



Ver índice

# Páginas WEB dinámicas



## Tipos de páginas web

Una sencilla clasificación de los tipos de páginas web podría ser esta:

Páginas estáticas  
Páginas dinámicas

### Páginas estáticas

Diremos que una página es estática cuando sus contenidos **no** pueden ser modificados –ni desde el *servidor* que la aloja (ordenador remoto) ni desde el *cliente* (navegador)– mediante ninguna intervención del usuario ni tampoco a través de ningún programa.

### Páginas dinámicas

Llamaremos dinámicas a las páginas cuyos contenidos **sí** pueden ser modificados –de forma automática o mediante la intervención de un usuario– bien sea desde el *cliente* y/o desde el *servidor*.

Para que esas modificaciones puedan producirse es necesario que *algo o alguien* especifique: *qué, cómo, cuándo, dónde y de qué forma* deben realizarse, y que exista otro *algo o alguien* capaz de acceder, interpretar y ejecutar tales instrucciones en el momento preciso.

Igual que ocurre en la vida cotidiana, las *especificaciones* y las *instrucciones* requieren: un *lenguaje* para definirlas; un *soporte* para almacenarlas y un *intérprete* capaz de *ejecutarlas*.

Somos capaces de entender unas instrucciones escritas en castellano pero si estuvieran escritas en *búlgaro* las cosas seguramente serían bastante distintas, y, por supuesto, a *un búlgar@* le pasaría justamente lo contrario.

Igual ocurre con los programas *intérpretes* de los lenguajes de script. Ellos también requieren órdenes escritas en su *propio idioma*.

### Scripts

Se llama *script* a un conjunto de *instrucciones* escritas en un *lenguaje* determinado que van **incrustadas** dentro de una *página WEB* de modo que su *intérprete* pueda acceder a ellas en el momento en el que se requiera su *ejecución*.

Cuando se **incrustan** *scripts* en

## Un ejemplo de página estática

Cualquier usuario que acceda a ésta –ya sea en *modo local*, o a través de un *servidor remoto*– visualizará siempre la misma fecha: *11 de febrero de 2009*.

```
<HTML>
<HEAD>
</HEAD>
<BODY>
Hoy es 4-5-2009 y son las 14:23:57 horas
</BODY>
</HTML>
```

**ejemplo1.html**

## Una par de páginas dinámicas

Si pulsas en el enlace del primero de estos dos ejemplos verás que la fecha que aparece en la página es la *fecha actual* de tu sistema, y además, cada vez que pulses el botón *Actualizar* de tu navegador podrás comprobar que se *actualiza* la hora.

Una *intervención* del usuario *modifica los contenidos*.

```
<HTML>
<HEAD>
<script language="JavaScript">
var son= new Date();
var fecha=son.getDate()+" - "+(son.getMonth()+1)+" - "+son.getFullYear();
var hora=son.getHours()+":"+son.getMinutes()+":"+son.getSeconds();
document.write('Hoy es '+fecha+' y son las '+hora+' horas');
</script>
</HEAD>
<BODY>
</BODY>
</HTML>
```

**ejemplo2.html**

En este otro ejemplo la modificación de los contenidos *no requiere intervención alguna* por parte del usuario. Cada *5 segundos* (fíjate donde dice *var frecuencia=5000*). *Cinco mil* es el período de actualización, expresado en *milisegundos*) se *rescribirán* de forma automática *la fecha y la hora*. Tenemos un *cronómetro* automático.

```
<HTML>
<HEAD>
<script language="JavaScript">
var reloj=0;
var frecuencia=5000;
function actualiza(){
var son= new Date();
var fecha=son.getDate()+" - "+(son.getMonth()+1)+" - "+son.getFullYear();
var hora=son.getHours()+":"+son.getMinutes()+":"+son.getSeconds();
var escribe='Hoy es '+fecha+' y son las '+hora+' horas';
var situa=document.getElementById('capa0');
situa.innerHTML=escribe;
reloj=setTimeout("actualiza()",frecuencia);
}
</script>
</HEAD>
<BODY onLoad="actualiza()";>
```

una página WEB empiezan a convivir en un mismo documento informaciones destinadas a distintos intérpretes.

Por una parte, el código HTML que ha de ser interpretado por el navegador, y por la otra, los scripts que han de ser ejecutados –dependiendo del lenguaje en el que hayan sido escritos– por su intérprete correspondiente.

La manera de diferenciar los contenidos es delimitar los scripts marcando su comienzo con una etiqueta de apertura `<script>` y señalando el final con una etiqueta de cierre `</script>`.

Lo que no está contenido entre esas etiquetas se considerará código HTML.

La posibilidad de insertar en un mismo documento scripts desarrollados en distintos lenguajes obliga a especificar cuál se ha utilizado en cada caso, para que en el momento en el que vayan a ser ejecutados se invoque el intérprete adecuado.

Para ello, dentro de la propia etiqueta de apertura (`<script>`) se inserta una referencia al tipo de lenguaje con esta sintaxis:

`language="nombre"`

Por ejemplo:

```
<script language="PHP">
.....
..... instrucciones ..
.....
</script>
```

indicaría que las instrucciones están escritas con la sintaxis de PHP.

Por el contrario, en este otro supuesto:

```
<script language="JavaScript">
.....
..... instrucciones ..
.....
</script>
```

estaría señalando que en las instrucciones contenidas en el script se ha utilizado sintaxis de JavaScript.

Para el caso concreto de PHP, existe una sintaxis alternativa, mucho más cómoda y que es la que se usa habitualmente.

Es la siguiente:

```
<?
.....
.....instrucciones..
.....
?>
```

```
<div class="capa0">
</div>
</BODY>
</HTML>
```

ejemplo3.html

### Ejercicio nº 1

Abre tu *block de notas* y escribe el código fuente del ejemplo nº 3 prestando especial atención a la transcripción de las mayúsculas y las minúsculas (JavaScript, igual que PHP, diferencia entre unas y otras) y también a las comillas y a los punto y coma que aparecen al final de cada línea.

Guarda el documento con el nombre `ejercicio1.html`, luego ábrélo con el navegador y comprueba el funcionamiento del cronómetro.

Una vez que hayas comprobado que funciona, prueba a sustituir el **5000** por otros valores numéricos y comprueba como se modifica la *frecuencia* del cronómetro.

## Servidores y clientes

Es frecuente observar en la calle que son muchas las personas que cuando se refieren a los servidores lo hacen con sí se trata de máquinas complejas, misteriosas, lejanas y enormes que, bajo esa aureola de *cripticismo*, parecen totalmente distintas al ordenador que usamos habitualmente. ¡Nada más lejos de la realidad!

Intentaremos aclarar algunos conceptos con ejemplos cotidianos. Pensemos en esos ordenadores remotos (también llamados **host**) como si se tratara de uno esos sitios desde los que se sirven comidas a domicilio.

Quizá lo primero en lo que se te ocurra pensar sea en una pizza, no porque desconozcas que también es posible comprar otras cosas sino por la popularidad de ese tipo de servicio. Algo similar ocurre con los **host**. La frecuencia con la que accedemos a ellos en demanda de páginas web hace que tendamos a identificarlos con ellas, pero... también los **host** ofrecen –o pueden ofrecer– más servicios. Sigamos con las comidas a domicilio.

Cada una de esas empresas puede atender peticiones de uno solo o de varios servicios distintos (pizzas, helados, o platos regionales, por citar algunos ejemplos), pero la oferta de cada uno de esos servicios requiere una infraestructura adecuada a cada caso. La oferta de pizzas exigirá disponer de un horno, y la de helados necesitará de una instalación frigorífica.

Pues bien, algo muy similar ocurre con los **host**. También éstos pueden ofrecer uno o varios servicios (páginas web, correo electrónico, transferencias FTP, noticias, etcétera) y también es necesario que cada servicio disponga de su propia infraestructura, que en este caso sería un programa distinto (software de servidor) para cada uno de ellos.

Como puedes ver, no basta con hablar de servidores. Es necesario especificar también qué es lo que sirven, para lo cual habría que decir: servidor de páginas web, servidor de correo, etcétera y tener presente que –aunque convivan en el mismo host– cada uno de ellos requiere su propio software y su propia configuración.

Resumiendo, cuando en lenguaje coloquial hablamos de un **servidor** estamos aludiendo un **host** (ordenador remoto) –el tamaño y la lejanía carecen de importancia– provisto de **programas** (software de servidor) que, cuando está accesible (conectado a Internet) y con el software activo (servidor en funcionamiento) > es capaz de atender peticiones y devolver a los **clientes** los documentos solicitados, o un mensaje de error, en el caso de que no estuvieran disponibles.

Veamos un ejemplo de como se desarrolla ese proceso de petición-respuesta.

Para leer el correo electrónico necesitas disponer de un **programa** –supongamos que es Outlook Express– instalado en tu ordenador y hacer, a través de él, una petición a un ordenador remoto (**host**). Si quisieras visualizar páginas web tendrías que utilizar un programa distinto –Internet Explorer, por ejemplo– capaz de realizar esta otra tarea.

Al programa que utilizamos para realizar cada petición le llamaremos **cliente**.

¿Qué es una petición?

Una petición es un conjunto de datos que un **cliente** (recuerda que el cliente siempre es uno de los

<? hará la misma función que <script language="PHP"> y ?> será equivalente a </script>.

## Lenguajes

Hay múltiples posibilidades en cuanto a lenguajes de *script*. Pero antes de hacer mención a algunos de ellos es conveniente hacer una clasificación previa.

Los lenguajes de *script* pueden clasificarse en dos tipos:

- Del lado del cliente
- Del lado del servidor

En la columna derecha hemos comentado algunos conceptos sobre *servidores* y *clientes* que pueden ser útiles a la hora de analizar las diferencias entre estos dos tipos de lenguaje.

### Lenguajes del lado del cliente

Diremos que un lenguaje es *del lado del cliente* cuando el *intérprete* que ha de ejecutar sus *scripts* es **accesible** desde éste –el *cliente*– sin que sea necesario hacer ninguna petición al *servidor*.

Seguramente te ha ocurrido alguna vez que al intentar acceder a una página web ha aparecido un mensaje diciendo que *la correcta visualización de la página requiere un plug-in determinado*, y que, a la vez, se te haya ofrecido la posibilidad de *descargarlo* en ese momento.

Eso ocurre porque cuando el *navegador* –que en el caso de las páginas web es el *cliente*– trata de interpretar la página, encuentra **incrustado** en ella *algo* (un fichero de sonido, una animación Flash, etcétera) que –de forma muy similar a lo que ocurre con los *scripts*– requiere un *intérprete adecuado* del que no dispone en ese momento.

Cuando los *scripts* contenidos en un documento son de este tipo, el *servidor* lo entrega al *cliente* si efectuar ningún tipo de modificación.

### Lenguajes del lado del servidor

Un lenguaje es *del lado del servidor* cuando la ejecución de sus *scripts* se efectúa, por *instancia* de este –el *servidor*–, **antes de dar respuesta a la petición**, de manera que el *cliente* no recibe el documento original sino el resultante de esa interpretación previa.

Cuando se usan estos tipos de lenguaje el *cliente* recibe un documento en el que cada *script*

programas instalados en tu ordenador) envía a través de Internet solicitando una respuesta determinada por parte de un ordenador remoto.

### ¿Qué contendría esa petición?

Cada tipo de **petición** tendrá contenidos distintos. Por ejemplo, cuando se trata de *leer mensajes de correo*, la petición realizada por el **cliente** (*Outlook Express*) contendría, entre otros, muchos de los datos de la configuración de la cuenta, tales como: el **protocolo** (forma de comunicación) –en el caso del correo lo habitual sería el protocolo **POP** (*Post Office Protocol*)–, el nombre de **host** donde está alojado el **buzón** (servidor POP ó servidor de correo entrante), el nombre de la cuenta, la contraseña de acceso, y algunas otras informaciones relativas a la gestión de esa cuenta tales como si deben conservarse o no los mensajes en el servidor, etcétera.

### ¿Qué ocurre con esa petición?

Cualquier **petición** pasa en primera instancia por un **servidor de nombres de dominio** (Domain Name Server) **DNS**, una especie de guía telefónica que contiene los nombres de los servidores y las direcciones IP a través de las cuales están conectados a Internet. Podría decirnos –los datos son ficticios– que **olmo.cnice.mecd.es** es el nombre de un **host** que está conectado a Internet a través de la dirección IP **111.112.113.114**

Una vez *resuelta* esa petición por el **servidor DNS** (*direcciónamiento de la petición a la IP correspondiente*) se comprobará si esa IP está activa (si efectivamente hay un ordenador conectado a través de ella) y, en caso de estarlo, se determinará si ese ordenador al que estamos accediendo es **capaz de atender la petición**.

### ¿Qué tiene que ocurrir para que pueda atenderse una petición?

Es necesario que el **ordenador remoto** tenga **instalado y funcionando el software de servidor adecuado al protocolo de nuestra petición**. Ello quiere decir –siguiendo con el ejemplo– que el ordenador remoto debe tener instalado y funcionando un software específico de **servidor de correo** capaz de interpretar el *protocolo POP3* especificado en la petición.

### ¡Cuidado!

**El ordenador remoto debe tener instalado y funcionando el software adecuado a cada tipo de petición (servicio) que deba atender.**

No basta con decir **servidor**, es preciso conocer los **servicios** que presta y es factible que un mismo ordenador preste –simultáneamente– varios **servicios**, siempre que tenga **instalado y activo** el software específico para cada uno de esos **servicios**.

Cuando el **ordenador remoto acepta la petición** el **software de servidor y/o las aplicaciones del lado del servidor** (software instalado en el ordenador remoto y vinculado con el software de servidor) **resuelven la petición** (comprobar que el *nombre de la cuenta* y la *contraseña* son correctas, comprobar si existen mensajes, borrarlos del *buzón* si así lo especifica la petición, etc.) y **devuelven al cliente** (recuerda que el *cliente* era nuestro *Outlook Express*) la información requerida.

Solo falta que una vez recibida la **respuesta** *Outlook Express (cliente)* *interprete* la información recibida y nos permita visualizar o imprimir el contenido de los mensajes *descargados* del servidor.

## Servidor y cliente en una misma máquina

Hasta ahora –al referirnos a servidores y clientes– hemos hecho alusión a dos *máquinas*: nuestro propio ordenador (*ordenador local*) en el que estarán instaladas las aplicaciones **cliente** y un **ordenador remoto** en el que se alojarían las aplicaciones de **servidor**. Eso es lo más *habitual*, pero *no es la única posibilidad*.

Dado que **servidor** y **cliente** son únicamente **aplicaciones** es perfectamente posible que ambas convivan dentro de la misma *máquina*.

La diferencia sustancial sería que ahora no es necesario el servidor de DNS para buscar la dirección IP. Utilizaríamos una IP (habitualmente la **127.0.0.1**) **reservada** para estos casos –preestablecida en la configuración del servidor– y a través de ella se canalizarían las *peticiones a nuestro propio servidor*. Ya hablaremos más adelante de esta IP.

## Esquemas de diferentes *peticiones* de páginas WEB

sustituido por los resultados de su ejecución.

Esto es algo a tener muy en cuenta, porque, en este caso, los usuarios no tendrán la posibilidad de visualizar el código fuente, mientras que cuando se trata de lenguajes del lado del cliente siempre es posible visualizar los scripts, bien sea de forma directa –mirando el código fuente de la página recibida– o leyendo el contenido de ficheros externos –vinculados a ella– que son bastante fáciles de encontrar en la caché del navegador.

La utilización de este tipo de scripts requiere que el intérprete del lenguaje sea accesible –esté del lado– desde el propio servidor.

#### ¿Cómo resuelve sus dudas el servidor?

Dado que en unos casos el servidor debe entregar el documento original –páginas estáticas o páginas dinámicas en las que se usan lenguajes del lado del cliente– mientras que en otros casos –páginas dinámicas usando lenguajes del lado del servidor– tiene que devolver el resultado de la ejecución de los scripts, es razonable que te preguntes: ¿cómo sabe el servidor lo que debe hacer en cada caso?

La respuesta es simple. Eso hay que decirselo. Y se le dice de una forma bastante simple. Se indica al poner la extensión al documento.

Si en la petición se alude a un documento con extensión .htm o .html el servidor entenderá que esa página no requiere la intervención previa de ningún intérprete de su lado y entre- gará la página tal cual.

Si en esa petición se aludiera a una extensión distinta –.php, por ejemplo– el servidor entendería que antes de servir la página debe leerla y requerir al intérprete de PHP que ejecute los scripts desarrollados en ese lenguaje (en caso de que los contuviera) y devolvería al cliente el documento que resultara de las eventuales ejecuciones de tales scripts.

#### Algunos lenguajes con nombre y apellidos

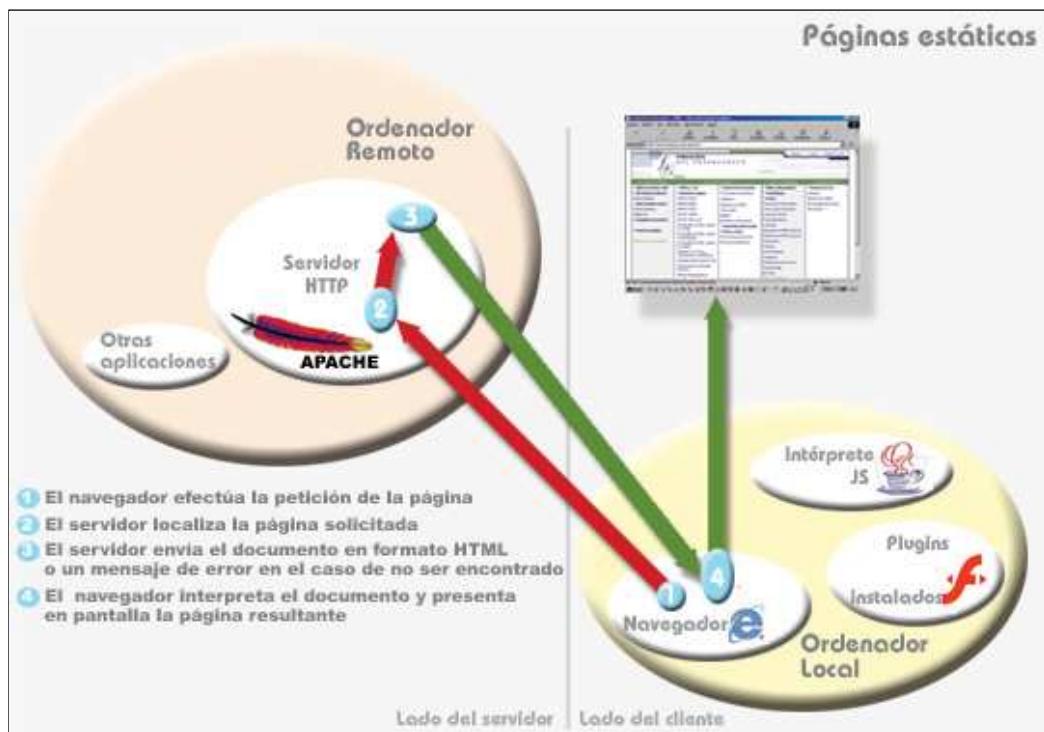
Sin pretender hacer una enumeración exhaustiva, los lenguajes de script más populares son los siguientes:

#### Del lado del cliente

- DHTML
- JavaScript
- VBScript

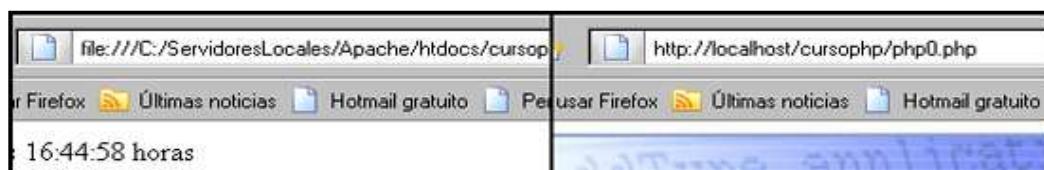
Intentaremos resumir de forma esquemática los procesos de algunos de los diferentes tipos de peticiones de páginas WEB.

Aquí tenemos la más sencilla de ellas:



Si observas con detenimiento el esquema de la parte superior es posible que encuentres algo que no te cuadre... porque en el esquema hay un servidor que parece imprescindible para atender las peticiones y sin embargo tú –sin tener instalado ningún servidor– eres capaz de visualizar tus propias páginas web sin más hacer un doble click sobre su ícono.

Eso es cierto, pero fíjate en las dos direcciones que aparecen en esta otra imagen.



La de la izquierda –consecuencia de haber hecho doble click sobre el ícono del documento– contiene como dirección una ruta (el path que conduce hasta el documento) mientras que en la de la derecha aparece el sintagma http al principio de la dirección.

En el primer caso no hemos hecho ninguna petición de página web sino que hemos abierto un documento cuya extensión (html) está asociada con Internet Explorer en nuestra configuración de Windows. El proceso ha sido exactamente el mismo que si hubiéramos hecho doble click sobre el ícono de un documento con extensión txt, con la única salvedad de que en este último caso se habría abierto el block de notas (por la asociación de extensiones y aplicaciones en la configuración de Windows).

En el segundo caso las cosas son distintas. Se incluye el sintagma http –acrónimo de HiperText Transfer Protocol– para indicar que ese es el protocolo que debe ser utilizado y que será preciso que el servidor que reciba la petición sea capaz de interpretarlo. Por eso a los servidores que alojan páginas web se les suele llamar servidores HTTP y se les requiere que soporten este protocolo.

Siguiendo con los esquemas, he aquí el correspondiente a una petición de página en la que hay incrustados scripts escritos en lenguaje del lado del cliente:

**DHTML** no es exactamente un lenguaje de programación. Se trata más bien de una serie de capacidades que se han ido añadiendo a los navegadores **modernos** mediante las cuales las páginas pueden contener **hojas de estilo** y/o organizarse en capas susceptibles de ser redimensionadas, modificadas, desplazadas y/o ocultadas.

**JavaScript** es uno de los lenguajes más populares. Cada navegador incluye su propio intérprete y es frecuente que los resultados de visualización sean *algo* distintos según el navegador y la versión que se utilice.

Parece ser que las versiones más recientes de los distintos navegadores se aproximan a un estándar –**ECMA Script-262**– que ha sido desarrollado por la **ECMA** (Asociación Europea de Normalización de Sistemas de Información y Comunicación), lo que hace suponer que en un futuro muy próximo todos los navegadores se ajustarán a esa especificación y que, con ello, las páginas web ya se visualizarán de forma idéntica en todos ellos.

**VBScript** es un lenguaje de *script* derivado de *VisualBasic* y diseñado específicamente para los navegadores de *Microsoft*.

#### Del lado del servidor

Los más populares de este tipo son:

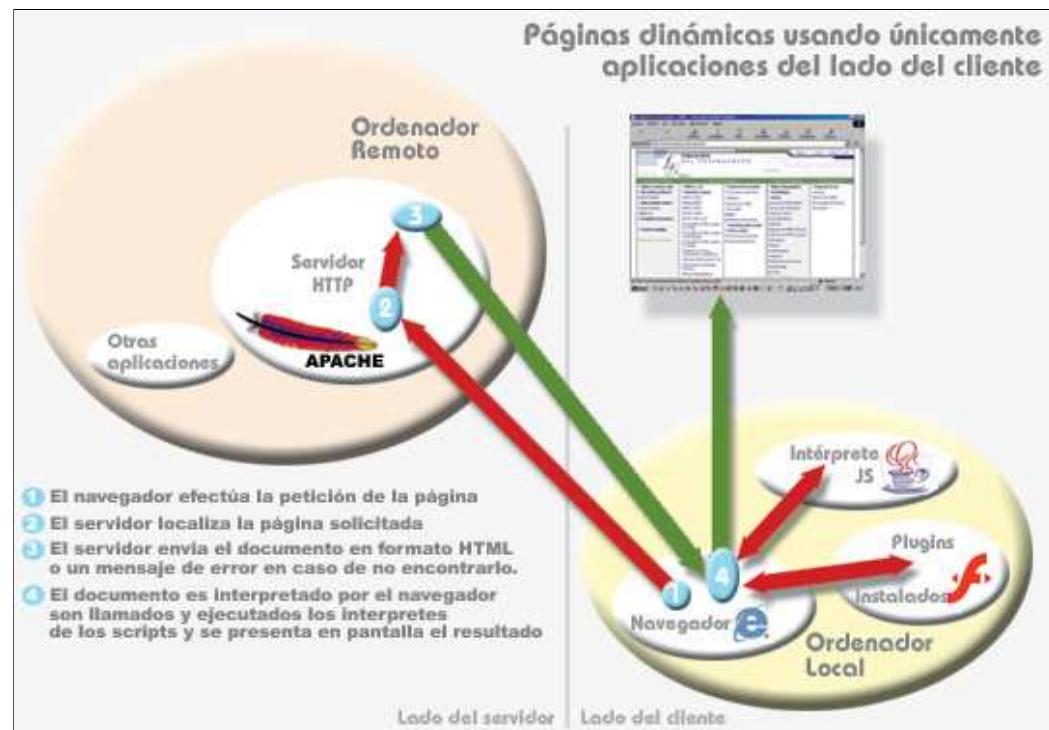
- PHP
- ASP
- Perl
- JSP

Cada uno de ellos tiene sus propias peculiaridades. Pero dado que aquí tratamos sobre **PHP** quizá sea conveniente –a modo de recordatorio– hacer algunas precisiones sobre los requisitos imprescindibles para trabajar con este lenguaje.

#### Requisitos para el uso del lenguaje PHP

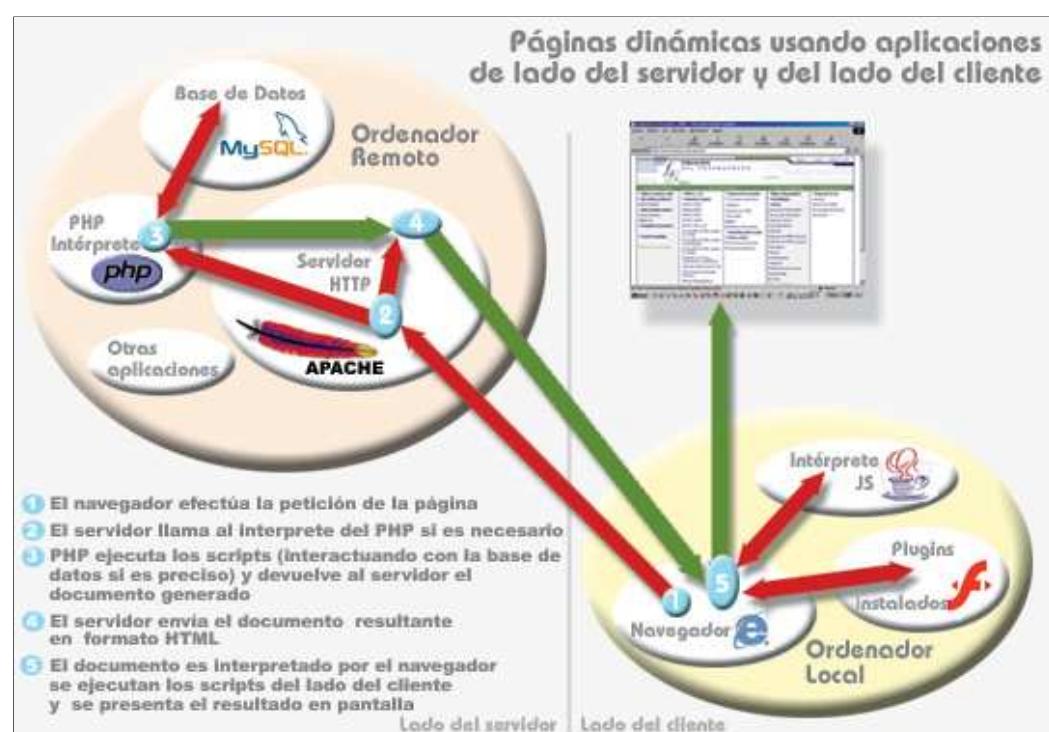
De acuerdo a lo comentado en los párrafos anteriores y en los esquemas que tenemos a la derecha, el uso del lenguaje PHP requiere tener *instalado y configurado*:

- Un **software de servidor** –configurado para interactuar con el intérprete de **PHP**– que soporte el protocolo **HTTP** y que en nuestro caso será el denominado servidor **Apache**.
- El intérprete de **PHP**.
- Un software de servidor de **bases de datos** capaz de ser gestionado mediante funciones propias de **PHP**.



Como puedes observar no requiere *nada distinto* a lo del supuesto anterior. La diferencia sería que en este caso se *harían llamadas* al *intérprete de JavaScript* –incluido en los navegadores, tal como comentamos al margen– y/o a eventuales *plugins* necesarios para interpretar otros tipos de *script*.

Y por último, el esquema más complejo: un ejemplo de *convivencia* en un mismo documento de varios *scripts* y varios tipos de lenguaje.



Aquí ya es preciso que, además de un servidor capaz de soportar el protocolo **HTTP**, esté instalado –del lado del servidor– un intérprete PHP, un **servidor de bases de datos MySQL** y que, además, estén configurados de modo que puedan interactuar entre ellos.

El lenguaje **PHP** dispone de funciones que le permiten acceder a muy diversos tipos de **servidores de bases de datos** pudiendo: **crear, consultar, borrar y modificar** tanto bases de datos como tablas y registros de las mismas.

Nosotros vamos a utilizar **MySQL**, unos de los gestores más potentes y populares que existen en este momento.

Utilizaremos el servidor de bases de datos conocido como **MySQL**.

En las páginas siguientes trataremos sobre la forma de realizar los procesos de instalación y configuración de estas aplicaciones.

## Diferentes servicios de *hosting*

Cuando empezamos a trabajar con páginas dinámicas elaboradas mediante scripts de **PHP** (lenguaje del lado del servidor) suele surgirnos la necesidad –o simplemente el deseo– de *publicarlas* en algún *espacio de alojamiento (hosting)*, sea *gratuito* o *de pago*.

Si queremos publicar páginas en las que utilicemos **PHP** y bases de datos **MySQL** habremos de buscar un **hosting** que, aparte de espacio de alojamiento, nos ofrezca *estos dos servicios*. Además, antes de elegir uno deberíamos informarnos sobre *la funcionalidad* que nos ofrece, ya que es importante conocer no sólo las versiones de PHP y MySQL de que dispone sino también *las restricciones que puedan existir para su uso* (bastante frecuentes y por razones de seguridad en la mayoría de los casos).

Si decides publicar tus páginas no te precipites en la elección de un **hosting**. A lo largo de este curso te iremos dando algunas pautas que te permitirán hacer una elección acorde con tus necesidades.

---

Anterior



Índice



Siguiente





Ver índice

# Instalación del editor Php Coder Pro!



## Editor Php Coder Pro!

A lo largo del curso va a sernos de gran utilidad el uso un editor de textos que nos permita identificar los números de línea de nuestros *scripts* y que a la vez nos ayude –mediante resaltado de textos– a depurar la sintaxis.

Existen multitud de editores y podrás utilizar –a tu gusto– este o cualquier otro que uses habitualmente. El hecho de sugerirte este obedece a al hecho de que es *gratuito, bastante completo, consume pocos recursos* y se va actualizando periódicamente.

## Instalación

La instalación de este editor no presenta ninguna dificultad. Se instala –y se desinstala– de la forma habitual en que se realizan estos procesos en Windows.

Por compatibilidad con el proceso de instalación automática (al que aludiremos en páginas posteriores) hemos optado por crear el directorio **C:\ServidoresLocales** para instalar dentro de él todos los programas que vamos a utilizar en el curso.



Al hacer doble click sobre el ícono del programa

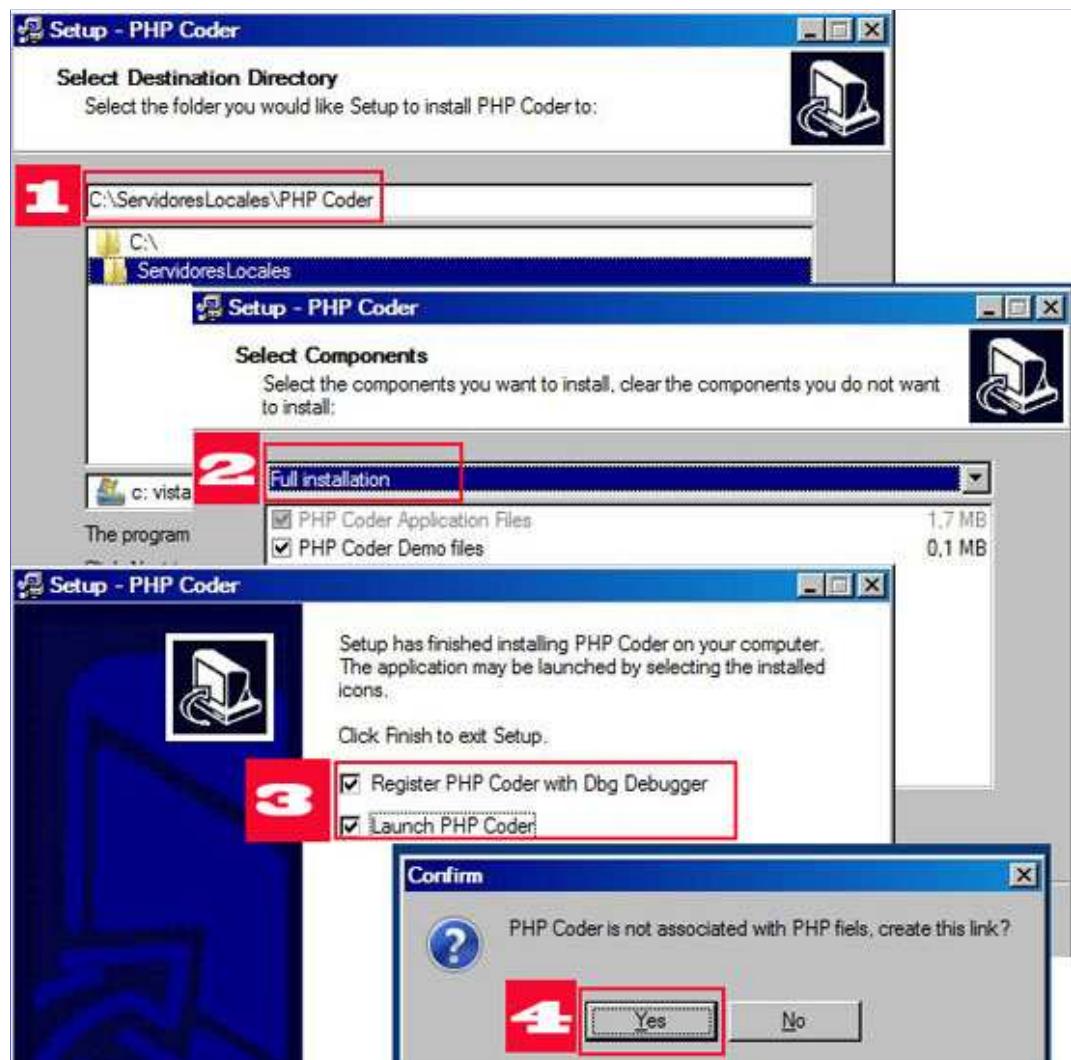


comienza el proceso de instalación cuya secuencia gráfica tenemos en la columna de la derecha.

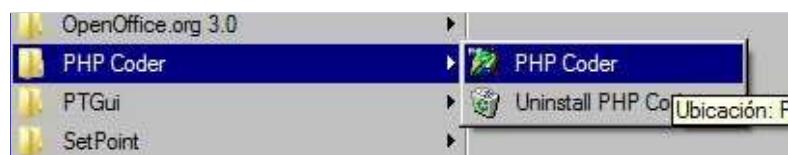
## El editor Php Coder Pro!

En el directorio Extras de este CD encontrarás un fichero llamado **phpcoder\_r2\_fp3.exe**. También podrás *descargar* este programa de instalación desde la dirección (el enlace de descarga está en la parte inferior de la página): <http://www.phpide.com/phpcoder.htm>.

La secuencia de la instalación (eliendo **ServidoresLocales** como directorio base para efectuar la instalación) es la que puedes ver en la imagen.



Al concluir la instalación se habrá creado un grupo de programas como el de la imagen inferior. Allí tienes el enlace para ejecutar el programa como para efectuar una eventual desinstalación



Anterior

Índice

Siguiente

## ¿Qué es un servidor WEB?

Podríamos definir un servidor WEB como una aplicación que permite acceder a los recursos contenidos en algunos de los directorios del ordenador que la alberga a usuarios remotos que realizan sus peticiones mediante el protocolo HTTP.

Por tanto, *instalar un servidor web* no es otra cosa que *instalar y configurar un programa* en una unidad o directorio de un ordenador cualquiera.

## ¿Qué es «Apache»?

Bajo este nombre suele hacerse referencia a **Apache Software Foundation**, organización norteamericana que se autodefine con el objetivo de «... facilitar ayuda organizativa, legal y financiera para los proyectos de desarrollo de software tipo Open Source (código abierto)».

Uno de los proyectos más populares de **Apache** es el desarrollo y suministro -de forma gratuita y libre- de un **software de servidor HTTP**, conocido también como el **servidor Apache**.

## La corta historia de «Apache»

*Apache Software Foundation* tiene su origen en febrero de 1995. Hasta ese momento el software más popular de servidores de HTTP era el desarrollado por Rob McCool, miembro del *Centro Nacional para Aplicaciones de Super computación* (NCSA), de la Universidad de Illinois.

El desarrollo de aquel primitivo software de servidor de NCSA (software de dominio público y código abierto) tuvo algunos problemas a mediados del año 1994.

Esta circunstancia obligó a que muchos de los *webmasters* que utilizaban aquella aplicación tuvieran que desarrollar sus propias extensiones y corregir de forma individual los fallos de funcionamiento de la aplicación original.

Un pequeño grupo de aquellos *webmasters* entró en contacto -vía e-mail- con el objetivo de coordinar y conjuntar sus trabajos de corrección y mejora de la aplicación original de NCSA.

## Proceso de instalación

En el CD-ROM que contiene los materiales de este curso hay un directorio llamado **Software**. En él tienes un fichero llamado **apache\_2.2.11-win32-x86-openssl-0.9.8i.msi**.

Al hacer *doble click* sobre su ícono deberá aparecer una pantalla como esta:



Si en vez de esta pantalla apareciera un mensaje de error, es muy probable que se deba a que no tienes instalada la aplicación **Windows Installer Redistributable** necesaria para ejecutar los ficheros con extensión **.msi**.

Las versiones modernas de Windows –a partir de Windows98– la tienen incorporada, pero si por cualquier circunstancia –usando Windows98– no dispones de ella, puedes descargarla [desde este enlace](#).

Al pulsar *Next* en la ventana anterior se llega a esta otra en la que es *preciso tener seleccionada* la opción *I accept the terms in the license agreement* para que se active el botón *Next* y poder continuar la instalación.



Fueron Brian Behlendorf y Cliff Skolnick quienes -a través de una lista de correo- coordinaron el trabajo y lograron establecer un espacio común para quienes participaran en el proyecto- en un ordenador instalado en California.

En febrero de 1995, ocho colaboradores del primitivo proyecto decidieron aunar sus esfuerzos de forma organizada y fundaron lo que fue conocido como *Grupo Apache*.

Usando como base el HTTPD 1.3 de NCSA y aplicando los patches desarrollados hasta ese momento, lanzaron la *primera versión oficial* (versión 0.6.2) del software de servidor de Apache en abril de 1995.

Aquella primera versión entró en un rápido e intenso proceso de mejoras y alcanzó una gran implantación como software de servidor –inicialmente era sólo para UNIX– para diferentes plataformas, una de las cuales es la versión para Windows cuya instalación te estamos explicando a la derecha, y que frente a otras alternativas de servidor local de HTTP tiene las ventajas de ser gratuita y muy eficaz.

## Manual o automático

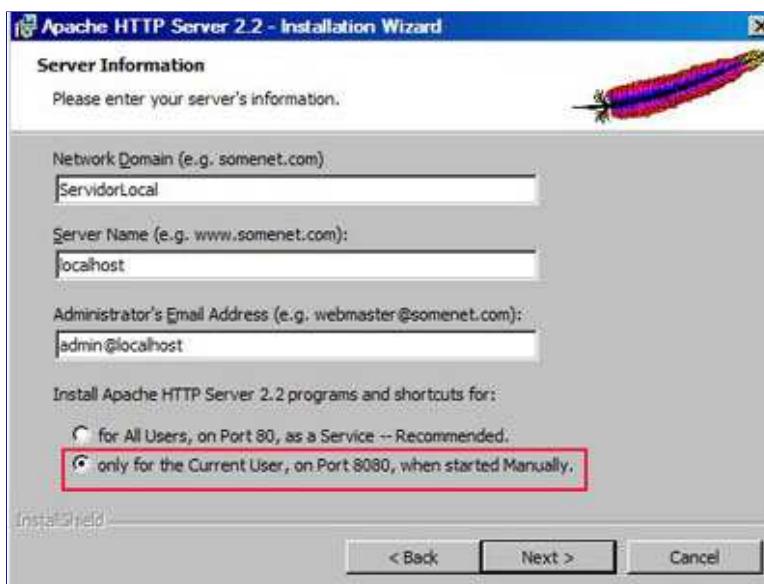
Al describir el proceso de instalación –aquí a la derecha– sugerímos elegir la opción *only for the Current user, on port 8080, when started manually* frente a la otra alternativa posible –la recomendada en la pantalla de instalación– *for All Users, on port 80, Run as a service*.

La instalación *recomendada* por Apache contempla el uso del servidor como *un servidor de red de área local* accesible desde el resto de los ordenadores de esa red. Es lógico que en esas circunstancias el servidor se configure con *autorranque* y que se inicie automáticamente en el momento en que se arranca el ordenador, con lo cual empezará a consumir a partir de ese instante –y en todo momento– *recursos del sistema*.

En nuestro caso, la situación es *especial*. Sólo vamos a utilizar el servidor para *probar* y *depurar* nuestros propios *scripts PHP* y en ello, seguramente vamos a ocupar sólo una pequeña parte del tiempo de uso del ordenador, por lo que parece ser más eficaz la opción de *arrancarlo* únicamente cuando sea necesario y, de esa forma, ahorrar esos recursos del sistema –que nos podrían ralentizar el uso de otros programas– cuando no necesitemos usar el servidor.

En cualquier caso, el proceso de instalación es idéntico para ambas opciones.

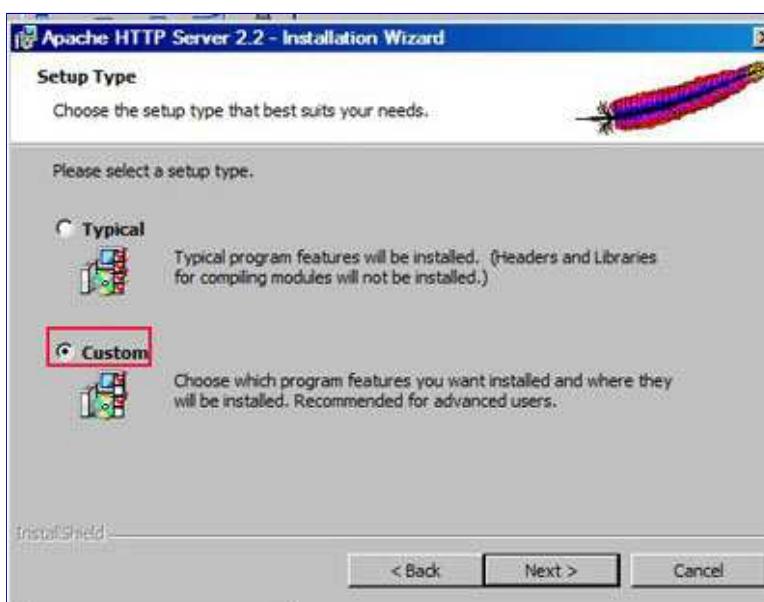
Al llegar a esta otra ventana –las anteriores solo requieren ir pulsando en el botón *Next*– es imprescindible llenar los campos *Network Domain*, *Server Name* y *Administrator's e-mail address*.



Como *Network Domain* y como *Administrator's e-mail address* puedes poner nombres cualesquier siempre que **no contengan espacios ni caracteres especiales** y en el caso de la dirección e-mail ésta debe contener el símbolo @. Como *Server Name* escribe *localhost*.

Por último, **elige la opción *only for the Current User, on port 8080, when started manually*** (al margen te comentamos el por qué de esa elección) y pulsa de nuevo *Next*.

En la pantalla que aparece inmediatamente después de la anterior se permite elegir el modo de instalación: *Typical* ó *Custom*. Elige la opción *Custom* y pulsa nuevamente sobre *Next*.



Te aparecerá esta nueva pantalla en la que el instalador nos indica el directorio por defecto en el que pretende instalar el *servidor Apache*.

## Directorio de instalación

Te hemos sugerido cambiar el directorio de instalación por razones de comodidad.

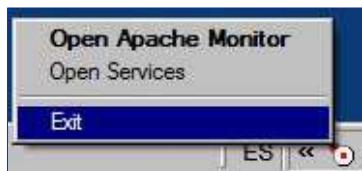
Pretendemos con ello agrupar todos los programas del curso en un único directorio (es la razón de definir **ServidoresLocales** para ese fin) y dentro de él asignar un nombre fácilmente identificable para cada uno de los programas (Apache, en este caso). Otra razón por la que te sugerimos utilizar esa estructura es porque hemos incluido junto con los materiales algunos ficheros de configuración y para realizar una instalación automática (para que puedes recurrir a ellos en caso de emergencia) que contemplan esa estructura.

## Desactivar el monitor

Al realizar la instalación se instala automáticamente el ícono del monitor de servicio en la barra de tareas tal como puedes ver en esta imagen.



Podemos cerrarlo pulsando con el botón derecho del ratón sobre el ícono y eligiendo la opción exit.



Durante el proceso de instalación de Apache se incluye en el menú inicio un enlace al Monitor de Apache. Esto significa que cada vez que reiniciemos el equipo el ícono del monitor se instalará en la barra de tareas. Para desactivarlo de forma definitiva bastará con eliminar tal acceso directo de la forma que puedes ver en la imagen de la derecha.

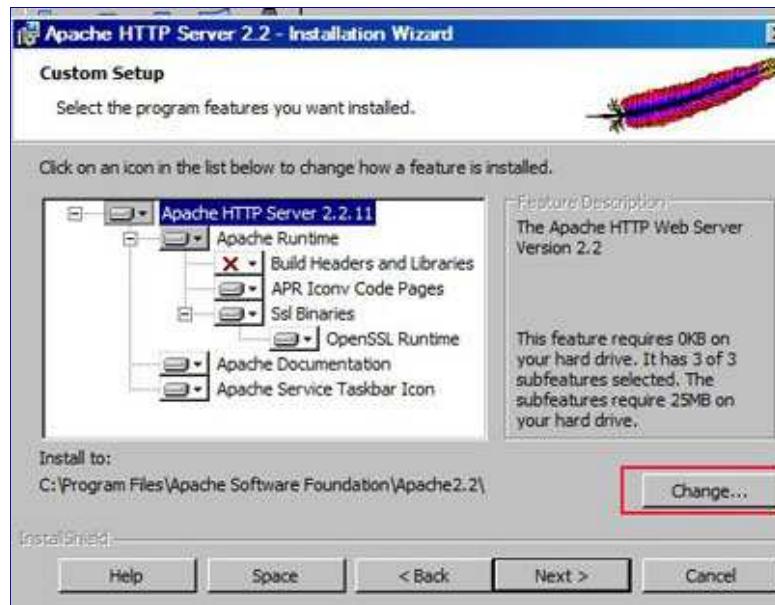
## Arrancar Apache

Contemplando la opción de arranque *manual* cada vez que queramos poner en marcha el servidor tendremos que acceder al ícono *Start Apache in console*, que podrás encontrar en la localización de la imagen que ves a la derecha.

Otra forma de efectuar el arranque sería ejecutar el programa **httpd.exe** que se encuentra en:

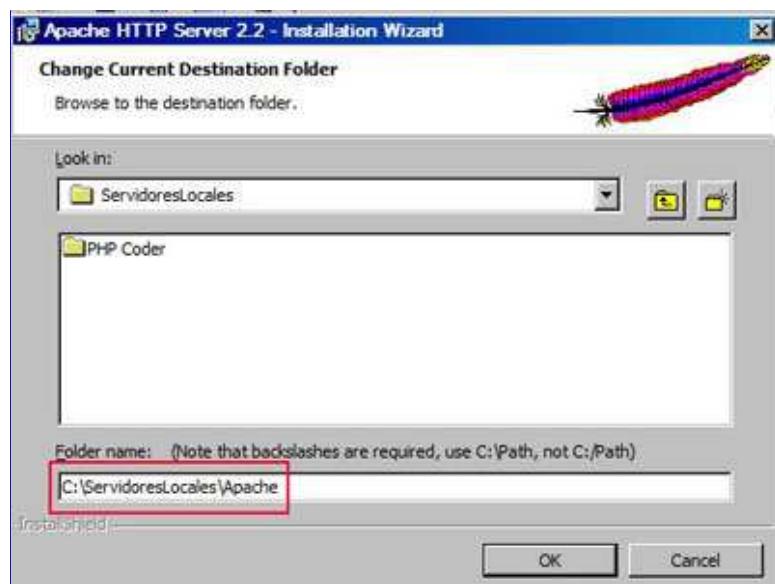
**c:\ServidoresLocales\Apache\bin**

haciendo doble click sobre su ícono o haciéndolo directamente desde el símbolo del sistema de la consola de Windows.



En este caso –aunque no es necesario hacerlo– **vamos a pulsar sobre Change y cambiarlo**. También al margen te explicamos las razones de esta opción.

Después del *Change* de la ventana anterior aparecerá esta otra. Ponla con la misma configuración que aparece en la imagen inferior, es decir, con *Look in* : (**C:\ServidoresLocales**) y deja como *Folder name*: **C:\ServidoresLocales\Apache**



Como podrás leer en la propia pantalla de instalación, es muy importante poner **la barra invertida (\)** detrás de C: al indicar el *Folder name*. No te olvides de hacerlo.

Una vez hayas pulsado *OK* volverá a aparecer la ventana que permite cambiar el directorio de instalación. Comprueba que ahora diga *Install to: C:\ServidoresLocales\Apache* y pulsa *Next* en esa ventana y en las siguientes.

Aparecerá una última ventana donde dice *Installation Wizard Completed* y una vez que hayas pulsado el botón *Finish* la instalación de Apache habrá terminado.

## Desactivar Apache monitor

En el momento del arranque, aparece una ventana de MS-DOS tal como la que ves en la segunda imagen.

### ¡No la cierres nunca!

Si lo haces sólo conseguirías **apagar el servidor** y además lo harías de una forma muy poco *ortodoxa*. Si te resulta *incómoda* minimízala pero no la cierras.

### Una página de prueba

Con el servidor *arrancado y funcionando*, bastará con que abras tu navegador y escribas en la barra de direcciones lo siguiente:

<http://localhost:8080>

o, también

<http://127.0.0.1:8080>

con lo que deberá *abrirse* una página idéntica a la ves un la última de las imágenes. Esa será la prueba definitiva de que el servidor está funcionando correctamente.

Si te estás preguntado por qué esas direcciones –*localhost* ó *127.0.0.1*– o por qué no hemos escrito el nombre de *ninguna de página web*, ¡ten un poco de paciencia! Trataremos de ello cuando *configuremos Apache*.

### Apagar Apache

Aunque se puede desconectar cerrando la ventana de MS-DOS, el método correcto es: **establecer** esa ventana como *ventana activa*, pulsar las teclas **Ctrl+C** y esperar unos pocos segundos. Transcurrido ese tiempo *la ventana MS-DOS* se *cerrará automáticamente* y se habrá **apagado** Apache.

### Desinstalación

El servidor *Apache* se desinstala como cualquier otro programa de Windows. Bastará con acceder a *Paneles de control*, opción de *Agregar o quitar programas* elegirlo en la lista de programas instalados y desinstalarlo.

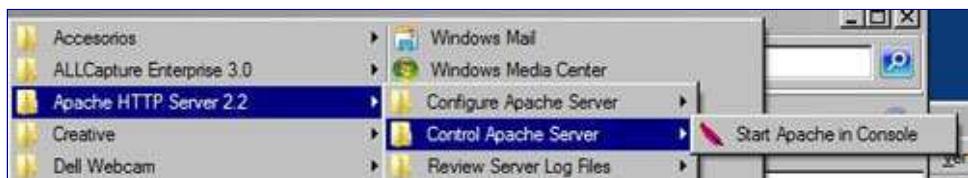
Antes de ejecutar ese proceso es necesario que el servidor esté **apagado** y saber que *quedrán algunos restos de la instalación* que –si fuera necesario– habría que **eliminar** manualmente.

El desinstalador de Apache no eliminará ni el directorio raíz –**C:\ServidoresLocales\Apache**– ni los subdirectorios **conf** y **htdocs** del mismo. La razón es la *seguridad*. El directorio **conf** contiene los archivos de configuración del servidor –en páginas posteriores veremos que hay que modificar sus valores por defecto– y al no eliminarlos nos estará dando la opción de meter

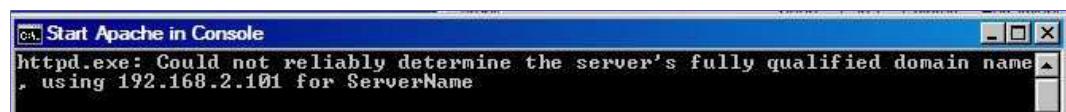


### Arrancar Apache

La imagen ilustra la localización del grupo de programas que crea el instalador de Apache.



Cuando se inicia Apache se abre siempre la consola de MS-DOS sin que aparezca en ella ningún mensaje tal como el que estás viendo en la imagen inferior.



Este mensaje de advertencia (sólo aparece cuando se ejecuta bajo Windows Vista) no afecta las pruebas que vamos a efectuar. No obstante, en los párrafos siguientes veremos como hacer algunas modificaciones en la configuración del servidor que evitan su aparición.



Página de prueba del servidor Apache. Observa la dirección:<http://localhost:8080>

### Configuración de Apache

Fichero inicial	<b>httpd.conf</b> (está en <b>c:\ServidoresLocales\Apache\conf\original</b> )
Guardar como (*)	<b>httpd.org</b>
Guardar como	<b>httpd.conf</b>
Modificaciones en el fichero inicial	
Línea	Cambios
46	Donde dice: (**)
	Listen 8080

reutilizarlos (sin tener que empezar de cero) en futuras instalaciones.

Lo mismo ocurre con **htdocs**. En ese directorio serán donde se almacenen los documentos que vayamos creando -páginas web, imágenes, etc.- y, como es lógico, un error en el proceso de desinstalación podría provocar su pérdida. Esa es la razón de que se conserven.

Por idénticas razones, si al reinstalar Apache ya existieran esos directorios no serían sobrescritos manteniéndose las configuraciones y documentos de la versión anterior.

## Nombres de dominio en redes locales

Cuando se utiliza un servidor a través de una red de área local, lo habitual es acceder a él escribiendo como dirección la IP de ordenador en el que se encuentra instalado el servidor.

Mediante un proceso similar al descrito a la derecha, se puede modificar el fichero HOSTS de uno (o varios) de los equipos de la red añadiendo una línea con el nombre del *dominio ficticio* precedida de la dirección IP del servidor.

Si la IP del servidor local fuera **168.0.12.234**, al incluir en el fichero **hosts** una línea como esta:  
**168.0.12.234 mispruebas.com** podríamos acceder a él tanto mediante:  
**http://168.0.12.234** como a través de:  
**http://mispruebas.com**

	cambiar por:	
		Listen 80
170	Donde dice: (***)	
		#ServerName localhost:8080
	cambiar por:	
		ServerName localhost
Observaciones sobre la configuración		
(*) Guardaremos el fichero modificado con dos nombres distintos. Uno de ellos para hacer la primera prueba del servidor (httpd.conf) y el otro (httpd.orig) para utilizarlo como base de las modificaciones posteriores.		
(**) Mediante esta modificación evitaremos que el servidor escuche el puerto 8080 y le obligaremos a utilizar el puerto por defecto (80).		
Observa que en la página de prueba anterior habíamos escrito: <b>http://localhost:8080</b> (para utilizar el puerto 8080 que era el que escuchaba el servidor) y una vez hecha esta modificación bastará poner: <b>http://localhost</b> ya que el puerto 80 es el puerto por defecto y no es necesario especificarlo.		
(*** ) Las razones de esta modificación son similares a las anteriores. Al quitar el :8080 ya no será preciso incluir este valor al pedir las páginas desde el navegador. Esta modificación nos evita el mensaje de error que aparecía en la consola de MS-DOS al realizar las pruebas anteriores		

## Nombres de dominio en un servidor local

Está opción **no es necesaria** para realizar la instalación propuesta para seguir este curso, pero es posible que te apetezca tener tu **propio dominio** local para poner una *dirección personalizada* en vez de usar **localhost** como nombre de dominio.

Si utilizas **Windows98** encontrarás un fichero con el nombre **hosts** –o **hosts.sam**– en el directorio **Windows**. Si tu sistema operativo es **W2000, NT ó XP Pro** tendrás que buscar ese fichero en: **\winnt\system32\drivers\etc\** y si se trata de **XP Home** lo encontrarás en: **\windows\system32\drivers\etc\**

Al editar ese fichero –**hosts.sam**– encontrarás algo similar a esto:

```
# For example:  
#  
#      102.54.94.97      rhino.acme.com      # source server  
#      38.25.63.10       x.acme.com          # x client host  
  
127.0.0.1      localhost
```

Siañades una línea como la señalada en rojo –la IP 127.0.0.1 ha de mantenerse– y pones cualquier nombre de dominio (**mispruebas.com**) en el caso del ejemplo

```
# For example:  
#  
#      102.54.94.97      rhino.acme.com      # source server  
#      38.25.63.10       x.acme.com          # x client host  
  
127.0.0.1      localhost  
127.0.0.1      mispruebas.com
```

bastará con que guardes el fichero –con los cambios– con el nombre **hosts** (sin ninguna extensión y en el mismo directorio). A partir de ese momento, ya podrás escribir en el navegador -con Apache en marcha- cualquiera de estas direcciones:

**http://localhost**

ó  
**http://127.0.0.1**

ó  
**http://mispruebas.com**



## ¿Qué es PHP?

El lenguaje **PHP** –acrónimo de Hypertext Pre Processor– suele definirse como: *interpretado* y de *alto nivel*. El código de sus instrucciones va insertado en páginas HTML (es un lenguaje de script) y es interpretado, siempre, en el servidor.

Se trata un lenguaje de estilo clásico, cercano en su sintaxis a C++.

## ¿Qué se necesita para trabajar con PHP?

Para trabajar con PHP es necesario tener instalado un servidor HTTP –en nuestro caso, Apache para Windows– y configurarlo de tal manera que sea posible la interacción entre ambas aplicaciones.

## ¿Podemos probar ahora PHP?

Si ya has realizado el proceso de instalación tal como lo hemos descrito en la columna de la derecha, puede que te preguntes si –como hemos hecho al instalar Apache– es posible hacer *ahora* una prueba de su funcionamiento.

La respuesta es NO. Analiza con un poco de detenimiento el proceso que hemos seguido y observarás que no hemos hecho una instalación sino que nos hemos limitado a descomprimir una serie de ficheros colocándolos en unos directorios determinados.

Lo hemos hecho así porque lo que acabamos de instalar no es una aplicación que pueda funcionar de forma autónoma, sino el intérprete de un lenguaje de scripts –del lado del servidor– que requiere ser invocado por medio de un software de servidor.

Así que, antes de que pueda ser utilizado, tendremos que indicarle al servidor el sitio donde ha de buscarse y cuándo debe utilizarlo.

## Instalando el PHP

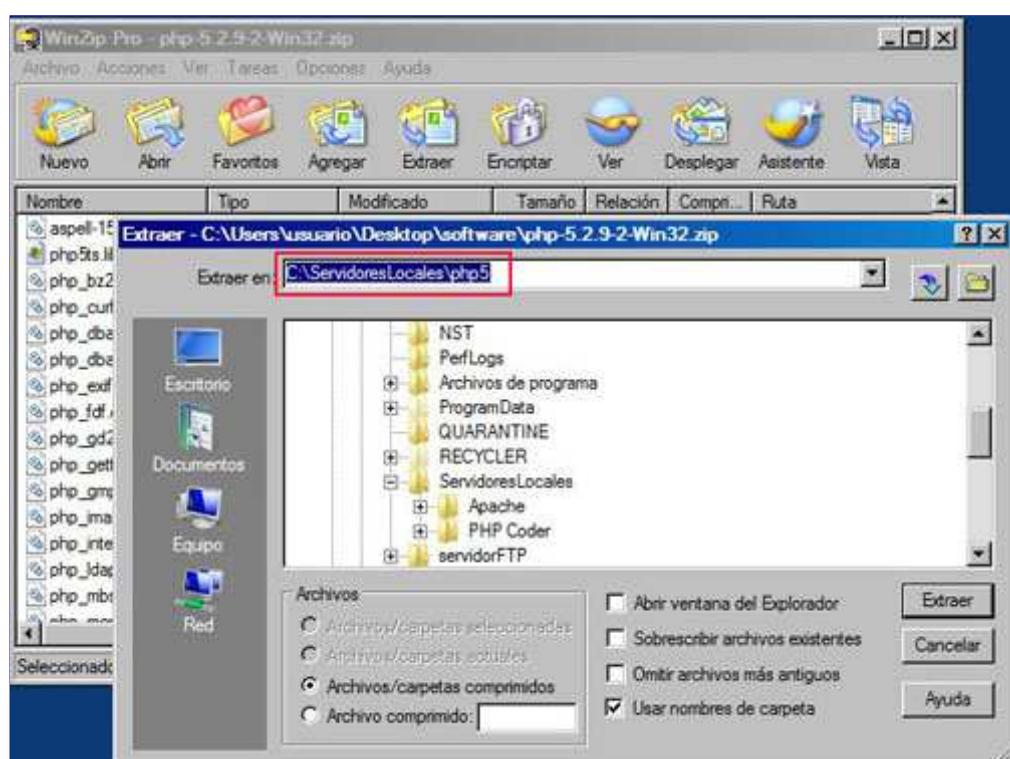
En el CD-ROM del curso –dentro de la carpeta Software– podrás encontrar un fichero llamado **php-5.2.9-2-Win32.zip**.



Si tienes instalado Winzip, al hacer doble click sobre el icono se te abrirá la ventana de este descompresor y al pulsar sobre el icono Extract se te abrirá una ventana como la que aparece en la imagen.

Si has seguido los pasos indicados en la página anterior para la instalación del servidor Apache, tendrás en tu ordenador un directorio llamado **C:\ServidoresLocales**.

Selecciona como directorio de descompresión **C:\ServidoresLocales\php5** –tal como estás viendo en la figura– y realiza la extracción.



Una vez concluida la descompresión se habrá creado el directorio **C:\ServidoresLocales\php5** que contendrá a su vez una serie de subdirectorios necesarios para el correcto funcionamiento de PHP.

El proceso de instalación de PHP habrá terminado. Solo resta la configuración de Apache y PHP para poder empezar a utilizarlo. Esto será lo que veremos en la página siguiente



Ver índice

# Configuración de Apache 2.2.11



## Para facilitar las cosas

A lo largo de este proceso –y en otros sucesivos– vas a encontrar ficheros que tienen el mismo nombre y que se diferencian sólo por la extensión.

Sería una buena idea tener Windows configurado de forma que se visualicen siempre las extensiones de todos los tipos de ficheros.

Te sugerimos que hagas esta modificación. Por si nunca has usado esa opción de Windows, te describimos cómo hacerlo.

Bastará con que hagas doble click en el ícono **Mi PC** y vayas a la opción **Ver** (en el caso de *Windows2000 o XP* habrás de ir a la opción **Herramientas**) de la ventana que se abre. En el submenú de esa opción elige **Opciones de Carpeta**.

Una vez en *Opciones de Carpeta* debes elegir la opción **Ver** de la nueva ventana y **buscar** la línea en la que dice **Ocultar extensiones para los tipos de archivos conocidos y, desmarcando su casilla de verificación** y pulsando sobre **Aplicar** y **Aceptar** ya podrás visualizar las extensiones de todos los ficheros.

## Efecto de los cambios en **httpd.conf**

La configuración de Apache permite múltiples opciones y ofrece muchas posibilidades. Tantas, que justificarían todo un curso dedicado al estudio de este servidor y sus opciones de configuración.

No entraremos en ese ámbito, pero sí trataremos de conocer –de forma somera– los *por qués* de las modificaciones que comentamos a la derecha.

En la *primera* modificación le hemos indicado a Apache que *deberá cargar* un **módulo** que se encuentra en el sitio que indican la **ruta** y el **nombre del fichero**. Este módulo es el que permite que el servidor interactúe con PHP cuando sea necesario.

La segunda de las modificaciones resulta de gran importancia a efectos de seguridad. En la parte derecha tienes comentados los efectos que produce el uso de esa directiva de configuración

## Modificación del fichero **httpd.conf** para configurar PHP como módulo de Apache

Tal como comentábamos en la página anterior la utilización de PHP requiere introducir algunos cambios en la configuración de Apache.

En el momento en el que se instala Apache se crea automáticamente en el *subdirectorio C:\ServidoresLocales\Apache\conf* un **fichero** llamado **httpd.conf** que contiene la configuración por defecto del servidor Apache. Como recordarás lo hemos editado y modificado durante el proceso de instalación para que el servidor utilice el puerto 80

Tendremos que hacer algunas modificaciones más. Pero, como precaución por si tenemos algún problema y necesitamos volver a utilizar el fichero de la configuración por defecto, vamos a hacer una **copia de seguridad**. Abriremos el documento **httpd.conf** con un editor de textos cualquiera –lo más cómodo será utilizar **PHP Coder** que señala los números de línea– y con la opción **guardar como** crearemos una copia con el nombre **httpd.orig**

### ¡Cuidado!

Si has utilizado el *block de notas* de Windows es probable que en el proceso anterior no te haya guardado como **httpd.orig** sino como **httpd.orig.txt**.

Comprueba los ficheros del directorio **c:\ServidoresLocales\Apache\conf** y si te ha ocurrido lo que comentamos tendrás que recurrir al conocido método de pulsar sobre el ícono del fichero con el botón derecho del ratón, elegir la opción *Cambiar nombre* y quitar el **.txt** que aparece al final del nombre del archivo.

Una vez hecho esto ya podremos hacer las modificaciones con toda tranquilidad, así que volveremos a abrir el fichero **httpd.conf** para hacer los cinco cambios siguientes:

Fichero inicial	<b>httpd.conf</b>
Guardar como	<b>httpd.conf</b>
Línea	Modificaciones en el fichero inicial
	Cambios
	Donde dice:
127	(línea en blanco)
	cambiar por:
	<b>LoadModule php5_module C:/ServidoresLocales/php5/php5apache2_2.dll</b>
	Donde dice:
217	<b>Options Indexes FollowSymLinks MultiViews</b>
	cambiar por:
	<b>Options -Indexes FollowSymLinks MultiViews</b>
	Donde dice:
239	<b>DirectoryIndex index.html</b>
	cambiar por:
	<b>DirectoryIndex index.html index.htm index.php</b>
	Donde dice:
382	(línea en blanco)
	cambiar por:
	<b>AddType application/x-httdp-php .php</b>
	Donde dice:
383	#
	sustituir por:
	<b>PHPIniDir "C:/ServidoresLocales/php5"</b>

Una vez efectuados estos cambios ya podremos guardar el fichero, sin cambiar su nombre original –**httpd.conf**–, y tendremos lista la nueva configuración de Apache.

Mediante la tercera de ellas le estamos diciendo a Apache que cuando reciba una **petición** –dirigida a uno cualquiera de los directorios accesibles a través de HTTP– en la que no se especifique ningún nombre de página, debe *comprobar* si en ese directorio existe alguna página llamada *index.html*.

En caso de que dicha página existiera la mostraría y en caso contrario *volvería a comprobar* para ver si existe alguna otra llamada *index.php* (el segundo nombre de página contenido en esa línea).

En caso de no encontrar esa *coincidencia* deberá continuar comprobando una a una, y de forma secuencial, la existencia de las páginas siguientes hasta encontrar alguna cuyo nombre coincidiera con uno de la lista establecida en la configuración de *httpd.conf*. En el caso de que no encontrara ninguna que coincida con los nombres indicados en esta *directiva* daría un mensaje de error del tipo: *File not found*.

Esta opción de configuración de Apache es la que nos permite escribir direcciones del estilo [www.isftic.mepsyd.es](http://www.isftic.mepsyd.es) en las que –sin especificar ningún nombre de página– nos aparece en pantalla el mismo contenido que si hubiéramos escrito:

[www.isftic.mepsyd.es/index.html](http://www.isftic.mepsyd.es/index.html)

Como podrás ver, esta opción de configuración añade mayor comodidad para al usuario.

Mediante la cuarta modificación *AddType application/x-httpd-php* se le indica a Apache que los únicos ficheros susceptibles de contener *scripts* que deban ser ejecutados por el intérprete de PHP son aquellos que tienen como extensión *.php*. Si un fichero con extensión distinta contuviera *scripts* PHP éstos **no serían ejecutados**.

Por último, la inclusión de la línea *PHPIniDir* (una novedad respecto a versiones anteriores de Apache y PHP) permite indicar la ruta de acceso al fichero de configuración de PHP (*php.ini*) sobre cuya configuración trataremos en la página siguiente.

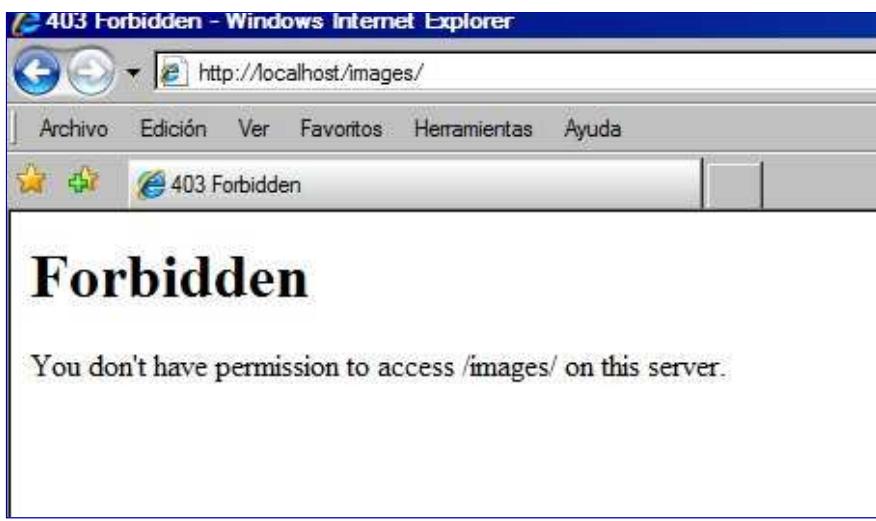
### Otras modificaciones en *httpd.conf*

A lo largo de curso iremos incorporando algunos servicios que requerirán algunos otros cambios en la configuración de Apache.

Habaremos de ellos cuando sea necesario realizarlos.

## Algunos detalles importantes sobre *httpd.conf*

Si –con la configuración descrita más arriba– intentamos acceder a la dirección: <http://localhost/images/> nos aparecería un mensaje con este:



esto ocurre como consecuencia de haber puesto el *signo menos* delante de *Indexes* (**cuidado!** debe ir *pegado* a *Indexes* sin ningún espacio intermedio) en la línea *Options -Indexes FollowSymLinks MultiViews*.

El subdirectorio **images** no contiene ficheros con nombre *index.html*, ni tampoco *index.php* ni *index.htm* (los especificados en la directiva *DirectoryIndex*) y el *signo menos* lo que hace es denegar el acceso (en el caso de no indicar el nombre de algún documento en la dirección) a los directorios que no los contengan.

Si no hubiéramos modificado esa directiva nos aparecería una lista con todos los ficheros contenidos en ese subdirectorio, tal como puedes ver en esta imagen.



Aunque no vamos a modificarlos, es conveniente saber que existen algunos otros elementos importantes en *httpd.conf*.

### DocumentRoot "C:/ServidoresLocales/Apache/htdocs"

Esta línea que se ha incluido automáticamente al hacer la instalación indica la ruta y el nombre del directorio en el que, **obligatoriamente**, han de estar los documentos –y los subdirectorios– susceptibles de ser servidos a través del protocolo HTTP.

Cualquier documento que estuviera *fuerza de este directorio sería inaccesible*, por lo tanto todos los documentos que vayamos generando a lo largo del curso, **deberemos guardarlos dentro de este directorio htdocs**.

Como es lógico, bastaría cambiar esa línea de la configuración para utilizar como *raíz del servidor* otro directorio cualquiera.

## **ServerName *localhost***

Esta otra línea –también contenida en httpd.conf– es la que determina el nombre del servidor y a través de ella se identifican las peticiones que el navegador realiza a ese servidor. Por esta razón, cuando probábamos la instalación de Apache, escribíamos como dirección <http://localhost>.

---

Anterior



Índice



Siguiente





Ver índice

# Configuración de PHP 5.2.9



## Un directorio importante

Antes de comenzar la configuración de php es conveniente que creamos dentro de **C:\ServidoresLocales** un subdirectorio con nombre **tmp**



que será utilizado en la configuración de php para contener las eventuales variables de sesión que fuera necesario manejar.

## Los cambios en php.ini

Con la configuración de **PHP** ocurre lo mismo que en el caso de Apache. También existen un montón de posibilidades de configuración –iremos viendo algunas de ellas a medida que vaya siendo necesario– a través de las cuales se puede modificar de forma sustancial el comportamiento de PHP.

Cuando nos hemos referido a la elección de *hosting* hemos comentado que sus niveles de prestaciones son distintos y que muchos de ellos tienen desactivadas algunas de las funciones. Pues bien, es en este fichero y en una línea donde dice *disable\_functions* = en la que se podría incluir la lista de funciones a desactivar.

Modificando *directivas* en este fichero podremos establecer también restricciones relativas a cuestiones de seguridad, así como activar *extensiones* por medio de las cuales –librerías dinámicas– se añaden nuevas funciones de PHP con las que podríamos crear ficheros en formato **PDF** o imágenes dinámicas, por citar dos ejemplos.

Ya iremos hablando de ello. Por el momento será suficiente con utilizar la configuración que comentamos a la derecha.

## Antes de empezar la configuración

Si abrimos nuestro directorio: **C:\ServidoresLocales\php5** encontraremos dentro de él cinco ficheros que deberemos copiar a nuestro sistema tal como se indica en la siguiente tabla

Ficheros originales	libeay32.dll libmcrypt.dll libmhash.dll libmysql.dll php5ts.dll	Copiar en el subdirectorio que se indica (dentro del que contiene el S.O.)
S.O.		Directorio
Windows98		\system
W2000		
Windows NT		\system32
Windows XP		
Windows Vista		

Si ya tuviéramos ficheros con ese nombre en el directorio de destino **tendremos que sobrescribirlos** sustituyendo los preexistentes por los que tenemos en **C:\ServidoresLocales\php5**.

## Configuración de PHP

En el directorio **C:\ServidoresLocales\php5** tenemos un fichero llamado **php.ini-dist**. Lo abriremos con un editor de textos –por ejemplo el *PHP Coder*– y haremos estas *modificaciones*:

Fichero inicial	<b>php.ini-dist</b>	
Guardar como	<b>php.ini</b>	Modificaciones en el fichero inicial
Línea		Cambios
	Donde dice:	
484	cambiar por:	<b>doc_root =</b>
		<b>doc_root=c:\ServidoresLocales\Apache\htdocs\</b>
	Donde dice:	
491	cambiar por:	<b>;extension_dir="./"</b>
		<b>extension_dir ="c:\ServidoresLocales\php5\ext"</b>
	Donde dice:	
616	cambiar por:	<b>;extension=php_gd2.dll</b>
		<b>extension=php_gd2.dll</b>
	Donde dice:	
617	cambiar por:	<b>;extension=php_gettext.dll</b>
		<b>extension=php_gettext.dll</b>
	Donde dice:	
623	cambiar por:	<b>;extension=php_mbstring.dll</b>
		<b>extension=php_mbstring.dll</b>
	Donde dice:	
624	cambiar por:	<b>;extension=php_mcrypt.dll</b>
		<b>extension=php_mcrypt.dll</b>
	Donde dice:	
630		

## Nuestras modificaciones

Vamos a comentar ahora las modificaciones del *php.ini* que describimos aquí a la derecha.

En la primera hemos asignado como valor de **doc\_root** una ruta que, como observarás, apunta el directorio **htdocs**. En otras palabras, le estamos diciendo a PHP en qué sitio del ordenador –fíjate que incluimos una ruta absoluta– se ubicarán los ficheros cuyos scripts debe interpretar.

Con la modificación donde dice **extension\_dir** le señalamos al intérprete de PHP en qué sitio debe buscar las **extensões** que pueda necesitar.

Estas extensiones, que vienen con la instalación de PHP, se descomprimen –por defecto– en un subdirectorio llamado **ext** y esa es la razón por la que la ruta incluida en esta modificación apunta a un directorio con ese nombre.

En las modificaciones siguientes lo único que hacemos es descomentar (quitar el punto y coma que llevan delante) algunas de las extensiones de php.

Descomentaremos las líneas alusivas a las librerías: **php\_gd2** (para el tratamiento y generación de imágenes dinámicas), **php\_mbstring** (esquemas de codificación de caracteres multibyte se han desarrollado para expresar más de 256 caracteres y que será necesario en algunas funciones de MySQL), **php\_mcrypt** (soporte para diferentes algoritmos de encriptación), **php\_mysql** (imprescindible para la comunicación entre PHP y MySQL).

La directiva **sesion.save\_path** indica la ruta de un directorio dónde se guardarán los ficheros de las sesiones.

## Eventuales problemas de instalación

Esperamos que las configuraciones no te hayan dado ningún problema y que todo te esté funcionando correctamente.

De no ser así, te sugeriríamos que fueras comprobando esta *lista de chequeo*:

- ¿Está instalado el servidor Apache en **c:\ServidoresLocales\Apache**?
- ¿Está instalado PHP5 en el directorio **c:\ServidoresLocales\php5**?
- ¿Has copiado en el directorio del sistema las librerías dinámicas (dll): *libeay32*, *libmcrypt*, *libmhash*, *libmysql* y *php5ts* que viene con la instalación de PHP?
- ¿Está el fichero *php.ini* en el subdirectorio **php5**?

		cambiar por:	:extension=php_mysql.dll
		Donde dice:	<b>extension=php_mysql.dll</b>
700		cambiar por:	<b>SMTP= localhost</b>
		Donde dice:	<b>SMTP = 127.0.0.1</b>
704		cambiar por:	<b>:sendmail_from= me@example.com</b>
		Donde dice:	<b>sendmail_from= admin@mispruebas.com</b>
992		cambiar por:	<b>:session.save_path = "/tmp"</b>
		Donde dice:	<b>session.save_path = C:/ServidoresLocales/tmp</b>

Las modificaciones de las líneas 700 y 704 (SMTP y send\_mail están relacionadas con el uso de un servidor de correo cuya instalación y configuración veremos en la página siguiente).

La modificación de la línea 992 está relacionada con el uso de sesiones y la analizaremos en el momento en que tratemos ese tema

## ¡Cuidado!

En la configuración de PHP (*php.ini*) –bajo Windows– debemos usar siempre la **barra invertida** (\) a la hora de escribir los *paths*.

Cuando tratamos la configuración de Apache (*httpd.conf*) –también bajo Windows– lo hemos hecho al revés, hemos escrito todos los paths utilizando la **barra normal** (/).

Ten muy presente que estas sintaxis **son distintas** y cuando efectúes modificaciones de configuración utiliza la adecuada a cada uno de los ficheros.

Una vez que hayamos modificado los apartados anteriores guardaremos el fichero con el nombre **php.ini** en el directorio **c:\ServidoresLocales\php5** dado que este es el lugar especificado en la directiva **PHPIniDir** de la configuración de Apache de la página anterior.

## Un script de prueba

Para comprobar que todo funciona correctamente deberemos escribir nuestro primer *script PHP*.

Abriremos nuestro editor –*PHP Coder*– y escribiremos **exactamente** esto:

```
<? phpinfo(); ?>
```

Ahora lo guardaremos en **C:\ServidoresLocales\Apache\htdocs** –recuerda que este es el que hemos configurado como directorio raíz de servidor– con el nombre **info.php**

Recuerda también que es probable que el *block de notas* haya añadido la extensión **.txt** y que el fichero puede haber sido guardado como **info.php.txt**. Lo comprobaremos mirando el directorio **htdocs** y cambiando el nombre si fuera necesario.

