



# PHP & MySQL

*Anexo 4.- Manejo de ficheros*



**DISTRIBUIDO POR:**

**CENTRO DE INFORMÁTICA PROFESIONAL S.L.**

C/ URGELL, 100  
08011 BARCELONA  
TFNO: 93 426 50 87

C/ RAFAELA YBARRA, 10  
48014 BILBAO  
TFNO: 94 448 31 33

[www.cipsa.net](http://www.cipsa.net)

**RESERVADOS TODOS LOS DERECHOS. QUEDA PROHIBIDO TODO TIPO DE REPRODUCCIÓN TOTAL O PARCIAL DE ESTE MANUAL, SIN PREVIO CONSENTIMIENTO POR EL ESCRITOR DEL EDITOR**

## Manejo de ficheros

Las aplicaciones web requieren casi siempre almacenar datos de modo que puedan emplearse posteriormente. Esto puede hacerse empleando ficheros o bases de datos.

El uso de ficheros permite almacenar y recuperar cualquier tipo de datos aunque preferiblemente se emplea para guardar datos de configuración y secciones de código HTML a modo de plantillas.

Las operaciones posibles en un fichero son la lectura y escritura de datos:

### Para guardar datos:

- Abrir el fichero, y si no existe crearlo.
- Escribir los datos deseados, o añadirlos al final del mismo si ya existía.
- Cerrar el fichero.

### Para leer datos:

- Abrir el fichero. Esta operación puede fallar si el fichero no existe o no se encuentra.
- Leer los datos necesarios.
- Cerrar el fichero.

## Apertura de ficheros

Tanto para guardar como para recuperar datos de un fichero, la primera operación necesaria es abrir el fichero llamando a la función ***fopen()***.

```
resource fopen ( string $filename , string $mode  
                [, bool $use_include_path = false  
                [, resource $context ]] )
```

### Ubicación del archivo

El primer parámetro determina el nombre y ruta del archivo a abrir.

Si sólo se indica el nombre del archivo sin ruta alguna, se entiende que el archivo a abrir se encuentra en la misma carpeta que el propio script PHP. Si se indica una ruta relativa se toma como referencia la ubicación del propio script PHP.

**Ejemplo:** El siguiente código abre el archivo “*datos.txt*” situado en la misma carpeta que el script PHP:

```
$res = fopen("datos.txt", "r");
```

**Ejemplo:** El siguiente código abre el archivo “datos.txt” situado dentro de la subcarpeta “ficheros”:

```
$res = fopen("ficheros/datos.txt", "r");
```

**Ejemplo:** El siguiente código abre el archivo “datos.txt” situado fuera de la carpeta en la que se encuentra el script PHP:

```
$res = fopen("../datos.txt", "r");
```

También pueden emplearse rutas relativas tomando como referencia la carpeta de publicación del servidor web. Su ruta puede obtenerse a partir de la matriz predefinida **\$\_SERVER** mediante la clase “DOCUMENT\_ROOT”:

**Ejemplo:** El siguiente código abre el fichero “datos.txt” situado dentro de la subcarpeta “prueba” de la carpeta de publicación del servidor web:

```
$document_root = $_SERVER["DOCUMENT_ROOT"];  
$res = fopen("$document_root/prueba/datos.txt", "r");
```

(\*) Con XAMPP, la ruta dada por **\$\_SERVER["DOCUMENT\_ROOT"]** en Windows es por defecto: “c:\xampp\htdocs\”

También pueden emplearse rutas absolutas (indicando la letra del disco C:\ en Windows) aunque no se recomienda, ya que puede provocar problemas al instalar la aplicación en servidores web con diferentes configuraciones o sistemas operativos.

También es posible acceder a ficheros remotos vía HTTP o FTP indicando la URL correspondiente. En el caso de utilizar HTTP, los ficheros sólo pueden abrirse en modo lectura.

```
$descriptor = fopen("http://www.images.com/ima03983.jpg", "r" );
```

En caso de utilizarse FTP, es necesario indicar el usuario y contraseña de una cuenta con permisos de acceso siguiente el siguiente formato:

**ftp://<usuario>:<clave>@<url\_fichero>**

```
$descriptor = fopen("ftp://roger:0000@datos.com/ima0394.jpg", "r+" );
```

(\*) El acceso a ficheros remotos puede ser restringida por la directiva **allow\_url\_fopen** en el archivo de configuración de PHP ( *php.ini* ).

## Modos de apertura

El segundo parámetro determina el *modo de apertura* que indica qué se va a hacer con el fichero. Los posibles valores son los siguientes:

Valor	Significado
<b>r</b>	El fichero se abre exclusivamente para leer desde el principio.
<b>r+</b>	El fichero se abre para leer y escribir desde el principio.
<b>w</b>	El fichero se abre para escribir desde el principio. Si ya existe se sobrescribe eliminando todos los datos existentes ( <i>truncado</i> ). Si no existe se crea.
<b>w+</b>	El fichero se abre para leer y escribir desde el principio. Si ya existe se sobrescribe eliminando todos los datos existentes ( <i>truncado</i> ). Si no existe se crea.
<b>a</b>	El archivo se abre para escribir a partir del fin del archivo ( añadiendo ). Si no existe se crea.
<b>a+</b>	El archivo se abre para leer y escribir a partir del fin del archivo ( añadiendo ). Si no existe se crea.
<b>x</b>	Equivalente a ( <b>w</b> ) pero si el archivo no existe no se crea y se muestra un aviso.
<b>x+</b>	Equivalente a ( <b>w+</b> ) pero si el archivo no existe no se crea mostrando un aviso.

Cuando se abre un fichero se crea un puntero que determina la posición del dato que va a ser leído o escrito a continuación. Cuando se abre un fichero con las opciones *r, r+, w, w+*; el puntero se posiciona al principio de modo que el fichero comienza a leerse o reescribirse desde el principio. En el caso de las opciones *a, a+*; el puntero se posiciona al final del fichero permitiendo escribir datos al final del mismo.

## Obtención del apuntador de fichero

La función ***fopen()*** retorna un apuntador al fichero (valor de tipo resource) que representa el fichero abierto. Este valor es requerido por el resto de funciones para leer o escribir en el mismo.

En caso de producirse un error en la apertura del fichero, ( o si éste no existe indicando los modos de apertura "**x**" y "**x+**" ), la función retorna un valor lógico FALSE.

**Ejemplo:** El siguiente código muestra la apertura de un fichero comprobando que el valor retornado por ***fopen()***:

```
// Obtencion ruta raiz
$document_root = $_SERVER["DOCUMENT_ROOT"];
// Apertura de fichero
@$res = fopen("$document_root/prueba/datos.txt", "r");
// Archivo abierto OK?
if ( !$res ) { // ERROR DE APERTURA DE FICHERO
    echo "Error de apertura de fichero";
    exit;
}
// Cerrado de fichero
fclose($res);
echo "OK";
```

**El empleo de '@' al principio de la línea del fopen() evita la visualización de un mensaje de advertencia si el fichero no es accesible.**

### Comprobación de la existencia de un fichero.

Para comprobar si un determinado fichero existe antes de intentar abrirlo PHP dispone de la función ***file\_exists()***:

```
bool file_exists ( string $filename )
```

La función recibe como parámetro el nombre y ruta del fichero cuya existencia quiere comprobarse y retorna un valor lógico cierto o falso indicando si existe o no.

```
// Obtencion ruta raiz
$document_root = $_SERVER["DOCUMENT_ROOT"];
// Existe el fichero?
if ( file_exists("$document_root/prueba/datos.txt") ) {
    // Apertura de fichero
    $res = fopen("$document_root/prueba/datos.txt", "r");
    // Cerrado de fichero
    fclose($res);
    echo "OK";
} else echo "El fichero no existe";
```

### Cerrado de fichero

Una vez terminadas las operaciones de lectura y/o escritura de un fichero *es preciso cerrarlo* con la función ***fclose()***. La función requiere como único parámetro el apuntador al fichero devuelto por la función ***fopen()***.

```
bool fclose ( resource $handle )
```

## Escritura de ficheros

La escritura de datos en fichero es bastante sencilla en PHP. Para ello pueden emplearse la función ***fwrite()***:

```
int fwrite ( resource $handle
            , string $string
            [, int $length ] )
```

(\*) La función ***fputs()*** es un alias de la función ***fwrite()***

El primer parámetro es el apuntador al fichero contra el que se quiere escribir. El segundo parámetro es la cadena con el texto a escribir, y el tercer parámetro opcional limita la escritura a la cantidad de bytes indicados. Si no se indica se escribe la cadena entera y la función retorna la cantidad de bytes escritos.

