PHP dinámico

Lo que hemos visto hasta el momento, solo nos ha servido para escribir una serie de scripts PHP y ver los resultados de su ejecución, pero aún no hemos visto de qué forma se puede lograr que **interactúen** el *cliente* y el *servidor*.

Veamos cómo hacerlo.

Envío a través del navegador

La forma más simple de que un **cliente** pueda *enviar* valores a un servidor es incluir esos valores en la propia petición, insertándolos directamente en la barra de direcciones del navegador.

Forma de envío

Deberá insertarse –en la barra de direcciones del navegador– lo siguiente:

*pagina.php***?***n1***=***v1***&***n2***=***v2*

donde

pagina.php será la dirección de la página que contiene el script que ha de procesar los valores transferidos.

? es un carácter obligatorio que indica que detrás de él van a ser insertados nombres de variables y sus valores.

n1, n2, etcétera representan los *nombres* de las variables.

= es el separador de los nombres de las variables y sus valores respectivos.

v1, v2,... simbolizan el valor asignado a cada una de las variables.

& es el símbolo que separa los distintos bloques *variable = valor*.

Los **nombres** de las variables **nunca llevan** el signo **$**.

Los **valores** de las variables –sean números o cadenas– **nunca se escriben entre comillas**.

Se pueden incluir tantos *nombre = valor* como se desee. La única restricción es la longitud máxima permitida por el método **GET** (el utilizado en este caso) que se sitúa en torno a los 2.000 caracteres.

Recepción de datos

Cuando es recibida por el *servidor* la **petición** de un documento con extensión **.php** en la que tras el signo **?** se incluyen una o varias parejas *nombre = valor*, los nombres de las variables y sus valores respectivos se incluyen, de forma automática, en variables predefinidas del tipo:

$HTTP_GET_VARS['*n1***']**

$HTTP_GET_VARS['*n2***']**

en las que *n1*, *n2*, ... coinciden **exactamente** con nombres asignados a cada una de las variables en esa transferencia.

Cada una de esas variables

¿Que valor tiene *register_globals* en tu *php.ini*?

Cuando hablábamos de la configuración de PHP, hacíamos alusión a la directiva register_globals y decíamos que tenía dos opciones de configuración: ON y OFF. Decíamos también que según el valor que tuviera esa configuración cambiaría el modo de afrontar diversos *asuntos*.

Habíamos optado por ponerla como: **register_globals=ON**, pero no está de más el comprobarlo.

Busquemos el fichero **php.ini** que estará en el directorio Windows –salvo que el directorio en que tienes instalado Windows se llame de otra manera– abrámoslo y comprobemos que es cierto que **register_globals=ON**. Si no fuera así, modificaríamos esa *directiva* y guardaríamos los cambios.

¡Cuidado!

Antes de hacer una modificación en **php.ini** ó en **httpd.conf desactiva el servidor** Apache. Cuando hayas acabado con los cambios vuelve a **ponerlo en marcha** (de lo contrario te aparecerá el mensaje: *no se puede encontrar el servidor*) y ya arrancará atendiendo a la nueva configuración. ¡No te olvides nunca de hacerlo así! Te evitarás un montón de sobresaltos.

Tabla de multiplicar

```
<?
/* Escribamos una instrucción, que imprima en pantalla
   el valor de una variable ($a), una "x" que hará funciones de aspa
   en la presentación, otra variable ($b), el signo igual
   y $a*$b que es el producto de ambas variables
   El  simbolo de multiplicar es (*)      */
print ($a." x ".$b." = ".$a*$b);
# Añadamos una bobadilla, para animar... ;-)
print ("<br> ¡¡Ya eres mayorcit@.. Deberías saberte la tabla");
?>
```

ejemplo13.php

Ejecuta este ejemplo y observa lo que aparece. ¿Sólo **x = 0**? ¿y la *otra línea*?

