Técnico en creación de portales web con software libre

Módulo 3: Creación de páginas web dinámicas mediante PHP

Juan Ángel Lorenzo del Castillo
juanangel@dec.usc.es

Departamento de Electrónica y Computación
Universidad de Santiago de Compostela

Parte I

Conceptos Básicos

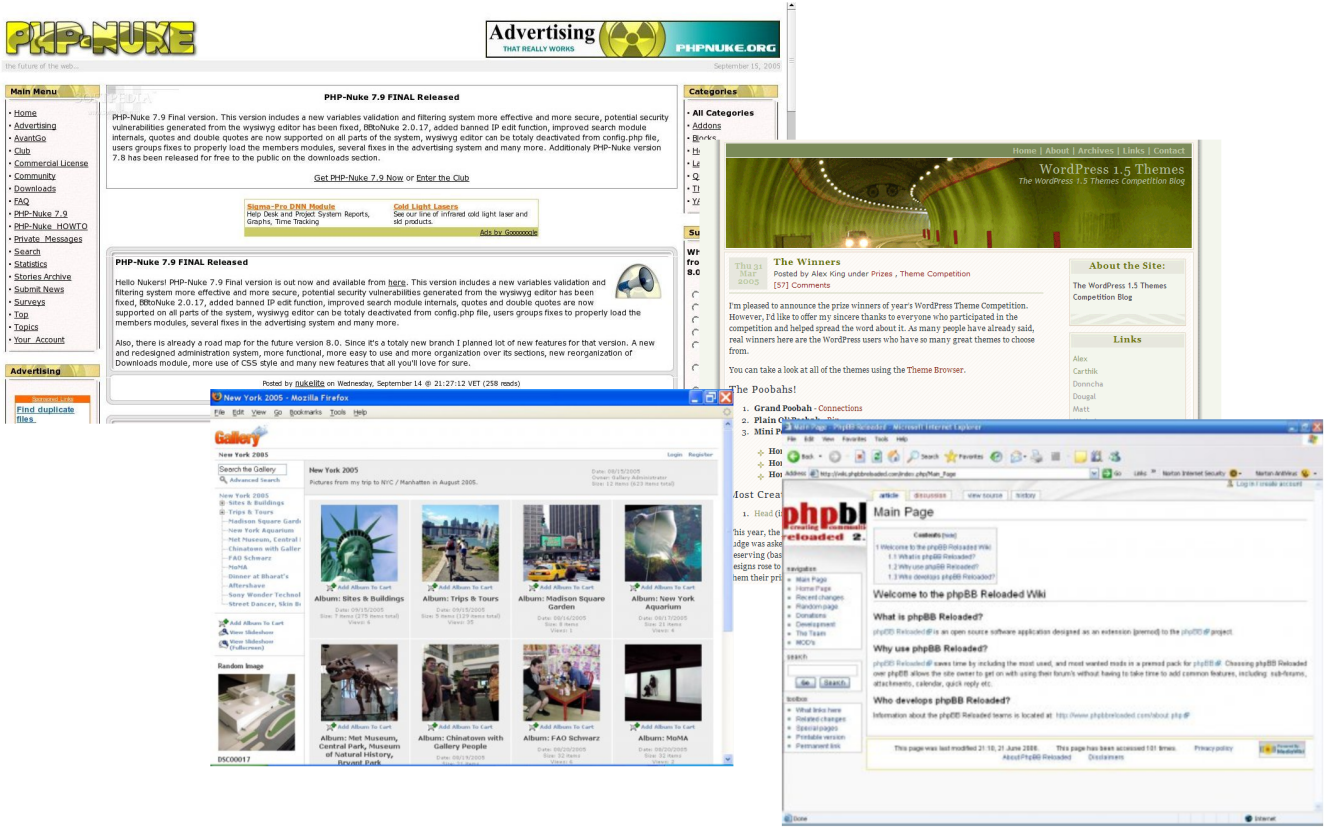
Contenidos

PHP (*PHP Hipertext Preprocessor*)

- Definición (*Personal Home Page*, *PHP Hipertext Preprocessor*): Lenguaje de secuencia de comandos de servidor diseñado para web.
- Versión 5.
- Creador: Rasmus Lerdorf
- Lenguaje interpretado en el servidor web, alto rendimiento.
- Licencia GPL. Adaptación a todos los SGBD (MySQL y otros).

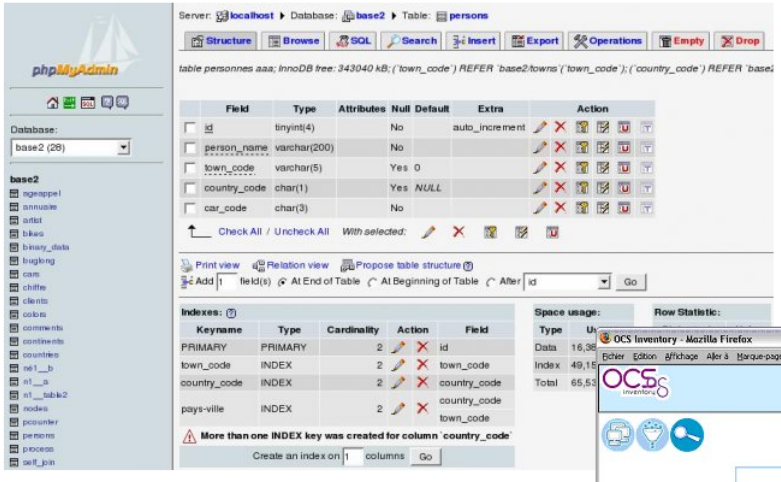
Ejemplos de aplicaciones creadas con PHP

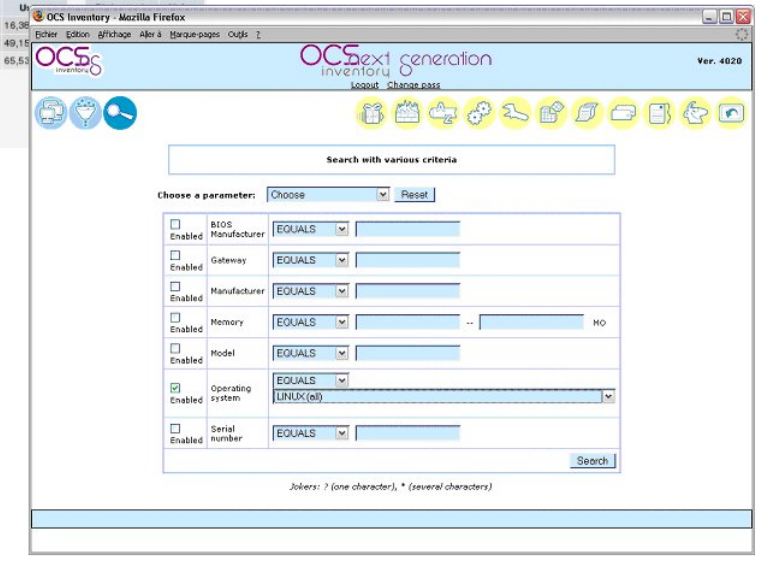
- Weblogs y Foros de discusión



Ejemplos de aplicaciones creadas con PHP

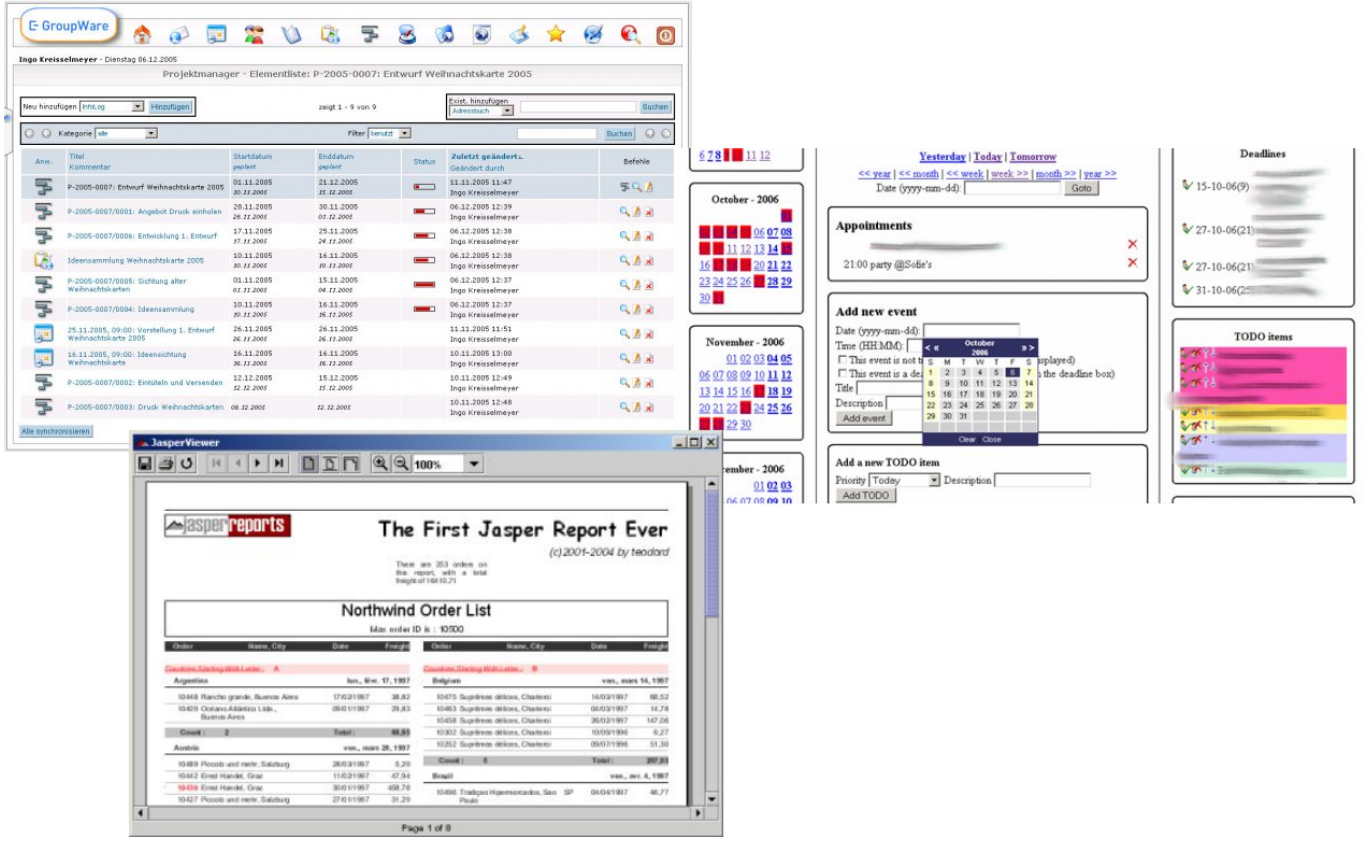
- Herramientas de administración





Ejemplos de aplicaciones creadas con PHP

- Herramientas de *groupware*



Ejemplos de aplicaciones creadas con PHP

- ¡Incluso juegos!

Ventajas de PHP

Ventajas:

- Potente
- Fácil de aprender.
- Libre distribución
- Bibliotecas incorporadas (generación pdf, emails, gifs, etc.).
- Permite el acceso a bases de datos y otras funcionalidades orientadas a la red
- Dispone de abundante soporte en la Web (`www.php.net`)
- Orientación a objetos completa desde la versión 5.