**Ejemplo:** El siguiente código muestra la apertura del fichero “*datos.txt*” y el guardado en el mismo de la tabla de multiplicar del 5 línea a línea. Si el fichero ya existe se sobrescriben sus datos:

```
// Obtencion ruta raiz
$document_root = $_SERVER["DOCUMENT_ROOT"];
// Existe el fichero?
@$res = fopen("$document_root/prueba/datos.txt", "w");
if ( $res ) {
    // Escritura de la tabla de multiplicar del 5
    for ( $i = 0; $i < 10; $i++ ) {
        $linea = "5 x ".$i." = ".$i*5;
        fwrite($res, $linea."\r\n");
        echo "X";
    }
    fclose($res);
}
```

## Formato de los datos

El formato de un fichero determina el modo en que se organiza la información guardada en él. Esta información es necesaria para poder posteriormente recuperar la información en almacenada.

Los ficheros pueden contener meras líneas de texto separadas por un salto de línea (***\r\n***) como en el caso del ejercicio.

```
5 x 0 = 0
5 x 1 = 5
5 x 2 = 10
5 x 3 = 15
5 x 4 = 20
5 x 5 = 25
5 x 6 = 30
5 x 7 = 35
5 x 8 = 40
5 x 9 = 45
```

## Programación Web en PHP

También es posible almacenar conjuntos de datos en cada línea separándolos con un carácter delimitador como una coma ( , ) un tabulador ( \t ), o cualquier otro carácter haciendo la función de separador.

**Ejemplo:** Supóngase una pequeña aplicación PHP provista de un formulario (*form.html*) donde el usuario puede introducir el nombre, apellidos y edad de una persona para su envío a un script PHP y guardado en un fichero:

```
<!DOCTYPE html>
<html>
<head>
<title>Insert title here</title>
</head>
<body>
  <form action="write.php" method="post">
    <label for="nombre">NOMBRE</label>
    <input type="text" name="nombre"><br />
    <label for="apellido">APELLIDO</label>
    <input type="text" name="apellido"><br />
    <label for="edad">EDAD</label>
    <input type="text" name="edad"><br />
    <input type="submit" value="ENVIAR">
  </form>
</body>
</html>
```

NOMBRE

APELLIDO

EDAD

La página PHP (*write.php*) recoge los datos enviados desde el formulario y los añade al final del fichero "*personas.dat*" separando cada valor mediante un tabulador (\t) mostrando un enlace de vuelta al formulario inicial para proseguir añadiendo datos.

```
<!DOCTYPE html>
<html>
<head>
</head>
<body>
<div>
  <?php
    // Hay datos?
    if ( count($_POST) > 0 ) {
      // Obtención de los datos introducidos en el formulario
      $nombre = filter_input(INPUT_POST, "nombre");
      $apellido = filter_input(INPUT_POST, "apellido");
      $edad = filter_input(INPUT_POST, "edad");
      // Apertura de fichero para añadir
      $document_root = $_SERVER["DOCUMENT_ROOT"];
      @$f = fopen("$document_root/prueba/personas.dat", "a");
      if ($f) {
        // Añadido de línea con datos de la persona introducida
        fwrite($f, "$nombre\t$apellido\t$edad\r\n");
        // Cerrado de fichero
        fclose($f);
      } else echo "ERROR ABRIENDO FICHERO"; // Fichero no pudo abrirse
    } else echo "NO HAY DATOS"; // No se enviaron datos.
  ?>
  <br />
  <a href="form.html">VOLVER AL FORMULARIO</a>
</div>
</body>
</html>
```



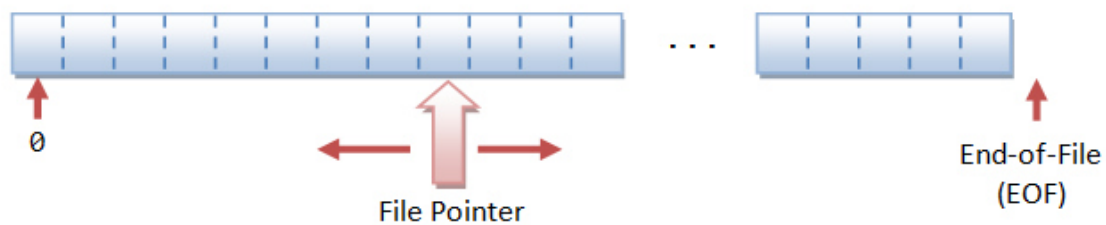
Tras la introducción de unas cuantas personas, el fichero de datos “personas.dat” se mostraría el siguiente contenido al ser abierto mediante el bloc de notas:

Roger	Petrov	12
Yuri	Estepanovich	23
Víctor	Tatamovich	27

### Lectura de ficheros

La lectura de un fichero comienza con la apertura del mismo con la función ***fopen()*** y la obtención del apuntador correspondiente.

Una vez abierto el fichero éste puede leerse de principio a fin. Cada lectura provoca el desplazamiento hacia delante del *puntero de fichero*. Este señala el siguiente valor a ser leído:



Representación visual del puntero de fichero en la lectura de un fichero

La lectura puede ser carácter a carácter o línea a línea en función de las funciones de PHP empleadas.

La lectura de todos los datos de un fichero se realiza empleando un bucle que debe detenerse cuando el *puntero de fichero* alcanza el fin del fichero (EOF) y no hay más datos que leer. Para ello se emplea la función ***feof()***:

```
bool feof ( resource $handle )
```

Esta función recibe como único argumento el descriptor del archivo y devuelve un valor lógico cierto si el fichero ha alcanzado el fin de fichero. Esto puede suceder nada más abrir el fichero si éste no tiene datos:

```
// Apertura de fichero para añadir
$document_root = $_SERVER["DOCUMENT_ROOT"];
$file = fopen("$document_root/prueba/personas.dat", "r");
while ( !feof($file) ) {
    // operaciones de lectura
}
// Cerrado de fichero
fclose($file);
```

## Lectura de caracteres

Los ficheros pueden leerse carácter a carácter empleando la función de PHP **fgetc()**:

```
string fgetc ( resource $handle )
```

La función recibe como único parámetro el descriptor de fichero y retorna el siguiente carácter leído o un valor lógico FALSO si se alcanza el fin de fichero (EOF).

