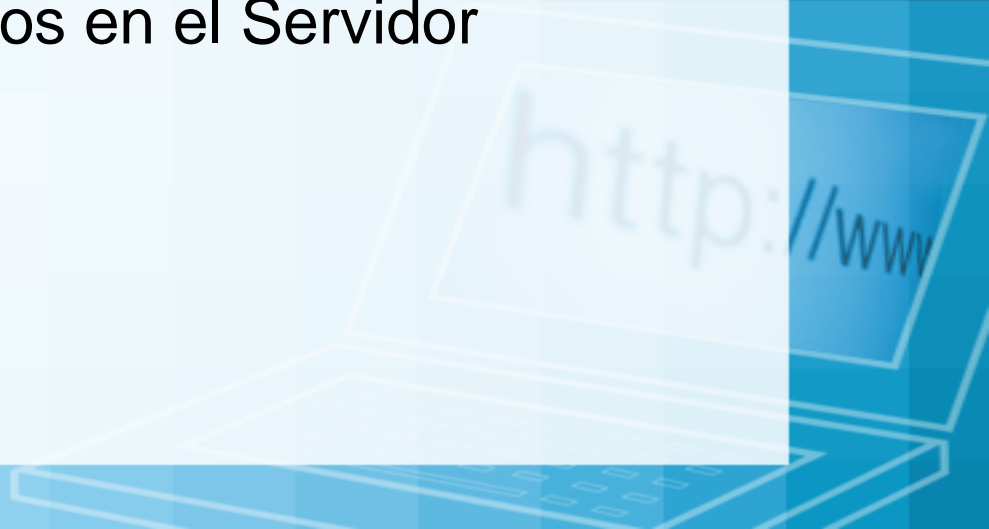


Archivos y directorios

Lenguajes Interpretados en el Servidor




Objetivos

- Hacer un repaso por los conceptos básicos relacionados con archivos.
- Adquirir dominio de las distintas operaciones que es posible realizar con archivos, tanto, operaciones a nivel interno, como externo.
- Lograr un amplio conocimiento de las distintas funciones que proporciona el lenguaje PHP para operaciones con archivos, como lectura y escritura.

Objetivos

- Tener conocimiento de las operaciones con archivo a nivel externo y de las funciones que proporciona PHP para su manejo.
- Adquirir dominio de las operaciones que se pueden realizar con directorios utilizando funciones de PHP.
- Alcanzar un perfecto dominio de los detalles que deben considerarse al subir archivos al servidor desde formulario HTML.

Contenido a desarrollar

1. ¿Qué es un archivo?
 2. Utilidad de los archivos.
 3. Ventajas de utilizar archivos.
 4. Operaciones con archivos.
 5. Verificar la existencia de un archivo.
 6. Abrir un archivo.
 7. Cerrar un archivo.
 8. Leer un archivo.
 9. Recorrido por las distintas formas y funciones de PHP para lectura de archivos.
- 
- A faint, stylized illustration of a laptop is visible in the bottom right corner of the slide. The screen of the laptop displays the text 'http://www' in a light blue font.

Contenido a desarrollar

10. Recorrer el contenido de un archivo.

- Utilización de la función `fseek()`.
- Utilización de la función `rewind()`.
- Utilización de la función `ftell()`.

11. Escribir en un archivo.

12. Funciones `fwrite()` y `fputs()`.

13. Operaciones a nivel externo.

- Función `copy()`.
- Función `rename()`.
- Función `unlink()`.

14. Manejo de directorios.



Contenido a desarrollar

10. Funciones PHP para directorios.
11. Procesamiento de los elementos de directorio.
12. Manejador de directorios.
13. Abrir y leer un directorio con opendir().
14. Utilización de la pseudo-clase dir.
15. Abrir y leer un directorio.
16. Utilización de la función scandir().
17. Obtener información de la ruta.
18. Cambiar un directorio.
19. Subida de archivos al servidor.



¿Qué es un archivo?