Qué vamos a aprender

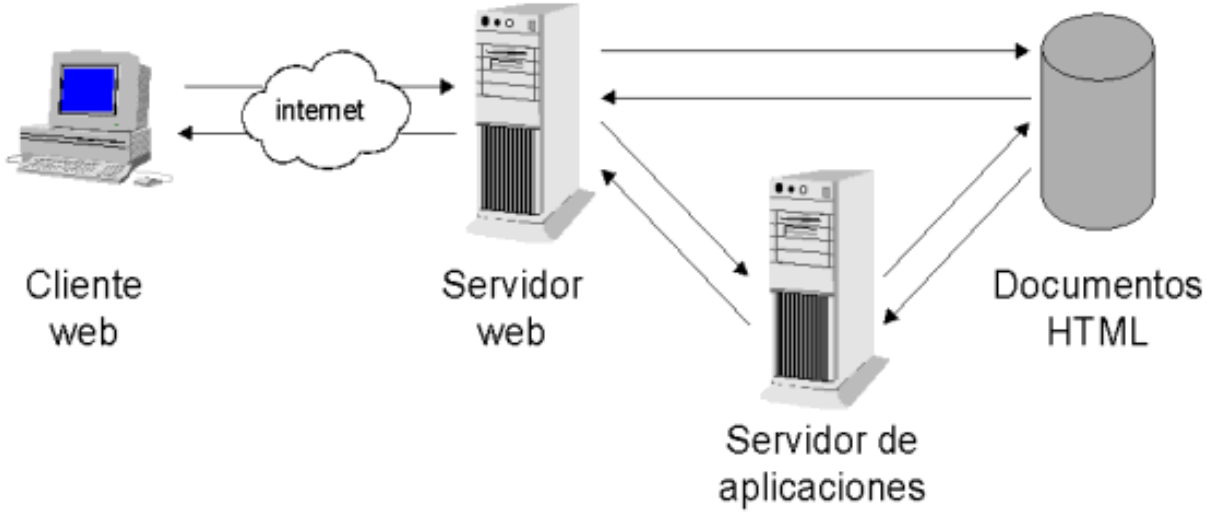
- PHP básico: variables, expresiones, formularios, estructuras de control.
- PHP avanzado: manipulación de ficheros, arrays, funciones.
- Orientación a Objetos en PHP.
- Control de sesiones.
- Manejo de fechas.
- Diseño de plugins para weblogs.
- Proyecto Fin de Módulo.

Plataforma y herramientas de programación

- SO: Linux
- Servidor web e intérprete de PHP: Apache 2.0
- Editor:
  - **Avanzado (frikis):** vi, emacs.
  - **Medio (menos frikis):** Quanta, Bluefish, Frontpage, Dreamweaver.
  - **Básico (mayor facilidad de uso):** kwrite, kate, wordpad, notepad.

Cómo funciona

- PHP es un lenguaje de *script*.
- Dentro de una página web se incrustan *scripts* (porciones de código) en PHP.
- Interpretación del código en tiempo real y generación de una página html cada vez que se visita la página.
- El usuario no ve el código PHP, sino los resultados en html que genera.



Contenidos

Procesado de un formulario html:

ordenpedido.html

```
<html>
<body>
 <h1>Formulario de Pedido</h1>
 <p>
 <form action="procesapedido.php"
         method=post>
  <table border="0" width="2">
  <tr bgcolor="#CCCCCC" >
    <td width="150">Artículo</td>
    <td width="15">Cantidad</td>
  </tr>

  <tr>
    <td>Neumáticos</td>
    <td align="center"><input type="text"
        name="neumatico" size="3"
        maxlength="3" ></td>
  </tr>

  <tr>
    <td>Aceite</td>
    <td align="center"><input type="text"
        name="aceite" size="3"
        maxlength="3"></td>
  </tr>

  <tr>
    <td>Bujías</td>
    <td align="center"><input type="text"
        name="bujias" size="3"
        maxlength="3"></td>
  </tr>

  <tr>
    <td colspan="2" align="center"><input
        type="submit" value="Enviar"
        name="EnviarOrden"></td>
  </tr>

  </table>
</body>
</html>
```

Script php:

procesapedido.php

```
<html>
 <head><title> Resultado del pedido </title></head>

<body>
 <h1>Garaje Paco</h1>
 <h2>Resultado del envío del pedido:</h2>

 <?php
  echo '<p>Orden procesada. Estilo 1</p>';
 ?>

</body>
</html>
```

¿Qué ha pasado aquí?

A tener en cuenta

- POST vs. GET

```
<form action="procesapedido.php"method=post>
```

- Estilos de etiqueta

```
<?php
  echo '<p>Orden procesada. Estilo XML </p>';
?>

<? echo '<p>Orden procesada. Estilo corto </p>'; ?>

<script language='php'>echo '<p>Orden procesada. Estilo script </p>';
</script>

<% echo '<p>Orden procesada. Estilo ASP </p>'; %>
```

- Formato de instrucción: terminados en ";"
- Comentarios como en C: /* */ , // y #

¿Qué ha pasado aquí?

A tener en cuenta

- POST vs. GET

```
<form action="procesapedido.php"method=post>
```

- Estilos de etiqueta

```
<?php
  echo '<p>Orden procesada. Estilo XML </p>';
?>

<? echo '<p>Orden procesada. Estilo corto </p>'; ?>

<script language='php'>echo '<p>Orden procesada. Estilo script </p>';
</script>

<% echo '<p>Orden procesada. Estilo ASP </p>'; %>
```

- Formato de instrucción: terminados en “;”
- Comentarios como en C: /* */ , // y #

¿Qué ha pasado aquí?

A tener en cuenta

- POST vs. GET

```
<form action="procesapedido.php"method=post>
```

- Estilos de etiqueta

```
<?php
  echo '<p>Orden procesada. Estilo XML </p>';
?>

<? echo '<p>Orden procesada. Estilo corto </p>'; ?>

<script language='php'>echo '<p>Orden procesada. Estilo script </p>';
</script>

<% echo '<p>Orden procesada. Estilo ASP </p>'; %>
```

- Formato de instrucción: terminados en “;”
- Comentarios como en C: /* */ , // y #

¿Qué ha pasado aquí?

A tener en cuenta

- POST vs. GET

```
<form action="procesapedido.php"method=post>
```

- Estilos de etiqueta

```
<?php
  echo '<p>Orden procesada. Estilo XML </p>';
?>

<? echo '<p>Orden procesada. Estilo corto </p>'; ?>

<script language='php'>echo '<p>Orden procesada. Estilo script </p>';
</script>

<% echo '<p>Orden procesada. Estilo ASP </p>'; %>
```

- Formato de instrucción: terminados en “;”
- Comentarios como en C: /* */ , // y #

¿Qué ha pasado aquí?

A tener en cuenta

- POST vs. GET

```
<form action="procesapedido.php"method=post>
```

- Estilos de etiqueta

```
<?php
  echo '<p>Orden procesada. Estilo XML </p>';
?>

<? echo '<p>Orden procesada. Estilo corto </p>'; ?>

<script language='php'>echo '<p>Orden procesada. Estilo script </p>';
</script>

<% echo '<p>Orden procesada. Estilo ASP </p>'; %>
```

- Formato de instrucción: terminados en “;”
- Comentarios como en C: /* */ , // y #

¿Qué ha pasado aquí?

A tener en cuenta

- POST vs. GET

```
<form action="procesapedido.php"method=post>
```

- Estilos de etiqueta

```
<?php
  echo '<p>Orden procesada. Estilo XML </p>';
?>

<? echo '<p>Orden procesada. Estilo corto </p>'; ?>

<script language='php'>echo '<p>Orden procesada. Estilo script </p>';
</script>

<% echo '<p>Orden procesada. Estilo ASP </p>'; %>
```

- Formato de instrucción: terminados en “;”
- Comentarios como en C: /* */ , // y #

¿Qué ha pasado aquí?

A tener en cuenta

- POST vs. GET

```
<form action="procesapedido.php"method=post>
```

- Estilos de etiqueta

```
<?php
  echo '<p>Orden procesada. Estilo XML </p>';
?>

<? echo '<p>Orden procesada. Estilo corto </p>'; ?>

<script language='php'>echo '<p>Orden procesada. Estilo script </p>';
</script>

<% echo '<p>Orden procesada. Estilo ASP </p>'; %>
```

- Formato de instrucción: terminados en “;”
- Comentarios como en C: /* */ , // y #

Un *script* un poco más útil:

- El ejemplo anterior podría haberse implementado íntegramente en html
- Añadimos contenido dinámico:

procesapedido2.php

```
<html>
  <head><title> Resultado del pedido </title>
  </head>

<body>
  <h1>Garaje Paco</h1>
  <h2>Resultado del envío del pedido:</h2>

  <?php
   echo '<p>Orden procesada a las ';
   echo date('H:i, jS F');
   echo '</p>';
  ?>

</body>
</html>
```

Invocación a la función `date`

- H: hora en formato 24 horas.
- i: minutos precedidos de un 0.
- j: día del mes sin utilizar un cero inicial.
- S: sufijo ordinal (*th*).
- F: nombre del mes.

Un *script* un poco más útil:

- El ejemplo anterior podría haberse implementado íntegramente en html
- Añadimos contenido dinámico:

procesapedido2.php

```
<html>
  <head><title> Resultado del pedido </title>
  </head>

<body>
  <h1>Garaje Paco</h1>
  <h2>Resultado del envío del pedido:</h2>

  <?php
   echo '<p>Orden procesada a las ';
   echo date('H:i, jS F');
   echo '</p>';
  ?>

</body>
</html>
```

Invocación a la función `date`

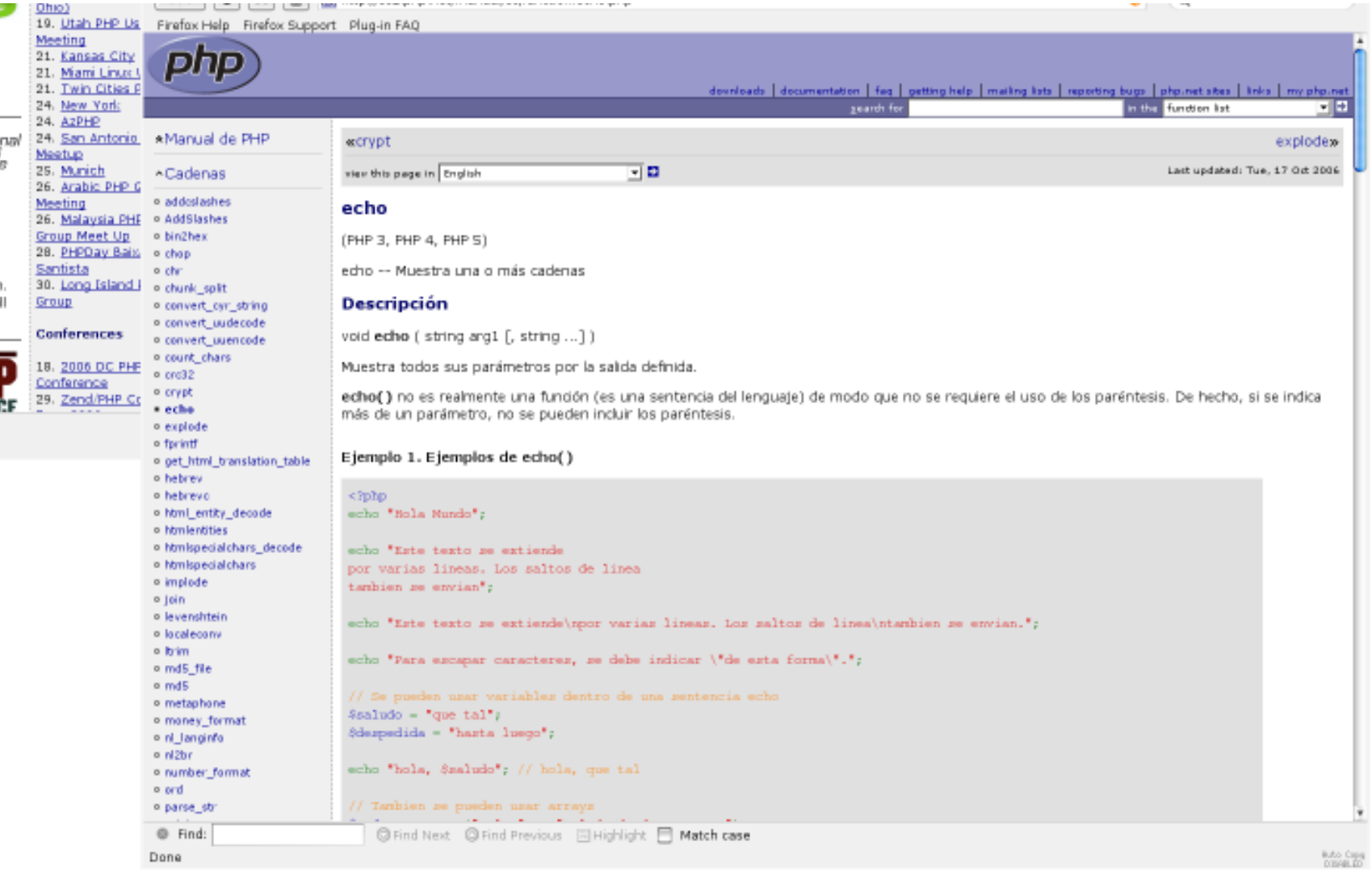
- H: hora en formato 24 horas.
- i: minutos precedidos de un 0.
- j: día del mes sin utilizar un cero inicial.
- S: sufijo ordinal (*th*).
- F: nombre del mes.