*(\*) Se recomienda emplear el operador '===' para detectar si la función devuelve realmente un valor lógico falso, ya que algunos valores devueltos podrían interpretarse erróneamente como falso con el operador convencional '==' (p.ej: "0")*

**Ejemplo:** El siguiente página PHP lee el fichero "personas.dat" carácter a carácter y lo muestra. Los saltos de línea del fichero (\r\n) se sustituyen por etiquetas <br/>:

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<body>

<?php
    // Apertura de fichero
    $document_root = $_SERVER["DOCUMENT_ROOT"];
    @$file = fopen("$document_root/prueba/personas.dat", "r");
    // Comprobacion de apertura de fichero
    if ( $file ) {
        // Bucle de lectura mientras EOF = false
        while ( !feof($file) ) {
            // Obtencion de caracter
            $c = fgetc($file);
            // Comprobacion salto de linea \r
            if ( $c == "\r" ) {
                fgetc($file); // Se lee el siguiente caracter "\n"
                echo "<br />"; // se visualiza "<br />"
            } else {
                // Visualizacion del caracter obtenido
                echo $c;
            }
        }
        echo "FIN DE FICHERO";
        // Cerrado de fichero
        fclose($file);
    } else echo "FICHERO NO ENCONTRADO";
?>

</body>

</html>
```

## Lectura de líneas

PHP dispone de funciones que permiten leer una línea desde la posición actual hasta el siguiente salto de línea. Para ello pueden emplearse las siguientes funciones:

La función ***fgets()*** devuelve la siguiente línea de texto tal cual:

```
string fgets ( resource $handle [, int $length ] )
```

Esta función requiere como primer parámetro el descriptor del fichero.

El segundo parámetro es opcional y permite limitar la cantidad de caracteres a leer. Si la línea es más larga se leen la cantidad de caracteres indicada, el resto de la línea será leída por la siguiente llamada a ***fgets()***. Si la línea tiene menos caracteres que los indicados se lee al completo. Por defecto, siempre se lee la línea completa.

La función retorna la cadena leída o un valor lógico *FALSE* en caso de alcanzarse el fin de fichero (*EOF*).

**(\*) Se recomienda emplear el operador '===' para detectar si la función devuelve realmente un valor lógico falso, ya que algunos valores devueltos podrían interpretarse erróneamente como falso con el operador convencional '==' (p.ej: "0")**

**Ejemplo:** Este es el mismo código del ejemplo anterior que lee y muestra el contenido del fichero "personas.dat" pero empleando la función ***fgets()*** para leer el fichero línea a línea en vez de carácter a carácter:

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<body>
<?php
    // Apertura de fichero
    $document_root = $_SERVER["DOCUMENT_ROOT"];
    @$file = fopen("$document_root/prueba/personas.dat", "r");
    // Comprobacion de apertura de fichero
    if ( $file ) {
        // Bucle de lectura mientras EOF = false
        while ( !feof($file) ) {
            // Obtencion de la línea
            $linea = fgets($file);
            echo "$linea<br/>";
        }
        echo "FIN DE FICHERO";
        // Cerrado de fichero
        fclose($file);
    } else echo "FICHERO NO ENCONTRADO";
?>
</body>
</html>
```

Otra función que dispone PHP para la lectura de una línea es ***fgetcsv()***:

```
array fgetcsv ( resource $handle
               [, int $length = 0
               [, string $delimiter = ","
               [, string $enclosure = '"'
               [, string $escape = "\" ]]] )
```

Esta función lee una línea del fichero como ***fgets()***, pero analiza su contenido buscando valores separados por un carácter delimitador y los retorna en una matriz escalar.

El primer parámetro es el descriptor de fichero. El segundo parámetro es opcional y limita el máximo nº de caracteres que se leerán. Lo habitual es indicar la longitud de la línea más larga presente en el fichero para mejorar la velocidad. Si no se indica se lee la línea hasta el final. El tercer parámetro es también opcional y permite indicar el carácter delimitador. Por defecto se toma la coma ( , ) como delimitador.

La función retorna los valores obtenidos en una matriz o un valor lógico FALSO si se alcanza el fin de fichero (EOF). En caso de leerse una línea vacía, la función retorna una matriz con un único valor null.

**Ejemplo:** El siguiente código lee el fichero “*personas.dat*” y muestra los datos de cada persona registrada mediante una lista. Cada línea se lee y procesa con la función ***fgetcsv()*** asumiendo una longitud máxima de 100 caracteres para obtener los datos almacenados separados por el tabulador ( \t ) como carácter delimitador.

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<body>
<?php
    // Apertura de fichero
    $document_root = $_SERVER["DOCUMENT_ROOT"];
    @$file = fopen("$document_root/prueba/personas.dat", "r");
    // Comprobacion de apertura de fichero
    if ( $file ) {
        // Bucle de lectura mientras EOF = false
        while ( !feof($file) ) {
            // Lectura de la siguiente linea max = 100 caracteres
            $datos = fgetcsv($file, 100, "\t");
            // Visualizacion de valores obtenidos
            echo "<ul><li>$datos[0]</li><li>$datos[1]</li><li>$datos[2]</li></ul>";
        }
        echo "FIN DE FICHERO";
        // Cerrado de fichero
        fclose($file);
    } else echo "FICHERO NO ENCONTRADO";
?>
</body>
</html>
```

### Escritura y lectura de una sola vez

PHP dispone también de funciones que permiten tanto leer todo el contenido de un fichero y volcarlo de una sola vez en una variable, como escribir un fichero volcando en él toda la información contenida en una variable de una sola vez. Para estas operaciones se emplean las funciones ***file\_put\_contents()*** y ***file\_get\_contents()***:

La función ***file\_put\_contents()*** que permite volcar el contenido de una variable en un fichero de una sola vez, sin necesidad siquiera de abrir y cerrar el fichero con las funciones ***fopen()*** y ***fclose()***:

```
int file_put_contents ( string $filename
                      , mixed $data
                      [, int $flags = 0
                      [, resource $context ]] )
```

- El primer parámetro indica el nombre y ruta del fichero a escribir. Si no existe se crea, y si ya existe se sobrescribe por defecto.
- El segundo parámetro es la información a escribir. Puede ser una cadena, una matriz o cualquier otro tipo de dato. Si se indica como parámetro una matriz se escriben sus valores en el fichero uno detrás de otro sin separación.
- El tercer parámetro es opcional y permite configurar el modo en que se abre el fichero permitiendo por ejemplo añadir los datos a un fichero ya existente indicando la constante predefinida ***FILE\_APPEND***.

La función retorna como valor el nº de bytes escritos en el fichero, o un valor lógico FALSO si se produce algún error al abrir, crear o escribir el fichero.

La función ***file\_get\_contents()*** por su parte permitir leer de una sola vez todo el contenido de un fichero y volcarlo a una variable. Al igual que la función anterior no requiere ni abrir ni cerrar el fichero:

```
string file_get_contents ( string $filename
                          [, bool $use_include_path = false
                          [, resource $context
                          [, int $offset = 0
                          [, int $maxlen ]]] ] )
```

El primer parámetro y único obligatorio es la ruta y nombre del fichero a abrir.

Hay que tener cierta prevención a la hora de leer ficheros especialmente grandes ya que podrían saturar la memoria del servidor. Para esos casos puede emplearse el parámetro (***\$maxlen***) para indicar la cantidad máxima de bytes a leer, acompañado del parámetro (***\$offset***) para indicar el nº de bytes a partir del que comenzar la lectura. De este modo es posible leer cualquier archivo en bloques por grande que sea.

**Ejemplo:** La función `file_get_contents()` puede emplearse para cargar de una sola vez ficheros con código HTML para ser empleados como plantillas para insertar valores generados desde PHP y enviarlos al usuario como respuesta.

Para ver un ejemplo supóngase que tenemos el siguiente formulario HTML que solicita al usuario los datos personales de un individuo:

```
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
    <form action="show.php" method="post">
        <label for="nombre">NOMBRE</label>
        <input type="text" name="nombre"><br />
        <label for="apellido">APELLIDO</label>
        <input type="text" name="apellido"><br />
        <label for="edad">EDAD</label>
        <input type="text" name="edad"><br />
        <input type="submit" value="ENVIAR">
    </form>
</body>
</html>
```

Los datos son enviados a la página `"show.php"` el cual obtiene los valores enviados y los muestra al usuario:

```
<?php
    if ( count($_POST) == 0 ) die("No hay datos");

    // Obtención de los datos introducidos en el formulario
    $nombre = filter_input(INPUT_POST, "nombre");
    $apellido = filter_input(INPUT_POST, "apellido");
    $edad = filter_input(INPUT_POST, "edad");
?>

<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
    <div>
        NOMBRE: <?php echo $nombre; ?><br/>
        APELLIDO: <?php echo $apellido?><br/>
        EDAD: <?php echo $edad?>
    </div>
</body>
</html>
```

El problema de este planteamiento es que el código HTML y el código PHP están juntos en el mismo archivo `"show.php"`. Esto no es necesariamente malo, pero lo recomendable es mantener el diseño y el contenido en archivos separados para poder editarlos de manera independiente diseñadores y programadores.

Para conseguirlo, debemos trasladar todo el código HTML a un archivo (`datos.template`). Este archivo servirá de plantilla para contener el diseño de la página de modo que pueda ser editado sin interferir en el código PHP:

Código del fichero plantilla (*datos.template*):

```
<!DOCTYPE html>
<html>
<head>
<title>Insert title here</title>
</head>
<body>
    <div>
        NOMBRE: #nombre#<br/>
        APELLIDO: #apellidos#<br/>
        EDAD: #edad#<br/>
    </div>
</body>
</html>
```

El archivo contiene únicamente código HTML con el diseño de la página. El lugar en el que debe mostrarse el contenido (nombre, apellido y edad del usuario introducido previamente), se determina mediante etiquetas entre caracteres almohadillas (#).

El script que recibe los datos (*show.php*) debe cargar el contenido HTML del fichero plantilla, sustituir cada una de las etiquetas por el valor correspondiente, y enviar el resultado al usuario:

```
<?php
    if ( count($_POST) == 0 ) die("No hay datos");

    // Obtención de los datos introducidos en el formulario
    $nombre = filter_input(INPUT_POST, "nombre");
    $apellido = filter_input(INPUT_POST, "apellido");
    $edad = filter_input(INPUT_POST, "edad");

    // Carga el fichero plantilla
    $document_root = $_SERVER["DOCUMENT_ROOT"];
    $plantilla = file_get_contents("$document_root/prueba/plantilla.template");
    if ( $plantilla === FALSE ) die("Error cargando plantilla");

    // Matriz de etiquetas presentes en la plantilla
    $etiquetas = ["#nombre#", "#apellidos#", "#edad#"];

    // Matriz de valores por los que sustituir cada etiqueta de la plantilla
    $valores = [$nombre, $apellido, $edad];

    // Sustitución de etiquetas por valores y envío de página resultante al usuario
    echo str_replace($etiquetas, $valores, $plantilla);
?>
```

El resultado es una página HTML que integra diseño y contenido de manera transparente al usuario:

```
<!DOCTYPE html>
<html>
<head>
<title>Insert title here</title>
</head>
<body>
    <div>
        NOMBRE: Roger<br/>
        APELLIDO: Petrov<br/>
        EDAD: 23<br/>
    </div>
</body>
</html>
```

## Funciones de manejo de ficheros

PHP define otras funciones adicionales que permiten obtener información de los archivos al margen de su lectura y/o escritura.

```
int filesize ( string $filename )
```

Esta función devuelve la longitud en bytes de un fichero dado su ruta y nombre.

```
bool is_readable ( string $filename )  
bool is_writable ( string $filename )
```

## Gestión de ficheros

PHP dispone de funciones que permiten además eliminar o copiar ficheros fácilmente:

La función **unlink()** permite eliminar un archivo indicando su ruta y nombre:

```
bool unlink ( string $filename [, resource $context ] )
```

Esta función elimina el fichero indicado dada su ruta y nombre.

```
bool copy ( string $source , string $dest [, resource $context ] )
```

Esta función el archivo con el nombre y ruta indicados en el primer parámetro (\$source), y lo copia con la ruta y nombre indicados en el segundo parámetro (\$dest).

## Funciones de manejo de carpetas

PHP define una serie de funciones que permiten la creación, eliminación e inspección de los contenidos de carpetas. Un directorio puede abrirse con intención de inspeccionar los archivos y subcarpetas que contiene. Para ello se emplea la función **opendir()**:

```
resource opendir ( string $path [, resource $context ] )
```

La función recibe como parámetro la ruta a la carpeta y retorna un *apuntador a la carpeta* o un valor lógico FALSE en caso de no existir el directorio indicado.

```
bool is_dir ( string $filename )  
bool is_file ( string $filename )
```

Las funciones **is\_dir()** y **is\_file()** que devuelven un valor lógico indicando si la el elemento en la ruta indicada es una carpeta o un fichero respectivamente.

Una vez completada las operaciones con una carpeta, debe cerrarse empleando la función **closedir()** e indicando como argumento el apuntador a carpeta devuelto por la función **opendir()**.

```
void closedir ([ resource $dir_handle ] )
```



## Inspección de carpetas

PHP permite abrir una carpeta indicando su ubicación para tener acceso a los ficheros contenidos en la misma mediante la función **opendir()**, que devuelve *un descriptor de carpeta* a partir del cual pueden recorrerse los ficheros y subcarpetas contenidas mediante la función **readdir()**:

```
string readdir ([ resource $dir_handle ] )
```

Esta función recibe como parámetro el *descriptor de carpeta* retornado por **opendir()**, y devuelve con cada llamada el nombre ( sin ruta ) de cada fichero y subcarpeta contenida. Una vez recorridos todos los ficheros y subcarpetas la función retorna un valor lógico falso:

**Ejemplo:** El siguiente código muestra el contenido de una carpeta “pruebas” indicando la longitud de los ficheros y el mensaje “[CARPETA]” por cada subcarpeta:

```
<?php
// Obtencion ruta de carpeta a inspeccionar
$ruta = $_SERVER["DOCUMENT_ROOT"]."/prueba";
// Existe la carpeta?
if ( is_dir( $ruta ) ) {
    // Apertura de la carpeta
    $dir = opendir($ruta);
    echo "<table>";
    // Obtencion del nombre del primer elemento en la carpeta
    $archivo = readdir( $dir );

    while ( $archivo != false ) {
        echo "<tr>";
        echo "<td>$archivo</td>";
        echo "<td align='right'>";
        // El archivo es una carpeta?
        if ( is_dir($ruta."/".$archivo) ) echo "[DIRECTORIO]";
        // El archivo es un fichero?
        if ( is_file($ruta."/".$archivo) ) echo filesize($ruta."/".$archivo)." bytes";
        echo "</td>";
        // Obtencion del nombre del siguiente elemento en la carpeta.
        $archivo = readdir( $dir );
    }

    echo "</table>";
    // Cierre de la carpeta
    closedir($dir);
} else {
    echo "Directorio no existente.";
}
```

.	[DIRECTORIO]
..	[DIRECTORIO]
.buildpath	174 bytes
.project	722 bytes
.settings	[DIRECTORIO]
copia.jpg	879394 bytes
copy.php	97 bytes
datos.txt	118 bytes
datos2.dat	0 bytes

Es importante destacar que la función **readdir()** únicamente devuelve el nombre y no la ruta de cada elemento contenido en la carpeta. Las funciones **filesize()**, **is\_dir()** y **is\_file()** requieren sin embargo la ruta y nombre del elemento por lo que debe concatenarse delante la ruta de la carpeta para obtener la ruta relativa de cada elemento.

