Abrir, leer, escribir y cerrar un archivo

Abrir un archivo

Los archivos en PHP se abren con la función *fopen()*, que requiere dos parámetros: el archivo que se quiere abrir y el modo en el que abrir el archivo. 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.

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

Si no es posible abrir el archivo, devuelve cero, por eso es frecuente utilizar esta función en una condición:

```
if (!$fp = 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.

```
$fp = fopen("http://localhost:8000", "r");
```

Los modos de acceso existentes para fopen son:

Modo	Descripción
r	Apertura para lectura. Puntero al principio del archivo
r+	Apertura para lectura y escritura. Puntero al principio del archivo
w	Apertura para escritura. Puntero al principio del archivo y lo sobreescribe. Si no existe se intenta crear.
w+	Apertura para lectura y escritura. Puntero al principio del archivo y lo sobreescribe. Si no existe se intenta crear.
a	Apertura para escritura. Puntero al final del archivo. Si no existe se intenta crear.

a+	Apertura para lectura y escritura. Puntero al final del archivo. Si 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.
c	Apertura para escritura. Si no existe se crea. Si existe no se sobreescribe ni da ningún error. Puntero al principio del archivo.
c+	Apertura para lectura y escritura. Mismo comportamiento que C.

Algunas **consideraciones** de la tabla:

- Si el archivo no es escribible, 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*

Leer un archivo

Una vez abierto el archivo, vamos a leerlo y guardar sus contenidos en una variable con *fread()*:

```
$file = "miarchivo.txt";

$fp = fopen($file, "r");

$contents = fread($fp, filesize($file));
```

La variable *$contents* guardará el contenido que obtengamos con la función *fread()*. Esta función requiere dos parámetros, el archivo abierto y la longitud que queremos leer de dicho archivo (en bytes). En este caso hemos empleado la función *filesize()* para obtener el tamaño del archivo y así devolver todo su contenido.

Cerrar un archivo

Finalmente, vamos a cerrar el archivo (no es obligatorio pero se recomienda):

```
fclose($fp);
```

Escribir en un archivo

Al igual que para **leer un archivo**, hay más de una forma de **escribir** en uno. La forma más básica es utilizar la función *fwrite()* (o *fputs()*, que es su alias):

```
$file = "miarchivo.txt";

$texto = "Hola que tal";

$fp = fopen($file, "w");

fwrite($fp, $texto);

fclose($fp);
```

Esta vez hemos empleado el **modo w**, que permite escribir sobreescribiendo el archivo.

Podemos **limitar la longitud de datos que queremos escribir** (todos los datos que había en el archivo se borrarán por completo igualmente):

```
$file = "miarchivo.txt";

$texto = "Hola que tal";

$fp = fopen($file, "w");

fwrite($fp, $texto, 4); // Escribirá sólo: Hola
```

Si el archivo *miarchivo.txt* no existe, se creará automáticamente con el **modo w de la función fopen**.

2. Puntero de archivo

Un **puntero de archivo** (*file pointer* o *handle*) 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()*.

PHP y su recolección de basuras 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()*.

Además de apuntar a un archivo, apunta a una **posición concreta en ese archivo**. En la mayoría de los casos cuando se abre un archivo el puntero apunta al principio (posición 0) o al final del archivo.

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:

```
$archivo = "miarchivo.txt";

// Abrimos el archivo

$fp = fopen($archivo, "r");

// Loop que parará al final del archivo, cuando feof sea true:

while(!feof($fp)){

   echo fread($fp, 4092);

}
```

El código anterior sólo cargará 4kb de datos de vez, lo que reduce el uso de memoria para grandes archivos.

Seeking es mover el puntero de un archivo. Para mover el puntero se puede usar la función *fseek()*, y para encontrar la posición de un puntero dado *ftell()*. Vamos a ver algunos ejemplos:

```
$file = "miarchivo.txt";

$texto = "Hola que tal 12345";

$fp = fopen($file, "w");

fwrite($fp, $texto);

fclose($fp);
```

Ahora abriremos el archivo para ir viendo *fseek* y *ftell*:

```
$fp = fopen($file, "r");

