PHP AVANZADO

Capitulo 1: ¿Qué es PHP?

PHP (acrónimo de "PHP: Hypertext Preprocessor") es un lenguaje de "código abierto" interpretado, de alto nivel, embebido en páginas HTML y ejecutado en el servidor.
Una respuesta corta y concisa, pero, ¿qué significa realmente? Un ejemplo nos aclarará las cosas:
Ejemplo 1-1. Un ejemplo introductorio

```
<html>
  <head>
    <title>Ejemplo</title>
  </head>
  <body>

    <?php
    echo "Hola, &iexcl;soy un script PHP!";
    ?>

  </body>
</html>
```

Puede apreciarse que no es lo mismo que un script escrito en otro lenguaje de programación como Perl o C -- En vez de escribir un programa con muchos comandos para crear una salida en HTML, escribimos el código HTML con cierto código PHP embebido (incluido) en el mismo, que producirá cierta salida (en nuestro ejemplo, producirá un texto). El código PHP se incluye entre etiquetas especiales de comienzo y final que nos permitirán entrar y salir del modo PHP.
Lo que distingue a PHP de la tecnología Javascript, la cual se ejecuta en la máquina cliente, es que el código PHP es ejecutado en el servidor. Si tuviésemos un script similar al de nuestro ejemplo en nuestro servidor, el cliente solamente recibiría el resultado de su ejecución en el servidor, sin ninguna posibilidad de determinar qué código ha producido el resultado recibido. El servidor web puede ser incluso configurado para que procese todos los archivos HTML con PHP.
Lo mejor de usar PHP es que es extremadamente simple para el principiante, pero a su vez, ofrece muchas características avanzadas para los programadores profesionales. No sienta miedo de leer la larga lista de características de PHP, en poco tiempo podrá empezar a escribir sus primeros scripts.
Aunque el desarrollo de PHP está concentrado en la programación de scripts en el lado del servidor, se puede utilizar para muchas otras cosas.

Capítulo 2. Una explicación sencilla

A continuación, le introduciremos a PHP en un pequeño y sencillo manual. Este documento explica cómo crear páginas web dinámicas para Internet con PHP, aunque PHP no solamente está diseñado para la creación de éstas. Consulte la sección titulada ¿Qué se puede hacer con PHP? para más información.
Las páginas web que utilizan PHP son tratadas como páginas de HTML comunes y corrientes, y puede crearlas y editarlas de la misma manera que lo hace con documentos normales de HTML.
¿Qué necesito?
En este manual vamos a asumir que usted cuenta con un servidor que soporta PHP y que todos los archivos con la extensión .php son manejados por PHP. En la mayoría de servidores, ésta es la extensión que toman los archivos PHP por defecto, pero pregunte al administrador de su servidor para estar seguro. Si su servidor soporta PHP, entonces no necesita hacer nada, solamente crear sus archivos .php y guardarlos en su directorio web, y el servidor, como por arte de magia, los analizará para usted. No hay necesidad de compilar nada, tampoco tiene necesidad de instalar otras herramientas. Mírelo de esta manera, estos archivos de PHP son tan simples como archivos de HTML con una nueva familia de etiquetas que le permiten una gran cantidad de cosas. La mayoría de las compañías de hospedaje de páginas web ofrecen el soporte que necesita para usar PHP, pero si por alguna razón ellos no lo hacen, considere leer la sección titulada Recursos PHP para mas información acerca de compañías de hospedaje que soportan PHP
Digamos que usted tiene limitado acceso a internet y se encuentra programando localmente. En este caso, querrá instalar un servidor de web como Apache, y PHP. Lo más seguro es que también quiera instalar una base de datos como MySQL. Puede instalar estos productos individualmente o simplemente localizar un paquete pre-configurado que automáticamente instale todos estos productos con solamente unos movimientos de su ratón. Es muy fácil instalar un servidor web con soporte para PHP en cualquier sistemas operativo, incluyendo Linux y Windows. En Linux, rpmfind y PBone le ayudarán a encontrar un RPM.

Su primera página con PHP
Comienze por crear un archivo llamado hola.php y colocarle en el "directorio raíz" (DOCUMENT_ROOT) con el siguiente contenido:
Ejemplo 2-1. Nuestro primer script PHP: hola.php

```
<html>
 <head>
  <title>Ejemplo de PHP</title>
 </head>
 <body>
 <?php echo "<p>Hola Mundo</p>"; ?>
</body>
</html>
```

Utilice su navegador web para acceder al archivo, con la URL terminando en "/hola.php". Si está programando localmente este URL lucirá algo como http://localhost/hola.php o http://127.0.0.1/hola.php pero esto depende de la configuración de su servidor web. Aunque este tema está fuera del alcance de este tutorial, también puede ver las directivas DocumentRoot y ServerName en la

configuración de su servidor (en Apache, esto es httpd.conf). Si todo está configurado correctamente, el archivo será analizado por PHP y el siguiente contenido aparecerá en su navegador:

```
<html>
 <head>
  <title>Ejemplo de PHP</title>
 </head>
 <body>
 <p>Hola Mundo</p>
 </body>
</html>
```

Note que esto no es como los scripts de CGI. El archivo no necesita ninguna clase especial de permisos para ser ejecutado. Piense en ellos como si fueran archivos HTML con un conjunto muy especial de etiquetas disponibles, y que hacen muchas cosas interesantes.
Este programa es extremadamente simple, y no necesita usar PHP para crear una página como ésta. Todo lo que hace es mostrar: Hola Mundo usando la sentencia echo().
Si ha intentado usar este ejemplo, y no produjo ningún resultado, preguntando si deseaba descargar el archivo, o mostró todo el archivo como texto, lo más seguro es que PHP no se encuentra habilitado en su servidor. Pídale a su administrador que active esta función por usted, o use el capítulo titulado Instalación en el manual. Si está trabajando localmente, lea también el capítulo dedicado a la instalación, y asegúrese de que todo esté configurado apropiadamente. Si el problema continúa, por favor use una de las muchas opciones para obtener ayuda con PHP.
El objetivo de este ejemplo es demostrar cómo puede usar las etiquetas PHP. En este ejemplo usamos <?php para indicar el inicio de la etiqueta PHP. Después indicamos la sentencia y abandonamos el modo PHP usando ?>. Puede salir de PHP y regresar cuantas veces lo desee usando este método. Para más información, puede leer la sección en el manual titulada Sintaxis básica de PHP.
Una nota acerca de editores de texto: Hay muchos editores de texto y Entornos Integrados de Desarrollo (IDE por sus siglas en Inglés) que puede usar para crear, editar, y organizar archivos PHP. Puede encontrar una lista parcial de éstos en Lista de editores de PHP. Si desea recomendar un editor, por favor visite la página mencionada anteriormente, y comunique su recomendación a las personas encargadas del mantenimiento para que lo incluyan en la lista. Contar con un editor que resalte la sintaxis de PHP puede ser de mucha ayuda.
Una nota acerca de los procesadores de palabras: Los procesadores de palabras como "StarOffice", "Microsoft word" y "Abiword" no son buenas opciones para editar archivos de PHP. Si desea usar uno de éstos programas para probar sus scripts, primero debe asegurarse de guardar el documento en formato de "Texto" puro, o PHP no será capaz de ejecutar el script.
Una nota acerca del "Bloc de Notas de Windows": Si desea escribir sus archivos PHP usando el "Bloc de Notas de Windows" o en algún otro editor de texto para Windows necesita asegurarse de que sus archivos sean guardados con la extensión .php (la mayoría de editores de texto en Windows automáticamente tratarán de

añadir la extensión .txt a los archivos a menos que tome los siguientes pasos para prevenirlo). Cuando guarde sus archivos y el programa le pregunte qué nombre le desea dar al archivo, use comillas para indicar el nombre (es decir, "hola.php"). Una alternativa es, en la lista de opciones "Archivos de Texto *.txt", seleccionar la opción "Todos los archivos *.*". Aquí puede escribir el nombre del archivo sin las comillas.
Ahora que ha creado un pequeño script de PHP que funciona correctamente, es hora de trabajar con el script de PHP más famoso; vamos a hacer una llamada a la función phpinfo() para obtener información acerca de su sistema y configuración como las variables predefinidas disponibles, los módulos utilizados por PHP, y las diferentes opciones de configuración. Tomemos unos segundos para revisar esta información.
Algo útil
Hagamos ahora algo que puede ser más útil. Vamos a chequear qué clase de navegador web utiliza. Para hacerlo, vamos a consultar la información que el navegador nos envía como parte de su petición HTTP. Esta información es guardada en una variable. Las variables siempre comienzan con un signo de dólar ("$") en PHP. La variable que vamos a utilizar en esta situación es $_SERVER["HTTP_USER_AGENT"].
Nota: $_SERVER es una variable reservada por PHP que contiene toda la información del servidor web. Es conocida como Autoglobal (o Superglobal). Consulte el manual en su sección titulada Autoglobales para mas información. Éstas son variables especiales que fueron introducidas en la versión 4.1.0 de PHP. Antes podíamos usar las matrices $HTTP_*_VARS, tales como $HTTP_SERVER_VARS. Aunque éstas han sido marcadas como obsoletas, tales matrices todavía existen. (También puede echar un vistazo a las notas relacionadas con el código antiguo.)
Para poder ver esta variable solo necesita:
Ejemplo 2-2. Impresión de una variable (elemento de la matriz)
```
<?php echo $_SERVER["HTTP_USER_AGENT"]; ?>
```
Un ejemplo de los resultados de este programa sería:
Mozilla/4.0 (compatible; MSIE 5.01; Windows NT 5.0)
Hay muchos tipos de variables en PHP. En el ejemplo anterior imprimimos una matriz. Las matrices pueden ser muy útiles.
$_SERVER es simplemente una variable que se encuentra disponible automáticamente para usted en PHP. Puede encontrar una lista en la sección titulada Variables Reservadas del manual, o puede generar una lista completa creando un archivo como el presentado a continuación:
Ejemplo 2-3. Consultar todas las variables predefinidas con phpinfo()
```
<?php phpinfo(); ?>
```
Si abre este archivo con su navegador, puede ver una página con información acerca de PHP, junto a una lista de todas las variables que puede usar.
Puede usar más de un declaración PHP dentro de una etiqueta PHP, y crear pequeños segmentos de código que pueden hacer más que un "echo". Por ejemplo, si quisiéramos detectar el uso de "Internet Explorer", haríamos algo así:
Ejemplo 2-4. Ejemplos de uso de estructuras de control y funciones
```
<?php
if (strstr($_SERVER["HTTP_USER_AGENT"], "MSIE")) {
   echo "Est&aacute; usando Internet Explorer<br />";
```

```
}
?>
```
Un ejemplo de los resultado del script puede ser:
```
Est&aacute; usando Internet Explorer<br />
```
A continuación introduciremos un par de conceptos nuevos. Tenemos una declaración "if". Si está familiarizado con la sintaxis básica del lenguaje "C", esto se vera lógico, Pero si no entiende "C", u otros lenguajes de programación donde encuentra la sintaxis usada anteriormente, probablemente debería conseguir un libro que le introduzca mejor a PHP, y lea los primeros capítulos, o también puede ver la parte del manual titulada Referencia del lenguaje. Puedes encontrar una lista de libros sobre PHP en /books.php.
El segundo concepto que introducimos fue la función llamada strstr(). strstr() es una función integrada de PHP que busca un cadena dentro de otra cadena más larga. En el caso anterior estamos buscando "MSIE" dentro de $_SERVER["HTTP_USER_AGENT"]. Si la cadena fue encontrada, la función devolverá verdadero ("TRUE"), la declaración "if" se evalúa a verdadero ("TRUE") y el código adentro de las llaves {} es ejecutado. De otra manera no resulta ejecutado. Tómese la libertad de crear ejemplos similares usando "if", "else" ("de otra manera"), y otras funciones como strtoupper() y strlen(). Cada página del manual contiene ejemplos que puede usar. Si no está seguro sobre el modo de uso éstas funciones, es recomendable que lea las páginas del manual tituladas Cómo leer una definición de función y la sección relacionada a Funciones en PHP
Podemos continuar y demostrar cómo puede saltar adentro y afuera del modo PHP en el medio de un bloque de código.
Ejemplo 2-5. Mezcla de los modos HTML y PHP
```
<?php
if (strstr($_SERVER["HTTP_USER_AGENT"], "MSIE")) {
?>
<h3>strstr debe haber devuelto verdadero</h3>
<center><b>Est&aacute; usando Internet Explorer</b></center>
<?php
} else {
?>
<h3>strstr debi&oacute; devolver falso</h3>
<center><b>No est&aacute; usando Internet Explorer</b></center>
<?php
}
?>
```
Un ejemplo de los resultados de este script puede ser:
```
<h3>strstr debe haber devuelto verdadero </h3>
<center><b>Est&aacute; usando Internet Explorer</b></center>
```
En vez de usar una sentencia PHP "echo" para demostrar algo, saltamos fuera del código PHP y escribimos HTML puro. Este es un punto muy importante y potente que debemos observar aquí, y es que la fluidez lógica del script está intacta. Sólamente las partes donde hay HTML serán enviadas a su navegador dependiendo de los resultados que strstr() devuelve (si fue verdadero [TRUE], o falso [FALSE]). En otras palabras, si la cadena MSIE fue encontrada o no.
Uso de Formularios HTML
Otra de las características de PHP es que gestiona formularios de HTML. El

concepto básico que es importante entender es que cualquier elemento de los formularios estará disponible automáticamente en su código PHP. Por favor refiérase a la sección titulada Variables fuera de PHP en el manual para más información y ejemplos sobre cómo usar formularios HTML con PHP. Observemos un ejemplo:
Ejemplo 2-6. Un formulario HTML sencillo

```
<form action="accion.php" method="POST">
 Su nombre: <input type="text" name="nombre" />
 Su edad: <input type="text" name="edad" />
 <input type="submit">
</form>
```

No hay nada especial en este formularo, es HTML limpio sin ninguna clase de etiquetas desconocidas. Cuando el cliente llena éste formulario y oprime el botón etiquetado "Submit", una página titulada accion.php es llamada. En este archivo encontrará algo así:
Ejemplo 2-7. Procesamiento de información de nuestro formulario HTML

```
Hola <?php echo $_POST["nombre"]; ?>.
Tiene <?php echo $_POST["edad"]; ?> a&ntilde;os
```

Un ejemplo del resultado de este script podría ser:

```
Hola José.
Tiene 22 años
```

Es aparentemente obvio lo que hace. No hay mucho más que decir al respecto. Las variables $_POST["nombre"] y $_POST["edad"] son definidas automáticamente por PHP. Hace un momento usamos la variable autoglobal $_SERVER, ahora hemos introducido autoglobal $_POST, que contiene toda la información enviada por el método POST. Fíjese en el atributo method en nuestro formulario; es POST. Si hubiéramos usado GET, entonces nuestra información estaría en la variable autoglobal $_GET. También puede utilizar la autoglobal $_REQUEST si no le importa el origen de la petición. Ésta variable contiene una mezcla de información GET, POST y COOKIE. También puede ver la función import_request_variables().
Use de código antiguo con nuevas versiones de PHP
Ahora que PHP ha crecido y se ha convertido en un lenguaje popular, hay muchos más recursos que contienen código que puede reusar en sus propios programas. Por lo general, las personas que se encargan del desarrollo de PHP tratan de que el lenguaje sea compatible con versiones anteriores, para que los programas escritos con versiones antiguas continúen funcionando cuando instale una nueva versión de PHP. En un mundo perfecto, nunca necesitaría modificar su código para hacerlo funcionar con versiones nuevas del lenguaje; pero, como todos sabemos, este no es un mundo perfecto, y usualmente son necesarios los cambios en su código.
Dos de los cambios mas importantes que afectan el código viejo son:

- La desaparición de las matrices $HTTP_*_VARS (que usualmente son usadas como globales al interior de una función o método). Las siguientes matrices autoglobales fueron introducidas en la versión 4.1.0 de PHP. Estas son: $_GET, $_POST, $_COOKIE, $_SERVER, $_FILES, $_ENV, $_REQUEST, y $_SESSION. Las antiguas $HTTP_*_VARS, como $HTTP_POST_VARS, todavía existen, y han existido desde PHP 3.
- Las variables externas que ya no son registradas automáticamente. En otras palabras, a partir de PHP 4.2.0, la directiva PHP register_globals está

deshabilitada (su valor es off) en php.ini. El método preferido para obtener acceso a éstos valores es por medio de las "matrices autoglobales" mencionados anteriormente. Los scripts, libros y tutoriales antiguos pueden asumir que ésta directiva es definida automáticamente como "on". Si es así, puede usar, por ejemplo, $id desde la URL http://www.example.com/foo.php?id=42. Por otra parte, no importa si el valor de la directiva es "on" u "off", $_GET['id'] está siempre disponible.

¿Y ahora qué?

Con lo que hemos visto, puede entender la mayor parte del manual, y también los ejemplos que están disponibles en los archivos. También puede encontrar otros ejemplos en los diferentes sitios de php.net en la sección de enlaces: /links.php. Para ver una presentación que muestra más acerca de lo que puede hacer PHP, visite los diferentes sitios con material relacionado a las conferencias realizadas: http://conf.php.net/ y http://talks.php.net/.

Capítulo 3: Consideraciones generales de instalación

Antes de instalar PHP, necesitais saber porque quereis utilizarlo. Existen tres campos principales en donde PHP se puede usar, tal y como se describe en la sección Qué se puede hacer con PHP?:

- Scripts en la parte del servidor
- Scripts en linea de comandos
- Aplicaciones gráficas clientes

El primero es el más tradicional y el principal campo de trabajo. Se necesitan tres cosas para que funcione. El analizador PHP (CGI ó módulo), un servidor web y un navegador. Dependiendo de la versión de sistema operativo que utiliceis, probablemente tengais un servidor web (p.ej: Apache en Linux y MacOS X ó IIS en Windows). También se puede alquilar espacio web en una empresa que ofrezca este servicio. De esta manera no se necesita instalar nada, solamente escribir los scripts PHP, subirlos al espacio alquilado y ver el resultado en vuestro navegador. Teneis dos maneras de utilizar PHP, si instalais vosotros el servidor y PHP. Existen módulos directos (también llamados SAPI) para muchos servidores web, como Apache, Microsoft Internet Information Server, Netscape e iPlanet. Muchos otros servidores soportan ISAPI, (p.ej.:OmniHTTPd). Si PHP no soporta un módulo para tu servidor web, siempre se puede usar como binario CGI. Esto significa que el

servidor se configura para usar el ejecutable para linea de comandos de PHP en el procesado de peticiones de ficheros PHP.
Si estais interesados en usar PHP desde la linea de comandos (p.ej.: para generar imagenes offline ó procesar ficheros de textos, etc) necesitais el ejecutable para linea de comandos. Para más información, lea la sección Usando PHP desde la línea de comandos. En este caso no se necesita ni servidor web, ni navegador.
Con PHP tambien se puede escribir aplicaciones gráficas usando la extensión PHP-GTK. Esta es una forma totalmente distinta de utilizar PHP que cuando se utiliza para escribir páginas web, ya que no se genera código HTML sino que se trabaja con ventanas y objetos dentro de las mismas. Para más información sobre PHP-GTK, visitar las páginas dedicadas a esta extensión. PHP-GTK no se incluye en la distribución oficial de PHP.
A partir de ahora, esta sección tratará sobre la configuración de PHP con servidores web, en Unix y Windows, tanto como módulo, como binario CGI.
PHP, el código fuente y distribuciones binarias para Windows se pueden encontrar en /. Recomendamos utilizar un servidor espejo cerca de donde esteis para bajaros la versión de PHP que querais.
Otros problemas
Si todavia teneis el mismo problema, alguien en la lista de correos sobre instalación de PHP, puede ayudaros. Primero, comprobar en los archivos de la lista si vuestro problema ya ha sido contestado. Los archivos se encuentran disponibles en las página de soporte /support.php. Para subscribirse a esta lista de correos mandar un correo vacío a php-install-subscribe@lists.php.net. La dirección de la lista es php-install@lists.php.net.
Si quereis conseguir ayuda en la lista de correo, intentar describir lo más detalladamente posible vuestro problema, los datos sobre vuestro sistema (sistema operativo que utilizais, versión de PHP, servidor web, si usais PHP como binario CGI ó como módulo, safe mode, etc...) y a ser posible código suficiente para poder reproducir vuestro problema.
Informes sobre Bugs
Si creeis que habeis encontrado un bug (error de programación) en PHP, mandarnos un informe. Probablemente los desarrolladores de PHP no lo conozcan y si no informais sobre el mismo no podrá arreglarse. Podeis informar sobre bugs a traves del sistema de seguimiento de bugs en http://bugs.php.net/. No mandar informes a la lista de correos ó en mensajes privados a los desarrolladores. El sistema de seguimiento tambien se puede utilizar para pedir nuevas caracteristicas en versiones futuras.

Capítulo 4: Configuración del comportamiento de PHP

El archivo de configuración

El archivo de configuración (llamado php3.ini en PHP 3, y simplemente php.ini a partir de PHP 4) es leído cuando arranca PHP. Para las versiones de PHP como módulo de servidor esto sólo ocurre una vez al arrancar el servidor web. Para la versión CGI y CLI, esto ocurre en cada llamada.
La localización por defecto de php.ini es definida en tiempo de compilación (Consultar la FAQ), pero puede ser cambiada en la versión CGI y CLI con la opción de la linea de comandos -c, consultar el cápitulo sobre como usar PHP desde la linea de comandos. También se puede definir la variable de entorno PHPRC con un "path" adicional para la búsqueda del archivo php.ini
Si php-SAPI.ini existe es usado en vez de php.ini.
Nota: El servidor web Apache cambia al directorio raiz al arrancar, por ello PHP intentará leer el archivo php.ini en el directorio raiz, si existe.
Las directivas php.ini gestionadas por extensiones están documentadas en cada una de las páginas de las extensiones respectivamente. La lista de directivas de núcleo se encuentra disponible en el apéndice.La mayoria de directivas PHP estan listadas en ini_set() con los respectivos permisos y enlaces a la documentacion.
Para obtener una lista completa de todas las directivas disponibles en su versión de PHP, por favor lea su archivo php.ini, el cual debe estar bien documentado.
Alternativamente, puede encontrar útil la última versión del archivo php.ini desde CVS.

Ejemplo 9-1. Ejemplo php.ini

```
; any text on a line after an unquoted semicolon (;) is ignored
[php] ; section markers (text within square brackets) are also ignored
; Boolean values can be set to either:
;    true, on, yes
; or false, off, no, none
register_globals = off
track_errors = yes

; you can enclose strings in double-quotes
include_path = ".:/usr/local/lib/php"

; backslashes are treated the same as any other character
include_path = ".;c:\php\lib"
```

Como cambiar los valores de la configuración

Ejecución de PHP como un módulo de Apache

Cuando se usa PHP como un módulo de Apache, se pueden cambiar valores de la configuración usando directivas en los archivos de configuración de apache, httpd.conf y .htaccess. Necesitará de los privilegios "AllowOverride Options" o "AllowOverride All" para hacerlo.

Con PHP 4 y PHP 5, hay varias directivas Apache que permiten cambiar la configuración de PHP desde los archivos de configuración de apache. Para obtener una lista de que directivas son del tipo PHP_INI_ALL, PHP_INI_PERDIR, ó PHP_INI_SYSTEM, consultar la lista que se encuentra en la documentación de la función ini_set().

Nota: Con PHP 3, existen directivas que corresponden a cada parámetro de configuración en php3.ini, con el prefijo "php3_".

php_value nombre valor
: Asigna el valor de la directiva especificada. Puede ser usado solamente con directivas del tipo PHP_INI_ALL y PHP_INI_PERDIR. Para borrar un valor previo, asignar none como valor

  Nota: No use php_value para definir valores booleanos. Debería usarse php_flag en su lugar (vea más abajo).

php_flag nombre on|off
: Usado para asignar una directiva de configuración booleana. Puede ser usado solamente con directivas del tipo PHP_INI_ALL y PHP_INI_PERDIR.

php_admin_value nombre valor
: Asigna el valor de la directiva especificada. Esto no puede usarse en archivos .htaccess. Todo tipo de directiva asignada con php_admin_value no puede ser cambiada con .htaccess ó directivas "virtualhost". Para borrar un valor previo, asignar none como valor.

php_admin_flag nombre on|off

Usado para asignar una directiva de configuración booleana. Esto no puede usarse en archivos .htaccess. Todo tipo de directiva asignada con php_admin_flag no puede ser cambiada con .htaccess ó directivas.