## Probando el primer script

Una vez instalados y configurados Apache y PHP y creado el fichero **info.php**, ha llegado el momento de comprobar si hemos hecho correctamente las configuraciones y si todo funciona bien.

Arrancaremos el servidor Apache y una vez que tengamos la ventana de MS-DOS abierta, deberemos visualizarla sin ningún mensaje (completamente en negro) lo que significaría que Apache está funcionando con el módulo PHP.

Si apareciera algún mensaje de error sería necesario corregir la línea del fichero de configuración a la que se aluda en el propio mensaje.

Solo faltaría abrir el navegador y escribir una de estas dos direcciones:

<http://localhost/info.php> o <http://127.0.0.1/info.php> y aparecería en pantalla una página como esta:

System	Windows NT EC-DSY 6.0 build 6001
Build Date	Apr 9 2009 08:22:37
Configure Command	cscript /nologo configure.js "--enable-snapshot-build" "--enable-debug-pack" "--with-snapshot-template=d:\php-sdk\snaps_5_2\wc6\x86\template" "--with-php-build=d:\php-sdk\snaps_5_2\wc6\x86\php_build" "--with-pdo-oci=D:\php-sdk\oracle\instantclient10\ sdk\shared" "--with-oci8=D:\php-sdk\oracle\instantclient10\ sdk\shared"
Server API	Apache 2.0 Handler
Virtual Directory Support	enabled
Configuration File (php.ini) Path	C:\Windows
Loaded Configuration File	C:\ServidoresLocales\php5\php.ini

De ser así, el proceso de instalación y configuración habría terminado y esa página nos estaría facilitando toda la información relativa a la configuración actual de nuestro PHP.

---

Anterior



Índice



Siguiente





Ver índice

# Autoinstalación de servidores



## Precauciones previas

Esta instalación no es otra cosa que una alternativa *automática* para el caso que decidas evitar o posponer la instalación manual que venimos describiendo en estas primeras páginas del curso.

Antes de efectuar esta instalación te sugerimos que desinstales cualquier versión anterior que pueda tener en tu equipo. Hacemos particular hincapié en la desinstalación de cualquier versión de Filezilla Server ya que puede ser una fuente de problemas.

## La instalación automática

La utilidad que comentamos aquí no hace otra cosa que realizar de forma automática los procesos descritos en páginas anteriores. Es decir:

- Crear el directorio de instalación (ServidoresLocales, por defecto)

- Instalar el servidor Apache.

- Instalar PHP4 y PHP5

- Copiar las librerías dll -de PHP4 y Php5- requeridas en el directorio system.

- Instalar MySQL configurando de forma automática el usuario *pepe*.

- Instalar el servidor FTP incluyendo todos los usuarios y permisos que se describen en la página de instalación manual de este servicio.

- Creación de los directorios servidorFTP (en el disco c:) tal como se describen en el proceso manual.

- Instalación del servidor de correo y configuración de cuentas de los usuarios: *juan*, *perico* y *andres* tal como se describen en el proceso de instalación manual de este servidor.

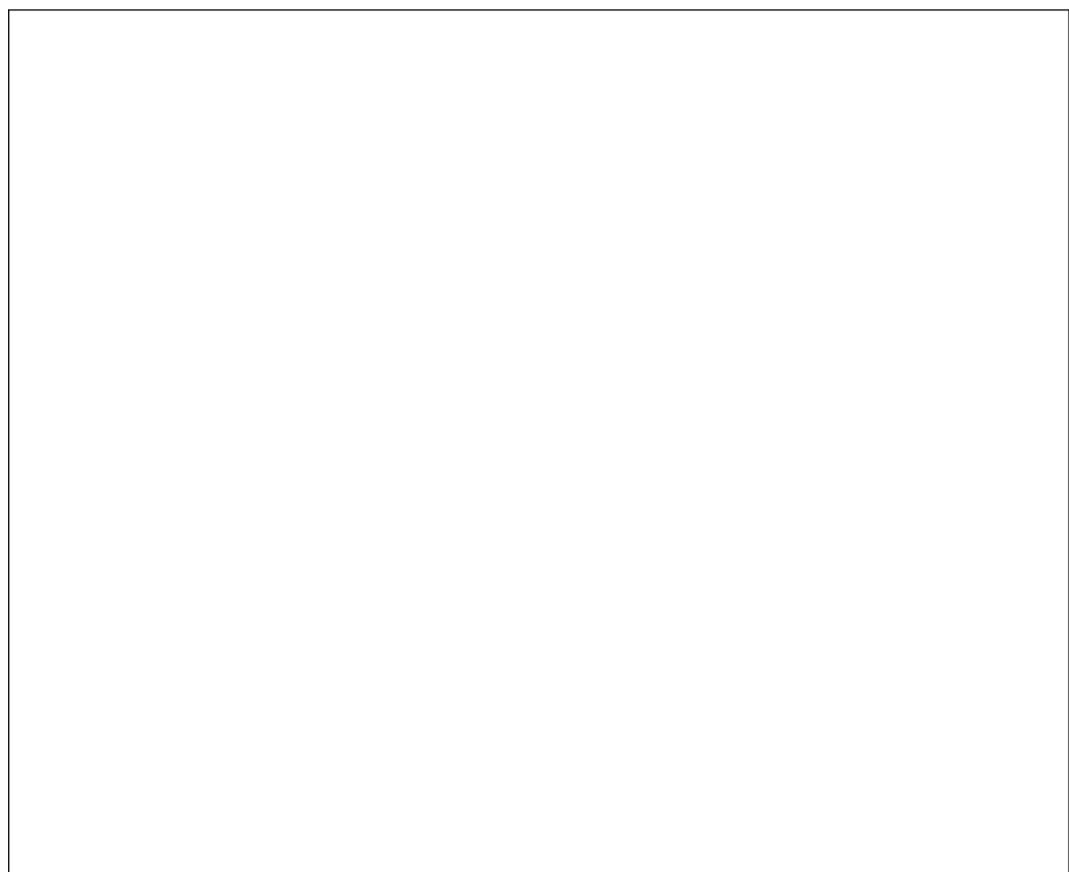
- Instalación del editor *Php-Coder*

- También instala manuales de ayuda de PHP y MySQL y una pequeña aplicación que permite gestionar cada uno de los servidores (arrancar y parar) y la posibilidad de trabajar con PHP4 o PHP5.

- Al cambiar la versión de PHP se reconfigura automáticamente la configuración del servidor.

## Instalación automática

En el directorio **software** del CD-ROM encontrarás un fichero llamado *Instalar\_servidores.exe*. Bastará con que hagas doble click sobre su ícono y comenzará el proceso de instalación automática cuyas ventanas y secuencia de instalación puedes ir viendo en la imagen.



Pulsando sobre la imagen podrás visualizar los diferentes pasos del proceso

Al ejecutar este instalador ya tendrás totalmente **instalados y configurados** en tu equipo: **Apache 2.2**, **PHP5**, **PHP4**, **MySQL**, **FileZilla Server** y el servidor de correo **Mercury**, además del **editor PHP-Coder**.

Una vez realizada esta instalación ya estaremos en condiciones de comenzar los desarrollos de los contenidos del curso tanto en su versión Iniciación como en la de Profundización.

### ¡Cuidado!

Recuerda que si utilizas **Windows Vista** como sistema operativo deberás ejecutar la aplicación de gestión de servidores (también el desinstalador) **con privilegios de Administrador**. En vez del clásico doble click sobre el ícono deberás pulsar con el botón derecho y elegir Ejecutar como Administrador.

De no hacerlo así -puede que en principio todo tenga apariencia de funcionar sin ese requisito- se te «*colgará*» el equipo cuando intentes poner en marcha el servidor FTP.

Durante el proceso de desinstalación del conjunto Servidores Locales también será necesarios los privilegios de Administrador para que pueda desinstalarse correctamente el servicio Filezilla Server.

Anterior



Índice



Siguiente



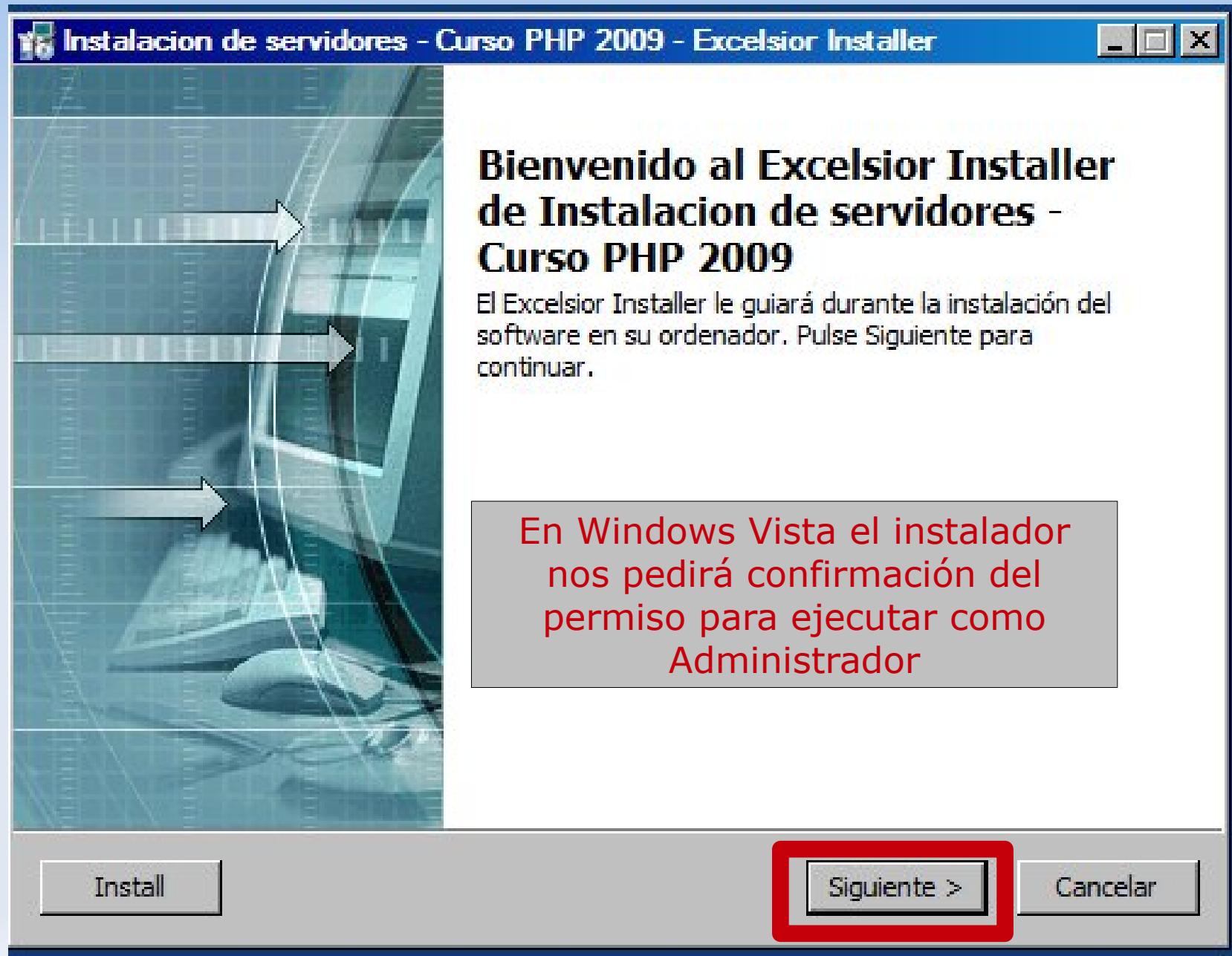
# Instalación automática de los servidores



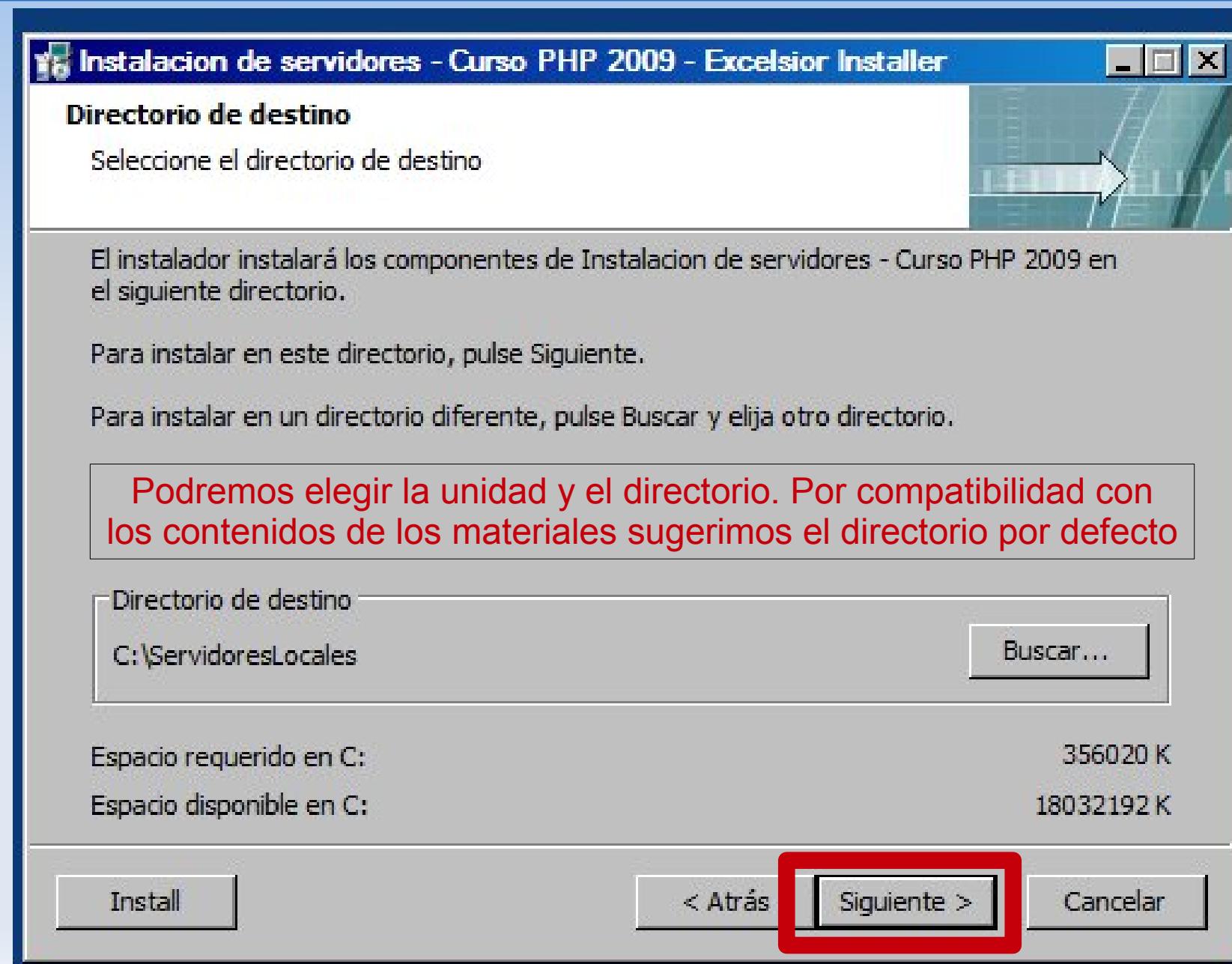
Este programa está en el directorio software del CD del curso

Si lo seleccionamos y pulsamos **enter**  
comenzará el proceso de instalación y configuración  
automática de todo el software del curso

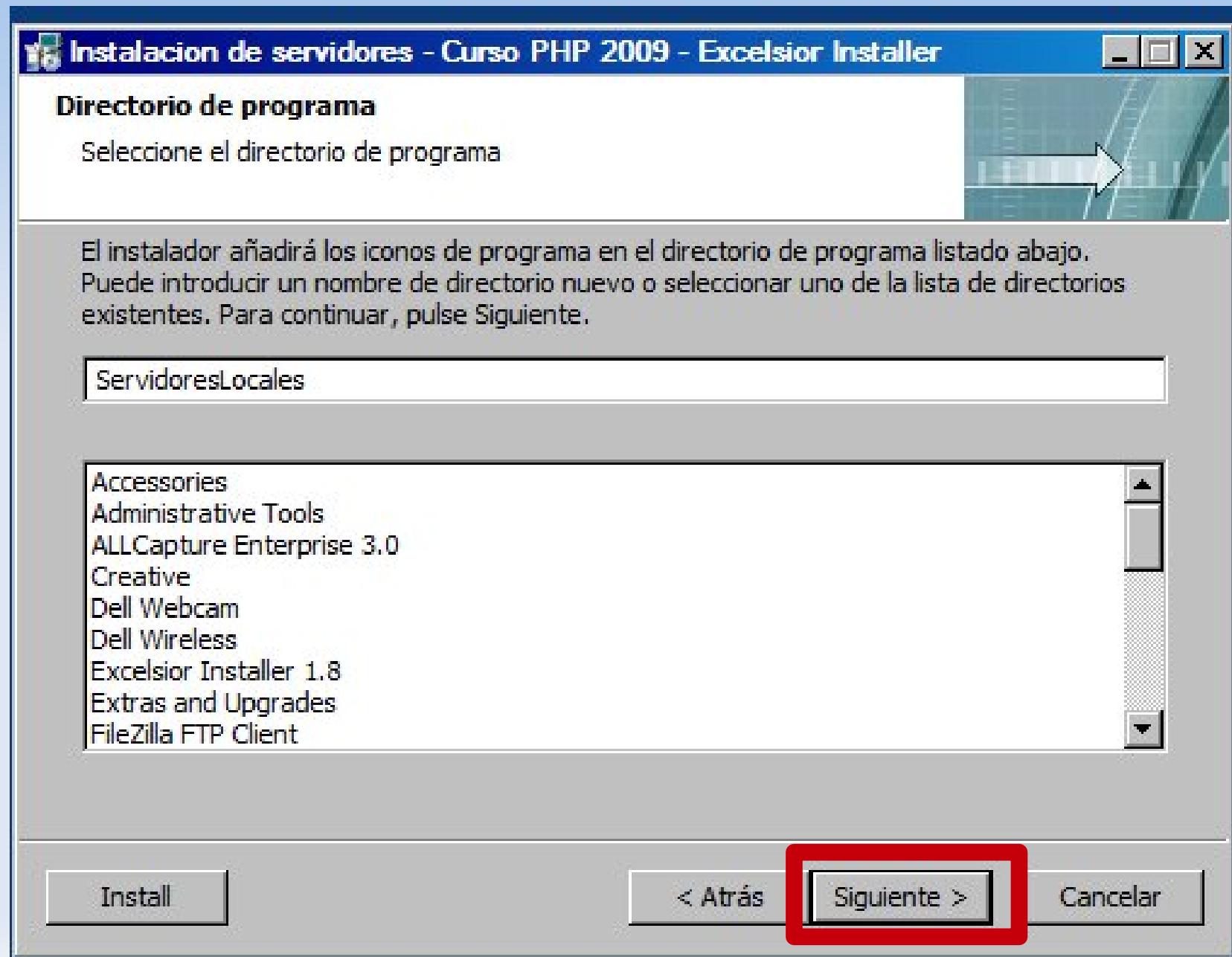
# Instalación automática de los servidores



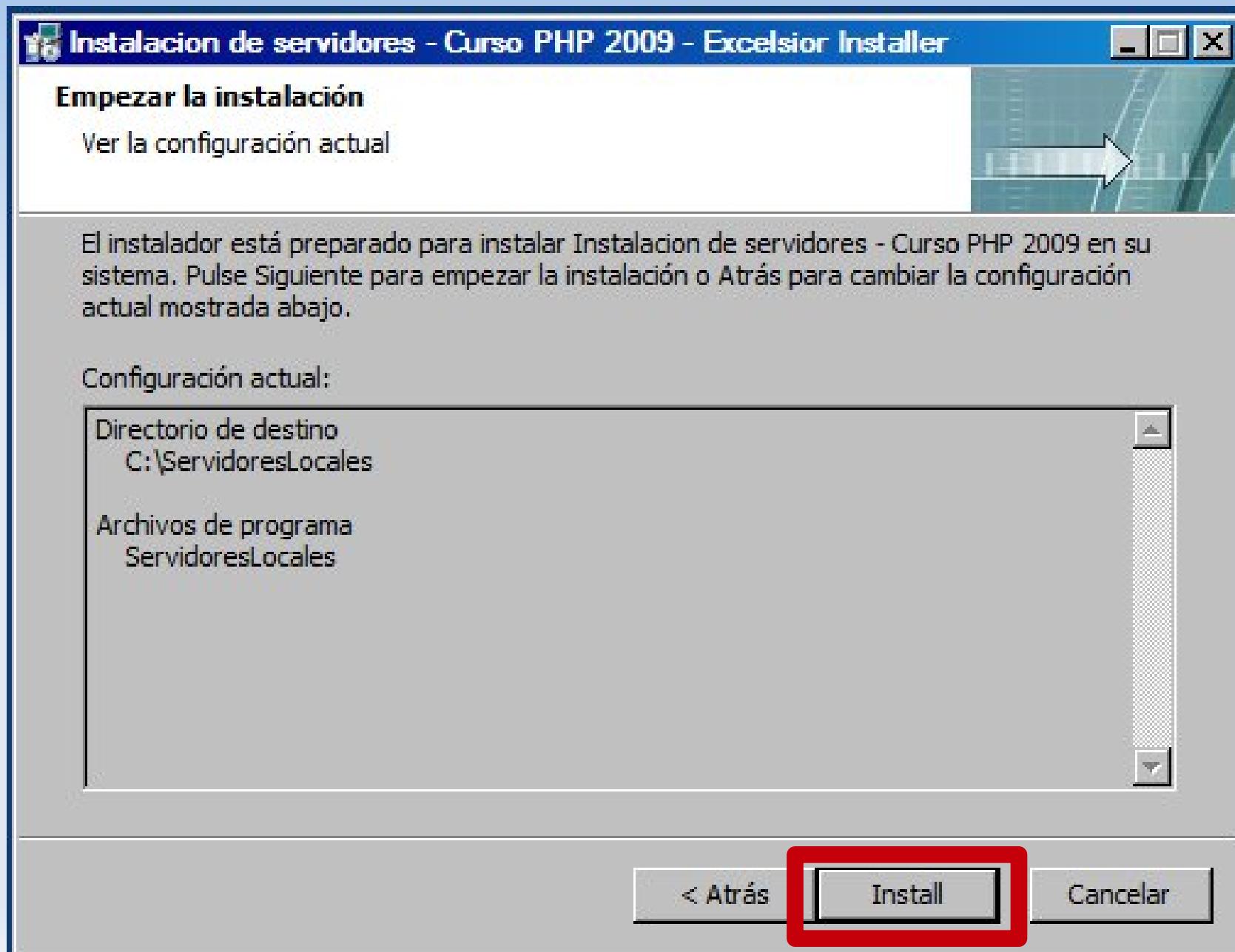
# Instalación automática de los servidores



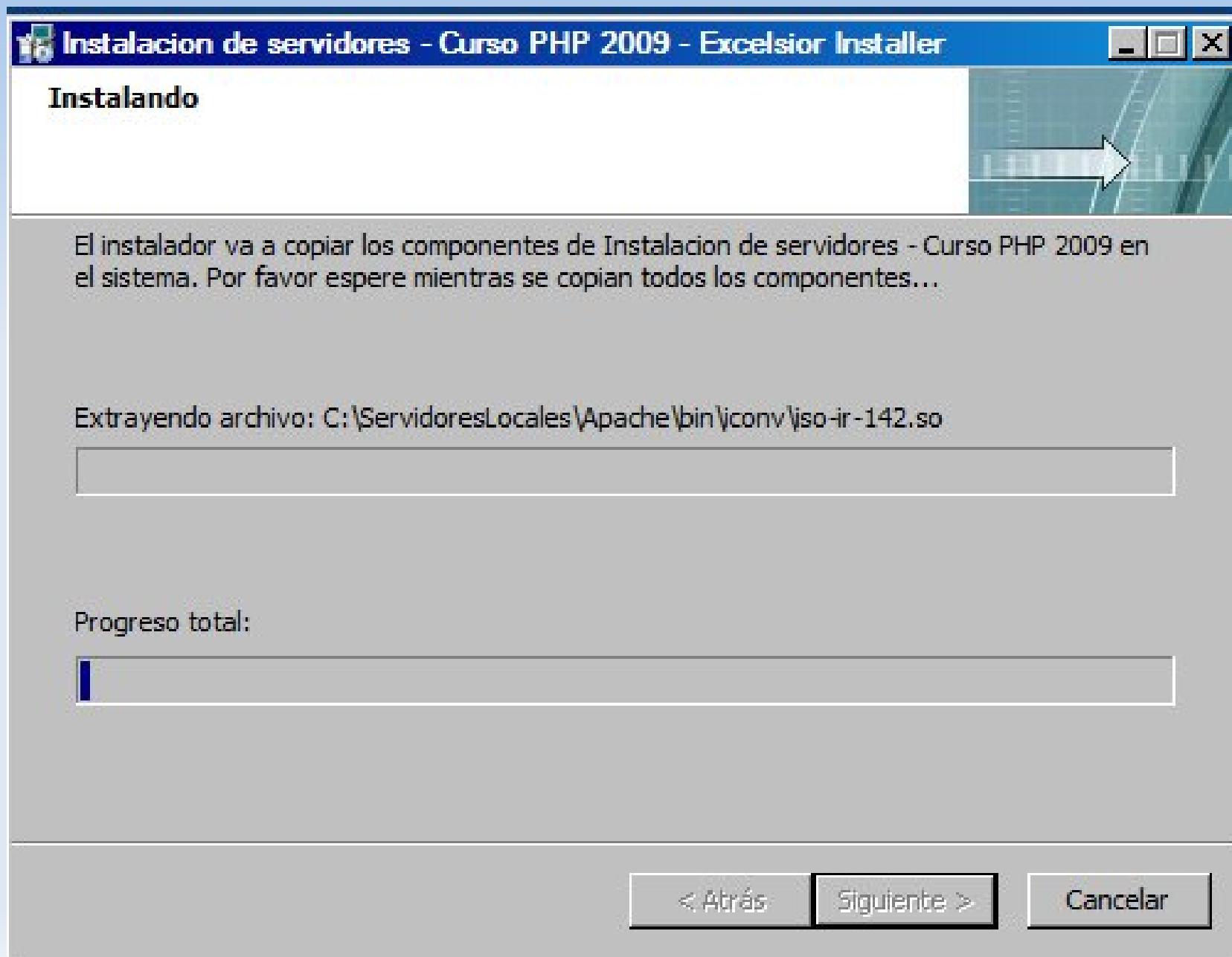
# Instalación automática de los servidores



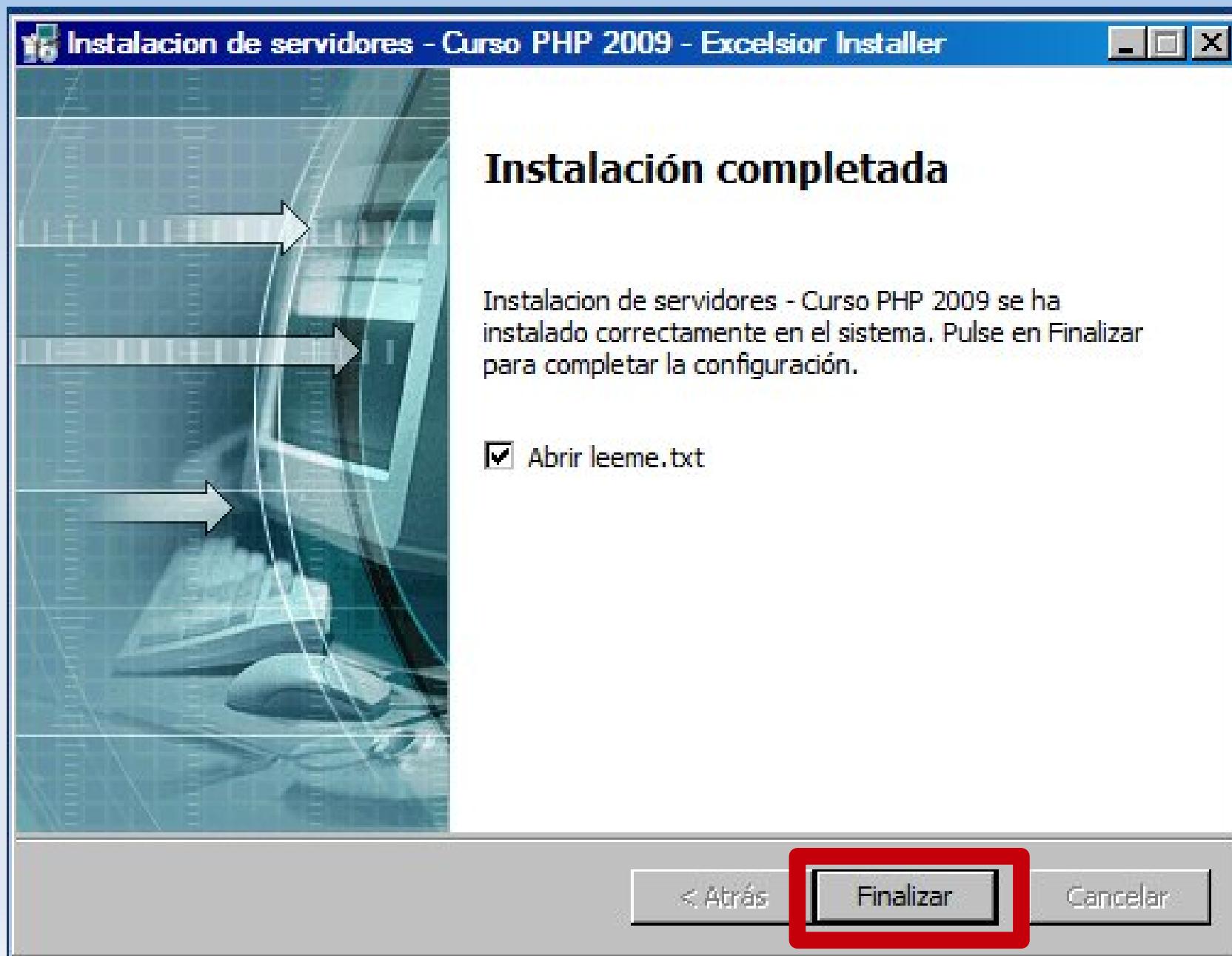
# Instalación automática de los servidores



# Instalación automática de los servidores

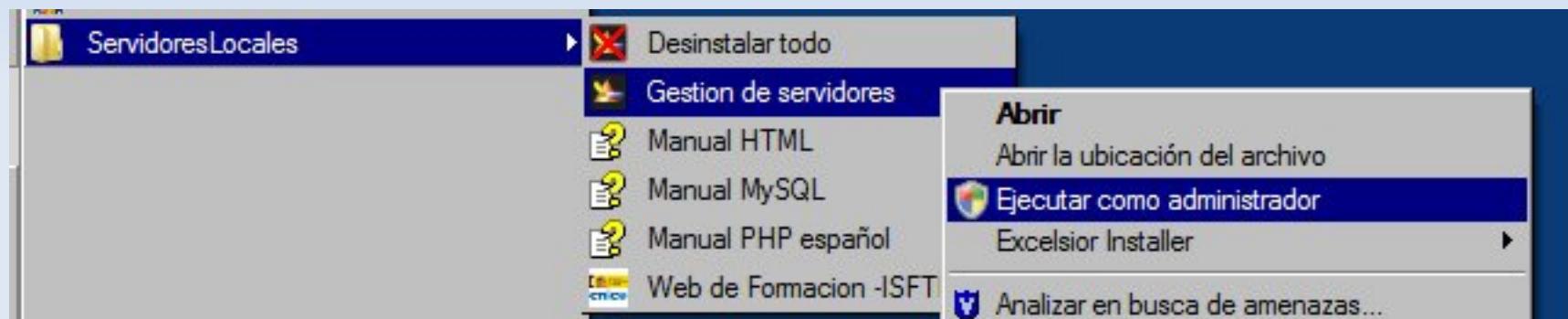


# Instalación automática de los servidores



# Instalación automática de los servidores

Durante el proceso anterior se creará un grupo de programas tal como el que se muestra en la imagen. Al pulsar sobre **Gestión de Servidores** se nos abrirá una ventana desde la que podremos activar /desactivar cada uno de ellos. Ya tenemos instalado y configurado todo lo necesario para trabajar con PHP.



## ¡Cuidado con Windows Vista!

Para ejecutar la aplicación sobre este sistema operativo debemos hacerlo como administrador. Para ello bastará pulsar con el **botón derecho** del ratón sobre el ícono Gestión de servidores y, después, elegir Ejecutar como administrador en el menú contextual que se despliega y que estamos viendo en la imagen

# Instalación automática de los servidores



Al pulsar sobre los iconos podremos activar/desactivar cada uno de los servidores: Apache+PHP, MySQL, Filezilla Server FTP y Mercury Mail



Ver índice

## Instalando MySQL 5.1.33



### Acceso a las bases de datos

El acceso y tratamiento de la información en bases de datos **MySQL** requiere que los usuarios estén identificados mediante un nombre (*login*) y –opcionalmente— una contraseña de acceso.

El propio instalador de MySQL incluye, por defecto, un *login* con nombre **root**, que utiliza como *password* **una cadena vacía**.

A cada usuario se le pueden asignar **privilegios** de modo que, por ejemplo, solo pueda realizar consultas, o acceder a tablas concretas. El usuario **root** goza de todos los privilegios posibles y podría ser usado para todos los supuestos de este curso. No obstante, como en situaciones reales es un usuario desaconsejable por el riesgo que entraña utilizar usuarios por defecto, vamos a crear un nuevo usuario –**con contraseña y con todos los privilegios**— que será el que utilizaremos en los ejemplos relativos a MySQL.

### Arrancar y parar el servidor MySQL

La gestión *habitual* de MySQL se realiza a través de la pantalla de MS-DOS (en el caso de Windows98) o mediante su equivalente *Símbolo del sistema* en las versiones de Windows más recientes.

Como opción alternativa, es posible configurar la opción de efectuar esos procesos desde Windows utilizando un programa llamado *MySQL Administrator*.

Utilizaremos la opción consola de DOS para realizar la puesta en servicio y la configuración de un nuevo usuario.

### Algunos comandos para ejecutar MySQL desde MS-DOS

Antes de empezar a ejecutar los comandos de MySQL es necesario situarse en el subdirectorio **bin** que está dentro del directorio de instalación que en nuestro caso es **C:\ServidoresLocales\mysql**.

Para ello hemos de escribir en el *prompt* de DOS lo siguiente:

### Proceso de instalación MySQL

Dentro del directorio **Software** del CD-ROM del curso podrás encontrar el fichero **mysql-5.1.33-win32.msi** que contiene el fichero de instalación de **MySQL**.



Al hacer doble click sobre su ícono comienza el proceso de instalación cuya secuencia de pantallas sucesivas puedes ir viendo en la secuencia de imágenes que tienes aquí debajo.

Usaremos el directorio **c:\ServidoresLocales\mysql** tanto para hacer la instalación del servidor como para albergar el directorio que ha de contener las bases de datos.

Pulsando sobre la imagen podrás visualizar los diferentes pasos del proceso

### Puesta en servicio desde la consola de MS-DOS

Una vez finalizado el proceso de instalación ya podremos poner en marcha el servidor. Lo haremos desde el símbolo del sistema usando los comandos de MS-DOS.

### Arrancar MySQL

Para poner en marcha MySQL basta con situarnos en **C:\ServidoresLocales\mysql\bin** (subdirectorio bin del directorio en el que se efectuó la instalación de MySQL) y ejecutar el comando:

CD C:\ServidoresLocales\mysql\bin

Una vez en el directorio **bin** (allí están los ejecutables de MySQL) los comandos básicos de arrancar y parar, así como el de creación de un usuario los tienes descritos junto a las ilustraciones de la derecha.

## Creación de un nuevo usuario

Será preciso que **creemos** un nuevo usuario. El usuario **root** no tiene contraseña y vamos a ver la opción de crear uno que requiera el uso de una contraseña para acceder.

Durante el proceso de creación le daremos los *máximos privilegios* (ALL PRIVILEGES) de modo que pueda gestionar *cualquier base de datos, tabla* y que además pueda *crearlas, borrarlas o modificarlas sin restricción alguna*.

Es importante que lo creamos con la sintaxis exacta ya que, los ejemplos de los temas relacionados con MySQL están desarrollados utilizando el usuario **pepe** con contraseña **pepa** (ambas en minúsculas).

## ¿Dónde se almacena la información?

Al instalar el programa se crea –dentro del directorio **mysql**– un subdirectorio llamado **data** destinado a contener todas las bases de datos que vayan a ser gestionadas por MySQL.

Cada base de datos estará contenida en un subdirectorio diferente que tendrá el mismo nombre que de la base que contiene.

El instalador de MySQL crea de forma automática **dos bases de datos** con los nombres: **mysql** y **test**.

¡No debes borrarlas!

La base de datos **mysql** contiene los datos relativos a usuarios y si no está presente, MySQL no funcionará.



La denominada **test** es una base de datos que permite chequear la instalación y la configuración de MySQL.

El directorio **bin** es el que contiene los programas que gestionan las *bases de datos* en MySQL.

C:\ServidoresLocales\mysql\bin>mysqld

a partir de su ejecución el servidor MySQL ya estará activo y podríamos gestionarlo desde PHP ó desde la propia consola MS-DOS.

### ¡Cuidado!

En algunas versiones de Windows es posible que, al arrancar el servidor, se quede el cursor intermitente en la ventana de MS-DOS sin que regrese al prompt. Basta con cerrar la ventana –Símbolo del Sistema– y reabrirla. El servidor seguirá activo y ya será posible la ejecución de comandos desde esta consola.

## Apagar el servidor MySQL

Para poder apagar el servidor es necesario utilizar un nombre de usuario. En este caso utilizaremos el nombre de usuario **root** (el usuario que se crea por defecto).

La sintaxis sería la siguiente:

C:\ServidoresLocales\mysql\bin>mysqladmin -u root shutdown

## Ejecutar instrucciones como usuario

El primer paso para poder ejecutar sentencias MySQL será tener el servidor activo. Para ello habrá que seguir el proceso descrito anteriormente. El paso siguiente sería ejecutar una instrucción como esta (solo en el caso del usuario **root** o de un usuario sin contraseña).

Ejecutaremos esta instrucción (con idéntica sintaxis) para ejecutar una sentencia MySQL.

C:\ServidoresLocales\mysql\bin>mysql --user=root mysql

Este mensaje de bienvenida y el *cambio de directorio* (observa que ahora el *prompt* ha cambiado y apunta hacia **mysql**) nos indican que MySQL está *listo para recibir instrucciones*.

```
c:\ServidoresLocales\mysql\bin>mysql --user=root mysql
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 1
Server version: 5.1.33-community MySQL Community Server (GPL)

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql>
```

Lo único que haremos desde aquí será **crear un usuario**. En adelante *nos comunicaremos con MySQL* a través de la web y usando como herramienta de comunicación el PHP.

Para crear el nuevo usuario utiliza exactamente la sintaxis que ves en la imagen.

```
mysql> GRANT ALL PRIVILEGES ON *.* TO pepe@localhost
-> IDENTIFIED BY 'pepa';
Query OK, 0 rows affected (0.05 sec)
```

### ¡Cuidado!

El pulsar **Enter** en MySQL no significa –como ocurre en DOS– que se vaya ejecutar el comando. Si observas la imagen, hemos pulsado **Enter** detrás de la palabra *localhost* de la primera línea y lo que ha ocurrido es que el cursor ha saltado hasta la segunda incluyendo automáticamente **->** que significa que *continua la instrucción anterior*. En MySQL las ejecución de las instrucciones requiere que haya un **;** inmediatamente antes del pulsar la tecla *Enter*.

## Desinstalación de MySQL

MySQL se desinstala desde la opción **Agregar o quitar programas** como cualquier otro programa de Windows. El proceso de desinstalación no elimina ni los ficheros **ini** ni tampoco el subdirectorio **data**. Este último se conserva como medida de seguridad ya que contiene todas las bases de datos y de eliminarlo se perdería la información. Si en algún momento tratas de desinstalar para hacer una nueva instalación, lo aconsejable sería mantener el directorio **data** y **buscar y eliminar** todos ficheros **my.\*** (los ini de la instalación anterior) antes de realizar la nueva instalación.

## Salir del *interface de usuario*

Para abandonar la interface de usuario basta con escribir **exit** tal como ves en la imagen. El sistema escribir su mensaje de despedida de forma automática y el prompt regresará a **C:\ServidoresLocales\mysql\bin>**

```
mysql> exit  
Bye  
c:\ServidoresLocales\mysql\bin>
```

### ¡Cuidado!

Independientemente de que puedas crear otros usuarios con otras contraseñas es imprescindible crear el usuario **pepe** con contraseña **pepa**. Todos los ejemplos que incluimos requieren este usuario.

## Acceso al *interface de usuario con contraseña*

Cuando un usuario registrado con contraseña (tal como ocurriría al usuario **pepe** creado en los párrafos anteriores) trate de acceder al *interface de usuario* deberá usar *siempre* una sintaxis como esta:

```
C:\ServidoresLocales\mysql\bin>mysql --user=pepe -p mysql  
Enter password: ****_
```

La única diferencia con la que hemos usado anteriormente estriba en la inclusión del modificador **-p** que indica que ese usuario requiere contraseña. Al hacerlo, nos pedirá que introduzcamos la clave (Enter password) y solo entonces nos permitirá el acceso.

Anterior



Índice



Siguiente





Ver índice

# Instalando phpMyAdmin



## ¿Qué es PHPMyAdmin?

PHPmyAdmin es, simplemente, un conjunto de utilidades y scripts escritos en lenguaje PHP que permiten gestionar bases de datos MySQL a través de una página web.

Mediante esta herramienta, sin conocer el lenguaje MySQL, podemos modificar, consultar, crear y borrar tanto bases de datos como tablas y registros contenidos en ellas.

También permite la gestión de usuarios –recuerda que MySQL requiere claves y contraseñas– y modificar sus privilegios de acceso.

## ¿Cuál es su utilidad?

La versión de MySQL para Windows no dispone de una interface propia que permita gestionar sus bases de datos a través de Windows.

La única posibilidad de gestión es a través de MS-DOS y eso requiere que el usuario sepa utilizar los comandos propios de las funciones MySQL.

Sin esos conocimientos de SQL, tendríamos como recurso la posibilidad de gestionar las bases de datos a través de nuestros propios scripts PHP, pero eso nos exigiría conocer con una cierta profundidad –a medida que avancemos en el curso lo iremos logrando– las funciones que PHP posee para la gestión de este tipo de bases de datos.

Es por eso que este conjunto de herramientas resulta *muy cómodo y fácil* de utilizar y está convirtiéndose –de hecho ya lo es– en el soporte estándar que la mayoría de los hosting facilitan a sus usuarios para gestionar las bases de datos alojadas en sus servidores.

## La advertencia

Al visualizar la página que tienes en la imagen de la derecha habrás visto un aviso sobre temas de seguridad.

¡No te preocunes!.

La razón de tal advertencia radica en la existencia del usuario **root** (el usuario por defecto) sin contraseña. Podríamos eliminarlo –ya tenemos al usuario **pepe** con todos los privilegios– y desaparecería el mensaje. Cuando estudiemos la

## Proceso de instalación

En el directorio **Software** del CD-ROM hay un fichero llamado **phpMyAdmin-3.1.3.1-all-languages.zip** que tenemos que **descomprimir obligatoriamente** (contiene scripts de PHP) en el directorio **c:\ServidoresLocales\Apache\htdocs**.

Al hacerlo se creará un directorio llamado **phpMyAdmin-3.1.3.1** al que vamos a *cambiar el nombre* por otro más cómodo y fácil, dado que al utilizar phpMyAdmin tendremos que escribir el nombre de ese directorio con bastante frecuencia.

Vamos a renombrarlo como **phpMyAdmin**.

Será necesario editar el fichero **Config.class.php** (contenido en el subdirectorio **libraries** de **phpMyAdmin**) y modificar la *Línea nº 18*. Dónde dice:

```
var $default_source = './libraries/config.default.php';
deberemos poner
var $default_source = './libraries/config.inc.php';
```

En el mismo subdirectorio **libraries** y, en él, un fichero llamado **config.default.php**. Abrámoslo con nuestro editor **PHP Coder**, guardémoslo (sin hacer ninguna modificación) con el nombre **config.inc.php**, y ya, en este último fichero hagamos los cambios que se detallan en la tabla siguiente.

Fichero inicial	<b>config.default.php</b>
Guardar como	<b>config.inc.php</b>
	Modificaciones en el fichero config.inc.php
Línea	Cambios
39	Donde dice: \$cfg['PmaAbsoluteUri'] = ":"; cambiar por: \$cfg['PmaAbsoluteUri'] = 'http://localhost/phpmyadmin/';
168	Donde dice: \$cfg['Servers'][\$i]['auth_type'] = 'cookie'; cambiar por: \$cfg['Servers'][\$i]['auth_type'] = 'config';
218	Donde dice: \$cfg['Servers'][\$i]['nopassword'] = false; cambiar por: \$cfg['Servers'][\$i]['nopassword'] = true;
345	Donde dice: \$cfg['Servers'][\$i]['AllowNoPasswordRoot'] = false; cambiar por: \$cfg['Servers'][\$i]['AllowNoPasswordRoot'] = true;

Hechas las modificaciones en ambos ficheros ya estaremos en condiciones de probar su

[Ver índice](#) [Búsqueda rápida](#) [◀ Página anterior](#) [Página siguiente ▶](#)

## Prueba de funcionamiento de phpMyAdmin

Ya estamos en condiciones de hacer una prueba de phpMyAdmin. **Arranquemos Apache** –dejando minimizada su ventana MS-DOS– y **arranquemos** también **MySQL**. Con ambos servidores activos escribamos <http://localhost/phpmyadmin> en nuestro navegador y se nos abrirá una página como esta:

relativo a MySQL ya lo haremos, pero por el momento, dejémoslo así.

## Los usuarios

En las imágenes de la derecha tienes la lista de usuarios actuales. Allí ves el archi mencionado usuario **root** y también al nuevo usuario **pepe** con su contraseña *encriptada*.

Las columnas marcadas con **Y/N** contienen las tablas de privilegios de cada usuario. Observa que tanto **root** como **pepe** tienen todos los privilegios, mientras que, el tercer usuario (con nombre en blanco y creado durante la instalación de MySQL) no tiene ninguno.

The screenshot shows the phpMyAdmin interface for MySQL localhost. The left sidebar lists databases: information\_schema (28), mysql (23), and test. The main area is titled 'MySQL localhost' with a sub-section 'User'. It displays a table with four rows:

	Host	User	Password	Select_priv
<input type="checkbox"/>	localhost	root		Y
<input type="checkbox"/>	127.0.0.1	root		Y
<input type="checkbox"/>	localhost			N
<input type="checkbox"/>	localhost	pepe	*C38E402D61D6E5B30487546DBED41C181C4267E1	Y

Explorando los enlaces de la parte izquierda de la pantalla podremos visualizar los contenidos de la tabla user que nos dará una imagen como esta:

	Host	User	Password	Select_priv
<input type="checkbox"/>	localhost	root		Y
<input type="checkbox"/>	127.0.0.1	root		Y
<input type="checkbox"/>	localhost			N
<input type="checkbox"/>	localhost	pepe	*C38E402D61D6E5B30487546DBED41C181C4267E1	Y

Anterior



Índice



Siguiente





Ver índice



## ¿Por qué transferir los documentos?

Dado que estamos en un curso de PHP y teniendo en cuenta que PHP es un *lenguaje del lado del servidor*, resulta imprescindible que, a partir de ahora, trabajemos *desde el servidor*.

Tanto los ejemplos como los ejercicios que tendrás que ir realizando van a requerir, de forma ineludible, que utilicemos continuamente el servidor.

Por esta razón es necesario que toda la documentación esté accesible a través de *nuestro propio* servidor.

## Proceso de transferencia

Ahora es el momento de *transferir* todos los materiales del curso al servidor que hemos instalado y configurado.

Vamos a *buscar* en el CD-ROM una carpeta llamada **cursophp** y vamos a *copiarla* íntegra dentro del directorio **C:\ServidoresLocales\Apache\htdocs**.

Una vez transferida esta carpeta *arrancaremos* nuestro servidor Apache y abriremos el navegador poniendo como dirección <http://localhost/cursophp/> y ya tendremos acceso al índice principal de este Curso.

Ahora podemos prescindir del CD-ROM. De aquí en adelante –con la excepción del software– para seguir el curso sólo será necesario que **arranquemos el servidor Apache** y accedamos a la dirección:

<http://localhost/cursophp/>

Ha llegado el momento de empezar con PHP. ¡Adelante!

Anterior



Índice



Siguiente





Ver índice

# Instalación del servidor FileZilla FTP



## ¿Qué es un servidor FTP?

Conceptualmente un servidor FTP no difiere en nada de un servidor HTTP.

Las diferencias entre los diversos tipos de servidores estriban en los programas que usan –software de servidor– en el tipo de petición que aceptan y en el protocolo que requieren las peticiones que atiende.

En el caso de **Apache**, se trata de un servidor *HTTP* porque sólo acepta peticiones a través de ese protocolo.

Al hablar de **servidores FTP** nos referimos a aquellos que únicamente aceptan peticiones realizadas por medio del protocolo conocido como **FTP** (File Transfer Protocol).

## Arrancar y apagar el servidor

Después de instalar el servidor aparecerán en Programas un grupo con unos iconos como estos.



Como resulta obvio, el señalado con **Start** permite la puesta en marcha del servidor y el señalado con **Stop** para detener su funcionamiento.

La opción **Server Interface** permite configurar del servidor y controlar su funcionamiento.

Para activar esta opción es necesario que hayamos puesto en marcha previamente el servidor.

Al activar el **Server Interface** por primera vez aparece una ventana como la que tienes a la derecha. Si marcamos la casilla de verificación y pulsamos **OK** ya no aparecerá en posteriores arranques.

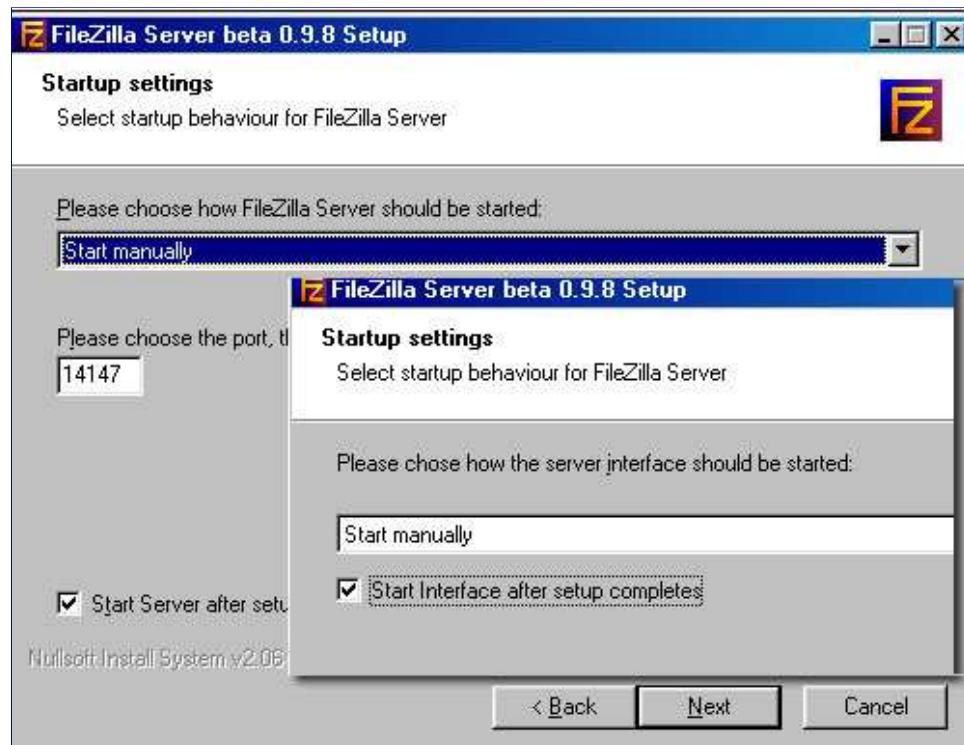
La ventana de Server Interface nos mostrará el estado del servidor e irá registrando todas las acciones que se realicen en él.

Si has optado por la instalación automática de todos los programas bastará con que pulses sobre el ícono correspondiente al servidor FTP desde la ventana de Gestión de Servidores.

## Instalación de un servidor FTP

Se trata del instalador de un *servidor FTP*, que –tal como te indicamos en el apartado Software del Curso– puedes descargar desde [FileZilla\\_Server\\_0\\_9\\_18](#).

Podemos instalarlo con las opciones *por defecto*, salvo en el caso de las dos ventanas que ves en la imagen:



en las que elegiremos *Start manually* en ambas opciones. De esta forma evitaremos que servidor arranque al conectar el ordenador –consumiendo menos recursos– y tendremos la opción de activarlo sólo en el momento en que necesitemos usarlo.

## Server Interface



## ¡Cuidado!

Sobre Windows Vista es necesario ejecutar este programa con **privilegios de administrador**.

En vez de doble click típico pulsa con el **botón derecho** del ratón y elige la opción de **ejecutar como Administrador**.



## Los iconos del Server Interface

El primero de los iconos permite arrancar y detener el servidor desde esta consola.

El segundo (en forma de candado) permite bloquear y desbloquear el servidor sin necesidad de detenerlo.

Desde el tercero podremos hacer modificaciones generales en la configuración. Optaremos por dejarlas con las opciones por defecto.

Desde el cuarto, uno de los más interesantes para nuestros propósitos podremos crear cuentas de usuarios así como realizar su configuración.

El quinto de los iconos -muy similar al anterior- permite acceder a la configuración de *grupos* de usuarios.

## Definición de los servicios

El directorio raíz (o *root*) de un servidor FTP –igual que ocurría en el caso de Apache– puede ser uno **cualquiera** de los existentes en el ordenador en el que tengamos instalado el *software de servidor*.

Imaginemos que hemos hecho la instalación de **Apache** y de **FTP** en un equipo que actúa como servidor de la *red local* de nuestro Centro.

Supongamos que es nuestra intención ofrecer tres servicios:

– **Gestion** al que sólo tendrá acceso el *secretario* del centro.

– **Documentacion** en el que sólo podrá insertar documentos el *secretario* y al que tendrán acceso todos los *profesores*.

– **Alumnos** que va a contener tres áreas distintas: *Materiales*, *Exámenes* y *Trabajos*. Podrán acceder a ellas -con los privilegios que comentamos al margen- *profesores* y *alumnos*.

Para lograr estos fines será necesario que el ordenador en el que

## Creación de cuentas de usuario

Empezaremos creando tres cuentas de usuario: *super* con *superi* como contraseña; *webmaster* con *webmaster* como contraseña; y *secre* utilizando *secret* como contraseña.

La forma de hacerlo es la que ves en la imagen inferior. Se pulsa en el ícono *user* de la ventana *Server interface* (el cuarto de los iconos de la imagen que tienes aquí arriba) y aparece la ventana que ves en la parte inferior.

La secuencia de creación de un usuario es la siguiente:

Se selecciona *general* (1).

y se pulsa en el botón *Add* de la ventana de usuarios (2). Se abrirá una nueva ventana. Se escribe -en la nueva ventana- el nombre de usuario (3).

Se deja la opción *None* (por defecto) en el menú de opciones *User should..* (4) y se pulsa el botón *OK*. Se escribe -en la nueva ventana- el nombre de usuario (5).



Al pulsar *ok* se cierra y se la ventana *Add user account*.

Se marca la opción *Enable account* (6).

Se marca la casilla de verificación *Password* (7) y se escribe la contraseña de usuario (8).

Repetiremos el proceso hasta crear las cuentas de los dos usuarios indicados al comienzo de este párrafo y acabaremos pulsando el botón *OK* que está situado en la **parte inferior izquierda** de la ventana de usuarios.

## Elección de los roots de cada usuario

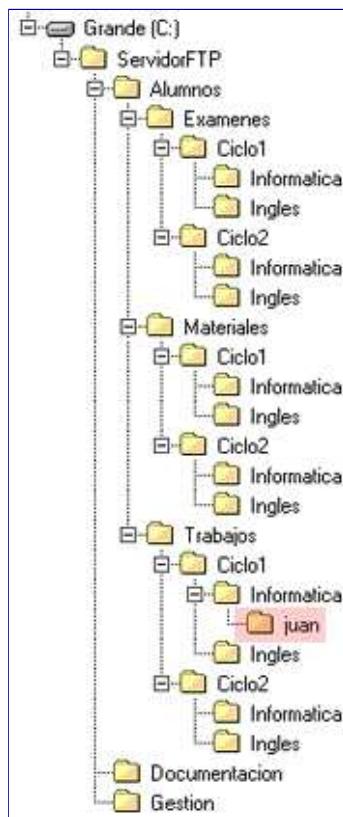
Dado que al usuario *webmaster* se le va a permitir únicamente el acceso a **c:\ServidoresLocales\Apache\htdocs** (para la gestión del servidor Apache) hemos de definir como root el directorio **c:\ServidoresLocales\Apache\htdocs**.

Para el caso de *super* (el usuario que controla en su integridad el servidorFTP) le estableceremos como root **c:\servidorFTP**. Cuando se trate del usuario *secre* a quien vamos a permitirle el acceso a **c:\ServidorFTP\Documentacion** y a **c:\ServidorFTP\Gestion** hemos de establecer como root el directorio **c:\ServidorFTP** (nivel superior a ellos y que, por tanto, los contiene a ambos).

## Directorios accesibles y privilegios

tenemos instalado el servidor ha de contener -entre otros- los directorios que ves en la imagen.

Deberemos crear el directorio **ServidorFTP** y todos los que contiene con excepción de **juan** (sombreado en rojo) que se generará de forma automática.



Configuraremos los usuarios de forma que **super** (usuario que administra el sistema) pueda acceder sin restricciones a todo el contenido del directorio **ServidorFTP** y todos sus subdirectorios.

Iguales privilegios habrá de tener el usuario **webmaster** en el Document Root del servidor Apache.

El usuario **secre** podrá acceder los directorios **Documentación** y **gestion**.

Dado que los restantes usuarios -tanto los profesores como los alumnos- van a ser varios, y van a tener acceso a los mismos espacios del servidor, crearemos **grupos** con cada uno de ellos.

Lo primero de todo, sería crear las cuentas de usuario tal como se describe al margen.

## Roots de usuarios

Cada usuario ha de disponer de un directorio raíz (su **root**) que **ha de contener todos los directorios a los que va a tener acceso**.

A la derecha tienes descritos los criterios de elección de esos **root**.

Es bastante frecuente el hecho de que un usuario **no deba acceder a los contenidos de todos los subdirectorios de su root**.

Cada uno de los directorios accesibles para un usuario (*Shared folders*) puede gozar de privilegios distintos. Al seleccionar un directorio se activan las casillas de verificación que ves en la imagen y desde ellas se pueden **conceder privilegios** a dos niveles: *ficheros y directorios*.

A nivel de ficheros cuenta con las opciones *Read* (descargar), *Write* (escribir, subir), *Delete* (borrar) y *Append* (añadir a un fichero preexistente cuando la transferencia ha sido interrumpida). Cada una de ellas puede configurarse como opción *permitida* ó *no permitida*.

Cuando se trata de *directorios* las opciones (también puede configurarse cada una de ellas como *permitida/no permitida*) son: *Create* (Crear), *Delete* (borrar), *List* (ver una lista de los contenidos) y *+SubDirs* (Cuando está activado **asigna automáticamente a todos los subdirectorios que contenga los mismos privilegios que al directorio actual**).

El proceso de establecimiento de la accesibilidad de un directorio (y la concesión de privilegios) es la que se detalla en la imagen.

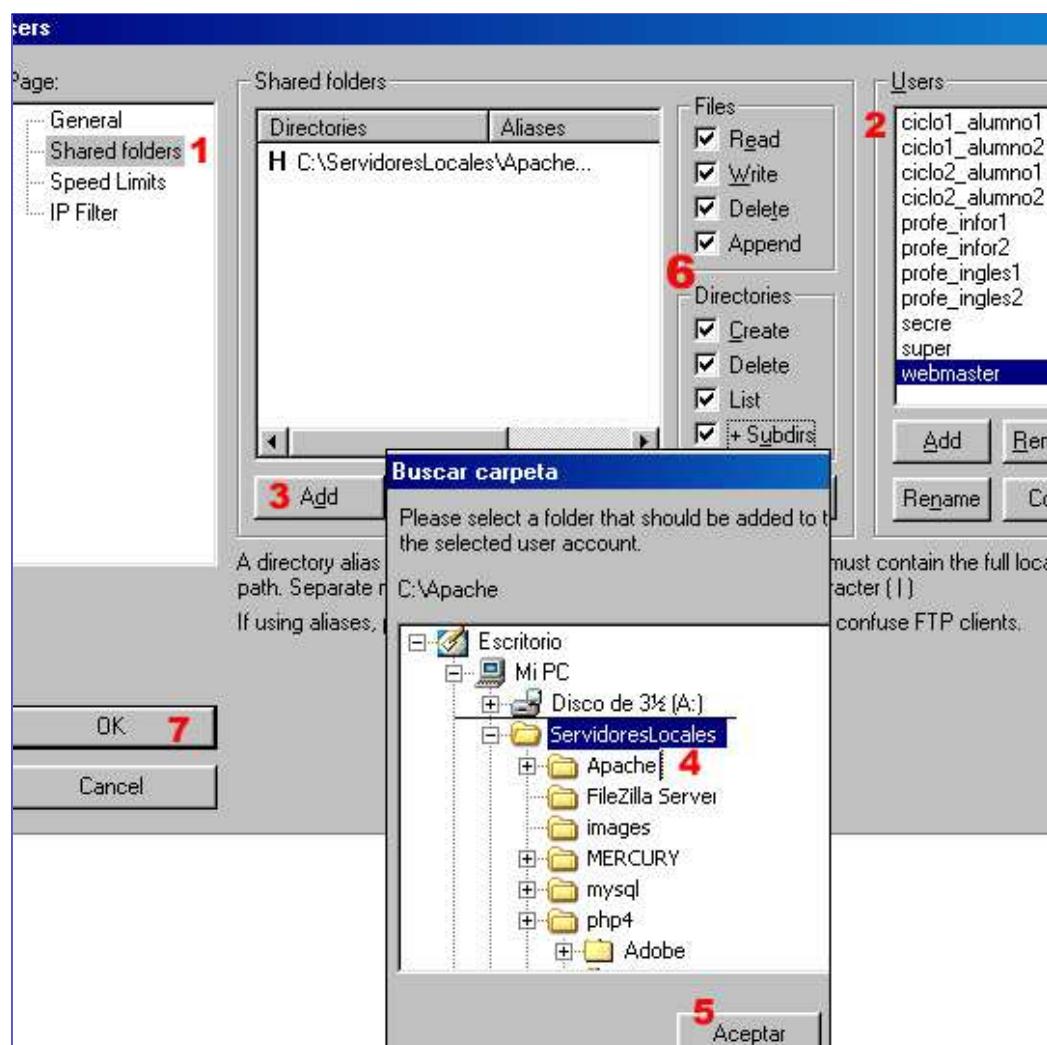
Se accede desde el icono *users* de la ventana *Server Interface* siguiendo la siguiente secuencia:

Se elige *Shared Folder* en la ventana de la izquierda (1).

Se elige un usuario *User* en la ventana de la derecha (2).

Se pulsa en el botón *Add* en la parte central de la ventana (debajo de *Shared Folders*) (3) con lo que se abre automáticamente la ventana *Buscar carpeta*.

Se elige un directorio (4) y se pulsa sobre el botón *Aceptar* (5).



Se establecen los privilegios -marcando o desmarcando las casillas de verificación correspondientes- para el directorio elegido (6).

Se pulsa OK para guardar los cambios de configuración (7).

El proceso puede repetirse cuantas veces sea necesario.

## Directorios accesibles por los usuarios **super** y **webmaster**

para los usuarios *super*, *webmaster* y *secre*.

El hecho de un directorio sea el *root* de un usuario no implica que pueda acceder a sus contenidos ya que para hacerlo es necesario que tenga -además- permisos de acceso.

## Privilegios de los usuarios

Esta sería la manera en la que podríamos asignar privilegios a cada uno de los usuarios.

### *super*

Sus privilegios en los diferentes directorios podrían ser:

#### - En C:\servidorFTP

**Todos.** Al ser el usuario *super* parece razonable concederte el mayor grado de libertad.

Al tener también activado el privilegio *+Subdirs* todos los directorios (de cualquier nivel) contenidos en c:\ServidorFTP gozarían de estos mismos privilegios. Por esa razón, ya no sería necesario especificar ninguno de ellos.

### *webmaster*

#### - En C:\ServidoresLocales\Apache\htdocs

**Todos.** Por idénticas razones al caso anterior.

### *secre*

Sus privilegios en los diferentes directorios podrían ser:

#### - En c:\ServidorFTP

**List.** Al concederle este privilegio podrían visualizarse (al acceder al *root* de este usuario mediante un cliente FTP) la lista de directorios y documentos que contiene.

Si no incluyéramos esta opción el cliente FTP nos daría un mensaje de error al acceder al *root* (no tendría permiso alguno) y obligaría al usuario *secre* a establecer como dirección del servidor alguno de los directorios a los que tiene permiso.

#### - En Documentación

**Todos.** Se entiende que este usuario es quien realiza la gestión completa de este directorio.

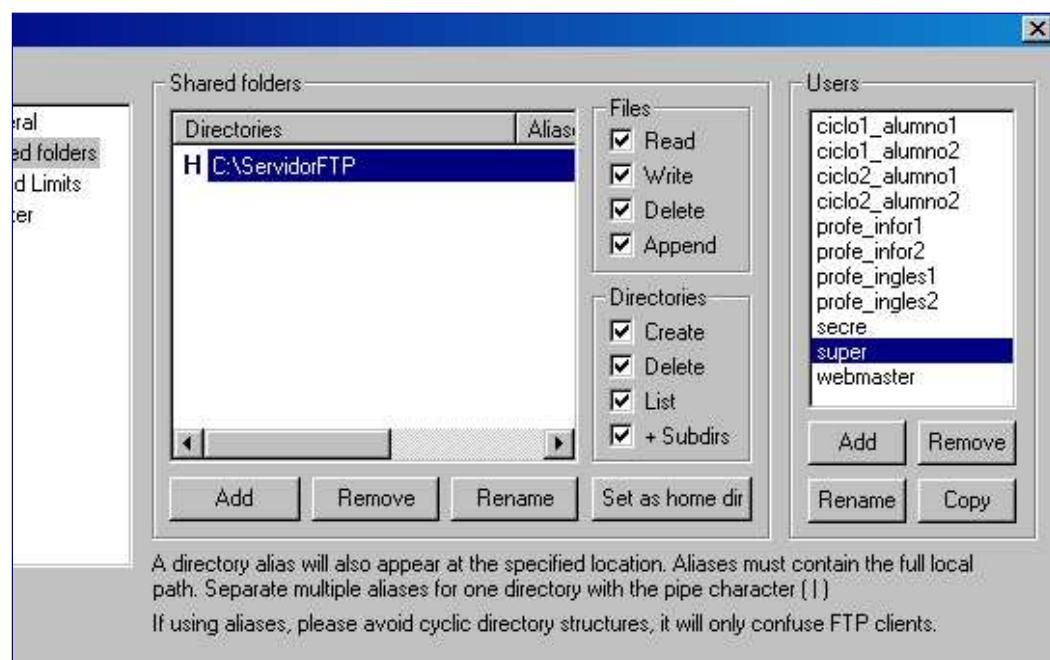
#### - En Gestión

**Todos.** Las razones son idénticas a las del caso anterior.

## Clients FTP

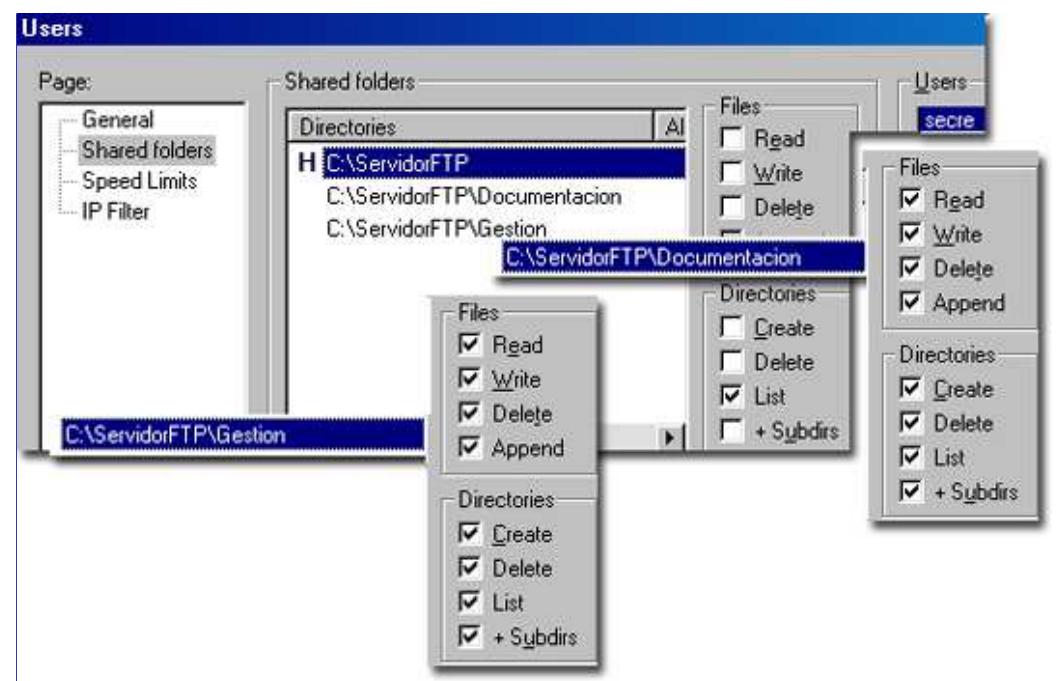
Para hacer una petición FTP, igual que en cualquier otro caso de petición, necesitamos disponer del software adecuado para realizarla. Recuerda que, en realidad, un

Esta es la configuración del usuario *super*.



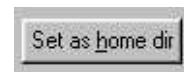
Para el usuario *webmaster* procederíamos de forma idéntica. La única diferencia sería el directorio (c:\ServidoresLocales\Apache\htdocs) sobre el que estableceríamos los mismos privilegios que se ven en la imagen para el usuario *super*.

## Directorios accesibles por del usuario *secre*



La razón de estas asignaciones las justificamos al margen.

Observarás que el directorio raíz está marcado con la letra **H**. Para cambiarlo bastará con seleccionar otro cualquiera y pulsar sobre el botón que ves en esta imagen.



## Acceso de usuarios

La dirección *localhost* apunta siempre hacia el directorio *root* de usuario. Eso quiere decir que si escribimos en el cliente FTP esa dirección y nos identificamos como *secre* (indicando la contraseña de usuario) veríamos algo como esto:

programa que se utiliza para realizar un determinado tipo petición a un servidor.

Existen varios clientes FTP en el mercado. El más popular de todos ellos es el WS\_FTP, que probablemente habrás usado –si has publicado alguna página web– para subir tus páginas al servidor.

Puedes descargar un cliente gratuito –con prestaciones muy similares a WS\_FTP– desde el sitio de FileZilla.

Las versiones más modernas de los navegadores también permiten realizar peticiones mediante este protocolo.

Si escribimos en la barra de direcciones del navegador –se requiere la versión 5 ó superior de IE–:

<ftp://super:super@localhost>

y un servidor FTP –con nombre *localhost*– está activo, veremos que aparecen en la ventana del navegador los iconos de los ficheros contenidos en el directorio root del servidor y que, a la vez, se nos ofrece la posibilidad de: borrar archivos; crear subdirectorios; copiar ficheros (desde el servidor a cualquier otro directorio de nuestro ordenador o viceversa) sin más que seguir métodos idénticos a los que se utilizan habitualmente en Windows.

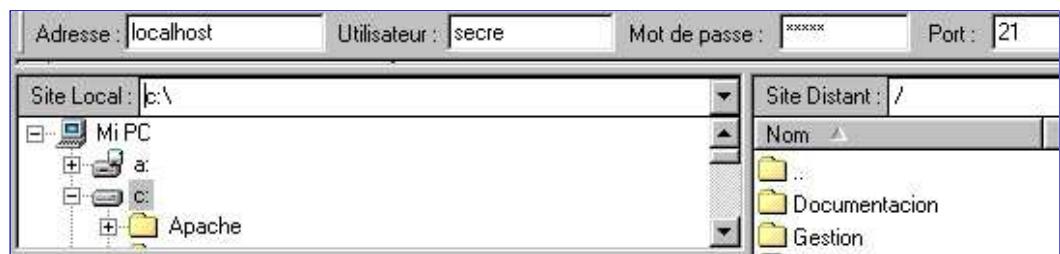
El acceso a un servidor FTP utilizando un navegador como cliente requiere tres datos: *nombre de usuario*, *nombre del servidor* y *contraseña*, que son los que aparecen en azul un poco más arriba.

En esa misma dirección aparecen –marcados en rojo–: (<ftp://>) que indica el tipo de protocolo que se utiliza en la petición; (:) cuya finalidad es la de actuar como separador entre el nombre de usuario y la contraseña; y, (@), que hace también función de separador, en este caso entre la contraseña y el nombre del servidor.

Pero, a riesgo de parecer reiterativos, queremos insistir en que para que una petición, como la que comentamos, pueda ser atendida se requiere, de forma imprescindible, que exista un servidor FTP activo.

Hay una posibilidad añadida. Mediante funciones de PHP también es posible gestionar servidores FTP sin necesidad de recurrir a ningún cliente específico. Lo trataremos en los contenidos de programación relativos a las funciones FTP.

## Grupos de usuarios



donde –como puedes observar– accedemos directamente al directorio C:\ServidorFTP (el root de este usuario) y visualizamos sus contenidos.

Si nuestro equipo estuviera conectado a una red de área local podríamos acceder al servidor FTP escribiendo en vez de localhost la dirección IP del equipo en el que tuvierámos instalado el servidor.

### ¡Cuidado!

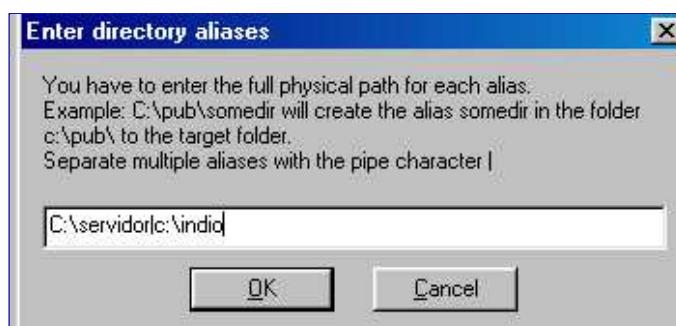
Es posible que el firewall de Windows no bloquee el acceso al servidor FTP sin darnos ningún mensaje de advertencia. Lo más conveniente para evitar problemas de esta índole sería abrir Paneles de Control -> Firewall de Windows -> Excepciones y, una vez allí, pulsar el botón Agregar Programas y añadir **Filezilla Server.exe**

## Utilización de alias

Este servidor también permite ser configurado para la utilización de *Alias*. Pulsando con el botón derecho del ratón sobre el nombre de uno de los directorios aparece un menú como este:



al elegir la opción *Edit Aliases* se abre una ventana como la que ves aquí debajo. Si introducimos en ella la ruta absoluta completa reemplazando el nombre del **último** directorio por una palabra distinta (en el ejemplo hemos incluido **web**) estaremos creando un alias. Podremos crear tanto como deseemos, es cuestión de incluirlos (completos) uno a continuación de otro separados por el carácter |.



Una vez creados los alias podremos utilizarlos para acceder a los espacios (sustituyendo el nombre del directorio por el alias). Aquí tienes imágenes del ejemplo. Observarás que **localhost/ServidoresLocales/Apache**, **localhost/servidor** y **localhost/indio** nos conducen al mismo sitio.



La configuración de grupos de usuarios es una opción que ofrece bastante interés. Mediante esta opción se pueden establecer privilegios comunes a una serie de usuarios. Esto facilita la configuración cuando se trata de grupos numerosos que van a compartir directorios y privilegios.

Los procesos de creación de grupos y de asignación de usuarios a cada uno de ellos los tienes descritos al margen.

### La opción *autocreate*

Supongamos que pretendemos que cada uno de los alumnos de nuestro supuesto *disponga* de un *subdirectorio* propio para poder *subir* y gestionar sus propios trabajos de cada materia.

Aparte de la ventaja de tener separadas sus actividades con la posibilidad de *borrar*, *añadir*, *modificar*, etcétera dentro de su *propio espacio* y a la vez *impedir* que pueda efectuar esos procesos en materiales ajenos.

Eso requeriría ir creando esos directorios para cada uno de los usuarios y esa podría ser una tarea lenta y pesada.

Mediante la opción *Autocreate* se puede configurar un directorio de forma que *durante el primer acceso* del usuario se cree de forma automática un subdirectorio con el mismo nombre que el del usuario.

El uso de esta opción requiere:

- Incluir en *Shared folders* la ruta del directorio base (en el que pretenden crearse los subdirectorios de cada uno de los usuarios pertenecientes al grupo) acabada con *:/*.

### Privilegios de los diferentes grupos de usuarios

En este enlace tienes el detalle y la justificación de los diferentes privilegios que hemos establecido para cada uno de los grupos de usuarios.

### Fichero de configuración de Filezilla Server

Dado que, por el número de usuarios y por la complejidad de la estructura, la labor de configuración y creación de los diferentes usuarios aquí propuestos puede resultarle lenta y tediosa hemos incluido un fichero que puedes encontrar en el directorio *cursophp* que puedes obtener desde aquí.

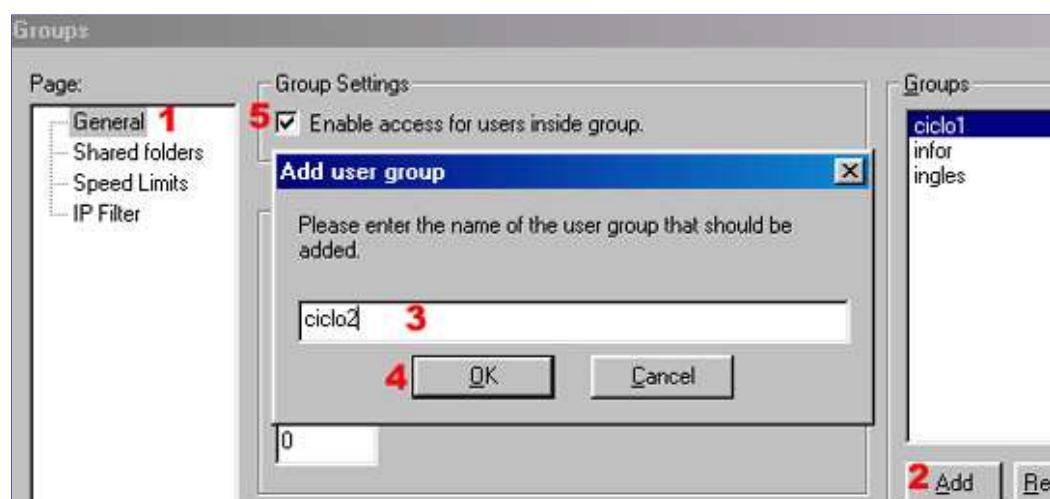


## Creación de grupos de usuarios

La creación de grupos de usuarios sigue un proceso muy similar al que hemos descrito para el caso de usuarios. Las diferencias más sustanciales son:

- Se accede a través del ícono *groups* (el quinto de la ventana *Server Interface*).
- No requiere insertar contraseñas. Estas serán privilegio exclusivo de los usuarios del grupo.
- Requiere activar la casilla de verificación *Enable access for users inside group*.

