

PRESENTACIÓN	2
QUÉ ES.....	2
QUÉ PUEDE HACER	3
INSTALACIÓN	3
BAJANDO LAS ÚLTIMAS VERSIONES	3
INSTALACIÓN SOBRE UNIX	3
CONFIGURACIÓN	6
EL LENGUAJE.....	7
SINTAXIS BÁSICA.....	7
<i>Insertando PHP.....</i>	7
<i>Separación de instrucciones.....</i>	7
<i>Comentarios.....</i>	7
TIPOS	8
<i>Strings.....</i>	8
<i>Arrays.....</i>	9
VARIABLES	10
<i>Variables predefinidas.....</i>	11
<i>Ámbito de una variable.....</i>	11
<i>Variables de variables.....</i>	11
<i>Variables externas.....</i>	12
<i>Variables de entorno</i>	13
CONSTANTES.....	13
OPERADORES	13
<i>Aritméticos.....</i>	13
<i>De Cadena</i>	13
<i>Asignación</i>	14
<i>De Bit.....</i>	14
<i>Lógicos</i>	14
<i>Comparación.....</i>	14
ESTRUCTURAS DE CONTROL.....	15
<i>Desvío.....</i>	15
<i>Iterativas.....</i>	15
<i>Inclusión</i>	16
FUNCIONES	16
ASPECTOS AVANZADOS	17
ORIENTACIÓN A OBJETOS.....	17
TEMPLATES.....	18
BASES DE DATOS: MySQL	22
SISTEMA DE FICHEROS	23
EJECUCIÓN DE PROGRAMAS	25
SOCKETS	26
SEMÁFOROS	27
RESUMEN FUNCIONES ARRAYS.....	28
RESUMEN FUNCIONES STRING.....	29
RESUMEN DE HTML BÁSICO	31
<i>Formato de la página</i>	31
<i>Links e imágenes</i>	31
<i>Tablas</i>	31
<i>Formato de texto</i>	32
<i>Listas</i>	32
<i>Formularios</i>	32
RESUMEN DE SQL.....	33

Presentación

Nota: Esta documentación no es más que un breve resumen de algunas de las posibilidades que ofrece PHP. Una documentación más detallada y completa puede encontrarse en la página oficial, <http://www.php.net/manual>.

PHP, que proviene de "PHP: Hypertext Preprocessor" es un lenguaje de scripting empotrado en HTML. Gran parte de su sintaxis se asemeja a C, Java o Perl, aunque incluye potentes características (acceso a bases de datos, DOM, generación de imágenes, etc).

El objetivo del lenguaje es permitir a los programadores escribir páginas dinámicas rápidamente.

Qué es

PHP es un lenguaje de 'scripting' que se inserta en páginas HTML:

```
<html>
<head>
    <title>Probando</title>
</head>
<body>
<?php echo "Esto es una prueba"; ?>
</body>
</html>
```

A diferencia de los scripts escritos en otros lenguajes como Perl o C que se dedican a imprimir líneas de código HTML, lo que se hace en PHP es empotrar el código (en el ejemplo anterior mostrar un simple mensaje) en la página HTML. El código PHP se inserta en la página mediante los tags de inicio '<?' y de final '?>' que nos permitirán salir del código HTML cuando nos interese.

Lo que diferencia a PHP de otros lenguajes de scripting (como Javascript) es que el código se ejecuta en el lado del servidor, es decir, el código se interpreta y se ejecuta antes de ser entregado por el servidor web. Así la página que recibiríamos en nuestro navegador sería algo parecido a:

```
<html>
<head>
    <title>Probando</title>
</head>
<body>
    Esto es una prueba
</body>
</html>
```

Qué puede hacer

A nivel básico, PHP puede hacer cualquier cosa que haga otro tipo de CGI, como recoger valores, generar contenido dinámico o enviar y recibir cookies.

Sin embargo, uno de los aspectos más importantes es la gran cantidad de bases de datos a las que se puede acceder: Adabas, InterBase, Informix, Oracle, Sybase, Velocis, PostgreSQL, MySQL, mSQL, FilePro, etc...

Permite además comunicarse con otros servicios usando protocolos como IMAP, SNMP, POP3 o incluso HTTP, aunque ofrece la posibilidad que trabajar a nivel de sockets para usar protocolos propios.

Instalación

Bajando las últimas versiones

Las últimas versiones, tanto de los fuentes como de los binarios, se pueden encontrar en <http://www.php.net>

Instalación sobre UNIX

Para realizar una instalación básica del php sobre Apache (<http://www.apache.org>) en un entorno UNIX debemos realizar los siguientes pasos

Descomprimir el source del Apache:

1. `gunzip apache_1.3.x.tar.gz`
2. `tar xvf apache_1.3.x.tar`

Descomprimir el source del php:

3. `gunzip php-4.0.x.tar.gz`
4. `tar xvf php-4.0.x.tar`

Relizar un configure en el apache falso (no hace falta poner ninguna otra opción aparte del `--prefix`).

5. `cd apache_1.3.x`
6. `./configure --prefix=/www`

Lanzar la configuración de PHP

7. `cd ../php-4.0.x`
8. `./configure --with-mysql --with-apache=../apache_1.3.x --enable-track-vars`

El script "configure" admite muchos parámetros para la configuración de PHP, la mayoría son para habilitar o deshabilitar módulos (para ello hacen falta librerías o programas extras; en este ejemplo se supone pre-instalado el MySQL).

Para ver todas las opciones del configure basta poner `./configure --help`.

Compilar e instalar:

- 9. make
- 10. make install

Realizar el configure verdadero en el Apache

- 11. cd ../apache_1.3.x
- 12. ./configure --prefix=/www -activate-module=src/modules/php/libphp.a

Aquí además de estas opciones se pueden poner todas aquellas que acepta normalmente el Apache: como activación de otros módulos (mod_java,mod_perl,mod_ssl,...) o parámetros propios (eapi,mm,...).

