Anexo A

PHP

¿Qué es PHP?

PHP (acrónimo de "PHP: Hypertext Preprocessor") es un lenguaje interpretado de alto nivel embebido en páginas HTML y ejecutado en el servidor.

Una respuesta corta y concisa, pero ¿que significa realmente? Un ejemplo nos aclarará las cosas:

Ejemplo 1-1. Un ejemplo introductorio

```
<html>
<head>
<title>Ejemplo PHP</title>
</head>
<body>
<?php echo "Hola, este es un ejemplo con PHP!"; ?>
</body>
</html>
```

Podemos ver que no es lo mismo que un script CGI 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 (introducido) en el mismo, que producirá cierta salida (en nuestro ejemplo, producir 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 que código ha producido el resultado recibido. El servidor web puede ser incluso configurado para que procese todos los ficheros HTML con PHP.

¿Qué se puede hacer con PHP?

Al nivel más básico, PHP puede hacer cualquier cosa que se pueda hacer con un script CGI, como procesar la información de formularios, generar páginas con contenidos dinámicos, o mandar y recibir cookies.

Quizás la característica más potente y destacable de PHP es su soporte para una gran cantidad de bases de datos. Escribir un interfaz vía web para una base de datos es una tarea simple con PHP. Las siguientes bases de datos están soportadas actualmente:

Adabas D Ingres Oracle (OCI7 and OCI8)
dBase InterBase PostgreSQL
Empress FrontBase Solid
FilePro mSQL Sybase
IBM DB2 MySQL Velocis
Informix ODBC Unix dbm

PHP también soporta el uso de otros servicios que usen protocolos como IMAP, SNMP, NNTP, POP3, HTTP y derivados. También se pueden abrir sockets de red directos (raw sockets) e interactuar con otros protocolos.

Corta historia de PHP

PHP fue concebido en otoño de 1994 por Rasmus Lerdorf (mailto:rasmus@php.net). Las primeras versiones no distribuidas al público fueron usadas en un sus páginas web para mantener un control sobre quien consultaba su currículum. La primera versión disponible para el público a principios de 1995 fue conocida como "Herramientas para paginas web personales"(Personal Home Page Tools). Consistían en un analizador sintáctico muy simple que solo entendía unas cuantas macros y una serie de utilidades comunes en las páginas web de entonces, un libro de visitas, un contador y otras pequeñas cosas. El analizador sintáctico fue reescrito a mediados de 1995 y fue nombrado PHP/FI versión 2. FI viene de otro programa que Rasmus había escrito y que procesaba los datos de formularios. Así que combinó las "Herramientas para paginas web personales", el "intérprete de formularios", añadió soporte para mSQL y PHP/FI vio la luz. PHP/FI creció a gran velocidad y la gente empezó a contribuir en el código.

Es difícil dar estadísticas exactas, pero se estima que a finales de 1996 PHP/FI se estaba usando al menos en 15.000 páginas web alrededor del mundo. A mediados de 1997 este número había crecido a mas de 50.000. A mediados de 1997 el desarrollo del proyecto sufrió un profundo cambio, dejó de ser un proyecto personal de Rasmus, al cual habían ayudado un grupo de usuarios y se convirtió en un proyecto de grupo mucho más organizado. El analizador sintáctico se rescribió desde el principio por Zeev Suraski y Andi Gutmans y este nuevo analizador estableció las

bases para PHP versión 3. Gran cantidad de código de PHP/FI fue portado a PHP3 y otra gran cantidad fue escrito completamente de nuevo.

Hoy en día (finales 1999), tanto PHP/FI como PHP3 se distribuyen en un gran número de productos comerciales tales como el servidor web "C2's StrongHold"y Redhat Linux. Una estimación conservativa basada en estadísticas de NetCraft (http://www.netcraft.com/) (ver también Estudio de NetCraft sobre servidores web (http://www.netcraft.com/survey/)), es que más de 1.000.000 de servidores alrededor del mundo usan PHP. Para hacernos una idea, este número es mayor que el número de servidores que utilizan el "Netscape's Enterprise server" en Internet.