Ejemplo 9-2. Ejemplo de configuración de Apache

```
<IfModule mod_php5.c>
  php_value include_path ".:/usr/local/lib/php"
  php_admin_flag safe_mode on
```

```
</IfModule>
<IfModule mod_php4.c>
  php_value include_path ".:/usr/local/lib/php"
  php_admin_flag safe_mode on
</IfModule>
<IfModule mod_php3.c>
  php3_include_path ".:/usr/local/lib/php"
  php3_safe_mode on
        </IfModule>
```

Atención
Las Constantes en PHP no existen fuera de PHP. Por ejemplo, en httpd.conf no se pueden usar constantes PHP tales como E_ALL ó E_NOTICE para definir la directiva error_reporting, ya que no tendrá ningún significado y será evaluada como 0. Usar los valores asociados de "bitmask" en su lugar. Estas constantes pueden ser usadas en php.ini

Modificación de la configuración de PHP usando el registro de Windows
Cuando se usa PHP en Windows, se pueden cambiar los valores de configuración para cada directorio por medio de los registros de Windows. Los valores de configuración se guardan en la llave de registro HKLM\SOFTWARE\PHP\Per Directory Values, en las subllaves correspondientes al PATH. Por ejemplo, los valores de configuracion del directorio c:\inetpub\wwwroot se guardarán en HKLM\SOFTWARE\PHP\Valores Por Directorio\c\inetpub\wwwroot. La configuración de un directorio es válida para todos los scripts ejecutados en el mismo y sus subdirectorios. Los valores en la llave deben de definirse con el nombre de la directiva de configuración de PHP y el valor tipo cadena. Las constantes PHP en los valores no son analizadas.
Otras interfaces con PHP
Independientemente del modo en que ejecute PHP, es posible cambiar ciertos valores en tiempo de ejecución usando ini_set(). Vea la documentación en la página sobre ini_set() para más información.
Si está interesado en una lista completa de parámetros de configuración en su sistema con sus valores actuales, puede ejecutar la función phpinfo(), y revisar la página resultante. También puede acceder a los valores de directivas de configuración individuales en tiempo de ejecución usando ini_get() o get_cfg_var() .

Capítulo 5. Sintaxis básica

Saliendo de HTML
Para interpretar un archivo, php símplemente interpreta el texto del archivo hasta que encuentra uno de los carácteres especiales que delimitan el inicio de código PHP. El intérprete ejecuta entonces todo el código que encuentra, hasta que encuentra una etiqueta de fin de código, que le dice al intérprete que siga ignorando el código siguiente. Este mecanismo permite embeber código PHP dentro de HTML: todo lo que está fuera de las etiquetas PHP se deja tal como está, mientras que el resto se interpreta como código.
Hay cuatro conjuntos de etiquetas que pueden ser usadas para denotar bloques de código PHP. De estas cuatro, sólo 2 (<?php. . .?> y <script language="php">. . .</script>) están siempre disponibles; el resto pueden ser configuradas en el fichero de php.ini para ser o no aceptadas por el intérprete. Mientras que el formato corto de etiquetas (short-form tags) y el estilo ASP (ASP-style tags) pueden ser convenientes, no son portables como la versión de formato largo de etiquetas. Además, si se pretende embeber código PHP en XML o XHTML, será obligatorio el uso del formato <?php. . .?> para la compatibilidad con XML.
Las etiquetas soportadas por PHP son:
Ejemplo 10-1. Formas de escapar de HTML
1. <?php echo("si quieres servir documentos XHTML o XML, haz como aquí\n"); ?>

2. <? echo ("esta es la más simple, una instrucción de procesado SGML \n"); ?>
<?= expression ?> Esto es una abreviatura de "<? echo expression ?>"

3. <script language="php">

```
    echo ("muchos editores (como FrontPage) no
          aceptan instrucciones de procesado");
  </script>

4.  <% echo ("Opcionalmente, puedes usar las etiquetas ASP"); %>
  <%= $variable; # Esto es una abreviatura de "<% echo . . ." %>
```

El método primero, <?php. . .?>, es el más conveniente, ya que permite el uso de PHP en código XML como XHTML.
El método segundo no siempre está disponible. El formato corto de etiquetas está disponible con la función short_tags() (sólo PHP 3), activando el parámetro del fichero de configuración de PHP short_open_tag, o compilando PHP con la opción --enable-short-tags del comando configure. Aunque esté activa por defecto en php.ini-dist, se desaconseja el uso del formato de etiquetas corto.
El método cuarto sólo está disponible si se han activado las etiquetas ASP en el fichero de configuración: asp_tags.
Nota: El soporte de etiquetas ASP se añadió en la versión 3.0.4.
Nota: No se debe usar el formato corto de etiquetas cuando se desarrollen aplicaciones o bibliotecas con intención de redistribuirlas, o cuando se desarrolle para servidores que no están bajo nuestro control, porque puede ser que el formato corto de etiquetas no esté soportado en el servidor. Para generar código portable y redistribuíble, asegúrate de no usar el formato corto de etiquetas.
La etiqueta de fin de bloque incluirá tras ella la siguiente línea si hay alguna presente. Además, la etiqueta de fin de bloque lleva implícito el punto y coma; no necesitas por lo tanto añadir el punto y coma final de la última línea del bloque PHP.
PHP permite estructurar bloques como:
Ejemplo 10-2. Métodos avanzados de escape

```
<?php
if ($expression) {
  ?>
   <strong>This is true.</strong>
  <?php
} else {
  ?>
   <strong>This is false.</strong>
  <?php
}
?>
```

Este ejemplo realiza lo esperado, ya que cuando PHP encuentra las etiquetas ?> de fin de bloque, empieza a escribir lo que encuentra tal cual hasta que encuentra otra etiqueta de inicio de bloque. El ejemplo anterior es, por supuesto, inventado. Para escribir bloques grandes de texto generamente es más eficiente separalos del código PHP que enviar todo el texto mediante las funciones echo(), print() o similares.
Separación de instrucciones
Las separación de instrucciones se hace de la misma manera que en C o Perl - terminando cada declaración con un punto y coma.
La etiqueta de fin de bloque (?>) implica el fin de la declaración, por lo tanto lo siguiente es equivalente:

```
<?php
  echo "This is a test";
?>

<?php echo "This is a test" ?>
```

Comentarios

PHP soporta el estilo de comentarios de 'C', 'C++' y de la interfaz de comandos de Unix. Por ejemplo:

```
<?php
  echo "This is a test"; // This is a one-line c++ style comment
  /* This is a multi line comment
     yet another line of comment */
  echo "This is yet another test";
  echo "One Final Test"; # This is shell-style style comment
?>
```

Los estilos de comentarios de una linea actualmente sólo comentan hasta el final de la linea o el bloque actual de código PHP, lo primero que ocurra.

```
<h1>This is an <?php # echo "simple";?> example.</h1>
<p>The header above will say 'This is an example'.
```

Hay que tener cuidado con no anidar comentarios de estilo 'C', algo que puede ocurrir al comentar bloques largos de código.

```
<?php
 /*
  echo "This is a test"; /* This comment will cause a problem */
 */
?>
```

Los estilos de comentarios de una linea actualmente sólo comentan hasta el final de la linea o del bloque actual de código PHP, lo primero que ocurra. Esto implica que el código HTML tras // ?> seráa impreso: ?> sale del modo PHP, retornando al modo HTML, el comentario // no le influye.

Capítulo 6. Tipos

Introducción
PHP soporta ocho tipos primitivos.
Cuatro tipos escalares:
- boolean
- integer
- float (número de punto-flotante, también conocido como 'double')
- string

Dos tipos compuestos:
- array
- object

Y finalmente dos tipos especiales:
- resource
- NULL

Este manual introduce también algunos pseudo-tipos por razones de legibilidad:
- mixed
- number
- callback

También puede encontrar algunas referencias al tipo "double". Considere al tipo double como el mismo que float, los dos nombres existen solo por razones históricas.
El tipo de una variable usualmente no es declarado por el programador; en cambio, es decidido en tiempo de compilación por PHP dependiendo del contexto en el que es usado la variable.
Nota: Si desea chequear el tipo y valor de una cierta expresión, use var_dump().
Nota: Si tan solo desea una representación legible para humanos del tipo para propósitos de depuración, use gettype(). Para chequear por un cierto tipo, no use

gettype(); en su lugar utilice las funciones is_tipo. Algunos ejemplos:

```
<?php
$bool = TRUE;  // un valor booleano
$str  = "foo";  // una cadena
$int  = 12;    // un entero

echo gettype($bool); // imprime "boolean"
echo gettype($str);  // imprime "string"

// Si este valor es un entere, incrementarlo en cuatro
if (is_int($int)) {
   $int += 4;
}

// Si $bool es una cadena, imprimirla
// (no imprime nada)
if (is_string($bool)) {
   echo "Cadena: $bool";
}
?>
```

Si quisiera forzar la conversión de una variable a cierto tipo, puede moldear la variable, o usar la función settype() sobre ella.

Note que una variable puede ser evaluada con valores diferentes en ciertas situaciones, dependiendo del tipo que posee en cada momento. Para más información, vea la sección sobre Manipulación de Tipos. Asimismo, puede encontrarse interesado en consultar las tablas de comparación de tipos, ya que éstas muestran ejemplos de las varias comparaciones relacionadas con tipos.

Booleanos

Este es el tipo más simple. Un boolean expresa un valor de verdad. Puede ser TRUE or FALSE.

Nota: El tipo booleano fue introducido en PHP 4.

Sintaxis

Para especificar un literal booleano, use alguna de las palabras clave TRUE o FALSE. Ambas son insensibles a mayúsculas y minúsculas.

```
<?php
$foo = True; // asignar el valor TRUE a $foo
?>
```

Usualmente se usa algún tipo de operador que deuelve un valor boolean, y luego éste es pasado a una estructura de control.

```
<?php
// == es un operador que prueba por
// igualdad y devuelve un booleano
if ($accion == "mostrar_version") {
   echo "La versi&oacute;n es 1.23";
}

// esto no es necesario...
if ($mostrar_separadores == TRUE) {
   echo "<hr>\n";
}

// ...porque se puede escribir simplemente
if ($mostrar_separadores) {
   echo "<hr>\n";
}
?>
```

Conversión a booleano

Para convertir explícitamente un valor a boolean, use el moldeamiento (bool) o (boolean). Sin embargo, en la mayoría de casos no es necesario usar el moldeamiento, ya que un valor será convertido automáticamente si un operador, función o estructura de control requiere un argumento tipo boolean.

Vea también Manipulación de Tipos.

Cuando se realizan conversiones a boolean, los siguientes valores son considerados FALSE:

- el boolean FALSE mismo
- el integer 0 (cero)
- el float 0.0 (cero)
- el valor string vacío, y el string "0"
- un array con cero elementos
- un object con cero variables miembro (sólo en PHP 4)
- el tipo especial NULL (incluyendo variables no definidas)

Cualquier otro valor es considerado TRUE (incluyendo cualquier resource).

Aviso
i-1 es considerado TRUE, como cualquier otro número diferente a cero (ya sea negativo o positivo)!

```
<?php
var_dump((bool) "");        // bool(false)
var_dump((bool) 1);        // bool(true)
var_dump((bool) -2);        // bool(true)
var_dump((bool) "foo");    // bool(true)
var_dump((bool) 2.3e5);    // bool(true)
var_dump((bool) array(12)); // bool(true)
var_dump((bool) array());  // bool(false)
var_dump((bool) "false");  // bool(true)
?>
```

Enteros

Un integer es un número del conjunto Z = {..., -2, -1, 0, 1, 2, ...}.
Vea también: Entero de longitud arbitraria / GMP, Números de punto flotante, y Precisión arbitraria / BCMath

Sintaxis

Los enteros pueden ser especificados en notación decimal (base-10), hexadecimal (base-16) u octal (base-8), opcionalmente precedidos por un signo (- o +).
Si usa la notación octal, debe preceder el número con un 0 (cero), para usar la notación hexadecimal, preceda el número con 0x.

Ejemplo 11-1. Literales tipo entero

```
<?php
$a = 1234; // numero decimal
$a = -123; // un numero negativo
$a = 0123; // numero octal (equivalente al 83 decimal)
$a = 0x1A; // numero hexadecimal (equivalente al 26 decimal)
?>
```

Formalmente, la posible estructura para literales enteros es:

```
decimal     : [1-9][0-9]*
            | 0

hexadecimal : 0[xX][0-9a-fA-F]+

octal       : 0[0-7]+

integer     : [+-]?decimal
            | [+-]?hexadecimal
            | [+-]?octal
```

El tamaño de un entero es dependiente de la plataforma, aunque un valor máximo de aproximadamente dos billones es el valor usual (lo que es un valor de 32 bits con signo). PHP no soporta enteros sin signo.

Aviso
Si un dígito inválido es pasado a un entero octal (p.ej. 8 o 9), el resto del número es ignorado. Ejemplo 11-2. Curiosidad de valores octales <?php var_dump(01090); // 010 octal = 8 decimal ?>

Desbordamiento de enteros

Si especifica un número más allá de los límites del tipo integer, será interpretado en su lugar como un float. Asimismo, si realiza una operación que resulta en un número más allá de los límites del tipo integer, un float es retornado en su lugar.

```php
<?php
$numero_grande =  2147483647;
var_dump($numero_grande);
// salida: int(2147483647)

$numero_grande =  2147483648;
var_dump($numero_grande);
// salida: float(2147483648)

// esto no ocurre con los enteros indicados como hexadecimales:
var_dump( 0x100000000 );
// salida: int(2147483647)

$millon = 1000000;
$numero_grande =  50000 * $millon;
var_dump($numero_grande);
// salida: float(50000000000)
?>
```

Aviso
Desafortunadamente, había un fallo en PHP que provocaba que esto no siempre funcionara correctamente cuando se presentaban números negativos. Por ejemplo: cuando hace -50000 * $millon, el resultado será -429496728. Sin embargo, cuando ambos operandos son positivos no se presenta ningún problema. Este problema fue resuelto en PHP 4.1.0.

No hay un operador de división de enteros en PHP. 1/2 produce el float 0.5. Puede moldear el valor a un entero para asegurarse de redondearlo hacia abajo, o puede usar la función round().

```php
<?php
var_dump(25/7);        // float(3.5714285714286)
var_dump((int) (25/7)); // int(3)
var_dump(round(25/7));  // float(4)
?>
```

Conversión a entero

Para convertir explícitamente un valor a integer, use alguno de los moldeamientos (int) o (integer). Sin embargo, en la mayoría de casos no necesita usar el moldeamiento, ya que un valor será convertido automáticamente si un operador, función o estructura de control requiere un argumento tipo integer. También puede convertir un valor a entero con la función intval().

Vea también Manipulación de Tipos.

Desde booleans

FALSE producirá 0 (cero), y TRUE producirá 1 (uno).

Desde números de punto flotante

Cuando se realizan conversiones desde un flotante a un entero, el número será redondeado hacia cero.

Si el flotante se encuentra más allá de los límites del entero (usualmente +/- 2.15e+9 = 2^31), el resultado es indefinido, ya que el flotante no tiene suficiente precisión para dar un resultado entero exacto. No se producirá una advertencia, ini siquiera una noticia en este caso!

Aviso

Nunca moldee una fracción desconocida a integer, ya que esto en ocasiones produce resultados inesperados.
<?php
echo (int) ((0.1+0.7) * 10); // imprime 7!
?>
 	
Para más información, consulte la advertencia sobre precisión-flotante
.

Desde cadenas

Desde otros tipos

Atención
El comportamiento de convertir desde entero no es definido para otros tipos. Actualmente, el comportamiento es el mismo que si el valor fuera antes convertido a booleano. Sin embargo, no confíe en este comportamiente, ya que puede ser modificado sin aviso.

Números de punto flotante
Los números de punto flotante (también conocidos como "flotantes", "dobles" o "números reales") pueden ser especificados usando cualquiera de las siguientes sintaxis:

```
<?php
$a = 1.234;
$b = 1.2e3;
$c = 7E-10;
?>
```

Formalmente:

```
LNUM         [0-9]+
DNUM         ([0-9]*[\.]{LNUM}) | ({LNUM}[\.][0-9]*)
EXPONENT_DNUM ( ({LNUM} | {DNUM}) [eE][+-]? {LNUM})
```

El tamaño de un flotante depende de la plataforma, aunque un valor común consiste en un máximo de ~1.8e308 con una precisión de aproximadamente 14 dígitos decimales (lo que es un valor de 64 bits en formato IEEE).

Precisión del punto flotante

Es bastante común que algunas fracciones decimales simples como 0.1 o 0.7 no puedan ser convertidas a su representación binaria interna sin perder un poco de precisión. Esto puede llevar a resultados confusos: por ejemplo, floor((0.1+0.7)*10) usualmente devolverá 7 en lugar del esperado 8 ya que el resultado de la representación interna es en realidad algo como 7.9999999999....
Esto se encuentra relacionado al hecho de que es imposible expresar de forma exacta algunas fracciones en notación decimal con un número finito de dígitos. Por ejemplo, 1/3 en forma decimal se convierte en 0.3333333. . ..
Así que nunca confíe en resultados de números flotantes hasta el último dígito, y nunca compare números de punto flotante para conocer si son equivalentes. Si realmente necesita una mejor precisión, es buena idea que use las funciones matemáticas de precisión arbitraria o las funciones gmp en su lugar.

Conversión a flotante

Para más información sobre cuándo y cómo son convertidas las cadenas a flotantes, vea la sección titulada Conversión de cadenas a números. Para valores de otros tipos, la conversión es la misma que si el valor hubiese sido convertido a entero y luego a flotante. Vea la sección Conversión a entero para más información. A partir de PHP 5, una noticia es generada si intenta convertir un objeto a flotante.

Cadenas

Un valor string es una serie de caracteres. En PHP, un caracter es lo mismo que un byte, es decir, hay exactamente 256 tipos de caracteres diferentes. Esto implica también que PHP no tiene soporte nativo de Unicode. Vea utf8_encode() y utf8_decode() para conocer sobre el soporte Unicode.
Nota: El que una cadena se haga muy grande no es un problema. PHP no impone límite práctico alguno sobre el tamaño de las cadenas, así que no hay ninguna razón para preocuparse sobre las cadenas largas.

Sintaxis

Un literal de cadena puede especificarse en tres formas diferentes.

- comillas simples
- comillas dobles
- sintaxis heredoc

Comillas simples

La forma más simple de especificar una cadena sencilla es rodearla de comillas simples (el caracter ').
Para especificar una comilla sencilla literal, necesita escaparla con una barra invertida (\), como en muchos otros lenguajes. Si una barra invertida necesita aparecer antes de una comilla sencilla o al final de la cadena, necesitará doblarla. Note que si intenta escapar cualquier otro caracter, ¡la barra invertida será impresa también! De modo que, por lo general, no hay necesidad de escapar la barra invertida misma.
Nota: En PHP 3, se generará una advertencia de nivel E_NOTICE cuando esto ocurra.
Nota: A diferencia de las otras dos sintaxis, las variables y secuencias de escape para caracteres especiales no serán expandidas cuando ocurren al interior de cadenas entre comillas sencillas.

```
<?php
echo 'esta es una cadena simple';

echo 'Tambi&eacute;n puede tener saltos de l&iacute;nea embebidos
en las cadenas de esta forma, ya que
es v&aacute;lido';

// Imprime: Arnold dijo una vez: "I'll be back"
echo 'Arnold dijo una vez: "I\'ll be back"';

// Imprime: Ha eliminado C:\*.*?
echo 'Ha eliminado C:\\*.*?';

// Imprime: Ha eliminado C:\*.*?
echo 'Ha eliminado C:\*.*?';

// Imprime: Esto no va a expandirse: \n una nueva linea
echo 'Esto no va a expandirse: \n una nueva linea';

// Imprime: Las variables no se $expanden $tampoco
echo 'Las variables no se $expanden $tampoco';
?>
```

Comillas dobles

Si la cadena se encuentra rodeada de comillas dobles ("), PHP entiende más secuencias de escape para caracteres especiales:

Tabla 11-1. Caracteres escapados

secuencia	significado
\n	alimentación de línea (LF o 0x0A (10) en ASCII)
\r	retorno de carro (CR o 0x0D (13) en ASCII)
\t	tabulación horizontal (HT o 0x09 (9) en ASCII)
\\	barra invertida
\$	signo de dólar
\"	comilla-doble
\[0-7]{1,3}	la secuencia de caracteres que coincide con la expresión regular es un caracter en notación octal
\x[0-9A-Fa-f]{1,2}	la secuencia de caracteres que coincide con la expresión regular es un caracter en notación hexadecimal

Nuevamente, si intenta escapar cualquier otro caracter, ¡la barra invertida será impresa también! Antes de PHP 5.1.1, la barra invertida en \{$var} no venía imprimiéndose.

Pero la característica más importante de las cadenas entre comillas dobles es el hecho de que los nombres de variables serán expandidos. Vea procesamiento de cadenas para más detalles.

Heredoc

Otra forma de delimitar cadenas es mediante el uso de la sintaxis heredoc ("<<<"). Debe indicarse un identificador después de la secuencia <<<, luego la cadena, y luego el mismo identificador para cerrar la cita.

El identificador de cierre debe comenzar en la primera columna de la línea.

Asimismo, el identificador usado debe seguir las mismas reglas que cualquier otra etiqueta en PHP: debe contener solo caracteres alfanuméricos y de subrayado, y debe iniciar con un caracter no-dígito o de subrayado.

Aviso

Es muy importante notar que la línea con el identificador de cierre no contenga otros caracteres, excepto quizás por un punto-y-coma (;). Esto quiere decir en especial que el identificador no debe usar sangría, y no debe haber espacios o tabuladores antes o después del punto-y-coma. Es importante también notar que el primer caracter antes del identificador de cierre debe ser un salto de línea, tal y como lo defina su sistema operativo. Esto quiere decir \r en Macintosh, por ejemplo.
Si esta regla es rota y el identificador de cierre no es "limpio", entonces no se considera un identificador de cierre y PHP continuará en busca de uno. Si, en tal caso, no se encuentra un identificador de cierre apropiado, entonces un error del analizador sintáctico resultará con el número de línea apuntando al final del script.
No es permitido usar la sintaxis heredoc al inicializar miembros de clase. Use otro tipo de sintaxis en su lugar.

Ejemplo 11-3. Ejemplo inválido

```
<?php
class foo {
   public $bar = <<<EOT
bar
EOT;
}
?>
```

El texto heredoc se comporta tal como una cadena entre comillas dobles, sin las comillas dobles. Esto quiere decir que no necesita escapar tales comillas en sus bloques heredoc, pero aun puede usar los códigos de escape listados anteriormente. Las variables son expandidas, aunque debe tenerse el mismo cuidado cuando se expresen variables complejas al interior de un segmento heredoc, al igual que con otras cadenas.

Ejemplo 11-4. Ejemplo de uso de una cadena heredoc

```
<?php
$cadena = <<<FIN
Ejemplo de una cadena
que se extiende por varias l&iacute;neas
usando la sintaxis heredoc.
FIN;

/* Un ejemplo mas complejo, con variables. */
class foo
{
  var $foo;
  var $bar;

  function foo()
  {
     $this->foo = 'Foo';
     $this->bar = array('Bar1', 'Bar2', 'Bar3');
  }
}

$foo = new foo();
$nombre = 'MiNombre';

echo <<<FIN
Mi nombre es "$nombre". Estoy imprimiendo algo de $foo->foo.
Ahora, estoy imprimiendo algo de {$foo->bar[1]}.
Esto deber&iacute;a imprimir una letra 'A' may&uacute;scula: \x41
FIN;
?>
```

Nota: El soporte heredoc fue agregado en PHP 4.

Procesamiento de variables

Cuando una cadena es especificada en comillas dobles o al interior de un bloque heredoc, las variables son interpretadas en su interior.

Existen dos tipos de sintaxis: una simple y una compleja. La sintaxis simple es la más común y conveniente. Esta ofrece una forma de interpretar una variable, un valor array, o una propiedad de un object.

La sintaxis compleja fue introducida en PHP 4, y puede reconocerse por las llaves que rodean la expresión.

Sintaxis simple

Si un signo de dólar ($) es encontrado, el analizador sintáctico tomará ambiciosamente tantos lexemas como le sea posible para formar un nombre de variable válido. Rodee el nombre de la variable de llaves si desea especificar explícitamente el final del nombre.

```
<?php
$cerveza = 'Heineken';
echo "El sabor de varias $cerveza's es excelente"; // funciona, "'" no es
un caracter valido para nombres de variables
echo "Tom&oacute; algunas $cervezas";  // no funciona, 's' es un
caracter valido para nombres de variables
echo "Tom&oacute; algunas ${cerveza}s"; // funciona
echo "Tom&oacute; algunas {$cerveza}s"; // funciona
?>
```

De forma similar, puede hacer que un índice de un array o una propiedad de un object sean interpretados. En el caso de los índices de matrices, el corchete cuadrado de cierre (]) marca el final del índice. Para las propiedades de objetos, se aplican las mismas reglas de las variables simples, aunque con las propiedades de objetos no existe un truco como el que existe con las variables.

```
<?php
// Estos ejemplos son especificos al uso de matrices al interior de
// cadenas. Cuando se encuentre por fuera de una cadena, siempre
rodee
// de comillas las claves tipo cadena de su matriz, y no use
// {llaves} por fuera de cadenas tampoco.

// Mostremos todos los errores
error_reporting(E_ALL);

$frutas = array('fresa' => 'roja', 'banano' => 'amarillo');

// Funciona pero note que esto trabaja de forma diferente por fuera de
// cadenas entre comillas
echo "Un banano es $frutas[banano].";

// Funciona
echo "Un banano es {$frutas['banano']}.";

// Funciona, pero PHP musca una constante llamada banano primero,
como
// se describe mas adelante.
echo "Un banano es {$frutas[banano]}.";

// No funciona, use llaves. Esto resulta en un error de analisis
sintactico.
echo "Un banano es $frutas['banano'].";

// Funciona
echo "Un banano es " . $frutas['banano'] . ".";

// Funciona
echo "Este cuadro tiene $cuadro->ancho metros de ancho.";

// No funciona. Para una solucion, vea la sintaxis compleja.
echo "Este cuadro tiene $cuadro->ancho00 cent&iacute;metros de
ancho.";
?>
```

Para cualquier cosa más sofisticada, debería usarse la sintaxis compleja.