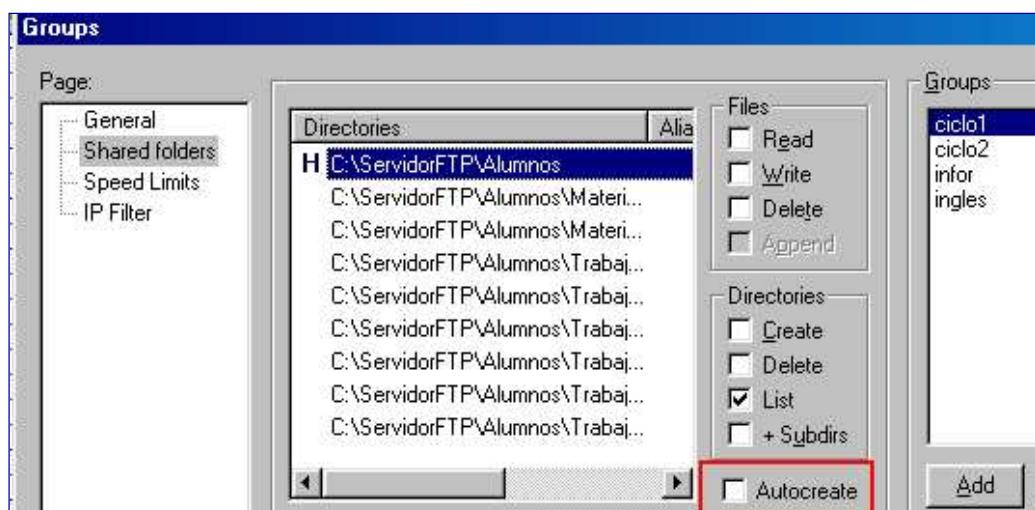
En nuestro ejemplo, crearemos cuatro grupos: *ingles* (grupo de los profesores de Inglés), *infor* (profesores de Informática), *ciclo1* (alumnos de primer ciclo) y *ciclo2* alumnos de segundo ciclo.



## Directarios accesibles y privilegios del grupo

La configuración de los directarios accesibles y de los privilegios en cada uno de ellos es idéntica a la descrita para el caso de usuarios no pertenecientes a un grupo.

La única diferencia estriba en que añade una nueva e interesante posibilidad a través de la opción *Autocreate*.



-FileZilla Server.xml- en el directorio de instalación de Filezilla Server (por defecto sería un directorio con ese mismo nombre dentro de Archivos de programa) sobrescribiendo el existente ya dispondrás de todos los usuarios y grupos aquí descritos junto con sus configuraciones respectivas.

### ¡Cuidado!

En el enlace que figura en el párrafo anterior y aquí tienes la configuración de los diferentes usuarios tal como se describen en esta página y en sus anexos.

### FTP y hosting

A la hora de la elección del *hosting*, otro de los factores que debemos tener en cuenta –o al menos conocer– son las opciones de FTP que ofrece cada uno de los proveedores de servicios.

Existen en la red situaciones de todo tipo. Hay casos en los que está activado el servicio y además es la única vía posible para poder *publicar* nuestras páginas.

Hay otros casos en los que está desactivado, como ocurre con el popular **geocities**, que **no permite** FTP y donde el *mantenimiento* de los espacios de alojamiento de páginas web de sus servidores requiere – de forma *ineludible* – acceder por medio de una página WEB específica que permite realizar las transferencias de ficheros por un *método alternativo* al **FTP** tradicional.

## Asignación de usuarios a un grupo

Una vez creado un grupo (o grupos) es necesario asignar los usuarios a ese grupo. El proceso es muy similar al de creación de usuarios no adscritos a ningún grupo. La única diferencia está en el punto (4) del proceso, ya que ahora hemos de elegir el grupo al que va a pertenecer el usuario (antes elegíamos *none*).

En el ejemplo hemos incluido dos usuarios por cada uno de los grupos:

- *profe\_ingles1* y *profe\_ingles2* en el grupo *ingles*.
- *profe\_infor1* y *profe\_infor2* en el grupo *infor*.
- *ciclo1\_alumno1* y *ciclo1\_alumno2* en el grupo *ciclo1*.
- *ciclo2\_alumno1* y *ciclo2\_alumno2* en el grupo *ciclo2*.

En todos los casos hemos incluido una contraseña *idéntica* al nombre de usuario.



Los miembros de un grupo recogen automáticamente todos los privilegios del *Shared Folder* del grupo al que pertenecen. No obstante, es posible añadir *nuevos directorios* y privilegios (añadidos a los específicos del grupo al que pertenecen) incluyéndolos en el *Shared Folder* del usuario.

### ¡Cuidado!

En el directorio **Extras** del CD-ROM hay una carpeta llamada **ServidorFTP** que contiene toda esta estructura de directorios. Puedes copiarla al directorio raíz de tu ordenador y ya tendrás la configuración descrita en la imagen de la columna de la izquierda.

Anterior

Índice

Siguiente

## Un servidor de correo

El servidor de correo Mercury Mail puede descargarse [www.pmail.com](http://www.pmail.com) o bien instalar directamente el fichero **m32-462.exe** que encontrarás en el directorio software de este CD.

## Finalidad de esta instalación

La instalación de este servidor de correo tiene un carácter puramente *experimental*.

Algunas funciones de PHP relacionadas con el envío de correo electrónico a través de una página web requieren disponer de un servidor de este tipo. Es la única forma en la que podremos ejecutar y comprobar el funcionamiento de los scripts que utilicen ese tipo de funciones.

## Modificación del fichero **php.ini**

Esta es una de las muchas modificaciones que tendremos que ir haciendo en el fichero **php.ini** a lo largo del curso. Recuerda que este fichero está en el directorio **Windows**.

Haremos de buscar las líneas que dicen:

[mail function]  
; For Win32 only.  
SMTP =

y cambiarlas por:

[mail function]  
; For Win32 only.  
SMTP = 127.0.0.1

y también  
; For Win32 only.  
;sendmail\_from=me@example

descomentando la última línea (quitando el punto y coma) y dejándola así:

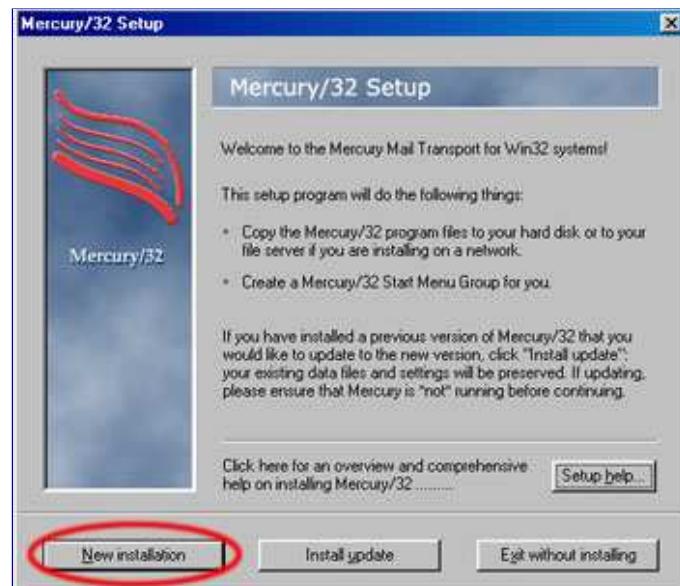
; For Win32 only.  
sendmail\_from=me@example

y sustituyendo *me@example* por *admin@mispruebas.com*.

## Proceso de instalación

Si no has descargado el programa de instalación desde el apartado *Software del curso* puedes hacerlo de este enlace de <http://www.pmail.com> eligiendo el fichero *Mercury mail transport system for win32 and NetWare systems v.4.62* o hacerlo directamente desde aquí.

Una vez descargado el programa, el proceso de instalación es el que describen las imágenes siguientes:



### ¡Cuidado!

Si has hecho la copia de

seguridad -que te hemos recomendado al configurar PHP- del fichero **php.ini** en c:\Apache\php4 deberías sustituirla por esta nueva versión modificada.

La finalidad no es otra que mantener la identidad de ambas copias.

## El proceso de instalación

La instalación, paso a paso la tienes descrita en las imágenes de la derecha. Los aspectos más significativos de este proceso son los siguientes:

– Elegir la opción **No NetWare Support**

– Elegir **No Pegasus Mail Integration**. De no hacerlo así nos obligaría a tener instalado -previamente- el *cliente* de correo de Pegasus Mail.

– Elegir las opciones **SMTP server Module**, **POP server module**, que serían las que nos permitieran enviar y recibir mensajes e modo local, y también **POP3 Client Module** que tiene relación con la recepción de **mensajes de cuentas externas**.

– La opción siguiente no tiene demasiada importancia para nuestros fines. Podemos elegir cualquiera de las dos sin que afecte sensiblemente al funcionamiento.

– Poner **127.0.0.1** donde dice: *this machine's Internet domain name* y dejar la opción por defecto **-admin-** donde dice: *user name for postmaster*.

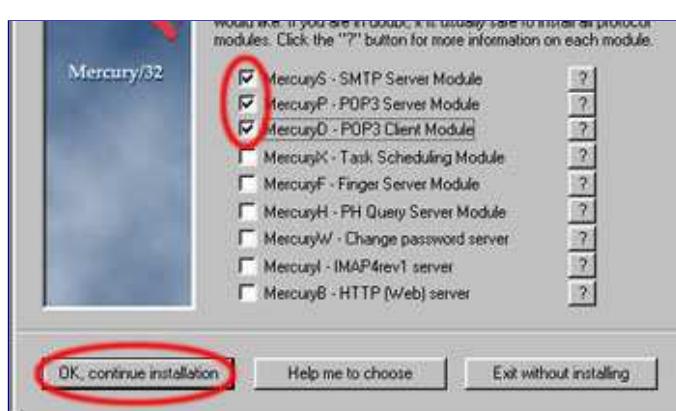
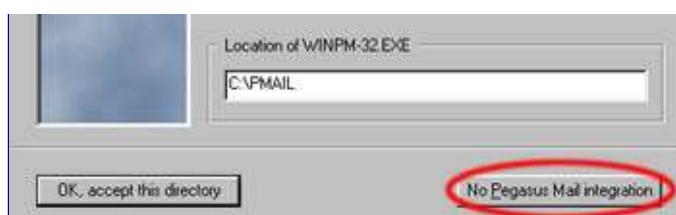
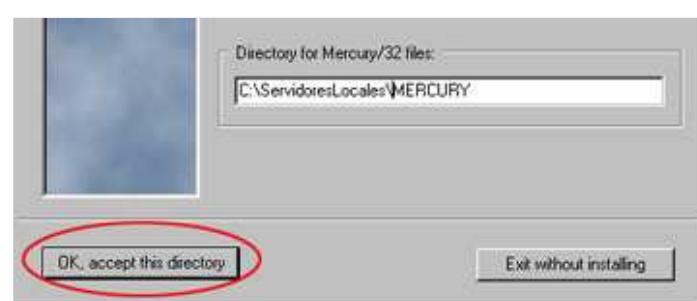
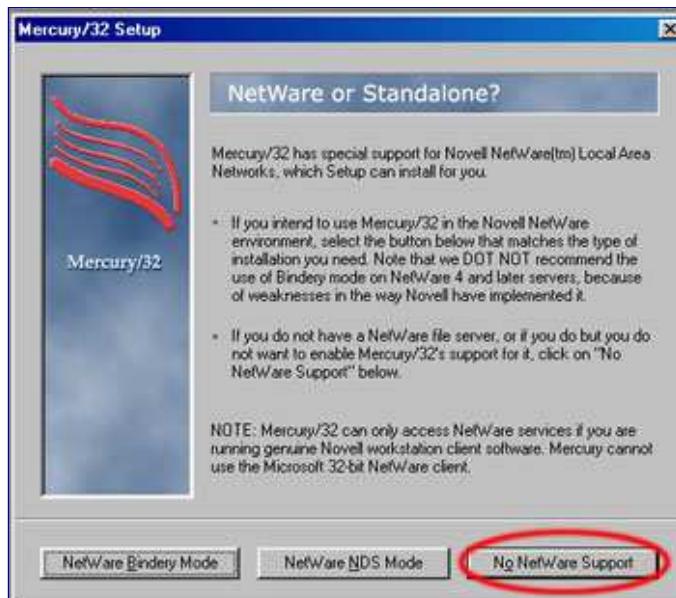
– Dónde dice: *Address of host via which to send mail* debes incluir la dirección del servidor SMTP a través del cual quieras que se envíen los mensajes a direcciones de correo externas.

En las pruebas hemos utilizado varias (los SMTP de nuestras cuentas de correo) aunque, como ves en la imagen, la prueba definitiva la hemos hecho utilizando la dirección *hermes.cnice.mecd.es*, que es el servidor SMTP para los usuarios de cuentas del CNICE.

Hemos optado por esa opción para desarrollar el proceso de configuración en su opción más compleja ya que, como sabes, ese servidor *requiere autenticación del usuario*.

El resto del proceso puede hacerse con las opciones por defecto hasta concluir el proceso de instalación.

## Configuración para una red de área local



Aunque hemos puesto la IP 127.0.0.1 pensando en servidor para pruebas, es posible que quieras utilizarlo como servidor de correo en una red local. En ese caso habrás de sustituir la IP por la correspondiente al ordenador en el que está instalado el servidor.

Para conocer esa IP, en el caso de que uses Windows98, bastaría con pulsar: **Inicio -> Ejecutar** y escribir en esa ventanita **winipcfg**. Si utilizas W2000, NT ó XP el proceso sería: **Inicio -> Programas -> Accesorios -> Símbolo del sistema** y una vez en esa ventana escribir **ipconfig**.

### Arrancar y parar el servidor

Para arrancar el servidor hay que ejecutar el programa **Mercury Loader**.

Lo encontrarás en el directorio en que hayas instalado el servidor de correo (por defecto, c:\Mercury).

Sabremos que está en marcha porque aparecerá una ventana nueva. Si la **minimizamos** aparecerá en la barra de tareas un ícono como este.



Este ícono solo aparece al minimizar la ventana. Si la cerramos se parará el servidor y desaparecerá el ícono.

A parte de la opción anterior, desde el menú del servidor tienes acceso a esta opción que ve en la imagen.

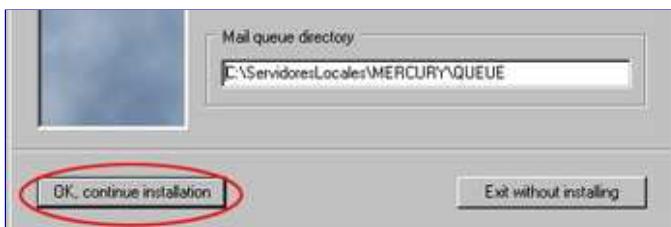
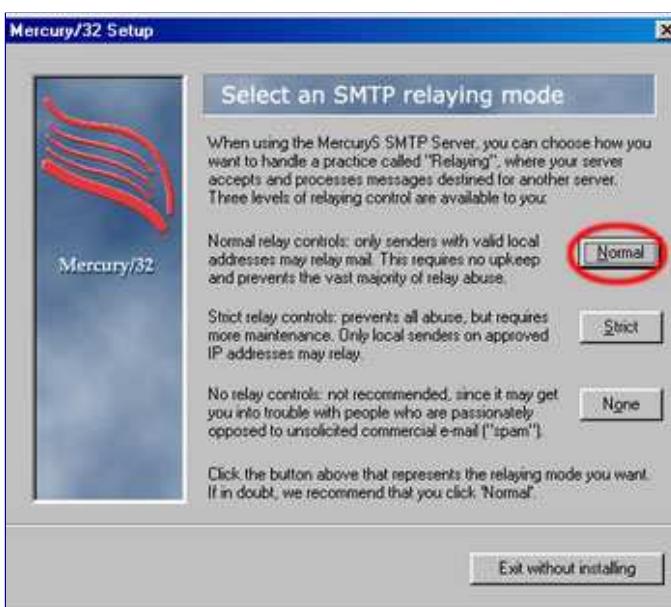
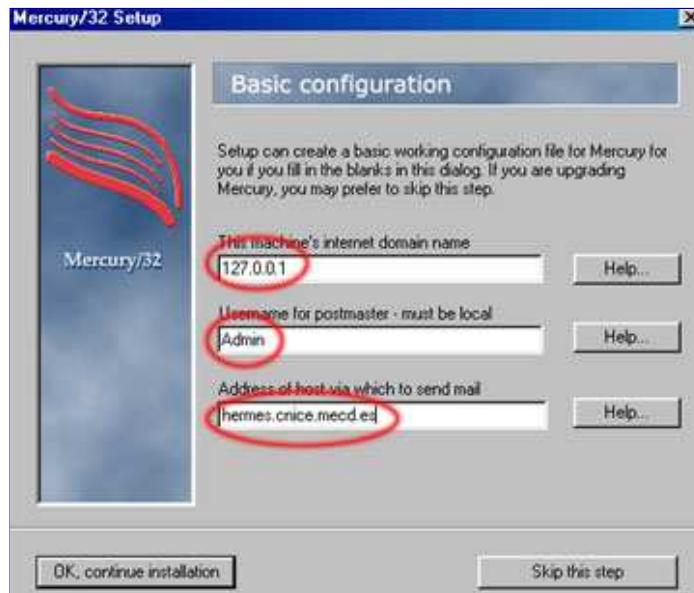
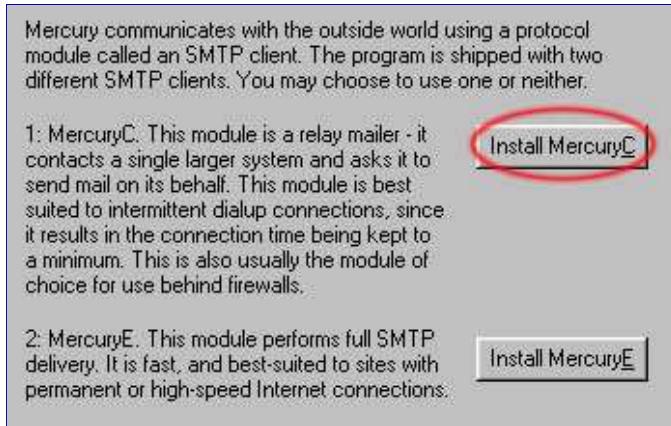


Cuando está funcionando el servidor -para pararlo- aparece un texto como este. Si estuviera parado, se podría activar desde la misma opción. Ahora aparecería con el texto: *Leave offline mode*

### Añadir usuarios

Después de arrancar debemos acceder -en el menú principal- a opción *Configuration* y una vez allí elegir *Manager local users*. Al abrirse esa ventana encontraremos el usuario *Admin* creado de forma automática durante el proceso de instalación.

Bastará con asignar un nombre de



### Crear cuentas de usuarios

cuenta, un nombre personal (no es imprescindible) y una contraseña.

Nosotros vamos a crear tres cuentas más. Añadiremos los usuarios *juan*, *perico* y *andres* y les pondremos contraseñas idénticas a los nombres respectivos.

### Crear un dominio local

Desde el menú del servidor, pulsando en *Configuration* y eligiendo *Mercury Core Module* nos aparece una ventana con varias pestañas tales como las que estás viendo.

En esta primera -donde dice General- no es preciso tocar nada. Está la dirección IP que habíamos introducido al instalar -127.0.0.1- y lo demás son los diferentes directorios de la instalación.

Al pulsar sobre la «pestaña» *Local domains* aparecen dos líneas que comienzan por 127. Podemos quitarlas desde el botón *Remove entry*.

Mediante el botón *Add New Domain* podemos insertar un nombre de dominio de nuestra elección.

Dado que al cambiar la configuración de php.ini hemos propuesto como dirección de correo *admin@mispuebas.com* usaremos como *localhost o server* **127.0.0.1** y asignaremos como *internet name* **mispuebas.com** tal como puedes ver en la imagen.

Como una segunda opción podremos asignar también *localhost* como internet name manteniéndole **127.0.0.1** como localhost o server.

### Configuración del cliente de correo

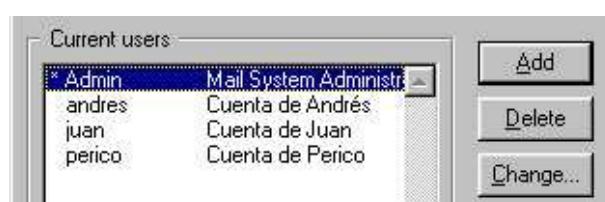
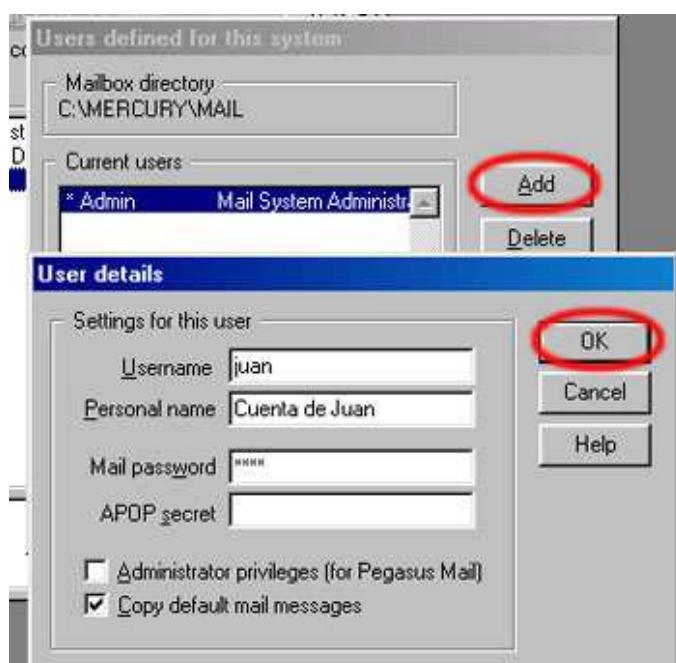
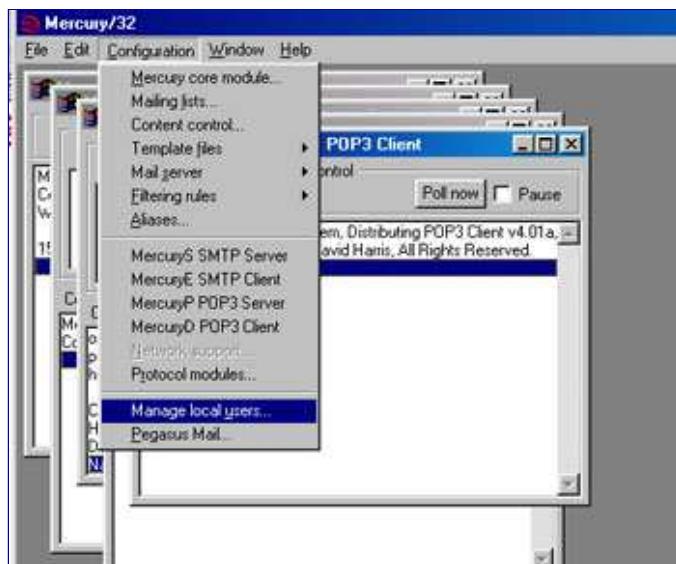
La configuración del cliente de correo no plantea problemas. Los únicos detalles a tener en cuenta son los relativos a la configuración de los servidores **SMTP** y **POP3**. En ambos casos se escribe la IP con la que hemos configurado el servidor, es decir: **127.0.0.1**

Como nombre de cuenta pondremos el mismo con el que las hemos creado (*juan*, *perico*, *andres*) sin añadir la @ ni el nombre del dominio.

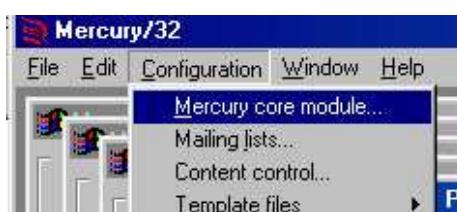
Si la instalación se realiza para trabajar en una red de área local, tendrás que cambiar el 127.0.0.1 por la IP del ordenador en el que está instalado el servidor.

### Prueba del servidor

Una vez configuradas las cuentas en el cliente de correo bastará con



### Configuración básica



enviar mensajes a las direcciones de usuarios locales (con el servidor Mercury activo) y comprobar que son recibidos en las cuentas destinatarias.

## Prueba desde PHP

Si ejecutar el script de prueba que tienes a la derecha (debes tener activos los servidores Apache y Mercury) deberá aparecerte una página con el texto: *Mensajes enviados con éxito.*

Será la prueba inequívoca de tanto el servidor con la configuración de PHP son las correctas. Si después abres el cliente de correo podrás comprobar que los usuarios *juan*, *perico* y *andres* han recibido ese mensaje.

## Leer correo externo

Aunque no forma parte de los contenidos de este curso, puede resultarte interesante configurar el servidor de correo de forma que puedan recibirse en una cuenta local los mensajes enviados a una -o varias- cuenta externa.

Tal como ves en las imágenes sólo debes acceder opción *Configuration* y seleccionar *MercuryD POP3 client*. Aparecerá una ventana como la que ves al margen.

Los pasos a seguir están numerados en la imagen.

El primero será establecer el *Check Every* (frecuencia con la que se comprueba la existencia de nuevos mensajes en la cuenta externa). Por defecto se autoconfigura en 30 segundos. Ahí podrás indicar el periodo de tiempo (en segundos) que estimes oportuno.

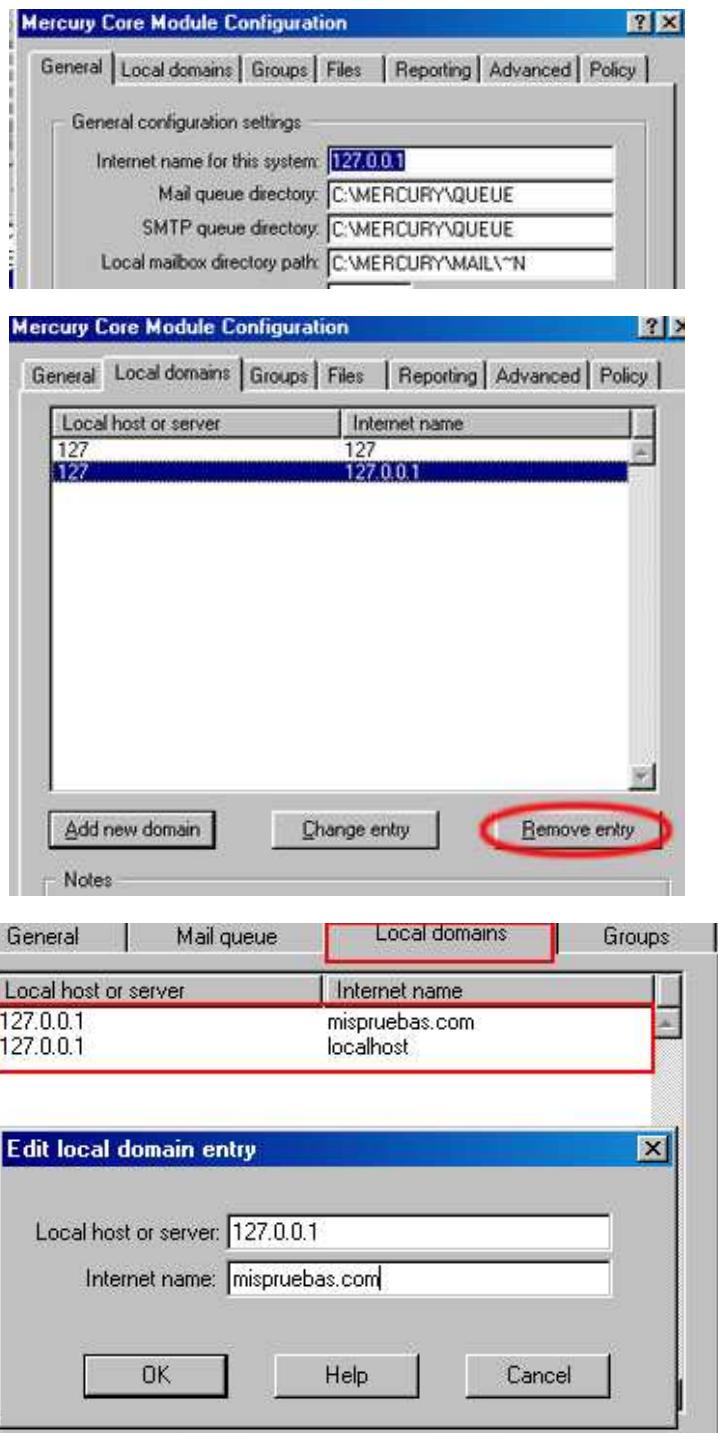
El segundo paso es pulsar el botón *Add* que abrirá la ventana que ves superpuesta en la imagen.

Debemos escribir el nombre de una cuenta externa, el nombre de su servidor POP3 y la contraseña de esa cuenta de correo, y asociarlas con una cuenta local. Después de pulsar el *OK* se cierra la ventana superior y deberemos pulsar el botón *Save* para guardar los cambios.

A partir de ese momento –siempre que el servidor de correo esté activo y tengamos abierta una conexión a internet– se comprobará la cuenta externa con la frecuencia indicada y si existieran mensajes serían transferidos a la cuenta local especificada en la configuración.

## Enviar mensajes a direcciones externas

Mercury permite enviar mensajes a



## Un script de prueba

Este es el código fuente de un script que nos permitirá comprobar si hemos configurado correctamente el servidor de correo. El contenido que aparezca en la página nos dirá lo que ha ocurrido. ¡No te preocupes si aún no entiendes el código! Ya hablaremos de él más adelante.

```
<?
if( mail("juan@mispruebas.com", "Una prueba definitiva","Bienvenid@ a PE
"From: Administrador de mispruebas.com <admin@mispruebas.com>
Reply-To: juan@mispruebas.com
Cc: perico@mispruebas.com
Bcc: andres@mispruebas.com
X-Mailer:PHP/" . phpversion())) {
    print "Mensajes enviados con exito";
} else{
    print "Se ha producido un error";
}
```

través del servidor configurado como *Address of host via which to send mail* ) a direcciones de correos correspondientes a dominios externos.

La configuración de esa opción requiere el proceso que ves a la derecha.

Eligiendo la opción *MercuryS SMTP Server* se abre una ventana como la que ves en la imagen. Activaremos la «pestaña» *Connection control* y desactivaremos la casilla de verificación que dice: *Do not permit SMTP relaying of non-local mail* y con ello daremos al servidor la opción de enviar a través del servidor SMTP externo los mensajes cuyo nombre de dominio no coincide con el configurado para el servidor local.

Aún hemos de solventar un pequeño problema en esta configuración. Al enviar mensajes desde una cuenta local hacia una cuenta externa se incluiría en el mensaje la dirección local como dirección de respuesta. Eso, obviamente, plantearía problemas al destinatario ya que sus respuestas no encontrarían ese dominio en la red.

El problema se resuelve modificando la configuración del *cliente de correo* e incluyendo como *dirección de respuesta* una cuenta externa. De esta forma, combinando esta configuración con la de lectura de cuentas externas podríamos gestionar desde nuestra cuenta local el envío y recepción de mensajes externos.

### Servidores SMTP que requieren autentificación

Cada día son más habituales los servidores de correo SMTP que *requieren autentificación*. Este es el caso, tal como comentamos más arriba, de las cuentas del CNICE.

En esas circunstancias es necesario acceder a la opción *Mercury SMTP Client* que nos abrirá una ventana como que la ves al margen. Bastará con llenar los campos correspondientes a *Credentials for SMTP Authentication, if required* poniendo en *Username* y *password* los mismos valores que usamos en la configuración de esa cuenta en el cliente de correo.

Una vez hecho esto, sólo queda guardar los cambios y el servidor estará listo para efectuar este tipo de envíos.

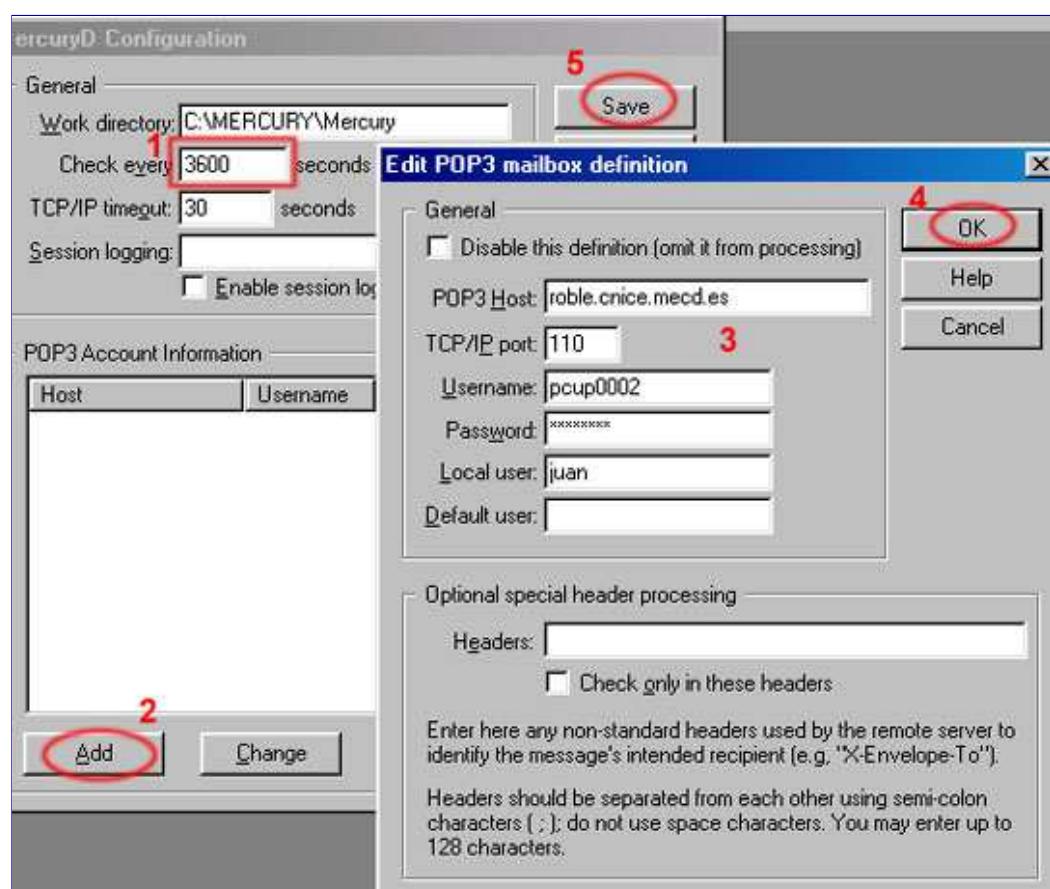
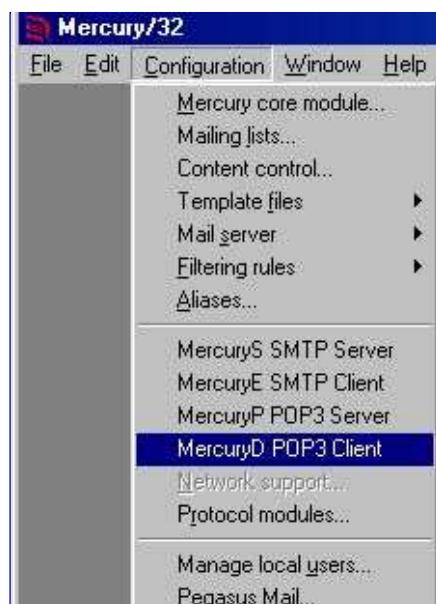
### Desinstalación del servidor de correo

La instalación de Mercury *no escribe*

?>

[Probar servidor de correo](#)

## Leer mensajes de cuentas externas

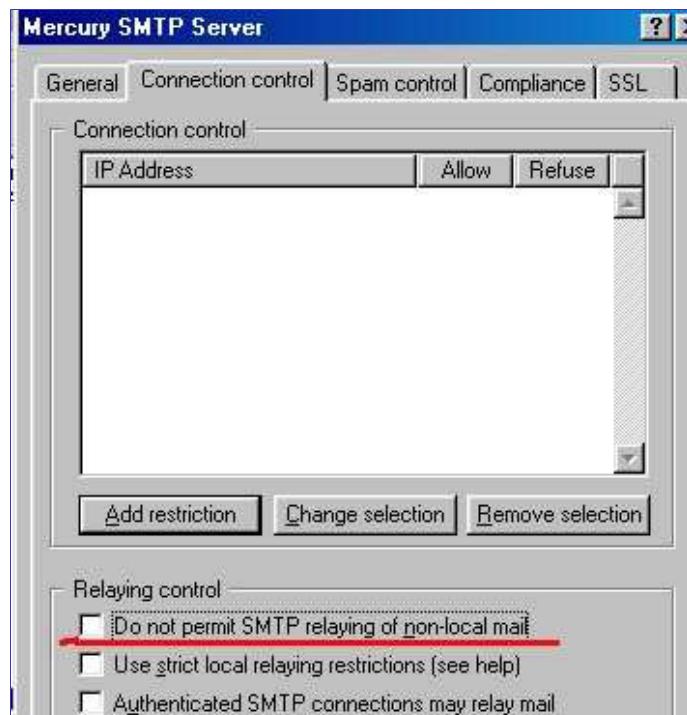
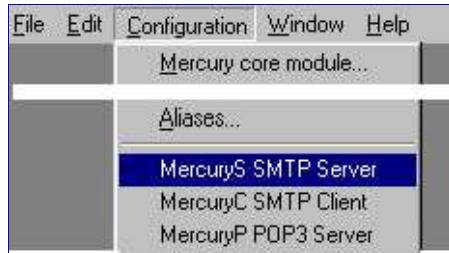


## Enviar mensajes a cuentas externas

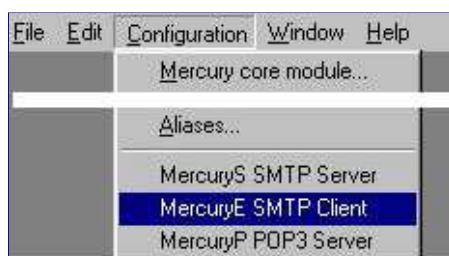
ello, el proceso de desinstalación no existe. Basta con *borrar* del directorio de instalación y habremos desinstalado el servidor.

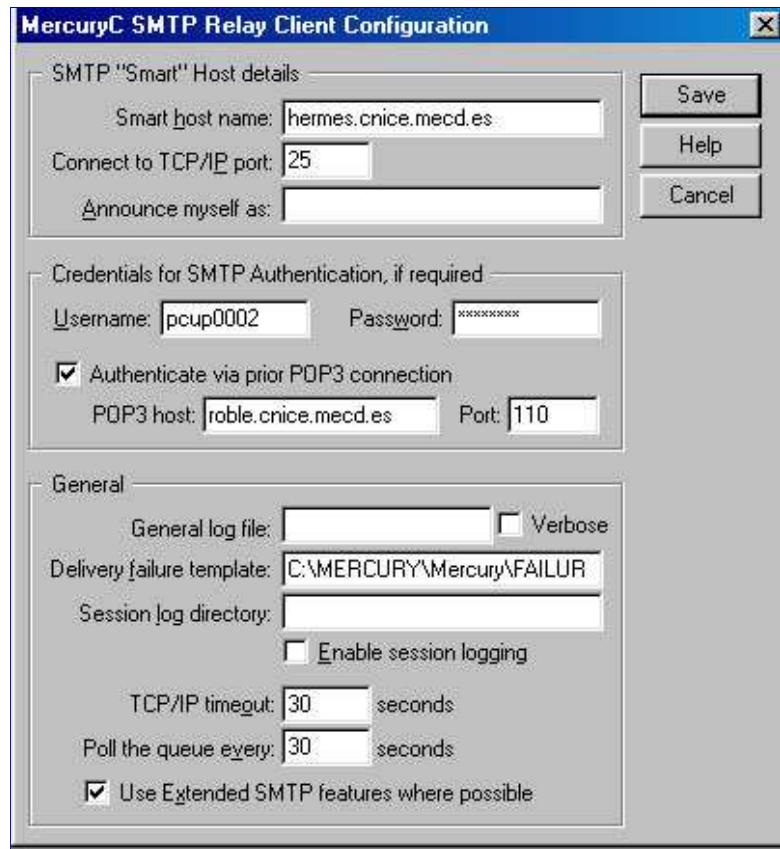
### ¡Cuidado!

La gestión de mensajes externos puede plantear problemas derivados de los filtros antispam de los servidores de correo de la red que pueden bloquear el envío o recepción de los mensajes enviados utilizando este servidor.



Nombre:	Juan Pruebas
Organización:	
Dirección de correo electrónico:	juan@mispruebas.com
Dirección de respuesta:	pcup0002@roble.cnice.mecd.es





---

Anterior



Índice



Siguiente



[Ver índice](#)

# PHP y HTML



## Páginas PHP

Las páginas PHP pueden ser páginas web *normales* a las que se cambia la extensión, poniendo **.php** en vez de **.htm** ó **.html**.

En una página cuyo nombre tenga por extensión **.php** se pueden insertar instrucciones –escritas en lenguaje PHP– anteponiendo **<?** a la primera instrucción y escribiendo des-pués de la última **?>**.

A cada uno de estos *bloques de instrucciones* le llamaremos un **script**.

No existe límite en cuanto al número de *scripts* distintos que pueden insertarse dentro de una página.

## Un poco de sintaxis

La primera instrucción PHP que conoceremos será esta:

```
echo "un texto..";
```

La instrucción **echo** seguida de un *texto entrecomillado* hará que el PHP escriba en la página web resultante lo contenido en esa cadena de texto.

Al final de cada instrucción debemos insertar siempre un punto y coma (**;**)

El (**;**) indicará a PHP que lo que viene a continuación es una nueva instrucción.

Para facilitar la depuración los *scripts* no suelen escribirse dos instrucciones en una misma línea.

```
print "un texto..";
```

La instrucción **print** tiene una función similar –no es exactamente la misma– a la descrita para **echo**.

```
print ("un texto..");
```

Esta es otra posibilidad –la más habitual– de utilizar **print**. Aquí encerramos entre paréntesis la cadena que pretendemos que aparezca impresa en la página web.

El hecho de que utilicemos paréntesis no nos evita tener que encerrar la *cadena* (texto) a imprimir entre comillas.

## Comillas dentro de comillas

## La primera página en PHP

Observemos este código fuente. Como verás, se trata de una página web muy simple que **no contiene ningún script PHP**.

Hemos guardado esa página con el nombre **ejemplo4.html** y luego la hemos vuelto a guardar –sin modificar nada en sus contenidos– como **ejemplo4.php**.

```
<html>
<head>
<title>Aprendiendo PHP</title></head>
<body>
Esta es una página supersimple
</body>
</html>
```

[Ver ejemplo4.html](#)[Ver ejemplo4.php](#)

Si visualizamos ambos ejemplos veremos que los resultados son idénticos.

## Los primeros script PHP

Editemos la página anterior (**ejemplo4.php**) y añadámosle nuestra primera etiqueta PHP guardándola como **ejemplo5.php**. Este sería el *código fuente*:

```
<html>
<head>
<title>Aprendiendo PHP</title></head>
<body>
Esta es una página supersimple
<?
echo "¿Aparecerá esta linea?";
?>
</body>
</html>
```

[ejemplo5.php](#)

Veamos ahora un ejemplo con las diferentes opciones de uso de las comillas

```
<html>
<head>
<title>Aprendiendo PHP</title></head>
<body>
<?
/* Las instrucciones PHP son las que aparecen en rojo.
   Las etiquetas en azul intenso son el código HTML.
   Todo lo que aparece en este color son líneas de comentario
   de las que hablaremos más adelante
   Cuando rescribas estos primeros scripts
   bastará que incluyas las instrucciones escritas en rojo */
/* ponemos <br> al final del texto para que cuando se
```

[Ver índice](#)[Búsqueda rápida](#)[Página anterior](#)[Página siguiente ►](#)

```
aparecerá en una línea diferente */
# aquí utilizamos solo unas comillas
echo "Este texto solo lleva las comillas de la instrucción<br>";
# aquí anidaremos comillas de distinto tipo
```

Existen dos tipos de comillas: dobles « " » (SHIFT+2) y sencillas ' ' (tecla ? en minúsculas).

Cuando es preciso *anidar* comillas deben utilizarse *tipos distintos* para las exteriores y para las interiores.

Para que una etiqueta echo interprete *unas comillas* como **texto** –y no como el final de la cadena– es necesario **antepone-nerles** un signo de *barra invertida*(\).

En ningún caso –ni con echo ni con print– está permitido sustituir las comillas exteriores (las que encierran la cadena) por \". Esta sintaxis solo es válida para indicar a PHP que debe interpretar las comillas como un carácter más.

En la página siguiente veremos las diferentes opciones de las *líneas de comentarios*.

Al realizar el ejercicio que te proponemos no es necesario que pongas los comentarios del ejemplo.

```
echo "La palabra 'comillas' aparecerá entrecomillada<br>";
# esta es otra posibilidad invirtiendo el orden de las comillas
echo 'La palabra "comillas" aparecerá entrecomillada<br>';
# una tercera posibilidad en la que utilizamos un mismo
# tipo de comillas. Para diferenciar unas de otras anteponemos
# la barra invertida, pero esta opción no podríamos utilizarla
# al revés.
# No podríamos poner \" en las comillas exteriores.
echo "La palabra \"comillas\" usando la barra invertida<br>";
?>
</body>
</html>
```

[Ver ejemplo6.php](#)

## Ejercicio nº 2

Crea un subdirectorio en el root de tu servidor Apache (**htdocs** si usas Windows, **/var/www** si utilizas Ubuntu) y ponle como nombre **prácticas**.

Escribe con tu editor un script similar al **ejemplo6.php** utilizando **print** y **print()** en vez de **echo**. Guarda el documento en la carpeta **prácticas** con el nombre **ejercicio2.php** y comprueba su funcionamiento.

Recuerda que para visualizarlo deberás escribir en tu navegador:

**<http://localhost/practicas/ejercicio2.php>**

[Anterior](#)

[Índice](#)

[Siguiente](#)

[Ver índice](#)

## Líneas de comentario



### ¿Por qué usar líneas de comentario?

A primera vista pueden parecer inútiles. ¿Para qué recargar las páginas con contenidos que no se van a ver ni ejecutar?

Las líneas de comentario sirven para poder recordar en un futuro qué es lo que hemos hecho al escribir un script y por qué razón lo hemos hecho así.

A medida que vayamos avanzando verás que en muchos casos tendremos que aplicar estrategias individuales para resolver cada problema concreto.

Cuando necesites hacer una corrección o una modificación al cabo de un tiempo verás que confiar en la memoria no es una buena opción. Es mucho mejor utilizar una línea de comentario que confiar en la memoria. ¡Palabra!

### Comentarios

Para insertar comentarios en los scripts de PHP podemos optar entre varios métodos y varias posibilidades:

#### • Una sola linea

Basta colocar los símbolos `//` al comienzo de la línea o detrás del punto y coma que señala el final de una instrucción.

También se puede usar el símbolo `#` en cualquiera de las dos posiciones.

#### • Varias líneas

Si un comentario va a ocupar más de una línea podremos escribir `/*` al comienzo de la primera de ellas y `*/` al final de la última. Las líneas intermedias no requieren de ningún tipo de marca.

Los comentarios para los que usemos la forma `/* ... */` no pueden anidarse. Si, por error, lo hiciéramos, PHP nos dará un mensaje de error.

### Ensayando líneas de comentario

En este ejemplo hemos incluido –marcados en rojo– algunos ejemplos de inserción de líneas de comentario.

```
<HTML>
<HEAD>
<TITLE>Ejemplo 7</TITLE></HEAD>
<BODY>
<?
// Este comentario no se verá en la página

echo "Esto se leerá <BR> "; // Esto no se leerá

/* Este es un comentario de
múltiples líneas y no se acabará
hasta que no cerremos así.... */
```

[Ver índice](#)[Búsqueda rápida](#)[Página anterior](#)[Página siguiente](#)

```
# Este es un comentario tipo shell que tampoco se leerá
# Este, tampoco

echo ("Aquí el tercer texto visible"); #comentario invisible

/* Cuidado con anidar
/* comentarios
multilinea con estos*/
al PHP no le gustan */

?>

</body>
</html>
```

[Ver ejemplo7.php](#)[Ver ejemplo7a.php](#)

Ejecutemos los dos ejemplos. En el **ejemplo7a** hemos quitado el `/*` que va delante de «comentarios» y el `*/` que va después de «multilinea con estos» (marcado en magenta en el código fuente) y funciona perfectamente.

En el caso del **ejemplo7** hemos mantenido el código exactamente como se muestra aquí arriba. Al ejecutarlo **nos dará un error**. Esto es una muestra, de la importancia que tiene el evitar **anidar** los comentarios

### Ejercicio nº 3

Escribe un script en el que se utilicen las funciones `print`, `print()` y `echo` añadiendo –con los diferentes formatos– líneas de comentario que expliquen la sintaxis de cada una de las funciones. Guárdalo como **ejercicio3.php** en el directorio **prácticas** y comprueba su funcionamiento.

[Anterior](#)[Índice](#)[Siguiente](#)



Ver índice

# Constantes



## ¿Qué es una constante?

Una **constante** es un valor –un número o una *cadena*– que **no va a ser modificado** a lo largo del proceso de ejecución de los *scripts* que contiene un documento.

Para mayor comodidad, a cada uno de esos valores se le **asigna un nombre**, de modo que cuando vaya a ser **utilizado baste** con escribir su nombre.

Cuando *ponemos nombre* a una **constante** se dice que **definimos esa constante**.

## ¿Cómo definir constantes?

En **PHP** las constantes se definen mediante la siguiente instrucción:

```
define("Nombre","Valor")
```

Los valores asignados a las constantes se mantienen en todo el documento, *incluso* cuando son invocadas desde *una función*.

No es necesario escribir *entre comillas* los valores de las constantes cuando se trata de constantes **numéricas**.

Si se realizan operaciones aritméticas con constantes tipo **cadena**, y su valor **comienza por una letra**, PHP les asigna **valor cero**.

Si una **cadena empieza por uno o varios caracteres numéricos**, al tratar de operarla aritméticamente PHP considerará **únicamente** el valor de los **dígitos anteriores a la primera letra o carácter no numérico**.

El **punto entre caracteres numéricos** es considerado como **separador de parte decimal**.

Tal como puedes ver en el **código fuente** del ejemplo que tienes al margen, es posible **definir constantes** a las que se **asigne como valor el resultado de una operación aritmética**.

## Ampliando echo

Mediante **una sola instrucción echo** se pueden **presentar** (en la ventana del navegador del cliente) de forma **simultánea** varias *cadenas de caracteres v/o constantes v*

## Un ejemplo con constantes

```
<HTML><HEAD><TITLE>Constantes</TITLE></HEAD>
<BODY>
<?

/* Definiremos la constante EurPta y le asignaremos el valor 166.386 */
define("EurPta",166.386);
/* Definiremos la constante PtaEur asignándole el valor 1/166.386
   En este caso el valor de la constante es el resultado
   de la operación aritmética dividir 1 entre 166.386*/
define("PtaEur",1/166.386);
/* Definimos la constante Cadena y le asignamos el valor:
   12Esta constante es una cadena*/
define("Cadena","12Esta constante es una cadena");
/* Definimos la constante Cadena2 y le asignamos el valor:
   12.54Constante con punto decimal*/
define("Cadena2","12.54Constante con punto decimal");

/* Comprobemos los valores.
   Observa la nueva forma en la que utilizamos echo
   Lo hacemos enlazando varias cadenas separadas con
   punto y/o coma, según se trate de echo o de print */

echo "Valor de la constante EurPta: ", EurPta, "<BR>";
echo "Valor de la constante PtaEur: ". PtaEur . "<BR>";
print "Valor de la constante Cadena: " . Cadena . "<BR>";
print "Valor de la constante Cadena x EurPta: " . Cadena*EurPta . "<br>";
print "Valor de la constante Cadena2 x EurPta: " . Cadena2*EurPta . "<br>";

echo "Con echo los números no necesitan ir entre comillas: " , 3, "<br>";
print "En el caso de print si son necesarias: " . "7" . "<br>";
print ("incluso entre paréntesis necesitan las comillas: "."45"."<br>");
print "Solo hay una excepción en el caso de print. ";
print "Si los números van en un print independiente no necesitan comillas
print 23;

# Pondremos la etiqueta de cierre del script y escribiremos
# una línea de código HTML
?>

<br>Ahora veremos los mismos resultados usando la function prueba<br><br>
<?

# Estamos dentro de un nuevo script abierto por el <? anterior

/* Aunque aún no la hemos estudiado, escribiremos una función
   a la que (tenemos que ponerle siempre un nombre)
   vamos a llamar prueba()
   Lo señalado en rojo es la forma de indicar el comienzo
   y el final de la función
   Lo marcado en azul son las instrucciones
   que deben ejecutarse cuando la función prueba()
   sea invocada */

function prueba(){
    echo "Valor de la constante EurPta: ". EurPta . "<BR>";
    print "Valor de la constante PtaEur: ". PtaEur. "<BR>";
    echo "Valor de la constante Cadena: ", Cadena , "<BR>";
    print ("Valor de la constante Cadena x EurPta: " .
          Cadena*EurPta . "<br>"); 
    print ("Valor de la constante Cadena2 x EurPta: " .
          Cadena2*EurPta . "<br>"); 
}

# Las funciones solo se ejecutan cuando son invocadas
```

*variables*. Basta con **ponerlas** una a continuación de otra utilizando **una coma** como *separador* entre cada una de ellas.

La forma anterior no es la única –ni la más habitual– de enlazar elementos mediante la instrucción **echo**. Si en vez de utilizar la **coma** usáramos un **punto** (el *concatenador* de cadenas) conseguiríamos el mismo resultado.

Cuando *enlazemos* elementos distintos –cadenas, constantes y/o números– hemos de tener muy en cuenta lo siguiente:

- Cada una de las sucesivas **cadenas** debe ir encerrada entre sus **propias** comillas.
- Los **nombres** de constantes **nunca** van entre comillas.

### Ampliando **print**

Las instrucciones **print** también permiten concatenar cadenas en una misma instrucción.

En este caso **solo es posible** usar el **punto** como elemento de unión. Si pusiéramos *comas* –como hacíamos con **echo**– PHP nos daría un **error**.

```
/* La función anterior no se ejecutará hasta que escribamos
   una línea –como esta de abajo– en la que ponemos
   únicamente el nombre de la función: prueba()
   */
?>

<?
prueba() ;
?>
</body>
</HTML>
```

[Ver ejemplo8.php](#)

### Ejercicio nº 4

Escribe un script (guárdalo como **ejercicio4.php**) en el que se definan dos constantes, una numérica y otra de cadena y en el que –mediante la las opciones **print** y **echo**– aparezca en la página web resultante un comentario sobre el tipo de cada una de ellas seguido de su valor. Intenta conseguir que los elementos concatenados aparezcan en la presentación separados por un espacio. ¡Observa el código fuente!

### ¡Cuidado!

Presta mucha atención a la sintaxis. Olvidar los «;» o no poner unas comillas suelen ser la causa de muchos mensajes de error.

Anterior



Índice



Siguiente





Ver índice

# Variables



## ¿Qué es una variable?

Podríamos decir que es un espacio de la *memoria RAM* del ordenador que se reserva –a lo largo del tiempo de ejecución de un script– para almacenar un determinado tipo de datos cuyos valores son susceptibles de ser modificados por medio de las instrucciones contenidas en el propio programa.

## Nombres de variables

En PHP todos los nombres de variable tienen que empezar por el símbolo \$.

Las nombres de las variables han de llevar una letra inmediatamente después del símbolo \$. \$pepe1 es un nombre válido, pero \$1pepe no es un nombre válido–.

Para PHP las letras mayúsculas y las minúsculas son distintas. La variable \$pepe es distinta de \$Pepe.

## Tipos de variables

En PHP no es necesario definir el tipo de variable, por lo tanto, una misma variable puede contener una cadena de caracteres en un momento del proceso y, posteriormente, un valor numérico, susceptible de ser operado matemáticamente.

## Definición de variables

PHP no requiere una definición previa de las variables. Se definen en el momento en que son necesarias y para ello basta que se les asigne un valor.

La sintaxis es esta:

**\$variable=valor;**

El valor puede ser una cadena (texto o texto y números que no requieren ser operados matemáticamente) o sólo un número. En el primero de los casos habría que escribir el valor entre comillas.

## Ámbito de las variables

Los valores de una variable definida en cualquier parte de un script –siempre que no sea dentro de una función– pueden ser utilizados desde cualquier otra parte de ese script, excepto desde dentro de las funciones que contuviera el propio

## Practicando con variables y sus ámbitos

Podemos comparar la memoria de un ordenador con el salón de un restaurante y la ejecución de un programa con los servicios que van a darse en la celebración del final de año. La forma habitual de hacer una reserva de mesa –espacio de memoria– para ese evento sería facilitar un nombre –nombre de la variable– y especificar además cuantos comensales –tipo de variable– prevemos que van a asistir.

Cuando acudimos a la cena de San Silvestre podremos sentarnos en esa mesa un número determinado de comensales –daremos un valor a la variable– y a lo largo de ella podremos levantarnos o incorporar un nuevo invitado –modificación del valor de la variable– siempre que sea alguien de nuestro ámbito quien realice la invitación.

Probablemente no permitiríamos que el cocinero decidiera quien debe sentarse o levantarse, pero si lo permitiríamos a cualquiera de nuestros invitados. La diferencia estaría –ámbito de la variable– en que el cocinero no pertenece a nuestro ámbito mientras que los invitados a nuestra mesa sí.

Quizá si celebráramos el evento otro día cualquiera no necesitaríamos hacer una reserva previa y bastaría con acudir a la hora deseada y hacer la reserva justo en el momento de sentarse.

El restaurante de PHP no necesita que hagamos ninguna reserva previa. Otros muchos lenguajes de programación, por el contrario, si la necesitan.

Siguiendo con lo que nos ocupa, aquí tienes un ejemplo del uso de las variables y la forma de utilizarlas en los diferentes ámbitos.

```
<HTML>
<HEAD>
</HEAD>
<BODY>
<?
# Definimos la variable $pepe como vacía
$pepe="";

# Definimos las variables $Pepe y $Pepa (ojo con mayúsculas y minúsculas)
$Pepe="Me llamo Pepe y soy serio y formal";
$Pepa="Me llamo Pepa y también soy seria y formal";

?>

<!-- esto es HTML, hemos cerrado el script -->

<center><b>Vamos a ver el contenido de las variables</b></center>

<!-- un nuevo script PHP -->

<?
echo "<br> El valor de la variable pepe es: ",$pepe;
echo "<br> No ha puesto nada porque $pepe esta vacía";
echo "<br> El valor de la variable Pepe es: ",$Pepe;
?>

<center><b><br>Invocando la variable desde una función</b></center>

<?
/* Escribiremos una function llamada vervariable
Observa la sintaxis. La palabra function delante
y el () al final seguidos de la llave.
Hasta que no cerremos la llave todas las líneas
serán consideradas parte de la función */

function vervariable(){
echo "<br> Si invoco la variable Pepe desde una función";
echo "<br>me aparecerá en blanco";
}
```

script o desde las que pudieran estar contenidas en un **fichero externo**.

Si una variable es definida **dentro de una función** sólo podrá ser utilizada dentro esa función.

Si en una **función** aludimos a una **variable externa** a ella PHP considerará esa llamada como si la variable tuviera valor **cero** (en caso de ser tratada como número) o una **cadena vacía** ("") es una cadena vacía).

Igual ocurriría si **desde fuera** de una función hiciéramos alusión a una variable definida en ella.

Si definimos dos variables con el mismo nombre, una dentro de una función y otra fuera, PHP las considerará distintas. La función utilizará –cuando sea ejecutada– sus propios valores sin que sus resultados modifiquen la variable **externa**.

## Variables globales

Lo comentado anteriormente, admite algunas **excepciones**.

Las funciones pueden utilizar valores de **variables externas a ellas** pero ello requiere **incluir dentro de la propia función** la siguiente instrucción:

**global nombre de la variable;**

Por ejemplo: **global \$a1;**

En una instrucción –**global**– pueden definirse como tales, de forma simultánea, varias variables. Basta con escribir los nombres de cada una de ellas separados por comas.

P. ej.: **global \$a1, \$a2, \$a3;**

## Variables superglobales

A partir de la versión **4.1.0** de **PHP** se ha creado un nuevo tipo de variables capaces de **comportarse como globales** sin necesidad de que se definan como tales.

Estas variables que *no pueden ser creadas por usuario*, recogen de forma automática *información muy específica* y tienen nombres preasignados que no pueden modificarse.

Las estudiaremos un poco más adelante. Por ahora, sólo citar los nombres de algunas de ellas:

**\$\_SERVER**, **\$\_POST**, **\$\_GET** o **\$\_ENV** son los de las más importantes.

```
}

/* esta llave de arriba señala el final de la función.
   Los contenidos que hay en adelante ya no pertenecen a ella */

/* Haremos una llamada a la función vervariable.
   Las funciones no se ejecutan hasta que no se les ordena
   y se hace de esta forma que ves aquí debajo:
   nombre de la función seguido de los famosos paréntesis */

vervariable();
?>

<!-- mas HTML puro -->
<center><b><br>Ver la variable desde la función
poniendo <i>global</i></b></center>

<?
# una nueva función

function ahorasi(){
    # aqui definiremos a $Pepe como global
    # la función leerá su valor externo
    global $Pepe;

    echo "<br><br> Hemos asignado ámbito global a la variable";
    echo "<br>ahora Pepe aparecerá";
    echo "<br>El valor de la variable Pepe es: ", $Pepe;

}
# hemos cerrado ya la función con la llave.
# Tendremos que invocarla para que se ejecute ahora
ahorasi();
?>

<center><b><br>Un solo nombre y dos <i>variables distintas</i></b><br>
Dentro de la función el valor de la variable es <br></center>

<?
function cambiaPepa(){

    $Pepa="Ahora voy a llamarme Luisa por un ratito";

    echo "<br>", $Pepa;
}

cambiaPepa();
?>
<center>... pero después de salir de la función
vuelvo al valor original...</center>
<?
echo "<br>", $Pepa;
?>

</BODY>
</HTML>
```

[Ver ejemplo9.php](#)

## Ejercicio nº 5

Escribe un script (guárdalo como **ejercicio5.php**) en el que una misma variable tome dos valores distintos sin utilizar ninguna función. Luego añade al script una función que presente ese mismo nombre de variable con un valor distinto de los anteriores, comprobando que esta última opción no modificó el último valor de aquellos.

[Anterior](#)

[Índice](#)

[Siguiente](#)



Ver índice

## Constantes predefinidas



### Constantes predefinidas

PHP dispone de algunas *constantes predefinidas* que no requieren la instrucción:

`define("Nombre", "Valor")`

Algunas de ellas son estas:

#### **\_FILE\_**

Recoge el nombre del fichero que se está ejecutando y la ruta completa de su ubicación en el servidor.

#### **\_LINE\_**

Recoge el número de línea (incluidas líneas en blanco) del fichero PHP cuyos scripts está interpretando. Puede resultar muy útil para depurar programas escritos en PHP.

#### **PHP\_OS**

Recoge información sobre el Sistema Operativo que utiliza el servidor en el que se está interpretando el fichero.

#### **PHP\_VERSION**

Recoge la versión de PHP que está siendo utilizada por el servidor.

¡Cuidado!

Por si existieran dudas –por problemas de visualización– tanto FILE como LINE tienen que llevar **dos guiones bajos** delante y otras dos detrás.

### Un ejemplo con constantes predefinidas

```
<HTML>
<HEAD>
</HEAD>
<BODY>
<?
# La constante del sistema __FILE__ nos devolverá
echo "La ruta completa de este fichero es: ";
echo __FILE__;

# La constante del sistema __LINE__ nos devolverá
# el número de línea que se está interpretando
# también cuenta las líneas en blanco
# cuenta las líneas y verás que devuelve ... 16

echo "<br>Esta es la línea: ", __LINE__, "del fichero";
echo "<br>Estamos utilizando la versión: ", PHP_VERSION, " de PHP";
echo "<br>El PHP se está ejecutando desde el sistema operativo: ", PHP_OS;

?>

</BODY>
</HTML>
```

[Ver ejemplo10.php](#)

### Ejercicio nº 6

Escribe un script (guárdalo como **ejercicio6.php**) que construya una pequeña página web en la que aparezca el texto "Estás utilizando la versión: 5.2.9-2 de PHP" (incluidas las comillas y extrayendo el número de versión de la constante predefinida correspondiente). Trata de que todas las etiquetas HTML que utilices estén recogidas en variables PHP, de manera que *no exista ninguna línea* en el código fuente de la página que esté fuera de las etiquetas <? ... ?>

Anterior



Índice



Siguiente





Ver índice

# Variables predefinidas



## Las tablas de valores

En las tablas que tenemos aquí a la derecha estamos viendo –clasificadas por tipos– algunas **variables predefinidas** de PHP con sus **valores actuales**.

Esta información (variables y valores) está siendo extraída de *la configuración de tu servidor* y de este documento.

Es muy probable que esté *llamando tu atención* el hecho de que **dos nombres** de variable distintos comparten el mismo **valor**.

Intentaremos ver el por qué de esa **duplicidad**.

Las variables de las columnas de la izquierda comienzan todas por **`$_`**, mientras que las de la derecha lo hacen por **`$HTTP`** y eso es algo **muy importante**.

En el primer caso se alude a las variables **superglobales** que hemos comentado en la página anterior, mientras que en el otro las variables *no tienen ese carácter*.

## Razones de la duplicidad

La **duplicidad** de variables se justifica por lo siguiente: Las variables **superglobales** se introdujeron en PHP a partir de la versión 4.1.0 y no existían con anterioridad.

Con muy buen criterio, los **desarrolladores** PHP optaron por mantener las variables predefinidas de las versiones anteriores para evitar que los programadores que migraran sus aplicaciones a estas nuevas versiones de PHP se vieran obligados a *reescribir el código* de sus scripts.

Es por esta razón, por la que se mantienen dos **variables** distintas para recoger el mismo valor.

Observa la sintaxis de los nombres de las variables y comprobarás la similitud que existe entre ambas. Los corchetes y sus contenidos son exactamente iguales en ambos casos.

Tal es el caso de estas dos, que, por cierto, devuelven en número de la IP a través de la que está accediendo el usuario a nuestro servidor:

## Variables de servidor

<code>\$_SERVER['HTTP_HOST']</code>	<code>\$HTTP_SERVER_VARS['HTTP_HOST']</code>
<b>localhost</b>	
<code>\$_SERVER['HTTP_USER_AGENT']</code>	<code>\$HTTP_SERVER_VARS['HTTP_USER_AGENT']</code>
<b>Mozilla/5.0 (Windows; U; Windows NT 5.1; es-ES; rv:1.9.0.10) Gecko/2009042316 Firefox/3.0.10 (.NET CLR 3.5.30729)</b>	
<code>\$_SERVER['HTTP_ACCEPT']</code>	<code>\$HTTP_SERVER_VARS['HTTP_ACCEPT']</code>
<b>text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8</b>	
<code>\$_SERVER['HTTP_ACCEPT_LANGUAGE']</code>	<code>\$HTTP_SERVER_VARS['HTTP_ACCEPT_LANGUAGE']</code>
<b>es</b>	
<code>\$_SERVER['HTTP_ACCEPT_ENCODING']</code>	<code>\$HTTP_SERVER_VARS['HTTP_ACCEPT_ENCODING']</code>
<b>gzip,deflate</b>	
<code>\$_SERVER['HTTP_ACCEPT_CHARSET']</code>	<code>\$HTTP_SERVER_VARS['HTTP_ACCEPT_CHARSET']</code>
<b>ISO-8859-1,utf-8;q=0.7,*;q=0.7</b>	
<code>\$_SERVER['HTTP_KEEP_ALIVE']</code>	<code>\$HTTP_SERVER_VARS['HTTP_KEEP_ALIVE']</code>
<b>300</b>	
<code>\$_SERVER['HTTP_CONNECTION']</code>	<code>\$HTTP_SERVER_VARS['HTTP_CONNECTION']</code>
<b>keep-alive</b>	
<code>\$_SERVER['HTTP_REFERER']</code>	<code>\$HTTP_SERVER_VARS['HTTP_REFERER']</code>
<b>http://localhost/cursophp/php15.php</b>	
<code>\$_SERVER['HTTP_CACHE_CONTROL']</code>	<code>\$HTTP_SERVER_VARS['HTTP_CACHE_CONTROL']</code>
<b>max-age=0</b>	
<code>\$_SERVER['PATH']</code>	<code>\$HTTP_SERVER_VARS['PATH']</code>
<b>C:\Archivos de programa\PC Connectivity Solution\;C:\WINDOWS\system32;C:\WINDOWS;C:\WINDOWS\System32\Wbem\;;C:\Archivos de programa\Archivos comunes\GTK2.0\bin</b>	
<code>\$_SERVER['SystemRoot']</code>	<code>\$HTTP_SERVER_VARS['SystemRoot']</code>
<b>C:\WINDOWS</b>	
<code>\$_SERVER['COMSPEC']</code>	<code>\$HTTP_SERVER_VARS['COMSPEC']</code>
<b>C:\WINDOWS\system32\cmd.exe</b>	
<code>\$_SERVER['PATHEXT']</code>	<code>\$HTTP_SERVER_VARS['PATHEXT']</code>
<b>.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH</b>	
<code>\$_SERVER['WINDIR']</code>	<code>\$HTTP_SERVER_VARS['WINDIR']</code>
<b>C:\WINDOWS</b>	
<code>\$_SERVER['SERVER_SIGNATURE']</code>	<code>\$HTTP_SERVER_VARS['SERVER_SIGNATURE']</code>
<b></b>	
<code>\$_SERVER['SERVER_SOFTWARE']</code>	<code>\$HTTP_SERVER_VARS['SERVER_SOFTWARE']</code>
<b>Apache/2.2.11 (Win32) PHP/5.2.9-2</b>	
<code>\$_SERVER['SERVER_NAME']</code>	<code>\$HTTP_SERVER_VARS['SERVER_NAME']</code>
<b>localhost</b>	
<code>\$_SERVER['SERVER_ADDR']</code>	<code>\$HTTP_SERVER_VARS['SERVER_ADDR']</code>
<b>127.0.0.1</b>	
<code>\$_SERVER['SERVER_PORT']</code>	<code>\$HTTP_SERVER_VARS['SERVER_PORT']</code>
<b>80</b>	
<code>\$_SERVER['REMOTE_ADDR']</code>	<code>\$HTTP_SERVER_VARS['REMOTE_ADDR']</code>
<b>127.0.0.1</b>	
<code>\$_SERVER['DOCUMENT_ROOT']</code>	<code>\$HTTP_SERVER_VARS['DOCUMENT_ROOT']</code>
<b>C:/ServidoresLocales/Apache/htdocs</b>	
<code>\$_SERVER['SERVER_ADMIN']</code>	<code>\$HTTP_SERVER_VARS['SERVER_ADMIN']</code>
<b>admin@localhost</b>	

y  
**\$HTTP\_SERVER\_VARS**  
['REMOTE\_ADDR']

cuya similitud resulta evidente.

Aunque los **valores** de ambas variables van a ser siempre **idénticos**, no ocurre lo mismo con su **ámbito**.

Las primeras son de ámbito **superglobal** y no necesitan ser declaradas expresamente con globales cuando se trata de utilizar sus valores dentro de cualquier función.

En el segundo caso va a ser **imprescindible** que sean *declaradas* como **globales** antes de que sus valores puedan ser utilizados dentro de una función.

### ¿Nombres algo raros?

Probablemente te extrañará –justificadamente– la longitud y la estructura un tanto rara de estos nombres de variables. Cuando tratemos el tema de los **arrays asociativos** veremos que esa es la sintaxis habitual de ese tipo de variables.

Las nombres de las variables de cada uno de los tipos, sólo se diferencian en lo contenido entre los **corchetes** porque se trata de *distintos elementos* del mismo **array asociativo** y –tal como veremos– esa es la sintaxis típica de los array.

### ¿Para qué sirven?

No vamos a agobiarte con una enumeración de variables y contenidos, pero a poco que observes las tablas de valores te darás cuenta de que es muy abundante y muy importante la información que recogen.

Si analizas las *variables de servidor* te darás cuenta de que aparece un montón de información relativa a su configuración: nombre, rutas, nombres de páginas, IP del servidor, etcétera.

Con los demás tipos ocurre algo similar.

### Los distintos tipos

Veamos los diferentes tipos de variables predefinidas que existen en PHP. Por ahora, no te preocupes demasiado sobre la forma de utilizarlas. Las incluimos aquí como una simple enumeración y con una breve descripción de su utilidad.

En temas posteriores haremos referencia a ellas. Por el momento nos bastará con conocer su existencia.

\$_SERVER[SCRIPT_FILENAME]	\$_HTTP_SERVER_VARS[SCRIPT_FILENAME]
<b>C:/ServidoresLocales/Apache/htdocs/cursophp/php16.php</b>	
\$_SERVER['REMOTE_PORT']	\$_HTTP_SERVER_VARS['REMOTE_PORT']
<b>1544</b>	
\$_SERVER[GATEWAY_INTERFACE]	\$_HTTP_SERVER_VARS[GATEWAY_INTERFACE]
<b>CGI/1.1</b>	
\$_SERVER[SERVER_PROTOCOL]	\$_HTTP_SERVER_VARS[SERVER_PROTOCOL]
<b>HTTP/1.1</b>	
\$_SERVER[REQUEST_METHOD]	\$_HTTP_SERVER_VARS[REQUEST_METHOD]
<b>GET</b>	
\$_SERVER[QUERY_STRING]	\$_HTTP_SERVER_VARS[QUERY_STRING]
\$_SERVER[REQUEST_URI]	\$_HTTP_SERVER_VARS[REQUEST_URI]
<b>/cursophp/php16.php</b>	
\$_SERVER[SCRIPT_NAME]	\$_HTTP_SERVER_VARS[SCRIPT_NAME]
<b>/cursophp/php16.php</b>	
\$_SERVER[PHP_SELF]	\$_HTTP_SERVER_VARS[PHP_SELF]
<b>/cursophp/php16.php</b>	
\$_SERVER[REQUEST_TIME]	\$_HTTP_SERVER_VARS[REQUEST_TIME]
<b>1243267298</b>	
\$_SERVER[argv]	\$_HTTP_SERVER_VARS[argv]
\$_SERVER[argc]	\$_HTTP_SERVER_VARS[argc]

## Variables de entorno

\$_ENV[ALLUSERSPROFILE]	\$_HTTP_ENV_VARS[ALLUSERSPROFILE]
<b>C:\Documents and Settings\All Users</b>	
\$_ENV[APPDATA]	\$_HTTP_ENV_VARS[APPDATA]
<b>C:\Documents and Settings\Usuario\Datos de programa</b>	
\$_ENV[CLIENTNAME]	\$_HTTP_ENV_VARS[CLIENTNAME]
<b>Console</b>	
\$_ENV[CommonProgramFiles]	\$_HTTP_ENV_VARS[CommonProgramFiles]
<b>C:\Archivos de programa\Archivos comunes</b>	
\$_ENV[COMPUTERNAME]	\$_HTTP_ENV_VARS[COMPUTERNAME]
<b>EC-DKS</b>	
\$_ENV[ComSpec]	\$_HTTP_ENV_VARS[ComSpec]
<b>C:\WINDOWS\system32\cmd.exe</b>	
\$_ENV[DEFLOGDIR]	\$_HTTP_ENV_VARS[DEFLOGDIR]
<b>C:\Documents and Settings\All Users\Datos de programa\McAfee\ DesktopProtection</b>	
\$_ENV[FP_NO_HOST_CHECK]	\$_HTTP_ENV_VARS[FP_NO_HOST_CHECK]
<b>NO</b>	
\$_ENV[HOMEDRIVE]	\$_HTTP_ENV_VARS[HOMEDRIVE]
<b>C:</b>	
\$_ENV[HOMEPATH]	\$_HTTP_ENV_VARS[HOMEPATH]
<b>\Documents and Settings\Usuario</b>	
\$_ENV[LANG]	\$_HTTP_ENV_VARS[LANG]
<b>es</b>	
\$_ENV[LOGONSERVER]	\$_HTTP_ENV_VARS[LOGONSERVER]
<b>\ EC-DKS</b>	
\$_ENV[NUMBER_OF_PROCESSORS]	\$_HTTP_ENV_VARS[NUMBER_OF_PROCESSORS]
<b>1</b>	
\$_ENV[OS]	\$_HTTP_ENV_VARS[OS]
<b>Windows_NT</b>	
\$_ENV[Path]	\$_HTTP_ENV_VARS[Path]

Estamos viendo los valores de las variables de entorno (**ENV**) y las de servidor (**SERVER**), pero, además de ellas, existen algunas otras cuyos nombres y utilidades vamos a comentarte.

#### Variables de sesión

Las identificaremos por los nombres **\$SESSION** o por **\$HTTP\_SESSION\_VARS**.

Este tipo de variables las utilizaremos cuando hagamos mención al uso de **sesiones**.

La utilización de sesiones –ya abundaremos en ello– es una forma de recoger, de *forma temporal* en un documento del mismo carácter, información específica generada a través de los accesos de cada uno de los usuarios.

Por ejemplo, cuando accedes al *Aula Virtual* de este curso y escribes tu clave y contraseña se crea un documento temporal en el servidor del CNICE con un número único y exclusivo para ese acceso –identificador de sesión– que te permite acceder a diferentes apartados de ese espacio sin necesidad de que reescribas, en cada una de las páginas, esos mismos valores.

El carácter efímero de las sesiones seguramente lo has comprobado más de una vez cuando al *actualizar* una página te ha aparecido un mensaje diciendo que *la sesión ha caducado*.

#### Variables del método POST

Las identificaremos por los nombres **\$HTTP\_POST\_VARS** o por **\$\_POST**.

Este tipo de variables –que utilizaremos con frecuencia– recogen la información que se envía desde el **cliente** para ser utilizada por el **servidor**.

Recuerda el carácter *dinámico* de PHP y que ese dinamismo (interacción *cliente* – *servidor*) requiere que el servidor guarde los datos remitidos por el cliente.

#### Variables del método GET

Las identificaremos por los nombres **\$HTTP\_GET\_VARS** o por **\$\_GET**.

Son *muy similares* a las anteriores. La existencia de los dos tipos se justifica porque también existen dos tipos de **métodos** (maneras) de enviar datos desde el *cliente* hasta el *servidor*.

Cuando el método de envío es el llamado GET los datos se recogen en variables de este tipo, y, por el contrario, si ese método envío fuera

<b>C:\Archivos de programa\PC Connectivity Solution;C:\WINDOWS\system32;C:\WINDOWS;C:\WINDOWS\System32\Wbem;;;C:\Archivos de programa\Archivos comunes\GTK\2.0\bin</b>	
<b>\$_ENV['PATHEXT']</b>	<b>\$HTTP_ENV_VARS['PATHEXT']</b>
<b>.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH</b>	
<b>\$_ENV['PROCESSOR_ARCHITECTURE']</b>	<b>\$HTTP_ENV_VARS['PROCESSOR_ARCHITECTURE']</b>
	<b>x86</b>
<b>\$_ENV['PROCESSOR_IDENTIFIER']</b>	<b>\$HTTP_ENV_VARS['PROCESSOR_IDENTIFIER']</b>
<b>x86 Family 15 Model 2 Stepping 7, GenuineIntel</b>	
<b>\$_ENV['PROCESSOR_LEVEL']</b>	<b>\$HTTP_ENV_VARS['PROCESSOR_LEVEL']</b>
	<b>15</b>
<b>\$_ENV['PROCESSOR_REVISION']</b>	<b>\$HTTP_ENV_VARS['PROCESSOR_REVISION']</b>
	<b>0207</b>
<b>\$_ENV['ProgramFiles']</b>	<b>\$HTTP_ENV_VARS['ProgramFiles']</b>
<b>C:\Archivos de programa</b>	
<b>\$_ENV['SESSIONNAME']</b>	<b>\$HTTP_ENV_VARS['SESSIONNAME']</b>
<b>Console</b>	
<b>\$_ENV['SystemDrive']</b>	<b>\$HTTP_ENV_VARS['SystemDrive']</b>
<b>C:</b>	
<b>\$_ENV['SystemRoot']</b>	<b>\$HTTP_ENV_VARS['SystemRoot']</b>
<b>C:\WINDOWS</b>	
<b>\$_ENV['TEMP']</b>	<b>\$HTTP_ENV_VARS['TEMP']</b>
<b>C:\DOCUME~1\Usuario\CONFIG~1\Temp</b>	
<b>\$_ENV['TMP']</b>	<b>\$HTTP_ENV_VARS['TMP']</b>
<b>C:\DOCUME~1\Usuario\CONFIG~1\Temp</b>	
<b>\$_ENV['USERDOMAIN']</b>	<b>\$HTTP_ENV_VARS['USERDOMAIN']</b>
<b>EC-DKS</b>	
<b>\$_ENV['USERNAME']</b>	<b>\$HTTP_ENV_VARS['USERNAME']</b>
<b>Usuario</b>	
<b>\$_ENV['USERPROFILE']</b>	<b>\$HTTP_ENV_VARS['USERPROFILE']</b>
<b>C:\Documents and Settings\Usuario</b>	
<b>\$_ENV['VSEDEFLOGDIR']</b>	<b>\$HTTP_ENV_VARS['VSEDEFLOGDIR']</b>
<b>C:\Documents and Settings\All Users\Datos de programa\McAfee\DesktopProtection</b>	
<b>\$_ENV['windir']</b>	<b>\$HTTP_ENV_VARS['windir']</b>
<b>C:\WINDOWS</b>	
<b>\$_ENV['AP_PARENT_PID']</b>	<b>\$HTTP_ENV_VARS['AP_PARENT_PID']</b>
	<b>1588</b>

## Variables GLOBALES

<b>\$GLOBALS['GLOBALS']</b>
<b>Array</b>
<b>\$GLOBALS['_ENV']</b>
<b>Array</b>
<b>\$GLOBALS['HTTP_ENV_VARS']</b>
<b>Array</b>
<b>\$GLOBALS['_POST']</b>
<b>Array</b>
<b>\$GLOBALS['HTTP_POST_VARS']</b>
<b>Array</b>
<b>\$GLOBALS['_GET']</b>
<b>Array</b>
<b>\$GLOBALS['HTTP_GET_VARS']</b>
<b>Array</b>
<b>\$GLOBALS['_COOKIE']</b>
<b>Array</b>
<b>\$GLOBALS['HTTP_COOKIE_VARS']</b>

POST se recogerían en aquellas.

