estudios abiertos
SEAS
GRUPO SANVALERO

4

4 UNIDAD DIDACTICA

PHP

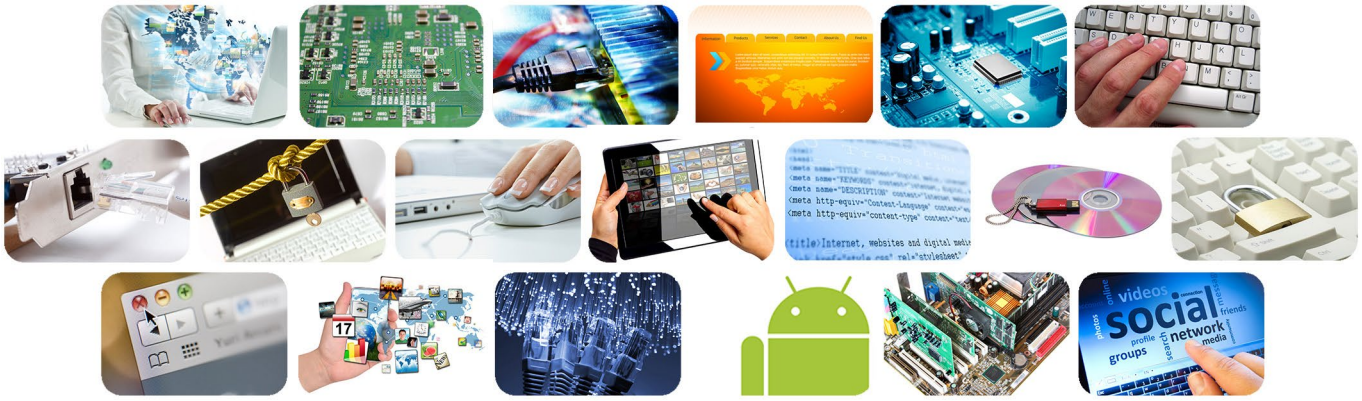
4. Conceptos avanzados de PHP

ÍNDICE

OBJETIVOS

- Explicar los conceptos de clases y objetos y cómo podemos utilizarlos en PHP.
- Explicar diferentes tipos de formularios que podemos utilizar en nuestras aplicaciones.
- Explicar el concepto de sesiones, su forma de trabajo y su implementación en PHP.
- Explicar el concepto de cookies, su forma de trabajo y su implementación en PHP.
- Conocer las operaciones básicas que podemos utilizar en el uso de ficheros.
- Introducir conceptos relativos a diferentes estrategias de seguridad que debemos implementar en nuestras aplicaciones.
- Conocer la forma en la que podemos enviar correos electrónicos desde nuestras aplicaciones en PHP.

INTRODUCCIÓN



En esta unidad vamos a adentrarnos en conocimientos más avanzados del lenguaje de programación PHP, aunque bien pueden ser utilizados e implementados en otros lenguajes.

Nos referimos a conceptos importantes y vitales como el de la programación orientada a objetos, el uso de sesiones y de cookies, operaciones básicas con ficheros y diferentes operaciones básicas que suelen ser implementadas en la utilización de formularios para nuestras aplicaciones.

Veremos la forma en la que PHP nos suministra funcionalidades para este tipo de operaciones comunes a cualquier entorno de programación, pero también veremos otras funcionalidades más particulares de PHP como el uso de algoritmos de encriptación tales como crypt(), md5() o sha1(), técnicas para detectar la inyección de código SQL o las diferentes maneras en las que podemos realizar un seguimiento de los errores que se puedan producir como consecuencia de la ejecución de nuestro código.

4.1. PHP orientado a objetos

Mediante la utilización de la programación orientada a objetos (POO), PHP nos va a ofrecer a los programadores la posibilidad de tener una programación más estructurada y fácil de implementar.

Veremos que gracias al concepto de objeto nos será posible desarrollar un código que podamos reutilizar todas las veces que estimemos oportuno sin tener que modificar nada.

Entender y desarrollar un código en base a una programación orientada a objetos es sinónimo de un alto nivel de conocimiento de programación y será el punto de partida en la creación de grandes proyectos, donde podremos compartir trabajo con otros programadores y asumir cada uno una parte independiente del proyecto, de tal forma que al final puedan unirse todas, dando un significado global a una aplicación.

Trabajando con programación orientada a objetos, cuando tengamos que utilizar alguna clase/objeto que haya desarrollado otra persona, no nos será necesario entender su funcionamiento, ni qué técnicas ha utilizado para desarrollarlo. Sólo nos hará falta conocer cómo acceder para poder usarlo en nuestro beneficio, proporcionándonos de esta forma un nivel de abstracción.

4.1.1. Concepto de clases y objetos

En un contexto de programación orientada a objetos, un objeto podrá ser cualquier elemento o concepto, tanto físico como podría ser un cliente, como conceptual que solo existe en el software y que, por ejemplo, podría ser un campo de entrada de texto o un archivo.

En la programación orientada a objetos, lo que diseñaremos y desarrollaremos serán objetos independientes, donde cada uno de ellos estará formado por unas propiedades y operaciones que van a interactuar entre ellas para lograr una serie de funcionalidades.

Las propiedades o variables serán los atributos que tendrán alguna relación con el objeto y las operaciones serán denominadas como métodos, funciones o acciones de los objetos, que utilizará el propio objeto para poder realizar tareas de modificación a sí mismo o para conseguir algún objetivo externo. Por lo tanto, podremos decir que un objeto estará formado por atributos y métodos.

Otro concepto importante relacionado con la programación orientada a objetos es la encapsulación. Los objetos tienen la capacidad de permitir o fomentar la ocultación de sus datos, de tal forma que el acceso a la información de su interior solo podrá ser posible mediante las operaciones con los objetos, a través de la interfaz del objeto. A esta ocultación de los datos se le denomina encapsulación.

Podremos actualizar los detalles de implementación de un objeto para obtener mayores beneficios en su rendimiento, añadir nuevas funcionalidades o resolver problemas, todo ello sin tener que modificar la interfaz de acceso al mismo.

Por otra parte, los objetos podrán ser agrupados en clases. Las clases representarán un conjunto de objetos que podrán variar, pero que tendrán un conjunto de características en común. Una clase contendrá objetos con operaciones que van a comportarse de la misma manera, y atributos que representarán las mismas cosas, aunque los valores de estos atributos puedan variar de un objeto a otro.

Por lo tanto, podemos decir que una clase será como una plantilla para un nuevo objeto. El modelo a partir del cual se va a implementar el objeto. A su vez, un objeto, será la instancia real de una clase, y a partir de ese momento, será cuando el objeto adquiera una entidad propia con sus características y funciones (los atributos y métodos).

A la hora de implementar la programación orientada a objetos en nuestros programas, los objetos serán entidades únicas, y serán instancias a una clase en concreto. La clase será definida con sus atributos y métodos, para posteriormente, cuando vayamos a crear un objeto, lo hagamos de tal forma que se base en una clase ya definida. A esto se le denomina instanciar una clase.

4.1.2. Crear clases, atributos y métodos

Para crear una clase, tendremos que utilizar la palabra reservada "class". La estructura básica de una clase tendrá un aspecto como el siguiente:

EJEMPLO

```
class nombre_de_clase
{
}
```

El siguiente paso será la definición de sus atributos y sus operaciones. Como ya hemos explicado, los atributos serán las variables de la clase y para su definición haremos uso de la palabra reservada "var".

En cuanto a los métodos, hará falta definir las funciones en la definición de la clase. Estas funciones podrán recibir parámetros o no hacerlo, y será necesario el uso de la palabra reservada "function" para crearlas.

Por lo tanto, la estructura completa de una clase sería la siguiente:

EJEMPLO

```
class nombre_de_clase
{
var $variable1;
var $variable2;
function primerafuncion($parametro1,$parametro2){
…. }
function segundafuncion(){
….}
}
```

4.1.3. Constructor de una clase

En muchas ocasiones, la clase va a contener una función que será ejecutada de forma automática al crear la clase. A este método se le denomina constructor y será una función que se ejecutará de manera automática al crear la instancia de la clase (el objeto) y que resultará de gran utilidad cuando sea necesario inicializar algún valor de la clase.

Los constructores también podrán recibir parámetros, que podrán ser opcionales, haciéndolos más útiles.

Aunque nosotros también podamos realizar llamadas al constructor de la clase, su principal objetivo siempre será llamarse a sí mismo en la construcción del objeto, para de esta forma poder inicializar los atributos con determinados valores o crear nuevos objetos que se puedan necesitar para el óptimo funcionamiento del objeto, por ejemplo.

En la mayoría de lenguajes de programación, el constructor es un método que va a compartir el mismo nombre que la clase. Esto también ocurría en las versiones PHP3 y PHP4. Sin embargo, desde PHP5, el constructor para todas las clases recibe el nombre **"__construct()"** (con dos guiones bajos).

NOTA

Para mantener la compatibilidad, PHP buscará un método con el mismo nombre de la clase si no puede encontrar "__construct()", pero este no debería ser siempre el caso, así que deberíamos usar siempre "__construct()".

El constructor podrá admitir el paso de parámetros, si creemos que son necesarios cuando creemos un nuevo objeto. Por lo tanto, su sintaxis dentro de una clase tendría que se algo como lo siguiente:

EJEMPLO

```
function __construct($valor)
{
//acciones
}
```

Por último, comentar que de igual forma que en una clase puede existir un constructor, también podrá existir un **destructor.**

Un destructor nos posibilitará poder ejecutar una funcionalidad en concreto antes de que la clase sea destruida, algo que ocurrirá de forma automática cuando sean anuladas todas las referencias a la clase o éstas no se correspondan con el ámbito.

De la misma forma que un constructor se crea con la palabra reservada "__construct()", un destructor se creará con la palabra reservada **"__destruct()".** Un destructor no podrá admitir parámetros.

4.1.4. Uso de objetos, instancias de una clase

Una vez que hayamos declarado una clase, el siguiente paso será crear el objeto con el que queramos trabajar. A este paso se le denomina **crear una instancia o instanciar una clase** y se podrá realizar mediante el uso de la palabra reservada "**new**".

En el proceso de instanciar una clase será necesario indicar a qué clase va a pertenecer el objeto, así como facilitarle los parámetros necesarios al método constructor si lo hubiera.

Veamos un ejemplo sencillo donde creamos dos objetos a partir de la misma clase, pero le pasamos valores diferentes al método constructor:

EJEMPLO

```
<?php
class doble
{
       function __construct($numero)
       {
echo "Método constructor llamado con el valor $numero<br>";
       }
}
$miobjeto= new doble(5);
$objeto2 = new doble(3);
?>
```

En el anterior script, en primer lugar hemos declarado una nueva clase denominada "cuadrado", y al declarar su método constructor, hemos admitido el paso de un parámetro, para que al crear cada nuevo objeto, muestre un mensaje en pantalla con el valor del parámetro pasado al instanciar la clase. Por lo tanto, el resultado tendría que ser la aparición en el navegador de los siguientes mensajes:

EJEMPLO

Método constructor llamado con el valor 5
Método constructor llamado con el valor 5

4.1.5. Uso de atributos de una clase

Para ampliar las funcionalidades de una clase será necesario el uso de atributos. Para definir un atributo dentro de una clase tendremos que definirlo en primer lugar con la palabra reservada **"var"**, y una vez que haya sido definido, para utilizarlo, tendremos que añadirle delante del nombre del atributo la palabra reservada **"$this->"**, de tal forma, que para hacer referencia a un atributo, tendremos que escribir **"$this->atributo".**

Partiendo del ejemplo anterior, vamos a ampliar nuestra clase con un nuevo atributo denominado **"$mult**", el cual lo inicializaremos con el valor "2" y que lo utilizaremos dentro de la clase constructor para multiplicar el parámetro pasado por este atributo, consiguiendo de esta forma obtener el doble del parámetro utilizado a la hora de instanciar la clase:

EJEMPLO

```
<?php
class doble
{
       var $mult=2;
       function __construct($numero)
       {
echo "Método constructor llamado con el valor $numero<br>";
echo "El valor doble del parámetro es :".$this->mult*$numero;
       echo "<br>";
       }
}
$miobjeto= new doble(5);
$objeto2 = new doble(3);
?>
```

También podríamos hacer uso del atributo desde fuera de la clase, es decir, desde nuestro script. Se haría de la siguiente forma:

EJEMPLO

```
<?php
$miobjeto= new doble(5);
$miobjeto->mult=3;
echo $miobjeto->mult;
?>
```

4.1.6. Uso de métodos de una clase

Para explicar la utilización de método de una clase, vamos a modificar la clase con la que estamos trabajando, de tal forma que en lugar de tener que realizar la multiplicación de forma manual, esta operación vamos a incluirla en el nuevo método que denominaremos multxnum. Además, añadiremos otro método que realizará el cálculo de la mitad del valor:

EJEMPLO

```
<?php
class doble
{
    var $mult=2;
    var $numinterno;
       function __construct($numero)
       {

       echo "Método constructor llamado con el valor
$numero<br>";
        $this->numinterno=$numero;
        echo "<br>";
       }
    function multxnum()
    {
        return $this->mult*$this->numinterno;
    }
    function divxnum()
    {
        return $this->numinterno/$this->mult;
    }
}
$miobjeto= new doble(5);
echo "El valor doble del parámetro es: ".$miobjeto->multxnum();
echo "<br>";
echo "El valor dividido por 2 del parámetro es: ".$miobjeto-
>divxnum();
?>
```

Como podemos observar, en primer lugar hemos utilizado el método constructor para guardar el parámetro pasado al instanciar el objeto en un atributo interno, que luego utilizaremos para realizar las operaciones.