El resultado es lógico. No hemos asignado valores a las variables **$a** y **$b** y por eso no escribió su valor. Sin embargo, a la hora de multiplicar -recuerda que una variable vacía es interpretada como cero a la hora de hacer operaciones– la interpretó como cero y –con muy buen criterio– nos respondió que cero por cero es cero.

Escribamos ahora en la barra de direcciones del navegador la siguiente dirección: http://localhost/cursoPHP/ejemplo13.php?a=21&b=456, (también puedes pulsar en este enlace) y podremos comprobar que **21 x 456 = 9576** .

Hagamos un nuevo *experimento*. Vayamos a **php.ini** y cambiemos la configuración. Pongamos **register_globals= OFF**. ¡No te olvides de apagar el servidor antes de hacer estos cambios!

Una vez hecho el cambio, pulsemos de nuevo en el enlace, y veremos que esta vez no ha recibido los valores de a y b y nos ha dado cero como resultado.

Lo intentaremos con este nuevo script manteniendo **register_globals=OFF**.

```
<?

/* Modifiquemos la instrucción anterior
   y utilicemos las variables predefinidas
   $HTTP_GET_VARS['a'], $HTTP_GET_VARS['b']
   en vez de $a y $b que utilizabamos en el ejemplo anterior */

# Pongamos un comentario de advertencia. Recuerda que <br>
# sirve para insertar un salto de línea en la salida
print ("Este resultado es el que utiliza $HTTP_GET_VAR<br>");

print ($HTTP_GET_VARS['a']." x ".$HTTP_GET_VARS['b']." = ".
                                $HTTP_GET_VARS['a']*$HTTP_GET_VARS
['b']);

/* Ahora trataremos de comprobar que también podemos
    utilizar la superglobal $_GET como $_GET['a'] y $_GET['b']
        con iguales resultados que las anteriores */


# Un comentario para identificar el origen del resultado

print("<br>El resultado siguiente ha sido generado usando $_GET <br>");

print ($_GET['a']." x ".$_GET['b']." = ".$_GET['a']*$_GET['b']);
```

contendrá como valor aquel que hubiera recibido a través de la petición.

Si la *versión* de **PHP** es *superior* a la **4.1.0**, esos mismos valores se incluirán en variables superglobales del tipo **$_GET** de modo que –en el supuesto de que la versión lo soporte– los valores de la petición **también** (esta opción no excluye la anterior) estarían disponibles en:

$_GET['*n1*']

$_GET['*n2*']

Según el modo en que esté configurado el **php.ini** podría caber una *tercera* posibilidad de registro de esos valores.

Si la directiva *register_globals* –en el fichero **php.ini**– está configurada con la opción **ON**, los valores transferidos desde el navegador –además de ser recogidos en las variables anteriores– son asignados a otras variables del tipo: **$n1**, **$n2**, ... cuyos nombres son el resultado de anteponer el símbolo **$** a los nombres de las variables contenidas en la petición.

La elección a la hora de escribir los scripts de uno u otro tipo de variable debe hacerse teniendo en cuenta que:

– Esta última –sin duda la más cómoda– tiene el problema de que *sólo es válida* cuando *register_globals=on* y, además, es la más *insegura* de todas.

– La superglobal **$_GET** tiene una sintaxis más corta que su alternativa y, además, añade como ventaja su condición de **superglobal**, que permite utilizarla en cualquier ámbito sin necesidad de declararla expresamente como **global**. Es la *opción del futuro*. Su único inconveniente es que puede no estar disponible en hostings que aún mantienen versiones antiguas de PHP.

Envío a través de formularios

La interacción *cliente–servidor* que acabamos de ver, resulta incómoda en su uso y no demasiado estética.

Hay una segunda opción –la de uso más frecuente– que es la utilización de formularios.

Los formularios no son ele- mentos propios de PHP –actúan del lado del **cliente**– y su estudio es más propio del ámbito de HTML que de PHP.