Compilar e instalar el Apache

- 13. make
- 14. make install

Copiar el archivo de configuración de PHP

- 15. cd ../php-4.0.x
- 16. cp php.ini-dist /usr/local/lib/php.ini

Puedes editar este fichero y ajustar varios parámetros sobre la ejecución del PHP (Ver apartado

Configuración).

Configurar el Apache para que reconozca el PHP

17. Edita tu `httpd.conf` y añade la línea:

```
AddType application/x-httpd-php .php
```

Que le indica que cuando se pida un fichero con extensión `.php` se lo pase al módulo de PHP.

`.php` suele ser la extensión habitual para PHP4 (la versión actual), también se usa `.php4`.

`.php3` para la versión 3

`.phtml` para la versión 2.

18. Arrancar el servidor de Apache (normalmente con `apachectl start`). Si está encendido, pararlo primero (`apachectl stop`).

Este es el método rápido para ponerlo sobre Apache, recordamos que PHP funciona sobre otros servidores web y como CGI. Además se puede configurar habilitando muchos otros módulos. Y la parte de configuración del Apache puede ser mucho más compleja si tiene que convivir con otro tipo de tecnologías (como puede ser Java o Perl).

En caso de usar Apache se recomienda utilizar DSO (carga dinámica de módulos) que facilita considerablemente la actualización de sus módulos (para más información al respecto mirar la documentación de Apache).

Configuración

Recordamos que PHP tiene un fichero de configuración (php.ini) donde se pueden configurar diversos comportamientos, la mayoría de estos parámetros ya tienen un valor adecuado con la configuración por defecto. Veamos algunos:

short_open_tag = [On]|Off

Admite <? como tag para empezar código PHP

safe_mode = On|[Off]

Activa el "safe mode" de PHP (ver más adelante).

max_execution_time = [30]

Tiempo máximo de ejecución de un script de PHP en segundos pasado este tiempo el script aborta si no lo ha modificado mediante `set_time_limit()`.

memory_limit = [8388608]

Máxima memoria que puede utilizar un script de PHP en bytes.

error_reporting = [7]

Tipos de errores de los que queremos ser avisados. Es una or binaria de los siguientes valores:

1 = Errores normales

2 = Warnings normales

4 = Errores del parser

8 = Notices - warnings que se pueden ignorar pero que a veces implican un bug

display_errors = [On] | Off

Mostrar los errores en la salida del script (en HTML).

log_errors = On | [Off]

Escribir los logs en un fichero de log.

include_path = path:path2:path3

Este parametro indica los lugares donde el PHP busca los scripts cuando realizamos un `include()` o `require()`. Un valor habitual es:

`include_path = ./:www/php`

Para que busque primero donde se está ejecutando el script y luego en un directorio global donde pueden haber bibliotecas de clases.

Safe Mode

Cuando `safe_mode` esta a On, PHP se ejecuta en un modo especial que intenta mejorar la seguridad del modelo de ejecución. Para ello se prohíbe cualquier acción sobre un fichero (abrir, ejecutar,...) que no tenga el mismo propietario que el propietario del script actual.

Además se deshabilitan algunas funciones como `set_time_limit()` para garantizar que se cumplen las políticas dictadas por la configuración global.

Como excepción a lo anterior cabe decir, que sí se pueden ejecutar aquellas aplicaciones que se encuentren en el directorio definido por el parametro `safe_mode_exec_dir`. Así podemos poner un link al sendmail en ese directorio para permitir que se envíen correos a través de las funciones de PHP.

El lenguaje

Sintaxis básica

Insertando PHP

Hay 4 maneras de insertar código PHP en una página HTML:

- 1) `<? echo "La forma simple\n"; ?>`
- 2) `<?php echo "Si hay XML rondando..."; ?>`
- 3) `<script language="php">`
 `echo "Para soportar editores de MocoSoft (léase`
 `FrontPeich)";`
 `</script>`
- 4) `<% echo ("La costumbre de ASP... -al paredon!!!-"); %>`

Aunque también puede obtenerse por usar código PHP que escribir el HTML necesario:

```
<?php
echo "<html>\n";
echo "<head><title>Probando</title></head>\n";
echo "<body>Esto es una prueba</body>\n";
echo "</html>\n";
¿>
```

Los nombres y palabras reservadas de PHP son case-sensitive, es decir, que distingue entre mayúsculas y minúsculas.

Separación de intrucciones

Las instrucciones se separan igual que en C o en Perl, es decir, con `';`. El tag de final `'?>` también implica separación de instrucciones. Así las sentencias siguientes son equivalentes:

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

<?php echo Esto es una prueba ¿>
```

Comentarios

```
Estilo C:           /* Comentario */
Estilo C++:         // Comentario
Estilo Shell Unix:  # Comentario
```

Cuando se use el estilo C ojo con la anidación.

Tipos

Los tipos básicos de PHP son:

- Array
- Reales
- Entero
- Objeto
- Strings

El tipo de una variable no lo decide el programador sino que se decide en tiempo de ejecución dependiendo del contexto en el que aparezca la variable.

Se pueden convertir a otros tipos mediante type-casting...

```
$unEntero = 10;  
$unReal = (double) $unEntero;
```

...o mediante la función `settype`...

```
settype( $unEntero, "double" )
```

Strings

Se pueden delimitar mediante `"` o `'`. La diferencia radica en que dentro de las `"` el intérprete de PHP expande variables. Así:

```
<?php  
$unaVariable = "probando";  
$laFrase = "Ahora estoy $unaVariable";  
echo $laFrase;  
;>
```

...daría como resultado...

Ahora estoy probando

Se concatenan mediante `'.'`

```
<?php  
$unaVariable = "probando";  
echo "Ahora estoy " . $unaVariable;  
;>
```

Se puede acceder a los caracteres de un string mediante `[]`

```
<?php  
$unaVariable = "probando";  
echo $unaVariable[3]; // 'b'  
;>
```


Arrays

Vectores (unidimensionales)

```
$unArray = array( "naranja", "manzana", "melon" )
```

...equivale a...

