    foreach($output as $line) echo htmlspecialchars("$line\n");
    echo "</pre>";
  }
?>
```

Se llama a la función `htmlspecialchars` para convertir cualquier carácter especial devuelto por el sistema en caracteres que HTML pueda entender y mostrar correctamente, ordenando el resultado. En función del sistema que utilices, el resultado de ejecutar este programa será algo parecido a lo siguiente (desde un comando `dir` de Windows):

```
Volume in drive C is Hard Disk
 Volume Serial Number is DC63-0E29

 Directory of C:\Program Files (x86)\Ampps\www

11/04/2018    11:58    <DIR>   .
11/04/2018    11:58    <DIR>   ..
28/01/2018    16:45    <DIR>   5th_edition_Examples
08/01/2018    10:34    <DIR>   cgi-bin
08/01/2018    10:34    <DIR>   error
29/01/2018    16:18             1,150 favicon.ico
           1 File(s)          1,150 bytes
           5 Dir(s) 1,611,387,486,208 bytes free
```

`exec` acepta tres argumentos:

- El propio comando (en el caso anterior, `$cmd`).
- Una matriz en la que el sistema colocará la salida del comando (en el caso anterior, `$output`).
- Una variable para contener el estado devuelto de la llamada (que, en el caso anterior es `$status`).

Si lo deseas, puedes omitir los parámetros `$output` y `$status`, pero no podrás saber la salida creada por la llamada o incluso si se completó con éxito.

También debes tener en cuenta el uso de la función `escapeshellcmd`. Es un buen hábito usarla siempre que se emite una llamada `exec`, porque desinfecta la cadena de comandos e impide la ejecución de comandos arbitrarios, si proporcionas entradas de usuario a la llamada.

Las funciones de llamada del sistema suelen estar deshabilitadas en los servidores web compartidos, ya que suponen un riesgo para la seguridad. Siempre debes intentar resolver tus problemas dentro de PHP, si puedes, e ir directamente al sistema solo si es realmente necesario. Además, ir al sistema es relativamente lento y necesitas codificar dos implementaciones si se espera que tu aplicación se ejecute en sistemas Windows y Linux/Unix.

¿XHTML o HTML5?

Debido a que los documentos XHTML necesitan estar bien elaborados, puedes analizarlos con analizadores XML estándar, a diferencia de HTML, que requiere un analizador sintáctico específico de HTML (que, afortunadamente, son los navegadores web más populares). Por esta razón, XHTML nunca llegó a ser realmente popular y cuando llegó el momento de idear un nuevo estándar la World Wide Web Consortium optó por apoyar HTML5 en lugar del nuevo estándar XHTML2.

HTML5 tiene algunas de las características de HTML4 y XHTML, pero es mucho más sencillo de usar y menos estricto para validar, y, afortunadamente, ahora hay un solo tipo de documento que necesitas colocar en el encabezamiento de un documento HTML5 (en lugar de la variedad de tipos estrictos, de transición y de conjuntos de marcos que se requerían anteriormente):

```
<!DOCTYPE html>
```

Solo la simple palabra `html` es suficiente para decirle al navegador que tu página web está diseñada para HTML5 y, debido a que las últimas versiones de los navegadores más populares han sido compatibles con la mayoría de la especificación HTML5 desde 2011 más o menos, este tipo de documento es generalmente lo único que necesitas, a menos que elijas atender a los navegadores más antiguos.

A todos los efectos, al escribir documentos HTML, los desarrolladores web pueden ignorar con toda seguridad los tipos de documentos XHTML antiguos y la sintaxis (como el uso de `<br />` en lugar de la etiqueta `<br>` más sencilla). Pero si te encuentras que tienes que atender a navegadores muy antiguos o a una aplicación inusual que se basa en XHTML, puedes obtener más información sobre cómo hacerlo en *http://xhtml.com*.

Preguntas

1. ¿Qué especificador de conversión `printf` utilizarías para visualizar un número en punto flotante?
2. ¿Qué declaración `printf` se podría utilizar para tomar la cadena de entrada `"Happy Birthday"` y obtener la cadena `"**Happy"`?
3. Para enviar la salida de `printf` a una variable en lugar de a un navegador, ¿qué función alternativa usarías?

4. ¿Cómo crearías una marca de tiempo Unix para las 7:11 a. m. del 2 de mayo de 2016 (7:11 a. m. on May 2, 2016)?
5. ¿Qué modo de acceso a archivos usarías con `fopen` para abrir un archivo en escritura y lectura con el archivo truncado y el puntero del archivo situado al principio?
6. ¿Cuál es el comando PHP para eliminar el archivo *file.txt*?
7. ¿Qué función de PHP se utiliza para leer un archivo entero de una sola vez, incluso a través de la web?
8. ¿Qué variable superglobal de PHP contiene los detalles de los archivos subidos?
9. ¿Qué función de PHP permite la ejecución de comandos del sistema?
10. ¿Cuál de los siguientes estilos de etiquetas se prefiere en HTML5: `<hr>` o `<hr />`?

Consulta "Respuestas del Capítulo 7" en la página 710 en el Apéndice A para comprobar las respuestas a estas preguntas.