En cualquier caso –por si te fuera necesario– aquí a la derecha tienes un formulario, en el que se comenta la sintaxis de sus elementos más comunes.

Interpretación de los datos recibidos a través de formularios

Igual que ocurría en el caso anterior, los datos enviados a través de un formulario son recogidos en diferentes tipos de variables predefinidas, pero ahora se añade una nueva particularidad.

Existe la posibilidad de dos métodos (**method**) de envío: '**GET**' y '**POST**'. En el caso anterior

```
?>
```

ejemplo14.php

Escribamos ahora en la barra de direcciones del navegador la siguiente dirección: http://localhost/cursoPHP/ejemplo14.php?a=21&b=456, (o pulsemos directamente en este enlace) y podremos comprobar que **21 x 456** *aún es igual* **9576**.

¡Cuidado!

Aquí, más que nunca, conviene reiterar, una vez más, los errores de sintaxis más frecuentes:

- Los nombres de variables son distintos si se cambian mayúsculas y minúsculas. Pon mucho cuidado en escribirlos correctamente.
- Los nombres de las variables predefinidas, tales como $HTTP_GET_VARS, $_GET, etcétera van en mayúsculas.
- No olvides poner punto y coma al final de cada línea de instrucciones.
- Presta atención a la apertura/cierre de comillas y mucha más atención aún si se trata de comillas anidadas. En este caso procura usar (") para las exteriores y (') para las interiores.

¡Cuidado!

En modo local puedes establecer las configuraciones de **php.ini** a tu antojo y, además, estás utilizando una versión –4.3.11– de PHP que permite **superglobales**. Esta versión –junto con la posibilidad de modificar php.ini– te permite utilizar cualquiera de las opciones de transferencia de variables.
Pero, si pretendes publicar tus páginas utilizando un **hosting** ajeno debes cerciorarte de cual es su versión de PHP –no todos tienen instaladas versiones superiores a 4.1.0– y conocer la configuración de sus php.ini.
Ten en cuenta que allí no vas a poder modificar las configuraciones y de no tener en cuenta estos aspectos, puedes verte obligad@ a **modificar** tu código fuente para adecuarlo a la configuración de tu *hosting*.

Un formulario

```
<HTML>
<HEAD>
</HEAD>
<BODY>

<!-- Un formulario debe empezar siempre con una etiqueta de este tipo <form ...> en la que será obligatorio indicar con esta sintaxis action='nombre.extension' nombre.extension debe contener el nombre (o la ruta completa en el caso de que estuviera en un directorio o hosting distinto del que alberga el documento que contiene el formulario desde el que se realiza la petición) Es opcional incluir method que puede tener dos valores method='GET' ó method='POST', por defecto (cuando no se indica) el envio se realizará usando method='GET' También es opcional -a los efectos de PHP- incluir name. Ese valor es útil cuando se incluyen scripts del lado del cliente del tipo JavaScript //--> <form name='mi_formulario' action='formu1.php' method='post'> <!-- Pueden incluirse textos dentro del formulario --> Escribe tu nombre: <!-- Uno de los tipos de campos posibles es el tipo texto su sintaxis (hablamos de HTML) requiere la etiqueta <input type='text'> que indica el contenido del texto esa etiqueta debe incluir obligatoriamente un name='nombre' el nombre a usar serán caracteres alfabéticos, sin tildes ni eñes y sin espacios. Salvo excepciones que comentaremos no puede usarse el mismo nombre para dos campos distintos el value='' puede no contener nada entre las comillas tal como ocurre aquí o contener el texto que por defecto queremos que aparezca en ese campo al cargar el formulario. el size=xx es opcional. Su utilidad es la de ajustar el tamaño de la ventana al número de caracteres que se indiquen //--> <input type='text' name='nombre' value='' size=15><br> Escribe tu clave: <!-- <input type='password'> solo se diferencia del anterior en que en el momento de rellenarlo se sustituyen los carecteres visualizados (no el contenido) por asteriscos //--> <input type='password' name='clave' value=''><br> Elige tu color de coche favorito:<br> <!-- Los <input type='radio'> permite optar entre varios valores posibles. Habrá que repetirlos tantas veces como opciones queramos habilitar. Todos los input –correspondientes a la misma opción– deben tener el mismo nombre (name) value='loquesea' deberá tener un valor distinto en cada uno de ellos. Ese valor (loquesea) será transferido a través del formulario Si queremos que una opción aparezca marcada (por defecto) al cargar el formulario, deberemos incluir en su etiqueta la palabra checked los contenidos de value no se visualizan en el navegador por lo que conviene incluir una descripción de los valores después de cerrar la etiqueta de cada input Al enviar el formulario solo se transmite el value correspondiente a la opción seleccionada //--> <input type='radio' name='color' value='Rojo'>Rojo</br> <input type='radio' checked name='color' value='Verde'>Verde</br> <input type='radio' name='color' value='Azul'>Azul</br> Elige los extras:<br> <!-- Cada uno de los <input type='checkbox'> requiere un nombre distinto (name) y un valor (value) permite optar entre varios Esos valor (loquesea) serám transferido a través del formulario cuando la casilla de verificación esté marcada Si queremos que una casilla aparezca marcada (por defecto) al cargar el formulario, deberemos incluir en su etiqueta la palabra checked los contenidos de value tampoco aquí se visualizan en el navegador por lo que conviene incluir una descripción de los valores después de cerrar la etiqueta de cada input Al enviar el formulario solo se transmite los value
```

