UNIDAD 4. MANEJO DE FICHEROS

Heike Bonilla Redondo
Fecha: 20-10-2015

Índice

Los ficheros .php que acompañan estos apuntes son más completos, con las comprobaciones y comentarios necesarios, que los que aparecen en el texto.

¡¡Te recomiendo que analices el código de los ejemplos en código!!

OPERACIONES BÁSICAS DE FICHEROS

En esta unidad veremos cómo podemos usar *PHP* para acceder al sistema de archivos del servidor para poder leer o guardar datos.

1.- Abrir

Para acceder a un archivo primero es necesario abrirlo. Para ello usaremos la función *fopen() que* tiene dos argumentos, el nombre del archivo a acceder y el modo de acceder a este. La función devuelve un **puntero** en el archivo si es satisfactoria o **cero** si no lo es. Los archivos se abren para realizar operaciones de lectura o escritura.

```
fopen(ruta al archivo, modo de acceso)
Por ejemplo:
```

```
$file = fopen("miarchivo.txt", "r");
```

Es frecuente comprobar con la siguiente expresión si la apertura se ha realizado con éxito:

```
if (!$file = fopen("miarchivo.txt", "r")){

    echo "No se ha podido abrir el archivo";

}
```

Se puede abrir un archivo pero también una **URL externa**, ya que fopen() realmente lo que hace es crear una conexión, por eso hay que cerrarla posteriormente.

PHP ofrece los siguientes modos de acceso

Modo	Descripción
'r'	Apertura para sólo lectura; coloca el puntero al fichero al principio del fichero.
'r+'	Apertura para lectura y escritura; coloca el puntero al fichero al principio del fichero.
'w'	Apertura para sólo escritura; coloca el puntero al fichero al principio del fichero y trunca el fichero a longitud cero. Si el fichero no existe se intenta crear.
'w+'	Apertura para lectura y escritura; coloca el puntero al fichero al principio del fichero y trunca el fichero a longitud cero. Si el fichero no existe se intenta crear.
'a'	Apertura para sólo escritura; coloca el puntero del fichero al final del mismo. Si el fichero no existe, se intenta crear.
'a+'	Apertura para lectura y escritura; coloca el puntero del fichero al final del mismo. Si el fichero no existe, se intenta crear.
'x'	Creación y apertura para sólo escritura. Puntero al principio del archivo. Si el

archivo ya existe dará error E_WARNING. Si no existe se intenta crear.

'x+' Creación y apertura para lectura y escritura. Mismo comportamiento que x.

Se llama apuntador a la posición del archivo en la que leemos o escribimos. Como podemos ver lo más habitual es que lo situemos al principio para leer todo su contenido o al final para ir añadiendo datos.

IMPORTANTE

- Si el archivo no tiene permiso de ecritura, abrirlo con *r+* fallará, incluso cuando sólo se intenta leer.
- *w* y *w+* eliminarán el contenido de cualquier archivo. Para sólo añadir y no borrar, se usa a y *a+*.
- Si quieres crear nuevos archivos y evitar sobreescribir sin querer un archivo existente, utiliza *x* o *x+*.
- Cuando se trabaja con **archivos binarios**, como imágenes, hay que añadir 'b' después del modo. Como *rb* o *r+b*

Puntero de archivo

Un **puntero de archivo** es una variable que hace referencia a un archivo. Es una variable que apunta a un archivo en concreto, y normalmente se obtiene cuando se abre con *fopen()*. Además de apuntar a un archivo, apunta a una posición concreta en ese archivo, dependerá del modo de apertura y de los movimientos realizados.

El puntero se va moviendo a lo largo del fichero según las operaciones que hacemos. Cada operación de lectura o escritura desde la posición que tenga el puntero en ese momento.

PHP cierra todos los punteros de archivos al final de la ejecución del script, aunque se considera una buena práctica cerrar los archivos manualmente con *fclose.*

La función *feof()* es utilizada con frecuencia en el manejo de archivos en PHP. Esta función comprueba si el puntero se encuentra al final del archivo. Se utiliza cuando se recorre un archivo línea por línea o para la lectura de grandes archivos, mediante un condicional.

