ÍNDICE

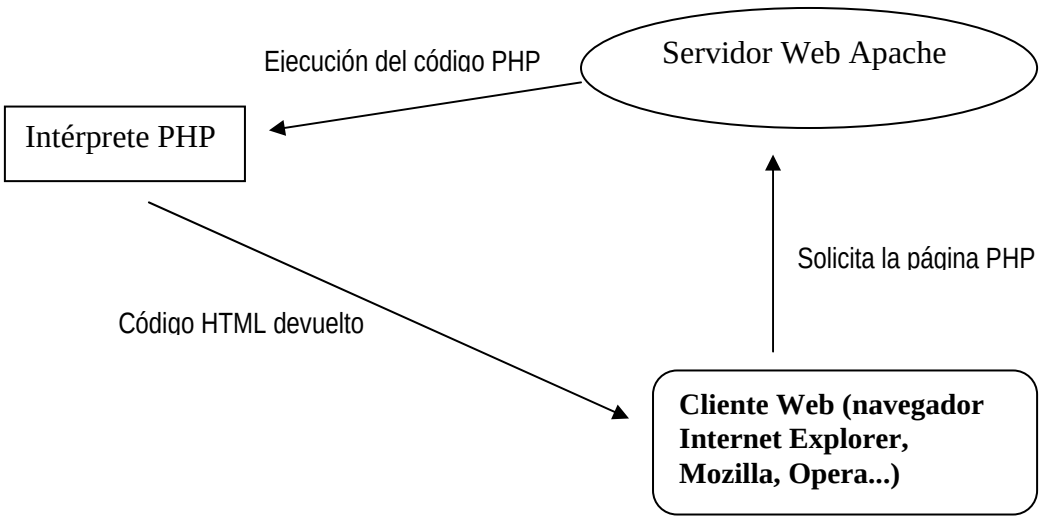
Tema 1

Instalación y configuración de APACHE y PHP

1.1. INTRODUCCIÓN

PHP corresponde a las iniciales de Personal Home Page, Procesador de Hipertexto. Este lenguaje de programación tiene una sintaxis similar a los lenguajes C y Perl. Se interpreta por un servidor web Apache bajo sistemas Unix/Linux (también han salido al mercado versiones para sistemas Windows, aunque no siempre podremos utilizar todas sus características bajo este sistema operativo).

Las páginas PHP son páginas webs con extensión .php o .phtml (otras extensiones comunes son .php3, .php4, .php5 o .inc) que incluyen código HTML, JavaScript y PHP embebido en ellas, y al ejecutarlas, se genera código HTML dinámicamente. Esto significa que al ejecutar las páginas PHP en el servidor web, como petición de un programa visualizador de páginas webs (cliente), se origina una respuesta en función de los datos que introduzca el usuario. El cliente no ve el código del programa PHP, ya que sólo le llegará el código HTML que genere el programa. El cliente (un navegador Internet Explorer, por ejemplo) realiza una petición de un programa a un servidor web (Apache) como si se tratara de cualquier otra página; el cliente no sabrá distinguirlo. Es el servidor web, quien reconoce que la página solicitada es una página PHP (por la extensión), se la envía al intérprete PHP, y éste procesa la página (por ejemplo, consulta una base de datos y genera una página HTML en función de los resultados obtenidos en la consulta), entonces devuelve los resultados al navegador cliente, que visualizará la página HTML resultante como cualquier otra página estática que le llegara.



PHP dispone de un gran número de librerías de funciones para realizar operaciones avanzadas como acceso a bases de datos, comunicaciones, transferencia de ficheros, correo electrónico, etc. En la dirección http://www.php.net podemos encontrar toda la documentación necesaria sobre PHP, como manuales, la referencia del lenguaje, entornos, etc., y además en varios idiomas.

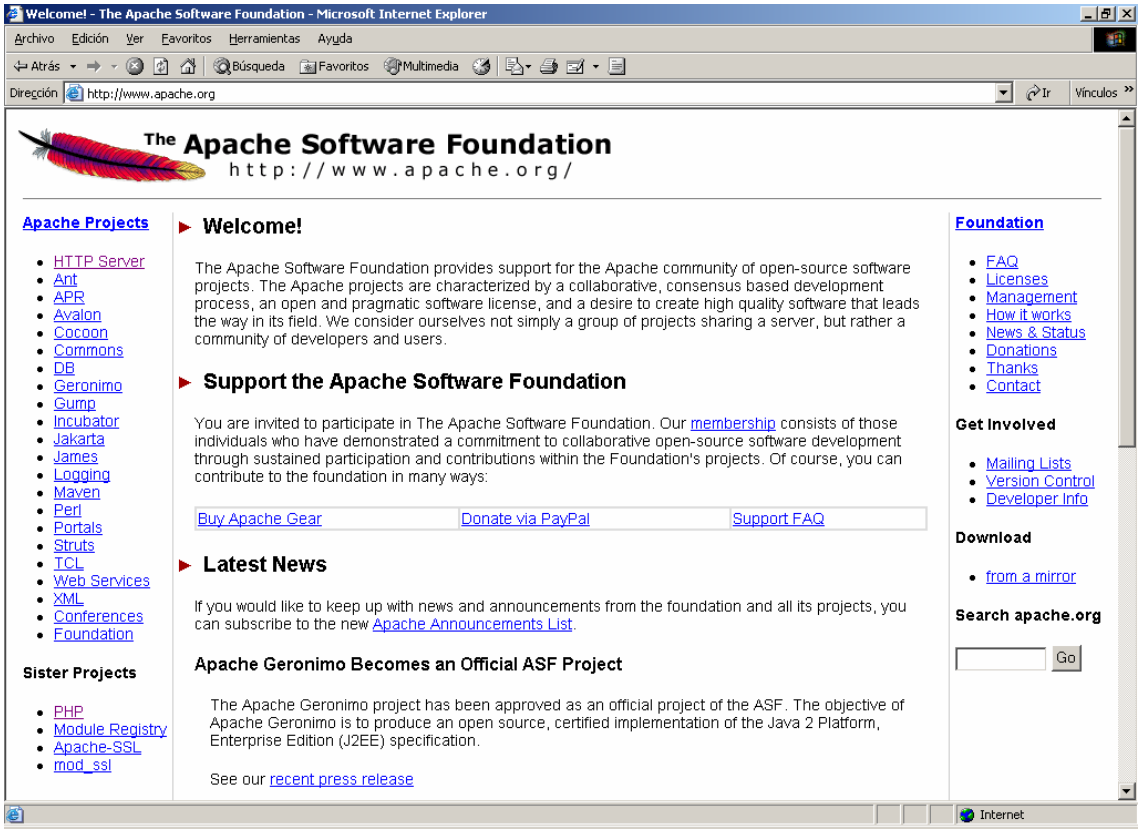
1.2. REQUERIMIENTOS

Para poder ejecutar páginas PHP necesitamos un servidor web Apache. Este servidor web Apache debe estar compilado para incorporar las características o funcionalidades PHP que necesitamos para nuestra aplicación. También es posible utilizar los archivos binarios que incluyen las distribuciones y que ya vienen preparados para admitir las características más habituales.

Igual ocurre con PHP, que admite una gran variedad de módulos, pero las versiones precompiladas no incluyen todos. Esto último conlleva que si necesitamos alguna funcionalidad concreta (como por ejemplo, soporte para acceder y comunicarse con una base de datos específica), tendremos que compilar nuestra propia versión de PHP para incluirla. Los programas fuentes del intérprete de PHP están escritos en C, por lo que podemos compilar dichas fuentes en una plataforma que disponga de un compilador de C e incorporar las funcionalidades que necesitemos para nuestra aplicación.

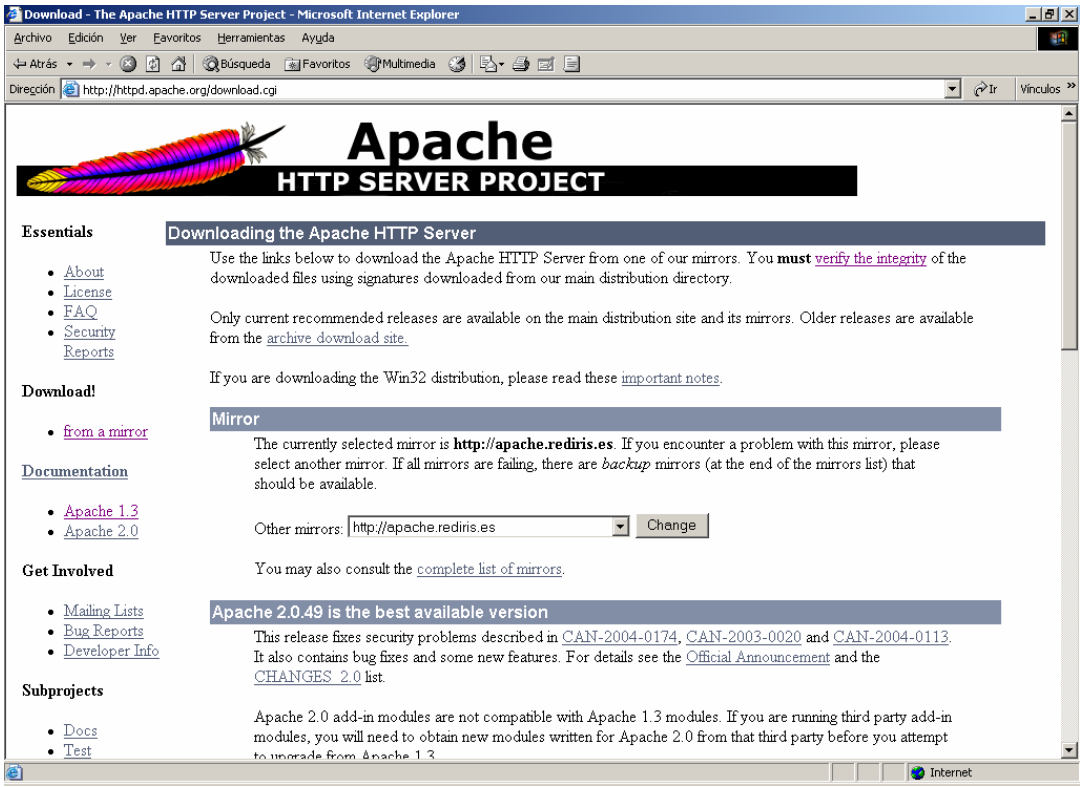
También existen versiones de Apache y PHP para Windows, aunque hay ciertas características que no estarán disponibles bajo estos sistemas operativos.

Podemos bajarnos las últimas versiones de Apache y PHP (tanto para Linux/Unix como para Windows) de su webs oficiales http://www.apache.org y http://www.php.net. En julio de 2004 apareció Apache 2.0. y ya disponemos de la versión estable de PHP 5.0.

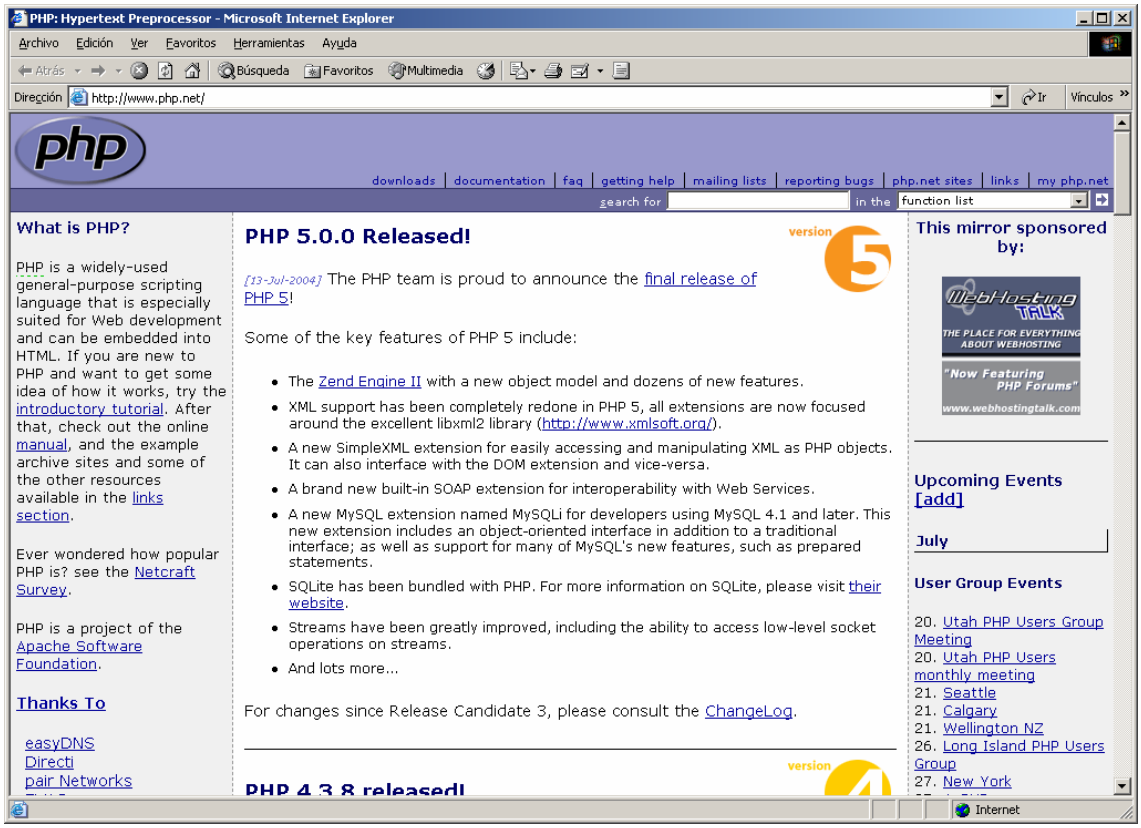


En la web de Apache también podemos encontrar una amplia documentación sobre Apache 2.0. y su versión anterior (Apache 1.3.), como tutoriales, manuales de referencia, preguntas frecuentes, notas específicas sobre cada plataforma, guía del usuario, etc., y además está disponible en varios idiomas. La documentación del servidor Apache 2.0 en español se encuentra en la dirección http://httpd.apache.org/docs-2.0/es/ :

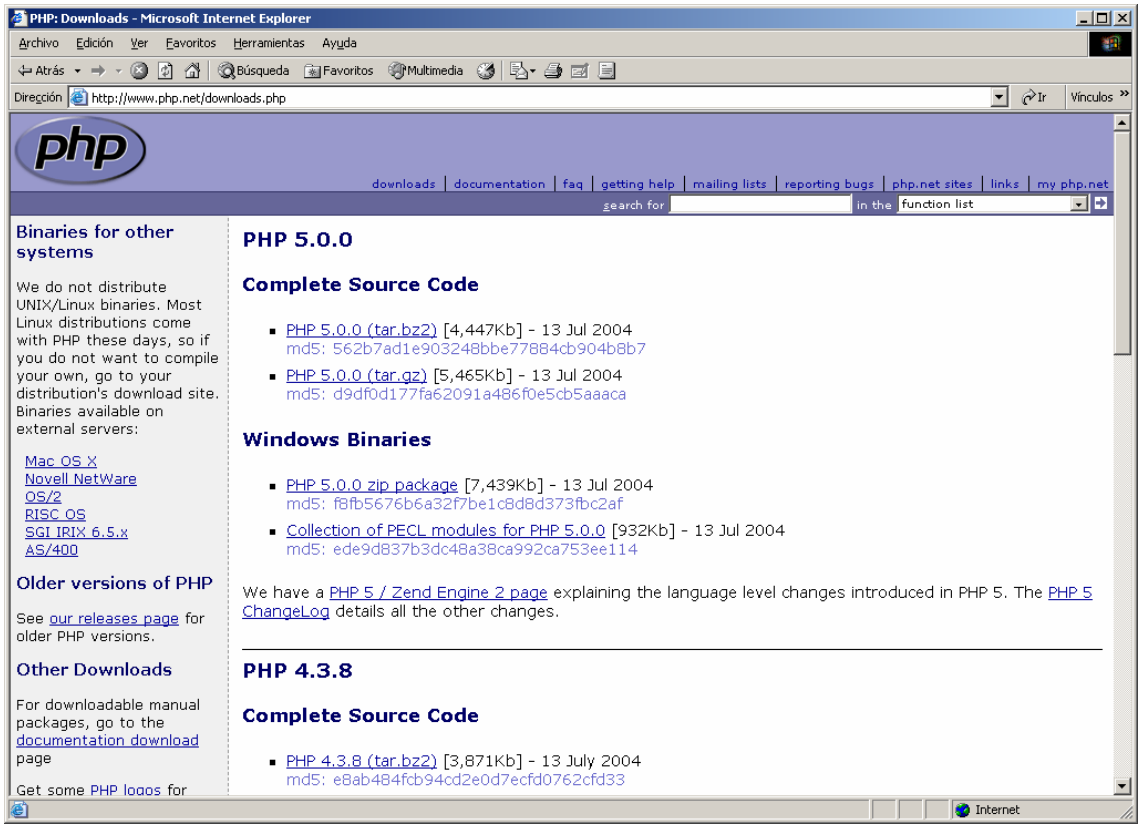
Podemos bajarnos Apache de la dirección http://httpd.apache.org/download.cgi, desde donde podemos elegir alguno de los mirrors disponibles para hacer la descarga:

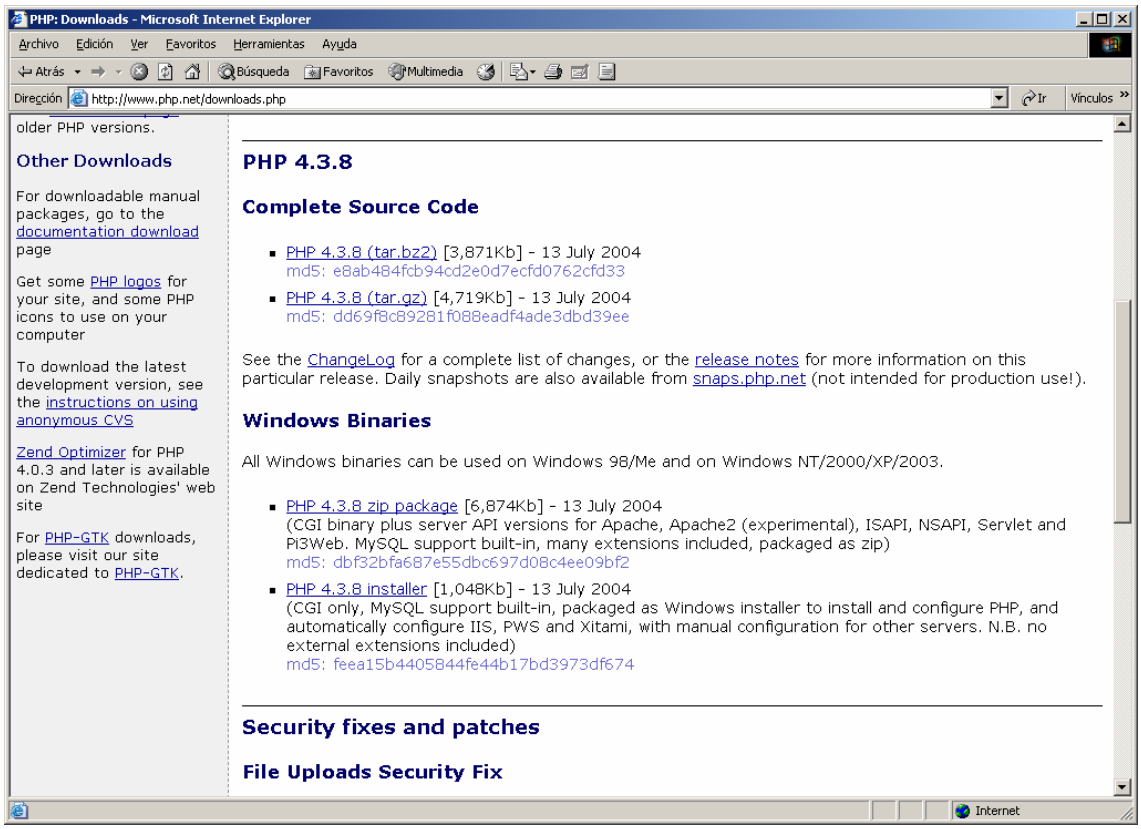


A mediados de julio de 2004, PHP anunciaba la salida de la versión estable de PHP 5.0., la versión anterior estable era la 4.3.8.:

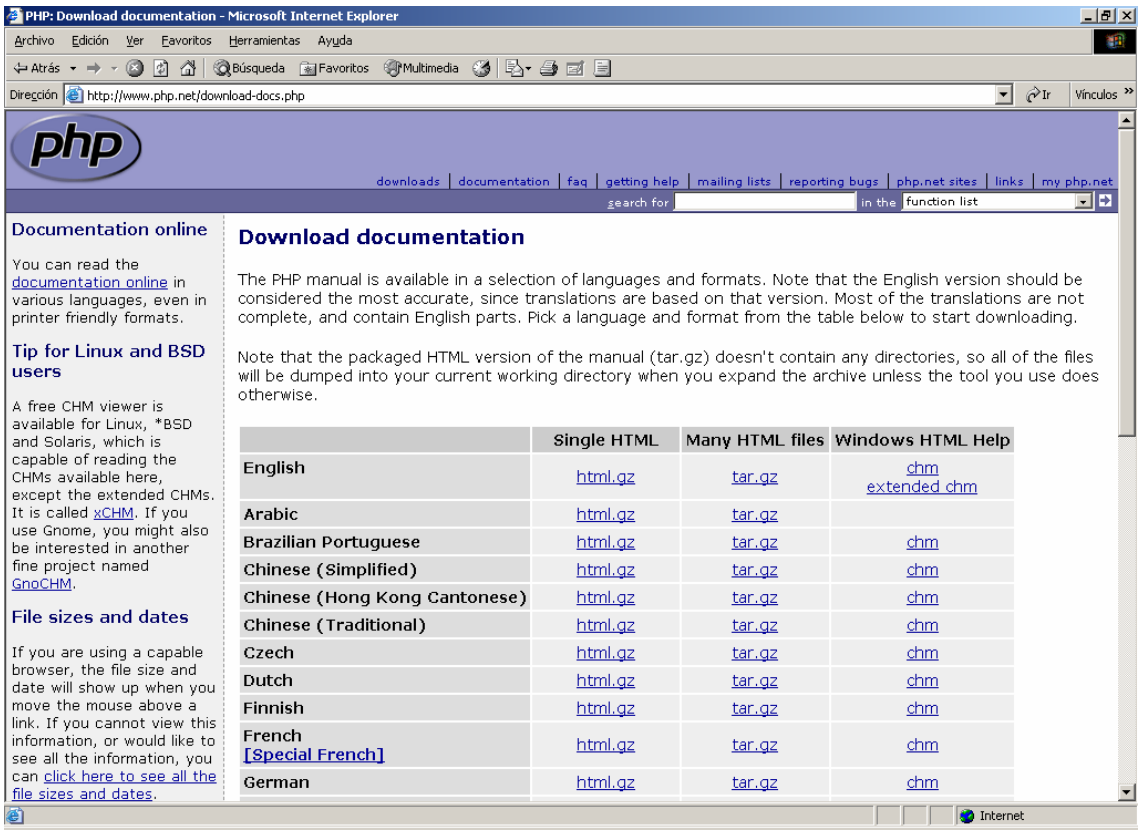


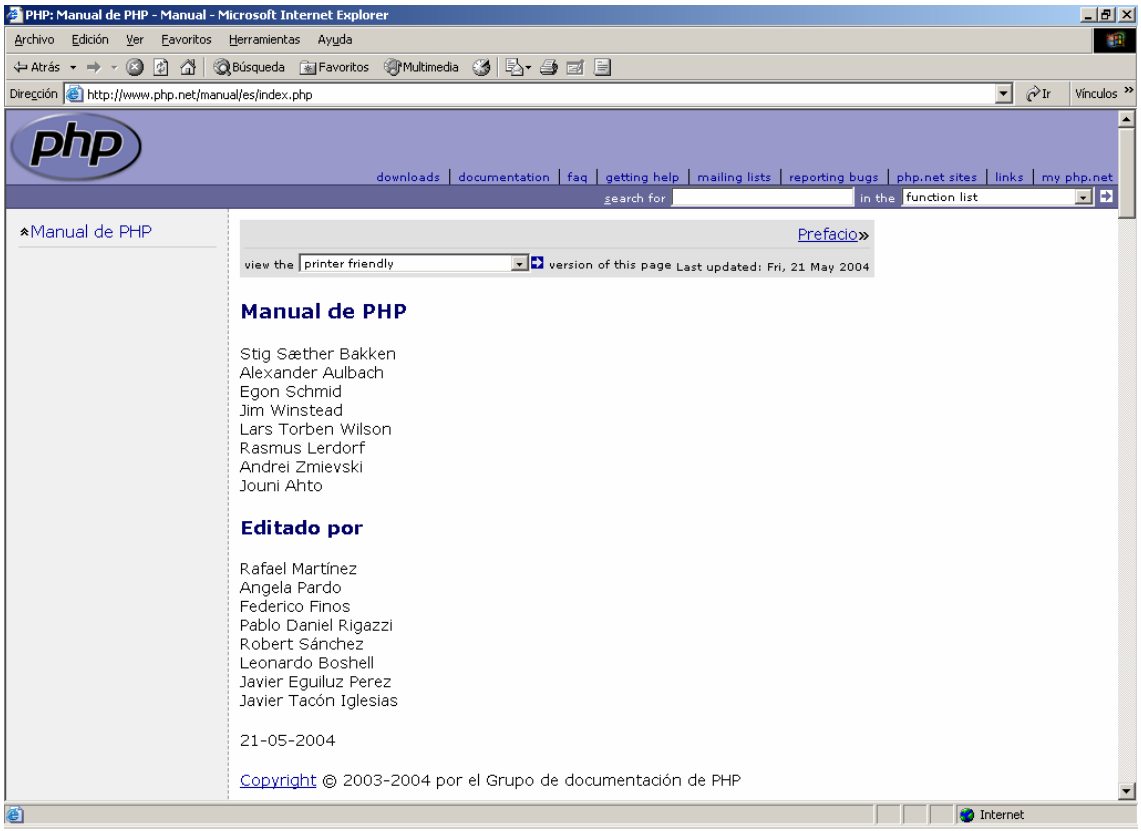
Desde la dirección http://www.php.net/downloads.php podemos bajarnos las últimas versiones disponibles de PHP, tanto los binarios para Windows, en formato zip y ejecutable, como las fuentes para Linux/Unix en tar.gz y tar.bz2. También hay enlaces hacia las páginas de documentación y las páginas para bajarse versiones para otros sistemas y versiones anteriores de PHP:

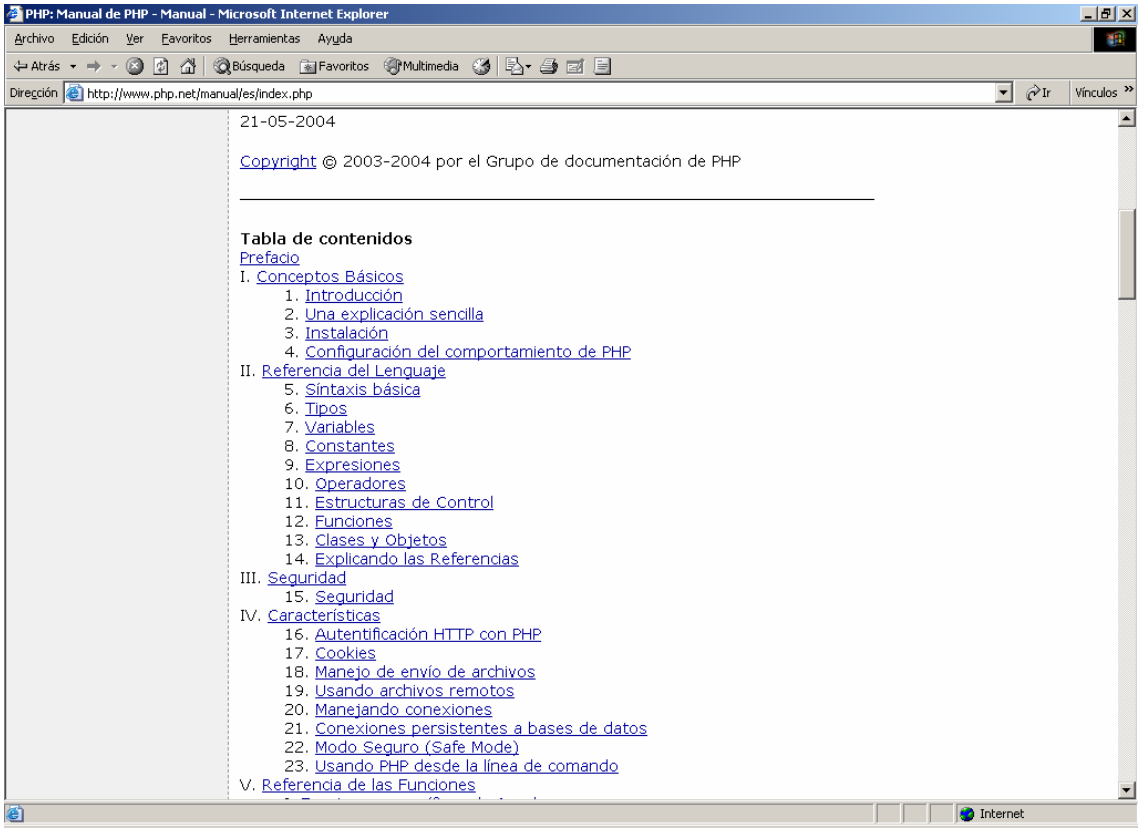

PHP: Downloads - Microsoft Internet Explorer
Archivo Edición Ver Favoritos Herramientas Ayuda
Atrás Búsqueda Favoritos Multimedia
Dirección http://www.php.net/downloads.php
Ir Vínculos
php
downloads | documentation | faq | getting help | mailing lists | reporting bugs | php.net sites | links | my php.net
search for in the function list
Binaries for other systems
We do not distribute UNIX/Linux binaries. Most Linux distributions come with PHP these days, so if you do not want to compile your own, go to your distribution's download site. Binaries available on external servers:
Mac OS X
Novell NetWare
OS/2
RISC OS
SGI IRIX 6.5.x
AS/400
Older versions of PHP
See our releases page for older PHP versions.
Other Downloads
For downloadable manual packages, go to the documentation download page
Get some PHP logos for
PHP 5.0.0
Complete Source Code
PHP 5.0.0 (tar.bz2) [4,447Kb] - 13 Jul 2004
md5: 562b7ad1e903248bbe77884cb904b8b7
PHP 5.0.0 (tar.gz) [5,465Kb] - 13 Jul 2004
md5: d9df0d177fa62091a486f0e5cb5aaaca
Windows Binaries
PHP 5.0.0 zip package [7,439Kb] - 13 Jul 2004
md5: f8fb5676b6a32f7be1c8d8d373fbc2af
Collection of PECL modules for PHP 5.0.0 [932Kb] - 13 Jul 2004
md5: ede9d837b3dc48a38ca992ca753ee114
We have a PHP 5 / Zend Engine 2 page explaining the language level changes introduced in PHP 5. The PHP 5 ChangeLog details all the other changes.
PHP 4.3.8
Complete Source Code
PHP 4.3.8 (tar.bz2) [3,871Kb] - 13 July 2004
md5: e8ab484fcb94cd2e0d7ecfd0762cfd33
Internet


PHP: Downloads - Microsoft Internet Explorer
Archivo Edición Ver Favoritos Herramientas Ayuda
Atrás Búsqueda Favoritos Multimedia
Dirección http://www.php.net/downloads.php
Ir Vínculos
older PHP versions.
Other Downloads
For downloadable manual packages, go to the documentation download page
Get some PHP logos for your site, and some PHP icons to use on your computer
To download the latest development version, see the instructions on using anonymous CVS
Zend Optimizer for PHP 4.0.3 and later is available on Zend Technologies' web site
For PHP-GTK downloads, please visit our site dedicated to PHP-GTK.
PHP 4.3.8
Complete Source Code
PHP 4.3.8 (tar.bz2) [3,871Kb] - 13 July 2004
md5: e8ab484fcb94cd2e0d7ecfd0762cfd33
PHP 4.3.8 (tar.gz) [4,719Kb] - 13 July 2004
md5: dd69f8c89281f088eadf4ade3dbd39ee
See the ChangeLog for a complete list of changes, or the release notes for more information on this particular release. Daily snapshots are also available from snaps.php.net (not intended for production use!).
Windows Binaries
All Windows binaries can be used on Windows 98/Me and on Windows NT/2000/XP/2003.
PHP 4.3.8 zip package [6,874Kb] - 13 July 2004
(CGI binary plus server API versions for Apache, Apache2 (experimental), ISAPI, NSAPI, Servlet and Pi3Web. MySQL support built-in, many extensions included, packaged as zip)
md5: dbf32bfa687e55dbc697d08c4ee09bf2
PHP 4.3.8 installer [1,048Kb] - 13 July 2004
(CGI only, MySQL support built-in, packaged as Windows installer to install and configure PHP, and automatically configure IIS, PWS and Xitami, with manual configuration for other servers. N.B. no external extensions included)
md5: feea15b4405844fe44b17bd3973df674
Security fixes and patches
File Uploads Security Fix
Internet

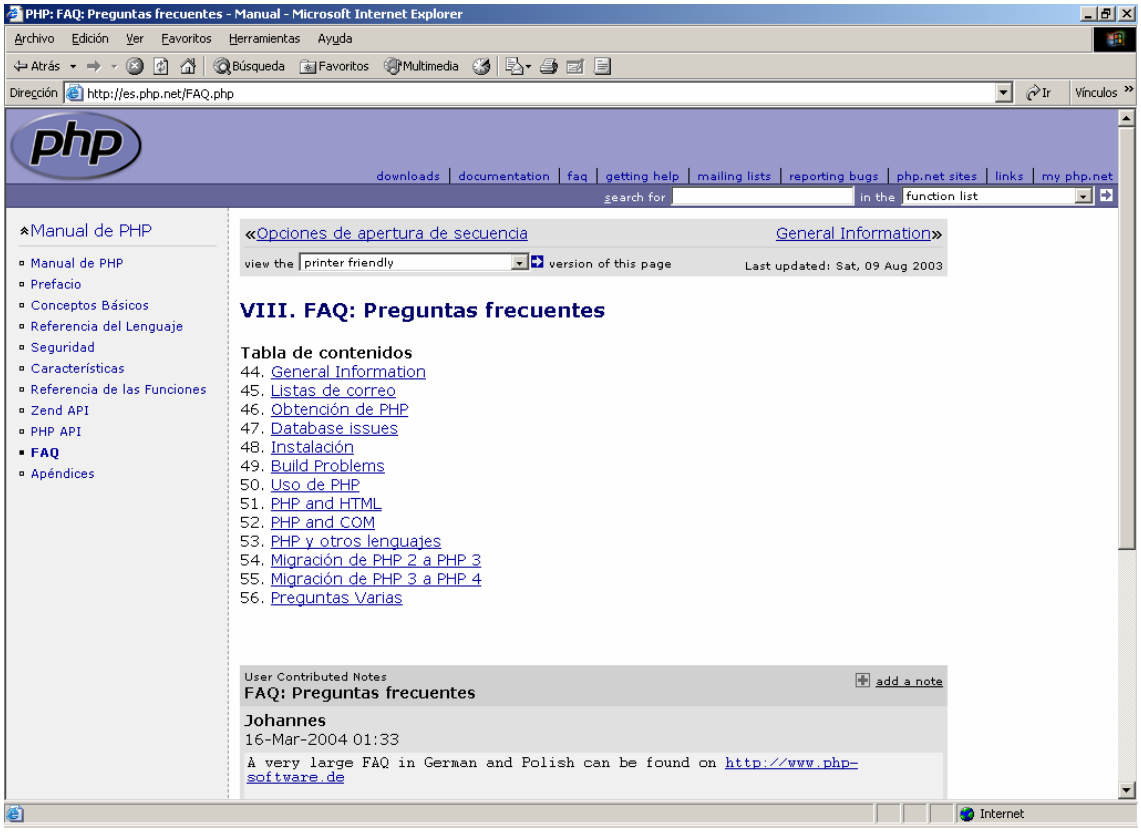
Por otro lado, desde esta dirección, http://www.php.net/downloads-docs.php, podemos bajarnos los manuales de referencia más actualizados de PHP en múltiples idiomas y formatos, como páginas HTML o en formato de la ayuda de Windows .chm. También existen versiones online de los manuales de referencia en varios idiomas; podemos acceder al manual en español en la dirección http://www.php.net/manual/es/index.php, donde el equipo de documentación de PHP mantiene bastante actualizada una traducción del manual de referencia:



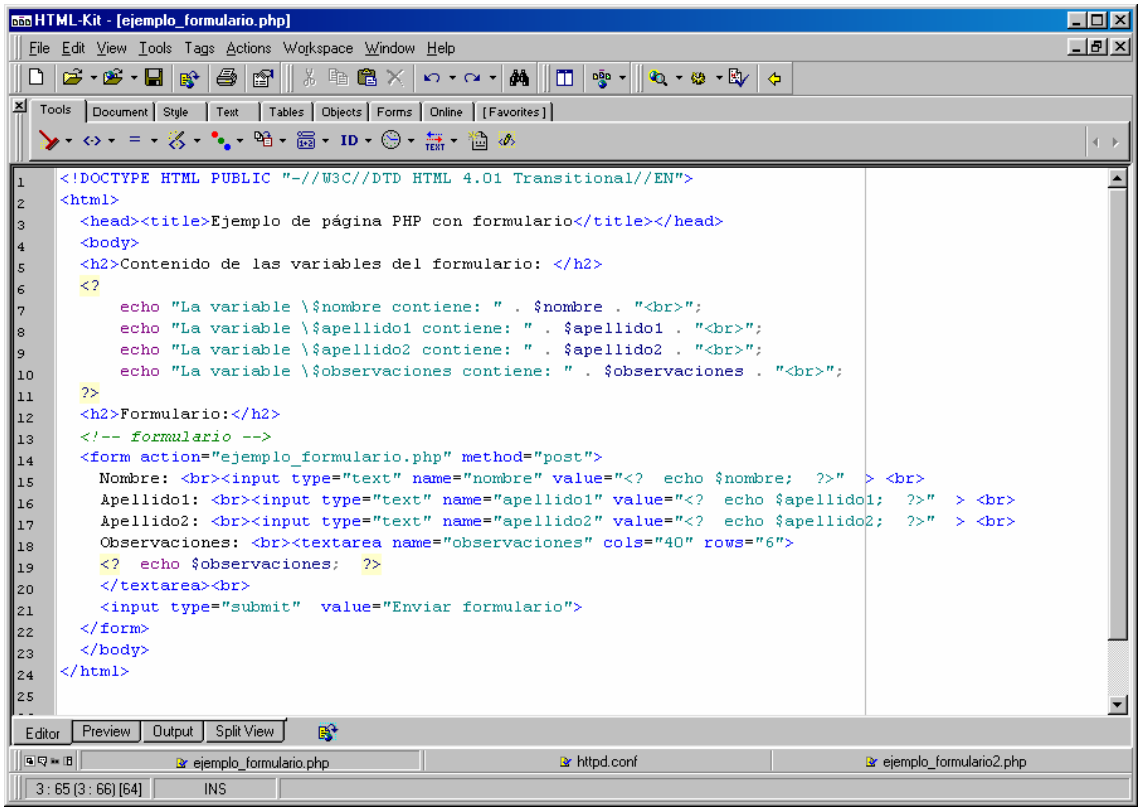

PHP: Manual de PHP - Manual - Microsoft Internet Explorer
Archivo Edición Ver Favoritos Herramientas Ayuda
Atrás Búsqueda Favoritos Multimedia
Dirección http://www.php.net/manual/es/index.php
Ir Vínculos
php
downloads | documentation | faq | getting help | mailing lists | reporting bugs | php.net sites | links | my php.net
search for in the function list
Manual de PHP
Prefacio»
view the printer friendly version of this page Last updated: Fri, 21 May 2004
Manual de PHP
Stig Sæther Bakken
Alexander Aulbach
Egon Schmid
Jim Winstead
Lars Torben Wilson
Rasmus Lerdorf
Andrei Zmievski
Jouni Ahto
Editado por
Rafael Martínez
Angela Pardo
Federico Finos
Pablo Daniel Rigazzi
Robert Sánchez
Leonardo Boshell
Javier Eguiluz Perez
Javier Tacón Iglesias
21-05-2004
Copyright © 2003-2004 por el Grupo de documentación de PHP
Internet


PHP: Manual de PHP - Manual - Microsoft Internet Explorer
Archivo Edición Ver Favoritos Herramientas Ayuda
Atrás Búsqueda Favoritos Multimedia
Dirección http://www.php.net/manual/es/index.php
Ir Vínculos
21-05-2004
Copyright © 2003-2004 por el Grupo de documentación de PHP
Tabla de contenidos
Prefacio
I. Conceptos Básicos
1. Introducción
2. Una explicación sencilla
3. Instalación
4. Configuración del comportamiento de PHP
II. Referencia del Lenguaje
5. Sintaxis básica
6. Tipos
7. Variables
8. Constantes
9. Expresiones
10. Operadores
11. Estructuras de Control
12. Funciones
13. Clases y Objetos
14. Explicando las Referencias
III. Seguridad
15. Seguridad
IV. Características
16. Autentificación HTTP con PHP
17. Cookies
18. Manejo de envío de archivos
19. Usando archivos remotos
20. Manejando conexiones
21. Conexiones persistentes a bases de datos
22. Modo Seguro (Safe Mode)
23. Usando PHP desde la línea de comando
V. Referencia de las Funciones
Internet

También podemos consultar la sección de preguntas frecuentes (FAQS) en español, en la dirección http://www.php.net/FAQ.php:



Para escribir el código PHP podemos usar cualquier editor de texto plano, como el vi en sistemas Unix/Linux. También podemos escribir las páginas PHP utilizando alguno de los programas más conocidos para generar páginas web, como Dreamweaver, o los editores Credit o Html-Kit. Es conveniente que el editor pueda distinguir entre el código PHP y HTML (ya que el código PHP coexiste con el código HTML, CSS y JavaScript habitual de las páginas web) para facilitar la escritura y depuración de nuestras páginas.