Sintaxis compleja (llaves)

Esta no es llamada compleja porque la sintaxis sea compleja, sino porque es posible incluir expresiones complejas de esta forma.

De hecho, de esta forma puede incluir cualquier valor que sea parte del espacio de nombres al interior de cadenas. Simplemente escriba la expresión en la misma forma que lo haría si se encontrara por fuera de una cadena, y luego la ubica entre { y }. Ya que no es posible escapar '{', esta sintaxis será reconocida únicamente cuando el caracter $ se encuentra inmediatamente después de {. (Use "{\$" para obtener una secuencia literal "{$"). Algunos ejemplos para aclarar el asunto:

```
<?php
// Mostremos todos los errores
error_reporting(E_ALL);

$genial = 'fant&aacute;stico';

// No funciona, imprime: Esto es { fant&aacute;stico}
echo "Esto es { $genial}";

// Funciona, imprime: Esto es fant&aacute;stico
echo "Esto es {$genial}";
echo "Esto es ${genial}";

// Funciona
echo "Este cuadro tiene {$cuadro->ancho}00 cent&iacute;metros de
ancho.";

// Funciona
echo "Esto funciona: {$matriz[4][3]}";

// Esto esta mal por la misma razon por la que $foo[bar] esta mal por
// fuera de una cadena. En otras palabras, aun funciona pero ya que
// PHP busca primero una constante llamada foo, genera un error de
// nivel E_NOTICE (constante indefinida).
echo "Esto esta mal: {$matriz[foo][3]}";

// Funciona. Cuando se usan matrices multi-dimensionales, use siempre
// llaves alrededor de las matrices al interior de cadenas
echo "Esto funciona: {$matriz['foo'][3]}";

// Funciona.
echo "Esto funciona: " . $arr['foo'][3];

echo "Puede incluso escribir {$obj->valores[3]->nombre}";

echo "Este es el valor de la variable llamada $nombre: {${$nombre}}";
?>
```

Acceso a cadenas y modificación por caracter

Los caracteres al interior de una cadena pueden ser consultados y modificados al especificar el desplazamiento, comenzando en cero, del caracter deseado después de la cadena entre llaves.

Nota: Para efectos de compatibilidad con versiones anteriores, aun puede usar corchetes tipo matriz para el mismo propósito. Sin embargo, esta sintaxis es obsoleta a partir de PHP 4.

Ejemplo 11-5. Algunos ejemplos de cadenas

```
<?php
// Obtener el primer caracter de una cadena
$cadena = 'Esta es una prueba.';
$primer = $cadena{0};

// Obtener el tercer caracter de una cadena
$tercer = $cadena{2};

// Obtener el ultimo caracter de una cadena.
$cadena = 'Esta es tambien una prueba.';
$ultimo = $cadena{strlen($cadena)-1};

// Modificar el ultimo caracter de una cadena
$cadena = 'Observe el mar';
$cadena{strlen($cadena)-1} = 'l';

?>
```

Funciones y operadores útiles

Las cadenas pueden ser concatenadas usando el operador '.' (punto). Note que el operador '+' (adición) no funciona para este propósito. Por favor refiérase a la sección Operadores de cadena para más información.

Existen bastantes funciones útiles para la modificación de cadenas.

Vea la sección de funciones de cadena para consultar funciones de uso general, o las funciones de expresiones regulares para búsquedas y reemplazos avanzados (en dos sabores: Perl y POSIX extendido).

Existen también funciones para cadenas tipo URL, y funciones para encriptar/descifrar cadenas (mcrypt y mhash).

Finalmente, si aun no ha encontrado lo que busca, vea también las funciones de tipo de caracter.

Conversión a cadena

Es posible convertir un valor a una cadena usando el moldeamiento (string), o la función strval(). La conversión a cadena se realiza automáticamente para usted en el contexto de una expresión cuando se necesita una cadena. Esto ocurre cuando usa las funciones echo() o print(), o cuando compara el valor de una variable con una cadena. El contenido de las secciones del manual sobre Tipos y Manipulación de Tipos ayudan a aclarar este hecho. Vea también settype().

Un valor boolean TRUE es convertido a la cadena "1", el valor FALSE se representa como "" (una cadena vacía). De esta forma, usted puede convertir de ida y vuelta entre valores booleanos y de cadena.

Un número integer o de punto flotante (float) es convertido a una cadena que representa el número con sus dígitos (incluyendo la parte del exponente para los números de punto flotante).

Las matrices son siempre convertidas a la cadena "Array", de modo que no puede volcar los contenidos de un valor array con echo() o print() para ver lo que se encuentra en su interior. Para ver un elemento, usted tendría que hacer algo como echo $arr['foo']. Vea más adelante algunos consejos sobre el volcado/vista del

contenido completo.
Los objetos son convertidos siempre a la cadena "Object". Si quisiera imprimir los valores de variables miembro de un object para efectos de depuración, lea los parágrafos siguientes. Si quiere conocer el nombre de clase del cual un objeto dado es instancia, use get_class(). A partir de PHP 5, el método __toString() es usado si resulta aplicable.
Los recursos son siempre convertidos a cadenas con la estructura "Resource id #1" en donde 1 es el número único del valor resource asignado por PHP durante tiempo de ejecución. Si quisiera obtener el tipo del recurso, use get_resource_type().
NULL se convierte siempre a una cadena vacía.
Como puede apreciar, el imprimir matrices, objetos o recursos no le ofroce información útil sobre los valores mismos. Consulte las funciones print_r() y var_dump() para conocer mejores formas de imprimir valores para depuración.
También puede convertir valores PHP a cadenas y almacenarlas permanentemente. Este método es conocido como seriación, y puede ser efectuado con la función serialize(). También puede seriar valores PHP a estructuras XML, si cuenta con soporte WDDX en su configuración de PHP.
Conversión de cadenas a números
Cuando una cadena es evaluada como un valor numérico, el valor resultante y su tipo son determinados como sigue.
La cadena será evaluada como un float si contiene cualquier caracter entre '.', 'e', o 'E'. De otra forma, evaluará como un entero.
El valor es dado por la porción inicial de la cadena. Si la cadena comienza con datos numéricos válidos, éstos serán el valor usado. De lo contrario, el valor será 0 (cero). Un signo opcional es considerado un dato numérico válido, seguido por uno o más dígitos (que pueden contener un punto decimal), seguidos por un exponente opcional. El exponente es una 'e' o 'E' seguida de uno o más dígitos.

```
<?php
$foo = 1 + "10.5";                // $foo es flotante (11.5)
$foo = 1 + "-1.3e3";              // $foo es flotante (-1299)
$foo = 1 + "bob-1.3e3";           // $foo es entero (1)
$foo = 1 + "bob3";                // $foo es entero (1)
$foo = 1 + "10 Cerditos";         // $foo es entero (11)
$foo = 4 + "10.2 Cerditos";       // $foo es flotante (14.2)
$foo = "10.0 cerdos " + 1;        // $foo es flotante (11)
$foo = "10.0 cerdos " + 1.0;      // $foo es flotante (11)
?>
```

Para más información sobre esta conversión, vea la página del manual Unix sobre strtod(3).
Si quisiera probar cualquiera de los ejemplos presentados en esta sección, puede cortar y pegar los ejemplos e insertar la siguiente línea para verificar por sí mismo lo que está sucediendo:

```
<?php
echo "\$foo==$foo; tipo es " . gettype ($foo) . "<br />\n";
?>
```

No espere obtener el código de un caractar convirtiéndolo a un entero (como lo haría en C, por ejemplo). Use las funciones ord() y chr() para convertir entre códigos de caracter y caracteres.
Matrices

Una matriz en PHP es en realidad un mapa ordenado. Un mapa es un tipo de datos que asocia valores con claves. Este tipo es optimizado en varias formas, de modo que puede usarlo como una matriz real, o una lista (vector), tabla asociativa (caso particular de implementación de un mapa), diccionario, colección, pila, cola y probablemente más. Ya que puede tener otra matriz PHP como valor, es realmente fácil simular árboles.
Una explicación sobre tales estructuras de datos se encuentra por fuera del propósito de este manual, pero encontrará al menos un ejemplo de cada uno de ellos. Para más información, le referimos a literatura externa sobre este amplio tema.
Sintaxis
Especificación con array()
Un array puede ser creado por la construcción de lenguaje array(). Ésta toma un cierto número de parejas clave => valor separadas con coma.

```
array( [clave =>] valor
    , ...
    )
// clave puede ser un integer o string
// valor puede ser cualquier valor
<?php
$matriz = array("foo" => "bar", 12 => true);

echo $matriz["foo"]; // bar
echo $matriz[12];    // 1
?>
```

Una clave puede ser un integer o un string. Si una clave es la representación estándar de un integer, será interpretada como tal (es decir, "8" será interpretado como 8, mientras que "08" será interpretado como "08"). Los valores flotantes en clave serán truncados a valores tipo integer. No existen tipos diferentes para matrices indexadas y asociativas en PHP; sólo existe un tipo de matriz, el cual puede contener índices tipo entero o cadena.
Un valor puede ser de cualquier tipo en PHP.

```
<?php
$matriz = array("unamatriz" => array(6 => 5, 13 => 9, "a" => 42));

echo $matriz["unamatriz"][6];    // 5
echo $matriz["unamatriz"][13];  // 9
echo $matriz["unamatriz"]["a"];  // 42
?>
```

Si no especifica una clave para un valor dado, entonces es usado el máximo de los índices enteros, y la nueva clave será ese valor máximo + 1. Si especifica una clave que ya tiene un valor asignado, ése valor será sobrescrito.

```
<?php
// Esta matriz es la misma que ...
array(5 => 43, 32, 56, "b" => 12);

// ...esta matriz
array(5 => 43, 6 => 32, 7 => 56, "b" => 12);
?>
```

Aviso
A partir de PHP 4.3.0, el comportamiento de generación índices descrito ha cambiado. Ahora, si agrega un elem una matriz cuya clave máxima actual es un valor negati entonces la siguiente clave creada será cero (0). Anteriormente, el nuevo índice hubiera sido establecido clave mayor existente + 1, al igual que con los índices positivos.

Al usar TRUE como clave, el valor será evaluado al integer 1. Al usar FALSE como clave, el valor será evaluado al integer 0. Al usar NULL como clave, el valor será evaluado a una cadena vacía. El uso de una cadena vacía como clave creará (o reemplazará) una clave con la cadena vacía y su valor; no es lo mismo que usar corchetes vacíos.

No es posible usar matrices u objetos como claves. Al hacerlo se producirá una advertencia: Illegal offset type.

Creación/modificación con sintaxis de corchetes cuadrados

Es posible modificar una matriz existente al definir valores explícitamente en ella. Esto es posible al asignar valores a la matriz al mismo tiempo que se especifica la clave entre corchetes. También es posible omitir la clave, agregar una pareja vacía de corchetes ("[]") al nombre de la variable en ese caso.

```
$matriz[clave] = valor;
$matriz[] = valor;
// clave puede ser un integer o string
// valor puede ser cualquier valor
```

Si $matriz no existe aun, ésta será creada. De modo que esta es también una forma alternativa de especificar una matriz. Para modificar un cierto valor, simplemente asigne un nuevo valor a un elemento especificado con su clave. Si desea remover una pareja clave/valor, necesita eliminarla mediante unset().

```
<?php
$matriz = array(5 => 1, 12 => 2);

$matriz[] = 56;    // Esto es igual que $matriz[13] = 56;
                   // en este punto del script

$matriz["x"] = 42; // Esto agrega un nuevo elemento a la
                   // matriz con la clave "x"

unset($matriz[5]); // Esto elimina el elemento de la matriz

unset($matriz);    // Esto elimina la matriz completa
?>
```

Nota: Como se menciona anteriormente, si provee los corchetes sin ninguna clave especificada, entonces se toma el máximo de los índices enteros existentes, y la nueva clave será ese valor máximo + 1. Si no existen índices enteros aun, la clave será 0 (cero). Si especifica una clave que ya tenía un valor asignado, el valor será reemplazado.

Aviso

A partir de PHP 4.3.0, el comportamiento de generación de índices descrito ha cambiado. Ahora, si agrega un elemento al final de una matriz en la que la clave máxima actual es negativa, la siguiente clave creada será cero (0). Anteriormente, el nuevo índice hubiera sido definido como la mayor clave + 1, al igual que ocurre con los índices positivos.

Note que la clave entera máxima usada para este caso no necesita existir actualmente en la matriz. Tan solo debe haber existido en la matriz en algún punto desde que la matriz haya sido re-indexada. El siguiente ejemplo ilustra este caso:

```
<?php
// Crear una matriz simple.
$matriz = array(1, 2, 3, 4, 5);
print_r($matriz);

// Ahora eliminar cada item, pero dejar la matriz misma intacta:
foreach ($matriz as $i => $valor) {
   unset($matriz[$i]);
}
print_r($matriz);

// Agregar un item (note que la nueva clave es 5, en lugar de 0 como
// podria esperarse).
$matriz[] = 6;
print_r($matriz);

// Re-indexar:
$matriz = array_values($matriz);
$matriz[] = 7;
print_r($matriz);
?>
```

El resultado del ejemplo seria:

```
Array
(
   [0] => 1
   [1] => 2
   [2] => 3
   [3] => 4
   [4] => 5
)
Array
(
)
Array
(
   [5] => 6
)
Array
(
   [0] => 6
   [1] => 7
)
```

Funciones útiles

Existe un buen número de funciones útiles para trabajar con matrices. Consulte la sección funciones de matrices.

Nota: La función unset() le permite remover la definición de claves de una matriz. Tenga en cuenta que la matriz NO es re-indexada. Si sólo usa "índices enteros comunes" (comenzando desde cero, incrementando en uno), puede conseguir el efecto de re-indexación usando array_values().

```
<?php
$a = array(1 => 'uno', 2 => 'dos', 3 => 'tres');
unset($a[2]);
/* producira una matriz que hubiera sido definida como
  $a = array(1 => 'uno', 3 => 'tres');
  y NO
  $a = array(1 => 'uno', 2 =>'tres');
*/

$b = array_values($a);
// Ahora $b es array(0 => 'uno', 1 =>'tres')
?>
```

La estructura de control foreach existe específicamente para las matrices. Ésta provee una manera fácil de recorrer una matriz.

Recomendaciones sobre matrices y cosas a evitar

¿Porqué es incorrecto $foo[bar]?

Siempre deben usarse comillas alrededor de un índice de matriz tipo cadena literal. Por ejemplo, use $foo['bar'] y no $foo[bar]. ¿Pero qué está mal en $foo[bar]? Es posible que haya visto la siguiente sintaxis en scripts viejos:

```
<?php
$foo[bar] = 'enemigo';
echo $foo[bar];
// etc
?>
```

Esto está mal, pero funciona. Entonces, ¿porqué está mal? La razón es que este código tiene una constante indefinida (bar) en lugar de una cadena ('bar' - note las comillas), y puede que en el futuro PHP defina constantes que, desafortunadamente para su código, tengan el mismo nombre. Funciona porque PHP automáticamente convierte una cadena pura (una cadena sin comillas que no corresponda con símbolo conocido alguno) en una cadena que contiene la cadena pura. Por ejemplo, si no se ha definido una constante llamada bar, entonces PHP reemplazará su valor por la cadena 'bar' y usará ésta última.

Nota: Esto no quiere decir que siempre haya que usar comillas en la clave. No querrá usar comillas con claves que sean constantes o variables, ya que en tal caso PHP no podrá interpretar sus valores.

```
<?php
error_reporting(E_ALL);
ini_set('display_errors', true);
ini_set('html_errors', false);
// Matriz simple:
$matriz = array(1, 2);
$conteo = count($matriz);
for ($i = 0; $i < $conteo; $i++) {
  echo "\nRevisando $i: \n";
  echo "Mal: " . $matriz['$i'] . "\n";
  echo "Bien: " . $matriz[$i] . "\n";
  echo "Mal: {$matriz['$i']}\n";
  echo "Bien: {$matriz[$i]}\n";
}
?>
```

Nota: El resultado del ejemplo seria:

```
Revisando 0:
Notice: Undefined index:  $i in /path/to/script.html on line 9
Mal:
Bien: 1
Notice: Undefined index:  $i in /path/to/script.html on line 11
Mal:
Bien: 1

Revisando 1:
Notice: Undefined index:  $i in /path/to/script.html on line 9
Mal:
Bien: 2
Notice: Undefined index:  $i in /path/to/script.html on line 11
Mal:
Bien: 2
```

Más ejemplos para demostrar este hecho:

```
<?php
// Mostrar todos los errores
error_reporting(E_ALL);

$matriz = array('fruta' => 'manzana', 'vegetal' => 'zanahoria');

// Correcto
print $matriz['fruta'];  // manzana
print $matriz['vegetal']; // zanahoria

// Incorrecto. Esto funciona pero tambi&eacute;n genera un error de
PHP de
// nivel E_NOTICE ya que no hay definida una constante llamada fruta
//
// Notice: Use of undefined constant fruta - assumed 'fruta' in...
print $matriz[fruta];    // manzana

// Definamos una constante para demostrar lo que pasa. Asignaremos
el
// valor 'vegetal' a una constante llamada fruta.
define('fruta', 'vegetal');

// Note la diferencia ahora
print $matriz['fruta'];  // manzana
print $matriz[fruta];    // zanahoria

// Lo siguiente esta bien ya que se encuentra al interior de una
// cadena. Las constantes no son procesadas al interior de
// cadenas, asi que no se produce un error E_NOTICE aqui
print "Hola $matriz[fruta]";  // Hola manzana

// Con una excepcion, los corchetes que rodean las matrices al
// interior de cadenas permiten el uso de constantes
print "Hola {$matriz[fruta]}";    // Hola zanahoria
print "Hola {$matriz['fruta']}";  // Hola manzana

// Esto no funciona, resulta en un error de interprete como:
// Parse error: parse error, expecting T_STRING' or T_VARIABLE' or
T_NUM_STRING'
// Esto se aplica tambien al uso de autoglobales en cadenas, por
supuesto
print "Hola $matriz['fruta']";
print "Hola $_GET['foo']";

// La concatenacion es otra opcion
print "Hola " . $matriz['fruta']; // Hola manzana
?>
```

Cuando habilita error_reporting() para mostrar errores de nivel E_NOTICE (como por ejemplo definiendo el valor E_ALL) verá estos errores. Por defecto,

error_reporting se encuentra configurado para no mostrarlos.
Tal y como se indica en la sección de sintaxis, debe existir una expresión entre los corchetes cuadrados ('[' y ']'). Eso quiere decir que puede escribir cosas como esta:

```
<?php
echo $matriz[alguna_funcion($bar)];
?>
```

Este es un ejemplo del uso de un valor devuelto por una función como índice de matriz. PHP también conoce las constantes, tal y como ha podido apreciar aquellas E_* antes.

```
<?php
$descripciones_de_error[E_ERROR]  = "Un error fatal ha ocurrido";
$descripciones_de_error[E_WARNING] = "PHP produjo una
advertencia";
$descripciones_de_error[E_NOTICE]  = "Esta es una noticia informal";
?>
```

Note que E_ERROR es también un identificador válido, asi como bar en el primer ejemplo. Pero el último ejemplo es equivalente a escribir:

```
<?php
$descripciones_de_error[1] = "Un error fatal ha ocurrido";
$descripciones_de_error[2] = "PHP produjo una advertencia";
$descripciones_de_error[8] = "Esta es una noticia informal";
?>
```

ya que E_ERROR es igual a 1, etc.
Tal y como lo hemos explicado en los anteriores ejemplos, $foo[bar] aun funciona pero está mal. Funciona, porque debido a su sintaxis, se espera que bar sea una expresión constante. Sin embargo, en este caso no existe una constante con el nombre bar. PHP asume ahora que usted quiso decir bar literalmente, como la cadena "bar", pero que olvidó escribir las comillas.
¿Entonces porqué está mal?
En algún momento en el futuro, el equipo de PHP puede querer usar otra constante o palabra clave, o puede que usted introduzca otra constante en su aplicación, y entonces se ve en problemas. Por ejemplo, en este momento no puede usar las palabras empty y default de esta forma, ya que son palabras clave reservadas especiales.
Nota: Reiterando, al interior de un valor string entre comillas dobles, es válido no rodear los índices de matriz con comillas, así que "$foo[bar]" es válido. Consulte los ejemplos anteriores para más detalles sobre el porqué, asi como la sección sobre procesamiento de variables en cadenas.
Conversión a matriz
Para cualquiera de los tipos: integer, float, string, boolean y resource, si convierte un valor a un array, obtiene una matriz con un elemento (con índice 0), el cual es el valor escalar con el que inició.
Si convierte un object a una matriz, obtiene las propiedades (variables miembro) de ese objeto como los elementos de la matriz. Las claves son los nombres de las variables miembro.
Si convierte un valor NULL a matriz, obtiene una matriz vacía.
Comparación
Es posible comparar matrices con array_diff() y mediante operadores de matriz.

Ejemplos

El tipo matriz en PHP es bastante versátil, así que aquí se presentan algunos ejemplos que demuestran el poder completo de las matrices.

```
<?php
// esto
$a = array( 'color'  => 'rojo',
         'sabor'  => 'dulce',
         'forma'  => 'redonda',
         'nombre' => 'manzana',
                  4       // la clave sera 0
       );

// es completamente equivalente con
$a['color']  = 'rojo';
$a['sabor']  = 'dulce';
$a['forma']  = 'redonda';
$a['nombre'] = 'manzana';
$a[]       = 4;       // la clave sera 0

$b[] = 'a';
$b[] = 'b';
$b[] = 'c';
// resultara en la matriz array(0 => 'a' , 1 => 'b' , 2 => 'c'),
// o simplemente array('a', 'b', 'c')
?>
```

Ejemplo 11-6. Uso de array()

```
<?php
// Array como mapa de propiedades
$mapa = array( 'version'         => 4,
               'SO'              => 'Linux',
               'idioma           => 'ingles',
               'etiquetas_cortas' => true
             );

// claves estrictamente numericas
$matriz = array( 7,
                 8,
                 0,
                 156,
                 -10
               );
// esto es lo mismo que array(0 => 7, 1 => 8,
...)

$cambios = array(       10, // clave = 0
                 5    =>  6,
                 3    =>  7,
                 'a'  =>  4,
                         11, // clave = 6 (el indice
entero maximo era 5)
                 '8'  =>  2, // clave = 8 (entero!)
                 '02' => 77, // clave = '02'
                 0    => 12  // el valor 10 sera
reemplazado por 12
               );

// matriz vacia
$vacio = array();
?>
```

Ejemplo 11-7. Colección

```
<?php
$colores = array('rojo', 'azul', 'verde', 'amarillo');

foreach ($colores as $color) {
  echo "&iquest;Le gusta el $color?\n";
}

?>
```

El resultado del ejemplo seria:

```
&iquest;Le gusta el rojo?
&iquest;Le gusta el azul?
&iquest;Le gusta el verde?
&iquest;Le gusta el amarillo?
```

Modificar los valores de la matriz directamente es posible a partir de PHP 5, pasándolos por referencia. Las versiones anteriores necesitan una solución alternativa:

Ejemplo 11-8. Colección

```
<?php
// PHP 5
foreach ($colores as &$color) {
  $color = strtoupper($color);
}
unset($color); /* se asegura de que escrituras subsiguientes a $color
no modifiquen el ultimo elemento de la matriz */

// Alternativa para versiones anteriores
foreach ($colores as $clave => $color) {
  $colores[$clave] = strtoupper($color);
}

print_r($colores);
?>
```

El resultado del ejemplo seria:

```
Array
(
   [0] => ROJO
   [1] => AZUL
   [2] => VERDE
   [3] => AMARILLO
)
```

Este ejemplo crea una matriz con base uno.

Ejemplo 11-9. Índice con base 1

```
<?php
$primercuarto  = array(1 => 'Enero', 'Febrero', 'Marzo');
print_r($primercuarto);
?>
```

El resultado del ejemplo seria:

```
Array
(
    [1] => 'Enero'
    [2] => 'Febrero'
    [3] => 'Marzo'
)
```

Ejemplo 11-10. Llenado de una matriz

```
<?php
// llenar una matriz con todos los items de un directorio
$gestor = opendir('.');
while (false !== ($archivo = readdir($gestor))) {
   $archivos[] = $archivo;
}
closedir($gestor);
?>
```

Las matrices son ordenadas. Puede también cambiar el orden usando varias funciones de ordenamiento. Vea la sección sobre funciones de matrices para más información. Puede contar el número de items en una matriz usando la función count().

Ejemplo 11-11. Ordenamiento de una matriz

```
<?php
sort($archivos);
print_r($archivos);
?>
```

Dado que el valor de una matriz puede ser cualquier cosa, también puede ser otra matriz. De esta forma es posible crear matrices recursivas y multi-dimensionales.