```
// El archivo es una carpeta?
if ( is_dir($ruta."/".$archivo)) echo "[DIRECTORIO]";
// El archivo es un fichero?
if ( is_file($ruta."/".$archivo)) echo filesize($ruta."/".$archivo). " bytes";
```

## La carpeta de trabajo

Se denomina *carpeta de trabajo* a la ubicación de referencia de las rutas relativas. La ruta de la carpeta actual de trabajo puede obtenerse mediante la función **getcwd()**.

```
string cwd ( void )
```

La función no recibe ningún argumento y devuelve una cadena con la ruta absoluta de la carpeta de trabajo actual, que por defecto coincide con la carpeta del fichero PHP que está ejecutándose. Esta puede también modificarse empleando la función **chdir()**:

```
bool chdir ( string $directory )
```

Esta función modifica el directorio de trabajo estableciéndolo en la ruta dada como argumento. A partir de entonces, todas las rutas relativas tomarán como referencia la nueva ubicación de la carpeta de trabajo.

**Ejemplo:** El siguiente código es una modificación del anterior en el que se emplea la función **chdir()** para establecer como carpeta de trabajo la carpeta inspeccionada. De este modo, se simplifica el acceso a los elementos en la carpeta siendo únicamente necesario indicar su nombre.

```
$ruta = $_SERVER["DOCUMENT_ROOT"]."/prueba";
if ( is_dir( $ruta) ) {
    $dir = opendir($ruta);
    // Modificación de la carpeta de trabajo
    chdir($ruta);
    echo "<table>";
    // Obtencion del nombre del primer elemento en la carpeta
    $archivo = readdir( $dir );
    while ( $archivo != false ) {
        echo "<tr>";
        echo "<td>$archivo</td>";
        echo "<td align='right'>";
        // El archivo es una carpeta?
        if ( is_dir($archivo)) echo "[DIRECTORIO]";
        // El archivo es un fichero?
        if ( is_file($archivo)) echo filesize($archivo). " bytes";
        echo "</td>";
        // Obtencion del nombre del siguiente elemento en la carpeta.
        $archivo = readdir( $dir );
    }
    echo "</table>";
    closedir($dir);
} else {
    echo "Directorio no existente.";
}
```

## Creación y eliminación de carpetas

PHP permite crear una carpeta en la ruta indicada con los permisos indicados mediante la función **mkdir()**:

```
bool mkdir ( string $pathname
            [, int $mode = 0777
            [, bool $recursive = false
            [, resource $context ]]] )
```

La función crea la carpeta en la ruta indicada en el primer parámetro (*\$pathname*) con los permisos indicados en el segundo parámetro opcional (*\$mode*)..

(\*) El parámetro *\$mode* consiste en tres componentes numéricos octales que especifican las restricciones de acceso para el propietario, el grupo de usuarios al que pertenece el propietario, y para todos los demás, en este orden. Un componente puede ser computado sumando los permisos necesarios para ese usuario objetivo base. El número 1 significa que se conceden derechos de ejecución, el número 2 significa que se puede escribir en el fichero, el número 4 significa que el fichero se puede leer. Sume estos números para especificar los derechos necesarios:

```
<?php
// Lectura y escritura para el propietario, nada para los demás
mkdir("/directorio/fichero", 0600);

// Lectura y escritura para el propietario, lectura para los demás
mkdir("/directorio/fichero", 0644);

// Todo para el propietario, lectura y ejecución para los otros
mkdir("/directorio/fichero", 0755);

// Todo para el propietario, lectura y ejecución para el grupo del propietario
mkdir("/directorio/fichero", 0750);
?>
```

**(\*) En servidores basados en Windows el valor del segundo parámetro es ignorado, y las carpetas creadas desde PHP reciben los permisos del usuario del servicio web Apache**

También es posible eliminar una carpeta desde PHP empleando la función **rmdir()**:

```
bool rmdir ( string $dirname [, resource $context ] )
```

La función elimina la carpeta indicada por la ruta relativa dada como primer argumento.

Ambas funciones **mkdir()** y **rmdir()** retornan un valor lógico cierto si la operación se ejecuta correctamente, y falso en caso contrario.

## Funciones de manejo de rutas

La ubicación de un fichero se componen de dos partes: el nombre del archivo, y la ruta que determina la secuencia de carpetas en las que está contenido.

- El valor de la matriz `$_SERVER['SCRIPT_FILENAME']` devuelve la ruta absoluta de la página/script PHP solicitado.
- El valor de la matriz `$_SERVER['PHP_SELF']` devuelve la ruta relativa de la página/script PHP solicitado.

```
$ruta_relativa = $_SERVER['PHP_SELF'];  
$ruta_absoluta = $_SERVER['SCRIPT_FILENAME'];  
echo "Ruta relativa [$ruta_relativa]<br />";  
echo "Ruta absoluta [$ruta_absoluta]<br />";
```

Ruta relativa [/prueba/folder.php]

Ruta absoluta [C:/xampp7/htdocs/prueba/folder.php]

PHP dispone de dos funciones específicas para obtener ambas partes a partir de una ruta completa.

La función **basename()** devuelve el nombre del fichero presente en la ruta indicada como primer parámetro:

```
string basename ( string $path [, string $suffix ] )
```

La función **dirname()** devuelve la ruta sin el nombre de fichero.

```
string dirname ( string $path [, int $levels = 1 ] )
```

```
$ruta_absoluta = $_SERVER['SCRIPT_FILENAME'];  
echo "ruta: $ruta_absoluta<br/>";  
echo "basename: ".basename($ruta_absoluta)."<br />";  
echo "dirname: ".dirname($ruta_absoluta)."<br />";
```

ruta: C:/xampp7/htdocs/prueba/folder.php

basename: folder.php

dirname: C:/xampp7/htdocs/prueba

(\*) En Windows, la barra (/) y la barra invertida (\) se usan como carácter separador de directorio. En otros entornos se usa la barra hacia delante (/).

## Archivos de configuración

Los archivos de configuración permiten almacenar secuencias de claves-valor empleados en múltiples páginas y scripts PHP de una aplicación. Esto evita que los valores se inscriban en el código de cada página y facilita su modificación desde el archivo de configuración

Los archivos de configuración tienen la misma forma que el archivo de configuración de PHP (*php.ini*), y sigue el siguiente formato:

```
; comentarios
[categoría1]
clave1 = valor1
clave2 = valor2
clave3 = valor3
[categoría2]
clave4 = valor4
clave5 = valor5
...
```

**(\*) Las líneas de comentario son aquellas que empiezan por el signo “;” y sus contenidos no participan en los datos contenidos en el fichero.**

Para obtener desde el código los valores almacenados en un determinado archivo de configuración es necesario emplear la función ***parse\_ini\_file()***:

```
array parse_ini_file ( string $filename
[, bool $process_sections = false
[, int $scanner_mode = INI_SCANNER_NORMAL ]] )
```

La función recibe como parámetro la ruta al archivo de configuración y devuelve un descriptor para acceder a sus datos. El valor devuelto es una matriz bidimensional que permite acceder a cada uno de los valores indicando la categoría y la clave correspondiente:

```
$valor = $descriptor[<categoría>][<clave>];
```

## ***Transferencia de Archivos***

Una operación relativamente habitual en las aplicaciones Web es la transferencia de archivos entre el servidor Web y el navegador del cliente. La operación de transferencia puede realizarse en ambos sentidos:

- Desde el servidor Web → Al navegador del cliente = *Descarga*.
- Desde el navegador del cliente → Al servidor Web = *Subida*.

### **Subida de archivos al servidor ( *Uploading* )**

La subida de archivos consiste en enviar un archivo desde el navegador del cliente al servidor Web para su almacenamiento o procesamiento ( p.ej: incluir sus datos en una base de datos..., etc ).

La subida de archivos se configura mediante ciertas directivas del archivo de configuración de PHP ( `php.ini` ). En concreto; la directiva ***file\_uploads*** debe estar habilitada ( valor 1 / "on" ) para que la subida de archivos al servidor sea posible.

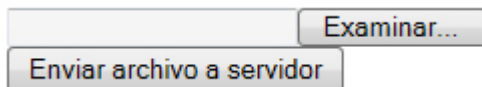
- ***file\_uploads***: ( valor por defecto: 1 ): Habilita o deshabilita la subida de archivos al servidor Web.
- ***upload\_max\_filesize***: ( valor por defecto: 2M ): Determina el tamaño máximo expresado en Megabytes que puede tener un archivo subido.
- ***post\_max\_size***: ( valor por defecto: 8M ): Determina el tamaño máximo expresado en Megabytes de los datos enviados al servidor Web mediante POST. ( Su valor debe ser siempre superior a `upload_max_filesize` ).
- ***upload\_tmp\_dir***: Los archivos subidos a servidor deben estar completos antes de poder ser utilizado, por lo que durante su la operación de transferencia se almacenan en una carpeta temporal en el servidor Web cuya ruta se indica en esta directiva. ( En XAMP por defecto: "`c:\xampp\tmp`" )

La selección y envío de un fichero a través de una página web desde el navegador del cliente realiza mediante un formulario HTML con las siguientes características:

- La etiqueta ***<form>*** debe llevar el atributo *method="post"*.
- La etiqueta ***<form>*** debe llevar el atributo *enctype = "multipart/form-data"* para habilitar el envío de un archivo al servidor.
- El formulario debe constar de un elemento ***<input type="file">*** que permite seleccionar el archivo a enviar al servidor.

**Ejemplo:** El siguiente código muestra un formulario que permite la subida de un archivo al servidor contra el script “*upload.php*”.

```
<!DOCTYPE html>
<html>
<head>
<title>Insert title here</title>
</head>
<body>
  <form action="upload.php" method="POST" enctype="multipart/form-data">
    <input type="file" name="fichero"><br>
    <input type="submit" value="Enviar archivo a servidor">
  </form>
</body>
</html>
```



### La matriz superglobal `$_FILES`

La gestión del archivo subido en el script PHP invocado desde el formulario se realiza mediante la matriz superglobal `$_FILES`. Esta es una matriz asociativa bidimensional que emplea dos claves: La primera hace referencia al valor del atributo *name* de la etiqueta `<input type='file'>`, y la segunda puede ser una de las siguientes claves:

- **'error'** = Devuelve un valor entero indicando el estado final de la transferencia.
- **'name'** = Devuelve el nombre del archivo enviado desde el cliente.
- **'size'** = Devuelve la longitud del archivo enviado expresada en bytes.
- **'type'** = Devuelve un identificador MIME del tipo del archivo enviado.
- **'tmp\_name'** = Devuelve el nombre temporal del archivo en la carpeta temporal del servidor.

La operación de subida de archivos es siempre delicada por lo que deben realizarse siempre varias comprobaciones para asegurar que la operación se ha completado correctamente.

### Obtención y almacenaje del archivo subido desde PHP

Lo primero es comprobar si existe la clave asociada al valor del atributo *name* del elemento `<input type='file'>` en la matriz superglobal `$_FILES`. En caso no de ser así, podemos suponer que el script NO ha sido llamado desde el formulario anterior:

```
// Comprobación de selección de archivo a subir.
if ( !isset($_FILES['fichero'])) exit();
```

A continuación debe comprobarse si existe algún error mediante el valor devuelto por la clave secundaria *"error"*. Los valores posibles están declarados como constantes predefinidas de PHP. Si el valor es 0 ( `UPLOAD_ERR_OK` ), la subida se ha realizado correctamente. Cualquier otro valor identifica un error.

## Programación Web en PHP

```
// Comprobación de estado de subida del archivo.
if ( $_FILES['fichero']['error'] != UPLOAD_ERR_OK ) {
    switch( $_FILES['fichero']['error'] ) {
        case UPLOAD_ERR_INI_SIZE:
            echo "El fichero excede directiva upload_max_filesize";
            break;
        case UPLOAD_ERR_FORM_SIZE:
            echo "El fichero excede el valor MAX_FILE_SIZE del formulario";
            break;
        case UPLOAD_ERR_PARTIAL:
            echo "El fichero se ha transferido parcialmente al servidor";
            break;
        case UPLOAD_ERR_NO_FILE:
            echo "No se ha subido ningún archivo";
            break;
        case UPLOAD_ERR_NO_TMP_DIR:
            echo "No existe la carpeta temporal de transferencia";
            break;
        case UPLOAD_ERR_INI_SIZE:
            echo "El fichero excede directiva upload_max_filesize";
            break;
        case UPLOAD_ERR_CANT_WRITE:
            echo "No hay permiso de escritura para transferir fichero";
            break;
        case UPLOAD_ERR_EXTENSION:
            echo "Una extension detuvo la subida del fichero";
            break;
    }
    exit();
};
```

Por último puede hacerse comprobaciones adicionales sobre el tipo y tamaño del archivo subido si se desea empleando para ello las claves “size” y “type”:

```
// Comprobación de tamaño -> No se admiten ficheros de más de 500Kbytes.
if ( $_FILES['fichero']['size'] > (0.5 * 1048576) ) {
    echo "No se permiten fichero de más de 500KB";
    exit();
}
// Comprobación de tipo -> Sólo se admiten imágenes de tipo JPG (MIME -> 'image/jpeg' )
IF ( $_FILES['fichero']['type'] != 'image/jpeg' ) {
    echo "Sólo se admiten imágenes en formato JPG";
    exit();
}
```

**(\*) El valor de la clave “type” de \$\_FILES indica el tipo MIME del fichero que se está subiendo. Este valor es una cadena que identifica el formato del archivo en función de su extensión. La lista completa de tipos MIME puede consultarse en:**

<http://www.iana.org/assignments/media-types/media-types.xhtml>

Cuando el archivo sube al servidor se copia en una carpeta temporal con un nombre aleatorio. Para completar la subida debe comprobarse primero que el archivo en la carpeta temporal es efectivamente un archivo subido mediante la función **is\_uploaded\_file()**:

```
bool is_uploaded_file ( string $filename )
```

La ruta del archivo en la carpeta temporal se obtiene a partir de la matriz superglobal **\$\_FILES** con la clave “tmp\_name”:

```
// El archivo en la carpeta temporal es el archivo recién subido???
if ( is_uploaded_file($_FILES['fichero']['tmp_name']) ) {
    // Comprobacion de archive subido correcta.
}
```



Por último debe transferirse el archivo subido a su carpeta final con su nombre correspondiente empleando la función ***move\_uploaded\_file()***. La función devuelve un valor lógico cierto si la operación se completa correctamente.