```
$unArray[0] = "naranja";  
$unArray[1] = "manzana";  
$unArray[2] = "melon";
```

Se pueden añadir elementos:

```
$unArray[] = "limón"; // Equivale a $unArray[3] = "limón";
```

Los índices no tienen porque ser numéricos, ni los valores del mismo tipo:

```
<?php  
$cuantos[ "arboles" ] = 5;  
$cuantos[ "torpedos" ] = "muchos";  
$cuantos[0] = 123.4;  
$cuantos[ $unaVariable ] = $otraVariable;  
>
```

Se pueden ordenar mediante asort(), arsort(), ksort(), rsort(), sort(), uasort(), usort(), uksort().

Se puede contar el numero de elementos mediante count().

Una forma de recorrerlas rapidamente:

```
while( list( $clave, $valor ) = each( $unArray ) )  
    echo "$clave = $valor<br>\n";
```

Multidimensionales

Basta con añadir otro índice:

```
$a[1]      = $f;           // Una dimension  
a["hola"] = $f;
```

```
$a[1][0]    = $f;         // Dos dimensiones  
$a["hola"][2] = $f;  
$a[3]["adios"] = $f;
```

```
$a["hola"][4]["adios"][0] = $f;           // Cuatro dimensiones
```

Hay varias formas de llenar un array multidimensional:

```
$a[ "color" ] = "rojo";  
$a[ "sabor" ] = "dulce";  
$a[ "forma" ] = "redonda";  
$a[ "nombre" ] = "manzana";  
$a[ 3 ]      = 4;
```

... equivale a ...

```
$a = array(  
    "color" => "rojo",  
    "sabor" => "dulce",  
    "forma" => "redonda",  
    "nombre" => "manzana",  
    3 => 4  
);
```

... y se pueden anidar ...

```
<?  
$a = array(  
    "manzana" => array(  
        "color" => "rojo",  
        "sabor" => "dulce",  
        "forma" => "redonda"  
    ),  
    "naranja" => array(  
        "color" => "naranja",  
        "sabor" => "acido",  
        "forma" => "redonda"  
    ),  
    "platano" => array(  
        "color" => "amarillo",  
        "sabor" => "pastoso",  
        "forma" => "aplatanado"  
    )  
);  
  
echo $a["manzana"]["sabor"]; // Escribiria "dulce"
```

Variables

Las variables en PHP se representan con el simbolo \$ seguido del nombre de la variable. Los nombres válidos son aquellos que empiezan por "_" o por letra, y seguidos de letras, números o "_".

En PHP las variables se asignan por valor, es decir, cuando se asigna una expresión a una variable el resultado de la evaluación de la expresión se copia a la variable de destino. Esto significa, por ejemplo, que después de asignar una variable a otra, variar el contenido de una de ellas no influye en la otra.

Sin embargo, a partir de la versión 4 de PHP apareció la asignación por referencia: la nueva variable referencia (es una "alias" o "apunta a") la variable original. Es decir, un cambio en cualquiera de las dos variables influye en la otra.

Para asignar por referencia basta con poner "&" delante de la variable a la que se quiere hacer referencia:

```
<?php
$a = 'David';
$b = &a;
$b = 'Feo';
echo $a;    // Escribe 'Feo' en ambos casos.
echo $b;
?>
```

Variables predefinidas

PHP ofrece una gran cantidad de variables predefinidas. Sin embargo, algunas de estas variables no estan totalmente documentadas debido a que dependen de la plataforma, de la versión y configuración del servidor o de otros factores. Algunas de ellas sólo no son accesibles cuando se ejecuta PHP desde la línea de comandos.

Algunos ejemplos:

- argv, argc: Array de parámetros y número de parametros pasados al script (si se ha lanzado desde línea de comandos).
- PHP_SELF: el nombre del script que se ejecuta.
- HTTP_COOKIE_VARS: array que contiene las cookies enviadas por el navegador del usuario.
- HTTP_GET_VARS: array que contiene las variables enviadas por GET.
- HTTP_POST_VARS: array que contiene las variables enviadas por POST.

Ámbito de una variable.

Si se declaran en el cuerpo principal del script tienen ámbito global. Declaradas dentro de una función tienen ámbito local.

Para acceder a las variables globales desde dentro de una función se puede usar "global" o bien el array \$GLOBALS:

```
$a = 1;

function unaFuncion()
{
    global $a
    echo $a;    // Escribe "1";
}

function otraFuncion()
{
    echo $GLOBALS["a"];    // Tambien escribe "1";
}
```

Variables de variables

```
$hola = "eres un torpedo";
$adios = "hola";
echo $$adios;    // Escribe "eres un torpedo";
```

Variables externas

Formularios HTML

Cuando se envia un formulario a un script PHP, cualquier variable definida en el formulario es accesible en el script de destino.

```
<form method=POST action=recoger.php>
<input type=text name=nombre>
<input type=submit>
</form>
```

Una vez enviado el script "recoger.php" tendra definida la variable \$nombre que contendrá el valor introducido en el formulario.

PHP soporta el uso de arrays en formularios (pero sólo de 1 dimensión).

```
<form method=POST action=recoger.php>
<input type=text name=datos[nombre]><br>
<input type=text name=datos[email]><br>
<select multiple name=cerveza[ ]>
<option value="rubia">Rubia</option>
<option value="sin alcohol">Sin alcohol</option>
<option value="guinness">Guinness</option>
</select>
<input type=submit>
</form>
```

```
--> recoger.php <--
```

```
echo "Datos personales:<br>\n";
while( list( $key, $val ) = each( $datos ) )
    echo "$key = $val<br>\n";

echo "Cervezas escogidas:<br>\n";
while( list( $key, $val ) = each( $cerveza ) )
    echo "$key = $val<br>\n";
```

Cookies

Se puede colocar cookies en el navegador del usuario mediante SetCookie.