Ejemplo 11-12. Matrices recursivas y multi-dimensionales

```
<?php
$frutas = array ( "frutas"  => array ( "a" => "naranja",
                                       "b" => "banano",
                                       "c" => "manzana"
                                     ),
                  "numeros" => array ( 1,
                                       2,
                                       3,
                                       4,
                                       5,
                                       6
                                     ),
                  "hoyos"  => array (      "primero",
                                       5 => "segundo",
                                            "tercero"
                                     )
                );

// Algunos ejemplos que hacen referencia a los valores de la matriz
anterior
echo $frutas["hoyos"][5];    // imprime "segundo"
echo $frutas["frutas"]["a"]; // imprime "naranja"
unset($frutas["hoyos"][0]);  // elimina "primero"

// Crear una nueva matriz multi-dimensional
$jugos["manzana"]["verde"] = "bien";
?>
```

Debe advertir que la asignación de matrices siempre involucra la copia de valores. También quiere decir que el apuntador interno de matriz usado por current() y otras funciones similares es reestablecido. Necesita usar el operador de referencia para copiar una matriz por referencia.

```
<?php
$matriz1 = array(2, 3);
$matriz2 = $matriz1;
$matriz2[] = 4; // $matriz2 cambia,
                // $matriz1 sigue siendo array(2, 3)

$matriz3 = &$matriz1;
$matriz3[] = 4; // ahora $matriz1 y $matriz3 son iguales
?>
```

Objetos

Inicialización de Objetos

Para inicializar un objeto, use la sentencia new, lo que instancia el objeto a una variable.

```php
<?php
class foo
{
  function hacer_foo()
  {
     echo "Haciendo foo.";
  }
}

$bar = new foo;
$bar->hacer_foo();
?>
```

Conversión a objeto

Si un objeto es convertido a un objeto, éste no es modificado. Si un valor de cualquier otro tipo es convertido a objeto, una nueva instancia de la clase stdClass es creada. Si el valor era NULL, la nueva instancia será vacía. Las matrices son convertidas a objeto usando las claves de la matriz como nombres de propiedades y con sus valores correspondientes. Para cualquier otro valor, una variable miembro llamada scalar contendrá el valor.

```php
<?php
$obj = (object) 'ciao';
echo $obj->scalar;  // imprime 'ciao'
?>
```

Recurso

Un recurso es una variable especial, que contiene una referencia a un recurso externo. Los recursos son creados y usados por funciones especiales. Vea el apéndice para un listado de todas estas funciones y los tipos de recurso correspondientes.

Nota: El tipo recurso fue introducido en PHP 4

Vea también get_resource_type().

Conversión a un recurso

Dado que los tipos de recurso contienen gestores especiales a archivos abiertos, conexiones con bases de datos, áreas de pintura de imágenes y cosas por el estilo, no es posible convertir cualquier valor a un recurso.

Liberación de recursos

Gracias al sistema de conteo de referencias introducido con el Motor Zend de PHP 4, se detecta automáticamente cuando un recurso ya no es referenciado (tal como en Java). Cuando este es el caso, todos los recursos que fueron usados para éste recurso se liberan por el recolector de basura. Por esta razón, rara vez se necesita liberar la memoria manualmente mediante el uso de alguna función free_result.

Nota: Los enlaces persistentes con bases de datos son especiales, ellos no son destruidos por el recolector de basura. Vea también la sección sobre conexiones persistentes.

NULL

El valor especial NULL representa que una variable no tiene valor. NULL es el único valor posible del tipo NULL.

Nota: El tipo null se introdujo en PHP 4.

Una variable es considerada como NULL si

- se ha asignado la constante NULL a la variable.
- no ha sido definida con valor alguno.
- ha sido eliminada con unset().

Sintaxis

Existe un solo valor de tipo NULL, y ese es la palabra clave NULL, insensible a mayúsculas y minúsculas.

```
<?php
$var = NULL;
?>
```

Vea también is_null() y unset().

Pseudo-tipos usados en esta documentación

mixed

mixed indica que un parámetro puede aceptar múltiples tipos (pero no necesariamente todos).

gettype() por ejemplo aceptará todos los tipos PHP, mientras que str_replace() aceptará cadenas y matrices.

number

number indica que un parámetro puede ser integer o float.

callback

Algunas funciones como call_user_func() o usort() aceptan llamadas de retorno definidas por el usuario como un parámetro. Las funciones tipo llamada de retorno no sólo pueden ser funciones simples, también pueden ser métodos de objetos incluyendo métodos estáticos de clase.

Una función de PHP es simplemente pasada usando su nombre como una cadena. Puede pasar cualquier función incorporada o definida por el usuario con la excepción de array(), echo(), empty(), eval(), exit(), isset(), list(), print() y unset().

Un método de un objeto instanciado es pasado como una matriz que contiene un objeto como el elemento con el índice 0 y un nombre de método como el elemento con índice 1.

Los métodos estáticos de clase pueden ser pasados también sin instanciar un objeto de esa clase al pasar el nombre de clase en lugar de un objeto como el elemento con índice 0.

Ejemplo 11-13. Ejemplos de funciones tipo llamada de retorno

```
<?php

// Una llamada de retorno de ejemplo
function mi_llamada_de_retorno() {
   echo '&iexcl;Hola mundo!';
}

// Un m&eacute;todo como llamada de
retorno de ejemplo
class MiClase {
   function miMetodoDeRetorno() {
      echo '&iexcl;Hola Mundo!';
   }
}

// Tipo 1: Llamada de retorno simple
call_user_func('mi_llamada_de_retorno');

// Tipo 2: Llamada de metodo estatico de
clase
call_user_func(array('MiClase',
'miMetodoDeRetorno'));

// Tipo 3: Llamada a un metodo de objeto
$obj = new MiClase();
call_user_func(array($obj,
'miMetodoDeRetorno'));
?>
```

Manipulación de Tipos

PHP no requiere (o soporta) la definición explícita de tipos en la declaración de variables; el tipo de una variable es determinado por el contexto en el que la variable es usada. Lo que quiere decir que si asigna un valor de cadena a la variable $var, $var se convierte en una cadena. Si luego asigna un valor entero a $var, ésta se convierte en entera.

Un ejemplo de la conversión automática de tipos de PHP es el operador de adición '+'. Si cualquiera de los operandos es un flotante, entonces todos los operandos son evaluados como flotantes, y el resultado será un flotante. De otro modo, los operandos serán interpretados como enteros, y el resultado será también un entero. Note que este NO modifica los tipos de los operandos mismos; el único cambio está en la forma como los operandos son evaluados.

```
<?php
$foo = "0";  // $foo es una cadena (ASCII 48)
$foo += 2;  // $foo es ahora un entero (2)
$foo = $foo + 1.3;  // $foo es ahora un flotante (3.3)
$foo = 5 + "10 Cerditos"; // $foo es entero (15)
$foo = 5 + "10 Cerdos";  // $foo es entero (15)
?>
```

Si los dos últimos ejemplos lucen extraños, consulte Conversión de cadenas a números.

Si desea forzar que una variable sea evaluada como un cierto tipo, consulte la sección sobre Moldeamiento de tipos. Si desea cambiar el tipo de una variable, vea settype().

Si quisiera probar cualquiera de los ejemplos en esta sección, puede usar la función var_dump().

Nota: El comportamiento de una conversión automática a matriz no se encuentra definido en el momento.

```
<?php
$a = "1";    // $a es una cadena
$a[0] = "f";  // Que hay de las posiciones de cadena? Que sucede?
?>
```

Ya que PHP (por razones históricas) soporta el uso de índices en cadenas mediante desplazamientos de posición usando la misma sintaxis que la indexación de matrices, el ejemplo anterior lleva a un problema: ¿debería $a convertirse en una matriz con un primer elemento "f", o debería "f" convertirse en el primer caracter de la cadena $a?

Las versiones recientes de PHP interpretan la segunda asignación como una identificación de desplazamiento de cadena, así que $a se convierte en "f", sin embargo el resultado de esta conversión automática debe considerarse indefinido.

PHP 4 introdujo la nueva sintaxis de llaves para acceder a los caracteres de una cadena, use esta sintaxis en lugar de la que fue presentada anteriormente:

```
<?php
$a    = "abc"; // $a es una cadena
$a{1} = "f";  // $a es ahora "afc"
?>
```

Vea la sección llamada Acceso a cadenas por caracter para más información.

Moldeamiento de Tipos

El moldeamiento de tipos en PHP funciona de forma muy similar a como ocurre en C: el nombre del tipo deseado es escrito entre paréntesis antes de la variable que debe ser moldeada.

```
<?php
$foo = 10;  // $foo es un entero
$bar = (boolean) $foo;  // $bar es un booleano
?>
```

Los moldeamiontos permitidos son:

- (int), (integer) - moldeamiento a entero
- (bool), (boolean) - moldeamiento a booleano
- (float), (double), (real) - moldeamiento a flotante
- (string) - moldeamiento a cadena
- (array) - moldeamiento a matriz

- (object) - moldeamiento a objeto

Note que las tabulaciones y los espacios son permitidos al interior de los paréntesis, así que las siguientes expresiones son funcionalmente equivalentes:

```
<?php
$foo = (int) $bar;
$foo = ( int ) $bar;
?>
```

Nota: En lugar de moldear una variable a cadena, puede también rodear la variable de comillas dobles.

```
<?php
$foo = 10;           // $foo es un entero
$cad = "$foo";       // $cad es una cadena
$fst = (string) $foo; // $fst es tambien una cadena

// Esto imprime "son lo mismo"
if ($fst === $cad) {
   echo "son lo mismo";
}
?>
```

Capítulo 7. Variables

Conceptos Básicos

En PHP las variables se representan como un signo de dólar seguido por el nombre de la variable. El nombre de la variable es sensible a minúsculas y mayúsculas. Los nombres de variables siguen las mismas reglas que otras etiquetas en PHP. Un nombre de variable valido tiene que empezar con una letra o una raya (underscore), seguido de cualquier número de letras, números y rayas. Como expresión regular se podria expresar como:
'[a-zA-Z_\x7f-\xff][a-zA-Z0-9_\x7f-\xff]*'
Nota: En nuestro caso, una letra es a-z, A-Z, y los caracteres ASCII del 127 al 255 (0x7f-0xff).

```
<?php
$var = "Bob";
$Var = "Joe";
echo "$var, $Var";      // outputs "Bob, Joe"

$4site = 'not yet';    // invalid; starts with a number
$_4site = 'not yet';    // valid; starts with an underscore
$täyte = 'mansikka';    // valid; 'ä' is ASCII 228 (Extendido)
?>
```

En PHP3, las variables siempre se asignan por valor. Esto significa que cuando se asigna una expresión a una variable, el valor íntegro de la expresión original se copia en la variable de destino. Esto quiere decir que, por ejemplo, después e asignar el valor de una variable a otra, los cambios que se efectúen a una de esas variables no afectará a la otra. Para más información sobre este tipo de asignación, vea Expresiones.

PHP4 ofrece otra forma de asignar valores a las variables: asignar por referencia. Esto significa que la nueva variable simplemente referencia (en otras palabras, "se convierte en un alias de" ó "apunta a") la variable original. Los cambios a la nueva variable afectan a la original, y viceversa. Esto también significa que no se produce una copia de valores; por tanto, la asignación ocurre más rápidamente. De cualquier forma, cualquier incremento de velocidad se notará sólo en los bucles críticos cuando se asignen grandes matrices u objetos.

Para asignar por referencia, simplemente se antepone un ampersandsigno "&" al comienzo de la variable cuyo valor se está asignando (la variable fuente). Por ejemplo, el siguiente trozo de código produce la salida 'Mi nombre es Bob' dos veces:

```
<?php
$foo = 'Bob';              // Asigna el valor 'Bob' a $foo
$bar = &amp;$foo;              // Referencia $foo v&iacute;a $bar.
$bar = "Mi nombre es $bar";  // Modifica $bar...
echo $foo;                // $foo tambi&eacute;n se modifica.
echo $bar;
?>
```

Algo importante a tener en cuenta es que sólo las variables con nombre pueden ser asignadas por referencia.

```
<?php
$foo = 25;
$bar = &amp;$foo;      // Esta es una asignaci&oacute;n v&aacute;lida.
$bar = &amp;(24 * 7);  // Inv&aacute;lida; referencia una
expresi&oacute;n sin nombre.

function test() {
   return 25;
}

$bar = &amp;test();    // Inv&aacute;lida.
?>
```

Variables predefinidas

PHP proporciona una gran cantidad de variables predefinidas a cualquier script que se ejecute. De todas formas, muchas de esas variables no pueden estar completamente documentadas ya que dependen de sobre qué servidor se esté ejecutando, la versión y configuración de dicho servidor, y otros factores. Algunas de estas variables no estarán disponibles cuando se ejecute PHP desde la línea de comandos.Para obtener una lista de estas variables podeis consultar la sección Variables predefinidas reservadas.

Aviso
A partir de PHP 4.2.0, el valor por defecto de la directiva PHP register_globals es off (desactivada). Este es un cambio importante en PHP. Teniendo register_globals off afecta el conjunto de variables predefinidas disponibles en el sistema. Por ejemplo, para obtener DOCUMENT_ROOT se usará $_SERVER['DOCUMENT_ROOT'] en vez de $DOCUMENT_ROOT ó $_GET['id'] de la URL http://www.example.com/test.php?id=3 en vez de $id ó $_ENV['HOME'] en vez de $HOME. Para más información sobre este cambio, podeis consultar el apartado de configuración sobre register_globals, el capítulo sobre seguridad Usando "Register Globals" , asi como los anuncios de lanzamiento de PHP 4.1.0 y 4.2.0 El uso de las variables reservadas predefinidas en PHP, como matrices superglobales es recomendable.

A partir de PHP 4.1.0, PHP ofrece un conjunto adicional de matrices predefinidas, conteniendo variables del servidor web, el entorno y entradas del usuario. Estas nuevas matrices son un poco especiales porque son automáticamente globales. Por esta razón, son conocidas a menudo como "autoglobales" ó "superglobales". Las superglobales se mencionan más abajo; sin embargo para una lista de sus contenidos y más información sobre variables predefinidas en PHP, consultar la sección Variables predefinidas reservadas. Podreis ver como las variables predefinidas antiguas ($HTTP_*_VARS) todavía existen. A partir de PHP 5.0.0, las matrices de tipo "long" de variables predefinidas, se pueden desactivar con la directiva register_long_arrays.

Variables variables: Las superglobales no pueden usarse como variables variables. Si ciertas variables no son definidas en variables_order, las matrices PHP predefinidas asociadas a estas, estarán vacias.

PHP superglobales

$GLOBALS
: Contiene una referencia a cada variable disponible en el espectro de las variables del script. Las llaves de esta matriz son los nombres de las variables globales. $GLOBALS existe dese PHP 3.

$_SERVER
: Variables definidas por el servidor web ó directamente relacionadas con el entorno en don el script se esta ejecutando. Análoga a la antigua matriz $HTTP_SERVER_VARS (la cual está todavía disponible, aunque no se use).

$_GET
: Variables proporcionadas al script por medio de HTTP GET. Análoga a la antigua matriz $HTTP_GET_VARS (la cual está todavía disponible, aunque no se use).

$_POST

Variables proporcionadas al script por medio de HTTP POST. Análoga a la antigua matriz $HTTP_POST_VARS (la cual está todavía disponible, aunque no se use).

$_COOKIE

Variables proporcionadas al script por medio de HTTP cookies. Análoga a la antigua matriz $HTTP_COOKIE_VARS (la cual está todavía disponible, aunque no se use).

$_FILES

Variables proporcionadas al script por medio de la subida de ficheros via HTTP . Análoga a la antigua matriz $HTTP_POST_FILES (la cual está todavía disponible, aunque no se use). Vea también Subiendo ficheros por método POST para más información.

$_ENV

Variables proporcionadas al script por medio del entorno. Análoga a la antigua matriz $HTTP_ENV_VARS (la cual está todavía disponible, aunque no se use).

$_REQUEST

Variables proporcionadas al script por medio de cuaquier mecanismo de entrada del usuario y por lo tanto no se puede confiar en ellas. La presencia y el orden en que aparecen las variables en esta matriz es definido por la directiva de configuración variables_order. Esta matriz no tiene un análogo en versiones anteriores a PHP 4.1.0. Vea también import_request_variables() .

Nota: Cuando se utiliza la linea de comandos, argv y argc no son incluidas aquí; estas variables se podrán encontrar en la matriz

$_SESSION

Variables registradas en la sesión del script. Análoga a la antigua matriz $HTTP_SESSION_VARS (la cual está todavía disponible, aunque no se use). Vea también la sección Funciones para el manejo de sessiones para más información.

Ambito de las variables

El ámbito de una variable es el contexto dentro del que la variable está definida. La mayor parte de las variables PHP sólo tienen un ámbito simple. Este ámbito simple también abarca los ficheros incluidos y los requeridos. Por ejemplo:

```
<?php
$a = 1;
include "b.inc";
?>
```

Aquí, la variable $a dentro del script incluido b.inc. De todas formas, dentro de las funciones definidas por el usuario aparece un ámbito local a la función. Cualquier variables que se use dentro de una función está, por defecto, limitada al ámbito local de la función. Por ejemplo:

```
<?php
$a = 1; /* global scope */

function Test()
{
  echo $a; /* reference to local scope variable */
}

Test();
?>
```
Este script no producirá salida, ya que la orden echo utiliza una versión local de la variable $a, a la que no se ha asignado ningún valor en su ámbito. Puede que usted note que hay una pequeña diferencia con el lenguaje C, en el que las variables globales están disponibles automáticamente dentro de la función a menos que sean expresamente sobreescritas por una definición local. Esto puede causar algunos problemas, ya que la gente puede cambiar variables globales inadvertidamente. En PHP, las variables globales deben ser declaradas globales dentro de la función si van a ser utilizadas dentro de dicha función. Veamos un ejemplo:
```
<?php
$a = 1;
$b = 2;

function Sum()
{
  global $a, $b;

  $b = $a + $b;
}

Sum();
echo $b;
?>
```
El script anterior producirá la salida "3". Al declarar $a y $b globales dentro de la función, todas las referencias a tales variables se referirán a la versión global. No hay límite al número de variables globales que se pueden manipular dentro de una función.
Un segundo método para acceder a las variables desde un ámbito global es usando la matriz $GLOBALS. El ejemplo anterior se puede reescribir así:

```php
<?php
$a = 1;
$b = 2;

function Sum()
{
  $GLOBALS["b"] = $GLOBALS["a"] + $GLOBALS["b"];
}

Sum();
echo $b;
?>
```

La matriz $GLOBALS es una matriz asociativa con el nombre de la variable global como clave y los contenidos de dicha variable como el valor del elemento de la matriz. $GLOBALS existe en cualquier ámbito, esto pasa porque $GLOBALS es una superglobal. Aqui teneis un ejemplo que demuestra el poder de las superglobales:

```php
<?php
function test_global()
{
  // Most predefined variables aren't "super" and require
  // 'global' to be available to the functions local scope.
  global $HTTP_POST_VARS;

  print $HTTP_POST_VARS['name'];

  // Superglobals are available in any scope and do
  // not require 'global'.  Superglobals are available
  // as of PHP 4.1.0
  print $_POST['name'];
}
?>
```

Otra característica importante del ámbito de las variables es la variable static. Una variable estática existe sólo en el ámbito local de la función, pero no pierde su valor cuando la ejecución del programa abandona este ámbito. Consideremos el siguiente ejemplo:

```php
<?php
function Test ()
{
  $a = 0;
  echo $a;
  $a++;
}
?>
```

Esta función tiene poca utilidad ya que cada vez que es llamada asigna a $a el valor 0 y representa un "0". La sentencia $a++, que incrementa la variable, no sirve para nada, ya que en cuanto la función termina la variable $a desaparece. Para hacer una función útil para contar, que no pierda la pista del valor actual del conteo, la variable $a debe declararse como estática:

```
<?php
function Test()
{
  static $a = 0;
  echo $a;
  $a++;
}
?>
```
Ahora, cada vez que se llame a la función Test(), se representará el valor de $a y se incrementará.
Las variables estáticas también proporcionan una forma de manejar funciones recursivas. Una función recursiva es la que se llama a sí misma. Se debe tener cuidado al escribir una función recursiva, ya que puede ocurrir que se llame a sí misma indefinidamente. Hay que asegurarse de implementar una forma adecuada de terminar la recursión. La siguiente función cuenta recursivamente hasta 10, usando la variable estática $count para saber cuándo parar:
```
<?php
function Test()
{
  static $count = 0;

  $count++;
  echo $count;
  if ($count < 10) {
     Test ();
  }
  $count--;
}
?>
```
En motor Zend 1, utilizado por PHP4, implementa los modificadores static y global para variables en términos de referencias. Por ejemplo, una variable global verdadera importada dentro del ámbito de una función con global, crea una referencia a la variable global. Esto puede ser causa de un comportamiento inesperado, tal y como podemos comprobar en el siguiente ejemplo:

```
<?php
function test_global_ref() {
  global $obj;
  $obj = &new stdclass;
}

function test_global_noref() {
  global $obj;
  $obj = new stdclass;
}

test_global_ref();
var_dump($obj);
test_global_noref();
```

```
var_dump($obj);
?>
```
Al ejecutar este ejemplo obtendremos la siguiente salida:
```
 NULL
object(stdClass)(0) {
}
```
Un comportamiento similar se aplica a static. Referencias no son guardadas estáticamente.
```
<?php
function &get_instance_ref() {
   static $obj;

   echo "Static object: ";
   var_dump($obj);
   if (!isset($obj)) {
       // Assign a reference to the static variable
       $obj = &new stdclass;
   }
   $obj->property++;
   return $obj;
}

function &get_instance_noref() {
   static $obj;

   echo "Static object: ";
   var_dump($obj);
   if (!isset($obj)) {
       // Assign the object to the static variable
       $obj = new stdclass;
   }
   $obj->property++;
   return $obj;
}

$obj1 = get_instance_ref();
$still_obj1 = get_instance_ref();
echo "\n";
$obj2 = get_instance_noref();
$still_obj2 = get_instance_noref();
?>
```
Al ejecutar este ejemplo obtendremos la siguiente salida:
```
Static object: NULL
Static object: NULL

Static object: NULL
Static object: object(stdClass)(1) {
  ["property"]=>
  int(1)
```

```
}
```
Este ejemplo demuestra que al asignar una referencia a una variable estática, esta no es recordada cuando se invoca la funcion &get_instance_ref() por segunda vez.

Variables variables

A veces es conveniente tener nombres de variables variables. Dicho de otro modo, son nombres de variables que se pueden establecer y usar dinámicamente. Una variable normal se establece con una sentencia como:

```
<?php
$a = "hello";
?>
```
Una variable variable toma el valor de una variable y lo trata como el nombre de una variable. En el ejemplo anterior, hello, se puede usar como el nombre de una variable utilizando dos signos de dólar. p.ej.
```
<?php
$$a = "world";
?>
```
En este momento se han definido y almacenado dos variables en el árbol de símbolos de PHP: $a, que contiene "hello", y $hello, que contiene "world". Es más, esta sentencia:
```
<?php
echo "$a ${$a}";
?>
```
produce el mismo resultado que:

```
<?php
echo "$a $hello";
?>
```
Por ej. ambas producen el resultado: hello world.

Para usar variables variables con matrices, hay que resolver un problema de ambigüedad. Si se escribe $$a[1] el intérprete necesita saber si nos referimos a utilizar $a[1] como una variable, o si se pretendía utilizar $$a como variable y el índice [1] como índice de dicha variable. La sintaxis para resolver esta ambigüedad es: ${$a[1]} para el primer caso y ${$a}[1] para el segundo.

Aviso
Tener en cuenta que variables variables no pueden usarse con Matrices superglobales. Esto significa que no se pueden hacer cosas como ${$_GET}. Si buscais un método para manejar la disponibilidad de superglobales y las antiguas HTTP_*_VARS, podeis intentar referiros a ellas

Variables externas a PHP

Formularios HTML (GET y POST)

Cuando se envía un formulario a un script PHP, las variables de dicho formulario pasan a estar automáticamente disponibles en el script gracias a PHP. Por ejemplo, consideremos el siguiente formulario:

Ejemplo 12-1. Variables de formulario simples
```
<form action="foo.php" method="POST">
  Name:  <input type="text" name="username"><br>
  Email: <input type="text" name="email"><br>
```

```
   <input type="submit" name="submit" value="Submit me!">
</form>
```

Dependiendo de tu configuración y preferencias personales, existen muchas maneras de acceder a los datos de tus formularios HTML. Algunos ejemplos:

Ejemplo 12-2. Accediendo datos de un formulario simple HTML POST

```
<?php
// Available since PHP 4.1.0

   print $_POST['username'];
   print $_REQUEST['username'];

   import_request_variables('p', 'p_');
   print $p_username;

// Available since PHP 3.  As of PHP 5.0.0, these long predefined
// variables can be disabled with the register_long_arrays directive.

   print $HTTP_POST_VARS['username'];

// Available if the PHP directive register_globals = on.  As of
// PHP 4.2.0 the default value of register_globals = off.
// Using/relying on this method is not preferred.

   print $username;
?>
```

Usando un formulario GET es similar excepto en el uso de variables predefinidas, que en este caso serán del tipo GET. GET también se usa con QUERY_STRING (la información despues del símbolo '?' en una URL). Por ejemplo http://www.example.com/test.php?id=3 contiene datos GET que son accesibles con $_GET['id'].

Nota: Matrices superglobales, como $_POST y$_GET, están disponibles desde PHP 4.1.0.

Como hemos dicho, antes de PHP 4.2.0, el valor por defecto de register_globals era on (activado). Y, en PHP 3 estba siempre activado. La comunidad PHP anima a no confiar en esta directiva ya que es preferible asumir que tiene el valor off (desactivada) y programar teniendo en cuenta esto.

Nota: La directiva de configuración magic_quotes_gpc afecta a valores Get, Post y Cookie, Si esta activada (on) el valor (It's "PHP!") sera convertido automaticamente a (It\'s \"PHP!\"). "Escaping" es necesario en inserciones a bases de datos.