A la vez que todo esto está pasando, el trabajo de desarrollo de la próxima generación de PHP está en marcha. Esta versión utiliza el potente motor de scripts Zend (http://www.zend.com/) para proporcionar altas prestaciones, así como soporta otros servidores web, además de apache, que corren PHP como módulo nativo.

Instalación

Bajándose la última versión

El código fuente y las distribuciones binarias para algunas plataformas (incluido Windows) se pueden encontrar en http://www.php.net/.

Instalación en sistemas UNIX

Esta sección le guiará a través de la configuración e instalación del PHP. Conocimientos y software necesarios:

- Habilidades básicas en UNIX (ser capaz de manejar el "make"y un compilador de C)
- Un compilador ANSI de C
- Un servidor web

Instrucciones Rápidas de Instalación (Versión Módulo de Apache)

1. gunzip apache_1.3.x.tar.gz
2. tar xvf apache_1.3.x.tar
3. gunzip php-3.0.x.tar.gz
4. tar xvf php-3.0.x.tar
5. cd apache_1.3.x
6. ./configure -prefix=/www
7. cd ../php-3.0.x
8. ./configure -with-mysql -with-apache=../apache_1.3.x -enable-track-vars
9. make
10. make install
11. cd ../apache_1.3.x

12. ./configure -prefix=/www -activate-module=src/modules/php3/libphp3.a
13. make
14. make install

En lugar de este paso quizás prefiera simplemente copiar el binario httpd encima del binario existente. Si lo hace, asegúrese antes de cerrar su servidor.

15. cd ../php-3.0.x
16. cp php3.ini-dist /usr/local/lib/php3.ini

Puede editar el archivo /usr/local/lib/php3.ini para ajustar opciones del PHP. Si prefiere tenerlo en otro sitio, utilice -with-config-file-path=/path en el paso 8.

17. Edite su archivo httpd.conf o srm.conf y añada:

AddType application/x-httpd-php3 .php3

Puede elegir la extensión que desee aquí. .php3 es simplemente nuestra sugerencia.

18. Utilice su método habitual para iniciar el servidor Apache (debe detener y reiniciar el servidor, no solamente hacerlo recargarse usando una señal HUP o USR1.)

Configuración

Hay dos maneras de configurar el PHP.

• Utilizando el script de "setup"que viene con el PHP. Este script le hace una serie de preguntas (casi como el script "install"del PHP/FI 2.0) y ejecuta el "configure"al final. Para ejecutar este script, escriba **./setup**.

Este script también creará un archivo llamado "do-conf", que contendrá las opciones pasadas a la configuración. Puede editar este archivo para cambiar algunas opciones sin tener que re-ejecutar el "setup". Escriba luego **./do-conf** para ejecutar la configuración con las nuevas opciones.

• Ejecutar el "configure"a mano. Para ver las opciones de que dispone, escriba **./configure –help**.

Los detalles sobre las distintas opciones de configuración son listados a continuación.

Módulo del Apache

Para configurar el PHP como módulo de Apache, responda "yes" a "Build as an Apache module?"(la opción

-with-apache=*DIR* es la que lo configura) y especifique el directorio base de la distribución de Apache. Si ha desempacado el Apache en /usr/local/www/apache_1.2.4, este será su directorio base de la distribución de Apache.

El directorio por defecto es /usr/local/etc/httpd.

Módulo fhttpd

Para configurar el PHP como módulo fhttpd, responda "yes"a "Build as an fhttpd module?"(la opción

-with-fhttpd=*DIR* es la que lo configura) y especifique el directorio base del fuente del fhttpd. El directorio por defecto es /usr/local/src/fhttpd. Si está ejecutando fhttpd, configurar PHP como módulo le dará mejor rendimiento, más control y capacidad de ejecución remota.