Ejemplo de editor que distingue la sintaxis de PHP

1.3. INSTALACIÓN Y CONFIGURACIÓN EN LINUX / UNIX

Una vez que sabemos dónde podemos descargarnos las versiones de Apache y PHP, y dónde podemos encontrar la documentación de referencia, vamos a hacer una instalación básica en un sistema Linux/Unix. Para dar soporte a funcionalidades específicas de nuestro sistema, como acceder a una base de datos concreta, debemos remitirnos a la documentación de estas aplicaciones para obtener más detalles de su instalación y configuración y a la referencia de PHP.

Como resultado de la descarga tendremos unos ficheros empaquetados y comprimidos como por ejemplo apache_1.3.31.tar.gz o httpd-2.0.49.tar.gz y php-4.3.7.tar.gz o php-5.0.tar.gz, según la versión que nos bajemos.

Para poder completar todo el proceso de instalación deberemos tener acceso como root a la máquina Linux/Unix. Es aconsejable leer los ficheros README e INSTALL existentes por si hubiera alguna nota especial en la versión que estamos utilizando para tenerla en cuenta en el proceso de compilación e instalación, sobre compatibilidad con el sistema u otras aplicaciones, o en caso de encontrar problemas.

NOTA: En la documentación de PHP se recomienda no utilizar Apache 2.0. y PHP en sistemas que estén en producción, tanto en Unix como en Windows. Con Apache 2.0. se recomienda usar la versión de PHP 4.3.0. o posterior para evitar problemas. La versión estable de PHP 5.0. ha salido a mediados de julio de 2004. Para nuestro ejemplo de instalación usaremos la versión 1.3.31. de Apache y la 4.3.7. de PHP.

Vamos a hacer la instalación en /usr/local. Supongamos que tenemos los .tar.gz en este directorio. Nos situamos primero en él y descomprimimos los paquetes:

```
cd /usr/local
tar zxvf apache-1.3.31.tar.gz
tar zxvf php-4.3.7.tar.gz
```

Creamos enlaces a los directorios que se crean al descomprimir y que contienen el código fuente de Apache y PHP para facilitar la instalación:

```
ln –s /usr/local/apache-1.3.31  /usr/local/apache
ln -s /usr/local/php-4.3.7 /usr/local/php
```

Indicamos las opciones de compilación de los fuentes de PHP. En este paso podemos configurar PHP con diferentes opciones, como por ejemplo qué extensiones estarán disponibles. Podemos ejecutar:

```
./configure --help
```

para obtener una lista de las características de configuración de PHP. En nuestro ejemplo configuramos PHP con Apache. Si quisiéramos que tuviera soporte MySQL deberíamos añadir la opción *--with-mysql*, para el soporte de PostgreSQL, la opción *--with-pgsql*; para desactivar la etiqueta corta de comienzo del código PHP <? la opción *--disable-short-tags*, etc. En la referencia de PHP podemos consultar el significado de las múltiples opciones de compilación y configuración disponibles.

Indicamos las opciones de configuración con el programa *configure*, compilamos los fuentes (con el programa *make*) y lo instalamos (con *make install*) como módulo de Apache. Debemos tener un compilador de C y ser root para ejecutar *make install*:

```
cd /usr/local/php
./configure --with-apache=/usr/local/apache
make
make install
```

Copiamos el fichero de configuración de PHP que viene con la distribución (php.ini-dist) con el nombre php.ini (por seguridad, para conservar la configuración por defecto que trae PHP tras la instalación):

```
cp php.ini-dist /usr/local/lib/php.ini
```

Configuramos, compilamos e instalamos Apache (indicándole el módulo de PHP):

```
cd /usr/local/apache
./configure --prefix=/usr/local/apache \
        --activate-module=src/modules/php4/libphp4.a
make
make install
```

Para que Apache sepa qué extensiones de ficheros debe pasar al intérprete de PHP, debemos indicárselas en su fichero de configuración, httpd.conf:

```
AddType application/x-httpd-php .php3
AddType application/x-httpd-php .php4
AddType application/x-httpd-php .php5
AddType application/x-httpd-php .php
AddType application/x-httpd-php .phtml
AddType application/x-httpd-php .inc
```

Para detener Apache podemos ejecutar:

```
/usr/local/apache/bin/apachectl stop
```

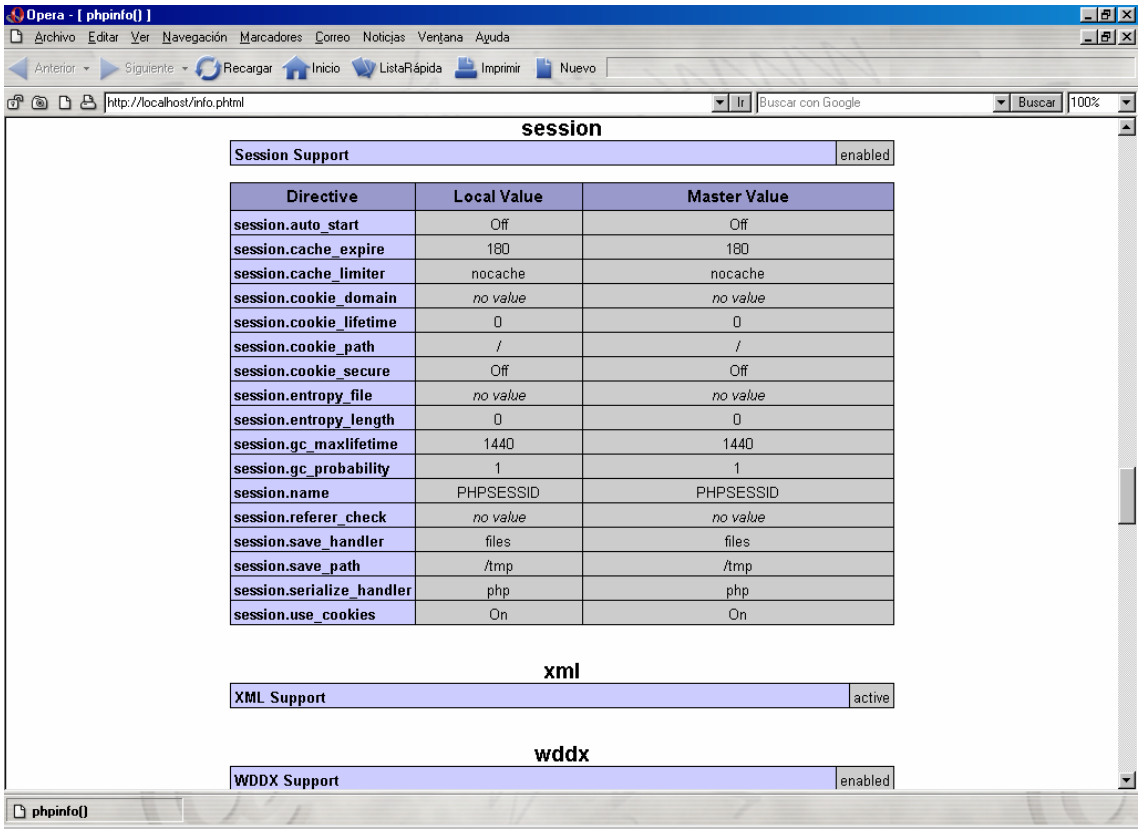
Y para arrancar Apache:

```
/usr/local/apache/bin/apachectl start
```

Podemos comprobar que la instalación ha ido bien y que tenemos PHP disponible ejecutando un script llamado info.php por ejemplo, que contenga la siguiente línea:

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

Debemos colocarlo en el directorio de documentos de Apache (por defecto es htdocs) y llamarlo desde el navegador. Si todo funciona bien obtendremos una página que muestra las variables y la configuración de PHP:

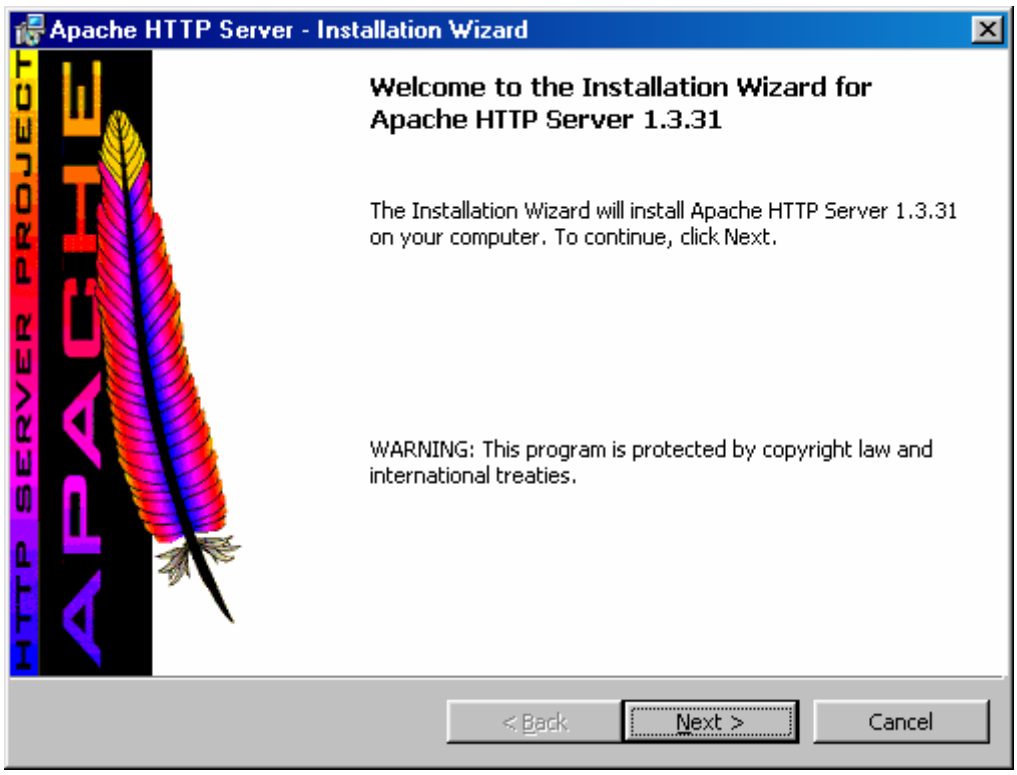


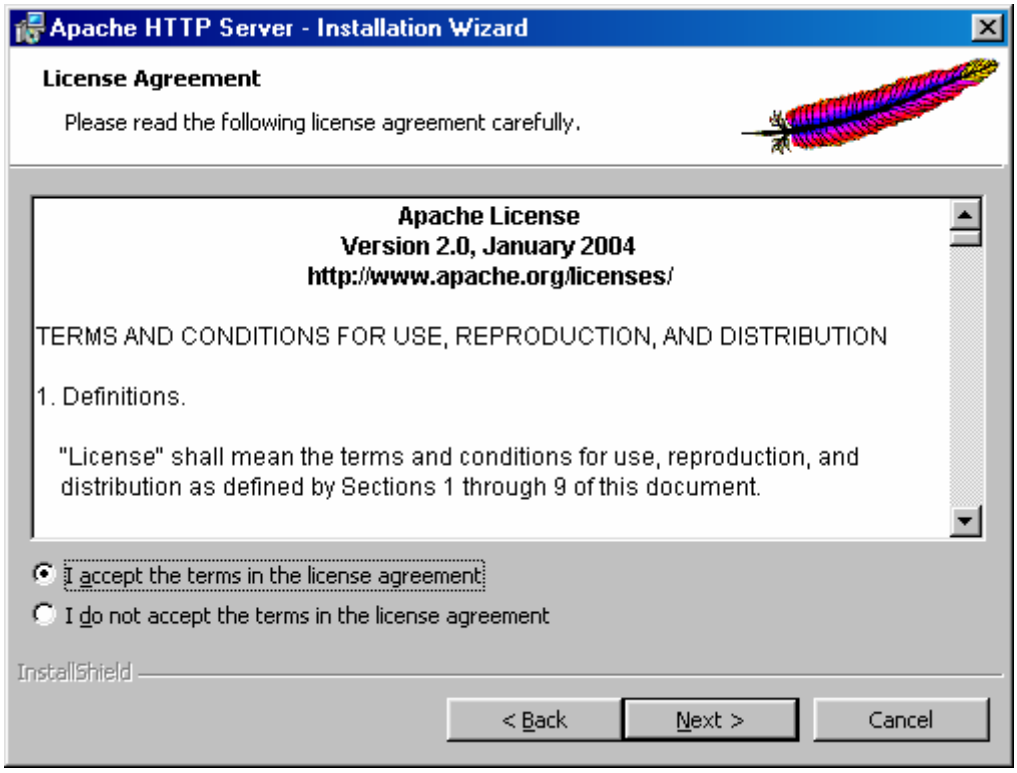
1.4. INSTALACIÓN Y CONFIGURACIÓN EN WINDOWS

En este apartado veremos la instalación y configuración de los binarios de Apache y PHP en Windows. Las versiones de PHP para Windows suelen estar disponibles en formato de fichero comprimido .zip o en un ejecutable .exe, y Apache como un paquete instalador de Windows .msi o como un ejecutable .exe.

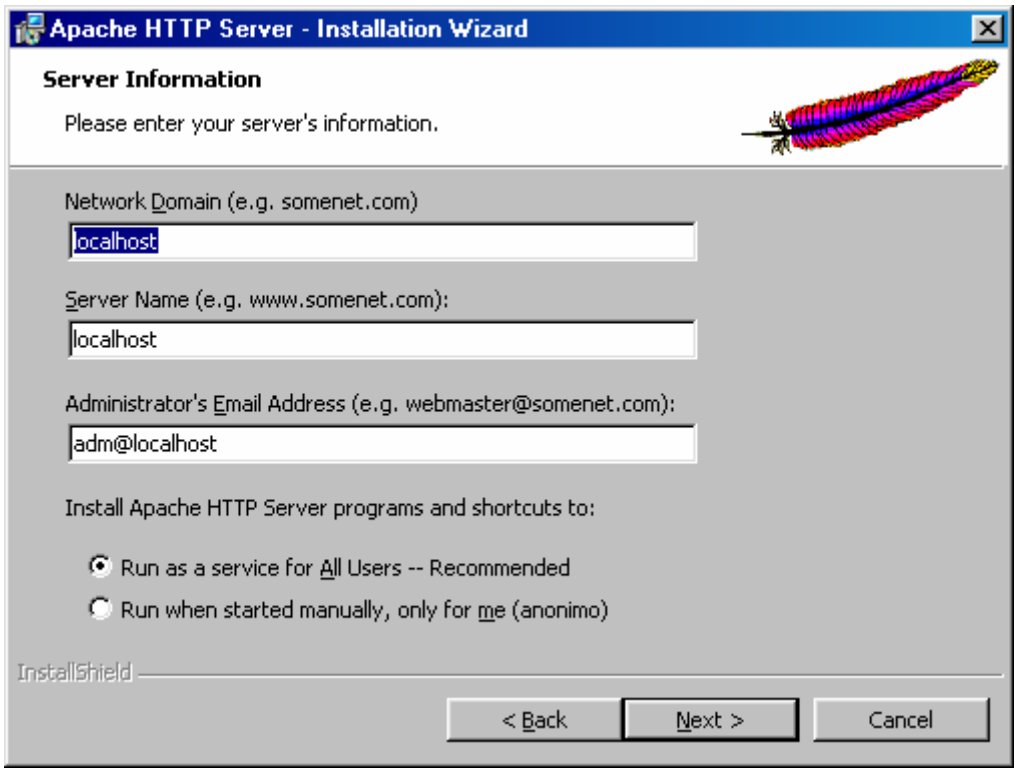
1.4.1. Instalación de Apache en Windows

La instalación de estos programas se hace de la forma habitual en Windows. Para Apache hacemos doble clic en su fichero de instalación (como apache-1.3.31-win32-x86-no_src.exe) y seguimos las instrucciones de las pantallas:

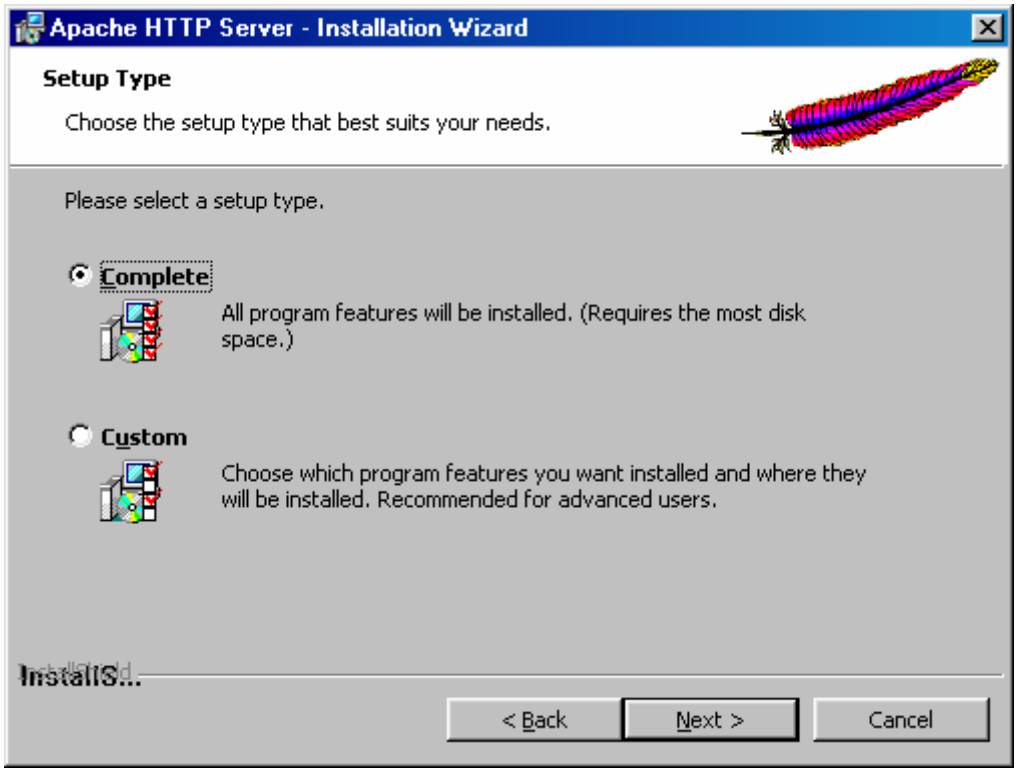


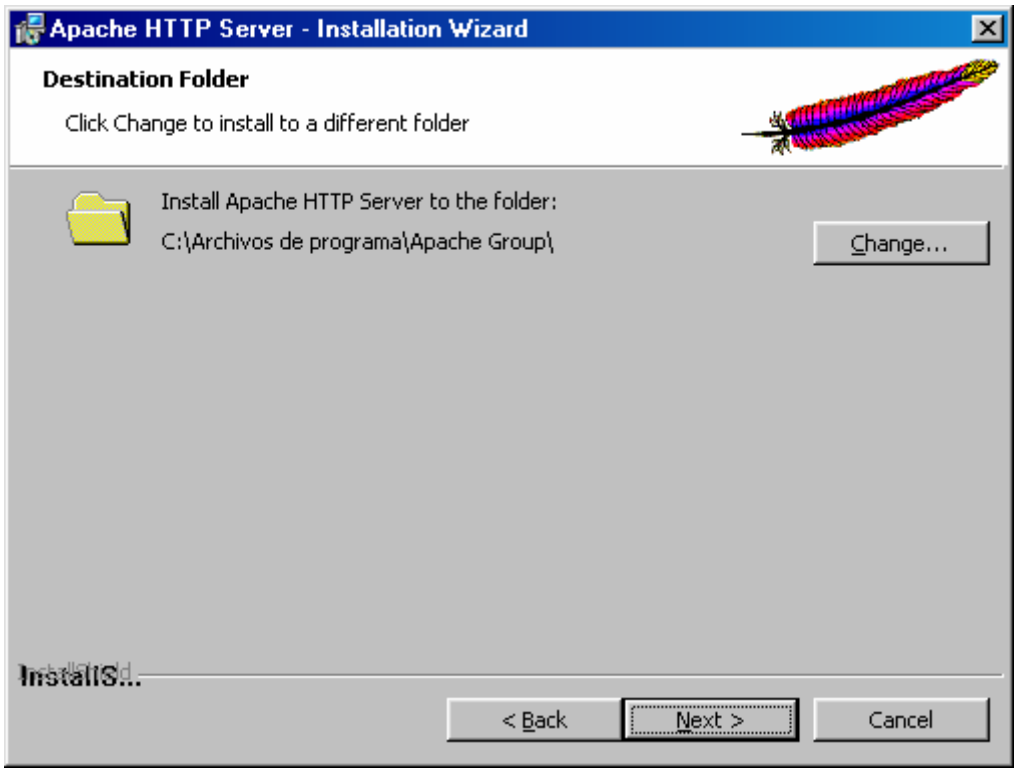


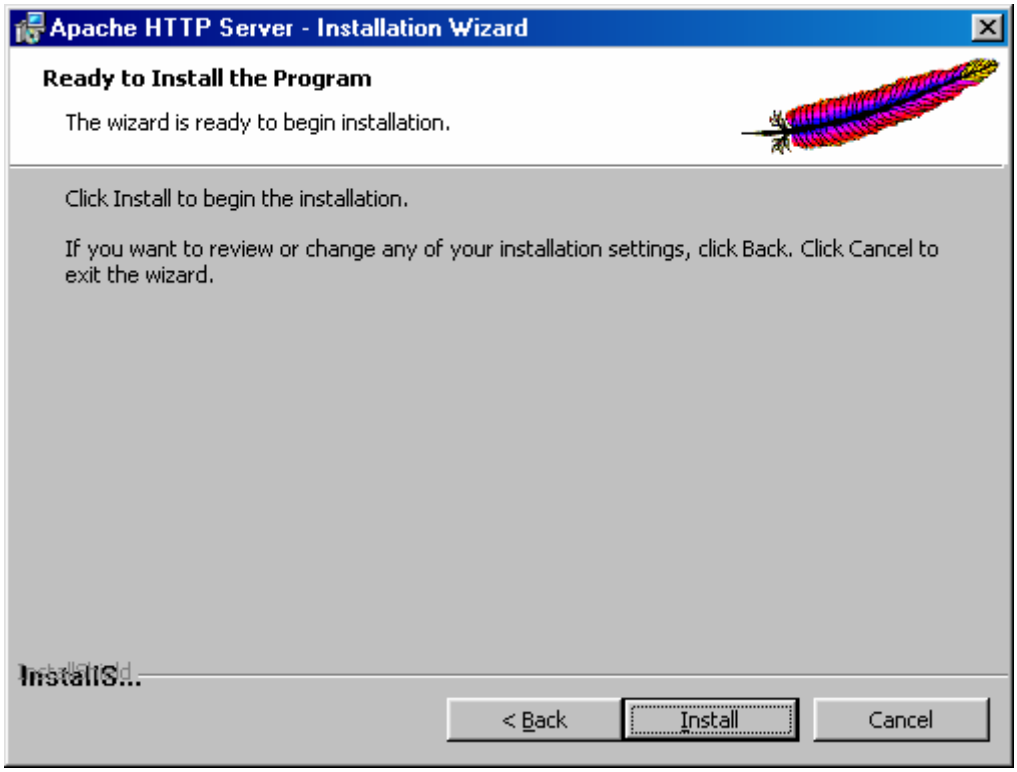
Nos solicita información del servidor:

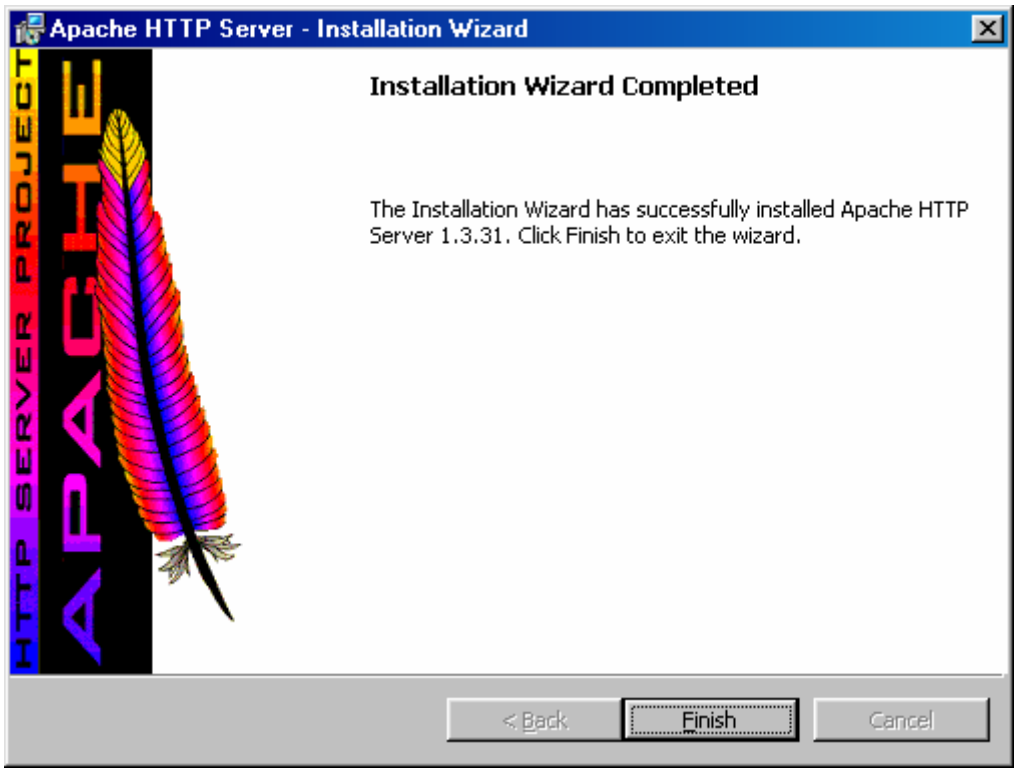


La versión completa incluye también la documentación de Apache:





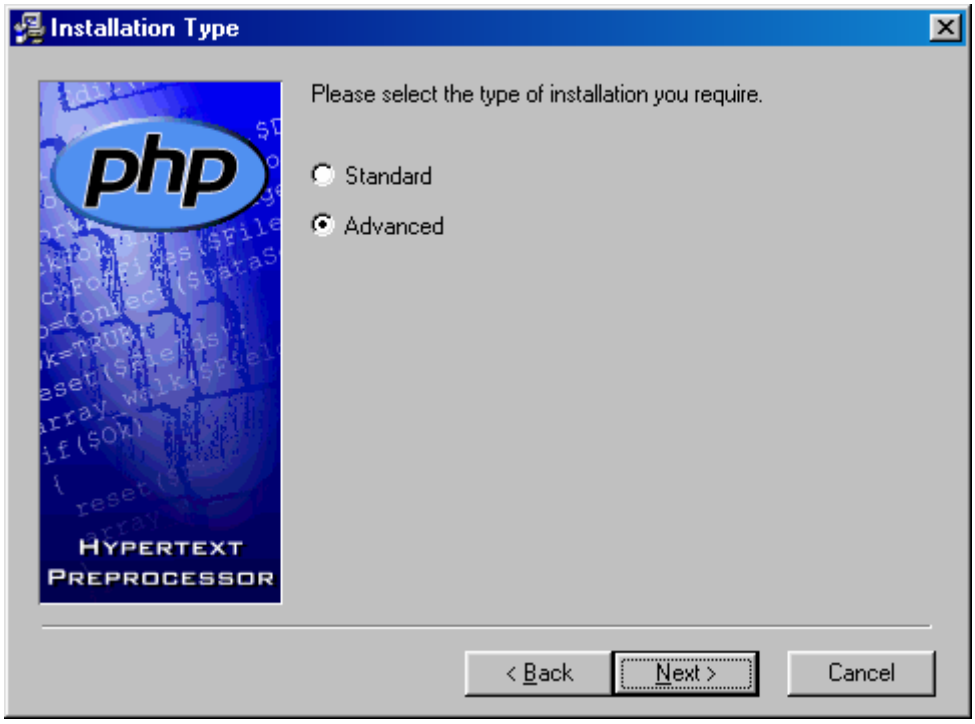

Apache HTTP Server - Installation Wizard
Ready to Install the Program
The wizard is ready to begin installation.
Click Install to begin the installation.
If you want to review or change any of your installation settings, click Back. Click Cancel to exit the wizard.
InstallS...
< Back
Install
Cancel


Apache HTTP Server - Installation Wizard
HTTP SERVER PROJECT
APACHE
Installation Wizard Completed
The Installation Wizard has successfully installed Apache HTTP Server 1.3.31. Click Finish to exit the wizard.
< Back
Finish
Cancel

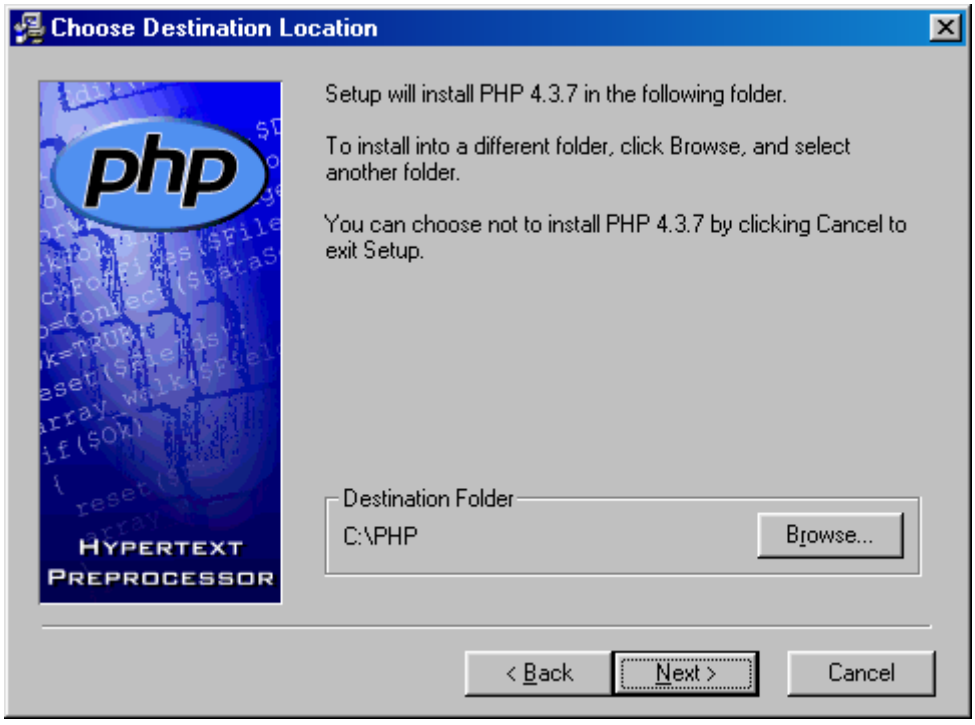
1.4.2. Instalación de PHP en Windows

Podemos instalar PHP para Windows de forma manual partiendo de un fichero .zip o usando un fichero ejecutable que nos guía en la instalación a través de un asistente. Este asistente nos muestra una serie de pantallas en las que solicita información para instalar PHP y establece los parámetros necesarios en el fichero php.ini y configura el servidor web para usar PHP. Al hacer doble clic sobre el ejecutable de PHP aparecerá el asistente y, deberíamos seguir las instrucciones de las sucesivas pantallas:

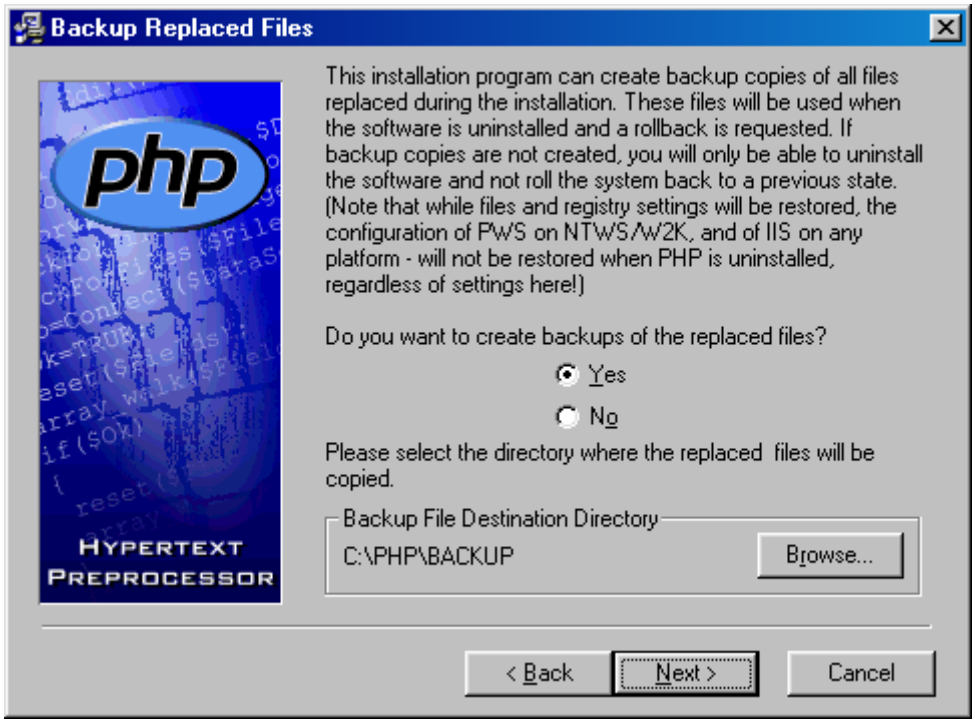
La instalación estándar suministra una configuración genérica y la avanzada permite seleccionar un mayor número de valores de configuración:



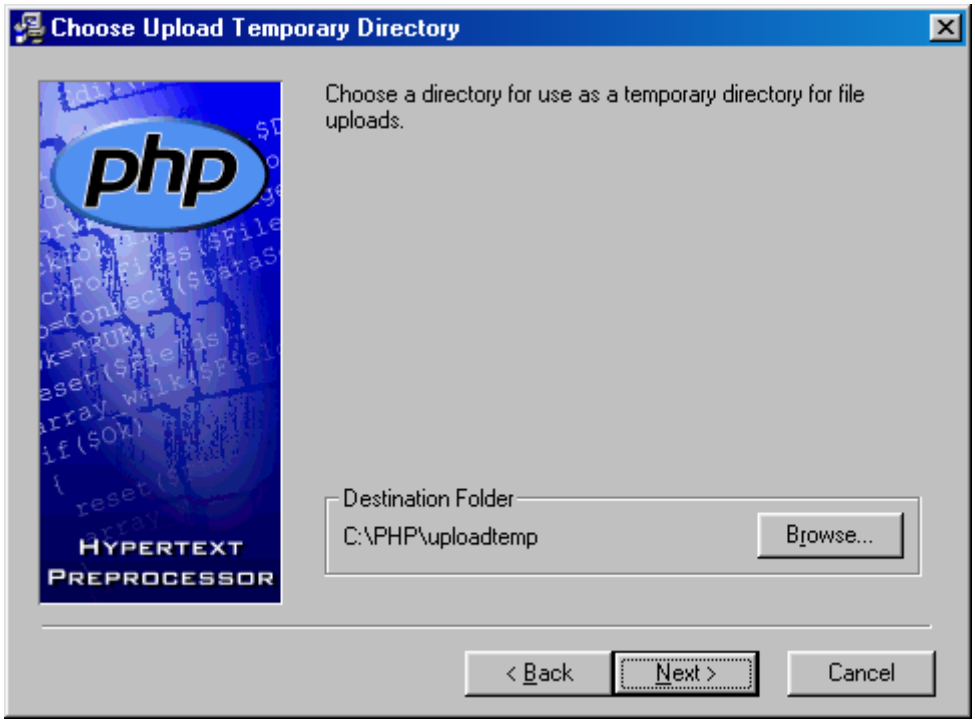
Elegimos el directorio de instalación:



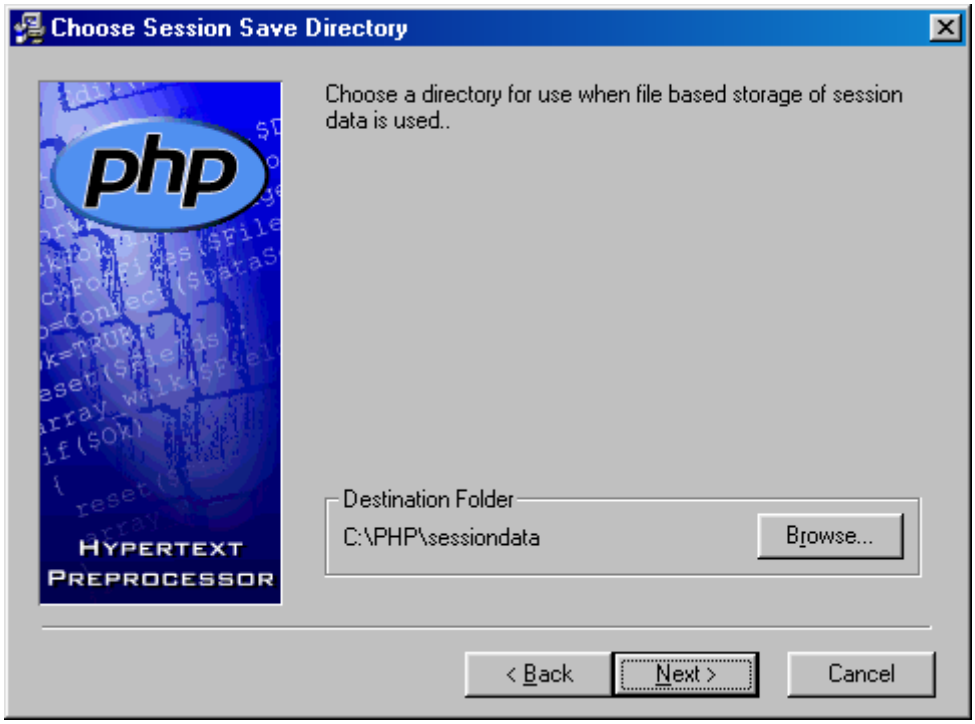
Podemos seleccionar hacer copia de seguridad de los ficheros modificados durante la instalación de PHP, que podrán ser usados durante la desinstalación de PHP.