Por cada operación, hemos creado un método, donde cada uno devolverá un valor mediante la palabra reservada "return".

Para hacer referencia a los métodos, al igual que con los atributos, haremos uso del símbolo "->" de tal forma que la sintaxis es $objeto->método().

En el ejemplo anterior, hemos usado los métodos directamente para incluirlos en el mensaje de salida a pantalla, pero también podríamos haber guardado su valor en una variable del programa principal, para poder utilizarla posteriormente en otras operaciones:

EJEMPLO

```
<?php
$miobjeto= new doble(5);
$eldoble=$miobjeto->multxnum();
echo "El valor doble del parámetro es: $eldoble";
echo "<br>";
$lamitad=$miobjeto->divxnum();
echo "El valor dividido por 2 del parámetro es: $lamitad";
?>
```

4.1.7. Visibilidad de atributos y métodos

Desde la versión PHP5 tenemos a nuestra disposición unos modificadores de acceso a través de los cuales podremos controlar la visibilidad de los atributos y los métodos. Tendremos los tres siguientes modificadores de acceso:

- Public. Ésta siempre es la opción predeterminada de cualquier atributo o método. De esta forma, podrán ser accedidos tanto dentro de clase, como desde el exterior cuando sea creado un objeto.
- Private. Estableciendo este modificador de acceso, solo se podrá tener acceso a los atributos y métodos desde el interior de la clase. Si intentáramos desde el exterior, obtendríamos un error en la ejecución del programa. En un capítulo posterior, hablaremos de herencia de objetos. Pues bien, los elementos privados no podrán ser heredados.
- Protected. Igual que el o modificador de acceso Private, pero en este caso sí que se permitirá la herencia de atributos y métodos Protected.

Para mostrar un ejemplo, vamos a seguir trabajando con la clase que hemos definido anteriormente. Podríamos declarar las funciones como públicas, puesto que vamos a usarlas todas desde fuera de la clase, pero las dos primeras variables, al ser solo de uso interno, podríamos declararlas como privadas:

EJEMPLO

```
<?php
class doble
{
    private  $mult=2;
    private  $numinterno;
       public function __construct($numero)
       {

       echo  "Método  constructor  llamado  con  el  valor
$numero<br>";
        $this->numinterno=$numero;
        echo "<br>";
       }
    public function multxnum()
    {
        return $this->mult*$this->numinterno;
    }
    public function divxnum()
    {
        return $this->numinterno/$this->mult;
    }
}
```

4.1.8. Herencia

Una de las técnicas más utilizadas en la programación orientada a objetos es la herencia. Con la herencia podremos crear nuevas clases que se basen en otras que ya estén creadas, pudiéndolas ampliar con nuevos métodos y atributos.

Para poder heredar una clase creada en PHP se tiene que utilizar la palabra reservada "*extends*" seguida de la clase original, que será denominada la superclase "*parent*", mientras que la clase que hereda recibirá el nombre de subclase "*child*". De una forma esquemática, así tendrían que ser declaradas:

EJEMPLO

```
class UNO{
//propiedades clase UNO
…
//métodos clase DOS
}
class DOS extends UNO {
//propiedades clase DOS
…
//métodos clase DOS
}
```

En este supuesto práctico, la clase DOS va a heredar todas los atributos y métodos de la clase UNO para que pueda incorporarlos y, por lo tanto, utilizarlos, junto con sus propios atributos y métodos.

Si no existiera esta posibilidad de heredar una clase, tendríamos que volver a definir los mismos métodos y atributos en la clase DOS, lo cual sería redundante y de poca utilidad.

Ahora veamos un ejemplo algo modificado y ampliado sobre el que hemos estado trabajando hasta ahora.

Modificaremos la clase padre con la que estábamos trabajando hasta ahora para eliminar la clase constructor, de tal forma que le pasaremos directamente el valor del número sobre el que vamos a realizar las operaciones, haciendo uso de su atributo desde el exterior ($miobjeto->numinterno). Por lo demás, la clase padre hace las mismas operaciones de multiplicar por 2 y dividir por 2 este parámetro.

A partir de aquí, crearemos una segunda clase hija de la primera, con una nueva función, que utilizaremos para volver a multiplicar el parámetro que hemos pasado a través del atributo por el resultado de la función "multxnum()" de la clase padre.

Veamos el ejemplo:

EJEMPLO

```
<?php
class doble
{
   var $mult=2;
   var $numinterno;
    public function multxnum()
    {
        return $this->mult*$this->numinterno;
    }
    public function divxnum()
    {
        return $this->numinterno/$this->mult;
    }

}
class dobleheredada extends doble
{
    function triple(){
        return($this->multxnum()*$this->numinterno);
    }
}
$miobjeto= new doble();
$miobjeto->numinterno=3;
$eldoble=$miobjeto->multxnum();
echo "El valor doble del parámetro es: $eldoble";
echo "<br>";
$lamitad=$miobjeto->divxnum();
echo "La mitad del parámetro es: $lamitad";
$objeto3=new dobleheredada();
$objeto3->numinterno=3;
echo "<br>";
echo "El doble del cuadrado del parámetro es: ".$objeto3-
>triple();
?>
```

La herencia podrá tener varias capas de profundidad. Podríamos declarar una tercera clase, que ampliará las funcionalidades de la segunda, y por lo tanto, heredará de la primera y de la segunda todos sus atributos y métodos.

4.1.9. Sobrescritura de clases

En el capítulo anterior, hemos visto cómo crear una subclase, que además de heredar los atributos y métodos de la superclase, define los suyos propios.

Pero en ocasiones puede que nos interese volver a declarar los mismos atributos y métodos con nuevas funcionalidades. A esta técnica se le llama sobrescritura o reemplazo de clases.

Vamos a realizar una modificación en el ejemplo anterior de herencia, modificando la clase hija para que tenga el mismo atributo y el mismo método que la clase de la que hereda, pero con distinto valor y funcionabilidad para el método:

EJEMPLO

```
<?php
class doble
{
   var $mult=2;
   var $numinterno;
    public function multxnum()
    {
        return $this->mult*$this->numinterno;
    }
    public function divxnum()
    {
        return $this->numinterno/$this->mult;
    }
}
class dobleheredada extends doble
{
    var $mult=3;
    function multxnum()

    {
        return $this->mult*$this->numinterno;
    }
}
$miobjeto= new doble();
$miobjeto->numinterno=3;
$eldoble=$miobjeto->multxnum();
echo "El valor doble del parámetro es: $eldoble";
echo "<br>";
$lamitad=$miobjeto->divxnum();
echo "La mitad del parámetro es: $lamitad";
$objeto3=new dobleheredada();
$objeto3->numinterno=3;
echo "<br>";
echo "El doble del cuadrado del parámetro es: ".$objeto3-
>multxnum();
?>
```

Como podemos observar, los cambios realizados a la subclase, no afectan para nada a la superclase. Una subclase heredará todos los atributos y operaciones de su superclase, a menos que les suministremos reemplazos.

No obstante, a partir de PHP5 tenemos la posibilidad de evitar que una función pueda ser reemplazada en una supuesta herencia. Para ello, será necesario añadir en nuestra primera clase a la función que no queremos que se reemplace, la palabra reservada "final". Quedaría de la siguiente forma:

EJEMPLO

```
final function multxnum()
    {
        return $this->mult*$this->numinterno;
    }
```

Si en la segunda clase que va a heredar la primera, intentáramos reemplazar la función "multxnum()", tendríamos un error como el siguiente:

EJEMPLO

Fatal error: Cannot override final method doble::multxnum() producto.php….

De la misma forma que podemos evitar que los métodos sean reemplazados, podremos realizar lo mismo con las clases. Si no queremos que la primera clase sea heredada, podríamos también utilizar la instrucción "final" por delante del nombre de la clase para evitarlo:

EJEMPLO

```
<?php
final class doble
{
   var $mult=2;
   var $numinterno;
   public function multxnum()
….
```

Si intentáramos definir la segunda clase para que herede de la primera, tendríamos un error como el siguiente:

EJEMPLO

Fatal error: Class dobleheredada may not inherit from final class (doble) in producto.php…

4.1.10. Polimorfismo

Seguramente, el polimorfismo es la característica más importante en la programación orientada a objetos.

Como es un concepto algo complejo, mejor vamos a explicarlo de la manera más práctica que podamos. El ejemplo que vamos a simular a continuación, es el que vamos a aprovechar para realizar un ejercicio final de programación orientada a objetos, como resumen de los visto anteriormente.

Imaginemos que tenemos la posibilidad de poder comprar dos coches: un Seat y un BMW. Ambos tienen una característica en común: avanzar. Ambos coches podrán "avanzar", pero cada uno lo hará de distintas maneras. Uno lo hará más rápido, y por lo tanto, consumirá más, y otro lo hará más lento y consumiendo menos.

Bien, pues mediante el polimorfismo, podemos para un mismo nombre de método, representar un código diferente en clases distintas. O dicho de otra manera, clases diferentes tendrán un comportamiento distinto, para el mismo método.

En el siguiente capítulo vamos a realizar un ejercicio completo donde podremos ver la forma de trabajo mediante polimorfismo.

4.1.11. Ejemplo explicado

En el siguiente ejemplo vamos a utilizar las siguientes clases:

Vehiculo. Será la clase padre de las subclases "Seat" y "BMW". Define un atributo protegido $Gasoil, es decir, podrá ser heredado, pero no podrá ser modificado desde el exterior de una clase. Este atributo es inicializado al valor 80 litros, y será el valor devuelto por el método getGasoil().

BMW. Subclase hija de "Vehículo" que añade un nuevo método denominado avanzar() y en el cual utilizará el atributo heredado "Gasoil" para indicar que su forma de "avanzar" es descontando 10 litros cada vez. Esta será la diferente forma de actuar de un mismo método en diferentes clases, en este caso, diferente a la subclase "Seat". Esta es la base del polimorfismo.

Seat. Subclase hija de "Vehículo" que añade también un nuevo método denominado avanzar() y en el cual utilizará el atributo heredado "Gasoil" para indicar que su forma de "avanzar", que esta vez será descontando 5 litros cada vez. Volvemos a comentar lo mismo. Mismo método, distinto resultado.

Conductor. Define un atributo privado $vehiculo, que por lo tanto, ni podrá ser heredado ni accesible desde el exterior de la clase. Este atributo servirá para almacenar un objeto que será pasado a la clase en la creación del objeto mediante su método constructor.

Dicho de otra forma, cuando instanciemos la clase desde el programa principal para crear un nuevo objeto, le pasaremos como parámetro un objeto. Como podemos ver en el ejemplo, los objetos que se pasarán a esta clase, serán objetos de las subclases "Seat" y "BMW", y por lo tanto, en el interior de esta clase, al almacenarlos en el atributo $vehiculo, podremos hacer uso del método "avanzar()" original de estas subclases, y el método "getGasoil()", original de la clase padre "Vehiculo".

En la zona de ejecución del programa, podremos ver como instanciamos las subclases "Seat" y "BMW", en los objetos $conductor1 y $conductor2 respectivamente para poder utilizarlas.

Para cada una de ellas, utilizaremos el método "avanzarVehiculo()" para poder comprobar cómo consume gasoil cada uno de ellos.

Al ejecutar estos métodos, el valor del atributo $Gasoil disminuye de forma diferente en cada objeto, lo cual podemos comprobarlo al mostrarlo por pantalla mediante el método $conductor1->Gasoil() y $conductor2>Gasoil().

A continuación, el código del ejemplo comentado:.

EJEMPLO

```
<?php
class Vehiculo{
   protected $Gasoil = 80; //Comienza con 80 litros de Gasoil
    public function getGasoil(){
       return $this->Gasoil;
    }
}
class BMW extends Vehiculo
   {
    public function avanzar()
   {
        $this->Gasoil - = 10;
     }
   }
class Seat extends Vehiculo{
    public function avanzar()
    {
        $this->Gasoil - = 5;
    }
  }
class conductor {
    private $vehiculo;
    function __construct($objeto){
       $this->vehiculo = $objeto;
    }
    public function avanzarVehiculo(){
       $this->vehiculo->avanzar();
    }
    public function Gasoil(){
       return $this->vehiculo->getGasoil();
    }
}
$conductor1 = new conductor (new Seat);
$conductor1->avanzarVehiculo();

$conductor2 = new conductor (new BMW);
$conductor2->avanzarVehiculo();

echo "Al Seat del conductor 1 le quedan " . $conductor1-
>Gasoil() . " litros de Gasoil<br/>";
// Al Seat del conductor 1 le quedan 75 litros de Gasoil
echo "Al BMW del conductor 2 le quedan " . $conductor2-
>Gasoil() . " litros de Gasoil<br/>";
// Al BMW del conductor 2 le quedan 70 litros de Gasoil
?>
```

4.2. Formularios y PHP

En este capítulo vamos a explicar diversos tipos de formularios que podemos usar en nuestras aplicaciones en PHP.

4.2.1. Autenticación de usuarios

En determinados momentos, nos interesará ofrecer mecanismos de seguridad para poder controlar que se puedan acceder a diferentes zonas de nuestras páginas web.