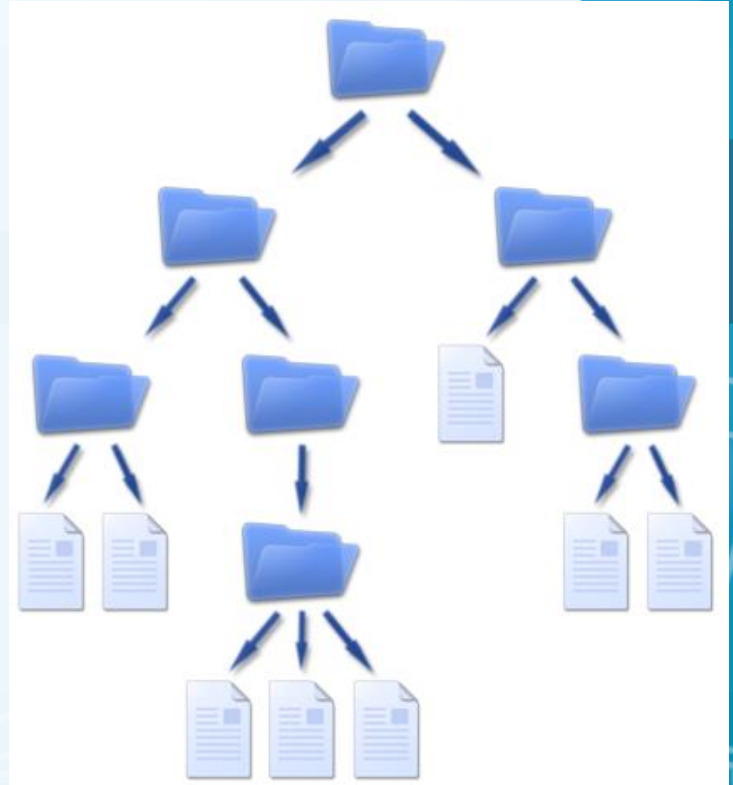
- Un **archivo** es un **conjunto de datos estructurados** que **son almacenados en algún medio** dentro del servidor web, en este caso, o en servidores remotos, para su posterior uso por las secuencias de comando o programas escritos en PHP.
- La forma en la que el **sistema operativo del servidor organiza, nombra, almacena y manipula los archivos** es a lo que se denomina **sistema de archivos**.

Utilidad de los archivos

- Los archivos se almacenan en directorios (carpetas) dentro del disco fijo (duro) del servidor web. Este medio de almacenamiento es permanente o persistente, ya que aún cuando se haya terminado el script, cerrado la sesión del usuario o incluso apagado el servidor, los datos seguirán almacenados de forma permanente.
- Los directorios se organizan jerárquicamente dentro de otros directorios, comenzando por el directorio raíz, que en el servidor web es la carpeta web www (caso del WampServer).

Utilidad de los archivos

- Entre los tipos de información que se pueden requerir almacenar de forma permanente se pueden mencionar: datos de usuario que ingresa a un sistema, productos que se agregan a un carrito de compras, preferencias del usuario cuando ingresa a un portal, etc.



Utilidad de los archivos

- Resulta importante recordar que la memoria RAM de las computadoras es volátil, y que por tanto toda la información cargada en ella desaparecerá al terminar la ejecución de un script, al cerrar el navegador o al apagar la computadora.
- Lo mismo ocurrirá si se satura la memoria con más información de la que se puede almacenar en ella.
- Como la memoria RAM es muy cara, resulta conveniente almacenar la información en memoria secundaria en forma de archivos.

Ventajas de utilizar archivos

- La información **es persistente**. Los datos almacenados en memoria secundaria (discos fijos) no se pierden al terminar el script, cerrar el navegador o apagar el equipo.
- La cantidad total de datos que se pueden almacenar en un archivo es superior a la que puede contener la memoria RAM.
- Permiten tener agrupada la información que esté relacionada de alguna forma. Por ejemplo, la música de nuestra preferencia, los datos de nuestros amigo(a)s, los datos de los libros de una biblioteca, etc.

Operaciones con archivos

- PHP proporciona una serie de funciones que permiten realizar operaciones con archivos a **nivel interno**, como: abrir, cerrar, leer, escribir y recorrer el archivo.



Verificar la existencia de un archivo

- La función `file_exists()`, verifica la existencia de un archivo antes de intentar utilizarlo.
- La sintaxis es la siguiente:

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

- `file_exists()` devuelve `true` si el archivo pasado como argumento en `$filename` existe en la ruta especificada también en `$filename`, o `false` si el archivo no existe.

Verificar la existencia de un archivo

```
<?php
//Comprobar si existe el archivo
$filename = '/path/to/file.txt';

if (file_exists($filename)) {
    echo "El fichero $filename existe";
} else {
    echo "El fichero $filename no existe";
}
?>
```



Abrir un archivo

- Para abrir un archivo, PHP provee la función `fopen()`. Con esta función se pueden abrir archivos locales o remotos.
- La sintaxis de esta función es la siguiente:

```
resource fopen($nombrearchivo, $modoapertura[,  
$rutabusqueda]);
```

- La forma en que esté establecido el parámetro `$nombrearchivo` indicará si el archivo debe ser buscado en el sistema de archivos del servidor (archivo local) o si está alojado en un servidor remoto, en cuyo caso la URL comenzará con `http://` o con `ftp://`.

Abrir un archivo

- Si PHP no encuentra el archivo en la ruta especificada en `$nombrearchivo` y el parámetro `$rutabusqueda`, estuviera establecido en 1, entonces lo buscará en los directorios definidos en el parámetro `include_path` del `php.ini`.