CGI versión

El valor por defecto es configurar el PHP como programa CGI. Si está ejecutando un servidor web para el que el PHP tiene soporte como módulo, debería elegir dicha solución por motivos de rendimiento. Sin embargo, la versión CGI permite a los usuarios del Apache el ejecutar distintas páginas con PHP bajo distintos identificadores de usuario. Por favor, asegúrese de haber leído el capítulo sobre Seguridad si va a ejecutar el PHP como CGI.

Opciones de soporte para Base de Datos

El PHP tiene soporte nativo para bastantes bases de datos (así como para ODBC):

Adabas D

-with-adabas=*DIR*

Compila con soporte para Adabas D. El parámetro es el directorio de instalación de Adabas D y por defecto vale /usr/local/adabasd.

Página de Adabas (http://www.adabas.com/)

dBase

-with-dbase

Habilita el soporte integrado para DBase. No se precisan librerías externas.

filePro

-with-filepro

Habilita el soporte integrado de sólo lectura para filePro. No se precisan librerías externas.

mSQL

-with-msql=*DIR*

Habilita el soporte para mSQL. El parámetro es el directorio de instalación de mSQL y por defecto vale /usr/local/Hughes. Este es el directorio por defecto de la distribución mSQL 2.0. **configure** detecta automáticamente qué versión de mSQL está ejecutándose y el PHP soporta tanto 1.0 como 2.0, pero si compila el PHP con mSQL 1.0 sólo podrá acceder a bases de datos de esa versión y viceversa.

Vea también Directivas de Configuración de mSQL en el archivo de configuración.

Página de mSQL (http://www.hughes.com.au)

MySQL

-with-mysql=*DIR*

Habilita el soporte para MySQL. El parámetro es el directorio de instalación de MySQL y por defecto vale /usr/local.

Este es el directorio de instalación de la distribución de MySQL.

Vea también Directivas de Configuración de MySQL en el archivo de configuración.

Página de MySQL (http://www.tcx.se)

iODBC

-with-iodbc=*DIR*

Incluye soporte para iODBC. Esta característica se desarrolló inicialmente para el iODBC Driver Manager, un gestor de controlador de ODBC de redistribución libre que ese ejecuta bajo varios sabores de UNIX. El parámetro es el directorio de instalación de iODBC y por defecto vale /usr/local.

Página de FreeODBC (http://users.ids.net/~bjepson/freeODBC/) o página de iODBC (http://www.iodbc.org)

OpenLink ODBC

-with-openlink=*DIR*

Incluye soporte para OpenLink ODBC. El parámetro es el directorio de instalación de OpenLink ODBC y por defecto vale /usr/local/openlink.

Página de OpenLink Software (http://www.openlinksw.com/)

Oracle

-with-oracle=*DIR*

Incluye soporte para Oracle. Se ha probado y debería funcionar al menos con las versiones de la 7.0 a la 7.3. El parámetro es el directorio ORACLE_HOME. No necesita especificar este parámetro si su entorno de Oracle ya está ajustado.

Página de Oracle (http://www.oracle.com)

PostgreSQL

-with-pgsql=*DIR*

Incluye soporte para PostgreSQL. El parámetro es el directorio base de la instalación de PostgreSQL y por defecto vale /usr/local/pgsql.

Vea también Directivas de Configuración de Postgres en el archivo de configuración.

Página de PostgreSQL (http://www.postgreSQL.org/)

Solid

-with-solid=*DIR*

Incluye soporte para Solid. El parámetro es el directorio de instalación y vale por defecto /usr/local/solid.

Página de Solid (http://www.solidtech.com)

Sybase

-with-sybase=*DIR*

Incluye soporte para Sybase. El parámetro es el directorio de instalación y vale por defecto /home/sybase.

Vea también Directivas de Configuración de Sybase en el archivo de configuración.

