PROGRAMACIÓN WEB EN ENTORNO SERVIDOR UD-9

FICHEROS

ÍNDICE

[APERTURA DE FICHEROS 3](#_Toc159853973)

[ESCRITURA EN FICHEROS 4](#_Toc159853974)

[LECTURA DE FICHEROS 5](#_Toc159853975)

[LECTURA DE FICHEROS COMPLETOS 6](#_Toc159853976)

[DESPLAZAMIENTO EN FICHEROS 7](#_Toc159853977)

[FUNCIONES ÚTILES CON FICHEROS 7](#_Toc159853978)

[SUBIENDO ARCHIVOS 8](#_Toc159853979)

[FUNCIONES DE DESPLAZAMIENTO 9](#_Toc159853980)

# APERTURA DE FICHEROS

Cuando usemos un fichero siempre vamos a seguir los siguientes pasos: apertura, lectura/escritura y cierre.

Al abrir un archivo hay que indicar tres características:

* El uso que se le va a dar (lectura, lecto-escritura o escritura).
* En el caso de que el archivo ya exista si se quiere sobreescribir, añadir nuevos datos al final o no hacer nada.
* Si se trata de un archivo binario o de un archivo de texto.

Para abrir un fichero usaremos la función fopen(‘nombreFichero’, ‘modo’[, blnIncludePath[, flujo]]) que devuelve un flujo de lectura o escritura.

Los modos pueden ser:

* r (read) 🡪 Lectura desde el principio.
* w (write) 🡪 Escritura con borrado.
* a (append) 🡪 Adjuntar al final del archivo.
* x 🡪 Escritura en archivo nuevo. Si el archivo ya existe avisará de ello.

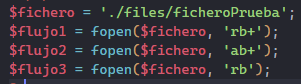
Se puede usar cualquiera de estos con un + que añadiría la opción contraria, es decir:

* r+ 🡪 Lectura y escritura desde el principio.
* w+ 🡪 Escritura y lectura.
* a+ 🡪 Escritura y lectura desde el final del archivo.
* x+ 🡪 Escritura y lectura en archivo nuevo.

Como opciones de archivo se puede usar:

* b 🡪 binario.
* t 🡪 texto (no es recomendable y sólo funciona en Windows).

Por ejemplo:

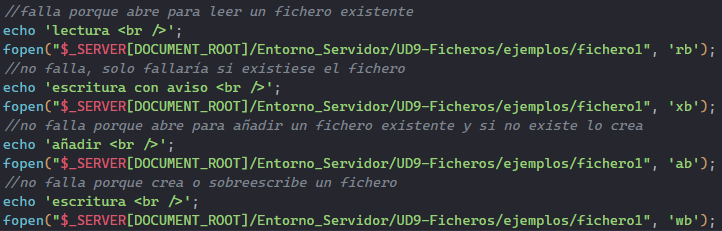


En el primer caso se creará un flujo para lectura y escritura desde el inicio del fichero.

En el segundo caso para añadir y leer desde el final del fichero.

En el último se genera un flujo de sólo lectura.

Otro ejemplo:



En la lectura del documento al abrirlo por primera vez lanzará un warning porque el archivo no existirá el resto de fopen() sí se lanzarán:

Texto

Descripción generada automáticamente con confianza media

En posteriores ejecuciones del archivo php el error se producirá en la escritura con aviso (la x) avisándonos de que ese archivo ya existe.

Texto

Descripción generada automáticamente con confianza baja

# ESCRITURA EN FICHEROS

Para escribir en un fichero usamos la función fwrite(flujo, cadena[, longitud]).

El parámetro longitud por defecto vale strlen(cadena) y puede ser recomendable indicarlo para garantizar la contabilidad entre plataformas.

Texto

Descripción generada automáticamente

Si ahora abrimos el archivo:



Cuando no se vaya a utilizar más un flujo se debe cerrar con la función fclose(flujo).



# LECTURA DE FICHEROS

Para leer de un fichero y saber si tenemos más datos que leer o estamos al final del fichero usaremos la función feof(flujo).

* **fgetc(flujo)**: Lee un carácter del archivo y se mueve al siguiente.

Texto

Descripción generada automáticamente

Imagen que contiene Patrón de fondo

Descripción generada automáticamente

* **fgets(flujo)**: Lee una línea del archivo y se mueve a la siguiente.