```
SetCookie( "unaCookie", "elValor", time() + 3600 );
```

Las cookies enviadas por el navegador se convierten directamente en variables PHP.

```
echo $unCookie; // Escribiria "elValor";
```

Variables de entorno

PHP hace accesibles las variables del entorno directamente .

```
echo $HOME; // Escribe el HOME del usuario propietario del script.
```

Constantes

PHP define varias constantes y ofrece formas de definir nuevas. Las constantes se comportan igual que las variables salvo por dos aspectos: las constantes se definen mediante la función define(), y no puede variarse su valor una vez definidas.

Algunas de las constantes a las que siempre tendremos acceso son:

__FILE__: El nombre del script que se está ejecutando.
__LINE__: La línea del script que se está ejecutando.
PHP_VERSION: String que contiene la versión de PHP
TRUE, FALSE: Constantes booleanas

Un ejemplo:

```
<?php
if ( !defined( "_MI_CONSTANTE_" ) )
    define( "_MI_CONSTANTE_", 25 );

echo __FILE__." : ".__LINE__." : "._MI_CONSTANTE_;

?php>
```

Operadores

Aritméticos

\$a + \$b	Suma
\$a - \$b	Resta
\$a * \$b	Multipliación
\$a / \$b	División
\$a % \$b	Módulo
++\$a, \$a++	Pre y Post incremento (equiv. \$a = \$a + 1)
--\$a, \$a--	Pre y Post decremento (equiv. \$a = \$a - 1)

De Cadena

. Concatenación

Asignación

El operador básico es el "=". Se puede preceder de otro operador con el resultado de aplicar una operación sobre una variable y dejar el resultado en la misma variable. P.e.:

```
$a = 3;  
$a += 3;    ( equiv. $a = $a + 3 )  
$b = "Estoy ";  
$b .= "probando"; ( $b = "Estoy probando"; )
```

De Bit

\$a & \$b	AND binaria
\$a \$b	OR binaria
~\$a	NOT binario
\$a << \$b	SHIFT LEFT \$b bits
\$a >> \$b	SHIFT RIGHT \$b bits

Lógicos

\$a AND \$b, \$a && \$b	AND lógica
\$a OR \$b, \$a \$b	OR lógica
\$a XOR \$b	XOR lógica
!\$a	NOT lógico

Comparación

\$a == \$b	Iguales
\$a != \$b	Diferentes
\$a < \$b	Menor
\$a > \$b	Mayor
\$a <= \$b	Menor o igual
\$a >= \$b	Mayor o igual

Estructuras de control

Desvío

```
if ( expresion ) // Para mas de una sentencia usar {}
    sentencia

if ( expresion )           // Por bloques;
    sentencia             if ( expresion ):
else                        sentencia
    sentencia             elseif( expresion ):
                           sentencia
if ( expresion )           else:
    sentencia             sentencia
elseif ( expresion )       endif;
    sentencia
else
    sentencia

switch( $i ) {
    case 0: sentencia; break;
    case 1: sentencia; break;
    case 2:
    default: sentencia; break;
}
```

Iterativas

```
while( expresion ) sentencia;

while( expresion ) {
    sentencial;
    sentencia2;
}

while( expresion ):
    sentencial;
    sentencia2;
endwhile;

do {
    sentencia;
} while( expresion );

for( $expr1; $expr2; $expr3 ) sentencia;

for( $expr1; $expr2; $expr3 ) {
    sentencial;
    sentencia2;
}

for( $expr1; $expr2; $expr3 ):
    sentencial;
    sentencia2;
endfor;
```

Inclusión

```
require( nombre_fichero );    // Incluye el fichero aunque la
                              // línea no se ejecute

include( nombre_fichero );    // Incluye el archivo cuando se
                              // ejecuta la línea. Puede usarse
                              // en bucle para incluir datos
                              // varias veces
```

Funciones

```
function unaFuncion( $arg1, $arg2, $arg3, ..., $argN )
{
    echo "Ejemplo";
    return $retval;
}
```

Se puede trabajar con los argumentos de una función mediante func_num_args(), func_get_arg(), func_get_args()

Se pasan por valor, aunque se pueden pasar por referencia:

```
function unaFuncion( &$referencia )
{
    $referencia = "hola";
}
```

Se pueden poner argumentos por defecto:

```
function unaFuncion( $valor = "hola" )
{
    echo $valor;
}
```

Se puede devolver cualquier tipo de información (tanto arrays como objetos) mediante return.

Funciones variables

```
function hola()
{
    echo "adios";
}

$tmp = "hola";
$tmp();    // Escribirá "adios";
```


Aspectos Avanzados

Orientación a objetos

PHP ofrece ciertos mecanismos para la programación orientada objetos. Aunque no contempla todavía todas las características de un verdadero lenguaje orientado a objetos sí ofrece encapsulación, herencia y cierto grado de polimorfismo.

```
<?php

class ObjPadre {                                // Nombre de la clase
    var $unAtributo;                            // atributos
    var $otroAtributo;

    function objPadre() {
        echo "Constructora";
    }

    function unaOperacion( $a ) {
        echo $a;
    }

    function otraOperacion( $b ) {
        echo $b;
    }
}

$unaVariable = new ObjPadre;
?>
```

En el código anterior se crea una instancia de ObjPadre.

Para invocar las operaciones de un objeto se usa el operador ->.