<code>implicit_flush</code>	Off	Off
<code>include_path</code>	<code>.;C:\php5\pear</code>	<code>.;C:\php5\pear</code>

Abrir un archivo

- El segundo parámetro `$modoapertura` indica la forma de acceso en que es abierto el archivo. Esta puede ser:

Valor	Significado
r	Modo de solo lectura. El puntero se coloca al inicio del archivo.
r+	Modo de lectura y escritura. El puntero se coloca al inicio del archivo.
w	Modo de solo escritura. Si el archivo no existe, entonces se crea, pero si ya existe, entonces se borra su contenido.
w+	Modo de lectura y escritura. Si no existe el archivo, se crea y si ya existe se borra todo su contenido.
a	Modo de solo escritura. Si no existe el archivo, se crea y si ya existe el puntero se coloca al final del archivo para añadir datos.
a+	Modo de lectura y escritura. Si no existe el archivo, se crea y si ya existe el puntero se coloca al final del archivo para añadir datos.
x	Modo de escritura prudente. Crea y abre un archivo local para escritura, comenzando al inicio del archivo. Si ya existe <code>fopen()</code> devuelve falso, si no PHP intenta crearlo.
x+	Modo de lectura y escritura prudente. Crea y abre un archivo local para lectura y escritura; comenzando al inicio del archivo. Si el archivo ya existe <code>fopen()</code> devuelve falso, si no existe, entonces intenta crearlo.

Abrir un archivo

- **b**: El modo de apertura binario, usado en combinación con los otros modos de apertura. Se aplica por defecto. Se aplica a sistemas de archivos que diferencian entre archivos binarios y archivos de texto. Caso de Windows, no así con UNIX y Macintosh.
- **t**: Usado para abrir archivos de texto en Windows, en donde se requiere traducir caracteres de final de línea `\n` a `\r\n`. Se usa con el modo `b` para portabilidad.

Abrir un archivo

- El tercer argumento `$rutabusqueda`, es opcional y tomará el valor de 1 o `true` si se desea indicar que a la hora de realizar la búsqueda del archivo especificado en el primer argumento de la función `fopen()`, se deben utilizar los directorios incluidos en la directiva `include_path` del archivo `php.ini`.
- En el caso de omitirse este argumento, PHP no utilizará estos directorios en la búsqueda.



Abrir un archivo

- Si la función `fopen()` se ejecuta satisfactoriamente devuelve un descriptor de archivo o una referencia a éste que deberá almacenarse en una variable.
- Esta variable se utilizará a lo largo de la secuencia de comandos para acceder al archivo cuando se desee leer o escribir en él.
- Ejemplo:

```
//Abrir archivo en modo solo lectura
$fr = fopen("myfile.txt", "r");

//Abrir archivo en modo binario de lectura y escritura
$fa = fopen("myfile.txt", "ba+");

//Abrir archivo en modo lectura/escritura (buscando en include_path)
$fw = fopen("code.php", "w+", true);
```

Cerrar un archivo

- Cerrar el archivo es la última operación que se debe realizar al manipular un archivo, con lo cual se elimina la referencia que se hace desde la secuencia de comandos al archivo.
- La función de PHP que realiza esta operación es `fclose()`, que devuelve el valor `true` si no se produce ningún problema al intentar cerrar el archivo o `false` si ocurre cualquier error.
- Una vez ejecutada esta función, para poder acceder al contenido del archivo, habrá que volver a abrirlo con `fopen()`.
- La sintaxis de la función `fclose()` es:

```
boolean fclose(resource $archivo);
```

Leer un archivo

- PHP tiene distintas y variadas formas para leer el contenido de un archivo. Lógicamente, se debería seleccionar la más conveniente de acuerdo a las necesidades.
- Se utilizan varias funciones de PHP para leer texto de un archivo. Esas funciones son:

- `fgetc()`
- `fgets()`
- `fgetss()`
- `fscanf()`
- `fread()`
- `readfile()`
- `file()`
- `file_get_contents()`



Leer un archivo con fgetc()

- La sintaxis de la función fgetc() es la siguiente:

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

- La función fgetc() devuelve un carácter del archivo referenciado por `$handle`. Si se ha llegado al final del archivo devuelve *false*.
- La forma más conveniente de leer un archivo utilizando la función fgetc() es haciendo uso de un ciclo o lazo, para que al momento de llegar al final del archivo pare la lectura.

Leer un archivo con fgetc()

- Ejemplo de lectura de archivo con fgetc():

```
$fp = fopen('archivo.txt', 'r');  
if (!$fp) {  
    die('No se pudo abrir archivo.txt');  
}  
while(false !== ($character = fgetc($fp))) {  
    echo "$character\n";  
}
```


Leer un archivo con fgets()

- La sintaxis de la función fgets() es la siguiente:
`string fgets (resource $handle [,int $length])`
- La función fgets() devuelve una cadena con una línea completa del archivo apuntado por `$handle`.
- La longitud de la cadena leída tendrá como máximo `$length-1` caracteres o bytes de longitud.
- Hay que mencionar que la operación de lectura puede terminar por una de tres causas:
 - Se han leído `$length-1` bytes.
 - Se ha leído el carácter de final de línea `"\n"`.
 - Se ha alcanzado el final de archivo.