¿Dónde encontrar los parámetros de una función determinada?

- ¡En la API!

www.php.net

¿Dónde encontrar los parámetros de una función determinada?



Contenidos

Variables

Variable: Símbolo que almacena un valor

- Identificador de una variable precedido por "$". Ej: `$contador`
- Cualquier longitud. Pueden incluir letras, números y guiones bajos.
- No pueden comenzar por un número.
- Sensibles a las mayúsculas: `$contador` ≠ `$ConTador`.
- Inicialización:
  - `$texto = 'hola';`
  - `$contador = 1;`
  - `$tmp = $contador;`
  - `$punto_flotante = 1.05;`

Variables

Tipos de datos:

- **Integer** (Entero): 32, -5, 0
- **Double** (Flotante): 5.67, -1.93
- **String** (Cadena): Se encierran entre comillas dobles o simples
  - Simples (’): admiten los caracteres de escape \’ (comilla simple) y \\ (barra). Las variables **no** se expanden.
  - Dobles ("): admiten más caracteres de escape, como *n*, *r*, *t*, \\, \$ y \". Las variables **sí** se expanden (**interpolación**).

```
$a = 3;
print ’a vale $a’; //Muestra: a vale $a
print "a vale $a"; //Muestra: a vale 3
```

- **Booleano**: *true* o *false*. El 0 tiene valor *false*.
- **Array** (Matriz): Para almacenar conjuntos de datos del mismo tipo.
- **Object** (Objeto): Para almacenar instancias de clases.
- **Resource** (Recurso): Recursos externos (como conexiones a una BBDD).
- **NULL** (Nulo): Variables a las que no se les ha asignado un valor.

Variables

Control de tipos:

- En PHP, el tipado es débil: el tipo de variable viene determinado por el valor que se le asigne.

```
$temporal = 5.4;     //temporal es double y vale 5.4
$temporal = "hola"; //ahora, temporal es string y vale "hola"
```

- `gettype()` devuelve el tipo de una variable.
- Funciones `is_type` para comprobar el tipo: `is_array()`,`is_bool()`, `is_float()`,`is_integer()`,`is_null()`,`is_numeric()`,`is_object()`, `is_resource()`,`is_scalar()`,`is_string()`

Conversión de tipos:

```
$temporal = 5;
$doble = (double)$temporal;
```

Variables

Variables de formulario

- Recogida de variables del formulario `ordenpedido.html`:
  - `$neumaticos = $_POST['neumatico'];`
  - `$aceite = $_POST['aceite'];`
  - `$bujias = $_POST['bujias'];`
- Otras posibilidades:
  - `$neumaticos = $HTTP_POST_VARS['neumatico'];`
  - `$neumaticos = $_REQUEST['neumatico'];`

Variables

Procesamos el pedido con un resultado más útil...

procesapedido3.php

```
<html>
<head><title> Resultado del pedido </title></head>

<body>
<h1>Garaje Paco</h1>
<h2>Resultado del envío del pedido:</h2>

<?php

  $neumaticos = $HTTP_POST_VARS['neumatico'];
  $aceite = $_POST['aceite'];
  $bujias = $_POST['bujias'];

  echo $neumaticos.' neumaticos<br>';
  echo $aceite.' garrafas de aceite<br>';
  echo "$bujias bujias<br>";

?>

</body>
</html>
```

Variables

Concatenación de cadenas

- Las cadenas se concatenan con el punto (.):

```
echo $neumaticos.' neumaticos<br>';
```

- Recordemos: ′ para literales, ″ para interpolación (sustitución de una variable en una cadena):

```
echo "$bujias bujias<br>";
```

Variables

Ámbito de las variables

Ámbito: Lugares dentro de las secuencias de comandos en los que resulta visible una variable dada.

- **Superglobales** o **predefinidas**: `$GLOBALS`, `$_SERVER`, `$_GET`, `$_POST`, `$_COOKIES`, `$_FILES`, `$_ENV`, `$_REQUEST`, `$_SESSION`. Siempre visibles.
- **Constantes:** Visibles globalmente, una vez declaradas.
- **Globales:** Visibles en toda la secuencia de comandos (fichero) pero no dentro de las funciones.
- **Locales:** Definidas dentro de una función.

Constantes

Constante: Símbolo que almacena un valor que no puede modificarse durante el resto de la ejecución del programa

- Se utiliza la función `define`:

```
define('CUATRO', 4);
```

- Se definen, por convención, en mayúsculas.
- Se invocan sin el símbolo "$":

```
echo CUATRO;
```

- Pueden consultarse las constantes propias y predefinidas por PHP, así como las variables, con `phpinfo();`
- Sólo pueden almacenar datos de tipo Boolean, integer, double o string (**valores escalares**).

Contenidos

Operadores

Aritméticos

`+, -, *, / y % (módulo)`

operador de asignación (=) e incrementos

Incrementos:

```
$a = 5;
$a += 8;  // $a vale 13
$a ++; //$a vale 14;

$a="Garaje";
$a.=" Paco"; //$a vale "Garaje Paco"
```

Pre y postincremento:

```
$a = 4;
echo ++$a;
   // preincremento:
   // incrementa $a y muestra 5

$a = 4;
echo $a++;
   // postincremento:
   // muestra 4 e incrementa $a
```

Operadores

Comparación

```
==, !=, <, >, <=, >=, === (idénticos), etc.
```

Lógicos

```
AND (&&), OR (||), NOT (!), XOR
```

Supresión de control de error

Antepuesto a una expresión, evita cualquier mensaje de error que pueda ser generado por la expresión.

```
@
```

Ejemplo: `$a = @(57/0) //Evita un error de división por cero.`

Operadores

Precedencia de operadores

Mayor a menor:

```
++, --
*, /, %
+,-
<, <=, >, >=
==, !=
&&
||
```

Contenidos

Estructuras de Control

Instrucciones `if`

```
if($bujias == 0)
     echo 'No solicitó ninguna bujía en la página anterior<br>';
```

Bloques de código entre llaves:

```
if(expresión1)
 {
      sentencia 1;
      sentencia 2;
 }
else if (expresión2)
      sentencia 2;
...
else if (expresión n)
      sentencia n;
else
      sentencia n+1;
```

Ejercicio

Modificar `procesapedido3.php` para que compruebe que los datos introducidos son válidos

Ejercicio

Modificar `procesapedido3.php` para que compruebe que los datos introducidos son válidos

```
<html>
<head><title> Resultado del pedido </title></head>

<body>
 <h1>Garaje Paco</h1>
 <h2>Resultado del envío del pedido:</h2>
 <?php

  $neumaticos = $HTTP_POST_VARS['neumatico'];
  $aceite = $_POST['aceite'];
  $bujias = $_POST['bujias'];

  if(!is_numeric($bujias) || !is_numeric($neumaticos) || !is_numeric($aceite))
     echo 'Al menos, uno de los datos introducidos no es una cantidad numérica<br>';
  else if($bujias == 0 || $neumaticos == 0 || $aceite == 0)
     echo 'No se ha solicitado, al menos, uno de los productos<br>';
  else{
     echo $neumaticos.' neumaticos<br>';
     echo $aceite.' garrafas de aceite<br>';
     echo "$bujias bujias<br>";
  }
 ?>

</body>
</html>
```

Estructuras de Control

Instrucciones `switch`

```
switch (expresión)
{
   case valor 1:
        sentencia 1;
        break;
   case valor 2:
        sentencia 2;
        break;
   ...
   case valor n:
        sentencia n;
        break;
   default
        sentencia n+1;
}
```

- `expresión` puede ser integer, float o string.

Estructuras de Control

Bucles `while`

```
while (condición)
   expresion;
```

Bucles `for`

```
for (expresión1; condición ; expresión2)
   expresión3;
```

Bucles `do...while`

```
do
   expresión;
while (condición);
```

Estructuras de Control

Salir de una estructura de control: `exit`

```
if(!is_numeric($numero))
{
  echo 'El valor introducido no es numérico<br>';
  exit;
}
```

Sintaxis alternativa para bucles de control

En lugar de la sentencia anterior:

```
if(!is_numeric($numero)):
  echo 'El valor introducido no es numérico<br>';
  exit;
endif;
```

- Se puede utilizar con `if`, `switch`, `while` y `for`, anteponiendo el prefijo `end`.

Ejercicio

Programa que muestre una tabla de multiplicar

Formulario `tablamultiplicar.html`:

```
<html>
<body>
<h1>Formulario de Pedido</h1>

<form action="tablamultiplicar.php" method=post>
<table>

<tr bgcolor="#CCCCCC">
    <td width="100">Tabla de multiplicar:</td>
</tr>
<br>
<tr>
    <td width="50">Número del que desea la tabla de multiplicar:</td>
    <td ><input type="text" name="numero" size="3" maxlength="3"></td>
</tr>

<tr>
    <td colspan="2" align="center"><input type="submit"
        value="Enviar" name="EnviarOrden"></td>
</tr>

</table>

</body>
</html>
```

Ejercicio

Programa que muestre una tabla de multiplicar

Programa `tablamultiplicar.php`

```
<html>
<head><title> Tabla de multiplicar </title></head>

<body>
<?php

  $numero = $_POST['numero'];

  if(!is_numeric($numero))
      echo 'No se puede calcular la tabla. El valor introducido no es numérico<br>';
  else
  {
      echo "<h2>Tabla de multiplicar del $numero</h2>";

      for($i=0;$i<=10;$i++)
      {
        echo "  $numero * $i = ".$numero*$i."<br>";
      }
  }
?>

</body>
</html>
```

Parte II

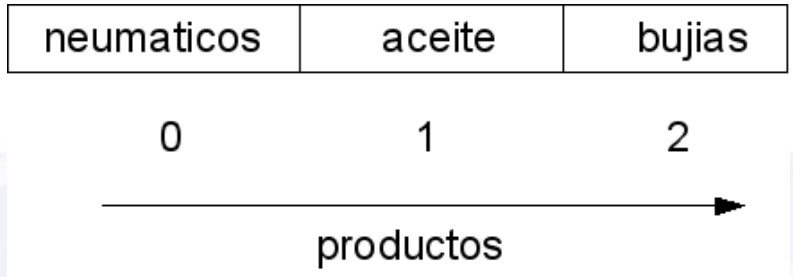
Matrices

Contenidos

¿Qué es una matriz?

- Las variables vistas hasta ahora sólo pueden almacenar un valor.
- **Matriz:** variable capaz de almacenar un conjunto o secuencia de valores.
- Dos tipos:
  - Matrices indexadas numéricamente.
  - Matrices asociativas.
- Las matrices pueden almacenar otras matrices. Se denominan entonces *matrices multidimensionales*.

Siguiendo con nuestro ejemplo del garaje...



- Almacenamos las variables del pedido en una matriz llamada `productos`.
- Se puede utilizar todo el conjunto de información como una unidad.
- Cada valor almacenado se denomina *elemento* de la matriz.
- Cada elemento lleva asociado un *índice* (también llamado *clave*).
  - Si el índice es numérico: *matriz indexada numéricamente*.
  - Si el índice no es numérico: *matriz no indexada numéricamente* o *matriz asociativa*.

Contenidos

Inicialización de matrices

Creamos la matriz `productos`

```
$productos = array('neumaticos','garrafas de aceite','bujias');
```

- Se utiliza la palabra reservada `array`.
- Siempre contienen elementos del mismo tipo.
- En este caso, contiene tres cadenas.

Inicialización automática de secuencias de números

```
$numeros = range(1,10);   //Secuencia del 1 al 10
$impares = range(1,10,2); //Secuencia del 1 al 10 de dos en dos
$letras = range('a','e',2); //Secuencia de 'a' a 'e' de dos en dos
```

Inicialización de matrices

Creación automática de una matriz

```
$productos[0]='neumaticos';
$productos[1]='garrafas de aceite';
$productos[2]='bujías';
```

- No es necesario utilizar la palabra `array`.
- Cada línea añade un elemento nuevo a la matriz.
- El tamaño de la matriz cambia automáticamente al añadir elementos.