```
bool move_uploaded_file ( string $filename , string $destination)
```

El primer parámetro requiere el nombre temporal del archivo subido, y el segundo la ruta y nombre final del archivo. El nombre puede obtenerse de la matriz superglobal ***\$\_FILES*** con la clave ***"name"***. Si se desea almacenar el archivo en una carpeta situada en una posición relativa al archivo .php del script en ejecución; puede emplearse la función ***dirname(\_\_FILE\_\_)*** para obtener la ruta absoluta del script actual.

```
// El archivo en la carpeta temporal es el archivo recién subido???
if ( is_uploaded_file($_FILES['fichero']['tmp_name'])) {
    // Copia del archivo subido de la carpeta temporal a la definitiva
    // Ruta absoluta a carpeta definitiva del fichero subido.
    $abs_descargado = dirname(__FILE__)."/ficheros/".$_FILES['fichero']['name'];
    if ( !move_uploaded_file($_FILES['fichero']['tmp_name'], $abs_descargado)) {
        echo "Error moviendo fichero a carpeta destino";
        exit();
    }
} else {
    // Error en el completado de la subida.
    echo "Error subiendo archivo : ".$_FILES['fichero']['name'];
    exit();
}
```

### Ejercicio Práctico:

Se desea crear una aplicación en PHP compuesta de un formulario que permite seleccionar un archivo para subirlo al servidor y una página de PHP que muestra una imagen con el fichero recién subido con las siguientes características:

Se requiere que la imagen sea de tipo JPG y un tamaño no superior a 500 KBytes.

Una vez completada la subida, la imagen debe mostrarse al usuario en la página de envío ***"upload.php"***.

### Código formulario de subida:

```
<!DOCTYPE html>
<html>
<head>
<title>Selecciona una imagen para subirla al servidor</title>
</head>
<body>
    <form action="upload.php" method="POST" enctype="multipart/form-data">
        <input type="file" name="fichero"><br>
        <input type="submit" value="Enviar archivo a servidor">
    </form>
</body>
</html>
```

## Código página web PHP de subida y visualización de la imagen. (upload.php)

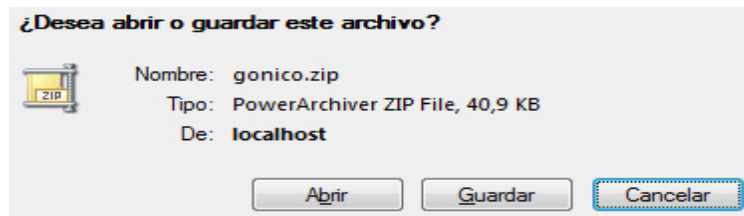
```
<?php
// Comprobación de seleccion de archivo a subir.
if ( !isset($_FILES['fichero'])) exit();
// Comprobación de estado de subida del archivo.
if ( $_FILES['fichero']['error'] != UPLOAD_ERR_OK) {
    switch( $_FILES['fichero']['error']) {
        case UPLOAD_ERR_INI_SIZE:
            echo "El fichero excede directiva upload_max_filesize";
            break;
        case UPLOAD_ERR_FORM_SIZE:
            echo "El fichero excede el valor MAX_FILE_SIZE del formulario";
            break;
        case UPLOAD_ERR_PARTIAL:
            echo "El fichero se ha transferido parcialmente al servidor";
            break;
        case UPLOAD_ERR_NO_FILE:
            echo "No se ha subido ningún archivo";
            break;
        case UPLOAD_ERR_NO_TMP_DIR:
            echo "No existe la carpeta temporal de transferencia";
            break;
        case UPLOAD_ERR_INI_SIZE:
            echo "El fichero excede directiva upload_max_filesize";
            break;
        case UPLOAD_ERR_CANT_WRITE:
            echo "No hay permiso para escritura en servidor denegado";
            break;
        case UPLOAD_ERR_EXTENSION:
            echo "Una extension detuvo la subida del fichero";
            break;
    }
    exit();
}
// Comprobación de tamaño ( 0.5 MBytes )
if ( $_FILES['fichero']['size'] > (0.5 * 1048576) ) {
    echo "No se permiten fichero de más de 500KB";
    exit();
}
// Comprobación de tipo ( MIME Type - "image/jpeg" )
if ( $_FILES['fichero']['type'] != 'image/jpeg' ) {
    echo "Sólo se admiten imágenes en formato JPG";
    exit();
}
// Ruta relativa al archivo subido -> empleada para mostrarla
$rel_descargado = "ficheros/".$_FILES['fichero']['name'];
// Ruta absoluta al archivo subido -> empleada para copiarlo completada la subida.
$abs_descargado = dirname(__FILE__)."/".$rel_descargado;
// Se ha completado la subida del archivo?
if ( is_uploaded_file($_FILES['fichero']['tmp_name'])) {
    // Copia del archivo subido de la carpeta temporal a la definitiva
    if ( !move_uploaded_file($_FILES['fichero']['tmp_name'], $abs_descargado)) {
        echo "Error moviendo fichero a carpeta destino";
        exit();
    }
} else {
    // Error en el completado de la subida.
    echo "Error subiendo archivo : ".$_FILES['fichero']['name'];
    exit();
}

?>
```

```
<!DOCTYPE html>
<html>
<head>
<title>Insert title here</title>
</head>
<body>
    <div>
        <img src='<?php echo $rel_descargado; ?>' />
    </div>
</body>
</html>
```

## Envío de archivos al usuario ( *Downloading* )

La transferencia de archivos del servidor al navegador del cliente se lleva a cabo automáticamente cuando el usuario hace clic sobre un enlace a un archivo diferente de una página HTML. Si el tipo del archivo es conocido por el navegador ( imagen, sonido, PDF..., etc ), éste puede mostrarlo directamente. Si el archivo no es conocido para el navegador ofrece la opción de descargarlo.



Ventana de solicitud de permiso de descarga de archivo por navegador

También es posible enviar al navegador del usuario un fichero desde un script empleando código PHP. Esto puede emplearse para permitir descargar al usuario un determinado archivo que no está en la carpeta de publicación del servidor, o hacerlo sin enviarle un vínculo a su URL.

Para ello debe modificarse la cabecera de la respuesta HTTP mediante el uso de la función **header()**. La llamada a la función debe realizarse obligatoriamente antes de enviar ningún valor al navegador del cliente.

En la cabecera deben indicarse los siguientes valores:

- **Tipo de fichero** → Este se indica mediante el parámetro de cabecera "**Content-type**" indicando el tipo MIME correspondiente. Por ejemplo; el tipo MIME: "*application/zip*" se corresponde con archivos comprimidos en ZIP, y los relaciona con la aplicación de descompresión instalada en el equipo del navegador. En el caso de un documento PDF el tipo MIME sería "*application/pdf*".
- **Indicador de fichero adjunto** → Esto indica que el fichero enviado forma parte del propio paquete de respuesta HTTP: Esto debe indicarse añadiendo el parámetro de cabecera "**Content\_disposition**" con el valor "*attachment*".
- **Nombre del fichero** → Este se indica mediante el parámetro de cabecera "**filename**" indicando el nombre del archivo.
- **Longitud de fichero** → Este se indica con el parámetro de cabecera "**Content-length**".

Para concluir debe incluirse el contenido del archivo cargándolo como parte de la respuesta HTTP. Para ello puede emplearse la función **readfile()**.