decíamos que se utilizaba el método **GET**, pero en el caso de los formularios son posibles ambos métodos. Conviene tenerlo en cuenta.

Método GET

No se diferencia en nada del descrito para el supuesto anterior. Utiliza las mismas variables predefinidas, las utiliza con idéntica sintaxis y se comporta de igual forma en lo relativo a las opciones de **register_globals**.

Los nombres de las variables son en este caso, los incluidos como **name** en cada una de las etiquetas del formulario.

Respecto a los valores de cada variable, éstos serían los recogidos del formulario. En los casos de campos tipo: *text*, *password* y *textarea* serían los valores introducidos por el usuario en cada uno de esos campos.

En el caso de los campos tipo *radio* –en el que varias opciones pueden tener el mismo nombre– recogería el valor indicado en la casilla marcada; mientras que si se trata de campos tipo *checkbox* se transferirían únicamente las variables –y los valores– que corresponden a las casillas marcadas.

Si se tratara de un campo tipo *hidden* se transferiría el valor contenido en su etiqueta y, por último, en el caso del *select* sería transferido como valor de la variable la parte del formulario contenida entre las etiquetas <option></option> de la opción seleccionada.

Método POST

En el caso de que el método de envío sea POST hay una diferencia a tener en cuenta en cuanto a las variables que recogen la información. Ahora será:

$HTTP_POST_VARS['*n1*']

quien haga la función atribuida en el método anterior a:

$HTTP_GET_VARS['*n1*']

y ocurrirá algo similar con las superglobales, que pasarían a ser del tipo:

$_POST['*n1*']

en sustitución del $_GET['*n1*'] usado en el caso del método GET

Si **register_globals** está en **On** el comportamiento de las *variables directas* es idéntico con ambos métodos.

Identificación del método de envío

PHP recoge en una variable el método utilizado para enviar los datos desde un formulario. Se trata de la variable **REQUEST_METHOD**.

Puede ser invocada como una *variable directa* (en caso de que **register globals** esté en **on**) o a través de una de las **variables de servidor**.