Página de Sybase (http://www.sybase.com)

Referencia del Lenguaje

Sintaxis básica

Hay cuatro formas de salir de HTML y entrar en el "modo de código PHP":

Ejemplo 5-1. Formas de salir de HTML

```
1. <? echo ("esta es la más simple, una instrucción de procesado SGML\n"); ?>
2. <?php echo("si quiere servir documentos XML, haga esto\n"); ?>
3. <script language="php">
echo ("a algunos editores (como FrontPage) no les gustan las intrucciones de procesado");
</script>
4. <% echo ("Puedes también usar etiquetas tipo ASP"); %>
<%= $variable; # Esto es una forma abreviada de "<%echo .." %>
```

La primera forma sólo está disponible si se han habilitado las etiquetas cortas. Esto se puede hacer a través de la función **short_tags()**, habilitando la opción de configuración short_open_tag en el archivo de configuración de PHP, o compilando PHP con la opción –enable-short-tags en **configure**.

La cuarta manera está disponible sólo si se han habilitado las etiquetas tipo ASP usando la opción de configuración asp_tags.

Nota: El soporte para las etiquetas tipo ASP se añadió en 3.0.4.

La etiqueta de cierre de un bloque incluirá el carácter de nueva línea final si hay uno presente.

Separación de instrucciones

Las instrucciones se separan igual que en C o perl - terminando cada sentencia con un punto y coma.

La etiqueta de cierre (?>) también implica el fin de la sentencia, así lo siguiente es equivalente:

```
<?php
echo "Esto es una prueba";
?>
<?php echo "Esto es una prueba" ?>
```

Comentarios

PHP soporta comentarios tipo 'C', 'C++' y shell de Unix. Por ejemplo:

```
<?php
echo "Esto es una prueba"; // Esto es un comentario tipo c++ para una línea
/* Esto es un comentario multilínea
```

```
otra línea más de comentario*/
echo "Esto es aún otra prueba";
echo "Una Prueba Final"; # Este es un comentario tipo shell
?>
```

El tipo de comentario de "una línea" sólo comenta, en realidad, hasta el fin de la línea o el bloque actual de código PHP, lo que venga primero.

```
<h1>Esto es un <?# echo "simple";?> ejemplo.</h1>
<p>La cabecera de arriba dirá 'Esto es un ejemplo'.
```

Se debería tener cuidado para no anidar comentarios de tipo 'C', lo cual puede ocurrir cuando se comentan grandes bloques.

```
<?php
/*
echo "Esto es una prueba"; /* Este comentario causará un problema */
*/
?>
```

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.

```
$var = "Bob";
$Var = "Joe";
echo "$var, $Var"; // produce la salida "Bob, Joe"
```

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 o "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 arrays u objetos.

Para asignar por referencia, simplemente se antepone un ampersand (&) 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
$bar = &$foo; // Referencia $foo vía $bar.
```

```
$bar = "Mi nombre es $bar"; // Modifica $bar...
echo $foo; // $foo también 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;
$bar = &$foo; // Esta es una asignación válida.
$bar = &(24 * 7); // Inválida; referencia una expresión sin nombre.
function test() {
return 25;
}
$bar = &test(); // Inválida.
?>
```

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.

A pesar de estos factores, aquí tenemos una lista de variables predefinidas disponibles en una instalación por defecto de PHP 3 corriendo como modulo de un Apache (http://www.apache.org/) 1.3.6 con su configuración también por defecto.

Para una lista de variables predefinidas (y muchas más información útil), por favor, vea (y use) **phpinfo()**.

Nota: Esta lista no es exhaustiva ni pretende serlo. Simplemente es una guía de qué tipo de variables predefinidas se puede esperar tener disponibles en un script.

Variables de PHP

Estas variables son creadas por el propio PHP.

argv

Array de argumentos pasados al script. Cuando el script se ejecuta desde la línea de comandos, esto da un acceso, al estilo de C, a los parámetros pasados en línea de comandos. Cuando se le llama mediante el método GET, contendrá la cadena de la petición.

argc

Contiene el número de parámetros de la línea de comandos pasados al script (si se ejecuta desde la línea de comandos).

PHP_SELF

El nombre del fichero que contiene el script que se esta ejecutando, relativo al directorio raíz de los documentos. Si PHP se está ejecutando como intérprete de línea de comandos, esta variable no está disponible.

HTTP_COOKIE_VARS

Un array asociativo de variables pasadas al script actual mediante cookies HTTP. Sólo está disponible si el seguimiento de variables ha sido activado mediante la directiva de configuración track_vars o la directiva <?php_track_vars?>.

HTTP_GET_VARS

Un array asociativo de variables pasadas al script actual mediante el método HTTP GET. Sólo está disponible si –variable tracking– ha sido activado mediante la directiva de configuración track_vars o la directiva <?php_track_vars?>.

HTTP_POST_VARS

Un array asociativo de variables pasadas al script actual mediante el método HTTP POST. Sólo está disponible si –variable tracking– ha sido activado mediante la directiva de configuración track_vars o la directiva <?php_track_vars?>.

Ámbito 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:

```
$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:

```
$a = 1; /* ámbito global */
Functcion Test () {
echo $a; /* referencia a una variable de ámbito local */
}
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:

```
$a = 1;
$b = 2;
Functcion 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 el array $GLOBALS propio de PHP3.

El ejemplo anterior se puede reescribir así:

```
$a = 1;
$b = 2;
Function Sum () {
$GLOBALS["b"] = $GLOBALS["a"] + $GLOBALS["b"];
}
Sum ();
echo $b;
```

El array $GLOBALS es un array asociativo con el nombre de la variable global como clave y los contenidos de dicha variable como el valor del elemento del array.

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:

```
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:

```
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:

```
Function Test () {
static $count = 0;
$count++;
echo $count;
if ($count < 10) {
```

```
Test ();
}
$count-;
}
```

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:

```
$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.

```
$$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:

```
echo "$a ${$a}";
```

produce el mismo resultado que:

```
echo "$a $hello";
```

p.ej. ambas producen el resultado: *hello world*.

Para usar variables variables con arrays, 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.

Estructuras de Control

Todo archivo de comandos 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:

```
if (expr)
sentencia
```

Como se describe en la sección sobre expresiones, expr se evalúa a su valor condicional. Si *expr* se evalúa como TRUE, PHP ejecutará la sentencia, y si se evalúa como FALSE - la ignorará.

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

```
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*:

```
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 condición, 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:

```
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:

```
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 alternativa 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 = 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:

```
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 múltiples 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:

```
/* ejemplo 1 */
$i = 1;
while ($i <= 10) {
print $i++; /* el valor impreso sería
$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:

```
$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, encapsulándolos con do..while(0), y usando la sentencia break. El siguiente fragmento de código demuestra esto:

```
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 implícitamente 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:

```
/* 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.

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 muchos valores diferentes, y ejecutar una parte de código distinta dependiendo de a qué valor es igual. Para ello sirve la sentencia switch.

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:

```
if ($i == 0) {
print "i es igual a 0";
}
if ($i == 1) {
print "i es igual a 1";
}
if ($i == 2) {
print "i es igual a 2";
}
switch ($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;
}
```

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:

```
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 sentencias 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.

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

```
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.

```
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;
```

Funciones

Funciones definidas por el usuario

```
Una función se define con la siguiente sintaxis:
function foo ($arg_1, $arg_2, ..., $arg_n) {
echo "Función de ejemplo.\n";
return $retval;
}
```

Cualquier instrucción válida de PHP puede aparecer en el cuerpo de la función, incluso otras funciones y definiciones de clases.

En PHP3, las funciones deben definirse antes de que se referencien. En PHP4 no existe tal requerimiento.

PHP no soporta la sobrecarga de funciones, y tampoco es posible redefinir u ocultar funciones previamente declaradas.

PHP3 no soporta un número variable de parámetros, aunque sí soporta parámetros por defecto (ver Valores por defecto 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:

```
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:

```
function add_some_extra(&$string) {
$string .= ’ y algo más.’;
}
$str = ’Esto es una cadena, ’;
add_some_extra($str);
echo $str; // Saca ’Esto es una cadena, y algo más.’
```