Una forma sencilla de conseguirlo es a través de la validación de usuarios mediante el mecanismo de autenticación básica del protocolo HTTP.

El protocolo http tiene una función denominada "*header()*" que enviará un mensaje en una ventana emergente "*Authentication Required*" al navegador del usuario, de tal forma que le estaremos obligando a introducir un usuario y un password.

Cuando el usuario haya introducido estos datos, el script se encargará de volver a cargar la página habiendo introducido estos valores en las variables "*PHP_AUTH_USER*" y "*PHP_AUTH_PW*", además de la variable "*AUTH_TYPE*" que es utilizada para indicar el tipo de autenticación.

Estas variables se encuentran localizadas en la variable superglobal que vimos en anteriores unidades *"$_SERVER*".

En la unidad 3 ya utilizamos la función "*header()*" para enviar una cabecera "*Location*", que en ese caso nos servía para redirigirnos hacia una página web que especificamos junto con el comando. Como podremos ver en el ejemplo que explicamos a continuación, en esta ocasión incluimos en la función una cabecera "*WWW-Authenticate*" que utilizaremos como medio de autenticación básico y para restringir el acceso a la página si fuera necesario.

Una cuestión muy importante que debemos tener en cuenta es que el código que va a comprobar la identidad de los usuarios deberá ser ejecutado antes de que cualquier salida sea enviada al navegador, es decir, antes de enviar las cabeceras. En nuestro ejemplo, primero ejecutamos la función "*validar()*" antes de enviar la cabecera mediante la función "*header()*".

Veamos el ejemplo:

EJEMPLO

```
<?php
function validar($user,$pass) {
$users = array('silvia' => 'silvia', 'pedro' => 'perico');
if (isset($users[$user]) && ($users[$user] == $pass)) {
return true; /*comprobamos si el usuario y contraseña envia-
dos existen en el array*/
}
else {
return false; //no existen en el array
}
}

//Comienza el código principal de validación
  if  (!  validar($_SERVER['PHP_AUTH_USER'],  $_SERVER['PHP_
AUTH_PW']))
{
header('WWW-Authenticate: Basic realm="Mi zona web"');
header('HTTP/1.0 401 Unauthorized');
echo "Tienes que introducir un usuario y contraseña válidos";
exit;
}
else {
    echo "Bienvenido/a".$_SERVER['PHP_AUTH_USER'];
}
?>
```

En nuestro sencillo ejemplo, primero definimos una función donde vamos a tener a nuestros usuarios almacenados en un array. Más adelante, cuando expliquemos cómo conectar con bases de datos, lo habitual será comprobar si los usuarios se encuentran almacenados en una tabla.

Mediante la función "*isset()*" comprobaremos si dichos valores existen en nuestro array, de tal forma que si existen, devolvemos el valor TRUE y si no existen, devolvemos un valor FALSE.

En el código principal del programa, utilizamos los valores TRUE o FALSE que devuelve la función para tener claro si el usuario existe, y en tal caso le damos la bienvenida, o no existe, para lo cual primero enviamos una cabecera de autenticación, y a continuación, otra cabecera de no autorizado, junto con el mensaje de aviso para indicar que el usuario no es válido.

Es importante conocer que primero siempre habrá que enviar la cabecera *WWW-Authenticate*, antes de la cabecera *http/1.0 401*.

4.2.2. Subida de ficheros

Con PHP también podremos realizar cargas de archivos mediante la utilización del navegador con la presentación de un formulario, donde el usuario elegirá el archivo que quiere subir al servidor.

Esta tarea tendrá dos fases.

En primer lugar, tendremos que realizar la presentación del formulario por pantalla para indicarle al usuario que tiene que subir un archivo.

A continuación, habrá que escribir el código en PHP para gestionar la subida de ese archivo.

Podríamos presentar un formulario sencillo HTML de la siguiente forma:

EJEMPLO

```
<HTML>
<HEAD>
<TITLE>Página de subida de archivos</title>
<BODY>
<h1>Subida de archivos</h1>
<form action=”2.php” method=”POST” enctype=”multipart/form-
data”>
Introduce el fichero: <input type=”file” name=”fichero”><br>
<input type=”hidden” name”MAX_FILE_SIZE” value = “1000000”>
<input type=”submit” value=”Enviar fichero”>
</form>
</BODY>
</HTML>
```

Vamos a explicar las características de este formulario:

- Con la etiqueta <form> para comenzar a crear el formulario, establecemos el atributo enctype=”multipart/form-data” para de esta forma poder permitir que el servidor pueda conocer que vamos a enviar un archivo con la información habitual de un formulario.
- Incluimos un campo de formularios para establecer el archivo con su tamaño máximo que podrá ser subido. Lo definimos como un campo oculto y lo hemos especificado en la línea: <input type=”hidden” name”MAX_FILE_SIZE” value = “1000000”>.
- Por último, será necesaria la introducción de una entrada de tipo “file” para poder seleccionar el archivo: <input type=”file” name=”fichero”>.

En la siguiente fase de recogido del archivo, vamos a diseñar un script en PHP para controlar determinados parámetros.

Al subir el archivo, se almacenará en un lugar temporal en el servidor, concretamente, su directorio predeterminado, de tal forma que si no lo movemos o cambiamos el nombre del archivo antes de terminar la ejecución de la secuencia de comandos, el archivo será eliminado.

Podremos controlar algunos parámetros relacionados con la subida del archivo mediante la variable superglobal $_FILES. Todas las entradas en $_FILES serán guardadas con el nombre de la etiqueta "<file>" que hayamos creado en el formulario HTML. En nuestro ejemplo, el elemento del formulario recibirá el nombre de "fichero", y por lo tanto, la variable $_FILES tendrá los siguientes contenidos:

- $_FILES['fichero]['tmp_name']. Lugar donde se almacena temporalmente el archivo en el servidor.
- $_FILES['fichero]['name']. Nombre del archivo en el sistema de ficheros.
- $_FILES['fichero]['size']. Tamaño del archivo en bytes.
- $_FILES['fichero]['type']. Tipo MIME del archivo. Por ejemplo si es text/plain o image/gif.
- $_FILES['userfile']['error']. Errores que se puedan producir en la carga de archivos. En el ejemplo podremos ver hasta 4 tipos de errores que se pueden producir.

Además, existen otras funciones que pueden ser muy útiles en determinados contextos.

Por ejemplo, mediante la función *"is_upload_file()",* podremos comprobar si el archivo ha sido subido con éxito.

Con la última función "*move_uploaded_file()",* podremos mover el archivo subido de la zona temporal a una localización definitiva.

Veamos cómo sería el archivo "2.php", que recibiría los datos del formulario diseñado anteriormente en otro fichero:

EJEMPLO

```
<?php
if (is_uploaded_file($_FILES['fichero']['tmp_name'])) {
   echo "Archivo ". $_FILES['fichero']['name'] ." subido con
éxtio.\n";
} else {
   echo "Posible ataque del archivo subido: ";
  echo "nombre del archivo '". $_FILES['fichero']['nombre_
tmp'] . "'.";
}
if ($_FILES['fichero']['error'])
{
          switch ($_FILES['fichero']['error'])
          {
                   case 1: // UPLOAD_ERR_INI_SIZE
               echo"El archivo sobrepasa el limite auto-
rizado por el                    servidor(archivo php.ini)
!";
                   break;
                   case 2: // UPLOAD_ERR_FORM_SIZE
                   echo "El archivo sobrepasa el limite au-
torizado en el formulario HTML !";
                   break;
                   case 3: // UPLOAD_ERR_PARTIAL
                   echo "El envio del archivo ha sido sus-
pendido durante la transferencia!";
                   break;
                   case 4: // UPLOAD_ERR_NO_FILE
                   echo "El archivo que ha enviado tiene un
tamaño nulo !";
                   break;
          }
}
if ((isset($_FILES['fichero']['archivo']))&&($_
FILES['fichero']['error'] ==
 UPLOAD_ERR_OK))
{
$ruta_destino = '/var/www/uploads/';
move_uploaded_file($_FILES['fichero']['tmp_name'],
$ruta_destino.$_FILES['fichero']['name']);
   }
?>
```

En este ejemplo se ha utilizado como ruta definitiva una ruta de Linux. Si el equipo servidor fuera Windows, tendríamos que especificar con \\ en lugar de solo una \.

4.2.3. Validación de campos

Una tarea muy importante en la recogida de datos de un formulario, es validar que los datos introducidos son correctos en cuanto a su formato.

Nos estamos refiriendo a tareas de comprobación como que un usuario ha entrado una dirección de email de forma correcta, o una dirección http con sintaxis perfecta o que no ha entrado un dato de tipo cadena de texto cuando se solicitaba un dato de tipo entero.

Antiguamente se escribían cientos de líneas de código para poder realizar estas verificaciones, pero por suerte, a día de hoy ya no es necesario gracias a la extensión de PHP filter_var.

Esta extensión nos va a suministrar un conjunto de funciones que nos van a posibilitar la validación de los datos de una manera relativamente sencilla. Incluye numerosos filtros a elegir en función del tipo de dato que queramos filtrar. La función filter_var utiliza los siguientes parámetros:

- $var. Variable que queremos filtrar.
- $filter. El filtro que queremos aplicar.
- $options. Algunos de los filtros pueden utilizar un array de opciones que modificarán su comportamiento.

A continuación, vamos a mostrar los filtros que están disponibles:

- FILTER_VALIDATE_BOOLEAN. Comprueba si la variable es un booleano.
- FILTER_VALIDATE_EMAIL. Comprueba si la variable es una dirección de correo electrónico correcta.
- FILTER_VALIDATE_FLOAT. Comprueba si la variable es del tipo float.
- FILTER_VALIDATE_INT. Comprueba si la variable es un número entero.
- FILTER_VALIDATE_IP. Comprueba si la variable es una dirección IP.
- FILTER_VALIDATE_REGEXP. Valida la variable contra una expresión regular enviada en la variable de opciones.
- FILTER_VALIDATE_URL. Comprueba si la variable es una URL de acuerdo con la RFC 2396.

Implementa también una serie de filtros que pueden ser utilizados para depurar las variables que le pasemos como argumentos:

- FILTER_SANITIZE_EMAIL. Elimina todos los caracteres excepto letras, números y !#$%&'*+-/=?^_`{|}~@.[].
- FILTER_SANITIZE_ENCODED. Codifica la cadena como una URL válida.

- FILTER_SANITIZE_MAGIC_QUOTES. Aplica la función *addslashes*.
- FILTER_SANITIZE_NUMBER_FLOAT. Elimina todos los caracteres excepto números, +- y opcionalmente ,.eE.
- FILTER_SANITIZE_NUMBER_INT. Elimina todos los caracteres excepto números y los signos + -.
- FILTER_SANITIZE_SPECIAL_CHARS. Escapa caracteres HTML y caracteres con ASCII menor a 32.
- FILTER_SANITIZE_STRING. Elimina etiquetas, opcionalmente elimina o codifica caracteres especiales.
- FILTER_SANITIZE_STRIPPED. Alias del filtro anterior.
- FILTER_SANITIZE_URL. Elimina todos los caracteres excepto números, letras y $-_.+!*'(),{}|\\^~[]`<>#%";/?:@&=.

Veamos unos cuantos ejemplos para comprobar la potencia de esta función.

Para comprobar si la variable que pasamos como argumento es un dato entero:

EJEMPLO

```
<?PHP
$var = '2.3';
if(filter_var($var, FILTER_VALIDATE_INT) === false){
     echo 'Valor incorrecto';
}else{
     echo 'Valor correcto';
}
?>
```

En el siguiente ejemplo, solo serán válidas aquellas variables que sean de tipo entero, y además, se encuentre su valor entre un rango proporcionado:

EJEMPLO

```
<?php
$a = '1';
$b = '-1';
$c = '3';
$options = array(
    'options' => array(
                        'min_range' => 0,
                        'max_range' => 3,
                        )
);
if (filter_var($a, FILTER_VALIDATE_INT, $options) !== FALSE) {
    echo "La variable a es correcta (valor entre 0 y 3).";
    echo "<br>";
}

if (filter_var($b, FILTER_VALIDATE_INT, $options) !== FALSE) {
    echo "La variable b es correcta (valor entre 0 y 3).\n";

                    echo "<br>";
}
if (filter_var($c, FILTER_VALIDATE_INT, $options) !== FALSE) {
    echo "La variable c es correcta (valor entre 0 y 3).\n";
    echo "<br>";
}
?>
```

En el siguiente ejemplo vamos a comprobar si las direcciones de email que hay almacenadas en variables tienen un formato correcto:

EJEMPLO

```
<?php
$email_a = 'cristian@midominio.com';
$email_b = 'cristian';

if (filter_var($email_a, FILTER_VALIDATE_EMAIL)) {
    echo "Esta dirección de email (email_a) es considerada
válida.";
    echo "<br>";
}
if (filter_var($email_b, FILTER_VALIDATE_EMAIL)) {
    echo "Esta dirección de email (email_b) es considerada
válida.";
    echo "<br>";
}
?>
```

Podemos también comprobar si determinadas direcciones IP tienen un formato correcto:

EJEMPLO

```
<?php
$ip_a = '192.168.1.25';
$ip_b = '23.34.36.36';
if (filter_var($ip_a, FILTER_VALIDATE_IP)) {
    echo "La dirección IP ".$ip_a." es válida.<br>";
}
if (filter_var($ip_b, FILTER_VALIDATE_IP)) {
    echo "La dirección IP ".$ip_b." es válida.";
}
?>
```

En cuanto al saneamiento de datos, veamos un sencillo ejemplo donde eliminamos lo que no nos interesa de una cadena de texto:

EJEMPLO

```
<?php
    $text = '<b>"Hola mundo!"</b>';
    echo $text;
    echo "<br>";
    echo filter_var($text, FILTER_SANITIZE_STRING);
?>
```

4.2.4. Paso de variables (Métodos GET y POST)

Hasta este momento, en los ejercicios previos, nuestros principales objetivos han sido realizar tareas con los datos introducidos por un usuario: hemos visto cómo autenticar un usuario, cómo se puede subir archivos al servidor o cómo podemos validar y sanear los datos introducidos.