PHP tambien entiende matrices en el contexto de variables de formularios. (vea la faq relacionada). Se puede, por ejemplo, agrupar juntas variables relacionadas ó usar esta característica para obtener valores de una entrada "select2 múltiple. Por ejemplo, vamos a mandar un formulario asi mismo y a presentar los datos cuando se reciban:

Ejemplo 12-3. Variables de formulario más complejas

```
<?php
if ($HTTP_POST_VARS['action'] == 'submitted') {
   print '<pre>';
```

```
  print_r($HTTP_POST_VARS);
  print '<a href="'. $HTTP_SERVER_VARS['PHP_SELF'] .'">Please try again</a>';

  print '</pre>';
} else {
?>
<form action="<?php echo $HTTP_SERVER_VARS['PHP_SELF']; ?>"
method="POST">
  Name:  <input type="text" name="personal[name]"><br>
  Email: <input type="text" name="personal[email]"><br>
  Beer: <br>
  <select multiple name="beer[]">
     <option value="warthog">Warthog</option>
     <option value="guinness">Guinness</option>
     <option value="stuttgarter">Stuttgarter Schwabenbräu</option>
  </select><br>
  <input type="hidden" name="action" value="submitted">
  <input type="submit" name="submit" value="submit me!">
</form>
<?php
}
?>
```

En PHP 3, el uso de matrices de variables de formularios está limitado a matrices unidimensionales. En PHP 4, no existe esta restricción.

IMAGE SUBMIT variable names

Cuando mandamos un formulario, es posible usar una imagen en vez del botón estandar de "mandar":

```
<input type="image" src="image.gif" name="sub">
```

Cuando el usuario hace click en cualquier parte de la imagen, el formulario que la acompaña se transmitirá al servidor con dos variables adicionales, sub_x y sub_y. Estas contienen las coordenadas del click del usuario dentro de la imagen. Los más experimentados puede notar que los nombres de variable enviados por el navegador contienen un guión en vez de un subrayado (guión bajo), pero PHP convierte el guión en subrayado automáticamente.

Cookies HTTP

PHP soporta cookies de HTTP de forma transparente tal y como están definidas en en las Netscape's Spec. Las cookies son un mecanismo para almacenar datos en el navegador y así rastrear o identificar a usuarios que vuelven. Se pueden crear cookies usando la función SetCookie(). Las cookies son parte de la cabecera HTTP, así que se debe llamar a la función SetCookie antes de que se envíe cualquier salida al navegador. Es la misma restricción que para la función header(). Los datos de una cookie estan disponibles en la matriz con datos de cookies apropiada, tal como $_COOKIE, $HTTP_COOKIE_VARS y también en $_REQUEST. vea la función setcookie() para más detalles y ejemplos.

Si se quieren asignar múltiples valores a una sola cookie, basta con añadir [] al nombre de la cookie para definirla como una matriz. Por ejemplo:

```
<?php
 setcookie("MyCookie[foo]", "Testing 1", time()+3600);
```

```
  setcookie("MyCookie[bar]", "Testing 2", time()+3600);
?>
```
Esto creara dos cookies separadas aunque MyCookie será una matriz simple en el script. Si se quiere definir una sola cookie con valores multiples, considerar primero el uso de la función serialize() ó explode() en el valor.
Nótese que una cookie reemplazará a una cookie anterior que tuviese el mismo nombre en el navegador a menos que el camino (path) o el dominio fuesen diferentes. Así, para una aplicación de carro de la compra se podría querer mantener un contador e ir pasándolo. P.ej.
Ejemplo 12-4. Ejemplo SetCookie
```
<?php
$count++;
setcookie("count", $count, time()+3600);
setcookie("Cart[$count]", $item, time()+3600);
?>
```
Puntos en los nombres de variables de entrada
Típicamente, PHP no altera los nombres de las variables cuando se pasan a un script. De todas formas, hay que notar que el punto no es un carácter válido en el nombre de una variable PHP. Por esta razón:
```
<?php
$varname.ext;  /* nombre de variable invalido */
?>
```
Lo que el intérprete ve es el nombre de una variable $varname, seguido por el operador de concatenación, y seguido por la prueba (es decir, una cadena sin entrecomillar que no coincide con ninguna palabra clave o reservada conocida) 'ext'. Obviamente, no se pretendía que fuese este el resultado.
Por esta razón, es importante hacer notar que PHP reemplazará automáticamente cualquier punto en los nombres de variables de entrada por guiones bajos (subrayados).
Determinando los tipos de variables
Dado que PHP determina los tipos de las variables y los convierte (generalmente) según lo necesita, no siempre resulta obvio de qué tipo es una variable dada en un momento concreto. PHP incluye varias funciones que descubren de qué tipo es una variable: gettype(), is_array(), is_float(), is_int(), is_object(), y is_string().

Capítulo 8. Constantes

Una constante es un identificador para expresar un valor simple. Como el nombre sugiere, este valor no puede variar durante la ejecución del script. (Las constantes especiales __FILE__ y __LINE__ son una excepción a esto, ya que actualmente no lo soin). Una constante es sensible a mayúsculas por defecto. Por convención, los identificadores de constantes suelen declararse en mayúsculas
El nombre de una constante sigue las mismas reglas que cualquier etiqueta en PHP. Un nombre de constante válido empieza con una letra o un caracter de subrayado, seguido por cualquier número de letras, números, o subrayados. Se podrían expresar mediante la siguiente expresión regular:
[a-zA-Z_\x7f-\xff][a-zA-Z0-9_\x7f-\xff]*
Nota: Para nuestros propósitos , entenderemos como letra los carácteres a-z, A-Z, y los ASCII del 127 hasta el 255 (0x7f-0xff).
El alcanze de una constante es global, Es decir, es posible acceder a ellas sin preocuparse por el ámbito de alcance.
Sintaxis
Se puede definir una constante usando la función define(). Una vez definida, no

puede ser modificada ni eliminada .
Solo se puede definir como constantes valores escalares (boolean, integer, float y string).
Para obtener el valor de una constante solo es necesario especificar su nombre. A diferencia de las variables, no se tiene que especificar el prefijo $. Tambien se puede utilizar la función constant(), para obtener el valor de una constante, en el caso de que queramos expresarla de forma dinámica Usa la función get_defined_constants() parar obtener una lista de todas las constantes definidas.
Nota: Las contantes y las variables (globales) se encuentran en un espacio de nombres distinto. Esto implica que por ejemplo TRUE y $TRUE son diferentes.
Si usas una constante todavia no definida, PHP asume que estás refiriéndote al nombre de la constante en si. Se lanzará un aviso si esto sucede. Usa la función defined() para comprobar la existencia de dicha constante.
Estas son las diferencias entre constantes y variables:

- Las constantes no son precedidas por un símbolo de dolar ($)
- Las contantes solo pueden ser definidas usando la función() define , nunca por simple asignación
- Las constantes pueden ser definidas y accedidas sin tener en cuenta las reglas de alcanze del ámbito.
- Las constantes no pueden ser redefinidas o eliminadas despues de establecerse; y
- Las constantes solo puede albergar valores escalares

Ejemplo 13-1. Definiendo constantes

```
<?php
define("CONSTANT", "Hello world.");
echo CONSTANT; // outputs "Hello world."
echo Constant; // outputs "Constant" and issues a notice.
?>
```

Constantes predefinidas
PHP ofrece un largo número de constantes predefinidas a cualquier script en ejecución. Muchas de estas constantes, sin embargo, son creadas por diferentes extensiones, y solo estarán presentes si dichas extensiones están disponibles, bien por carga dinámica o porque has sido compiladas.

Capítulo 9. Expresiones

Las expresiones son la piedra angular de PHP. En PHP, casi cualquier cosa que escribes es una expresión. La forma más simple y ajustada de definir una expresión es "cualquier cosa que tiene un valor".
Las formas más básicas de expresiones son las constantes y las variables. Cuando escribes "$a = 5", estás asignando '5' a $a. '5', obviamente, tiene el valor 5 ó, en otras palabras '5' es una expresión con el valor 5 (en este caso, '5' es una constante entera).
Después de esta asignación, se espera que el valor de $a sea 5 también, de manera que si escribes $b = $a, se espera también que se comporte igual que si escribieses $b = 5. En otras palabras, $a es una expresión también con el valor 5. Si todo va bien, eso es exactamente lo que pasará.
Las funciones son un ejemplo algo más complejo de expresiones. Por ejemplo, considera la siguiente función:

```
<?php
function foo () {
   return 5;
}
```

?>
Suponiendo que estés familiarizado con el concepto de funciones (si no lo estás échale un vistazo al capítulo sobre funciones), asumirás que teclear $c = foo() es esencialmente lo mismo que escribir $c = 5, y has acertado. Las funciones son expresiones que valen el valor que retornan. Como foo() devuelve 5, el valor de la expresión 'foo()' es 5. Normalmente las funciones no devuelven un valor fijo, sino que suele ser calculado.
Desde luego, los valores en PHP no se limitan a enteros, y lo más normal es que no lo sean. PHP soporta tres tipos escalares: enteros, punto flotante y cadenas (los tipos escalares son aquellos cuyos valores no pueden 'dividirse' en partes menores, no como los arrays, por ejemplo). PHP también soporta dos tipos compuestos (no escalares): arrays y objetos. Se puede asignar cada uno de estos tipos de valor a variables o bien retornarse de funciones, sin ningún tipo de limitación.
Hasta aquí, los usuarios de PHP/FI 2 no deberían haber notado ningún cambio. Sin embargo, PHP lleva las expresiones mucho más allá, al igual que otros lenguajes. PHP es un lenguaje orientado a expresiones, en el sentido de que casi todo es una expresión. Considera el ejemplo anterior '$a = 5'. Es sencillo ver que hay dos valores involucrados, el valor de la constante entera '5', y el valor de $a que está siendo actualizado también a 5. Pero la verdad es que hay un valor adicional implicado aquí, y es el valor de la propia asignación. La asignación misma se evalúa al valor asignado, en este caso 5. En la práctica, quiere decir que '$a = 5', independientemente de lo que hace, es una expresión con el valor 5. De esta manera, escribir algo como '$b = ($a = 5)' es como escribir '$a = 5; $b = 5;' (un punto y coma marca el final de una instrucción). Como las asignaciones se evalúan de derecha a izquierda, puedes escribir también '$b = $a = 5'.
Otro buen ejemplo de orientación a expresiones es el pre y post incremento y decremento. Los usuarios de PHP/FI 2 y los de otros muchos lenguajes les sonará la notación variable++ y variable--. Esto son las operaciones de incremento y decremento. En PHP/FI 2, la instrucción '$a++' no tiene valor (no es una expresión), y no puedes asignarla o usarla de ningún otro modo. PHP mejora las características del incremento/decremento haciéndolos también expresiones, como en C. En PHP, como en C, hay dos tipos de incremento - pre-incremento y post-incremento. Ambos, en esencia, incrementan la variable y el efecto en la variable es idéntico. La diferencia radica en el valor de la propia expresion incremento. El preincremento , escrito '++$variable', se evalúa al valor incrementado (PHP incrementa la variable antes de leer su valor, de ahí el nombre 'preincremento'). El postincremento, escrito '$variable++', se evalúa al valor original de $variable antes de realizar el incremento (PHP incrementa la variable después de leer su valor, de ahí el nombre 'postincremento').
Un tipo muy corriente de expresiones son las expresiones de comparación. Estas expresiones se evalúan a 0 o 1, significando FALSO (FALSE) o VERDADERO (TRUE), respectivamente. PHP soporta > (mayor que), >= (mayor o igual que), == (igual que), != (distinto), < (menor que) y <= (menor o igual que). Estas expresiones se usan frecuentemente dentro de la ejecución condicional como la instrucción if.
El último tipo de expresiones que trataremos, es la combinación operador-asignación. Ya sabes que si quieres incrementar $a en 1, basta con escribir '$a++' o ++$a'. Pero qué pasa si quieres añadir más de 1, por ejemplo 3? Podrías escribir '$a++' múltiples veces, pero no es una forma de hacerlo ni eficiente ni cómoda. Una práctica mucho más corriente es escribir '$a = $a + 3'.

'$a + 3' se evalúa al valor de $a más 3, y se asigna de nuevo a $a, lo que resulta en incrementar $a en 3. En PHP, como en otros lenguajes como C, puedes escribir esto de una forma más concisa, que con el tiempo será más clara y también fácil de entender. Añadir 3 al valor actual de $a se puede escribir como '$a += 3'. Esto quiere decir exactamente "toma el valor de $a, súmale 3, y asígnalo otra vez a $a". Además de ser más corto y claro, también resulta en una ejecución más rápida. El valor de '$a += 3', como el valor de una asignación normal y corriente, es el valor asignado. Ten en cuenta que NO es 3, sino el valor combinado de $a más 3 (ése es el valor asignado a $a). Cualquier operación binaria puede ser usada en forma de operador-asignación, por ejemplo '$a -= 5' (restar 5 del valor de $a), '$b *= 7' (multiplicar el valor de $b por 7), etc.
Hay otra expresión que puede parecer extraña si no la has visto en otros lenguaes, el operador condicional ternario:

```
<?php
$first ? $second : $third
?>
```

Si el valor de la primera subexpresión es verdadero (distinto de cero), entonces se evalúa la segunda subexpresión, si no, se evalúa la tercera y ése es el valor.
El siguiente ejemplo te ayudará a comprender un poco mejor el pre y post incremento y las expresiones en general:

```
<?php
function double($i) {
   return $i*2;
}
$b = $a = 5;        /* asignar el valor cinco a las variables $a y $b */
$c = $a++;          /* postincremento, asignar el valor original de $a (5) a $c */
$e = $d = ++$b;     /* preincremento, asignar el valor incrementado de $b (6) a
                     $d y a $e */

/* en este punto, tanto $d como $e son iguales a 6 */

$f = double($d++);  /* asignar el doble del valor de $d antes
                     del incremento, 2*6 = 12 a $f */
$g = double(++$e);  /* asignar el doble del valor de $e despu&eacute;s
                     del incremento, 2*7 = 14 a $g */
$h = $g += 10;      /* primero, $g es incrementado en 10 y termina valiendo 24.
                     despu&eacute;s el valor de la asignaci&oacute;n (24) se asigna a
$h,
                     y $h tambi&eacute;n acaba valiendo 24. */
?>
```

Al principio del capítulo hemos dicho que describiríamos los distintos tipos de instrucciones y, como prometimos, las expresiones pueden ser instrucciones. Sin embargo, no todas las expresiones son instrucciones. En este caso, una instrucción tiene la forma 'expr' ';', es decir, una expresión seguida de un punto y coma. En '$b=$a=5;', $a=5 es una expresión válida, pero no es una instrucción en sí misma. Por otro lado '$b=$a=5:' sí es una instrucción válida.
Una última cosa que vale la pena mencionar, es el valor booleano de las expresiones. En muchas ocasiones, principalmente en condicionales y bucles, no estás interesado en el valor exacto de la expresión, sino únicamente si es TRUE ó

FALSE Las constantes TRUE y FALSE son los dos posibles valores booleanos.
Cuando es necesario, una expresión se puede transformar automáticamente al tipo booleano. Consultar la sección sobre type-casting para obtener detalles de como se hace.
PHP brinda una completa y potente implementación de expresiones, y documentarla enteramente está más allá del objetivo de este manual. Los ejemplos anteriores, deberían darte una buena idea de qué son las expresiones y cómo construir expresiones útiles. A lo largo del resto del manual, escribiremos expr para indicar una expresión PHP válida.

Capítulo 10. Operadores

Un operador es algo a lo que usted entrega uno o más valores (o expresiones, en jerga de programación) y produce otro valor (de modo que la construcción misma se convierte en una expresión). Así que puede pensar sobre las funciones o construcciones que devuelven un valor (como print) como operadores, y en aquellas que no devuelven nada (como echo) como cualquier otra cosa.
Existen tres tipos de operadores. En primer lugar se encuentra el operador unario, el cual opera sobre un único valor, por ejemplo ! (el operador de negación) o ++ (el operador de incremento). El segundo grupo se conoce como operadores binarios; éste grupo contiene la mayoría de operadores que soporta PHP, y una lista se encuentra disponible más adelante en la sección Precedencia de Operadores .
El tercer grupo consiste del operador ternario: ?:. Éste debe ser usado para seleccionar entre dos expresiones, en base a una tercera, en lugar de seleccionar dos sentencias o rutas de ejecución. Rodear las expresiones ternarias con paréntesis es una muy buena idea.
Precedencia de Operadores
La precedencia de un operador indica qué tan "cerca" se agrupan dos expresiones. Por ejemplo, en la expresión 1 + 5 * 3, la respuesta es 16 y no 18, ya que el operador de multiplicación ("*") tiene una mayor precedencia que el operador de adición ("+"). Los paréntesis pueden ser usados para marcar la precedencia, si resulta necesario. Por ejemplo: (1 + 5) * 3 evalúa a 18. Si la precedencia de los operadores es la misma, se utiliza una asociación de izquierda a derecha.
La siguiente tabla lista la precedencia de los operadores, con aquellos de mayor precedencia listados al comienzo de la tabla. Los operadores en la misma línea tienen la misma precedencia, en cuyo caso su asociatividad decide el orden para evaluarlos.
Tabla 15-1. Precedencia de Operadores

Asociatividad	Operadores	Información Adicional
no-asociativo	new	new
izquierda	[	array()

Asociatividad	Operadores	Información Adicional
no-asociativos	++ --	incremento/decremento
no-asociativos	! ~ - (int) (float) (string) (array) (object) @	tipos
izquierda	* / %	aritmética
izquierda	+ - .	aritmética, y cadena
izquierda	<< >>	manejo de bits
no-asociativos	< <= > >=	comparación
no-asociativos	== != === !==	comparación
izquierda	&	manejo de bits, y referencias
izquierda	^	manejo de bits
izquierda	|	manejo de bits
izquierda	&&	lógicos
izquierda	||	lógicos
izquierda	? :	ternario
derecha	= += -= *= /= .= %= &= |= ^= <<= >>=	asignación
izquierda	and	lógicos
izquierda	xor	lógicos
izquierda	or	lógicos
izquierda	,	varios usos

La asociatividad de izquierda quiere decir que la expresión es evaluada desde la izquierda a la derecha, la asociatividad de derecha quiere decir lo contrario.

Ejemplo 15-1. Asociatividad

```
<?php
$a = 3 * 3 % 5; // (3 * 3) % 5 = 4
$a = true ? 0 : true ? 1 : 2; // (true ? 0 : true) ? 1 : 2 = 2

$a = 1;
$b = 2;
$a = $b += 3; // $a = ($b += 3) -> $a = 5, $b = 5
?>
```

Use paréntesis para incrementar la legibilidad del código.

Nota: Aunque ! tiene una mayor precedencia que =, PHP permitirá aun expresionas similares a la siguiente: if (!$a = foo()), en cuyo caso la salida de foo() va a dar a $a.

Operadores de Aritmética

¿Recuerda la aritmética básica del colegio? Éstos operadores funcionan tal como aquéllos.

Tabla 15-2. Operadores de Aritmética

Ejemplo	Nombre	Resultado
-$a	Negación	El opuesto de $a.
$a + $b	Adición	Suma de $a y $b.
$a - $b	Substracción	Diferencia entre $a y $b.
$a * $b	Multiplicación	Producto de $a y $b.
$a / $b	División	Cociente de $a y $b.
$a % $b	Módulo	Resto de $a dividido por $b.

El operador de división ("/") devuelve un valor flotante en todos los casos, incluso si los dos operandos son enteros (o cadenas que son convertidas a enteros).
Nota: El resto de $a % $b es negativo para valores negativos de $a.
Operadores de Asignación
El operador básico de asignación es "=". A primera vista, usted podría pensar en él como "es igual a". No lo haga. Lo que quiere decir en realidad es que el operando de la izquierda recibe el valor de la expresión a la derecha (es decir, "se define a").
El valor de una expresión de asignación es el valor que se asigna. Es decir, el valor de "$a = 3" es 3. Esto le permite hacer una que otra cosa curiosa:

```
<?php

$a = ($b = 4) + 5; // $a es igual a 9 ahora, y $b ha sido definido a 4.

?>
```

En conjunto con el operador básico de asignación, existen "operadores combinados" para todos los operadores de aritmética binaria, unión de matrices y de cadenas, que le permiten usar un valor en una expresión y luego definir su valor como el resultado de esa expresión. Por ejemplo:

```
<?php

$a = 3;
$a += 5; // define $a como 8, como si hubiesemos dicho: $a = $a + 5;
$b = "&iexcl;Hola ";
$b .= "a todos!"; // define $b como "&iexcl;Hola a todos!", tal como $b = $b . "a todos!";

?>
```

Note que la asignación copia la variable original en la nueva (asignación por valor), de modo que cualquier cambio a una no afecta a la otra. Esto puede resultar de importancia si necesita copiar algo como una matriz de gran tamaño al interior de un ciclo reducido. A partir de PHP4, es soportada la asignación por referencia, usando la sintaxis $var = &$otra_var;, pero esto no es posible en PHP3. 'Asignación por referencia' quiere decir que ambas variables terminan apuntando a los mismos datos y que nada es realmente copiado.
Operadores Bit a Bit
Los operadores bit a bit le permiten activar o desactivar bits individuales de un entero. Si los parámetros tanto a la izquierda y a la derecha son cadenas, el operador bit a bit trabajará sobre los valores ASCII de los caracteres.

```
<?php
echo 12 ^ 9; // Imprime '5'

echo "12" ^ "9"; // Imprime el caracter Backspace (ascii 8)
                 // ('1' (ascii 49)) ^ ('9' (ascii 57)) = #8

echo "hallo" ^ "hello"; // Imprime los valores ascii #0 #4 #0 #0 #0
                        // 'a' ^ 'e' = #4
?>
```

Tabla 15-3. Operadores Bit a Bit

Ejemplo	Nombre	Resultado
$a & $b	Y	Los bits que están activos tanto en $a como en $b son activados.
$a | $b	O	Los bits que están activos ya sea en $a o en $b son activados.
$a ^ $b	O exclusivo (Xor)	Los bitos que estén activos en $a o $b, pero no en ambos, son activados.
~ $a	No	Los bits que estén activos en $a son desactivados, y vice-versa.
$a << $b	Desplazamiento a izquierda	Desplaza los bits de $a, $b pasos a la izquierda (cada paso quiere decir "multiplicar por dos")
$a >> $b	Desplazamiento a derecha	Desplaza los bits de $a, $b pasos a la derecha (cada paso quiere decir "dividir por dos")
Aviso		
No realice desplazamientos a derecha para más de 32 bits en sistemas de 32 bits. No realice desplazamientos a izquierda en caso de que resulte en un número de más de 32 bits.		

Operadores de Comparación

Los operadores de comparación, como su nombre indica, le permiten comparar dos valores. Puede que también se encuentre interesado en consultar las tablas de comparación de tipos, ya que éstas muestran ejemplos de varios tipos de comparaciones relacionadas con tipos.

Tabla 15-4. Operadores de Comparación

Ejemplo	Nombre	Resultado
$a == $b	Igual	TRUE si $a es igual a $b.
$a === $b	Idéntico	TRUE si $a es igual a $b, y son del mismo tipo. (A partir de PHP 4)
$a != $b	Diferente	TRUE si $a no es igual a $b.
$a <> $b	Diferente	TRUE si $a no es igual a $b.
$a !== $b	No idénticos	TRUE si $a no es igual a $b, o si no son del mismo tipo. (A partir de PHP 4)
$a < $b	Menor que	TRUE si $a es escrictamente menor que $b.
$a > $b	Mayor que	TRUE si $a es estrictamente mayor que $b.
$a <= $b	Menor o igual que	TRUE si $a es menor o igual que $b.
$a >= $b	Mayor o igual que	TRUE si $a es mayor o igual que $b.

Si compara un entero con una cadena, la cadena es convertida a un número. Si compara dos cadenas numéricas, ellas son comparadas como enteros. Estas reglas también se aplican a la sentencia switch.

```
<?php
var_dump(0 == "a"); // 0 == 0 -> true
var_dump("1" == "01"); // 1 == 1 -> true

switch ("a") {
```

```
case 0:
  echo "0";
  break;
case "a": // nunca se ejecuta ya que "a" ya ha coincidido con 0
  echo "a";
  break;
}
?>
```

Para varios tipos, la comparación se realiza de acuerdo con la siguient tabla (en orden).