En el primer caso la variable se llamaría:

$REQUEST_METHOD

y en el segundo:

$HTTP_SERVER_VARS

```
correspondiente a la opcion marcadas //--> <input type='checkbox' name="acondicionado" value="Aire"> Aire acondicionado<br> <input type='checkbox' checked name="tapiceria" value="Tapicieria"> Tapiceria en piel<br> <input type='checkbox' name="llantas" value="aluminio"> Llantas de aluminio<br> ¿Cual es el precio máximo<br> que estarías dispuesto a pagar? <!-- La etiqueta <input type='select'> requiere un nombre y requiere también una etiqueta de cierre </select> Entre ambas -apertura y cierre- deben incluirse las diferentes opciones entre las de etiquetas <option>valor<option> Al enviar el formulario se transmite lo contenido después de opción en la opción seleccionada si dentro de una etiqueta option escribimos selected será esa la que aparezca por defecto al cargarse el formulario//--> <select name="precio"> <Option>Menos de 6.000 euros</option> <Option>6.001 - 8.000 euros</option> <Option selected >8.001 - 10.000 euros</option> <Option>10.001 - 12.000 euros</option> <Option>12.001 - 14.000 euros</option> <Option>Más de 14.000 euros</option> </select>
<!-- Las áreas de texto deben tener una etiqueta de apertura <textarea name='checkbox'> seguida de una etiqueta de cierre </textarea> Dentro de la etiqueta de apertura puede incluirse rows=xx (indicará el número de filas) cols=yy (indicará el ancho expresado en número de caracteres) y opcionalmente un value='lo que sea...' que puede contener el texto que -por defecto- pretendemos que aparezca en ese espacio en el momento de cargar rl formulario //--> <br> Escribe aquí cualquier otro comentario:<br> <textarea rows=5 cols=50 name='texto'></ textarea><br> <!-- El <input type='hidden'> permite insertar en un formulario una valor oculto que no requiere ser cumplimentado por el usuario y que no aparece visible en el documento requiere un name y un value //--> <input type="hidden" name='oculto' value='Esto iría oculto'><br> <!-- El <input type='submit'> es el encargado de ejecutar la action incluida en la etiqueta de apertura del formulario que en este caso sería la llamada a la página que se indica en la action El texto que incluyamos en value='enviar...' será el que se visualice en el propio botón de envio //--> <input type="submit" value="enviar"> <!-- El <input type='reset'> permite borrar todos los contenidos del formulario y reestablecer los valores por defecto de cada campo //--> <input type="reset" value="borrar"> <!-- La etiqueta </form> es la etiqueta de cierre del formulario //--> </FORM> </BODY> </HTML>
```

No incluiremos la opción *Ver ejemplo* hasta que hayamos elaborado los documentos **formuxx.php** incluidos en la **action** de este formulario.

Scripts para recoger los datos del formulario anterior

Insertaremos ahora los diferentes tipos scripts utilizables, especificando las condiciones de utilización de cada uno de ellos.

Los hemos llamado **formu1.php**, **formu2.php**, etcétera.

Debajo del código fuente de cada uno de ellos, hemos incluido dos enlaces. En el primero de ellos se utiliza el formulario que vemos aquí arriba modificando únicamente el valor de **action** para adecuarlo al nombre de cada script.

En el segundo de los enlaces utilizaremos un formulario en el que, además de las modificaciones anteriores, se utilice **GET** como método.

De esta forma, tendrás la oportunidad de comprobar el funcionamiento o no funcionamiento anunciado en cada uno de los supuestos.

Sería buena idea que *experimentaras* con ellos tanto bajo register_globals=ON como cuando esté en modo OFF.

Este primero funcionará tanto cuando el método sea POST como cuando sea GET. Requiere que el **php.ini** contenga la opción **register_globals=ON**

```
<?
echo "El method que ha usado fué: ",$REQUEST_METHOD,"<br>";
echo $nombre,"<br>";
echo $clave,"<br>";
echo $color,"<br>";
echo $acondicionado,"<br>";
echo $tapiceria,"<br>";
echo $llantas,"<br>";
echo $precio,"<br>";
echo $texto,"<br>";
echo $oculto,"<br>";
?>
```

«Con action = POST»	«Con action = GET»