Leer un archivo con fgets()

- Ejemplo de lectura de archivo con fgets():

```
define('ARCHIVO', 'productos.dat');  
$file = fopen(ARCHIVO, 'r');  
$numlinea = 0;  
while(!feof($file)){  
    $numlinea++;  
    //Se utiliza el 4096 como número de bytes para poder  
    //leer cualquier línea independientemente de su  
    //longitud  
    $buffer = fgets($file, 4096);  
    echo "Línea $numlinea: $buffer<br>\n";  
}  
fclose($file);
```

Leer un archivo con fgets()

```
<?php
$file = $_SERVER['DOCUMENT_ROOT']. "/examples/datos.dat";
//Verificar la existencia del archivo
if(!file_exists($file)){
    die("No existe el archivo o directorio.");
}
$fh = fopen($file, "r"); //Selección del modo de apertura
//Recorrer el archivo línea por línea hasta llegar al final
while(!feof($fh)){
    $line_text = fgets($fh);
    print $line_text;
}
//Cerrar el manejador de archivo
fclose($fh);
?>
```

Leer un archivo con fgetss()

- Si en lugar de leer un archivo de texto se pretende leer un archivo con código HTML y además, se desean omitir las etiquetas de este lenguaje durante la lectura del documento la función apropiada para tal efecto es fgetss().

- La sintaxis de la función es:

```
string fgetss(resource $handle [,int $length [,string $allowable_tags]]);
```

- El argumento **\$handle** representa el manejador del archivo que se desea leer, **\$length** es la cantidad de bytes que se van a leer del archivo y **\$allowable_tags** indica mediante una cadena de caracteres, las etiquetas HTML que no deben ser omitidas en la línea leída.

Leer un archivo con fscanf()

- Cuando la información almacenada en el archivo sigue un determinado formato, se puede ahorrar mucho trabajo empleando la función fscanf().
- La función fscanf() obtiene los datos de un archivo siguiendo un formato predeterminado. Posteriormente, la información extraída puede ser tratada por el programador.
- La sintaxis de la función fscanf() es la siguiente:

```
mixed fscanf(resource $handle, string $format[, mixed &$var1, ...]);
```
- Donde, **\$handle** es el manejador del archivo por leer, **\$format** es la cadena de caracteres que representa el formato en el que deben ser leídos los valores.

Leer un archivo con fscanf()

- Cuando sólo se proporcionan los dos primeros argumentos de la función fscanf(), los valores que se obtienen en la lectura se devuelven en una matriz.
- El resto de argumentos son opcionales y si se utilizan, representan las variables en donde se asignarán los valores obtenidos durante la lectura.
- Estos parámetros deben ser pasados por referencia y el número de parámetros utilizados debe coincidir con el número de valores a leer.
- Cuando se utiliza esta lista de parámetros opcionales la función devuelve como resultado el número de valores asignados.

```
define("ARCHIVO", "datos.dat");
$file = fopen(ARCHIVO, "r") or die("ERROR: Archivo no encontrado.");
$tabla = "<table>\n\t<thead>\n\t<tr>\n\t\t";
$tabla .= "<th>\n\t\t\tNombre\n\t\t</th>\n\t\t";
$tabla .= "<th>\n\t\t\tEdad\n\t\t</th>\n\t\t";
$tabla .= "<th>\n\t\t\tCantidad de datos\n\t\t</th>\n\t\t";
$tabla .= "<tr>\n\t</thead>\n\t\t";
$tabla .= "<tbody>\n\t\t";
while(!feof($file)){
    //Se leen los datos del nombre y la edad comprobando que la línea no está en blanco
    if($leidos = fscanf($file, "%s\t\t%d\n", &$nombre, &$edad)){
        //Se crea la fila de la tabla HTML con los datos leídos del archivo
        $tabla .= "<tr>\n\t\t<td>\n\t\t\t$nombre\n\t\t</td>\n\t\t\t";
        $tabla .= "<td>\n\t\t\t$edad\n\t\t</td>\n\t\t\t";
        $tabla .= "<td>\n\t\t\t$leidos\n\t\t</td>\n\t</tr>\n";
    }
}
$tabla .= "</tbody>\n</table>\n";
echo $tabla;
fclose($file);
```

Leer un archivo con fread()

- Lee un número específico de caracteres de un archivo, tratándolo como un archivo binario que consta de bytes, sin tener en cuenta finales de línea u otros caracteres especiales.
- La sintaxis de la función fread() es:
`string fread(resource $handle, int $length);`
- La función lee el número de caracteres indicados por `$length` a partir del puntero o referencia al archivo dada por `$handle`.
- La función fread() devuelve el número de bytes leídos desde el archivo.

Leer un archivo con fread()

- Es importante tener en cuenta que en sistemas que diferencian entre archivos binarios y de texto, como el caso de Windows, el archivo debería ser abierto en modo binario ('b') al usar la función fopen()).

- Ejemplo:

```
$filename = "datos.dat";  
$filehandle = fopen($filename, "rt");  
$contents = fread($filehandle, filesize($filename));  
print "<pre>$contents</pre>";  
fclose($filehandle);
```

Leer un archivo con readfile()

- Esta función lee el contenido de un archivo y lo muestra por la salida estándar o flujo de salida, que por lo regular es el navegador.
- La sintaxis es la siguiente:

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

- La función devuelve el número de bytes que han sido leídos del archivo. Si ocurre un error devuelve falso y se imprime un mensaje de error.
- Ejemplo:

```
$bytes = @readfile("data.dat");  
echo "$bytes le&iacute;dos";
```

Leer un archivo con file()

- Permite leer todo el contenido de un archivo, sin utilizar un manejador o apuntador, en lugar de ello lo devuelve en forma de *array*, una línea en cada posición.