2.- Cerrar

Mientras hacemos operaciones con el archivo lo debemos mantener abierto, pero al terminar de trabajar con él hay que cerrarlo para que el sistema operativo pueda disponer de él, ya que mientras está abierto el sistema operativo lo bloquea para que otros programas no puedan escribir y destruir mutuamente lo que escriben.

Para cerrar un archivo abierto se usa la función *fclose()* pasándole como parámetro la variable que contiene el manejador del archivo.

```
fclose(archivo)
```

Ejemplo En este código se abre un fichero en modo de lectura. En función de si se ha conseguido, se muestra un mensaje de confirmación o de error. Finalmente el archivo se cierra.

```php
<?php
$ruta = "utils.php";
if ($archivo=fopen($ruta, "r")) {
print "Archivo $ruta abierto para lectura.";
} else {
print "No se pudo abrir el archivo: $ruta.";
```

```
}
fclose($archivo);
?>
```

Ejemplo El fichero que abrimos lo podemos localizar mediante una ruta absoluta o relativa. Aunque en *Windows* las rutas se construyan usando la contrabarra " \ ", *PHP* admite que se separen los directorios con la barra normal " / " como en *Linux*. Esta última es la forma que elegiremos ya que las rutas relativas que construyamos de esta forma serán válidas tanto si instalamos nuestra aplicación en *Linux* como en *Windows*.

```
<?php
$ruta_absoluta = "c:/CursoPHP/htdocs/index.php";
$ruta_relativa = "../practicasPHP/config.php";
$archivo1 = fopen($ruta_absoluta, "r");
$archivo2 = fopen($ruta_relativa, "r");
fclose($archivo1);
fclose($archivo2);
?>
```

Ejemplo. Es incluso posible abrir archivos que estén alojados en otros servidores, aunque lo más habitual es que solo tengamos permisos de lectura.

```
<?php
$url = "http://www.google.es/index.html";
$archivo = fopen($url, "r");
fclose($archivo);
?>
```

3.- Lectura

Lectura básica con fread()

Lo más habitual es que queramos leer un archivo. Una de las formas más sencillas es mediante la función *fread()* que lee un número de bytes de un archivo. En conjunción con la función *filesize()* que nos devuelve el tamaño del archivo en bytes se puede usar para leer todo el archivo. Normalmente el máximo coincide con el tamaño del cluster, lo más habitual 8KB (8182 bytes, podemos comprobarlo utilizando Chkdsk.

```
fread(archivo, tamaño)
```

Ejemplo Lectura del archivo "*prueba.txt*". Para que el archivo funcione tendremos que haberlo creado en la misma carpeta que este script, con el contenido que deseemos.

```
<?php
If($archivo = fopen("prueba.txt", "r")){
      $tamano = filesize("prueba.txt");
If($texto = fread($archivo, $tamano))
      echo $texto;
fclose($archivo);
}
?>
```

Lectura con fgets

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

Obtiene una línea desde el **puntero de un archivo**. La **lectura** termina cuando se llegue a *$length*, se llegue a una nueva línea o se alcance el final del archivo.

Ejemplo:

```
$fp = fopen("miarchivo.txt", "r");
while (!feof($fp)){
    $linea = fgets($fp);
    echo $linea;
}
fclose($fp);
```

Lectura con file_get_contents

Devuelve un archivo entero a una cadena, se puede especificar el comienzo desde *$offset* hasta *$maxlen*.

Si falla devuelve false, pero genera un E*WARNING si* $filename *no se pudo encontrar*, $maxlen *es menor que cero*.

```
// Devuelve el contenido de una web
$web = file_get_contents("http://www.ejemplo.com");
echo $web;
// Devuelve contenido de un archivo en el directorio actual
$contenido = file_get_contents("miarchivo.txt");
echo $contenido;
```

4.- Escritura

Escritura básica con fwrite()

Otra acción que querremos hacer habitualmente es añadir datos a un archivo. Para ello se usa una función muy sencilla *fwrite() o fputs()*, que escribe en la posición en la que está el apuntador. Por lo general, si hemos abierto el archivo en modo "a" escribiremos al final del archivo.