Este otro **requiere** que el método especificado en el formulario de envío sea **POST**. El valor de register_globals no afectaría a su funcionalidad.

```
<?
echo "El method usado fué: ",$HTTP_SERVER_VARS[REQUEST_METHOD],"<br>";
echo  $HTTP_POST_VARS['nombre'],"<br>";
echo  $HTTP_POST_VARS['clave'],"<br>";
echo  $HTTP_POST_VARS['color'],"<br>";
echo  $HTTP_POST_VARS['acondicionado'],"<br>";
echo  $HTTP_POST_VARS['tapiceria'],"<br>";
echo  $HTTP_POST_VARS['llantas'],"<br>";
echo  $HTTP_POST_VARS['precio'],"<br>";
echo  $HTTP_POST_VARS['texto'],"<br>";
echo  $HTTP_POST_VARS['oculto'],"<br>";
```

[REQUEST_METHOD]

Cuando PHP permita el uso de variables **superglobales** se puede utilizar:

$_SERVER[REQUEST_METHOD]

Una **advertencia** importante.

Observa que en este caso **no se incluyen comillas** dentro del corchete como ocurría con todos los nombres de variable anteriores.

Diferencias ente los métodos GET y POST

Las diferencias entre uno y otro método son las siguientes:

Método GET

Las particularidades de este método son las siguientes:

– Al ser enviado el formulario se *carga* en el navegador la dirección especificada como **action**, se le añade un **?** y a continuación se incluyen los datos del formulario. Todos los datos de la petición **van a ser visibles** desde la *barra de direcciones del navegador*.

– Únicamente son aceptados los caracteres ASCII.

– Tiene una limitación en el tamaño máximo de la cadena que contiene los datos a transferir. En IE esa limitación es de 2.083 caracteres.

Método POST

No tiene las limitaciones indicadas para el caso de **GET** en lo relativo a **visibilidad** ni en cuanto a aceptación de caracteres no ASCII.

Este método de transferencia de datos es el más habitual cuando se utilizan formularios.

Tipos de contenidos de los formularios

Respecto a los formularios, vamos a contemplar un último aspecto: la *forma de encriptar* de los datos en el momento de la transmisión (**ENCTYPE**).

Puede especificarse dentro de la etiqueta**<form>** utilizando la sintaxis: **enctype**='*valor*'.

En el caso de que no se especifique, tomará el valor **application / x-www-form-urlencoded**, sea **GET** o **POST** el método que se utilice.

El método **POST** admite **multipart/form-data** como una opción alternativa a la anterior. Suele utilizarse cuando se trata de enviar grandes cantidades de datos, formularios en los que se adjuntan ficheros, datos *no ASCII* o contenidos *binarios*.

Las diferencias básicas entre ambos modos de encriptación son las siguientes:

En el tipo **application/x-www-form-urlencoded** los nombres de control y los valores se transforman en *secuencias de escape*, es decir, convirtiendo cada byte en una cadena **%HH**, donde **HH** es la *notación hexadecimal del valor del byte*.

Además, los *espacios* son convertidos en **signos +**, los *saltos de línea* se representan

```
?>
```

«Con action = POST»	«Con action = GET»

Para utilizar eficazmente este script **es necesario** que el método especificado en el formulario de envío sea **GET**.
El valor de register_globals no afectaría a su funcionalidad.

```
<?
echo "El method  usado fué: ",$HTTP_SERVER_VARS[REQUEST_METHOD],"<br>";
echo  $HTTP_GET_VARS['nombre'],"<br>";
echo  $HTTP_GET_VARS['clave'],"<br>";
echo  $HTTP_GET_VARS['color'],"<br>";
echo  $HTTP_GET_VARS['acondicionado'],"<br>";
echo  $HTTP_GET_VARS['tapiceria'],"<br>";
echo  $HTTP_GET_VARS['llantas'],"<br>";
echo  $HTTP_GET_VARS['precio'],"<br>";
echo  $HTTP_GET_VARS['texto'],"<br>";
echo  $HTTP_GET_VARS['oculto'],"<br>";
?>
```