```
int readfile ( string $filename
               [, bool $use_include_path = false
               [, resource $context ]] )
```

Esta función lee al completo el archivo en la ubicación indicada como argumento y lo escribe como parte del paquete de respuesta para el usuario.

**Ejemplo:** El siguiente código envía al usuario el archivo “1.jpg” contenido en la carpeta “ficheros” mediante código:

```
<?php
$imagen = "1.jpg";
$ruta = 'ficheros/';

// tipo
header('Content-type: application/pdf');
// nombre
header('Content-disposition: attachment; filename="'.$imagen.'');
// tamaño
$longitud = filesize($ruta.$imagen);
header('Content-length: '.$longitud);

// archivo fuente.
readfile($ruta.$imagen);
?>
```

## Ejercicios

**1.-** Crea una aplicación web que solicite un usuario y contraseña, y compruebe la identificación consultando un archivo almacenado en el servidor Web. El archivo de claves debe llamarse psw.dat y debe almacenar en cada línea el nombre y contraseña de cada usuario siguiendo el siguiente formato:

```
<usuario1>,<clave1>  
<usuario2>,<clave2>  
<usuario3>,<clave3>  
<usuario4>,<clave4>
```

Si el usuario está registrado en el fichero se le da la bienvenida. En caso contrario se rechaza la petición indicando un error *HTTP403 – Forbidden*.

**2.-** Crea una aplicación

**3.-** Crea una aplicación web tipo FORO público donde cualquier persona puede acceder y ver los mensajes escritos.

Sólo los usuarios registrados en el archivo psw.dat generado anteriormente pueden publicar mensajes. Los mensajes deben almacenarse en un fichero foto.dat junto con el nombre del usuario que lo escribió, su imagen, y la fecha en que lo envió. Todos los mensajes publicados deben ser visibles

**3.-** Crea una aplicación Web para registro de votaciones. La aplicación debe constar de un formulario de entrada con los siguientes campos: *Nombre, Apellidos, DNI, Fecha nacimiento*, un grupo de tres botones de opción para *Candidato1, Candidato2, y Candidato3*; y un botón de envío.

Cada vez que un usuario acceda a la página y complete los datos de su voto, éstos deben almacenarse en un fichero de modo que puedan recuperarse más tarde.

**4.-** Complementa la aplicación con otra página que permite obtener los resultados de las votaciones del ejercicio anterior. Esta página debe requerir una autenticación para poder vista.

El nombre de usuario y contraseña deben ser *“alumno”* y *“cipsa”* respectivamente. La página debe mostrar en una tabla el nombre de cada uno de los candidatos incluyendo el porcentaje de votos obtenido.

**5.-** Crea una aplicación de gestión bancaria. La aplicación utiliza un pequeño fichero con la siguiente forma:

```
NCuenta_1, PIN1, Nombre_1, Apellido_1, Saldo1
NCuenta_2, PIN2, Nombre_2, Apellido_2, Saldo2
NCuenta_3, PIN3, Nombre_3, Apellido_3, Saldo3
NCuenta_4, PIN4, Nombre_4, Apellido_4, Saldo4
```

La aplicación debe constar de una página *index.php* inicial en la que el usuario pueda introducir su *número de cuenta* y el *PIN*.

Una vez que el usuario se autentifica correctamente accede a la página *gestión.php*. Esta página debe indicar al usuario la fecha del último acceso registrado mediante cookies. Esta página muestra al usuario su nombre, apellido, el saldo del que dispone y tres botones *ingreso* y *reintegro* ). Cada uno de los botones redirige al usuario a las página *ingreso.php* y *reintegro.php*.

La página *ingreso.php* se compone de un formulario compuesto de una caja de texto y un botón "Ingresar". Debe validarse que el valor introducido sea una cifra numérica mayor de cero e inferior a diez mil euros.

La página *reintegro.php* se compone de un formulario con una caja de texto y un botón de "reintegro". Debe validarse que el valor introducido sea una cifra numérica válida mayor que cero e inferior o igual al saldo del usuario.

En ambos casos, el usuario debe ser redirigido a la página de gestión donde debe mostrarse actualizado su saldo.

**6.-** Modifica la aplicación anterior para que se registren todos los accesos realizados contra la aplicación Web. La aplicación debe almacenar en una carpeta *logs* situada en el servidor un archivo por cada día con el nombre *logDDMMMAA.log* (donde "DD" se corresponde con el día, MM con el mes, y AA con el año de la fecha).

Por cada acceso a la aplicación debe generarse una línea en el correspondiente archivo almacenando el nombre del usuario, la hora y la IP de su ordenador siguiendo el siguiente formato: *<usuario>,<hora>,<ip>*

Ej:

```
roger, 20:35, [192.34.55.43]
```

## Práctica

**1.-** Crea una aplicación web que permita registrar el nombre, contraseña y foto de perfil de un usuario en un fichero psw.dat. Para ello crea las siguientes páginas:

Una página de registro de usuarios ( *registro.php* ) provista de un formulario con los siguientes campos:

- Una caja de texto para indicar el nombre de usuario.
- Dos cajas de texto tipo contraseña para indicar la misma contraseña dos veces.
- Un selector de archivo para poder subir la foto de perfil asociada al usuario.

Comprobaciones:

- Debe indicarse un nombre de usuario.
- Debe indicarse una contraseña y debe ser la misma en las dos cajas de texto de contraseña.
- Debe indicarse una imagen como foto de perfil. La imagen debe ser formato JPG de un tamaño no superior a 500Kb.
- Tanto el nombre de usuario como la contraseña no debe contener comas.
- Ante cualquier error en estas validaciones debe volver a mostrarse el formulario indicando el error correspondiente.

Los datos deben almacenarse del siguiente modo:

- Los datos del usuario y su contraseña deben almacenarse en un fichero "*usuarios.dat*" dentro de la carpeta *perfiles* con el siguiente formato.

```
usuario,contraseña  
usuario,contraseña  
usuario,contraseña
```

- Las imágenes de perfil de los usuarios deben almacenarse en la misma carpeta con el nombre del usuario correspondiente.

**2.-** Crea una aplicación de tipo FORO que permita a cualquier persona ver los mensajes publicados, pero solo los usuarios registrados pueden publicar nuevos mensajes. La aplicación debe constar de las siguientes páginas.

Una página principal ( *foro.php* ) → Que muestra todos los mensajes publicados indicando por cada mensaje:

- Foto de perfil y nombre del usuario que publicó el mensaje.
- Fecha de publicación del mensaje
- Contenido del mensaje.

Página de publicación ( *publicación.php* ) → Debe constar de un formulario con los siguientes elementos:

- Una caja de texto para el nombre del usuario
- Una caja de texto para la contraseña del usuario
- Una caja de texto multi-línea para el contenido del mensaje.

Comprobaciones:

- El campo nombre de usuario y contraseña no deben estar vacías y deben corresponderse con alguno de los usuarios registrados en el fichero “usuarios.dat” creado anteriormente.
- El campo contenido no debe estar vacío.
- El mensaje no puede contener caracteres “<” o “>”. En su defecto, estos deben ser eliminados antes del guardado del mensaje en el fichero.

Los datos de los mensajes publicados deben almacenarse del siguiente modo en un fichero “*mensajes.dat*” dentro de la carpeta “*foro*”.

Por cada mensaje se indica una línea inicial con la etiqueta “<mensaje>”. Las siguientes dos líneas indican el nombre del usuario y la fecha de publicación del mensaje respectivamente. Las siguientes líneas corresponden con el mensaje indicado. Tras el mensaje se añade una línea </mensaje>.

```
<mensaje>
Usuario
Fecha
Texto mensaje
Texto mensaje
Texto mensaje
</mensaje>
<mensaje>
...
</mensaje>
```

Cada vez que se publica un nuevo mensaje debe añadirse al final del fichero.