### Variables de transferencia de ficheros

Las identificaremos por el nombre **\$HTTP\_FILES\_VARS** o por **\$\_FILES**.

Para el caso de *transferencia de ficheros* desde el *cliente* al *servidor* –«*subir ficheros*»– es necesario un *procedimiento* distinto de los anteriores.

Será en este caso cuando se utilicen variables de este tipo.

### El tipo GLOBALS

A diferencia de las anteriores, las variables de este tipo disponen de una sintaxis única –**\$GLOBALS**– sin que quepa ninguna otra opción.

Su finalidad es recoger en una *tabla* los nombres de todas la variables establecidas como globales –en cada momento– así como sus valores. Si observas la tabla que tienes aquí a la derecha, quizás te sorprenda leer *nombre* o *página*. ¿De donde han salido esos valores?. Bueno... en esta página utilizamos *scripts PHP* y esos son los nombres de unas variables que hemos incluido en esos *script*.

Conocida la existencia de los diferentes tipos de *variables predefinidas* y vista esta tabla –a modo de ejemplo de su utilidad– no será preciso que profundicemos más en el asunto.

Lo trataremos en el momento en el que tengamos que hacer uso de cada una de ellas.

<b>Array</b>
<b>\$GLOBALS['_SERVER']</b>
<b>Array</b>
<b>\$GLOBALS['HTTP_SERVER_VARS']</b>
<b>Array</b>
<b>\$GLOBALS['_FILES']</b>
<b>Array</b>
<b>\$GLOBALS['HTTP_POST_FILES']</b>
<b>Array</b>
<b>\$GLOBALS['_REQUEST']</b>
<b>Array</b>
<b>\$GLOBALS['opcion']</b>
<b>I</b>
<b>\$GLOBALS['anterior']</b>
<b>php15.php</b>
<b>\$GLOBALS['siguiente']</b>
<b>php17.php</b>
<b>\$GLOBALS['clave']</b>
<b>php17.php</b>
<b>\$GLOBALS['nombre']</b>
<b>nombre</b>

---

Anterior

Índice

Siguiente





Ver índice

## Otras variables



### Valores de las variables

Cuando hablábamos de las variables, nos referíamos a su **ámbito** y comentábamos que las variables definidas dentro de una función pierden sus valores en el momento en el que abandonemos el ámbito de esa función, es decir, cuando finaliza su ejecución.

Decíamos también que si el **ámbito** en el que hubiera sido definida fuera *externo a una función* los valores sólo se perderían **-temporalmente-** mientras durara la eventual ejecución de las instrucciones de aquella y que, una vez acabado ese proceso, volvían a recuperar sus valores.

Bajo estas condiciones, si invocáramos repetidamente la misma función obtendríamos cada vez el mismo resultado.

Las posibles modificaciones que pudieran haberse efectuado (a través de las instrucciones contenidas en la función) en el valor inicial de las variables, se perderían cada vez que abandonáramos la función con lo cual, si hiciéramos *llamadas sucesivas*, se repetirían tanto el valor inicial como el resultado.

### Variables estáticas

Para poder conservar el último valor de una variable definida *dentro de una función* basta con definirla como **estática**.

La instrucción que permite establecer una variable como **estática** es la siguiente:

**static nombre = valor;**

P. ej: si la variable fuera **\$a** y el **valor inicial** asignado fuera **3** escribiríamos:

**static \$a=3;**

La variable conservará el último de los valores que pudo habersele asignado durante la ejecución de la **función** que la contiene. No retornará el **valor inicial** hasta que se actualice la página.

### Variables de variables

Además del método habitual de asignación de nombres a las variables -poner el signo **\$** delante de una palabra-, existe la posibilidad de

### Variabes estáticas

```
<?
# Observa que hemos prescindido de los encabezados HTML.
# No son imprescindibles para la ejecución de los scripts
/* Escribamos una función y llamémosla sinEstaticas
   Definamos en ella dos variables sin ninguna otra especificación
   e insertemos las instrucciones para que al ejecutarse
   se escriban los valores de esas variables */

function sinEstaticas() {

# Pongamos aquí sus valores iniciales
    $a=0;
    $b=0;

# Imprimamos estos valores iniciales

    echo "Valor inicial de $a: ",$a,"<br>";
    echo "Valor inicial de $b: ",$b,"<br>";

/* Modifiquemos esos valores sumando 5 al valor de $a
   y restando 7 al valor de $b.
   $a +=5 y $b -=7 serán quienes haga esas
   nuevas asignaciones de valor
   ya lo iremos viendo, no te preocupes     */

    $a +=5;
    $b -=7;

# Visualicemos los nuevos valores de las variables
    echo "Nuevo valor de $a: ",$a,"<br>";
    echo "Nuevo valor de $b: ",$b,"<br>";

}

# Escribamos ahora la misma función con una modificación que será
# asignar la condición de estática a la variable $b
# Llamemos a esa función: conEstaticas

function conEstaticas() {

# Definimos $b como estática
    $a=0;
        static $b=0;

    echo "Valor inicial de $a: ",$a,"<br>";
    echo "Valor inicial de $b: ",$b,"<br>";

    $a +=5;
    $b -=7;

    echo "Nuevo valor de $a: ",$a,"<br>";
    echo "Nuevo valor de $b: ",$b,"<br>";

}

# Insertemos un texto que nos ayude en el momento de la ejecución

print ("Esta es la primera llamada a sinEstaticas()<br>");

# Invoquemos la función sinEstaticas;

sinEstaticas();
# Añadamos un nuevo comentario a la salida
print ("Esta es la segunda llamada sinEstaticas()<br>");
print ("Debe dar el mismo resultado que la llamada anterior<br>");

# Invoquemos por segunda vez sinEstaticas;
sinEstaticas();
```

que tomen como nombre el valor de otra variable previamente definida.

La forma de hacerlo sería esta:

```
$$nombre_variable_previa;
```

Veamos un ejemplo.

Supongamos que tenemos una variable como esta:

```
$color="verde";
```

Si ahora queremos definir una nueva variable que utilice como nombre el valor (*verde*) que está contenido en la variable previa (\$color), habríamos de poner algo como esto:

```
 $$color="es horrible";
```

¿Cómo podríamos visualizar el valor de esta *nueva variable*?

Habrá tres formas de escribir la instrucción:

```
print $$color;  
o  
print ${$color};  
o también  
print $verde;
```

Cualquiera de las instrucciones anteriores nos produciría la misma salida: *es horrible*.

Podemos preguntarnos ¿cómo se justifica que existan dos sintaxis tan similares como \$\$color y \${\$color}? ¿Qué pintan las llaves?.

La utilización de las llaves es una forma de evitar situaciones de interpretación confusa.

Supongamos que las variables tienen un nombre un poco *más raro*.

Por ejemplo que \$color no se llama así sino \$color[3] (podría ser que \$color fuera un array –una lista de colores– y que esta variable contuviera el tercero de ellos).

En este supuesto, al escribir: `print $$color[3]` cabría la duda de si el número 3 pertenece (es un índice) a la variable \$color o si ese número corresponde a \$\$color.

Con `print ${$color[3]}` no habría lugar para esas dudas. Estaríamos aludiendo de forma inequívoca a 3 como índice de la variable \$color.

¿Qué ocurre cuando la variable previa cambia de valor?

Cuando la variable utilizada para definir una variable de variable cambia de valor no se modifica ni el nombre de esta última ni tampoco su valor.

Puedes ver este concepto, con un

```
# Hagamos ahora lo mismo con la función conEstaticas  
  
print ("Esta es la primera llamada a conEstaticas()<br>");  
  
conEstaticas();  
  
print ("Esta es la segunda llamada a conEstaticas()<br>");  
print ("El resultado es distinto a la llamada anterior<br>");  
  
conEstaticas();  
  
?>
```

ejemplo11.php

## Variables de variables

```
<?  
# Definamos una variable y asignémosle un valor  
$color="rojo";  
# Definamos ahora una nueva variable de nombre variable  
# usando para ello la variable anterior  
 $$color=" es mi color preferido";  
  
# Veamos impresos los contenidos de esas variables  
print ( "El color ".$color. $$color .<br>");  
#o también  
print ( "El color ".$color. ${$color}.<br>");  
# o también  
print ( "El color ".$color. $rojo.<br>");  
  
# advirtamos lo que va a ocurrir al visualizar la página  
  
print ("Las tres líneas anteriores deben decir lo mismo<br>");  
print ("Hemos invocado la misma variable de tres formas diferentes<BR>");  
  
# cambiemos ahora el nombre del color  
$color="magenta";  
  
/* La variable $rojo seguirá existiendo.  
   El hecho de cambiar el valor a $color  
   no significa que vayan a modificarse  
   las variables creadas con su color anterior  
   ni que se creen automáticamente variables  
   que tengan por nombre el nuevo valor de $color */  
  
# Pongamos un mensaje de advertencia para que sea visualizado en la salida  
  
print ("Ahora la variable $color ha cambiado a magenta<br>");  
print ("pero como no hemos creado ninguna variable con ese color<br>");  
print ("en las líneas siguientes no aparecerá nada <br>");  
print ("detrás de la palabra magenta <br>");  
  
# Escribimos los print advertidos  
print (" El color ".$color.$$color."<br>");  
print (" El color ".$color.${$color}.<br>");  
  
# Comprobemos que la variable $rojo creada como variable de variable  
# cuando $color="rojo" aún existe y mantiene aquel valor  
  
print ("Pese a que $color vale ahora ".$color."<br>");  
print ("la vieja variable $rojo sigue existiendo <br>");  
print ("y conserva su valor. Es este: ".$rojo);  
  
?>
```

ejemplo12.php

## Ejercicio nº 7

poco más de detalle, en el *código fuente* del ejemplo.

### La solución del ejercicio

Las diferencias que habrás podido observar al realizar el *ejercicio nº7* están originadas por la interpretación que hacen PHP y el navegador de algunos caracteres especiales, tales como \$, ", o <, que al ser interpretados como *símbolos del lenguaje* no se imprimen en pantalla.

Cuando pretendamos que aparezcan *escritos* tendremos que indicarlo de una forma especial.

En [este enlace](#) podrás ver la forma de hacerlo.

Abre tu editor creando un documento nuevo. Copia el código fuente del ejemplo anterior y pégalo en ese documento. Ahora guárdalo como **ejercicio7.php**. Abre en tu navegador *ejercicio7.php* y compáralo con el que visualizas al pulsar sobre **ejemplo12.php**. Comprueba que existen algunas diferencias entre ambos.

### ¡Cuidado!

Aunque en los enunciados no se advierta, como norma general, los ejercicios deberás guardarlos siempre en el directorio **prácticas** que has creado al realizar el ejercicio nº 2

---

Anterior



Índice



Siguiente





Ver índice

# Tipos de variables



## Tipos de variables

En PHP no es necesaria una definición previa del tipo de variables.

Según los valores que se les vayan asignando, las variables podrán cambiar de tipo –de modo automático– y se irán adaptando a los valores que contengan en cada momento.

Las variables en PHP pueden ser de tres tipos:

– **Enteras** (tipo *Integer*)

– **De coma flotante** (tipo *Double*)

– **Cadenas** (tipo *String*)

Cualquier número entero cuyo valor esté comprendido entre  $\pm 2^{31}$  será interpretado por PHP como de tipo *Integer*.

Si el valor de una variable es un *número decimal* o –siendo entero– desborda el intervalo anterior, bien por asignación directa o como resultado de una operación aritmética, PHP la convierte a tipo *Double*.

Cualquier variable a la que se le asigne como valor el contenido de una cadena de caracteres (letras y/o números delimitados por comillas) es interpretada por PHP como tipo *String*.

## Determinación de tipos de variables

PHP dispone de la función:

**gettype(variable)**

que devuelve una cadena de caracteres indicando el tipo de la variable que contiene.

La cadena devuelta por esta función puede ser: **Integer**, **double** o **string**.

## Forzado de tipos

PHP permite *forzar* los tipos de las variables. Eso quiere decir que se puede *obligar* a PHP a asignar *un tipo determinado a una variable determinada*, siempre que los valores que contenga estén dentro del rango del nuevo tipo de variable.

## Tipos de variables

En el cuadro siguiente podemos ver los tres tipos de variables que utiliza PHP.

Las variables en PHP				
Tipo	Ejemplo	Valor máximo	Valor mínimo	Observaciones
<b>Integer</b>	\$a=1234	2147483647	-2147483647	Cualquier valor numérico entero (dentro de este intervalo) que se asigne a una variable será convertido a este tipo
<b>Double</b>	\$a=1.23			Cualquier valor numérico decimal, o entero fuera del intervalo anterior, que se asigne a una variable la convertirá a este tipo
<b>String</b>	\$a="123"			Cualquier valor <i>entrecomillado</i> (sean números o letras) que se asigne a una variable la convertirá a este tipo

## Determinación del tipo de variable utilizada

Dado que PHP gestiona las variables de forma automática y modifica los tipos de acuerdo con los valores que va tomando durante la ejecución del *script*, se puede recurrir a la función **gettype(nombre de la variable)** para determinar el tipo de la variable actual.

En la tabla siguiente tienes algunos ejemplos de aplicación de esa función.

Podemos observar –en la columna *Sintaxis*– que para visualizar el resultado anteponemos **echo** a **gettype**. Es decir, le indicamos a PHP que muestre el resultado obtenido al determinar el tipo de variable.

Ejemplos de determinación del tipo de una variable		
Variable	Sintaxis	Devuelve
\$a1=347	echo gettype(\$a1)	<b>integer</b>
\$a2=2147483647	echo gettype(\$a2)	<b>integer</b>
\$a3=-2147483647	echo gettype(\$a3)	<b>integer</b>
\$a4=23.7678	echo gettype(\$a4)	<b>double</b>
\$a5=3.1416	echo gettype(\$a5)	<b>double</b>
\$a6="347"	echo gettype(\$a6)	<b>string</b>
\$a7="3.1416"	echo gettype(\$a7)	<b>string</b>
\$a8="Solo literal"	echo gettype(\$a8)	<b>string</b>
\$a9="12.3 Literal con número"	echo gettype(\$a9)	<b>string</b>
\$a10=""	echo gettype(\$a10)	<b>string</b>

## Forzado de tipos

Aquí tienes algunos ejemplos de *forzado de tipos*. Te sugerimos que eches un vistazo a las advertencias que hemos puesto después de esta tabla.

Forzado de tipos		
Variable	Sintaxis	Devuelve
\$a1=347	echo gettype((real)\$a1)	<b>double</b>
\$a2=2147483647	echo gettype((double)\$a2)	<b>double</b>
\$a3=-2147483647	echo gettype((float)\$a3)	<b>double</b>
\$a4=23.7678	echo gettype((int)\$a4)	<b>integer</b>
\$a5=3.1416	echo gettype((integer)\$a5)	<b>integer</b>
\$a6="347"	echo gettype((double)\$a6)	<b>double</b>
\$a7="3.1416"	echo gettype((int)\$a7)	<b>integer</b>
\$a8="3.1416"	echo gettype((string)\$a8)	<b>string</b>

Los tipos se pueden *forzar* tanto en el momento de definir la variable como después de haber sido definida.

### Forzado y asignación simultánea de valores

Al asignar un valor a una variable, se puede *forzar* su tipo de la siguiente forma. Si deseamos que la variable pase a ser tipo de **double** basta con **anteponer** a su valor –entre paréntesis– tal como se indica una de las expresiones:

**(double), (real) o (float).**

Por ejemplo:

```
$a=((double)45); o
$a=(float)45; o
$a=(real)45;
```

cualquiera de ellas produciría el mismo efecto: convertir la variable \$a a tipo **Double**.

Para *forzar* una variable a tipo **Integer** podemos **anteponer** a su valor una de estas expresiones:

**(integer), o (int).**

Por ejemplo:

```
$b=integer)4.5); o
$b=int)45);
```

producirían el mismo efecto: convertir la variable \$b a tipo **Integer**.

Para *forzar* una variable a tipo **String** basta con **anteponer** a su valor (entre paréntesis): **(string)**.

Por ejemplo:

```
$c=(string)4.5);
```

convertiría la variable \$c a tipo **String**.

### Forzado de tipos en variables ya definidas

La forma más aconsejable de *forzado de tipos* en variables que ya estuvieran definidas previamente, es el uso de la siguiente instrucción:

**settype(var, tipo)**

donde *var* es el nombre de la variable cuyo tipo pretendemos modificar y *tipo* una expresión que puede contener (**entre comillas**) uno de estos valores: **'double'**, **'integer'**, o **'string'** según se trate de forzar a: *coma flotante*, *entero*, o *cadena*.

Un ejemplo podría ser este:

**settype(\$a,'integer')**

que convertiría a *tipo entero* la variable \$a.

<b>\$a8="Solo literal"</b>	<b>echo gettype((double)\$a8)</b>	<b>double</b>
<b>\$a9="12.3 Literal con número"</b>	<b>echo gettype((int)\$a9)</b>	<b>integer</b>

### ¡Cuidado!

Al modificar los tipos de variables pueden modificarse sus valores.

Si forzamos a entera una variable que contenga un número decimal se perdería la parte decimal y la variable modificada solo contendría el valor de la parte entera.

Si tratamos de convertir a numérica una variable alfanumérica el nuevo valor sería cero.

Aquí tienes algunos ejemplos relacionados con la advertencia anterior

Nuevos valores de la variable		
Valor inicial	Sintaxis	Nuevo valor
<b>\$a1=347</b>	<b>echo ((real)\$a1)</b>	<b>347</b>
<b>\$a2=2147483647</b>	<b>echo ((double)\$a2)</b>	<b>2147483647</b>
<b>\$a3=-2147483647</b>	<b>echo ((float)\$a3)</b>	<b>-2147483647</b>
<b>\$a4=23.7678</b>	<b>echo ((integer)\$a5)</b>	<b>23</b>
<b>\$a5="3.1416"</b>	<b>echo ((double)\$a6)</b>	<b>3.1416</b>
<b>\$a6="347"</b>	<b>echo ((int)\$a7)</b>	<b>347</b>
<b>\$a7="3.1416"</b>	<b>echo ((string)\$a7)</b>	<b>3.1416</b>
<b>\$a8="Solo literal"</b>	<b>echo ((int)\$a8)</b>	<b>0</b>
<b>\$a9="12.3 Literal con número"</b>	<b>echo ((double)\$a9)</b>	<b>12.3</b>
<b>\$a10=""</b>	<b>echo ((int)\$a9)</b>	<b>0</b>

### Forzado de tipos usando settype()

Aquí tienes algunos ejemplos del uso de esa función. La tabla está organizada en bloques de **tres filas** que corresponden a la ejecución de tres instrucciones y a la visualización del resultado de cada una de ellas.

El resultado de **settype** –primera fila– solo podrá ser **1** ó **0** según la instrucción se haya ejecutado con éxito o no haya podido realizarse.

En la **segunda fila** comprobamos el nuevo tipo de variable obtenida mediante la ejecución de la instrucción anterior y en la **tercera** visualizamos los nuevos valores de la variable, que pueden haber cambiado como consecuencia del cambio de tipo.

Forzado de tipos con settype()		
Variable	Sintaxis	Devuelve
<b>\$a1=347</b>	<b>echo (settype(\$a1,'double'))</b>	<b>1</b>
	<b>echo gettype(\$a1)</b>	<b>double</b>
	<b>echo \$a1</b>	<b>347</b>
<b>\$a2=2147483647</b>	<b>echo (settype(\$a2,'double'))</b>	<b>1</b>
	<b>echo gettype(\$a2)</b>	<b>double</b>
	<b>echo \$a2</b>	<b>2147483647</b>
<b>\$a3=-2147483647</b>	<b>echo settype(\$a3,'double')</b>	<b>1</b>
	<b>echo gettype(\$a3)</b>	<b>double</b>
	<b>echo \$a3</b>	<b>-2147483647</b>
<b>\$a4=23.7678</b>	<b>echo settype(\$a4,'integer')</b>	<b>1</b>
	<b>echo gettype(\$a4)</b>	<b>integer</b>
	<b>echo \$a4</b>	<b>23</b>
<b>\$a5=3.1416</b>	<b>echo settype(\$a5,'integer')</b>	<b>1</b>
	<b>echo gettype(\$a5)</b>	<b>integer</b>
	<b>echo \$a5</b>	<b>3</b>
<b>\$a6="347"</b>	<b>echo settype(\$a6,'double')</b>	<b>1</b>
	<b>echo gettype(\$a6)</b>	<b>double</b>

La ejecución de la instrucción `settype` devuelve (da como resultado) un valor que puede ser: `true` o `false` (1 ó 0) según la *conversión* se haya realizado con éxito o no haya podido realizarse.

## Operaciones con distintos tipos de variables

PHP permite la realización de operaciones aritméticas con cualquiera de los tres tipos de variables y adecúa el resultado al tipo más apropiado.

En la tabla de la derecha puedes ver algunos ejemplos, pero, en resumen, ocurre lo siguiente:

- Al operar con dos enteros, si el resultado está dentro del rango de los enteros, devuelve un entero.

- Si al operar con dos enteros el resultado desborda el rango *entero*, convierte su valor, de forma automática, al tipo *coma flotante*

- Al operar un *entero* con una variable tipo *coma flotante* el resultado es de *coma flotante*.

- Al operar con una *cadena* lo hace como si se tratara de un *entero*. Si hay caracteres numéricos al comienzo, los extrae (hasta que aparezca un *punto* o un *carácter no numérico*) y los opera como un número *entero*.

- Si una cadena *no comienza* por un carácter numérico PHP la operará tomando su valor numérico como CERO.

	<code>echo \$a6</code>	347
<code>\$a7="3.1416"</code>	<code>echo settype(\$a7,'integer')</code>	1
	<code>echo gettype(\$a7)</code>	<b>integer</b>
	<code>echo \$a1</code>	3
<code>\$a8="Solo literal"</code>	<code>echo settype(\$a8,'double')</code>	1
	<code>echo gettype(\$a8)</code>	<b>double</b>
	<code>echo \$a8</code>	0
<code>\$a9="12.3 Literal con número"</code>	<code>echo settype(\$a9,'integer')</code>	1
	<code>echo gettype(\$a9)</code>	<b>integer</b>
	<code>echo \$a9</code>	12

## Tipos de variable de los operadores y de los resultados

La tabla siguiente contiene –en cada fila– los valores asignados a dos variables (A y B) y el resultado de la suma de ambas. A continuación se recogen los tipos de variable de cada una de ellas y el del resultado. El tipo de este último –generado por PHP– estará condicionado por el valor del resultado de cada una de las operaciones.

Resultados de operaciones y tipos de variables resultantes					
Valores			Tipos de variables		
A	B	A+B	A	B	A+B
12	16	28	integer	integer	integer
12	2147483647	2147483659	integer	integer	double
-12	-2147483640	-2147483652	integer	integer	double
12	1.2456	13.2456	integer	double	double
1.2456	12	13.2456	double	integer	double
1.2456	123.4567	124.7023	double	double	double
12	abc	12	integer	string	integer
1.2456	abc	1.2456	double	string	double
12	12abc	24	integer	string	integer
12	12.34567abc	24.34567	integer	string	double
1.2456	12.34567abc	13.59127	double	string	double
1.2456	12.3e2abc	1231.2456	double	string	double
abc	12abc	12	string	string	integer
abc	12.34567abc	12.34567	string	string	double
12abc	12.34567abc	24.34567	string	string	double

Anterior



Índice



Siguiente





Ver índice

## Utilizando formularios



### PHP dinámico

Lo que hemos visto hasta el momento, solo nos ha servido para escribir una serie de scripts PHP y ver los resultados de su ejecución, pero aún no hemos visto de qué forma se puede lograr que interactúen el cliente y el servidor.

Veamos cómo hacerlo.

### Envío a través del navegador

La forma más simple de que un cliente pueda enviar valores a un servidor es incluir esos valores en la propia petición, insertándolos directamente en la barra de direcciones del navegador.

### Forma de envío

Deberá insertarse –en la barra de direcciones del navegador– lo siguiente:

`pagina.php?n1=v1&n2=v2`

donde

`pagina.php` será la dirección de la página que contiene el script que ha de procesar los valores transferidos.

`?`  es un carácter obligatorio que indica que detrás de él van a ser insertados nombres de variables y sus valores.

`n1, n2, etcétera` representan los nombres de las variables.

`=` es el separador de los nombres de las variables y sus valores respectivos.

`v1, v2, ...` simbolizan el valor asignado a cada una de las variables.

`&` es el símbolo que separa los distintos bloques `variable = valor`.

Los nombres de las variables nunca llevan el signo `$`.

Los valores de las variables –sean números o cadenas– nunca se escriben entre comillas.

Algunos caracteres especiales (& por ejemplo) no pueden escribirse directamente dado que se prestan a confusión (no sabría si es un valor de una variable o si se trata de el símbolo de unión).

### ¿Qué valor tiene `register_globals` en tu `php.ini`?

Cuando hablábamos de la configuración de PHP, no hicimos ninguna alusión a la directiva `register_globals` que tenía dos opciones de configuración: ON y OFF. Según el valor que tenga esa configuración cambiará el modo de afrontar diversos asuntos.

En las últimas versiones de PHP viene configurado por defecto `register_globals=OFF`, pero no está de más el comprobarlo.

Empezaremos localizando la ubicación del fichero `php.ini` que está utilizando nuestro PHP. El método más cómodo es ejecutar el script `http://localhost/cursophp/info.php` y comprobar la ruta que se señala en el apartado **Loaded Configuration File**. De esta forma no tendremos duda alguna sobre su localización. Editaremos ese fichero, buscaremos la línea donde dice `register_globals`, comprobaremos que está en OFF y modifiquemos esa directiva dejándola como `register_globals=ON` y guardaremos los cambios..

#### ¡Cuidado!

Antes de hacer una modificación en `php.ini` ó en `httpd.conf` desactiva el servidor Apache.

Cuando hayas acabado con los cambios vuelve a ponerlo en marcha (de lo contrario te aparecerá el mensaje: *no se puede encontrar el servidor*) y ya arrancará atendiendo a la nueva configuración. ¡No te olvides nunca de hacerlo así! Te evitarás un montón de sobresaltos.

Recuerda también, si estás trabajando bajo Linux, que la modificación este tipo de ficheros de configuración requiere los privilegios de superusuario.

### Tabla de multiplicar

```
<?
/* Escribamos una instrucción, que imprima en pantalla
   el valor de una variable ($a), una "x" que hará funciones de aspa
   en la presentación, otra variable ($b), el signo igual
   y $a*$b que es el producto de ambas variables
   El símbolo de multiplicar es (*)      */
print ($a." x ".$b." = ".$a*$b);
# Añadamos una bobadilla, para animar... ;)
print ("<br> ;¡Ya eres mayorcito!.. Deberías saber la tabla");
?>
```

`ejemplo13.php`

Ejecuta este ejemplo y observa lo que aparece. ¿Sólo `x =0`? ¿y la otra línea?

El resultado es lógico. No hemos asignado valores a las variables `$a` y `$b` y por eso no escribió su valor. Sin embargo, a la hora de multiplicar –recuerda que una variable vacía es interpretada como cero a la hora de hacer operaciones– la interpretó como cero y –con muy buen criterio– nos respondió que cero por cero es cero.

Escribamos ahora en la barra de direcciones del navegador la siguiente dirección: `http://localhost/cursophp/ejemplo13.php?a=21&b=456`, (también puedes pulsar en este enlace) y podremos comprobar que  $21 \times 456 = 9576$ .

Hagamos un nuevo experimento. Vayamos a `php.ini` y cambiemos la configuración. Pongamos `register_globals= OFF`. ¡No te olvides de apagar el servidor antes de hacer estos cambios!

Una vez hecho el cambio, pulsemos de nuevo en el enlace, y veremos que esta vez no ha recibido los valores de `a` y `b` y nos ha dado cero como resultado.

En esos casos es necesario sustituir en carácter por su codificación URL que representa cada carácter anteponiendo el signo % al valor de su código ASCII expresando en formato hexadecimal.

En este enlace tienes una tabla de conversión.

Se pueden incluir tantos *nombre* = *valor* como se deseé. La única restricción es la longitud máxima permitida por el método **GET** (el utilizado en este caso) que se sitúa en torno a los 2.000 caracteres.

## Recepción de datos

Cuando es recibida por el **servidor** la **petición** de un documento con extensión **.php** en la que tras el signo ? se incluyen una o varias parejas *nombre* = *valor*, los nombres de las variables y sus valores respectivos se incluyen, de forma automática, en variables predefinidas del tipo:

**\$HTTP\_GET\_VARS['n1']**

**\$HTTP\_GET\_VARS['n2']**

en las que *n1*, *n2*, ... coinciden **exactamente** con los nombres asignados a cada una de las variables en esa transferencia.

Cada una de esas variables contendrá como valor aquél que hubiera recibido a través de la petición.

Si la **versión de PHP** es superior a la **4.1.0**, esos mismos valores se incluirán en variables superglobales del tipo **\$\_GET** de modo que –en el supuesto de que la versión lo soporte– los valores de la petición **también** (esta opción no excluye la anterior) estarían disponibles en:

**\$\_GET['n1']**

**\$\_GET['n2']**

Según el modo en que esté configurado el **php.ini** podría haber una **tercera** posibilidad de registro de esos valores.

Si la directiva **register\_globals** –en el fichero **php.ini**– está configurada con la opción **ON**, los valores transferidos desde el navegador –además de ser recogidos en las variables anteriores– son asignados a otras variables del tipo: **\$n1**, **\$n2**, ... cuyos nombres son el resultado de anteponer el símbolo **\$** a los nombres de las variables contenidas en la petición.

La elección a la hora de escribir los scripts de uno u otro tipo de variable debe hacerse teniendo en cuenta que:

Lo intentaremos con este nuevo script manteniendo **register\_globals=OFF**.

```
<?

/* Modifiquemos la instrucción anterior
y utilicemos las variables predefinidas
$HTTP_GET_VARS['a'], $HTTP_GET_VARS['b']
en vez de $a y $b que utilizabamos en el ejemplo anterior */

# Pongamos un comentario de advertencia. Recuerda que <br>
# sirve para insertar un salto de línea en la salida
print ("Este resultado es el que utiliza $HTTP_GET_VAR<br>");

print ($HTTP_GET_VARS['a']."' x ".$HTTP_GET_VARS['b']."' = ".
$HTTP_GET_VARS['a']*$HTTP_GET_VARS['b']);

/* Ahora trataremos de comprobar que también podemos
utilizar la superglobal $_GET como $_GET['a'] y $_GET['b']
con iguales resultados que las anteriores */

# Un comentario para identificar el origen del resultado

print("<br>El resultado siguiente ha sido generado usando $_GET <br>");
print ("$_GET['a']."' x ".$_GET['b']."' = ".$_GET['a']*$_GET['b']);

?>
```

ejemplo14.php

Escribamos ahora en la barra de direcciones del navegador la siguiente dirección: <http://localhost/cursoPHP/ejemplo14.php?a=21&b=456>, (o pulsemos directamente en este enlace) y podremos comprobar que **21 x 456 aún es igual 9576**.

### ¡Cuidado!

Aquí, más que nunca, conviene reiterar, una vez más, los errores de sintaxis más frecuentes:

Los nombres de variables son distintos si se cambian mayúsculas y minúsculas. Pon mucho cuidado en escribirlos correctamente.

Los nombres de las variables predefinidas, tales como **\$HTTP\_GET\_VARS**, **\$\_GET**, etcétera van en mayúsculas.

No olvides poner punto y coma al final de cada línea de instrucciones.

Presta atención a la apertura/cierre de comillas y mucha más atención aún si se trata de comillas anidadas. En este caso procura usar ("") para las exteriores y ('') para las interiores.

### ¡Cuidado!

En modo local puedes establecer las configuraciones de **php.ini** a tu antojo y, además, estás utilizando una versión –5.2.9-2– de PHP que permite **superglobales**. Esta versión –junto con la posibilidad de modificar **php.ini**– te permite utilizar cualquiera de las opciones de transferencia de variables.

Pero, si pretendes publicar tus páginas utilizando un **hosting** ajeno debes cerciorarte de cuál es su versión de PHP –no todos tienen instaladas versiones superiores a 4.1.0– y conocer la configuración de sus **php.ini**.

Ten en cuenta que allí no vas a poder modificar las configuraciones y de no tener en cuenta estos aspectos, puedes verte obligad@ a **modificar** tu código fuente para adecuarlo a la configuración de tu **hosting**.

## Un formulario

```
<html>
```

cómida— tiene el problema de que sólo es válida cuando `register_globals=on` y, además, es la más *insegura* de todas.

– La superglobal `$_GET` tiene una sintaxis más corta que su alternativa y, además, añade como ventaja su condición de **superglobal**, que permite utilizarla en cualquier ámbito sin necesidad de declararla expresamente como **global**. Es la *opción del futuro*. Su único inconveniente es que puede no estar disponible en hostings que aún mantienen versiones antiguas de PHP.

## Envío a través de formularios

La interacción *cliente-servidor* que acabamos de ver, resulta incómoda en su uso y no demasiado estética.

Hay una segunda opción –la de uso más frecuente– que es la utilización de formularios.

Los formularios no son elementos propios de PHP –actúan del lado del **cliente**– y su estudio es más propio del ámbito de HTML que de PHP.

En cualquier caso –por si te fuera necesario– aquí a la derecha tienes un formulario, en el que se comenta la sintaxis de sus elementos más comunes.

## Interpretación de los datos recibidos a través de formularios

Igual que ocurría en el caso anterior, los datos enviados a través de un formulario son recogidos en diferentes tipos de variables predefinidas, pero ahora se añade una nueva particularidad.

Existe la posibilidad de dos métodos (**method**) de envío: '**GET**' y '**POST**'. En el caso anterior decíamos que se utilizaba el método **GET**, pero en el caso de los formularios son posibles ambos métodos. Conviene tenerlo en cuenta.

### Método GET

No se diferencia en nada del descrito para el supuesto anterior. Utiliza las mismas variables predefinidas, las utiliza con idéntica sintaxis y se comporta de igual forma en lo relativo a las opciones de `register_globals`.

Los nombres de las variables son en este caso, los incluidos como `name` en cada una de las etiquetas del formulario.

Respecto a los valores de cada variable, éstos serían los recogidos del formulario. En los casos de

```
<head>
</head>
<body>
<!-- Un formulario debe empezar siempre con una etiqueta de este tipo
     <form ...> en la que será obligatorio indicar con esta sintaxis
     action='nombre.extension'
     nombre.extension debe contener el nombre (o la ruta completa
     en el caso de que estuviera en un directorio o hosting distinto
     del que alberga el documento que contiene el formulario desde
     el que se realiza la petición)

     Es opcional incluir method que puede tener dos valores
     method='GET' ó method='POST', por defecto (cuando no se indica)
     el envío se realizará usando method='GET'

     También es opcional -a los efectos de PHP- incluir name.
     Ese valor es útil cuando se incluyen scripts del lado del cliente
     del tipo JavaScript //-->

<form name='mi_formulario' action='formul.php' method='post'>
<!-- Pueden incluirse textos dentro del formulario -->
Escribe tu nombre:

<!-- Uno de los tipos de campos posibles es el tipo texto
     su sintaxis (hablamos de HTML) requiere la etiqueta
     <input type='text'> que indica el contenido del texto
     esa etiqueta debe incluir obligatoriamente un name='nombre'
     el nombre a usar serán caracteres alfabéticos, sin tildes
     ni éñes y sin espacios. Salvo excepciones que comentaremos
     no puede usarse el mismo nombre para dos campos distintos
     el value='' puede no contener nada entre las comillas
     tal como ocurre aquí o contener el texto que por defecto queremos
     que aparezca en ese campo al cargar el formulario.
     el size=xx es opcional. Su utilidad es la de ajustar el
     tamaño de la ventana al número de caracteres que se indiquen //-->

<input type='text' name='nombre' value='' size=15><br>
Escribe tu clave:

<!-- <input type='password'> solo se diferencia del anterior
     en que en el momento de rellenarlo se sustituyen los caracteres
     visualizados (no el contenido) por asteriscos //-->

<input type='password' name='clave' value=''><br>
Elige tu color de coche favorito:<br>

<!-- Los <input type='radio'> permite optar entre varios
     valores posibles. Habrá que repetirlos tantas veces como
     opciones queramos habilitar.
     Todos los input -correspondientes a la misma opción-
     deben tener el mismo nombre (name)
     value='loquesea' deberá tener un valor
     distinto en cada uno de ellos. Ese valor (loquesea)
     será transferido a través del formulario
     Si queremos que una opción aparezca marcada (por defecto)
     al cargar el formulario, deberemos incluir en su etiqueta
     la palabra checked
     los contenidos de value no se visualizan en el navegador
     por lo que conviene incluir una descripción de los
     valores después de cerrar la etiqueta de cada input
     Al enviar el formulario solo se transmite el value
     correspondiente a la opción seleccionada //-->

<input type='radio' name='color' value='Rojo'>Rojo<br>
<input type='radio' checked name='color' value='Verde'>Verde<br>
<input type='radio' name='color' value='Azul'>Azul<br>
Elige los extras:<br>

<!-- Cada uno de los <input type='checkbox'>
     requiere un nombre distinto (name) y un valor (value)
     permite optar entre varios
     Esos valor (loquesea)
     serán transferidos a través del formulario
     cuando la casilla de verificación esté marcada
     Si queremos que una casilla aparezca marcada (por defecto)
     al cargar el formulario, deberemos incluir en su etiqueta
     la palabra checked
     los contenidos de value tampoco aquí
```

campos tipo: `text`, `password` y `textarea` serían los valores introducidos por el usuario en cada uno de esos campos.

En el caso de los campos tipo `radio` –en el que varias opciones pueden tener el mismo nombre– recogería el valor indicado en la casilla marcada; mientras que si se trata de campos tipo `checkbox` se transferirían únicamente las variables –y los valores– que corresponden a las casillas marcadas.

Si se tratara de un campo tipo `hidden` se transferiría el valor contenido en su etiqueta y, por último, en el caso del `select` sería transferido como valor de la variable la parte del formulario contenida entre las etiquetas `<option></option>` de la opción seleccionada.

## Método POST

En el caso de que el método de envío sea POST hay una diferencia a tener en cuenta en cuanto a las variables que recogen la información. Ahora será:

### `$HTTP_POST_VARS['n1']`

quien haga la función atribuida en el método anterior a:

### `$HTTP_GET_VARS['n1']`

y ocurrirá algo similar con las superglobales, que pasarán a ser del tipo:

### `$_POST['n1']`

en sustitución del `$_GET['n1']` usado en el caso del método GET

Si `register_globals` está en On el comportamiento de las *variables directas* es idéntico con ambos métodos.

## Identificación del método de envío

PHP recoge en una variable el método utilizado para enviar los datos desde un formulario. Se trata de la variable `REQUEST_METHOD`.

Puede ser invocada como una *variable directa* (en caso de que `register_globals` esté en on) o a través de una de las *variables de servidor*.

En el primer caso la variable se llamaría:

### `$_REQUEST_METHOD`

y en el segundo:

### `$HTTP_SERVER_VARS[REQUEST_METHOD]`

Cuando PHP permita el uso de variables *superglobales* se puede

se visualizan en el navegador por lo que conviene incluir una descripción de los valores después de cerrar la etiqueta de cada input  
Al enviar el formulario solo se transmite los **value** correspondiente a la opcion marcadas //-->

```
<input type='checkbox' name="acondicionado" value="Aire">
    Aire acondicionado<br>
<input type='checkbox' checked name="tapiceria" value="Tapiceria">
    Tapiceria en piel<br>
<input type='checkbox' name="llantas" value="aluminio">
    Llantas de aluminio<br>
¿Cuál es el precio máximo<br>
que estarías dispuesto a pagar?

<!-- La etiqueta &lt;input type='select'&gt;
    requiere un <b>nombre
    y requiere también una etiqueta de cierre
        </select>
    Entre ambas -apertura y cierre-
    deben incluirse las diferentes opciones
    entre las de etiquetas
        <option>valor</option>
    Al enviar el formulario se transmite lo
    contenido despues de opción
    en la opción seleccionada
    si dentro de una etiqueta option
        escribimos selected será esa
    la que aparezca por defecto al cargarse el formulario//-->

<select name="precio">
<Option>Menos de 6.000 euros</option>
<Option>6.001 - 8.000 euros</option>
<Option selected>8.001 - 10.000 euros</option>
<Option>10.001 - 12.000 euros</option>
<Option>12.001 - 14.000 euros</option>
<Option>Más de 14.000 euros</option>
</select>

<!-- Las áreas de texto deben tener una etiqueta de apertura
    &lt;textarea name='checkbox'&gt;
        seguida de una etiqueta de cierre &lt;/textarea&gt;
        Dentro de la etiqueta de apertura puede incluirse
        rows=xx (indicará el número de filas)
        cols=yy (indicará el ancho expresado en número de caracteres)
        y opcionalmente un value='lo que sea...'
        que puede contener el texto que -por defecto-
        pretendemos que aparezca en ese espacio
        en el momento de cargar rl formulario //--&gt;

&lt;br&gt; Escribe aquí cualquier otro comentario:&lt;br&gt;
&lt;textarea rows=5 cols=50 name='texto'&gt;&lt;/textarea&gt;&lt;br&gt;

<!-- El &lt;input type='hidden'&gt;
    permite insertar en un formulario una valor <i>oculto
    que no requiere ser cumplimentado por el usuario
    y que no aparece visible en el documento
    requiere un name y un value //-->

<input type="hidden" name='oculto' value='Esto iría oculto'><br>

<!-- El &lt;input type='submit'&gt;
    es el encargado de ejecutar la action
    incluida en la etiqueta de apertura del formulario
    que en este caso seria la llamada
    a la página que se indica en la action
    El texto que incluyamos en value='enviar...'
    será el que se visualice en el propio botón de envio //--&gt;

&lt;input type="submit" value="enviar"&gt;
<!-- El &lt;input type='reset'&gt;
    permite borrar todos los contenidos
    del formulario y reestablecer los valores
    por defecto de cada campo //--&gt;

&lt;input type="reset" value="borrar"&gt;
<!-- La etiqueta &lt;/form&gt;
    es la etiqueta de cierre del formulario //--&gt;
&lt;/FORM&gt;</pre>
```

utilizar:

```
</BODY>
</HTML>
```

## **\$\_SERVER[REQUEST\_METHOD]**

Una advertencia importante.

Observa que en este caso **no se incluyen comillas** dentro del corchete como ocurría con todos los nombres de variable anteriores.

## Diferencias entre los métodos GET y POST

Las diferencias entre uno y otro método son las siguientes:

### Método GET

Las particularidades de este método son las siguientes:

- Al ser enviado el formulario se **carga** en el navegador la dirección especificada como **action**, se le añade un ? y a continuación se incluyen los datos del formulario. Todos los datos de la petición **van a ser visibles** desde la **barra de direcciones del navegador**.
- Únicamente son aceptados los caracteres ASCII.

- Tiene una limitación en el tamaño máximo de la cadena que contiene los datos a transferir. En IE esa limitación es de 2.083 caracteres.

### Método POST

No tiene las limitaciones indicadas para el caso de **GET** en lo relativo a **visibilidad** ni en cuanto a aceptación de caracteres no ASCII.

Este método de transferencia de datos es el más habitual cuando se utilizan formularios.

## Tipos de contenidos de los formularios

Respecto a los formularios, vamos a contemplar un último aspecto: la **forma de encriptar** de los datos en el momento de la transmisión (**ENCTYPE**).

Puede especificarse dentro de la etiqueta **<form>** utilizando la sintaxis: **enctype='valor'**.

En el caso de que no se especifique, tomará el valor **application / x-www-form-urlencoded**, sea **GET** o **POST** el método que se utilice.

El método **POST** admite **multipart/form-data** como una opción alternativa a la anterior. Suele utilizarse cuando se trata de enviar grandes cantidades de datos, formularios en los que se adjuntan ficheros, datos no ASCII o contenidos binarios.

No incluiremos la opción **Ver ejemplo** hasta que hayamos elaborado los documentos **formuxx.php** incluidos en la **action** de este formulario.

## Scripts para recoger los datos del formulario anterior

Insertaremos ahora los diferentes tipos scripts utilizables, especificando las condiciones de utilización de cada uno de ellos.

Los hemos llamado **formu1.php**, **formu2.php**, etcétera.

Debajo del código fuente de cada uno de ellos, hemos incluido dos enlaces. En el primero de ellos se utiliza el formulario que vemos aquí arriba modificando únicamente el valor de **action** para adecuarlo al nombre de cada script.

En el segundo de los enlaces utilizaremos un formulario en el que, además de las modificaciones anteriores, se utilice **GET** como método.

De esta forma, tendrás la oportunidad de comprobar el funcionamiento o no funcionamiento anunciado en cada uno de los supuestos.

Sería buena idea que **experimentaras** con ellos tanto bajo **register\_globals=ON** como cuando esté en modo **OFF**.

Este primero funcionará tanto cuando el método sea **POST** como cuando sea **GET**. Requiere que el **php.ini** contenga la opción **register\_globals=ON**

```
<?
echo "El method que ha usado fué: ",$REQUEST_METHOD,"<br>";
echo $nombre,"<br>";
echo $clave,"<br>";
echo $color,"<br>";
echo $acondicionado,"<br>";
echo $tapiceria,"<br>";
echo $llantas,"<br>";
echo $precio,"<br>";
echo $texto,"<br>";
echo $oculto,"<br>";
?>
```

«Con action = POST»

«Con action = GET»

Este otro **requiere** que el método especificado en el formulario de envío sea **POST**. El valor de **register\_globals** no afectaría a su funcionalidad.

```
<?
echo "El method usado fué: ",$HTTP_SERVER_VARS[REQUEST_METHOD],"<br>";
echo $HTTP_POST_VARS['nombre'],"<br>";
echo $HTTP_POST_VARS['clave'],"<br>";
echo $HTTP_POST_VARS['color'],"<br>";
echo $HTTP_POST_VARS['acondicionado'],"<br>";
echo $HTTP_POST_VARS['tapiceria'],"<br>";
echo $HTTP_POST_VARS['llantas'],"<br>";
echo $HTTP_POST_VARS['precio'],"<br>";
echo $HTTP_POST_VARS['texto'],"<br>";
echo $HTTP_POST_VARS['oculto'],"<br>";
?>
```

«Con action = POST»

«Con action = GET»

Para utilizar eficazmente este script **es necesario** que el método especificado en el formulario de envío sea **GET**.

El valor de **register\_globals** no afectaría a su funcionalidad.

Las diferencias básicas entre ambos modos de encriptación son las siguientes:

En el tipo **application/x-www-form-urlencoded** los nombres de control y los valores se transforman en *secuencias de escape*, es decir, convirtiendo cada byte en una cadena %HH, donde HH es la notación hexadecimal del valor del byte.

Además, los espacios son convertidos en **signos +**, los saltos de línea se representan como **%0D%0A**, el nombre y el valor se separan con el signo = y los diferentes bloques nombre/valor, se separan con el carácter &.

En cuanto a la encriptación tipo **multipart/form-data**, sigue las reglas de las transferencias MIME, que comentaremos más adelante cuando tratemos el tema del correo electrónico.

### ¡Cuidado!

Cuando se incluye una cadena vacía ("") como valor de **action** en un formulario se recargará el mismo documento como respuesta al envío del formulario.

## La seguridad en los envíos de datos

El tema de la seguridad es una preocupación constante entre los usuarios de Internet.

Cuando utilizamos las técnicas que venimos comentando en esta página –nos referimos siempre al caso de servidores remotos– corremos dos tipos de riesgo de seguridad que no estaría de más tener en cuenta.

El riesgo de que la información sea interceptada durante el proceso de transmisión desde el cliente hasta el servidor lo compartimos con todos los demás usuarios de la Red, pero hay otro –el riesgo de daños en los contenidos de nuestro espacio de servidor– que es **exclusivamente** nuestro.

La *transparencia* del método GET es tal, que incluso muestra –en el momento del envío– todos los datos en la barra de direcciones del navegador. Eso permite que cualquier usuario pueda conocer a simple vista la ruta completa hasta el script, así como los nombres y valores de las variables.

Cuando se usa el método POST los formularios son un poco más

```
<?
echo "El method usado fué: ",$_HTTP_SERVER_VARS[REQUEST_METHOD], "<br>";
echo $_HTTP_GET_VARS['nombre'], "<br>";
echo $_HTTP_GET_VARS['clave'], "<br>";
echo $_HTTP_GET_VARS['color'], "<br>";
echo $_HTTP_GET_VARS['acondicionado'], "<br>";
echo $_HTTP_GET_VARS['tapiceria'], "<br>";
echo $_HTTP_GET_VARS['llantas'], "<br>";
echo $_HTTP_GET_VARS['precio'], "<br>";
echo $_HTTP_GET_VARS['texto'], "<br>";
echo $_HTTP_GET_VARS['oculto'], "<br>";
?>
```

«Con action = POST»

«Con action = GET»

Este otro **requiere** que el método especificado en el formulario de envío sea **POST** y que la versión de PHP soporte variables **superglobales**.

El valor de register\_globals no afectaría a su funcionalidad.

```
<?
echo "El method usado fué: ",$_SERVER[REQUEST_METHOD], "<br>";
echo $_POST['nombre'], "<br>";
echo $_POST['clave'], "<br>";
echo $_POST['color'], "<br>";
echo $_POST['acondicionado'], "<br>";
echo $_POST['tapiceria'], "<br>";
echo $_POST['llantas'], "<br>";
echo $_POST['precio'], "<br>";
echo $_POST['texto'], "<br>";
echo $_POST['oculto'], "<br>";
?>
```

«Con action = POST»

«Con action = GET»

En este supuesto sería necesario que el método especificado en el formulario de envío sea **GET** y que la versión de PHP instalada en el servidor que lo aloja soporte variables **superglobales**.

El valor de register\_globals no afectaría a su funcionalidad.

```
<?
echo "El method que ha usado fué: ",$_SERVER[REQUEST_METHOD], "<br>";
echo $_GET['nombre'], "<br>";
echo $_GET['clave'], "<br>";
echo $_GET['color'], "<br>";
echo $_GET['acondicionado'], "<br>";
echo $_GET['tapiceria'], "<br>";
echo $_GET['llantas'], "<br>";
echo $_GET['precio'], "<br>";
echo $_GET['texto'], "<br>";
echo $_GET['oculto'], "<br>";
?>
```

«Con action = POST»

«Con action = GET»

## La variable **\$REQUEST**

PHP también dispone –a partir de su versión 4.1.0– de la variable **\$REQUEST** (de tipo superglobal) que aúna las funcionalidades de **\$\_GET** y **\$\_POST** y que recoge en variables del tipo **\$\_REQUEST['nombre']** tanto los valores transferidos mediante el método **GET** como mediante **POST**.

**\$REQUEST**, a diferencia de **\$\_GET** y **\$\_POST**, no dispone de equivalentes en versiones anteriores. Ello quiere decir, **no existe** una variable (no superglobal) del tipo **\$\_HTTP\_REQUEST\_VARS** con **\$\_HTTP\_GET\_VARS** o con **\$\_HTTP\_POST\_VARS**.

En este ejemplo hemos incluido dos scripts que solo se diferencian en el *method* especificado en el formulario.

discretos, pero igual de transparentes. El código de cualquier formulario estará accesible sólo con ir a la opción Ver código fuente y allí estarán de nuevo todos los datos: nombre del script, nombres de las variables, etcétera, con lo que, cualquier usuario y desde cualquier sitio, puede acceder a ese script.

No haría falta ni usar *nuestro* formulario. Bastaría guardar una copia del mismo en el ordenador del visitante y después –haciendo ligerísimos retoques– se podría acceder a nuestro script sin necesidad de utilizar el formulario alojado en nuestro servidor.

Si pensamos que uno de nuestros scripts puede estar diseñado con el fin de modificar algunos de los contenidos de nuestro espacio –borrar datos, por ejemplo– seguramente sería cuestión de empezar a preocuparnos, y mucho más si en nuestro servidor tenemos datos *importantes*.

Existen formas de evitar, o al menos reducir, este tipo de riesgos. Restringir a usuarios autorizados el uso de algunos subdirectorios es una de ellas, almacenar datos importantes fuera del directorio root del servidor es otra y el uso de algunas de las variables predefinidas como elementos de protección puede ser una tercera.

Hacemos este comentario a *título meramente informativo*. Por el momento nos basta con manejar los formularios, pero queremos que tengas presente la existencia de ese tipo de riesgos. Más adelante, veremos la manera de *tratar de evitarlos*.

### Ejercicio nº 8

Crea dos documentos llamados respectivamente *color1.php* y *color2.php*.

En el primero de ellos incluye un formulario en el que –mediante un menú de opciones– se pueda elegir uno de entre cinco colores (especifica como *value* el código hexadecimal de cada uno de ellos).

Al enviar los datos, deberá cargarse la página *color2.php* con el color de fondo correspondiente a la opción seleccionada en el formulario.

Al ejecutarlos podremos comprobar que, independientemente del método usado, *\$\_REQUEST* recoge los valores transferidos desde el formulario.

```
<?
/* Al ejecutar por primera vez el script la variable pepe será nula ya que no se ha transferido aún el formulario. Al pulsar sucesivamente en el botón Enviar iremos visualizando los valores que se vayan transfiriendo */
print "He recibido la variable pepe con valor: ".$_REQUEST['pepe'];
/* al enviar el formulario se recargará este mismo documento ya que hemos puesto action="" */
?>
<form name="prueba" method="post" action="">
<input type="text" name="pepe" value=''>
<input type="submit" value="enviar">
```

«Con action = POST»

«Con action = GET»

### Ejercicio nº 9

Crea dos documentos llamados respectivamente **formulario1.php** y **visor1.php**. En el primero de ellos incluye un formulario que permita recoger datos personales y académicos de tus alumnos, utilizando todos los tipos de campos de formulario que conozcas. Los datos insertados en ese formulario deberán ser visualizados después de su envío –con cualquier configuración de *register\_globals* y con cualquier versión PHP– a través del documento *visor1.php*. Utiliza los recursos estéticos –fondos, colores, tipografía, etcétera– que estimes oportunos para una correcta presentación

### Ejercicio nº 10

Con criterios similares en cuanto a estética y funcionalidad a los del ejercicio anterior, te proponemos que crees dos nuevos documentos con nombres **formulario2.php** y **visor2.php**. En este caso el formulario deberá poder recoger el nombre y apellidos de un alumno hipotético al que debemos formularle dos preguntas. La primera de ellas con cuatro posibles respuestas entre las que deba elegir como válida una de ellas. La segunda, también con cuatro respuestas, deberá permitir marcar las respuestas correctas que pueden ser: todas, ninguna, o algunas de ellas.

El alumno debería poder insertar sus datos personales en el formulario y elegir las respuestas a las preguntas formuladas. Al pulsar en el botón de envío, los datos del alumno y las respuestas elegidas deben visualizarse a través del documento **visor2.php**.

Anterior

Índice

Siguiente





## Operaciones aritméticas

En páginas anteriores hemos podido ver que PHP permite utilizar un tipo de variables –las numéricas– cuyos valores puedan ser *operados* de la misma forma que se hace con los números en la vida cotidiana.

Los resultados de las operaciones pueden utilizarse de *forma directa* o ser recogidos en una *nueva variable*. Aquí a la derecha tienes un ejemplo de ambas opciones.

Si asignamos a una nueva variable el resultado de una operación el valor contenido en ella no se modifica, aunque cambien los de las variables que intervinieron su creación.

### Sintaxis de print y echo

Si queremos *encadenar* en una sola instrucción –*echo* ó *print*– el resultado de una operación junto con otras variables (o cadenas) es **imprescindible** poner entre paréntesis las instrucciones de la operación.

Esta norma, solo tiene dos excepciones: en caso de que el *print* solo contenga la propia operación o cuando utilicemos *echo* y el separador sea una **coma**.

## Operadores aritméticos

### Suma

**\$a + \$b**

### Diferencia

**\$a - \$b**

### Producto

**\$a \* \$b**

### Cociente

**\$a / \$b**

### Cociente entero

**(int)(\$a / \$b)**

### Resto de la división

**\$a % \$b**

### Raíz cuadrada

**Sqrt(\$a)**

**Potencia a<sup>b</sup>**

## Operaciones aritméticas

```
<?
# definiamos dos variables numéricas asignandoles valores
$a=23; $b=34;
/* hagamos una suma y escribamos directamente los resultados
utilizando las instrucciones print y echo
con todas sus posibles opciones de sintaxis */
print("La suma de $a + $b es: " . $a . "+" . $b . "=" . ($a+$b) . "<br>");
print "La suma de $a + $b es: " . $a . "+" . $b . "=" . ($a+$b) . "<BR>";
print ("La suma de $a + $b es: " . $a . "+" . $b . "=" . ($a+$b) . "<BR>");
echo "La suma de $a + $b es: " . $a . "+" . $b . "=" . ($a+$b) . "<BR>";
echo "La suma de $a + $b es: " , $a , "+" , $b . "=" , ($a+$b) . "<BR>";
echo "La suma de $a + $b es: " , $a , "+" , $b , "=" , $a+$b , "<BR>";
# guardemos ahora el resultado de esa operación en una nueva variable
$c=$a+$b;
/*ahora presentemos el resultado utilizando esa nueva variable
adviertiendo el la salida*/
print ("Resultados recogidos en una nueva variable<br>");
print "La suma de $a + $b es: " . $a . "+" . $b . "=" . $c . "<BR>";
print ("La suma de $a + $b es: " . $a . "+" . $b . "=" . $c . "<BR>");
echo "La suma de $a + $b es: " . $a . "+" . $b . "=" . $c . "<BR>";
echo "La suma de $a + $b es: " , $a , "+" , $b . "=" , $c . "<BR>";
echo "La suma de $a + $b es: " , $a , "+" , $b , "=" , $c , "<BR>";
/* modifiquemos ahora los valores de $a y $b comprobando que el cambio
no modifica lo contenido en la variable $c */
$a=513; $b=648;
print ("<br> C sigue valiendo: " . $c . "<br>");
# experimentemos con los paréntesis en un supuesto de operaciones combinadas
# tratemos de sumar la variable $a con la variable $b
# y multiplicar el resultado por $c.
# Si escribimos print($a+$b*$c) nos hará la multiplicación antes que la suma
print "<br>No he puesto paréntesis y el resultado es: ".($a+$b*$c);
# Si escribimos print((($a+$b)*$c)) nos hará la suma y luego multiplicará
print "<br>He puesto paréntesis y el resultado es: ".(($a+$b)*$c);
?>
```

ejemplo16.php

Operaciones aritméticas				
Operación	Sintaxis	A	B	Resultado
Suma	<b>\$a+\$b</b>	12	-7.3	4.7
Diferencia	<b>\$a-\$b</b>	12	-7.3	19.3
Producto	<b>\$a*\$b</b>	12	-7.3	-87.6
Cociente	<b>\$a/\$b</b>	12	-7.3	-1.64383561644
Cociente entero	<b>(int)(\$a/\$b)</b>	12	-7.3	-1
Resto de la división	<b>\$a%\$b</b>	12	5	2
Potencias a <sup>b</sup>	<b>pow(\$a,\$b)</b>	12	5	248832
Potencias a <sup>b</sup>	<b>pow(\$a,\$b)</b>	-7.3	-3	-0.00257058174836
Raíz cuadrada	<b>Sqrt(\$a)</b>	12		3.46410161514
Raíz cuadrada	<b>Sqrt(\$a)</b>	-7.3		NAN
Raíz enésima	<b>pow(\$a,(1/\$b))</b>	12	3;	2.28942848511

Redondeos			
tipo	Sintaxis	A	Resultado
Parte entera	<b>(int)\$a</b>	12	12
Parte entera	<b>(int)\$a</b>	-7.3	-7
Parte entera	<b>(int)\$a</b>	-13.8546	-13

**pow(\$a,\$b)**

Raíz (de índice b) de a

**pow(\$a,1/\$b)**

## Redondeo de resultados

PHP tiene tres opciones de redondeo:

**Redondeo por defecto**

**floor(\$z)**

**Redondeo por exceso**

**ceil(\$z)**

**Redondeo tradicional**

**round(\$z)**

Al realizar una operación cuyo resultado no es un número real PHP devuelve la cadena -1.#IND.

## Orden de operación

Cuando una misma instrucción contiene una secuencia con varias operaciones el orden de ejecución de las mismas sigue los mismos criterios que las matemáticas. No se realiza una ejecución secuencial sino que se respeta el *orden de prioridad matemático*. Es decir, las potencias y raíces tienen prioridad frente a los productos y los cocientes, y estos, son prioritarios respecto a la suma y las diferencias.

Igual que en matemáticas se pueden utilizar los paréntesis para *modificar* el orden de ejecución de las operaciones, e igual que *allí* PHP también permite encerrar paréntesis dentro de paréntesis.

Parte entera	(int)\$a	-24.5	-24
Parte entera	(int)\$a	13.8546	13
Parte entera	(int)\$a	24.5	24
Redondeo por defecto	floor(\$a)	12	12
Redondeo por defecto	floor(\$a)	-7.3	-8
Redondeo por defecto	floor(\$a)	-13.8546	-14
Redondeo por defecto	floor(\$a)	-24.5	-25
Redondeo por defecto	floor(\$a)	13.8546	13
Redondeo por defecto	floor(\$a)	24.5	24
Redondeo por exceso	ceil(\$a)	12	12
Redondeo por exceso	ceil(\$a)	-7.3	-7
Redondeo por exceso	ceil(\$a)	-13.8546	-13
Redondeo por exceso	ceil(\$a)	-24.5	-24
Redondeo por exceso	ceil(\$a)	13.8546	14
Redondeo por exceso	ceil(\$a)	24.5	25
Redondeo	round(\$a)	12	12
Redondeo	round(\$a)	-7.3	-7
Redondeo	round(\$a)	-13.8546	-14
Redondeo	round(\$a)	-24.5	-25
Redondeo	round(\$a)	13.8546	14
Redondeo	round(\$a)	24.5	25

### ¡Cuidado!

Cuando realices operaciones combinadas, no olvides establecer **–mediante paréntesis–** las prioridades que sean necesarias. ¡No temas abusar de ellos! Te evitarán muchos problemas.

## Ejercicio nº 11

Crea un documento con el nombre **ejercicio11.php** e incluye en él un formulario que permita introducir valores numéricos en dos campos de texto. Al enviarlo (puedes usar el mismo documento para el formulario y para la visualización de resultados) deberán aparecer los resultados de: *sumar*, *restar*, *multiplicar* y *dividir* ambos números. Deberá aparecer también el resultado (redondeado por exceso) de *elevant la suma de ambos números a la cuarta potencia* y la *raíz quinta del cubo de la suma de ambos números*.

Anterior



Índice



Siguiente



## Logarítmos y trigonometría

La sintaxis para el uso de las funciones logarítmicas y trigonométricas es esta:

- Logaritmo neperiano

**log(\$a)**

- Logaritmo decimal

**Log10(\$a)**

- $e^a$

**Exp(\$a)**

En cuanto a las funciones trigonométricas debe tenerse en cuenta que tanto en las directas como en las inversas los valores de los angulos se expresan en radianes.

- El número pi

La función **pi()** devuelve el valor del número pi

- Seno de a

**Sin(\$a)**

- Coseno de a

**Cos(\$a)**

- Tangente de a

**Tan(\$a)**

- Convierte grados en radianes

**deg2rad(\$a)**

- Arco cuyo Seno es a

**Asin(\$a)**

- Arco cuyo Coseno es a

**Acos(\$a)**

- Arco cuya Tangente es a

**Atan(\$a)**

- Convierte radianes en grados sexagesimales

**rad2deg(\$a)**

## Logaritmos y trigonometría

En el cuadro siguiente se resumen las operaciones aritméticas en PHP y su sintaxis

Operaciones con logaritmos				
Operación	Sintaxis	A	B	Resultado
Logaritmo neperiano	<b>log(\$a)</b>	12		2.48490664979
Logaritmo neperiano	<b>log(\$a)</b>	-7.3		NAN
Logaritmo decimal	<b>Log10(\$a)</b>	12		1.07918124605
Logaritmo decimal	<b>Log10(\$a)</b>	-7.3		NAN
exponencial $e^a$	<b>Exp(\$a)</b>	12		162754.791419
exponencial $e^a$	<b>Exp(\$a)</b>	-7.3		0.000675538775194
exponencial $10^a$	<b>pow(10,\$a)</b>	12		1.0E+12
exponencial $10^a$	<b>pow(10,\$a)</b>	-7.3		5.01187233627E-8

Funciones trigonométricas				
Operación	Sintaxis	A	B	Resultado
Seno de A (radianes)	<b>Sin(\$a)</b>	12		-0.536572918
Seno de PI (radianes)	<b>Sin(pi())</b>	pi()		1.22460635382E-16
Coseno de A (radianes)	<b>Cos(\$a)</b>	12		0.843853958732
Coseno de PI (radianes)	<b>Cos(pi())</b>	pi()		-1
Tangente de A (radianes)	<b>Tan(\$a)</b>	12		0.843853958732
Tangente de PI (radianes)	<b>Tan(pi())</b>	pi()		-1.22460635382E-16
Tangente de PI/2 (radianes)	<b>Tan(pi())</b>	pi()/2		1.63317787284E+16
Pasa grados a radianes	<b>deg2rad(\$a)</b>	45		0.785398163397

Funciones trigonométricas inversas				
Operación	Sintaxis	A	B	Resultado
Arco seno de A (en radianes)	<b>Asin(\$a)</b>	0.8		0.927295218002
Arco seno de A (en radianes)	<b>Asin(\$a)</b>	12		NAN
Arco coseno de A (en radianes)	<b>Acos(\$a)</b>	0.8		0.643501108793
Arco coseno de A (en radianes)	<b>Acos(\$a)</b>	12		NAN
Arco tangente de A (en radianes)	<b>Atan(\$a)</b>	0.8		0.674740942224
Arco tangente de A (en radianes)	<b>Atan(\$a)</b>	12		1.48765509491
Pasa radianes a grados	<b>rad2deg(\$a)</b>	pi()/4		45

Anterior



Índice



Siguiente



## Formatos de las variables numéricas

Para asignar valores numéricos a una variable, en PHP, puede utilizarse uno de los siguientes sistemas de numeración:

Los **números enteros** pueden escribirse en una cualquiera de estas bases:

- **Base decimal**

**\$a=número**

No se pueden insertar ceros a la izquierda cuando se escriben números en base decimal.

- **Base octal**

**\$a=0número octal**

Basta poner un CERO delante del número para que sea interpretado como escrito en base OCTAL. Obviamente, sólo admite los dígitos de 0 a 7.

- **Base hexadecimal**

**\$a=0xnúmero hexadecimal**

Si se escribe CERO EQUIS (0x) delante del número, PHP lo interpretará como expresado en hexadecimal. En este caso, admitirá como dígitos de 0 a 9 y de A a F.

Un número de **coma flotante** puede escribirse de cualquiera de estas formas:

- **Notación decimal.**

**\$a=número**

Se pueden utilizar un cero a la izquierda del punto decimal.

- **Notación científica**

**\$a=número e exponente**

Se puede utilizar un cero a la izquierda del punto decimal

Ejemplo: **\$a=1.2e5** asigna a \$a el valor:  $1.2 \times 10^5$

Otro ej: **\$a=1.2e-5** asigna a \$a el valor:  $1.2 \times 10^{-5}$

## Cambios de base

PHP permite hacer todo tipo de **cambios de base**. Para evitar ser reiterativos, observa los ejemplos. Allí tienes las diferentes funciones mediante las que se puede realizar ese proceso.

## Formas de asignar valores a la variables

Cuando se asignan valores numéricos a una variable PHP cabe la posibilidad de hacerlo en distintas bases. Estos son algunos ejemplos.

Asignación de valores en distintas bases			
Base	Sintaxis	Valor decimal	Aplicable a
Base Decimal	<b>\$a=17</b>	17	Números enteros
Base Octal	<b>\$a=017</b>	15	Números enteros
Base Hexadecimal	<b>\$a=0x17</b>	23	Números enteros
Base Hexadecimal	<b>\$a=0x1A3B</b>	6715	Números enteros
Notación decimal	<b>\$a=123000;</b>	123000	Coma flotante
Base Decimal	<b>\$a=0.174</b>	0.174	Coma flotante
Notación científica	<b>\$a=1.23e5;</b>	123000	Coma flotante
Notación científica	<b>\$a=23.4e-2;</b>	0.234	Coma flotante

## Cambios de base

PHP dispone de funciones que permiten obtener **una cadena de caracteres** con la expresión, en una **nueva base**, de un número escrito en **otra base cualquiera**.

Asignación de valores en distintas bases				
Valor de la variable	Base	Nueva base	Sintaxis	Expresión
\$a=1234	10	8	<b>decoct(\$a)</b>	2322
\$a=1234	10	16	<b>dechex(\$a)</b>	4d2
\$a=1234	10	2	<b>decbin(\$a)</b>	10011010010
\$a=1234	8	10	<b>octdec(\$a)</b>	668
\$a=1234	16	10	<b>hexdec(\$a)</b>	4660
\$a=1010011	2	10	<b>bindec(\$a)</b>	83
\$a=1234	7	14	<b>base_convert(\$a,7,14)</b>	254
\$a=1234	5	18	<b>base_convert(\$a,5,18)</b>	ae
\$a=1234	18	5	<b>base_convert(\$a,18,5)</b>	202123

## Formato de presentación de números

En PHP es posible establecer el formato de la presentación de los valores numéricos utilizando alguna de estas funciones.

**number\_format (número)**

Presenta la parte entera del número (sin decimales) y utiliza como separador de miles una coma (,).

**number\_format (número , número de cifras decimales)**

Presenta el número de cifras decimales que se indiquen y utiliza como separador decimal un punto (.) y el separador de miles es una coma (,).

**number\_format (número , núm decimales , "sep. decimal" , "sep. miles")**

Permite establecer el número de cifras decimales de la presentación así como el carácter que se establezca como separador de decimales y como separadores de miles.

¡Cuidado!. No te olvides de escribir los caracteres de separación *entre comillas*.

Aquí tienes algunos ejemplos.

### Formatos de presentación de número

## Presentaciones numéricas

La *presentación* de los valores numéricos permite una gran variedad de formatos.

El número de cifras decimales, los separadores de decimales y los separadores de mil pueden configurarse a voluntad.

Los ejemplos de aplicación y la sintaxis son los que tienes en la parte derecha de esta página.

Valor inicial	Nº de decimales	Sep. dec.	Sep. miles	Sintaxis	Resultado
\$a=1234567.234	0	,		number_format(\$a)	1,234,567
\$a=1234567.234	2	.	,	number_format(\$a,2)	1,234,567.23
\$a=1234567.234	1	,	.	number_format(\$a ,2 ,"," ,".")	1.234.567,2
\$a=1234567.234	1	'	esp	number_format(\$a ,2 ,"''"','"")	1 234 567'2

### Ejercicio nº 12

Modifica el [ejercicio11.php](#) de forma que los resultados obtenidos al realizar los cálculos aparezcan con un espacio como separador de miles, un punto como separador de decimales y cuatro cifras decimales.

Anterior



Índice



Siguiente





[Ver índice](#)

## Números aleatorios



### El origen de la semilla

#### El valor Unix Epoch

El conocido como *tiempo UNIX* –o también *Unix Epoch*- es un sistema referencia de tiempo cuya unidad son los segundos y que tiene su valor cero a las **0:00:00 horas (GMT)** del día uno de enero de 1970.

#### La función `time()`

La función `time()` devuelve **una cadena** con el número de segundos transcurridos desde el comienzo de la **Unix Epoch**.

¿Que cuantos son? Pues mira, hasta este mismo instante han transcurrido:

1 243 523 762 segundos

desde el comienzo del *tiempo UNIX*.

#### La función `microtime()`

Usando la función `microtime()` se obtiene **una cadena** a la que, además del número de segundos transcurridos desde el comienzo de la **Unix Epoch**, se añade – al comienzo de ella y separado por un espacio– la parte decimal de ese tiempo expresada en microsegundos.

Este es el valor de la cadena devuelta por `microtime()` en este instante:

0.39062800 1243523762

donde, como verás, aparece la fracción decimal del tiempo *Unix Epoch* delante de su valor entero.

#### (double)`microtime()`

Dado que con `microtime()` obtenemos **una cadena**, es posible convertirla en número de coma flotante anteponiendo a esa función `(double)`.

Una vez cambiado el tipo de variable el valor devuelto por `microtime` se convierte en:

0.390663

que como puedes observar es un número con **seis decimales** (si las últimas cifras son ceros no se visualizarán).

Si se multiplica el valor anterior por **1.000.000** obtenemos un número de **seis cifras** y valor entero que puede cambiar de valor cada **millonésima**

## Números aleatorios

PHP dispone de dos funciones capaces de generar números aleatorios. Se trata de la función `rand()` y de la **función mejorada `mt_rand()`**.

El valor **mínimo** del intervalo que contiene los números aleatorios que pueden ser generados es **CERO** en ambos casos y los valores **máximos** de cada opción pueden determinarse mediante las funciones `getrandmax()`, para el primer generador, y `mt_getrandmax()`, para el segundo.

Veamos cuales son esos valores en cada uno de los casos.

Valores máximos de los generadores de números aleatorios			
Generador <code>rand()</code>	Generador <code>mt_rand()</code>		
Sintaxis	Valor máximo	Sintaxis	Valor máximo
<code>echo getrandmax()</code>	<b>32767</b>	<code>echo mt_getrandmax()</code>	<b>2147483647</b>

Según las librerías que esté usando PHP, puede ocurrir que los valores máximos con ambos generadores sean iguales.

## La forma más simple

La forma más simple -y más desaconsejable- de generación de un número aleatorio es esta:

Generación de un número aleatorio			
Generador <code>rand()</code>	Generador <code>mt_rand()</code>		
Sintaxis	Nº aleatorio	Sintaxis	Nº aleatorio
<code>echo rand()</code>	<b>26474</b>	<code>echo mt_rand()</code>	<b>299791159</b>

## Una semilla que mejora la aleatoriedad

Si antes de escribir la función `rand()` en un caso, o `mt_rand()` en el otro escribimos:

`srand((double)microtime()*1000000)`, en el primer caso y/o

`mt_srand((double)microtime()*1000000)`, en el segundo

estaremos introduciendo una **semilla** (al margen comentamos la forma en que se genera) que mejora sustancialmente la aleatoriedad del los números obtenidos

[Ver índice](#)

[Búsqueda rápida](#)

[Página anterior](#)

[Página siguiente](#)

Generación de un número aleatorio con semilla		
Generador <code>rand()</code>	Generador <code>mt_rand()</code>	
Sintaxis	Nº aleatorio	Nº aleatorio
<code>srand((double)microtime()*1000000); echo rand()</code>		<b>11892</b>
Generador <code>mt_rand()</code>		
Sintaxis	Nº aleatorio	
<code>mt_srand((double)microtime()*1000000); echo mt_rand()</code>		<b>1186193619</b>

Manteniendo la sintaxis anterior -no te olvides de las **semillitas famosas**- se pueden generar números aleatorios comprendidos dentro del intervalo que preestablezcamos.

de segundo.

390683

¡Fíjate que ha cambiado! La diferencia no es otra cosa que el tiempo transcurrido entre los instantes en que se ejecutaron ambas instrucciones.

Este número entero es utilizado por **srand** y **mt\_rand** como *semilla* generadora de números aleatorios.

Bastaría con añadir los valores de los **extremos** de ese **intervalo** como parámetros de la función.  
La sintaxis sería esta:

**rand(extremo inferior , extremo superior)**

y para la función mejorada

**mt\_rand(extremo inferior , extremo superior)**

Generación de un número aleatorio delimitando intervalos	
Generador <b>rand()</b>	
Sintaxis	Nº aleatorio
<b>srand((double)microtime()*1000000); echo rand(1,300)</b>	<b>104</b>
Generador <b>mt_rand()</b>	
Sintaxis	Nº aleatorio
<b>mt_srand((double)microtime()*1000000); echo mt_rand(1,300)</b>	<b>75</b>

Anterior

Índice

Siguiente

[Ver índice](#)

## Concatenación de cadenas



### Las variables de cadena

A las variables tipo **cadena** se les puede asignar los valores de dos formas:

Escribiendo el *contenido entre comillas*:

```
$var="Texto del contenido";
```

y por medio de la *sintaxis de documento incrustado* que es la siguiente:

**\$var= <<< EOD**

... contenido de la cadena...

... puede ir ....

.. en varias líneas...

**EOD;**

donde **EOD** es una *palabra cualquiera* que debe repetirse exactamente *igual* al final de la instrucción.

### Peculiaridades

El nombre de la variable, el signo igual que la precede, los tres símbolos < y el **EOD** deben escribirse en la *misma línea*, que esta vez **no** irá acabada en *punto y coma* (observa que no acaba allí la instrucción).

Puede incluirse el texto (valor de la variable) en tantas líneas como se desee, pero hay que tener en cuenta, que a la hora de visualizar el documento *no se mantiene esa estructura* ya que HTML sólo entiende la etiqueta <BR> como indicador de salto de línea.

El *cierre* de la instrucción debe hacerse —siempre— escribiendo el **EOD** en una nueva línea —independiente— que ahora sí tiene que llevar el *punto y coma* que indica a PHP el final de una instrucción.

### Escribiendo cadenas

Aquí tienes un ejemplo en el que se utilizan las dos formas de asignación de valores a una cadena.

```
<HTML>
<HEAD>
<TITLE>Ejemplo 17 - PHP</TITLE>
</HEAD>
<BODY>
<?
$cadena1="Esto es una cadena de texto";
```

[Ver índice](#)[Búsqueda rápida](#)[Página anterior](#)[Página siguiente](#)

Se escribe en varias líneas y tiene la sintaxis siguiente. *Después de escribir el nombre de la variable y el signo igual se ponen los tres <<< y un nombre cualquiera*. En este caso, **Pepe**. Luego hay que saltar de linea y escribir el texto con las líneas que se desee, pero cuidado... a la hora de visualizar la cadena con la instrucción echo todo este texto se verá seguido ya que para que se visualizaran saltos de línea en una página web habría que poner las famosas etiquetas <BR>. Se indica el *final de la cadena* escribiendo de nuevo el nombre asignado en la primera línea -**Pepe**- pero teniendo la precaución de escribirlo **en una linea nueva** al final de todo el texto... Así como lo ves en el código fuente.

**Pepe;**

**\$cadena3= <<<Pepe**

Esta es otra cadena con el nombre Pepa puedo escribir Pepa cuantas veces quiera porque el PHP no interpretará el final de documento incrustado hasta que no la escriba en una sola linea y seguida del punto y coma **Pepe;**

```
echo $cadena1,"<BR>";
echo $cadena2,"<BR>";
echo $cadena3,"<BR>";
?>
</BODY>
</HTML>
```

[ejemplo17.php](#)[Anterior](#)[Índice](#)[Siguiente](#)



[Ver índice](#)

# Operaciones con cadenas



## La concatenación de cadenas

Para concatenar (unir), en una sola, varias *porciones* de texto hemos venido utilizando –en las instrucciones *print* y *echo*– un punto (.).

### El operador .

Este *punto* es un elemento muy importante que, además de la forma que hemos visto en las páginas anteriores, tiene los siguientes usos:

#### Unir dos cadenas y recogerlas en una variable

Con la sintaxis:

`$a="cad1" . "cad2";`

o mediante

`$a= $b . $c`

podemos obtener una nueva variable formada por la *unión* dos trozos. Pero seguramente te preguntarás *¿qué ocurre si juntamos una variable cadena y una numérica? o ¿qué ocurre si juntamos dos variables numéricas?*

En cualquiera de los supuestos –puedes verlo en el ejemplo– las variables serán tratadas por PHP –con independencia de lo que puedan contener– como de tipo *cadena* y la variable que contiene el resultado es del tipo *string*.