«Con action = POST»	«Con action = GET»

Este otro **requiere** que el método especificado en el formulario de envío sea **POST** y que la versión de PHP soporte variables **superglobales**.
El valor de register_globals no afectaría a su funcionalidad.

```
<?
echo "El method  usado fué: ",$_SERVER[REQUEST_METHOD],"<br>";
echo  $_POST['nombre'],"<br>";
echo  $_POST['clave'],"<br>";
echo  $_POST['color'],"<br>";
echo  $_POST['acondicionado'],"<br>";
echo  $_POST['tapiceria'],"<br>";
echo  $_POST['llantas'],"<br>";
echo  $_POST['precio'],"<br>";
echo  $_POST['texto'],"<br>";
echo  $_POST['oculto'],"<br>";
?>
```

«Con action = POST»	«Con action = GET»

En este supuesto sería necesario que el método especificado en el formulario de envío sea **GET** y que la versión de PHP instalada en el servidor que lo aloja soporte variables **superglobales**.
El valor de register_globals no afectaría a su funcionalidad.

```
<?
echo "El method que ha usado fué: ",$_SERVER[REQUEST_METHOD],"<br>";
echo  $_GET['nombre'],"<br>";
echo  $_GET['clave'],"<br>";
echo  $_GET['color'],"<br>";
echo  $_GET['acondicionado'],"<br>";
echo  $_GET['tapiceria'],"<br>";
echo  $_GET['llantas'],"<br>";
echo  $_GET['precio'],"<br>";
echo  $_GET['texto'],"<br>";
echo  $_GET['oculto'],"<br>";
?>
```

«Con action = POST»	«Con action = GET»

La variable $_REQUEST

PHP también dispone –a partir de su versión 4.1.0– de la variable **$_REQUEST** (de tipo superglobal) que aúna las funcionalidades de **$_GET** y **$_POST** y que recoge en variables del tipo **$_REQUEST['***nombre***']** tanto los valores transferidos mediante el método **GET** como mediante **POST**

$_REQUEST, a diferencia de $_GET y $_POST, no dispone de equivalentes en versiones anteriores. Ello quiere decir, **no existe** una variable (no superglobal) del tipo $HTTP_REQUEST_VARS con $HTTP_GET_VARS o con $HTTP_POST_VARS.

En este ejemplo hemos incluido dos scripts que solo solo se diferencian en el *method* especificado en el formulario.

Al ejecutarlos podremos comprobar que, independientemente del método usado, $_REQUEST recoge los valores transferidos desde el formulario.

```
<?
/* Al ejecutar por primera vez el script la variable pepe será nula
   ya que no se ha transferido aún el formulario. Al pulsar sucesivamente
   en el botón Enviar iremos visualizando los valores que se vayan
```

como **%0D%0A**, el *nombre y el valor se separan con el* signo **=** y los *diferentes bloques nombre/valor*, se separan con el carácter **&**.

En cuanto a la encriptación tipo **multipart/form-data**, sigue las reglas de las transferencias MIME, que comentaremos más adelante cuando tratemos el tema del correo electrónico.

¡Cuidado!

Cuando se incluye una *cadena vacía* ("") como valor de **action** en un formulario se recargará el mismo documento como respuesta al envío del formulario.

La seguridad en los envíos de datos

El tema de la seguridad es una preocupación constante entre los usuarios de Internet.

Cuando utilizamos las técnicas que venimos comentando en esta página –nos referimos siempre al caso de servidores remotos– corremos dos tipos de riesgo de seguridad que no estaría de más tener en cuenta.

El riesgo de que la información sea interceptada durante el proceso de transmisión desde el cliente hasta el servidor lo compartimos con todos los demás usuarios de la Red, pero hay otro –el *riesgo de daños en los contenidos de nuestro espacio de servidor*– que es **exclusivamente** nuestro.