- Sintaxis:

```
array file(string $filename [,int $flags = 0 [,resource $context]]);
```

- El nombre del archivo puede ser una ruta completa, relativa o, incluso, una URL. Tenga en cuenta que si cada elemento del array corresponde a una línea en el archivo, incluirá también un carácter de fin de línea al final de cada uno de los elementos de la matriz.
- Si trabaja en Macintosh puede ser que PHP no reconozca los finales de línea. En ese caso tendrá que habilitar la opción de configuración en tiempo de ejecución en la directiva `auto_detect_line_endings`.

Recorrer el contenido de un archivo

- A veces puede ser necesario colocar el puntero de lectura en una parte específica del archivo para poder leer o escribir cadenas de texto a partir de esa posición.
- PHP dispone de varias funciones relacionadas con el posicionamiento en determinados puntos de un archivo que facilitan el trabajo:
 - `rewind()`.
 - `fseek()`.
 - `ftell()`.



Utilización de la función fseek()

- La función fseek() le permite acceder de forma aleatoria a un archivo, moviéndose a la posición de un byte específico dentro del archivo.

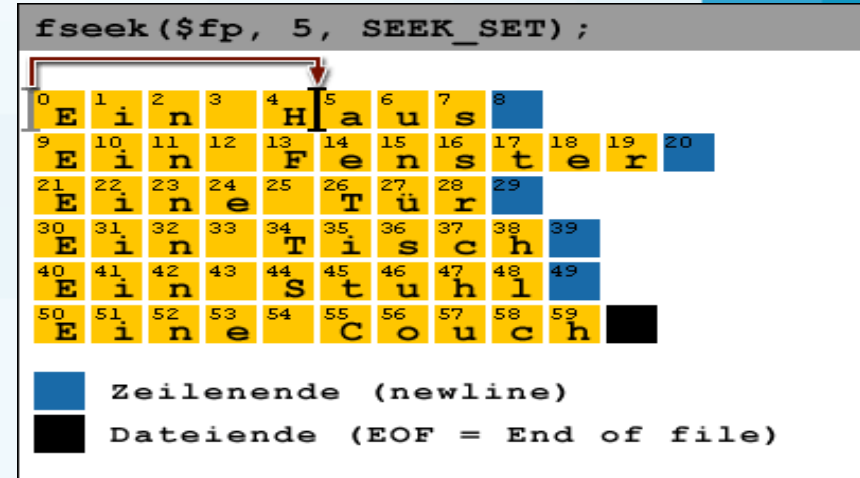
- Sintaxis:

```
int fseek(resource $handle, int $offset [,int $whence=SEEK_SET])
```

- Una llamada a fseek() establece el puntero de lectura/escritura del archivo a un punto que comienza desde la posición indicada por **\$whence**, cuando se establece un valor, o desde el comienzo del archivo (byte 0) cuando no se establece, y se mueve a partir de esa posición \$offset bytes.
- El parámetro opcional **\$whence** toma el valor predeterminado SEEK_SET que indica el principio del archivo. Otros valores posibles para este parámetro son: SEEK_CUR, para la posición actual del puntero al archivo y SEEK_END para ubicarlo en el final.

Utilización de la función `rewind()`

- La función `rewind()` sitúa el puntero de lectura/escritura al principio del archivo.
- Sintaxis:
`bool rewind (resource $handle)`
- Esta función recibe un manejador de archivo como argumento y devuelve un valor booleano que será verdadero en caso de éxito o falso de producirse algún fallo.



Utilización de la función ftell()

- Si se han leído ciertos datos de un archivo y se desea conocer en qué posición quedó el puntero de lectura/escritura cuando terminó de leer puede usar la función ftell().

- Sintaxis:

```
int ftell(resource $handle)
```

- La función ftell() recibe un manejador de archivos como argumento y devolverá la posición del byte actual, el número de bytes desde el comienzo del archivo y la posición desde donde comenzará la próxima operación de lectura.
- Usando el resultado anterior junto a la función seek() se puede devolver la posición exacta en el archivo.
- Tome en cuenta que si está usando la lectura de texto, los caracteres de retorno de carro y salto de línea forman parte de la cuenta de bytes.

Escribir en un archivo

- Para escribir datos en un archivo se dispone de las funciones: `fwrite()` y `fputs()`. Ambas funciones se pueden utilizar de forma indistinta, ya que en la práctica trabajan de la misma forma, incluso, puede decirse que una es un alias de la otra.
- La sintaxis de ambas funciones es la siguiente:

```
int fwrite(resource $handle, string $string [, int $length]);  
int fputs(resource $handle, string $string [, int $length]);
```
- La escritura en un archivo puede implicar comenzar a escribir desde cero en un archivo o agregar datos en un archivo existente.