#### Añadir contenidos a una variable tipo string

Si utilizamos una sintaxis como esta:

`$a .= "cad1"  
o de este otro tipo  
$a .=$b`

(presta mucha atención al punto que va delante del signo igual) se *añadiría* al **valor actual** de la variable **\$a** el contenido indicado después del signo igual.

Fíjate en la importancia del punto. Si está presente, se *añaden* nuevos contenidos a la variable; pero en el caso de que no lo estuviera, se *sustituirían* esos contenidos, con lo cual se perdería la información que esa variable pudiera contener.

## Uniendo cadenas

Aquí tienes un ejemplo de concatenación de variables tipo *string*.

```
<?
#definamos y asignemos valores a variables tipo cadena
$cadena1="Esto es una cadena de texto";
$cadena2="Esta es una segunda cadena de texto";

#hagamos lo mismo con variables numéricas
$cadena3=127;
$cadena4=257.89;

# unámoslas mezclando tipos

$union1=$cadena1 . $cadena2;
$union2=$cadena1 . $cadena3;
$union3=$cadena3 . $cadena4;

#veamos que ha ocurrido

echo $union1,"<br>";
echo $union2,"<br>";
echo $union3,"<br>";

# modifiquemos ahora una cadena
# añadiéndole contenidos

$cadena3 .= " Este es el texto que se añadirá a la variable cadena3";
# imprimamos los resultados
echo $cadena3,"<br>";
# añadimos ahora un nuevo trozo, esta vez
# a partir de una cadena escrita con la
# sintaxis de documento incrustado
$cadena3 .= <<<Pepito
Ahora le añado a la cadena
este trocillo asignado con el "formato"
de documento incrustado
Pepito;
# visualicemos el resultado
echo $cadena3,"<br>";
?>
```

`ejemplo18.php`

### ¡Cuidado!

Observa en el ejemplo que, excepcionalmente, la *sintaxis de documento incrustado* permite introducir comillas (sin ningún método especial), pero recuerda que en cualquier otro caso hay que recurrir al truco del que hablábamos en [aquí](#).

## Ejercicio nº 13

Utiliza los dos métodos de creación de variables de cadena y lo relativo al tratamiento de caracteres especiales para crear un script –al que llamaremos **ejercicio13.php**– que permita explicar el significado de las etiquetas **<BODY>**, **<HEAD>** y **<HTML>**. Resalta, con comillas, las palabras de tu explicación que consideres más importantes.

[Anterior](#)

[Índice](#)

[Siguiente](#)





Ver índice

# Array escalar y asociativo



## ¿Qué es un array?

Un **array** es sencillamente una tabla de valores.

Cada uno de los elementos de esa **tabla** se identifica por medio de un **nombre** (común para todos) y un **índice** (que diferenciaría a cada uno de ellos).

La sintaxis que permite definir elementos en un array es esta:

**\$nombre[indice]**

**\$nombre** utiliza exactamente la misma sintaxis empleada para definir variables, con la única particularidad de que ahora deben añadirse los corchetes y los índices.

El **índice** puede ser *un número* (habría que escribirlo dentro del corchete *sin comillas*), *una cadena* (que habría que poner en el corchete encerrada entre *comillas sencillas* *'-'*), o *una variable PHP* en cuyo caso tampoco necesitaría ir entre comillas.

Cuando los *índices* de un array son *números* se dice que es **escalar** mientras que si fueran *cadenas* se le llamaría array **asociativo**.

## Arrays escalares

Los elementos de un **array escalar** puede escribirse con una de estas sintaxis:

**\$a[]=valor**

ó

**\$a[xx]=valor**

En el primero de los casos PHP asigna los índices de forma automática atribuyendo a cada elemento el valor **entero siguiente** al último asignado.

Si es el **primero** que se define le pondrá índice **0** (CERO).

En el segundo de los casos, seremos nosotros quienes pongamos (**xx**) el **número** correspondiente al **valor del índice**.

Si ya existiera un elemento con ese índice, se cambiaría el valor de su contenido, en caso contrario crearía un nuevo elemento del array y se le asignaría como valor lo especificado detrás del signo igual que de los

## Tablas (arrays) unidimensionales

Mediante el uso de arrays podemos utilizar el **mismo nombre** para varias variables **diferenciándolas entre sí** mediante **índices distintos**

Tablas unidimensionales					
Array escalar			Array asociativo		
Variable	Indice	Valor	Variable	Indice	Valor
\$a[0]	0	Domingo	\$a['Primero']	Primero	Domingo
\$a[1]	1	Lunes	\$a['Segundo']	Segundo	Lunes
\$a[2]	2	Martes	\$a['Tercero']	Tercero	Martes
\$a[3]	3	Miércoles	\$a['Cuarto']	Cuarto	Miércoles
\$a[4]	4	Jueves	\$a['Quinto']	Quinto	Jueves
\$a[5]	5	Viernes	\$a['Sexto']	Sexto	Viernes
\$a[6]	6	Sábado	\$a['Septimo']	Septimo	Sábado

## Uso de arrays

```
<?
# Crearemos un array escalar (basta con definir un elemento)
$a[2]="Este elemento es el segundo del array";
# creamos un nuevo elemento de ese array
# esta vez de forma automática
# si ponemos corchetes vacíos va añadiendo índices automáticamente
$a[]="¿Será este tercero?";
# comprobemos que le ha puesto índice 3
echo "El elemento ".$a[3]." tiene índice 3 (siguiente a 2) <br>";
# ahora insertemos un nuevo elemento con índice 32
$a[32]="Mi índice es 32";
# insertemos otro elemento de forma automática
$a[]="¿Irá a parar al índice 33 este elemento?";
# la inserción se hará con índice 33, comprobémoslo
print "Vemos que contiene el elemento de índice 33 ...".$a[33]."<br>";
# ¿qué ocurrirá si pido que imprima el elemento 21 que nadie ha definido
# seguramente estará vacío, ¡comprobémoslo!
print ("Aquí--> ".$a[21]. "<--- si es que hay algo<br>");
# ahora crearemos un nuevo array llamado $b
# insertémosle de forma automática su PRIMER elemento
$b[]="Estoy empezando con el array b y mi índice será cero";
# comprobemos que efectivamente ha empezado con índice CERO
print ($b[0]."<br>");
# veamos ahora eso de los arrays asociativos
# creamos uno llamado $c con varios elementos
$c["objeto"]="coche";
$c["color"]="rojo";
$c["tamaño"]="ideal";
$c["marca"]="Ferrari";
$c["precio"]="prohibitivo para un humilde docente";
# encadenemos variables para hacer una salida
# pondremos cadenas " " para que no aparezcan los textos
# pegados unos a otros..
$salida=<H2> El ". $c["objeto"] . " ".$c["marca"].".$c["color"];
$salida .= " tiene el tamaño ideal ".$c["tamaño"];
$salida .= " y su precio es ".$c["precio"];
$salida .=</H2>;
print $salida;
# sigamos experimentando ahora
# ¿qué ocurriría si nos olvidamos de poner nombre al índice
# e insertamos un corchete vacío ¿lo crearía?; que índice pondría?
# probemos ....
$c[]="¿creará un array escalar nuevo y le pondrá índice cero?";
# tratemos ahora de visualizar esa variable
```

misma forma que ocurría con las variables— debería ir entre comillas si fuera una cadena o sin ellas, si se tratara de números.

## Arrays asociativos

Los elementos de un **array asociativo** pueden escribirse usando la siguiente sintaxis:

**\$a[índice]=valor**

En este caso estamos obligados a escribir el nombre del índice que habrá de ser una **cadena** y debe ponerse entre comillas.

Tanto en este supuesto como en el anterior, es posible —y bastante frecuente— utilizar como índice el contenido de una variable. El modo de hacerlo sería:

**\$a[\$ind]=valor**

En este caso, sea cual fuere el valor de la variable **\$ind**, el nombre de la variable **nunca** se pone entre comillas.

```
# probemos a escribir $c[0] porque PHP  
# habrá entendido que queremos un array escalar  
# y como no existe ninguno con ese nombre empezará por cero  
# comprobémoslo  
echo $c[0];  
?>
```

ejemplo19.php

## Ejercicio nº 14

Crea un formulario -puedes llamarlo **formulario14.php**- en el que se permita introducir un número cualquiera en una caja de texto. Al enviar el formulario deberá aparecer -un mensaje **en letra**- indicando **el resto de dividir entre doce** el valor transferido.

El procedimiento podría ser el siguiente:

- Crear un array conteniendo los nombres de todos restos posibles (división exacta, uno, dos, tres... hasta once).
- Asignar como índices los valores numéricos correspondientes a los literales que contiene cada elemento.
- Comprobar el resto de la división e imprimir el valor del elemento del array cuyo índice coincide con ese resto.

Anterior



Índice



Siguiente





Ver índice

# Arrays bidimensionales



## Arrays bidimensionales

Los *arrays bidimensionales* pueden entenderse como algo muy similar a una *tabla de doble entrada*.

Cada uno de los elementos se identifica –sigue siendo válido el nombre único que se usaba en los unidimensionales – por un nombre (*\$nombre*) seguido de dos (*[]*) que contienen los *índices* (en este caso son dos índices) del array.

Los *índices* pueden ser de tipo **escalar** –equivalentes al número de fila y columna que la celda ocupa en la tabla– o puede ser **asociativos** lo que equivaldría en alguna medida a usar como índices los nombres de la fila y de la columna.

**J**¡Cuidado!

No dejes de tener en cuenta lo que hemos advertido al hablar de arrays unidimensionales.

En este supuesto, también, se empiezan a numerar los arrays escalares a partir de **CERO**.

## Arrays escalares

Los elementos de un **array bidimensional** escalar pueden escribirse usando una de estas sintaxis:

```
$a[][]=valor
o
$a[xx][]=valor
o
$a[][xx]=valor
o también
$a[xx][yy]=valor
```

En el primero de los casos PHP asigna automáticamente como **primer índice** el valor que sigue al último asignado y, si es el **primero** que se define, le pondrá como índice **0** (CERO).

Sea cual fuere el valor de primer índice al **segundo** se le asignará **cero** ya que es en este mismo momento cuando se habrá creado el **primero** y, por tanto, **aún carecerá de elementos**.

En el segundo de los casos, asignamos un valor al **primer índice** (**xx**) y será el segundo quien se incremente en **una unidad** respecto al de valor más alto de todos aquellos cuyo **primer índice** coincide con el especificado.

## Arrays bidimensionales

Como ejemplo de array bidimensional emplearemos una tabla de resultados de una *liga de fútbol* en la que intervienen **cinco equipos** que –como en toda liga que se precie– se juega a *doble partido*.

En este primer supuesto utilizaremos **arrays escalares**, por lo tanto los equipos serán identificados con números desde **cero** hasta **cuatro**.

```
<?
# rellenamos el array desde [0][0] hasta [0][4]
# la inserción automática haría que este primero fuera [0][0]

$a[][]=" ";
# ahora pondremos cero como índice del primer array y dejemos que PHP
# nos vaya insertando automáticamente el segundo

$a[0][]="3-2";$a[0][]="5-3";$a[0][]="7-1";$a[0][]="0-2";

#ahora desde [1][0] hasta [1][4]
#este primero lo dejamos como automático en ambos índices
# de esta forma el primero tomará valor uno (siguiente al anterior)
# de forma automática
$a[][]="0-11";
# repetimos el proceso anterior
$a[1][]=" ";$a[1][]="2-1";$a[1][]="1-0";$a[1][]="1-2";
# y repetimos de nuevo, ahora crearía 2 como primer índice
$a[][]="0-0";
#insertariamos los restantes valores de índice 2
$a[2][]="1-3";$a[2][]=" ";$a[2][]="1-4";$a[2][]="2-0";
# nuevo incremento del primer índice
$a[][]="1-0";
# rellenamos
$a[3][]="6-3";$a[3][]="14-3 ";$a[3][]=" ";$a[3][]="1-0";
# nuevo y último incremento de primer índice
$a[][]="1-1";
# rellenamos de nuevo
$a[4][]="2-3";$a[4][]="0-1 ";$a[4][]="1-1";$a[4][]="";

# como verás el proceso no tiene complicaciones, pero ... pesadillo si es
# ¿verdad que si tuviéramos una base de datos sería más fácil?
# estamos en ello, todo se andará...

# tendríamos que ver esos valores pero.. escribir "a mano"
# una tabla puede ser una tortura, así que mejor introducimos
# una bucle, otro recurso que estudiaremos pronto
# para esa labor repetitiva de mostrar en una tabla
# todos los datos del array

# Sería algo como esto
# creamos la etiqueta de apertura de una tabla

print ("<TABLE BORDER=2>");
# ahora dos bucles anidados (rojo uno, magenta el otro)
# para llenar las celdas de cada fila (el magenta)
# y para insertar las etiquetas <TR> utilizaremos el rojo

for ($i=0;$i<5;$i++){
    print("<tr>");
    for($j=0;$j<5;$j++) {
        print("<td>".$a[$i][$j]."</td>");
    }
}

#ponemos la etiqueta de cierre de la tabla

print("</table>");
?>
```

La tercera opción es bastante similar a la anterior. Ahora se modificaría automáticamente el primer índice y se escribiría el contenido (`xx`) como valor del segundo.

En la cuarta de las opciones se asignan libremente cada uno de los índices (`xx` e `yy`) poniéndoles valores numéricos.

### Arrays asociativos

Los elementos de un *array* asociativo **bidimensional** se pueden escribir usando la siguiente sintaxis:

```
$a["indice1"]["indice2"] = valor
```

En este caso, los índices serán **cadenas** y se escribirán entre comillas.

### Arrays mixtos

PHP permite utilizar también arrays *mixtos*. Sería este el caso de que uno de ellos fuera escalar y el otro asociativo.

Igual que ocurría con los unidimensionales, también aquí podemos utilizar valores de variables como índices.

Utilizando el script anterior, con ligeros *retoques estéticos*, hemos construido esta tabla:

Todos los resultados de la liguilla					
Indice	0	1	2	3	4
0		3-2	5-3	7-1	0-2
1	0-11		2-1	1-0	1-2
2	0-0	1-3		1-4	2-0
3	1-0	6-3	14-3		1-0
4	1-1	2-3	0-1	1-1	

Con el mismo procedimiento –en este caso hemos usado resultados diferentes– hemos construido esta otra tabla

Resultados de la liguilla					
Indice	Juvencia	Mosconia	Canicas	Condal	Piloñesa
Juvencia		3-2	5-3	7-1	0-2
Mosconia	0-11		2-1	1-0	1-2
Canicas	0-0	1-3		1-4	2-0
Condal	1-0	6-3	14-3		1-0
Piloñesa	1-1	2-3	0-1	1-1	

¿Por qué no intentas modificar el script y tratas de reproducir estas tablas? Desde luego, es sólo una sugerencia.

Anterior



Índice



Siguiente



## Funciones de cadenas

Como complemento a las descritas en la página anterior, añadimos aquí algunas otras funciones PHP que también permiten manejar *cadenas de caracteres*.

### AddSlashes(cadena)

Inserta el carácter \ delante los siguientes: ", ', \ y **NUL** (el bit nulo).

### stripslashes(cadena)

Quita las marcas añadidas a una cadena con la función **AddSlashes()**.

### chunk\_split(cad, n, sep)

Devuelve la cadena (**cad**) después de haberle insertado, cada **n** caracteres, la cadena indicada en el parámetro **sep**.

Si no se indica **sep** PHP pondrá un **espacio**.

Si no se establece el parámetro **n** insertará el separador cada **76** caracteres.

Esta función coloca siempre *un separador al final de la cadena*.

### parse\_str(cadena)

Devuelve las variables –con su valor– indicadas dentro de la cadena (observa la sintaxis del ejemplo).

Dentro de la cadena cada variable se denomina con un nombre que va seguido de un signo igual. Los espacios se señalan con el signo + y los separadores de variables son signos &.

### explode(sep, cad,n)

Devuelve un **array** cuyos elementos contienen cada una de las porciones de la cadena (**cad**) comprendidas entre dos de los caracteres señalados como (**sep**) hasta el máximo de porciones señaladas (**n**). Los caracteres separadores no son incluidos en las cadenas resultantes. Si no se indica la cantidad de porciones, será fraccionada toda la cadena.

Si se indica número, el último *trozo* contendrá *toda* la cadena restante.

### implode(sep, array)

Devuelve una cadena formada por todos los elementos del **array** separados mediante los caracteres indicados en **sep**.

*join/can array*

## Formatos de salida

Estos son algunos ejemplos en los que continuamos con las aplicaciones de las funciones PHP al manejo de cadenas:

Marcas, divisiones y uniones de cadenas		
Variable cadena	Sintaxis	Resultado
\$a="Esta 'y \' y también NUL"	AddSlashes(\$a)	\$a="Esta ' l\l y también NUL"
\$a="Esta ' y \\ y también NUL"	stripslashes(\$a)	Esta ' y también el nulo
\$a="Esta es una cadena larga que presuntamente será troceada"	chunk_split(\$a,5,"")	Esta -es un-a cad-en-a l-arg-a -que p-resun-tamen-te se-rá tr-ocead-a-
\$a="Esta es una cadena larga que presuntamente será troceada"	chunk_split(\$a,5)	Esta es un a cad ena l arga que p resun tamen te se rá tr ocead a
\$a="Esta es una cadena larga que presuntamente será troceada"	chunk_split(\$a,76,"")	Esta es una cadena larga que presuntamente será troceada-
\$todo="v1=Esto+sera+una+variable&v2=est0+otra&p[]= <b>incluso+un+array</b> "		
Divide la cadena <b>\$todo</b> en sus componentes	parse_str(\$todo); echo \$v1; echo \$v2; echo \$p[0];	Esto sera una variable esto otra incluso un array
\$a="Esta cadena sera devuelta en trozos"		
Recogerá en un array cada uno de los trozos delimitados por los separadores	\$trozo1=explode(" ",\$a); echo \$trozo1[0]; echo \$trozo1[1]; echo \$trozo1[2]; echo \$trozo1[3]; echo \$trozo1[4]; echo \$trozo1[5]; >	Esta cadena sera devuelta en trozos
Recogerá en un array cada uno de los trozos delimitados por los separadores	\$trozo2=explode("a",\$a); echo \$trozo2[0]; echo \$trozo2[1]; echo \$trozo2[2]; echo \$trozo2[3]; echo \$trozo2[4]; echo \$trozo2[5];	Est c den ser devuel en trozos
Recogerá en un array cada uno de los trozos delimitados por los separadores hasta un máximo de 3	\$trozo3=explode(" ",\$a,3); echo \$trozo3[0]; echo \$trozo3[1]; echo \$trozo3[2]; echo \$trozo3[3]; echo \$trozo3[4]; echo \$trozo3[5];	Esta cadena sera devuelta en trozos
Recogerá en un array cada uno de los trozos delimitados por los separadores hasta un máximo de 3	\$trozo4=explode("a",\$a,3); echo \$trozo4[0]; echo \$trozo4[1]; echo \$trozo4[2]; echo \$trozo4[3]; echo \$trozo4[4]; echo \$trozo4[5];	Est c dena sera devuelta en trozos
implode(" ",\$trozo1)		Esta cadena sera devuelta en trozos
implode("**",\$trozo2)		Est* c*den* ser* devuel* en trozos
implode("-",\$trozo3)		Esta-cadena-sera devuelta en trozos
implode(":",\$trozo4)		Est: c:dena sera devuelta en trozos
join(" ",\$trozo1)		Esta cadena sera devuelta en trozos
join("**",\$trozo2)		Est* c*den* ser* devuel* en trozos
join("-",\$trozo3)		Esta-cadena-sera devuelta en trozos
join(":",\$trozo4)		Est: c:dena sera devuelta en trozos
\$cadena="Esta cadena será dividida con la función strtok"		
\$strocin = strtok (\$cadena, " ");  while (\$strocin) { echo "\$strocin "; \$strocin = strtok (" "); }		Esta cadena será dividida con la

Es idéntica a **implode**.

### strtok(cad , sep)

Esta función divide la cadena **cad** en trozos delimitados por el separador que se indica en **sep**.

Cuando se invoca la primera vez –extrae el primer trozo– debe llevar las sintaxis **strtok(cadena,sep)**.

Al invocarla sucesivamente, se escribe **solo strtok (" ")** e irá recogiendo de forma secuencial los trozos sucesivos.

## Encriptación de cadenas

PHP dispone de funciones que permiten *codificar* o *encriptar* cadenas de caracteres.

### bin2hex(cadena)

Devuelve una cadena ASCII que contiene la representación **hexadecimal** de la **cadena**. La conversión se realiza byte a byte, con los 4 bits superiores primero.

### crypt(cadena)

Devuelve la cadena *encriptada* utilizando una *semilla aleatoria* de dos caracteres.

Por su carácter aleatorio, si se ejecuta dos veces seguidas –tal como puedes observar en el ejemplo– dará dos resultados diferentes.

### crypt(cadena,"xx")

Devuelve la cadena *encriptada* utilizando como *semilla* los dos caracteres (entre comillas) que se escriben como segundo parámetro de la función.

Tanto en este supuesto como en el anterior, los dos primeros caracteres de la cadena encriptada coinciden con los que han sido utilizados como *semilla*.

### md5(cadena,"xx")

Aplica el algoritmo **md5** –lo comentamos a la derecha– y devuelve la *huella digital* generada por él.

### crc32(cadena)

Aplica el algoritmo **crc32** de *comprobación de integridad* y devuelve el valor del mismo.

Se utiliza muchísimo en los programas de *compresión* y *descompresión* de ficheros. Se aplica en el momento de *comprimir* y se incluye el valor obtenido dentro del fichero comprimido. Después de la *descompresión* se vuelve a aplicar el mismo algoritmo y se comparan ambos valores. La coincidencia será

		función strtok
\$trocin = strtok (\$cadena," "); echo \$trocin," "; \$trocin1 = strtok (""); echo \$trocin1," "; \$trocin2 = strtok (""); echo \$trocin2," ";		Esta cadena Esta
\$trocin = strtok (\$cadena,"a"); while (\$trocin) { echo "\$trocin "; \$trocin = strtok ("a"); }		Est c den será dividid con l función strtok
<b>Encriptaciones y codificaciones</b>		
Variable cadena	Sintaxis	Resultado
\$a="Esta es la cadena"	bin2hex(\$a)	45737461206573206c6120636164656e610a
\$a="Encriptame"	crypt(\$a)	\$1\$jQ5.w22.\$QPBJQ/mNSxxdZ2ccJpZo41
\$a="Encriptame"	crypt(\$a)	\$1\$3v1.O.5.\$MxTxISqC2VwZiDyZjdz0oo
\$a="Encriptame"	crypt(\$a,"zq")	zqQ4qOeELzPFg
\$a="Encriptame"	crypt(\$a,"zq")	zqQ4qOeELzPFg
\$a="Encriptame"	crypt(\$a,"@\$")	@\$eKFJms3tL.
\$a="Encriptame"	md5(\$a)	67c3ca0aefda2595138168a85e7b33a0
\$a="Encriptame"	md5(\$a)	67c3ca0aefda2595138168a85e7b33a0
\$a="Encriptame"	crc32(\$a)	-1128189886
<b>Búsquedas y recuentos de caracteres</b>		
Variable cadena	Sintaxis	Resultado
\$a="Contando caracteres"	count_chars(\$a,0)	Array
\$a="Contando caracteres"	\$b=count_chars(\$a,0); echo \$b[97];	3
\$a="Contando caracteres"	\$b=count_chars(\$a,0); echo \$b[ord("o")]	2
\$a="Pepe Perez el perverso pecador en penitencia"	substr_count(\$a,"Pe");	2
\$a="Pepe Perez el perverso pecador en penitencia"	substr_count(\$a,"pe");	4

La función **count\_char(\$a,0)** devuelve un **array** cuyos *índices* son los *códigos ASCII* de los caracteres y cuyos *valores* son el *número de veces* que se repite cada uno de ellos.

La función **substr\_count(\$a,"cadena")** determina el *número de veces* que aparece la **cadena** dentro de **\$a**. Diferencia entre mayúsculas y minúsculas.

## El algoritmo md5

Este algoritmo presenta como peculiaridades que –tenga la dimensión que tenga la cadena a la que se aplique– genera siempre una *huella digital* que no es otra cosa que una cadena formada por **32 caracteres** y que **no dispone** de ningún mecanismo *inverso*.

Seguramente habrás vivido esta experiencia. En muchos espacios de Internet –grupos de noticias, cuentas de correo web, etcétera– que requieren un login y una contraseña cuando utilizas la opción de *recuperar contraseña* no te envían tu contraseña anterior, sino que te generan y envían una nueva.

Esto ocurre porque, por razones evidentes de seguridad, las contraseñas se almacenan usando estas *huellas digitales* y resulta imposible recuperar los valores originales.

La única solución en estos casos es crear una nueva contraseña (suelen hacerlo con las funciones de números aleatorios), enviarla de forma automática por correo electrónico y sustituir el valor anterior del registro de usuarios por el resultado de la codificación **md5** de la nueva contraseña.

Anterior

Índice

Siguiente

la garantía de que el fichero obtenido  
es idéntico al original.



Ver índice

# Arrays multidimensionales



## Arrays multidimensionales

PHP permite el uso de arrays con dimensión superior a dos. Para modificar la dimensión del array basta con ir añadiendo nuevos índices.

**\$a[x][y][z]=valor;**

asignaría un valor al elemento de índices **x**, **y** y **z** de un array tridimensional y

**\$a[x][y][z][w]=valor;**

haría lo mismo, ahora con un array de dimensión **cuatro**.

Pueden tener cualquier tipo de **índices**: **escalares**, **asociativos** y, también, **mixtos**.

## La función **array()**:

Para asignar valores a una matriz puede usarse la función **array()**, que tiene la siguiente sintaxis:

```
$a= array (
    índice 0 => valor,
    ....,
    índice n => valor,
);
```

Por ejemplo:

```
$z=array (
    0 => 2,
    1 => "Pepe",
    2 => 34.7,
    3 => "34Ambrosio",
);
```

producirá igual resultado que:

```
$z[0]=2;
$z[1]="Pepe";
$z[2]=34.7;
$z[3]="34Ambrosio";
```

## Anidando en **array()**:

La función **array()** permite escribir arrays de cualquier dimensión utilizando la técnica de **anidado**.

Si pretendemos escribir los elementos de este array:

```
$z[0][0]=34;
$z[0][1]=35;
$z[0][2]=36;
$z[1][0]=134;
$z[1][1]=135;
$z[1][2]=136;
```

podríamos hacerlo así:

## Arrays multidimensionales

Esta es la forma en la que hemos definido el array tridimensional que utilizaremos en el ejemplo.

```
<?
$b = array(
    "Juvencia" => array(
        "Juvencia" => array (
            "Resultado" => " ",
            "Amarillas" => " ",
            "Rojas" => " ",
            "Penalty" => " "
        ),
        "Mosconia" => array (
            "Resultado" => "3-2",
            "Amarillas" => "1",
            "Rojas" => "0",
            "Penalty" => "1"
        ),
        "Canicas" => array (
            "Resultado" => "5-3",
            "Amarillas" => "0",
            "Rojas" => "1",
            "Penalty" => "2"
        ),
        "Condal" => array (
            "Resultado" => "7-1",
            "Amarillas" => "5",
            "Rojas" => "2",
            "Penalty" => "1"
        ),
        "Piloñesa" => array (
            "Resultado" => "0-2",
            "Amarillas" => "1",
            "Rojas" => "0",
            "Penalty" => "0"
        ),
        "Mosconia" => array(
            "Juvencia" => array (
                "Resultado" => "0-11 ",
                "Amarillas" => "4",
                "Rojas" => "2",
                "Penalty" => "4"
            ),
            "Mosconia" => array (
                "Resultado" => " ",
                "Amarillas" => " ",
                "Rojas" => " ",
                "Penalty" => " "
            ),
            "Canicas" => array (
                "Resultado" => "2-1",
                "Amarillas" => "0",
                "Rojas" => "0",
                "Penalty" => "2"
            ),
            "Condal" => array (
                "Resultado" => "1-0",
                "Amarillas" => "1",
                "Rojas" => "0",
                "Penalty" => "0"
            ),
            "Piloñesa" => array (
                "Resultado" => "1-2",
                "Amarillas" => " "
            )
        )
    )
);
```

```

$z=array(
  0 => array (
    0 => 34,
    1 => 35,
    2 => 36,
  ),
  1 => array (
    0 => 134,
    1 => 135,
    2 => 136,
  )
);

```

Como puedes observar, se trata de sustituir los *valores* asignados a los elementos de una primera función **array()** por otra nueva función **array** que contiene los segundos índices así como los valores asignados a los mismos.

El anidado sucesivo permitiría generar arrays de cualquier dimensión.

Aunque en el ejemplo anterior nos hemos referido a un **array escalar**, idéntico procedimiento sería válido para **arrays asociativos** con sólo cambiar los números por **cadenas** escritas entre **comillas**.

Este podría ser un ejemplo de **array asociativo**:

```

$z["a"]["A"]=34;
$z["a"]["B"]=35;
$z["a"]["C"]=36;
$z["b"]["A"]=134;
$z["b"]["B"]=135;
$z["b"]["C"]=136;

```

que podría definirse también de esta forma:

```

$z=array(
  "a" => array (
    "A" => 34,
    "B" => 35,
    "C" => 36,
  ),
  "b" => array (
    "A" => 134,
    "B" => 135,
    "C" => 136,
  )
);

```

A medida que la dimensión se hace mayor la sintaxis requiere muchísima más atención y los errores son poco menos que inevitables. **Refresquemos** un poco la memoria.

No olvides *los punto y coma* del final de las instrucciones.

Cuidado con las **formas anidadas** y también con los **paréntesis**.

Cierra cada uno de los paréntesis que abras y no olvides que los paréntesis se **anidan**, ya sabes... el primero que se abre siempre con el último que se cierra, el segundo con el penúltimo, etcétera.

No dejes de prestar atención a las comillas. Recuerda que hay que cerrarlas siempre y que hay que diferenciarlas en los casos en que van **comillas dentro de otras**

```

  "Amarillas" => "1",
  "Rojas"      => "0",
  "Penalty"    => "0"
),
"Canicas"  => array(
  "Juvencia" => array (
    "Resultado" => "0-0",
    "Amarillas" => "0",
    "Rojas"     => "1",
    "Penalty"   => "1"
  ),
  "Mosconia"  => array (
    "Resultado" => "1-3",
    "Amarillas" => "2",
    "Rojas"     => "0",
    "Penalty"   => "1"
  ),
  "Canicas"  => array (
    "Resultado" => " ",
    "Amarillas" => " ",
    "Rojas"     => " ",
    "Penalty"   => " "
  ),
  "Condal"    => array (
    "Resultado" => "1-4",
    "Amarillas" => "2",
    "Rojas"     => "1",
    "Penalty"   => "1"
  ),
  "Piloñesa"  => array (
    "Resultado" => "2-0",
    "Amarillas" => "1",
    "Rojas"     => "0",
    "Penalty"   => "0"
  ),
),
"Condal"   => array(
  "Juvencia" => array (
    "Resultado" => "1-0 ",
    "Amarillas" => "4",
    "Rojas"     => "1",
    "Penalty"   => "2"
  ),
  "Mosconia"  => array (
    "Resultado" => "6-3",
    "Amarillas" => "1",
    "Rojas"     => "2",
    "Penalty"   => "3"
  ),
  "Canicas"  => array (
    "Resultado" => "14-3",
    "Amarillas" => "1",
    "Rojas"     => "0",
    "Penalty"   => "0"
  ),
  "Condal"    => array (
    "Resultado" => " ",
    "Amarillas" => " ",
    "Rojas"     => " ",
    "Penalty"   => " "
  ),
  "Piloñesa"  => array (
    "Resultado" => "1-0",
    "Amarillas" => "3",
    "Rojas"     => "1",
    "Penalty"   => "0"
  ),
),
"Piloñesa" => array(
  "Juvencia" => array (
    "Resultado" => "1-1",
    "Amarillas" => "0",
    "Rojas"     => "0",
    "Penalty"   => "1"
  ),
  "Mosconia"  => array (
    "Resultado" => "2-3",
    "Amarillas" => "1",
    "Rojas"     => "0",
    "Penalty"   => "1"
  ),
)

```

comillas.  
Una última advertencia. ¡No te desesperes con los errores de sintaxis! Son inevitables.

## Enmendando un olvido

Cuando hemos hablado de las funciones matemáticas hemos olvidado mencionar una de ellas.

Se trata de la función **valor absoluto**.

La sintaxis es la siguiente:

**Abs(\$a);**

Un ejemplo:

```
<?
$MiSaldo=-347.513 €;
$MisDeseos=Abs($MiSaldo);
$MisDeseos .= " €";
echo $MisDeseos;
?>
```

resultaría :

**347.513 €**

```
"Rojas"      => "0",
"Penalty"    => "0"
),
"Canicas"   => array (
"Resultado" => "0-1",
"Amarillas" => "0",
"Rojas"      => "0",
"Penalty"    => "0"
),
"Condal"     => array (
"Resultado" => "1-1",
"Amarillas" => "1",
"Rojas"      => "2",
"Penalty"    => "0"
),
"Piloñesa"   => array (
"Resultado" => " ",
"Amarillas" => " ",
"Rojas"      => " ",
"Penalty"    => " "
),
)
)
?>
```

Utilizando este array hemos construido la tabla que hemos puesto como ejemplo.

**ejemplo21.php**

Anterior

Índice

Siguiente

## Funciones de salida

Ya conocemos algunas de las funciones que PHP utiliza para mostrar información –salidas– en la ventana del navegador del cliente.

Recordémoslas:

### **echo**

La función **echo**, aunque admite también la forma **echo()**, no requiere de forma obligatoria el uso de los **paréntesis**.

Detrás de la *instrucción echo* pueden insertarse: variables, cadenas (éstas entre comillas) y números (éstos sin comillas) separadas por **comas**.

Este es un ejemplo de código:

```
$a=24; $b="Pepe";
$c=<br>;
echo $a,$b,25,
"Luis",$c;
```

que produciría esta salida:

24Pepe25Luis

Observa los **valores** que hay detrás de **echo**. Como verás, no es necesario insertar todos los valores en la misma línea.

### **print()**

La función **print()** sólo puede contener *dentro del paréntesis* una sola variable, o el conjunto de varias de ellas enlazadas por un punto.

Aquí tienes algunos ejemplos:

```
print(25.3)
produciría esta salida
25.3
```

```
print("Gonzalo")
escribiría
Gonzalo
```

```
$z=3.1416;
print($z);
escribiría
3.1416
```

Recuerda también que es posible utilizar *dentro del paréntesis* el **concatenador de cadenas**.

```
$h=3;
$f=" hermanos"
print("Heladeria ".$h.$f)
```

## Funciones de salida

Aquí tienes un resumen de las diversas opciones de la función **printf()**:

Sintaxis			
<code>printf(cadena de formato,variable1,variable2,..)</code>			
Cadena de formato			
<code>"%[relleno][alineación][ancho][precisión][tipo]"</code>			
	Carácter	Valor	Sintaxis
Relleno	0	0	<code>printf("%02d",32)</code>
	*	*	<code>printf("%*20d",32)</code>
	espacio 1)	'	<code>printf("%'20d",32)</code>
	-	'-	<code>printf("%'-20d",32)</code>
Observaciones			
En este apartado prestaremos atención únicamente a los caracteres marcados en rojo, que son los que corresponden a las diferentes formas de relleno. Los demás parámetros los iremos tratando uno en los apartados siguientes.			
Cuando se pretende <i>llenar con ceros</i> –a la izquierda– basta escribir el <b>0</b> inmediatamente detrás del signo %			
Si se trata de llenar con un carácter distinto de cero debe escribirse <b>inmediatamente después de % una comilla simple</b> ' seguida del carácter de relleno.			
Si se pretende llenar con <b>espacios forzados</b> se puede escribir la comilla simple ' e <b>inmediatamente después</b> teclear la combinación <b>ALT+0160</b> (carácter ASCII 160) usando el teclado numérico. Aunque obviamente no se visualiza el espacio si se conserva tal como puede verse en el ejemplo 1)			
Obsérvese que como la tipografía es de <b>ancho variable</b> y que según el carácter que se use como relleno se modifica el ancho de la presentación.			
Quizá convenga recordar que <b>32</b> es en este caso la variable a la que pretendemos dar formato y que ese valor podría ser sustituido por el nombre de una variable que contenga valores numéricos.			
	Carácter	Valor	Sintaxis
Alineación	Ninguno	Dcha	<code>printf("%02d",32)</code>
	-	Izda	<code>printf("%-20d",32)</code>
	Ninguno	Dcha	<code>printf("%*20d",32)</code>
	-	Izda	<code>printf("%*-20d",32)</code>
	Ninguno	Dcha	<code>printf("%020s",32)</code>
	-	Izda	<code>printf("%-20s",32)</code>
	Ninguno	Dcha	<code>printf("%*20s",32)</code>
	-	Izda	<code>printf("%*-20s",32)</code>
Observaciones			
En los casos en que figura <b>Ninguno</b> en la columna <b>Carácter</b> tratamos de indicar que no es necesario escribir nada en la cadena de formato. Cuando aparece un signo (-) estamos indicando que debe insertarse un <b>signo menos</b> .			
Fíjate que en los cuatro primeros supuestos el identificador de tipo es <b>d</b> , lo cual hace que considere la variable como <b>número</b> , mientras que en los cuatro últimos ese valor es <b>s</b> , con lo cual considera la variable como tipo <b>cadena</b> .			
Cuando tratamos de llenar una <b>variable numérica</b> con <b>ceros</b> por la derecha PHP los omite para no alterar el valor numérico en la presentación			
Con cualquier otro <b>carácter de relleno</b> (incluidos los caracteres numéricos con ' delante) sí efectúa el relleno.			
	Carácter	Valor	Sintaxis
Ancho	Entero	14	<code>printf("%*14d",32)</code>
	Entero	17	<code>printf("%*-17d",32)</code>
	Decimal	14.5	<code>printf("%*14.5d",32)</code>
	Decimal	17.8	<code>printf("%*17.8d",32)</code>
	Decimal	14.5	<code>printf("%*14.5f",32)</code>
	Decimal	11.8	<code>printf("%*11.8f",32)</code>
Observaciones			
El ancho (nº de caracteres totales) puede especificarse mediante <b>un número entero</b> para todo tipo de variables			
Si se expresa mediante <b>un número decimal</b> y la variable es tipo <b>coma flotante</b> la parte decimal indica la <b>precisión</b> (nº de cifras decimales) y la parte entera el <b>ancho</b> como número de caracteres de la			

## Salidas con formato

Ni la función **echo**, ni tampoco **print** permiten establecer una **presentación** (formato) en sus salidas, excepto que alguna de las variables que se use contenga el resultado de una función **number\_format**.

La función **printf()** ofrece un gran número de posibilidades en este sentido. Tanto la sintaxis como los valores de los diferentes parámetros –cuando se trate de presentar números– las tienes resumidas aquí a la derecha.

En la página siguiente veremos el uso de **printf()** para el tratamiento de variables tipo **cadena**.

parte entera o de la parte decimal, según se rellene a la derecha o a la izquierda.			
Tipo	Valor	Sintaxis	Resultado
Tipo	Presentación en forma binaria	<code>printf("%*14b",17)</code>	*****10001
	Caracter correspondiente al código ASCII	<code>printf("%*14c",97)</code>	a
	Número presentado como entero	<code>printf("%*14d",17.83)</code>	*****17
	Número presentado con decimales	<code>printf("%*14f",17.45)</code>	*****17.450000
	Presentación en forma octal	<code>printf("%*14o",17)</code>	*****21
	Presentación en hexadecimal	<code>printf("%*14x",170)</code>	*****aa
	Presentación en hexadecimal	<code>printf("%*14X",170)</code>	*****AA
	Presentación como >cadena	<code>printf("%*14s",170)</code>	*****170

Anterior



Índice



Siguiente



## Un ejemplo

### Dos interpretaciones

Recordemos que cuando un documento tiene extensión **php** es objeto de **dos interpretaciones** antes de ser visualizado en el navegador. En primer lugar es PHP quien ejecuta sus scripts y devuelve al cliente el documento resultante y, más tarde, será el propio navegador quien realice una **segunda interpretación** –del documento recibido– cuyo resultado será la visualización de la página.

### Los saltos de línea

Cualquier *salto de línea* que se inserte en un documento será respetado por PHP y enviado al navegador, pero, como éste solo entiende como tales sus propias etiquetas **<br>**, no serán visualizados, aunque sí estarán en el código fuente de la página visualizada.

Hemos de considerar un nuevo concepto. PHP permite utilizar algunos caracteres especiales que son transformados durante la ejecución del script.

Uno de ellos –no es el único– es **\n** que es interpretado y convertido por el intérprete de PHP en un *salto de línea* cuyo efecto será visible en el código fuente del documento enviado al navegador, pero que –por no ser una etiqueta **<br>**– no producirá efecto alguno en la visualización de la página.

### **nl2br(\$A)**

Esta función inserta de **forma automática** etiquetas HTML de salto de línea (**<br>**). Por cada salto de línea que exista en el texto de la cadena inserta una etiqueta **<br>** en la salida HTML.

### **strtr(\$a,\$dicc)**

Busca en la cadena **\$a** las *palabras* coincidentes con los **índices** del **array asociativo** (**\$dicc**) y las sustituye por los **valores** asociados a esos índices.

### **get\_meta\_tags(\$a)**

Devuelve un **array asociativo** cuyos **índices** son los valores de la propiedad **name** de las etiquetas **<meta>** (escritas siempre en minúsculas, sin comillas, y reemplazando, en el caso de que

```

<?
/* definamos algunas variables de cadena
tal como se describe en sus contenidos
e incluyamos caracteres especiales \n */
$cadena1="Este texto está escrito
en varias líneas
y hemos saltado de una a otra
pulsando enter";
$cadena2="Aquí\nseparamos\nlas\nlíneas\ncon\nsin\npulsar\nenter";
$cadena3=<<<Prueba
Nuevamente texto en varias líneas
ahora usando sintaxis de documento incrustado.
Seguiremos probando
Prueba;
$cadena4=<<<OtraPrueba
Ahora\ninsertaré\nalgo\ncomo\nesto
OtraPrueba;
# definamos una variable conteniendo saltos de linea HTML
# y vayamos construyendo una variable de salida
# en la que uniremos las variables anteriores insertando
# entre ellas saltos de linea para facilitar la visualización
# en el navegador
$saltador="  
><br><br>";
$salida=$cadena1.$saltador;
$salida .= $cadena2.$saltador;
$salida .= $cadena3.$saltador;
$salida .= $cadena4.$saltador;
# visualicemos el resultado
print $salida;
# apliquemos ahora a la variable salida
# la función nl2br y veamos el resultado
print nl2br($salida);
?>

```

ejemplo22.php

Si ejecutas el ejemplo y visualizas el código fuente a través del navegador podrás observar como los resultados del primer **print** generan saltos de línea en éste. Y en el caso del código correspondiente al segundo print, podrás visualizar etiquetas **<BR />** que son el resultado de la aplicación de la función **nl2br**.

### Las cadenas y las etiquetas HTML

Cadenas	Sintaxis	Resultado
\$a="Esto es\nun texto escrito\nen varias líneas\nsin etiquetas ";	nl2br(\$a)	Esto es un texto escrito en varias líneas sin etiquetas 

<a href="#">Ver índice</a>	<a href="#">Búsqueda rápida</a>	<a href="#">Página anterior</a>	<a href="#">Página siguiente</a>
"sun" =>"sol"; \$a="Lundi es un dia good si hace sun"		...	
\$a=index.php"	\$b=get_meta_tags(\$a); echo \$b[keywords]	Programación, PHP, Apache, MySQL	
\$a="index.php"	\$b=get_meta_tags(\$a); echo \$b[description]	Materiales para un curso a distancia	

especiales por un guión bajo \_) de la página web cuya dirección (absoluta o relativa) se indica en la cadena **\$a**. Los valores de los elementos del array son los contenidos de esas etiquetas.

#### htmlspecialchars(\$a)

Convierte en *entidades de HTML* algunos caracteres (los que se indican a la derecha). Con ello se consigue su visualización y se impide que sean interpretados como *signos del lenguaje HTML*.

#### htmlentities(\$a)

Es una función similar a la anterior, pero en este caso afecta a *todos los caracteres* que tienen equivalentes como *entidad HTML* utilizando el juego de caracteres ISO-8859-1. Recuerda que son éstos.

\$a=<H1>A</H1>"	echo \$a	A
\$a=<H1>A</H1>"	echo htmlspecialchars(\$a)	<H1>A</H1>

La tabla de sustituciones de **htmlspecialchars** es esta:

Sustituye & por &#amp;  
" por &quot;  
< por &lt;  
> por &gt;

#### Ejercicio nº 16

Modifica el script del ejercicio nº 15 de forma que los saltos de línea introducidos en el textarea del formulario sean sustituidos por etiquetas <BR> de HTML.

Anterior



Índice



Siguiente



## Funciones de cadenas

Algunas de las funciones que permiten manejar los formatos de las *cadenas de caracteres* son estas:

### **chr(*n*)**

Devuelve el carácter cuyo código ASCII es *n*.

### **ord(*cadena*)**

Devuelve el *código ASCII* del **primero** de los caracteres de la *cadena*.

### **strlen(*cadena*)**

Devuelve la *longitud* (número de caracteres) de la *cadena*. Los espacios son considerados como un carácter más.

### **strtolower(*cadena*)**

Cambia todos los caracteres de la *cadena* a *minúsculas*.

### **strtoupper(*cadena*)**

Convierte en *mayúsculas* todos los caracteres de la *cadena*.

### **ucwords(*cadena*)**

Convierte a *mayúsculas* la *primera letra* de cada palabra.

### **ucfirst(*cadena*)**

Convierte a *mayúsculas* la primera letra de la *cadena* y pone en *minúsculas* todas las demás.

### **ltrim(*cadena*)**

Elimina todos los *espacios* que pudiera haber al *principio* de la *cadena*.

### **rtrim(*cadena*)**

Elimina todos los *espacios* que existieran al *final de la cadena*.

### **trim(*cadena*)**

Elimina los *espacios* tanto al *principio* como al *final* de la *cadena*.

### **chop(*cadena*)**

Elimina los *espacios* al *final* de la *cadena*. Es *idéntica* a **rtrim**.

Advertencia

## Formatos en cadenas

### **printf(*cadena de formato,variable1,variable2,..*)**

#### *Cadena de formato*

Dentro de la *cadena de formato* deben repetirse tantos formatos como variables se pretenda manejar

`"%[rell1][alin1][anc1][prec1][tipo1][sepa1]%"`

Hemos de mencionar aquí los **separadores** ya que no fueron mencionados en la página anterior

Se puede introducir una *cadena de separación* al final de una *cadena de formato* que puede hacer, entre otras, función de separación entre dos *cadenas*.

Por ejemplo, `printf("%*15.2f Euros",1475.875)` nos devolvería:

`*****1475.88 Euros`

La función **printf()** permite presentar varios valores o variables con distintos formatos utilizando la sintaxis que se indica más arriba.

Este ejemplo :

`printf("%*15.2f Euros=%*18.0f Pesetas",1475.875,1475.875*166.386)`

devuelve como resultado:

`*****1475.88 Euros=*****245565 Pesetas`

Existe otra función PHP con características muy similares a la anterior. Se trata de **sprintf()**.

La sintaxis es idéntica **sprintf (*cadena de formato, variable1,variable2, ...*)** y su única diferencia con **printf** es que, mientras que **printf()** imprime las variables utilizando el formato indicado, **sprintf()** puede guardar en una nueva variable la *cadena resultante de la aplicación del formato*.

## Otras funciones con cadenas

Estos son algunos ejemplos de aplicación de las funciones de manejo de cadenas

### **Código ASCII y viceversa**

Función	Ejemplo	Resultado
<code>chr(código ASCII)</code>	<code>chr(97)</code>	<code>a</code>
<code>ord("cadena")</code>	<code>ord("abadesa")</code>	<code>97</code>

### **Longitudes y conversiones mayúsculas/minúsculas**

Función	Ejemplo	Resultado
<code>strlen("cadena")</code>	<code>strlen("Mide la longitud de esta cadena")</code>	<code>31</code>
<code>strtolower("cadena")</code>	<code>strtolower("CONVIERTA A MINÚSCULAS")</code>	<code>convierte a minúsculas</code>
<code>strtoupper("cadena")</code>	<code>strtoupper("pasa a mayúsculas")</code>	<code>PASA A MAYÚSCULAS</code>
<code>ucwords("cadena")</code>	<code>ucwords("todas empiezan por mayúscula")</code>	<code>Todas Empiezan Por Mayúscula</code>
<code>ucfirst("cadena")</code>	<code>ucfirst("mayúscula al principio")</code>	<code>Mayúscula al principio</code>

### **Eliminar espacios**

Tanto **trim**, como **ltrim** y **rtrim** eliminan, además de los espacios, las secuencias: \n, \r, \t, \v y \0; llamadas también **caracteres protegidos**.

### substr(cadena,n)

Si el valor de **n** es **positivo** extrae **todos los caracteres** de la cadena a partir del que ocupa la posición **enésima** a contar desde la izquierda.

Si el valor de **n** es **negativo** serán extraídos **los n últimos caracteres** contenidos en la cadena.

### substr(cadena,n,m)

Si **n** y **m** son **positivos** extrae **m** caracteres a partir del que ocupa la posición **enésima**, de izquierda a derecha.

Si **n** es **negativo** y **m** es **positivo** extrae **m** (contados de izquierda a derecha) a partir del que ocupa la posición **enésima** contada de derecha a izquierda.

Si **n** es **positivo** y **m** es **negativo** extrae la cadena comprendida entre el **enésimo** carácter (contados de izquierda a derecha) hasta el **emésimo**, contando en este caso de derecha a izquierda

Si **n** es **negativo** y **m** también es **negativo** extrae la porción de cadena comprendida entre el **emésimo** y el **enésimo** caracteres contando, en ambos casos, de derecha a izquierda. Si el valor absoluto de **n** es **menor** que el de **m** devuelve una **cadena vacía**.

### strrev(cadena)

Devuelve la cadena **invertida**

### str\_repeat(cadena, n)

Devuelve la cadena **repetida** tantas veces como indica **n**.

### str\_pad(cad, n, rell, tipo)

Añade a la cadena **cad** los caracteres especificados en **rell** (uno o varios, escritos entre comillas) hasta que alcance la longitud que indica **n** (un número)

El parámetro **tipo** puede tomar uno de estos tres valores (sin comillas): **STR\_PAD\_BOTH** (rellena por ambos lados) **STR\_PAD\_RIGHT** (rellena por la derecha) **STR\_PAD\_LEFT** (rellena por la izquierda).

Si se omite la cadena de **Relleno** utilizará **espacios** y si se omite el **tipo** rellenará por la **derecha**

Función	Ejemplo	Resultado
ltrim("cadena")	ltrim("\n\nEliminar espacios")	Eliminar espacios
rtrim("cadena")	rtrim("Eliminar espacios\n\n")	Eliminar espacios
trim("cadena")	trim("\n\nEliminar espacios\n\n")	Eliminar espacios
chop("cadena")	chop("\n\nEliminar espacios\n\n")	Eliminar espacios
<b>Extraer porciones de una cadena</b>		
Función	Ejemplo	Resultado
substr("cadena",n)	substr("Extrae caracteres",3)	rae caracteres
substr("cadena",n)	substr("Extrae caracteres",0)	Extrae caracteres
substr("cadena",n)	substr("Extrae caracteres",-5)	teres
substr("cadena",n)	substr("Extrae caracteres",-2)	es
substr("cadena",n,m)	substr("Extrae caracteres",2,6)	trae c
substr("cadena",n,m)	substr("Extrae caracteres",0,8)	Extrae c
substr("cadena",n,m)	substr("Extrae caracteres",2,-3)	trae caracte
substr("cadena",n,m)	substr("Extrae caracteres",-7,5)	acter
substr("cadena",n,m)	substr("Extrae caracteres",-7,-5)	ac
substr("cadena",n,m)	substr("Extrae caracteres",-5,-7)	
<b>Modificaciones de cadenas</b>		
Función	Ejemplo	Resultado
strrev("cadena")	strrev("Invierte la cadena")	anedac al etreivnl
str_repeat("cadena",n)	str_repeat("Rep",5)	RepRepRepRepRep
str_pad("cadena",n,"Relleno",Tipo)	str_pad("Pepe",10,"*",STR_PAD_BOTH)	***Pepe***
str_pad("cadena",n,"Relleno",Tipo)	str_pad("Pepe",10,"*",STR_PAD_LEFT)	*****Pepe
str_pad("cadena",n,"Relleno",Tipo)	str_pad("Pepe",10,"*",STR_PAD_RIGHT)	Pepe*****
str_pad("cadena",n,"Relleno",Tipo)	str_pad("Pepe",10,"")	Pepe*****
str_replace ("lo que dice",lo que dira,"Cadena")	str_replace("e","a","Pepe")	Papa
str_replace ("lo que dice",lo que dira,"Cadena")	str_replace("pe","pa","Pepepe")	Pepapa
str_replace ("lo que dice",lo que dira,"Cadena")	str_replace("Pepe","Luis","Pepe")	Luis
substr_replace ("Cadena",lo que dira,n,m)	substr_replace("Pepe","Luis",2,-1)	PeLuise

### ¡Cuidado!

Como tu **buen criterio** ya habrá podido advertir, no se trata aquí de **aprender** todas estas opciones de formato –ni las que veremos en las páginas siguientes– sino de que dispongas de una referencia de consulta a la que recurrir en el momento en el que necesites utilizar estas funciones.

Pensamos que **sí** es importante que sepas que PHP dispone de todos estos recursos y conozcas su potencialidad para hacer presentaciones de las formas más variadas.

### Ejercicio nº 15

Crea un formulario **-formulario15.php-** mediante el que puedas transferir un valor numérico y un texto (un input tipo **text** y otro **textarea**).

Trata de que el script que los reciba **-visor15.php-**, permita visualizar sus valores de forma que el valor numérico tenga 2 decimales y acabe con la palabra euros, y que la parte entera se complete (por la izquierda) con asteriscos hasta tener una longitud de 12 caracteres.

El contenido del **textarea** debería visualizarse íntegramente en minúsculas, excepto los diez primeros caracteres que deberían verse en mayúsculas.

Anterior

Índice

Siguiente





Ver índice

# Operadores bit a bit



## Comentario previo

Incluimos la sintaxis de este tipo de operadores a título meramente informativo. Rara vez será necesario utilizarlos en nuestras aplicaciones PHP.

Su utilidad suele limitarse a la gestión de periféricos y algunas operaciones de cálculo de carácter muy reiterativo en la que se puede conseguir un rendimiento muy superior a los operadores tradicionales.

En el ámbito propio del PHP pueden tener algún interés a la hora de elaborar rutinas para *criptar* el código fuente de algunos scripts que por su importancia pueden requerir ese tipo de protección.

Los que sí han de resultarnos de gran interés serán el resto de los operadores. Los iremos viendo en páginas sucesivas.

## Operadores bit a bit

### \$A & \$B

El operador **&** compara los valores binarios de cada uno de los bits de las cadenas \$A y \$B y devuelve 1 en el caso que ambos sean 1, y 0 en cualquier otro caso.

Cuando las variables \$A y \$B son **cadenas** compara los valores binarios de los códigos ASCII de sus caracteres y devuelve los caracteres ASCII correspondientes al resultado de esa comparación.

### \$A | \$B

Funciona de forma idéntica al anterior y *devuelve 1* cuando **al menos** el valor de uno de los bits comparados es 1, y devolverá 0 cuando **ambos** sean 0.

### \$A ^ \$B

Devuelve 1 cuando los bits comparados son **distintos**, y 0 cuando son **iguales**.

### \$A << \$B

Realiza la operación  $$A * 2^B$ .

Hace el cálculo añadiendo **\$B CEROS** (binarios) a la derecha de la cadena binaria \$A.

### \$A >> \$B

## Manejando operadores bit a bit

Desarrollamos aquí algunos ejemplos de manejo de los operadores bit a bit.

El operador &					
Números		Números como cadenas		Cadenas alfanuméricas	
Variables	Valores binarios	Variables	Valores binarios	Variables	Valores binarios
\$a=12	1100	\$A="12"	110001110010	\$A1="Rs"	10100101110011
\$b=23	10111	\$B="23"	110010110011	\$B1="aZ"	11000011011010
\$a&\$b=4	100	\$A&\$B=02	110000110010	\$A1&\$B1=@R	10000001010010

En los casos de cadenas hemos diferenciado en rojo el valor binario correspondiente al primer carácter. Esos valores binarios corresponden a la forma binaria del código ASCII de cada uno de los caracteres

Puedes observar que el tratamiento es distinto cuando los mismos valores numéricos se asignan como entero y como cadena.

Al asignarlos como cadena opera los valores binarios de los códigos ASCII de los caracteres, mientras que cuando se trata de números compara los valores de las expresiones binarias de los valores de cada uno de ellos

El operador					
Números		Números como cadenas		Cadenas alfanuméricas	
Variables	Valores binarios	Variables	Valores binarios	Variables	Valores binarios
\$a=12	1100	\$A="12"	110001110010	\$A1="Rs"	10100101110011
\$b=23	10111	\$B="23"	110010110011	\$B1="aZ"	11000011011010
\$a \$b=31	11111	\$A \$B=33	110011110011	\$A1 \$B1=s{	1110011111011

Se comporta de forma idéntica al anterior en lo relativo a números y cadenas.

El operador ^					
Números		Números como cadenas		Cadenas alfanuméricas	
Variables	Valores binarios	Variables	Valores binarios	Variables	Valores binarios
\$a=12	1100	\$A="12"	110001110010	\$A1="Rs"	10100101110011
\$b=23	10111	\$B="23"	110010110011	\$B1="aZ"	11000011011010
\$a^\$b=27	11011	\$A^\$B=000001	000011000001	\$A1^\$B1=3)	01100110101001

Los criterios de tratamiento de números y cadenas coinciden con los operadores anteriores.

El operador <<					
Números		Números como cadenas		Cadenas alfanuméricas	
Variables	Valores binarios	Variables	Valores binarios	Variables	Valores binarios
\$a=12	1100	\$A="12"	110001110010	\$A1="Rs"	10100101110011
\$b=2	10	\$B=2	10	\$B1=2	10
\$a<<b=48	110000	\$A<<\$B=48	110000	\$A1<<\$B1=0	

El operador << multiplica el valor de la primera cadena por 2 elevado al valor de la segunda.

Al ser un operador matemático solo tiene sentido cuando ambas variables son números naturales. En las cadenas alfanuméricas extrae los números que pudiera haber al comienzo y, en caso de no haberlos, toma valor cero.

El operador >>					
Números		Números como cadenas		Cadenas alfanuméricas	
Variables	Valores binarios	Variables	Valores binarios	Variables	Valores binarios

Divide el valor **\$A** entre  **$2^{\$B}$** .  
Hace la operación en la cadena  
*binaria* quitando ***\$B CEROS*** (por la  
derecha) de la cadena ***\$A***.

~ \$A

Invierte los valores de los bits de la cadena **\$A** convirtiendo los CEROS en UNO y los UNO en CERO.

Variables	Valores binarios	Variables	Valores binarios	Variables	Valores binarios
\$a=12	1100	\$A="12"	110001110010	\$A1="Rs"	10100101110011
\$b=2	10	\$B=2		\$B1=2	10
\$a>>b=3	11	\$A>>\$B=3		\$A1>>\$B1=0	

Para este operador (`>>`) son aplicables los mismos comentarios hechos en el párrafo anterior.

## Anterior



Índice



Siguiente





Ver índice

# Operadores de comparación



## Antes de empezar

Si es este tu primer contacto con el mundo de la programación es probable que estés pensando que *todo esto está muy bien* pero que a tí lo que te interesa es *hacer cosas*. En una palabra, que quieras usar PHP como herramienta para el desarrollo de tus proyectos.

Esta página y las siguientes van a ser básicas para el éxito en ese lógico y razonable afán utilitaria.

Pensemos en nuestro día a día. En cada momento de nuestra vida *hacemos algo*. Cada cosa que hacemos suele requerir casi siempre esta secuencia de acciones: *comparar, evaluar, optar y hacer*.

Supongamos que queremos *cambiar de coche*. Lo más prudente será empezar por *comparar las características* (potencia, diseño, precio, etc.) de los diferentes modelos.

Salvo en casos excepcionales, no tomaremos una decisión de compra a través de un solo parámetro sino que haríamos una *evaluación conjunta* de todos esos factores (*menos potencia pero mejor precio y diseño muy similar*, por citar un ejemplo) y sería a través de esa evaluación como *optaría-mos* por una marca o modelo.

Una vez ejercida la opción –y no antes– sería el momento de *realizar la compra del nuevo coche*.

PHP, y en general todos los lenguajes de programación, disponen de herramientas que permiten emular cada uno de esos procesos de la conducta humana. Los iremos viendo en esta página y en las siguientes.

## Operadores de comparación

PHP dispone de los siguientes operadores de comparación:

### \$A == \$B

El operador **== compara** los valores de dos variables y devuelve **1** (CIERTO) en el caso de que sean iguales y el valor **NUL** –carácter ASCII 0– (FALSO) cuando son distintas.

Mediante este operador se pueden comparar variables de distinto tipo.

## Manejando operadores de comparación

Desarrollamos aquí algunos ejemplos de manejo de los operadores de comparación.

**El operador ==**

A	B	Operador	Sintaxis	Resultado
\$A=123	\$B=123	==	\$A==\$B	1
\$A=123.0	\$B=123	==	\$A==\$B	1
\$A=123	\$B="123"	==	\$A==\$B	1
\$A=123	\$B="123ABC"	==	\$A==\$B	1
\$A=123	\$B=124	==	\$A==\$B	
\$A=123	\$B=124	==	ord(\$A==\$B)	0
\$A="abc"	\$B="ABC"	==	\$A==\$B	
\$A="abc"	\$B="abc"	==	\$A==\$B	1
\$A="abc"	\$B="abcd"	==	\$A==\$B	

Los valores de la columna señalada como *Resultados* se obtienen mediante la función `echo($A==$B)`; Podemos ver que, en los casos que es cierta la igualdad, imprime un **1**, mientras que cuando es falsa *no imprime nada*. Se justifica este extremo porque en caso de no coincidencia el valor devuelto es **NUL** (código ASCII 0) y ese carácter carece de símbolo gráfico.

Hemos marcado en **rojo** una excepción.

En ese caso, la instrucción es `echo ord($A==$B)`; y –como recordarás– dado que `ord`, que devuelve el código ASCII del carácter contenido en el paréntesis que le acompaña, podemos comprobar a través del **cero** de este resultado, que, efectivamente, la no coincidencia devuelve **NUL**.

**El operador ===**

A	B	Operador	Sintaxis	Resultado
\$A=123	\$B=123	==	\$A==\$B	1
\$A=123.0	\$B=123	==	\$A==\$B	
\$A=123	\$B="123"	==	\$A==\$B	
\$A=123	\$B="123ABC"	==	\$A==\$B	
\$A=123	\$B=124	==	\$A==\$B	
\$A=123	\$B=124	==	ord(\$A==\$B)	0
\$A="abc"	\$B="ABC"	==	\$A==\$B	
\$A="abc"	\$B="abc"	==	\$A==\$B	1
\$A="abc"	\$B="abcd"	==	\$A==\$B	

Observa que los valores señalados en **rojo** –a diferencia de lo que ocurre con el operador anterior– devuelven **NUL** como resultado.

En este caso no sólo compara valores sino que también compara tipos de variables. Al ser una de ellas tipo numérico y la otra cadena alfanumérica el resultado no puede ser otro que **NUL**.

**El operador !=**

A	B	Operador	Sintaxis	Resultado
\$A=123	\$B=123	!=	\$A!=\$B	
\$A=123.0	\$B=123	!=	\$A!=\$B	
\$A=123	\$B="123"	!=	\$A!=\$B	
\$A=123	\$B="123ABC"	!=	\$A!=\$B	
\$A=123	\$B=124	!=	\$A!=\$B	1
\$A=123	\$B=124	!=	ord(\$A!=\$B)	49
\$A="abc"	\$B="ABC"	!=	\$A!=\$B	1
\$A="abc"	\$B="abc"	!=	\$A!=\$B	
\$A="abc"	\$B="abcd"	!=	\$A!=\$B	1

Para comparar una *cadena* con un *número* se extrae el *valor entero de la cadena* (si lleva dígitos al comienzo los extrae y en caso contrario le asigna el valor cero) y utiliza ese valor para hacer la comparación.

Cuando se comparan cadenas *discrimina* entre *mayúsculas* y *minúsculas* ya que utiliza los códigos ASCII de cada uno de los caracteres para hacer la comparación.

La comparación se hace de izquierda a derecha y devuelve **1** (CIERTO) sólo en el caso que coincidan exactamente los contenidos de ambas cadenas.

#### \$A === \$B

El operador **==** es similar al anterior, pero realiza la comparación en sentido **estricto**.

Para que devuelva **1** es necesario que sean iguales *los valores de las variables y también su tipo*.

#### \$A != \$B

El operador **!=** devuelve **1** cuando los valores de las variables **son distintos** (en general **!** indica negación, en este caso podríamos leer «*no igual*») y devuelve **NUL** cuando son iguales.

Este operador **no compara** en sentido **estricto**, por lo que puede considerar iguales los valores de dos variables de distinto tipo.

#### \$A < \$B

El operador **<** devuelve **1** cuando los valores de **\$A** son **menores** que los de **\$B**.

Los criterios de comparación son los siguientes:

- Los *valores numéricos* siguen el criterio matemático.
- Cuando se trata de un *número* y una *cadena* extrae el valor numérico de ésta (es cero si no hay ningún dígito al principio de la misma) y hace una comparación matemática.

– En el supuesto de dos cadenas, compara **uno a uno** –de izquierda a derecha– los códigos ASCII de cada uno de los caracteres (primero con primero, segundo con segundo, etcétera).

Si al hacer esta comprobación encuentra –en la primera cadena– un carácter cuyo código ASCII es mayor que el correspondiente de la segunda cadena, o encuentra que todos son iguales en ambas cadenas devuelve **NUL**. Solo en el caso de no existir ninguno mayor y sí haber al menos uno menor devolverá **UNO**.

En los ejemplos señalados en **rojo** Puedes comprobar el carácter **no estricto** de este operador. Devuelve **NUL** porque considera que **no son distintos**, lo que equivale a interpretar que los considera **iguales** pese a que las variables sean de distinto tipo.

**El operador <**

A	B	Operador	Sintaxis	Resultado
\$A=123	\$B=123	<	\$A<\$B	
\$A=123	\$B="123"	<	\$A<\$B	
\$A=123.0	\$B="123"	<	\$A<\$B	
\$A=123	\$B="123ABC"	<	\$A<\$B	
\$A=123	\$B=124	<	\$A<\$B	1
\$A=123	\$B=124	<	ord(\$A<\$B)	49
\$A="abc"	\$B="ABC"	<	\$A<\$B	
\$A="abc"	\$B="abc"	<	\$A<\$B	
\$A="bcd"	\$B="abcd"	<	\$A<\$B	1
\$A="aacd"	\$B="abcd"	<	\$A<\$B	1
\$A="abc"	\$B="abcd"	<	\$A<\$B	1
\$A="abcd"	\$B="abc"	<	\$A<\$B	
\$A="A"	\$B="a"	<	\$A<\$B	1
\$A="a"	\$B="A"	<	\$A<\$B	
\$A="aBC"	\$B="A"	<	\$A<\$B	
\$A="123"	\$B=124	<	\$A<\$B	1
\$A=123	\$B="124"	<	\$A<\$B	1

Observa los ejemplos señalados en **rojo**. Cuando las cadenas tienen distinta longitud este operador considera que los caracteres que faltan en la cadena más corta son **NUL**. Esa es la razón por la que en el primer caso devuelve CIERTO (NUL es menor que d) y el segundo FALSO (d no es menor que NUL).

**El operador <=**

A	B	Operador	Sintaxis	Resultado
\$A=123	\$B=123	<=	\$A<=\$B	1
\$A=123.0	\$B=123	<=	\$A<=\$B	1
\$A=123	\$B="123"	<=	\$A<=\$B	1
\$A=123	\$B="123ABC"	<=	\$A<=\$B	1
\$A=123	\$B=124	<=	\$A<=\$B	1
\$A=123	\$B=124	<=	ord(\$A<=\$B)	49
\$A="abc"	\$B="ABC"	<=	\$A<=\$B	
\$A="abc"	\$B="abc"	<=	\$A<=\$B	1
\$A="abc"	\$B="abcd"	<=	\$A<=\$B	1
\$A="A"	\$B="a"	<=	\$A<=\$B	1
\$A="a"	\$B="A"	<=	\$A<=\$B	
\$A="aBC"	\$B="A"	<=	\$A<=\$B	
\$A="123"	\$B=124	<=	\$A<=\$B	1
\$A=123	\$B="124"	<=	\$A<=\$B	1

Hemos modificado la instrucciones marcadas en **rojo** para comprobar el código ASCII de carácter que devuelve, en el caso de ser cierto, el resultado de la comparación. El valor que aparece (49) como resultado es el código ASCII del carácter 1.

**El operador >**

A	B	Operador	Sintaxis	Resultado
\$A=123	\$B=123	>	\$A>\$B	
\$A=123	\$B="123"	>	\$A>\$B	
\$A=123	\$B="123ABC"	>	\$A>\$B	
\$A=123	\$B=124	>	\$A>\$B	
\$A=123	\$B=124	>	ord(\$A>\$B)	0
\$A="abc"	\$B="ABC"	>	\$A>\$B	1
\$A="abc"	\$B="abc"	>	\$A>\$B	
\$A="abc"	\$B="abcd"	>	\$A>\$B	
\$A="A"	\$B="a"	>	\$A>\$B	
\$A="a"	\$B="A"	>	\$A>\$B	1
\$A="aBC"	\$B="A"	>	\$A>\$B	1

– Cuando las cadenas tengan distinta longitud, considerá (a efectos de la comparación) que los caracteres *que faltan* en la cadena más corta son NUL (ASCII 0).

### \$A <= \$B

Se comporta de forma idéntica al anterior. La única diferencia es que ahora aceptará como *ciertos* los casos de *igualdad* tanto en el caso de números como en el de códigos ASCII.

### \$A > \$B

Es idéntico –en el modo de funcionamiento– a `$A < $B`. Solo difiere de éste en el *criterio de comparación* que ahora requerirá que los valores de `$A` sean *mayores* que los de la variable `$B`.

### \$A >= \$B

Añade al anterior la posibilidad de certeza en caso de igualdad.

<code>\$A="123"</code>	<code>\$B=124</code>	<code>&gt;</code>	<code>\$A&gt;\$B</code>	
<code>\$A=123</code>	<code>\$B="124"</code>	<code>&gt;</code>	<code>\$A&gt;\$B</code>	

El operador <code>&gt;=</code>				
A	B	Operador	Sintaxis	Resultado
<code>\$A=123</code>	<code>\$B=123</code>	<code>&gt;=</code>	<code>\$A&gt;=\$B</code>	1
<code>\$A=123</code>	<code>\$B="123"</code>	<code>&gt;=</code>	<code>\$A&gt;=\$B</code>	1
<code>\$A=123</code>	<code>\$B="123ABC"</code>	<code>&gt;=</code>	<code>\$A&gt;=\$B</code>	1
<code>\$A=123</code>	<code>\$B=124</code>	<code>&gt;=</code>	<code>\$A&gt;=\$B</code>	
<code>\$A=123</code>	<code>\$B=124</code>	<code>&gt;=</code>	<code>ord(\$A&gt;=\$B)</code>	0
<code>\$A="abc"</code>	<code>\$B="ABC"</code>	<code>&gt;=</code>	<code>\$A&gt;=\$B</code>	1
<code>\$A="abc"</code>	<code>\$B="abc"</code>	<code>&gt;=</code>	<code>\$A&gt;=\$B</code>	1
<code>\$A="abc"</code>	<code>\$B="abcd"</code>	<code>&gt;=</code>	<code>\$A&gt;=\$B</code>	
<code>\$A="A"</code>	<code>\$B="a"</code>	<code>&gt;=</code>	<code>\$A&gt;=\$B</code>	
<code>\$A="a"</code>	<code>\$B="A"</code>	<code>&gt;=</code>	<code>\$A&gt;=\$B</code>	1
<code>\$A="aBC"</code>	<code>\$B="A"</code>	<code>&gt;=</code>	<code>\$A&gt;=\$B</code>	1
<code>\$A="123"</code>	<code>\$B=124</code>	<code>&gt;=</code>	<code>\$A&gt;=\$B</code>	
<code>\$A=123</code>	<code>\$B="124"</code>	<code>&gt;=</code>	<code>\$A&gt;=\$B</code>	

### Ejercicio nº 17

Escribe un script en el que uses cada uno de los operadores de comparación que hemos descrito en esta página.

Eligiendo las variables adecuadas, debes escribir dos ejemplos –para cada uno de los operadores– de modo que uno de ellos devuelva CIERTO y el otro FALSO.

Guarda el documento como **ejercicio17.php** y comprueba que su funcionamiento es correcto y que los resultados de las comparaciones coinciden con lo que habías previsto.

Anterior



Índice



Siguiente





Ver índice

# Operadores lógicos



## Operadores lógicos

Mediante operadores lógicos es posible evaluar un conjunto de **variables lógicas**, es decir, aquellas cuyos valores sean únicamente: VERDADERO o FALSO (1 ó NUL).

El resultado de esa evaluación será siempre 1 ó NUL.

### \$A AND \$B

El operador **AND** devuelve VERDADERO (1) en el caso de que **todas** las variables lógicas comparadas sean verdaderas, y FALSO (NUL) cuando **alguna** de ellas sea falsa.

### \$A && \$B

El operador **&&** se comporta de forma idéntica al operador **AND**. La única diferencia entre ambos es que *operan con distinta precedencia*.

Más abajo veremos el orden de precedencia de los distintos operadores.

### \$A OR \$B

Para que el operador **OR** devuelva VERDADERO (1) es suficiente que **una sola** de las variables lógicas comparadas sea **verdadera**. Únicamente devolverá FALSO (NUL) cuando **todas** ellas sean FALSAS.

### \$A || \$B

El operador **||** se comporta de forma idéntica al operador **OR**.

Su única diferencia es el orden de precedencia con el que opera.

### \$A XOR \$B

El operador **XOR** devuelve VERDADERO (1) sólo en el caso de que sea **cierta una sola** de las variables, y FALSO (NUL) cuando ambas sean ciertas o ambas sean falsas.

### ! \$A

Este operador **NOT** (negación) devuelve VERDADERO (1) si la variable lógica \$A es FALSA y devuelve FALSO (NUL) si el valor de esa variable \$A es VERDADERO.

## Sintaxis alternativa

Tal como hemos descrito los distintos operadores lógicos sería necesario que \$A y \$B contuvieran valores

## Operadores lógicos

Aquí tienes las *tablas de verdad* de los distintos operadores lógicos.

El operador AND						
Variables	\$a=32; \$b=0; \$c=-7; \$d=4.32; \$f="23 Ptas"			A	B	C
Ejemplo de sintaxis	\$x=\$a>\$b; \$y=\$a>\$c; \$z=\$a>\$f; echo (\$x AND \$y AND \$z);					
Otra sintaxis	echo (\$a>\$b AND \$a>\$c AND \$a>\$f);					
Condición A	Condición B	Condición C	A	B	C	Resultado
\$a>\$b	\$a>\$c	\$a>\$f	1	1	1	1
\$a>\$b	\$a>\$c	\$a<\$f	1	1		
\$a>\$b	\$a<\$c	\$a>\$f	1		1	
\$a<\$b	\$a>\$c	\$a>\$f		1	1	
\$a<\$b	\$a<\$c	\$a>\$f			1	
\$a<\$b	\$a>\$c	\$a<\$f		1		
\$a>\$b	\$a<\$c	\$a<\$f	1			
\$a<\$b	\$a<\$c	\$a<\$f				

El operador &&						
Variables	\$a=32; \$b=0; \$c=-7; \$d=4.32; \$f="23 Ptas"			A	B	C
Condición A	Condición B			Condición C	A	Resultado
\$a>\$b	\$a>\$c	\$a>\$f	1	1	1	1
\$a>\$b	\$a>\$c	\$a<\$f	1	1		
\$a>\$b	\$a<\$c	\$a>\$f	1		1	
\$a<\$b	\$a>\$c	\$a>\$f		1	1	
\$a>\$b	\$a<\$c	\$a>\$f			1	
\$a<\$b	\$a>\$c	\$a<\$f		1		
\$a>\$b	\$a<\$c	\$a<\$f	1			
\$a<\$b	\$a<\$c	\$a<\$f				

El operador OR						
Variables	\$a=32; \$b=0; \$c=-7; \$d=4.32; \$f="23 Ptas"			A	B	C
Condición A	Condición B			Condición C	A	Resultado
\$a>\$b	\$a>\$c	\$a>\$f	1	1	1	1
\$a>\$b	\$a>\$c	\$a<\$f	1	1		1
\$a>\$b	\$a<\$c	\$a>\$f	1		1	1
\$a<\$b	\$a>\$c	\$a>\$f		1	1	1
\$a<\$b	\$a<\$c	\$a>\$f			1	1
\$a<\$b	\$a>\$c	\$a<\$f		1		1
\$a>\$b	\$a<\$c	\$a<\$f	1			1
\$a<\$b	\$a<\$c	\$a<\$f				

El operador						
Variables	\$a=32; \$b=0; \$c=-7; \$d=4.32; \$f="23 Ptas"			A	B	C
Condición A	Condición B			Condición C	A	Resultado
\$a>\$b	\$a>\$c	\$a>\$f	1	1	1	1
\$a>\$b	\$a>\$c	\$a<\$f	1	1		1
\$a>\$b	\$a<\$c	\$a>\$f	1		1	1
\$a<\$b	\$a>\$c	\$a>\$f		1	1	1
\$a<\$b	\$a>\$c	\$a<\$f		1		1
\$a>\$b	\$a<\$c	\$a<\$f	1			1
\$a<\$b	\$a<\$c	\$a<\$f				

lógicos, y eso requeriría un paso previo para asignarles valores de ese tipo.

Habrá que recurrir a procesos de este tipo:

```
$A = $x>$y;
$b= $x >=$z;
$A && $B;
```

pero se obtendría el mismo resultado escribiendo:

```
$x>$y && $x >=$z;
```

que, aparte de ser la forma habitual de hacerlo, nos evita dos líneas de instrucciones.

Aunque el propio ejemplo se auto comenta, digamos que al utilizar operadores lógicos se pueden sustituir las variables lógicas por expresiones que den como resultado ese tipo de valores.

### Orden de precedencia

Cuando se usan los operadores lógicos se plantean situaciones similares a lo que ocurre con las operaciones aritméticas.

Dado que permiten trabajar con secuencias de operaciones sería posible, por ejemplo, una operación de este tipo:

```
$a<$b OR $c<$b && $a<
```

Surgirían preguntas con estas:

- ¿qué comparación se haría primero OR ó &&?
- ¿se harían las comparaciones en el orden natural?
- ¿alguno de los operadores tiene prioridad sobre el otro?

Igual que en las matemáticas, también aquí, hay un orden de prioridad que es el siguiente:

**NOT, &&, ||, AND, XOR** y, por último, **OR**.

De esta forma la operación && se realizaría antes que ||, mientras que si pusiéramos AND en vez de && sería la operación || la que se haría antes y, por lo tanto, los resultados podrían variar de un caso a otro.

Aquí también es posible, de la misma manera que en la aritmética, utilizar paréntesis para priorizar una operación frente a otra.

Es muy importante prestar atención a la *construcción correcta* de estas estructuras. Un descuido en la atención a las prioridades puede ser origen –lo es frecuentemente– de resultados incoherentes que suelen ser detectados bajo una apariencia aleatoria.