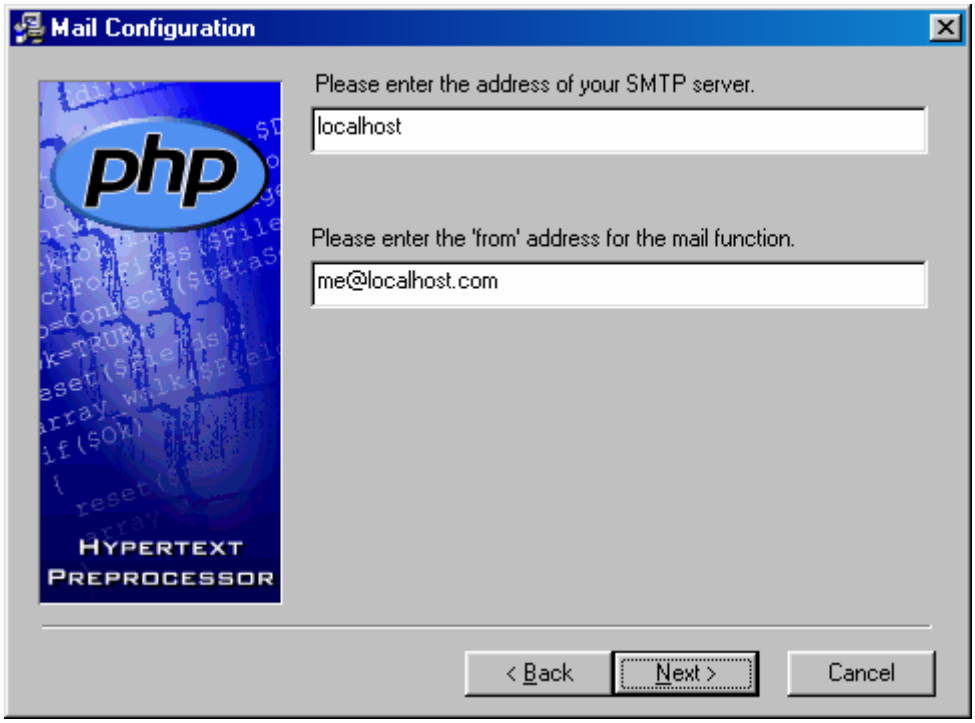
Seleccionamos el directorio temporal usado en la subida de ficheros desde páginas PHP al servidor Web:



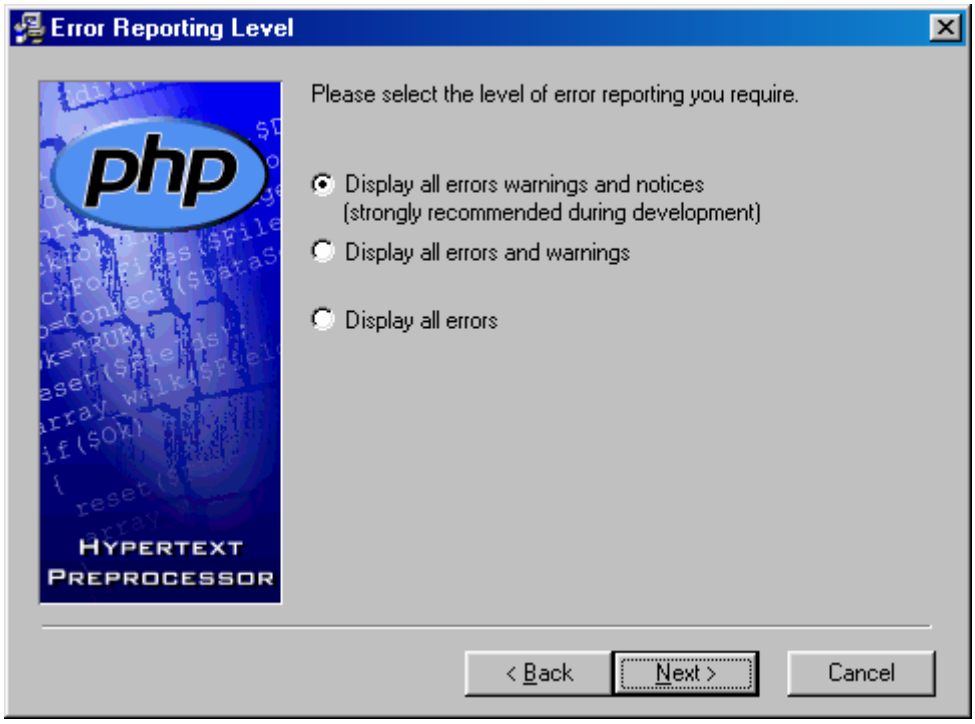
El directorio usado para mantener las sesiones de los clientes:



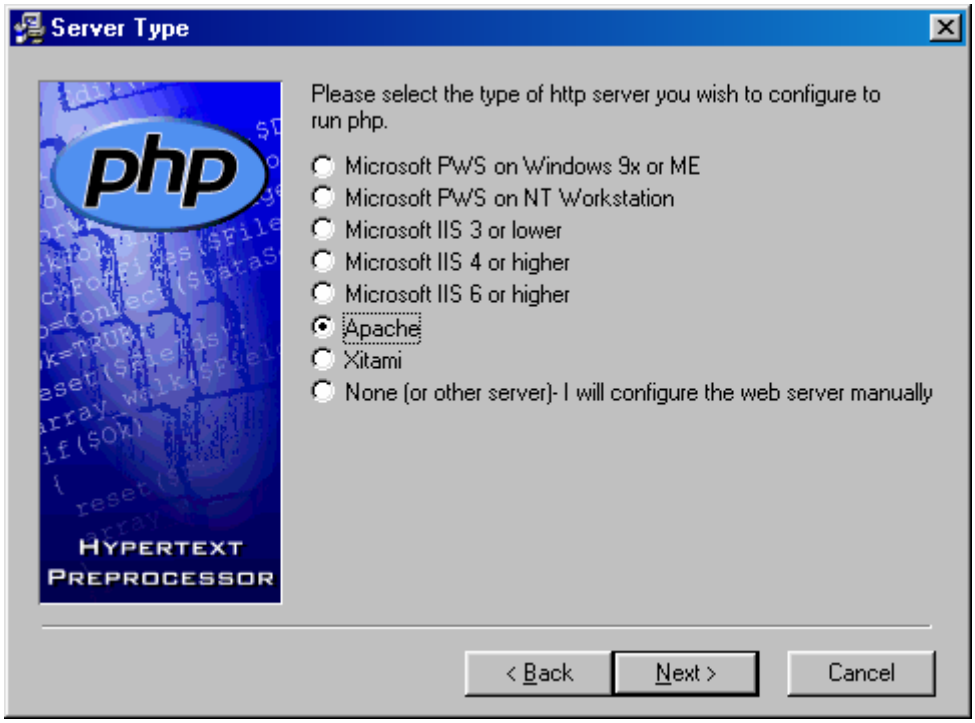
Nombre del servidor SMTP y dirección de correo usados en la función de envío de correos desde PHP:



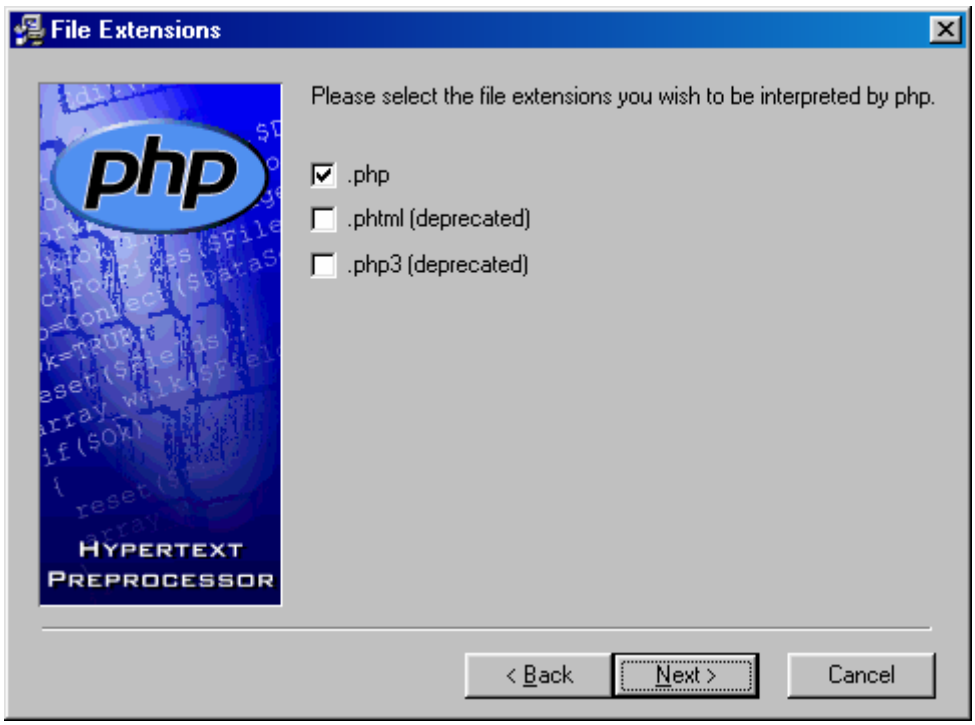
También podemos seleccionar el nivel de detalle de la información sobre los errores ocurridos en nuestras páginas PHP:

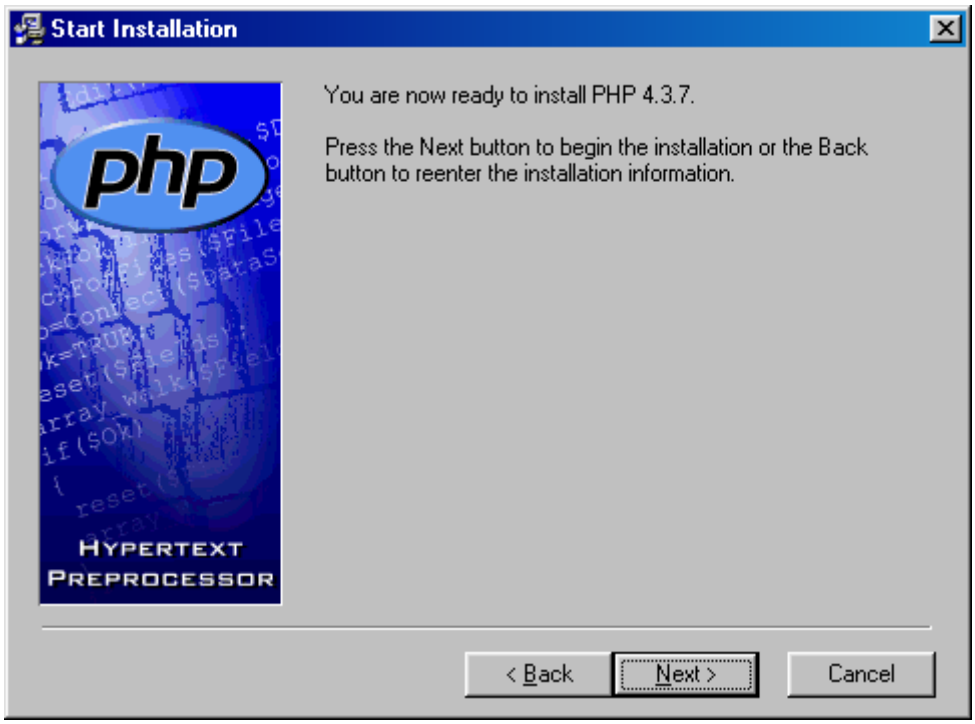


Asimismo, es posible concretar el servidor Web que tenemos instalado en nuestro sistema (PHP también puede ejecutarse junto a otros servidores Webs distintos a Apache):



Indicamos las extensiones de los ficheros que serán interpretados por PHP (para incluir más extensiones, al igual que para cambiar los anteriores valores de configuración introducidos a lo largo del asistente u otros, deberíamos editar el fichero php.ini de forma manual. Por ejemplo, se podrían incluir también las extensiones .php4, .php5, .inc, etc).





Si le es posible, este asistente intentará conectarse a alguna dirección web para configurar también automáticamente el servidor Web seleccionado.

Si instalamos PHP desde un fichero .zip, debemos descomprimir este fichero en un directorio, por ejemplo C:\php, y ajustar manualmente la configuración de Apache y PHP editando los ficheros httpd.conf y php.ini.

El fichero de configuración http.conf se halla en "C:\Archivos de Programas\Apache Group\Apache\ conf\" si hemos instalado Apache en el directorio por defecto. Al menos, debemos indicar la siguiente información:

- El nombre del servidor Web se encuentra en la línea que comienza por 'ServerName'. Si la línea está comentada comenzará por el carácter (#), así que lo quitamos:

```
ServerName http://localhost
```

- Indicamos el directorio de PHP:

```
ScriptAlias /php "C:\php"
```

- Definimos la extensión que tendrán nuestras páginas PHP:

```
AddType application/x-httpd-php .php
AddType application/x-httpd-php .phtml
AddType application/x-httpd-php .php4
AddType application/x-httpd-php .php5
AddType application/x-httpd-php .inc
```

- Asignamos la aplicación asociada a las extensiones PHP, el intérprete de PHP:

```
Action application/x-httpd-php "/php/php.exe"
```

Por defecto, los ficheros que son accesibles desde el navegador se encuentran en la carpeta htdocs del directorio de instalación de Apache. Si queremos especificar otro directorio buscamos la línea que comienza por "*DocumentRoot"*:

```
DocumentRoot  "C:\wwwroot"
<Directory "C:\wwwroot">
.....
</Directory>
```

Para configurar PHP, localizamos el fichero de configuración php.ini-dist que aparecerá en el directorio donde hemos descomprimido el .zip, y lo renombramos a php.ini, lo editamos con un editor de texto plano y hacemos los siguientes cambios:

- Buscamos la expresión *"extension_dir"* para indicarle el directorio de PHP:

```
extension_dir = C:\php
```

- Indicamos el directorio donde se buscarán los ficheros incluidos en las páginas PHP. Para ello buscamos la expresión *"include_path"*:

```
include_path = C:\Archivos de Programas\Apache Group\Apache\
include\
```

- Para añadir el soporte de alguna extensión en Windows, debemos localizar la dll correspondiente y quitarle el carácter (;) del comienzo de la línea para que deje de estar comentada, por ejemplo, si quisiéramos el soporte para MySQL, descomentaríamos la línea:

```
;extension = php_mysql.dll
```

  Hay múltiples extensiones, como *oracle.dll*, *pgsql.dll*, *php_sockets.dll*, *php_imap.dll*, *php_pdf.dll*, para distintas funcionalidades. Debemos consultar la documentación de las aplicaciones y de PHP para configurar las extensiones que necesitemos adecuadamente, por ejemplo, para activar la extensión de Oracle, deberemos tener instalada y configurada correctamente esta base de datos para acceder a ella desde nuestras páginas PHP.

- Copiamos el fichero php.ini en el directorio de Windows.

Para comprobar que todo funciona bien, arrancamos Apache (lo podemos hacer desde el ejecutable de su directorio de instalación, por defecto será C:\Archivos de Programas\Apache Group\ Apache\apache.exe, o desde el menú de inicio: Inicio->Programas->Apache HTTP Server->Start Apache. Podemos crear un fichero .php de prueba con la siguiente línea para comprobar que se está interpretando correctamente el código PHP:

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

Colocamos este script PHP en el directorio de documentos de Apache, y lo visualizamos a través del navegador. Debemos obtener una página con la configuración y valor de las variables de PHP.

RECUERDE

- PHP es un lenguaje de programación con sintaxis similar a los lenguajes C y Perl. Se interpreta por un servidor web Apache bajo sistemas Unix/Linux, aunque también han salido versiones para sistemas Windows.

- En las webs oficiales de PHP y Apache, www.php.net y www.apache.org, podemos consultar una amplia documentación sobre la instalación, configuración y referencia de ambos, y también podemos descargarnos las últimas versiones disponibles para diferentes plataformas en varios formatos.

- Para sistemas Linux/Unix hemos visto una instalación común de Apache y PHP partiendo de los ficheros fuentes comprimidos y empaquetados, y para sistemas Windows a partir de los ficheros binarios.

- Las páginas PHP son páginas webs con extensión .php o .phtml (otras extensiones comunes son .php3, .php4, .php5 o .inc) que incluyen código HTML, JavaScript y PHP embebido en ellas, y al ejecutarlas, se genera código HTML dinámicamente.

- Al ejecutar una página PHP en el servidor web, como petición de un programa visualizador de páginas webs (un cliente, como un navegador Mozilla, Opera o Internet Explorer), el servidor web (generalmente Apache) reconoce que la página solicitada es una página PHP (por la extensión) y se la envía al intérprete PHP. Éste procesa el código y genera una página HTML que se devuelve al cliente, quien visualizará la página HTML resultante como cualquier otra página estática que le llegara.

- PHP dispone de un gran número de librerías de funciones para realizar operaciones avanzadas como acceso a bases de datos, comunicaciones, transferencia de ficheros, correo electrónico, etc.

Tema 2

Sintaxis y elementos del lenguaje

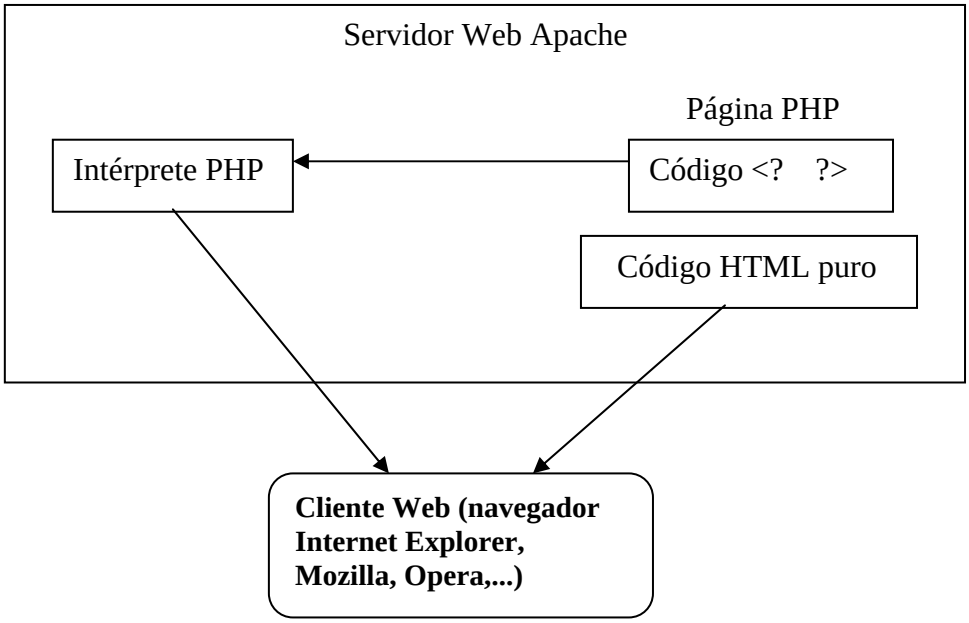
2.1. SEPARACIÓN E IDENTIFICACIÓN DEL CÓDIGO PHP

Debemos tener en cuenta que una página PHP no es un programa binario compilado que se ejecuta en la máquina cliente (la que solicita la página), sino que es un código interpretado en el servidor Web que genera el código HTML. Este código HTML tiene que ser interpretado a su vez en el cliente Web (el navegador, como Internet Explorer o Mozilla).

El objetivo de PHP es generar páginas HTML (que también pueden incluir código JavaScript y hojas de estilo). El código PHP convive en el mismo fichero con el lenguaje HTML, por lo que ambos códigos deben estar separados correctamente, para distinguir qué partes del fichero son interpretadas por PHP y cuáles se envían directamente al navegador cliente. Las formas más habituales para separar y delimitar el código PHP son:

<? Nuestro código php ?>
<?php Nuestro código php ?>
<script language="php"> Nuestro código php </script>

Estos controles de código PHP pueden ir en cualquier posición dentro de la página, pero no pueden anidarse. El servidor Apache pasa las partes de código comprendidas entre estas etiquetas (código PHP) al intérprete PHP para que lo procese (que a su vez generará código HTML), y aquellos otros trozos de código que estén fuera de estos controles, los suministra al cliente tal como los encuentra. El siguiente esquema ilustra la separación y procesamiento que hace el servidor Web Apache con el código de la página PHP que solicita un cliente:



2.2. VARIABLES

Los identificadores de variables en PHP siempre comienzan por el carácter $ y se distingue entre mayúsculas y minúsculas. No necesitan una declaración previa, como en otros lenguajes. La primera vez que aparece una variable supone una declaración implícita.

Algunos ejemplos de nombres de variables son:

$cantidad
$totales
$nombre
$a

Hay varios tipos de variables:

- números (enteros o de coma flotante)
- cadenas de caracteres
- arrays
- objetos

pero como no hay una declaración explícita de las variables que pueda definir qué tipo de datos va a contener, el propio PHP determina el tipo en función del contenido actual de la variable (que por tanto puede variar según el valor que le asignemos en cada momento a lo largo de la ejecución del programa). Por ejemplo:

```
$cantidad=13;
$nombre=”Luis”;
```

La variable $cantidad es de tipo entero y la variable $nombre es una cadena de caracteres (por el contenido que tiene cada una). Es decir, si le asignamos a la variable $cantidad el valor 13, la estamos declarando implícitamente de tipo int. Si le asignamos después el valor “hola” entonces la estamos redeclarando como de tipo string. Todas las sentencias del lenguaje terminan en ; . El operador de asignación es = .

También podemos especificar de forma explícita el tipo del valor asignado a una variable. Para ello debemos poner entre paréntesis el tipo. A esto se le llama hacer un “casting” a la variable. Los tipos son:

- (int)
- (double)
- (string)
- (array)
- (object)

Por ejemplo:

```
$cantidad=5;
$precio_unidad=200;
$impuesto=1,16;
$total=(int) $cantidad * $precio_unidad * $impuesto;
```

El resultado de la operación de multiplicación es un resultado real, porque al menos uno de los operandos implicados lo es ($impuesto), sin embargo, la variable $total tendrá un valor entero porque hemos indicado una conversión del tipo de coma flotante (double) devuelto a tipo entero (int).

Es muy importante que tengamos en cuenta el comportamiento de las variables. El programa (la página PHP) no permanece en memoria tras su ejecución, sino que el intérprete PHP la procesa por completo y los resultados (el código HTML) generados se envían al navegador cliente. Esto significa que, una vez se ha interpretado la página, se pierde todo el contenido de las variables que ha utilizado. Por tanto, debemos especificar qué valores de variables queremos transferir en los enlaces entre páginas, aunque sea una llamada a la misma página en la que estamos. En el siguiente capítulo veremos las formas que existen para transferir variables entre llamadas a páginas.

Existen dos palabras reservadas del lenguaje, global y static, que sirven para definir el ámbito de una variable dentro de una función:

- global: Se utiliza dentro de una función para indicar que la variable a la que acompaña es una variable global (se usa en el programa principal) y por tanto, conservará el valor que tiene en el programa principal. Si se usa una variable dentro de una función con igual nombre que una variable global (del programa principal), se tomaría como variable local dentro de la función, es decir, la variable local ocultaría el valor de la variable global. Por ello, si queremos usar una variable del programa principal dentro de una función, debemos declararla como global.

- static: También se usa dentro de una función, e indica que la variable a la que acompaña es una variable estática, que mantiene su valor entre las sucesivas llamadas a la función (por lo que sólo se inicializa en la primera llamada a la función).

Veremos ejemplos de estos modificadores en los siguientes apartados sobre funciones (2.13 Funciones con variables globales y 2.14 Funciones con variables estáticas).

2.3. SENTENCIAS

Una sentencia puede estar formada por una asignación, una llamada a una función o una estructura de control de flujo. Las sentencias simples terminan en ; . Cuando estas sentencias se agrupan forman sentencias compuestas, y se delimitan con un carácter **{** al comienzo y con el carácter **}** al final.

Si olvidamos el ; se producirá un error durante la ejecución de la página PHP. Debemos tener en cuenta que el código PHP se interpreta en el servidor Web (se interpreta, no compila), y es posible que no aparezcan todos los errores existentes si el flujo de ejecución del programa no "pasa" por ciertas líneas. Por lo que otras ejecuciones, contemplando otros valores de entradas u otras condiciones lógicas, que sí "pasen" por otras líneas de código, originen errores no detectados en otras ejecuciones de la página.

2.4. OPERADORES

En este apartado veremos los principales operadores del lenguaje PHP.

2.4.1. Operadores aritméticos

+	Suma.
-	Resta.
*	Multiplicación.
/	División.
%	Módulo (resto de la división).
++	Operador de incremento.
--	Operador de decremento.

2.4.2. Operadores lógicos

and o &&	Devuelve verdadero si las dos expresiones son verdaderas.
or o ||	Devuelve verdadero si una de las dos expresiones es verdadera.
xor	Devuelve true si sólo una de las expresiones es verdadera.
!	Cambia una expresión de verdadera a falsa y viceversa.

2.4.3. Operadores de asignación

=	Operador básico de asignación.
*=	Operador de multiplicación.
/=	Operador de división.
+=	Operador de suma.
-=	Operador de resta.
%=	Operador de módulo.
.=	Operador de concatenación de cadenas.

2.4.4. Operadores de comparación

==	Igual que.
!=	Distinto que.
>	Mayor que.
<	Menor que.
>=	Mayor o igual que.
<=	Menor o igual que.
===	Idéntico (valor y tipo).

2.4.5. Operador de concatenación

Este operador sirve para unir dos cadenas de texto (tipo string) en una sola, y es el operador punto (.), o bien la forma .= como hemos visto en los operadores de asignación, para añadir una cadena al final de otra.

2.4.6. Operadores de bit

&	Operador and.
|	Operador or.
^	Operador xor.
~	Operador not.
<<nº	Operador desplazamiento izquierda.
>>nº	Operador desplazamiento derecha.

2.4.7. Operador de ejecución

Este operador nos permite ejecutar órdenes o comandos, y está formado por la doble comilla invertida (` `). Por ejemplo, esta sentencia:

$listado= `ls –l`;

ejecuta la orden del sistema operativo linux ls –l para obtener un listado de los archivos, y guarda el resultado en la variable $listado.

2.5. COMENTARIOS

Podemos usar comentarios de línea simple o de un grupo de líneas:

```
// Código comentado de línea simple
/*
        Código comentado que ocupa varias líneas.
        El código comentado no se ejecutará.
        $cadena="Hola mundo";
*/
```

2.6. CADENAS DE CARACTERES

Para definir una cadena de caracteres podemos utilizar las comillas dobles o simples. Pero hay una diferencia entre ellas: se expande el contenido de las variables si la cadena que las contiene va entre comillas dobles, no simples.
Por ejemplo:

```
$valor=13;
$cadena="Ejemplo de cadena con comillas dobles, \$valor es  $valor";
```

Al ejecutar estas dos líneas, la variable $cadena contendrá:
"Ejemplo de cadena con comillas dobles, $valor es 13"

Es decir, se ha sustituido la variable $valor por su contenido dentro de la cadena. Para escapar el carácter $, que como vimos, en PHP indica el comienzo de una variable, y para que no se interprete como tal, sino como una carácter más, debemos ponerle delante una barra invertida (\ ). Si hubiésemos utilizado comillas simples, entonces PHP no trata de expandir el contenido de las variables.

Para escapar las comillas dobles dentro de una cadena, usaremos también la barra invertida (\ ). Por ejemplo:

```
$enlace="<a href=\"http://www.google.com \"> Haga clic para ir al buscador Google </a>";
```

La primera y última comilla delimitan la cadena de caracteres asignada a la variable $enlace, y las comillas de dentro están escapadas para que estén convenientemente emparejadas. De este modo, las comillas que delimitan la cadena http://www.google.com son comillas que PHP tiene que pasar al código HTML que se genera.

2.7. ARRAYS

Un array es un conjunto de datos agrupados. En PHP los arrays pueden contener datos de diferentes tipos y además los índices para acceder a sus elementos pueden ser numéricos o de texto (arrays asociativos). Los índices numéricos comienzan por cero. Los arrays también pueden ser multidimensionales, es decir, arrays cuyo contenido es otro array.

Podemos declarar un array dándole valores directamente:

```
$libro['titulo']="Más grandes que el amor";
$libro['autor']="Dominique  Lapierre";
$libro['num_paginas']=447;
```

o bien mediante la función **array( )**:

```
$libro=array ("titulo" =>"Más grandes que el amor",
        "autor" =>"Dominique  Lapierre",
        "num_paginas" =>447);
```

Estas dos formas anteriores son equivalentes. Un ejemplo de array bidimensional podría ser:

```
$biblioteca[0][0] = "Sultana";
$biblioteca[0][1] = "Las hijas de Sultana";
$biblioteca[0][2] = "La canción del Mirlo";
$biblioteca[1][0] = "Programación en PHP";
$biblioteca[1][1] = "Páginas JSP";
$biblioteca[1][2] = "Introducción a ASP.NET";
```

También podemos combinar índices de texto y numéricos en un mismo array:

```
$biblioteca["narrativa"][0] = "Sultana";
$biblioteca["narrativa"][1] = "Las hijas de Sultana";
$biblioteca["narrativa"][2] = "La canción del Mirlo";
$biblioteca["programacion"][0] = "Programación en PHP";
$biblioteca["programacion"][1] = "Páginas JSP";
$biblioteca["programacion"][2] = "Introducción a ASP.NET";
```

Usando la función **array( )** en un array bidimensional:

```
$cine = array ( "sabado" => array ("matinal" => "El Rey León",
                                   "tarde" => "Los invasores",
                                   "noche" => "La Pasión" ),
          "domingo" => array ("matinal" => "Buscando a Nemo",
          "tarde" => "Falsas apariencias",
          "noche" => "La mala educación" ) );
```

2.8. ESTRUCTURAS DE CONTROL

Las estructuras de control que podemos utilizar son:

- if .. else .. elseif
- switch
- while
- do .. while
- for

2.8.1. Condicional if .. else .. elseif

Su sintaxis es la siguiente (la inclusión de la sentencia “else” y “elseif” es opcional, por eso va entre corchetes):

```
if (expresión)   {
        código para expresión “verdadera”
}  [else {
        código para expresión “falsa”
} ]
```

o bien:

```
if (expresión1)   {
        código para expresión1 “verdadera”
}  [  elseif (expresión2) {
        código para expresión2 “verdadera”, cuando expresión1 es falsa.

}  else  {
        código cuando el resto de expresiones son falsas.
} ]
```

Veamos un ejemplo de su uso:

```
<html>
  <body>
        <?
              $valor= 13;
              if ( $valor==13 )  {
                    echo “el valor es 13”;
              }else  {
                    echo “el valor es distinto de 13”;
              }
        ?>
  </body>
</html>
```

2.8.2. Condicional múltiple switch

```
switch ($variable)   {
        case valor1:
              código
              break;
        case valor2:
              código
              break;
        ...
}
```

Si omitimos **break** se seguirían ejecutando las sentencias que hubiera debajo, hasta encontrar algún otro break o hasta el final del switch si no hay ninguno.

Veamos un ejemplo de su uso:

```
<html>
  <body>
        <?
              $respuesta = 2;
              switch  ($respuesta)   {
                    case 1:
                          echo “La respuesta es 1”;
                          break;
                    case 2:
                          echo “La respuesta es 2”;
                          break;
                    case 3:
                          echo “La respuesta es 3”;
                          break;
              }
        ?>
  </body>
</html>
```

2.8.3. Bucle while

```
while (expresión)  {
        código
}
```

Ejemplo de su uso:

```
        <?
              $contador= 0;
              while ($contador <= 7)  {
                    echo “contador : “ . $contador . “<br>”;
                    $contador++;
              }
        ?>
```

2.8.4. Bucle do .. while

```
do  {
        código
} while (expresión)
```

Ejemplo de su uso:

```
<?
        $contador= 7;
        do  {
                echo “Quedan “ . $contador . ” segundos para el final <br>”;
                $contador--;
        } while ($contador >= 0);
?>
```

2.8.5. Bucle for

```
for  (inicialización; expresión; iteradores)  {
        código
}
```

Ejemplo de su uso:

```
<?
        for  ($i=0; $i<=10; $i++)  {
                echo “\$i contiene: “ . $i . “<br>”;
        }
?>
```

2.9. VARIABLES VARIABLES

En PHP existen las variables que hacen referencia al contenido de otra variable, es decir, una variable contiene el nombre de otra variable. Por ejemplo:

```
$temario="Temas de programación";
$apartado="temario";
echo "$temario <br>";
echo "$$apartado <br>";
```

Estas dos últimas líneas de código mostrarían exactamente lo mismo, la cadena "Temas de programación". La variable $apartado contiene el nombre de la variable $temario ("temario"). Cuando usamos $$apartado, nos estamos refiriendo a la variable cuyo nombre está contenido en la variable $apartado, es decir, $temario.