Pero aún no hemos visto la forma en la que un formulario envía esos datos para que posteriormente puedan ser procesados.

Un formulario en HTML, siempre comienza y termina con la etiqueta <FORM>, pero será a través de su etiqueta <METHOD> donde se especifique los dos métodos posibles de envío de valores: GET y POST:

- GET. Mediante el método GET, los datos serán enviados a través de la URL y unidos por el símbolo **"&"** que genera el formulario, después pulsar el botón SUBMIT (Enviar). De hecho, en realidad no sería necesario realizar un formulario para enviar estos datos. Simplemente tendremos que crear una URL con los datos como la de nuestro ejemplo, para que pudieran ser enviados mediante sencillos links. Sería un formato como el siguiente:

EJEMPLO

http://www.mipagina.com/index.php?page=mia&variable2&variable3=valor3

y serán recogidos de esta forma:

EJEMPLO

```
$nombre_de_variable=$_GET['nombre_de_objeto'] ;
```

- POST. Este método solo puede ser utilizado mediante un formulario, y la información no será enviada por una URL, sino que será enviada de forma transparente al usuario. Es preferible usarlo si vamos a pasar grandes campos de texto, vamos a subir imágenes, o si simplemente no queremos crear una URL tan larga y favorecer las denominadas URL's amigables, que tanto gustan a los motores de búsqueda como Google. Las variables enviadas mediante este método podrán ser recogidas con el siguiente formato:

EJEMPLO

```
$nombre_de_variable=$_POST['nombre_de_objeto'] ;
```

De esta forma, tendremos que decidir qué método queremos usar para definirlo en el formulario en primer lugar, y posteriormente en el script que recoge los datos, utilizar $_POST o $_GET. Veamos un ejemplo para cada caso.

Envío de variables mediante método POST.

En este primer archivo, crearíamos el formulario para recoger los datos del usuario y enviarlos al segundo script en PHP que recogerá y mostrará los datos:

EJEMPLO

```
<HTML>
<HEAD>
<TITLE>Página de subida de archivos</title>
<BODY>
<h1>Subida de archivos</h1>
<form action="2.php" method="post">
      Nombre:<br/>
      <input type="text" name="nombre"/><br/>
      E-mail:<br/>
      <input type="text" name="email"/><br/>
      Comentario:<br/>
<textarea type="text" rows="5" cols="50" name="comentar-
io"></textarea><br/>
      <input type="submit" name="boton"/>
</form>
</BODY>
</HTML>
```

En el script "2.php" recogemos los datos enviados y para comprobar que se han enviado de forma correcta, los mostramos en pantalla:

EJEMPLO

```
<html>
<head>
<title>Métodos GET y POST</title>
</head>

<body>
<h1>Ejemplo de uso de métodos GET y POST en formularios</h1>
<?php
echo "El nombre que ha introducido es ".$_POST['nombre'];
echo "<br>";
echo "El email introducido es: ".$_POST['email'];
echo "<br>";
echo "El comentario introducido es: ".$_POST['comentario'];
?>
<br>
</body>
</html>
```

Envío de variables mediante método GET.

En este primer archivo, crearíamos el formulario para recoger los datos del usuario. Observamos que solo cambia el método de envío.

EJEMPLO

```
<HTML>
<HEAD>
<TITLE>Página de subida de archivos</title>
<BODY>
<h1>Subida de archivos</h1>

<form action="2.php" method="get">
       Nombre:<br/>
       <input type="text" name="nombre"/><br/>
       E-mail:<br/>
       <input type="text" name="email"/><br/>
       Comentario:<br/>
<textarea type="text" rows="5" cols="50" name="comentario"></
textarea><br/>
       <input type="submit" name="boton"/>
</form>
</BODY>
</HTML>
```

En el script "2.php" recogemos los datos enviados y para comprobar que se han enviado de forma correcta, los mostramos en pantalla. Comprobaremos, que solo cambia la variable superglobal $_POST por $_GET:

EJEMPLO

```
<html>
<head>
<title>Métodos GET y POST</title>
</head>
<body>
<h1>Ejemplo de uso de métodos GET y POST en formularios</h1>
<?php
echo "El nombre que ha introducido es ".$_GET['nombre'];
echo "<br>";
echo "El email introducido es: ".$_GET['email'];
echo "<br>";
echo "El comentario introducido es: ".$_GET['comentario'];
?>
<br>
</body>
</html>
```

4.3. Cookies en PHP

En el anterior capítulo, hemos explicado cómo crear sesiones de tal forma que podamos guardar información relativa al usuario que esté visitando nuestro sitio web.

Con las cookies podremos también realizar este tipo de operaciones, ofreciéndonos una serie de ventajas frente al uso de sesiones.

4.3.1. Qué son y cómo funcionan

Una cookie es un conjunto de información que las secuencias de comandos de una página podrán almacenar en el equipo del cliente, de tal forma que el navegador del cliente pueda utilizarlas en cada solicitud a una página.

Una cookie será un fichero de texto que va a almacenar en su interior parejas de datos con su nombre y su valor. Cuando un navegador realiza una petición a una página web, el servidor puede crear un cookie y devolverla al navegador como parte de la respuesta. A partir de ese momento, el navegador envía la cookie de vuelta al servidor cada vez que realice una nueva solicitud a la página web alojada en el servidor. Esto permite al servidor acceder a los datos de la cookie y usar dichos datos para controlar el acceso a su página web.

Por defecto, el periodo de vida de una cookie será válido hasta que el usuario cierre el navegador. Aunque también podrían ser configuradas para que sean “persistentes”.

Un aspecto muy a tener en cuenta si optamos por una estrategia de cookies, es que los usuarios puede que tengan deshabilitadas las cookies en sus navegadores, y esto nos supondría un grave problema.

Las cookies suelen ser utilizadas para permitir a los usuarios saltarse las páginas de “login” o formularios de registro, al tener ya guardados y cargados sus datos de usuario y password. Pero también pueden ser utilizadas para personalizar las páginas con información, como por ejemplo, estado del tiempo, de las carreteras, marcadores deportivos, etc.

Una cookie, como decimos, es un archivo de texto, que podría tener un aspecto como el siguiente:

EJEMPLO

```
PHPSESSID=o9mqiad9dmtadh60f02h0h97a3
user_id=86
email=hola@prueba.com
userName=Silvia
passwordCookie=abretesesamo
```

4.3.2. Creación de una Cookie

Para crear una cookie y configurarla en el navegador, podremos utilizar la función "**setcookie()".** Esta función tendrá la siguiente sintaxis:

EJEMPLO

```
setcookie ($name, $value, $expire, $path, $domain, $secure,
$httponly)
```

Los únicos parámetros que serán obligatorios serán el nombre y el valor inicial, el resto serán opcionales:

- **$name**. Nombre de la cookie.
- **$value**. Valor de la cookie. Por defecto es una cadena vacía.
- **$expire**. La fecha de expiración de la cookie formato timestamp. Si está configurada a 0, la cookie expirará cuando el usuario cierre el navegador. Es su valor por defecto.
- **$path**. La ruta en el servidor donde la cookie estará disponible. Si se configura con el valor "/", la cookie estará disponible en todos los directorios del servidor actual. El valor por defecto puede ser configurado en el fichero PHP.INI.
- **$domain**. El dominio para el que la cookie estará disponible. Por defecto es el nombre del servidor que está almacenando la cookie.
- **$secure**. Si está a TRUE, la cookie estará disponible solo si es enviada usando el protocolo HTTPS. Por defecto se encuentra con valor FALSE.
- **$httponly**. Si está a TRUE, la cookie solamente estará disponible a través del protocolo HTTP y no podrá ser utilizada por lenguajes como Javascript. Su valor por defecto es FALSE.

Para recuperar el valor de una cookie, pondremos la variable superglobal **$_COOKIE**, que al igual que **$_SESSION**, es un array asociativo.

NOTA

Las cookies son parte de las cabeceras HTTP, así es que setcookie() tendrá que ser llamada antes que cualquier otra salida sea enviada al navegador. Ésta es la misma limitación que tiene la función header().

Veamos un ejemplo donde vamos a crear un array que va a contener tres cookies, cada una con su valor, y un tiempo de vida de 1 hora. A continuación, utilizaremos de nuevo el bucle FOREACH para recorrer el array $_COOKIE y mostrar todos sus valores:

EJEMPLO

```
<?php
// configuramos los cookies
setcookie("cookie[tres]", "cookietres", time()+3600);
setcookie("cookie[dos]", "cookiedos",time()+3600);
setcookie("cookie[one]", "cookieuno",time()+3600);
// después de recargar la página, tendrá que visualizar lo
siguiente
if (isset($_COOKIE['cookie'])) {
    foreach ($_COOKIE['cookie'] as $name => $value) {
        echo "$name : $value <br />\n";
    }
}
?>
```

4.3.3. Eliminación de una Cookie

Para eliminar una cookie tendremos que establecer un valor de fecha ya expirada a la cookie. Podríamos hacerlo de la siguiente forma:

EJEMPLO

```
<?php
// configuramos la fecha de expiración en una hora atrás en
el tiempo
setcookie ("cookie[tres]", "", time() - 3600);
?>
```

4.3.4. Ventajas y desventajas del uso de Cookies

En el capítulo siguiente explicaremos el uso de sesiones, y sus diferencias con las cookies. De momento, vamos a adelantar una serie de ventajas de las cookies sobre las sesiones:

- Más fáciles de crear y recibir por navegador.
- Requieren menos tareas de mantenimiento desde el punto de vista del servidor.
- Por norma general, pueden persistir más en el tiempo que una sesión.
- Útil es para formularios del tipo "Recuérdame en este equipo".
- Útil es para almacenar opciones personalizadas de usuarios.

Como norma general, deberemos utilizar las cookies en situaciones donde la seguridad es un problema menor y solo un mínimo de datos se están almacenando.

Si la seguridad es un tema importante y hay mucha información a recordar, será mejor el uso de las sesiones. Sin embargo, el uso de sesiones puede requerir un poco más de esfuerzo en desarrollar nuestros scripts.

Además, podríamos nombrar también los siguientes inconvenientes:

- Los datos se almacenan en el cliente (equipo del usuario). Eso significa que el usuario puede "modificar" esos valores.
- Solo pueden almacenarse cierta cantidad de datos.
- En cada carga se pasan todos los datos del usuario, lo que incrementa el consumo de ancho de banda.

4.4. Sesiones en PHP

Las sesiones son una serie de variables que estarán almacenadas en nuestro servidor y que nos van a proporcionar información sobre nuestros usuarios y serán diferentes para cada uno de ellos.

En este capítulo explicaremos cómo crear y cerrar sesiones, así como las principales ventajas del paso de variables con el uso de sesiones.

4.4.1. Qué son y cómo funcionan

Mediante el uso de sesiones podremos realizar un seguimiento a nuestros usuarios durante el tiempo que estén conectados a la misma.

La sesión será creada desde que el usuario se conecte a nuestro sitio web, y a partir de este momento todas las solicitudes y visitas que realice a las diferentes páginas podrán ser seguidas gracias a la sesión que se creó al comienzo.

La creación de una sesión lleva asociada la creación de un identificador (id de sesión) de tal manera que relacionará de forma unívoca al usuario con dicha sesión, para de esta forma, mediante nuestras técnicas de programación, poder registrar una serie de variables como variables de sesión, que serán almacenadas en el servidor. La única información de una sesión que podrá ser consultada desde el lado del cliente será este Id. de sesión.

El Id. de sesión será almacenado como una "cookie" en el equipo cliente, aunque también podrá ser propagado mediante una URL.

Las operaciones básicas que realizaremos en el uso de sesiones serán las siguientes:

- Iniciar una sesión.
- Registrar variables de sesión.
- Utilizar variables de sesión.
- Anular las variables registradas y eliminar la sesión.

El enfoque de control de sesiones ha sufrido alguna modificación desde versiones anteriores de PHP, gracias a la aparición de las variables superglobales. En el caso de las sesiones, la variable global $_SESSION, en realidad es un array asociativo que va a contener las variables de sesión que estén disponibles para nuestro script actual.

Por otra parte, es necesario conocer que el soporte para las sesiones estará habilitado por defecto en PHP y que la ruta donde será almacenada toda la información relativa a las sesiones, se podrá consultar o modificar en la opción "SESSION.SAVE_PATH" del fichero PHP.INI.

4.4.2. Creación de una sesión

Para comenzar a beneficiarnos del uso de las sesiones, en primer lugar tendremos que ejecutar las instrucciones precisas para crearlas.

Esta operación la realizaremos mediante la función session _start().