```
<?php $unaVariable->unaOperacion( "Hola" );?>
```

La herencia se consigue "extendiendo" las clases:

```
<?php

class ObjHijo extends ObjPadre {
    var $unAtributo = "hola";

    function unaOperacion( $a ) {
        echo "probando";
    }
}
?>
```

Los atributos y las operaciones se heredan. Sin embargo si tienen el mismo nombre que en la clase padre, entonces se sobrescriben.

La constructora de la clase padre debe invocarse explícitamente.

Templates

Aunque uno de los principales beneficios de usar PHP es que se puede insertar el código directamente entre sentencias HTML, en algunos proyectos interesa separar la lógica de la aplicación de su diseño gráfico. Una herramienta que nos va a ser útil para abordar este tema es la clase FastTemplate. Mediante esta clase y diferentes plantillas (templates) HTML podremos hacer esta separación.

Un template consiste en un archivo que contiene un trozo de HTML, por ejemplo, el cuerpo principal de nuestra web. Entre este código se encuentran insertadas varias palabras entre llaves '{}' que haran de variables del template. Lo que haremos será asignar valores a estas variables y generar el código resultante.

Un ejemplo:

```
--- cuerpo.tpl ---
```

```
<html>
<head>
<title>{TITULO}</title>
</head>
<body bgcolor="{COLORFONDO}">
Esto es una prueba.
Ahora son las {HORA}
</body>
</html>
```

A partir de este template y un script PHP generaremos la página HTML.

```
<?php

include( "class.FastTemplate.php" );

$t = new FastTemplate( "../template" );    // Directorio donde se
                                           //encuentran los
                                           //templates

$t->define( array( "principal" => "cuerpo.tpl" ) );

// A cada template que usemos deberemos asignarle un nombre para
// poder referenciarlo. En nuestro ejemplo el template se llama
// "principal"

$t->assign( TITULO, "Una página de prueba" );    // Asignamos los
$t->assign( COLORFONDO, "white" );              // valores a las
$t->assign( HORA, Date( "T" ) );                // variables

// Parseamos el código: sustituimos los valores de las
// variables en el template

$t->parse( FINAL, principal );                // El resultado lo guardamos
                                           // en una variable
$t->FastPrint( FINAL );                      // Mostramos el resultado
¿>
```

El resultado que obtenemos es el siguiente:

```
<html>
<head>
<title>Una página de prueba</title>
</head>
<body bgcolor="white">
Esto es una prueba.
Ahora son las 18:03
</body>
</html>
```

La potencia de la clase FastTemplate radica a la hora de hacer listados. Imaginemos que queremos sacar un listado de productos (nombre y precio) en la página HTML, cada producto en una fila diferente de una tabla. El diseñador gráfico nos ha entregado 2 templates, el cuerpo principal de la página y el código HTML para cada producto de la lista. Algo parecido a:

```
--- main.tpl ---

<html>
<head>
<title>Listado de productos</title>
</head>
<body bgcolor=white>
Listado de productos:<br>
<table border=0>
{LISTADO}
</table>
</body>
</html>

--- producto_lista.tpl ---

<tr>
    <td>{NOMBRE}</td>
    <td>{PRECIO} Ptas.</td>
</tr>
```

El script necesario para obtener el resultado deseado podría ser:

```
<?php

include( "class.FastTemplate.php" );

$productos = array(
    array(
        "nombre" => "Destornillador",
        "precio" => 1200 ),
    array(
        "nombre" => "Taladro",
        "precio" => 5000 ),
    array(
        "nombre" => "Martillo",
        "precio" => 2000 )
);

$t = new FastTemplate( "../templates" );

$t->define( array(
    "principal" => "main.tpl",
    "producto" => "producto_lista.tpl" ) );

while( list( , $producto ) = each( $productos ) ) {
    $t->assign( NOMBRE, $producto[ nombre ] );
    $t->assign( PRECIO, $producto[ precio ] );

    // Parseamos una línea de producto con las variables
    // anteriores y lo concatenamos ( nótese el '.' ) con
    // La variable interna LISTADO
    $t->parse( LISTADO, ".producto" );
}

// Ahora listado contiene el código HTML con todos los
// productos. Falta parsear el cuerpo principal para substituir
// {LISTADO} con el resultado obtenido.

$t->parse( FINAL, "principal" );
$t->FastPrint( FINAL );
?>
```

Con este código obtenemos una página parecida a:

```
<html>
<head>
<title>Listado de productos</title>
</head>
<body bgcolor=white>
Listado de productos:<br>
<table border=0>
<tr>
    <td>Destornillador</td>
    <td>1200 Ptas.</td>
</tr>
<tr>
    <td>Taladro</td>
    <td>5000 Ptas.</td>
</tr>
<tr>
    <td>Martillo</td>
    <td>2000 Ptas.</td>
</tr>
</table>
</body>
</html>
```

La ventaja de este sistema es que ahora el diseñador sólo tendría que modificar los templates si quisiera por ejemplo usar una lista en lugar de una tabla mientras que el script PHP no haría falta tocarlo:

```
--- main.tpl ---

<html>
<head>
<title>Listado de productos</title>
</head>
<body bgcolor=white>
Listado de productos:<br>
<ul>
{LISTADO}
</ul>
</body>
</html>

--- producto_listado.tpl ---

<li>{NOMBRE} ({PRECIO}.- Ptas)</li>
```

Bases de datos: MySQL

Actualmente PHP soporta el acceso a una gran cantidad de bases de datos, desde las comerciales como Oracle, Informix, Adabas, etc. hasta las OpenSource como PostgreSQL o MySQL. Ésta última, aunque no ofrece todas las características de un SGBD comercial, se ha convertido en uno de los motores de SQL más usados en aplicaciones web de pequeño/mediano tamaño debido a su velocidad y a la gran cantidad de sentencias SQL extendidas que incorpora.

A continuación se presentan las funciones PHP más usadas para atacar a este tipo de base de datos:

```
int mysql_connect( [host] [,usuario] [,contraseña] )
```

Establece la conexión con el servidor MySQL. Devuelve un identificador de conexión o falso si se ha producido un error.

```
int mysql_select_db( base_datos [,identificador] )
```

Selecciona la base de datos a usar en la conexión "identificador". Devuelve cierto si hay éxito, falso si se ha producido un error.

```
int mysql_query( consulta_SQL [,identificador] )
```

Lanza una consulta SQL en la conexión "identificador". Devuelve un identificador de consulta si el servidor MySQL ha podido interpretar y ejecutar la consulta, falso si se ha producido un error.