Para evitar algún tipo de ambigüedad posible al utilizar las variables variables, PHP tiene el operador **{ }** para poder determinar el orden de sustitución de los contenidos (como cuando estamos trabajando con arrays). La última línea del ejemplo anterior podría haberse escrito como:

```
echo  "${$apartado} <br>";
```

Las variables variables son muy útiles para leer ficheros de configuración de la forma:
Parámetro = valor
Con esta asignación:
${$parametro} = $valor;
tendríamos una variable PHP con el nombre del parámetro y con el contenido del correspondiente valor de configuración.

2.10. ASIGNACIÓN POR VALOR O POR REFERENCIA

Hasta la versión 4 de PHP todas las asignaciones de variables se hacían por valor, es decir, no existía la asignación por referencia. A partir de PHP 4, también podemos realizar asignaciones por referencia.
Por ejemplo:

```
$variable1=$variable2;
```

Es una asignación por valor: la variable $variable1 toma el contenido (valor) de la variable $variable2, y ambas variables conservan su valor por separado. Una modificación en $variable1 sólo le afectará a $variable1, y no a $variable2. Son variables distintas e independientes.

```
$variable2=30;
$variable1=$variable2;
echo “contenido de variable1: ” . $variable1 . “<br>”;
echo “contenido de variable2: ” . $variable2 . “<br>”;
```

Estas dos últimas líneas mostrarán lo mismo: el valor 30.

```
$variable2=30;
$variable1=$variable2;
$variable2=100;
echo “contenido de variable1: ” . $variable1 . “<br>”;
echo “contenido de variable2: ” . $variable2 . “<br>”;
```

Ahora, las dos últimas líneas mostrarán valores distintos, $variable1 tiene el valor 30 y $variable2 tiene valor 100. Las modificaciones de una de ellas no le afecta a la otra, porque la asignación de $variable2 a $variable1 era una asignación por valor.

Sin embargo, si hacemos:

```
$variable1=&$variable2;
```

estamos asignándole a $variable1 el mismo espacio de almacenamiento que tiene la variable $variable2, es decir, es como si fuesen la misma variable pero con distinto nombre. Estamos haciendo una asignación por referencia (para lo cual usamos el operador &). Al realizar un cambio sobre una de ellas, también se reflejará en la otra. Esto significa que si modificamos $variable2, la variable $variable1 tomará el nuevo valor asignado a $variable2.

```
$variable2=30;
$variable1=&$variable2;
$variable2=100;
echo “contenido de variable1: ” . $variable1 . “<br>”;
echo “contenido de variable2: ” . $variable2 . “<br>”;
```

En este caso, las dos últimas líneas de código mostrarán el mismo contenido: el valor 100. Al modificar $variable2, también estamos modificando $variable1, porque $variable1 está apuntando al valor que tiene $variable2.

2.11. FUNCIONES

PHP dispone de una gran cantidad de funciones integradas clasificadas en distintas categorías (de tratamiento de cadenas, de tratamiento de arrays, funciones matemáticas, criptográficas, de expresiones regulares, de comunicaciones, etc). Para tener una referencia completa y actualizada de ellas, podemos consultar la referencia del lenguaje en la dirección http://www.php.net, donde podemos encontrarla en varios idiomas.

Para declarar nuestras funciones usaremos la siguiente sintaxis:

```
function nombre_función (argumentos)  {
        global variables_globales
        variables
        sentencias
        return valor_devuelto
}
```

La palabra reservada **global** sirve para especificar que se trata de variables globales (como hemos visto en un apartado anterior), ya que todas las variables que declaremos en una función son locales a esa función, salvo que explícitamente indiquemos que se trata de una variable global.

El uso de **return** no es obligatorio, lo usaremos cuando queramos que nuestra función devuelva un valor. Al ejecutarse una sentencia con return, se termina la ejecución de la función y se devuelve el valor que lo acompaña.

Ejemplo:

```
<?
function operacion ($a, $b,$operador)  {
        $resultado=0;
        if ($operador=="1") {  // operación suma
                $resultado=$a+$b;
```

```
        }else {  //  operación resta
                $resultado=$a-$b;
        }
        return $resultado;
}
?>
<html>
        <body>
        <head><title>Ejemplo de página PHP que usa una función</title></head>
        <?
                $variable1=10;
                $variable2=25;
                $oper=1;
                $resultado=operacion($variable1,$variable2,$oper);
                echo “Resultado de la operación: “ . $resultado;
        ?>
        </body>
</html>
```

2.12. FUNCIONES PARA DEFINIR CONSTANTES

PHP posee dos funciones usadas en el tratamiento de constantes. La función **define** nos permite definir una constante. Las constantes no van precedidas por el símbolo $ habitual de las variables.
La sintaxis de esta función es:

booleano **define** (string nombre_constante, valor_constante);

Devuelve un valor verdadero si todo va bien y se puede definir la constante, y valor falso si la operación falla. Ejemplo de su uso:

```
define (“VELOCIDAD_MEDIA“, 60);
define (“COLOR_FONDO“, “azul”);
```

Por convención, los nombres de las constantes suelen escribirse con todas sus letras en mayúsculas.

La función **defined** nos permite comprobar si una constante está definida. Su sintaxis es:

booleano **defined** (string nombre_constante);

Devuelve verdadero si la constante pasada como argumento está definida, y falso si no lo está.
Por ejemplo:

```
define ("VELOCIDAD_MEDIA", 60);
if (defined ("VELOCIDAD_MEDIA"))  {
    echo "La constante VELOCIDAD_MEDIA está definida y su valor es" . VELOCIDAD_MEDIA . "<br>";
} else {
    echo "La constante VELOCIDAD_MEDIA no está definida";
}
```

2.13. FUNCIONES CON VARIABLES GLOBALES

Como hemos visto, las variables usadas dentro de una función son variables locales, salvo que explícitamente indiquemos que se trata de una variable global. Por ejemplo:

```
<?
function muestraSaludo ( )  {
        echo "La variable \$saludo contiene: " . $saludo;
}
?>
<html>
        <body>
        <head><title>Funciones con variables globales</title></head>
        <?
                $saludo="Hola";
                muestraSaludo( );
        ?>
        </body>
</html>
```

Esta página PHP no mostraría ningún saludo porque la única variable $saludo que utiliza y a la que se le da el valor “Hola”, es una variable global, del programa principal, y la variable $saludo que se utiliza dentro de la función muestraSaludo ( ) es una variable local. Como dentro de la función no se le da ningún valor a la variable local $saludo, la sentencia echo no puede escribir nada en su lugar. Para referirnos a la variable global del programa principal $saludo, debemos modificar el ejemplo anterior de esta forma:

```
<?
function muestraSaludo ( ) {
        global $saludo;    // así indicamos que la variable $saludo es global, y no local
        echo “La variable \$saludo contiene: “ . $saludo;
}
?>
<html>
        <body>
        <head><title>Funciones con variables globales</title></head>
        <?
                $saludo=”Hola”;
                muestraSaludo( );
        ?>
        </body>
</html>
```

Esta página PHP sí mostrará el contenido de la variable $saludo del programa principal, porque en la función hemos especificado que $saludo se trata de una variable global.

2.14. FUNCIONES CON VARIABLES ESTÁTICAS

Como vimos en el apartado de variables, una variable estática es aquella que mantiene su valor entre sucesivas llamadas a la función que la declara como static. Veamos un ejemplo de función que tiene una variable estática:

```
<?
function cuenta ( ) {
        static $contador=0;    // así indicamos que la variable $contador es estática
```

```
        $contador++;
        echo “La variable \$contador vale: “ . $contador;
}
?>
```

Como la variable $contador es estática, sólo se inicializa la primera vez que se ejecuta la función cuenta. En cada llamada a la función se incrementa su valor en uno sobre el valor anterior, y se muestra en pantalla. La variable $contador contiene el número de veces que se ha invocado a la función cuenta.

2.15. FUNCIONES RECURSIVAS

Una función recursiva es aquélla que se llama a sí misma. En algún momento debe satisfacerse la condición del caso base para que termine la recursión (la función deja de llamarse a sí misma). Un ejemplo típico de función recursiva es el factorial de un número:

```
<?
function factorial($numero )  {
        if ($numero==1)  {  // caso base, se acaba la recursividad
                return 1;
        }else {
                return $numero*factorial($numero-1);
        }
}
?>
<html>
        <body>
        <head><title>Funciones recursivas</title></head>
        <?
                $num=5;
                $resultado=factorial($num);
                echo “El factorial de ”.$num.“ vale: “.$resultado;
        ?>
        </body>
</html>
```

Es decir, el factorial de un número se define como el producto de dicho número por el factorial de dicho número menos uno. Para calcular ese otro factorial volvemos a llamar a la misma función factorial. En el ejemplo:

```
factorial($numero)= $numero * factorial($numero-1)
factorial(5)= 5 * factorial(4)
        5 * 4 * factorial(3)
        5 * 4 * 3 * factorial(2)
        5 * 4 * 3 * 2 * factorial(1)
        5 * 4 * 3 * 2 * 1              // se termina la recursión
        120
```

Cuando se llama a factorial(1), se llega al caso base, se devuelve el valor 1 y la función factorial deja de llamarse a sí misma. Cuanto más alto sea el número del que queremos obtener su factorial, mayor profundidad tendrá la pila de llamadas a la función recursiva, por lo que debemos tener cuidado para no desbordar la memoria (el factorial de 1000 con esta solución supondría llamar 1000 a la función factorial).

2.16. INCLUIR FICHEROS

PHP dispone de la función include ( ) para incluir ficheros dentro de nuestras páginas. Es conveniente separar cabeceras, pies, ficheros de configuración, funcionalidades relacionadas, ... en diversos ficheros, e incluirlos en nuestro programa principal cuando sea necesario. De esta forma es más fácil el mantenimiento y la reusabilidad de esos módulos (por ejemplo, si cambia algo de una cabecera, como el logotipo o un valor de configuración, sólo es necesario actualizarlo en el fichero que lo contiene y nada más, el resto de páginas que tienen incluido el fichero actualizado tomará la modificación).

La sintaxis de la función include ( ) es:

include (fichero_a_incluir);

El fichero a incluir puede contener únicamente una salida HTML, o un conjunto de funciones PHP, o salida HTML y código PHP. Por ejemplo, podríamos mantener la función factorial anterior en un fichero independiente junto con otras operaciones matemáticas, e incluirlo en aquellas páginas PHP que lo necesiten:

fichero operaciones.php

```
<?
function factorial($numero )  {
        if ($numero==1) {   // caso base, se acaba la recursividad
                return 1;
        }else {
                return $numero*factorial($numero-1);
        }
}
?>
```

fichero ejemplo_include.php

```
<?
        include(operaciones.php);
?>
<html>
        <body>
        <head><title>Funciones recursivas</title></head>
        <?
                $num=5;
                $resultado=factorial($num);
                echo "El factorial de ".$num." vale: ".$resultado;
        ?>
        </body>
</html>
```

La página ejemplo_include.php puede usar la función factorial como si estuviese definida en ella misma, porque incluye el fichero operaciones.php que la contiene. Además, el fichero operaciones.php puede incluirse en otras páginas que lo necesiten, y en caso de cambiar la función factorial recursiva, por una implementación iterativa más eficiente por ejemplo, todas las páginas PHP que la usen tomarán la nueva versión.

PHP dispone también de la función include_once ( ) para incluir ficheros en nuestras páginas. La diferencia con la función include ( ) anterior es que hará la inclusión si la página no tiene ya incluido el fichero. Esto es importante cuando tenemos un fichero que incluye a otros, y por ejemplo, alguno de esos otros también incluye a su vez otros ficheros comunes. Para evitar que pudiera darse el caso de incluir

dos veces el mismo fichero (con las posibles redefiniciones de constantes o variables), podemos usar include_once ( ). Su sintaxis es análoga a include ( ):

include_once (fichero_a_incluir);

2.17. CLASES

Una clase es un conjunto de datos y funciones que actúan sobre ellos. En PHP podemos declarar una clase con la siguiente sintaxis:

```
class nombre_clase  {
        // Declaración de variables de la clase
        var $variable1;
        var $variable2;
        // Declaración de funciones de la clase
        function nombre_funcion1 ( )  {
                // sentencias de la función
        }
}
```

Para crear un nuevo elemento de la clase (instanciar un objeto de la clase), usaremos el operador new:
$objeto=new nombre_clase;

Para acceder a los datos y funciones de la clase se utiliza el operador -> . Con $this nos referimos a elementos de la clase. Como ejemplo, vamos a definir la clase Pedido, en un fichero independiente, que puede ser incluido en aquellas páginas PHP que necesiten crear objetos de esta clase.

fichero clase_Pedido.php

```
<?
Class Pedido {
        var  $referencia;
        var  $cliente;
        var  $fecha_solicitud;
        var  $fecha_salida;
        var  $precio;
```

```
        // Función que inicializa el estado del pedido
        function hacerPedido ($referencia, $cliente, $fecha_solicitud, $precio)  {
                $this->referencia=$referencia;   // inicializa los datos de la clase
                $this->cliente=$cliente;
                $this->fecha_solicitud=$fecha_solicitud;
                $this->precio=$precio;
        }

        // Función que especifica la fecha de salida del pedido
        function indicarSalida ($fecha_salida)  {
                $this->fecha_salida=$fecha_salida;
        }

        // Función que aplica el IVA pasado al precio del pedido
        function incrementaIVA ($iva)  {
                $this->precio += $this->precio * $iva;
        }

        // Función que muestra los detalles del pedido
        function muestra ( )  {
                echo "----------------------------------------------------------------------";
                echo "información del pedido con referencia: " .$this->referencia . "<br>";
                echo "Cliente: " . $this->cliente . "<br>";
                echo "Fecha de solicitud: " . $this->fecha_solicitud . "<br>";
                echo "Precio: " . $this->precio . "<br>";
                echo "Fecha de salida: " . $this->fecha_salida . "<br>";
                echo "----------------------------------------------------------------------";
        }
}  // fin de definición de la clase
?>
```

Ahora vamos a crear una página PHP que incluya la clase Pedido, cree un objeto de esta clase y acceda a sus métodos:

fichero pag_gestionarPedido.php

```
<?
        include(clase_Pedido.php);  // Fichero que contiene la definición de la clase Pedido
?>
<html>
        <body>
        <head><title>Página que gestiona un pedido</title></head>
<?
        $pedido1= new Pedido;
        $pedido1->hacerPedido ("A34B45", "Luis Cabrera", "15/05/04", 200);
        $iva=0,16;
        $pedido1->incrementaIVA($iva);
        $pedido1->indicarSalida ("17/05/04");
        $pedido1->muestra( );
?>
        </body>
</html>
```

RECUERDE

- Debemos separar el código PHP y el código HTML en los programas. La forma más habitual es <? código PHP ?> .

- Todas las variables en PHP deben comenzar por el carácter $.

- Las constantes en PHP no comienzan por el carácter $. Se definen con la función define ( ).

- El tipo de cada variable depende del contenido actual que tenga. Podemos hacer un "casting" a un tipo distinto al valor asignado.

- En PHP una variable puede contener el nombre de otra variable (son las variables variables).

- El valor de las variables se interpreta dentro de las comillas dobles, no dentro de comillas simples.

- Es necesario escapar ciertos caracteres con una barra invertida (\ ) para que no sean interpretados por PHP.

- Las asignaciones a variables pueden ser por valor o por referencia. Para que sea una asignación por referencia usamos el operador & .

- Las variables dentro de una función son variables locales, salvo que se indiquen como globales o estáticas, con los modificadores global y static.

- Los arrays pueden tener índices numéricos o de texto. Pueden ser multidimensionales.

- Las clases se definen con la palabra reservada class. Podemos crear un nuevo objeto de la clase con el operador new, y acceder a los miembros de la clase con el operador -> .

- Podemos separar nuestro código en distintos archivos e incluirlos en nuestras páginas PHP con la función include ( ) o include_once ( ).

Tema 3

Comunicación de datos entre páginas. Procesado de formularios

3.1. FORMULARIOS

Podemos interaccionar con una página PHP pasándole un valor al llamarla, por ejemplo, a través de un enlace o al escribir su URL en la barra de direcciones de un navegador, o bien, a través de un formulario: cargamos una página que contenga un formulario, con los campos necesarios, se rellenan en el cliente por el usuario, y se envía al servidor donde se procesa. Por tanto, el programa destinatario de un formulario HTML puede ser una página PHP (el valor del elemento 'action' de un formulario).

NOTA DE CONFIGURACIÓN: Es muy importante tener en cuenta el valor de la directiva de configuración 'register_globals' (se encuentra en el fichero php.ini), porque influye en el conjunto de variables predefinidas disponibles en el sistema y en la forma de acceder a ellas. Habitualmente, esta directiva estaba activada por defecto, con lo que todas las variables globales estaban disponibles en cualquier página PHP como variables individuales y podíamos acceder a ellas directamente. A partir de PHP 4.2.0, el valor por defecto de esta directiva ha cambiado, tiene el valor *off* (desactivada), y por tanto no podremos acceder a las variables globales directamente como variables individuales sino a través de las matrices asociativas correspondientes, las habituales $HTTP_SERVER_VARS, $HTTP_POST_VARS, $HTTP_GET_VARS, $HTTP_FILES_VARS, $HTTP_COOKIE_VARS, $HTTP_SESSION_VARS y $GLOBALS. A partir de la versión 4.1.0 de PHP, también existen las nuevas matrices autoglobales $_SERVER, $_POST, $_GET, $_FILES, $_COOKIE, $_REQUEST y $_SESSION. El valor de 'register_globals' podemos modificarlo según nuestras preferencias. Es recomendable usar las nuevas matrices globales asociativas del tipo $_. A lo largo de este capítulo supondremos que 'register_globals' está activada, y por tanto, en los ejemplos de código mostrados estarán disponibles directamente las variables globales que representan las entradas de los usuarios a través de formularios o de variables pasadas en los enlaces. Si probamos estos ejemplos en un sistema cuya configuración tenga la directiva 'register_globals' desactivada, no funcionarán porque estaremos intentando acceder a variables que no existen. En los capítulos siguientes 4 y 5, sobre sesiones y variables predefinidas, veremos las otras formas posibles de acceso a las variables globales, a través de los arrays mencionados, para que conozcamos las distintas posibilidades de acceso a las variables predefinidas en nuestras páginas PHP.

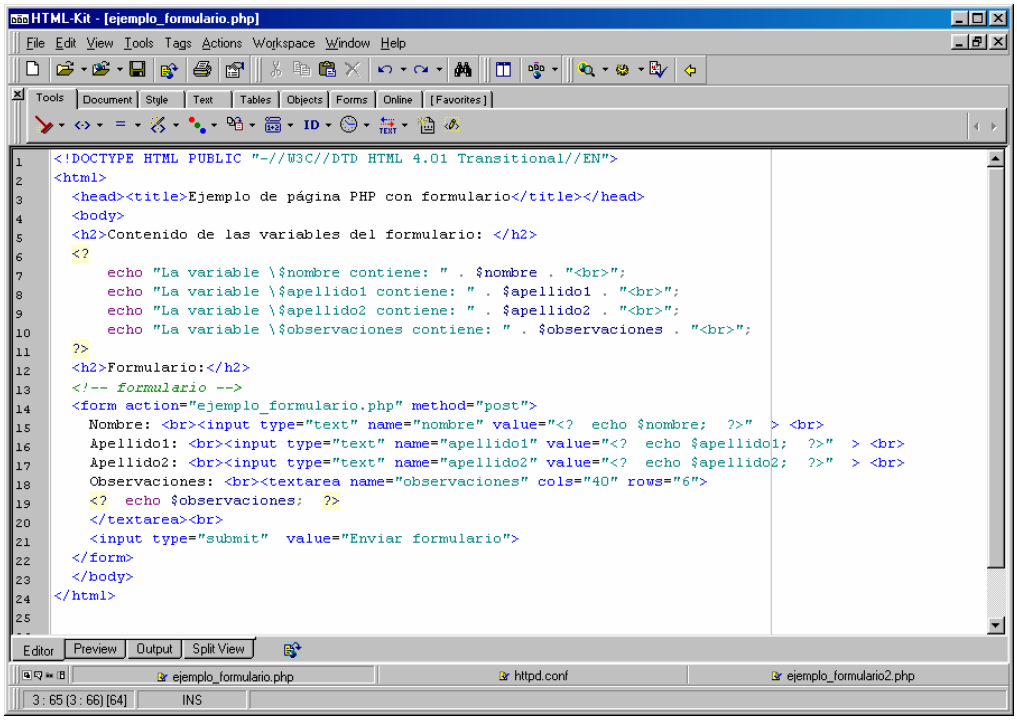
Por tanto, con la directiva 'register_globals' activada, cuando definimos un elemento en un formulario (un campo de texto, un combo, un conjunto de casillas checkbox ...), automáticamente se crea la variable correspondiente para cada uno de ellos en la página PHP destino del formulario. Cada variable correspondiente contendrá el valor que hayamos introducido en el navegador.

El nombre de la variable disponible en la página PHP destino del formulario es el mismo que tiene el campo en dicho formulario (precedido por el carácter $ habitual que identifica las variables en PHP).

Veamos un ejemplo:

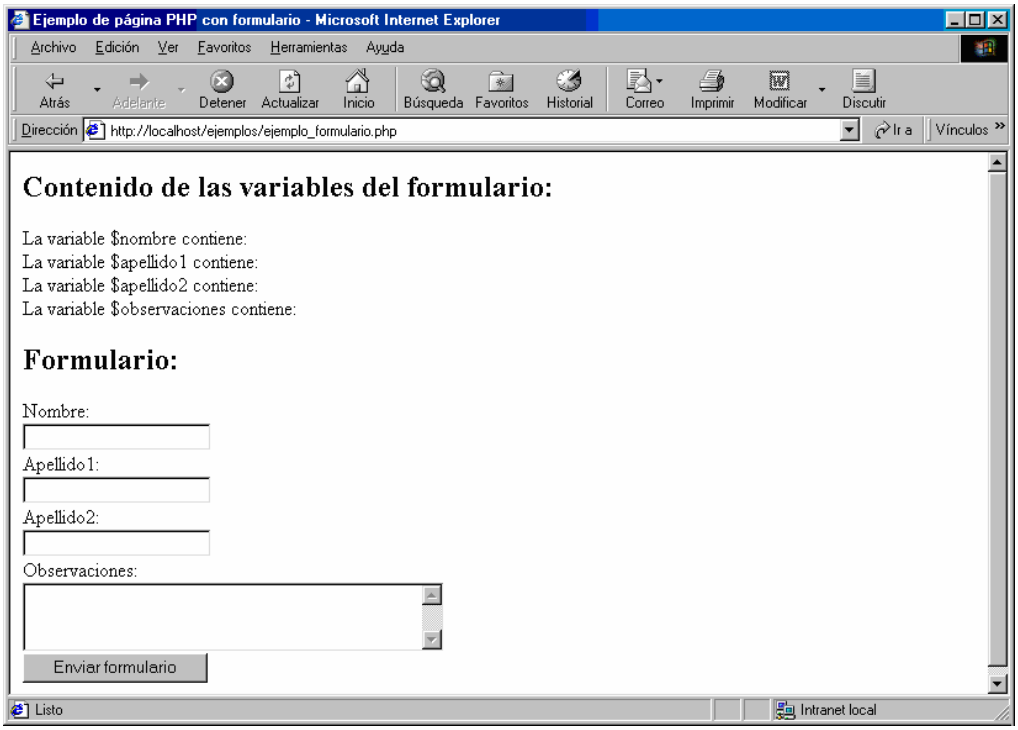
ejemplo_formulario.php

```
<html>
        <head><title>Ejemplo de página PHP con formulario</title></head>
        <body>
        <h2>Contenido de las variables del formulario: </h2>
        <?
                echo “La variable \$nombre contiene: “ . $nombre . “<br>”;
                echo “La variable \$apellido1 contiene: “ . $apellido1 . “<br>”;
                echo “La variable \$apellido2 contiene: “ . $apellido2 . “<br>”;
                echo “La variable \$observaciones contiene: “ . $observaciones . “<br>”;
        ?>
        <h2>Formulario:</h2>
        <!-- formulario -->
        <form action=”ejemplo_formulario.php” method=”post”>
            Nombre: <br><input type=”text” name=”nombre” value=”<?  echo $nombre;  ?>”  > <br>
            Apellido1: <br><input type=”text” name=”apellido1” value=”<?  echo $apellido1;  ?>” > <br>
            Apellido2: <br><input type=”text” name=”apellido2” value=”<?  echo $apellido2;  ?>”  > <br>
                Observaciones: <br><textarea name=”observaciones” cols=”30” rows=”7”>
                                        <?  echo $observaciones;  ?>
                                        </textarea><br>
                <input type=”submit”  value=”Enviar formulario”  >
        </form>
        </body>
</html>
```

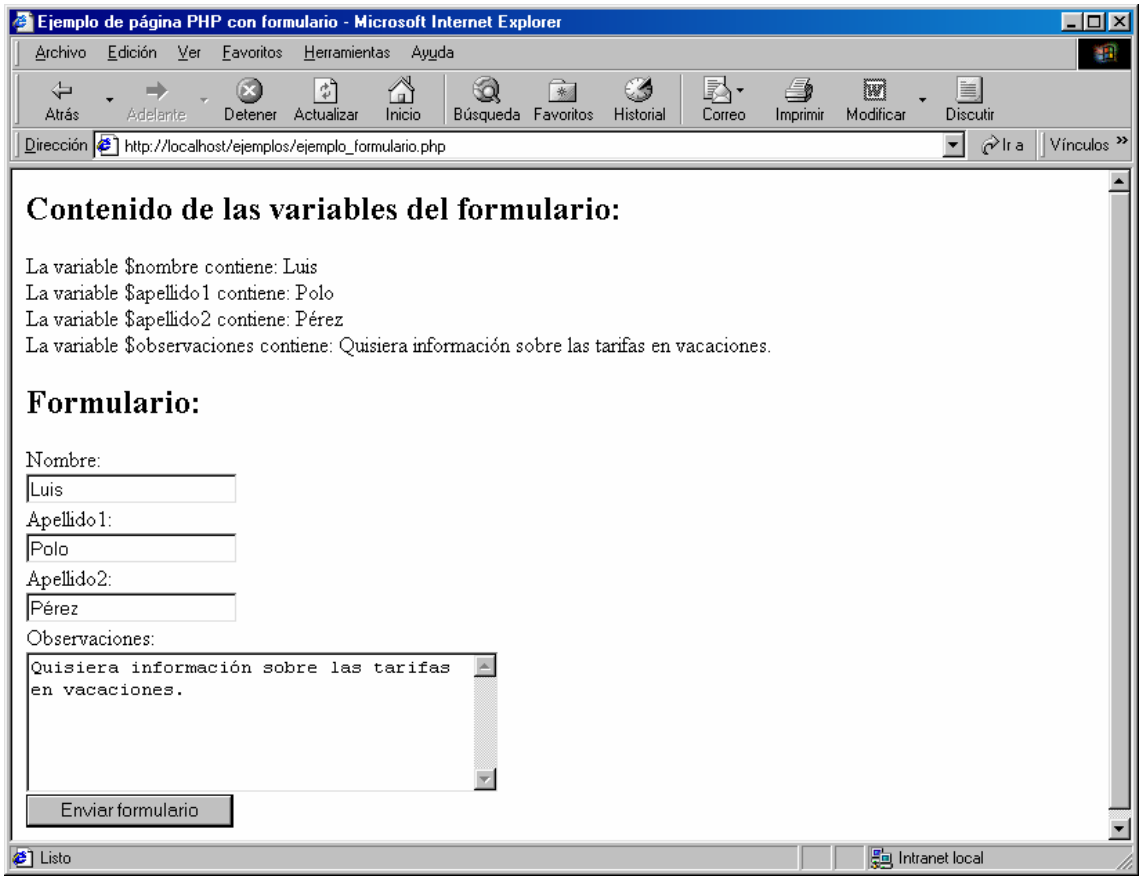


Editando el código de la página PHP en el editor HTML-Kit

Esta página PHP muestra el contenido de las variables del formulario que presenta a continuación. La primera vez que cargamos la página en el navegador (aún no hemos rellenado el formulario), las variables del formulario todavía no han sido creadas ni tienen ningún valor asignado, por ello las primeras líneas no mostrarán nada en el lugar dedicado a las variables correspondientes a cada campo del formulario. El resultado de la primera ejecución de ejemplo_formulario.php es:



Cuando rellenamos los campos del formulario, y pulsamos el botón de submit ('Enviar formulario'), el navegador envía el formulario, con todos sus campos y el valor que hayamos introducido, al programa indicado en el atributo 'action' de la etiqueta <form> del HTML, que, como podemos ver en el código, es la propia página PHP ejemplo_formulario.php. Es decir, la propia página que muestra el formulario puede ser también la encargada de procesarlo (también podría ser otra página PHP distinta). Cuando la página ejemplo_formulario.php recibe el formulario (segunda llamada a dicha página), como ya sí le llegan las variables correspondientes a los campos del formulario, puede mostrar su valor tanto en las líneas informativas del principio como en los campos, ya que el atributo 'value' de las etiquetas <input> y el <textarea> del formulario HTML toman como valor el propio al correspondiente a la variable PHP que se crea cuando se envía el formulario al servidor:



3.2. DISTINCIÓN DEL ESTADO DEL FORMULARIO

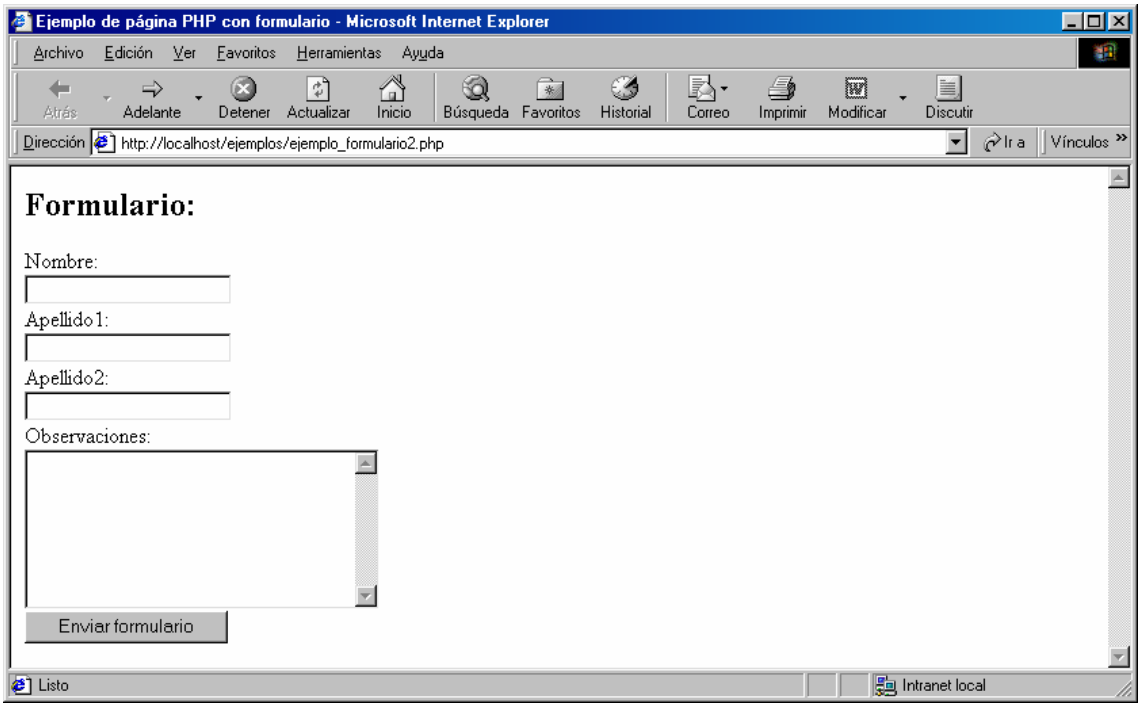
Aunque sea la misma página PHP la que presenta el formulario y la que lo procesa cuando éste se envía, podemos modificar el código anterior para que las sucesivas llamadas a la página aparezcan como páginas distintas. Si es la primera llamada a la página, presentamos el formulario para que se rellene, y si el formulario contiene datos, entonces los tratamos (se muestran por pantalla, se utilizan en una consulta a una base de datos o se guardan, etc). El ejemplo anterior modificado quedaría:

ejemplo_formulario2.php

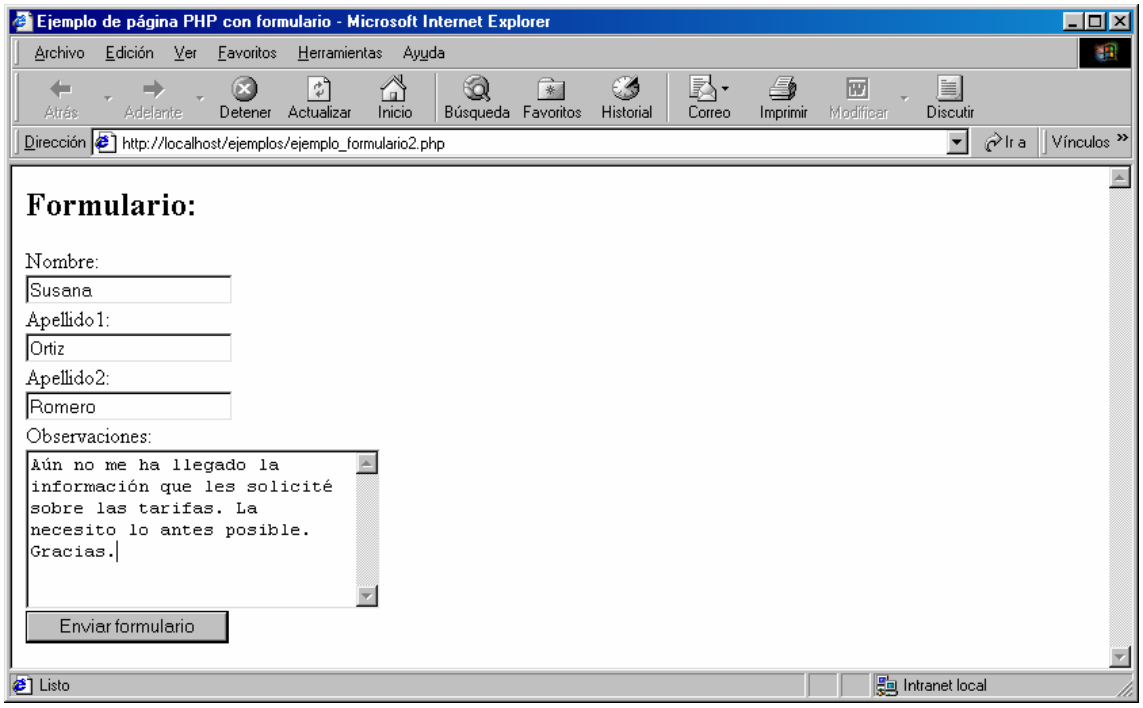
```
<html>
	<head><title>Ejemplo de página PHP con formulario</title></head>
	<body>
	<?
		if  ( !empty($nombre) )  {  // la variable $nombre tiene contenido
			echo “<h2>Contenido de las variables del formulario: </h2>”;
			echo “La variable \$nombre contiene: “ . $nombre . “<br>”;
			echo “La variable \$apellido1 contiene: “ . $apellido1 . “<br>”;
			echo “La variable \$apellido2 contiene: “ . $apellido2 . “<br>”;
			echo “La variable \$observaciones contiene: “ . $observaciones . “<br>”;
		}else  {  // presentamos formulario para que se rellene
	?>
	<h2>Formulario:</h2>
	<!-- formulario -->
	<form action=”ejemplo_formulario2.php” method=”post”>
	    Nombre: <br><input type=”text” name=”nombre” value=”<?  echo $nombre;  ?>”  > <br>
	    Apellido1: <br><input type=”text” name=”apellido1” value=”<?  echo $apellido1;  ?>” > <br>
	    Apellido2: <br><input type=”text” name=”apellido2” value=”<?  echo $apellido2;  ?>”  > <br>
		Observaciones: <br><textarea name=”observaciones” cols=”30” rows=”7”>
				<?  echo $observaciones;  ?>
				</textarea><br>
		<input type=”submit”  value=”Enviar formulario”  >
	</form>
	<?	}
	?>
	</body>
</html>
```

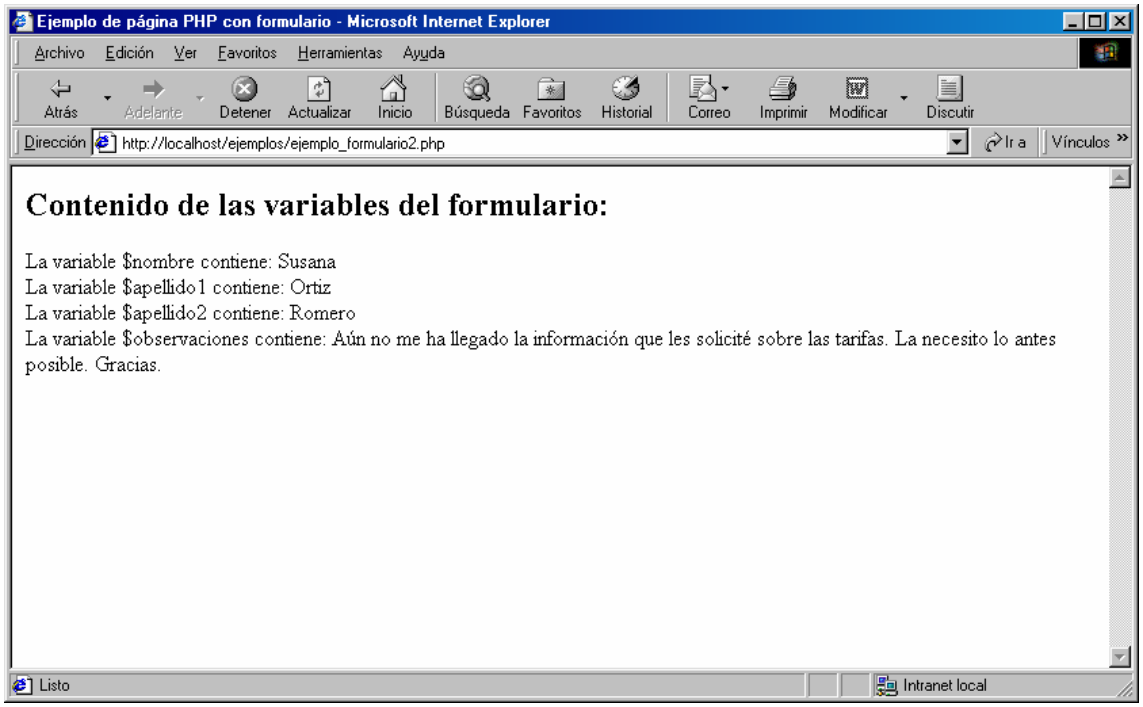
La función **empty( )** devuelve verdadero cuando la variable pasada está vacía. Si la variable $nombre aún no tiene contenido, presentamos el formulario para poder leerla y asignarle un valor. Si ya lo tiene, significa que el formulario ya se rellenó y envío, así que mostramos el valor de los campos del formulario y no presentamos el formulario (es similar a escribir el código de presentación del formulario en una página, y el código que lo procesa en otra página PHP distinta).