// Si lo hemos abierto con r, siempre empieza desde el principio:

echo ftell($fp) . "<br>"; // Devuelve 0

// Colocamos el apuntador en la posicion 10:

fseek($fp, 10);

// Mostramos la posición actual:

echo ftell($fp) . "<br>"; // Devuelve 10
```

```
// Se puede indicar una posición sin contenido:
fseek($fp, 1000);
echo ftell($fp) . "<br>"; // Devuelve 1000
// Para ir al final del archivo se emplea un tercer argumento en fseek:
fseek($fp, 0, SEEK_END);
echo ftell($fp) . "<br>"; // Devuelve 18
// Para mover el apuntador a una posición relativa se emplea SEEK_CUR:
fseek($fp, -5, SEEK_CUR);
echo ftell($fp) . "<br>"; // Devuelve 13
```

No es necesario emplear *fseek()* para mover el puntero, también se puede hacer cuando se lee o se escribe un archivo:

```
// Cambiar el puntero leyendo un archivo:
$file = "miarchivo.txt";
$fp = fopen($file, "r");
// Leemos 10 bytes
$datos = fread($fp, 10);
echo ftell($fp); // Devuelve 10
// Cambiar el puntero escribiendo un archivo:
$file = "miarchivo.txt";
$texto = "12345";
$fp = fopen($file, "w");
// Escribimos los 5 bytes del texto:
fwrite($fp, $texto);
echo ftell($fp); // Devuelve 5
```

3. Obtener información de un archivo

Se puede **obtener información de un archivo** además de su **contenido**: tamaño, última vez que se ha accedido o modificado, número de links, etc. La función principal para obtener esta información es con la función *stat()*, en esta tabla se pueden ver los 12 elementos que devuelve el *array*.

```
$file = "miarchivo.txt";
$texto = "Todos somos muy ignorantes, lo que ocurre es que no todos ignoramos las mismas cosas.";
```

```
$fp = fopen($file, "w");
fwrite($fp, $texto);


$datos = stat($file);


echo $datos[3] . "<br>"; // Número de enlaces, 1
echo $datos[7] . "<br>"; // Tamaño en bytes, 85
echo $datos[8] . "<br>"; // Momento de último acceso, 1444138104
echo $datos[9] . "<br>"; // Momento de última modificación, 1444138251
```

La extensión fileinfo de **PHP** ofrece también una serie de **funciones para obtener información de archivos**. Primero se crea un *recurso* con la función **_finfo_open() *o la* clase finfo()**_, con la configuración que se especifique, y después se utiliza finfo_file() o finfo::file() para obtener la información:

- finfo_open

```
resource finfo::__construct ([int $options = FILEINFO_NONE [, string $magic_file = NULL ]])
```

Crea un **recurso fileinfo**, abre una base de datos mágica y la devuelve a su recurso. _$magic*file* es opcional, es el archivo de la base de datos mágica. El primer argumento, *$options*, es una **constante de fileinfo**, también opcional. Puedes obtener la lista entera de constantes fileinfo aquí.

- finfo_file

```
string finfo::file ( resource $finfo, string $file_name = null [, int $options = FILEINFO_NONE [, resource $context = null ]] )
```

El argumento *$finfo* es el recurso que se ha creado con _finfo*open*. _$file*name* el archivo del que se quiere obtener información.

```
$file = "miarchivo.txt";
$finfo = new finfo(FILEINFO_MIME); // Devuelve el tipo mime
echo $finfo->file($file); // Devuelve: text/plain; charset=us-ascii
```

Para **cerrar un recurso fileinfo abierto** se utiliza _finfo_close()_.

4. 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() . "\n"; // Directorio: /directorio/actual
chdir('nuevo/directorio');
echo getcwd() . "\n"; // 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 = "Slim";
$archivos = scandir($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] => .
)
 */
```

- chroot

bool chroot (string $directory)

Cambia el directorio raíz al directorio *$directory*. Requiere privilegios de administrador. Es necesario tener en cuenta que el directorio que se señale ha de estar preparado incluyendo los archivos necesarios para ser un directorio root.

- 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: $gestor\n";
    echo "Entradas:\n";
    // Iteramos sobre el directorio:
    while (false !== ($entrada = readdir($gestor))) {
        echo "$entrada\n";
    }
    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()*.