Tabla 15-5. Comparación con Varios Tipos

Tipo del Operando 1	Tipo del Operando 2	Resultado
null o string	string	Convertir NULL a "", comparación numérica o de léxico
bool o null	cualquiera	Convertir a bool, FALSE < TRUE
object	object	Las clases internas pueden definir su propia comparación, clases diferentes son incomparables, la misma clase - comparan propiedades en la misma forma que las matrices (PHP 4), PHP 5 tiene su propia explicación
string, resource o number	string, resource o number	Traducir las cadenas y recursos a números, matemática usual
array	array	Una matriz con menos elementos es menor, si una clave del operando 1 no se encuentra en el operando 2 entonces las matrices son incomparables, de otra forma - comparar valor por valor (vea el siguiente ejemplo)
array	cualquiera	array es siempre mayor
object	cualquiera	object es siempre mayor

Ejemplo 15-2. Transcripción de la comparación de matrices estándar

```
<?php
// Las matrices son comparadas de esta forma con los operadores de comparacion estandar
function comparacion_matrices_estandar($op1, $op2)
{
  if (count($op1) < count($op2)) {
     return -1; // $op1 < $op2
  } elseif (count($op1) > count($op2)) {
     return 1; // $op1 > $op2
  }
  foreach ($op1 as $clave => $val) {
     if (!array_key_exists($clave, $op2)) {
        return null; // incomparable
     } elseif ($val < $op2[$clave]) {
        return -1;
     } elseif ($val > $op2[$clave]) {
```

```
        return 1;
      }
    }
  }
  return 0; // $op1 == $op2
}
?>
```

Operador Ternario
Otro operador condicional es el operador "?:" (o ternario).
Ejemplo 15-3. Asignación de un valor predeterminado

```
<?php
// Ejemplo de uso de: el Operador Ternario
$accion = (empty($_POST['accion'])) ? 'predeterminada' : $_POST['accion'];

// La sentencia anterior es identica a este bloque if/else
if (empty($_POST['accion'])) {
  $accion = 'predeterminada';
} else {
  $accion = $_POST['accion'];
}

?>
```

La expresión (expr1) ? (expr2) : (expr3) evalúa a expr2 si expr1 evalúa a TRUE, y expr3 si expr1 evalúa a FALSE.
Nota: Por favor note que el operador ternario es una sentencia, y que no evalúa a una variable, sino al resultado de una sentencia. Es importante saber esto si se desea devolver una variable por referencia. La sentencia return $var == 42 ? $a : $b; en una función con retorno-por-referencia no funcionará por lo que se ha mencionado y una advertencia es generada en versiones posteriores de PHP.
Operadores de Control de Errores
PHP ofrece soporte para un operador de control de errores: el signo de arroba (@). Cuando es colocado al comienzo de una expresión en PHP, cualquier mensaje de error que pudiera generarse a causa de esa expresión será ignorado.
Si la característica track_errors está habilitada, cualquier mensaje de error generado por la expresión será almacenado en la variable $php_errormsg. La variable será sobrescrita en cada instancia de error, así que realice sus chequeos de forma temprana si quiere usarla.

```
<?php
/* Error intencional de archivo */
$mi_archivo = @file ('archivo_que_no_existe') or
  die ("La apertura de archivo ha fallado: el error fue '$php_errormsg'");

// esto funciona con cualquier expresion, no solo con funciones:
$valor = @$cache[$llave];
// no producira una anotacion si el indice $llave no existe.

?>
```

Nota: El operador @ trabaja sólo sobre expresiones. Una simple regla de oro es: si usted puede tomar el valor de algo, entonces puede usar el operador @ sobre ese

algo. Por ejemplo, puede usarlo al inicio de variables, llamadas a funciones y sencencias include(), constantes, y así sucesivamente. No puede usarlo sobre definiciones de función o clase, ni sobre estructuras condicionales como if y foreach, y así sucesivamente.
Vea también error_reporting() y la sección del manual sobre funciones de Gestión de Errores y Registros.

Aviso
En la actualidad, el operador de prefijo "@" para control de errores deshabilitará incluso el reporte de errores en casos de fallos críticos que terminarán la ejecución del script. Entre otras cosas, esto quiere decir que si usa "@" para eliminar los errores de una cierta función, y ésta no se encuentra disponible o ha sido escrita de forma incorrecta, el script se detendrá en ese punto sin dar indicación alguna del motivo.

Operadores de ejecución
PHP soporta un operador de ejecución: las comillas invertidas (``). ¡Note que no se trata de comillas sencillas! PHP intentará ejecutar el contenido entre las comillas como si se tratara de un comando del intérprete de comandos; su salida será devuelta (es decir, no será simplemente volcada como salida; puede ser asignada a una variable). El uso del operador de comillas invertidas es idéntico al de shell_exec().

```
<?php
$salida = `ls -al`;
echo "<pre>$salida</pre>";
?>
```

Nota: El operador de comillas invertidas es deshabilitado cuando se encuentra activo safe mode o cuando se deshabilita shell_exec().
Operadores de Incremento/Decremento
PHP ofrece soporte de operadores de pre- y post-incremento y decremento, estilo-C.
Nota: Los operadores de incremento/decremento no afectan a los valores booleanos. Decrementar valores NULL tampoco tiene efecto, aunque incrementarlos resulta en 1.
Tabla 15-6. Operadores de Incremento/decremento

Ejemplo	Nombre	Efecto
++$a	Pre-incremento	Incrementa $a en uno, y luego devuelve $a.
$a++	Post-incremento	Devuelve $a, y luego incrementa $a en uno.
--$a	Pre-decremento	Decrementa $a en uno, luego devuelve $a.
$a--	Post-decremento	Devuelve $a, luego decrementa $a en uno.

Aquí hay un script sencillo de ejemplo:

```
<?php
echo "<h3>Postincremento</h3>";
$a = 5;
echo "Debe ser 5: " . $a++ . "<br />\n";
echo "Debe ser 6: " . $a . "<br />\n";

echo "<h3>Preincremento</h3>";
$a = 5;
```

```
echo "Debe ser 6: " . ++$a . "<br />\n";
echo "Debe ser 6: " . $a . "<br />\n";

echo "<h3>Postdecremento</h3>";
$a = 5;
echo "Debe ser 5: " . $a-- . "<br />\n";
echo "Debe ser 4: " . $a . "<br />\n";

echo "<h3>Predecremento</h3>";
$a = 5;
echo "Debe ser 4: " . --$a . "<br />\n";
echo "Debe ser 4: " . $a . "<br />\n";
?>
```

PHP sigue la convención de Perl cuando trabaja con operaciones aritméticas sobre variables de caracter, y no la convención de C. Por ejemplo, en Perl 'Z'+1 se convierte en 'AA', mientras que en C 'Z'+1 se convierte en '[' (ord('Z') == 90, ord('[') == 91). Note que las variables de caracter pueden ser incrementadas pero no decrementadas.

Ejemplo 15-4. Operaciones Aritméticas sobre Variables de Caracter

```
<?php
$i = 'W';
for ($n=0; $n<6; $n++) {
   echo ++$i . "\n";
}
?>
```

El resultado del ejemplo seria:

```
X
Y
Z
AA
AB
AC
```

Incrementar o decrementar valores booleanos no tiene efecto.

Operadores de Lógica

Tabla 15-7. Operadores de Lógica

Ejemplo	Nombre	Resultado
$a and $b	Y	TRUE si tanto $a como $b son TRUE.
$a or $b	O	TRUE si cualquiera de $a o $b es TRUE.
$a xor $b	O exclusivo (Xor)	TRUE si $a o $b es TRUE, pero no ambos.
! $a	No	TRUE si $a no es TRUE.
$a && $b	Y	TRUE si tanto $a como $b son TRUE.
$a || $b	O	TRUE si cualquiera de $a o $b es TRUE.

La razón para tener las dos variaciones diferentes de los operadores "and" y "or" es que ellos operan con precedencias diferentes. (Vea Precedencia de Operadores.)

Operadores de Cadena

Existen dos operadores para datos tipo string. El primero es el operador de concatenación ('.'), el cual devuelve el resultado de concatenar sus argumentas a lado derecho e izquierdo. El segundo es el operador de asignación sobre concatenación ('.='), el cual adiciona el argumento del lado derecho al argumento

en el lado izquierdo. Por favor consulte Operadores de Asignación para más información.

```
<?php
$a = "&iexcl;Hola ";
$b = $a . "Mundo!"; // ahora $b contiene "&iexcl;Hola Mundo!"

$a = "&iexcl;Hola ";
$a .= "Mundo!";    // ahora $a contiene "&iexcl;Hola Mundo!"
?>
```

Operadores de Matrices

Tabla 15-8. Operadores de Matrices

Ejemplo	Nombre	Resultado
$a + $b	Unión	Unión de $a y $b.
$a == $b	Igualdad	TRUE si $a y $b tienen las mismas parejas llave/valor.
$a === $b	Identidad	TRUE si $a y $b tienen las mismas parejas llave/valor en el mismo orden y de los mismos tipos.
$a != $b	No-igualdad	TRUE si $a no es igual a $b.
$a <> $b	No-igualdad	TRUE si $a no es igual a $b.
$a !== $b	No-identidad	TRUE si $a no es idéntico a $b.

El operador + adiciona la matriz del lado derecho a aquél al lado izquierdo, al mismo tiempo que cualquier llave duplicada NO es sobrescrita.

```
 <?php
$a = array("a" => "manzana", "b" => "banano");
$b = array("a" => "pera", "b" => "fresa", "c" => "cereza");

$c = $a + $b; // Union de $a y $b
echo "Uni&oacute;n de \$a y \$b: \n";
var_dump($c);

$c = $b + $a; // Union de $b y $a
echo "Uni&oacute;n de \$b y \$a: \n";
var_dump($c);
?>
```

Cuando sea ejecutado, este script producirá la siguiente salida:

```
Uni&oacute;n de $a y $b:
array(3) {
  ["a"]=>
  string(7) "manzana"
  ["b"]=>
  string(6) "banano"
  ["c"]=>
  string(6) "cereza"
}
```

Unión de $b y $a:
array(3) {
  ["a"]=>
  string(4) "pera"
  ["b"]=>
  string(5) "fresa"
  ["c"]=>
  string(6) "cereza"
}
Los elementos de las matrices son considerados equivalentes en la comparación si éstos tienen la misma clave y valor.
<?php
$a = array("manzana", "banano");
$b = array(1 => "banano", "0" => "manzana");

var_dump($a == $b); // bool(true)
var_dump($a === $b); // bool(false)
?> Vea también las secciones del manual sobre el tipo Array y funciones de Matrices.
Operadores de Tipo
PHP tiene un operador único de tipo: instanceof es usado para determinar si un objeto dado, sus padres o sus implementaciones de interfaces son de una clase de objeto especificada.
El operador instanceof fue introducido en PHP 5. Antes de esta versión, is_a() era utilizado, pero is_a() ha sido marcado como obsoleto desde entonces en favor de instanceof.
<?php
class A { }
class B { }

$cosa = new A;

if ($cosa instanceof A) {
   echo 'A';
}
if ($cosa instanceof B) {
   echo 'B';
}
?>
Dado que $cosa es un object de tipo A, pero no de tipo B, sólo el bloque dependiente del tipo A será ejecutado:

A

Capítulo 11. Estructuras de Control

Todo script PHP se compone de una serie de sentencias. Una sentencia puede ser una asignación, una llamada a función, un bucle, una sentencia condicional e incluso una sentencia que no haga nada (una sentencia vacía). Las sentencias normalmente acaban con punto y coma. Además, las sentencias se pueden agrupar en grupos de sentencias encapsulando un grupo de sentencias con llaves. Un grupo de sentencias es también una sentencia. En este capítulo se describen los diferentes tipos de sentencias.

if

La construcción if es una de las más importantes características de muchos lenguajes, incluido PHP. Permite la ejecución condicional de fragmentos de código. PHP caracteriza una estructura if que es similar a la de C:

```
<?php
if (expr)
    sentencia
?>
```

Como se describe en la sección sobre expresiones, expr se evalúa a su valor condicional (boolean). Si expr se evalúa como TRUE, PHP ejecutará la sentencia, y si se evalúa como FALSE - la ignorará. Se puede encontrar más información sobre los valores evaluados como FALSE en la sección Convirtiendo a un valor condicional (boolean)'.

El siguiente ejemplo mostraría a es mayor que b si $a fuera mayor que $b:

```
<?php
if ($a > $b)
   print "a es mayor que b";
?>
```

A menudo, se desea tener más de una sentencia ejecutada de forma condicional. Por supuesto, no hay necesidad de encerrar cada sentencia con una cláusula if. En vez de eso, se pueden agrupar varias sentencias en un grupo de sentencias. Por ejemplo, este código mostraría a es mayor que b si $a fuera mayor que $b, y entonces asignaría el valor de $a a $b:

```
<?php
 if ($a > $b) {
     print "a es mayor que b";
     $b = $a;
 }
?>
```

Las sentencias if se pueden anidar indefinidamente dentro de otras sentencias if, lo cual proporciona una flexibilidad completa para ejecuciones condicionales en las diferentes partes de tu programa.

else

A menudo queremos ejecutar una sentencia si se cumple una cierta condicion, y una sentencia distinta si la condición no se cumple. Esto es para lo que sirve else. else extiende una sentencia if para ejecutar una sentencia en caso de que la expresión en la sentencia if se evalúe como FALSE. Por ejemplo, el siguiente código mostraría a es mayor que b si $a fuera mayor que $b, y a NO es mayor que b en cualquier otro caso:

```
<?php
 if ($a > $b) {
     print "a es mayor que b";
 } else {
     print "a NO es mayor que b";
 }
?>
```

La sentencia else se ejecuta solamente si la expresión if se evalúa como FALSE, y si hubiera alguna expresión elseif - sólo si se evaluaron también a FALSE (Ver elseif).

elseif

elseif, como su nombre sugiere, es una combinación de if y else. Como else, extiende una sentencia if para ejecutar una sentencia diferente en caso de que la

expresión if original se evalúa como FALSE. No obstante, a diferencia de else, ejecutará esa expresión alternativa solamente si la expresión condicional elseif se evalúa como TRUE. Por ejemplo, el siguiente código mostraría a es mayor que b, a es igual a b o a es menor que b:

```
<?php
 if ($a > $b) {
    print "a es mayor que b";
 } elseif ($a == $b) {
    print "a es igual que b";
 } else {
    print "a es mayor que b";
 }
?>
```

Puede haber varios elseifs dentro de la misma sentencia if. La primera expresión elseif (si hay alguna) que se evalúe como TRUE se ejecutaría. En PHP, también se puede escribir 'else if' (con dos palabras) y el comportamiento sería idéntico al de un 'elseif' (una sola palabra). El significado sintáctico es ligeramente distinto (si estas familiarizado con C, es el mismo comportamiento) pero la línea básica es que ambos resultarían tener exactamente el mismo comportamiento.

La sentencia elseif se ejecuta sólo si la expresión if precedente y cualquier expresión elseif precedente se evalúan como FALSE, y la expresión elseif actual se evalúa como TRUE.

Sintaxis Alternativa de Estructuras de Control

PHP ofrece una sintaxis altenativa para alguna de sus estructuras de control; a saber, if, while, for, y switch. En cada caso, la forma básica de la sintaxis alternativa es cambiar abrir-llave por dos puntos (:) y cerrar-llave por endif;, endwhile;, endfor;, or endswitch;, respectivamente.

```
<?php if ($a==5): ?>
A es igual a 5
 <?php endif; ?>
```

En el ejemplo de arriba, el bloque HTML "A es igual 5" se anida dentro de una sentencia if escrita en la sintaxis alternativa. El bloque HTML se mostraría solamente si $a fuera igual a 5.

La sintaxis alternativa se aplica a else y también a elseif. La siguiente es una estructura if con elseif y else en el formato alternativo:

```
<?php
 if ($a == 5):
    print "a es igual a 5";
    print "...";
 elseif ($a == 6):
    print "a es igual a 6";
    print "!!!";
 else:
    print "a no es ni 5 ni 6";
 endif;
?>
```

Mirar también while, for, e if para más ejemplos.

while

Los bucles while son los tipos de bucle más simples en PHP. Se comportan como su

contrapartida en C. La forma básica de una sentencia while es:
while (expr) sentencia
El significado de una sentencia while es simple. Le dice a PHP que ejecute la(s) sentencia(s) anidada(s) repetidamente, mientras la expresión while se evalúe como TRUE. El valor de la expresión es comprobado cada vez al principio del bucle, así que incluso si este valor cambia durante la ejecución de la(s) sentencia(s) anidada(s), la ejecución no parará hasta el fin de la iteración (cada vez que PHP ejecuta las sentencias en el bucle es una iteración). A veces, si la expresión while se evalúa como FALSE desde el principio de todo, la(s) sentencia(s) anidada(s) no se ejecutarán ni siquiera una vez.
Como con la sentencia if, se pueden agrupar multiples sentencias dentro del mismo bucle while encerrando un grupo de sentencias con llaves, o usando la sintaxis alternativa:
while (expr): sentencia ... endwhile;
Los siguientes ejemplos son idénticos, y ambos imprimen números del 1 al 10:

```
<?php
/* ejemplo 1 */

 $i = 1;
 while ($i <= 10) {
    print $i++;  /* el valor impreso ser&iacute;a
                    $i antes del incremento
                    (post-incremento) */
 }

 /* ejemplo 2 */

 $i = 1;
 while ($i <= 10):
    print $i;
    $i++;
 endwhile;
?>
```

do..while
Los bucles do..while son muy similares a los bucles while, excepto que las condiciones se comprueban al final de cada iteración en vez de al principio. La principal diferencia frente a los bucles regulares while es que se garantiza la ejecución de la primera iteración de un bucle do..while (la condición se comprueba sólo al final de la iteración), mientras que puede no ser necesariamente ejecutada con un bucle while regular (la condición se comprueba al principio de cada iteración, si esta se evalúa como FALSE desde el principio la ejecución del bucle finalizará inmediatamente).
Hay una sola sintaxis para los bucles do..while:

```php
<?php
 $i = 0;
 do {
    print $i;
 } while ($i>0);
?>
```

El bucle de arriba se ejecutaría exactamente una sola vez, después de la primera iteración, cuando la condición se comprueba, se evalúa como FALSE ($i no es más grande que 0) y la ejecución del bucle finaliza.

Los usuarios avanzados de C pueden estar familiarizados con un uso distinto del bucle do..while, para permitir parar la ejecución en medio de los bloques de código, encapsulandolos con do..while(0), y usando la sentencia break. El siguiente fragmento de código demuestra esto:

```php
<?php
do {
    if ($i < 5) {
        print "i no es lo suficientemente grande";
        break;
    }
    $i *= $factor;
    if ($i < $minimum_limit) {
        break;
    }
    print "i es correcto";
    /* procesa i */
 } while(0);
?>
```

No se preocupes si no entiende esto completamente o en absoluto. Se pueden codificar archivos de comandos e incluso archivos de comandos potentes sin usar esta 'propiedad'.

for

Los bucles for son los bucles más complejos en PHP. Se comportan como su contrapartida en C. La sintaxis de un bucle for es:

for (expr1; expr2; expr3) sentencia

La primera expresión (expr1) se evalúa (ejecuta) incondicionalmente una vez al principio del bucle.

Al comienzo de cada iteración, se evalúa expr2 . Si se evalúa como TRUE, el bucle continúa y las sentencias anidadas se ejecutan. Si se evalúa como FALSE, la ejecución del bucle finaliza.

Al final de cada iteración, se evalúa (ejecuta) expr3.

Cada una de las expresiones puede estar vacía. Que expr2 esté vacía significa que el bucle debería correr indefinidamente (PHP implicitamente lo considera como TRUE, al igual que C). Esto puede que no sea tan inútil como se podría pensar, puesto que a menudo se quiere salir de un bucle usando una sentencia break condicional en vez de usar la condición de for.

Considera los siguientes ejemplos. Todos ellos muestran números del 1 al 10:

```php
<?php
 /* ejemplo 1 */

 for ($i = 1; $i <= 10; $i++) {
    print $i;
 }

 /* ejemplo 2 */

 for ($i = 1; ;$i++) {
    if ($i > 10) {
        break;
    }
    print $i;
 }

 /* ejemplo 3 */

 $i = 1;
 for (;;) {
    if ($i > 10) {
        break;
    }
    print $i;
    $i++;
 }

 /* ejemplo 4 */

 for ($i = 1; $i <= 10; print $i, $i++) ;
?>
```

Por supuesto, el primer ejemplo parece ser el mas elegante (o quizás el cuarto), pero uno puede descubrir que ser capaz de usar expresiones vacías en bucles for resulta útil en muchas ocasiones.

PHP también soporta la "sintaxis de dos puntos" alternativa para bucles for.

for (expr1; expr2; expr3): sentencia; ...; endfor;

Otros lenguajes poseen una sentencia foreach para traducir un array o una tabla hash. PHP3 no posee tal construcción; PHP4 sí (ver foreach). En PHP3, se puede combinar while con las funciones list() y each() para conseguir el mismo efecto. Mirar la documentación de estas funciones para ver un ejemplo.

foreach

PHP 4 (PHP3 no) incluye una construcción foreach, tal como perl y algunos otros lenguajes. Esto simplemente da un modo fácil de iterar sobre matrices. foreach funciona solamente con matrices y devolverá un error si se intenta utilizar con otro tipo de datos ó variables no inicializadas. Hay dos sintaxis; la segunda es una extensión menor, pero útil de la primera:

foreach(expresion_array as $value) sentencia

foreach(expresion_array as $key => $value) sentencia

La primera forma recorre el array dado por expresion_array. En cada iteración, el

valor del elemento actual se asigna a $value y el puntero interno del array se avanza en una unidad (así en el siguiente paso, se estará mirando el elemento siguiente).
La segunda manera hace lo mismo, salvo que la clave del elemento actual será asignada a la variable $key en cada iteración.
Nota: Cuando foreach comienza su primera ejecución, el puntero interno a la matriz se reinicia automáticamente al primer elemento de la matriz. Esto significa que no se necesita llamar a reset() antes de un bucle foreach.
Nota: Hay que tener en cuenta queforeach trabaja con una copia de la matriz especificada y no la lista en si, por ello el puntero de la lista no es modificado como en la función each(), y los cambios en el elemento de la matriz retornado no afectan a la matriz original. De todas maneras el puntero interno a la matriz original avanza al procesar la matriz. suponiendo que bucle foreach se ejecuta hasta el final, el puntero interno a la matriz estar/aacute; al final de la matriz.
Nota: foreach no soporta la caracteristica de suprimir mensajes de error con '@'.
Puede haber observado que las siguientes son funcionalidades idénticas:

```
<?php
$arr = array("one", "two", "three");
reset ($arr);
while (list(, $value) = each ($arr)) {
   echo "Value: $value<br>\n";
}

foreach ($arr as $value) {
   echo "Value: $value<br>\n";
}
?>
```

Las siguientes también son funcionalidades idénticas:

```
<?php
reset( $arr );
while( list( $key, $value ) = each( $arr ) ) {
   echo "Key: $key; Valor: $value&lt;br&gt;\n";
}

foreach( $arr as $key => $value ) {
   echo "Key: $key; Valor: $value&lt;br&gt;\n";
}
?>
```

Algunos ejemplos más para demostrar su uso:

```
<?php
/* foreach ejemplo 1: s&oacute;lo valor*/
$a = array(1, 2, 3, 17);

foreach($a as $v) {
   print "Valor actual de \$a: $v.\n";
}

/* foreach ejemplo 2: valor (con clave impresa para ilustrar) */
$a = array(1, 2, 3, 17);

$i = 0; /* s&oacute;lo para prop&oacute;sitos demostrativos */

foreach($a as $v) {
   print "\$a[$i] => $v.\n";
   $i++;
}

/* foreach ejemplo 3: clave y valor */
$a = array(
   "uno" => 1,
   "dos" => 2,
   "tres" => 3,
   "diecisiete" => 17
);

foreach($a as $k => $v) {
   print "\$a[$k] => $v.\n";
}

/* foreach ejemplo 4: matriz multi-dimensional */

$a[0][0] = "a";
$a[0][1] = "b";
$a[1][0] = "y";
$a[1][1] = "z";

foreach($a as $v1) {
   foreach ($v1 as $v2) {
      print "$v2\n";
   }
}

/* foreach ejemplo 5: matriz din&aacute;mica */

foreach(array(1, 2, 3, 4, 5) as $v) {
   print "$v\n";
}
?>
```

break
break escapa de la estructuras de control iterante (bucle) actuales for, while, o switch.
break accepta un parámetro opcional, el cual determina cuantas estructuras de control hay que escapar.

```
<?php
$arr = array ('one', 'two', 'three', 'four', 'stop', 'five');
while (list (, $val) = each ($arr)) {
  if ($val == 'stop') {
     break;    /* You could also write 'break 1;' here. */
  }
  echo "$val<br>\n";
}

/* Using the optional argument. */

$i = 0;
while (++$i) {
  switch ($i) {
  case 5:
     echo "At 5<br>\n";
     break 1;  /* Exit only the switch. */
  case 10:
     echo "Al 10; saliendo<br>\n";
     break 2;  /* Exit the switch and the while. */
  default:
     break;
  }
}
?>
```

continue
continue se usa dentro de la estructura del bucle para saltar el resto de la iteración actual del bucle y continuar la ejecución al comienzo de la siguiente iteración.
Nota: Tener en cuenta que en PHP la declaración switch es considerada una estructura de bucle por continue.
continue accepta un parámetro opcional, el cual determina cuantos niveles (bluces) hay que saltar antes de continuar con la ejecución.

```
<?php
while (list ($key, $value) = each ($arr)) {
  if (!($key % 2)) { // skip odd members
     continue;
  }
  do_something_odd ($value);
}

$i = 0;
while ($i++ < 5) {
  echo "Outer<br>\n";
  while (1) {
     echo "  Middle<br>\n";
     while (1) {
        echo "  Inner<br>\n";
        continue 3;
     }
     echo "Esto nunca se imprime.<br>\n";
  }
  echo "Y esto tampoco.<br>\n";
}
?>
```

switch

La sentencia switch es similar a una serie de sentencias IF en la misma expresión. En muchas ocasiones, se quiere comparar la misma variable (o expresión) con nuchos valores diferentes, y ejecutar una parte de código distinta dependiendo de a qué valor es igual. Para ello sirve la sentencia switch.

Nota: Tener en cuenta que al contrario que otros lenguajes de programación, continue se aplica a switch y funciona de manera similar a break. Si teneis un switch dentro de un bucle y deseais continuar con el paso siguiente en el bucle externo, usar continue 2.