La *transparencia* del método GET es tal, que incluso muestra –en el momento del envio– todos los datos en la barra de direcciones del navegador. Eso permite que cualquier usuario pueda conocer a simple vista la ruta completa hasta el script, así como los nombres y valores de las variables.

Cuando se usa el método POST los formularios son un poco más *discretos*, pero igual de transparentes. El *código* de cualquier formulario estará accesible sólo con ir a la opción *Ver código fuente* y allí estarán de nuevo todos los datos: nombre del script, nombres de las variables, etcétera, con lo que, cualquier usuario y desde cualquier sitio, puede acceder a ese script.

No haría falta ni usar *nuestro* formulario. Bastaría guardar una copia del mismo en el ordenador del *visitante* y después –haciendo ligerísimos retoques– se podría acceder a nuestro script sin necesidad de utilizar el formulario alojado en nuestro servidor.

Si pensamos que uno de nuestros scripts puede estar diseñado con el fin de modificar algunos de los contenidos de nuestro espacio –**borrar** datos, por ejemplo– seguramente sería cuestión de empezar a preocuparnos, y mucho más si en nuestro servidor tenemos datos *importantes*.

Existen formas de evitar, o al menos reducir, este tipo de riesgos. Restringir a usuarios autorizados el uso de algunos subdirectorios es una de ellas, almacenar datos importantes fuera del directorio root del servidor es otra y el uso de algunas de las variables predefinidas como elementos de protección puede ser

```
   transfiriendo */
print "He recibido la variable pepe con valor: ".$_REQUEST['pepe'];
/* al enviar el formulario se recargará este mismo documento
ya que hemos puesto action="" */
?>
<form name="prueba" method="post" action="">
<input type="text" name="pepe" value=''>
<input type="submit" value="enviar">
```

«Con action = POST»	«Con action = GET»

Ejercicio nº 9

Crea dos documentos llamados respectivamente **formulario1.php** y **visor1.php**. En el primero de ellos incluye un formulario que permita recoger datos personales y académicos de tus alumnos, utilizando todos los tipos de campos de formulario que conozcas. Los datos insertados en ese formulario deberán ser visualizados después de su envío –con cualquier configuración de register_globals y con cualquier versión PHP– a través del documento visor1.php. Utiliza los recursos estéticos –fondos, colores, tipografía, etcétera– que estimes oportunos para una correcta presentación

Ejercicio nº 10

Con criterios similares en cuanto a estética y funcionalidad a los del ejercicio anterior, te proponemos que crees dos nuevos documentos con nombres **formulario2.php** y **visor2.php**. En este caso el formulario deberá poder recoger el nombre y apellidos de un alumno hipotético al que debemos formularle dos preguntas. La primera de ellas con cuatro posibles respuestas entre las que deba elegir como válida una de ellas. La segunda, también con cuatro respuestas, deberá permitir marcar las respuestas correctas que pueden ser: todas, ninguna, o algunas de ellas.
El alumno debería poder insertar sus datos personales en el formulario y elegir las respuestas a las preguntas formuladas. Al pulsar en el *botón* de envío, los datos del alumno y las respuestas elegidas deben visualizarse a través del documento **visor2.php**.

una tercera.

Hacemos este comentario a *título meramente informativo*. Por el momento nos basta con manejar los formularios, pero queremos que tengas presente la existencia de ese tipo de riesgos. Más adelante, ve- remos la manera de *tratar de evitarlos*.

Ejercicio nº 8

Crea dos documentos llamados respectivamente *color1.php* y *color2.php*.
En el primero de ellos incluye un formulario en el que -mediante un menú de opciones– se pueda elegir uno de entre cinco colores (especifica como *value* el código hexadecimal de cada uno de ellos).
Al enviar los datos, deberá cargarse la página *color2.php* con el color de fondo correspondiente a la opción seleccionada en el formulario.