La función *fwrite()* usa dos parámetros. El primero el puntero del archivo y el segundo la cadena que queremos escribir.

```
fwrite(archivo, cadena de texto)
```

Ejemplo. En este ejemplo se usa esta función dos veces para escribir dos frases en el archivo .

```
<?php
$archivo = fopen("refranes.txt", "a");
fwrite($archivo, "Si las barbas de tu vecino ves cortar
...\r\n");
fwrite($archivo, "...pon las tuyas a remojar".PHP_EOL);
fclose($archivo);
?>
```

La cadena " \r\n " sirve para insertar un salto de línea en el archivo en el sistema operativo *Windows*. En *Linux* se usa solo la cadena " \n ".

Si lo que queremos es **sobrescribirlo** abriremos el archivo en modo **"w"**, de tal forma que se borre el contenido al abrirlo y dispondremos de un fichero vacío.

Constante PHP_EOL

PHP dispone de una constante de sistema que nos permite insertar el código de fin de línea de manera más cómoda.

Esta constante tiene el valor correspondiente al salto de línea dependiendo el SO del servidor.

file_put_contents

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

Escribe información *$data* en un archivo *$filename*. Viene a ser equivalente a hacer *fopen()*, *fwrite()* y *fclose()* a la vez. Si el archivo *$filename* no existe, se crea, a no ser que la bandera FILE_APPEND esté establecida. Las banderas pueden ser:

FILE_USE_INCLUDE_PATH. Busca *$filename* en el directorio _include*path*.

FILE_APPEND. Si el archivo *$filename* ya existe, añade la información en el mismo en lugar de sobreescribirlo.

LOCK_EX. Adquiere **acceso exclusivo** mientras se ejecuta la escritura. Es como llamar a *flock()* entre la llamada a **fopen()** y **fwrite()**.

La función devuelve el **número de bytes** escritos en el archivo *$filename*.

```
// Uso junto con file_get_contents y añadiendo al ya existente:
$archivo = "miarchivo.txt";
$actual = file_get_contents($archivo);
$actual .= "Esto añade más texto al archivo\n";
file_put_contents($archivo, $actual);
// Podemos utilizar también la bandera FILE_APPEND para añadir al final sin eliminar:
$archivo = "miarchivo.txt";
$mascontenido = "¡Todavía más contenido!\n";
// También vamos a emplear la bandera LOCK_EX para evitar cualquier modificación mientras:
file_put_contents($archivo, $mascontenido, FILE_APPEND | LOCK_EX);
```

5.- Crear un fichero

Si queremos crear un archivo bastará que lo abramos con el modo "a" o "w". Recuerda que hay diferencia entre los dos modos.

- a crea el fichero sino existe y coloca el puntero al final si existe.
- w abre el fichero al inicio, de manera que borra el contenido.

6.- Eliminar un fichero.

La función *unlink()* que recibe como parámetro una ruta a un fichero lo borra. En el caso de que no lo haya conseguido, por no tener permisos o sencillamente porque no existe el archivo, devuelve FALSE.

```
unlink(archivo)
```

Ejemplo Este programa intenta borrar un archivo y en el caso de no conseguirlo muestra un mensaje de error.

```
<?php
if (!unlink("refranes.txt")) {
echo "No se ha podido borrar el archivo.";
}
?>
```

7.- Rebobinar fichero con rewind

La función rewind() mueve el puntero de un archivo al principio del mismo.

Otras funciones para manejar ficheros

Estas son algunas de las funciones que pueden ayudarnos en el manejo de ficheros, algunas ya las conocemos te la unidad anterior.

file_exists().

Devuelve TRUE si el archivo por el que preguntamos existe.

file_size().

Ya ha aparecido al hablar de la lectura de un archivo. Devuelve el tamaño en bytes del mismo.

is_file().

Devuelve TRUE si el archivo es un fichero.

is_dir().

Devuelve TRUE si el archivo es un directorio.

is_readable().