Ten muy en cuenta que al depurar programas no siempre se ven

\$a<\$b	\$a>\$c	\$a<\$t	1	1
\$a>\$b	\$a<\$c	\$a<\$f	1	1
\$a<\$b	\$a<\$c	\$a<\$f		

El operador XOR				
Variables	\$a=32; \$b=0; \$c=-7; \$d=4.32; \$f="23 Ptas"	A	B	Resultado
Condición A	Condición B			
\$a>\$b	\$a>\$c	1	1	
\$a>\$b	\$a<\$c	1		1
\$a<\$b	\$a>\$c		1	1
\$a<\$b	\$a<\$c			

## Ejemplo de uso de los operadores lógicos

```
<?
# asignemos valores a cuatro variables
$a=3;
$b=6;
$c=9;
$d=17;
# utilicemos operadores de comparación
# y recojamos sus resultados en nuevas variables
$x= $a<$b;
$y= $a<$b;
$z=$c>$b;
# apliquemos un operador lógico (por ejemplo &&)
# e imprimamos el resultado

print("Resultado FALSO si no sale nada: ".($y && $z)."  
");
# hagamos la misma comparación sin utilizar la variables $y y $z
# que deberá darnos el mismo resultado
print("<br>Resultado FALSO si no sale nada: ".($a<$b && $c>$b)."  
");
/* veamos ahora que ocurre al ampliar la estructura
¿qué ocurriría si escribieramos
 $a<$b OR $c>$b && $d<$a ?
    El operador && tiene preferencia ante OR,
    luego haría primero la comparación $c>$b && $d<$a
    9 > 6 es cierto, 17 < 3 es falso, luego como &&
    solo devuelve cierto cuando ambos sean ciertos
    el resultado de esa opción es FALSO.
    Veamos que ocurre al comparar $a<$b OR falso (resultado anterior)
    como 3 < 6 es cierto OR operará con cierto y falso
    que dará como resultado CIERTO, ya que basta que se
    cumpla una de las condiciones */
/* vamos a comprobarlo mediante este operador condicional
no conocemos aun su sintaxis pero adelántemosla un poco...
si el el contenido del paréntesis que va detrás del if es cierto
imprimirá cierto y en caso contrario imprimirá falso
    Aquí debería imprimirnos cierto */
if($a<$b OR $c>$b && $d<$a) {
    print "cierto<br>";
} else{
    print "falso<br>";
}

# Cambiemos la estructura anterior por $a<$b || $c>$b AND $d<$a
# ahora se operará primero || que como antes dará cierto
# pero ese resultado operado mediante AND con falso dará FALSO
# AL CAMBIAR LOS OPERADORES POR SUS SIMILARES el resultado el DISTINTO

if($a<$b || $c>$b AND $d<$a) {
    print "cierto<br>";
} else{
    print "falso<br>";
}
# un paréntesis nos devolverá a la situación anterior
# escribamos $a<$b || ($c>$b AND $d<$a)
# veremos que el resultado es CIERTO como ocurría en el primer caso

if($a<$b || ($c>$b AND $d<$a)) {
    print "cierto<br>";
} else{
    print "falso<br>";
}
```

fácilmente este tipo de errores de programación.

Puede que para determinados valores los resultados sean los esperados y que, sin embargo, al utilizar otros distintos pueda manifestarse la incoherencia.

Si te ocurriera esto no pienses que es el ordenador el que está *haciendo cosas raras*, procura revisar los paréntesis y los criterios de prioridad de los distintos operadores contenidos en la estructura lógica.

?>

ejemplo23.php

### Ejercicio nº 18

Define tres variables: *langosta*, *angula* y *caviar*; y asígnale valores numéricos. Luego, escribe una estructura lógica que devuelva **cierto** en el caso de que **dos de ellas superen** los precios respectivos de: **50**, **70** y **500**, y **falso** en cualquier otro caso. Compruébalo modificando los valores de las variables y viendo los resultados y guárdalo como **ejercicio18.php**.

Anterior



Índice



Siguiente





Ver índice

## Operadores de incremento



### Operadores de incremento

Los caracteres **`++`** y **`--`** escritos al lado del nombre de una variable producen *incrementos* o *decrementos* de **una unidad** en el valor de la misma.

De igual forma, los operadores **`+=n`** y **`-=n`** escritos a la derecha del nombre de una variable producen *incrementos* o *decrementos* de **n** unidades en el valor de la variable.

Como veremos a continuación, los operadores **`++`** y **`--`** se comportan de distinta forma según estén situados a la **izquierda** o a la **derecha** de la variable.

### Tipos válidos

Estas operaciones sólo tienen sentido en variables numéricas –enteras o no–, pero, si se aplican a variables de *cadena* les asignan previamente *valor cero*, salvo en una *curiosa excepción* que puedes ver en la primera de las tablas que tienes aquí a la derecha.

### Operadores de preincremento

#### **`++$A` y `--$A`**

Este operador **incrementa** el valor de la variable en una unidad (**+1** o **-1**) **antes** de ejecutar el contenido de la instrucción.

#### **`$A+=n` y `$A-=n`**

Este operador **incrementa** el valor de la variable en **n** unidades (**+n** o **-n**) y luego ejecuta el contenido de la instrucción.

### Operadores de post-incremento

#### **`$A++` y `$A--`**

Cuando los operadores **`++`** o **`--`** están situados a la derecha de la variable los incrementos **no se producen** hasta que se ejecute la *instrucción siguiente*.

### Operadores de preincremento

El operador <b><code>++\$A</code></b>					
Variables numéricas			Variables alfanuméricas		
Valor inicial	Sintaxis	Resultado	Valor inicial	Sintaxis	Resultado
<code>\$a=23</code>	<code>echo ++\$a</code>	24	<code>\$b="pepe"</code>	<code>echo ++\$b</code>	<code>pepf</code>
<code>\$a=23</code>	<code>echo ++\$a*2</code>	48	<code>\$b="pepe"</code>	<code>echo ++\$b*2</code>	0

El operador <b><code>--\$A</code></b>					
Variables numéricas			Variables alfanuméricas		
Valor inicial	Sintaxis	Resultado	Valor inicial	Sintaxis	Resultado
<code>\$a=23</code>	<code>echo --\$a</code>	22	<code>\$b="pepe"</code>	<code>echo --\$b</code>	<code>pepe</code>
<code>\$a=23</code>	<code>echo --\$a*2</code>	44	<code>\$b="pepe"</code>	<code>echo --\$b*2</code>	0

El operador <b><code>\$A +=n</code></b>					
Variables numéricas			Variables alfanuméricas		
Valor inicial	Sintaxis	Resultado	Valor inicial	Sintaxis	Resultado
<code>\$a=23</code>	<code>Búsqueda rápida</code>		<code>\$b="pepe"</code>	<code>Página anterior</code>	<code>Página siguiente</code>
<code>\$a=23</code>	<code>echo 2*\$a+=5;</code>	56	<code>\$b="pepe"</code>	<code>echo 2*\$b+=5;</code>	10
	<code>echo 2*\$a</code>	56		<code>echo 2*\$b</code>	10

El operador <b><code>\$A -=n</code></b>					
Variables numéricas			Variables alfanuméricas		
Valor inicial	Sintaxis	Resultado	Valor inicial	Sintaxis	Resultado
<code>\$a=23</code>	<code>echo \$a=5;</code>	18	<code>\$b="pepe"</code>	<code>echo \$b=5;</code>	-5
<code>\$a=23</code>	<code>echo \$a</code>	18		<code>echo \$b</code>	-5
<code>\$a=23</code>	<code>echo 2*\$a=5;</code>	36	<code>\$b="pepe"</code>	<code>echo 2*\$b=5;</code>	-10
	<code>echo 2*\$a</code>	36		<code>echo 2*\$b</code>	-10

### Operadores de post-incremento

El operador <b><code>\$A++</code></b>		
Valor inicial de la variable	Sintaxis	Resultado
<code>\$a=23</code>	<code>echo \$a++;</code>	23
	<code>echo \$a</code>	24

El operador <b><code>\$A--</code></b>		
Valor inicial de la variable	Sintaxis	Resultado
<code>\$a=23</code>	<code>echo \$a--;</code>	23
	<code>echo \$a</code>	22

Anterior

Índice

Siguiente



Ver índice

# Operadores condicionales



## Operadores condicionales

Este tipo de operadores son el auténtico *cerebro* de cualquier aplicación que desarrollemos en PHP o en cualquier otro lenguaje de programación.

Los operadores condicionales son la herramienta que permite tomar decisiones tales como: *hacer o no hacer*, y también: *hacer algo bajo determinadas condiciones y otra cosa distinta en caso de que no se cumplan*.

### Condiciones

Aunque para simplificar los ejemplos vamos a utilizar en ellos una sola condición, este operador permite incluir como tal cualquier estructura lógica, del tipo que hemos visto en la página anterior, por compleja que esta sea.

### Alternativas de sintaxis

Como iremos viendo a lo largo de estas líneas, este operador permite diferentes formas de sintaxis que podemos utilizar según nuestra conveniencia.

La forma más simple es:

```
if(condición)
    ..instrucción...;
```

Si se cumple la *condición* establecida en el paréntesis se ejecutará la primera instrucción que se incluya a continuación de ella.

Cualquier otra instrucción que hubiera a continuación de esa primera no estaría afectada por el condicional y se ejecutaría en cualquier circunstancia.

Observa que, aunque hemos puesto *if(condición)* en una línea independiente, *no lleva punto y coma detrás*.

```
if(condición){
    ..instrucción 1... ;
    ..instrucción 2... ;
    ....;
}
```

Es una ampliación del caso anterior. Cuando es necesario que –en caso de que se cumpla la condición o condiciones– se ejecute más de una instrucción, se añade una **{** para indicar que habrá varias instrucciones, se escriben estas y

## El operador if

```
<?
# Definamos dos variables y lasignémolas valores.
# Hubieran podido obtenerse por cualquier otro procedimiento:
# desde un array,
# a través de un formulario cuya action ejecute este script, etc.

$A=3; $B="3";
if ($A==$B)
    print ("A es igual B");

# cualquier otra instrucción que incluyeramos de aquí
# en adelante se ejecutaría independientemente de que la condición
# se cumpla ó no ya que esta forma de if (sin llaves)
# únicamente considera la primera instrucción
# comprobémoslo en este otro supuesto

if ($A<$B)
    print ("A es menor que B");
    print("<br>A no es menor que b, pero esto saldrá<br>");
    print("Esta es la segunda instrucción. No la condicionará el if");
?>
```

ejemplo24.php

```
<?
$A=3; $B="3";
# en este caso cerraremos entre llaves las lineas
# que deben ejecutarse si se cumple la condición

if ($A==$B) {
    print ("A es igual B");
    echo "<br>";
    echo "Este if tiene varias instrucciones contenidas entre llaves";
}
# una sintaxis alternativa a las llaves
# sustituymos la { por : y la } por endif
if ($A==$B):
    print ("A es igual B");
    echo "<br>";
    echo "Hemos cambiado {} por : endif";
endif;
?>
```

ejemplo25.php

```
<?
$a=5;
# observa que ponemos la etiqueta de fin de script
# después de la llave de apertura
if ($a==5){ ?>
<!-- Aquí estamos poniendo HTML puro
    no estamos dentro del script PHP //-->
<H1>Esto no ha sido interpretado por PHP</H1>

<!-- en la linea siguiente a este comentario
    volveremos a PHP para insertar la llave que indica el fin del if //-->
<? } ?>

<?
# hagamos lo mismo cambiando {} por : endif
```

mediante } se señala el final.

```
if(condición):  
    ..instrucción 1... ;  
    ..instrucción 2... ;  
    ....;  
endif;
```

Esta otra forma del condicional se comporta como la anterior pero con otra sintaxis. Se **sustituye** la { de apertura por : y la } de cierre por endif

```
if(condición){ ?>  
    ..Etiquetas HTML... ;  
    ..HTML... ;  
    ....;  
<? } ?>
```

PHP permite la utilización del operador condicional if con esta sintaxis. Una primer script PHP establece la condición. Todo lo contenido entre ese primer script y el de cierre: <? } ?> será código HTML (está fuera del script), que se insertará en el documento sólo en el caso de que se cumpla la condición.

```
if(condición) : ?>  
    ..Etiquetas HTML... ;  
    ..HTML... ;  
    ....;  
<? endif; ?>
```

Idéntica a la anterior, con la sintaxis : , endif.

### If ... else

El operador condicional tiene una interesante ampliación. En conjunción con else permite añadir instrucciones que sólo serían ejecutadas en caso de no cumplirse la condición.

Esta nueva opción se habilita mediante la siguiente sintaxis:

```
if(condición){  
    ... instrucciones...  
    ... a ejecutar cuando  
    se cumple la condición  
    } else {  
    ... instrucciones...  
    ... a ejecutar cuando NO  
    se cumple la condición  
}
```

permitiendo también la sintaxis alternativa :, endif, aunque en este caso hay que hacer una precisión -puedes verla aquí debajo- la llave de cierre que iba delante de else se elimina y no es sustituida por ningún carácter ni símbolo especial.

```
<?if(condición): ?>
```

```
... código HTML  
... a ejecutar cuando  
se cumple la condición
```

```
<? else: ?>
```

```
--> -->  
<!-- Aquí estamos poniendo HTML puro  
no estamos dentro del script PHP //-->  
<H2>Esto tampoco sido interpretado por PHP</H2>  
  
<!-- en la linea siguiente a este comentario  
volveremos a PHP para insertar la llave que indica el fin del if //-->  
<? endif; ?>
```

ejemplo26.php

### La estructura if ... else

```
<?  
$A=3; $B="4";  
if ($A==$B) {  
#estas instrucciones se ejecutarían si se cumple la condición  
    print ("A es igual B");  
    echo "<br>";  
    echo "Este if tiene varias intrucciones";  
} else {  
# estas se ejecutarían en el caso de no cumplirse  
# las condiciones especificadas en el fi  
    print("A no es igual que B");  
    echo "<br>";  
    echo ("La estructura de control se ha desviado al else");  
}  
?>
```

ejemplo27.php

```
<?  
$a=3;  
# observa que ponemos la etiqueta de fin de script  
# después de los dos puntos  
if ($a==5): ?>  
<!-- Aquí estamos poniendo HTML puro  
no estamos dentro del script PHP //-->  
<H1>Esto no es PHP. A es igual 5</H1>  
  
<!-- en la linea siguiente a este comentario  
volveremos a PHP para insertar el else seguido de dos puntos  
y cerramos de nuevo el script con ?>//-->  
<? else: ?>  
  
<!-- Aquí más HTML para el (else)  
caso de que no se cumpla la condición //-->  
<H2>Esto no es PHP. Es el resultado del ELSE</H2>  
  
<!--  
volveremos a PHP para insertar en endif que indica el fin del if //-->  
<? endif; ?>
```

ejemplo28.php

### El operador condicional ternario

```
<? $a=5;  
($a==8) ? ($B="El valor de a es 8") : ($B="El valor de a no es 8");  
echo $B;  
?>
```

Ejemplo con a=8

Ejemplo con a=5

### La estructura if ... elseif... else

... código HTML...  
... a ejecutar cuando NO se cumple la condición

<? endif; ?>

En algunos casos resulta útil y cómodo el uso de esta otra posibilidad de sintaxis:

(condición) ? (opc1) : (opc2)

Si se cumple la condición se ejecuta la opc1, pero en el caso de que no se cumpla se ejecutará la opc2.

### If ... elseif .. else

Otra posibilidad dentro de la estructura de los operadores condicionales es la inclusión de elseif.

Esta es la sintaxis. (Dentro de ella tienes los comentarios explicativos).

if(condición1){

... instrucciones...

... a ejecutar cuando se cumple la condición1

}elseif(condición2){

... instrucciones...

... a ejecutar cuando se cumple la condición2 sin cumplirse condición1

} else {

... instrucciones...

... a ejecutar cuando NO se cumple ni la condición1 ni la condición2

}

### Condicionales anidados

El anidado no es otra cosa que el equivalente a los paréntesis dentro de paréntesis en las matemáticas. Y este operador lo permite, con una única condición, que verás en esta muestra de sintaxis.

```
if(condición1){  
    ... instrucciones...  
    if(condición2){  
        ... instrucciones...  
    } else {  
        ...instrucciones  
    }  
}  
  
}else{  
    ... instrucciones...  
    ...instrucciones...  
}
```

Observa que todo el bloque if.. else... marcado en azul se cierra antes de abrir la opción else marcada en marrón. Es obligatorio que así sea. De igual forma, podríamos insertar bloques sucesivos hasta llenar a

```
<? $a=1;  
if ($a==1){  
    echo "El valor de la variable A es 1";  
}elseif ($a==2){  
    echo "El valor de la variable A es 2";  
}elseif ($a==3){  
    echo "El valor de la variable A es 3";    echo "La variable A no es 1, ni 2, ni 3";  
?>
```

Ejemplo con a=3

Ejemplo con a=-7

```
<? $a=1;  
if ($a==1): ?>  
    <H1>A es igual a 1</H1>  
<? elseif($a==2): ?>  
    <H1>A es igual a 2</H1>  
<? elseif($a==3): ?>  
    <H1>A es igual a 3</H1>  
<? else: ?>  
    <H1>A no es igual ni a 1, ni a 2, ni a 3</H1>  
<? endif;  
?>
```

Ejemplo con a=2

Ejemplo con a=8

### Ejercicio nº 19

Diseña un formulario –ejercicio19a.php– con un input tipo texto en el que puedes escribir números. Al pulsar el botón de enviar debe llamar a un script –ejercicio19b.php– que debe decirnos si el número enviado fue: positivo, cero o negativo.

A la página ejercicio19b.php añádele un enlace HTML que permita volver a la página anterior.

### Ejercicio nº 20

En el ejercicio nº 10 –puedes verlo pulsando aquí– diseñaste un cuestionario en el que formulabas dos preguntas. Utilizando un formulario similar, pero únicamente con la primera pregunta –puedes modificarlo y guardarla como ejercicio20a.php– debes crear un script de modo que al recibir el formulario muestre en pantalla «Respuesta correcta» ó «Respuesta incorrecta».

Como es lógico, en ese script –puedes llamarlo ejercicio20b.php– debes incluir en una variable el valor de la respuesta correcta y compararla con la recibida a través del formulario.

### Ejercicio nº 21

Amplía el ejercicio anterior a las dos preguntas que se formulaban en el nº10. Ahora deberíamos saber si ha sido correcta o no la respuesta a cada una de las preguntas. Puedes llamar ejercicio21a.php y ejercicio21b.php a los documentos que crees para este ejercicio.

### Restringir accesos

Las variables predefinidas `$_SERVER['HTTP_REFERER']` (en el caso de PHP 4.1.0 o superior) y

crear una estructura tan amplia como fuera necesaria.

Como ves, todo un mundo de posibilidades.

## Una aplicación a la seguridad

En páginas anteriores hemos hecho algunas alusiones a la seguridad.

Decíamos que los envíos de información por medio de los formularios no eran seguros porque, dada la transparencia de su código, pueden ser reproducidos y utilizados desde cualquier otro sitio distinto a nuestro servidor.

Una sencillo condicional puede resolver ese problema. Lo puedes ver en el ejemplo que tienes aquí a la derecha.

## La función exit()

PHP dispone de una función **exit()** muy útil a los efectos del comentario anterior.

Cuando se ejecuta **exit()** se **interrumpe** la ejecución del script con lo que la respuesta del servidor a la petición del cliente incluirá únicamente los contenidos generados antes de su ejecución.

**\$HTTP\_SERVER\_VARS['HTTP\_REFERER']** (en todos los casos) recogen la ruta completa de la página desde la que hemos accedido a la actual.

```
<?
# el condicional if estable como condición
# que el acceso a este script proceda de la dirección indicada
# en este caso hemos puesto como condición que ese valor
# sea la dirección de esta página
if($_SERVER['HTTP_REFERER']=="http://localhost/cursophp/php37.php"){
# si accedemos desde esta página, el enlace que tienes aquí debajo
# veremos que aparece este print, es decir se visualizaría todo
# lo contenido antes del else
    print "ejecuto sin problemas el script.";
    print "Vienes de:".$_SERVER['HTTP_REFERER'];
}
else{
# si accedes desde un sitio diferente te aparecerá este mensaje
# puedes probar escribiendo en tu navegador
# http://localhost/cursophp/ejemplo35.php
# y comprobarás que aparece este mensaje
"No puedes ver esta página";
    exit;
}
?>
```

ejemplo35.php

Anterior

Índice

Siguiente



Ver índice

# La función switch



## La función switch

Una alternativa al uso de condicionales del tipo `if` es la función `switch`.

Se trata de un *condicional* que *evalúa una variable* y, según su valor, ejecuta unas instrucciones u otras.

Su sintaxis es la siguiente:

```
switch ( variable ) {
    case n1:
        instrucciones caso n1...
        .....
    case n2:
        instrucciones caso n2...
        .....
}
```

Cuando se usa esta sintaxis sólo se ejecutan aquellas instrucciones que han sido incluidas *a partir* de la etiqueta en la que el número que *sigue a case* coincide con el valor de la variable.

La forma más habitual de uso de esta función es esta:

```
switch ( variable ) {
    case n1:
        instrucciones caso n1...
        .....
    break;
    case n2:
        instrucciones caso n2...
        .....
    break;
}
```

Esta opción incluye antes de **cada nuevo case** la función de ruptura `break`. Cuando PHP *encuentra el break* *interrumpe* la ejecución y *no la reanuda* hasta la instrucción siguiente a la `}` que cierra la función `switch`.

Insertando `break` en cada una de las opciones `case`, sólo se ejecutarán las instrucciones contenidas entre `case num` y ese `break`.

**default:**

Bajo este nombre (`default:`) se pueden incluir –dentro de la función `switch`– un conjunto de instrucciones que *solo serán ejecutadas* en el

```
<?
# esta es la variable que controlará la función switch
$si=1;

switch ($si) {

#insertamos la etiqueta case 0 y a continuación
# las instrucciones correspondientes

    case 0:
        print "i es igual a 0 - No he puesto el break<br>";

# insertamos la etiqueta case 1 y a continuación
# las instrucciones correspondientes
# como no hemos puesto break y en este ejemplo $si=1
# se ejecutarán todas las instrucciones escritas
# de aquí en adelante

    case 1:
        print "i es igual a 1 - No he puesto el break<br>";
    case 2:
        print "i es igual a 2 - No he puesto el break<br>";

};

# ahora incluiremos break al final de las intrucciones de cada case
# con ello lograremos que solo se ejecuten las instrucciones correspondientes
# a cada uno de ellos
switch ($si) {
    case 0:
        print "i es igual a 0 - Ahora lleva break<br>";
        break;
    case 1:
        print "i es igual a 1 - Ahora lleva break<br>";
        break;
    case 2:
        print "i es igual a 2 - Ahora lleva break<br>";
        break;
}
```

[Ver índice](#)
[Búsqueda rápida](#)
[Página anterior](#)
[Página siguiente](#)

?&gt;

[ejemplo36.php](#)

```
<? $i=3;
switch ($i) {
    case 0:
        print "La variable i es 0<br>";
        break;
    case 1:
        print "La variable i es 1<br>";
        break;
    case 2:
        print "La variable i es 2<br>";
        break;
# al introducir default y dado que $i=3 se ejecutarán
# las instrucciones contenidas aquí ya que la variable
# no coincide con ninguno de los case establecidos
    default:
        print "La variable i es mayor que dos o menor que cero";
        break;
}
?>
```

caso que el valor de la variable *no coincide* con ninguno de los **case**. Su comportamiento es equivalente a **else** en el condicional **if**.

Anterior

Índice

Siguiente





Ver índice

# Bucles while



## Los bucles

La necesidad de *repetir* la ejecución de instrucciones es algo habitual en el mundo de la programación.

Frente a la alternativa –poco práctica– de escribir esas instrucciones todos los lenguajes de programación disponen de funciones que pueden ejecutar un bloque de instrucciones de forma repetitiva.

## La instrucción while

Como ocurría en el caso de *if*, el parámetro *condición* permite cualquier estructura lógica, y también dispone de distintas opciones de sintaxis.

**while(condición)**  
...*instrucción*

Con esta sintaxis estaremos indicando que la *instrucción siguiente* (sólo una instrucción) ha de ejecutarse continua y repetidamente hasta que deje de cumplirse la condición establecida.

**while(condición){**  
...*instrucción*  
.....  
**}**

De forma similar a la utilizada en el caso de *if*, también en este caso, las llaves hacen la función de *contenedores* de las instrucciones cuya ejecución debe repetirse mientras se cumpla la condición.

**while(condición):**  
...*instrucción*  
.....  
**endwhile;**

También aquí se mantiene la similitud con la sintaxis del condicional *if*. La llave (*{*) pueden sustituirse por (*:*) y en este caso en vez de (*}*) habría que escribir **endwhile**.

**while(condición) :?**  
...*etiquetas HTML*  
.....  
**<? endwhile; ?>**

También *while* permite *cerrar el script PHP* después de (*:*) o de la sintaxis alternativa (*{}*) e insertar etiquetas HTML, indicando más tarde el final del bucle con **<? } ?>** o **<? endwhile; ?>**, según proceda.

## Whiles anidados

```
<?
# asignemos un valor a la variable $A
$A=0;
/* establezcamos la condición menor que cinco
   e insertemos dentro de la instrucción algo que modifique
   el valor de esa variable de modo que en algún momento
   deje de cumplirse la condición;
   de no ocurrir esto, el bucle se repetiría indefinidamente
   en este ejemplo el autoincremento ++ de la variable
   hará que vaya modificándose su valor*/
while ($A<5) echo "El valor de A es: ",$A++,"<br>";
# comprobemos que este while solo ejecuta una instrucción
# la delimitada por el punto y coma anterior
print("Esto solo aparecerá una vez. While no lo incluye en su bucle");
?>
```

ejemplo38.php

```
<?
$A=0;
/* utilicemos ahora el bucle para crear un tabla HTML
   empiezamos escribiendo la etiqueta de apertura de esa tabla
   fuera del bucle (ya que esa se repite una sola vez)
   y utilicemos el bucle para escribir las celdas y sus contenidos */
print ("<table width=300 border=2>");

while ($A<=5){
    echo "<tr><td align=center>";
    print $A;
}
# esta instrucción es importantísima
# si no modificamos el valor de $A el bucle sería infinito
    $A++;
    print("</td></tr>");
}
# cerremos la etiqueta table
print "</table>";
?>
```

ejemplo39.php

```
<?
# utilicemos whiles anidados para construir una tabla de
$filas=5; $columnas=3;
# insertemos la etiqueta de apertura de la tabla
print ("<table border=2 width=400 align=center>");

# un primer while rojo que utiliza la condición filas mayor que cero
# en este caso, la variable tendrá que ir disminuyendo su valor con $filas
# para escribir las etiquetas <tr> y </tr>
# y el modificador de la variable filas
# y un segundo while (magenta) para insertar las etiquetas correspondiente
# a las celdas de cada fila
while ($filas>0):
    echo "<tr>";
    $filas--;
        while ($columnas>0):
            echo "<td>";
            print "fila: ".$filas." columna: ".$columnas;
            print "</td>";
        
```

Una nueva similitud sintáctica con *if*. En el caso de *while* también es posible insertar un *while* dentro de otro *while* utilizando una sintaxis de este tipo:

```
while(condición1):
...instrucción
  while(condición2) {
    ...instrucción
    ....
  }
.....
endwhile;
```

En esta descripción hemos utilizado dos sintaxis distintas. Por si acaso dudaras de si es necesario o no hacerlo de esa forma, te diremos que **no es necesario** nunca. El hecho de la *anidación* no limita un ápice las posibilidades de la sintaxis.

### Ver código fuente

PHP dispone de la función

```
show_source('pag');
```

que permite visualizar el código fuente del *documento* que se indica en el parámetro *pag*.

Es muy útil para los propósitos de este curso, pero presenta un problema de **seguridad**.

Si escribes –en el parámetro *pag*– la dirección completa de una web cualquiera (que tenga extensión *php*) se visualizará su contenido, salvo que el PHP instalado en el servidor que la aloja la tenga expresamente desactivada.

### Recuerda...

En HTML se puede asignar el color fondo a una celda incluyendo *bgcolor=RGB(x,y,z)* dentro de la etiqueta *<TD>*.

*x*, *y* ,*z* son los valores de las componentes de cada color primario.

```
$columnas--;
endwhile;
/* ¡muy importante!. Tendremos que reasignar a la variable columnas
   su valor inicial para que pueda ser utilizado en la proxima fila
   ya que el bucle (magenta) va reduciendo ese valor a cero
   y en caso de no restaurar el viejo valor no volvería a ejecutarse
   ya que no cumple la condición de ser mayor que cero */
$columnas=3;
echo "</TR>";
endwhile;
# por ultimo la etiqueta de cierre de la tabla
print "</table>";
?>
```

ejemplo40.php

## Insertando condicionales en un bucle *while*

En este nuevo ejemplo hemos modificado ligeramente el anterior, incluyendo un condicional *if*. No incluimos aquí el código fuente para evitar la monotonía de repetir íntegramente y con ligeras modificaciones el supuesto anterior.

Para visualizar ese código bastará que pulses en el enlace *Ver código fuente*. Las modificaciones que hemos introducido aparecen marcadas y podrás localizarlas rápidamente.

Utilizaremos a menudo esta forma de visualización del código fuente de los scripts. Es una opción de uso muy simple, utilizando la función *show\_source()*, la que aprovechamos para comentar al margen.

Ver nuevo ejemplo

Ver código fuente

## Ejercicio nº 22

Escribe un script –**ejercicio22.php**– en el que, mediante un bucle **while**, construya una tabla cuyas celdas tengan como *colores de fondo* una escala de grises que comience en *RGB(0,0,0)* y acabe en *RGB(255,255,255)* a intervalos de 5 unidades.

Recuerda que los diferentes tonos de grises se forman combinando valores iguales de los tres colores primarios.

Anterior

Índice

Siguiente



Ver índice



## Bucles do ... while

### El bucle do... while

Estamos ante una variante del bucle *while* que hemos visto en la página anterior.

La sintaxis es la siguiente:

```
do {  
    ...instrucción 1...  
    .... instrucción2...  
} while(condición);
```

Se diferencia de *while* en que en este caso se comprueba la condición **después** de haber ejecutado las instrucciones contenidas en el bucle, con lo cual, en el caso de que desde el comienzo *no se cumplieran las condiciones establecidas en while*, las instrucciones del bucle se ejecutarían *una vez*.

Respecto a la sintaxis, como puedes observar, detrás de **do** se inserta una llave (**{**) que señala el comienzo de las instrucciones pertenecientes al bucle. El final de esas instrucciones lo señala la otra llave (**}**) que precede a **while(condición)**.

### break

La función **break** –de forma similar a lo que ocurría en *switch*– permite interrumpir la ejecución de bucle.

Tal como puede verse en el ejemplo, podría –entre otras posibilidades– utilizarse para evitar la primera ejecución de las instrucciones contenidas en el bucle, en el caso de que, desde un primer momento, no se cumplieran las condiciones establecidas en **while**.

### El bucle do ... while

```
<?  
$A=0;  
do {  
    ++$A;  
    echo "Valores de A usando el do: ",$A,"<br>";  
} while($A<5);  
$B=7;  
do {  
    echo "Pese a que B es mayor que 5 se ejecuta una vez. B= ",$B,"<br>";  
} while($B<5);  
?>
```

[ejemplo42.php](#)

```
<?  
$A=500;  
do {  
    if ($A>=500) {  
        echo "No puede ejecutarse el bucle, porque no se cumple la condición";  
        break;  
    }  
    ++$A;  
    echo "Valores de A usando el do: ",$A,"<br>";  
}  
} while($A<500);  
echo "<BR>He salido del bucle porque A es: ",$A;  
?>
```

[ejemplo43.php](#)

El ejemplo que tienes aquí debajo es similar al ejemplo nº 40 de la página anterior. Sólo hemos sustituido los bucles *while* que allí habíamos utilizado por otros del tipo *do ... while*.

[ejemplo44.php](#)

[Ver código fuente](#)

No lo hemos comentado en la página anterior pero **break** se comporta en el caso de **while** de forma idéntica a la descrita aquí.

[Anterior](#)

[Índice](#)

[Siguiente](#)

[Ver índice](#)

## Bucles for



### El bucle for

Se trata de una nueva forma –de uso bastante habitual– que permite establecer un bucle que se repetirá mientras una variable numérica se mantenga dentro de un intervalo establecido en la sintaxis del propio bucle– indicándose, también en la propia instrucción, el criterio de modificación de esa variable en cada ejecución del bucle.

La sintaxis es la siguiente:

```
for ( desde ; hasta ; incre ){
.....
...instrucciones....
}
```

El parámetro **desde** permite asignar un **valor inicial** a una variable (**\$var=num**) que hará funciones de **controladora de iteraciones**.

El parámetro **hasta** establece la condición que limita el valor máximo que puede alcanzar la variable de **control**.

El parámetro **incre** (con una sintaxis del tipo **\$variable++;** **\$variable--;** **++\$variable** **--\$variable;** **\$variable+=n** o **\$variable -=n**) establece los **incrementos o decrementos** de la variable **controladora** en cada iteración del bucle.

Las instrucciones contenidas entre **{ }** serán ejecutadas cada vez que se reitere el bucle.

### Variantes del bucle for

El bucle for permite algunas variantes respecto a su forma más general. Son estas:

```
for ( desde ; ; incre ){
.....
...instrucciones....
}
```

En este caso se omite el valor del parámetro **hasta** (observa que no se omite el separador de parámetros **(;)**) con lo que en realidad se está asignando a **hasta** el valor NUL.

Cuando se utiliza esta sintaxis, el bucle se repetiría de forma **indefinida** (la variable podría tomar cero como valor, pero, cero es distinto de NUL) salvo que -tal como puedes ver en el ejemplo- se escriba

### La estructura for

Aquí tienes algunos ejemplos de las diferentes variantes del bucle for.

```
<?
for ($i = 1; $i <= 10; $i++) {
    print $i."<br>";
}
?>
```

[ejemplo45.php](#)

```
<?
for ($i = 1;;$i++) {
    if ($i > 10) {
        break;
    }
    print $i."<br>";
}
?>
```

[ejemplo46.php](#)

```
< ?
$ i = 1;
for (;;) {
    if ($i > 10) {
        break;
    }
    print $i."<br>";
    $i++;
}
?>
```

[ejemplo47.php](#)[Ver índice](#)[Búsqueda rápida](#)[Página anterior](#)[Página siguiente ►](#)

```
for ($i = 1; $i <= 10; print $i."<br>", $i++) ;
?>
```

[ejemplo48.php](#)

```
<?
for($i = 1; $i <=10;$i++) :
    echo $i,"<br>";
endfor;
?>
```

[ejemplo49.php](#)

en las **instrucciones** un operador condicional con una opción de **ruptura del bucle** -el famoso **break** que ya hemos visto al estudiar la instrucción **while** y otras anteriores-.

```
for ( ; ; ){  
.....  
...instrucciones....  
.....  
}  
En este caso no se inserta ningún parámetro pero si se escriben los ; delimitadores de los mismos.
```

Si observas el ejemplo verás que el control se realiza fuera del **for**. El valor de la variable **contador** se asigna fuera del bucle, los incrementos de esa variable están escritos en las líneas de **instrucciones** y llevan un operador **condicional** con la función **break** para permitir la salida.

```
for( desd ; hast ; inst , incr )
```

Esta nueva variante de **for** permite insertar instrucciones a través del tercer parámetro de la función.

Si insertamos como tercer parámetro una conjunto de instrucciones, **separadas por comas**, se ejecutarán de igual forma que si estuvieran contenidas entre { y }

En este caso, el modificador de la variable de control (**incr**) se incluye como una instrucción más –separada por una coma– de las contenidas en ese tercer parámetro de la función.

```
for ( desde ; hasta ; incre ):  
.....  
...instrucciones....  
.....  
endfor;
```

Esta sintaxis es alternativa a la primera de las descritas. Sustituye la { por dos puntos (: ) y la } por **endfor**.

```
<? for ($i = 1; $i <= 10;$i++):?>  
    <H1>Esto se repetirá 10 veces</H1>  
<? endfor; ?>
```

ejemplo50.php

Como puedes observar en este último ejemplo también es aplicable aquí la sintaxis de los *dos scripts PHP*. El primero contiene las instrucciones del bucle y el segundo señala el final del mismo.

Entre ambos *scripts* se escribe el código HTML

### ¡Cuidado!

A la hora de programar bucles hay que evitar el riesgo de convertirlo en un bucle indefinido.

Cuando esto ocurre –el error es humano– al abrir la página que lo contiene parecerá que nuestro navegador se ha quedado **colgado** aunque en realidad estará esperando a que sea atendida la **petición**.

Si llega a planteártelo ese problema, tendrás que recurrir a la **socorrida** solución de pulsar ALT+CTRL+DEL para abortar la petición del navegador.

### Ejercicio nº 23

Siguiendo criterios similares a los del *ejemplo nº 40* elabora un script que permita construir una tabla de 5 filas y 7 columnas que contengan los sucesivos números naturales desde 1 hasta 35. Utiliza bucles del tipo **for**, que igual que while y do while permiten ser anidados. Guárdalo como **ejercicio23.php**

### Ejercicio nº 24

En este ejercicio –**ejercicio24.php**– trataremos de crear una tabla como la anterior, esta vez de una sola fila y seis columnas, contenido cada celda un **número aleatorio** comprendido entre **1 y 49** en la que habremos de evitar la posibilidad de que un número se repita dos veces (podría ser una forma de llenar la *primitiva*).

Para ello te sugerimos que guardes en un array los valores de los números aleatorios que se van generando y que, antes de guardar cada uno de ellos, se ejecute un bucle que compruebe si entre los registrados ya existe un valor igual al obtenido. Si no existiera ese valor se guardaría el dato, en caso contrario se repetiría la extracción.

Anterior



Índice



Siguiente





[Ver índice](#)

## Bucles foreach



### El bucle foreach

El bucle **foreach** es específico de los **array** y aplicable a ellos tanto si son **escalares** como si son de tipo **asociativo**.

Tiene dos posibles opciones. En una de ellas lee únicamente los valores contenidos en cada elemento del array. En el otro caso lee además los índices del array.

### Lectura de valores

Utiliza la sintaxis:

```
foreach( array as var ){  
...instrucciones...  
}
```

donde **array** es el **nombre del array** (sin incluir índices ni corchetes), **as** es una **palabra obligatoria** y **var** el nombre de una variable (puede ser creada al escribir la instrucción ya que no requiere estar previamente definida).

Las instrucciones escritas entre las **{** } permiten el tratamiento o visualización de los valores obtenidos.

La variable **var** no podrá ser utilizada para **guardar** valores. Hemos de tener en cuenta que su valor se **repite** en cada iteración del bucle y que al acabar este sólo contendrá el último de los valores leídos.

### Lectura de índices y valores

Con una sintaxis como la que sigue se pueden leer no sólo los valores de un array sino también sus índices.

```
foreach( array as v1 => v2 ) {  
...instrucciones...  
}
```

donde **array** es el nombre de la matriz, **as** es una **palabra obligatoria**, **v1** es el nombre de la variable que recogerán los índices, los caracteres **=>** (son obligatorios) son el separador entre ambas variables y, por último, **v2** es el nombre de la variable que recoge el valor de cada uno de los elementos del array.

Tanto esta función como la anterior realizan una **lectura secuencial** que comienza en el **primer valor** del array.

### Arrays bidimensionales

```
<?  
/* definimos un array escalar utilizando la sintaxis  
   nombre del array=array (valores de los elemento separados por comas)  
   si los valores son números no es necesario encerrarlos entre comillas */  
$a=array("a","b","c","d","e");  
/* definimos ahora un nuevo array, esta vez asociativo  
   utilizando la sintaxis clave => valor tal como puedes ver */  
$b=array(  
    "uno"  =>"Primer valor",  
    "dos"  =>"Segundo valor",  
    "tres"  =>"Tercer valor",  
);  
# establecemos el bucle que leerá el array $a  
# recogiendo en la variable $pepe los valores extraídos  
# y escribimos los valores  
foreach($a as $pepe) {  
echo $pepe,"<br>";  
};  
# repetimos el mismo proceso, ahora con $b que es un array asociativo  
foreach($b as $pepe) {  
echo $pepe,"<br>";  
};  
?>
```

**ejemplo51.php**

```
<?  
$a=array("a","b","c","d","e");  
$b=array(  
    "uno"  =>"Primer valor",  
    "dos"  =>"Segundo valor",  
    "tres"  =>"Tercer valor",  
);  
  
# en este caso extraeremos índices y valores de ambos arrays  
# usaremos $pepe para recoger los índices y $pepe para recoger los valores  
# y separaremos ambas variables por => que es el separador obligatorio  
# para estos casos  
  
foreach($a as $pepe=>$pepa) {  
    echo "Índice: ",$pepe," Valor: ",$pepa,"<br>";  
};  
foreach($b as $pepe=>$pepa) {  
    echo "Índice: ",$pepe," Valor: ",$pepa,"<br>";  
};  
?>
```

**ejemplo52.php**

### El bucle foreach en arrays bidimensionales

```
<?  
# definamos un array bidimensional  
$z=array(  
    0 => array (  
        0 => 34,  
        1 => 35,  
        2 => 36,  
    ),  
    1 => array (
```

Cuando se trata de arrays bidimensionales la lectura de los valores que contienen sus elementos requiere el uso de **dos bucles anidados**.

Cuando un array de este tipo es sometido al bucle **foreach** se extrae como *índice* el *primero de ellos* y como *valor* un **array unidimensional** que tiene como *índice* el *segundo del array original* y como valor el de aquél.

La lectura de los valores de cada elemento requiere utilizar un segundo bucle que los vaya extrayendo a partir del array unidimensional obtenido por medio del bucle previo.

La sintaxis sería de este tipo:

```
foreach($a as $i1=>$na){  
    foreach($na as $i2=>$val){
```

```
        ..$i1 es el primer índice...  
        ..$i2 es el segundo índice...  
        ..$na nuevo array  
        ..$valor es el valor  
        ....  
    }
```

En el caso de arrays con dimensiones superiores sería necesario proceder del mismo modo, y habría que *utilizar tantos bucles foreach* como *índices* contuviera el array.

```
        1 => 135,  
        2 => 136,  
    )  
};  
# intentemos leer indices y valores mediante un bucle foreach  
# y veamos como los valores que extraemos son Arrays (unidimensionales)  
# consecuencia del caracter bidimensional del array original  
  
foreach($z as $pepe=>$pepa) {  
    echo "Índice: ",$pepe," Valor: ",$pepa,"<br>";  
};  
/* anidemos dos bucles foreach de la siguiente forma  
en el primero extraemos un array que es tratado por  
el segundo foreach para extraer el segundo indice  
y el valor realmente contenido en ese elemento bidimensional */  
foreach($z as $ind1=>$valor1) {  
    foreach($valor1 as $ind2=>$valorReal) {  
        echo "Ind. 1: ",$ind1,"Ind. 2: ",$ind2," Valor: ",$valorReal,"<br>";  
    };  
};  
?>
```

[ejemplo52a.php](#)

### Ejercicio nº 25

Crea un array bidimensional que contenga los nombres de cinco alumnos y las calificaciones de tres materias (biología, física y latín por ejemplo). El primer índice puede ser el número de lista y los segundos pueden ser de tipo asociativo ('nombre', 'biología', 'física', 'latín', por ejemplo). Asignales valores a los elementos del array y completa el script de forma que se visualice un listado con los nombres de los alumnos y las calificaciones de cada una de las materias.

[Anterior](#)



[Índice](#)



[Siguiente](#)





[Ver índice](#)

## La instrucción continue



### La instrucción continue

Si la instrucción `break` permite interrumpir el desarrollo de un bucle, mediante `continue` se puede *impedir* que, bajo unas condiciones determinadas, se ejecuten *algunas o todas* las instrucciones de un bucle sin que por ello se interrumpa la ejecución de las iteraciones siguientes.

Esta instrucción es aplicable tanto a bucles `for` como a los de tipo `while` o `do while`.

Seguramente los ejemplos nos ayudarán a aclarar un poquito más la idea. En todos ellos hay un **condicional** que contiene la función `continue`.

El primero de ellos (*un bucle for*) tiene como condición: `$i % 2 == 0`, que, como recordarás, significa que *el resto de la división de \$i entre dos(\$i % 2)* sea igual (==) a cero.

En este supuesto (*condición de múltiplo de dos*) se activará la opción `continue` y por lo tanto en esa iteración no se ejecuta la instrucción `echo` o, lo que es lo mismo, *no se imprimirán en pantalla los múltiplos de 2*.

En el segundo ejemplo (caso de *bucle while*) la condición establecida para que se ejecute `continue` es que el valor de la variable **no sea múltiplo de tres**, en cuyo caso `echo` sólo imprimirá los **múltiplos de 3**.

El tercer ejemplo utiliza *un bucle do ... while* para presentar en pantalla los **múltiplos de 11**.

### La instrucción continue n

La instrucción `continue` puede utilizar un parámetro `n` con valor **entero positivo** que cuando no se indica toma por defecto el valor 1.

La idea es la siguiente. Cuando tenemos **bucles anidados** el intérprete de PHP los considera **numerados** correlativamente **-de dentro hacia fuera-** a partir de **UNO**.

Cuando es ejecutada `continue n` se *redirecciona la iteración al bucle*, cuyo **número coincide** con el valor de `n`.

### Ejemplos de continue

```
<?
for ($i=0;$i<=10;$i++) {

    #condicion de multiplo de 2
    if ($i % 2 ==0 ) {
        continue ;
    }

    echo "La variable I vale ",$i,"<br>";
}
?>
```

**ejemplo53.php**

```
<?
$i = 0;
while ($i++ < 14) {

    #condicion de no multiplo de 3 usando para distinto la sintaxis !=
    if ($i % 3 !=0){
        continue ;
    }

    echo "El valor de i es: ",$i,"<br>";
}
?>
```

**ejemplo54.php**

```
<?
$i = 0;
do {

    # condicion de no multiplo de 11. fijate en la sintaxis alternativa
    # observa que aquí distinto lo hemos escrito <>

    if ($i % 11 <>0 ){
        continue ;
    }

    echo "El valor de i es: ",$i,"<br>";
}while ($i++ < 100)
?>
```

**ejemplo55.php**

### Ejemplos de continue n

Obviamente, el valor de n no puede ser nunca mayor que el número de bucles anidados en el script.

Analicemos los ejemplos que tenemos a la derecha.

En el primer caso el bucle **for** sería el **UNO** y el **while** sería el **DOS**.

Cuando se cumpla la condición que activa **continue 2**, se redirecciona la iteración al paso siguiente del bucle **DOS**, en el caso del ejemplo al paso siguiente de **while**.

En el segundo ejemplo, como puedes ver, hemos anidado a tres niveles y hemos escrito **continue 3**, aunque a la hora de ejecutar los ejemplos podrás ver las **tres variantes** posibles de ese script modificando los valores del n de **continue**.

Fíjate en un **matiz** importante. Cuando el intérprete lee la instrucción **for** por primera vez lo hace a partir del valor  **inicial** de la variable que controla las iteraciones, pero ni **do... while** ni **while** tienen esa opción dado que por sí mismos no modifican las variables de control. Estos trabajan con condiciones mientras que **for** lo hacen con su variable de control.

Esa es la razón por la que en los ejemplos de los casos **continue 1** y **continue 2** la variable **k** no pasa del valor **0**, ya que al sobrepasar **j** el valor **5**, el bucle **while** no se ejecuta.

Si quieras que esas variables se reinicien al modo de **for** tendrás que añadir –dentro del **if** que contiene el **continue** correspondiente y antes de **continue**– una línea donde asignes a esas variables su valor  **inicial**.

```
<?
$j=0;
while (++$j <5) {
    for($i=1;$i<5;$i++) {

        if ($i==3){
            continue 2;
        }
        echo "El valor de j es: ",$j, " y el de i es: ",$i,"<br>";

    }
}
?>
```

Caso continue 1

Caso continue 2

```
<?
$j=0;$k=0;
do {
while (++$j <=5) {
    for($i=1;$i<=5;$i++) {

        if ($i==2){
            continue 3;
        }
        echo "El valor de k es: ",$k,
        " y el valor de j es: ",$j, " y el de i es: ",$i,"<br>";

    }
}
}while ($k++ <=5);
?>
```

Caso continue 1

Caso continue 2

Caso continue 3

Anterior

Índice

Siguiente



variable sea tipo **string** y NUL en caso contrario.

#### **unset(variable)**

Destruye la variable indicada. Si después de aplicar **unset** sobre una variable aplicamos de nuevo la función **isset**, nos devolverá NUL indicando que **ya no está definida**.

La función **unset** se puede aplicar tanto sobre *variables* como sobre un *array* y también sobre *un elemento de un array*.

```
echo "<h3>Eliminando variables</h3>";
echo "La dimensión de a es: ",count($a),"<br>";
unset($a[0]);
echo "Nueva dimensión de a: ",count($a),"<br>";
echo "La dimensión de b es: ",count($b),"<br>";
unset($b);
echo "Nueva dimensión de b: ",count($b),"<br>";
echo " ¿Sigue definida la variable b ? ",isset($b),"<br>";
?>
```

[ejemplo59.php](#)

[Anterior](#)

[Índice](#)

[Siguiente](#)



Ver índice

# Recuentos en arrays



## Recuento de valores

### Recuento de los valores contenidos en una matriz

Algunas de las posibilidades de obtener información sobre los contenidos de un array son las siguientes:

**\$n=array\_count\_values(ar)**

Con **array\_count\_values** la variable **\$n** será un array que tendrá como índices cada uno de los valores distintos que contenga el array ar, y como valores el resultado de contar el número de veces que se repite cada uno de los valores contenidos en el array inicial.

Distingue entre *mayúsculas* y *minúsculas*.

Cuando el array inicial (ar) contiene números enteros (sea ar escalar o asociativo) \$n será un array escalar. En caso contrario, será asociativo.

### Búsqueda de elementos en un array

**clav=array\_keys(arr)**

Devuelve un array escalar (clav) que contiene como valores los índices del array inicial (arr).

**clav=array\_keys(arr, valor)**

Devuelve un array escalar (clav) que contiene como valores los índices de los elementos del array inicial cuyo valor coincide con el indicado mediante el parámetro valor.

**valores=array\_values(arr)**

Esta función recoge en una nueva matriz (valores) todos los valores contenidos en otro array.

Es una forma de conversión de un array asociativo en otro escalar.

```
<?
$a=array(1,2,3,1,1,2,3,3,4,4,4,0,1);
$b=array("blanco","azul","blanco","blanco","azul","Blanco","Azul");
$c=array(
    "a"=>"rojo",
    "b" =>"verde",
    "c" =>"rojo",
    "d" =>"rojo",
    "e" =>"verde",
    "f" =>"Rojo",
    "g" =>"Verde");
echo "<h3>Cuenta valores del array()</h3>";
$contador=array_count_values($a);
foreach($contador as $valor=>$veces){
    echo "El valor \"$valor,\" se repite ",
        $veces," en la matriz a<br>";
}
echo $contador[0], "<br>";
echo $contador[1], "<br>";
echo $contador[2], "<br>";
echo $contador[3], "<br>";
echo $contador[4], "<br>";
$contador1=array_count_values($b);
foreach($contador1 as $valor=>$veces){
    echo "El valor \"$valor,\" se repite ",
        $veces," en la matriz a<br>";
}
echo $contador1["Blanco"], "<br>";
$contador2=array_count_values($c);
foreach($contador2 as $valor=>$veces){
    echo "El valor \"$valor,\" se repite ",
        $veces," en la matriz a<br>";
}
echo $contador2["rojo"], "<br>";
echo $contador2["Verde"], "<br>";
echo $contador2["verde"], "<br>";
echo $contador2["Rojo"], "<br>";
echo "<h3>Devuelve las claves de un array</h3>";
$claves=array_keys($a);
foreach($claves as $v){
echo "El valor \"$v,\" es una de las claves<br>";
}
$claves1=array_keys($a,1);
foreach($claves1 as $v){
echo "El valor \"$v,\" es una de las claves de elementos
    de la matriz cuyo valor es <b>1</b><br>";
}
echo "<h3>Devuelve los valores de un array</h3>";
$valores=array_values($c);
foreach($valores as $v){
echo "$v, \" Este es un de los valores de
    de la matriz c<br>";
}
?>
```

[Ver índice](#)
[Búsqueda rápida](#)
[Página anterior](#)
[Página siguiente](#)

```
echo $contador1["Blanco"], "<br>";
$contador2=array_count_values($c);
foreach($contador2 as $valor=>$veces){
    echo "El valor \"$valor,\" se repite ",
        $veces," en la matriz a<br>";
}
echo $contador2["rojo"], "<br>";
echo $contador2["Verde"], "<br>";
echo $contador2["verde"], "<br>";
echo $contador2["Rojo"], "<br>";
echo "<h3>Devuelve las claves de un array</h3>";
$claves=array_keys($a);
foreach($claves as $v){
echo "El valor \"$v,\" es una de las claves<br>";
}
$claves1=array_keys($a,1);
foreach($claves1 as $v){
echo "El valor \"$v,\" es una de las claves de elementos
    de la matriz cuyo valor es <b>1</b><br>";
}
echo "<h3>Devuelve los valores de un array</h3>";
$valores=array_values($c);
foreach($valores as $v){
echo "$v, \" Este es un de los valores de
    de la matriz c<br>";
}
?>
```

[ejemplo60.php](#)
[Anterior](#)
[Índice](#)
[Siguiente](#)

## Localización de valores en una matriz

### **in\_array(valor,array)**

La función **in\_array** busca en la matriz (**array**) el valor (numérico o cadena) contenido en el parámetro **valor**. Si lo encuentra devuelve 1, y, si no existiera devolvería NUL.

## Posicionamientos en una matriz

Mediante estas funciones se puede modificar la posición del **puntero interno** de una matriz y determinar los **índices** de los elementos a los que apunta en cada momento.

### **key(array)**

Devuelve el **índice** del elemento de la matriz al que **apunta** en ese momento el **puntero interno** de la matriz.

**¡Cuidado!**

Fíjate en el ejemplo. Pese a que hemos *pedido* a PHP que nos muestre, mediante la instrucción **echo \$a[3]**, esa lectura no mueve el **puntero interno**, que sigue señalando a la primera posición (devuelve CERO).

### **reset(array)**

Desplaza el **puntero interno** a la posición del **primer índice** del array.

### **end(array)**

Desplaza el **puntero interno** a la posición del **último índice** del array.

### **pos(array)**

Mantiene el **puntero interno** en la posición del **actual**.

### **next(array)**

Avanza el **puntero interno** en una posición respecto a la **actual**.

### **prev(array)**

Retrocede el **puntero interno** en una posición respecto a la **actual**.

## Moviendo el puntero interno

```
<?
$a=array(1,2,3,1,1,2,3,3,4,4,4,0,1);
$b=array("blanco","azul","blanco","blanco","azul","Blanco","Azul");
$c=array(
    "a"=>"rojo",
    "b" =>"verde",
    "c" =>"rojo",
    "d" =>"rojo",
    "e" =>"verde",
    "f" =>"Rojo",
    "g" =>"Verde");

echo "<h3>Busca un valor en una matriz</h3>";

echo " Busca el valor en la matriz: <b>#",in_array(3,$a),
    "#</b> Si no ha puesto nada no estaba, si 1 lo encontró <BR>";
echo " Busca el valor en la matriz: <b>#",in_array(7,$a),
    "#</b> Si no ha puesto nada no estaba, si 1 lo encontró <BR>";
echo " Busca el valor en la matriz: <b>#",in_array("gris",$b),
    "#</b> Si no ha puesto nada no estaba, si 1 lo encontró <BR>";
echo " Busca el valor en la matriz: <b>#",in_array("blanco",$b),
    "#</b> Si no ha puesto nada no estaba, si 1 lo encontró <BR><br>";

Ver índice Búsqueda rápida ◀ Página anterior Página siguiente ▶

echo "Este es el valor asociado al indice 3 de la matriz a: ",$a[3], "<br>";
echo "El puntero interno apunta a la clave: ",key($a), "<br>";
echo "Este es el valor siguiente al anterior: ",next($a), "<br>";
echo "El puntero interno apunta a la clave: ",key($a), "<br>";
echo "Este es el primer valor de la matriz a: ",reset($a), "<br>";
echo "El puntero interno apunta a la clave: ",key($a), "<br>";
echo "Este es el ultimo valor de la matriz a: ",end($a), "<br>";
echo "El puntero interno apunta a la clave: ",key($a), "<br>";
echo "Este es el penultimo valor de la matriz a: ",prev($a), "<br>";
echo "El puntero interno apunta a la clave: ",key($a), "<br>";
echo "Este es el mismo valor anterior: ",pos($a), "<br>";
echo "El puntero interno apunta a la clave: ",key($a), "<br>";

echo "Este el valor asoaciado al indice 4 de la matriz b: ",$b[4], "<br>";
echo "El puntero interno apunta a la clave: ",key($b), "<br>";
echo "Este es el valor siguiente al anterior: ",next($b), "<br>";
echo "El puntero interno apunta a la clave: ",key($b), "<br>";
echo "Este es el primer valor de la matriz a: ",reset($b), "<br>";
echo "El puntero interno apunta a la clave: ",key($b), "<br>";
echo "Este es el ultimo valor de la matriz a: ",end($b), "<br>";
echo "El puntero interno apunta a la clave: ",key($b), "<br>";
echo "Este es el penultimo valor de la matriz a: ",prev($b), "<br>";
echo "El puntero interno apunta a la clave: ",key($b), "<br>";
echo "Este es el mismo valor anterior: ",pos($b), "<br>";
echo "El puntero interno apunta a la clave: ",key($b), "<br>";

?>
```

[ejemplo61.php](#)

Anterior

Índice

Siguiente

## Ordenaciones de arrays

Los elementos de un array se van ordenando según se van definiendo. Por tanto, su orden *no es el mismo* que el de los *valores de sus índices*.

Las funciones PHP que ordenan los elementos de un array permiten dos opciones.

Con una de ellas es posible la ordenación de los elementos **sin modificar los valores de los índices**, mientras que la otra **sí modifica los índices**.

En el segundo de los casos la modificación puede afectar incluso al *tipo de índices* dado que **los resultados** de las ordenaciones –tanto si hemos partido de un array *escalar* como si lo hemos hecho desde uno *asociativo*– es siempre un array *escalar*.

### Ordenación por valores sin mantener índices

#### **sort(array)**

Ordena los valores del array en sentido *creciente* y lo *reindexa* asignando índice CERO al menor de los valores.

#### **rsort(array)**

Ordena la matriz en sentido *decreciente* de sus valores y la *reindexa* asignando índice CERO al mayor de estos.

### Ordenación por índices

#### **ksort(array)**

Ordena la matriz según sus índices y en sentido creciente de estos.

#### **krsort(array)**

Ordena la matriz por *índices* en sentido *decreciente* de los mismos.

### Ordenación por valores manteniendo índices

#### **asort(array)**

Ordena la matriz según sus *valores* en sentido *creciente* y *mantiene* los índices del array original.

## Ordenaciones de arrays

```
<?
$a=array(1,2,3,1,1,2,3,3,4,4,4,0,1);
$b=array("blanco","azul","blanco","blanco","azul","Blanco","Azul");
$c=array(
    "b" =>"verde",
    "c" =>"rojo",
    "e" =>"verde",
    "f" =>"Rojo",
    "g" =>"Verde",
    "a"=>"rojo",
    "d" =>"rojo",);

sort ($a);

echo "<h3>Ordenación por valores usando sort</h3>";
foreach ($a as $clave=>$valor){
echo "Clave: ",$clave," Valor: ",$valor, "<br>";
}

sort ($c);

echo "<h3>Ordenación por valores usando sort</h3>";
foreach ($c as $clave=>$valor){
echo "Clave: ",$clave," Valor: ",$valor, "<br>";
}

rsort($a);

echo "<h3>Ordenación inversa por valores usando rsort</h3>";
foreach ($a as $clave=>$valor){
echo "Clave: ",$clave," Valor: ",$valor, "<br>";
}

ksort($b);

echo "<h3>Ordenación por claves usando ksort</h3>";
foreach ($b as $clave=>$valor){
echo "Clave: ",$clave," Valor: ",$valor, "<br>";
}

krsort($b);

echo "<h3>Ordenación inversa por claves usando krsort</h3>";
foreach ($b as $clave=>$valor){
echo "Clave: ",$clave," Valor: ",$valor, "<br>";
}

asort($c);

echo "<h3>Ordenación por valores manteniendo indices </h3>";
foreach ($c as $clave=>$valor){
echo "Clave: ",$clave," Valor: ",$valor, "<br>";
}

arsort($c);

echo "<h3>Ordenación inversa por valores manteniendo indices arsort</h3>";
foreach ($c as $clave=>$valor){
echo "Clave: ",$clave," Valor: ",$valor, "<br>";
```

Ordena la matriz por valores en sentido decreciente y sigue **manteniendo** los índices originales.

## Ordenación mediante función definida por usuario

PHP permite que el usuario pueda definir **funciones** en las que establezca sus criterios particulares de ordenación. Las funciones PHP que permiten usar esta característica son las siguientes:

### **uasort(array, funcion)**

Ordena la matriz utilizando los criterios establecidos por la **función** definida por el usuario y mantiene los índices del array.

### **usort(array, funcion)**

Ordena la matriz por **valores** utilizando los criterios definidos en la **función** de usuario y modifica los índices.

### **uksort(array, funcion)**

Ordena la matriz por **claves** utilizando los criterios definidos en la **función**.

En el ejemplo hemos definido una función de comparación siguiendo el criterio de ser o no ser múltiplo de 2.

Trataremos las **funciones** en un tema aparte. La utilidad de la que hemos incluido en el ejemplo es la siguiente: Si el valor de la variable es par le asignamos un número negativo como respuesta y en caso contrario uno positivo.

De esta forma los valores del array que devuelven negativos se consideran anteriores en la ordenación a los que dan como resultado un número positivo.

```
}

echo "<h3>Ordenación por valores mediante
función de usuario manteniendo indices</h3>";

/* esta función recoge el valor de la variable $a
y aplicar el operador de comparación ternario
de forma que si el valor de la variable es impar
devuelve como valor -2 y si es par devuelve 2
el 2 y el menos 2 únicamente establecen criterios de
comparación de modo que los valores -2 serán considerados
anteriores a los valores +2 */

function micomparar (&$a) {
    return ($a%2!=0) ? -2 : 2;
}

uasort ($a, micomparar);

foreach ($a as $clave=>$valor){
echo "Clave: ",$clave," Valor: ",$valor, "<br>";
}

echo "<h3>Ordenación por clave mediante función de usuario </h3>";

uksort ($a, micomparar);

foreach ($a as $clave=>$valor){
echo "Clave: ",$clave," Valor: ",$valor, "<br>";
}

echo "<h3>Ordenación por valores mediante función de usuario </h3>";

usort ($a, micomparar);

foreach ($a as $clave=>$valor){
echo "Clave: ",$clave," Valor: ",$valor, "<br>";
}

?>
```

ejemplo62.php

Anterior



Índice



Siguiente





Ver índice

## Modificación de arrays



### Modificaciones en arrays

#### **var= range(*inf,sup*)**

Crea una **nueva matriz** (**var**) **escalar** en la que los valores de los elementos serán los números enteros (ordenados) pertenecientes al intervalo comprendido entre los valores **inf** y **sup**, incluidos estos.

Los valores **inf** y **sup** deben ser números enteros.

#### **shuffle(*array*)**

Intercambia de modo **aleatorio** los valores de un array y los reindexa.

Igual que ocurría en caso de los números aleatorios, la función **shuffle** deberá ir precedida de una **semilla** del tipo **rand**.

En el ejemplo hemos usado como **semilla** la función: **rand(time())**.

#### **var= array\_flip(*array*)**

Devuelve un **array** (**var**) que contiene como **valores** los **índices** de la matriz **array** y como **índices** los **valores** de aquella.

Como quiera que los **valores** pueden estar **repetidos** y **no es posible** que lo estén los **índices**, esta función, en caso de valores repetidos, toma cada uno de esos **valores una sola vez**, lo utiliza como índice del nuevo array y asigna como **valor** del nuevo elemento el mayor de los **índices** –del array original– de los elementos que contuvieran ese valor.

### Insertando elementos en un arrays

#### **array\_unshift(*arr, v1,v2,..*)**

Inserta al **principio de la matriz** **arr** los valores **v1, v2, etcétera** que pueden ser tantos como se deseen y deben estar separados por comas.

#### **array\_push(*array, v1,v2,..*)**

Inserta al **final de la matriz** **array** los valores **v1, v2, etcétera**, que igual que en el caso anterior, pueden ser tantos como se deseen y deben estar separados por comas.

Tanto **array\_unshift** como **array\_push** asignan a los nuevos elementos **índices numéricos**.

#### **array\_pad(*array, n, var*)**

### Modificación de arrays

```
<?
$a=array(1,2,3,1,1,2,3,3,4,4,4,0,1);
$b=array("blanco","azul","blanco","blanco","azul","Blanco","Azul");
$c=array(
    "b" =>"verde",
    "c" =>"rojo",
    "e" =>"verde",
    "f" =>"Rojo",
    "g" =>"Verde",
    "a"=>"rojo",
    "d" =>"rojo");
$c=array(
    "b" =>"verde",
    "c" =>"rojo",
    "e" =>"verde",
    "f" =>"Rojo",
    "g" =>"Verde",
    "a"=>"rojo",
    "d" =>"rojo");

echo "<h3>Crea una matriz de números enteros</h3>";

$r=range(7,11);

foreach($r as $clave=>$valor){
echo "Clave: ",$clave," Valor: ",$valor,"<br>";
}

echo "<h3>Intercambia aleatoriamente elementos en una matriz</h3>";

rand (time());
shuffle ($r);

foreach($r as $clave=>$valor){
echo "Clave: ",$clave," Valor: ",$valor,"<br>";
}

echo "<h3>Intercambia valores e indices</h3>";

$p=array_flip($a);

foreach($p as $clave=>$valor){
echo "Clave: ",$clave," Valor: ",$valor,"<br>";
}
echo "<br>";

$q=array_flip($c);

foreach($q as $clave=>$valor){
echo "Clave: ",$clave," Valor: ",$valor,"<br>";
}

echo "<h3>Inserta elementos al principio de una matriz</h3> ";
array_unshift($a,97,"Pepe",128);

foreach($a as $clave=>$valor){
echo "Clave: ",$clave," Valor: ",$valor,"<br>";
}
echo "<br>";
```

Inserta nuevos elementos en **array** y les asigna el valor contenido en **var**. Insertará tantos nuevos elementos como sea necesario para que el **array** alcance una longitud de **n** elementos.

Si el valor de **n** es **positivo** inserta los elementos **al final del array**, si fuera **negativo** los insertaría **al comienzo** del mismo.

A los nuevos elementos del array se les asignan índices numéricos.

#### array\_merge(\$a, \$b)

Crea un nuevo **array escalar** en el que se incluyen todos los **elementos** contenidos en los arrays **\$a** y **\$b**.

### Quitar elementos de un array

#### array\_shift(\$a)

La función **array\_shift** extrae el **primer elemento** del array **\$a**.

#### array\_pop(\$a)

La función **array\_pop** extrae el **último elemento** del array **\$a**.

#### array\_slice(\$a,n)

La función **array\_slice** extrae **n** elementos del array **\$a**.

Si el valor de **n** es **positivo** extraerá **todos** los elementos a partir del que ocupa la posición **n** contando desde **primero** hasta el **último** según el orden de creación de los elementos.

Si el valor de **n** es **negativo** extraerá todos los elementos a partir del **enésimo**, esta vez, contando desde el **último** hasta el **primero**.

#### array\_slice(\$a,n, m)

La función **array\_slice** con **dos parámetros** permite extraer **una parte** de los valores de una matriz siguiendo estos criterios:

Si **n** y **m** son **positivos**, extraerá **m** elementos a partir del que ocupa la posición **enésima** de **primero a último**.

Cuando **n** es **negativo** y **m** es **positivo** se extraerán **m** elementos contados a partir del **enésimo**, esta vez recorriendo el array de **último a primero**.

En el caso en que **n** tenga valor **positivo** y **m** sea **negativo** extraerá los comprendidos entre el **enésimo** contado de **primero a último** y el **emésimo** contado desde el **último hasta el primero**.

Si **n** es **negativo** y **m** es también **negativo** extraerá los caracteres

```
foreach($c as $clave=>$valor){  
echo "Clave: ",$clave," Valor: ",$valor,"<br>";  
}  
  
echo "<h3>Inserta elementos al final de una matriz</h3>";  
  
array_push($a,3.4,"Luis",69);  
  
foreach($a as $clave=>$valor){  
echo "Clave: ",$clave," Valor: ",$valor,"<br>";  
}  
echo "<br>";  
  
array_push($c,3.4,"Luis",69);  
  
foreach($c as $clave=>$valor){  
echo "Clave: ",$clave," Valor: ",$valor,"<br>";  
}  
  
echo "<h3>Inserta elementos iguales  
al principio o al final de una matriz</h3>";  
  
$wz1=array_pad($a,25,"relleno");  
  
foreach($wz1 as $clave=>$valor){  
echo "Clave: ",$clave," Valor: ",$valor,"<br>";  
}  
  
echo "<br>";  
  
$wz2=array_pad($c,-17,"relleno");  
  
foreach($wz2 as $clave=>$valor){  
echo "Clave: ",$clave," Valor: ",$valor,"<br>";  
}  
  
echo "<h3>Fusiona dos matrices</h3>";  
  
$wz3=array_merge($a,$b);  
  
foreach($wz3 as $clave=>$valor){  
echo "Clave: ",$clave," Valor: ",$valor,"<br>";  
}  
echo "<h3>Extrae el primer elemento de una matriz</h3>";  
  
array_shift ($a);  
  
foreach($a as $clave=>$valor){  
echo "Clave: ",$clave," Valor: ",$valor,"<br>";  
}  
echo "<br>";  
  
array_shift ($c);  
  
foreach($c as $clave=>$valor){  
echo "Clave: ",$clave," Valor: ",$valor,"<br>";  
}  
echo "<h3>Extrae el ultimo elemento de una matriz</h3>";  
  
array_pop($a);  
  
foreach($a as $clave=>$valor){  
echo "Clave: ",$clave," Valor: ",$valor,"<br>";  
}  
echo "<br>";  
  
array_pop ($c);  
  
foreach($c as $clave=>$valor){  
echo "Clave: ",$clave," Valor: ",$valor,"<br>";  
}  
echo "<h3>Extrae elementos de una matriz</h3>";  
  
$zz1=array_slice($a,3);  
  
foreach($zz1 as $clave=>$valor){  
echo "Clave: ",$clave," Valor: ",$valor,"<br>";  
}
```

comprendidos entre el **enésimo** contado de *último a primero* y el **emésimo** contado en el mismo sentido.

En este caso se requiere que el valor absoluto de **n** sea mayor que el de **m**.

En caso de no cumplirse esta condición devolverá un *array vacío*.

## Invertir el orden de un array

### array\_reverse(array)

Devuelve un nuevo array cuyos elementos están en orden inverso al del array original.

De esta forma el elemento que ocupaba la última posición pasa a ocupar la primera y así sucesivamente.

**¡Cuidado!**

Recuerda que *las posiciones iniciales de los elementos de un array* no tienen relación con sus índices sino con la **secuencia** en la que fueron creados.

Y otra cosa, mucho cuidado con la aplicación de todas estas funciones y con los **índices** de los arrays resultantes.

Fíjate en los ejemplos y verás que algunas estas funciones **reindexan** los resultados y los convierten en escalares aún en el caso de que originalmente fueran asociativos.

```
}

echo "<br>";

$zz2=array_slice($a,-3);

foreach($zz2 as $clave=>$valor){
echo "Clave: ",$clave," Valor: ",$valor,"<br>";
}

echo "<br>";

$zz3=array_slice($b,3,4);

foreach($zz3 as $clave=>$valor){
echo "Clave: ",$clave," Valor: ",$valor,"<br>";
}

echo "<br>";

$zz4=array_slice($b,3,-2);

foreach($zz4 as $clave=>$valor){
echo "Clave: ",$clave," Valor: ",$valor,"<br>";
}

echo "<br>";

$zz5=array_slice($b,-5,-2);

foreach($zz5 as $clave=>$valor){
echo "Clave: ",$clave," Valor: ",$valor,"<br>";
}

?>
```

**ejemplo63.php**

Anterior

Índice

Siguiente



[Ver índice](#)

## Funciones de usuario



### ¿Qué son las funciones de usuario?

De igual forma que ocurre con el navegador en el caso del HTML, **PHP** lee e interpreta las instrucciones contenidas en los scripts de forma secuencial.

Es decir, las instrucciones se van ejecutando en el mismo orden en el que aparecen en el documento original, con la **excepción** de las **funciones**.

En este caso, los bloques de instrucciones son *puestos a disposición* de PHP, pero **no se ejecutarán** hasta el momento en que sean requeridas de forma expresa.

### ¿Dónde deben insertarse?

Aunque en *versiones antiguas* de PHP era necesario **definir** la función antes de invocarla, a partir de la versión 4 **no es necesaria** esa organización secuencial.

La función pueden estar escrita dentro de cualquier script y en cualquier parte del documento, sin que tenga importancia alguna el lugar en el que se incluya la llamada a la misma.

También es posible –y bastante habitual– incluir funciones de uso frecuente en *documentos externos* de modo que pueden ser **compartidas**.

En este caso, además de *invocarla* es necesario indicar a PHP el lugar donde debe buscarla. Hablaremos de ello cuando estudiemos lo relativo a **include**.

### Definición de la función

Las funciones de usuario requieren la siguiente sintaxis:

```
function nombre(){
....  
... instrucciones ...  
....  
}
```

Es imprescindible respetar estrictamente la sintaxis que requiere de forma obligatoria los siguientes elementos:

- La palabra **function** debe estar

## Funciones de usuario

Imaginémonos, allá por el mes de junio, sentados ante una mesa tratando de hacer la *declaración de la renta*. Para ese menester seguramente nos pertecharíamos –además del impreso oficial de la declaración– con: calculadora, lápiz, goma de borrar e incluso con las disposiciones legales relativas al impuesto.

Seguramente iríamos leyendo y cumplimentando el *impreso oficial –script PHP–* partiendo de la primera página –**orden secuencial**– y continuando de forma ordenada hasta el final.

La *calculadora*, la *goma*, etcétera –**funciones**– estarían *disponibles* para ser utilizadas –**invocadas**– tantas veces como fuera preciso y, además, no serían *elementos exclusivos* de ese documento, sino que seguirían *disponibles* para otros usos –**scripts**– distintos, tales como *la planificación financiera de nuestras vacaciones*, por citar un ejemplo.

Las **funciones de usuario** son, como *la calculadora* o *la goma*, **herramientas** diseñadas para *facilitar las tareas* y susceptibles de ser usadas en una o varias situaciones –**scripts**– diferentes.

### Ejemplos de funciones de usuario

Con este primer ejemplo obtendremos *una página en blanco*. El script contiene una función pero no hay ninguna instrucción que la invoque y por lo tanto no se ejecutaría.

```
<?
function a1(){
    for($i=1;$i<=10;$i++){
        echo $i,"<br>";
    }
?>
```

**ejemplo64.php**

```
<?
a1();
?>
<!-- Hemos escrito un script con una llamada
a la función al que aún no está definida.
Tendremos que hacerlo, pero no importa
la parte del documento en la que lo hagamos
La pondremos en este nuevo script PHP //--&gt;

&lt;?
function a1(){
    for($i=1;$i&lt;=10;$i++){
        echo $i,"&lt;br&gt;";
    }
?&gt;</pre>

```

**ejemplo65.php**

En este otro ejemplo veremos las diversas situaciones que pueden plantearse respecto al **ámbito** de las variables.

```
<?
# definamos dos variables y asignémosles un valor
$a=5; $b=47;
```

escrita en minúsculas.

– El *nombre de la función*, que debe seguir criterios similares a los de los nombres de variables, aunque en este caso no se antepone el símbolo \$ ni ningún otro.

– Los paréntesis (), incluso cuando no contengan nada.

– Las *llaves de apertura {} y cierre {}* dentro de las cuales se escribirán las instrucciones correspondientes a ella.

## Ejecución de la función

Las funciones PHP **no se ejecutan** en tanto no sean **invocadas**.

Para *invocar una función* la sintaxis es la siguiente:

*nombre()*

Al ser *llamada* con esta sintaxis –desde cualquier script– se **ejecutarán** las instrucciones contenidas en ella.

## Ámbito de las variables

Resumamos lo ya comentado cuando tratamos el tema de las variables.

– Las funciones no leen valores de variables definidas fuera de su ámbito salvo que dentro de la propia función se definan de forma expresa como **globales**.

– Si una función modifica el valor de una variable **global**, el nuevo valor persiste después de abandonar la función.

– Si dentro de una función se utiliza un nombre de variable idéntico al de otra externa a ella (sin definirla global) la nueva variable se inicia con valor nulo y los eventuales valores que pudiera ir conteniendo se pierden en el momento en que se acaba su ejecución.

## Asignación de valores a variables

A las variables no globales se les pueden asignar sus valores iniciales de dos formas:

- Incluyéndolas en una línea de instrucciones *contenido* en la propia función.

- Insertando los nombres de variable y sus valores dentro del paréntesis que –de forma obligatoria– debe seguir al nombre de la función. En este caso la sintaxis sería:

**function nom (\$a=v1,\$b=v2)**

donde \$a y \$b son nombres de variables a utilizar en el ámbito de la

```
# escribamos una función al y pidámosle que imprima sus valores

function al(){
echo "Este es el valor de $a en la función al: ",$a,"<br>";
echo "Este es el valor de $b en la función al: ",$b,"<br>";
}
# hagamos una llamada a la función anterior
# no nos escribirá ningún valor porque esas variables no pertenecen
# al ámbito de la función y serán consideradas como vacías
# en el ámbito de la función
al();
# escribamos una nueva función, definamos como global $a
# y comprobemos que ahora si la hemos incluido en el ámbito
# de la función
function a2(){
global $a;
echo "Este es el valor de $a en la función a2: ",$a,"<br>";
echo "Este es el valor de $b en la función a2: ",$b,"<br>";
}
# invoquemos esta nueva función y veamos que ahora
# si se visualiza el valor de $a pero no el de $b
a2();
# creamos una nueva función y ahora modifiquemos dentro de ella
# ambas variables
function a3(){
global $a;
$a +=45;
$b -=348;
echo "Este es nuevo valor de $a en la función a3: ",$a,"<br>";
echo "Este es el valor de $b en la función a3: ",$b,"<br>";
}
# invoquemos la función a3
a3();
# comprobemos -desde fuera del ámbito de la función
# que ocurrió con los valores de las variables
echo "El valor de $a HA CAMBIADO después de ejecutar a3 es: ",$a,"<br>";
echo "El valor de $b NO HA CAMBIADO después de ejecutar a3 es: ",$b,"<br>";
# probemos que ocurre con una variable superglobal
# veremos que sin ser definida expresamente en a4
# si pertenece a su ámbito y por lo tanto visualizamos su contenido
function a4(){
print "La superglobales si están: ".$_SERVER['SERVER_NAME']."<br>";
}
# invoquemos esta nueva función
a4();
?>
```

**ejemplo66.php**

```
<?
$a=-13; $b=7482; $c="Ambrosio";
# esta es una forma alternativa de asignar valores a una variable
# del ámbito de la función
function al($a=56, $b=25){
echo "El valor de $$a en la función al: ", $a,"<br>";
echo "El valor de $$b en la función al: ", $b,"<br>";
}
al();
echo "El valor de $a después de ejecutar la función es: ",$a,"<br><br>";

# Pasando valores desde la llamada a la función #
/* Definimos una función fun1 e incluyamos dentro de su paréntesis
nombres de variables, separados por comas pero ahora sin asignarles
ningún valor */
function fun1($x,$y,$z){
    print "Valor de la variable x: ".$x."<br>";
    print "Valor de la variable y: ".$y."<br>";
    print "Valor de la variable z: ".$z."<br>";
}

# debemos hacer la llamada a la función pero ahora
# lo haremos de forma distinta.
```

función y **v1** y **v2** los valores asignados a cada una de ellas.

En este paréntesis pueden incluirse –separándolas con **comas**– cuantas parejas **var = val** sean necesarias.

• Una forma alternativa a la anterior sería la siguiente:

```
function nom ($a,$b)
```

donde habría que asignar los valores de cada una de las variables *desde la llamada a la función*, que ahora tendría esta sintaxis:

```
nombre (valor1, valor2,...);
```

en la que se escriben los *valores* separados por comas, y *encerrados entre comillas* cuando se trata de variables alfanuméricas.

Si el número de valores contenidos en la llamada fuera mayor que el número de variables definidas en la función, los excedentes serían ignorados y, si fuera inferior, se asignaría valor nulo a las variables a las que no se transfiriera ningún valor.

• También es posible incluir en la llamada a la función los *nombres de algunas variables* definidas en el ámbito externo a la función. Se haría de la siguiente forma:

```
nombre ($var1, var2,...);
```

## Pasar por referencia

Tal como hemos visto, las funciones PHP pueden recibir **valores** de variables externas y utilizar esos **valores** sin que el valor original de las mismas –salvo que se les asigne la condición de globales *dentro* de la función– sufra modificación.

Una manera de *lograr* que los *valores una variable externa* puedan ser modificados por una función, es lo que se llama en *argot informático* «*pasar variables por referencia*».

La forma de hacerlo es esta:

Hay que **anteponer** al nombre de la variable el símbolo **&** y PHP interpretará que la estamos *pasando por referencia*.

El **&** puede anteponerse tanto en *la definición de la función* como en *la llamada a la función*, tal como puedes ver en el ejemplo.

La segunda de las opciones *nos concede mayor libertad* dado que permite usar una sola función y *decidir en cada llamada* la forma de pasar los parámetros.