Los siguientes dos ejemplos son dos modos distintos de escribir la misma cosa, uno usa una serie de sentencias if, y el otro usa la sentencia switch:

```
<?php
if ($i == 0) {
   print "i equals 0";
} elseif ($i == 1) {
   print "i equals 1";
} elseif ($i == 2) {
   print "i equals 2";
}

switch ($i) {
   case 0:
      print "i equals 0";
      break;
   case 1:
      print "i equals 1";
      break;
   case 2:
      print "i equals 2";
      break;
}
?>
```

Es importante entender cómo se ejecuta la sentencia switch para evitar errores. La sentencia switch ejecuta línea por línea (realmente, sentencia a sentencia). Al comienzo, no se ejecuta código. Sólo cuando se encuentra una sentencia case con un valor que coincide con el valor de la expresión switch PHP comienza a ejecutar las sentencias. PHP continúa ejecutando las sentencias hasta el final del bloque switch, o la primera vez que vea una sentencia break. Si no se escribe una sentencia break al final de una lista de sentencias case, PHP seguirá ejecutando las sentencias del siguiente case. Por ejemplo:

```
<?php
 switch ($i) {
    case 0:
       print "i es igual a 0";
    case 1:
       print "i es igual a 1";
    case 2:
       print "i es igual a 2";
 }
?>
```

Aquí, si $i es igual a 0, ¡PHP ejecutaría todas las sentecias print! Si $i es igual a 1, PHP ejecutaría las últimas dos sentencias print y sólo si $i es igual a 2, se obtendría la conducta 'esperada' y solamente se mostraría 'i es igual a 2'. Así, es importante no olvidar las sentencias break (incluso aunque pueda querer evitar escribirlas intencionadamente en ciertas circunstancias).

En una sentencia switch, la condición se evalúa sólo una vez y el resultado se compara a cada sentencia case. En una sentencia elseif, la condición se evalúa otra vez. Si tu condición es más complicada que una comparación simple y/o está en un bucle estrecho, un switch puede ser más rápido.

La lista de sentencias de un case puede también estar vacía, lo cual simplemente

pasa el control a la lista de sentencias del siguiente case.

```
<?php
switch ($i) {
    case 0:
    case 1:
    case 2:
        print "i es menor que 3, pero no negativo";
        break;
    case 3:
        print "i es 3";
 }
?>
```

Un caso especial es el default case". Este "case" coincide con todo lo que no coincidan los otros case. Por ejemplo:

```
<?php
 switch ($i) {
    case 0:
        print "i es igual a 0";
        break;
    case 1:
        print "i es igual a 1";
        break;
    case 2:
        print "i es igual a 2";
        break;
    default:
        print "i no es igual a 0, 1 o 2";
 }
?>
```

La expresión case puede ser cualquier expresión que se evalúe a un tipo simple, es decir, números enteros o de punto flotante y cadenas de texto. No se pueden usar aquí ni arrays ni objetos a menos que se conviertan a un tipo simple.

La sintaxis alternativa para las estructuras de control está también soportada con switch. Para más información, ver Sintaxis alternativa para estructuras de control.

```
<?php
switch ($i):
    case 0:
        print "i es igual 0";
        break;
    case 1:
        print "i es igual a 1";
        break;
    case 2:
        print "i es igual a 2";
        break;
    default:
        print "i no es igual a 0, 1 o 2";
 endswitch;
?>
```

declare
La construcción declare es usada para definir directivas de ejecución para un bloque de código. La sintaxis de declare es similar a la de las otras estructuras de control:
declare (directiva) sentencia
Directiva permite asignar el comportamiento del bloque declare. Actualmente una sola directiva es reconocida: la directiva ticks (Consultar más abajo la información sobre la directiva ticks)
La sentencia es lo que se ejecuta -- Como se ejecuta y que efectos secundarios tiene depende de la directiva definida en la directiva.
El constructor declare se puede usar tambien globalmente, afectando a todo el código que le sigue.

```
<?php
// Estos son lo mismo:

// se puede usar este:
declare(ticks=1) {
  // script completo aqui
}

// o este:
declare(ticks=1);
// script completo aqui
?>
```

Ticks
Un "tick" es un evento que ocurre por cada N sentencias de bajo nivel ejecutadas dentro del bloque declare. El valor de N es especificado por ticks=N como directiva dentro de declare.
El evento que ocurre en cada "tick" es especificado usando la función register_tick_function(). Ver el ejemplo más abajo para más detalles. Tener en cuenta que mas de un evento puede ocurrir por cada "tick"

Ejemplo 16-1. Perfilar una sección de código PHP

```
<?php
// A function that records the time when it is called
function profile ($dump = FALSE)
{
  static $profile;

  // Return the times stored in profile, then erase it
  if ($dump) {
     $temp = $profile;
     unset ($profile);
     return ($temp);
  }

  $profile[] = microtime ();
}

// Set up a tick handler
register_tick_function("profile");

// Initialize the function before the declare block
profile ();

// Run a block of code, throw a tick every 2nd statement
declare (ticks=2) {
  for ($x = 1; $x < 50; ++$x) {
     echo similar_text (md5($x), md5($x*$x)), "<br />;";
  }
}

// Display the data stored in the profiler
print_r (profile (TRUE));
?>
```

Este ejemplo perfila el código PHP dentro del bloque 'declare', grabando la hora, una sentencia si y otra no, cuando fue ejecutada. Esta información puede ser usada para encontrar areas en donde el codigo es lento. Este proceso se puede implementar de diferentes maneras: usando "ticks" es más conveniente y facil de implementar.
"Ticks" es una manera muy buena de eliminar errores, implementando simples trabajos en paralelo, I/O en modo desatendido y otras tareas.
return
Si se llama desde una función, return() termina inmediatamente la ejecución de la función y retorna su argumento como valor de la función. return() tambien terminará la ejecución de una sentencia eval()ó un script PHP.
Si el script actual ha sido incluido ó requerido con include() ó require(), el control es transferido al script que llamo al script incluido. Además, si el script actual fue incluido, el valor dado a return() será retornado como el valor de la llamada include(). Si return() es invocado desde el script principal, la ejecución terminara

inmediatamente. Si el script actual fue incluido con las opciones de configuración auto_prepend_file ó auto_append_file, la ejecución terminara inmediatamente. Para más información, consultar Retornando valores.
Nota: Tener en cuenta que ya que return() es un constructor del lenguaje y no una función, los paréntesis alrededor de sus argumentos, son solo necesarios si el argumento contiene una expresion, no se suelen utilizar tan a menudo, cuando retornan una variable.
require()
La sentencia require() incluye y evalua el archivo especificado.
require() incluye y evalua el archivo especificado. Información detallada de como esta inclusión funciona se puede encontrar en la documentación de la función include().
require() y include() son idénticas en todos los aspectos excepto en el modo de actuar ante un error. include() produce un Warning mientras que require() produce un Error Fatal. En otras palabras, no dude en utilizar require() si quiere que un fichero no encontrado cuelgue el procesamiento de la página. include() no se comporta de esta manera, el script seguirá funcionando de todas maneras. Asegurarse que include_path este configurado bien.
Ejemplo 16-2. ejemplos básicos de require()

```
<?php

require 'prepend.php';

require $somefile;

require ('somefile.txt');

?>
```

consultar la documentación de include() para más ejemplos.
Nota: Con anterioridad a PHP 4.0.2, se aplica lo siguiente: require() siempre intentará leer el fichero a incluir, incluso si la linea donde se encuentra require() nunca es ejecutada. Sin embargo, si la linea donde se encuentra require() no es ejecutada, tampoco lo hará el código incluido.
Nota: Puesto que esto es una construcción del lenguaje y no una función, no puede ser llamado usando funciones variables

Aviso
Versiones de PHP para Windows anteriores a 4.3.0, no soportan el acceso remoto a archivos para esta función, no funcionará ni activando siquiera allow_url_fopen.

include()
La sentencia include() incluye y evalúa el archivo especificado.
Esta documentación también se aplica a la función require(). require() y include() son idénticas en todos los aspectos excepto en el modo de actuar ante un error. include() produce un Warning mientras que require() produce un Error Fatal. En otras palabras, no dude en utilizar require() si quiere que un fichero no encontrado cuelgue el procesamiento de la página. include() no se comporta de esta manera, el script seguirá funcionando de todas maneras. Asegurarse que include_path este

configurado bien.
Cuando un fichero es incluido, el código que contiene hereda la variable scope de la linea en donde el include ocurre. Cualquier variable disponible en esa linea en el fichero desde donde se hace la inclusión estará disponible en el fichero incluido a partir de ese momento.
Ejemplo 16-3. Ejemplo básico de la función include()
vars.php

```
<?php

$color = 'green';
$fruit = 'apple';

?>
```

test.php

```
<?php

echo "A $color $fruit"; // A

include 'vars.php';

echo "A $color $fruit"; // A green apple

?>
```

Si la inclusión ocurre dentro de una función en el fichero donde se incluye, todo el código contenido en el fichero incluido se comportará como si hubiese sido definido dentro de esta función.

Ejemplo 16-4. Incluyendo desde funciones

```
<?php

function foo()
{
  global $color;

  include 'vars.php';

  echo "A $color $fruit";
}

/* vars.php is in the scope of foo() so     *
 * $fruit is NOT available outside of this  *
 * scope.  $color is because we declared it *
 * as global.                               */

foo();                    // A green apple
echo "A $color $fruit";  // A green

?>
```

Cuando un fichero es incluido, el intérprete sale del modo PHP y entra en modo HTML al principio del archivo referenciado, y vuelve de nuevo al modo PHP al final. Por esta razón, cualquier código dentro del archivo referenciado que debiera ser ejecutado como código PHP debe ser encerrado dentro de etiquetas válidas de comienzo y fin de PHP.
Si "URL fopen wrappers" esta activada en PHP (como está en la configuración inicial), se puede especificar el fichero que se va a incluir usando una URL (via HTTP u otro mecanismo soportado, consultar Apéndice M) en vez de un fichero local. Si el servidor destino interpreta el fichero destino como código PHP, variables pueden ser mandadas al fichero incluido usando una cadena URL de petición, tal como se hace con HTTP GET. Esto no es lo mismo que incluir un fichero y que este fichero herede las variables del fichero padre; el script es ejecutado en el servidor remoto y el resultado es incluido en en script local.

Aviso
Versiones de PHP para Windows anteriores a 4.3.0, no soportan el acceso remoto a archivos para esta función, no funcionará ni activando siquiera allow_url_fopen.

Ejemplo 16-5. include() a traves de HTTP

```
<?php

/* This example assumes that www.example.com is configured to parse .php
 * files and not .txt files. Also, 'Works' here means that the variables
 * $foo and $bar are available within the included file.*/

// Won't work; file.txt wasn't handled by www.example.com as PHP
include 'http://www.example.com/file.txt?foo=1&bar=2';

// Won't work; looks for a file named 'file.php?foo=1&bar=2' on the
// local filesystem.
include 'file.php?foo=1&bar=2';

// Works.
include 'http://www.example.com/file.php?foo=1&bar=2';

$foo = 1;
$bar = 2;
include 'file.txt';  // Works.
include 'file.php';  // Works.

?>
```

Ya que include() y require() son contructores especiales del lenguaje, se deben de incluir dentro del bloque de una sentencia, si están dentro de un bloque condicional.

Ejemplo 16-6. include() y bloques condicionales

```
<?php

// This is WRONG and will not work as desired.
if ($condition)
   include $file;
else
   include $other;


// This is CORRECT.
if ($condition) {
   include $file;
} else {
   include $other;
}

?>
```

Es posible ejecutar una sentencia return dentro de un archivo incluido para terminar el procesado de ese archivo y volver al archivo de comandos que lo llamó. Tambien es posible retornar valores de ficheros incluidos. Se puede coger el valor

de la llamada "include" como se haría con una función normal.
Nota: En PHP3, return no puede aparecer dentro de un bloque a menos que sea un bloque de función, en el cual return se aplica a esa función y no al archivo completo.
Ejemplo 16-7. include() y return()
return.php
```
<?php

$var = 'PHP';

return $var;

?>
```

noreturn.php
```
<?php

$var = 'PHP';

?>
```

testreturns.php
```
<?php

$foo = include 'return.php';

echo $foo; // prints 'PHP'

$bar = include 'noreturn.php';

echo $bar; // prints 1

?>
```

$bar es igual a 1 porque la inclusión salio bien. Notar la diferencia entre los dos ejemplos anteriores. el primero usa return() dentro del fichero incluido y el segundo no. Otras maneras de incluir ficheros en variables es con fopen(), file() ó usando include()con Funciones de control de salida.
Nota: Puesto que esto es una construcción del lenguaje y no una función, no puede ser llamado usando funciones variables
require_once()
La función require_once() incluye y evalua el fichero especificado durante la ejecución del script. Se comporta de manera similar a require(), con la única diferencia que si el código ha sido ya incluido, no se volverá a incluir. Consultar la documentación de la función require() para obtener más información.
require_once() debería de usarse en casos en los que un mismo fichero puede ser incluido y evaluado más de una vez durante la ejecución de un script, y se quiere estar seguro que se incluye una sola vez para evitar problemas con redefiniciones de funciones, valores de funciones, etc.

Para consultar ejemplos que usen require_once() y include_once(), ver el código de PEAR incluido con las últimas versiones de PHP.
Nota: require_once() fue incorporado en PHP 4.0.1pl2
Nota: El comportamiento de de require_once() y include_once() puede que no sea el esperado en sistemas operativos en los que mayúsculas y minúsculas se traten igual (como en Windows)
Ejemplo 16-8. Con require_once() no importan las mayúsculas y minúsculas en Windows

```
<?php
require_once("a.php"); // this will include a.php
require_once("A.php"); // this will include a.php again on Windows!
?>
```

Aviso
Versiones de PHP para Windows anteriores a 4.3.0, no soportan el acceso remoto a archivos para esta función, no funcionará ni activando siquiera allow_url_fopen.

include_once()
La función include_once() incluye y evalua el fichero especificado durante la ejecución del script. Se comporta de manera similar a include(), con la única diferencia que si el código ha sido ya incluido, no se volverá a incluir.
include_once() debería de usarse en casos en los que, un mismo fichero puede ser incluido y evaluado más de una vez durante la ejecución de un script, y se quiere estar seguro que se incluye una sola vez para evitar problemas con redefiniciones de funciones, valores de funciones, etc.
Para consultar ejemplos que usen include_once() e require_once(), ver el código de PEAR incluido con las últimas versiones de PHP.
Nota: include_once() fue incorporado en PHP 4.0.1pl2
Nota: El comportamiento de de include_once() y require_once() puede que no sea el esperado en sistemas operativos en los que mayúsculas y minúsculas se traten igual (como en Windows)
Ejemplo 16-9. Con include_once() no importan las mayúsculas y minúsculas en Windows

```
<?php
include_once("a.php"); // this will include a.php
include_once("A.php"); // this will include a.php again on Windows!
?>
```

Aviso
Versiones de PHP para Windows anteriores a 4.3.0, no soportan el acceso remoto a archivos para esta función, no funcionará ni activando siquiera allow_url_fopen.

Capítulo 12. Funciones

Funciones definidas por el usuario
Una función se puede definir con la siguiente sintaxis:
Ejemplo 17-1. Psuedo código para demostrar el uso de funciones

```
<?php
function foo ($arg_1, $arg_2, ..., $arg_n)
{
  echo "Funci&oacute;n de ejemplo.\n";
  return $retval;
}
?>
```

Cualquier instrucción válida de PHP puede aparecer en el cuerpo de la función, incluso otras funiones y definiciones de clases.
En PHP3, las funciones deben definirse antes de que se referencien. En PHP4 no existe tal requerimiento.Excepto cuando una función es definida condicionalmente como en los ejemplos siguientes.
Cuando una función es definida condicionalmente como se puede ver en estos dos ejemplos, su definición debe ser procesada antes que sea llamada.
Ejemplo 17-2. Funciones Condicionales

```
<?php

$makefoo = true;

/* We can't call foo() from here
   since it doesn't exist yet,
   but we can call bar() */

bar();

if ($makefoo) {
  function foo ()
  {
   echo "I don't exist until program execution reaches me.\n";
  }
}

/* Now we can safely call foo()
   since $makefoo evaluated to true */

if ($makefoo) foo();

function bar()
{
  echo "I exist immediately upon program start.\n";
}

?>
```

Ejemplo 17-3. Funciones dentro de funciones

```
<?php
function foo()
{
  function bar()
  {
   echo "I don't exist until foo() is called.\n";
  }
}

/* We can't call bar() yet
   since it doesn't exist. */

foo();

/* Now we can call bar(),
   foo()'s processesing has
   made it accessible. */

bar();

?>
```

PHP no soporta la redefinición de funciones previamente declaradas.
Nota: Los nombres de funciones se pueden llamar con mayusculas o minusculas, aunque es una buena costumbre el llamar a las funciones tal y como aparecen en su definición.
PHP3 no soporta un número variable de parámetros, aunque sí soporta parámetros por defecto (ver Valores por defecto de de los parámetros para más información). PHP4 soporta ambos: ver Listas de longitud variable de parámetros y las referencias de las funciones func_num_args(), func_get_arg(), y func_get_args() para más información.
Parámetros de las funciones
La información puede suministrarse a las funciones mediante la lista de parámetros, una lista de variables y/o constantes separadas por comas.
PHP soporta pasar parámetros por valor (el comportamiento por defecto), por referencia, y parámetros por defecto. Listas de longitud variable de parámetros sólo están soportadas en PHP4 y posteriores; ver Listas de longitud variable de parámetros y la referencia de las funciones func_num_args(), func_get_arg(), y func_get_args() para más información. Un efecto similar puede conseguirse en PHP3 pasando un array de parámetros a la función:
Ejemplo 17-4. Pasando matrices a funciones

```
<?php
function takes_array($input)
{
  echo "$input[0] + $input[1] = ", $input[0]+$input[1];
}
?>
```

Pasar parámetros por referencia
Por defecto, los parámetros de una función se pasan por valor (de manera que si cambias el valor del argumento dentro de la función, no se ve modificado fuera de ella). Si deseas permitir a una función modificar sus parámetros, debes pasarlos por referencia.
Si quieres que un parámetro de una función siempre se pase por referencia, puedes anteponer un ampersand (&) al nombre del parámetro en la definición de la función:
Ejemplo 17-5. Pasando parámetros de funciones por referencia

```
<?php
function add_some_extra(&$string)
{
  $string .= ' y algo m&aacute;s.';
}
$str = 'Esto es una cadena, ';
add_some_extra($str);
echo $str;    // Saca 'Esto es una cadena, y algo m&aacute;s.'
?>
```

Parámetros por defecto
Una función puede definir valores por defecto para los parámetros escalares estilo C++:
Ejemplo 17-6. Uso de parámetros por defecto en funciones

```
<?php
function makecoffee ($type = "cappucino")
{
  return "Hacer una taza de $type.\n";
}
echo makecoffee ();
echo makecoffee ("espresso");
?>
```

La salida del fragmento anterior es:
Hacer una taza de cappucino.
Hacer una taza de espresso.
El valor por defecto tiene que ser una expresión constante, y no una variable, miembro de una clase ó llamada a una función.
Destacar que cuando se usan parámetros por defecto, estos tienen que estar a la derecha de cualquier parámetro sin valor por defecto; de otra manera las cosas no funcionarán de la forma esperada. Considera el siguiente fragmento de código:

Ejemplo 17-7. Uso incorrecto de parámetros por defecto en funciones

```
<?php
function makeyogurt ($type = "acidophilus", $flavour)
{
  return "Haciendo un bol de $type $flavour.\n";
}

echo makeyogurt ("mora");  // No funcionar&aacute; de la manera
esperada
?>
```

La salida del ejemplo anterior es:

```
Warning: Missing argument 2 in call to makeyogurt() in
/usr/local/etc/httpd/htdocs/php3test/functest.html on line 41
Haciendo un bol de mora.
```

Y ahora, compáralo con:

Ejemplo 17-8. Uso correcto de parámetros por defecto en funciones

```
<?php
function makeyogurt ($flavour, $type = "acidophilus")
{
  return "Haciendo un bol de $type $flavour.\n";
}

echo makeyogurt ("mora");  // funciona como se esperaba
?>
```

La salida de este ejemplo es:

```
Haciendo un bol de acidophilus mora.
```

Lista de longitud variable de parámetros

PHP4 soporta las listas de longitud variable de parámetros en las funciones definidas por el usuario. Es realmente fácil, usando las funciones func_num_args(), func_get_arg(), y func_get_args().

No necesita de ninguna sintaxis especial, y las listas de parámetros pueden ser escritas en la llamada a la función y se comportarán de la manera esperada.

Devolviendo valores

Los valores se retornan usando la instrucción opcional return. Puede devolverse cualquier tipo de valor, incluyendo listas y objetos.

Ejemplo 17-9. Us0 de return()

```
<?php
function square ($num)
{
  return $num * $num;
}
echo square (4);  // saca '16'.
?>
```

No puedes devolver múltiples valores desde una función, pero un efecto similar se puede conseguir devolviendo una lista.

Ejemplo 17-10. Retornando una matriz para obtener múltiples valores

```
<?php
function small_numbers()
{
  return array (0, 1, 2);
}
list ($zero, $one, $two) = small_numbers();
?>
```

Para retornar una referencia desde una función, se tiene que usar el operador de referencias & tanto en la declaración de la función como en la asignación del valor de retorno a una variable;

Ejemplo 17-11. Retornando una referencia desde una función

```
<?php
function &returns_reference()
{
  return $someref;
}

$newref =& returns_reference();
?>
```

Funciones variables

PHP soporta el concepto de funciones variable, esto significa que si una variable tiene unos paréntesis añadidos al final, PHP buscará una función con el mismo nombre que la evaluación de la variable, e intentará ejecutarla. Entre otras cosas, esto te permite implementar retrollamadas (callbacks), tablas de funciones y demás.

Las funciones varibles no funcionarán con construcciones del lenguaje, tal como echo(), print(), unset(), isset(), empty(), include(), require() y derivados. Se necesitará usar una función propia para utilizar cualquiera de estos constructores como funciones variables.

Ejemplo 17-12. Ejemplo de función variable

```
<?php
function foo()
{
  echo "In foo()<br>\n";
}

function bar($arg = '')
{
  echo "In bar(); argument was '$arg'.<br>\n";
}

// This is a wrapper function around echo
function echoit($string)
{
  echo $string;
}

$func = 'foo';
$func();        // This calls foo()

$func = 'bar';
$func('test');  // This calls bar()

$func = 'echoit';
$func('test');  // This calls echoit()
?>
```

También se puede llamar a un metodo de un objeto usando la característica variable de las funciones.

Ejemplo 17-13. Ejemplo sobre el metodo variable

```
<?php
class Foo
{
  function Var()
  {
     $name = 'Bar';
     $this->$name(); // This calls the Bar() method
  }

  function Bar()
  {
     echo "This is Bar";
  }
}

$foo = new Foo();
$funcname = "Var";
$foo->$funcname();  // This calls $foo->Var()

?>
```

Funciones internas (incorporadas)
PHP tiene incorporadas muchas funciones y construcciones. Existen también funciones que requieren extensiones específicas de PHP para que no fallen con un error fatal del tipo "undefined function". Por ejemplo, para usar funciones image, tal como imagecreatetruecolor(), se necesita compilar PHP con soporte para GD. O para usar mysql_connect() se necesita compilar PHP con soporte para MySQL. Existen muchas funciones en el núcleo de PHP que se incluyen en cada version de PHP, como las funciones string y variable. Una llamada a la función phpinfo() ó get_loaded_extensions() mostrará que extensiones están cargadas en tu versión de PHP. Tener tambien en cuenta que muchas extensiones se encuentran activadas por defecto y que el manual de PHP se encuentra dividido en partes, según estas extensiones. Vea los capítulos configuración, instalación y los capitulos sobre cada extensión, para obtener información sobre como configurar vuestro PHP
La explicación de como leer e intrerpretar un prototipo de función se encuentra en la sección del manual titulada como leer la definición de una función. Es importante entender que devuelve una función ó si la función trabaja directamente en el valor entregado a la misma. Por ejemplo, str_replace() devuelve una cadena modificada mientras que usort() trabaja directamente en el valor entregado a la misma. Cada página del manual contiene información específica sobre las diferentes funciones existentes, parametros que utilizan, valores devueltos, cambios de comportamiento, etc. Es importante conocer estas diferencias para poder escribir correctamente código PHP.

Capítulo 13. Clases y Objetos (PHP 4)

Class

Una clase es una colección de variables y funciones que trabajan con éstas

variables. Una clase es definida usando la siguiente sintaxis:

```
<?php
class Carrito {
  var $items;  // Items en nuestro carrito de compras

  // Agregar $num articulos de $artnr al carrito

  function agregar_item($artnr, $num) {
    $this->items[$artnr] += $num;
  }

  // Tomar $num articulos de $artnr del carrito

  function retirar_item($artnr, $num) {
    if ($this->items[$artnr] > $num) {
      $this->items[$artnr] -= $num;
      return true;
    } elseif ($this->items[$artnr] == $num) {
      unset($this->items[$artnr]);
      return true;
    } else {
      return false;
    }
  }
}
?>
```

Esto define una clase llamada Carrito que consiste de una matriz asociativa de artículos en el carrito y dos funciones para agregar y eliminar elementos del carrito.