La primera vez que ejecutamos la página ejemplo_formulario2.php obtendremos esta salida:



Rellenamos los campos del formulario y pulsamos el botón de submit 'Enviar formulario'. La página destino del formulario es la propia ejemplo_formulario2.php, pero en esta ocasión sólo se mostrará el contenido que se introdujo en el formulario, y no se presentará nuevamente el formulario porque la página PHP distingue el estado de las variables para saber si se han introducido o no los datos.


Ejemplo de página PHP con formulario - Microsoft Internet Explorer
Archivo Edición Ver Favoritos Herramientas Ayuda
Atrás Adelante Detener Actualizar Inicio Búsqueda Favoritos Historial Correo Imprimir Modificar Discutir
Dirección http://localhost/ejemplos/ejemplo_formulario2.php
Ir a Vínculos
Formulario:
Nombre:
Susana
Apellido1:
Ortiz
Apellido2:
Romero
Observaciones:
Aún no me ha llegado la información que les solicité sobre las tarifas. La necesito lo antes posible. Gracias.
Enviar formulario
Listo
Intranet local


Ejemplo de página PHP con formulario - Microsoft Internet Explorer
Archivo Edición Ver Favoritos Herramientas Ayuda
Atrás Adelante Detener Actualizar Inicio Búsqueda Favoritos Historial Correo Imprimir Modificar Discutir
Dirección http://localhost/ejemplos/ejemplo_formulario2.php
Ir a Vínculos
Contenido de las variables del formulario:
La variable $nombre contiene: Susana
La variable $apellido1 contiene: Ortiz
La variable $apellido2 contiene: Romero
La variable $observaciones contiene: Aún no me ha llegado la información que les solicité sobre las tarifas. La necesito lo antes posible. Gracias.
Listo
Intranet local

3.3. CAMPOS OCULTOS EN FORMULARIOS

Los formularios también pueden utilizarse para transferir datos que no son visibles, los campos de tipo oculto:

<input type="hidden" name="nombre_variable" value="valor_variable">

Al enviar un campo de este tipo dentro de un formulario ocurre igual que con el resto de campos, en la página PHP de destino se crea automáticamente una variable con el nombre del campo. La única diferencia con el resto de campos es que no tiene una representación visible dentro del formulario y por tanto, no le damos valor directamente rellenando una entrada, sino por ejemplo a partir de otras entradas del formulario o como información complementaria que necesita la página de destino.

En el siguiente ejemplo, una página PHP presenta un campo para obtener un número y además contiene un campo oculto cuyo valor es también otro número. La otra página PHP destinataria del formulario, compara los dos valores que le llegan (el valor introducido y el valor del campo oculto) y muestra un mensaje para indicar si son iguales o no:

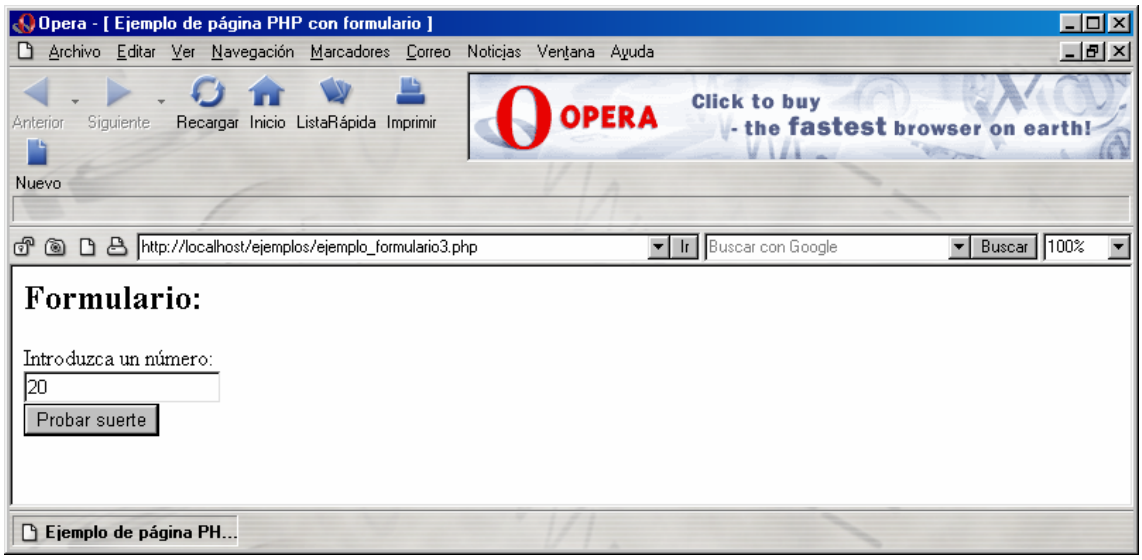
ejemplo_formulario3.php

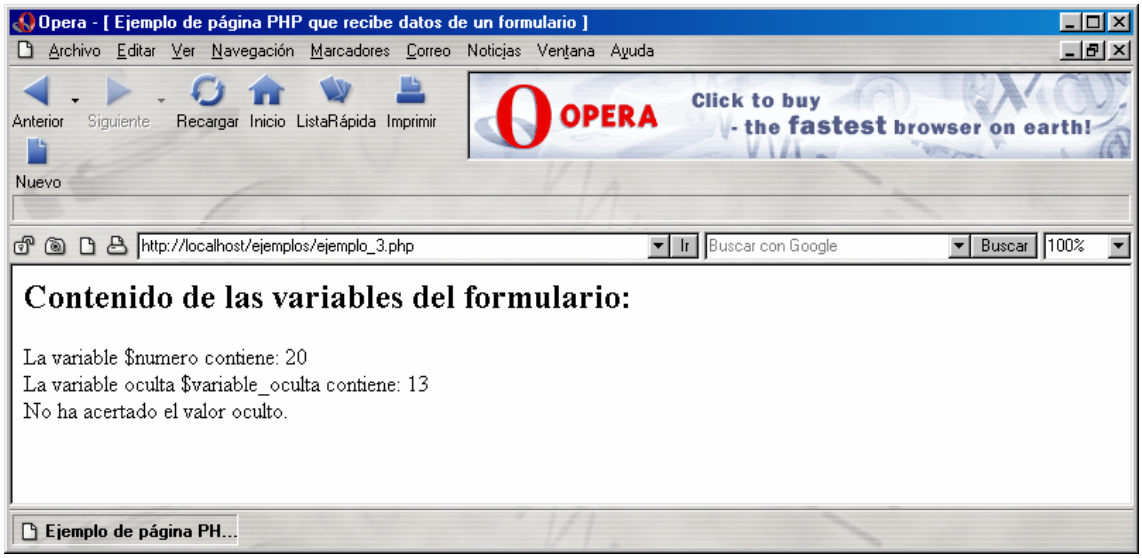
```
<html>
        <head><title>Ejemplo de página PHP con formulario</title></head>
        <body>
        <?
                $numero_oculto="13";
         ?>
        <h2>Formulario:</h2>
        <form action="ejemplo_3.php" method="post">
           Introduzca un número: <br><input type="text" name="numero">
           <input type="hidden" name="variable_oculta" value="<?  echo $numero_oculto;  ?>"  > <br>
           <input type="submit"  value="Probar suerte"  >
        </form>
        </body>
</html>
```

ejemplo_3.php

```
<html>
        <head><title>Ejemplo de página PHP que recibe datos de un formulario</title></head>
        <body>
          <h2>Contenido de las variables del formulario:</h2>
          <?
                echo “La variable \$numero contiene: “ . $numero . “<br>”;
                echo “La variable oculta \$variable_oculta contiene: “ . $variable_oculta . “<br>”;
                 if  ( $numero==$variable_oculta )  {
                        echo “Ha acertado el valor oculto. <br>”;
                 }else  {
                        echo “No ha acertado el valor oculto. <br>”;
                }
         ?>
        </body>
</html>
```

Al ejecutar estas páginas:





Si la página ejemplo_3.php se llamara directamente, las variables $número y $variable_oculta no estarían definidas, por lo que en su lugar no se mostraría ningún valor en pantalla. Para que estas dos variables estén definidas, la llamada de esta página debe hacerse al pulsar el botón de submit del formulario de la página ejemplo_formulario3.php, donde se generan estas variables y se comunican a ejemplo_3.php.

3.4. FORMULARIOS DE TAMAÑO VARIABLE

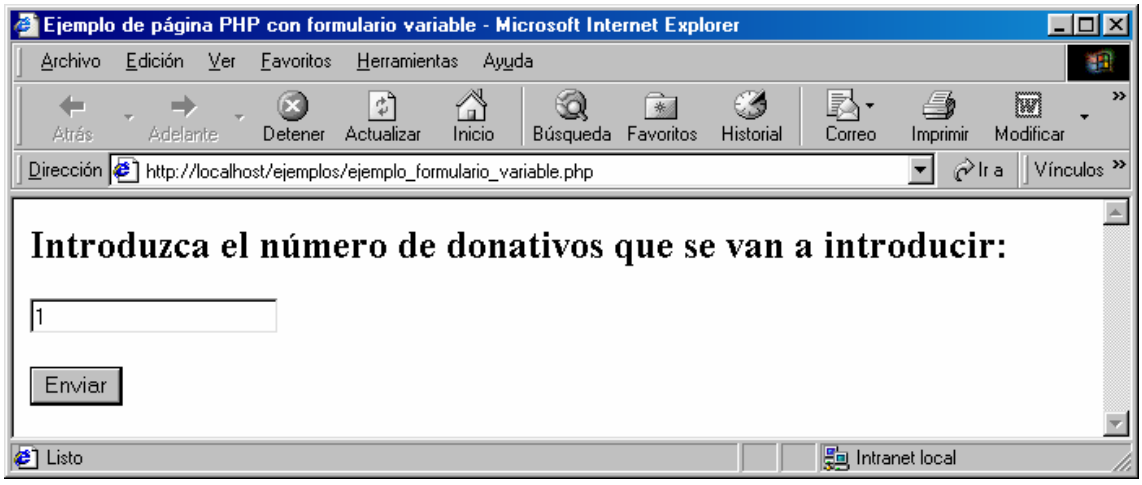
Como hemos visto en los apartados anteriores, podemos escribir todo el código que presenta un formulario y el que lo procesa en la misma página PHP, o bien, podemos separarlos en dos páginas distintas, una que presenta el formulario y otra destinataria que recibe los valores introducidos en el navegador. Podemos extender la primera técnica y programar una página PHP que distingue entre varios estados posibles y genera el código HTML necesario para cada uno. Por ejemplo, podríamos tener una página que detecta que es la primera vez que se llama, entonces presenta un formulario para pedir el número de datos que quieren leerse. Una vez sabido esto, la página presenta otro formulario con tantos campos a rellenar como se había pedido, y por último, una vez que se rellenan y envían nuevamente al servidor, la misma página PHP inicial detecta que ya tiene los datos del último formulario y los procesa. El código podría ser:

ejemplo_formulario_variable.php

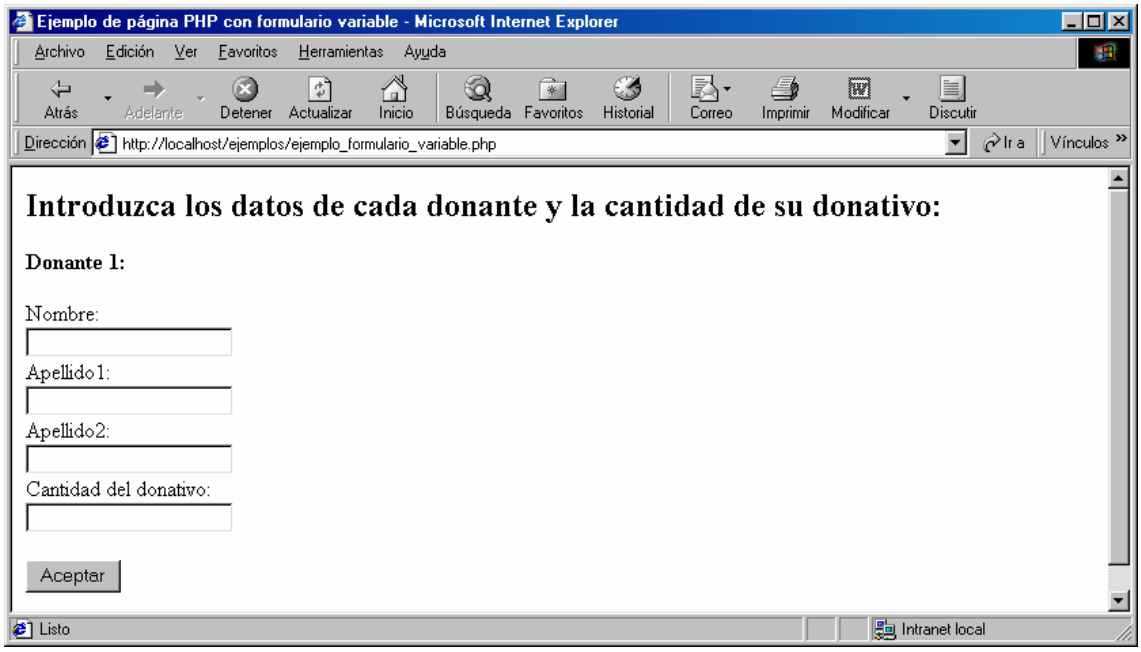
```
<html>
        <head><title>Ejemplo de página PHP con formulario variable</title></head>
        <body>
        <?
          // primero leemos el número de donativos que se necesitan
          if  ( empty($numero_donativos) )  {  // la variable $numero_donativos está vacía
                echo “<h2>Introduzca el número de donativos que se van a introducir: </h2>”;
         ?>
            <!-- Primer formulario -->
            <form action=”ejemplo_formulario_variable.php” method=”post”>
              <input type=”text” name=”numero_donativos” value=”<? echo $numero_donativos; ?>”>
              <p><input type=”submit”  value=”Enviar”  >
            </form>
        <?
          }
          // a continuación presentamos otro formulario para recoger los donativos
          if  ( empty($boton_aceptar)  &&   !empty($numero_donativos) )  {
                // tenemos el número de donativos pero aún no se ha rellenado el 2º formulario
         ?>
            <!-- Segundo formulario -->
            <form action=”ejemplo_formulario_variable.php” method=”post”>
         <?
                echo “<h2>Introduzca los datos de cada donante y la cantidad de su donativo: </h2>”;
                for ($i=0; $i<$numero_donativos; $i++)  {
                        echo "<h4>Donante ".($i+1).": </h4>";
                        echo “Nombre: <br>“;
                        echo “<input type=text name=nombre[$i]><br>”;
                        echo “Apellido1: <br>“;
                        echo “<input type=text name=apellido1[$i]><br>”;
                        echo “Apellido2: <br>“;
                        echo “<input type=text  name=apellido2[$i]><br>”;
                        echo “Cantidad del donativo: <br>“;
                        echo “<input type=text  name=donativo[$i]><p>”;
                }
```

```
            echo “<input type=hidden name=numero_donativos value=”.$numero_donativos.“>”;
            echo “<p><input type=submit  name=boton_aceptar value=Aceptar><p>”;
            echo “</form>”;
        }
        // por último se procesan los valores leídos:  se muestran los donantes y la suma total
        if  ( $boton_aceptar==”Aceptar” )  {  // se ha enviado el 2º formulario
            echo "<h2>Datos de los donantes: </h2>";
            $total_donativos=0;
            for ($i=0; $i<$numero_donativos; $i++)  {
              $total_donativos+=$donativo[$i];
              echo “Donante: “.$nombre[$i]. “ “ .$apellido1[$i]. “ “ .$apellido2[$i]. “<br>“;
              echo “Cantidad: ” . $donativo[$i]  . “<br>”;
              echo “----------<p>”;
            }
            echo “Total de donativos: ” . $total_donativos . " Euros";
        }
        ?>
    </body>
</html>
```

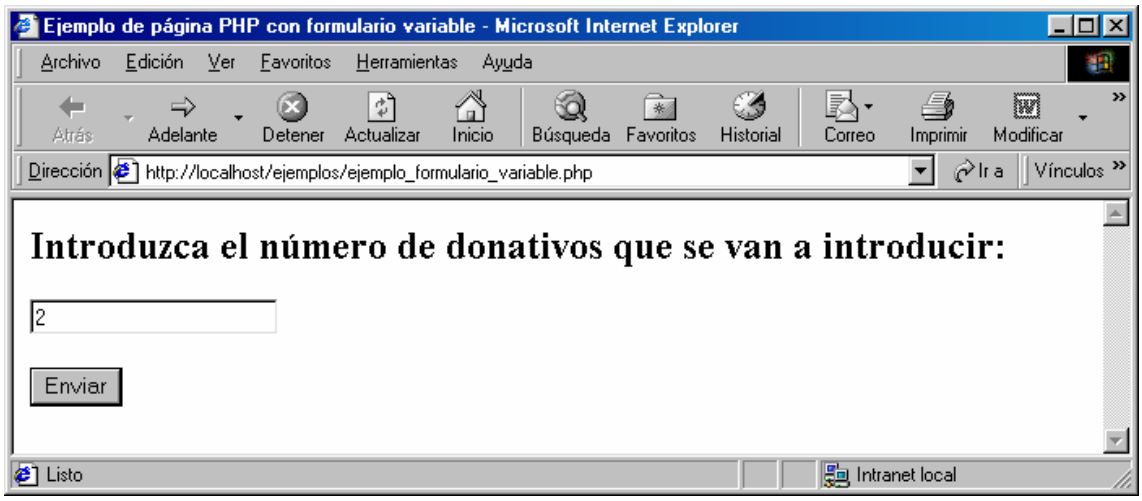
La primera vez que ejecutamos esta página PHP la variable $numero_donativos está vacía, así que se entra en el primer if y la página muestra el formulario que la pide. El resto de las condiciones siguientes no se cumplen (el que presenta el segundo formulario y el que muestra los datos en el último paso). Obtendremos esta apariencia en el navegador:

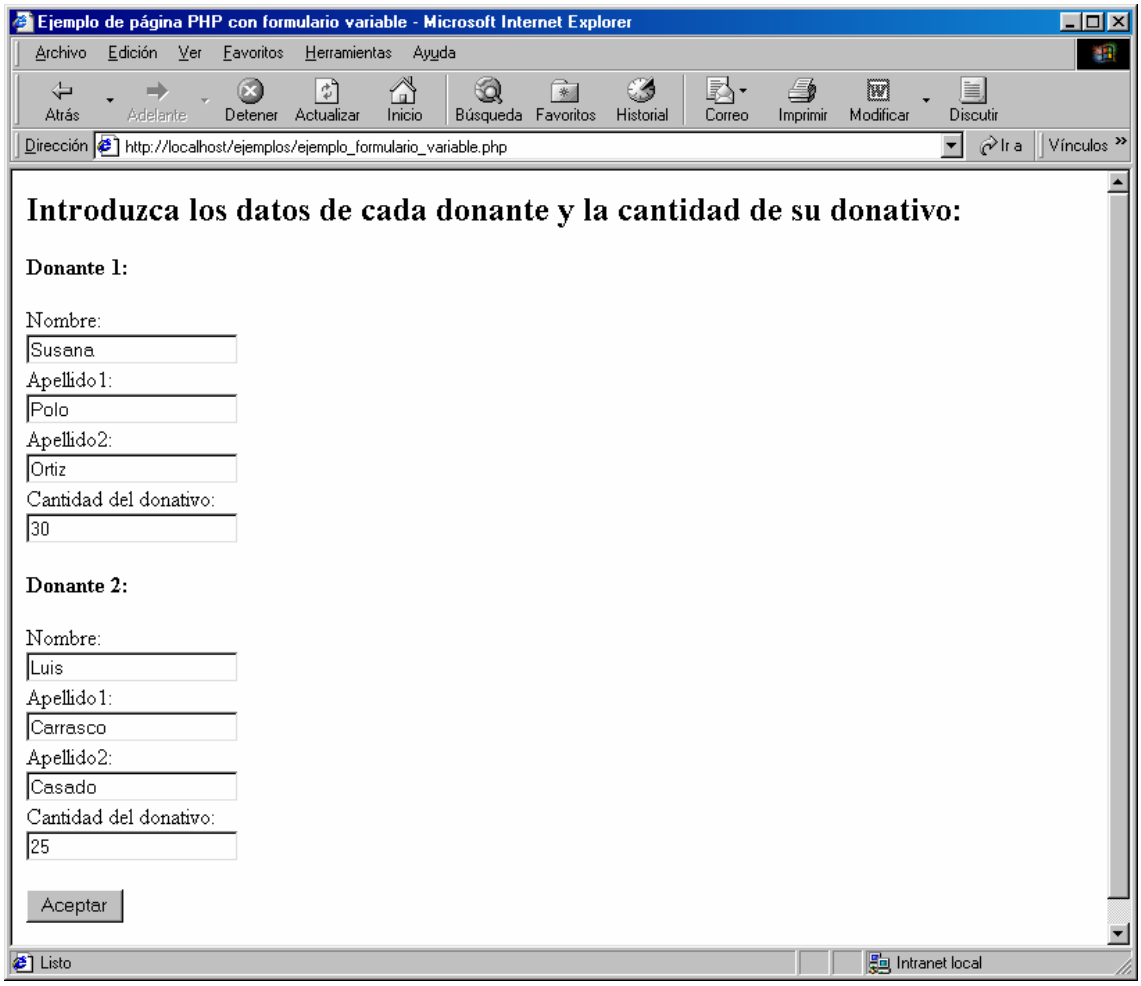


El número que introduzcamos en este campo determinará las entradas para recoger los datos de cada donante en el siguiente formulario que presenta la página ejemplo_formulario_variable.php (se cumple el segundo if). Es decir, para distintos valores de la variable $numero_donantes distinta apariencia del segundo formulario de datos mostrado (se repite un conjunto de campos sobre nombre, apellidos y cantidad para cada donativo):

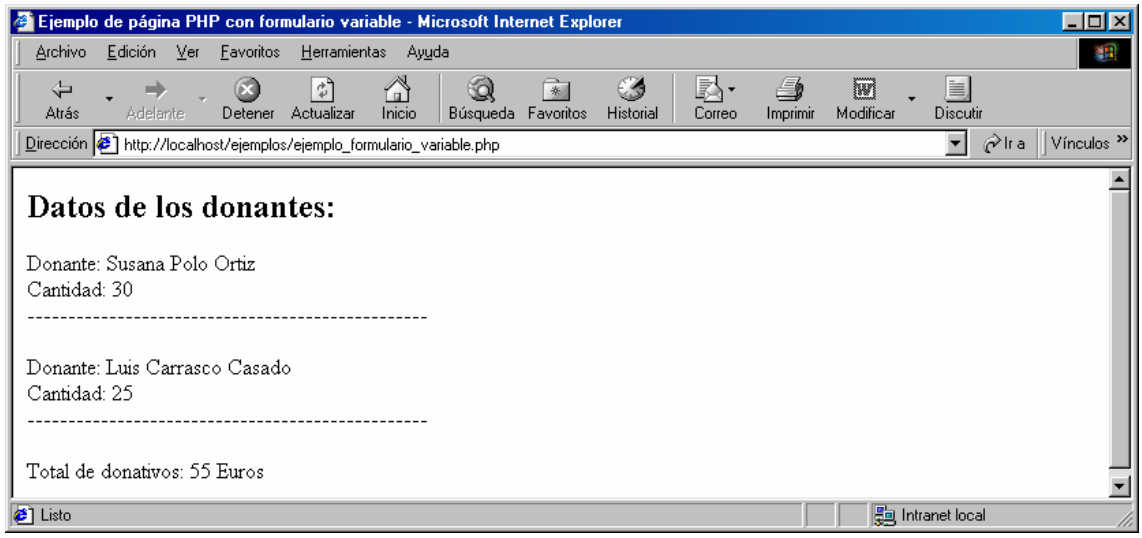


Sin embargo, si en el primer formulario introducimos un valor de 2 y enviamos el formulario, se nos presentará otro formulario con los campos necesarios para introducir los datos de dos donantes, porque la longitud del segundo formulario depende del valor introducido en el primero:





La apariencia del último posible estado de esta página PHP también depende de los valores anteriores introducidos. Cuando se rellena el segundo formulario y se envía, la página muestra los datos recogidos y la suma total del donativo:



Observemos que los datos nombre, apellido1, apellido2 y donativo de cada donante se guardan en su correspondiente array $nombre[ ], $apellido1[ ], $apellido2[ ], $donativo[ ], cuyas longitudes serán iguales al valor de $numero_donantes introducido en el primer formulario. Podemos ver que el nombre de la variable correspondiente a un campo de formulario (atributo 'name' de los campos 'input' del HTML) puede ser una celda de un array, se trata igual que si fuese una variable individual.

Es importante también tener en cuenta que entre la segunda y tercera llamada a la página PHP se perdería el valor de la variable $numero_donantes, si ésta no es pasada explícitamente en la siguiente llamada (se pasa como un campo de tipo oculto, 'hidden') para que sea conocida al entrar en el último estado de la página, cuando muestra los datos de los donantes (esta variable coincide con la longitud de los arrays que contienen los datos).

3.5. TRANSFERIR UN FICHERO AL SERVIDOR

Otro uso de los formularios consiste en transferir ficheros entre el cliente y el servidor, por ejemplo, para actualizar las imágenes que se muestran en nuestra aplicación.

Para ello, necesitamos definir un formulario con los siguientes atributos en la etiqueta <form>:

enctype="multipart/form-data" method="post"

Dentro del formulario debemos incluir una etiqueta <input type="file">. Esta etiqueta es la que se muestra en los navegadores como un campo de entrada para escribir una ruta junto a un botón "Examinar" que abre una ventana donde seleccionar un fichero.
Al ejecutar un formulario de este tipo, introducir un nombre de fichero y pulsar el botón de envío del formulario, PHP copia el fichero indicado en el cliente a un directorio temporal del servidor con un nombre único que él decide. Este fichero subido al servidor es temporal, se borra al terminar la sesión. Por eso debemos copiarlo explícitamente a otro directorio del servidor si queremos conservarlo.

PHP define automáticamente unas cuantas variables que nos informan del fichero transferido para que sepamos localizarlo y copiarlo a otro directorio del servidor. Los nombres de estas variables son (donde "fichero" es el nombre que se le ha dado al campo tipo file, es decir, <input type="file" name="fichero">:

- $fichero, es el nombre del fichero temporal en el servidor, con la ruta de acceso completa. Nos permite localizar dónde se ha subido.

- $fichero_name, es el nombre que tenía el fichero en el equipo cliente.

- $fichero_type, es el tipo de fichero (imagen, comprimido,...)

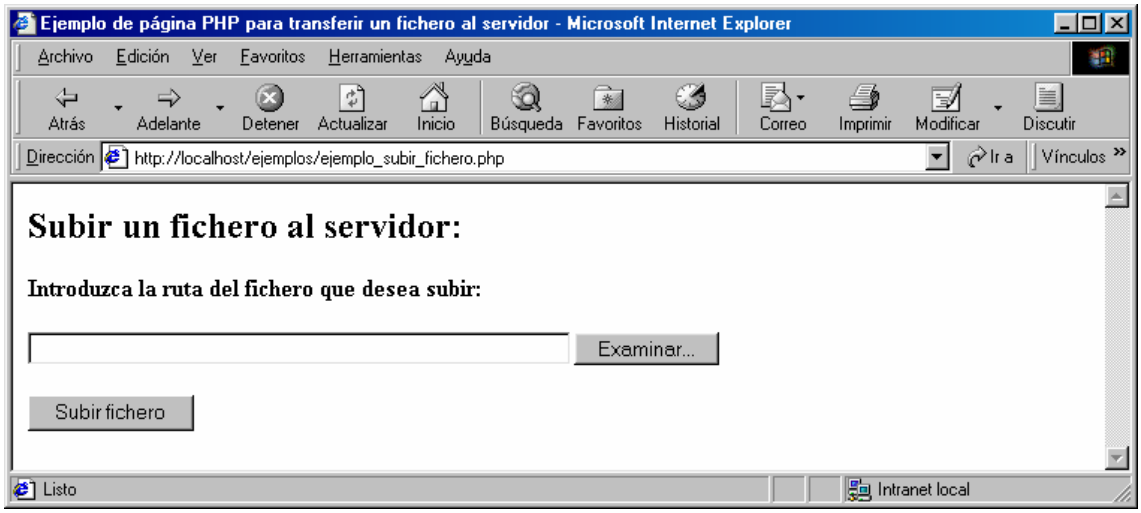
- $fichero_size, es el tamaño en bytes del fichero. Nos permite controlar el espacio que se irá ocupando en el servidor a medida que vamos subiendo ficheros.

Veamos un ejemplo:

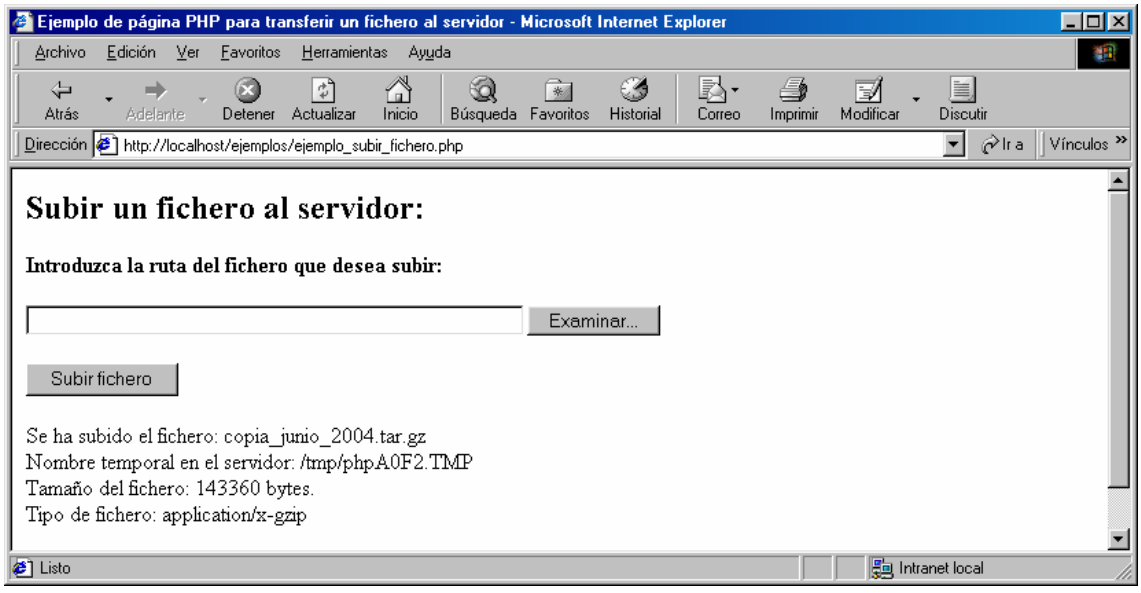
ejemplo_subir_fichero.php

```
<html>
        <head><title>Ejemplo de página PHP para transferir un fichero al servidor</title></head>
        <body>
        <h2>Subir un fichero al servidor:</h2>
        <form action="ejemplo_subir_fichero.php" method="post" enctype="multipart/form-data">
          <strong>Introduzca la ruta del fichero que desea subir: </strong><p>
          <p><input type="file" name="fichero" size="50"><br>
          <p><input type="submit"  value="Subir fichero" >
        </form>
        <?
                // ruta del servidor para copiar el fichero subido
                $ruta_destino="/ficheros/usuarios/";
                // se ha introducido el nombre de un fichero en el  formulario
                if ($fichero !="")  {
                        $destino=$ruta_destino . $fichero_name
                        if ( copy($fichero,$destino) )  {
                                echo " Se ha subido el fichero: " . $fichero_name ."<br>";
                                echo " Nombre temporal en el servidor: " . $fichero ."<br>";
                                echo " Tamaño del fichero: " . $fichero_size ." bytes.<br>";
                                echo " Tipo de fichero: " . $fichero_type ."<br>";
                        }else { // no se puede copiar el fichero
                                echo " No se ha podido copiar el fichero " . $fichero_name;
                        }
                }
         ?>
        </body>
</html>
```

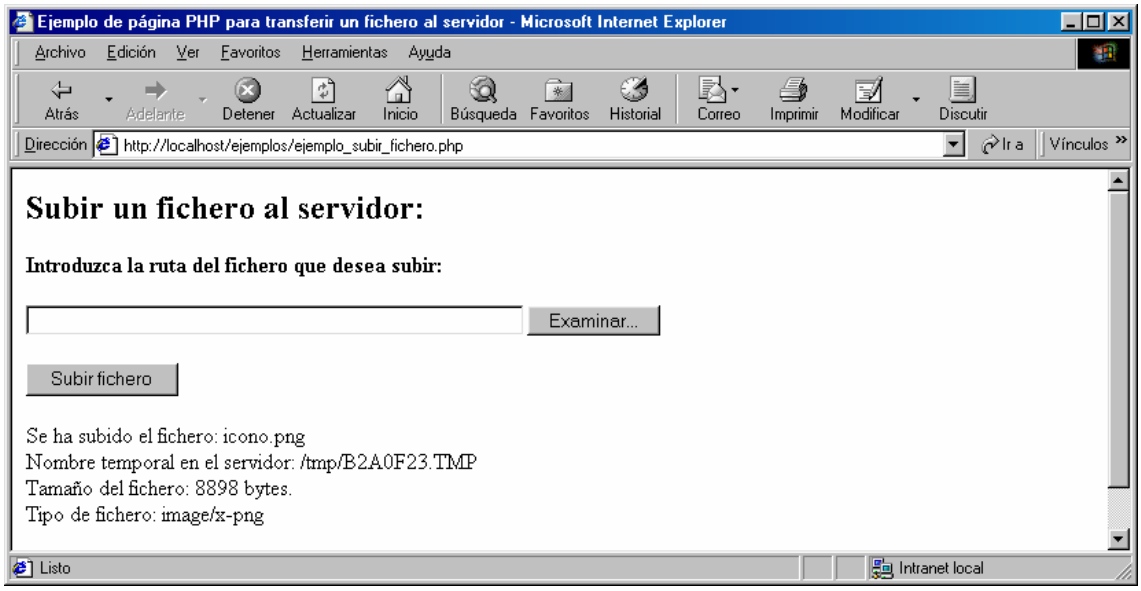
La apariencia de esta página al ejecutarla es la siguiente:



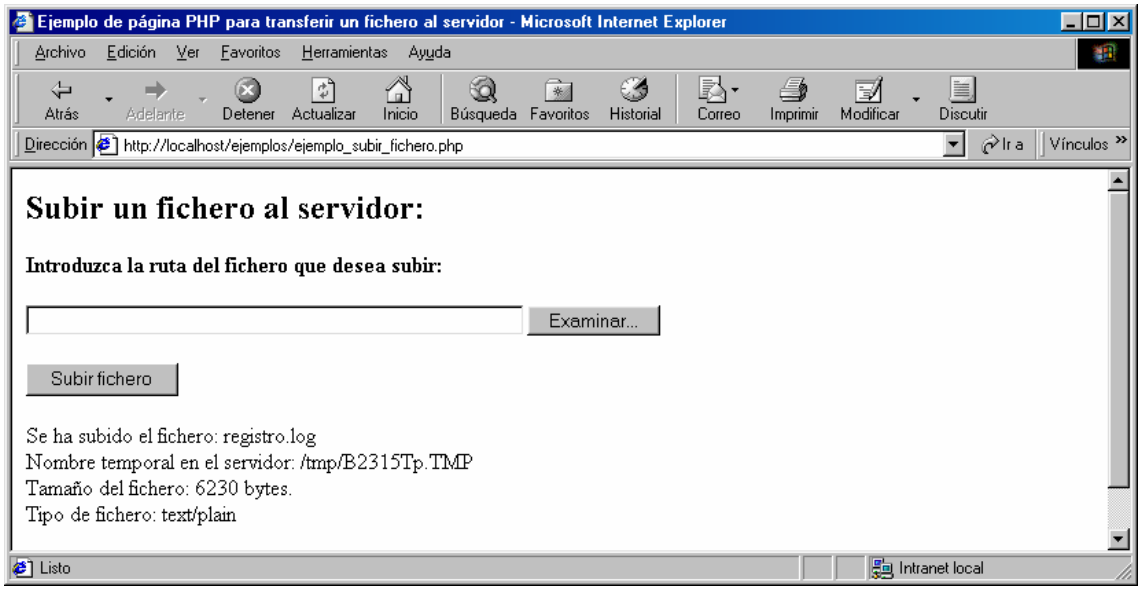
Ante distintas elecciones y tipos de archivos para subirlos al servidor, obtendremos distintas salidas en las variables que nos informan acerca del nombre temporal del fichero utilizado en el servidor, su tamaño y tipo. Por ejemplo, al subir un fichero empaquetado y comprimido .tar.gz:



Si subimos una imagen png:



O bien, un archivo de log:



3.6. ENVIAR VARIOS FICHEROS AL SERVIDOR

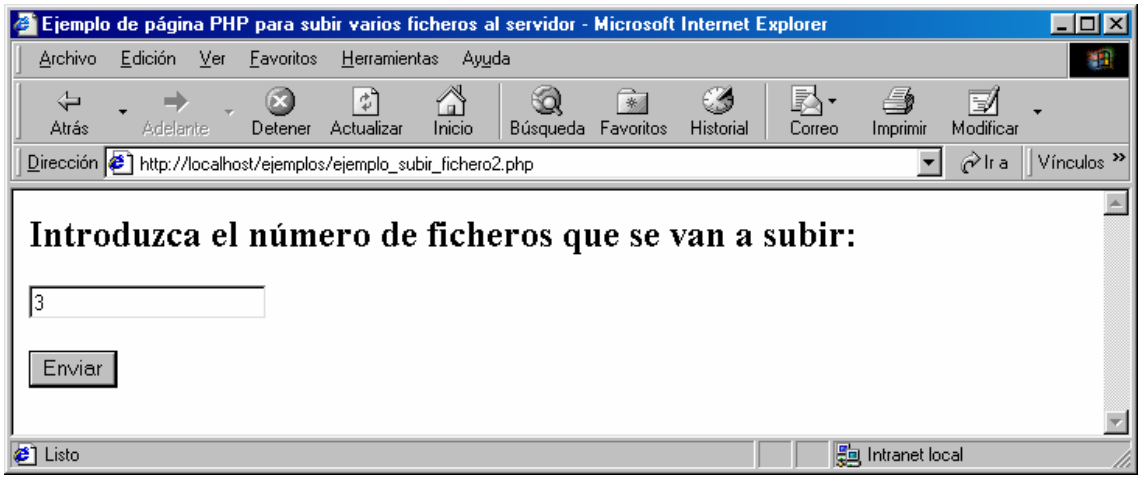
En este apartado veremos un ejemplo de cómo subir varios ficheros al servidor en una única operación. Primero pedimos el número de ficheros a transferir, después, en la siguiente ejecución de la página se genera el formulario con los campos de tipo file necesarios, y por último se envían los ficheros que hemos indicado.

ejemplo_subir_fichero2.php

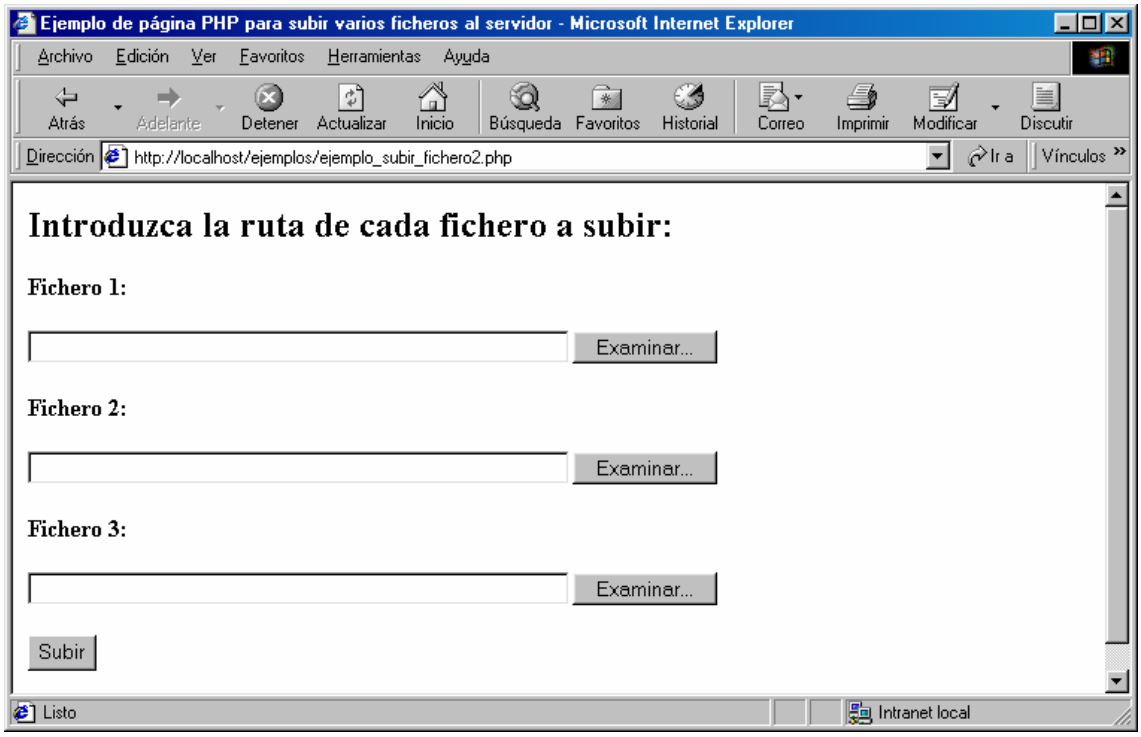
```
<html>
        <head><title>Ejemplo de página PHP para subir varios ficheros al servidor</title></head>
        <body>
        <?
          // primero leemos el número de ficheros a subir
          if  ( empty($numero_ficheros) )  {  // la variable $numero_ficheros está vacía
                echo “<h2>Introduzca el número de ficheros que se van a subir: </h2>”;
         ?>
            <!-- Primer formulario -->
            <form action=”ejemplo_subir_fichero2.php” method=”post”>
              <input type=”text” name=”numero_ficheros” value=”<? echo $numero_ficheros; ?>”>
              <p><input type=”submit”  value=”Enviar”  >
            </form>
        <?
          }
          // a continuación presentamos otro formulario para indicar las rutas de
          // cada fichero a subir
          if  ( empty($boton_aceptar)  &&   !empty($numero_ficheros) )  {
                // tenemos el número de ficheros pero aún no se ha rellenado el 2º formulario
         ?>
            <!-- Segundo formulario -->
            <form action=”ejemplo_subir_fichero2.php” method=”post” enctype=”multipart/form-data”>
        <?
                echo “<h2>Introduzca la ruta de cada fichero a subir: </h2>”;
                for ($i=0; $i<$numero_ficheros; $i++)  {
                        echo "<h4>Fichero ".($i+1).": </h4>";
                        echo “<input type=file name=fichero”. $i . “ size=50><br>”;
                }
                echo “<input type=hidden name=numero_ficheros value=”.$numero_ficheros.“>”;
                echo “<p><input type=submit  name=boton_aceptar value=Subir><p>”;
                echo “</form>”;
          }
```

```
        // por último se procesan los valores leídos:  se suben los ficheros al servidor
        // ruta del servidor para copiar cada fichero subido
        if  ( $boton_aceptar==”Subir” )  {  // se ha enviado el 2º formulario
                $ruta_destino=”/ficheros/usuarios/”;
                echo "<h2>Ficheros subidos al servidor: </h2>";
                for ($i=0; $i<$numero_ficheros; $i++)  {
                        $fichero=”fichero”.$i;
                        $fichero_tmp=$$fichero;
                        $fichero_name="fichero".$i."_name";
                        $fichero_name=$$fichero_name;
                        $fichero_size="fichero".$i."_size";
                        $fichero_size=$$fichero_size;
                        $fichero_type="fichero".$i."_type";
                        $fichero_type=$$fichero_type;
                        // se ha introducido el nombre de un fichero en el  formulario
                        if ($fichero_tmp !="")  {
                                $destino=$ruta_destino. $fichero_name;
                                if ( copy($fichero_tmp,$destino) )  {
                                    echo “ Fichero ” . ($i+1) .”: ”.$fichero_name .”<br>”;
                                    echo “ Nombre temporal en el servidor: ” . $fichero_tmp .”<br>”;
                                    echo “ Tamaño del fichero: ” . $fichero_size .” bytes.<br>”;
                                    echo “ Tipo de fichero: ” . $fichero_type .”<br>”;
                                    echo “ ----------<p>”;
                                }else { // no se puede copiar el fichero
                                    echo “ No se ha podido copiar el fichero ” . $fichero_name;
                                }
                        }
                }
         ?>
        </body>
</html>
```

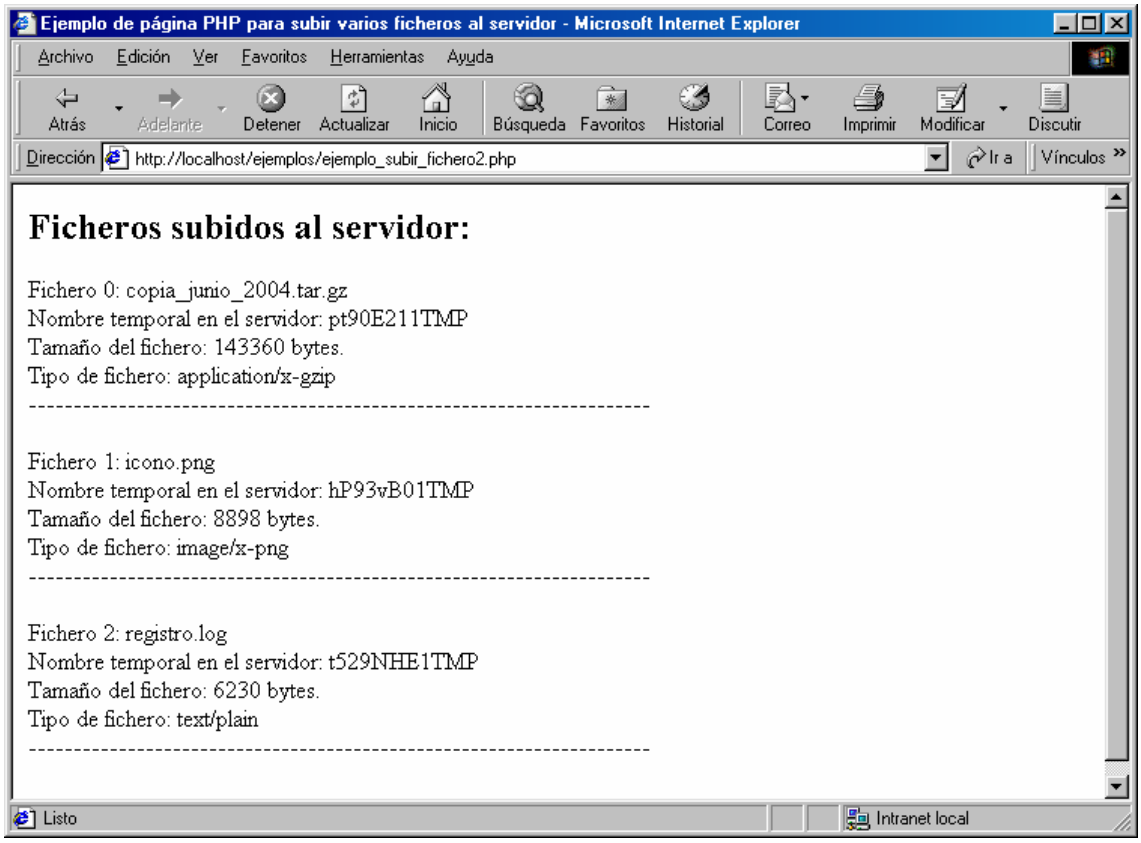
La primera ejecución de esta página PHP mostrará una apariencia como ésta:



En la siguiente llamada a la página, se entrará en el segundo if y presentará el segundo formulario, con tantos campos tipo file como hayamos indicado en el primer formulario. Como en el ejemplo anterior, el formulario que presenta los campos para introducir las rutas de los ficheros a subir al servidor, debe incluir los atributos enctype="multipart/form-data" method="post". Es necesario incluir el valor de la variable $numero_ficheros para la siguiente llamada a la página (la tercera), ya que es necesaria para conocer cuántos ficheros se han subido, y si no se pasa explícitamente para la siguiente llamada a la página, se pierde su valor. El navegador presentará un aspecto similar a éste:



Los campos tipo file de este segundo formulario tienen un atributo name="fichero$i", por tanto, el nombre de cada uno de estos campos se va a diferenciar del resto por un índice (que va desde 0 al número de ficheros menos 1), es decir, que se llamarán "fichero0", "fichero1", "fichero2",... Cuando se procesan los datos enviados en el segundo formulario (tercera llamada a la página) se utilizan **variables variables** para localizar cada nombre temporal de fichero dado por PHP para identificarlo en el servidor, el nombre que tenía en el cliente, el tamaño y tipo. Para cada fichero subido existirán un conjunto de variables: $fichero0, $fichero0_name, $fichero0_size, $fichero0_type para el primer fichero subido, $fichero1, $fichero1_name, $fichero1_size, $fichero1_type para el siguiente, y así sucesivamente. Podemos procesar esta información como en el ejemplo anterior, se hace una copia hacia otro directorio destino y se muestran los datos de cada fichero subido. La última ejecución de la página mostrará una apariencia como ésta:



3.7. PASAR INFORMACIÓN ENTRE PÁGINAS A TRAVÉS DE ENLACES

En los apartados anteriores hemos visto que podemos pasar información entre páginas usando formularios. También podemos hacerlo a través de enlaces. Cuando le pasamos un valor a un enlace, en la página destino PHP crea, al igual que en la página destino de un formulario, la variable con el valor que le hemos asignado. El nombre de la variable y su valor se indican a continuación de la ruta del enlace y el carácter **?** . El nombre de la variable y su valor van separados por el carácter = . Por ejemplo:

```
<a href=”pagina2.php?variable1=valor1”> enlace que pasa una variable a pagina2.php </a>
```

Al pulsar este enlace se llamará a pagina2.php, y en ella estará disponible la variable $variable1 con el valor valor1. Supongamos que tenemos estas dos páginas PHP:

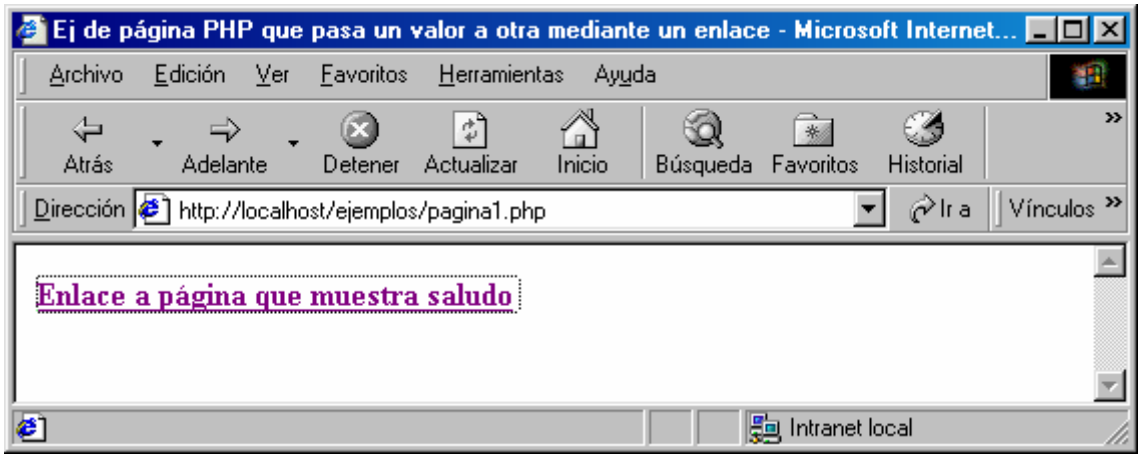
pagina1.php

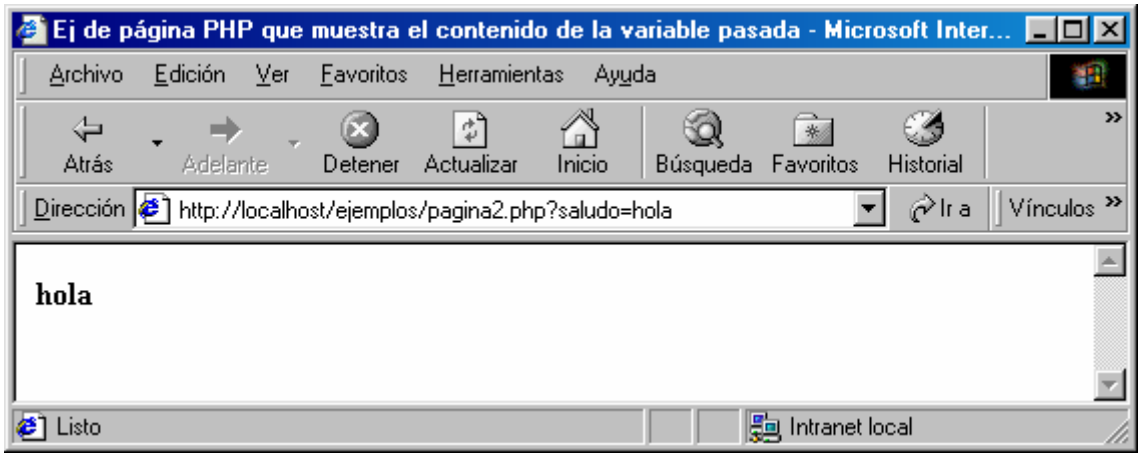
```
<html>
        <head><title>Ej de página PHP que pasa un valor a otra mediante un enlace</title></head>
        <body>
            <h4><a href=”pagina2.php?saludo=hola”>Enlace a página que muestra saludo </a></h4>
        </body>
</html>
```

pagina2.php

```
<html>
        <head><title>Ej de página PHP que muestra el contenido de la variable pasada</title></head>
        <body>
                <?
                        echo “<h4>”.$saludo.“</h4>”;
                ?>
        </body>
</html>
```

Al ejecutar pagina1.php y pulsar en el enlace que muestra para navegar hasta pagina2.php, veremos:





Cuando pasamos datos de esta forma, a través de enlaces, son visibles en la ventana del navegador y además pueden ser modificados, con lo que es posible cambiar la información pasada en la llamada a la página destino. Por ello, debemos tener cuidado con la información que pasamos por este método.

También podemos pasar varios valores a la página destino. En este caso, cada pareja de variable y valor se separa por el carácter & . Por ejemplo:

 enlace que pasa varias variables a la página muestraFecha.php

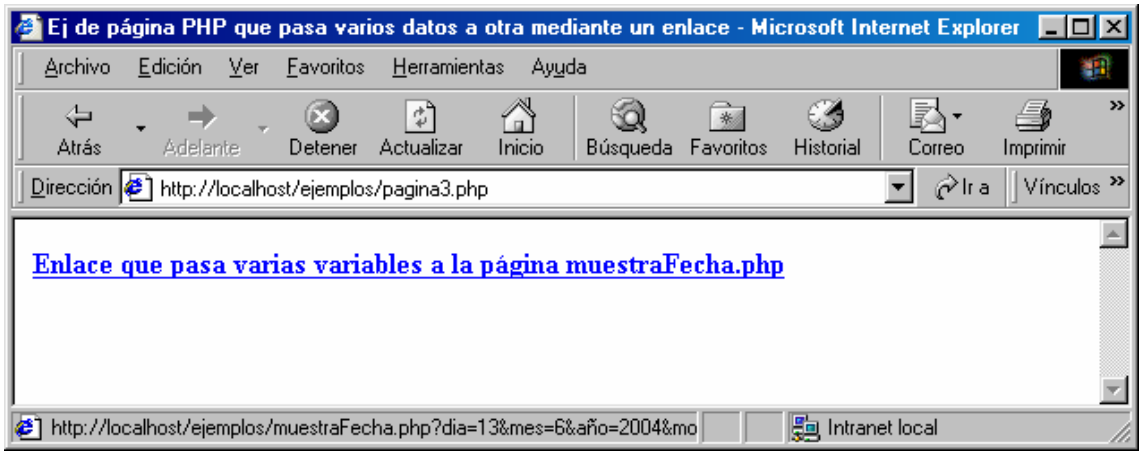
En la página destino muestraFecha.php se crearán automáticamente las cuatro variables $dia, $mes, $año y $modo, con los valores que le hemos asignado a cada una en el enlace.

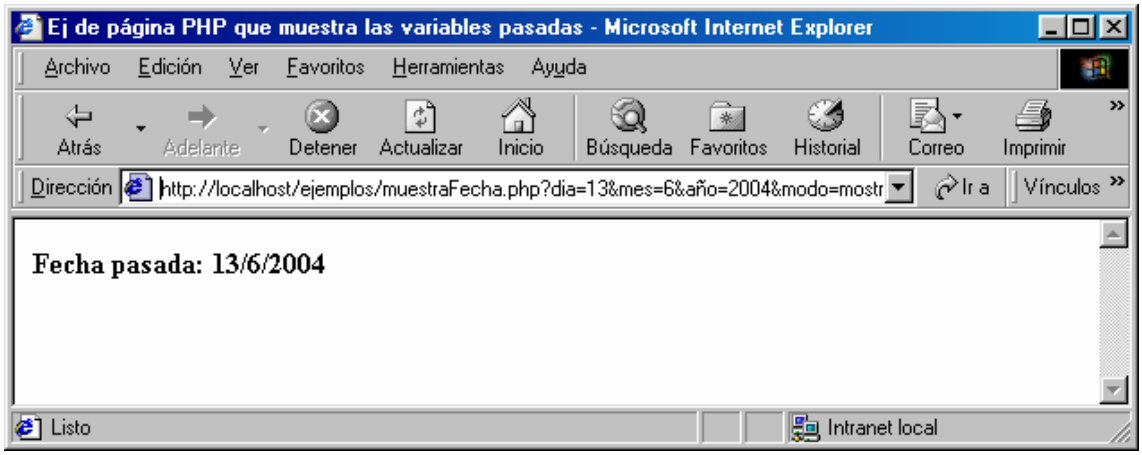
pagina3.php

```
<html>
        <head><title>Ej  de página PHP que pasa varios datos a otra mediante un enlace</title></head>
        <body>
            <h4><a  href="muestraFecha.php?dia=13&mes=6&año=2004&modo=mostrar"> Enlace  que
            pasa varias variables a la página muestraFecha.php </a>
        </h4>
        </body>
</html>
```

muestraFecha.php

```
<html>
        <head><title>Ej  de página PHP que muestra las variables pasadas</title></head>
        <body>
        <?
            if ($modo=="mostrar") {
                    echo "<h4>Fecha pasada: ".$dia."/".$mes. "/".$año."</h4>";
                    }
        ?>
        </body>
</html>
```





Además, al pasar valores a una página a través de un enlace, debemos tener en cuenta que los espacios en blanco y algunos caracteres que tienen un significado especial en HTML, pueden causar problemas. Por ejemplo, el carácter & dentro del valor dado a una de las variables pasadas, porque se confundiría con el carácter separador de los pares variable-valor, las comillas, el carácter =, ... Para poder incluir los espacios en blanco y esos caracteres especiales, PHP dispone de la función urlencode ( ) , que toma como argumento una cadena de caracteres y la devuelve codificando los caracteres que pueden dar algún problema en un enlace, y de la función urldecode ( ), que realiza la operación inversa, es decir, toma como argumento una cadena de caracteres codificada por urlencode ( ), y devuelve la cadena original. La sintaxis de estas funciones es:

```
String urlencode (string cadena);

String urldecode (string cadena);
```

Usaremos la función urlencode ( ) para pasar los datos en un enlace para que sean correctamente interpretados, y en la página destino la función urldecode ( ) para decodificar los valores pasados y obtener los caracteres originales. Por ejemplo:

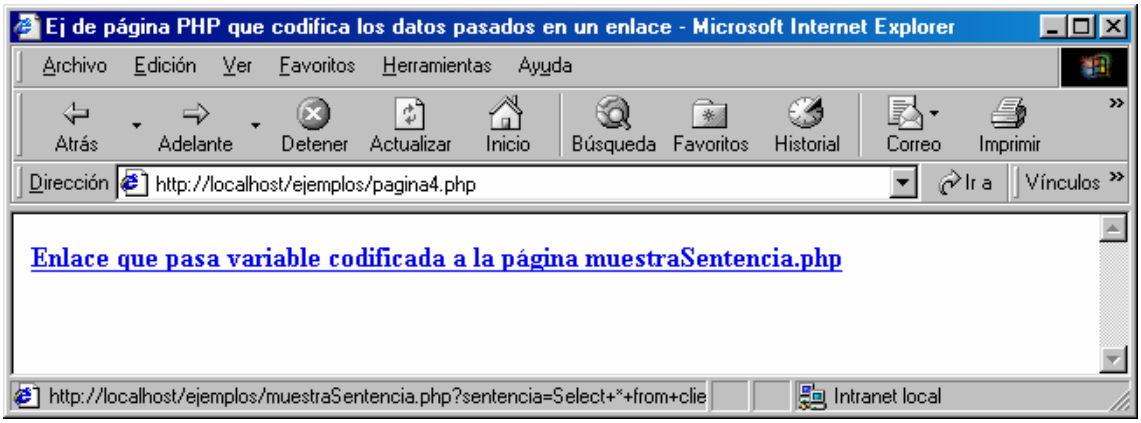
Pagina4.php

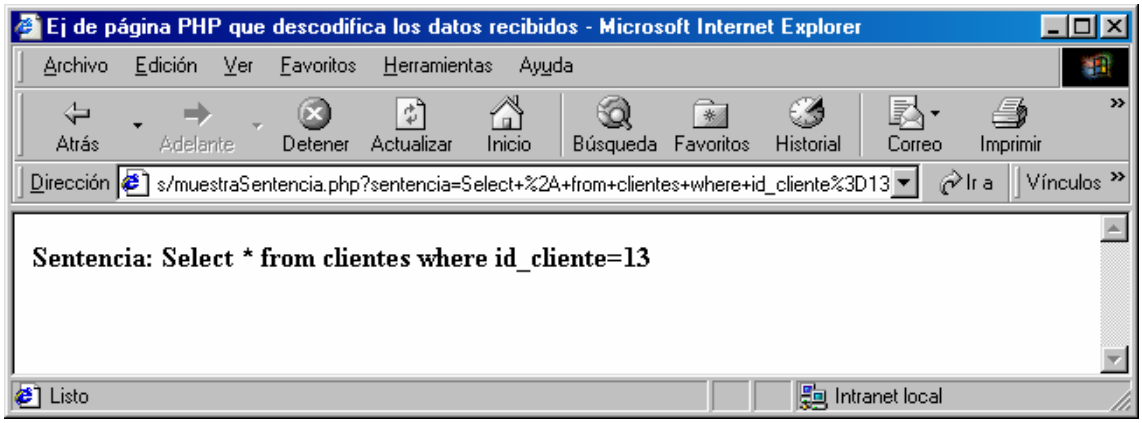
```
<html>
        <head><title>Ej  de página PHP que codifica los datos pasados en un enlace</title></head>
        <body>
                <?
                        $sentencia="Select * from clientes where id_cliente=13";
                ?>

                <h4><a href="muestraSentencia.php?sentencia=<? echo urlencode($sentencia); ?>" >
                    Enlace que pasa  variable codificada a la página muestraSentencia.php </a>
                </h4>
        </body>
</html>
```

muestraSentencia.php

```
<html>
        <head><title>Ej de página PHP que descodifica los datos recibidos</title></head>
        <body>
                <?
                        echo "<h4>Sentencia: ".urldecode($sentencia)."</h4>";
                ?>
        </body>
</html>
```

El navegador mostraría:





Podemos observar en la barra de direcciones del navegador que el valor dado a la variable 'sentencia' se ha pasado codificado (los espacios en blanco se han sustituido por el carácter +, el asterisco por el hexadecimal %2A, el carácter = por el hexadecimal %3D), y que así, le llega correctamente el contenido de $sentencia a la página muestraSentencia.php, donde se muestra.

RECUERDE

- Podemos pasar información entre páginas PHP a través de formularios o a través de enlaces. Es muy importante tener en cuenta el valor de la directiva de configuración 'register_globals' (se encuentra en el fichero php.ini), porque influye en el conjunto de variables predefinidas disponibles en el sistema y en la forma de acceder a ellas. A lo largo de este capítulo hemos supuesto que esta directiva estaba activada, por lo que las variables globales correspondientes a las entradas de usuario a través de formularios o las variables pasadas en enlaces, están disponibles automáticamente como variables globales en la página.

- La página destino que procesa un formulario puede ser otra página PHP o bien, la misma que presenta dicho formulario. Una misma página PHP puede actuar de diversos modos según el estado en que se encuentre (por ejemplo, mostrar un formulario si es la primera vez que se llama, procesar dicho formulario una vez recibido, hacer una consulta y presentar datos, pedir otro nuevo formulario complementario si no se han introducido correctamente todos los datos, etc). Podemos distinguir en qué estado se encuentra la página en cada ejecución por el valor de las variables que le llegan.

- Los formularios pueden incluir también campos de tipo oculto (<input type="hidden">), para pasar información complementaria a la página destino. Estos campos no tienen una representación visual en el formulario, y pueden tomar su valor a partir de otras variables del formulario, como resultado de alguna expresión, para mantener información de estado entre distintas llamadas a la misma página PHP, ...

- El número de campos que contiene un formulario puede variar en función de algún valor (tomado en una ejecución anterior de la página, por ejemplo), y que distintas ejecuciones muestren formularios con distinta longitud (formularios de tamaño variable).

- Podemos transferir ficheros al servidor usando formularios con los atributos enctype="multipart/form-data" method="post". En la página destino se crearán automáticamente algunas variables que nos informarán de la ruta del fichero en el servidor, el nombre en el equipo cliente, el tamaño y tipo, que podremos consultar para operar con el fichero subido.

– Al pasar variables a través de enlaces, si éstas pueden incluir espacios en blanco o caracteres especiales, como &, =, comillas, ... es necesario utilizar las funciones urlencode ( ) y urldecode ( ) para que estos caracteres se interpreten adecuadamente.

Tema 4

Sesiones

4.1. INTRODUCCIÓN

A partir de la versión 4.0., PHP también tiene integrada la posibilidad de mantener y gestionar sesiones de trabajo para cada cliente que está usando la aplicación. A través de las sesiones podemos guardar y mantener entre distintas páginas la información concerniente a cada usuario (esta información estará disponible en cada página PHP que conserve la sesión, y por tanto, no tendremos que pasarla como el resto de datos, por el método de formularios o enlaces).

Hay aplicaciones web en las que es fundamental el seguimiento de las acciones que realiza un cliente mientras navega por sus páginas para su correcto funcionamiento. Por ejemplo, en un tienda virtual, debemos conocer el carro que está asociado a cada comprador, el pedido que está haciendo, y quién es dicho comprador, para poder mostrar adecuadamente los productos que está comprando, hacer el cálculo correcto de los precios de productos que ha seleccionado, guardar su pedido y datos personales, etc.

Sería muy tedioso tener que pasar toda la información identificativa necesaria para gestionar el sistema (que puede aumentar si aplicamos también ofertas personalizadas por ejemplo, histórico de pedidos, el operador de logística asociado, notificaciones al comercio donde se está comprando, etc) entre cada página.

Otro ejemplo de aplicación web que necesita gestionar sesiones de trabajo para cada cliente es una plataforma de teleformación, donde se debe saber quién es el alumno que ha entrado, presentar los cursos en los que está matriculado, guardar las notas de sus ejercicios y exámenes, los temas visitados en el curso que está viendo, etc. Por tanto, es fundamental mantener el identificador del alumno y del curso actuales para cada cliente que esté entrando en la plataforma.

La solución está en mantener la información que está asociada a cada cliente y que es necesaria durante la navegación entre las distintas páginas de la aplicación para poder identificarlo, en variables de sesión. A cada visitante que acceda a la aplicación, se le asigna un identificador único, identificador de sesión, que se almacena en una cookie en el equipo del cliente o se propaga en la URL.

4.2. GESTIÓN DE SESIONES

Veamos las principales funciones disponibles en PHP para gestionar sesiones:

4.2.1. Función session_start

Su sintaxis es:

```
booleano session_start ( );
```

Usaremos esta función para crear una sesión de trabajo (en caso de no haberla creado todavía) o para continuarla (si ya la creamos antes). Esta función envía una cabecera, por lo que hay que llamarla antes que cualquier otra salida HTML. Al crearse la sesión, PHP guarda el identificador de la misma en una variable de entorno llamada $PHPSESSID.

4.2.2. Función session_name

Su sintaxis es:

```
string session_name (string nombre);
```

Con esta función podemos fijar el nombre de la sesión activa con el argumento que le pasamos. Si no le pasamos argumento sólo nos devuelve el nombre de la sesión activa.

No es obligatorio establecer un nombre a la sesión activa con esta función, ya que PHP proporciona uno por defecto. Si queremos fijar el nombre de la sesión a uno propuesto por nosotros, debemos llamar a session_name ( ) antes de la función session_start ( ) o session_register ( ) (esta función la veremos en el siguiente apartado), y por supuesto, antes de cualquier salida HTML.

4.2.3. Función session_id

Su sintaxis es:

```
string session_id (string id);
```

Esta función devuelve el identificador de la sesión actual. Si se le pasa el argumento id, reemplazará el identificador de sesión actual. También está disponible la constante SID, que contiene una cadena con el nombre y el id de sesión actuales.

4.2.4. Función session_regenerate_id

Su sintaxis es:

booleano session_regenerate_id ( );

Esta función actualiza el identificador de sesión actual con uno nuevo, y conservará la información de sesión actual. Devuelve verdadero si la operación se realiza correctamente, y falso en caso contrario. Al llamar a esta función se enviará una nueva cookie de sesión con el nuevo identificador de sesión.
Veamos un ejemplo de su uso:

ejemplo_actualiza_id_sesion.php

```
<?php
        session_start();
        $id_sesion_antigua = session_id();
        session_regenerate_id();
        $id_sesion_nueva = session_id();
?>
<html>
        <head><title>Ejemplo de página PHP con sesiones</title></head>
        <body>
        <?
                echo "Sesión antigua: $id_sesion_antigua<br>";
                echo "Sesión nueva: $id_sesion_nueva<br>";
                echo "PHPSESSID: “.$PHPSESSID.” <br>";
        ?>
        </body>
</html>
```

4.2.5. Función session_register

Su sintaxis es:

booleano session_register (string nombre_variable1, string nombre_variable2, ...);

Con esta función registramos las variables pasadas como argumento como variables pertenecientes a la sesión de trabajo. Como estas variables formarán parte de la sesión, no tenemos que pasarlas explícitamente al resto de páginas que comparten la sesión (aquellas páginas que llaman a session_start ( ) para continuar la sesión activa).

4.2.6. Función session_unregister

Su sintaxis es:

booleano session_unregister (string nombre_variable);

Esta función elimina una variable de la sesión de trabajo (la deja como si no hubiese sido registrada).

4.2.7. Función session_is_registered

Su sintaxis es:

booleano session_is_registered (string nombre_variable);

Con esta función podemos saber si la variable pasada como argumento está registrada en la sesión de trabajo activa, es decir, si hemos llamado antes a la función session_register ( ) con dicha variable. Devuelve verdadero si está registrada y falso en caso contrario.

4.2.8. Función session_encode

Su sintaxis es:

string session_encode ( );

A esta función no le pasamos ningún argumento. Devuelve una cadena que contiene todos los valores de la sesión de trabajo activa codificados.

4.2.9. Función session_decode

Su sintaxis es:

string session_decode (string cadena_sesión_codificada);

Esta función realiza la operación inversa de session_encode ( ), es decir, le pasamos la cadena codificada y devuelve los valores originales de las variables de la sesión.

4.2.10. Función session_destroy

Su sintaxis es:

booleano session_destroy ( );

Al llamar a esta función eliminaremos la sesión de trabajo activa con todas las variables de sesión registradas. Por lo que, no se guardará ni mantendrá información de sesión hasta que se vuelva a llamar a session_start ( ) para crearla y continuarla y se guarden variables de sesión.