¡Cuidado!

```
# Vamos a incluir en la llamada
# los valores que queremos asignar a las variables de la función
# Escribiremos dentro del paréntesis de la llamada
# los valores de cada una de las tres variables
# separados por comas
# (si se trata de una cadena, pongámolas entre comillas)
# y veremos con la función recoge esos valores asignados
# en la llamada

fun1(14,"Robustiano",23.4);
/* si esta llamada contuviera más de tres valores
   los ultimos serían ignorados */
fun1(49.3,"Eustaquio",78,"Lupicio",456);
# si contuviera menos de tres valores
# PHP nos daría un mensaje de error
# advirtiendo que falta un valor
# pero nos devolvería los valores
fun1("Desiderio","Bailador");

# esos mensajes de error podríamos evitarlos
# poniendo una arroba delante de la llamada a la función
@fun1("Nuevo Desiderio","Nuevo Bailador");

# también podría utilizarse una sintaxis como esta
# en la que dejamos en blanco (entre comillas)
# el espacio correspondiente al segundo valor
# aunque si incluimos las comas.
# La variable que ocupa esa posición
# sería considerada como nula
fun1("La luna","",verde");

# también podríamos incluir en la llamada nombres de variables
# definidas en el ámbito general del script
# en este caso la función usaria esos valores

fun1($a,$b,$c);

?>
```

ejemplo67.php

```
<? $a=3; $b=2;
function a1(&$a,$b) {
    $a=pow($a,2);
    $b=pow($b,3);
echo "El cuadrado de a dentro de la función es: ",$a, "<br>";
echo "El cubo de b dentro de la función es: ",$b, "<br><br>";
}

a1($a,$b);

echo "Al salir de la función a conserva la modificación: ",$a, "<br>";
echo "Por el contrario, b no la conserva: ",$b, "<br><br>";

$c=8; $d=12;
function b1($a,$b) {
    $a=pow($a,2);
    $b=pow($b,3);
echo "El cuadrado de a dentro de la función es: ",$a, "<br>";
echo "El cubo de b dentro de la función es: ",$b, "<br><br>";
}

b1(&$c,$d);

echo "Al salir de la función c conserva la modificación: ",$c, "<br>";
echo "Por el contrario, d no la conserva: ",$d, "<br><br>";
```

ejemplo68.php

Si tratas de ejecutar **una función** en la que colocas el & en la **llamada** a la función y te aparece un mensaje como este:

«Warning: Call-time pass-by-reference has been deprecated -argument passed by value; If you would like to pass it by reference, modify the declaration of function(). If you would like to enable call-time pass-by-reference, you can set allow\_call\_time\_pass\_reference to true in your INI file».

lo que estará ocurriendo es que el **php.ini** del servidor tiene configurada en **Off** la directiva:

**allow\_call\_time\_pass\_reference**

y eso suele ocurrir con algunos **hostings** y también con la configuración por defecto de algunas versiones de PHP anteriores a la que estamos utilizando.

### Otra forma de definir funciones de usuario

Existe otra opción de definición de funciones de usuario que puede resultar de mucho interés. En este caso la **función** se define en tres bloques:

– **Definición** de la función, **llave de apertura** y **cierre** del script PHP.

– **Contenido** de la función formado exclusivamente por código HTML, que se escribiría cuando fuera invocada la función que lo contiene.

– **Cierre** de la función (**llave de cierre**) contenido en un script PHP, es decir, entre las etiquetas de apertura **<?** y cierre **?>** de PHP.

Cuando es invocada una función definida de esta forma –puedes verlo en el ejemplo– PHP se limita a **escribir** en el documento final los textos contenidos entre la etiqueta de apertura y cierre de la función.

Las funciones de esta forma son particularmente útiles para la construcción de espacios web que contienen una serie de páginas en las que se repiten las mismas estructuras.

### Ejercicio nº 26

En este ejercicio –**ejercicio26.php**– utilizaremos **una función** para construir tablas similares a las que hemos construido en el ejercicio nº 23. Pero incorporaremos una innovación respecto a aquél. Ahora la función debe permitir construir tablas de cualquier dimensión –nº de filas y/o columnas– y el número de estas habremos de incluirlo en la llamada a esa función.

## Otras funciones de usuario

```
<? function Encabezado() { ?>
<!-- Hemos abierto la función y cerrado la etiqueta PHP
todo esto es código HTML //-->
<HTML>
<HEAD>
<TITLE>Titulo de mi página</TITLE></HEAD>
<BODY BGCOLOR="#FF0000">
<!-- Esta nueva llamada a PHP
insertando la llave de cierre de la función
indicará a PHP que debe escribir todo lo
contenido entre la { y esta } //-->
<? } ?>

<? function Pie() { ?>
<HR>
</BODY>
</HTML>
<? } ?>
<!-- Utilizaremos esas dos funciones para
crear una página web. Llamamos a la función Encabezado
luego escribimos un texto y por ultimo insertamos
el Pie de página con la función Pie //-->
<? Encabezado(); ?>
Este es texto que aparecerá en el cuerpo de la página.
Está fuera de los scripts de php y será considerado
como un texto HTML. Debajo aparecerá la línea horizontal
que insertaremos mediante una nueva llamada a la función Pie

<? Pie(); ?>
```

ejemplo68a.php

Anterior

Índice

Siguiente



Ver índice

# Funciones que devuelven valores



## Funciones que devuelven valores

Las funciones PHP pueden ser llamadas a partir de un *script* y posteriormente **recoger** –en ese mismo *script*– los resultados de su ejecución.

Para conseguir este resultado debemos escribir **dentro de la función** la instrucción **return** seguida de la variable o la instrucción cuyo resultado queremos que sea devuelto al *script* desde el que ha sido **llamada** la función.

Tal como podemos ver en el ejemplo, los valores devueltos por **return** pueden ser presentados directamente en la página o recogidos por una variable.

También es posible que la función genere un **array** y que este sea devuelto a una variable que se convertiría a ese tipo de forma automática.

Otra opción de recoger los valores devueltos por **return** es invocar la función mediante una llamada del tipo:

**list(v1, v2,..)=llamada**

Las variables **v1**, **v2**, etc. recogerán los valores de los elementos del array devuelto por la función.

## Ejemplos de funciones que devuelven valores

```
<?
# asignamos valores a dos variables
Ver índice      Búsqueda rápida      ◀ Página anterior      ▶ Página siguiente ▶

# insertando return delante de la operación
function a1($a,$b){
    return pow($a,$b);
}
# incluimos en la instrucción echo una llamada
# a la función y en ella pasamos los valores
# recogidos en las variables a y b
# return conseguirá que se ejecute esa función
# y que echo recoja e imprima el resultado
echo "El valor de a elevado a b es: ",a1($a,$b), "<br>";
# esta otra función generará y devolverá un array
# con los resultados de la ejecución del bucle for
function a2($a,$b){
    for ($i=0;$i<=$b;$i++){
        $z[]=$pow($a,$i);
    }
    return $z;
}
# hacemos una llamada a la función
$p=a2($a,$b);
# leemos el array devuelto desde fuera de la función
foreach($p as $clave=>$valor){
echo "El valor de a (3) elevado a: ".$clave," es: ".$valor,"<br>";
}
echo "<br>";
# otra forma de leer el array con los resultados de la función
list($r,$s,$t)=a2($a,$b);
echo "Este es el valor recogido en la variable r :",$r,"<br>";
echo "Este es el valor recogido en la variable s :",$s,"<br>";
echo "Este es el valor recogido en la variable t :",$t,"<br>";
?>
```

ejemplo69.php

Anterior



Índice



Siguiente





Ver índice

# Funciones de fecha



## Funciones de fecha

PHP es pródigo en cuanto a posibilidades de manejo de fechas y horas. Para ello, cuenta, entre otras, con las siguientes funciones:

### **date (cadena de formato)**

Devuelve valores de **fecha y hora actuales** utilizando los parámetros que se señalan en la tabla para establecer el formato de salida de los datos. Dentro de la **misma cadena de formato** puede contener tantos parámetros como se deseen.

Como puedes ver en el ejemplo que hay al final de la tabla respeta los caracteres separadores (*espacios, dos puntos, guiones, etcétera*) que se hubieran incluido en la **cadena de formato** siempre que no coincidan con ninguno de los parámetros PHP para esta cadena.

**¡Cuidado!**

No olvides que PHP se ejecuta en el servidor que suele estar en un ordenador remoto. Por lo tanto, **fecha y hora locales** se refieren al lugar donde está instalado el servidor y que en nuestro caso servidor y cliente coinciden en un mismo equipo y coincidirán la hora del sistema con la del servidor.

Pero si alojas esta página en un *hosting australiano*, PHP nos devolvería los valores con hora y fecha de las *antípodas*.

### **date (formato, número)**

Esta función nos devuelve la fecha y hora del **tiempo Unix** (¿recuerdas aquello tan fino de Unix Epoch?) indicado en el parámetro **número**.

Recuerda también que ese número indica **segundos** contados a partir de la **0:00:00** (GMT) del día 1 de Enero de 1970.

### **gdate(cadena formato)**

Se comporta de forma idéntica a **date()** con la única diferencia de que devuelve la hora y fecha **GMT**.

Si te fijas en el ejemplo habrá una o dos horas de diferencia según accedas a esta página en verano o invierno.

### **gdate (formato, número)**

Los mismos comentarios que con el caso anterior. La única diferencia es

## Parámetros de la función **date()**

Parámetros de formato de <b>date()</b>			
Valor	Funcionalidad	Sintaxis	Resultado
<b>A</b>	AM-PM	date("A")	PM
<b>a</b>	am-pm	date("a")	pm
<b>d</b>	Día del mes en formato de 2 dígitos	date("d")	01
<b>j</b>	día del mes sin ceros a la izquierda	date("j")	1
<b>F</b>	Nombre del mes (texto completo)	date("F")	June
<b>M</b>	Nombre del mes (3 letras)	date("M")	Jun
<b>m</b>	Nº del mes (de 01 a 12) con dos dígitos	date("m")	06
<b>n</b>	Nº del mes (de 1 a 12) sin dos dígitos	date("n")	6
<b>Y</b>	Año con cuatro dígitos	date("Y")	2009
<b>y</b>	Año con dos dígitos	date("y")	09
<b>G</b>	Hora 0-23 sin ceros a la izquierda	date("G")	12
<b>H</b>	Hora 0-23 con dos dígitos	date("H")	12
<b>g</b>	Hora 1-12 sin ceros a la izquierda	date("g")	12
<b>h</b>	Hora 01-12 con dos dígitos	date("h")	12
<b>i</b>	Minutos de 00 a 59 con dos dígitos	date("i")	48
<b>s</b>	Segundos de 00 a 59 con dos dígitos	date("s")	11
<b>I</b>	día semana en texto completo	date("l")	Monday
<b>D</b>	Día de la semana (tres letras)	date("D")	Mon
<b>w</b>	día semana de 0 (domingo) a 6 (sábado)	date("w")	1
<b>z</b>	días transcurridos del año actual	date("z")	151
<b>t</b>	Número de días mes actual	date("t")	30
<b>L</b>	Año actual bisiesto (1), no bisiesto (0)	date("L")	0
<b>Z</b>	Diferencia (seg.) horaria local con GMT	date("Z")	7200
<b>U</b>	Segundos Unix Epoch	date("U")	1243853291
<b>S</b>	Sufijo ordinal inglés	date("S")	st

Un ejemplo de fecha actual:

```
<? echo "Son las ", date("h : i : s"), " y hoy es ", date("j-n-Y")?>
```

devolvería: **Son las 12 : 48 : 11 y hoy es 1-6-2009**

Este otro script devolverá la fecha y hora en la que el *tiempo Unix* era de 456.573.426 segundos.

```
<"Fué a las ", date("h:i:s",456573426), " del ", date("j-n-Y",456573426) ?>
```

devolvería: **Fué a las 11 : 57 : 06 del 20-6-1984**

## Ejemplos de la función **gdate()**

Un ejemplo de fecha actual en hora **GMT** (observa la diferencia horaria):

```
<? echo "Son las ", gdate("h : i : s"), " y hoy es ", gdate("j-n-Y")?>
```

que devuelve **hora GMT**.

#### checkdate(*mes,día,año*)

Comprueba si los valores de los parámetros **mes** están dentro del rango permitido (de 1 a 12), si el parámetro **día** es un valor válido para ese mes (considera años bisiestos) y si el valor del año pertenece al rango 0 a **32767**.

Devuelve VERDADERO si los valores corresponden a una fecha correcta y FALSO en el caso de que no ocurra así.

#### gettimeofday()

Esta función devuelve un **array asociativo** con los siguientes índices:

**sec**

El valor asociado a este índice del array recoge la **hora actual** (Unix Each) expresada en segundos

**usec**

El valor asociado a **usec** recoge la fracción en **microsegundos de hora actual** (Unix Each)

**minuteswest**

Devuelve los minutos al Oeste de Greenwich

**dsttime()**

Devuelve el tipo de corrección horaria según horarios de verano/invierno. El valor UNO corresponde a horario de verano, el valor CERO al de invierno y MENOS UNO en el caso en que sea desconocido.

#### getdate()

Devuelve un array asociativo con parámetros de la fecha actual.

Los índices de este array y sus valores son los que puedes ver en el ejemplo.

#### getdate(*número*)

Interpreta el **número** como una fecha *Unix Each* (segundos transcurridos desde el día 1 de Enero de 1970) y devuelve un array asociativo con los valores relativos a esa fecha.

Los índices de este array y sus valores son idénticos a los de **getdate()** y puedes verlos en el ejemplo.

#### microtime()

Esta función devuelve la fracción de **microsegundos** de la hora actual expresada en *tiempo Unix*.

#### time()

Esta función devuelve la hora actual en **segundos** expresada en *tiempo Unix*.

#### mktime (*hora, min, seg, mes, día, año , horario*)

Devuelve el **tiempo Unix** de la fecha

devolvería: **Fué a las 10 : 45 : 11 y hoy es 05 / 06 / 2009**

Este otro ejemplo devolverá la fecha y hora GMT coincidente con el *tiempo Unix* 456.573.426.

<?"Fué a las ", gmdate("h:i:s",**456573426**)," del ", gmdate("j-n-Y",**456573426**) ?>

devolvería: **Fué a las 09 : 57 : 06 del 20-6-1984**

## Checkdate()

Ejemplos de checkdate()			
Mes	Día	Año	Sintaxis
10	32	1987	Checkdate(10,32,1987)
10	31	1987	Checkdate(10,31,1987)
2	29	2000	Checkdate(2,29,2000)
2	29	2001	Checkdate(2,29,2001)

## gettimeofday()

Ejemplos de gettimeofday()	
Sintaxis	Devuelve
\$z= gettimeofday(); echo \$z;	Array
echo \$z[sec];	1243853291
echo \$z[usec];	281252
echo \$z[minuteswest];	-120
echo \$z[dsttime];	1

## getdate()

Ejemplos de getdate()		
Funcionalidad	Sintaxis	Devuelve
Devuelve un <b>array asociativo</b>	\$s=getdate(); echo \$s;	Array
Este índice devuelve los segundos de la hora actual	echo \$s[seconds]	11
Este índice devuelve los minutos de la hora actual	echo \$s[minutes]	48
Este índice devuelve la hora de la hora actual	echo \$s[hours]	12
Este índice devuelve el día del mes actual	echo \$s[mday]	1
Este índice devuelve el nº del día de la semana	echo \$s[wday]	1
Este índice devuelve el nº del mes	echo \$s[mon]	6
Este índice devuelve el año	echo \$s[year]	2009
Este índice devuelve nº del día en el año actual	echo \$s[yday]	151
Este índice devuelve el día de la semana	echo \$s[weekday]	Monday
Este índice devuelve el nombre del mes	echo \$s[month]	June

## getdate(*número*)

Ejemplos de getdate( <i>número</i> )		
Funcionalidad	Sintaxis	Devuelve
Devuelve un <b>array asociativo</b>	\$s=getdate( <b>125748</b> ); echo \$s;	Array

pasada como parámetro a la función. Es fundamental mantener la secuencia de los datos.

Si se omiten argumentos (sólo pueden omitirse por la derecha) tomará los de la fecha actual.

El parámetro **horario** es opcional y admite los valores **0** (horario de invierno), **1** (horario de verano).

Cuando el parámetro **día** es *cero* devuelve *el último día del mes anterior*, pero si pasamos *cero* como parámetro de **mes** nos dará un error.

Este índice devuelve los segundos de la hora actual	<code>echo \${seconds}</code>	48
Este índice devuelve los minutos de la hora actual	<code>echo \${minutes}</code>	55
Este índice devuelve la hora de la hora actual	<code>echo \${hours}</code>	11
Este índice devuelve el día del mes actual	<code>echo \${mday}</code>	2
Este índice devuelve el nº del día de la semana	<code>echo \${wday}</code>	5
Este índice devuelve el nº del mes	<code>echo \${mon}</code>	1
Este índice devuelve el año	<code>echo \${year}</code>	1970
Este índice devuelve nº del día en el año actual	<code>echo \${yday}</code>	1
Este índice devuelve el día de la semana	<code>echo \${weekday}</code>	Friday
Este índice devuelve el nombre del mes	<code>echo \${month}</code>	January

## mktime()

Ejemplos de <i>mktime()</i>									
H	Min	Sec	Mes	Día	Año	Horario	Tiempo Unix		Fecha
23	12	57	6	16	1973	0	109116777	23:12:57	16-Jun-1973
23	12	57	6	16	1973	1	109116777	22:12:57	16-Jun-1973
25	12	57	6	16	1973	1	109123977	00:12:57	17-Jun-1973
23	97	57	6	16	1973	1	109121877	23:37:57	16-Jun-1973
23	12	57	14	16	1973	1	130284777	22:12:57	16-Feb-1974
23	12	57	14	0	1973	1	128902377	22:12:57	31-Jan-1974

En los ejemplos puede verse como para valores *fueras de rango* (mes mayor de 12, minutos mayor de 60, etcétera) la función realiza la corrección correspondiente.

Anterior

Índice

Siguiente



[Ver índice](#)

## Funciones de calendario



### Los distintos calendarios

#### Días julianos

El sistema de **días julianos** fue creado por *Joseph Justus Scaliger* en **1582** y fue llamado así en recuerdo de su padre, *Julius Cesar Scaliger*.

Se trata de un sistema de *cuenta de días* que tiene su origen en el **1 de Enero del año 4713 a.C.** y que acabará el **31 de Diciembre de 3267**.

Este ciclo es producto de multiplicar tres ciclos menores: uno de 28 años denominado solar, otro de 19 años, que incorpora las fases lunares y uno de 15 años denominado de indicación.

El **día juliano** es el número resultante de contar los días transcurridos desde la fecha definida por *Scaliger* como comienzo del ciclo.

#### Calendario juliano

El emperador romano **Julio Cesar** ordenó en el año **44 a.C.** la reforma del calendario. Sustituyó el lunar adoptando, con modificaciones, uno solar de origen egipcio que data del **4000 a.C.**

Con la asesoría de *Sosigenes de Alejandría* fijó la duración de cada año en **365,25 días**, insertando un día suplementario –en febrero– cada cuatro años, **-bis sextus dies ante calendas Martii-** (dos sextos días antes de las calendas de marzo), haciendo **bisestos** a todos los años cuyo número de orden sea divisible por cuatro.

#### Calendario gregoriano

A lo largo la Edad Media se siguió manteniendo en gran parte de Europa el calendario juliano con la única adaptación de fijar la fecha de referencia de la cuenta de años en el nacimiento de Cristo.

Pero dado que la duración real del ciclo de translación de la tierra alrededor del sol es de **365,2422 días** solares medios, el calendario juliano –con años de 365,25 días– empezaba a acumular un error importante.

El Papa *Gregorio XIII* realizó la corrección en el año **1582**. Se descontaron **diez días** y es por eso que en 1582 al cuatro de octubre le

### gregoriantojd(*mes,día,año*)

Realiza la *cuenta de días julianos* correspondiente a la **fecha gregoriana** pasada en los parámetros *mes*, *día* y *año*. El script `<? echo gregoriantojd (9, 27, 1999) ?>` nos devolverá: **2451449** que es el día juliano correspondiente a la fecha gregoriana: **27 de setiembre de 1999**.

Este otro script nos devolverá el día juliano correspondiente a la fecha actual.

```
<? echo gregoriantojd (date("n"), date("j"), date("Y")) ?>
```

Así que, para tu conocimiento y efectos pertinentes, hoy día **1 -6-2009** estamos celebrando el **día juliano** número **2454984**.

### jdtogregorian(*nº de días julianos*)

Esta función devuelve en **fecha gregoriana** –con formato: *mes, día y año*– el **día juliano** pasado como parámetro.

Por si quieras ir preparando las celebraciones del **2.500.000 día juliano** debes saber que coincidirá con la fecha **8/31/2132**

### jdtojulian(*nº de días julianos*)

Con esta función puedes obtener la **fecha juliana** a partir de un valor de la **Cuenta de Días Julianos**.

Este script de PHP

```
<? echo jdtojulian(gregoriantojd (date("n"), date("j"), date("Y"))) ?>
```

nos devuelve la fecha actual según el **calendario juliano**.

Para que sepas en que día vives, hoy es **5/19/2009** según el calendario juliano.

### juliantojd(*mes,día,año*)

Convierte a **Cuenta de días Julianos** la fecha pasada (*mes, día y año*) del **calendario juliano**.

Por ejemplo `<? echo juliantojd(7,25,2001) ?>` nos devolverá **2452129** que corresponde a la cuenta de días correspondiente a la fecha **25/7/2001** expresada según el **calendario juliano**.

### jdtojewish(*nº de días julianos*)

Esta función nos devuelve la fecha (*mes, día y año*) según el **calendario judío** a partir de una fecha expresada en días julianos.

```
<? echo jdtojewish (gregoriantojd (date("n"), date("j"), date("Y"))) ?>
```

nos dará la fecha actual según el **calendario judío** que es: **10/9/5769**.

### jewishtojd(*mes,día,año*)

Nos devuelve el **día juliano** correspondiente a una determinada fecha del **calendario judío**.

Por ejemplo:

```
<? echo jdtogregorian(jewishtojd(7,21,5758)) ?>
```

nos devolverá **3/19/1998** que es la **fecha gregoriana** correspondiente al **día 21 del séptimo mes del año 5758** según el **calendario judío**.

siguió el día *quince* (viernes).

Para evitar sucesivos desfases se modificaron las condiciones de los años bisiestos que, en lo sucesivo, habrían de cumplir la condición de que su ordinal sea divisible por 4 y que no acabe en 00 con la excepción de los múltiplos de 400 que tendrían condición bisiestos.

### Calendario judío

La era judía comienza a contar desde un supuesto año de la creación del mundo, que se calcula sumando las edades de las distintas generaciones mencionadas en la Biblia.

El año judío se corresponde con el cristiano sumándole a éste 3.760 años.

El año judío es solar como el cristiano, pero sus meses son lunares, por lo que cada dos o tres años tiene que añadirse un mes bisiesto para adecuar al año solar el cómputo de los meses lunares.

### Calendario republicano francés

El Calendario Republicano fue adoptado por la Convención Francesa partiendo de las propuestas técnicas formuladas por el matemático **Lagrange**.

Es un intento de adaptar el calendario al *sistema decimal* y *eliminar referencias religiosas*.

El comienzo del año coincidía con el día 22 de Septiembre, equinoccio de otoño, y se fijó su día uno del año uno el 22 de Septiembre de 1792, día de la proclamación de la República.

Consta 12 meses de 30 días, a los que se añaden cinco días complementarios (seis en los años que son divisibles por 4 y no por 100) que son festivos y no se asignan a ningún mes.

Los meses se dividen en tres décadas de 10 días.

El calendario fue de aplicación civil en Francia y sus colonias americanas y africanas hasta **1806**.

### Día de Pascua

El Día de Pascua fue fijado en el Concilio de Nicea (año 325) como el domingo siguiente a la primera luna llena posterior al equinoccio de Primavera.

Este equinoccio se supone que siempre coincide con el 21 de marzo.

El algoritmo que usa PHP para su cálculo se basa en el que desarrolló

## jdtofrench(*nº de días julianos*)

Esta función nos devuelve la fecha según el **calendario republicano francés** correspondiente al día juliano especificado como parámetro.

```
<? echo jdtofrench (gregoriantojd (5, 7, 1796)) ?>
```

nos dará la fecha del **calendario republicano francés** que se corresponde con el **7 de Mayo de 1796** (gregoriano) que según parece es: **8/18/4**.

Sólo convierte fechas comprendidas entre los años 1 y 14 (fechas Gregorianas del 22 de septiembre de 1792 al 22 de septiembre de 1806) que se corresponden con el período de vigencia oficial de este calendario.

## frenchtojd(*mes,día,año*)

Convierte una fecha del **calendario republicano francés** en su equivalente en **días julianos**.

Por ejemplo:

```
<? echo jdgregorian(frenchtojd(6,7,8)) ?>
```

nos devolverá **2/26/1800**, que es la fecha gregoriana correspondiente al **día 7 del sexto mes del año 8** según el **calendario republicano francés**.

Igual que la función anterior sólo convierte fechas comprendidas entre los años 1 y 14 del calendario francés.

## jdmonthname(*día juliano, calendario*)

Devuelve del nombre del mes correspondiente al **día juliano** en el **calendario** señalado.

Ejemplos de jdmonthname()						
Fecha gregoriana	Gregoriano abreviado	Gregoriano	Juliano abreviado	Juliano	Judío	Republicano francés
3/1/1803	Jan	January	Dec	December	Tevet	Nivose
3/2/1803	Feb	February	Jan	January	Shevat	Pluviose
3/3/1803	Mar	March	Feb	February	Adar	Ventose
3/4/1803	Apr	April	Mar	March	Nisan	Germinal
3/5/1803	May	May	Apr	April	Iyyar	Floreal
3/6/1803	Jun	June	May	May	Sivan	Prairial
3/7/1803	Jul	July	Jun	June	Tammuz	Messidor
3/8/1803	Aug	August	Jul	July	Av	Thermidor
3/9/1803	Sep	September	Aug	August	Elul	Fructidor
3/10/1803	Oct	October	Sep	September	Tishri	Vendemiaire
3/11/1803	Nov	November	Oct	October	Heshvan	Brumaire
3/12/1803	Dec	December	Nov	November	Kislev	Frimaire
Parámetro calendario	0	1	2	3	4	5

Los parámetros señalados en la fila inferior son los correspondientes a los tipos de nombres de mes señalados en los encabezados de la tabla.

## easter\_date(*año*)

Devuelve -en tiempo Unix- la media noche del **día de Pascua** del año establecido como parámetro.

Esta función sólo es válida cuando los valores del año están comprendidos entre **1970** y **2037** (tiempo UNIX).

El script `echo date( "j-n-Y", easter_date(2006))` nos señala que la **Pascua** del año 2006 ha sido el **16-4-2006**.

## easter\_days(*año*)

Dionisio Exiguo en el año 532.

Para los años anteriores a 1753, (calendario Juliano) usa un ciclo simple de 19 años para calcular las fases de la luna.

En los años posteriores a esa fecha (Calendario Gregoriano) se añaden dos factores de corrección que tratan de hacer ese ciclo más preciso.

Devuelve el número de días del período comprendido entre el **21 de marzo** y el **día de Pascua**. Si no se especifica el año, se asume el actual.

No tiene las limitaciones de la función anterior y es aplicable a años fuera del intervalo de tiempo UNIX.

El script **easter\_days(2006)** nos señala que la Pascua del año 2006 ha sido **26 días después del 21 de Marzo**.

### ¡Cuidado!

Si piensas en la posibilidad de utilizar estas funciones en alguna aplicación concreta que pretendas publicar en un *hosting* de la red cerciórate antes de que estén habilitadas.

---

Anterior



Índice



Siguiente





[Ver índice](#)

## Clases y objetos



### Clases y objetos

Aunque PHP no es un lenguaje **orientado a objetos**, sí tiene recursos que permiten definir *clases* y construir *objetos*.

El uso de **clases y objetos** no añade ninguna funcionalidad nueva a las posibilidades de PHP. Su verdadera utilidad es la de *hacer la programación de otra manera*, con un código más legible y reutilizable.

### Las clases

Una clase no es otra cosa que una especie de *plantilla* en la que se pueden definir una serie de **variables** —que pueden contener valores predefinidos— y un conjunto de **funciones** que pueden ser *invocadas* desde cualquier parte del documento.

La sintaxis es la siguiente:

```
class nombre {
    ...
    ... definición de variables...
    ...
    ... constructores (opcional)...
    ... definición de funciones...
}
```

Vayamos por partes.

Dentro de una *clase* podemos *definir las variables* que serán utilizadas por sus *funciones internas* y a las que es posible (no es imprescindible hacerlo) *asignar* valores.

Para definir una variable es **obligatorio** anteponer **var** a su nombre y en el caso de que tratemos de asignarle un valor, bastará con poner detrás del nombre el signo **=** seguido del valor.

Ni que decir tiene que el nombre de la variable utiliza la sintaxis habitual de PHP y que si los valores asignados son tipo **cadena** tienen que ir *entre comillas*.

**var \$pepe="Jose"**

es una sintaxis válida, pero

**\$pepe="Jose"**

no lo es, le falta el **var** y si lo escribimos así nos dará un **error**.

Más adelante hablaremos de los

### El ejemplo más simple

En este ejemplo podemos ver como **utilizar una clase** para definir una *plantilla* que va a multiplicar **siete** por **ochos** y que nos va a devolver el resultado de esa operación cada vez que sea invocada.

```
<?
class Multiplica{
    var $factor1=7;
    var $factor2=8;
    function curratelo(){
        echo $this->factor1*$this->factor2;
    }
}
$objeto= new Multiplica;
$objeto->curratelo();
?>
```

[Ver ejemplo70.php](#)

### Invocando varias veces el mismo objeto

En este ejemplo puedes observar cómo al *invocar* dos veces a **\$objeto** el valor que nos devuelve es el resultado de la **última llamada** y también como se pueden crear **dos objetos** distintos.

```
<?
class Multiplica{
    var $resultado;
    function curratelo($a,$b){
        $this->resultado=$a*$b;
    }
    function imprimelo(){
        echo $this->resultado,<br>;
    }
}
$objeto= new Multiplica;
$objeto->curratelo(7,3);
$objeto->curratelo(11,4);
$objetol= new Multiplica;
$objetol->curratelo(-23,11);
$objeto->imprimelo();
$objetol->imprimelo();
?>
```

[Ver ejemplo71.php](#)

### Recogiendo resultados en un array

En este ejemplo vemos cómo se puede *invocar* reiteradamente una **función** utilizando el mismo objeto y como pueden recogerse esos resultados en un **array**.

```
<?
class Multiplica{
    var $resultado;
    var $indice=0;
```

**constructores**, pero dado su carácter *opcional* veamos antes las funciones.

Las **funciones** definidas dentro de las **clases** tienen una sintaxis **idéntica** al resto de las funciones PHP con una **salvedad** importante:

Siempre que desde una **función** – contenida en una clase – se trate de **invocar una variable** definida **en la misma clase** ha de hacerse con la siguiente sintaxis:

**\$this->variable**

Prestemos mucha atención. El **\$** va siempre **delante** de la palabra **this** y solo se escribe **una vez** y en esa posición.

El nombre de la variable (**que va siempre después** de **->** no lleva pegado el **\$**).

¡Observa los ejemplos!

## Crear y utilizar objetos

Las **clases** son solo **plantillas** y sus **funciones** no se ejecutan hasta que se les **ordene**.

Para poder utilizarlas primero debemos **crear un objeto** y luego **ejecutar sobre ese objeto** la función o funciones que deseemos.

### Creación de objetos

Para crear un **objeto** en el que se vaya a utilizar **una clase determinada** debemos usar la siguiente sintaxis:

**\$nombre = new clase**

donde **nombre** es una *palabra cualquiera* con la que identificar el **objeto** (como si se tratara de una variable) y **clase** es el **nombre** de una de las clases definidas.

¡Cuidado!

Fíjate que *detrás del nombre de la clase no hemos puesto los ()* que suelen utilizarse para invocar las **funciones**.

### Utilizando objetos

Una vez definido un **objeto** ya se le pueden aplicar las **funciones** definidas en la **clase**. La sintaxis es la siguiente:

**\$nombre->funcion()**

Donde **\$nombre** es la *misma variable* utilizada a la hora de **crear el objeto**, el **->** es obligatorio, **funcion** es el nombre de **una de las funciones** definidas en la **clase** invocada y donde los **()** sí **son obligatorios** y además – como ocurría en las demás funciones PHP – puede contener **valores**,

```
<?
    $this->resultado[$this->indice] = $a * $b;
    $this->indice++;
}

function imprimelo() {
    foreach($this->resultado as $valor) {
        echo $valor, "<br>";
    }
}

$objeto = new Multiplica;

$objeto->curratelo(7, 3);
$objeto->curratelo(11, 4);
$objeto->curratelo(-23, 11);
$objeto->imprimelo();
?>
```

[Ver ejemplo72.php](#)

## Ejemplos de uso de un constructor

```
<?
class Multiplica{
    var $factor1=7;
    var $factor2=8;
    function Multiplica(){
        print $this->factor1*$this->factor2."<br>";
    }
}

$objeto= new Multiplica;

?>
```

[Ver ejemplo73.php](#)

```
<?
class Multiplica{

    var $producto;

    function Multiplica($a=3,$b=7) {
        $this->producto=$a*$b;
        print $this->producto."<br>";
    }
}

$objeto= new Multiplica;

$objeto->Multiplica(90,47);

$objeto->Multiplica(47);

$objeto->Multiplica();

?>
```

[Ver ejemplo74.php](#)

## Un ejemplo más completo

En este ejemplo puedes ver como el **constructor** crea automáticamente los valores de la **primera**

variables, etcétera separadas por comas.

## Reiterando llamadas a objetos

Si hacemos varias *llamadas* a una función utilizando el *mismo objeto* los resultados se van *sobrescribiendo* sobre los de la llamada anterior.

Si queremos conservar *varios resultados* obtenidos de la aplicación de la *misma función* tenemos dos opciones:

Crear varios objetos

Utilizar arrays

En los ejemplos podemos ver ambos supuestos.

## Constructores

Cuando se define *una función* cuyo nombre es *idéntico* al de la *clase* que la contiene recibe el nombre de **constructor**.

Esa función -el **constructor**- se ejecuta de forma *automática* en el momento en que se define un *nuevo objeto (new)*

Según como esté *definido*, el **constructor** puede ejecutarse de distintas formas:

- Con los *valores predefinidos* en las variables de la *clase*.

- Mediante la asignación de *valores preestablecidos* en los propios parámetros de la función *constructor*.

En el ejemplo puedes ver la sintaxis de esta forma en la que se le asignan los valores **3** y **7** a las variables **\$a** y **\$b**.

En el mismo ejemplo puedes ver también la *utilidad añadida* del **constructor**.

Cuando se le *pasan* valores la función se ejecuta *sin tomar en consideración* los asignados por defecto en la función y cuando se le *pasan sólo parte* de esos valores utiliza los **valores recibidos** y para los *no asignados en la llamada* utiliza los valores del constructor.

## Clases extendidas

PHP también tiene la posibilidad de crear **clases extendidas** cuya *virtud* es poder disponer tanto de *variables y funciones propias* como de *todas las variables y funciones* de la **clase padre**.

## Sintaxis de las clases extendidas

fila de la tabla resultante.

```
<?
Class Operaciones {
    var $inicializada=32;
    var $num1;
    var $num2;
    var $suma;
    var $diferencia;
    var $producto;
    var $cociente;
    var $contador=0;
    function Operaciones ($val1=45,$val2=55) {
        $this->contador +=1;
        $c=$this->contador;
        $this->num1[$this->contador]=$val1;
        $this->num2[$c]=$val2;
        $this->suma[$c]=$val1+$val2;
        $this->diferencia[$c]=$val1-$val2;
        $this->producto[$c]=$val1*$val2;
        $this->cociente[$c]=$this->inicializada*$val1/$val2;
    }

    function imprime() {
        print "<table align=center border=1>";
        print "<td>Num 1</td><td>num2</td><td>Suma</td>";
        print "<td>Diferencia</td><td>Producto</td><td>Cociente</td><tr>";
        foreach($this->num1 as $clave=>$valor){
            print "<td align=center>".$valor."</td>";
            print "<td align=center>".$this->num2[$clave]."</td>";
            print "<td align=center>".$this->suma[$clave]."</td>";
            print "<td align=center>".$this->diferencia[$clave]."</td>";
            print "<td align=center>".$this->producto[$clave]."</td>";
            print "<td align=center>".$this->cociente[$clave]."</td><tr>";
        }
        print "</table>";
    }
}
$objeto= new Operaciones;

for ($i=1;$i<11;$i++){
    for ($j=1;$j<11;$j++){
        $objeto -> Operaciones($i,$j);
    }
}

$objeto-> imprime();

?>
```

[Ver ejemplo75.php](#)

## Un ejemplo de clase extendida

En este ejemplo puedes ver como las **clases extendidas** utilizan variables de la **clase padre**, pueden tener **constructores propios** pero **solo ejecutan su propio constructor pero no el de la clase padre**.

Para que el **constructor** de la **clase padre** sea ejecutado desde la clase extendida tiene que ser invocado expresamente.

```
<?
class Multiplica{

    var $factor1=7;
    var $factor2=8;

    function Multiplica() {
```

Para crear **clases extendidas** se requiere utilizar la siguiente sintaxis:

```
class nuev extends base {  
...  
... definición de variables....  
...  
... constructores (opcional)...  
...  
... definición de funciones...  
...  
}
```

Como habrás podido deducir *nuev* es el *nombre de la nueva clase (la extendida)*, *base* es el nombre de la clase *padre* y *extends* es la *palabra clave* que indica a PHP que se trata de una clase extendida.

PHP no permite las *herencias múltiples*. No es posible crear una *clase extendida de otra clase extendida*.

## Funciones con Clases y objetos

Existen algunas funciones que pueden resultarte útiles a la hora de trabajar con clases y objetos. Son las siguientes:

**method\_exists(obj, func)**

Comprueba si está definida la función **func** (función y método son sinónimos) en el objeto **obj**.

Devuelve un valor *booleano*. Certo (true) en el caso de que exista esa función y falso en el caso de que no exista.

**get\_class\_vars(clase)**

Crea un **array asociativo** cuyos *índices* son los *nombres de las variables* y cuyos valores coinciden con los *valores preasignados* a cada una de esas variables.

En este **array** solo se recogen las variables que han sido *inicializadas* asignándoles un *valor*.

**get\_class\_methods(clas)**

Devuelve un **array** conteniendo los valores de todos los **métodos** (funciones) definidas en la **clase clas**.

**get\_object\_var(obj)**

Devuelve **todas las variables** (y sus valores) contenidas en el **objeto obj**.

## La llamada ::

PHP permite llamar a una función definida en una **clase** sin necesidad de **crear** un objeto.

La sintaxis es la siguiente:

```
    print $this->factor1*$this->factor2."<br>";  
    print "Esto está en el constructor de la clase padre<br>";  
}  
  
class MeSeOlvido extends Multiplica{  
  
    var $divisor=5;  
    function MeSeOlvido(){  
        print $this->factor1*$this->factor2/$this->divisor."<br>";  
    }  
}  
  
$objeto= new MeSeOlvido;  
  
$objeto->Multiplica();  
?>
```

[Ver ejemplo76.php](#)

## Ejemplo de funciones PHP con clases y objetos

En este ejemplo puedes comprobar que las clases pueden escribirse en ficheros externos y posteriormente ser incluidas en un script PHP mediante la función **include**.

```
<?  
include("ejemplo75.php");  
  
$busca="imprime";  
  
if(method_exists ( $objeto, $busca )){  
    echo "Existe la función $busca <br>";  
}else{  
    echo "No existe la función $busca <br>";  
}  
  
$r=get_class_vars ("Operaciones");  
foreach ($r as $pepe=>$pepito){  
    echo "$pepe -->$pepito<br>";  
}  
  
$s=get_class_methods("Operaciones");  
foreach($s as $clave){  
    echo $clave,"<br>";  
}  
print_r(get_object_vars($objeto));  
?>
```

[Ver ejemplo77.php](#)

## Ejemplo de utilización de ::

```
<?  
class A {  
    function ejemplo() {  
        echo "Este es el resultado de la función ejemplo<br>";  
    }  
}  
  
# llamo a la función sin crear un nuevo objeto  
# usando ::  
A::ejemplo();  
  
#ahora creo el objeto $b y llamo a la función  
  
$b = new A;  
$b->ejemplo();  
?>
```

[Ver ejemplo78.php](#)

*clase :: funcion()*

donde **clase** es el nombre de una clase y **funcion()** es una función definida dentro de esa clase.

Su funcionalidad es la misma que si escribiéramos:

**\$nombre = new clase**

**\$nombre -> funcion ()**

## Métodos con el mismo nombre

Los objetos de clases extendidas pueden usar tanto las funciones contenidas en la propia clase extendida como en la clase padre. Sin embargo se puede plantear una situación (tendremos oportunidad de verla en los temas relativos a los ficheros PDF) en la que existan **dos funciones con idéntico nombre**, una en la clase extendida y otra en la clase padre.

En esa situación, siempre que se invoque el método, se ejecutará el incluido de la clase para la que se haya definido el objeto.

En el ejemplo puedes ver las diferentes opciones posibles -objetos creados sobre ambas clases- y también las, ya comentadas, de ejecución de los métodos sin creación previa de objetos mediante la sintaxis:

*nombre\_clase :: metodo()*

En este caso, si el método se invoca desde una función de la clase extendida en vez de escribir el nombre asignado a la clase padre se especifica la referencia a esta mediante la palabra: **parent**

## Bibliotecas de clases

Una de las ventajas más importantes de las clases es la posibilidad de reutilización de **rutinas** -propias o desarrolladas por otros- en nuestros scripts. Existen algunos sitios en la red en los que pueden obtener una gran cantidad de materiales de este tipo. Uno de los sitios más populares puedes encontrarlo en la dirección: [phpclasses.org](http://phpclasses.org).

Trataremos con una cierta profundidad la forma de utilización de estas clases de *terceros* en los apartados relativos a la creación de ficheros en formato PDF.

## Métodos con el mismo nombre

```
<?
# creamos una clase padre
class Padre{
    function prueba() {
        echo "Esto está en la función prueba de la clase PADRE<br>";
    }
}

# creamos una clase extendida (Hija) con una función de igual nombre
# función ejemplo
class Hija extends Padre{
    function prueba() {
        echo "Esto está en la función prueba de la clase HIJA<br>";
    }
# creamos un nuevo objeto de la clase hija que invoque
# ambas pruebas, la de la clase hija y la de la clase padre
    function ambas(){
        print "Ejecución de la función ambas<br>";
        # mediante $this-> requerimos la ejecución de método prueba
        # de la clase actual
        $this->prueba();
        # al señalar parent:: requerimos la ejecución de método prueba
        # de la clase padre
        parent::prueba();
    }
}
# creamos un objeto de la clase padre y por tanto invocará
# los métodos contenidos en esa clase sin considerar los de la
# clase extendida.
$objeto1 =new Padre();
print "<BR>Ejecutamos el método prueba().
El \'$objeto1 utiliza la clase padre<br>";
$objeto1->prueba();
# creamos un nuevo objeto sobre la clase extendida.
# Podrán invocarse métodos de la clase padre y de la extendida
# pero si coinciden los nombres de esos métodos prevalecerá
# el de la clase extendida.
$objeto2 =new Hija();
print "<BR>Ejecutamos el método prueba().
El \'$objeto2 utiliza la clase Hija<br>';
$objeto2->prueba();

print "<BR>Ejecutamos el método ambas()<br>";
# aquí invitamos otro método de la clase extendida
$objeto2->ambas();
# también podemos invocar el método de la clase que deseemos sin crear el
Print "<br>Ahora vamos a usar Padre::Prueba()<br>";
Padre::Prueba();
Print "<br>Ahora vamos a usar Hija::Prueba()<br>";
Hija::Prueba();
?>
```

Ejecutar ejemplo

Anterior

Índice

Siguiente



Ver índice

## La opción INCLUDE



### Utilización de ficheros externos

PHP dispone de funciones que permiten insertar en un documento una parte o la totalidad de los contenidos de otro. Esta opción resulta muy interesante, tanto desde el punto de vista operativo como en lo relativo a la seguridad.

Estas son algunos de los recursos que permiten ese tipo de *inclusiones*:

`include("nom. del fichero")`

El parámetro *nom. del fichero* es una cadena que contiene el *path* y el nombre del fichero cuyos contenidos pretendemos incluir.

Pueden incluirse ficheros con cualquier extensión aunque es muy habitual utilizar archivos con extensión *.inc.php*.

La primera parte (*inc*) nos permitirá identificar este tipo de ficheros mientras que la extensión *php* obligaría a que (si un usuario malicioso pretende visualizar el contenido del fichero) fuera interpretado por PHP y, como consecuencia de ello, solo devolvería el resultado sin permitir la visualización de informaciones privadas (contraseñas por ejemplo) que pudiera contener.

Este tipo de ficheros pueden contener: *texto, etiquetas HTML y funciones*.

Si no contiene funciones se podrá insertar tantas veces como se invoque y se insertará, además, todo su contenido tal como puedes ver en el ejemplo.

Si el fichero contiene funciones solo podrá ser invocado **una vez** ya que si se hiciera una segunda llamada se produciría un **error** por duplicidad en los nombres de las funciones.

Como verás en el ejemplo, es posible incluir cualquier tipo de funciones. Tanto las de la forma:

`<? function nombre { ?>`

.....  
... código HTML ...

.....  
`<? } ?>`

### Los ficheros a incluir

Este primer ejemplo de fichero a incluir contiene únicamente texto y etiquetas HTML pero no contiene ninguna llamada a ninguna función PHP, ni tampoco ningún script de este lenguaje. Le hemos guardado con dos extensiones: **ejemplo1.inc.php** y **ejemplo1.inc**.

Si pulsas sobre las opciones *Ver ejemplo* podrás comprobar que los resultados de visualización son distintos dependiendo de la extensión de cada fichero.

```
<h3><font color="#ff0000">Este sería un texto  
que se incluiría dentro de la página  
mediante las funciones  
include o require</font></h3><br>
```

[Ver ejemplo1.inc](#)

[Ver ejemplo1.inc.php](#)

Este otro fichero –que también hemos incluido con dos extensiones– contiene **funciones PHP** que pueden ser invocadas desde cualquier otro documento PHP.

Dado que las funciones contienen etiquetas HTML al abrir -mediante el navegador- el documento con extensión *.inc* serán interpretadas y se visualizarán parte de los contenidos. Cuando lleva extensión *.php* se visualizará una página en blanco ya que esas etiquetas están contenidas en funciones que no son invocadas desde el propio script.

```
<?  
function Encabezado() { ?>  
    <HTML>  
    <head>  
    <title>  
    Pruebas con la función include  
    </title>  
    </head>  
    <BODY>  
    <center><br>  
    <font size=6 face="Times" color="#0000ff">Pruebas PHP</font><br>  
    <hr width=75%>  
<? } ?>  
  
<? function Pie() { ?>  
    <center><hr width=50%>  
    <font size=2 face="Arial" color="#ff0000">Luchando con PHP</font>  
    <hr width=50%></center>  
    </body>  
    </html>  
<? } ?>  
  
<? function Calcula($a,$b) {  
    return $a*$b;  
} ?>
```

[Ver ejemplo2.inc](#)

[Ver ejemplo2.inc.php](#)

### Incluyendo ficheros

Aquí tienes el código de un documento en el que se *invocan* mediante la función **include** los dos documentos anteriores y se ejecutan sus funciones.

porciones de código HTML en cualquier script, como las del tipo:

```
function nombre {  
....  
... instrucciones PHP ...  
....  
}
```

que permiten invocar funciones repetitivas a partir de cualquier documento PHP.

Definidas las funciones en el fichero a *incluir* y colocado al comienzo de la página un *script* que contenga el **include** y la ruta de ese fichero, bastará con invocar cualquiera de las funciones, en cualquier punto del documento, para que esa llamada sea sustituida por el resultado de la ejecución de la función.

Como es lógico, solo serán visualizados en el navegador del cliente los resultados de la ejecución de las funciones que hayan sido *invocadas*.

### La función require()

Tiene la misma sintaxis que **include** y una funcionalidad similar, aunque con algunas diferencias.

Igual que ocurría con aquél, cuando un fichero es invocado por **require** esa llamada lo que hace es **sustituirse a sí misma** por el contenido del fichero especificado.

A diferencia de **include**, la etiqueta **require** lee y escribe —en el documento a partir del que es invocada— el archivo referenciado completo y **no acepta condicionales** que sí son aceptados por **include**.

### Evitar errores por duplicidad de llamadas

Tanto en el caso de usar la instrucción **include** como con **require**, si se intenta incluir **dos o más veces** un fichero que contenga funciones, se producirá un **error** (PHP no permite que dos funciones tengan el mismo nombre) y se interrumpirá la ejecución del script.

Los errores de ese tipo puede evitarse usando las funciones:

```
include_once("fichero")  
y  
require_once("fichero")
```

que a diferencia de **include** y **requiere** van a **impedir** que un mismo fichero pueda incluirse dos veces.

En los ejemplos vas a poder comprobar que no aparece el

```
<!-- empezaríamos incluyendo el fichero que contiene  
     las funciones. No escribiría nada hasta que las  
     funciones que contiene no fueran invocadas //-->  
<? include("ejemplo2.inc.php") ?>  
<!-- Insertaremos un script PHP que invoque  
     la función encabezado. Debe llevar las etiquetas  
     de apertura y cierre <? y ?> de PHP //-->  
<? Encabezado() ?>  
<!-- Insertaremos código HTML  
     según nuestra conveniencia //-->  
Aquí iría el contenido de la página<br>  
.... esto es texto HTML.....<br>  
.....<br><br><br>  
<!-- Incluimos el fichero ejemplo1.inc.php y dado que no contiene  
     ninguna función, insertará aquí todo su contenidos //-->  
<? include("ejemplo1.inc.php") ?>  
<!-- Insertaremos más código HTML -->  
.....<br>  
.....<br><br><br>  
<!-- Incluimos nuevamente el fichero ejemplo1.inc.php Puede repetirme  
     la inclusión porque no contiene funciones  
     si las contuviera habría un conflicto de duplicidad  
     porque una función no puede estar definida dos veces  
     con el mismo nombre. La instrucción include  
     como en todos los casos deberá ir dentro de un script PHP  
     y por tanto entre <? y ?> //-->  
<? include("ejemplo1.inc.php") ?>  
<!-- Ahora ejecutaremos la función PHP Calcula() pasando como  
     parámetros 7 y 9. El return de la función nos devolverá  
     el resultado que imprimiremos aquí //-->  
  
<? print "Aquí va el resultado de la multiplicación: ".Calcula(7,9); ?>  
<!-- Por último invocaremos la función Pie() -->  
<? Pie() ?>
```

[Ver ejemplo79.php](#)

### La función include\_once

```
<!-- Repetimos la inclusión de ambos ficheros  
     pero veremos que no aparece mensaje de error  
     por duplicidad de funciones y además  
     pese a hacer una doble inserción de ejemplo1  
     solo se visualiza una vez por efecto del filtro  
     establecido por include_once //-->  
<? include_once("ejemplo2.inc.php") ?>  
<? Encabezado() ?>  
Aquí iría el contenido de la página<br>  
.....<br>  
.....<br><br><br>  
<? include_once("ejemplo1.inc.php") ?>  
.....<br>  
no aparecerá nada aquí debajo<br><br><br>  
<? include_once("ejemplo1.inc.php") ?>  
<? include_once("ejemplo2.inc.php") ?>
```

[Ver ejemplo80.php](#)

El resultado de este otro ejemplo es idéntico al anterior. Sólo hemos sustituido **include\_once** por **require\_once**, que tiene una funcionalidad similar a la anterior.

mensaje de error cuando se utiliza esta función y que el texto que –anteriormente– era incluido dos veces ahora solo aparece una vez.

## Comprobando los ficheros incluidos

PHP dispone de dos funciones que permiten recoger en un *array* la lista de ficheros que se han insertado en el documento por medio de las instrucciones **require**, **require\_once** y con **include\_once** e **include**.

Tales funciones son estas:

**\$v=get\_included\_files()**

Recoge en un **array escalar** (contenido en la variable \$v) los nombres de los ficheros incluidos en el archivo en uso por **include\_once**.

**\$v=get\_required\_files()**

Igual que la función anterior recoge en un **array escalar** (contenido en la variable \$v) los nombres de los ficheros incluidos en el archivo en uso mediante **require\_once**.

## Mejorando la seguridad

Hemos hablado de la función **show\_source**, que permitía visualizar el código fuente de los scripts –no sólo locales sino de cualquier URL- si esta función no estaba desactivada en el php.ini.

Esa posibilidad de ver no sólo permite el *espionaje industrial* (ver la técnica de construcción de los scripts, etcétera) sino que permite ver también las claves y contraseñas de acceso a las bases de datos que pudieran contener los scripts.

Aparte de simplificar el trabajo la opción de incluir ficheros externos permite **guardar** la *información confidencial* fuera del **root** del servidor y usarla mediante estas llamadas.

De ese modo –*show\_source*– permitiría visualizar el nombre de ese fichero externo pero no su contenido.

Si creáramos un directorio –por ejemplo como *subdirectorio* de **c:\Apache** (fuera del root del servidor)– y le ponemos como nombre **sg**, podríamos guardar allí los ficheros *ejemplo2* y *ejemplo1*, con lo cual sus contenidos no serían visibles con *show\_source*.

En este caso la instrucción **include** ha de contener el *path* y sería la siguiente:

**include("C:\Apache\sg\fichero")**

```
<? require_once("ejemplo2.inc.php") ?>
<? Encabezado() ?>
Aqui iria el contenido de la página<br>
.....<br><br><br>
<? require_once("ejemplo1.inc.php") ?>
.....<br>
....nada de aqui en adelante ...<br><br><br>
<? require_once("ejemplo2.inc.php") ?>
<? require_once("ejemplo1.inc.php") ?>
```

[Ver ejemplo81.php](#)

## La función get\_included\_files()

```
<? include_once("ejemplo2.inc.php") ?>
<? Encabezado() ?>
<? include_once("ejemplo1.inc.php") ?>
<? Pie() ?>

Lista de fichero utilizados por include

<?
$z= get_included_files();
foreach($z as $clave=>$valor) {
echo "Clave: ",$clave," Valor: ",$valor,"<br>";
};
?>
```

[Ver ejemplo82.php](#)

### ¡Cuidado!

La manera de escribir los **path** difiere de un sistema operativo a otro. Bajo *Windows* debemos usar \ como separador, mientras que otros S.O. (Unix, Linux, etcétera) requieren utilizar /.

Para publicar tus páginas en un **hosting** no *Windows* tendrás que modificar tus scripts. Tenlo en cuenta

## Ejercicio nº 27

Crea un subdirectorio con nombre **seguridad** en **C:/ServidoresLocales/Apache/**. Despues crea un documento similar a *ejemplo2.inc.php* y guárdalo allí.

Crea otro documento –ejercicio27.php– en que incluyas algunas de aquellas funciones. Presta atención de incluir el path correcto y comprueba su funcionamiento.

## Utilizando include para gestión de fechas

Las funciones que incluye PHP para el manejo de fechas solo contemplan periodos posteriores a 1970. Para el caso de fechas anteriores a esta, existen un funciones que pueden descargarse desde: <http://phplens.com/lens/dl/adodb-time.zip>.

El archivo comprimido contiene un fichero **-adodb-time.inc.php-** con funciones PHP de que se comportan de forma idéntica a las nativas de PHP y que, además, permiten utilizar fechas anteriores a 1970 y valores negativos del tiempo Unix.

Para utilizar estas funciones bastaría con poner en el script: **include("adodb\_time.inc.php");** y sustituir las funciones de fecha de acuerdo con lo que se indica en esta tabla:

Función PHP	Función Adodb-time
<b>getdate()</b>	<b>adodb_getdate()</b>

<a href="#">date()</a>	<a href="#">adodb_date()</a>
<a href="#">gmdate()</a>	<a href="#">adodb_gmdate()</a>
<a href="#">mktime()</a>	<a href="#">adodb_mktime()</a>

En este enlace puedes comprobar los resultados de la aplicación de estas funciones que, como verás, son idénticos a los que hemos visto en el tema [Funciones de fecha](#) con la salvedad de que en este caso se admiten **fechas anteriores a 1970** y **tiempos Unix negativos**.

---

[Anterior](#)



[Índice](#)



[Siguiente](#)



## Utilización de ficheros externos

PHP dispone de funciones mediante las cuales se pueden **crear, modificar, borrar y leer** ficheros de cualquier tipo así como extraer información sobre ellos y sus contenidos.

### Abrir o crear ficheros

Para crear o modificar ficheros se utiliza la instrucción:

**\$f1=fopen(fichero,modo)**

**\$f1** es una variable que recoge el *identificador del recurso*, un valor importante (será utilizado para referirnos a este fichero en instrucciones posteriores), **fichero** es el nombre (con extensión) del fichero a abrir o crear y deberá escribirse entre comillas, y **modo**, que es una cadena que debemos poner entre comillas, el indicador del *modo de apertura* elegido.

En la tabla de la derecha hemos enumerado las opciones de ese parámetro.

Si el fichero que pretendemos abrir está en un directorio distinto al del script, debe incluirse el *path* completo delante del nombre del fichero y la cadena resultante debe ir entre comillas.

### ¡Cuidado!

Si incluimos, junto con el nombre del fichero, **un path** hay que tener muy presente que bajo **Windows** hemos de utilizar siempre el separador *anti-slash* (\).

### Cerrar ficheros

Una vez finalizado el uso de un fichero es necesario **cerrarlo**. Para ello PHP dispone de la siguiente intrucción:

**fclose(\$f1)**

Esta función - que devuelve un valor *booleano*- permite cerrar el fichero especificado en **\$f1** que, como recordarás, es el valor del **identificador de recurso** que le fue asignado automáticamente por PHP en el momento de la apertura.

### Punteros internos

Dentro de funciones para

## Función fopen()

En esta tabla tienes los parámetros *modo* de la función **fopen** de apertura de ficheros.

Valores del parámetro <i>modo</i> de la función fopen	
Valor	Funcionalidad
<b>r</b>	Abre el fichero en modo <b>lectura</b> y coloca el <b>puntero</b> al comienzo del fichero
<b>r+</b>	Abre el fichero en modo <b>lectura y escritura</b> y coloca el <b>puntero</b> al comienzo del fichero
<b>w</b>	Abre el fichero en modo <b>escritura</b> y coloca el <b>puntero</b> al comienzo del fichero, reduce su tamaño a <b>cero</b> y si el <b>fichero no existe intenta crearlo</b>
<b>w+</b>	Abre el fichero en modo <b>lectura y escritura</b> y coloca el <b>puntero</b> al comienzo del fichero, reduce su tamaño a <b>cero</b> y si el <b>fichero no existe intenta crearlo</b>
<b>a</b>	Abre el fichero en modo <b>escritura</b> y coloca el <b>puntero</b> al <b>final</b> del fichero y si <b>no existe intenta crearlo</b>
<b>a+</b>	Abre el fichero en modo <b>lectura y escritura</b> y coloca el <b>puntero</b> al <b>final</b> del fichero y si <b>no existe intenta crearlo</b>

## Un fichero de pruebas

Hemos creado un fichero llamado **domingo.txt** para poder utilizarlo en los ejemplos. Su contenido es exactamente el siguiente (incluidos los saltos de línea):

```
Esto es un ejemplo para comprobar
si funcionan o no los saltos de linea
en un documento de texto
que será leido desde php
```

## Un ejemplo de algunas funciones sobre ficheros

```
<?
/* abrimos con w+ con lo cual borramos el contenido
   y creamos el fichero en el caso de que no existiera */
$f1=fopen("sabado.txt","w+");
# escribimos en el fichero vacio
fwrite($f1,"Esta es la primera linea que escribimos en el fichero<br>");
#cerramos el fichero
fclose($f1);
echo "<H2>Este es el resultado después del primer fwrite</H2><br>";
include("sabado.txt");
# abrimos con r+ con lo cual sobreescribiremos
# en el fichero preexistente
$f1=fopen("sabado.txt","r+");
# escribimos en al principio del fichero preexistente
# ya que al abrir un fichero en este modo el puntero
# se situa al comienzo del fichero
fputs($f1,"Esto se sobreescribe");
#cerramos el fichero
fclose($f1);
echo "<H2>Este es el resultado después del segundo fwrite</H2><br>";
include("sabado.txt");
# abrimos con at con lo cual AÑADIREMOS
# al fichero preexistente ya que el modo de apertura
# situa el puntero al final del fichero
$f1=fopen("sabado.txt","at");
# escribimos al final del fichero preexistente
fputs($f1," Esto se añadirá al final<br>");
#cerramos el fichero
fclose($f1);
echo "<H2>Este es el resultado después del tercer fwrite</H2><br>";
include("sabado.txt");
echo "<h2>Leyendo con fgetc</h2><br>";
# abrimos con r+ con lo cual podemos LEER y AÑADIR
# al fichero preexistente
```

situar sus *punteros internos* y también para determinar la posición a la que **apuntan** en un momento determinado.

Se trata de las siguientes:

### **feof(\$f1)**

Es un operador *booleano* que devuelve CIERTO (1) si el puntero señala el final del fichero y FALSO si no lo hace.

### **rewind(\$f1)**

Coloca el *puntero interno* al **comienzo** del fichero indicado por el *identificador del recurso \$f1*.

### **fseek(\$f1, posición)**

Sitúa el apuntador del fichero señalado por el *identificador del recurso \$f1* en la posición (expresada en bytes) señalada por **posición**.

### **ftell(\$f1)**

Devuelve (expresada en bytes) la **posición actual** del puntero interno del fichero.

**¡Cuidado!**

Antes de utilizar funciones es necesario que el fichero que señala el *identificador de recursos* haya sido **abierto**.

## **Lectura de ficheros**

La lectura de los contenidos de un fichero puede hacerse de dos maneras: *sin apertura previa* o *con apertura previa* del mismo.

### **Lectura de ficheros sin apertura previa**

Las funciones que permiten la **lectura** de ficheros *sin haber sido abiertos* previamente son las siguientes:

### **readfile(fichero)**

Escribe directamente en el punto de inserción del *script* el *contenido completo* del fichero.

**¡Atención!**

La función *readfile* escribe el contenido del fichero *sin necesidad* de ir precedido por *echo* ni *print*.

Si se pone **echo** o se recoge en una variable, *además de su contenido* añadirá un **número** que indica el **tamaño del fichero** expresado en *bytes*.

### **\$var=file(fichero)**

Crea **\$var** –un array escalar– cuyos elementos tienen como valores los

```
--- ---, ---, ---, - , -
# leemos el primer carácter del fichero
# ya que el apuntador esta en el principio
$z=fgetc($f1);
# imprimimos el primer carácter
echo "He leido el primer carácter: ",$z,"<br>";
/* leemos el segundo carácter del fichero
ya que el apuntador se ha movido a esa posición al leer
anteriormente el primer carácter.
OBSERVA que NO HEMOS CERRADO AUN EL FICHERO */
$z=fgetc($f1);
# este es el nuevo valor de la variable $z
echo "He leido el segundo carácter: ",$z,"<br>";
/* leemos el siguiente carácter del fichero
ya que el apuntador se ha movido a una nueva posición
Recuerda que NO HEMOS CERRADO AUN EL FICHERO */
$z=fgetc($f1);
# este es ahora el valor de la variable $z
echo "He leido el tercer carácter: ",$z,"<br>";
echo "<h2>Ahora el puntero está en el tercer carácter<br>";
echo "fgets empezará a leer a partir de el</h2>";
$z=fgets($f1,200);
echo "Con fgets he leido esto: ",$z,"<br>";
#Ahora cerramos el fichero
fclose($f1);
echo "<br><h2>Al abrir el fichero de nuevo fgets
comienza desde el principio</h2><br>";
#Abrimos de nuevo el fichero
$f1=fopen("sabado.txt","r");
#Leemos su contenido
$za=fgets($f1,5000);
#Presentamos el contenido
echo $za;
#Ahora cerramos el fichero
fclose($f1);
echo "-----<br>";
?>
<h2>Aqui veremos el contenido (sin etiquetas HTML)de una pagina web</h2>
<?
# Escribimos la dirección completa de la página que puede ser
# el resultado de unir el valor de la variable $_SERVER['DOCUMENT_ROOT']
# (ruta completa del directorio raíz de servidor)
# con el nombre del directorio que la contiene y nombre del fichero
# la abrimos en modo solamente lectura
$f1=fopen($_SERVER['DOCUMENT_ROOT']."/cursophp/php53.php","r");
# Escribimos un bucle para que vaya leyendo
# cada una de las líneas hasta llegar al final del fichero
while (!feof($f1)) {
    $z = fgetss($f1, 1024);
    echo $z,"<br>";
}
#Cerramos el fichero
fclose($f1);
#Borramos el fichero antes de salir
unlink("sabado.txt");
?>
```

[Ver ejemplo83.php](#)

## **Otro ejemplo completando el anterior**

```
<?
# Abrimos el fichero en modo lectura
$f1=fopen("domingo.txt","r");
# Al anteponer echo a fpassthru
# NOS APARECERA AL FINAL EL TAMAÑO DEL FICHERO
echo fpassthru($f1), "<br>";
/* Abrimos de nuevo el fichero
RECUERDA QUE FPASSHRU LO CIERRA AUTOMATICAMENTE
DESPUES DE EJECUTAR LA INSTRUCCIÓN */
$f1=fopen("domingo.txt","r");
# Este bucle nos escribirá cada una de las
#líneas del fichero
while(!feof($f1)){
$z=fgets($f1,4000);
echo $z,"<br>";
```

contenidos de cada una de las **líneas del fichero**.

Una línea termina allí donde se haya insertado un **salto de línea** en el fichero original.

### Lectura de ficheros con apertura previa

Para la utilización de estas funciones los ficheros **han de ser abiertos** en un **modo** que permita la **lectura**.

La función que permite la lectura completa del fichero es:

### fpassthru(\$f1)

Esta función presenta algunas peculiaridades importantes:

- **Cierra** el fichero de forma automática después de la lectura. Por esa razón, si se escribe la función **fclose** a continuación de **fpassthru**, se produce un error.

- Si el resultado se recoge en una variable, o si va precedido de **echo**, además de escribir el contenido del mismo, añadirá el **número de bytes** que indican su **tamaño**.

### fgets(\$f1,long)

Extrae del fichero señalado por el **\$f1** una cadena –que **comienza** en la **posición actual** del puntero– y cuya **longitud** está limitada por el **menor** de estos tres valores:

- El valor (en bytes) indicado en **long**.

- La **distancia** (también en bytes) desde la **posición actual del puntero** hasta el **final** del fichero.

- La **distancia** que hay entre la **posición actual** del puntero y el **primer salto de línea**.

### fgetc(\$f1)

Extrae el carácter **siguiente** al señalado por la **posición actual del puntero**.

### Escribir en un fichero

Una vez abierto un fichero -en modo que permita escritura- la función PHP que nos permite **escribir** en él es la siguiente:

### fwrite(\$f1,"texto",long)

donde: **\$f1** sigue siendo el **identificador de recurso**, **texto** la cadena de texto a insertar en el fichero y **long** el número máximo de caracteres que han de insertarse.

Si la cadena de texto tiene **menor o igual longitud** que el parámetro **long** la escribirá en su totalidad, en caso

```
}

# Situamos el puntero
#al comienzo del fichero
rewind($f1);
# Reescribimos el fichero
while(!feof($f1)){
$z=fgets($f1,4000);
echo $z,"<br>";
}
# Situamos de nuevo el puntero
#al comienzo del fichero
rewind($f1);
# Situamos el puntero
#señalando el byte número 15 del fichero
fseek($f1,15);
# Releemos el fichero
#ahora la primera linea estar incompleta
#LE FALTARAN LOS 15 PRIMEROS CARACTERES
while(!feof($f1)){
$z=fgets($f1,4000);
echo $z,"<br>";
}
# volvemos el puntero al comienzo del fichero
rewind($f1);
#leemos la primera linea
$z=fgets($f1,4000);
echo $z,"<br>";
# Determinamos LA POSICION ACTUAL DEL PUNTERO
echo ftell($f1),"<br>";
# Cerramos el fichero
fclose($f1);
echo "_____<br>";

# leemos el fichero y lo presentamos
# en diferentes modalidades
$pepe=readfile("domingo.txt");
readfile("domingo.txt");
echo $pepe, "<br>";
#leemos el fichero y lo recogemos
#en un array
$z=file("domingo.txt");
#Al presentar la variable solo
#nos aparecerá la palabra array
echo $z,"<br>";
# presentamos el contenido del array
foreach($z as $linea=>$texto) {
echo "Linea: ",$linea," Texto: ",$texto,"<br>";
};
# copiamos el fichero con mensaje de resultado
if (!copy("domingo.txt", "otrodomingo.txt")) {
print("Error en el proceso de copia<br>\n");
} else{
print "<br>Fichero copiado con éxito";
}
# renombramos un fichero con mensaje de resultado
if (!rename("otrodomingo.txt", "otrolunes.txt")) {
print("Error en el proceso de renombrado<br>");
} else{
print "<br>Fichero renombrado con éxito";
}
unlink("otrolunes.txt");
echo "Ultima modificación a las: ",date("h:i:s A",
filemtime ("domingo.txt"))," del dia ",
date("j-n-Y", filemtime ("domingo.txt"));
echo "<br>El tamaño del fichero es: ", filesize("domingo.txt"), "
bytes<br>";
echo "<br>El fichero es tipo: ", filetype("domingo.txt"), " <br>";
echo "<br>Saldrá un 1 si el fichero existe: ",file_exists("domingo.txt");
?>
```

[Ver ejemplo84.php](#)

### Los valores del array devuelto por la función stat

Indice	Significado	Sintaxis	Resultado
0	Dispositivo	<? echo \$d[0] ?>	2

contrario sólo escribirá el número de caracteres indicados.

También es posible utilizar:

**fputs(\$f1,"texto",long)**

que en realidad es un alias de la función anterior.

¡Atención!

Estas funciones realizan la inserción de la cadena a partir de la posición a la que apunte el puntero en el momento de ser invocadas.

Si el fichero ya existiera y contuviera datos los nuevos datos se sobreescrivirían sobre el contenido anterior.

Para poder añadir contenidos a un fichero el puntero deberá apuntar el final del fichero preexistente y estar abierto en un modo que permita añadir contenidos.

## Borrado de ficheros

Para borrar ficheros se utiliza la siguiente instrucción:

**unlink(fichero)**

**fichero** ha de ser una cadena que contenga el nombre y la extensión del fichero y, en su caso, también el path.

## Duplicado de ficheros

La función:

**copy(fich1, fich2)**

Copia el fichero **fich1** (debe indicarse nombre y extensión) en otro fichero cuyo nombre y extensión se establecen en la cadena **fich2**.

Esta función devuelve un valor booleano indicando si la copia se ha realizado con éxito TRUE (1) o FALSE (nul) si por alguna razón no ha podido realizarse la copia.

## Renombrar ficheros

La función:

**rename(fich1, fich2)**

cambia el nombre del fichero **fich1** (hay que poner nombre y extensión) por el indicado en la cadena **fich2**.

También devuelve TRUE o FALSE.

Si tratamos de cambiar el nombre a un fichero inexistente nos dará error.

1	I node	<? echo \$d[1] ?>	0
2	Modo de protección de I node	<? echo \$d[2] ?>	33206
3	Número de enlaces	<? echo \$d[3] ?>	1
4	Id de usuario del propietario	<? echo \$d[4] ?>	0
5	Id de grupo del propietario	<? echo \$d[5] ?>	0
6	tipo de dispositivo si es un inode device *	<? echo \$d[6] ?>	2
7	Tamaño en bytes	<? echo \$d[7] ?>	126
8	Fecha del último acceso	<? echo \$d[8] ?>	1243021087
9	Fecha de la última modificación	<? echo \$d[9] ?>	1243002280
10	Fecha del último cambio	<? echo \$d[10] ?>	1243021087
11	Tamaño del bloque para el sistema I/O *	<? echo \$d[11] ?>	-1
12	Número de bloques ocupados *	<? echo \$d[12] ?>	-1

Los valores señalados con \* devuelven -1 en algunos sistemas operativos, entre ellos Windows>

## Ejemplo de un contador de visitas

```
<?
/* comprobamos si existe el fichero contador. Si existe
leemos las visitas registradas e incrementamos su valor en una unidad
Si no existe, registramos 1 como valor de número de visitas*/
if(file_exists("contador.txt")){
/* abrimos el fichero en modo lectura y escritura (r+) con lo que
el puntero se colocará al comienzo del fichero */
    $f1=fopen("contador.txt","r+");
    # leemos el contenido del fichero
    $visitas=(int)(fgets($f1,10));
    # lo aumentamos en una unidad
    $visitas++;
    # colocamos el puntero al comienzo del fichero para que
    # al guardar en nuevo valor sobreescriba el anterior
    rewind($f1);
}
else{
/*abrimos el fichero en modo lectura y escritura con (w+)
de modo que se cree automáticamente al no existir*/
    $f1=fopen("contador.txt","w+");
#asignamos uno como valor a número de visitas
    $visitas=1;
}
/* escribimos el número de visitas en el fichero. En cualquiera
de los casos el puntero estará al comienzo del fichero, por tanto
cuando existan valores serán sobreescritos */
fwrite($f1,$visitas,10);
print("Esta página ha sido visitada ".$visitas." veces");
fclose($f1);
?>
```

[Ver contador](#)

## Guardar y leer datos transferidos mediante un formulario

Aunque el modo más habitual de guardar información suele ser los servidores de bases de datos (MySQL, por ejemplo) la utilización de ficheros ofrece interesantes posibilidades de almacenamiento de información.

Este es un ejemplo muy sencillo, en el que mediante un formulario tal como el que aparece en el recuadro puede transferirse y almacenarse la información en un fichero.

```
<form name="fichero" method="post" action="escribe.php">
<input type="text" name="nombre">
<input type="text" name="apellido">
<input type="edad" name="edad">
<input type="submit" value="enviar">
</form>
```

## Funciones informativas

PHP dispone de funciones que nos facilitan información sobre ficheros. Algunas de ellas son las siguientes:

#### file\_exists(fichero)

Esta función devuelve TRUE si el fichero existe, en caso contrario devuelve FALSE.

#### filesize(fichero)

Devuelve el tamaño del fichero expresándolo en bytes. En caso de que el fichero no existiera nos dará un error.

#### filetype(fichero)

Devuelve una cadena en la que se indica el tipo del fichero. En caso de que el fichero no existiera nos dará un error.

#### filemtime(fichero)

Devuelve –en tiempo Unix– la fecha de la última modificación del fichero.

#### stat(fichero)

Devuelve un array que contiene información sobre el fichero.

En la tabla de la derecha puedes ver los contenidos asociados a cada uno de sus índices.

Recogeremos en un array, que llamaremos \$d, el resultado de la función stat aplicada sobre el fichero domingo.txt.

Para ello vamos a utilizar la siguiente sintaxis:

\$d=stat("domingo.txt")

El contenido y significado de los valores asociados a los índices de array \$d son los que tenemos en la tabla de la derecha.

### Otras funciones

Existen otras muchas funciones relacionadas con el manejo de ficheros y directorios, así como con los permisos de acceso, etcétera.

Hemos resumido únicamente las de mayor interés.

Si quieras profundizar en este tema a través de este enlace podrás acceder al capítulo del *Manual de Referencia oficial de PHP*, en el que se describen las funciones relacionadas con el manejo de ficheros.

Los datos transferidos mediante un formulario como el anterior podrían ser registrados y visualizados mediante un script como este:

```
<?
/*abrimos el fichero en modo a+ para permitir que
se cree en caso de no existir, que permita los modos lectura
y escritura y que escriba al final del fichero */
$f1=fopen("escribiente.txt","a+");
# hacemos un bucle para leer los valores transferidos
# desde el formulario y recogidos en el array $_POST
foreach($_POST as $v){
/* añadimos "\r\n" a cada valor para que se inserte
un salto de línea y que cada valor sea recogido en
una línea distinta en el fichero
Limitamos las entradas a 150 caracteres*/
fwrite($f1,$v."\r\n",150);
}
/* para comprobar que los nuevos datos han sido agregados
y visualizar el contenido integro del fichero situamos el
puntero interno al comienzo del mismo */
rewind($f1);
/* creamos un bucle que vaya leyendo todas las líneas
hasta encontrar el final del fichero */
while (!feof($f1)) {
/* vamos leyendo el contenido de cada línea del fichero
y aunque establecemos en 250 el número de caracteres
dado que los saltos del línea aparecerán antes
serán ellos los puntos de interrupción de cada lectura*/
$z = fgets($f1,250);
#presentamos el resultado de las lecturas
echo $z,"<br>";
}
# cerramos el fichero
fclose($f1);
?>
```

### Ejercicio nº 28

Crea un script -puedes llamarlo ejercicio28.php- de modo que al ejecutarlo escriba en un fichero la fecha y hora en que se produjo el acceso.

Ese mismo script deberá presentar en la página la fecha y hora de la visita anterior y si es la primera vez que se accede deberá aparecer un mensaje diciendo: «Esta es la primera vez que accedes a esta página».

Para obtener los valores de fecha y hora, deberás utilizar las funciones adodb a las que se alude en la página en la que se trata la opción **Include**.

Anterior



Índice



Siguiente





Ver índice

# Transferencia de ficheros



## Comprobación de la configuración

Antes de empezar con este tema debemos comprobar cuál es la configuración de nuestro **php.ini**.

Si por alguna circunstancia los valores no coincidieran con los que tenemos aquí a la derecha, tendríamos que abrir **php.ini** y modificar aquellas directivas.

Si te apetece, y como simple experimento, podemos cambiar el límite del tamaño máximo de transferencia poniendo:

**upload\_max\_filesize=500K**

un valor más reducido, que nos servirá para hacer una prueba muy sencilla.

## Transferencia de ficheros

La transferencia de un fichero requiere dos documentos: un **formulario** que la inicie y un **script** que la recoja.

### El formulario

Se diferencia del que hemos visto en páginas anteriores en tres aspectos. Dos de ellos se refieren a cambios dentro de la etiqueta **<form>** y el tercero es un nuevo tipo de **input** del que aún no hemos hablado.

En la etiqueta **<form>** hemos de incluir –obligatoriamente– el **method='POST'** y **ENCTYPE = "multipart/form-data"** ya que no soporta ni otro método ni otra forma de encriptación.

El cuerpo del **formulario** ha de contener un nuevo tipo de input que utiliza la siguiente sintaxis:

**<input type='file' name='nm'>**

### La tranferencia

Una vez enviado el formulario, el **fichero transferido** se guarda en un **directorio temporal** del servidor –salvo que **php.ini** especifique una cosa distinta– con un **nombre** que le es asignado de forma automática, y, además, en las **variables predefinidas \$\_FILES** (si la versión admite **superglobales**) y en **\$HTTP\_POST\_FILES** se recogerán datos relativos al contenido del fichero y a los resultados de la

## Requisitos de configuración

Cuando publicamos en un *hosting* no tenemos acceso al fichero de configuración **php.ini** pero si podemos conocer su configuración mediante el script **info.php**.

Recordemos que ese fue nuestro primer script –lo hemos creado y utilizado– para comprobar nuestra instalación y lo hemos guardado con ese nombre en el root de nuestro servidor –C:\Apache\htdocs– así que podremos acceder a él escribiendo como dirección <http://localhost/info.php>.

La abundante lista que nos muestra **info.php** contiene las siguientes líneas (las podemos localizar fácilmente porque están ordenadas alfabéticamente), que en nuestro caso –configuración por defecto– tendrán los valores que vemos en la imagen.

file_uploads	On	On
upload_max_filesize	2M	2M
upload_tmp_dir	no value	no value

Es imprescindible que **file\_uploads=On** (tal como aparece en la imagen) y resulta muy útil también conocer el valor de **upload\_max\_filesize** que por defecto –tal como ves en la imagen– es de **2Mb**. La primera directiva nos dice que PHP sí permite *subir* ficheros al servidor y la segunda nos indica el **tamaño máximo** (en Mbytes) de los ficheros que pueden ser objeto de esa transferencia.

## Formulario para transferencia de archivos

Observa que en el formulario hemos insertado una variable **oculta** (hidden) con el fin de limitar el **tamaño máximo** e **impedir** la transferencia de ficheros que **excedan** ese tamaño.

```
<HTML>
<BODY>
<FORM ENCTYPE="multipart/form-data" ACTION="ejemplo86.php" METHOD="post">
# con este input "oculto" establecemos el límite máximo
# del tamaño del fichero a transferir. En este ejemplo 1.000.000 bytes
<INPUT type="hidden" name="lim_tamano" value="1000000">
<p><b>Archivo a transferir</b><br>
<INPUT type="file" name="archivo"></p>
<p><INPUT type="submit" name="enviar" value="Aceptar"></p>
</FORM>
</BODY>
</HTML>
```

[ejemplo85.php](#)

Puedes utilizar el script invocado por la **action** de formulario anterior (es este que tienes aquí debajo) para hacer algunas comprobaciones.