```
array mysql_fetch_array( id_consulta )
```

Devuelve un array correspondiente al registro de la base de datos donde se encuentra el cursor SQL y avanza éste. Devuelve falso si no hay más registros.

```
int mysql_free_result( id_consulta )
```

Libera los recursos usados por la consulta "id_consulta". Se lanza automáticamente al finalizar el script PHP.

```
int mysql_close( [identificador] )
```

Cierra la conexión "identificador". Devuelve cierto si hay éxito, falso si se ha producido un error.

Un posible script que atacase a una base de datos podría ser:

```
<?php

$linkID = mysql_connect( "localhost", "usuario", "password" );

if ( !$linkID ) exit;
if ( !mysql_select_db( "dbase", $linkID ) exit;

$query = "SELECT nombre,apellidos FROM tabla";
$queryID = mysql_query( $query, $linkID );

while( $campos = mysql_fetch_array( $queryID ) ) {
    echo $campos[nombre];
    echo $campos[apellidos];
}
mysql_free_result( $queryID );
mysql_close( $linkID );
?>
```

Sistema de ficheros

PHP nos proporciona una serie de funciones para interactuar con el sistema de ficheros. Las más importantes son las de lectura de ficheros, así que empezaremos por ellas:

```
int fopen(string filename, string mode [,int use_include_path] )
```

fopen nos retorna un descriptor sobre el cual podemos utilizar otras operaciones. *Filename* indica el fichero que deseamos abrir, *mode* puede ser uno de los siguientes:

r	Modo lectura
r+	Modo lectura/escritura. Puntero al principio del fichero
w	Modo escritura. Truncar a 0. Intentar crear.
w+	Modo lectura/escritura. Truncar a 0. Intentar crear.
a	Modo escritura. Puntero al final. Intentar crear.
a+	Modo lectura/escritura. Puntero al final. Intentar crear

si *use_include_path* vale 1, el fichero se buscará también en los directorios indicados por el parámetro de configuración *include_path*.

Filenames especiales

Si *filename* empieza por "http://" se abre el fichero correspondiente realizando una conexión HTTP 1.0.

Si empieza por "ftp://" se realiza una conexión en **modo pasivo** para acceder al fichero. Se puede acceder en modo lectura o escritura pero no los dos a la vez.

Si es "php://stdin" o "php://stderr" o "php://stdout" se abren los correspondientes streams asociados.

```
int feof(int fp)
```

Nos devuelve cierto en caso de haber llegado al final del fichero *fp* en caso contrario devuelve falso.

string fread(int fp, int length)

Nos devuelve hasta *length* bytes del fichero *fp*.

Variantes útiles de esta función son *fgetc()* (para leer un sólo carácter), *fgets()* (para leer una línea usando '\n' como delimitador) y *fgetss()* (lee del fichero intentando eliminar los tags HTML y de PHP).

int fwrite (int fp, string string [, int length])

Escribe el contenido de *string* en *fp*. Si *length* se especifica solo escribe ese número de bytes.

```
<?php
$fpo = fopen("http://dsl.upc.es","r");
$fpd = fopen("copia","w");
while ( !feof($fpo) ) {
    $buf = fread($fpo,4096);
    fwrite($fpd,$buf);
}
fclose($fpo);
fclose($fpd);
?>
```

int fseek (int fp, int offset [, int whence])

Posiciona el puntero de lectura escritura del archivo *fp* en la posición calcula por *whence+offset*. Donde *whence* puede ser:

SEEK_SET = inicio del fichero
SEEK_CUR = posición actual
SEEK_END = final del fichero

si no se especifica se usa SEEK_SET. ftell() hace lo contrario devolviendonos el offset desde el principio de fichero.

string tempnam(string dir, string prefix)

Nos proporciona un nombre de fichero unico con prefijo *prefix* en el directorio *dir*, aunque en segun que sistemas (Windows,Linux,...) el directorio *dir* si esta configurada alguna zona temporal en el sistema.

array file(string filename [, int use_include_path])

Lee el contenido de *filename* y nos devuelve un array en el cual cada posición es una de las líneas del fichero. *filename* y *use_include_path* tienen el mismo significado y consideraciones que en *fopen()*.

```
<?php
$url_content = file("http://dsl.upc.es");
$fpd = fopen(tempnam("/mytmp","backup"),"w");
while ( list($num,$linea) = each($url_content))
    fwrite($fpd,$linea);
fclose($fpd);
?>
```


Pasamos a resumir brevemente otras funciones que tienen relación con los ficheros y el sistema de ficheros.

Con basename() y dirname() podemos obtener los componentes de un path, basename nos devuelve el nombre del fichero y dirname el path, de tal manera de `path = dirname + basename`.

Con file_exists() podemos verificar que el fichero exista.

Luego tenemos diversas funciones que nos devuelven información de un fichero fileatime() nos devuelve el último tiempo de acceso, filemtime() el último tiempo de modificación, filegroup() el grupo al que pertenece el fichero, fileowner() el usuario propietario del fichero, fileperms() nos devuelve los permisos del fichero, filesize() nos devuelve su tamaño en bytes.

Con las siguientes funciones podemos saber el tipo de fichero con el que estamos tratando: is_dir() para saber si es un directorio, is_executable() para saber si el fichero se puede ejecutar, is_file() para saber si es un fichero normal o especial, is_link() nos indica si el fichero es un link, is_readable() si tenemos acceso de lectura y is_writable() si tenemos acceso de escritura.

Otras nos permiten cambiar atributos de los ficheros: chgrp() para cambiar el grupo de un fichero, chown() para cambiar el propietario, chmod() para cambiar los privilegios, con rename() podemos cambiar su nombre y con touch() podemos cambiar su tiempo de modificación.

También tenemos funciones para obtener información del sistema de ficheros linkinfo() nos devuelve información sobre un link, diskfree() la cantidad de disco libre en un directorio y readlink() nos permite saber a que fichero apunta un link simbolico.

Y por ultimo aquellas que nos permiten modificar el sistema de ficheros: copy() copia un fichero origen otro destino, link() crea un hard link, mkdir() nos permite crear un nuevo directorio, y con rmdir() podemos eliminarlo, symlink() nos permite crear un link simbolico, y con unlink() podemos eliminar un fichero.

Ejecución de programas

PHP también nos permite ejecutar programas externos y recoger su salida. Veamos cuales son estas funciones y como se utilizan:

```
string escapeshellcmd (string command)
```

Esta función parsea la cadena que le pasamos y "escapa" (pone una \ delante) los caracteres especiales del shell. Esta función se ha de pasar siempre sobre aquellas partes de un comando (o todo el comando) que provengan de fuentes no fiables (usuarios, bases de datos,...) para evitar fallos importantes en la seguridad de la aplicación.

```
string exec (string command [, string array [, int return_var]])
```

Ejecuta el comando *command* y nos devuelve la última línea de la ejecución de dicho comando. Si se le pasa el parametro *array* este sera relleno con las líneas de la ejecución del comando. Si se pasa una variable *return_var*, en esta se devolverá el status code que devuelve el comando al finalizar su ejecución.

```
void passthru (string command [, int return_var])
```

Igual que exec(), pero envia la salida del comando directamente al browser, útil para enviar información binaria al navegador.

Operador ``

Ejecuta el comando que se encuentre entre las comillas invertidas y nos devuelve la salida de este comando.

```
<?php  
$ls = `ls -la`;  
>
```

```
int popen (string command, string mode)
```

Popen crea una pipe hacia/desde el comando *command*. Si el modo es *r* entonces de la pipe podremos leer el resultado del comando. Si el modo es *w* podremos escribir en la entrada estandar del comando.

Devuelve un file descriptor como el que devuelve fopen(), sobre el que se pueden utilizar fread(), fwrite(),...

Este file descriptor se ha cerrar mediante pclose().

Sockets

PHP también tiene funciones que nos permiten realizar conexiones AF_INET y AF_UNIX mediante sockets. Esto es útil para realizar interfaces web para aplicaciones que tienen puertos TCP o sockets UNIX para realizar operaciones sobre la aplicación.

Para abrir la conexión se utiliza:

```
int fsockopen (string hostname, int port [,int errno [,string  
errstr [,double timeout]])
```

En *hostname* especificaremos el nombre del servidor o fichero al que queremos conectarnos. En *port* pondremos el puerto TCP al que queremos conectarnos o en caso de socket UNIX un 0. Además podemos pasar dos variables *errno* y *errstr* en las que se nos devolvera el código de error y un mensaje identificativo en caso de no haberse podido realizar la conexión. El ultimo parámetro, *timeout*, nos permite indicar el tiempo máximo que permitimos para realizar la conexión.

fsockopen devuelve un descriptor en caso de haber podido realizar la conexión que podemos utilizar en las funciones fread(), fputs(), fclose(), ...

Además es útil conocer las siguientes funciones:

```
string gethostbyaddr (string ip_address)
```

Nos permite resolver una ip a su nombre de host, en caso de no poder resolverse nos devuelve la ip.

```
string gethostbyname (string hostname)
```

Nos permite obtener la dirección ip asociada a un host.

Semáforos

Las aplicaciones web son inherentemente concurrentes, así que hemos de tener en cuenta el hecho de que varios usuarios pueden estar realizando operaciones a la vez sobre los datos.

La base de datos nos ayuda controlando algunas de estas concurrencias (inserciones, deletes,...) garantizando la integridad de los datos, y proporcionando mecanismos de bloqueo a nivel de tabla, registro,... para casos más complicados.

Pero hay veces que nuestra aplicación no hará uso de una base de datos. Para estos casos se podría utilizar un fichero como bloqueo.

Así podemos intentar crear un fichero para adquirir el lock, si podemos crearlo continuamos sino esperamos un poco i volvemos a probar. Cuando terminamos borramos el fichero.

Más elegante, en caso de no tener BD, es utilizar las funciones de semáforos que nos proporciona PHP.

Son 3:

```
int sem_get(int key, int [max_acquire] , int [perm] )
```

Con esta llamada obtenemos un `id_sem` del semáforo con identificador `key`. Con `max_acquire` podemos indicar el número de procesos que pueden tener el semáforo a la vez, con esto podemos crear semáforos no binarios. `Perm` indica los permisos del semáforo con respecto a otros procesos.

```
int sem_acquire(int sem_identifier)
```

Intentamos obtener el semáforo con identificador `sem_identifier`, bloqueandonos si hace falta hasta conseguirlo.

```
int sem_release(int sem_identifier)
```

Liberamos el semáforo con identificador *sem_identifier* si lo habíamos obtenido previamente. Al finalizar el script se liberan automáticamente todos los semáforos que tuvieramos y no se hubiesen liberado, dando un warning.

Ejemplo:

```
<?php
$idssem = sem_get($WRITE_LOCK_KEY);
sem_acquire($idssem) or die ("Error obteniendo semáforo");

// región crítica.

sem_release($idssem);
</>
```

Como siempre al utilizar funciones de regiones críticas hay que tener cuidado de no caer en deadlocks.

Resumen funciones Arrays

```
void asort( array )
```

Ordena el array manteniendo la asociación de índices.

```
int count( array )
```

Devuelve el número de elementos del array.

```
string mixed( array )
```

Devuelve el elemento actual según el apuntador interno del array.

```
array each( array )
```

Devuelve el siguiente par clave-valor del array.

```
void end( array )
```

Sitúa el apuntador interno al final del array.

```
bool in_array( string valor, array )
```

Devuelve cierto si *valor* se encuentra en el array.

```
string key( array )
```

Devuelve la clave del elemento actual del array.