Veamos un ejemplo para ilustrar cómo registrar variables de sesión y comprobar que se mantienen entre las distintas páginas que comparten la sesión de trabajo:

ejemplo_sesiones1_pagina1.php

```
<?php
        session_start();
        session_register("contador");
        session_register("usuario");
        if (empty($contador)) {
                $contador=1;
        }else {
                $contador++ ;
        }
        $usuario="Luis";
        // comienza una sesión o continúa la sesión activa
        // Registra las variables $contador y $usuario como datos de la sesión con session_register
?>
```

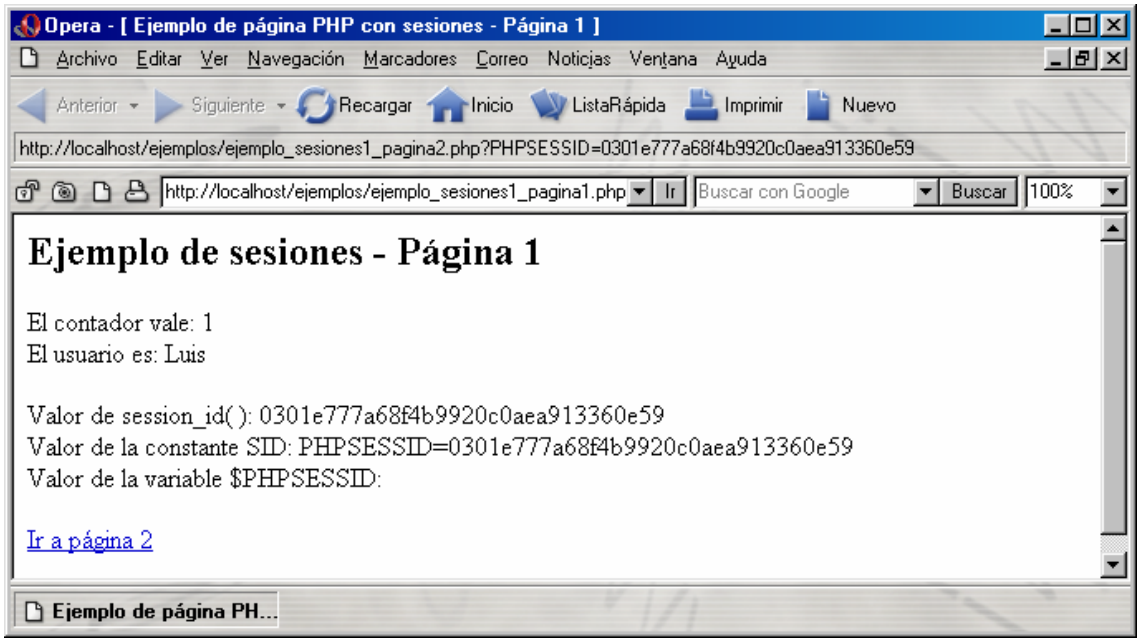
```
<html>
	<head>
	   <title>Ejemplo de página PHP con sesiones – Página 1</title>
	</head>
	<body>
		<h2>Ejemplo de sesiones – Página 1</h2>
	<?
		echo "El contador vale: ".$contador." <br>";
		echo "El usuario es: ".$usuario." <p>";
		echo "Valor de session_id( ): ".session_id()." <br>";
		echo "Valor de la constante SID: ".SID." <br>";
		echo "Valor de la variable \$PHPSESSID: ".$PHPSESSID." <p>";
		echo “<a href=’ejemplo_sesiones1_pagina2.php’>Ir a página 2</a>";
	?>
	</body>
</html>
```

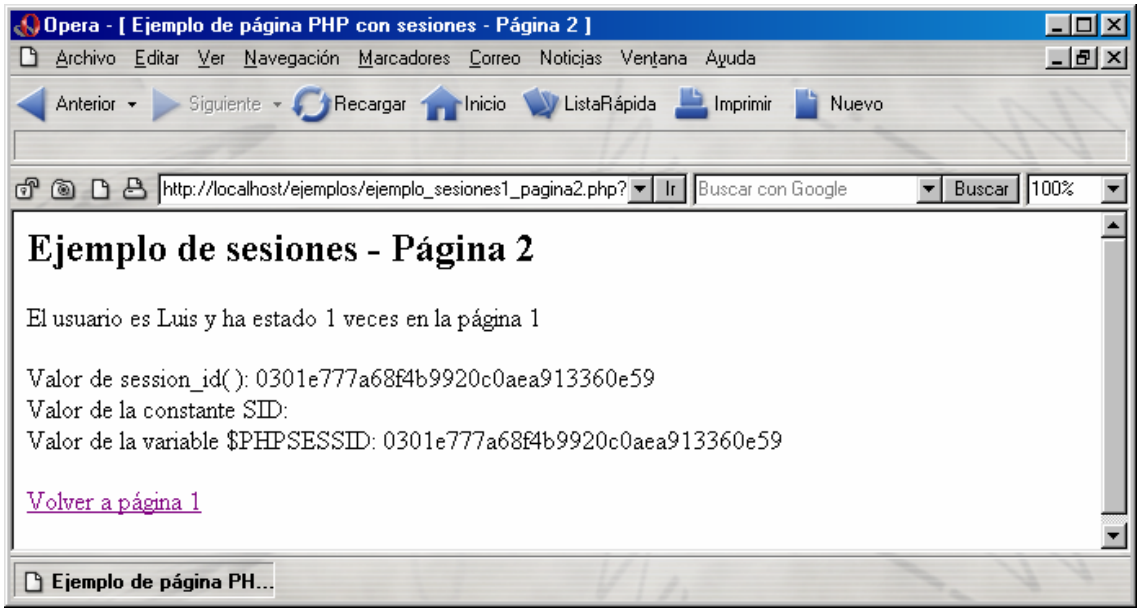
ejemplo_sesiones1_pagina2.php

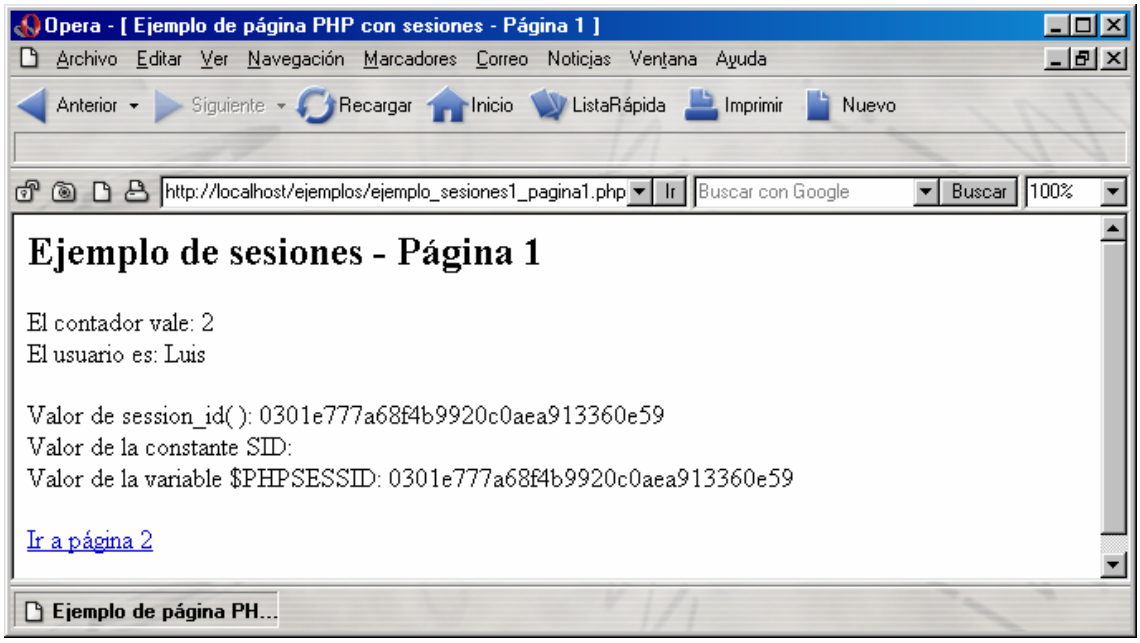
```
<?php
	session_start();
	session_register(“contador”);
	session_register(“usuario”);
	// comienza una sesión o continúa la sesión activa
	// Registra las variables $contador y $usuario como datos de la sesión con session_register
?>
<html>
	<head>
	   <title>Ejemplo de página PHP con sesiones – Página 2</title>
	</head>
	<body>
		<h2>Ejemplo de sesiones – Página 2</h2>
	<?
		echo "El usuario es ".$usuario." y ha estado “.$contador.” veces en la página 1<p>";
		echo "Valor de session_id( ): ".session_id()." <br>";
		echo "Valor de la constante SID: ".SID." <br>";
```

```
            echo "Valor de la variable \$PHPSESSID: ".$PHPSESSID." <p>";
            echo “<a href=’ejemplo_sesiones1_pagina1.php’>Volver a página 1</a>”;
        ?>
        </body>
</html>
```

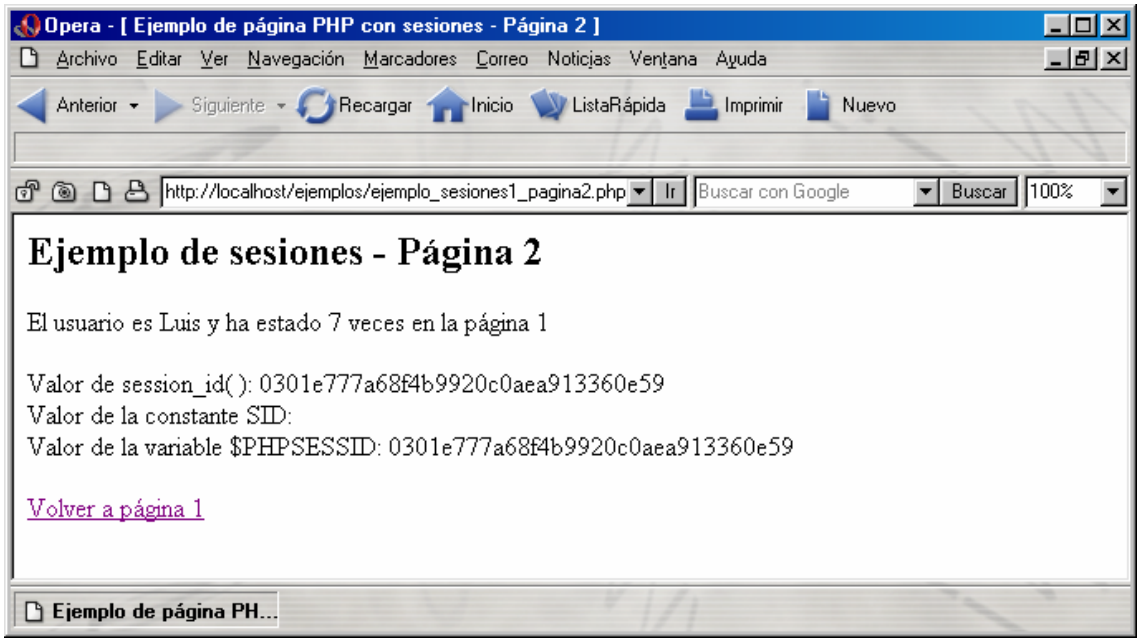
El resultado de ejecutar estas páginas y navegar entre ellas es el que se muestra a continuación:







...



Podemos ver cómo son conocidas las variables de sesión que hemos registrado, $contador y $usuario, entre las dos páginas que mantienen la sesión, donde podemos acceder a ellas para consultarlas o modificarlas. También se muestran los datos identificativos de la sesión de trabajo actual, el id de sesión devuelto por la función session_id( ), el valor de la constante SID (formada por nombre_de_sesion=session_id o por una cadena vacía) y el valor de la variable de entorno $PHPSESSID, que también almacena el identificador de sesión.

4.3. ARRAYS $HTTP_SESSION_VARS y $_SESSION

También tenemos disponible el array $HTTP_SESSION_VARS para acceder a los datos de la sesión, y a partir de la versión 4.1.0 de PHP, el array $_SESSION.

Si queremos usar $HTTP_SESSION_VARS dentro de una función o método, debemos declararlo como global. Sin embargo, $_SESSION está siempre disponible como variable global, es decir, no se debe usar el modificador global para utilizarlo dentro de funciones o métodos, es siempre conocido en cualquier contexto del script.

Se recomienda usar $_SESSION (o $HTTP_SESSION_VARS con PHP 4.0.6. o inferior), hace el código más legible y también porque permitir separar las variables que le llegan al script de los datos de usuario.

Cuando usamos estos arrays, no es necesario usar las funciones session_register(), session_unregister() y session_is_registered() para registrar, desregistrar o comprobar si una variable está en la sesión de trabajo actual. Para acceder a una variable de sesión en este caso lo haremos como si fuese una variable normal, con la sintaxis propia de los arrays asociativos (los vimos en el apartado 2.7). Para saber si está definida una variable de sesión podemos usar la función isset() y para eliminar una variable de sesión, usaremos la función unset(), que borra la variable pasada como argumento. En ambas funciones pasaremos el valor del array asociativo ($_SESSION o $HTTP_SESSION_VARS) que tiene como clave el nombre de la variable.

NOTA:

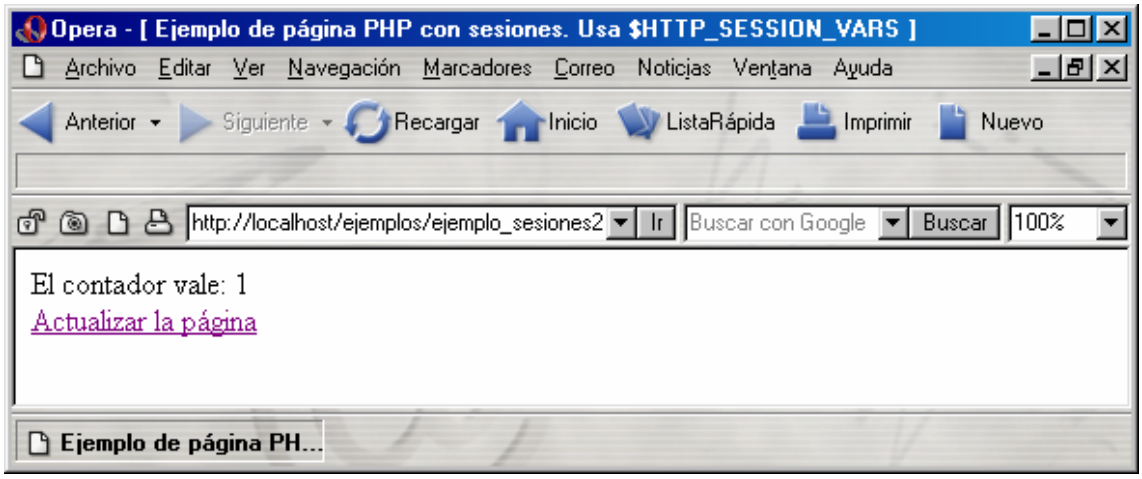
Es muy importante saber que, a partir de PHP 4.2.0, el valor por defecto de la directiva PHP 'register_globals' es *off* (desactivada), y, con esta configuración de PHP, deberemos usar los arrays asociativos $_SESSION (o $HTTP_SESSION_VARS con PHP 4.0.6. o inferior) para mantener la información de sesión, y no las funciones session_register(), session_unregister() y session_is_registered(). El resto de funciones, como session_start() o session_destroy() están disponibles en ambos casos. Podemos modificar el valor de esta directiva en el archivo de configuración de PHP, el fichero php.ini, aunque es recomendable mantener desactivada 'register_globals' por motivos de seguridad, rendimiento y legibilidad del código.

Veamos algunos ejemplos:

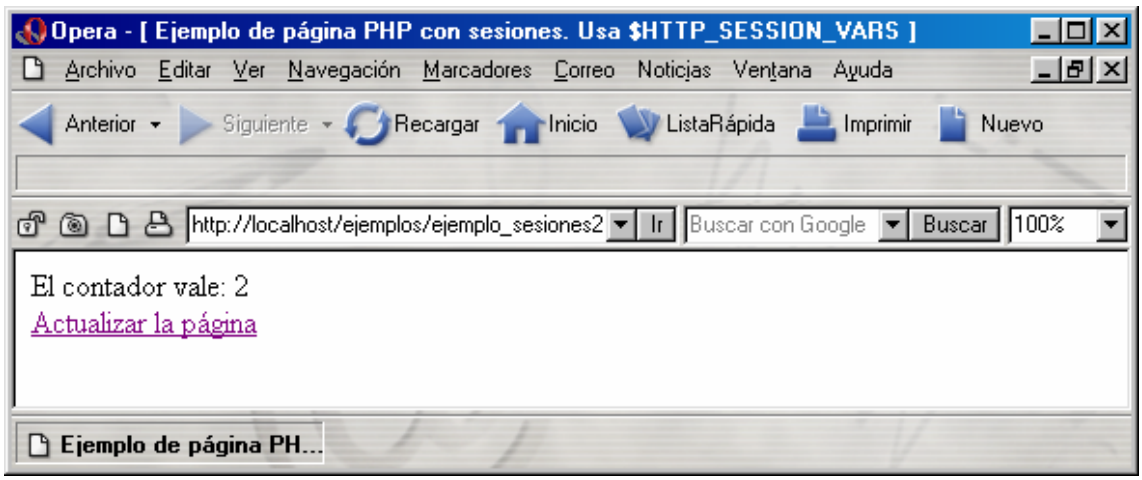
ejemplo_sesiones2.php

```
<?php
        session_start();
        if (isset($HTTP_SESSION_VARS['contador'])) {
                $HTTP_SESSION_VARS['contador']++;
        }else {
                $HTTP_SESSION_VARS['contador'] = 1;
        }
        // comienza una sesión o continúa la sesión activa
        // Si no existe la variable contador en la sesión, la define e inicializa
        // Si existe la variable de sesión contador, la incrementa
?>
<html>
        <head>
            <title>Ejemplo de página PHP con sesiones. Usa $HTTP_SESSION_VARS</title>
        </head>
        <body>
        <?
                echo "El contador vale: ".$HTTP_SESSION_VARS["contador"]." <br>";
                echo “<a href=’ejemplo_sesiones2.php’>Actualizar la página</a>”;
        ?>
        </body>
</html>
```

Este ejemplo mantiene un contador como información de sesión (por eso no tenemos que pasar dicha variable cada vez que actualizamos la página o volvemos a ella a través del enlace que apunta a sí misma). En cada llamada a la página se incrementa el contador, y muestra el número de veces que se ha visitado la página. En el navegador veríamos:



Al actualizar la página va incrementándose el contador:



Debemos comprobar que nuestro navegador acepta las cookies, para que funcione la gestión de sesiones (por defecto, el mantenimiento de la sesión de trabajo se basa en cookies, directiva de configuración '**session.use_cookies**' en **php.ini**).

Usando el array $_SESSION, el ejemplo anterior quedaría:

ejemplo_sesiones3.php

```
<?php
        session_start();
        if (isset($_SESSION['contador'])) {
                        $_SESSION['contador']++;
        }else {
```

```
                $_SESSION['contador'] = 1;
            }
        // comienza una sesión o continúa la sesión activa
        // Si no existe la variable contador en la sesión, la define e inicializa
        // Si existe la variable de sesión contador, la incrementa
?>
<html>
        <head>
            <title>Ejemplo de página PHP con sesiones. Usa $_SESSION</title>
        </head>
        <body>
        <?
            echo "El contador vale: ".$_SESSION["contador"]." <br>";
            echo “<a href=’ejemplo_sesiones3.php’>Actualizar la página</a>”;
        ?>
        </body>
</html>
```

Ejemplo para eliminar una variable de sesión con el array $HTTP_SESSION_VARS:

```
<?php
        session_start();
        unset($HTTP_SESSION_VARS['contador']);
?>
```

Y con el array $_SESSION:

```
<?php
        session_start();
        unset($_SESSION['contador']);
?>
```

4.4. OPCIONES DE CONFIGURACIÓN

En este apartado vamos a ver las principales opciones de configuración relacionadas con el sistema de control de sesiones. Podemos consultarlas y modificarlas en el archivo de configuración de PHP, php.ini:

- session.save_path: por defecto indica la ruta donde se crean los archivos de las sesiones, que es el directorio /tmp.

- session.name: especifica el nombre de la sesión que se usa como nombre de la cookie. Su valor por defecto es PHPSESSID. Podemos modificarlo por cualquier otro nombre formado por caracteres alfanuméricos.

- session.auto_start: indica si el sistema de control de sesiones inicia una sesión automáticamente al comenzar la petición de alguna página. Por defecto está desactivado, por lo que, para que comience una sesión debemos llamar a la función session_start( ) que ya hemos visto.

- session.use_cookie: indica si el sistema de control de sesiones puede usar cookies para guardar el identificador de sesión en el lado del cliente. Por defecto está activado, por eso se envía una cookie para mantener la sesión (en este caso, el navegador del cliente deberá estar configurado para aceptarlas para que funcione el mantenimiento de la sesión de trabajo).

- session.cookie_lifetime: esta opción indica la duración en segundos de la cookie que se manda al navegador. Por defecto tiene el valor 0, y se interpreta como que la cookie debe estar hasta que se cierra el navegador.

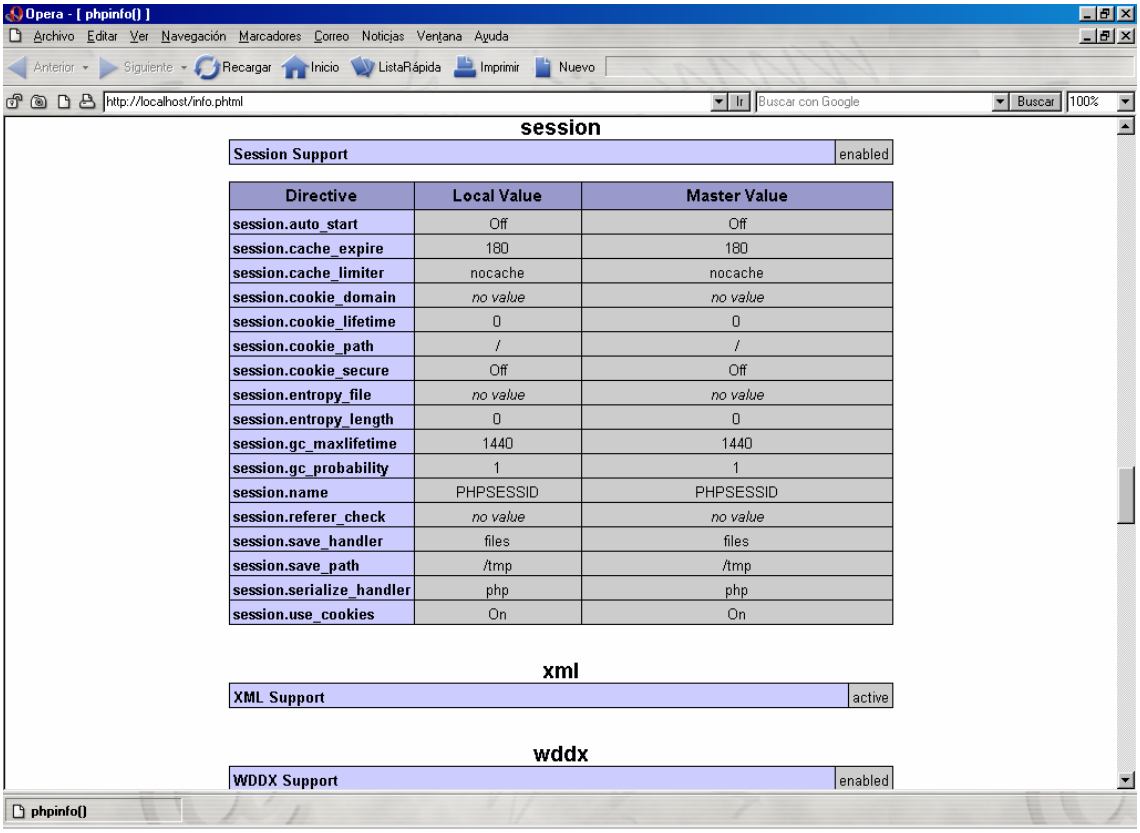
- session.cache_limiter: especifica el método de control de la caché en la páginas de la sesión. Por defecto su valor es 'nocache', que significa no guardar las páginas en la caché, por lo que debería servir siempre la versión actualizada de la página.

- session.cache_expire: este parámetro actúa en combinación con el anterior, especifica el tiempo de vida en minutos de las páginas de la sesión que se encuentran en la caché. Por defecto tiene el valor 180, pero no se tendrá en cuenta si session.cache_limiter tiene el valor 'nocache'.

Vimos en el primer capítulo del manual que la llamada a la función phpinfo( ) era muy útil para conocer los datos de configuración de nuestro sistema (versiones usadas de PHP y Apache, sistema operativo, módulos incorporados, variables PHP y variables de entorno predefinidas con los valores que están tomando, configuración del control de las sesiones, etc). Es suficiente un script como el siguiente para que podamos ver toda esta información:

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

La siguiente captura muestra la porción dedicada a la configuración de sesiones de la salida que nos proporciona phpinfo(). Podemos consultar esta información para conocer el método de mantenimiento de las sesiones en nuestro sistema:



Por ejemplo, en este sistema las sesiones se basan en cookies, los ficheros para mantenerlas se crean en el directorio /tmp y las páginas que intervengan en una sesión no deben cachearse.

RECUERDE

- A partir de la versión 4.0., PHP también tiene integrada la posibilidad de mantener y gestionar sesiones de trabajo para cada cliente que está usando la aplicación.

- A cada visitante que acceda a la aplicación, se le asigna un identificador único, identificador de sesión, que se almacena en una cookie en el equipo del cliente o se propaga en la URL.

- Existen distintas funciones para el manejo de sesiones, como session_start() para comenzar una sesión o continuar la sesión activa, session_id() que devuelve el identificador de sesión, session_register( ) para registrar variables de sesión, etc.

- También tenemos disponible el array $HTTP_SESSION_VARS para acceder a los datos de la sesión, y a partir de la versión 4.1.0 de PHP, el array $_SESSION.

- Si queremos usar $HTTP_SESSION_VARS dentro de una función o método, debemos declararlo como global. Sin embargo, $_SESSION es siempre conocido en cualquier contexto del script.

- Se recomienda usar $_SESSION (o $HTTP_SESSION_VARS con PHP 4.0.6. o inferior), hace el código más legible y también porque permitir separar las variables que le llegan al script de los datos de usuario. Son arrays asociativos.

- Debemos tener en cuenta que, a partir de PHP 4.2.0, el valor por defecto de la directiva PHP 'register_globals' es *off* (desactivada), y, con esta configuración de PHP, deberemos usar los arrays asociativos $_SESSION o $HTTP_SESSION_VARS para mantener la información de sesión, y no las funciones session_register(), session_unregister() y session_is_registered().

- Podemos modificar el valor de la directiva 'register_globals' en el archivo de configuración de PHP, el fichero php.ini, aunque es recomendable mantenerla desactivada por motivos de seguridad, rendimiento y legibilidad del código.

- Podemos consultar y modificar las opciones de configuración relacionadas con el sistema de control de sesiones en el archivo php.ini, como activar el uso de cookies para mantener las sesiones, el directorio donde se crearán los archivos de cookies, el tiempo de vida de las cookies o el control de la caché de las páginas que intervengan en las sesiones.

Tema 5

Variables predefinidas

5.1. INTRODUCCIÓN

PHP proporciona una gran cantidad de variables predefinidas a cualquier script que se ejecute y, además, a partir de PHP 4.1.0, existe un conjunto adicional de arrays asociativos predefinidos (cuyos nombres comienzan por $_), que contienen variables del servidor web, del entorno, entradas del usuario, ficheros subidos al servidor, cookies y variables de sesión. Estos nuevos arrays se comportan de modo especial porque son automáticamente globales, PHP las denomina "autoglobales" ó "superglobales", y son conocidas en cualquier contexto del script, es decir, no tienen que declararse como globales dentro de las funciones o métodos.

Los arrays predefinidos del tipo ($HTTP_*_VARS) existentes antes de los nuevos mencionados ($_) también siguen disponibles. Para acceder a ellos en funciones o métodos, debemos declararlos con el modificador 'global'. Si estamos usando las últimas versiones de PHP, se recomienda utilizar las nuevas matrices autoglobales para recuperar la información, aunque las antiguas siguen disponibles por compatibilidad con versiones anteriores.

NOTA: Volvemos a recordar que, a partir de PHP 4.2.0, el valor por defecto de la directiva PHP 'register_globals' tiene el valor *off* (desactivada). Éste es un cambio muy importante en PHP porque esta configuración afecta al conjunto de variables predefinidas disponibles en el sistema y a la forma de acceder a ellas. Por ejemplo, para obtener el valor de SCRIPT_NAME, que contiene la ruta del script actual, deberemos usar $_SERVER['SCRIPT_NAME'] (acceder a la variable a través de la matriz predefinida $_SERVER) en vez de $SCRIPT_NAME, como variable predefinida individual disponible directamente. Esto ocurre con cada variable de servidor, de sesión, de entrada del usuario a través de formularios o enlaces y con las variables de entorno, en lugar de estar disponibles las variables tal cual directamente, deberemos acceder a ellas usando su nombre como clave en el array asociativo autoglobal correspondiente ($_SERVER, $_POST, $_GET, $_FILES, $_COOKIE, $_REQUEST, $_SESSION o $GLOBALS).

En los siguientes apartados veremos cada tipo de variables y los arrays asociativos predefinidos disponibles para acceder a ellas.

5.2. VARIABLES DE SERVIDOR

Estas variables contienen información del servidor, como rutas, cabeceras HTTP, datos del protocolo de comunicación entre el cliente y el servidor, como direcciones IP, puertos, etc.

Podemos acceder a ellas directamente, como variables simples, si la directiva '**register_globals**' en el fichero **php.ini** está activada. Todas estas variables individuales no son globales, por lo que debemos declararlas como globales en funciones o métodos que hagan uso de ellas. Además, a partir de PHP 4.1.0 está disponible el array asociativo predefinido $_SERVER y en versiones anteriores podemos usar el también array asociativo $HTTP_SERVER_VARS. Ambos arrays contienen en principio la misma información.

$_SERVER pertenece al grupo de matrices superglobales mencionado antes, por lo que está disponible en todos los contextos a lo largo de un script, sin tener que declararse como global, podemos acceder a ella dentro de funciones o métodos, mientras que, en el caso del antiguo array $HTTP_SERVER_VARS es necesario declararlo como global dentro de funciones o métodos para poder acceder a él.

Para acceder a la información del servidor a través de estos arrays, usaremos como clave el nombre de la variable, por ejemplo, la variable de servidor $PHP_SELF, disponible si '**register_globals**' está activada, tiene su equivalente en el array $_SERVER como $_SERVER['PHP_SELF'], y en el array $HTTP_SERVER_VARS como $HTTP_SERVER_VARS['PHP_SELF'].

Veamos los principales nombres de variables de servidor:

- 'PHP_SELF': Se refiere al nombre de archivo del script PHP que está ejecutándose actualmente, relativo a la raíz de documentos. Por ejemplo, $_SERVER['PHP_SELF'] en un script en la dirección http://localhost/ejemplos/ejemplo_sesiones1.php daría como valor de esta variable "/ejemplos/ejemplos_sesiones1.php".

- 'GATEWAY_INTERFACE': Indica la revisión de la especificación CGI que está usando el servidor; por ejemplo, 'CGI/1.1'.

- 'SERVER_NAME': Contiene el nombre del servidor bajo el que está siendo ejecutado el script o página PHP actual.

- ‘SERVER_SOFTWARE’: Contiene la cadena de identificación del servidor, con el software principal que tiene instalado.

- ‘SERVER_PROTOCOL’: Es el nombre y revisión del protocolo de información mediante el cual fue solicitada la página.

- ‘REQUEST_METHOD’: Contiene el método de petición usado para acceder a la página, como GET o POST.

- ‘QUERY_STRING’: Contiene la cadena de consulta de la página, si la tenía.

- ‘DOCUMENT_ROOT’: Esta variable indica el directorio raíz de documentos bajo el que está siendo ejecutado el script actual.

- ‘HTTP_ACCEPT_CHARSET’: Indica el juego de caracteres de la petición actual, como por ejemplo 'iso-8859-1' o 'utf-8'.

- ‘HTTP_ACCEPT_LANGUAGE’: Indica el idioma de la petición actual, por ejemplo: 'en'.

- ‘HTTP_REFERER’: Contiene la dirección de la página previa (si existe) desde la que se llegó a la página actual.

- ‘HTTP_USER_AGENT’: Indica el navegador del cliente, podemos usar esta información para personalizar la salida de las páginas.

- ‘REMOTE_ADDR’: Es la dirección IP desde donde el cliente ha solicitado la página actual.

- ‘REMOTE_HOST’: Es el nombre de la máquina desde donde el cliente ha solicitado la página actual.

- ‘REMOTE_PORT’: Es el puerto usado en la máquina del cliente para comunicarse con el servidor web.

- ‘SCRIPT_FILENAME’: Contiene la ruta absoluta del nombre del script actual.

- ‘SERVER_PORT’: Es el puerto en el equipo servidor usado para comunicarse con el cliente. Por ejemplo, en la configuración predeterminada con el protocolo HTTP este valor es el ‘80’.

- ‘SCRIPT_NAME’: Contiene la ruta del script actual.

- ‘REQUEST_URI’: Contiene la URI indicada para acceder a la página actual.

5.3. COOKIES HTTP

Una cookie es información que el servidor web le solicita a la aplicación cliente (navegador) que guarde en su disco duro para recuperarla con posterioridad. Por ejemplo, el método predeterminado para mantener las sesiones de cada cliente se basa en las cookies que el servidor envía a cada usuario conectado. La información que se guarda en las cookies la define el servidor, y está formada por una cadena de texto.

El array asociativo disponible para consultar las cookies que tengamos en cada momento era $HTTP_COOKIE_VARS (como el resto de arrays del tipo $HTTP_*_VARS, debe declararse como global para ser usado en funciones y métodos), y a partir de la versión PHP 4.1.0. también está disponible la matriz autoglobal $_COOKIE.

Para recuperar la información que contienen podemos usar como índice del array el nombre de la cookie y su contenido es el valor correspondiente, por ejemplo, para recuperar el valor de una cookie llamada ‘id_curso’, podríamos hacerlo a través de $HTTP_COOKIE_VARS[‘id_curso’] o de $_COOKIE[‘id_curso’].

PHP dispone de la función setcookie ( ) para enviar una cookie al cliente, su sintaxis es:

int setcookie (string nombre_cookie, string valor, int caducidad, string ruta, string dominio, int seguro);

Las cookies deben enviarse antes de mandar cualquier otra información de cabecera y debemos situar las llamadas a esta función antes de cualquier salida HTML.

Todos los parámetros, excepto el nombre de la cookie, son opcionales. Los parámetros son:

- Nombre_cookie: es el nombre con el que identificaremos la cookie.

- Valor: es el contenido dado a la cookie, es una cadena.

- Caducidad: indica el periodo de tiempo de permanencia de la cookie. Si se omite se asume que este tiempo es el que dure la sesión de trabajo del navegador. Podemos usar la función time( ) + duración para establecer este parámetro. La función time( ) devuelve la hora actual como el número de segundos transcurridos desde las 00:00:00 del 1 de enero de 1970.

- Dominio: es el nombre de dominio (que debe tener al menos dos puntos) para el cual será válida la cookie. Esto significa que el navegador devolverá la cookie a cualquier equipo perteneciente a ese dominio. Si no especificamos este parámetro, se asume que la cookie sólo es válida para el equipo que la envió.

- Ruta: indica el directorio raíz de las páginas a las que se les devolverá la cookie. Podemos utilizar este parámetro para delimitar las páginas que pueden ver la cookie.

- Seguro: Indica que la cookie sólo debe transmitirse a través de un canal seguro, sobre una conexión segura HTTPS.

Podemos usar esta función especificando sólo el parámetro nombre_cookie para borrarla del cliente remoto.
Para saltar los parámetros de tipo cadena pasando una cadena de texto vacía (“”) y los parámetros de tipo entero pasando un cero (0).

El siguiente ejemplo envía una cookie llamada ‘id’, cuyo valor es un identificador único y caduca dentro de una hora. El identificador único lo formamos a partir de la llamada a una función que obtiene valores aleatorios, rand( ), y el valor de fecha correspondiente al momento actual:

```
<?php
        srand(time( )*10000);  // establece la semilla del generador de números aleatorios
        $valor_id=rand( ).time( );
        setcookie(“id”, $valor_id, time( ) + 3600 );
?>
```

5.4. VARIABLES HTTP ENVIADAS POR EL MÉTODO GET

El array asociativo disponible para consultar las variables pasadas a nuestra página por el método GET, es $HTTP_GET_VARS y a partir de PHP 4.1.0., también está disponible la matriz autoglobal $_GET.
Por ejemplo, para acceder a la variable 'id_usuario' pasada en este enlace, http://miservidor.com/ejemplo?id_usuario=13, podemos hacerlo como $HTTP_GET_VARS['id_usuario'] o como $_GET['id_usuario'], y si la directiva **'register_globals'** está activada, también estará disponible la variable individual $id_usuario.