Aviso

NO es posible separar la definición de una clase en varios archivos. Asimismo NO es posible separar la definición de una clase en bloques PHP diferentes, a menos que la separación sea al interior de una declaración de método. Lo siguiente no funciona:

```
<?php
class prueba {
?>
<?php
  function prueba() {
     print 'Bien';
  }
}
?>
```

Sin embargo, lo siguiente es permitido:

```
<?php
class prueba {
  function prueba() {
     ?>
      <?php
     print 'Bien';
  }
}
?>
```

Las siguientes notas de precaución son válidas para PHP 4.

Atención
El nombre stdClass es usado internamente por Zend y es reservado. No puede tener una clase con el nombre stdClass en PHP.
Atención
Los nombres de función __sleep y __wakeup son mágicos en las clases PHP. No puede tener funciones con éstos nombres en cualquiera de sus clases a menos que desee usar la funcionalidad mágica asociada con ellas. Vea más información a continuación.
Atención
PHP reserva todos los nombres de función que comienzan con __ como mágicos. Se recomienda que no use nombres de función con __ en PHP a menos que desee usar alguna funcionalidad mágica documentada.

En PHP 4, sólo se permiten inicializadores constantes para variables var. Para inicializar variables con valores no-constantes, necesita una función de inicialización que sea llamada automáticamente cuando un objeto es construido a partir de la clase. Tal función es llamada constructora (vea más información a continuación).

```
<?php
class Carrito {
  /* Ninguna de estas expresiones funciona en PHP 4. */
  var $fecha_hoy = date("Y-m-d");
  var $nombre = $primer_nombre;
  var $duenyo = 'Fred ' . 'Jones';
  /* Aunque, las matrices que contienen valores constantes funcionan
*/
  var $items = array("VCR", "TV");
}

/* Asi es como debe declararse. */
class Carrito {
  var $fecha_hoy;
  var $nombre;
  var $duenyo;
  var $items = array("VCR", "TV");

  function Carrito() {
     $this->fecha_hoy = date("Y-m-d");
     $this->nombre = $GLOBALS['primer_nombre'];
     /* etc. . . */
  }
}
?>
```

Las clases son tipos, es decir, son planos usados para variables reales. Necesita crear una variable del tipo deseado con el operador new.

```
<?php
$carrito = new Carrito;
$carrito->agregar_item("10", 1);

$otro_carrito = new Carrito;
$otro_carrito->agregar_item("0815", 3);
?>
```

Esto crea los objetos $carrito y $otro_carrito, ambos de la clase Carrito. La función agregar_item() del objeto $carrito es llamada para agregar 1 item del artículo número 10 al $carrito. Se agregan 3 items del artículo número 0815 al $otro_carrito.

Ambos, $carrito y $otro_carrito, tienen funciones agregar_item(), retirar_item() y una variable items. Estas son variables y funciones diferentes. Puede pensar sobre los objetos como algo similar a las directorios en un sistema de archivos. En un sistema de archivos es posible tener dos archivos LEAME.TXT diferentes, siempre y cuando estén en directorios diferentes. Tal y como con los directorios, en donde es necesario escribir las rutas de nombres completas para llegar a cada archivo a partir del directorio del nivel superior, es necesario especificar el nombre completo de la función que desea llamar: en términos de PHP, el directorio de nivel superior sería el espacio de nombres global, y el separador de ruta sería ->. De tal modo que los nombres $carrito->items y $otro_carrito->items hacen referencia a dos variables diferentes. Note que la variable se llama $carrito->items, no

$carrito->$items, es decir, un nombre de variable en PHP solo tiene un único signo de dólar.

```
<?php
// correcto, un solo $
$carrito->items = array("10" => 1);

// invalido, ya que $carrito->$items se convierte en $carrito->""
$carrito->$items = array("10" => 1);

// correcto, pero puede o no ser lo que se busca:
// $carrito->$mivar se convierte en $carrito->items
$mivar = 'items';
$carrito->$mivar = array("10" => 1);
?>
```

Al interior de una definición de clase, no se conoce el nombre bajo el que el objeto será accesible en su programa: en el momento en que la clase Carrito fue escrita, no se conocía que el objeto se llamaría $carrito u $otro_carrito más adelante. Por lo tanto, no es posible escribir $carrito->items al interior de la clase Carrito. En su lugar, para poder acceder a sus propias funciones y variables desde el interior de una clase, es posible usar la pseudo-variable $this, la cual puede leerse como 'mi propio' o 'el objeto actual'. Por lo tanto, '$this->items[$num_art] += $num' puede leerse como 'agregar $num al contador $num_art de mi propia matriz de items', o 'agregar $num al contador $num_art de la matriz de items al interior del objeto actual'.

Nota: Usualmente, la pseudo-variable $this no es definida si el método en el que se encuentra es llamado estáticamente. Sin embargo, esta no es una regla estricta: $this se define si un método es llamado estáticamente desde el interior de otro objeto. En este caso, el valor de $this es aquél del objeto que hace la llamada. Esto se ilustra en el siguiente ejemplo:

```php
<?php
class A
{
  function foo()
  {
     if (isset($this)) {
        echo '$this se define (';
        echo get_class($this);
        echo ")\n";
     } else {
        echo "\$this no se define.\n";
     }
  }
}

class B
{
  function bar()
  {
     A::foo();
  }
}

$a = new A();
$a->foo();
A::foo();
$b = new B();
$b->bar();
B::bar();
?>
```

El resultado del ejemplo seria:

```
$this se define (a)
$this no se define.
$this se define (b)
$this no se define.
```

extends

Con frecuencia es necesario tener clases con variables y funciones similares a otra clase existente. De hecho, es una buena práctica definir una clase genérica que pueda ser usada en todos sus proyectos y adaptar ésta clase para las necesidades de cada uno de sus proyectos específicos. Para facilitar esto, las clases pueden ser extensiones de otras clases. La clase extendida o derivada tiene todas las variables y funciones de la clase base (esto es llamado 'herencia', a pesar del hecho de que nadie muera) y lo que se agregue en la definición extendida. No es posible substraer de una clase, es decir, remover la definición de cualquier función o variable existente. Una clase extendida siempre depende de una clase base única, lo que quiere decir que no se soporta herencia múltiple. Las clases son extendidas usando la palabra clave 'extends'.

```
<?php
class Carrito_Con_Nombre extends Carrito {
  var $duenyo;

  function definir_duenyo ($nombre) {
     $this->duenyo = $nombre;
  }
}
?>
```
Esto define una clase Carrito_Con_Nombre que tiene todas las variables y funciones de Carrito, más una variable adicional $duenyo y una función adicional definir_duenyo(). Se crea un carrito con nombre en la forma usual y ahora es posible definir y obtener el dueño del carrito. Aun es posible usar las funciones normales de un carrito sobre los carritos con nombre:
```
<?php
$carrito_n = new Carrito_Con_Nombre;  // Crear un carrito con
nombre
$carrito_n->definir_duenyo("kris");  // Nombrar el carrito
print $carrito_n->duenyo;           // imprimir el nombre del duenyo
$carrito_n->agregar_item("10", 1);    // (funcionalidad heredada de
carrito)
?>
```
Esto también se conoce como una relación "padre-hijo". Se crea una clase, padre, y se usa extends para crear una clase basada en la clase padre: la clase hija. Es posible incluso usar esta nueva clase hija y crear otra clase basada en esta clase hija.
Nota: ¡Las clases deben ser definidas antes de ser usadas! Si desea que la clase Carrito_Con_Nombre extienda la clase Carrito, tendrá que definir la clase Carrito primero. Si desea crear otra clase llamada Carrito_amarillo_con_nombre basada en la clase Carrito_Con_Nombre, debe definir Carrito_Con_Nombre primero. Resumiendo: el orden en que se definen las clases es importante.
Constructores
Los constructores son funciones en una clase que son llamadas automáticamente cuando se crea una nueva instancia de una clase con new. Una función se convierte en constructor cuando tiene el mismo nombre que la clase. Si una clase no tiene constructor, el constructor de la clase base es llamado, si existe.
```
<?php
class Auto_Carrito extends Carrito {
  function Auto_Carrito() {
     $this->agregar_item("10", 1);
  }
}
?>
```
Esto define una clase Auto_Carrito que es un Carrito más un constructor que inicializa el carrito con un item del número de artículo "10" cada vez que un nuevo Auto_Carrito se crea con "new". Los constructores pueden recibir argumentos y tales argumentos pueden ser opcionales, lo que los hace mucho más útiles. Para poder usar aun la clase sin parámetros, todos los parámetros deben ser opcionales, al proveer valores predeterminados.

```
<?php
class Constructor_Carrito extends Carrito {
  function Constructor_Carrito($item = "10", $num = 1) {
     $this->agregar_item ($item, $num);
  }
}

// Comprar lo mismo de antes.
$carrito_predeterminado = new Constructor_Carrito;

// Comprar esta vez en serio...
$carrito_diferente = new Constructor_Carrito("20", 17);
?>
```

También puede usar el operador @ para callar los errores que ocurren en el constructor, p.ej. @new.

```
<?php
class A
{
  function A()
  {
     echo "Soy el constructor de A.<br />\n";
  }

  function B()
  {
     echo "Soy una funci&oacute;n regular llamada B en la clase A.<br
/>\n";
     echo "No soy un constructor en A.<br />\n";
  }
}

class B extends A
{
}

// Esto llama a B() como un constructor
$b = new B;
?>
```

La función B() en la clase A se convertirá de repente en un constructor en la clase B, aun cuando nunca fue esa la intención. A PHP 4 no le importa si la función está siendo definida en la clase B, o si ha sido heredada.

Atención
PHP 4 no llama constructores de la clase base automáticamente desde un constructor de una clase derivada. Es su responsabilidad propagar la llamada a constructores más arriba en la jerarquía cuando sea apropiado.

Los destructores son funciones que son llamadas automáticamente cuando un objeto es destruido, ya sea con unset() o simplemente al finalizarse su contexto. No hay destructores en PHP. Es posible usar register_shutdown_function() en su

lugar para simular la mayoría de efectos de los destructores.

Operador de Resolución de Contexto (::)

Atención
Lo siguiente es válido únicamente para PHP 4 y versiones posteriores.

Algunas veces es útil referirse a funciones y variables en clases base o referirse a funciones en clases que no tienen aun alguna instancia. El operador :: es usado en tales casos.

```
<?php
class A {
  function ejemplo() {
     echo "Soy la funci&oacute;n original A::ejemplo().<br />\n";
  }
}

class B extends A {
  function ejemplo() {
     echo "Soy la funci&oacute;n redefinida B::ejemplo().<br />\n";
     A::ejemplo();
  }
}

// no hay un objeto de la clase A.
// esto imprime
//  Soy la funcion original A::ejemplo().<br />
A::ejemplo();

// crear un objeto de clase B.
$b = new B;

// esto imprime
//  Soy la funcion redefinida B::ejemplo().<br />
//  Soy la funcion original A::ejemplo().<br />
$b->ejemplo();
?>
```

El ejemplo anterior llama la función ejemplo() en la clase A, pero no hay un objeto de la clase A, así que no podemos escribir $a->ejemplo() o algo semejante. En su lugar llamamos ejemplo() como una 'función de clase', es decir, una función de la clase misma, no de un objeto de tal clase.

Existen funciones de clase, pero no existen variables de clase. De hecho, no hay un objeto en absoluto al momento de la llamada. Por lo tanto, una función de clase no puede usar variables del objeto (pero puede usar variables locales y globales), y quizás no pueda usar $this en absoluto.

En el ejemplo anterior, la clase B redefine la función ejemplo(). La definición original en la clase A es cubierta y ya no se encuentra disponible, a menos que haga referencia específicamente a la implementación de ejemplo() en la clase A usando el operador ::. Escriba A::ejemplo() para hacerlo (en realidad, debería usar parent::ejemplo(), como se muestra en la siguiente sección).

En este contexto, hay un objeto actual y puede tener variables de objeto. Por lo tanto, cuando se usa DESDE EL INTERIOR de una función de objeto, es posible

usar $this y variables de objeto.
parent
Es posible que se encuentre escribiendo código que hace referencia a variables y funciones de las clases base. Esto es particularmente cierto si su clase derivada es una refinación o especialización del código en su clase base.
En lugar de usar el nombre literal de la clase base en su código, debería usar el nombre especial parent, el cual hace referencia al nombre de su clase base tal y como se entrega en la declaración extends de su clase. Al hacer esto, evita usar el nombre de su clase base en más de un lugar. Llegado el caso de que su árbol de jerarquía cambie durante la implementación, el cambio se puede efectuar con facilidad simplemente modificando la declaración extends de su clase.

```
<?php
class A {
  function ejemplo() {
      echo "Soy A::ejemplo() y ofrezco funcionalidad b&aacute;sica.<br
/>\n";
  }
}

class B extends A {
  function ejemplo() {
      echo "Soy B::ejemplo() y ofrezco funcionalidad adicional.<br
/>\n";
      parent::ejemplo();
  }
}

$b = new B;

// Esto hace la llamada a B::ejemplo(), la cual llama a su vez a
A::ejemplo().
$b->ejemplo();
?>
```

parent
Es posible que se encuentre escribiendo código que hace referencia a variables y funciones de las clases base. Esto es particularmente cierto si su clase derivada es una refinación o especialización del código en su clase base.
En lugar de usar el nombre literal de la clase base en su código, debería usar el nombre especial parent, el cual hace referencia al nombre de su clase base tal y como se entrega en la declaración extends de su clase. Al hacer esto, evita usar el nombre de su clase base en más de un lugar. Llegado el caso de que su árbol de jerarquía cambie durante la implementación, el cambio se puede efectuar con facilidad simplemente modificando la declaración extends de su clase.

```
<?php
class A {
   function ejemplo() {
       echo "Soy A::ejemplo() y ofrezco funcionalidad b&aacute;sica.<br
/>\n";
   }
}

class B extends A {
   function ejemplo() {
       echo "Soy B::ejemplo() y ofrezco funcionalidad adicional.<br
/>\n";
       parent::ejemplo();
   }
}

$b = new B;

// Esto hace la llamada a B::ejemplo(), la cual llama a su vez a
A::ejemplo().
$b->ejemplo();
?>
```

Seriación de objetos, objetos en sesiones

Nota: En PHP 3, los objetos perdían su asociación de clase a lo largo del proceso de seriación y decodificación. La variable resultante es de tipo objeto, pero no tiene clase ni métodos, de modo que es inútil (se ha convertido en algo como una matriz con una sintaxis curiosa).

Atención
La siguiente información es válida para PHP >= 4 únicamente.

serialize() devuelve una cadena que contiene una representación tipo secuencia-de-bytes de cualquier valor que pueda ser almacenado en PHP. unserialize() puede causar que éstas cadenas recreen los valores de variable originales. Usando el método de seriación para guardar un objeto guardará todas las variables en un objeto. Las funciones en un objeto no se guardarán, sólo el nombre de la clase.

Para poder usar unserialize() con un objeto, la clase de ese objeto necesita ser definida. Es decir, si tiene un objeto $a de la clase A en pagina1.php y codifica ésta variable, obtendrá una cadena que hace referencia a la clase A y contiene todos los valores de variables contenidas en $a. Si desea tener la capacidad de revertir el proceso de seriación en pagina2.php, recreando $a de la clase A, la definición de la clase A debe estar presente en pagina2.php. Esto puede conseguirse por ejemplo almacenando la definición de la clase A en un archivo de inclusión e incluyéndolo tanto en pagina1.php como en pagina2.php.

```
<?php
// clase_a.inc:

  class A {
    var $uno = 1;

    function mostrar_uno() {
       echo $this->uno;
    }
  }

// pagina1.php:

  include("clase_a.inc");

  $a = new A;
  $s = serialize($a);
  // almacenar $s en alguna parte en donde pagina2.php lo pueda
encontrar.
  $fp = fopen("almacenamiento", "w");
  fwrite($fp, $s);
  fclose($fp);

// pagina2.php:

  // esto es necesaria para revertir la seriacion apropiadamente.
  include("clase_a.inc");

  $s = implode("", @file("almacenamiento"));
  $a = unserialize($s);

  // ahora use la funcion mostrar_uno() del objeto $a.
  $a->mostrar_uno();
?>
```

Si está usando sesiones y usa session_register() para registrar objetos, éstos objetos son seriados automáticamente al final de cada página PHP, y son decodificados de vuelta automáticamente en cada una de las siguientes páginas. Esto quiere decir, básicamente, que tales objetos pueden aparecer en cualquiera de sus páginas una vez hacen parte de su sesión.

Es bastante recomendable que incluya las definiciones de clase de todos esos objetos registrados en todas sus páginas, incluso si no va a usar realmente éstas clases en todas sus páginas. Si no lo hace y un objeto está siendo decodificado sin que su definición de clase esté presente, perderá su asociación de clase y se convertirá en un objeto de la clase stdClass sin ninguna función disponible, es decir, se hará prácticamente inútil.

De modo que si en el ejemplo anterior $a se hacía parte de una sesión ejecutando session_register("a"), debería incluírse el archivo clase_a.inc en todas sus páginas, no sólo en pagina1.php y pagina2.php.

Las funciones mágicas __sleep y __wakeup

serialize() revisa si su clase tiene una función con el nombre mágico __sleep. De ser así, esa función es ejecutada antes de cualquier intento de seriación. Puede limpiar el objeto y su intención es que devuelva una matriz con los nombres de todas las variables de ese objeto que deberían ser seriadas.
El uso planeado para __sleep es cerrar todas las conexiones de bases de datos que pueda tener el objeto, aplicando datos pendientes o realizando tareas similares de limpieza. Asimismo, la función resulta útil si tiene objetos bastante grandes que no necesitan ser guardados en su totalidad.
De forma semejante, unserialize() revisa por la presencia de una función con el nombre mágico __wakeup. Si está presente, ésta función puede reconstruir cualquier recurso que el objeto pueda tener.
El uso planeado para __wakeup es reestablecer cualquier conexión con bases de datos que hayan podido perderse durante la seriación y realizar otras tareas de reinicialización.
Las referencias al interior del constructor
Crear referencias al interior del constructor puede llevar a resultados confusos. Esta sección tipo-tutorial le ayuda a evitar problemas.

```
<?php
class Foo {
  function Foo($nombre) {
     // crear una referencia al interior de la matriz global $refglobal
     global $refglobal;
     $refglobal[] = &$this;
     // definir el nombre al valor pasado
     $this->definirNombre($nombre);
     // e imprimirlo
     $this->imprimirNombre();
  }

  function imprimirNombre() {
     echo "<br />", $this->nombre;
  }

  function definirNombre($nombre) {
     $this->nombre = $nombre;
  }
}
?>
```

Revisemos si existe una diferencia entre $bar1, que ha sido creado usando el operador de copia = y $bar2 que ha sido creado usando el operador de referencia =&...

```
<?php
$bar1 = new Foo('definido en el constructor');
$bar1->imprimirNombre();
$refglobal[0]->imprimirNombre();

/* salida:
definido en el constructor
definido en el constructor
definido en el constructor */

$bar2 =& new Foo('definido en el constructor');
$bar2->imprimirNombre();
$refglobal[1]->imprimirNombre();

/* salida:
definido en el constructor
definido en el constructor
definido en el constructor */
?>
```

Aparentemente no hay ninguna diferencia, pero en realidad hay una bastante importante: $bar1 y $refglobal[0] _NO_ son referenciados, NO son la misma variable. Esto se debe a que "new" no devuelve una referencia por defecto, en su lugar devuelve una copia.

Nota: No existe una pérdida de rendimiento (ya que desde PHP 4 se usa el conteo de referencias) al devolver copias en lugar de referencias. Al contrario, con frecuencia es mejor trabajar simplemente con copias en lugar de referencias, ya que crear referencias toma cierto tiempo mientras que crear copias prácticamente no toma nada de tiempo (a menos que ninguna de ellas sea una matriz u objeto grande y una de ellas se modifique y luego las otras subsecuentemente, entonces sería buena idea usar referencias para modificarlas todas al mismo tiempo).

Para probar lo que se dice más arriba, veamos el siguiente código.

```
<?php
// ahora cambiaremos el nombre. que espera que pase?
// puede que espere que tanto $bar1 como $refglobal[0] cambien sus
nombres...
$bar1->definirNombre('definido desde afuera');

// como se ha mencionado antes, ese no es el caso.
$bar1->imprimirNombre();
$refglobal[0]->imprimirNombre();

/* salida:
definido desde afuera
definido en el constructor */

// veamos que cambia entre $bar2 y $refglobal[1]
$bar2->definirNombre('definido desde afuera');

// por suerte, no solo son iguales, son la misma variable, de modo que
// $bar2->nombre y $refglobal[1]->nombre son el mismo tambien
$bar2->imprimirNombre();
$refglobal[1]->imprimirNombre();

/* salida:
definido desde afuera
definido desde afuera */
?>
```

Otro ejemplo final, intente entenderlo.

```php
<?php
class A {
  function A($i) {
     $this->valor = $i;
     // intente descubrir porque no necesitamos una referencia aqui
     $this->b = new B($this);
  }

  function crearRef() {
     $this->c = new B($this);
  }

  function echoValor() {
     echo "<br />","clase ",get_class($this),': ',$this->valor;
  }
}


class B {
  function B(&$a) {
     $this->a = &$a;
  }

  function echoValor() {
     echo "<br />","clase ",get_class($this),': ',$this->a->valor;
  }
}

// intente entender porque usar una simple copia produciria
// un resultado no deseado en la linea marcada con *
$a =& new A(10);
$a->crearRef();

$a->echoValor();
$a->b->echoValor();
$a->c->echoValor();

$a->valor = 11;

$a->echoValor();
$a->b->echoValor(); // *
$a->c->echoValor();

?>
```

El resultado del ejemplo seria:

```
clase A: 10
clase B: 10
clase B: 10
clase A: 11
clase B: 11
clase B: 11
```

Comparación de objetos

En PHP 4, los objetos son comparados en una forma muy simple: Dos instancias de objeto son iguales si tienen los mismos atributos y valores, y son instancias de la misma clase. Reglas similares se aplican cuando se comparan dos objetos usando el operador de identidad (===).

Ejemplo de comparación de objetos en PHP 4

```
<?php
function bool_a_cadena($bool) {
  if ($bool === false) {
        return 'FALSE';
  } else {
        return 'TRUE';
  }
}

function compararObjetos(&$o1, &$o2) {
  echo 'o1 == o2 : '.bool_a_cadena($o1 == $o2)."\n";
  echo 'o1 != o2 : '.bool_a_cadena($o1 != $o2)."\n";
  echo 'o1 === o2 : '.bool_a_cadena($o1 === $o2)."\n";
  echo 'o1 !== o2 : '.bool_a_cadena($o1 !== $o2)."\n";
}

class Bandera {
  var $bandera;

  function Bandera($bandera=true) {
        $this->bandera = $bandera;
  }
}

class BanderaCambiante extends Bandera {

  function encender() {
     $this->bandera = true;
  }

  function apagar() {
     $this->bandera = false;
  }
}
```

```
$o = new Bandera();
$p = new Bandera(false);
$q = new Bandera();

$r = new BanderaCambiante();

echo "Comparar instancias creadas con los mismos par&aacute;metros\n";
compararObjetos($o, $q);

echo "\nComparar instancias creadas con par&aacute;metros diferentes\n";
compararObjetos($o, $p);

echo "\nComparar una instancia de una clase padre con una de una subclase\n";
compararObjetos($o, $r);
?>
```

El resultado del ejemplo seria:

```
Comparar instancias creadas con los mismos par&aacute;metros
o1 == o2 : TRUE
o1 != o2 : FALSE
o1 === o2 : TRUE
o1 !== o2 : FALSE

Comparar instancias creadas con par&aacute;metros diferentes
o1 == o2 : FALSE
o1 != o2 : TRUE
o1 === o2 : FALSE
o1 !== o2 : TRUE

Comparar una instancia de una clase padre con una de una subclase
o1 == o2 : FALSE
o1 != o2 : TRUE
o1 === o2 : FALSE
o1 !== o2 : TRUE
```

Que es la salida que podemos esperar dadas las reglas de comparación descritas anteriormente. Solo las instancias con los mismos valores para sus atributos y de la misma clase son consideradas iguales e idénticas.
Incluso en los casos en donde tenemos composición de objetos, se aplican las mismas reglas de comparación. En el ejemplo siguiente creamos una clase contenedora que almacena una matriz asociativa de objetos Bandera.

```
<?php
class ConjuntoBanderas {
  var $conjunto;

  function ConjuntoBanderas($matrizBanderas = array()) {
     $this->conjunto = $matrizBanderas;
  }

  function agregarBandera($nombre, $bandera) {
```

```
        $this->conjunto[$nombre] = $bandera;
    }

    function eliminarBandera($nombre) {
        if (array_key_exists($nombre, $this->conjunto)) {
            unset($this->conjunto[$nombre]);
        }
    }
}


$u = new ConjuntoBanderas();
$u->agregarBandera('bandera1', $o);
$u->agregarBandera('bandera2', $p);
$v = new ConjuntoBanderas(array('bandera1'=>$q, 'bandera2'=>$p));
$w = new ConjuntoBanderas(array('bandera1'=>$q));

echo "\nObjetos compuestos u(o,p) y v(q,p)\n";
compararObjetos($u, $v);

echo "\nu(o,p) y w(q)\n";
compararObjetos($u, $w);
?> El resultado del ejemplo seria:

Objetos compuestos u(o,p) y v(q,p)
o1 == o2 : TRUE
o1 != o2 : FALSE
o1 === o2 : TRUE
o1 !== o2 : FALSE

u(o,p) y w(q)
o1 == o2 : FALSE
o1 != o2 : TRUE
o1 === o2 : FALSE
o1 !== o2 : TRUE
```