Acceso a los elementos de una matriz

Se accede a través del índice

```
echo $productos[0]; //Devuelve "neumaticos"
```

Cambio del contenido de un elemento

```
$productos[1] = 'carburador'; //cambio de "aceite" por "carburador"
```

Adición de un elemento

```
$productos[3] = 'junta de la trócola';
```

Acceso a los elementos de una matriz

Acceso a través de un bucle `for`

```
for($i=0;$i<3;$i++)
{
  echo 'Producto '.$i.' = '.$productos[$i].'<br>';
}
```

Acceso a través de un bucle `foreach`

```
foreach($productos as $elemento)
{
  echo 'Producto = '.$elemento.'<br>';
}
```

- En cada iteración, la variable `$elemento` contiene un elemento de la matriz `$productos`.
- Se utiliza más habitualmente con matrices asociativas...
- ... pero nos permite aseguramos de recorrer todos los elementos.

Contenidos

Inicialización de matrices asociativas

Creamos una matriz `precios`

```
$precios = array('neumaticos'=>100, 'aceite'=>10, 'bujias'=>4);
```

- En este caso, los índices no son números, sino los nombres de los productos.
- Los índices se denominan *claves* (los nombres de los productos, en el ejemplo).
- Los elementos referenciados por las claves se denominan *valores* (los precios de los productos, en el ejemplo).
- Notación: *clave* $=>$ *valor*.

Inicialización de matrices asociativas

Creación automática de una matriz asociativa

```
$precios = array('neumaticos'=>100);
$precios['aceite'] = 10;
$precios['bujias'] = 4;
```

- O bien:

```
$precios['neumaticos'] = 100;
$precios['aceite'] = 10;
$precios['bujias'] = 4;
```

Acceso a los elementos de una matriz

Acceso a través de un bucle `foreach`

```
foreach($precios as $clave=>$valor)
{
  echo $clave.' = '.$valor.'<br>';
}
```

- En cada iteración, la variable `$clave` contiene el índice de un elemento de la matriz `$precios`.
- En cada iteración, la variable `$valor` contiene el elemento de la matriz `$precios` correspondiente a la clave `$clave`.

Acceso a los elementos de una matriz

Acceso mediante `each`

```
while($elemento = each($precios))
{
  echo $elemento['key'];
  echo ' - ';
  echo $elemento['value'];
  echo '<br>';
}
```

- `each` almacena en `$elemento`, para cada iteración, una matriz con pares clave-valor.
- Se accede a los elementos medianta las palabras reservadas **`key`** para la clave o índice y **`value`** para el valor del elemento.
- También es posible acceder a los elementos mediante los índices 0 (para la clave) y 1 (para el valor).
- El bucle se detiene tras llegar al último elemento de la matriz.

Acceso a los elementos de una matriz

Acceso mediante `list`

```
while(list($producto,$precio) = each($precios))
{
  echo "$producto - $precio<br>";
}
```

- En este caso, `list` convierte los elementos 0 y 1 de la matriz devuelta por `each` en dos variables llamadas `producto` y `precio`.
- `each` va recorriendo cada elemento de `$precios`. Si quisiéramos volver a utilizar `$precios` en otro bucle, debemos hacer que se vuelva a apuntar al primer elemento. Para ello, se utiliza la orden `reset()`.

```
reset($precios);
```

Contenidos

Operadores de matriz

Permiten realizar operaciones sobre las matrices

Operador	Nombre	Ejemplo	Resultado
+	Unión	$a + $b	Unión de $a y $b. La matriz `b` se adjunta a `a`.
==	Igualdad	$a == $b	*True* si $a y $b contienen los mismos elementos.
===	Identidad	$a === $b	*True* si $a y $b contienen los mismos elementos en el mismo orden.
!=	Desigualdad	$a! = $b	*True* si $a y $b no contienen los mismos elementos.
<>	Desigualdad	$a <> $b	Igual que !=.
!==	No identidad	$a! == $b	*True* si $a y $b no contienen los mismos elementos en el mismo orden.

Operadores de matriz

A tener en cuenta:

- La operación de unión adjunta una matriz a otra, pero no se sobrescriben elementos con las mismas claves.

```
$precios1 = array('neumaticos'=>100, 'aceite'=>10, 'bujias'=>4);
$precios2 = array('volantes'=>50, 'aceite'=>8, 'asientos'=>3);
$preciosTotal = $precios1 + $precios2;
```

- `$preciosTotal` contendrá los elementos `{'neumaticos'=>100, 'aceite'=>10, 'bujias'=>4, 'volantes'=>50, 'asientos'=>3}`

Contenidos

Definición de la matriz

Con índices numéricos:

```
$productos = array(
             array('NM','neumaticos',100),
             array('ACT','aceite',10),
             array('BUJ','bujias',4));
```

Con índices no numéricos:

```
$productos2 = array (
     array('Codigo'=>'NM','Descripcion'=>'neumaticos','Precio'=>100),
     array('Codigo'=>'ACT','Descripcion'=>'aceite','Precio'=>10),
     array('Codigo'=>'BUJ','Descripcion'=>'bujias','Precio'=>4));
```

Acceso a una matriz indexada numéricamente

Manualmente:

```
echo '|'.$productos[0][0].'|'.$productos[0][1].'|'.$productos[0][2].'|<br>';
echo '|'.$productos[1][0].'|'.$productos[1][1].'|'.$productos[1][2].'|<br>';
echo '|'.$productos[2][0].'|'.$productos[2][1].'|'.$productos[2][2].'|<br>';
```

Mediante bucles anidados:

```
for($fila=0;$fila<3;$fila++)
{
  for($columna=0;$columna<3;$columna++)
  {
    echo '|'.$productos[$fila][$columna];
  }
  echo '|<br>';
}
```

Acceso a una matriz asociativa

Mediante un bucle:

```
for($fila=0;$fila<3;$fila++)
{
  echo '|'.$productos2[$fila]['Codigo'].
       '|'.$productos2[$fila]['Descripcion'].
       '|'.$productos2[$fila]['Precio'].'|<br>';
}
```

Acceso a una matriz asociativa

Mediante bucles anidados (`foreach`):

```
for($fila=0;$fila<3;$fila++)
{
  foreach($productos2[$fila] as $clave=>$valor)
  {
    echo '|'.$valor;
  }
  echo '|<br>';
}
```

Mediante bucles anidados (`while`):

```
for($fila=0;$fila<3;$fila++)
{
  while(list($clave,$valor) = each($productos2[$fila]))
  {
    echo '|'.$valor;
  }
  echo '|<br>';
}
```

Contenidos

Ordenación de matrices

La función `sort`

Ordenación alfabética

```
$productos = array('neumaticos','aceite','bujias');
sort($productos); //Devuelve: aceite bujias neumaticos
```

Ordenación numérica

```
$precios = array(100,10,4);
sort($precios); // Devuelve: 4 10 100
```

- Los elementos de la matriz se ordenan ascendentemente.

Ordenación de matrices

Opciones de la función `sort`

Ordena numéricamente: SORT_NUMERIC

```
$matriz = array(2,22,12);
sort($matriz,SORT_NUMERIC); //Devuelve 2,12,22
```

Ordena como cadenas de texto: SORT_STRING

```
$matriz = array(2,22,12);
sort($matriz,SORT_STRING); //Devuelve 12,2,22
```

Ordenación de matrices asociativas

Ordenación por valor: `asort`

```
$precios = array('neumaticos'=>100, 'aceite'=>10, 'bujias'=>4);
asort($precios);
// Devuelve:
    bujias = 4
    aceite = 10
    neumaticos = 100
```

Ordenación por clave: `ksort`

```
$precios = array('neumaticos'=>100, 'aceite'=>10, 'bujias'=>4);
ksort($precios);
// Devuelve:
    aceite = 10
    bujias = 4
    neumaticos = 100
```

Funciones inversas de ordenación

`rsort`, `arsort` y `krsort`

- Misma función que `sort`, `asort` y `ksort`.
- Ordenación en orden descendente.

Funciones de reordenación

Reordenación aleatoria: `shuffle()`. Ejemplo: `pag_ppal_garaje.php`

```
<?php
   $imagenes = array('rueda.jpg', 'aceite.jpg', 'bujia.jpg', 'puerta.jpg',
               'volante.jpg', 'termostato.jpg', 'limpiaparabrisas.jpg',
               'zapata.jpg', 'pedal_freno.jpg');
   shuffle($imagenes);
?>

<html>
<head>
   <title>Garaje Paco</title>
</head>
<body>
   <center>
    <h1>Garaje Paco</h1>
    <table width = 100%>
      <tr>
<?php
  for ( $i = 0; $i < 3; $i++ )
  {
    echo '<td align="center"><img src="';
    echo $imagenes[$i];
    echo '"width="100" height="100"></td>';
  }
?>
      </tr>
    </table>
  </center>
</body>
</html>
```

Funciones de reordenación

Adición de un elemento: `array_push()`

```
$numeros = array();
for($i=10;$i>0;$i--)
{
  array_push($numeros,$i);
}

for($i=0;$i<10;$i++)
{
  echo "$numeros[$i]";  //Devuelve: 10987654321
}
```

Eliminación del último elemento: `array_pop()`

```
array_pop($numeros);
for($i=0;$i<9;$i++)
{
  echo "$numeros[$i]";  //Devuelve: 1098765432
}
```

Inversión del orden de los elementos: `array_reverse()`

```
$numeros = range(1,10);
$numeros_reordenados = array_reverse($numeros);
for($i=0;$i<10;$i++)
{
   echo "$numeros_reordenados[$i]";  //Devuelve: 10987654321
}
```

Funciones de desplazamiento por una matriz

Todas las matrices constan de un puntero interno que apunta al elemento actual de la matriz. Podemos obtener otras posiciones de la matriz con las siguientes funciones:

Función	**Resultado**
`current($matriz)`	Devuelve el elemento actual.
`each($matriz)`	Devuelve el elemento actual y avanza el puntero.
`next($matriz)`	Avanza el puntero y devuelve el elemento actual.
`reset($matriz)`	Devuelve el puntero al primer elemento.
`end($matriz)`	Coloca el puntero en la última posición.
`prev($matriz)`	Retrocede el puntero y devuelve el elemento actual.

Ejemplo

```
$precios = array('neumaticos'=>100, 'aceite'=>10, 'bujias'=>4);

$elemento = end($precios);     //$elemento = 4
$elemento = current($precios); //$elemento = 4
$elemento = prev($precios);    //$elemento = 10
$elemento = current($precios); //$elemento = 10
$elemento = reset($precios);   //$elemento = 100
```

Contar elementos de una matriz

`count()` y `sizeof()`

```
$productos = array('neumaticos','aceite','bujias','aceite','neumaticos');
echo 'Tamaño de la matriz: '.count($productos).'<br>';
echo 'Tamaño de la matriz: '.sizeof($productos).'<br>';
   // Devuelve en ambos casos: 5
```

Número de elementos repetidos: `array_count_values()`

```
$repeticiones = array_count_values($productos);

foreach($repeticiones as $clave=>$valor)
   echo $clave.' = '.$valor.'<br>';

  //Devuelve:
    neumaticos = 2
    aceite = 2
    bujias = 1
```

Conversión de matrices en variables escalares

extract()

- Convierte una matriz asociativa en un conjunto de variables escalares que contengan los elementos de la matriz.

```
$precios = array('neumaticos'=>100, 'aceite'=>10, 'bujias'=>4);
extract($precios);

echo $neumaticos.'<br>';  // Devuelve 100
echo $aceite.'<br>';      // Devuelve 10
echo $bujias.'<br>';      // Devuelve 4
```

- Se crean variables con los nombres de cada una de las claves de la matriz.
- Dichas variables contienen los valores asociados a cada una de las claves.

Conversión de matrices en variables escalares

extract()

- Opciones más utilizadas:

EXTR_OVERWRITE:	Sobreescribe la variable existente cuando tiene lugar una colisión (comportamiento por defecto).
EXTR_PREFIX_ALL:	Coloca delante de todos los nombres de variables la cadena que se le indique.