Escribir en un archivo

- Debe tomarse en cuenta que cuando se abre un archivo para escritura, si el archivo no existe se creará, y si existe se sobrescribirá en él perdiéndose los datos que pudieran existir.
- Cuando el archivo sea abierto para agregar datos, si el archivo existe, entonces se adicionarán los datos al final del archivo. Si el archivo no existe, será creado.



Escribir en un archivo

- Para escribir datos en un archivo se necesitará
 1. Abrir el archivo en modo de escritura.
 2. Realizar la operación de escritura.
 3. Cerrar el archivo.



Funciones `fwrite()` y `fputs()`

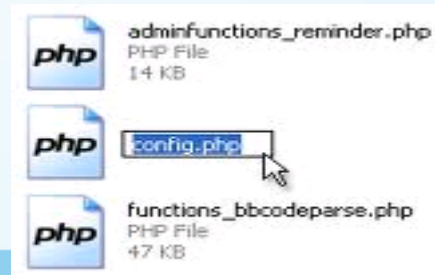
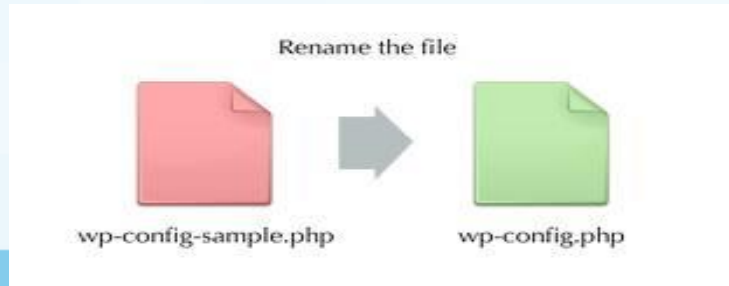
- La función `fwrite()` escribe una cadena de texto en un archivo y devuelve el número de bytes escritos.
- Recibe tres argumentos, el primero `$handle`, es un manejador de archivo que apunta al archivo abierto.
- El segundo, `$string` contiene los caracteres que van a ser escritos en el archivo.
- El tercer argumento, `$length` es opcional, y si se proporciona la escritura parará luego de que se hayan escrito `$length` caracteres o hasta que se llegue al final de `$string`, lo que suceda primero.
- `fputs()` es un alias de `fwrite()`.

Ejemplo con fputs()

```
//Contador de visitas en archivo
$filename = 'sitevisitors.txt';
if (file_exists($filename)) {
    $count = file('sitevisitors.txt');
    $count[0] ++;
    $fp = fopen("sitevisitors.txt", "w");
    fputs ($fp, "$count[0]");
    fclose ($fp);
    echo $count[0];
}
else {
    $fh = fopen("sitevisitors.txt", "w");
    if($fh==false)
        die("unable to create file");
    fputs ($fh, 1);
    fclose ($fh);
    $count = file('sitevisitors.txt');
    echo $count[0];
}
```

Operaciones a nivel externo

- Las operaciones más comunes con archivos a nivel externo son:
- Copiar archivos con la función `copy()`.
- Renombrar archivos con la función `rename()`.
- Eliminar archivo con la función `unlink()`.



Función copy()

- La sintaxis de la función copy() es la siguiente:

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

- Esta función realiza una copia del archivo indicado en `$source` hacia `$destiny`.
- El primer argumento, `$source` puede incluir la ruta de acceso al archivo a copiar.
- Puede utilizarse un tercer argumento opcional, `$context`, que representa un recurso de contexto válido creado con `stream_context_create()`.
- La función devuelve `true` en caso de éxito en la operación y `false` en caso contrario.

Función rename()

- La sintaxis de la función rename() es la siguiente:

```
bool rename(string $oldname,string $newname[,resource $context]);
```

- La función se puede utilizar para cambiar el nombre de un archivo de `$oldname` a `$newname`.
- También puede utilizarse para mover un archivo de una ubicación a otra, ya que ambos argumentos `$oldname` y `$newname` pueden incluir las rutas junto con los nombres de los archivos.
- Puede utilizarse un tercer argumento opcional, `$context`, que representa un recurso de contexto válido creado con `stream_context_create()`.
- Devuelve true en caso de éxito en la operación y falso en caso contrario.

Función unlink()

- La sintaxis de la función unlink() es la siguiente:
`bool unlink(string $filename [,resource $context]);`
- Esta función se utiliza para borrar el archivo `$filename` del servidor.
- El argumento `$filename` puede incluir la ruta al archivo junto al nombre del mismo.
- La función unlink() devolverá `true` en caso de éxito o `false` en caso contrario.

Manejo de directorios

- PHP pone a disposición varias funciones para trabajar con directorios en el sistema de archivos.
- Desde un script PHP se puede abrir un directorio y leer su contenido, tal y como lo hace un comando ls en UNIX o dir en DOS.
- Puede cambiar a un nuevo directorio, listar el directorio actual, eliminar un directorio, etc.