Se modifica el archivo a este contenido:

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Patrón de fondo

Descripción generada automáticamente con confianza baja

* **fgetss(flujo, longitud[, indicadores]):** Lee la longitud indicada, eliminando las etiquetas HTML y PHP que encuentre.
* **fgetcsv(flujo[, longitud[, delimitador[, contenedor[, escape]]]])**: devuelve una matriz con los campos de un fichero. La longitud es de la línea más larga y si se pone a 0, indica que se deja cualquier longitud, pero funciona más lento. Delimitador indica el carácter que sirve para separar los campos. Contenedor es el carácter con el que están rodeados los campos.

Texto

Descripción generada automáticamente

* **fread(flujo, nºBytes):** Lee el número de bytes indicados en el flujo que se le pasa.

# LECTURA DE FICHEROS COMPLETOS

Para leer de un fichero y saber si tenemos más datos que leer o estamos al final del fichero, usaremos la función feof(flujo):

* **readFile((‘fichero’[, blnIncludePath[, flujo]])**: lee el fichero y lo muestra tal cual.

Texto

Descripción generada automáticamente

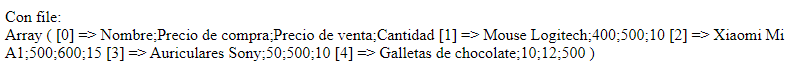
Texto

Descripción generada automáticamente

* **file((‘fichero’[, blnIncludePath[, flujo]])**: lee el fichero y lo deja en un array, guardando cada fila en una posición del array.

Texto

Descripción generada automáticamente



* **file\_get\_contents((‘fichero’[, blnIncludePath[, flujo]])**: lee el fichero y devuelve una cadena con su contenido.

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

# DESPLAZAMIENTO EN FICHEROS

* **rewind(flujo)**: se coloca al principio del fichero apuntado por el flujo.
* **ftell(flujo)**: indica en que posición estamos dentro del fichero apuntado por el flujo.
* **fseek(flujo, desplazamiento[, puntoDePartida])**: se desplaza lo indicado en desplazamiento desde el puntoDePartida que puede tener los valores:
  + SEEK\_SET inicio (por defecto)
  + SEEK\_CUR ubicación actual
  + SEEK\_END final del fichero

Texto

Descripción generada automáticamente



# FUNCIONES ÚTILES CON FICHEROS

* **file\_exists(‘fichero’)**: indica si existe un fichero.
* **file\_size(‘fichero’):** indica el tamaño en bytes de un fichero-
* **unlink(‘fichero’)**: elimina un fichero.
* **flock(flujo, operación[, blnQuedaEspera])**: bloquea un fichero por medio del SO. Las operaciones pueden ser:
  + LOCK\_SH bloqueo compartido (lectura)
  + LOCK\_EX bloqueo exclusivo (escritura)
  + LOCK\_UN para liberar un bloqueo.

# SUBIENDO ARCHIVOS

Un fichero es un elemento en binario que no es ni un número o ni un texto: imagen, vídeo, música, doc, iso… Necesitaremos usar siempre el method POST y añadir enctype=”multipart/form-data”. Por último, usar el input de tipo archivo (file). Los archivos subidos almacenan en el servidor en el array $\_FILES con el nombre del campo del tipo file del formulario.

Una captura de pantalla de un celular con texto e imagen

Descripción generada automáticamente con confianza media

Cada archivo cargado en $\_FILES tiene:

* name: nombre
* tmp\_name: ruta temporal
* size: tamaño en bytes
* type: tipo MIME
* error: si hay error, contiene un mensaje. Si ok → 0.Tabla

  Descripción generada automáticamente

Texto

Descripción generada automáticamente

Se puede subir más de un archivo generando un array de los campos de entrada. Tendremos que comprobar que existen los distintos elementos del array para ver que se ha movido.

Texto

Descripción generada automáticamente

# FUNCIONES DE DESPLAZAMIENTO

* scandir(‘directorio’[, orden[, flujo]]) lista el contenido de un directorio, el orden puede ser 0, por defecto, ascendente o 1 descendente
* is\_dir(‘directorio’) indica si existe un directorio con el nombre dado
* is\_file(‘directorio’) indica si existe un archivo con el nombre dado
* chdir(‘directorio’) cambia de directorio al indicado. Funciona igual que el comando cd del SO
* getcwd() devuelve el directorio actual
* chmod(‘fichero’, ‘modo’) cambia el modo del fichero. Funciona igual que el comando chmod del SO
* dirname(‘ruta’) devuelve un string con la ruta del directorio padre de la ruta pasada por parámetro.