Esta función, en primer lugar, comprobará si ya existe una sesión abierta mediante la comprobación de un Id. de sesión actual, y si no, creará una en el mismo momento. Podremos consultar el Id. de sesión mediante la función session_id(). Esta función será también utilizada cuando queramos propagar el identificador de la sesión de usuario a otras páginas dentro de nuestro sitio web para mantener la información del usuario.

Si se diera la circunstancia de que la sesión ya estuviera creada, realizaría una carga de las variables de sesión que estén registradas para que podamos volver a utilizarlas. Por lo tanto, una buena práctica será realizar una llamada a la función session_start() antes de realizar cualquier otro tipo de operaciones relacionadas con sesiones.

Existe una segunda alternativa para la creación de forma automática para las sesiones y es a través de la opción session.auto_start en el archivo de configuración PHP.INI. Pero tiene un inconveniente, y es que si habilitamos esta opción, no podremos utilizar objetos como variables de sesión.

Veamos un ejemplo sencillo en el que crearemos una sesión y mostraremos el valor del Id. de sesión:

EJEMPLO

```
<?php
echo " ";
session_start();
?>
<HTML>
<BODY>
<?php
echo "El identificador de la sesión es: " . session_id();
?>
</BODY>
</HTML>
```

Una vez que hemos creado nuestra primera sesión, podemos comenzar a registrar variables de sesión almacenando sus valores en el array asociativo $_SESSION, de tal forma que podemos especificar el nombre de la variable como el índice, y a continuación, su valor. Veamos un ejemplo:

EJEMPLO

```
<?php
session_start();
$_SESSION[“nombre”] = “Silvia Ruiz”;
$_SESSION[“edad”] = 32;
if (!isset($_SESSION[‘contador’])) {
  $_SESSION[‘contador’] = 0;
} else {
  $_SESSION[‘contador’]++;
}
echo “El valor de la variable NOMBRE es: “.$_
SESSION[“nombre”].”<br>”;
echo “El valor de la variable EDAD es: “.$_SESSION[“edad”].”<br>”;
echo “El valor de la variable CONTADOR es: “.$_
SESSION[“contador”].”<br>”;
?>
```

En el ejemplo anterior podemos aprender, en primer lugar, cómo se registran las variables en la variable superglobal $_SESSION. Pero además, podemos comprobar cómo esos valores son guardados cada vez que refrescamos la página con el botón “Actualizar” del navegador. En el caso de la variable “contador”, podemos comprobar que no solo mantiene su valor inicial, sino que es modificado en cada carga de la página, aumentando su valor, mientras el resto de las variables permanecen inalteradas.

En esta ocasión, hemos visualizado las variables y sus contenidos realizando un llamamiento directo, utilizando como índice el nombre de la variable, pero también podríamos utilizar el bucle “FOREACH” para recorrer todos los valores que estén almacenados en la variable $_SESSION de la siguiente forma:

EJEMPLO

```
echo “Mostrar las variables de sesión<br>”;
foreach ($_SESSION as $indice => $valor) {
echo “$indice: $valor<br>”;
}
```

Si necesitamos eliminar o dejar de registrar una variable que haya sido añadida previamente a $_SESSION, podremos utilizar la función unset() de la siguiente forma:

EJEMPLO

```
unset($_SESSION[‘contador’]);
```

4.4.3. Cerrar una sesión

En el apartado anterior hemos comprobado cómo podemos dejar de registrar una variable.

Tendremos que tener cuidado con el uso de la función unset() en el uso de sesiones, ya que si, por ejemplo, intentáramos destruir una sesión mediante la instrucción unset($_SESSION), estaríamos deshabilitando el registro de las variables a través de la variable superglobal $_SESSION.

Esto está directamente relacionado con la forma de cerrar una sesión. Para finalizar la sesión actual, en primer lugar, tendremos que borrar todas las variables registradas en el proceso, y a continuación, podremos destruir la sesión mediante la función **sesión_destroy()**.

Veamos un ejemplo de cómo sería el cierre correcto de una sesión:

EJEMPLO

```
$_SESSION = array();
session_destroy();
```

En la primera instrucción eliminamos todas las variables registradas en la sesión. Realmente estamos reinicializando la variable superglobal $_SESSION mediante la asignación de la función **array()**, que creará un nuevo array vacío.

A continuación, destruimos la sesión de forma definitiva.

4.4.4. Paso de variables por sesión

Existen tres formas de mantener y continuar una sesión mientras el usuario sigue navegando por nuestra página web:

- Mediante la propagación del Id. de sesión por URL.
- Por cookies.
- Propagación automática.

4.4.4.1. Por URL

Mediante este método, podremos enviar el Id. de sesión en las URL como un parámetro más. Este método, tendrá sus ventajas e inconvenientes.

Por una parte, será más fiable que las cookies con respecto a que nos aseguramos de que el cliente siempre recibirá el Id. de sesión, mientras que con las cookies, no podremos asegurar que todos los clientes las tengan habilitadas en su navegador.

Por otra parte, en lo relativo a la seguridad, será menos fiable, ya que exponer el PHPSESSID de esta forma nos podrá traer problemas, ya que se está haciendo público dicho identificador a otras personas o programas que pudieran estar al acecho.

En este caso, podremos hacer uso de la constante "SID" que va a contener el nombre de la sesión y el identificador de la forma "name=ID" o una cadena vacía, de tal forma.

Veamos un ejemplo de cómo sería el paso de Id. de sesión mediante propagación por URL.

En primer lugar, tendríamos el archivo donde estableceríamos el valor de las variables que queremos registrar, y donde crearemos el enlace para propagar el Id. de sesión:

EJEMPLO

```
<?php
session_start();
$_SESSION["nombre"] = "Silvia Ruiz";
$_SESSION["edad"] = 32;
if (!isset($_SESSION['contador'])) {
  $_SESSION['contador'] = 0;
} else {
  $_SESSION['contador']++;
}
$id_sesion=SID;
echo  "<a  href=\"sesion2.php?$id_sesion\">Pasar  variables</
a>";
?>
```

Por otra parte, tendríamos el fichero "sesion2.php" donde para comprobar que todo se ha recibido correctamente, visualizamos todos los valores:

EJEMPLO

```
<?php
session_start();
echo "Mostrar las variables de sesión<br>";
foreach ($_SESSION as $indice => $valor) {
      echo "$indice: $valor<br>";
}
?>
```

4.4.4.2. Por Cookie

Según podemos observar el manual oficial de PHP, las sesiones que están basadas en cookies son más seguras que las que están basadas en URL. A partir de la versión 5.2.13 de PHP, podremos encontrar por defecto en el fichero PHP.INI los siguientes valores para configuraciones de las sesiones, que podrán ser modificadas en el mismo momento en el que estemos ejecutando nuestro script:

- session.use_cookies. Mediante esta opción podremos especificar si PHP usará cookies para propagar las sesiones. Por defecto se establece en 1.
- session.use_only_cookies. Mediante esta opción podremos especificar si queremos que PHP solo use cookies para almacenar el Id. de sesión en el navegador del cliente, y por lo tanto, prohibiendo la opción de propagarlo por URL. Su valor predeterminado es 1, y está fuertemente recomendado para evitar la suplantación de sesiones (hijacking).

NOTA

Hay que contemplar la posibilidad de que el usuario tenga deshabilitadas las cookies en su ordenador, ya que no podremos obligarle a que las tenga habilitadas.

4.4.4.3. Propagación automática

Afortunadamente, tenemos una tercera alternativa para la propagación del Id. de sesión. Es decir, sin tener que utilizar la constante "SID" para dicha propagación, y de esta forma evitar el riesgo que supone para la seguridad filtrar "SID".

Realizar esta propagación automática será posible gracias a la opción que podemos encontrar en PHP.INI y cuyo nombre es session.use_trans_sid. Mediante esta opción podremos especificar que el Id. de sesión sea comunicado a través de URL entre el navegador y el usuario de forma transparente. De esta forma, no tendremos que incluir el identificador en cada petición o respuesta. Por defecto encontraremos su valor a 0. En este caso, podremos hacer uso de la constante "SID" que va a contener el nombre de la sesión y el identificador de la forma "name=ID" o una cadena vacía, si estamos usando la opción de sesiones basadas en cookies. Si queremos usar sesiones que no estén basadas en cookies sino en URL, deberemos de poner la opción session.use_only_cookies a 0 y a continuación, esta opción a 1.

Para habilitarla, podremos editar el fichero PHP.INI tal y como hemos indicado, o bien ejecutar la función ini_set() de la siguiente forma:

EJEMPLO

```
ini_set("sesión.use_trans_sid","1");
```

De esta forma, en el enlace que proporcionamos en nuestro anterior ejemplo para pasar al otro fichero, donde comprobaremos que las variables han sido pasadas correctamente, bastará con escribir lo siguiente:

EJEMPLO

```
echo "<a href=\"sesion2.php">Pasar variables</a>";
```

4.4.5. Manejo de sesiones con clases

Para seguir practicando con el manejo de sesiones, y profundizar algo más en el uso de clases y objetos, vamos a realizar un ejemplo que va a constar de tres archivos:

- clase_sesion.php. Que va a contener la clase que utilizaremos para las principales funciones.
- prueba.php. Realizará la instancia de la clase y registrará los primeros valores mediante los métodos de la clase y realizará el paso de valores al siguiente script, prueba2.php.
- prueba2.php. Mostrará por pantalla los valores actuales de la sesión mediante la instancia de la clase y recuperación de valores.

Veamos el código con el que definimos la clase, en el fichero clase_sesion.php:

EJEMPLO

```
<?php
class Sesion
{
       function Sesion()
       {
           session_start();
       }
       public function set($nombre,$valor)
       {
           $_SESSION[$nombre] = $valor;
       }
       public function get($nombre)
       {
           if (isset($_SESSION[$nombre])) {
               return $_SESSION[$nombre];
           } else {
               return false;
           }
       }
       public function borrar_variable($nombre)
       {
           unset ($_SESSION[$nombre]);
       }
       public function  borrar_sesion()
       {
           $_SESSION = array();
           session_destroy();
       }
}
?>
```

Veamos el código con el que instanciamos la clase, y registramos los primeros valores, en el fichero clase_sesion.php:

EJEMPLO

```
<?php
require("clase_sesion.php");
$sesion = new Sesion();
$sesion->set("nombre","Silvia Ruiz");
$sesion->set("edad",32);
$id_sesion = SID;
echo  "<a  href=\"prueba2.php?$id_sesion\">Pasar  variables</
a>";
?>
```

Veamos el código con el que instanciamos la clase, y recuperamos valores de la sesión, eliminamos valores y eliminamos la sesión mediante los métodos de la clase original, en el fichero clase_sesion.php:

EJEMPLO

```
<?php
require("clase_sesion.php");
$sesion = new Sesion();
echo $sesion->get("nombre") . "<br>";
echo $sesion->get("edad") . "<br>";
$sesion->borrar_variable("nombre");
$sesion->borrar_sesion();
?>
```

4.4.6. Ventajas y desventajas del uso de sesiones

Una sesión, como una cookie, nos suministra una forma de poder guardar datos de un usuario en una serie de páginas. La principal diferencia entre ambas, es que las cookies son almacenadas en el equipo del cliente y los datos de las sesiones son guardados en el servidor. Esto puede ser una ventaja o un inconveniente según se mire.

Pero las sesiones tienen otras ventajas sobre las cookies:

- Las sesiones por lo general son más seguras, porque los datos no son enviados entre el cliente y el servidor de forma repetida.
- Las sesiones nos permitirán almacenar más información que la que podemos almacenar en una cookie.
- Aunque un usuario no acepte trabajar con cookies en su navegador, podremos seguir trabajando igualmente.

4.5. Manejo de ficheros

PHP nos va a ofrecer un conjunto de funciones y procedimientos mediante los cuales podremos acceder a un fichero de texto para poder leerlo, modificarlo, añadir contenido o eliminarlo.

Cualquier operación que suponga el manejo de ficheros llevará tres operaciones asociadas:

- La primera será la apertura del fichero, donde especificaremos el tipo de acción que vamos a realizar sobre dicho fichero.
- La segunda será ejecutar la acción.
- Por último, habrá que cerrar el fichero.

4.5.1. Abrir un fichero (FOPEN)

Para abrir un fichero utilizaremos la instrucción ***fopen().*** Mediante esta función podremos abrir un fichero ya sea vía HTTP o FTP.

Su sintaxis será la siguiente:

EJEMPLO

```
fopen ( string $filename , string $mode [, bool $use_inclu-
de_path = false ] )
```

La cadena ***$filename*** será el fichero que queramos abrir y tendrá que contener la ruta completa del archivo.

La cadena ***$mode*** tendrá que contener el modo de apertura para indicar la forma en la que queremos que el fichero sea abierto. Un archivo puede ser abierto con los siguientes modos:

- **'r'.** Apertura para solo lectura; coloca el puntero al archivo al principio del archivo.
- **'r+'.** Apertura para lectura y escritura; coloca el puntero al archivo al principio del archivo.
- **'w'.** Apertura para solo escritura; coloca el puntero al archivo al principio del archivo y trunca el archivo a longitud cero. Si el archivo no existe, se intenta crear.
- **'w+'**. Apertura para lectura y escritura; coloca el puntero al archivo al principio del archivo y trunca el archivo a longitud cero. Si el archivo no existe se intenta crear.