- Ejemplo:

```
$precios = array('neumaticos'=>100, 'aceite'=>10, 'bujias'=>4);
extract($precios,EXTR_PREFIX_ALL,'articulo');
echo $articulo_neumaticos.'<br>';
echo $articulo_aceite.'<br>';
echo $articulo_bujias.'<br>';
```

Parte III

Manipulación de ficheros

Contenidos

Operaciones que vamos a realizar

- Escritura de archivos
  1. Abrir un archivo
  2. Escribir datos en el archivo
  3. Cerrar el archivo
- Lectura de archivos
  1. Abrir un archivo
  2. Leer datos del archivo
  3. Cerrar el archivo

Contenidos

Apertura de archivos: sintaxis

`fopen()`

```
@ $fp = fopen("$DOCUMENT_ROOT/pedidos.txt", 'ab');
```

- `@`: Supresión de error. Si se produce fallo, no muestra errores al usuario. Los controlaremos nosotros y mostraremos la información oportuna al usuario.
- `$fp`: Identificador de fichero.
- Parámetros: `fopen(`*ruta archivo*, *modo apertura*`)`

Apertura de archivos: sintaxis

`fopen()`

```
@ $fp = fopen("$DOCUMENT_ROOT/pedidos.txt", 'ab');
```

- `@`: Supresión de error. Si se produce fallo, no muestra errores al usuario. Los controlaremos nosotros y mostraremos la información oportuna al usuario.
- `$fp`: Identificador de fichero.
- Parámetros: `fopen(`*ruta archivo*, *modo apertura*`)`

Apertura de archivos: sintaxis

`fopen()`

```
@ $fp = fopen("$DOCUMENT_ROOT/pedidos.txt", 'ab');
```

- `@`: Supresión de error. Si se produce fallo, no muestra errores al usuario. Los controlaremos nosotros y mostraremos la información oportuna al usuario.
- `$fp`: Identificador de fichero.
- Parámetros: `fopen(`*ruta archivo*, *modo apertura*`)`

Apertura de archivos: sintaxis

`fopen()`

```
@ $fp = fopen("$DOCUMENT_ROOT/pedidos.txt", 'ab');
```

- `@`: Supresión de error. Si se produce fallo, no muestra errores al usuario. Los controlaremos nosotros y mostraremos la información oportuna al usuario.
- `$fp`: Identificador de fichero.
- Parámetros: `fopen(`*ruta archivo*, *modo apertura*`)`

Apertura de archivos: opciones

@: Supresión de error

```
@ $fp = fopen("$DOCUMENT_ROOT/pedidos.txt", 'ab');

if (!$fp)
{
 echo '<p><strong> Su pedido no puede ser procesado en este momento.  '
 .'Inténtelo más tarde.</strong></p></body></html>';
 exit;
}
```

Ruta del archivo

```
$DOCUMENT_ROOT/pedidos.txt
```

- `$HTTP_SERVER_VARS['DOCUMENT_ROOT']`: Variable predefinida del servidor.
- Utilizamos la variable abreviada `$DOCUMENT_ROOT`.
- Apunta a la base del árbol de documentos del servidor web (`/var/www`, en nuestro caso).
- Si no se especifica la ruta, el archivo se busca en el mismo directorio que el *script* php.
- Direcciones relativas: `$DOCUMENT_ROOT/../pedidos.txt` busca un fichero en el directorio superior a `$DOCUMENT_ROOT` (`/var` en nuestro caso).

Apertura de archivos: opciones

Modo de apertura

Modo	Descripción
`r`	*Lectura*
`r+`	*Escritura y lectura*, comenzando por el inicio del archivo.
`w`	*Escritura*. Si el archivo ya existe, borra el contenido. Si no existe, intenta crearlo.
`w+`	*Escritura y lectura*. Si el archivo ya existe, borra el contenido. Si no existe, intenta crearlo.
`x`	*Escritura con advertencia*. Si el archivo existe, no lo abre, devuelve `false`, y se genera una advertencia.
`x+`	*Escritura y lectura con advertencia*. Si el archivo existe, no lo abre, devuelve `false`, y se genera una advertencia.
`a`	*Adjunción*. Comienza a escribir al final de los datos existentes. Si el archivo no existe, intenta crearlo.
`a+`	*Adjunción*. Abre para lectura y escritura. Comienza a escribir al final de los datos existentes. Si el archivo no existe, intenta crearlo.
`b`	*Modo binario*. Se utiliza junto con una de las opciones anteriores. Portabilidad para sistemas que distingan entre archivos binarios y de texto (Windows).
`t`	*Modo texto*. Exclusiva para Windows.

Cierre de archivos: sintaxis

fclose()

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

- Devuelve *true* si ha conseguido cerrar correctamente el fichero apuntado por `$fp`, y *false* en caso contrario.

Contenidos

Sintaxis

`fwrite()`

```
fwrite($fp, $stringPedido, strlen($stringPedido));
```

- Escribe la cadena almacenada en `$stringPedido` en el archivo al que apunta `$fp` hasta alcanzar el final de la cadena o escribir un número de bytes igual a `strlen($stringPedido)` (optativo).
- Parámetros: `fwrite(`*puntero*, *string cadena* [,*int longitud*]`)`

Alternativa: `file_put_contents()`

```
file_put_contents(string archivo, string datos);
```

- Escribe la cadena *datos* en el archivo *archivo*, sin necesidad de invocar a `fopen()` ni `fclose()`.

Sintaxis

`fwrite()`

```
fwrite($fp, $stringPedido, strlen($stringPedido));
```

- Escribe la cadena almacenada en `$stringPedido` en el archivo al que apunta `$fp` hasta alcanzar el final de la cadena o escribir un número de bytes igual a `strlen($stringPedido)` (optativo).
- Parámetros: `fwrite(`*puntero*, *string cadena* [,*int longitud*]`)`

Alternativa: `file_put_contents()`

```
file_put_contents(string archivo, string datos);
```

- Escribe la cadena *datos* en el archivo *archivo*, sin necesidad de invocar a `fopen()` ni `fclose()`.

Sintaxis

`fwrite()`

```
fwrite($fp, $stringPedido, strlen($stringPedido));
```

- Escribe la cadena almacenada en `$stringPedido` en el archivo al que apunta `$fp` hasta alcanzar el final de la cadena o escribir un número de bytes igual a `strlen($stringPedido)` (optativo).
- Parámetros: `fwrite(`*puntero*, *string cadena* [,*int longitud*]`)`

Alternativa: `file_put_contents()`

```
file_put_contents(string archivo, string datos);
```

- Escribe la cadena *datos* en el archivo *archivo*, sin necesidad de invocar a `fopen()` ni `fclose()`.

Sintaxis

`fwrite()`

```
fwrite($fp, $stringPedido, strlen($stringPedido));
```

- Escribe la cadena almacenada en `$stringPedido` en el archivo al que apunta `$fp` hasta alcanzar el final de la cadena o escribir un número de bytes igual a `strlen($stringPedido)` (optativo).
- Parámetros: `fwrite(`*puntero*, *string cadena* [,*int longitud*]`)`

Alternativa: `file_put_contents()`

```
file_put_contents(string archivo, string datos);
```

- Escribe la cadena *datos* en el archivo *archivo*, sin necesidad de invocar a `fopen()` ni `fclose()`.

Sintaxis

`fwrite()`

```
fwrite($fp, $stringPedido, strlen($stringPedido));
```

- Escribe la cadena almacenada en `$stringPedido` en el archivo al que apunta `$fp` hasta alcanzar el final de la cadena o escribir un número de bytes igual a `strlen($stringPedido)` (optativo).
- Parámetros: `fwrite(`*puntero*, *string cadena* [,*int longitud*]`)`

Alternativa: `file_put_contents()`

```
file_put_contents(string archivo, string datos);
```

- Escribe la cadena *datos* en el archivo *archivo*, sin necesidad de invocar a `fopen()` ni `fclose()`.

Ejemplo

almacenapedido.html

```
<html>
<body>
 <h1>Garaje Paco</h1>
 <h2>Formulario de Pedido</h2>
 <p>
 <form action="almacenapedido.php"
  method=post>

  <table border="0">
  <tr bgcolor="#CCCCCC">
   <td width="150">Artículo</td>
   <td width="15">Cantidad</td>
  </tr>

  <tr>
   <td>Neumáticos</td>
   <td align="center"><input type="text"
    name="neumatico" size="3"
    maxlength="3"></td>
  </tr>

  <tr>
   <td>Aceite</td>
   <td align="center"><input type="text"
    name="aceite" size="3"
    maxlength="3"></td>
  </tr>

  <tr>
   <td>Bujías</td>
   <td align="center"><input type="text"
    name="bujias" size="3"
    maxlength="3"></td>
  </tr>

  <tr>
   <td>Dirección de envío</td>
   <td align=center><input type="text"
    name="direccion" size=40
    maxlength=40></td>
  </tr>

  <tr>
   <td colspan="2" align="center">
   <input type="submit" value="Enviar"
    name="EnviarOrden"></td>
  </tr>

  </table>

</body>
</html>
```

Ejemplo

almacenapedido.php (I)

```
<?php
  $neumaticos = $_POST['neumatico'];
  $aceite = $_POST['aceite'];
  $bujias = $_POST['bujias'];
  $direccion = $_POST['direccion'];

  $DOCUMENT_ROOT = $_SERVER['DOCUMENT_ROOT'];
?>
<html>
<head>
  <title>Resultado del pedido</title>
</head>

<body>
<h1>Garaje Paco</h1>
<h2>Resultado del envío del pedido:</h2>

<?php
 $fecha = date('H:i, jS F');

 echo '<p>Orden procesada a las ';
 echo $fecha;
 echo '</p>';

 $total = 0;
 $total = $neumaticos + $aceite + $bujias;

if( $total == 0)
{
 echo 'No se ha solicitado ningún
    artículo!<br />';
 exit;
}
else
{
 echo 'Artículos solicitados: '
   .$total.'<br />';

 if ( $neumaticos>0 )
  echo $neumaticos.' neumaticos<br/>';
 if ( $aceite>0 )
  echo $aceite.' garrafas de
    aceite<br/>';
 if ( $bujias>0 )
  echo $bujias.' bujías<br/>';
}

$cantidadTotal = 0.00;

define('PRECIONEUMATICO', 100);
define('PRECIOACEITE', 10);
define('PRECIOBUJIA', 4);
```

Ejemplo

almacenapedido.php (y II)

```
$cantidadTotal = $neumaticos * PRECIONEUMATICO + $aceite * PRECIOACEITE +
                 $bujias * PRECIOBUJIA;

$cantidadTotal=number_format($cantidadTotal, 2, '.', ' ');

echo '<p>El total del pedido es '.$cantidadTotal.' euros </p>';
echo '<p>Dirección de envío del pedido: '.$direccion.'</p>';

$stringPedido = $fecha."\t".$neumaticos." neumáticos \t".$aceite." garrafas aceite\t"
    .$bujias." bujías\t euros".$cantidadTotal ."\t". $direccion."\n";

@ $fp = fopen("$DOCUMENT_ROOT/pedidos.txt", 'ab');  // abrimos archivo

flock($fp,LOCK_EX);
if (!$fp)
{
 echo '<p><strong> Su pedido no puede ser procesado en este momento.  '
 .'Inténtelo más tarde.</strong></p></body></html>';
 exit;
}

fwrite($fp, $stringPedido, strlen($stringPedido));
flock($fp,LOCK_UN);
fclose($fp);

echo '<p><b>Pedido almacenado.</b></p>';
?>
</body>
</html>
```

Contenidos

Sintaxis

- Previamente es necesario abrir el fichero con opciones de lectura

```
@ $fp = fopen("$DOCUMENT_ROOT/pedidos.txt", 'rb');
```

Ejemplo

verpedido.php

```
<?php
  $DOCUMENT_ROOT = $_SERVER['DOCUMENT_ROOT'];
?>
<html>
<head> <title>Garaje Paco: pedidos</title> </head>
<body>
  <h1>Garaje Paco</h1>
  <h2>Pedidos</h2>
<?php
  @  $fp = fopen("$DOCUMENT_ROOT/pedidos.txt", 'rb');

  if (!$fp)
  {
    echo '<p><strong>No hay pedidos pendientes.'
    .'Por favor, inténtelo más tarde.</strong></p>';
    exit;
  }

  while (!feof($fp))
  {
    $pedido= fgets($fp, 999);
    echo $pedido.'<br />';
  }
  fclose($fp);
?>
</body>
</html>
```

¿Qué ha pasado aquí?