Funciones PHP para directorios

Función	Descripción
<code>chdir()</code>	Permite cambiar de directorio activo.
<code>mkdir()</code>	Crea dentro del árbol de directorios un nuevo directorio en la ubicación indicada.
<code>rmdir()</code>	Elimina un directorio que deberá estar vacío y se deberán tener permisos de escritura sobre ese directorio.
<code>chroot()</code>	Cambiar al directorio raíz.
<code>opendir()</code>	Devuelve un manejador de directorio que puede usarse con <code>readdir()</code> , <code>closedir()</code> y <code>rewinddir()</code> .
<code>closedir()</code>	Cierra un manejador de directorio abierto antes con <code>opendir()</code> .
<code>readdir()</code>	Lee el siguiente archivo del manejador de directorio que se haya abierto con <code>opendir()</code> .
<code>rewinddir()</code>	Retrocede el puntero del manejador de directorio al principio del directorio
<code>scandir()</code>	Devuelve un array de archivos y directorios de una ruta dada.

Funciones para directorios

- La sintaxis de las tres principales funciones para trabajar con directorios:
- `bool chdir(string $directory);`

Cambia el directorio actual al indicado en `$directory`.

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

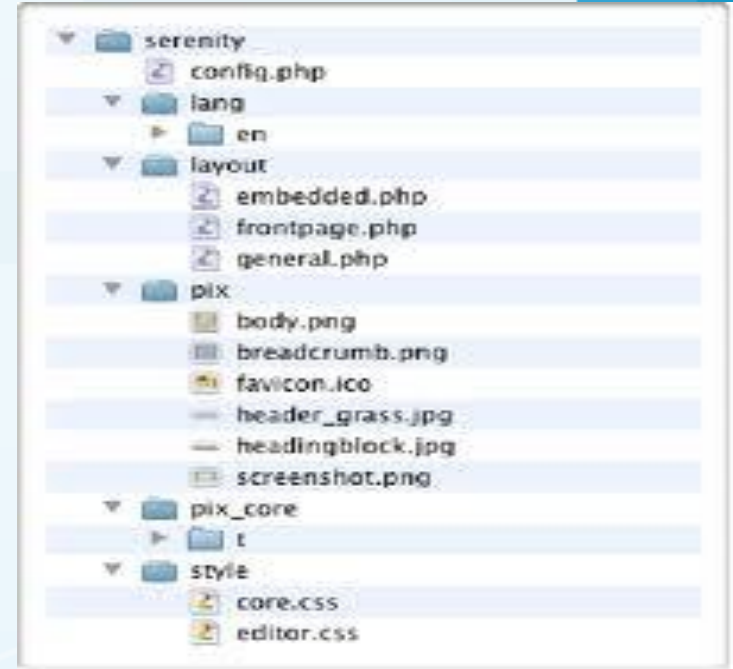
Intenta crear el directorio especificado por `$pathname`.

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

Elimina el directorio especificado en `$dirname`.

Procesamiento de los elementos de directorio

- PHP proporciona varias alternativas para realizar operaciones sobre directorios.
 1. Utilizar un manejador de directorios.
 2. Por medio de la pseudo-clase `dir`.
 3. Utilizando la función `scandir()`



Manejador de directorios

- Un manejador de directorios es un recurso que representa una conexión lógica con un directorio determinado y que permite leer la lista con los nombres de los elementos del mismo (archivos y subdirectorios).
- El procesamiento de un directorio debe comenzar con la creación de un manejador de directorio.
- Para ello se utiliza la función `opendir()`.

Abrir y leer un directorio con opendir()

- Para abrir una conexión lógica a un directorio con un script PHP, se utiliza la función opendir(), cuya sintaxis es la siguiente:

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

- La función crea un manejador de directorio (valor devuelto) para permitir el acceso al directorio tal y como está almacenado en el disco por el sistema operativo sin tener en cuenta su estructura interna.
- Una vez devuelto el manejador de directorio se pueden utilizar las funciones readdir(), rewinddir() y closedir() para trabajar con el directorio.
- Si el directorio no puede ser abierto devolverá *false*.

Abrir y leer un directorio con opendir()

```
<?php
$manejador = opendir(".");
echo "Elementos del directorio actual:<br />";
while($elemento = readdir($manejador)) {
    echo "$elemento <br />\n";
}
closedir($manejador);
?>
```



Utilización de la pseudo-clase dir

- PHP proporciona una pseudo-clase para el manejo de directorios, llamada clase dir. Aunque no aporta ninguna funcionalidad adicional a la mostrada por la función opendir(), muestra una opción orientada a objetos para procesar directorios:

```
Directory dir(string $directory[, resource $context]);
```

- El objeto creado al instanciar a la pseudo-clase cuenta con tres métodos y dos propiedades. Estas últimas son solo de consulta.
- Los métodos son: read(), rewind() y close().
- Las propiedades son: \$path (ruta de acceso al directorio) y \$handle (manejador del directorio).