Si deseas pasar una variable por referencia a una función que no toma el parámetro por referencia por defecto, puedes anteponer un ampersand al nombre del parámetro en la llamada a la función:

```
function foo ($bar) [
$bar .= ’ y algo más.’;
}
$str = ’Esto es una cadena, ’;
foo ($str);
echo $str; // Saca ’Esto es una cadena, ’
foo (&$str);
echo $str; // Saca ’Esto es una cadena, y algo más.’
```

Parámetros por defecto

Una función puede definir valores por defecto para los parámetros escalares estilo C++:

```
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 o miembro de una clase.

En PHP 4.0 también es posible especificar unset como parámetro por defecto. Esto significa que el argumento no tomará ningún valor en absoluto si el valor no es suministrado.

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:

```
function makeyogurt ($type = "acidophilus", $flavour) {
return "Haciendo un bol de $type $flavour.\n";
}
echo makeyogurt ("mora"); // No funcionará 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:

```
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.

Devolver valores

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

```
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.

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

Clases y Objetos

class

Una clase es una colección de variables y de funciones que acceden a esas variables. Una clase se define con la siguiente sintaxis:

```
<?php
class Cart {
var $items; // Items en nuestro carro de la compra
// Añadir $num artículos de tipo $artnr al carro
function add_item ($artnr, $num) {
$this->items[$artnr] += $num;
}
// Sacar $num artículos del tipo $artnr del carro
function remove_item ($artnr, $num) {
if ($this->items[$artnr] > $num) {
$this->items[$artnr] -= $num;
```

```
return true;
} else {
return false;
}
}
}
?>
```

El ejemplo define una clase llamada Cart que consiste en un array asociativo de artículos en el carro y dos funciones para meter y sacar ítems del carro

Las clases son tipos, es decir, son plantillas para variables. Tienes que crear una variable del tipo deseado con el operador new.

```
$cart = new Cart;
$cart->add_item("10", 1);
```

Este ejemplo crea un objeto $cart de clase Cart. La función add_item() de ese objeto se llama para añadir un item del artículo número 10 al carro.

Las Clases pueden ser extensiones de otras clases. Las clases extendidas o derivadas tienen todas las variables y funciones de la clase base y lo que les añadas al extender la definición. La herencia múltiple no está soportada.

```
class Named_Cart extends Cart {
var $owner;
function set_owner ($name) {
$this->owner = $name;
}
}
```

Ese ejemplo define una clase Named_Cart (carro con nombre o dueño) que tiene todas las variables y funciones de Cart, y además añade la variable $owner y una función adicional set_owner(). Un carro con nombre se crea de la forma habitual y, una vez hecho, puedes acceder al propietario del carro. En los carros con nombre también puedes acceder a las funciones normales del carro:

```
$ncart = new Named_Cart; // Creamos un carro con nombre
$ncart->set_owner ("kris"); // Nombramos el carro
print $ncart->owner; // Imprimimos el nombre del propietario
$ncart->add_item ("10", 1); // Funcionalidad heredada de Cart
```

Entre funciones de una clase, la variable $this hace referencia al propio objeto. Tienes que usar $this->loquesea para acceder a una variable o función llamada loquesea del objeto actual.

Los constructores son funciones de una clase que se llaman automáticamente al crear una nueva instancia (objeto) de una clase. Una función se convierte en constructor cuando tiene el mismo nombre que la clase.

```
class Auto_Cart extends Cart {
function Auto_Cart () {
$this->add_item ("10", 1);
}
}
```

Este ejemplo define una clase Auto_Cart que es un Cart junto con un constructor que inicializa el carro con un item del tipo de artículo "10"cada vez que se crea un nuevo Auto_Cart con "new".

Los constructores también pueden recibir parámetros y estos parámetros pueden ser opcionales, lo que los hace más útiles.

```
class Constructor_Cart extends Cart {
function Constructor_Cart ($item = "10", $num = 1) {
$this->add_item ($item, $num);
}
}
// Compramos las mismas cosas aburridas de siempre
$default_cart = new Constructor_Cart;
// Compramos las cosas interesantes
$different_cart = new Constructor_Cart ("20", 17);
```