A tener en cuenta

- El fichero se intenta abrir con `@`, para que no muestre errores. Si el fichero no existe, `fopen()` devuelve *false*. Lo comprobamos con el `if`:

```
if (!$fp) // Comprueba si if == false
```

- Para indicar que se lea mientras no se alcance el final del fichero, se utiliza la función `feof()`:

```
while (!$feof($fp))
```

- Para leer texto de un archivo, línea a línea, utilizamos `fgets()`:

```
$pedido = fgets($fp, 999);
```

`fgets()` lee hasta encontrar un caracter de nueva línea (\n), un fin de fichero (`EOF`), o haber leído 998 bytes del archivo.

¿Qué ha pasado aquí?

A tener en cuenta

- El fichero se intenta abrir con `@`, para que no muestre errores. Si el fichero no existe, `fopen()` devuelve *false*. Lo comprobamos con el `if`:

```
if (!$fp) // Comprueba si if == false
```

- Para indicar que se lea mientras no se alcance el final del fichero, se utiliza la función `feof()`:

```
while (!$feof($fp))
```

- Para leer texto de un archivo, línea a línea, utilizamos `fgets()`:

```
$pedido = fgets($fp, 999);
```

`fgets()` lee hasta encontrar un caracter de nueva línea (\n), un fin de fichero (`EOF`), o haber leído 998 bytes del archivo.

¿Qué ha pasado aquí?

A tener en cuenta

- El fichero se intenta abrir con `@`, para que no muestre errores. Si el fichero no existe, `fopen()` devuelve *false*. Lo comprobamos con el `if`:

```
if (!$fp) // Comprueba si if == false
```

- Para indicar que se lea mientras no se alcance el final del fichero, se utiliza la función `feof()`:

```
while (!$feof($fp))
```

- Para leer texto de un archivo, línea a línea, utilizamos `fgets()`:

```
$pedido = fgets($fp, 999);
```

`fgets()` lee hasta encontrar un caracter de nueva línea (\n), un fin de fichero (`EOF`), o haber leído 998 bytes del archivo.

Alternativas a `fgets()`

- `fgetss()` funciona igual que `fgets()` salvo que elimina, por motivos de seguridad, todas las etiquetas HTML y PHP, excepto las indicadas en la cadena *etiquetas_permitidas*.

```
string fgetss(resource fp, int longitud, [string etiquetas_permitidas]);
```

- `fgetcsv()` divide cada línea del archivo leído en campos utilizando el carácter de delimitación *delimitador*. Los campos se almacenan como elmentos del array devuelto por la función.

```
array fgetcsv(resource fp, int longitud, [string delimitador]);
```

Alternativas a `fgets()`

- `fgetss()` funciona igual que `fgets()` salvo que elimina, por motivos de seguridad, todas las etiquetas HTML y PHP, excepto las indicadas en la cadena *etiquetas_permitidas*.

```
string fgetss(resource fp, int longitud, [string etiquetas_permitidas]);
```

- `fgetcsv()` divide cada línea del archivo leído en campos utilizando el carácter de delimitación *delimitador*. Los campos se almacenan como elmentos del array devuelto por la función.

```
array fgetcsv(resource fp, int longitud, [string delimitador]);
```

Lectura de todo el archivo

- `readfile()` abre un archivo, imprime el contenido en el navegador, y cierra el archivo.

```
int readfile(string nombre_de_archivo);
```

- `fpassthru()` muestra el contenido del fichero, tras abrirlo con `fopen()`, desde la posición del puntero (`fp`, en nuestros ejemplos) hasta el final del fichero.

```
int fpassthru(resource fp);
```

- `file()` es idéntico a `readfile()`, pero cada línea la almacena como un elemento de la matriz que devuelve. Ejemplo:

```
$array_archivo = file("$DOCUMENT_ROOT/pedidos.txt");

foreach($array_archivo as $linea)
{
   echo $linea.'<br>';
}
```

Lectura de todo el archivo

- `readfile()` abre un archivo, imprime el contenido en el navegador, y cierra el archivo.

```
int readfile(string nombre_de_archivo);
```

- `fpassthru()` muestra el contenido del fichero, tras abrirlo con `fopen()`, desde la posición del puntero (`fp`, en nuestros ejemplos) hasta el final del fichero.

```
int fpassthru(resource fp);
```

- `file()` es idéntico a `readfile()`, pero cada línea la almacena como un elemento de la matriz que devuelve. Ejemplo:

```
$array_archivo = file("$DOCUMENT_ROOT/pedidos.txt");

foreach($array_archivo as $linea)
{
   echo $linea.'<br>';
}
```

Lectura de todo el archivo

- `readfile()` abre un archivo, imprime el contenido en el navegador, y cierra el archivo.

```
int readfile(string nombre_de_archivo);
```

- `fpassthru()` muestra el contenido del fichero, tras abrirlo con `fopen()`, desde la posición del puntero (`fp`, en nuestros ejemplos) hasta el final del fichero.

```
int fpassthru(resource fp);
```

- `file()` es idéntico a `readfile()`, pero cada línea la almacena como un elemento de la matriz que devuelve. Ejemplo:

```
$array_archivo = file("$DOCUMENT_ROOT/pedidos.txt");

foreach($array_archivo as $linea)
{
   echo $linea.'<br>';
}
```

Otros modos de lectura

- `fgetc()` lee el siguiente carácter del fichero especificado. Ejemplo:

```
while(!feof($fp))
{
  $caracter = fgetc($fp);
  if ($caracter == "\n")
    echo '<br>';
  else
    echo $caracter;
}
```

- `fread()` lee un número arbitrario de bytes de un archivo, o hasta final de fichero, si el número de bytes es superior al tamaño del fichero, y lo devuelve como string. Ejemplo:

```
$cadena = fread($fp,300);
echo $cadena;
```

Otros modos de lectura

- `fgetc()` lee el siguiente carácter del fichero especificado. Ejemplo:

```
while(!feof($fp))
{
  $caracter = fgetc($fp);
  if ($caracter == "\n")
    echo '<br>';
  else
    echo $caracter;
}
```

- `fread()` lee un número arbitrario de bytes de un archivo, o hasta final de fichero, si el número de bytes es superior al tamaño del fichero, y lo devuelve como string. Ejemplo:

```
$cadena = fread($fp,300);
echo $cadena;
```

Otras funciones:

- `file_exists()`: comprueba la existencia de un archivo sin abrirlo. Ejemplo:

```
if(file_exists("$DOCUMENT_ROOT/pedidos.txt"))
{
   echo 'Existen pedidos pendientes de ser procesados <br>';
}else
{
   echo 'No existen pedidos pendientes de ser procesados <br>';
}
```

- `filesize()` devuelve el tamaño de un fichero. Ejemplo:

```
$fp = fopen("$DOCUMENT_ROOT/pedidos.txt", 'rb');
echo nl2br(fread($fp,filesize("$DOCUMENT_ROOT/pedidos.txt")));
fclose($fp);
```

La función `nl2br` transforma los caracteres de nueva línea (\n) en saltos de línea en HTML `<br>`.

- `unlink()` elimina un fichero. Ejemplo:

```
unlink("$DOCUMENT_ROOT/pedidos.txt");
```

Otras funciones:

- `file_exists()`: comprueba la existencia de un archivo sin abrirlo. Ejemplo:

```
if(file_exists("$DOCUMENT_ROOT/pedidos.txt"))
{
   echo 'Existen pedidos pendientes de ser procesados <br>';
}else
{
   echo 'No existen pedidos pendientes de ser procesados <br>';
}
```

- `filesize()` devuelve el tamaño de un fichero. Ejemplo:

```
$fp = fopen("$DOCUMENT_ROOT/pedidos.txt", 'rb');
echo nl2br(fread($fp,filesize("$DOCUMENT_ROOT/pedidos.txt")));
fclose($fp);
```

La función `nl2br` transforma los caracteres de nueva línea (\n) en saltos de línea en HTML `<br>`.

- `unlink()` elimina un fichero. Ejemplo:

```
unlink("$DOCUMENT_ROOT/pedidos.txt");
```

Otras funciones:

- `file_exists()`: comprueba la existencia de un archivo sin abrirlo. Ejemplo:

```
if(file_exists("$DOCUMENT_ROOT/pedidos.txt"))
{
   echo 'Existen pedidos pendientes de ser procesados <br>';
}else
{
   echo 'No existen pedidos pendientes de ser procesados <br>';
}
```

- `filesize()` devuelve el tamaño de un fichero. Ejemplo:

```
$fp = fopen("$DOCUMENT_ROOT/pedidos.txt", 'rb');
echo nl2br(fread($fp,filesize("$DOCUMENT_ROOT/pedidos.txt")));
fclose($fp);
```

La función `nl2br` transforma los caracteres de nueva línea (\n) en saltos de línea en HTML `<br>`.

- `unlink()` elimina un fichero. Ejemplo:

```
  unlink("$DOCUMENT_ROOT/pedidos.txt");
```

Otras funciones:

Desplazarse dentro de un archivo

- `rewind()`: restablece el puntero del archivo al principio de éste.
- `ftell()`: indica la distancia, en bytes, a la que se encuentra el puntero dentro del archivo. Ejemplo:

```
echo 'Posición del puntero del fichero: '.ftell($fp).'<br>';
rewind($fp);
echo 'Posición del puntero del fichero tras rebobinar: '.ftell($fp).'<br>';
```

- `fseek()` establece el puntero del archivo en otro punto del archivo. Sintaxis:

```
int fseek(resource fp, int desplazamiento [, int punto_de_partida]);
```

  - Por defecto, establece el puntero *desplazamiento* bytes desde el inicio del archivo.
  - Si `punto_de_partida = SEEK_CUR`, se establece el desplazamiento desde la posición actual del puntero.
  - Si `punto_de_partida = SEEK_END`, se establece el desplazamiento desde la posición final del fichero.

Otras funciones:

Desplazarse dentro de un archivo

- `rewind()`: restablece el puntero del archivo al principio de éste.
- `ftell()`: indica la distancia, en bytes, a la que se encuentra el puntero dentro del archivo. Ejemplo:

```
echo 'Posición del puntero del fichero: '.ftell($fp).'<br>';
rewind($fp);
echo 'Posición del puntero del fichero tras rebobinar: '.ftell($fp).'<br>';
```

- `fseek()` establece el puntero del archivo en otro punto del archivo. Sintaxis:

```
int fseek(resource fp, int desplazamiento [, int punto_de_partida]);
```

  - Por defecto, establece el puntero *desplazamiento* bytes desde el inicio del archivo.
  - Si `punto_de_partida = SEEK_CUR`, se establece el desplazamiento desde la posición actual del puntero.
  - Si `punto_de_partida = SEEK_END`, se establece el desplazamiento desde la posición final del fichero.

Otras funciones:

Bloqueo de archivos

- Dos usuarios podrían intentar escribir simultáneamente en un fichero, dando lugar a un conflicto.
- Para evitarlo, se pueden bloquear los ficheros para que únicamente puedan ser accedidos en un momento determinado por un sólo usuario. Se utiliza `flock()`:

```
bool flock(resource fp, int operacion);
```

- Posibles valores del parámetro *operación*:

Operación	Descripción
`LOCK_SH`	*Bloqueo de lectura*. El archivo se puede compartir con otros lectores.
`LOCK_EX`	*Bloqueo de escritura*. El archivo no se puede compartir con nadie.
`LOCK_UN`	Liberar bloqueo existente.
`LOCK_NB`	Se impide el bloqueo al intentar obtenerlo.

Otras funciones:

Bloqueo de archivos

- Dos usuarios podrían intentar escribir simultáneamente en un fichero, dando lugar a un conflicto.
- Para evitarlo, se pueden bloquear los ficheros para que únicamente puedan ser accedidos en un momento determinado por un sólo usuario. Se utiliza `flock()`:

```
bool flock(resource fp, int operacion);
```

- Posibles valores del parámetro *operación*:

Operación	**Descripción**
`LOCK_SH`	*Bloqueo de lectura*. El archivo se puede compartir con otros lectores.
`LOCK_EX`	*Bloqueo de escritura*. El archivo no se puede compartir con nadie.
`LOCK_UN`	Liberar bloqueo existente.
`LOCK_NB`	Se impide el bloqueo al intentar obtenerlo.