Abrir y leer un directorio

```
<?php
$directorio = dir(".");
echo "Elementos del directorio actual:";
echo $directorio->path, "<br />\n";
while($entrada = $directorio->read()){
    echo $entrada, "<br />\n";
}
$directorio->close();
?>
```



Utilización de la función scandir()

- Las últimas versiones de PHP incorporan una única función que permite en una sola operación recuperar todos los elementos de un directorio. La sintaxis de esta función es:

```
array scandir(string $directory[, int $sorting_order =  
SCANDIR_SORT_ASCENDING) ;
```

- En donde el parámetro `$directory` es el directorio local o remoto a procesar y `$sorting_order` es un parámetro opcional que si no es proporcionado toma el valor indicado por la constante `SCANDIR_SORT_ASCENDING` (0) que indicará que el ordenamiento será ascendente.

Utilización de la función scandir()

- La función devuelve un array con la lista de todos los elementos encontrados dentro del directorio (archivos y subdirectorios).

```
//Obtendrá los elementos del directorio actual  
//listándolos en orden descendente.
```

```
$directorio = scandir(".", 1);  
foreach($directorio as $valor){  
    echo $valor . "<br />";  
}
```



Obtener información de la ruta

- La función `dirname()` devuelve el nombre del directorio de una ruta en el que se encuentra el archivo pasado como argumento con toda su ruta.

```
string dirname(string $path);
```

- La función `basename()` devuelve el nombre del archivo sin el componente del directorio; es decir, únicamente una cadena con el nombre del archivo.

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

Obtener información de la ruta

```
<?php
```

```
    $path = "c:/wamp/www/examples/file.php";
```

```
    //Salida: c:/wamp/www/examples
```

```
    echo dirname($path) , "<br />\n";
```

```
    //Salida: file.php
```

```
    echo basename($path) , "<br />\n";
```

```
?>
```



Cambiar un directorio

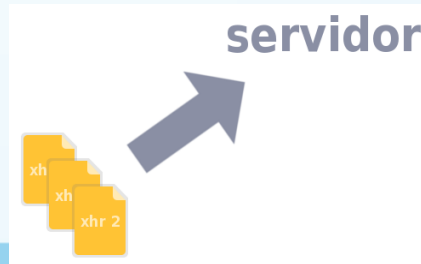
- Se le denomina así al hecho de cambiarse de un directorio a otro. Para realizar esa operación se utilizar la función `chdir()`.

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

- Se pueden utilizar nombres de rutas relativas y absolutas en el argumento **`$directory`**.
- Devuelve *true* en caso de éxito en la operación y *false* en caso de fallo.

Subida de archivos al servidor

- La subida de archivos al servidor es una operación que puede ser frecuente y de utilidad para las aplicaciones web del lado del servidor.
- Esta operación consiste básicamente en que los usuarios puedan enviar archivos desde una página web al servidor y que éstos sean almacenados en alguna carpeta dentro del servidor web.



Subida de archivos al servidor

- Para poder enviar un archivo al servidor web, debe crear un formulario que incluya un elemento input de tipo file:

```
<input type="file" name="archivo" accept="image/*" />
```

- El elemento form donde se incluye este elemento input type="file" debe incluir como atributo enctype el valor multipart/form-data.

```
<form action="script.php" method="POST"  
enctype="multipart/form-data">
```

- Un control de datos oculto (hidden) que se denomine MAX_FILE_SIZE, que contendrá el tamaño máximo del archivo que el usuario subirá al servidor. Este tamaño estará dado en bytes.

```
<input type="hidden" name="MAX_FILE_SIZE" value="102400" />
```


Subida de archivos al servidor

- Cuando los datos del formulario son enviados al servidor, deben ser procesados. PHP proporciona la matriz asociativa superglobal `$_FILES` una vez que el archivo es enviado, los índices asociativos de dicha matriz pueden ser utilizados para acceder a las propiedades del archivo.
- El archivo enviado al servidor queda alojado temporalmente en el directorio especificado en la directiva de configuración del archivo `php.ini`. Esta directiva es `upload_tmp_dir`, por lo general, en WampServer, con el siguiente valor:

```
upload_tmp_dir = "c:/wamp/tmp"
```

Subida de archivos al servidor

- Para poder enviar un archivo al servidor web, debe crear un formulario que incluya un elemento input de tipo file:

```
<input type="file" name="archivo" accept="image/*" />
```

- El elemento form donde se incluye este elemento input type="file" debe incluir como atributo enctype el valor multipart/form-data.

```
<form action="script.php" method="POST"  
enctype="multipart/form-data">
```

- Un control de datos oculto (hidden) que se denomine MAX_FILE_SIZE, que contendrá el tamaño máximo del archivo que el usuario subirá al servidor. Este tamaño estará dado en bytes.

```
<input type="hidden" name="MAX_FILE_SIZE" value="102400" />
```

Funciones PHP para directorios

Clave	Descripción
name	Nombre real del archivo transferido.
tmp_name	Nombre que se asigna al archivo de forma temporal en el servidor una vez que se termina de cargar. Si no se hubiera completado esta operación, entonces obtendremos none como valor.
size	Tamaño en bytes del archivo enviado al servidor.
type	El tipo MIME del archivo, si el navegador proporciona esta información.
error	El código de error asociado con el envío del archivo. Este índice asociativo se añadió a partir de la versión 4.2.0 de PHP.

FIN