```
void ksort( array )
```

Ordena el array según sus claves.

void list(string variables)

Asigna valores a las variables como si fueras un array.

```
<?php

list( $a, $b, $c ) = array( "1","2","3" );

echo $a; // 1
echo $b; // 2
echo $c; // 3
</>
```

string next(array)

Avanza el apuntador interno del array y devuelve el elemento.

string prev(array)

Retrocede el apuntador interno del array y devuelve el elemento.

void reset(array)

Sitúa el apuntador interno del array al principio.

void sort(array)

Ordena un array según sus valores.

Resumen funciones String

string addslashes(string str)

Devuelve un string con '\\' delante de los caracteres que necesitan ser espaciados para construir consultas SQL. Estos caracteres son la comilla simple '' , las comillas dobles '', la antebarra \' y el carácter NUL.

```
<?php
$query = sprintf( "DELETE WHERE nombre = '%s'",
    addslashes( "Eres un 'torpedo'" ) );
</>
```

echo string

Escribe por la salida estándar el string

array explode(string separador, string cadena [,int limite])

Devuelve un array con las subcadenas obtenidas de partir *cadena* según el delimitador *separador*. Si se especifica *limite*, el array contendrá *limite* elementos siendo el último el resto de la cadena.

```
<?php
$tmp = "Esto/es/una/prueba";
$array = explode( "/", $tmp );
</>
```

string htmlentities(string)

Convierte todos los posibles caracteres en entidades HTML.

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

// Esto es una &gt; prueba &lt;
¿>
```

string implode(string separador, array trozos)

Devuelve un string donde se han unido todos los elementos del array *trozos* separandólos por el string *separador*.

```
<?php
echo implode( ":", array( "eres", "un", "torpedo" ) );

// eres:un:torpedo
¿>
```

string md5(string)

Devuelve el hash md5 del un string.

print string

Escribe por la salida estándar el string

printf(string formato [,parámetros])

Produce una salida formateada.

string sprintf(string formato [,parámetros])

Devuelve una cadena con formato.

string strip_tags(string)

Elimina los tags HTML o PHP que haya en una cadena.

string stripslashes(string)

Elimina las '\\' introducidas por addslashes

int strlen(string)

Devuelve la longitud de la cadena.

string strtolower(string)

Convierte todas las mayúsculas de la cadena a minúsculas.

string strtoupper(string)

Convierte todas las minúsculas de la cadena a mayúsculas.

```
string str_replace( string pat, string subs, string original )
```

Reemplaza todas las ocurrencias de *pat* por *subs* en la cadena *original*.

```
string substr( string origen, int posicion [,int longitud] )
```

Devuelve la subcadena de *origen* que empieza en *posicion* (con *longitud* caracteres).

```
string trim( string )
```

Elimina los espacios blancos del principio y final de la cadena. Los espacios blancos incluyen '\n', '\r', '\t', '\v', '\0' y el espacio normal.

```
string ucfirst( string )
```

Pone en mayúsculas el primer carácter de la cadena.

```
string ucwords( string )
```

Pone en mayúsculas el primer carácter de todas las palabras de la cadena.

Resumen de HTML Básico

Formato de la página

```
<html>
<head>
<title> titulo </title>
</head>
<body>
...
</body>
</html>
```

Links e imágenes

```
<a href="url"></a>

```

Tablas

```
<table border="tamaño del borde" background="color de fondo">
<tr>
  <td> Fila 1 Celda 1 </td>
  <td> Fila 1 Celda 2 </td>
</tr>
<tr>
  <td> Fila 2 Celda 1 </td>
  <td> Fila 2 Celda 2 </td>
</tr>
</table>
```

Formato de texto

```
<br> salto de linea
<pre>Texto sin formatear</pre>
<k>texto</k>
<b>texto</b>
```

Cambiar el tipo de letra,tamaño y color:

```
<font face="tipo de letra" size="tamaño" color="color">
texto
</font>
```

Listas

```
<ul>
<li>Elemento 1</li>
<li>Elemento 2</li>
<li>Elemento 3</li>
</ul>
```

Formularios

```
<form method="POST|GET" action="script_destino">
campos del formulario
</form>
```

Campos del formulario

Entrada de texto

```
<input type=text name="nombre_campo" value="valor por defecto">

<textarea name="nombre_campo" [wrap=soft]>contenido</textarea>
```

Lista desplegable

```
<select name="nombre_campo" [multiple]>
<option value="opcion1"> Opcion 1 </option>
<option value="opcion2"> Opcion 2 </option>
</select>
```

Marca on/off

```
<input type=checkbox name="nombre_campo" [checked]>
```

Grupo de selección

```
<input type=radio name="nombre_radio" value="valor del radio">
<input type=radio name="mismo_nombre" value="otro valor del radio">
```

Botones

```
<input type=submit name="nombre" value="etiqueta">
<input type=reset name="nombre" value="etiqueta">
<input type=button name="nombre" value="etiqueta">
<input type=image name="nombre" src="imagen">
```


Resumen de SQL

INSERT INTO tabla [(campos)] VALUES (valores)

INSERT INTO productos (5,'camiseta',500,10)

INSERT INTO productos (id_prodo,nombre,precio) VALUES (10,'gorra',300)

UPDATE tabla SET campo=nuevo_valor [WHERE condicion]

UPDATE ofertas SET descuento=0

UPDATE ofertas SET descuento=20 WHERE id_producto=10

DELETE FROM tabla [WHERE condicion]

DELETE FROM productos WHERE precio < 1000

DELETE FROM ofertas WHERE id_producto = 10

SELECT campos FROM tablas [WHERE condicion]

SELECT * FROM productos

SELECT nombre,precio FROM productos WHERE precio > 1500

SELECT p.nombre,o.descuento

FROM productos p,ofertas o

WHERE p.id_producto = o.id_producto

SELECT count(*)

FROM productos p,ofertas o

WHERE p.id_producto = o.id_producto

AND p.precio > 1500 AND o.descuento > 20