Parte IV

Reutilización de código: funciones

Contenidos

¿Por qué reutilizar código?

- Permite desarrollar código más coherente y fiable
- Contribuye a reducir los costes
- El código es más sencillo de mantener con menos esfuerzo
- Mejora la uniformidad de los resultados
- Permite crear nuevos proyectos combinando componentes de código ya existente, con un mínimo desarrollo desde cero

Contenidos

Utilizar `require()`

- Permite cargar un archivo en la secuencia de comandos de un programa en PHP.
- El archivo puede contener todos los elementos que se suelen incluir en una secuencia de comandos (instrucciones PHP, etiquetas HTML, etc.)

fecha.php

```
<?php
  echo '<p>'.date('H:i, jS F');
?>
```

index.php

```
<html>
<body>

  <?php
    echo '<h3>La fecha actual es:</h3>';
    require('fecha.php');
    echo '<p>Terminada inclusión de archivos';
  ?>

</body>
</html>
```

Utilizar `require()`

- Permite cargar un archivo en la secuencia de comandos de un programa en PHP.
- El archivo puede contener todos los elementos que se suelen incluir en una secuencia de comandos (instrucciones PHP, etiquetas HTML, etc.)

fecha.php

```
<?php
  echo '<p>'.date('H:i, jS F');
?>
```

index.php

```
<html>
<body>

  <?php
    echo '<h3>La fecha actual es:</h3>';
    require('fecha.php');
    echo '<p>Terminada inclusión de archivos';
  ?>

</body>
</html>
```

Utilizar `require()`

- Permite cargar un archivo en la secuencia de comandos de un programa en PHP.
- El archivo puede contener todos los elementos que se suelen incluir en una secuencia de comandos (instrucciones PHP, etiquetas HTML, etc.)

fecha.php

```
<?php
  echo '<p>'.date('H:i, jS F');
?>
```

index.php

```
<html>
<body>

  <?php
    echo '<h3>La fecha actual es:</h3>';
    require('fecha.php');
    echo '<p>Terminada inclusión de archivos';
  ?>

</body>
</html>
```

Utilizar `include()`

- Casi idéntica a `require()`.
- Diferencia: cuando fallan, `require()` devuelve un error y termina, mientras que `include()` devuelve una advertencia.

`require_once()` e `include_once()`

- Funcionan igual que `include()` y `require()`, pero se asegura de que el archivo especificado sólo se incluye una vez.

Contenidos

Llamar a funciones

Sin parámetros

- No requieren parámetros (valores de entrada a la función, que van entre los paréntesis).

```
phpinfo();
```

Con parámetros

- Se pasan colocando los datos o el nombre de una variable con los datos dentro del paréntesis.

```
funcion(2);
funcion(2,'Enero');
funcion(2,$mes);
date('H:i, jS F');
```

Llamar a funciones

Sin parámetros

- No requieren parámetros (valores de entrada a la función, que van entre los paréntesis).

```
phpinfo();
```

Con parámetros

- Se pasan colocando los datos o el nombre de una variable con los datos dentro del paréntesis.

```
funcion(2);
funcion(2,'Enero');
funcion(2,$mes);
date('H:i, jS F');
```

Sintaxis de una función

Ejemplo

```
string fgetss(resource fp, int longitud, [string etiquetas_permitidas]);
```

- 3 parámetros de entrada:
  - `resource fp`: *fp* deberá ser de tipo *puntero a fichero*.
  - `int longitud`: *longitud* deberá ser de tipo entero.
  - `[string etiquetas_permitidas]`: **opcional**. De tipo *string*.
- Salida:
  - `string`: la función devuelve una cadena de texto (*string*).

Uso

```
$fp = fopen("pedidos.txt", 'rb');
$longitud = 999;
$vars = "<h1> <h2>";

$pedido = fgetss($fp, $longitud, $vars);
```

Sintaxis de una función

Ejemplo

```
string fgetss(resource fp, int longitud, [string etiquetas_permitidas]);
```

- 3 parámetros de entrada:
  - `resource fp`: *fp* deberá ser de tipo *puntero a fichero*.
  - `int longitud`: *longitud* deberá ser de tipo entero.
  - `[string etiquetas_permitidas]`: **opcional**. De tipo *string*.
- Salida:
  - `string`: la función devuelve una cadena de texto (*string*).

Uso

```
$fp = fopen("pedidos.txt", 'rb');
$longitud = 999;
$vars = "<h1> <h2>";

$pedido = fgetss($fp, $longitud, $vars);
```

Sintaxis de una función

Uso de mayúsculas y minúsculas

```
$pedido = fgetSs($fp, $longitud, $vars);
$pedido = Fgetss($fp, $longitud, $vars);
$pedido = FGETSS($fp, $longitud, $vars);
```

- Al contrario que con los nombres de variables, las llamadas a funciones no discriminan entre mayúsculas y minúsculas.
- Se recomienda mantener cierta uniformidad.

Contenidos

Estructura de una función

Declaración

- Se utiliza la palabra clave `function`.
- Se incluye el nombre de la función, los parámetros de entrada, y el código que se ejecutará cada vez que se invoque a la función.
- Un ejemplo sencillo:

```
<?php
 function mi_funcion()
 {
    echo 'mi_funcion ha sido invocada';
 }
?>
```

- O también:

```
<?php
  function mi_funcion()
  {
?>
    mi_funcion ha sido invocada
<?php
  }
?>
```

Designación de funciones

- No se puede utilizar el mismo nombre asignado a una función existente.
- El nombre de la función sólo puede contener letras, dígitos y guiones bajos.
- El nombre de la función no puede comenzar por un dígito.

Parámetros

- Permiten pasar datos dentro de una función.

```
function muestra_tabla($matrizdatos)
{
  echo '<table border = 1>';
  reset($matrizdatos);
  $valor = current($matrizdatos);

  while($valor)
  {
    echo "<tr><td>$valor</td></tr>";
    $valor = next($matrizdatos);
  }
  echo "</table>";
}
```

- Podemos invocarlo así:

```
$datos = array('linea uno','linea dos','linea tres');
muestra_tabla($datos);
```

Parámetros opcionales

- En el ejemplo anterior, pueden definirse parámetros opcionales, con valores predeterminados en caso de que el invocador no los pase a la función.

```
function muestra_tabla2($matrizdatos,$borde=1,$espaciado=4,$espaciadocelda=4)
{
  echo "<table border = $borde cellspacing = $espaciado
                        cellpadding = $espaciadocelda>";
  reset($matrizdatos);
  $valor = current($matrizdatos);

  while($valor)
  {
    echo "<tr><td>$valor</td></tr>";
    $valor = next($matrizdatos);
  }
  echo "</table>";
}
```

- Se puede invocar sin parámetros optativos o con ellos:

```
$datos = array('linea uno','linea dos','linea tres');
muestra_tabla2($datos);
muestra_tabla2($datos,1,5);
```

Número variable de parámetros

- Se pueden declarar funciones que acepten un número variable de parámetros.

```
function prueba_de_argumentos()
{
  echo "Número de parámetros: ";
  echo func_num_args();

  echo "<br>";
  $argumentos = func_get_args();

  foreach($argumentos as $arg)
  {
    echo $arg.'<br>';
  }
}
```

- Invocación:

```
prueba_de_argumentos('a','b','c');
```

  - `func_num_args()`: Devuelve el número de parámetros pasados a la función.
  - `func_get_args()`: Devuelve una matriz con los elementos.
  - `func_get_arg()`: Permite acceder a los argumentos uno a uno, pasándole como parámetro el número de argumento, comenzando por 0.

Ámbito

El ámbito de una variable controla dónde resulta visible y el lugar en el que se puede utilizar. Reglas:

- **Variables locales:** El ámbito de una variable declarada dentro de una función abarca desde el punto en que es declarada hasta el final de la función.
- **Variables globales:** El ámbito de una variable declarada fuera de una función abarca desde el punto en que es declarada hasta el final del archivo, pero no en el interior de las funciones.
- **Variables superglobales:** Visibles tanto dentro como fuera de las funciones.
- El uso de `require()` e `include()` no afecta al ámbito. Se aplicará el ámbito del lugar donde se utilicen esas instrucciones.
- Con la palabra clave `global` podemos especificar manualmente que una variable definida o utilizada dentro de una función tiene ámbito global.
- Las variables se pueden eliminar manualmente llamando a `unset($variable)`.

Paso de parámetros por valor y por referencia

Paso por valor

- Al pasar un parámetro, se creará una nueva variable que contiene el valor pasado. Es una **copia** del original, y su modificación no afectará al valor de la variable original. Ejemplo:

```
function incremento($valor, $cantidad = 1)
{
  $valor = $valor + $cantidad;
}
```

- Llamada a la función:

```
$valor = 10;
incremento($valor);
echo $valor;
```

- El resultado de `$valor` fuera de la función no varía.

Paso de parámetros por valor y por referencia

Paso por referencia

- Al pasar un parámetro, en vez de crear una nueva variable, la función recibe una referencia a la variable original. Cualquier cambio realizado sobre esa referencia afecta a la variable original. Ejemplo:

```
function incremento(&$valor, $cantidad = 1)
{
  $valor = $valor + $cantidad;
}
```

- Llamada a la función:

```
$a = 10;
echo $a.'<br>';
incremento($a);
echo $a.'<br>';
```

- El resultado de `$valor` fuera de la función varía. Nótese que para definir el paso de una variable por referencia, se utiliza el símbolo "`&`".

Devolver valores desde una función

- Con `return` se detiene la ejecución de una función. Ejemplo:

```
function prueba_return()
{
  echo 'Esta sentencia se ejecutará';
  return;
  echo 'Esta sentencia no se llegará a ejecutar';
}
```

- También se utiliza para devolver valores:

```
function mayor($x,$y)
{
  if(!isset($x)||!isset($y))
      return false;
  else if ($x>=$y)
      return $x;
  else
      return $y;
}
```

Parte V

Control de sesiones

Contenidos

Control de sesiones

- HTTP es un protocolo **sin estado**: no permite conservar el estado (datos, información sobre el usuario, etc.) entre dos transacciones. No es posible saber si dos solicitudes provienen de un mismo usuario.
- El **control de sesiones** permite realizar el seguimiento del usuario durante una sola sesión en un sitio Web.
- **Utilidad:** personalización del entorno según las preferencias del usuario, accesos a determinadas zonas según su nivel de autorización, implementación de carritos de la compra...

Control de sesiones en PHP

- El control y seguimiento de una sesión se hace mediante un **Id. de sesión**: número aleatorio que se almacena en el ordenador del usuario en forma de *cookie* o pasándolo en la URL.
- Proceso para implementar sesiones en PHP:
  - Iniciar una sesión.
  - Registrar variables de sesión.
  - Utilizar variables de sesión.
  - Anular las variables registradas y eliminar la sesión.

Contenidos

Implementación de sesiones

- Con PHP no es necesario definir las *cookies* manualmente.
- Para trabajar con una sesión, primero es necesario inicializarla:

```
session_start();
```

- Comprueba si ya existe un Id. de sesión actual.
  - Si no existe, crea un Id nuevo (una nueva sesión).
  - Si existe, carga las variables de sesión registradas.
- Se debe invocar `session_start()` al inicio de todas las secuencias de comandos PHP que trabajen con sesiones.