- **'a'.** Apertura para solo escritura; coloca el puntero al archivo al final del archivo. Si el archivo no existe, se intenta crear.
- **'a+'**. Apertura para lectura y escritura; coloca el puntero al archivo al final del archivo. Si el archivo no existe, se intenta crear.
- **'x'.** Creación y apertura para solo escritura; coloca el puntero al archivo al principio del archivo. Si el archivo ya existe, la llamada a fopen() fallará devolviendo FALSE y generando un error de nivel E_WARNING. Si el archivo no existe, se intenta crear.
- **'x+'.** Creación y apertura para lectura y escritura; de otro modo tiene el mismo comportamiento que 'x'.
- **'c'**. Abrir el archivo para solo escritura. Si el archivo no existe, se crea. Si existe, no es truncado (a diferencia de 'w'), ni la llamada a esta función falla (como en el caso con 'x'). El puntero al archivo se posiciona en el principio del archivo. Esto puede ser útil si se desea obtener un bloqueo asistido (véase flock()) antes de intentar modificar el archivo, ya que al usar 'w' se podría truncar el archivo antes de haber obtenido el bloqueo (si se desea truncar el archivo, se puede usar ftruncate() después de solicitar el bloqueo).
- **'c+'**. Abrir el archivo para lectura y escritura; de otro modo tiene el mismo comportamiento que 'c'.

El tercer parámetro opcional ***$use_include_path*** puede ser establecido a '1' o TRUE, si se desea buscar un archivo en ***include_path*** también. Para Linux, la ruta por defecto es include_path=".:/php/includes" y para Windows la ruta por defecto es include_path=".;c:\php\includes".

Podemos utilizar también la función **die()** para que en el caso de que se haya producido un error en la apertura del fichero, se cierre la conexión y envíe un mensaje avisando del error. Veamos un ejemplo de ello:

EJEMPLO

```
<?php
@fopen( "contador.txt",  "r") or die("El fichero no ha podido
ser abierto");
?>
```

Si observamos, hemos añadido delante de la instrucción **fopen()** el símbolo **"@".** Con este símbolo nos aseguramos que no visualizará los mensajes de error del sistema, y solo mostrará el que nosotros especificamos en la función **die().**

La función fopen() devolverá un puntero al comienzo del archivo para que pueda ser utilizado, y lo tendremos que guardar en una variable **$fichero=fopen(fichero,modo).**

Por ejemplo, si quisiéramos abrir el fichero "**contador.txt**" en modo lectura y posteriormente en modo escritura, tendría que hacerse de la siguiente forma:

EJEMPLO

```
<?php
$fichero=@fopen( "contador.txt",  "r") or die("El fichero no ha
podido ser abierto");
$fichero=@fopen( "contado2r.txt",  "w");
?>
```

Cuando abrimos el fichero en modo escritura, no hace falta que incluyamos el mensaje de error, puesto que si el fichero no existe, será creado en ese mismo momento.

4.5.2. Cerrar un fichero (FCLOSE)

Una vez que hayamos todas las operaciones que necesitemos sobre un fichero, será siempre conveniente cerrarlo de forma correcta.

Para ello, utilizaremos la función fclose().

Si tenemos varios ficheros abiertos, y solo deseamos cerrar uno de ellos, le podremos pasar como argumento de la función la variable que contiene el apuntador de dicho fichero.

Veamos como cerrar el fichero abierto anteriormente:

EJEMPLO

```
<?php
$fichero=@fopen( "contador.txt",  "r") or die("El fichero no ha
podido ser abierto");
$fichero=@fopen( "contado2r.txt",  "w");
fclose($fichero);
?>
```

4.5.3. Leer un fichero (FGETS)

Con la función fgets() podremos leer cadenas de texto del fichero indicado.

Su sintaxis es la siguiente:

EJEMPLO

```
string fgets ($fichero [, int $longitud ] )
```

Mediante esta función podremos obtener un número de caracteres según especifiquemos en la variable $longitud. En realidad, esta variable tendrá un número cuyo valor será superior en 1 a los caracteres que finalmente serán mostrados en cada línea. La lectura de esta cadena acabará si el carácter de retorno de carro es encontrado o si se llega al final del archivo.

Si no especificamos una longitud, seguirá leyendo hasta que alcance el final de línea.

Veamos un ejemplo donde le especificaremos un parámetro de longitud:

NOTA

```
<?php
$fichero = @fopen("contador.txt", "r");
if ($fichero) {
    while (($linea = fgets($fichero,5)) !== false) {
        echo $linea;
        echo "<br>";
    }
    fclose($fichero);
}
```

Supongamos que el fichero tiene el siguiente contenido:

EJEMPLO

LA DONNA E MOBILE
CUAL PIU MAL VENTO

Tendríamos que obtener un resultado como el siguiente:

LA D
ONNA
E M
OBIL
E
CUAL
PIU
MAL
VEN
TO

Figura 4.69. Resultado de la función fgets()

4.5.4. Escribir un fichero (FPUTS)

Seguramente, la operación que más realicemos será la de escribir texto en los ficheros. Para ello utilizaremos la función **fputs(),** que tendrá la siguiente sintaxis:

EJEMPLO

```
int fputs ($fichero, string $cadena [, int $longitud ] )
```

Esta función escribirá el contenido de la variable **$cadena** en el archivo que esté apuntando el apuntador **$fichero.**

Si especificamos el parámetro opcional **$longitud**, el proceso de escritura se detendrá al haber escrito la cantidad de bytes especificados en dicha variable, a no ser que se llegue al final del fichero.

La función **fputs()** devolverá el número de bytes escritos, o el valor **FALSE,** si se produjo algún tipo de error en el proceso de escritura.

Veamos un ejemplo:

EJEMPLO

```
<?php
$fichero=”contador.txt”;
$cadena = “Estas palabras queremos añadir”;
if ($apuntador= fopen($fichero,”a”))
{
    fputs($apuntador,$cadena);
}
fclose($apuntador);
?>
```

4.6. Seguridad

En este capítulo vamos a explicar diferentes medidas de seguridad que podemos adoptar para aportar seguridad a nuestras aplicaciones PHP.

4.6.1. Seguridad en aplicaciones web

A medida que vayamos adquiriendo conocimiento y seguridad en la construcción de páginas web, ya sea con PHP+MySQL o con cualquier otro lenguaje de programación, podremos ir creando cada vez estructuras más complejas, en función de cuáles sean nuestras necesidades.

No obstante, existe una arquitectura común para todas las aplicaciones web, distribuida en tres niveles por sus diferentes funcionalidades.

Para empezar, tendremos que tener en cuenta el sistema de interfaz de usuario, es decir, la parte que gestiona la interacción entre nuestros usuarios y nuestro sistema, y que en una aplicación web típica es el navegador de Internet.

El siguiente nivel es donde reside la lógica de la aplicación. En esta parte están implicados los servidores web y los servidores de aplicaciones, que utilizarán diferentes tecnologías para crear conocimiento o procesar la información con un fin concreto.

En el tercer nivel se encuentra el almacén de datos, es decir, el repositorio de toda la información relativa a nuestra página web. Ejemplos de estos repositorios pueden ser un árbol LDAP, una base de datos relacional, un almacén con datos en XML o un simple fichero de datos.

Una vez conocidos los tres niveles, es obvio concluir que existe una relación funcional entre todos ellos. Por ejemplo, nuestros usuarios, mediante el envío de información de su navegador de Internet preferido, podrán interactuar con el servidor web donde tengamos alojada nuestra aplicación, que a su vez, tendrá una relación con nuestro almacén de datos, para extraer información o para escribir información del mismo.

Por lo tanto, cuando tengamos que tomar decisiones en lo relativo a la seguridad en nuestras aplicaciones, tendremos que pensar en soluciones para cada uno de estos niveles.

En este capítulo, apuntaremos algunas de las principales soluciones que podemos implementar mediante el lenguaje de programación PHP para aportar seguridad a nuestras aplicaciones.

4.6.2. Seguridad en PHP

Para comenzar a hablar de temas relativos a seguridad en el lenguaje de programación PHP, comenzaremos por referirnos a su intérprete. Al poder instalarlo como módulo de un servidor como Apache o bien como binario CGI, tendrá permisos para acceder a archivos, ejecutar comandos u otro tipo de operaciones que puedan ser realizadas en el servidor.

Éste será el primer problema de seguridad al que nos enfrentaremos.

En los siguientes apartados, explicaremos algunos consejos generales de seguridad, diferentes combinaciones de opciones de configuración y las situaciones en que pueden ser útiles, describiendo diferentes consideraciones relacionadas con la programación de acuerdo a diferentes niveles de seguridad.

4.6.3. Manejo de errores

PHP nos va a ofrecer la posibilidad de poder concretar la claridad con la que queremos que los errores nos sean ofrecidos por el intérprete.

Esta tarea la podremos realizar, en primer lugar, conociendo los diferentes tipos de eventos de error que pueden ser producidos, y a continuación, pudiendo modificar estos tipos de eventos según nuestras necesidades. Por defecto, PHP ofrecerá informes sobre todos los tipos de errores que se puedan producir, excepto de los avisos.

Para poder consultar estos tipos de errores, podemos acudir a la siguiente tabla, donde veremos una serie de constantes definidas que podremos utilizar para variar este nivel de informes:

Valor	Nombre	Significado
1	E_ERROR	Nos informará de los errores graves en tiempo de ejecución
2	E_WARNING	Nos informará de los errores que no sean graves en tipo de ejecución
4	E_PARSE	Nos informará de los informes de análisis
8	E_NOTICE	Nos informará de los avisos, que serán notificaciones de que algo que hemos escrito es un error
16	E_CORE_ERROR	Nos informará de los fallos que se produzcan al iniciar el motor de PHP
32	E_CORE_WARNING	Nos informará de los fallos que no sean graves al arrancar el motor de PHP
64	E_COMPILE_ERROR	Nos informará de los errores que se hayan producido en la tarea de compilación
128	E_COMPILE_WARNING	Nos informará de los errores que no sean graves que se hayan producido en la tarea de compilación
256	E_USER_ERROR	Nos informará de los errores que haya provocado el usuario
512	E_USER_WARNING	Nos informará de las advertencias que pueda provocar el usuario
1024	E_USER_NOTICE	Nos informará de los avisos que pueda provocar el usuario
2047	E_ALL	Nos informará de todos los errores y advertencias que se puedan producir
2048	E_STRICT	Nos informará del uso de un comportamiento obsoleto y que no es recomendado. No se incluirá en E_ALL pero resultará útiles para poder modificiar el factor del código.

Figura 4.70. Tabla de errores.

En la anterior tabla, cada constante es el resultado de un tipo de error en concreto que podremos destacar o ignorar si lo estimamos oportuno. Si, por ejemplo, solo utilizamos el nivel de error E_ERROR, solo informaremos de los errores graves que se puedan producir.

Podremos realizar combinaciones de estas constantes para poder personalizar nuestros distintos niveles de error.

Por ejemplo, el nivel de error que se muestra como predeterminado, mediante el cual se nos informará de todos los errores que no sean avisos, se podrá especificar de la siguiente manera:

EJEMPLO

```
E_ALL & ~E_NOTICE
```

Como podemos observar en el anterior ejemplo, hemos utilizado dos operadores aritméticos. Por un lado el símbolo "&" será el equivalente al operador AND y el símbolo "~" será equivalente al operador NOT. Por lo tanto, esta expresión también podría ser escrita de la siguiente forma:

EJEMPLO

```
E_ALL AND NOT E_NOTICE
```

El nivel de error E_ALL es el equivalente a la combinación del resto de los errores, menos el error E_STRICT y podría ser sustituido por los otros niveles utilizando el operador OR, que puede ser representado por el símbolo "|", de la siguiente forma:

EJEMPLO

```
E_ERROR | E_WARNING | E_PARSE | E_NOTICE | E_CORE ERROR | E_
CORE_WARNING | E_COMPILE_ERROR | E_COMPILE_WARNING | E_USER_
ERROR | E_USER_WARNING | E_USER_NOTICE
```

Igualmente, el nivel de informes de error predeterminado que hemos comentado anteriormente también podría ser especificado por todos los errores menos los avisos combinados con OR:

EJEMPLO

```
E_ERROR | E_WARNING | E_PARSE | E_CORE ERROR | E_CORE_WAR-
NING | E_COMPILE_ERROR | E_COMPILE_WARNING | E_USER_ERROR |
E_USER_WARNING | E_USER_NOTICE
```

Para poder modificar estos niveles de error podemos acudir al fichero de configuración PHP.INI o mediante una secuencia de comandos.

En lo que respecta al archivo PHP.INI, podremos modificar estas configuraciones en las siguientes cuatro líneas destacadas:

EJEMPLO

```
error_reporting = E_ALL & E~NOTICE
display_errors = On
log_errors = Off
track_errors = Off
```

Estos parámetros globales nos servirán para los siguientes propósitos:

- Informarnos de todos los errores menos los avisos.
- Mostrarnos los mensajes de error como HTML en resultado estándar.
- No registrar los mensajes de error en el disco.
- No realizar el seguimiento de errores y almacenar el error en la variable.

Cuando estemos en una fase de depuración del código, una buena práctica será subir el nivel de error_reporting.

En la fase de código de producción, si queremos suministrar mensajes de errores propios, otra buena práctica podría ser desactivar display_errors y activar log_errors, mientras seguimos manteniendo el nivel de error_reporting en un valor elevado. Con estas prácticas, podremos hacer referencia a errores detallados en los registros por si apareciera algún problema.