Devuelve TRUE si el archivo se puede abrir para lectura.

is_writeable().

Devuelve TRUE si el archivo se puede abrir para escritura.

Ejemplo Mediante este programa se muestran las propiedades del archivo "prueba.txt".

```
<?php
$ruta = "prueba.txt";
if (file_exists($ruta)) {
      echo "$ruta tiene un tamaño de ";
      echo filesize($ruta) . " bytes.<br>";
if (is_file($ruta)) {
      echo "$ruta es un fichero.<br>";
}
if (is_dir($ruta)) {
      echo "$ruta es un directorio.<br>";
}
if (is_readable($ruta)) {
```

```
      echo "$ruta se puede abrir para lectura.<br>";
}
if (is_writable($ruta)) {
      echo "$ruta se puede abrir para escritura.<br>";
}
} else {
      echo "$ruta no existe.";
}
?>
```

NL2BR () La función nl2br convierte los saltos de línea a etiquetas
 con lo que podremos ver esos saltos de línea en cualquier navegador.

Funciones de directorios

Las funciones de directorios vienen de la extensión directories de **PHP**. Hay un total de 9 funciones disponibles:

- chdir

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

Cambia el directorio actual al directorio *$directory*.

- getcwd

```
string getcwd ( void )
```

Obtiene el directorio actual.

```
echo getcwd() . "<br>"; // Directorio: /directorio/actual

chdir('nuevo/directorio');

echo getcwd() . "<br>"; // Directorio:
/directorio/actual/nuevo/directorio
```

- scandir

```
array_scandir (string $directory [, int $sorting_order =
SCANDIR_SORT_ASCENDING [, resource $context ]] )
```

Devuelve un *array* con los archivos y directorios que se encuentran en *$directory*. El _$sorting*order* indica el orden en que devolverá el listado: **SCANDIR_SORT_ASCENDING (0), SCANDIR_SORT_DESCENDING (1) o SCANDIR_SORT_NONE** (sin orden):

```
$directorio = "Slimvistas;
$archivos = scandir($directorio, 1);
print_r($archivos);
/*
Array
(
    [0] => View.php
    [1] => Slim.php
    [2] => Router.php
    [3] => Middleware
    [4] => LogWriter.php
```

```
    [5] => Log.php
    [6] => Http
    [7] => Helper
    [8] => Environment.php
    [9] => ..
    [10] => .
)
 */
```

- opendir

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

Abre un **gestor de directorio** para ser usado en llamadas posteriores como *closedir()*, *readdir()* y *rewinddir()*.

- readdir

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

Lee una entrada desde un gestor de directorio. _$dir*handle* es el gestor de directorio previamente abierto por *opendir().* Si no se especifica se usa la última conexión abierta por *opendir()*. Devuelve el nombre de la siguiente entrada del directorio.

- closedir

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

Cierra un gestor de directorio abierto _$dir*handle*. Si no se especifica se asumirá la última conexión abierta por *opendir()*.

Ejemplo de las funciones *opendir*, *readdir* y *closedir*:

```
if ($gestor = opendir('Slim')) {
    echo "Gestor de directorio: $gestor <br>"";
    echo "Entradas <br>"";
    // Iteramos sobre el directorio:
    while (false !== ($entrada = readdir($gestor))) {
        echo "$entrada <br>"";
    }
    closedir($gestor);
}
// Devuelve todos los archivos del directorio especificado
```

- rewinddir

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

Restablece la secuencia de directorio indicada por _$dir*handle* al comienzo del directorio. De nuevo, si no se especifica el gestor, se asumirá la última conexión abierta por *opendir()*.

AMPLIACIÓN. OTRAS FUNCIONES DE LECTURA

readfile

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

Es una versión más simplificada de _file_get*contents()*. **Lee un archivo y lo escribe en el búfer de salida.** La función devuelve el número de **bytes leídos del archivo**.

```
// Devuelve el contenido de un archivo
readfile("miarchivo.txt");
// Devuelve el contenido y el número de bytes, 79
$contenido = readfile("miarchivo.txt");
echo $contenido;
```