En el tema 3 vimos un ejemplo sobre una página PHP que presentaba un enlace a otra página a la que le pasaba los valores de una fecha para que los mostrara. La versión de este ejemplo usando notación de arrays asociativos ($HTTP_GET_VARS o $_GET) en lugar de variables individuales, quedaría de esta forma:

pagina3.php

```
<html>
        <head><title>Ej  de página PHP que pasa varios datos a otra mediante un enlace</title></head>
        <body>
            <h4><a  href="muestraFecha.php?dia=13&mes=6&año=2004&modo=mostrar"> Enlace  que
            pasa varias variables a la página muestraFecha.php </a>
        </h4>
        </body>
</html>
```

Usando la matriz $HTTP_GET_VARS:

muestraFecha.php

```
<html>
        <head><title>Ej  de página PHP que muestra las variables pasadas</title></head>
        <body>
                <?
                    if ($HTTP_GET_VARS['modo']=="mostrar") {
                            echo "<h4>Fecha pasada: ". $HTTP_GET_VARS['dia'] . "/" .
                            $HTTP_GET_VARS['mes'] . "/" . $HTTP_GET_VARS['año'] . "</h4>";
```

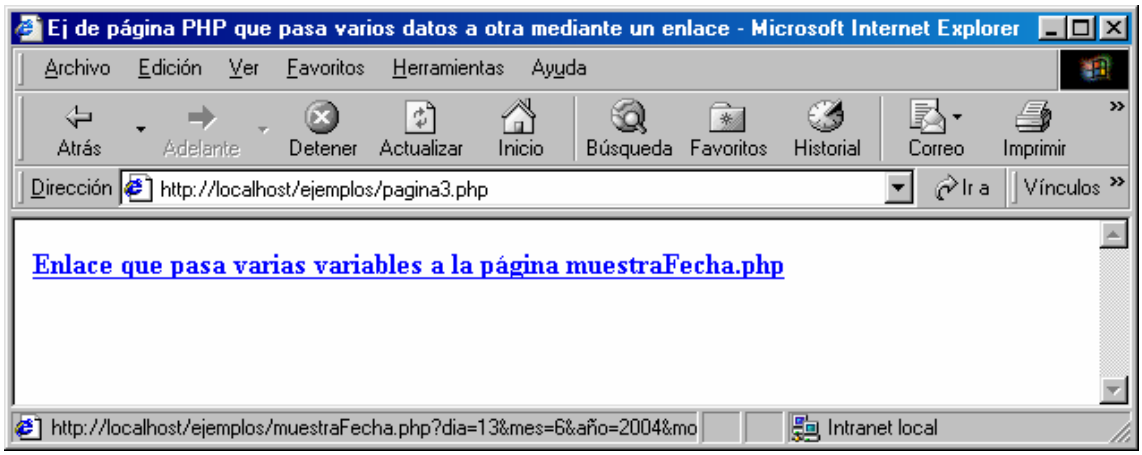
```
                }
            ?>
        </body>
</html>
```

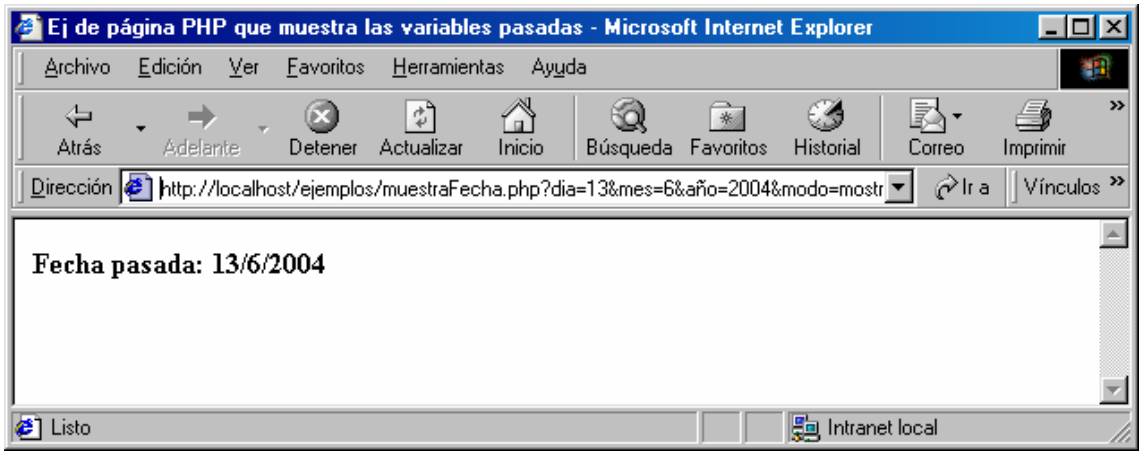
Usando la matriz $_GET:

muestraFecha.php

```
<html>
    <head><title>Ej  de página PHP que muestra las variables pasadas</title></head>
    <body>
        <?
            if (($_GET['modo']=="mostrar") {
                echo "<h4>Fecha pasada: ".$_GET['dia'] ."/" . $_GET['mes'] . "/" .
                $_GET['año']."</h4>";
            }
        ?>
    </body>
</html>
```

El resultado en el navegador es similar a la primera versión de este ejemplo que vimos en el capítulo 3:





5.5. VARIABLES HTTP ENVIADAS POR EL MÉTODO POST

Para consultar las variables pasadas a nuestra página por el método POST, tenemos el array asociativo $HTTP_POST_VARS y a partir de PHP 4.1.0., también está disponible la matriz autoglobal $_POST.
Por ejemplo, para acceder a la variable 'curso' pasada a nuestra página a través de un formulario por el método POST, podemos hacerlo como $HTTP_POST_VARS['curso'] o como $_POST['curso'], y si la directiva **'register_globals'** está activada, también estará disponible la variable individual $curso.

En el capítulo 3 vimos un ejemplo sobre una página PHP que presentaba un formulario con un campo para obtener un número y además tenía un campo oculto con otro valor. La otra página PHP destinataria del formulario, enviado por método POST, comparaba los dos valores que le llegaban (el valor introducido y el valor del campo oculto) y mostraba un mensaje para indicar si eran iguales o no. Utilizando las matrices asociativas $HTTP_POST_VARS o $_POST, nuestro ejemplo quedaría:

ejemplo_formulario3.php

```
<html>
        <head><title>Ejemplo de página PHP con formulario</title></head>
        <body>
        <?
                $numero_oculto="13";
         ?>
        <h2>Formulario:</h2>
        <form action="ejemplo_3.php" method="post">
          Introduzca un número: <br><input type="text" name="numero">
```

```
        <input type="hidden" name="variable_oculta" value="<?  echo $numero_oculto;  ?>"  > <br>
        <input type="submit"  value="Probar suerte"  >
    </form>
    </body>
</html>
```

Usando la matriz $HTTP_POST_VARS :

ejemplo_3.php

```
<html>
    <head><title>Ejemplo de página PHP que recibe datos de un formulario</title></head>
    <body>
      <h2>Contenido de las variables del formulario:</h2>
      <?
            echo "La variable \$numero contiene: " . $HTTP_POST_VARS['numero'] . "<br>";
            echo "La variable oculta \$variable_oculta contiene: " .
            $HTTP_POST_VARS['variable_oculta'] . "<br>";
            if  ($HTTP_POST_VARS['numero']==$HTTP_POST_VARS['variable_oculta'] )  {
                  echo "Ha acertado el valor oculto. <br>";
            }else  {
                  echo "No ha acertado el valor oculto. <br>";
            }
       ?>
    </body>
</html>
```

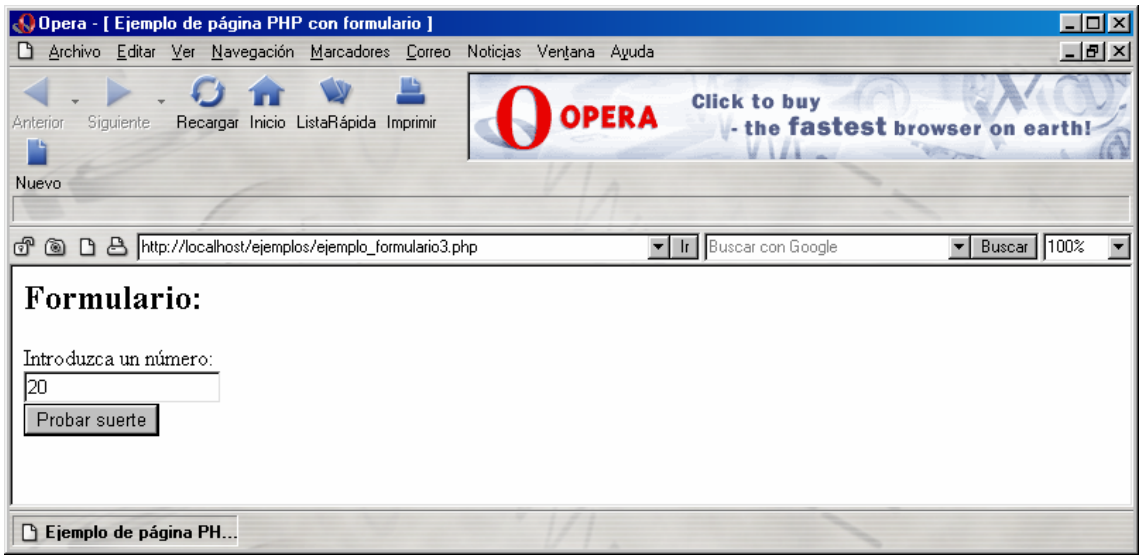
Usando la matriz $_POST :

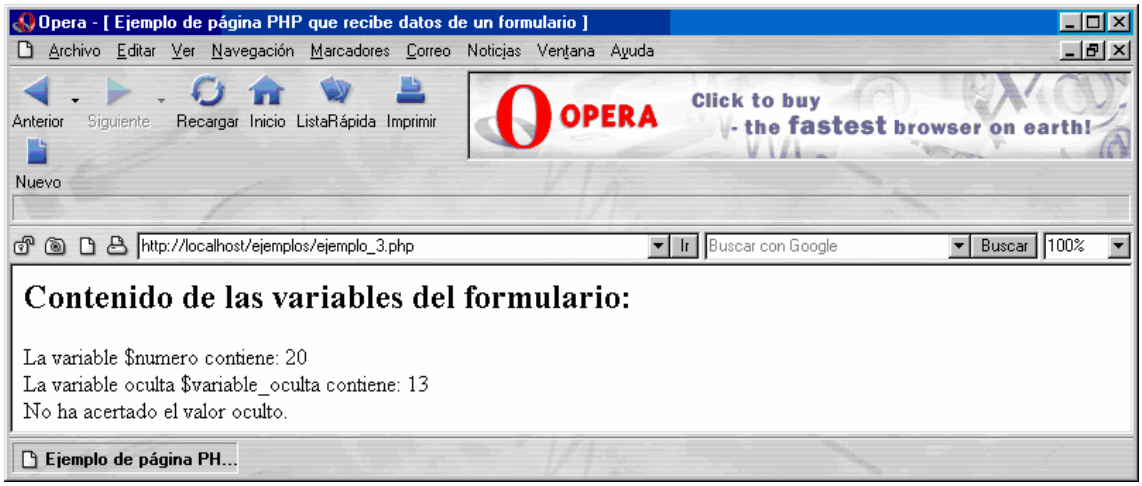
ejemplo_3.php

```
<html>
    <head><title>Ejemplo de página PHP que recibe datos de un formulario</title></head>
    <body>
      <h2>Contenido de las variables del formulario:</h2>
```

```
<?
        echo "La variable \$numero contiene: " . $_POST['numero'] . "<br>";
        echo "La variable oculta \$variable_oculta contiene: " . $_POST['variable_oculta'] .
"<br>";
        if ($_POST['numero']==$_POST['variable_oculta'] ) {
            echo "Ha acertado el valor oculto. <br>";
        }else {
            echo "No ha acertado el valor oculto. <br>";
        }
    ?>
    </body>
</html>
```

Al ejecutar estas páginas:





5.6. VARIABLES DE PETICIÓN

Existe otra matriz autoglobal que incluye los contenidos de los arrays $_GET, $_POST, y $_COOKIE, la matriz $_REQUEST. Como es autoglobal, es conocida en todos los contextos del script, y no tenemos que declararla como global dentro de funciones o métodos.

Accedemos a su contenido usando la notación de arrays asociativos que hemos visto, donde el nombre de la variable es la clave del array para acceder a su contenido, por ejemplo, podemos obtener el contenido de una variable de nombre 'usuario' pasada a través de un formulario por el método POST a la página actual, como $_REQUEST['usuario']. Al igual que el resto de matrices que estamos viendo, si la directiva **'register_globals'** está activada, entonces estas variables también estarán disponibles en el contexto global del script como variables individuales, y la variable anterior sería conocida también como $usuario.

5.7. VARIABLES HTTP DE SUBIDA DE ARCHIVOS AL SERVIDOR

Para consultar las variables relacionadas con los ficheros subidos al servidor (con formularios <form enctype="multipart/form-data" method="post">, está disponible el array asociativo $HTTP_FILES_VARS y a partir de PHP 4.1.0., también está disponible la matriz autoglobal $_FILES, y como en los anteriores casos, si la directiva **'register_globals'** está activada, también existirán las variables individuales que vimos en los apartados 3.5 y 3.6 que informaban sobre el nombre temporal del fichero subido en el servidor, su nombre en el cliente, tamaño y tipo de fichero. La forma de acceder a los arrays $HTTP_FILES_VARS y $_FILES es similar al resto de matrices que estamos viendo, usando la notación de arrays asociativos.

5.8. VARIABLES DE SESSIÓN

Ya vimos en el capítulo anterior, el tratamiento de las sesiones en PHP, y los arrays implicados, $HTTP_SESSION_VARS y $_SESSION (a partir de PHP 4.1.0.). Además, si la directiva de configuración 'register_globals' está activada, las variables registradas como variables de sesión estarán también disponibles como variables individuales en la página.

5.9. VARIABLES GLOBALES

PHP proporciona una matriz asociativa autoglobal con las referencias a todas las variables que están definidas actualmente en el contexto global del script. Los nombres de las variables son las claves de la matriz. Esta matriz es $GLOBALS y ha estado disponible desde PHP 3.0.0. Es otra forma de acceder a cada una de las variables globales de nuetra página.

Por ejemplo, para conocer el valor de una variable global en nuestra página llamada $usuario a través de este array, lo haríamos como $_GLOBALS['usuario'].

RECUERDE

- PHP proporciona una gran cantidad de variables predefinidas a cualquier script que se ejecute, como variables del servidor web, del entorno, entradas del usuario, ficheros subidos al servidor, cookies y variables de sesión. Estas variables globales estarán disponibles como variables individuales si la directiva de configuración '**register_globals**' (se encuentra en el fichero **php.ini**) está activada.

- Debemos tener en cuenta que, a partir de **PHP 4.2.0**, el valor por defecto de '**register_globals**' es *off* (desactivada), por lo que las variables globales no estarán disponibles directamente como variables individuales, sino que deberemos obtenerlas a partir de las matrices predefinidas disponibles o cambiar el valor de esta directiva si queremos acceder a las variables globales como variables individuales.

- PHP también dispone de las matrices asociativas $HTTP_SERVER_VARS, $HTTP_POST_VARS, $HTTP_GET_VARS, $HTTP_FILES_VARS, $HTTP_COOKIE- _VARS y $HTTP_SESSION_VARS. Estos arrays **no** son **autoglobales**.

- A partir de la versión **4.1.0 de PHP**, también existen las nuevas matrices **autoglobales** $_SERVER, $_POST, $_GET, $_FILES, $_COOKIE, $_REQUEST y $_SESSION. La matriz $GLOBALS está disponible desde PHP 3.0.0. y también es **autoglobal**.

- Para acceder a la información de todas estas matrices asociativas, escribiremos como clave del array el nombre de la variable que buscamos.

Tema 6

Acceso a bases de datos

6.1. INTRODUCCIÓN

PHP proporciona una gran número de librerías de funciones para acceder a multitud de bases de datos de forma nativa o mediante ODBC. La generación de contenidos dinámicos normalmente se basa en la información contenida en bases de datos que se consultan en función de los datos solicitados por los usuarios a través de las páginas PHP de nuestra aplicación web.

Podemos consultar la referencia del lenguaje en su sitio web oficial, www.php.net, para una descripción detallada de cada una de ellas, donde veremos que PHP puede trabajar con PostgreSQL, MySQL, mSql, Informix, SQL Server, Oracle, Sybase, Interbase, Solid, ODBC, etc. Como ejemplo de uso de una de estas librerías, en este capítulo veremos las funciones PHP de acceso a MySQL, por ser una gestor de base de datos ampliamente utilizado junto con PHP. Existen versiones de MySQL tanto para sistemas Linux/Unix como para Windows. Debemos tener instalado MySQL en nuestro sistema y además, asegurarnos de que nuestra configuración de Apache y PHP soporta el acceso a bases de datos MySQL (por ejemplo, si en un sistema Linux/Unix no hemos compilado nuestras fuentes con soporte para MySQL, no tendremos disponibles las funciones PHP de acceso a este gestor de base de datos). Esto es aplicable a cualquier gestor de base de datos, es decir, para que podamos acceder desde PHP a una base de datos concreta, debemos asegurarnos de que dicha base de datos esté instalada correctamente y de que nuestra configuración en Apache y PHP nos permite hacerlo, ya que no todas las distribuciones compiladas para Windows traen soporte para todas las bases de datos, y en sistemas Linux/Unix donde compilemos los fuentes de Apache y PHP debemos incorporar el soporte que necesitemos.

En el fichero de configuración de PHP, php.ini, existen diversas directivas que establecen los valores predeterminados para ciertas operaciones con las bases de datos. Podemos consultarlos o modificarlos para adaptarlos a nuestras preferencias.

Es necesario tener conocimientos de SQL para entender los ejemplos de este capítulo, ya que usaremos este lenguaje para realizar operaciones sobre la base de datos, como consultas, inserciones, modificaciones o eliminaciones de datos.

La filosofía del trabajo con bases de datos desde PHP es prácticamente igual para todos los gestores de bases de datos. Normalmente necesitaremos una función para conectarnos a la base de datos, otra función para realizar consultas, otra para acceder a los resultados de la consulta,... y además suelen estar disponibles otras funciones auxiliares que nos informan del número de resultados obtenidos, filas afectadas, número de campos obtenidos en la consulta, etc. Las funciones de acceso a una base de datos suelen comenzar con un prefijo identificativo de la base de datos en cuestión, por ejemplo, las

funciones de acceso a MySQL comienzan por mysql_, las funciones de Informix por ifx_, las funciones de ODBC por odbc_, las de Oracle por oci_ o por ora_, las de PostgreSQL por pg_, etc.

6.2. CONECTAR A MYSQL DESDE PHP

La conexión se realiza en dos pasos, primero se establece la conexión con el motor de la base de datos indicando el equipo que alberga la base de datos, el usuario con el que nos conectamos y su clave, y después se selecciona una base de datos:

```
$enlace = mysql_connect($servidor, $usuario, $clave);
$id_conexion = mysql_select_db($base_datos, $enlace);
```

Es habitual utilizar la función die( ) al intentar hacer la conexión para terminar la ejecución del script PHP si ha habido algún problema y no hemos podido conectarnos:

```
$id_conexion = mysql_select_db($base_datos, $enlace) or
        die("No se puede conectar a la base de datos");
```

Con la función mysql_connect( ) establecemos una conexión no persistente. También existen las denominadas conexiones persistentes, que aumentan de algún modo el rendimiento de las conexiones porque una conexión persistente no termina cuando acaba el programa, y una posible posterior solicitud de conexión verifica si hay alguna abierta de las mismas características para utilizarla antes de abrir otra nueva. PHP dispone también de la función mysql_pconnect( ), que utiliza los mismos argumentos que mysql_connect( ), y que realiza una conexión persistente.

6.3. REALIZAR UNA CONSULTA A MYSQL DESDE PHP

La función para realizar una consulta con MySQL es mysql_query( ), le pasamos una cadena con la consulta a realizar y la conexión a la base de datos. Devuelve un descriptor con los resultados de la consulta. Si la operación falla $resultado tendrá un valor nulo equivalente a falso, por lo que aquí también podemos usar la función die( ) para terminar el script si una consulta no devuelve resultados.

```
$consulta = "Select * from pedidos where id_pedido>20";
$resultado = mysql_query($consulta, $id_conexion) or
        die("No hay datos");
```

La función mysql_query( ) también se usa para realizar otras operaciones sobre la base de datos (INSERT, UPDATE o DELETE). En estos casos la función devuelve TRUE o FALSE para indicar si la operación de inserción, modificación o borrado ha tenido éxito. Para la operación SELECT hemos visto que devuelve un nuevo identificador de resultado si la consulta devuelve datos o FALSE en caso contrario.

6.4. RECUPERAR LOS DATOS DE UNA CONSULTA A MYSQL DESDE PHP

Una vez que hemos realizado una consulta y obtenido un descriptor con los datos como en el ejemplo anterior, podemos recuperar los valores por filas o bien individualmente.

Para recuperar los datos individualmente tenemos la función mysql_result( ), a la que le pasamos el descriptor de la consulta obtenido con la función mysql_query( ), el número de fila del resultado que queremos recuperar (debemos tener en cuenta que este número comienza por cero) y el campo del resultado (que puede ser el número de campo o el nombre del campo, y el número de campo también comienza por cero):

```
$enlace = mysql_connect($servidor, $usuario, $clave);
$id_conexion = mysql_ select_db($base_datos, $enlace) or
        die("No se puede conectar a la base de datos");

$consulta = "Select * from pedidos where id_pedido>20";
$resultado = mysql_query($consulta, $id_conexion) or
        die("No hay datos");

$valor = mysql_result($resultado, $numero_fila, $campo);
```

También podemos recuperar los datos de la consulta por filas como arrays, es decir, un array por cada fila, con la función mysql_fetch_array( ), a la que le pasamos el descriptor de la consulta obtenido con la función mysql_query( ). Las filas se recuperan secuencialmente y existe un puntero que se actualiza automáticamente cada vez que ejecutamos la función mysql_fetch_array( ):

```
$enlace = mysql_connect($servidor, $usuario, $clave);
$id_conexion = mysql_ select_db($base_datos, $enlace) or
        die("No se puede conectar a la base de datos");

$consulta = "Select * from pedidos where id_pedido>20";
$resultado = mysql_query($consulta, $id_conexion) or
          die("No hay datos");

$fila = mysql_fetch_array($resultado);
```

Una vez que tenemos el array con los campos de la fila actual podemos recorrerlo usando la notación de arrays habitual, indicando los nombres de campo como índices.

También existe la función mysql_data_seek( ) para situar el puntero de lectura en una posición concreta del conjunto de resultados de la consulta. Recibe el descriptor de la consulta y el número de fila en que deseamos situar el puntero:

```
booleano mysql_data_seek($resultado, $numero_fila);
```

6.5. OBTENER EL NÚMERO DE FILAS OBTENIDAS EN UNA CONSULTA A MYSQL DESDE PHP

Para ello tenemos la función mysql_num_rows( ), que recibe el descriptor de la consulta y devuelve el número de filas del conjunto de resultados obtenido:

```
$numero_filas = mysql_num_rows($resultado);
```

6.6. OBTENER EL NÚMERO DE CAMPOS OBTENIDOS EN UNA CONSULTA A MYSQL DESDE PHP

La función mysql_num_fields( ), que recibe el descriptor de la consulta, devuelve el número de campos o columnas del conjunto de resultados obtenido:

```
$numero_campos = mysql_num_fields($resultado);
```

6.7. OBTENER EL NOMBRE DE CAMPO EN UNA CONSULTA A MYSQL DESDE PHP

Para obtener el nombre de un campo de un conjunto de resultados de una consulta disponemos de la función mysql_field_name( ), que recibe el descriptor de la consulta (obtenido con la función anterior mysql_query( )) y el número de campo buscado y devuelve una cadena con el nombre del campo:

```
$nombre_campo = mysql_field_name($resultado,$numero_campo);
```

6.8. EJEMPLO DE CONSULTA DE DATOS

Veamos un ejemplo de una página PHP que se conecta a una base de datos MySQL llamada 'bd_almacen' con el usuario 'nobody', realiza una consulta a la tabla de clientes y recorre las filas obtenidas para mostrar los datos:

ejemplo_mysql_consulta.php

```
<html>
<body>
<h2>Datos de los clientes: </h2>
<?php
    // nos conectamos a la base de datos
    $enlace = mysql_connect("localhost", "nobody", "nobody");
    $id_conexión = mysql_select_db("bd_almacen", $enlace);
    $consulta = "SELECT nombre, apellidos, direccion FROM clientes";
    // realizamos la consulta
    $resultado = mysql_query($consulta, $id_conexion);
```

```
 // presentamos los datos
if ($fila = mysql_fetch_array($resultado)){
        echo "<table border = '1'> \n";
        echo "<tr> \n";
        echo "<td><b>Nombre</b></td> \n";
        echo "<td><b>Apellidos</b></td> \n";
        echo "<td><b>Dirección</b></td> \n";
        echo "<td><b>Código_cliente</b></td> \n";
        echo "</tr> \n";
        do {
                echo "<tr> \n";
                echo "<td>".$fila["nombre"]."</td> \n";
                echo "<td>".$fila["apellidos"]."</td>\n";
                echo "<td>".$fila["direccion"]."</td>\n";
                echo "<td>".$fila["codigo"]."</td>\n";
                echo "</tr> \n";
        } while ($fila = mysql_fetch_array($resultado));
        echo "</table> \n";
 } else {
        echo "No hay datos de clientes";
 }
?>
</body>
</html>
```

6.9. EJEMPLO DE INSERCIÓN DE DATOS

En el siguiente ejemplo se presenta una página web con un formulario que contiene los campos para introducir los datos del nuevo cliente que vamos a introducir en la base de datos anterior 'bd_almacen'. Una vez recogidos y enviados los datos a la segunda página PHP, ejemplo_mysql_insercion.php, se forma la sentencia INSERT correspondiente y se utiliza la función mysql_query( ) para añadir el nuevo registro a la tabla clientes.

ejemplo_2.php

```
<html>
<body>
	<h2>Introduzca los datos del nuevo cliente: </h2>
	<form method="post" action="ejemplo_mysql_insercion.php">
		Nombre: <input type="text" name="nombre"><br>
		Apellidos: <input type="text" name="apellidos"><br>
		Dirección: <input type="text" name="direccion"><br>
		Código_cliente: <input type="text" name="codigo"><br>
		<input type="submit" name="enviar" value="Enviar">
	</form>
</body>
</html>
```

ejemplo_mysql_insercion.php

```
<html>
<body>
<?php
	if (!empty($codigo))  {
		// procesamos los datos recibidos del formulario anterior
		// para formar la sentencia INSERT y añadir el nuevo cliente
		$enlace = mysql_connect("localhost", "nobody", "nobody");
		$id_conexión = mysql_select_db("bd_almacen", $enlace);
		$sql = "INSERT INTO clientes (nombre, apellidos, direccion, codigo) ";
		$sql .= "VALUES ('$nombre', '$apellidos', '$direccion', '$codigo')";
		// Ejecutamos la sentencia de inserción
		mysql_query($sql, $id_conexion);
		echo "El nuevo cliente ha sido introducido. \n";
	} else {
		echo "Debe indicar un código de cliente. \n";
	}
?>
</body>
</html>
```

6.10. EJEMPLO DE ACTUALIZACIÓN DE DATOS

En este ejemplo se presenta una página web con un formulario para introducir los nuevos datos del código de cliente especificado. Una vez recogidos y enviados los datos a la segunda página PHP, ejemplo_mysql_actualizacion.php, se forma la sentencia UPDATE correspondiente y se utiliza la función mysql_query( ) para modificar los datos de ese cliente en la tabla clientes.

ejemplo_3.php

```
<html>
<body>
    <h2>Introduzca los nuevos datos para el código de cliente indicado: </h2>
    <form method="post" action="ejemplo_mysql_actualizacion.php">
            Nombre: <input type="text" name="nombre"><br>
            Apellidos: <input type="text" name="apellidos"><br>
            Dirección: <input type="text" name="direccion"><br>
            Código_cliente: <input type="text" name="codigo"><br>
            <input type="submit" name="enviar" value="Enviar">
    </form>
</body>
</html>
```

ejemplo_mysql_actualizacion.php

```
<html>
<body>
<?php
        if (!empty($codigo))  {
            // procesamos los datos recibidos del formulario anterior
            // para formar la sentencia UPDATE y actualizar los datos del cliente indicado
            $enlace = mysql_connect("localhost", "nobody", "nobody");
            $id_conexión = mysql_select_db("bd_almacen", $enlace);
            $sql = "UPDATE clientes SET nombre='$nombre', apellidos='$apellidos'";
            $sql .= ", direccion='$direccion' WHERE codigo='$codigo'";
```

```
            // Ejecutamos la sentencia de actualización
            mysql_query($sql, $id_conexion);
            echo "Los datos del cliente han sido actualizados. \n";
        } else {
            echo "Debe indicar un código de cliente. \n";
        }
    ?>
    </body>
</html>
```

6.11. EJEMPLO DE ELIMINACIÓN DE DATOS

Ahora veremos un ejemplo de cómo eliminar un registro de una tabla. Presentamos un formulario web para recoger el código del cliente que deseamos eliminar y en la segunda página PHP, ejemplo_mysql_borrado.php, se forma la sentencia DELETE correspondiente y se utiliza la función mysql_query( ) para eliminar los datos de ese cliente en la tabla clientes.

ejemplo_4.php

```
<html>
<body>
    <h2>Introduzca el código del cliente que desea eliminar: </h2>
    <form method="post" action="ejemplo_mysql_borrado.php">
        Código_cliente: <input type="text" name="codigo"><br>
        <input type="submit" name="enviar" value="Enviar">
    </form>
</body>
</html>
```

ejemplo_mysql_borrado.php

```
<html>
<body>
<?php
        if (!empty($codigo))  {
            // formamos la sentencia DELETE para eliminar el cliente indicado
            $enlace = mysql_connect("localhost", "nobody", "nobody");
            $id_conexión = mysql_select_db("bd_almacen", $enlace);
            $sql = "DELETE FROM clientes WHERE codigo='$codigo'";
            // Ejecutamos la sentencia de eliminación
            mysql_query($sql, $id_conexion);
            echo "El cliente ha sido eliminado. \n";
        } else {
            echo "Debe indicar un código de cliente. \n";
        }
?>
</body>
</html>
```

NOTA: Los ejemplos anteriores son ilustrativos de la operación en cuestión que se está realizando en cada caso. En una aplicación en producción deberíamos hacer más comprobaciones como por ejemplo, que exista el código del cliente cuyos datos vamos a actualizar, y además, para que sea posible realizar operaciones de consulta, inserción, modificación o borrado de datos, el usuario con el que nos conectamos a la base de datos MySQL debe tener los permisos necesarios.

6.12. OTRAS FUNCIONES PHP DE ACCESO A MYSQL

Veamos otras funciones PHP comunes de acceso a MySQL. El resto de gestores de bases de datos disponen de funciones con características similares para realizar operaciones análogas. Para una descripción detallada de todas ellas podemos consultar el manual de referencia de PHP:

6.12.1. Función mysql_affected_rows

```
int mysql_affected_rows([int $id_conexión]);
```

Devuelve el número de filas afectadas por la última operación UPDATE o DELETE asociada al identificador de conexión especificado. Si no se indica identificador de conexión, se asume el de la última conexión abierta.
Para recuperar el número de filas devuelto por una instrucción SELECT debemos usar mysql_num_rows( ).

6.12.2. Función mysql_change_user

```
int mysql_change_user(string $usuario, string $clave [, string $base_de_datos [, int
$id_conexion]]);
```

Cambia el usuario registrado en la conexión activa, o si se especifica, en la conexión determinada por el identificador de conexión. Si se indica la base de datos, ésta será la base de datos por defecto después del cambio de usuario.

6.12.3. Función mysql_close

```
int mysql_close([int $id_conexión]);
```

Cierra la conexión a la base de datos MySQL asociada al identificador de conexión indicado. Si no se especifica identificador de conexión, se toma el de la última conexión abierta. Devuelve TRUE si la conexión se cierra correctamente y FALSE en caso de error.

6.12.4. Función mysql_create_db

```
int mysql_create_db(string $base_de_datos
 [,int $id_conexión]);
```

Crea una base de datos MySQL en el servidor asociado al identificador de conexión especificado.

6.12.5. Función mysql_drop_db

```
int mysql_drop_db(string $base_de_datos
  [,int $id_conexión]);
```

Elimina una base de datos del servidor asociado al identificador de conexión especificado. Devuelve TRUE si la operación se realiza correctamente y FALSE en caso de error.

6.12.6. Función mysql_errno

```
int mysql_errno([int $id_conexión]);
```

Devuelve el número de error asociado a la última operación realizada.

6.12.7. Función mysql_error

```
string mysql_error([int $id_conexión]);
```

Devuelve el mensaje de error asociado a la última operación realizada.

6.12.8. Función mysql_fetch_lengths

```
array mysql_fetch_lengths(int $resultado);
```

Devuelve una matriz que contiene las longitudes de cada campo de la última fila recuperada con mysql_fetch_array( ), o falso en caso de error. El índice del array comienza en cero.

6.12.9. Función mysql_field_type

```
string mysql_field_type(int $resultado, int $numero_campo);
```

Devuelve el tipo del campo indicado.

6.12.10. Función mysql_field_len

```
int mysql_field_len(int $resultado, int $numero_campo);
```

Devuelve la longitud del campo indicado.

RECUERDE

- PHP proporciona una gran número de librerías de funciones para acceder a multitud de bases de datos de forma nativa o mediante ODBC.

- La generación de contenidos dinámicos normalmente se basa en la información contenida en bases de datos que se consultan en función de los datos solicitados por los usuarios a través de las páginas PHP de nuestra aplicación web.

- Podemos consultar la referencia del lenguaje en su sitio web oficial, www.php.net, para una descripción detallada de cada una de las funciones de las librerías disponibles de acceso a bases de datos. PHP puede trabajar con PostgreSQL, MySQL, mSql, Informix, SQL Server, Oracle, Sybase, Interbase, Solid, ODBC, etc.

- Los fundamentos del trabajo con bases de datos desde PHP son prácticamente iguales para todos los gestores de bases de datos. Normalmente necesitaremos una función para conectarnos a la base de datos, otra función para realizar consultas, recuperar los datos, ... y además suelen estar disponibles otras funciones auxiliares e informativas.

- Las funciones de acceso a una base de datos suelen comenzar con un prefijo identificativo de la base de datos en cuestión, por ejemplo, las funciones de acceso a MySQL comienzan por mysql_, las funciones de ODBC por odbc_, las de PostgreSQL por pg_, etc.

- Como ejemplo de uso de una de estas librerías, en este capítulo hemos visto las principales funciones PHP de acceso a MySQL y ejemplos sobre cómo realizar consultas, añadir registros, actualizar o eliminar datos sobre una base de datos MySQL.

A

Apache. Servidor web. Suele usarse en combinación con PHP. Podemos descargarnos las últimas versiones disponibles y su documentación de www.apache.org .

C

Compilador. Aplicación que genera un programa ejecutable a partir de los ficheros que contienen su código fuente.

Cookie. Fichero con información que el servidor web le solicita a la aplicación cliente (navegador) que guarde en su disco duro para recuperarla con posterioridad, como por ejemplo preferencias sobre las páginas visitadas, idioma elegido, etc.

CSS. Cascading Style Sheets, Hojas de Estilo en Cascada. Se utilizan para dar formato a documentos HTML o XML, separando los datos del formato de presentación.

D

DHTML. Dinamic HTML (HTML Dinámico). Se forma de una conjunción de HTML, hojas de estilo en cascada (CSS) y lenguajes de script como JavaScript o VBScript.

H

HTML. HiperText Markup Language. Lenguaje para la creación de páginas Web relacionadas entre sí por hipertexto.

HTTP. HiperText Transport Protocol. Protocolo de comunicaciones utilizado por los navegadores de Internet para el acceso a recursos Web, como ficheros de imágenes, de video, páginas HTML, páginas PHP, páginas ASPX, etc.

I

Intérprete. Aplicación que analiza el código fuente de entrada, lo traduce y lo ejecuta. No genera un fichero ejecutable como lo hace un compilador.

J

JavaScript. Lenguaje de script desarrollado por Netscape y utilizado en los navegadores de Internet.

M

MySql. Base de datos relacional. Se usa ampliamente junto con PHP.

P

PHP. Personal Home Page, Procesador de Hipertexto. Lenguaje de programación con sintaxis similar a los lenguajes C y Perl. Se interpreta por un servidor web Apache bajo sistemas Unix/Linux (también han salido al mercado versiones para sistemas Windows, aunque no siempre podremos utilizar todas sus características bajo este sistema operativo). Las páginas PHP son páginas webs con extensión .php o .phtml (otras extensiones comunes son .php3, .php4, .php5 o .inc) que incluyen código HTML, JavaScript y PHP embebido en ellas, y al ejecutarlas, se genera código HTML dinámicamente.

S

Script. Conjunto de instrucciones ejecutables.

Sesión PHP. Conjunto de información de trabajo y seguimiento de cada cliente conectado a la aplicación PHP, que permite mantener el estado de cada cliente y diferenciar y personalizar las páginas presentadas a cada uno de ellos en función de los datos introducidos y sus acciones. Son necesarias en aplicaciones dinámicas complejas como las de comercio electrónico, plataformas de formación online ...,

SQL. Structured Query Language, Lenguaje de Consulta Estructurado. Es el lenguaje de base de datos relacional estándar. Es un lenguaje declarativo que nos permite crear tablas, vistas, índices, manipular los datos y establecer derechos de acceso de los usuarios y comprobaciones de integridad sobre la base de datos.

X

XML. eXtensible Markup Language, Lenguaje de Marcado Extensible. Se utiliza para la construcción de documentos de datos estructurados multi-propósito.

- Creación de aplicaciones Web con PHP, Prentice Hall.

- Guía Práctica para usuarios PHP, Anaya Multimedia.

- Desarrollo Web con PHP y MySQL. Programación. Anaya Multimedia – Anaya Interactiva.

- Programación con PHP. Anaya Multimedia – Anaya Interactiva.