Si activamos track_errors, podremos revisar los errores de nuestro propio código, y no tener que esperar a que PHP nos suministre su funcionalidad de manera predeterminada.

No será necesario conservar el comportamiento del procesamiento de errores de forma predeterminada por PHP, ni tenemos por qué utilizar los mismos parámetros en todos los archivos, ya que podremos modificar este nivel de errores mediante la función error_reporting().

Podremos pasarle como parámetros a esta función las constantes de niveles de informes de errores o una combinación de las mismas, y estaríamos realizando la misma configuración como si lo hiciéramos directamente en el archivo PHP.INI. La función devolverá el nivel de informe de error que había anteriormente.

Podríamos utilizar esta función, por ejemplo, de la siguiente forma:

EJEMPLO

```
//desactivamos el informe de errores
$nivel_anterior = error_reporting(0);
//insertamos a partir de aquí el código que va generar las
advertencias
//volvemos a activar el informe de errores
error_reporting($nivel_anterior);
```

Con este ejemplo, estaremos desactivando los informes de error, permitiéndonos ejecutar el código que nos está generando advertencias que no deseamos que en principio sean vistas.

No obstante, nunca es aconsejable dejar desactivados los informes de error de manera permanente, ya que nos complicará la localización y la corrección de los errores en el código.

Podremos mostrar nuestros propios errores mediante la función trigger_error(). A esta función tendremos que pasarle como argumentos un mensaje de error, y de manera opcional, podremos asignarle un tipo de error a elegir entre E_USER_ERROR, E_USER_WARNING o USER_NOTICE, siendo el predeterminado E_USER_NOTICE si no se especifica ninguno.

La forma de utilizarla sería de la siguiente forma:

EJEMPLO

```
trigger_error("Este es el mensaje de error que queremos
mostrar",E_USER_ERROR)";
```

4.6.4. Nombres de ficheros

PHP está sujeto a la seguridad integrada en la mayoría de sistemas de servidores con respecto a los permisos de archivos y directorios. Esto permite controlar qué archivos en el sistema de archivos se pueden leer. Se debe tener cuidado con los archivos que son legibles, para garantizar que son seguros para la lectura por todos los usuarios que tienen acceso al sistema de archivos.

Desde que PHP fue diseñado para permitir el acceso a nivel de usuarios para el sistema de archivos, es perfectamente posible escribir un script PHP que le permita leer archivos del sistema como /etc/passwd, modificar sus conexiones de red, enviar trabajos de impresión masiva, etc. Esto tiene algunas implicaciones obvias, es necesario asegurarse que los archivos que se van a leer o escribir son los apropiados.

Veamos un ejemplo sencillo donde podemos observar rápidamente lo vulnerable que puede ser un sistema si no prestamos especial atención o no somos conscientes de las consecuencias que pueden acarrear nuestras decisiones a la hora de desarrollar nuestro código.

Comenzamos por desarrollar un script donde un usuario indica que quiere borrar un archivo en su directorio home. Mediante el siguiente código suponemos que previamente hemos introducido en un formulario el nombre del usuario y del archivo que queremos borrar. El siguiente código se encargaría de borrarlo:

EJEMPLO

```
<?php
// eliminar un archivo del directorio personal del usuario
$usuario = $_POST['nombre_usuario_enviado'];
$fichero = $_POST['nombre_fichero_enviado'];
$directoriohome  = "/home/$usuario";
unlink("$directoriohome/$fichero");
echo "El archivo ha sido eliminado!";
?>
```

Como el nombre de usuario y el nombre del archivo son enviados desde un formulario, podríamos escribir un nombre de archivo y un nombre de usuario que pertenecen a otro usuario, o también podríamos eliminar el archivo a pesar que se supone que no estaría permitido hacerlo. Para evitarnos todos estos problemas, deberíamos siempre implementar una solución de autenticación en la operación. Las consecuencias de un mal desarrollo como el anterior podemos observarlas rápidamente si las variables enviadas son "../etc/" y "passwd". El código entonces se ejecutaría efectivamente como:

EJEMPLO

```
<?php
// eliminar un archivo del directorio personal del usuario
$usuario = $_POST['nombre_usuario_enviado']; // "../etc"
$fichero = $_POST['nombre_fichero_enviado']; // "passwd"
$directoriohome  = "/home/$usuario"; // "/home/../etc"
unlink("$directoriohome/$fichero"); // "/home/../etc/passwd"
echo "El archivo ha sido eliminado!";
?>
```

Como premio acabamos de eliminar el fichero de passwords de nuestro sistema, que reside en la ruta "/etc/passwd".

Principalmente, serán dos las medidas a las que deberemos prestar especial atención:

- Establecer permisos limitados al usuario web de PHP.
- Revisar todas las variables que se envían.

Además, en función del sistema operativo donde resida nuestro servidor, tendremos una gran variedad de archivos a los que tendremos que vigilar:

- Entradas de dispositivos (/dev/ o COM1).
- Archivos de configuración (archivos /etc/ y archivos .ini).
- Las conocidas carpetas de almacenamiento (/home/, Mis documentos), etc.

Por esta razón, por lo general es más fácil crear una política en donde se prohíba todo excepto lo que expresamente se permite.

Tendremos que tener especial cuidado también con el uso de las funciones "include" y "require". Veamos el siguiente código:

EJEMPLO

```
include("/usr/local/lib/bienvenida/$username");
```

Este código pretende mostrar un mensaje de bienvenida personalizado para el usuario. Aparentemente no es peligroso, pero ¿qué ocurriría si el usuario introduce como nombre la cadena "../../../../etc/passwd"? Se mostraría el fichero de passwords del sistema.

La directiva más útil en relación con la seguridad en PHP es open_basedir. Esta directiva indica a PHP a qué ficheros puede acceder y a cuáles no. El valor de esta directiva es una lista de prefijos de ficheros separados por una coma en Unix y por punto y coma en Windows.

Las restricciones configuradas aquí, afectan a los scripts de PHP y a los ficheros de datos. La opción recomendada es activar esta opción incluso en aquellos servidores con un único sitio web, y debe de apuntar un nivel por encima del directorio raíz del servidor web.

Una posible configuración sería:

EJEMPLO

```
open_basedir = /var/www/
```

Es importante recordar la diferencia entre establecer las restricciones a un prefijo frente a establecer las restricciones a un directorio, por ejemplo:

EJEMPLO

```
open_basedir=/var/www
// permitiría acceso tanto a ficheros de /var/www como de /
var/www2
open_basedir=/var/www/
// permitiría acceso a los ficheros que estén únicamente en /
var/www/
```

4.6.5. Encriptación de textos

Uno de los aspectos fundamentales para aumentar la seguridad en nuestro código en PHP, es dotar de seguridad y encriptación a las contraseñas que usemos en nuestras aplicaciones. Por ejemplo, en el proceso de validación de usuario y contraseña.

La forma habitual de trabajo es recoger la contraseña del usuario mediante un formulario, y almacenarla ya encriptada en la base de datos, de tal forma que no sea legible ni para el propio usuario.

Seguramente, cuando alguna vez hayamos olvidado nuestra contraseña de acceso a alguna de nuestras páginas web favoritas, habremos podido comprobar que la solución que nos ofrece la propia página web es enviarnos a nuestro correo electrónico una nueva contraseña, en lugar de enviarnos la que originalmente escribimos. La razón de la creación de esta nueva contraseña es que la original fue almacenada encriptada y los algoritmos utilizados en la encriptación son de una sola vía, es decir, pueden aplicarse para encriptar, pero no para desencriptar.

Esta propiedad la podremos encontrar en las siguientes tres funciones para encriptar con php: crypt(), md5() y sha1().

Crypt()

La función **crypt()** devolverá el hash de un string utilizando el algoritmo basado en DES estándar de Unix o algoritmos alternativos que puedan estar disponibles en el sistema. Su sintaxis es la siguiente:

EJEMPLO

```
string crypt ( string $str [, string $salt ] )
```

Algunos sistemas operativos soportan más de un tipo de hash. De hecho, a veces el algoritmo estándar DES es sustituido por un algoritmo basado en MD5. El tipo de hash se dispara mediante el argumento salt. Antes de 5.3, PHP determinaba los algoritmos disponibles en el momento de la instalación, basado en el crypt() del sistema. Si no se proporciona salt, PHP intentará auto-generar ya sea un salt estándar de dos caracteres (DES) o uno de doce caracteres (MD5), dependiendo de la disponibilidad del crypt() de MD5.

En sistemas donde la función crypt() soporta múltiples tipos de hash, podremos utilizar las siguientes constantes en 0 o 1, en función de si el tipo dado está disponible:

- CRYPT_STD_DES - Hash estándar basado en DES con un *salt* de dos caracteres del alfabeto "./0-9A-Za-z". Utilizar caracteres no válidos en el salt causará que crypt() falle.
- CRYPT_EXT_DES - Hash extendido basado en DES. El *salt* es un string de nueve caracteres que consiste en un guion bajo seguido de 4 bytes del conteo de iteraciones y 4 bytes del salt. Estos están codificados como caracteres imprimibles, 6 bits por carácter, por lo menos, el carácter significativo primero. Los valores del 0 al 63 son codificados como "./0-9A-Za-z". Utilizar caracteres no válidos en el salt causará que crypt() falle.
- CRYPT_MD5 - Hash MD5 con un salt de doce caracteres comenzando con 1.
- CRYPT_BLOWFISH - Hash Blowfish con un *salt* como sigue: "$2a$", un parámetro de costo de dos dígitos, "$" y veintidós dígitos en base64 del alfabeto "./0-9A-Za-z". Utilizar caracteres fuera de este rango en el salt causará que crypt() devuelva un string de longitud cero.
- CRYPT_SHA256 - Hash SHA-256 con un *salt* de dieciséis caracteres prefijado con 5. Si el strnig del salt inicia con 'rounds=<N>$', el valor numérico de N se utiliza para indicar cuántas veces el bucle del hash se debe ejecutar, muy similar al parámetro de costo en Blowfish.
- CRYPT_SHA512 - Hash SHA-512 con un *salt* de dieciséis caracteres prefijado con 6. Si el string del salt inicia con 'rounds=<N>$', el valor numérico de N se utiliza para indicar cuántas veces el bucle del hash se debe ejecutar, muy similar al parámetro de costo en Blowfish.

Veamos un ejemplo de sus posibles usos:

EJEMPLO

```
<?php
$password = 'ladonnaemobile';
if (CRYPT_STD_DES == 1) {
    echo 'Standard DES: ' . crypt($password, 'dl') . "\n";
    echo "<br>";
}
if (CRYPT_EXT_DES == 1) {
    echo 'Extended DES: ' . crypt($password, '_H9..escr');
    echo "<br>";
}
if (CRYPT_MD5 == 1) {
    echo 'MD5:           ' . crypt($password, '$1$escribelo-
quequieras$');
    echo "<br>";
}
if (CRYPT_BLOWFISH == 1) {
          echo   'Blowfish:            '   .   crypt($password,
'$2a$07$escribeloquequieras$');
    echo "<br>";
}
if (CRYPT_SHA256 == 1) {
    echo 'SHA-256:      ' . crypt($password, '$5$rounds=5000
$escribeloquequieras$');


    echo "<br>";
}
if (CRYPT_SHA512 == 1) {
    echo 'SHA-512:      ' . crypt($password, $6$rounds=5000$
escribeloquequieras$');
    echo "<br>";
}
?>
```

Sha1

Calcula el hash **sha1** de un string. Utiliza el algoritmo *Secure Hash Algorithm* 1 (SHA1). SHA-0 y SHA-1 producen una salida resumen de 160 bits (20 bytes) de un mensaje que puede tener un tamaño máximo de 264 bits, y se basa en principios similares a los usados por el profesor Ronald L. Rivest del MIT en el diseño de los algoritmos de resumen de mensaje MD4 y MD5.

Su sintaxis es la siguiente:

EJEMPLO

```
string sha1 ( string $str [, bool $raw_output = false ] )
```

Donde "*str*" será la cadena a encriptar. Si se establece el "*raw_output"* opcional en TRUE, entonces el resumen sha1 será devuelto en formato binario sin tratar con una longitud de 20, de otra manera, el valor retornado será un número hexadecimal de 40 caracteres.

Veamos un sencillo ejemplo de su uso:

EJEMPLO

```
<?php
function cryptconsha1($string) {
// Creamos un salt
 $salt = sha1($string."%*4!#$;.k~'(_@");
 $string = sha1("$salt$string$salt");

  return $string;
}

echo "<br>";
echo 'SHA1: '.cryptconsha1($password);
?>
```

MD5

Calcula el hash **md5** de un string, utilizando el algoritmo de "resumen de mensajes" o MD5 *Message-Digest Algorithm*.

Su sintaxis es la siguiente:

EJEMPLO

```
string md5 ( string $str [, bool $raw_output = false ] )
```

Donde "*str*" será la cadena a encriptar. Si se establece el "*raw_output"* opcional en TRUE, entonces el resumen md5 será devuelto en formato binario sin tratar con una longitud de 16.

Ésta es una de las funciones más utilizadas para encriptar contraseñas, ya que siempre genera el mismo resultado para la misma cadena y también es de una sola vía, es decir, una vez codificada una cadena no es posible volver a descodificarla.