Registro y utilización de variables de sesión

- *Registrar una variable*: Almacenarla en la sesión para que pueda ser accedida desde cualquier secuencia de comandos PHP mientras dure la sesión o hasta que sea anulada manualmente.

```
$_SESSION['variable'] = 5;
```

- Para acceder a una variable de sesión:

```
$var = $_SESSION['variable'];
```

- Es aconsejable comprobar siempre si esa variable está registrada en la sesión, con `isset()` o `empty()`:

```
if(isset($_SESSION['variable']))...
```

Registro y utilización de variables de sesión

- Para anular el registro de una variable:

```
unset($_SESSION['variable']);
```

- Una vez terminada la sesión, es aconsejable anular el registro de todas las variables, borrando el Id. de sesión:

```
session_destroy();
```

Contenidos

Ejemplo de inicio de sesión y paso de variables

- Creación de una sesión e inicialización de una variable:

iniciosesion.php

```
<?php
 session_start();

 $_SESSION['var_sesion'] = "Hola CEI !";

 echo '<h3>iniciosesion.php:</h3>
    <p>El contenido de $_SESSION[\'var_sesion\'] es '
    .$_SESSION['var_sesion'].'<br />';
?>

<a href="accesosesion.php">Siguiente página</a>
```

Ejemplo de inicio de sesión y paso de variables

- Acceso a la variable de sesión desde otra página y anulación del registro de la variable:

accesosesion.php

```
<?php
 session_start();

 echo '<h3>accesosesion.php:</h3>
    <p>El contenido de $_SESSION[\'var_sesion\'] es '
    .$_SESSION['var_sesion'].'<br />';

 unset($_SESSION['var_sesion']);
?>

<a href="finalsesion.php">Siguiente página</a>
```

Ejemplo de inicio de sesión y paso de variables

- Se intenta acceder a la variable de sesión desde una tercera página, pero no es posible, porque la variable fue anulada en la página anterior. Se elimina la sesión:

finalsesion.php

```
<?php
 session_start();

 echo '<h3>finalsesion.php:</h3>
       <p>El contenido de $_SESSION[\'var_sesion\'] es '
       .$_SESSION['var_sesion'].'<br />';

 session_destroy();
?>
```

Parte VI

Interactuar con el sistema de archivos

Contenidos

Introducción a la carga de archivos

upload.html

```
<html>
<head>
  <title>Administración - Subir archivos</title>
</head>
<body>
<h1>Subir nuevo archivo de texto</h1>
<form enctype="multipart/form-data" action="upload.php" method=post>
  <input type="hidden" name="MAX_FILE_SIZE" value="1000000">
  Subir este archivo: <input name="userfile" type="file">
  <input type="submit" value="Send File">
</form>
</body>
</html>
```

- El formulario puede utilizar PUT o POST.
- Debe incluir el atributo `enctype="multipart/form-data"`.
- También debe incluir un campo (podemos ponerlo como oculto) para establecer el tamaño máximo de fichero que se puede subir: `<input type='hidden' name='MAX_FILE_SIZE' value='1000000'>`.
- Se utiliza una entrada de tipo `file` para poder introducir la ruta y nombre del archivo.

Introducción a la carga de archivos

- Al cargar un archivo, se almacena en el directorio temporal predeterminado del servidor web (Por lo general, `/tmp`).
- El archivo se almacena con un nombre temporal. Si no se cambia el nombre o se mueve de ubicación antes de terminar la ejecución de la secuencia de comandos PHP, el fichero se eliminará.
- La información necesaria para trabajar sobre el fichero se encuentran en la matriz superglobal `$_FILES`. Si el elemento `file` del formulario se llama `userfile`:
  - `$_FILES['userfile']['tmp_name']` contiene el lugar en el que el archivo se ha almacenado temporalmente.
  - `$_FILES['userfile']['name']` es el nombre real del archivo.
  - `$_FILES['userfile']['size']` es el tamaño del archivo en bytes.
  - `$_FILES['userfile']['type']` es el tipo MIME del archivo (ej: `text/plain` o `image/gif`).
  - `$_FILES['userfile']['error']` devuelve cualquier error asociado a la carga del archivo.

Introducción a la carga de archivos

upload.php (I)

```
<html>
<head>
  <title>Enviando...</title>
</head>
<body>
<h1>Enviando archivo...</h1>
<?php

  if ($_FILES['userfile']['error'] > 0)
     {
    echo 'Problema: ';
    switch ($_FILES['userfile']['error'])
    {
      case 1:  echo 'Excedido tamaño máximo de subida';  break;
      case 2:  echo 'Excedido tamaño máximo de fichero';  break;
      case 3:  echo 'Archivo subido parcialmente';  break;
      case 4:  echo 'No se ha podido subir el fichero';  break;
    }
    exit;
  }

  // Tiene el fichero un tipo MIME correcto?
  if ($_FILES['userfile']['type'] != 'text/plain')
  {
    echo 'Problema: El archivo no está en texto plano';
    exit;
  }
```

Introducción a la carga de archivos

upload.php (II)

```
// Copiamos el fichero a la ubicación deseada
$upfile = '/home/usuario/uploads/'.$_FILES['userfile']['name'];

if (is_uploaded_file($_FILES['userfile']['tmp_name']))
{
   if (!move_uploaded_file($_FILES['userfile']['tmp_name'], $upfile))
   {
      echo 'Problema: No se pudo mover el fichero al directorio de destino';
      exit;
   }
}
else
{
  echo 'Problema: Posible ataque al archivo enviado. Fichero: ';
  echo $_FILES['userfile']['name'];
  exit;
}

echo 'Fichero subido con éxito<br><br>';

// reformateo de los contenidos del fichero
$fp = fopen($upfile, 'r');
$contents = fread ($fp, filesize ($upfile));
fclose ($fp);
```

Introducción a la carga de archivos

upload.php (III)

```
    $contents = strip_tags($contents);
    $fp = fopen($upfile, 'w');
    fwrite($fp, $contents);
    fclose($fp);

    // Mostramos lo que se ha enviado
    echo 'Vista previa del
          fichero enviado:<br><hr>';
    echo $contents;
    echo '<br><hr>';

?>
</body>
</html>
```

- En el ejemplo anterior, nos aseguramos de que se sube un fichero de texto, es decir, de tipo `text/plain`.
- `strip_tags` elimina de un *string* cualquier etiqueta HTML o PHP.

Contenidos

Leer desde directorios

browsedir.php

```
<html> <head> <title>Examinar Directorios</title>  </head>
<body>
<h1>Examinando</h1>
<?php
  $current_dir = '/var/www';
  $dir = opendir($current_dir);

  echo "<p>El directorio actual es $current_dir</p>";
  echo '<p>Contenido del directorio:</p><ul>';
  while ($file = readdir($dir))
  {
      echo "<li>$file</li>";
  }
  echo '</ul>';
  closedir($dir);
?>
</body> </html>
```

- `opendir()` abre un directorio para su lectura. Devuelve un indicador de directorio.
- Con `readdir()` leemos un nombre de archivo. Con cada invocación, lee el fichero siguiente. Cuando no quedan archivos que leer, devuelve `false`.
- `closedir()` cierra el directorio.
- Con `rewindir()` podemos restablecer la lectura de los nombres de archivo al principio del directorio.

Obtener información sobre el directorio actual

- Si añadimos las siguientes líneas al fichero `upload.php`:

```
$tmp = $_FILES['userfile']['tmp_name'];
echo "<p>tmp: $tmp";
echo "<p>dirname de $tmp: ". dirname($tmp);
echo "<p>basename de $tmp: ". basename($tmp);
```

- Si `$tmp` vale, por ejemplo, `/tmp/phpHA1E8H`:
- `dirname` nos devolverá `/tmp`.
- `basename` nos devolverá `phpHA1E8H`.

Crear y eliminar directorios

- `mkdir(ruta,permisos)` permite crear directorios con los permisos especificados. Los permisos obtenidos son el resultado de realizar la operación AND sobre la inversa del `umask` actual con los permisos solicitados. Para evitarnos problemas:

```
$oldumask = umask(0);
mkdir("/tmp/testing",0777);
umask($oldumask);
```

- `rmdir(ruta)` elimina un directorio. Dicho directorio deberá estar vacío.

```
rmdir("/tmp/testing");
```

Contenidos

Obtener información sobre archivos

- Modificamos parte de nuestro programa `browsedir.php` para que cada fichero mostrado incluya un enlace:

browsedir2.php

```
<html>
<head>
  <title>Examinar Directorios</title>
</head>
<body>
<h1>Examinando</h1>
<?php
  $current_dir = '/var/www';
  $dir = opendir($current_dir);

  echo "<p>El directorio actual es $current_dir</p>";
  echo '<p>Contenido del directorio:</p><ul>';
  while ($file = readdir($dir))
  {
      echo '<li><a href="filedetails.php?file='.$file.'">'.$file.'</a></li>';
  }
  echo '</ul>';
  closedir($dir);
?>
</body>
</html>
```

Obtener información sobre archivos

- Mostramos información sobre el archivo seleccionado en el código anterior:

filedetails.php (I)

```
<html>
<head>
  <title>Detalles del archivo</title>
</head>
<body>
<?php
  $current_dir = '/var/www/';

  $file = $_GET['file'];
  $file = basename($file);  // Por seguridad, nos aseguramos de
                            // quedarnos sólo con el nombre del archivo
  echo '<h1>Detalles del archivo: '.$file.'</h1>';
  $file = $current_dir.$file;

  echo '<h2>Fecha del archivo</h2>';
  echo 'Último acceso: '.date('j F Y H:i', fileatime($file)).'<br>';
  echo 'Última modificación: '.date('j F Y H:i', filemtime($file)).'<br>';

  $user = posix_getpwuid(fileowner($file));
  echo 'Propietario del archivo: '.$user['name'].'<br>';

  $group = posix_getgrgid(filegroup($file));
  echo 'Grupo del archivo: '.$group['name'].'<br>';
```

Obtener información sobre archivos

filedetails.php (II)

```
  echo 'Permisos del archivo: '.decoct(fileperms($file)).'<br>';
  echo 'Tipo de archivo: '.filetype($file).'<br>';
  echo 'Tamaño del archivo: '.filesize($file).' bytes<br>';
  echo '<h2>Tests</h2>';

  echo 'is_dir: '.(is_dir($file)? 'true' : 'false').'<br>';
  echo 'is_executable: '.(is_executable($file)? 'true' : 'false').'<br>';
  echo 'is_file: '.(is_file($file)? 'true' : 'false').'<br>';
  echo 'is_link: '.(is_link($file)? 'true' : 'false').'<br>';
  echo 'is_readable: '.(is_readable($file)? 'true' : 'false').'<br>';
  echo 'is_writable: '.(is_writable($file)? 'true' : 'false').'<br>';
?>
</body>
</html>
```

Operaciones sobre archivos

- PHP dispone de funciones similares a las de UNIX para trabajar sobre archivos o modificar sus propiedades:

```
chmod('fichero.txt',0644);
```

- `chmod()` Cambia los permisos del fichero.

```
touch('fichero.txt');
```

- Con `touch()`, si el fichero no existe, se crea. Si ya existe, se actualiza su hora de modificación.
- Para eliminar archivos:

```
unlink($nombre_archivo);
system("del fichero.txt");  //versiones antiguas de Windows
```

- Para copiar y mover archivos:

```
copy($ruta_origen,$ruta_destino);
rename($archivo_antiguo,$archivo_nuevo);
```

Funciones de ejecución de comandos en el servidor

1. `exec()`: Se le pasa como parámetro el comando a ejecutar. En `$resultado` nos devuelve una matriz de *strings* que contiene las líneas de la salida resultante de la ejecución del comando.

```
exec("ls -la",$resultado);
```

2. `passthru()`: Repite el resultado de la salida en el navegador.

```
passthru("cat $file");
```

3. `system()`: Repite el resultado de la salida en el navegador, pero intenta eliminar el resultado tras cada línea. Devuelve la última línea del resultado.

```
$resultado = system("cat $file");
```

4. Apóstrofes invertidos: Operadores de ejecución. El resultado se puede almacenar en una cadena.

```
$resultado = `ls -al`;
```

Interactuar con el entorno

- `getenv()` permite recuperar variables de entorno (podemos obtener la lista con `phpinfo()`). Por ejemplo, para obtener la ruta desde la que el usuario llegó hasta la página actual:

```
$ruta = getenv("HTTP_REFERER");
echo $ruta.'<br>';
```

- `putenv()` Permite establecer una variable de entorno.

```
putenv("PATH=/usr/local/bin");
```

Parte VII

Autentificación de usuarios

Parte VIII

Manipulación de cadenas

Parte IX

Control de sesiones

Parte X

Orientación a Objetos en PHP

Parte XI

Gestión de errores

Parte XII

Trabajar con fechas

Parte XIII

Poyecto de Fin de Módulo

Técnico en creación de portales web con software libre

Módulo 3: Creación de páginas web dinámicas mediante PHP

Juan Ángel Lorenzo del Castillo
juanangel@dec.usc.es

Departamento de Electrónica y Computación
Universidad de Santiago de Compostela