Veamos un sencillo ejemplo de su uso:

EJEMPLO

```
<?php
function cryptconmd5($string) {
// Creamos un salt
 $salt = md5($string."%*4!#$;.k~'(_@");
 $string = md5("$salt$string$salt");
  return $string;
}
echo "<br>";
```

4.6.6. Inyección SQL

Como veremos a partir de la próxima unidad, será muy común trabajar contra una base de datos (MySQL, PostgreSQL, Oracle, etc.), a través de las cuales realizaremos unas consultas para obtener datos que estén almacenados en las mismas. Estas consultas dependerán de parámetros que recibiremos por los métodos GET o POST. Al utilizar estas consultas con formularios, una de las desventajas es que permitimos al usuario de cada web que en cierta manera modifique las consultas SQL de nuestra web.

Un ataque de inyección SQL es una vulnerabilidad existente en el proceso de validación de entradas a una base de datos en una aplicación. Esta aplicación puede ser, por ejemplo, un formulario de entrada de datos o cualquier otra que necesite acceder a una base de datos para poder realizar su función. Mediante un ataque de inyección SQL se podrán ejecutar sentencias SQL no deseadas, que pueden ser desde borrado de tablas hasta cambios de permisos o la obtención de contraseñas u otros datos sensibles de estas tablas.

Mediante esta técnica, un usuario malintencionado podría construir una consulta SQL para obtener resultados no deseados. El caso más habitual suele ser el de suplantación de identidad en un formulario para realizar las tareas de login para permitir el acceso a una zona restringida, sin que sea necesario conocer ningún usuario y/o contraseña.

Supongamos la siguiente situación: tenemos un nombre de usuario "*admin*" y una contraseña "*abcde*" guardados en una base de datos. Para realizar la identificación obtenemos los parámetros por POST mediante las variables $_POST['usuario'] y $_POST['clave']. Esta introducción de datos por parte del usuario, podremos guardarla en unas variables, por ejemplo, de la siguiente forma:

EJEMPLO

```
$usua = $_POST['usuario'];
$clav = $_POST['clave'];
```

Tras ello realizamos la siguiente consulta SQL que almacenaremos en otra variable para posteriormente ser ejecutada.

EJEMPLO

```
$sql="SELECT * FROM usuarios WHERE user = '$usua' AND
password='$clav'";
```

Con la utilización de la anterior consulta, estaremos intentando comprobar si un usuario con ese password existe en nuestra tabla de USUARIOS. Es decir, con el usuario anteriormente citado, la consulta sería como la siguiente:

EJEMPLO

```
$sql="SELECT * FROM usuarios WHERE user = 'admin' AND
password='abcde'";
```

Si existe un usuario con ese nombre y usuario, es que la identificación es correcta y por tanto, permitiremos que el usuario acceda a la zona restringida para usuarios.

Un usuario experto, conocedor de cómo trabaja una inyección de código SQL, le bastaría con cambiar el valor de las variables $usua y $clav por lo siguiente:

EJEMPLO

```
$usua=' or '1'='1;
$clav=' or '1'='1;
```

Una vez sustituidas las variables en la consulta que va a ser ejecutada, quedaría de la siguiente forma:

EJEMPLO

```
$sql="SELECT * FROM usuarios WHERE user = '' or '1'='1' AND
password=' ' or '1'='1';
```

Como podemos observar, vemos que una de las condiciones se convierte en una pregunta "uno es igual que uno", por lo tanto esta, consulta devolverá valor trae, y por tanto, un acceso no deseado a la sección restringida de nuestra web.

Actualmente, existen hasta tres técnicas distintas de evitar un ataque por inyección de código SQL:

- Prepared Statements. Los prepared statements son sentencias pre-compiladas, en las cuales tendremos que indicar los parámetros que van a introducir los usuarios. Así podremos especificarle a la base de datos, por un lado, el código que vamos a ejecutar, y por otra parte, le diferenciaremos las variables que vamos a utilizar. De esta forma, el motor de bases de datos podrá distinguir los datos de entrada y evitar la inyección de instrucciones SQL.

  Otra de la ventaja que aportar es la optimización de tiempo de ejecución si la misma sentencia vamos a utilizarla en más de una ocasión, ya que el motor de la base de datos, por cada instrucción SQL, tiene que analizar, compilar y optimizar la forma en la que se ejecutará. Por lo tanto, si la preparamos una sola vez y la ejecutamos en varias ocasiones, con los mismos o diferentes argumentos, el tiempo de ejecución será inferior.

  Podemos encontrar más información en el enlace oficial del manual de PHP: http://www.php.net/manual/es/pdo.prepared-statements.php

- Stored Procedures. Los *stored procedures* son más conocidos que los *prepared statements* entre los desarrolladores gracias a su utilización extensiva en la programación con bases de datos. Se escriben procedimientos en el lenguaje del DBMS (SQL) y desde la zona donde hayamos escrito el código se invocarán estos procedimientos con las variables ingresadas por el usuario como los parámetros. De esta manera, el DBMS podrá distinguir correctamente las variables del código, evitando de nuevo la inyección.

  Podemos encontrar más información en el enlace oficial del manual de PHP: http://www.php.net/manual/es/pdo.prepared-statements.php.

- Escapar todo dato ingresado por el usuario. Esta es la manera más utilizada por los programadores de prevenir la inyección de SQL, pero también es la menos recomendada. La idea es que cada vez que el usuario introduzca los datos que utilizaremos en una sentencia, escapemos los caracteres especiales (como comillas simples o dobles, barras invertidas "\", o caracteres de comentario "--" o "#", etc) para que el dato sea un solo string y el motor de BD no lo confunda con código a ejecutar.

Un ejemplo de función que permite escapar los caracteres especiales, cuando utilizamos php y mysql, es la función mysql_real_escape_string().

No obstante, esta técnica es frágil y en principio será mejor utilizarla solo cuando ya contamos con un código inseguro y reescribirlo utilizando *prepared statements* requeriría un costo inaceptable. Si empezamos a desarrollar nuestro código desde el principio, siempre será mejor utilizar *prepared statements* o *stored procedures*.

Por ejemplo, en nuestro formulario de login anterior, tendríamos que hacer lo siguiente:

EJEMPLO

```
$usua=mysql_real_escape_string($_POST['usuario']);
$clav=mysql_real_escape_string($_POST['clave']);
```

Podemos encontrar más información en el enlace oficial del manual de PHP: http://php.net/manual/es/function.mysql-real-escape-string.php

4.7. Envío de mails

PHP incorpora una función sencilla de utilizar para poder enviar correos electrónicos desde nuestras aplicaciones. Se trata de la función **mail()**.

Su sintaxis es la siguiente:

EJEMPLO

```
bool mail ( string $to , string $subject , string $message [,
string $additional_headers [, string $additional_parameters
]] )
```

Los parámetros que hay que pasar a la función son los siguientes:

- **$to**. Receptor o receptores del correo.
- **$subject**. Título del mail a ser enviado.
- **$message**. Mensaje a enviar.
- **$additional_headers**. String a ser insertado al final de la cabecera del email. Se usa típicamente para añadir cabeceras extra (From, Cc y Bcc).

Una aplicación clásica de esta función suele ser en los formularios de contacto, donde ofrecemos la posibilidad a los usuarios de que se comuniquen con los responsables de la página web.

Veamos un sencillo ejemplo de su utilización.

EJEMPLO

```
<?php
$nombre_origen = “”; //Nombre de remitente
$email_origen = “”; // Email desde donde se envía el correo
$email_copia = “”; // Dirección de copia del correo
$email_ocultos = “”; // Direcciones para correo oculto
$email_destino = “”; // Correo destino
$nombre_destinatario=””; //a quien va dirigido
$asunto= “”; // asunto del correo
$mensaje = “”;
$formato= “html”; //opción para enviarlo como html si lo de-
jas vacio
                          // lo manda como texto plano.
//*********************************************************
*//

//a continuación las cabeceras de un correo como para quien
va, de quien, etc.
$headers = “To: $nombre_destinatario <$email_destino> \r\n”;
$headers .= “From: $nombre_origen <$email_origen> \r\n”;
$headers .= “Return-Path: <$email_origen> \r\n”;
$headers .= “Reply-To: $email_origen \r\n”;
$headers .= “Cc: $email_copia \r\n”;
$headers .= “Bcc: $email_ocultos \r\n”;
$headers .= “MIME-Version: 1.0 \r\n”;
//*********************************************************
*//
//si el formato no es html entonces que lo envié como texto
plano
if($formato == “html”)
{
$headers  .=  “Content-Type:  text/html;  charset=iso-8859-1
\r\n”;
}
else
{
$headers  .=  “Content-Type:  text/plain;  charset=iso-8859-1
\r\n”;
  }
// si todo está bien en correo se envía
if (@mail($email_destino, $asunto, $mensaje, $headers))
{
echo “Su correo ha sido correctamente enviado”;
}
else
{
echo “Error en el envío del correo”;
}
?>
```

Para el correcto funcionamiento de esta función, tendremos que tener configurado un servidor de envío de correo electrónico SMTP en el fichero de configuración PHP.INI.

Para ello, tendremos que localizar en este fichero, las siguientes etiquetas y modificarlas con nuestra configuración:

EJEMPLO

```
[mail function]
; For Win32 only.
SMTP = localhost //aquí el servidor SMTP
smtp_port = 25  //aquí el puerto de envío del servidor SMTP
; For Win32 only.
sendmail_from = me@localhost.com //aquí la cuenta desde donde
enviamos el correo
```

4.8. Creación de ficheros PDF mediante la librería FPDF

Mediante el uso de la librería FPDF (Free PDF), vamos a poder crear de forma cómoda y sencilla documentos PDF, con grandes opciones de personalización como la inclusión de cabeceras, pies de página, colores, imágenes, saltos del línea, fuentes, etc.

Antes de explicar un ejemplo sencillo para su uso, es conveniente conocer la forma en la que tendremos que instalarla.

En primer lugar, descargaremos la versión más reciente desde su página oficial (http://www.fpdf.org/es/download.php), en la cual nos encontraremos un archivo comprimido zip con tutoriales y explicaciones de cada función en particular (también disponibles en la página oficial). El archivo más importante del fichero comprimido, es el llamado **fpdf.php,** aunque también necesitaremos tener en nuestro directorio local, el archivo fpdf.css y la carpeta de font. Por ejemplo, la podemos descomprimir y guardar los ficheros necesarios en la ruta c:\xampp\htdocs. Este dato es importante, ya que conocer la ruta será indispensable para incluirla en la instrucción **require**.

Para indicar que vamos a utilizar esta librería, tendremos que hacer referencia a la localización del fichero mediante la función **require**:

EJEMPLO

```
require("fpdf.php");
```

A continuación, para indicar la creación del documento PDF, tendremos que escribirlo de la siguiente forma:

EJEMPLO

```
$pdf=new FPDF();
```

Veamos, a continuación, un ejemplo sencillo de su uso:

EJEMPLO

```
<?php
require('fpdf.php');
$pdf = new FPDF();
$pdf->AddPage();
$pdf->SetFont('Arial','B',16);
$pdf->Cell(40,10,'¡Hola, Mundo!');
$pdf->Output();
?>
```

En primer lugar, hemos creado una nueva clase, que utilizará el método constructor **FPDF()** para asignar unos valores por defecto: tamaños de página A4 alargado y con una unidad de medida que será el milímetro.

A continuación, hacemos uso del método **AddPage().** Este método será llamado al comienzo de cada página del documento PDF que queramos crear. Si quisiéramos un documento de 5 páginas, esto conllevaría el llamamiento a la función AddPage() hasta en 5 ocasiones.

Después utilizamos el método **SetFont(),** para a través de sus parámetros escoger el tipo de fuente, en este caso, en Arial, negrita y tamaño 16.

A continuación, creamos una celda con **Cell().** Una celda es una superficie rectangular, con borde si se quiere, que contiene texto. Se imprime en la posición actual. Especificamos sus dimensiones, el texto (centrado o alineado), si queremos dibujar o no los bordes, y dónde se ubicará la posición actual después de imprimir la celda (a la derecha, debajo o al principio de la siguiente línea).

Finalmente, el documento se cierra y se envía al navegador con **Output().** También podríamos haberlo guardado en un fichero pasando como parámetro el nombre del archivo.

RESUMEN

- Se ha explicado la forma de crear clases y objetos, así como de utilizar sus atributos y métodos para conseguir una programación más optimizada.
- Se han dado a conocer la forma en la que se pueden heredar las clases y el concepto de polimorfismo y su forma de implementación.
- Se han explicado y desarrollado formularios para poder autentificar usuarios, subir ficheros al servidor, validar campos introducidos por usuarios y cómo enviar variables mediante los métodos GET y POST.
- Se ha explicado el concepto de sesión y de cookies, sus diferentes ventajas e inconvenientes, así como desarrollado aplicaciones para comprobar su funcionamiento.
- Se han explicado las operaciones y funciones básicas para realizar operaciones con ficheros: apertura, lectura, escritura y cierre de archivos.
- Se han introducido aspectos importantes relativos a implementar diferentes medidas de seguridad en aplicaciones PHP.
- Se ha explicado la vulnerabilidad de la inyección de código SQL y cómo prevenirnos de ella.
- Se han explicado diferentes funciones para encriptación de textos: crypt(), md5() y sha1().
- Se han explicado las diferentes maneras que tenemos de controlar los errores de compilación o tiempo de ejecución en PHP y cómo podemos modificar la configuración según nos interese para depurar nuestro código.
- Se ha dado a conocer la función mail() para poder enviar correos electrónicos y se ha desarrollado una aplicación para pruebas de envíos de correo electrónico.