Envía un fichero cualquiera –de menos de 500K– y verás que el error es **0**. Repite el envío, ahora con un fichero que sobrepase ese tamaño, y comprobarás que el error toma valor **1** dado que la directiva **upload\_max\_filesize=500K** del fichero **php.ini** habrá bloqueado la transferencia.

```
<?
/* Mediante el bucle foreach leemos el array $_FILES.
Observa la sintaxis. Escribimos como nombre del array
$_['archivo'] con lo cual foreach leerá los elementos
del array que tienen 'archivo' como primer índice
(coincide con el name que hemos puesto
en la etiqueta input=file del formulario) */

foreach ($_FILES['archivo'] as $indice=>$valor) {
```

El formato de `$_FILES` y de `$HTTP_POST_FILES` es el de array bidimensional.

El primero de sus índices es el nombre de variable usado para la transferencia (el especificado como `name='nm'` en el input type='file').

Los segundos índices –se trata de un array asociativo– son estos: `name`, `type`, `tmp_name`, `error` y `size`.

En ellos se recogen: el nombre original de fichero transferido, su formato, el nombre con el que ha sido guardado en el directorio temporal, el tipo de error de transferencia y el tamaño del archivo.

El error puede ser CERO o UNO. Si es CERO indica que la transferencia se ha realizado con éxito. En caso contrario, el valor de ese error es UNO.

## Copia del fichero

Tal como hemos visto, el fichero transferido aún no está en el servidor. Por el momento se encuentra en un directorio temporal y será preciso hacer una copia en nuestro espacio de servidor. Para este proceso puede utilizarse una función que ya hemos visto en páginas anteriores:

`copy(fich1, fich2)`

donde `fich1` sería el fichero temporal y `fich2` el del nuevo fichero.

El primero de los nombres es el valor contenido en:

`$_FILES['nm']['tmp_name']`

donde `nm` es el valor incluido como `name` en el formulario usado para la transferencia y `tmp_name` es una palabra reservada que debe escribirse exactamente con esa sintaxis.

Si la versión de PHP fuera anterior a 4.1.0 habría que utilizar `$HTTP_POST_FILES` en vez de `$_FILES`. En otros casos podrían usarse indistintamente ambos tipos de variable.

El valor `fich2` podría ser un nombre cualquiera asignado en el propio script –podemos verlo en el ejemplo– o el nombre original del fichero transferido. En este caso habría que recogerlo del elemento del array anterior cuyo segundo índice es `name`.

En la cadena `fich2` podría incluirse –recuerda que debes ponerlo entre comillas– un `path` señalando el directorio o subdirectorio donde

```

print $indice."--->".$valor."<br>";

}

/* Dependiendo del navegador que estés utilizando puede ocurrir
que varíen los valores del índice type sean distintos.
Cuando se trata de un fichero jpg, con IE devolverá image/jpeg,
mientras que con Mozilla, Firefox, Opera y Netscape
devolverá image/jpg.*/

/* repetimos el proceso anterior ahora usando la otra
variable predefinida (recuerda que esta no es superglobal)
$HTTP_POST_FILES y podremos visualizar los índices
y los valores del fichero transferido */

foreach ($HTTP_POST_FILES['archivo'] as $indice=>$valor){
    print $indice."--->".$valor."<br>";
}
?>

```

Cuando está habilitada la opción de transferencias de ficheros es conveniente –en previsión de sorpresas desagradables– tomar algunas cautelas. Una de ellas sería limitar la posibilidad de transferencia a determinados tipos de archivos –imágenes, por ejemplo– impidiendo con ello que pudieran transferirse al servidor ficheros de riesgo, tales como: ejecutables, virus, etcétera.

Cuando se establece este tipo de limitaciones, PHP comprueba los contenidos de los ficheros sin tomar en consideración la extensión de los mismos. Con ello se evita el riesgo de que puedan escondérse –cambiando la extensión– ficheros distintos de los permitidos.

Aquí tienes un ejemplo de `script` que impide la transferencia de ficheros con extensión distinta a `.jpg` o `.gif`.

```

<?
/* filtramos el tipo de archivos recibidos
de forma que solo se permitan imágenes en formato
jpg ó gif. Si el fichero transferido tuviera formato
distinto, la función exit() acabaría la ejecución del script */

if(!($_FILES['archivo']['type']=="image/jpeg" OR
      $_FILES['archivo']['type']=="image/jpg" OR
      $_FILES['archivo']['type']=="image/gif")){
    print "El formato ".$FILES['archivo']['type']."
          " no está permitido";
    exit();
} else{
    # anidamos este segundo condicional
    # para guardar en una variable
    # la extensión real del fichero
    # mas adelante la utilizaremos
    if ($_FILES['archivo']['type']=="image/jpeg" OR
        $_FILES['archivo']['type']=="image/jpg" ){
        $extension=".jpg";
    } else{
        $extension=".gif";
    }
    /* filtremos ahora el tamaño de modo que no supere
    el máximo establecido en el hidden del formulario
    (lógicamente ese valor no puede superar el valor máximo
    de la configuración de php, pero si puede ser menor)
    y también evitaremos archivos sin contenido,
    es decir con tamaño CERO */
    if($_FILES['archivo']['size']>$_POST['lim_tamano']
       OR $_FILES['archivo']['size']==0){
        print "El tamaño ".$FILES['archivo']['size']."' excede el límite";
        exit();
    }

    # asignemos un nombre a la imagen transferida
    # de modo que se guarde en el servidor
    # con un nombre distinto, asignado por nosotros
    # con ello, podemos evitar duplicidades de nombres
    # ya que si existiera un fichero con el mismo nombre
    # que el enviado por el cliente, se sobreescibiría

    $nuevo_nombre="foto_abuelita";
    # añadámosle la extensión real de fichero que teníamos

```

no incluirlo, el fichero se copiaría en el directorio desde el que se está ejecutando el script.

### Sintaxis alternativa

La opción anterior tiene una alternativa, igual de eficiente y mucho más segura. Se trata de:

**move\_uploaded\_file(fich1, fich2)**

que tiene la misma utilidad que *copy* y añade algunas ventajas tales como:

– Comprobar que el fichero ha sido transferido mediante el método POST e impedir que se *copie* en el servidor en caso de no cumplirse esa condición.

– Si la opción **safe mode=on** (configuración de php.ini en modo seguro) –por defecto está Off– comprueba, además, que la transferencia del fichero ha sido realizada por el mismo usuario que hace la petición de ejecución del script, evitando que alguien, *maliciosamente*, pueda *move* ficheros desde el directorio temporal hasta el espacio de servidor.

### ¡Cuidado..!

Al usar esta función bajo Windows conviene indicar en el parámetro **fich2** la ruta absoluta completa junto con el nombre del fichero ya que –de no hacerlo así– en algunas ocasiones la imagen no será transferida al directorio desde el que se ejecuta el script.

### register\_globals = ON

Con esta opción habilitada (ON) existe una posibilidad añadida.

En ese caso, la sintaxis podría ser más simple ya que, en el script indicado en la *action* del formulario, se crearía una variable cuyo nombre coincide con el valor de **name** en el **<INPUT type="FILE">** del formulario.

Esta variable (supongamos que su nombre es \$archivo) tiene una importante peculiaridad, ya que, además de guardar el fichero transferido, contiene (con el formato que tenemos a continuación) tres valores muy importantes:

- **\$archivo\_name** (se añade **\_name** al nombre de la variable) y recogerá el *nombre del fichero transferido*.

- **\$archivo\_size** que contiene el *tamaño del fichero*.

- **\$archivo\_type** que recoge el *tipo de fichero*.

```
# recogida en la variable nuevo_nombre  
  
$nuevo_nombre .= $extension;  
# aceptemos la transferencia siempre que el archivo tenga nombre  
if ($_FILES['archivo']['tmp_name'] != "none") {  
/* con la función copy  
pasaremos el archivo que está en el directorio temporal  
al subdirectorío que contiene el script que estamos  
ejecutando. Podríamos incluir un path y copiarlo  
a otro directorio */  
    if (copy($_FILES['archivo']['tmp_name'], $nuevo_nombre)) {  
        echo "<h2>Se ha transferido el archivo</h2>";  
    }  
} else{  
    echo "<h2>No ha podido transferirse el fichero</h2>";  
}  
?  
?
```

Usando «copy»

Usando «move\_uploaded\_file»

### Ejemplo de script para el caso register\_globals=ON

Aquí debajo incluimos el código del script que –de forma alternativa a los anteriores– podría utilizarse con esta opción de configuración.

Pese a su simplicidad te sugerimos no usarlo. Los anteriores compensan con creces su mayor complejidad sintáctica con la versatilidad –funcionan con cualquier opción de register global– y con la seguridad que añaden a las transacciones.

```
<?  
  
if ($archivo != "none" AND $archivo_size != 0  
    AND $archivo_size <= $lim_tamano) {  
  
    if (copy ($archivo, $archivo_name)) {  
        echo "<h2>Se ha transferido el archivo $archivo_name</h2>";  
        echo "<br>Su tamaño es: $archivo_size bytes<br>";  
        echo "<br>El fichero es tipo: $archivo_type <br>";  
    }  
} else{  
    echo "<h2>No ha podido transferirse el fichero</h2>";  
    echo "<h3>su tamaño no puede exceder de $lim_tamano bytes</h3>";  
}  
?  
?
```

### Ejercicio nº 29

Modifica el formulario que has creado en el ejercicio nº 9 añadiéndole la opción para que tus alumnos puedan enviarte –junto con los datos– una fotografía.

Modifica también el script asociado a aquel formulario de forma que puedas asignar un nombre a la fotografía, visualizarla –previamente copiada al directorio de trabajo– junto con los datos personales y guardar los datos personales y el nombre de la imagen en un fichero.

Anterior



Índice



Siguiente





Ver índice

## Funciones FTP



### Dos tipos de transferencias

En la página anterior hemos hablado de la manera de transferir información entre el ordenador de un usuario y un servidor web.

Aquí trataremos algo –similar a primera vista– un poco distinto. Es el caso las transferencias –en los dos sentidos– entre servidores (un servidor HTTP y un servidor FTP).

En la configuración descrita en la instalación del servidor FTP hemos establecido que ambos servidores tengan por **root** el mismo directorio, pero no es la única opción posible.

Es totalmente factible que uno de los servidores esté alojado en un *hosting* de Londres y el otro en Sydney, por poner un ejemplo de lugares distantes.

Imaginemos que todo esto es cierto.

### Transferencias FTP

Los dos primeros pasos para poder utilizar las funciones FTP han de ser: **abrir la conexión y pasar el login**. El último sería **cerrar la conexión**.

#### Abrir la conexión

**\$x=ftp\_connect (host,pt)**

Esta función –en la que **host** es una cadena con el *nombre del servidor FTP* (no te olvides de ponerlo entre comillas) y **pt** es el número del *puerto* a través del cual se efectúa la conexión FTP– **abre** una conexión con el servidor FTP.

Si se omite **pt** se asigna por defecto el valor **21** que es el *puerto* que se usa habitualmente para acceder a este tipo de servidores.

La variable **\$x** recogerá el **identificador de conexión** que será utilizado por las demás funciones.

#### «Loguearse»

Utilizaremos este término del argot informático –*horrible, verdad?*– para referirnos al hecho de que el usuario se **acredite** como autorizado en el servidor FTP.

**ftp\_login(\$x,user,pass)**

### Requisitos del sistema

El uso de estas funciones requiere que PHP tenga activada la opción **FTP (enabled)**, que en nuestro caso está activada en la configuración por defecto, tal como puedes ver a través de tu **info.php**, que tendrá un apartado idéntico al que observamos en la siguiente imagen.

**ftp**

FTP support	enabled
-------------	---------

Esto en cuanto a PHP. Pero además de esta configuración será **imprescindible** disponer de un **servidor FTP accesible y activo**. En este tipo de transferencias intervienen dos servidores: el servidor HTTP (nuestro Apache) y el servidor FTP, cuyo procedimiento de instalación y configuración hemos descrito en el apartado **Servidor de FTP**. Antes de poder utilizar las funciones que aquí describimos deberás tener instalado y activo el servidor FTP que allí se describe.

Nos conviene tener muy presente que esta versión de PHP no admite **localhost** como nombre de servidor (en la versión 4 esto no ocurría) y que por esa razón hemos de referirnos al servidor FTP mediante su dirección IP (**127.0.0.1**, en nuestro caso de servidor local) y que hemos creado diferentes usuarios (con privilegios distintos) entre ellos **admin**.

### Comprobación de *login* y conexión

```
<?
# conexión con el servidor FTP
if($x=@ftp_connect ("127.0.0.1",21)){
    echo "Conexión FTP activada<br>";
}else{
    echo "No se activo lo conexión FTP<br>";
}
# registro de usuario
if(@ftp_login($x,"super","superi")){
    echo "El login y la password han sido aceptados";
}else{
    echo "Error en login o password";
}
#desconexión
ftp_quit($x);
?>
```

**ejemplo91.php**

### Lista de contenidos del subdirectorio *cursophp*

```
<?
if($x=@ftp_connect ("127.0.0.1",21)){
    echo "Conexión FTP activada<br>";
}else{
    echo "No se activo lo conexión FTP";
}
if(@ftp_login($x,"webmaster","webmaster")){
    echo "El login y la password han sido aceptados<BR><BR>";
}else{
    echo "Error en login o password";
}
$list=ftp_nlist($x,"/cursophp");
foreach($list as $c=>$v){
    print "Indice: ".$c." Valor: ".$v."<br>";
```

Una vez *abierta* la conexión es preciso comenzar la sesión utilizando la función `ftp_login` con los siguientes parámetros:

- `$x`, que es la variable en la que se recogía el resultado de `ftp_connect`.
- `user`, que es el *nombre de usuario*.
- `pass`, que es la *password* del usuario.

Esta función devuelve un valor *booleano* que será **1** en el caso en que se inicie la sesión correctamente o **NUL** si no lo hace.

### Cerrar la conexión

Mediante la función:

`ftp_quit($x)`

se **cierra** la conexión abierta con el *identificador* indicado en la variable `$x`.

### Gestión de directorios en el servidor FTP

Una vez *logueados* y con la conexión activa ya podremos utilizar *funciones FTP* tales como:

`ftp_cdup($x)`

Nos sitúa en el directorio raíz del servidor FTP.

`ftp_pwd($x)`

Nos devuelve una cadena con el nombre del directorio actual.

`ftp_chdir($x, nuevodir)`

Cambia el acceso actual al directorio especificado por la cadena `nuevodir`, en caso de que exista.

`ftp_pwd($x)`

Indica el nombre del directorio al que estamos accediendo en este momento.

`ftp_mkdir($x, nomdir)`

Crea un **subdirectorio** –en el directorio actual– cuyo nombre es el nombre indicado en la cadena `nomdir`.

`ftp_rmdir($x, nomdir)`

Borra el *directorio* especificado en la cadena `nomdir`.

Para que un directorio pueda ser **borrado** se requiere que esté **vacío** y que esté incluido dentro del directorio *actual*.

### Información sobre los

```

    }
    print "<H1>Lista completa</H1>";
    $listacompleta=ftp_rawlist($x,"/");
    foreach($listacompleta as $c=>$v){
        print "Indice: ".$c." Valor: <H1>".$v."</H1>";
    }
    ftp_quit($x);
?>

```

ejemplo92.php

El resultado de la ejecución del script anterior podría producir una salida similar a esta:

```

Conexión FTP activada
El login y la password han sido aceptados
Indice: 0 Valor: //configuracion.txt
Indice: 1 Valor: //AreaAlumnos

```

### Lista completa

Indice: 0 Valor:	1	2a	2b	2c	3	4	5	6	7	8	9
-rw-r--r-- 1 ftp ftp 17064 Mar 04 21:11 configuracion.txt											
<b>Indice: 1 Valor:</b>											
drwxr-xr-x 1 ftp ftp 0 Mar 20 18:06 AreaAlumnos											

Tal como puedes ver en la imagen, la cadena devuelta por la función `ftp_rawlist` tiene dos resultados distintos. La primera de las cadenas comienza por **-** lo cual indica que se trata de un archivo o documento. En el segundo de los casos ese primer carácter es **d** e indica que se trata de un directorio.

Los **nueve caracteres** siguientes especifican los permisos de acceso a los ficheros y/o directorios. Se subdividen en tres bloques de igual tamaño que corresponden a los tres niveles de usuarios habituales en sistemas Unix/Linux (propietario, grupo y resto de usuarios). Para nuestros propósitos bastará con que consideremos los privilegios del primer bloque, es decir los del *propietario*.

El primero carácter de cada bloque sólo puede ser **r** ó **-**. Si se trata de un *fichero* y está marcado con **r** indica que se permite el acceso a él en modo *lectura* y si se trata de un directorio indica que está permitida la visualización de su contenido.

El segundo de los caracteres (puede ser **w** ó **-**) indica, si se trata de un fichero, que está permitida la modificación del fichero. Cuando se trata de un directorio significa que se pueden añadir o suprimir ficheros.

El tercero de los caracteres indicaría (**x** ó **-**) que el fichero –si se trata de un ejecutable– tiene permisos para ser ejecutado. Cuando se trata de un directorio, indica que pueden conocerse los atributos de los ficheros que contiene y que está permitido el acceso a él y a sus subdirectorios.

El signo **-** significa la negación del atributo en todas las opciones.

El siguiente carácter, el número **1**, está asociado con sistemas Linux/Unix e indicaría el número de **vínculos duros** contra el archivo, que es otra cosa que una forma de asignar nombres distintos a un mismo fichero.

Los dos grupos siguientes –parece que no demasiado relevantes para nuestros propósitos– ser los nombres del usuario y grupo al que pertenece.

A continuación aparece el tamaño del archivo (cero si se trata de un directorio), la fecha y hora de su creación y el nombre del archivo o directorio.

### Una miscelánea de las funciones FTP

```

<?
# Conexión con el el servidor ftp utilizando su dirección IP
if(!$x=@ftp_connect ("127.0.0.1",21)){
    echo "No se activo la conexión FTP";
    exit();
}
# Identificación de usuario webmaster (manejaremos ficheros en Apache)
if(!@ftp_login($x,"webmaster","webmaster")){
    echo "Error en login o password";
    exit();
}
/* comprobamos el nombre del directorio actual del servidor FTP
que será el root correspondiente al usuario registrado (aparecerá /) */
echo "El directorio actual es: ",ftp_pwd($x), "<br>";

```

## contenidos de los directorios del servidor FTP

### ftp\_nlist(\$x, nomdir)

Devuelve una *array escalar* con los *nombres* de los *ficheros* y *subdirectorios* contenidos en el directorio que se indica en **nomdir**.

Si se trata del *directorio actual*, el parámetro **nomdir** puede especificarse como una cadena vacía ("").

Si la información se refiere a un *subdirectorio* del *actual* bastará poner su nombre como valor del parámetro **nomdir**.

En cualquier otro caso **nomdir** contendrá la *ruta* completa.

### ftp\_rawlist(\$x, nomdir)

Igual que la función anterior, **ftp\_rawlist** también devuelve un *array escalar*, pero en este caso con información ampliada.

Este array detalla, además del nombre del fichero, el *tamaño*, el *tipo*, la *fecha de la última modificación* y los *permisos* de lectura y/o escritura.

## Transferencia de ficheros

Las transferencias de ficheros pueden realizarse en *ambos sentidos*.

### Desde el servidor FTP hasta el servidor HTTP

Mediante la función:

### ftp\_get(\$x,nloc,nrem,modo)

se transfiere un *fichero* desde un *servidor FTP* hasta un directorio del *servidor HTTP* en el que se está ejecutando PHP.

La cadena **nloc** contiene el *nombre* con el que el fichero será *copiado en el directorio actual del servidor web* y la cadena **nrem** contiene el *nombre* (incluyendo el *path*) del fichero que debe ser trasferido.

El parámetro **modo** puede contener uno de estos valores: **FTP\_ASCII** o **FTP\_BINARY**

### Desde el servidor HTTP hasta el servidor FTP

Para realizar transferencias en *sentido contrario* al anterior se utiliza la siguiente sintaxis:

### ftp\_put(\$x,nrem,nloc,modo)

Se comporta de forma idéntica a la

```
/* intentamos cambiar a un subdirectorio indicando la ruta absoluta partiendo del directorio root del usuario actual.
En caso de error (ruta incorrecta o falta de permisos de accesos nos daría un mensaje de error. Si el cambio tiene éxito nos indicaría el nombre del nuevo directorio */
if(!@ftp_chdir($x,"/cursophp/pdf")){
    print "No tienes permisos de acceso a este directorio<br>";
    print "o la ruta es incorrecta.;Comprueba los datos!<br>";
}else{
    echo "Hemos cambiado al directorio: ",ftp_pwd($x),"<br>";
}
# comprobamos el nombre del sistema operativo del servidor de FTP
echo "El S.O: del servidor FTP es: ",ftp_systype ($x),"<br>";
/* obtenemos una matriz contenido la lista de ficheros y directorios del subdirectorio "cursophp/fuentes" del del directorio actual*/
$list=ftp_nlist($x,"/cursophp/fuentes");
# escribimos la lista de ficheros contenidos en ese directorio
echo "Lista de ficheros del subdirectorio cursophp/fuentes<br>";
foreach ($list as $valor){
    echo $valor,"<br>";
}
# obtenemos una lista completa de los contenidos de ese subdirectorio
$list=ftp_rawlist($x,"/cursophp/fuentes");
# ordenamos el array que contiene la lista anterior
sort($list);
echo "Contenidos del subdirectorio cursophp/fuentes<br>";
/* extrae los elementos del array eliminando los espacios repetidos mediante la función preg_replace en la que \s+ indica uno o más espacios que serán sustituidos por uno solo (' ') */
foreach($list as $v){
    $v=preg_replace('/\s+/', ' ', $v);
    # imprimimos la cadena completa
    print "<br><br><br>".$v."<br>";
    # convertimos la cadena en un array
    # utilizando los espacios como separadores
    $extrae=explode(" ",$v);
    # leemos los elementos del array y comentamos sus valores
    foreach($extrae as $indice=>$cont){
        switch($indice){
            case 0:
                print "El elemento de indice".$indice." es: ".$cont."<br>";
                if (substr($cont,0,1)=="d"){
                    print "Es un directorio<br>";
                }elseif(substr($cont,0,1)=="-"){
                    print "Es un fichero<br>";
                }
                if (substr($cont,1,1)=="r"){
                    print "Tiene permisos de LECTURA<br>";
                }elseif(substr($cont,1,1)=="-"){
                    print "No tiene permisos de LECTURA<br>";
                }
                if (substr($cont,2,1)=="w"){
                    print "Tiene permisos de ESCRITURA<br>";
                }elseif(substr($cont,2,1)=="-"){
                    print "No tiene permisos de ESCRITURA<br>";
                }
                break;
            case 4:
                print "El tamaño de este fichero es: ".$cont." bytes<br>";
            break;
            case 8:
                print "El nombre del fichero o directorio es: ".$cont."<br>";
            break;
        }
    }
}
# regresamos al directorio cursophp
ftp_chdir($x,"/cursophp/");
/* creamos un subdirectorio (del directorio actual que es cursophp) con nombre experimento anteponiendo @# para evitar mensajes de error en caso de que ya existiera */
@ftp_mkdir($x, "experimento");
/* copiamos el fichero enol.jpg desde el directorio que se indica en el tercer parámetro (cursophp)
al directorio del servidor FTP que se indica en el segundo parámetro. Le ponemos por nombre lago_enol.jpg */
ftp_put($x, ".../cursophp/experimento/lago_enol.jpg",
        ".../cursophp/enol.jpg",FTP_BINARY);
# obtenemos el tamaño del fichero transferido
```

función anterior. La cadena **nrem** sigue siendo el *nombre* y el *path* del servidor FTP (donde vamos a copiar el fichero) y **nloc** contiene el nombre del fichero en el servidor web (origen de la transferencia).

## Modificación de ficheros en el servidor FTP

### ftp\_rename(\$x,nant,nnuevo)

Cambia el nombre del fichero **nant** por el indicado en la cadena **nnuevo**.

### ftp\_delete(\$x,fichero)

Elimina -en el servidor FTP- el fichero indicado en la cadena **fichero**.

## Información sobre ficheros del en el servidor FTP

### ftp\_size(\$x,nomfile)

Devuelve el tamaño (en bytes) del fichero que se indica en la cadena **nomfile**.

### ftp\_mdtm(\$x,nomfile)

Esta función devuelve la *fecha de la última modificación* del fichero indicado en la cadena **nomfile**. Esta fecha se indica en *tiempo Unix*.

### Ejercicio nº 31

Crea un formulario y un script mediante los cuales puedes *subir* al directorio Documentacion del servidor FTP un fichero cualquiera.

Utiliza el nombre de usuario y contraseña adecuados para poder acceder al directorio indicado y evita que quede copia del fichero en el servidor Apache.

Intenta mejorar el script para limitar el tamaño del archivo y restringir la transferencia a formatos \*.jpg ó \*.gif.

### ¡Cuidado!

Observa los path de los ejemplos. Al anteponer **..**/ estaremos indicando una ruta absoluta desde al root de servidor FTP y con **./** aludimos a un

```
echo "El tamaño de fichero transferidos es: ",  
      ftp_size($x,"../cursophp/experimento/lago_enol.jpg")," bytes<br>";  
/* escribimos la fecha de la última modificación del fichero transferido  
que coincidirá con la fecha y hora en la que se realizó la transferencia.  
Convertimos a formato de fecha convencional el tiempo UNIX que devuelve  
la función ftp_mdtm */  
print "La fecha de modificación del fichero es:";  
print date("d-m-Y H:i:s",ftp_mdtm($x,"./experimento/lago_enol.jpg"));  
# cambiamos el nombre del fichero lago_enol.jpg por lago_covadonga.jpg  
# en el servidor FTP  
@ftp_rename($x,"./experimento/lago_enol.jpg",  
           "./experimento/lago_covadonga.jpg");  
/* creamos un enlace de descarga directa del fichero haciendo una llamada  
mediante el protocolo ftp:// utilizando la sintaxis:  
ftp://usuario:contraseña@nombre del servidor  
seguidos de la ruta (en el servidor FTP) y el nombre del fichero */  
print "<BR><a href='ftp://webmaster:webmaster@localhost";  
print "/cursophp/experimento/lago_covadonga.jpg'> Descargar</a>";  
/* transferimos al directorio cursophp con nombre liborio.jpg  
un fichero procedente del directorio experimento  
cuyo nombre es lago_covadonga.jpg*/  
ftp_get($x,"../cursophp/liborio.jpg",  
       "./experimento/lago_covadonga.jpg",FTP_ASCII);  
/* comprimimos un fichero alojado en cursophp  
para transferirlo comprimido al servidor FTP */  
#empezamos leyendo el fichero y guardándolo en una cadena  
$f1=fopen("cabina.jpg","r");  
while (!feof($f1)) {  
    $cadena .= fgets($f1,1024);  
}  
fclose($f1);  
# comprimimos la cadena obtenida del fichero anterior  
$c1=gzencode($cadena,3,FORCE_GZIP);  
# guardamos la cadena comprimida en un fichero  
$f=fopen("cabina.jpg.gz","w");  
fwrite($f,$c1);  
fclose($f);  
/* al servidor el fichero comprimido. No es necesario indicar  
la ruta actual ya que ha sido creado en el mismo directorio  
en el que se está ejecutando el script */  
ftp_put($x, "./experimento/cabina.jpg.gz",  
        "cabina.jpg.gz",FTP_BINARY);  
#eliminamos el fichero comprimido del directorio cursophp  
unlink("cabina.jpg.gz");  
# cerramos la conexión con el servidor ftp  
ftp_quit($x);  
# establecemos un enlace de descarga para el fichero comprimido  
print "<BR><a href='ftp://webmaster:webmaster@localhost";  
print "/cursophp/experimento/cabina.jpg.gz'>Descarga comprimido</a>";  
?>
```

ejemplo93.php

### ¡Cuidado!

Al ejecutar el script del ejemplo anterior **por segunda vez** (sobre Linux Ubuntu) puede aparecerse un mensaje de error del tipo **overwrite permission denied**.

Este problema puede ser causado por un defecto de configuración del seidor FTP. Hemos podido comprobar que, algunas veces, por una extraña razón, aparecen en el fichero de configuración dos líneas seguidas dicendo **AllowOverwrite Off** y **AllowOverwrite On**. La configuración correcta es **AllowOverwrite On** (para permitir sobreescribir). Bastaría con eliminar la línea marcada con Off (o reemplazar el Off por On) para solventar el problema.

Anterior



Índice



Siguiente



subdirectorio del actual.

Si te aparece el error que comentamos al margen puedes editar el fichero de configuración (con GADMIN PROFTPD) y sustituir su contenido por el que puedes encontrar en un fichero llamado **config\_proftpd.txt** en el directorio **soft\_linux** del CD.



Ver índice

# Funciones de compresión



## Funciones de compresión

### zlib

Algunas de esas funciones son estas:

**\$f=gzopen(fich,mod, path)**

Abre el fichero identificado por el parámetro *fich* y lo hace en modo especificado en el parámetro modo **r** o **w** según se trate de modo *lectura* o *escritura*.

Cuando se trata del modo de escritura el parámetro **w** debe ir seguido de un número comprendido entre **cero** y **nueve** que especifica el *grado de compresión* pretendido.

El parámetro *path* es opcional y puede contener un valor lógico (cero ó uno). Cuando el valor de este parámetro es **1** permite incluir en el parámetro *fich* la **ruta del directorio o subdirectorio** que alberga el fichero que tratamos de abrir

Si se incluye una *ruta* sin especificar el valor **1** en el parámetro *path* aparecerá un error.

**gzclose(\$f)**

Cierra el fichero asociado al identificador de recurso **\$f**. Esta función devuelve TRUE en caso de éxito o FALSE si se produjera un error.

**gzeof(\$f)**

Esta función devuelve 1 (TRUE) en el caso de que el puntero apunte al final del fichero abierto e identificado mediante **\$f**.

También devuelve TRUE en caso de error.

Si el fichero estuviera abierto y el puntero apunta a una posición distinta del final del fichero devolverá FALSE.

**gzseek(\$f,desplaza)**

Desplaza -dentro del fichero identificado por **\$f**- el puntero -a partir de su posición actual- la cantidad de **bytes** indicados en el parámetro **desplaza**

**gztell(\$f)**

Devuelve la posición actual del puntero.

**gzrewind(\$f)**

## Herramientas de compresión

Existen varias herramientas para compresión de ficheros. Las más populares son las funciones de la biblioteca **bzip2** de Julian Seward que generan ficheros comprimidos que se reconocen por su extensión (bz2) y la función de **zlib** de Jean-loup Gailly y Mark Adler para leer y grabar archivos comprimidos con extensión **gz**. En esta página veremos el uso de la segunda de las opciones. Empezaremos comprobando en *info.php* que la opción está activada. Deberemos ver algo como esto:

### zlib

ZLib Support	enabled
Stream Wrapper support	compress.zlib://
Stream Filter support	zlib.inflate,zlib.deflate
Compiled Version	1.2.3
Linked Version	1.2.3

En la versión de PHP que estamos utilizando esta opción se activa por defecto y **no es necesario modificar ningún parámetro** en *php.ini*.

## Ejemplo de compresión y lectura de un fichero

En este ejemplo trataremos de utilizar las funciones de compresión comentadas al margen. Si observas las formas de apertura de los ficheros verás que son similares a las utilizadas para la gestión de ficheros. Los modos de apertura para **escritura** son: "**w0**" a "**w9**" siendo los valores de **cero a nueve** los indicadores de los niveles de compresión. Para **lectura** debe usarse el modo "**r**" sin indicar ningún nivel de compresión.

```
<?
# asignamos un nombre al fichero con extensión "gz"
$fichero ='prueba.gz';
# abrimos el fichero en modo escritura (w)
# con el nivel máximo de compresión (9)
$f=gzopen($fichero,"w9",0);
$cadena="Este es el primer bloque de texto que hemos
         introducido en el fichero comprimido. ";
$cadena .= "Añadimos este segundo bloque";
echo "<i>Esta es la cadena inicial:</i> ".$cadena."<br>";
# escribimos (comprimida) la cadena en el fichero
gzwrite($f,$cadena);
# cerramos el fichero
gzclose($f);
#abrimos el fichero en modo lectura
$f=gzopen($fichero,"r");
echo "<i>Estos son los tres primeros caracteres de la cadena:</i> ";
# escribimos los tres primeros caracteres, el puntero (por defecto)
# apunta al comienzo de la cadena
echo gzread($f, 3)."<br>";
# desplazamos el puntero hasta el carácter nº 8
gzseek($f,8);
echo "<i>Estos son los seis caracteres siguientes al octavo:</i> ";
# escribimos seis caracteres a partir del octavo
echo gzread($f, 6)."<br>";
echo "<i>Ahora el puntero está en:</i> ";
# buscamos la posición actual de puntero
echo gztell($f)."<br>";
# movemos el puntero hasta el comienzo del fichero
gzrewind($f);
echo "<i>Estos son los diez primeros caracteres de la cadena:</i> ";
```

Coloca el puntero al comienzo del fichero

#### **gzread(\$f, longitud)**

Devuelve una cadena -después de descomprimida- de longitud igual a la indicada en el parámetro **longitud**. La lectura comienza en la **posición actual del puntero** y acaba cuando la longitud de la cadena leída y descomprimida sea igual al valor del parámetro **longitud** o cuando se haya alcanzado el *final del fichero*.

#### **gzpassthru (\$f)**

Esta función **escribe** en la salida (no necesita la función echo) el contenido del fichero desde la posición actual del puntero hasta el final del fichero. Como es lógico, si estuviera *precedida* de **gzrewind** escribiría el fichero completo.

#### **¡Cuidado!**

La función **gzpassthru** cierra automáticamente el fichero después de escribir su contenido. Si pones **gzclose** después de esta función te dará **error** y si quieras seguir utilizando el fichero tendrás que volver a abrirlo con la función **gzopen**.

#### **gzwrite(\$f, cadena, long)**

Esta función **escribe** en el fichero comprimido que se identifica por **\$f** la cadena contenida en el parámetro **cadena**.

Opcionalmente puede llevar el tercer parámetro (**longitud**) en cuyo caso solo escribirá los primeros **longitud bytes** de la cadena. Si el parámetro **longitud** existe y es *mayor* que la longitud de la cadena, insertará la *cadena completa*.

#### **gzputs(\$f, cadena, long)**

Esta función es idéntica a **gzwrite**.

#### **readgzfile(\$fichero,path)**

Esta función **abre de forma automática** el fichero indicado como parámetro **fichero**, además lo **lee** y lo **escribe de forma automática** sin necesidad de usar echo ni ninguna otra función de salida.

Si el fichero no está en el mismo directorio que el *script* -además de incluir la ruta en la cadena **fichero**- es necesario añadir el segundo parámetro **-path-** con valor **1**.

```
"-----"
echo gzread($f, 10)."  
# volvemos el puntero al comienzo del fichero
    gzrewind($f);
    echo "<i>Escribimos el fichero completo:</i> ";
# con gzpasthrw escribimos el fichero completo
# el puntero está al principio porque allí lo ha situado gzrewind
# no necesitamos utilizar "echo" ni "print" ya que gzpassthru
# escribe directamente el contenido del fichero
    gzpassthru($f);
# tenemos que volver a abrir el fichero ya que gzpassthru
# se encargó de cerrarlo después de leerlo
$f=gzopen($fichero,"r");
    echo "<br><i>Aquí estará todo el fichero:</i> ";
    gzpassthru ($f);
# la función readgzfile abre el fichero, imprime su contenido y lo cierra
echo "<br><i>Aqui se imprime la cadena
completa usando readgzfile</i>: <br>";
    readgzfile($fichero);
/* con gzfile también se abre el fichero,
pero ahora el contenido no se presenta
directamente. Es recogido en un array.
Para visualizarlo debemos imprimir
el primer elemento del array. */
$z=gzfile($fichero);
    echo "<br><i>Este es el primer elemento (0)
del array generado por gzfile</i>: ".$z[0];
# gzip cierra el fichero.
# No podemos poner gzclose porque nos daría error
?>
```

comprime1.php

## **Utilizando un directorio distinto**

El ejemplo anterior está desarrollado para el supuesto que el *script* y el *fichero comprimido* estén en el mismo directorio.

Si quieras utilizar estas funciones utilizando ficheros alojados en un directorio distinto, solo tendrás que recordar que algunas funciones deben incluir el parámetro complementario **1**. Estos son las modificaciones que deberías efectuar:

La variable que recoge el nombre del fichero debe incluir el **path**, por ejemplo: **\$fichero ='/subdirectorio/prueba.gz'**

La función **gzopen** debe incluir el tercer parámetro (path) con valor **1**, por ejemplo: **\$f=gzopen(\$fichero,"r",1);**

También las funciones **gzfile** y **readgzfile** -que abren automáticamente el fichero- deberán incluir ese valor **1** como parámetro añadido. Por ejemplo: **readgzfile(\$fichero,1)** ó **\$z=gzfile(\$fichero,1)**

## **Elección del grado óptimo de compresión**

Puede parecer -a primera vista- que la condición óptima de compresión sería elegir el **nivel 9** y eso es cierto si tomamos únicamente en consideración el tamaño final del fichero comprimido. Sin embargo no existe una relación lineal entre reducción de tamaño/nivel de compresión. Sin que pueda considerarse ninguna referencia exacta -la compresión alcanzable depende del contenido del fichero y en consecuencia no puede establecerse una relación funcional- puede comprobarse experimentalmente que -aparentemente- a partir del grado 2 la reducción de tamaño del fichero es mínima y que cuando se aumenta el grado de compresión a niveles máximos (tratándose de ficheros de un cierto tamaño) el tiempo de ejecución aumenta sustancialmente como consecuencia de la reiteración de la ejecución de los algoritmos de compresión.

## **Compresión de cadenas**

En este ejemplo utilizamos *las tres funciones de compresión de cadenas* así como las opciones de descompresión y lectura de cada una de ellas.

<?

## Comprimiendo cadenas

Las funciones anteriores permiten la creación, lectura y modificación de ficheros comprimidos

Sin embargo, existen otras funciones PHP que permiten comprimir *cadenas*. Aquí tienes algunas de ellas.

### gzcompress(cadena, nivel)

Esta función devuelve una **cadena comprimida** a partir de una *original* especificada en el parámetro **cadena**. El nivel de compresión (valores entre 0 y 9) se especifica en el parámetro **nivel**.

Las cadenas resultantes de esta función pueden *descomprimirse* aplicando la función **gzuncompress** que te comento más abajo.

### gzdeflate(cadena, nivel)

Se comporta de forma idéntica a la función anterior. La única salvedad parece ser que utiliza un *algoritmo de compresión distinto*.

Las cadenas resultantes de esta función también pueden *descomprimirse* aplicando la función **gzdeflate**.

### gzencode(cad, niv, opc)

Esta función devuelve la cadena **cad** comprimida con el nivel especificado en **niv** y permite dos opciones de compresión: **FORCE\_GZIP** ó **FORCE\_DEFLATE** que se pueden especificar como tercer parámetro (**opc**) sin encerrar entre comillas.

El *valor por defecto* (cuando no se especifica el parámetro opción) es **FORCE\_GZIP**

## Descomprimiendo cadenas

### gzuncompress(cadena)

Con esta función se obtiene una cadena -descomprimida- a partir de la cadena comprimida indicada en el parámetro **cadena**- siempre que esta hubiera sido comprimida usando la función **gzcompress**

### gzinflate(cadena)

Funciona igual que la anterior. La única diferencia es que esta *descomprime* las cadenas que han sido comprimidas con **gzdeflate**

## Funciones para buferización de salidas

### ob\_start()

Esta función activa la *buferización* de

```
# creamos una cadena de ejemplo
$cadena="Esta es la cadena a comprimir. Intentaremos que sea larga
porque parece que si la hacemos muy corta en vez de reducirse
su tamaño parece que aumenta. Y como sigue siendo enormemente
grande la cadena comprimida intentaremos hacerla aun mayor
a ver que pasa ";
# comprimimos con la función gzcompress
$c=gzcompress($cadena,9);
echo "<br>".$c;
# descomprimimos con la función gzcompress
$dc=gzuncompress($c);
echo "<br>".$dc."<br>";
# ahora utilizamos la función gzencode
$c1=gzencode($cadena,9,FORCE_GZIP);
echo "<br>".$c1."<br>";
/* el resultado lo guardamos en un fichero con extensión gz
pero abierto en modo "normal", es decir escribiendo
dentro del fichero la cadena "tal cual" fue devuelta
por gzencode*/
$f=fopen("pepe.gz","w");
fwrite($f,$c1);
fclose($f);
# abrimos el fichero anterior utilizando las funciones
# de lectura de fichero comprimidos
$f=ezopen("pepe.gz","r");
readgzfile("pepe.gz");
gzclose($f);
# borramos el fichero una vez leido
unlink("pepe.gz");
# otra opción de compresión de cadenas utilizando la función
# gzdeflate
$c2= gzdeflate($cadena,9);
echo "<br><BR>".$c2;
# con la función gzinflate podemos descomprimir la cadena
# comprimida generada por gzdeflate
$dc2=gzinflate($c2);
echo "<br>".$dc2;
?>
```

comprime2.php

## Economizando espacio en el servidor

Las opciones de compresión pueden permitirnos un cierto ahorro de espacio de servidor. Las páginas HTML podrían almacenarse comprimidas y ser *llamadas* a través de un script de descompresión que permita visualizarlas. En este ejemplo se efectúa la compresión de una página web (una de las páginas de estos materiales guardada en formato HTML) cuyo tamaño original es de 29.015 bytes. El fichero comprimido resultante ocupa 9.886 bytes. Como verás, el fichero se reduce a poco más del 30% del original.

PHP a partir de su activación.

Dicho de otra forma, *impide que las salidas generadas por el script se envíen al cliente y por tanto no serán visualizadas en el navegador*. A partir del momento de activar esa **buferización**, todas las salidas generadas se almacenan en una variable específica llamada: **ob\_get\_contents()**

#### ob\_end\_clean()

Esta función **desactiva** la **buferización** iniciada por **ob\_start** y borra los contenidos de la variable **ob\_get\_contents()**

#### ob\_clean()

Esta función vacía el buffer de salida pero sin *desactivar la bufferización*. Las salidas posteriores a esta función seguirían siendo recogidas en el *buffer*.

### Cabeceras para transferir información comprimida

Cuando un servidor recibe una petición de una página web el navegador del cliente siempre envía información sobre su disposición a aceptar diferentes tipos de contenidos.

Una de las informaciones que suelen recibirse con la *peticIÓN del navegador* se refiere a su capacidad para aceptar *contenidos codificados* y esto suelen hacerlo mediante el envío de una cabecera que diría algo similar a esto: **Accept-Encoding:gzip,deflate** o **Accept-Encoding: gzip**.

Esta posibilidad es una característica común a todas las versiones modernas de los navegadores (es posible que algunas versiones antiguas no acepten esta codificación) y bastará con se incluya en la respuesta (el documento transferido por el servidor al cliente) la cabecera: **Header('Content-Encoding: gzip')** para que el navegador sepa que la *información llegará codificada* y que debe activar -de forma automática- sus *mecanismos de traducción de ese tipo de contenidos*.

### Algunas limitaciones

En todos estos ejemplos hemos dado por supuesto que los navegadores de los clientes **aceptan la codificación gzip**, pero es evidente que si eso no ocurriera la página se visualizaría erróneamente.

#### Ejercicio nº 30

```
<?
# Creamos una variable "vacía"
$cadena="";
# Abrimos el fichero en modo lectura (r)
$f1=fopen("prueba.html","r");
/* hacemos un bucle para leer el fichero
hasta encontrar el final (feof) y vamos recogiendo
el contenido en la variable */
while (!feof($f1)) {
    $cadena .= fgets($f1, 1024);
}
/*comprimimos la cadena con gzencode
con lo cual la propia función añade los "encabezados"
de formato gzip*/
$c1=gzencode($cadena,3,FORCE_GZIP);
/* abrimos un nuevo fichero modo escritura (w)
con "fopen", es decir como un fichero normal con extensión GZ */
$fp=fopen("prueba.html.gz","w");
/* escribimos la cadena "tal cual"
en este fichero */
fwrite($fp,$c1);
# cerramos el fichero comprimido
fclose($fp);
echo "La compresión ha terminado";
?>
```

comprime3.php

El fichero comprimido mediante el script anterior no puede ser visualizado directamente. Requiere ser descomprimido antes de ser enviado al navegador. Y eso podría hacerse mediante un script como este:

```
<?
#abrimos el fichero comprimido con "gzopen"
$f=gzopen("prueba.html.gz","r");
/* leemos el contenido completo
en forma transparente ya que readgzfile descomprime
la salida*/
readgzfile("prueba.html.gz");
# cerramos el fichero
gzclose($f);
?>
```

Visualizar fichero comprimido

### Economizando tiempo de transferencia

No solo se puede economizar espacio en el servidor. También es posible enviar *comprimidas* -desde el servidor hasta el cliente- las páginas web.

En ese caso, será el propio navegador el que interprete la información comprimida y la presente de una manera transparente. Lo que habremos ahorrado habrá sido tiempo de transferencia pero, igual que ocurría en el comentario anterior, esa reducción del volumen de información a transferir afecta únicamente al contenido de la página y no a otros elementos que puede incluir, tales como imágenes, etcétera.

Este es un ejemplo de un script que **comprime** una página web y **la envía comprimida** al cliente.

```
<?
/* activamos la bufferización de la salida
para que no se presenten los resultados del script
directamente en la página
¡¡Cuidado con no dejar líneas en blanco delante del script
ya que vamos a insertar luego Headers!! */
ob_start();
# abrimos y leemos el fichero html
$f1=fopen("prueba.html","r");
fpassthru($f1);
# recogemos el contenido del buffer en la variable $cadena
$cadena = ob_get_contents();
```

Escribe un script que guarde en un fichero comprimido la imagen **aviones4.jpg** que tienes en el directorio *images*.

Crea un segundo script que permita visualizarla en el navegador y evalúa la economía de espacio que aporta la compresión de este tipo de archivos.

```
# comprimimos la cadea con gzencode
# para que incluya los encabezados "gzip"
$cd=gzencode($cadena,3,FORCE_GZIP);
# desactivamos la "buferización"
# y borramos el contenido del buffer
ob_end_clean();
# insertamos la cabeceras
# indicando el tipo de contenido y el tamaño
    Header('Content-Encoding: gzip');
    Header('Content-Length: ' . strlen($cd));
# presentamos el contenido (cadena comprimida) que será
# "traducido" automáticamente por el navegador
echo $cd;
?>
```

[Ejecutar script](#)

El ejemplo anterior *comprimía* el contenido del fichero antes de enviarlo. En este que incluimos a continuación partimos del supuesto de que la página **ya está comprimida en el servidor**. Por tanto, tendremos que leer el fichero comprimido y enviarlo, de igual forma, al cliente.

```
<?
ob_start();
/* En este caso abrimos el fichero con "gzopen"
ya que se trata de un fichero comprimido
# todo lo demás es idéntico al ejemplo anterior*/
$f1=gzopen("prueba.html.gz","r");
gzpassthru($f1);
$cadena = ob_get_contents();
$cd=gzencode($cadena,3,FORCE_GZIP);
ob_end_clean();
    Header('Content-Encoding: gzip');
    Header('Content-Length: ' . strlen($cd));
echo $cd;
?>
```

[Ejecutar script](#)

[Anterior](#)

[Índice](#)

[Siguiente](#)



Ver índice

## Mensajes de correo



### Correo electrónico

PHP dispone de una función que permite el envío de todo tipo de *mensajes de correo electrónico* desde una página web.

Son escasos los **hosting** que tienen activada esta función.

El motivo aducido, por muchos de ellos, para esta restricción no es otra que el riesgo de *abusos* mediante los mensajes conocidos como **spam**.

Como siempre, si tu interés es *publicar* y tienes necesidad de este servicio procura consultar sobre su disponibilidad en el *hosting* que elijas.

### Sintaxis

La forma más simple de la función de correo es esta:

**mail(dest,asunto,mensaje)**

Donde **dest** es *la dirección del destinatario*, **asunto** es el texto que aparecerá como *Asunto* en el encabezado que recibirá el destinatario y **mensaje** el texto que aparecerá en el *cuerpo del mensaje*.

No te olvides de escribir *entre comillas* esos tres parámetros de la función.

### Mensaje con cabeceras MIME

Una forma más completa es la siguiente:

**mail(dest,asunto,mens,cabez)**

Como puedes ver, en este caso añadimos un nuevo parámetro a la función (**cabez**) que debe estar **entre comillas** y que puede contener, separando con comas, lo siguiente:

**From: Nombre <e-mail>**

El texto que escribas en el parámetro **Nombre** (**¡cuidado, no lleva comillas!**) será el *nombre del remitente*, que aparecerá en el campo **De:** cuando el destinatario reciba el mensaje.

Donde dice **e-mail** es *obligado* escribir *una dirección de correo* que debe ir contenida **entre < >**, tal

### Requisitos del sistema

La utilización de las funciones de correo electrónico requiere disponer de un servidor de correo electrónico instalado y activo y la modificación de la configuración inicial del fichero *php.ini*.

Si no tienes instalado este servidor puedes hacerlo ahora. En la página [Servidor de correo](#) tienes detallados ambos procesos (instalación y modificación de *php.ini*).

### El mensaje más simple

Aquí tienes un ejemplo, el más simple, del uso de esta función:

```
<?
# insertamos la función mail (rojo) dentro de un condicional
# para tener una confirmación de envío y/o aviso de error
# es algo muy habitual para conocer el éxito de la ejecución
# de funciones
if(mail("juan@mispruebas.com", "Mi primer mensaje","Este es el texto")){
    print "mensaje enviado";
} else{
    print "el mensaje no ha podido enviarse";
}
?>
```

**ejemplo94.php**

### Un ejemplo más completo

En el ejemplo siguiente insertaremos algunos nuevos elementos comentados al margen. Observa con mucha atención la *estructura* del código. Fíjate que hemos insertado en **líneas diferentes** cada uno de los conceptos: *From*, *Reply-To*, etcétera y que **no hemos dejado ningún espacio** al comienzo de esas líneas.

No es por *capricho* ni por afán *estético*. Si insertáramos algún carácter delante de esas líneas se plantearían problemas en la estructura del mensaje y si no incluyéramos un *salto de línea* para cada uno de los conceptos también tendríamos ese mismo problema.

La sintaxis MIME es muy estricta en este sentido. ¡Tengamos mucho cuidado en esto!

Hay otra posibilidad sintáctica –como alternativa a los saltos de línea– ya conocida. Podríamos escribir todo en una sola línea sustituyendo los saltos de línea que ves aquí por **\n**, de forma que el script tuviera un aspecto similar al siguiente:

**mail("juan@localhost","Asunto","Contenido","\nReply-To: ... \nCc:.....\nBcc: ...")**

donde, como ves, los **\n** sustituyen a los saltos de línea.

```
<?
mail("juan@mispruebas.com","Varios destinatarios","Cuerpo del mensaje",
"From: CursoPHP <juan@mispruebas.com>
Reply-To: andres@mispruebas.com
Cc: perico@mispruebas.com, andres@mispruebas.com
Bcc: andres@mispruebas.com, perico@mispruebas.com
X-Mailer: PHP/" . phpversion());
?>
```

**Usando código de ejemplo**

**Utilizando «separador \n»**

como puedes ver en el ejemplo.

#### Reply-To: correo

En una nueva línea, y sin cerrar las comillas de la cadena iniciada en el encabezado anterior, podemos indicar la dirección de respuesta del mensaje.

La dirección que escribamos donde dice **correo** (fíjate que no va entre comillas) será la dirección a la que se enviará la respuesta si el destinatario una vez recibido tu correo desea responder mediante la opción **Responder** de su programa de correo electrónico.

#### Cc: correo1,correo2,...

De igual forma que en el caso anterior –en una nueva línea y sin cerrar comillas– podemos incluir en **correo1**, **correo2**, etcétera, las direcciones de correo de las personas a las que deseamos enviar copia del mensaje.

No te olvides de separar con comas cada una de las direcciones que, como puedes ver en los ejemplos, no van entre comillas.

#### Bcc: correo1,correo2,...

Esta opción es idéntica en cuanto a su funcionamiento a la anterior, con la única diferencia que esas direcciones no serán visibles por los destinatarios de los mensajes.

#### X-Mailer: PHP/.phpversion()

Es una *frivolidad* que se puede incluir en el **encabezado** del mensaje y que indica que versión de PHP con que ha sido creado.

#### Mucho cuidado!

Debemos insistir en sugerirte una **especial atención** a la sintaxis de este tipo de scripts.

Los contenidos generados por la función *mail* deben ser *interpretados* por el servidor de correo y –tanto en este caso como en los que veremos a continuación– los requisitos de formato de éstos servidores son muy estrictos.

De no adaptarse exactamente a sus especificaciones pueden producirse efectos extraños tales como: envíos incorrectos y/o apariencias indeseadas en los mensajes.

## El mismo ejemplo, utilizando variables

Aquí tenemos un ejemplo donde los parámetros de envío proceden de variables PHP.

Recordemos que esas variables pueden transferirse –mediante un formulario– a un script que se encargue de su envío.

Como puedes observar, hemos puesto las direcciones de los destinatarios de las copias –visibles y ocultas– en sendos arrays y hemos añadido una función que: lee los array, los une en una cadena separándolos con comas y, por último, quita la última coma añadida utilizando la función **substr**.

```
<?
#variables destinatario, asunto, texto, etc.
$destino="andres@mispruebas.com";
$envia="Andrés Curso PHP";
$remite="andres@mispruebas.com";
$asunto="Mensaje experimental";
$texto="Esto es una prueba. No es spam";

#array de destinatarios de copias visibles

$c[0]="perico@mispruebas.com";
$c[1]="juan@mispruebas.com";

#crear la cadena con las direcciones
# y añadir las comas de separación

foreach($c as $pegar) {
$coco .= $pegar;
$coco.= ",";
};

#quitamos la coma del final de la cadena

$l=strlen($coco);

$coco=substr($coco,0,$l-1);

#array de destinatarios de copias OCULTAS

$b[0]="perico@mispruebas.com";
$b[1]="andres@mispruebas.com";

#crear la cadena con las direcciones
# y añadir las comas de separación

foreach($b as $pegar) {
$bco .= $pegar;
$bco.= ",";
};

#quitamos la coma del final de la cadena

$l=strlen($bco);

$coco=substr($bco,0,$l-1);

mail($destino, $asunto, $texto,
"From: $envia <$remite>
Reply-To: $remite
Cc: $coco
Bcc:$bco
X-Mailer: PHP/" . phpversion());
?>
```

ejemplo96.php

Anterior

Índice

Siguiente

[Ver índice](#)

## Formatos MIME



### Formato de los mensajes de correo electrónico

En la página anterior hemos hablado acerca de la manera de enviar un e-mail y veímos la forma de insertar el cuarto parámetro de la función `mail` para incluir algunos elementos de los encabezados MIME.

El formato de los mensajes está especificado en una serie de normas conocidas como el **MIME (Multipurpose Internet Mail Extensions)** en las que se establecen los contenidos y la sintaxis de las diferentes partes de un mensaje.

Recordemos que la función

`mail(dest, asunt, men, cab)`

tiene cuatro parámetros y que las especificaciones del MIME aluden a los dos últimos, es decir a *men* (el **cuerpo** del mensaje) y *cab* que es el **encabezado** del mismo.

Respecto a *dest* (destinatario) y *asunt* no se requieren más comentarios que reiterar la necesidad de incluir esos valores (e-mail del destinatario y asunto) bien directamente, como parámetro en la función, o a través de una variable tal como hemos comentado en la página anterior.

### Cabeceras de los mensajes

Los diferentes elementos de la cabecera de un mensaje deben insertarse **siempre** separados por **saltos de línea** bien pulsando *Enter* o incluyendo la secuencia **\n dentro de la misma línea**.

No pueden incluirse espacios, ni al comienzo de las nuevas líneas ni después de \n, y las **comillas** —que han de contener todo el encabezado— se abren delante del primero de ellos y no se cierran hasta después de haber escrito el último.

Pueden contener lo siguiente:

**Date: xxxx**

**Date:** debe escribirse con esta sintaxis **exactamente**.

El parámetro *xxxx* es una **cadena** que contendrá la fecha de envío del mensaje y que puede obtenerse a través de una de las funciones de

### Las cabeceras MIME de un mensaje

Aquí tienes un ejemplo con los diferentes elementos del encabezado de un mensaje. Como ves, he incluido todos los elementos dentro de la función `mail`.

```
<?
mail("juan@mispruebas.com", "Cabeceras", "Prueba de cabeceras",
      "Date: 24 de Junio de 2001"
MIME-Version: 1.0
From: Estudiante Perico<perico@mispruebas.com>
Cc:perico@mispruebas.com
Bcc:andres@mispruebas.com
Reply-To: perico@mispruebas.com
X-Mailer: PHP/".$phpversion());
?>
```

[ejemplo98.php](#)

Una forma un poco más depurada del script anterior podría ser esta que incluimos aquí debajo.

Sus particularidades son las siguientes:

Recogemos los **datos** en variables e insertamos en la función `mail` esas variables  
La variable **\$cabecera** tiene algunas singularidades:

La vamos construyendo añadiendo subcademas: date, from, etc. etc.

En cada subcadena dejamos **pegado** el contenido a las comillas iniciales

[Ver índice](#)

[Búsqueda rápida](#)

[Página anterior](#)

[Página siguiente](#)

```
<?
# datos del mensaje
$destinatario="juan@mispruebas.com";
$titulo="Cabeceras en variables";
$mensaje="Nueva prueba de cabeceras";
$responder="andres@mispruebas.com";
$remite="andres@mispruebas.com";
$remitente="Otra vez Andres"; //sin tilde para evitar errores de servidor
# cabeceras
$cabecera ="Date: ".date("l j F Y, G:i")."\n";
$cabecera .="MIME-Version: 1.0\n";
$cabecera .="From: ".$remitente."<".$remite.">\n";
$cabecera .="Return-path: ".$remite."\n";
$cabecera .="X-Mailer: PHP/".$phpversion()."\n";

if( mail($destinatario, $titulo, $mensaje,$cabecera)){
    echo "mensaje enviado";}else{print "el mensaje no ha podido enviarse";
}
?>
```

[ejemplo99.php](#)

### Algunas funciones PHP que *incorporamos* en estos ejemplos

Podrás ver en estos ejemplos algunas *funciones raras* que vamos a comentar seguidamente:

**uniqid(pre,bol)**

fecha de PHP tal como puedes ver en el ejemplo.

#### MIME-Version: 1.0

Este elemento de la cabecera especificará la versión MIME que ha de utilizar el cliente de correo para poder interpretar adecuadamente el contenido de los mensajes.

#### From: remitente<e-mail>

Este elemento de la cabecera permite indicar el nombre del remitente (remitente) y su dirección e-mail siguiendo la sintaxis que se especifica. El nombre, como un elemento independiente y la dirección e-mail dentro de <>.

¡Cuidado!

No debemos poner comillas ni en el nombre del remitente, ni en la dirección e-mail, ni en la fecha, etcétera.

Respecto a Cc: y Bcc: ; Reply-To: y X-Mailer: son válidos los comentarios que hemos hecho en la página anterior.

Si no se especifica lo contrario, los mensajes se envían como texto sin formato, pero existen opciones que permiten especificar el formato que ha de tener un mensaje.

La especificación de un formato obliga a incluir otro elemento en cabecera del mensaje:

#### Content-Type:

Este elemento debe ir seguido de la especificación en la que se indique el tipo de contenido. Tiene la sintaxis: tipo/subtipo

El MIME establece un gran variedad de opciones para este propósito. Hablaremos de dos de ellas:

#### text/plain

El text/plain es la opción por defecto y señala que el contenido del mensaje es de tipo texto (text) y del subtipo sin formato (plain)

#### text/html

Como la opción anterior, es tipo texto, pero en este caso, el subtipo es html con lo cual el mensaje se visualizará en formato html siempre que el cliente de correo permite esa posibilidad.

#### Mensajes multiparte

Los tipos anteriores permiten enviar mensajes simples (sin ficheros adjuntos) en uno u otro formato, pero el MIME nos da opciones para insertar dentro de un mismo mensaje

Genera un identificador único basado en la hora actual del sistema expresada en microsegundos y con una longitud de 13 caracteres.

El parámetro pre permite establecer una cadena o número (puede ser una cadena vacía) que se antepone al identificador generado por la función.

Opcionalmente permite el segundo parámetro bol que debe ser un valor booleano (TRUE ó FALSE) o también 0 ó 1.

Cuando este parámetro es TRUE añade al final de la cadena generada anteriormente otra subcadena numérica -generada aleatoriamente- de nueve dígitos, que refuerza la unicidad del identificador.

#### ereg\_replace(busca, reemplaza, cadena)

Busca en la cadena especificada en el parámetro cadena (que puede ser una cadena o una variable que contenga una cadena) las subcadenas especificadas en busca (pueden ser expresiones regulares) y sustituye esas subcadenas por el contenido del parámetro reemplaza.

Esta función devuelve la cadena modificada.

#### strip\_tags(cadena, excepciones)

Suprime todas las etiquetas HTML contenidas en cadena salvo las que se indiquen en excepciones.

Por ejemplo: strip\_tags(\$cadena, '<i><u><b>') eliminaría todas las etiquetas HTML, salvo las indicadas aquí y sus correspondientes cierres.

Si no se especifican excepciones elimina todas las etiquetas.

#### base64\_encode(cadena)

Devuelve una cadena codificada en base64. Esta codificación se hace para permitir que las informaciones binarias puedan ser correctamente manipuladas por sistemas que no generan correctamente los 8 bits, tal como ocurre frecuentemente en los cuerpos de los mensajes de correo electrónico.

#### base64\_decode(cadena)

Realiza el proceso inverso a la anterior. Decodifica una cadena previamente codificada en base64.

#### chunk\_split(cadena, longitud, separador)

Devuelve una cadena obtenida al insertar en la cadena especificada -a intervalos del número de caracteres especificados en el parámetro numérico longitud- el contenido de una subcadena indicada en el parámetro separador.

Por defecto -cuando no se especifican los parámetros- longitud es igual a 76 caracteres y el separador es la cadena \r\n (retorno y salto de línea).

Esta función se utiliza para convertir al formato especificado en la RFC 2045 (especificación para MIME) las cadenas obtenidas por base64\_encode.

Es el formato habitual de los ficheros adjuntos de los e-mail.

## Mensaje con contenido alternativo

```
<?
# creamos la variables "salto" para "mayor comodidad
# un salto es la secuencia retorno de carro-nueva linea
# dos saltos es algo similar pero duplicado

$UN_SALTO="\r\n";
$DOS_SALTOS="\r\n\r\n";

# creamos el remitente, etc. y tambien la que parte que
```

elementos de diferentes tipos y subtipos.

Las opciones de mayor interés son las siguientes:

#### **multipart/alternative**

Es la forma de especificar que el mensaje tiene *varias partes* (*multipart*) de las que el destinatario *ha de ver una sola* (*alternative*).

Se podría utilizar en casos en los que sea necesario prever la posibilidad de que un mensaje con formato HTML pueda ser visualizado como *texto plano* cuando el *cliente de correo* no soporte HTML.

Podemos hacer un mensaje *a medida* que se presentará de una forma u otra según el *cliente* utilizado para leerlo.

#### **multipart/mixed**

Cuando en el **Content-Type** se establece el tipo *multiparte* y el subtipo *mezclado* (*mixed*) será cuando tengamos la posibilidad de *adjuntar ficheros* al mensaje.

Las *diferentes partes* de un mensaje deben ir separadas – tanto en modo *alternativo* como *mezclado*– y para ello hay que incluir un nuevo elemento en el encabezado. Se trata de un *separador* al que se llama **boundary**.

#### **boundary=cadena**

Dentro del **encabezado** y *siempre en línea aparte* (fíjate que en los ejemplos o está en línea aparte o aparece el \n) debemos incluir el elemento **boundary=** (sin símbolo de \$ delante) y detrás del signo igual una cadena (en este caso **entre comillas**) que en principio puede ser una cadena cualquiera que no contenga espacios, aunque lo habitual es incluirla con el formato que podemos ver en los ejemplos.

### **El cuerpo del mensaje**

En su formato más simple el cuerpo del mensaje contiene únicamente texto, pero cuando se trata de *multipartes* deberá contener necesariamente: los *separadores* de las diferentes partes, los *encabezados* de cada una de las partes y sus respectivos *contenidos*.

La secuencia habría de ser de este tipo:

Separador  
Content-type  
Content-Transfer-Encoding  
\*\*Content-Disposition  
\*\*Lectura del fichero  
\*\*Codificación  
Inserción en cuerpo  
Separador

```
# contiene el código HTML del mensaje

$destinatario="juan@mispruebas.com";
$titulo="Mensaje alternativo Texto Plano - HTML ";
$mensaje=<html><head></head><body bgcolor='#ff0000'>";
$mensaje .=<font face='Arial' size=6>Prueba HTML. </font>";
$mensaje .=<aquí pueden ir tildes: á, é, í, ó, ú, ñ</body></html>";
$responder="andres@mispruebas.com";
$remite="andres@mispruebas.com";
$remitente="Andres Perez y Perez";
// omitimos las tildes en encabezados para evitar errores de servidor

# creamos el separador de bloques del mensaje
# anteponiendo _separador" aunque podríamos haber puesto "tiburcio"
# generamos un identificador único utilizando un número aleatorio
# como "semilla" y luego lo codificamos con la función md5

$separador ="_separador".md5 (uniqid (rand()));

# creamos la variable cabecera con los elementos
# ya utilizados en los ejemplos anteriores y ponemos al final
# de cada elemento UN SALTO DE LINEA

$cabecera = "Date: ".date("l j F Y, G:i").$UN_SALTO;
$cabecera .= "MIME-Version: 1.0\n";
$cabecera .= "From: ".$remitente."<".$remite.">".$UN_SALTO;
$cabecera .= "Return-path: ".$remite.$UN_SALTO;
$cabecera .= "Cc:perico@mispruebas.com".$UN_SALTO;
$cabecera .= "Reply-To: ".$remite.$UN_SALTO;
$cabecera .= "X-Mailer: PHP/". phpversion().$UN_SALTO;

# AQUÍ DEFINIMOS EL CONTENIDO MULTIPART, fíjate que lo acabamos con ";"

$cabecera .= "Content-Type: multipart/alternative;".$UN_SALTO;

# insertamos BOUNDARY (fíjate que dejo un espacio
# en BLANCO DELANTE y ponemos al FINAL los DOS SALTOS DE LINEA

$cabecera .= " boundary=$separador".$DOS_SALTOS;

# colocamos el primer separador (con los dos guiones delante)
# antes de insertar la primera parte del mensaje
# que es el texto plano para el caso de que el cliente de correo
# no soporte HTML

$texto_plano ="--$separador".$UN_SALTO;

# especificamos el tipo de contenido y la codificación
# e inserto DOS SALTOS AL FINAL ya que ahí acaba la cabecera de esta parte
$texto_plano .= "Content-Type:text/plain; charset=\"ISO-8859-1\"".$UN_SALTO;
$texto_plano .= "Content-Transfer-Encoding: 7bit".$DOS_SALTOS;

# cambiamos las etiquetas "<br>" por saltos de línea
# y luego quitamos todas las etiquetas HTML del cuerpo del mensaje
# ya que el texto plano no debe llevar ese tipo de etiquetas

$extractor= strip_tags(eregi_replace("<br>", $UN_SALTO, $mensaje));

$texto_plano .= $extractor;

# insertamos un nuevo separador para señalar el final
# de la primera parte del mensaje y el comienzo de la segunda
# en este caso ponemos UN SALTO delante del separador ya que de lo contrario
# al componer el mensaje se uniría con la cadena texto_plano anterior
# que no tiene SALTO DE LINEA AL FINAL

$texto_html = $UN_SALTO."--$separador".$UN_SALTO;

# especificamos el encabezado HTML para el siguiente bloque
# y ponemos en la última línea los DOS SALTOS DE LINEA

$texto_html .= "Content-Type:text/html; charset=\"ISO-8859-1\"".$UN_SALTO;
$texto_html .= "Content-Transfer-Encoding: 7bit".$DOS_SALTOS;
#añado la cadena que contiene el mensaje
$texto_html .= $mensaje;

# insertamos SOLAMENTE un SALTO DE LINEA
# estamos al final del mensaje
```

.....  
otra parte

14

### Separador final

Los apartados señalados con \*\* sólo se incluirán en el caso de que junto con el mensaje se adjunten ficheros.

## Content-type

Los tipos y subtipos más habituales son los siguientes:

*Para incluir textos:  
los ya mencionados  
**text/plain**  
**text/html***

*Para imágenes:*  
según el tipo de imagen  
**image/jpeg**  
**image/gif**

*Para sonidos:  
audio/basic*

*Para vídeo:  
video/mpeg*

*Para ejecutables, comprimidos y otros ficheros adjuntos:  
application/octet-stream*

En cualquier caso, si quieres utilizar algún otro tipo de archivo puedes consultar en la web las especificaciones del MIME.

A parte de *tipo/subtipo* puede añadirse a *Content-type* -en el caso de texto- separado por *punto y coma*, la especificación del tipo de alfabeto (*charset=*) seguida del tipo de codificación (te sugerimos el "ISO-8859-1" que hace alusión al alfabeto latino).

Cuando se trata de **ficheros adjuntos** deberemos poner, después del *punto y coma, name=* seguido del *nombre y extensión* del fichero que se adjunta.

## Content-Transfer-Encoding

Este apartado del encabezado puede especificar una de los siguientes codificaciones:

**7BIT  
8BIT  
BASE64  
BINARY  
QUOTED-PRINTABLE**

La transferencia codificada en **7bit** representa la codificación habitual en el formato ASCII de 7 bits.

No permite caracteres ASCII con un código mayor que 127.

**Quoted-printable** constituye una de las alternativas al formato ASCII de 7 bits.

Esta codificación suele usarse cuando la mayoría de los caracteres del mensaje puede escribirse con formato ISO-ASCII de 7 bits.

```
$texto_html .= $UN_SALTO;  
  
# unimos ambas cadenas para crear el cuerpo del mensaje  
  
$mensaje=$texto_plano.$texto_html;  
  
# enviamos el mensaje utilizando  
  
if( mail($destinatario, $titulo, $mensaje,$cabecera)){  
    echo "mensaje enviado ";}else{print "ha habido errores en el envio"  
}  
  
?>
```

## **ejemplo100.php**

## Mensaje con ficheros adjuntos

```

<?
# definimos estas variables igual que en el ejemplo anterior

$UN_SALTO="\r\n";
$DOS_SALTOS="\r\n\r\n";

#incluimos en varias, asunto, un texto en HTML
# remitente, etc. etc.

$destinatario="perico@mispruebas.com";
$titulo="Mensaje con dos fichero adjuntos";
$mensaje=<html><head></head><body bgcolor="#ff0000">;
$mensaje .=<font face="Arial" size=6>Prueba HTML </font>;
$mensaje .=</body></html>;
$responder="andres@mispruebas.com";
$remite="andres@mispruebas.com";
$remitente="Andres otra vez";

# definimos el separador de parte
# con el mismo procedimiento del ejemplo anterior

$separador = "_separador_de_trozos_".md5 (uniqid (rand()));

# insertamos los datos de la cabecera del mensaje

$cabecera = "Date: ".date("l j F Y, G:i").$UN_SALTO;
$cabecera .= "MIME-Version: 1.0".$UN_SALTO;
$cabecera .= "From: ".$remitente."<".$remite.">".$UN_SALTO;
$cabecera .= "Return-path: ".$remite.$UN_SALTO;
$cabecera .= "Reply-To: ".$remite.$UN_SALTO;
$cabecera .= "X-Mailer: PHP/".$phpversion().$UN_SALTO;

# especificamos el tipo de contenido mutipart/mixed
# ya que ahora insertaremos ficheros de distinto tipo

$cabecera .= "Content-Type: multipart/mixed;".$UN_SALTO;

# insertamos el valor de boundary haciéndola igual a $separador
# y acabamos con DOS SALTOS porque es el FINAL DE LA CABECERA

$cabecera .= " boundary=$separador".$DOS_SALTOS;

/* Parte primera del envio -Mensaje en formato HTML
=====
Separador inicial
-----
$texto ="--$separador".$UN_SALTO;

/* Encabezado parcial
-----
/* especificamos que este primer elemento
será texto y que irá codificado en formato 7 bits */

```

Prevé que los caracteres con códigos ASCII superiores 127 se expresen mediante un mecanismo especial.

Evita, entre otras cosas, que las *letras con tilde* y algunos otros caracteres especiales se visualicen incorrectamente. Es la forma de codificación más recomendable para *textos*.

La codificación en **base64** convierte cadenas binarias en cadenas de texto, con lo cual pueden ser enviadas de forma más segura. Es la forma de codificación habitual de las imágenes y los ficheros *exe*, *zip*, etcétera.

### Content-Disposition

Se utiliza únicamente cuando se insertan ficheros adjuntos.

Permite dos opciones: **inline** o **attachment**.

**Inline** permite que los contenidos se visualicen junto con el cuerpo del mensaje mientras que con **attachment** sólo aparecerán como ficheros adjuntos.

Por lo que hemos podido comprobar *Outlook Express* no suele respetar esa condición y presenta siempre las imágenes en el mensaje. Sin embargo, sí funciona en los correos web.

Este elemento del encabezado lleva *-separada por punto y coma-* una segunda parte.

El **filename=**, donde se puede especificar entre comillas un nombre y una extensión (igual o distinta de la original) con la que se denominará al fichero en el mensaje recibido.

### Lectura del fichero

Cuando se trata de insertar un *fichero* el proceso es el típico de lectura de ficheros, es decir:  
Hay que crear el identificador de recurso del fichero en modo **lectura**. Recoger en una variable el *buffer* de lectura.

Cerrar el fichero.

### Codificación

Una vez recogido en el fichero a transmitir en una variable, el paso siguiente es *codificar* esa variable.

Utilizaremos la codificación más habitual y flexible **-base64-** que requerirá el uso de dos nuevas funciones PHP:

### base64\_encode chunk\_split

Mediante la primera se realiza la codificación propiamente dicha mientras que la segunda organiza el

```
$texto .= "Content-Type: text/html; charset=\"ISO-8859-1\"".$UN_SALTO;
$texto .= "Content-Transfer-Encoding: 7bit".$DOS_SALTOS;

/* Contenido de esta parte del mensaje
-----
# ya teniamos escrito el texto del mensaje más arriba
# simplemente lo añadimos a la cadena de texto

$texto .= $mensaje;

#la variable $texto recoge esta parte del documento
# la uniremos al final con las siguientes

/* Separador de partes
-----
$adj1 = $UN_SALTO."--$separador".$UN_SALTO;

/* Parte segunda de mensaje -Fichero adjunto n° 1
=====
/* Encabezado parcial
-----
# especificamos el tipo de contenido image/jpeg
# ya que ese será el documento que vamos a enviar
# ponemos el nombre del fichero (debemos tenerlo en el servidor
# con ese mismo nombre)
# establecemos in line como disposición para que pueda ser visualizado
# directamente en el cuerpo del mensajes
# en filename le asignamos el nombre con el que queremos que sea
# recibido por el destinatario
# por ultimo especificamos la codificacion como base64

$adj1 .= "Content-Type: image/jpeg;";
$adj1 .= " name=\"casa08.jpg\"".$UN_SALTO;
$adj1 .= "Content-Disposition: inline; ";
$adj1 .= "filename=\"leoncio.jpg\"".$UN_SALTO;
$adj1 .= "Content-Transfer-Encoding: base64".$DOS_SALTOS;

/* Lectura previa del fichero a adjuntar
-----
# abrimos el fichero en modo lectura (r)
# y leemos todo su contenido midiendo previamente
# su longitud con filesize
# recogemos en $buff el contenido del fichero
# y cerramos después

$fp = fopen("casa08.jpg", "r");
$buff = fread($fp, filesize("casa08.jpg"));
fclose($fp);

/* Codificación del fichero a adjuntar
-----
# codificamos en base 64 y troceamos en lineas de 76 caracteres
# y añadimos el resultado a la variable adj1

$adj1 .=chunk_split(base64_encode($buff));

/* Separador de partes
-----
$adj2 = $UN_SALTO."--$separador".$UN_SALTO;

/* Tercera parte de mensaje -Fichero adjunto n° 2
=====
/* Encabezado parcial
-----
# los contenidos del encabezado son similares al caso anterior
# con la salvedad de que el contenido es ahora
# application/octet-stream ya que contiene un fichero ejecutable
# y la disposicion es attachment, no tiene sentido tratar
# de visualizar un fichero zip

$adj2 .= "Content-Type: application/octet-stream;";
$adj2 .= " name=\"apachito.zip\"".$UN_SALTO;
$adj2 .= "Content-Disposition: attachment;
filename=\"apachito.zip\"".$UN_SALTO;
```

fichero codificado en trozos, de 76 caracteres cada uno, insertando detrás de cada uno un *salto de línea*.

Si *analizas* un mensaje de correo que contenga un fichero adjunto –propiedades, ver codificación–, podrás ver esa fragmentación *tan cuidada* -un montón de líneas de texto *muy raro*- perfectamente alineadas por los márgenes por efecto de *chunk\_split*.

### Inserción en el cuerpo

La fase final del proceso es la de *agrupar* los diferentes *trozos* en una sola variable, que será la que se insertará como parámetro texto en la función e-mail.

**Cuidado!**

La inserción de ficheros adjuntos requiere que éstos estén disponibles en el servidor por lo que, antes de enviarlos, habrá que *subirlos* al servidor utilizando un proceso como el que hemos analizado cuando hablábamos de *Transferencia de ficheros*.

### Sobre los ejemplos

Hemos incluido dos ejemplos relativos al envío de ficheros en formato HTML y con ficheros adjuntos.

No entraremos en el estudio detallado del MIME.

Quedaremos únicamente en sus aspectos funcionales en cuanto a los requerimientos de formato, separadores, etc.

Quizá te parezca *obsesivo* el hincapié que hacemos en los ejemplos sobre los detalles de la sintaxis.

Nuestra insistencia se debe al carácter *sumamente estricto* de la especificación MIME, donde un *simple salto de línea* puede ser la causa de que script deje de funcionar.

```
$adj2 .="Content-Transfer-Encoding: base64".$DOS_SALTOS;

/* Lectura previa del fichero a adjuntar
-----
# abrimos el fichero en modo lectura (r)
# y leemos todo su contenido midiendo previamente
# su longitud con filesize
# recogemos en $buff el contenido del fichero
# y cerramos después

    $fp = fopen("apachito.zip", "r");
    $buff = fread($fp, filesize("apachito.zip"));
    fclose($fp);

/* Codificación del fichero a adjuntar
-----
$adj2 .=chunk_split(base64_encode($buff));

/* Separador final YA NO HAY MAS PARTES
-----
$adj2 .= $UN_SALTO."--$separador".$UN_SALTO;

/* Unión de todas las PARTES
-----
# unimos en la variable mensaje todos los elementos
# y lo hacemos por el orden en el que fueron creados

$mensaje=$texto.$adj1.$adj2;

/* Envio del mensaje
-----
if(mail($destinatario, $titulo, $mensaje,$cabecera)){
    echo "mensaje enviado";}else{print "ha habido problemas";
}

?>
```

ejemplo101.php

### Ejercicio nº 32

Diseña un script de forma que al cargarse la página que lo contiene se envíe –de forma automática– un mensaje de correo, indicando la fecha y hora de acceso, al usuario [juan@mispruebas.com](mailto:juan@mispruebas.com)

Anterior

Índice

Siguiente



Ver índice

## Imágenes dinámicas



### Imágenes dinámicas

PHP permite la *creación dinámica* de imágenes. Quiere esto decir que una imagen puede ser presentada en la página web sin necesidad de ser almacenada previamente en el servidor y, además, con un contenido que puede ser modificado en cada instante.

Esta posibilidad que ofrece PHP puede resultar muy útil a la hora de presentar gráficos estadísticos ya que permitiría utilizar **valores actuales** obtenidos, por ejemplo, de una *base de datos*.

#### Formatos GIF

Aunque son abundantes los materiales que aluden a este formato gráfico -incluso en las páginas oficiales PHP- los formatos GIF **sólo funcionan en modo lectura**.

Parece ser que existe un conflicto sobre los derechos de propiedad del *algoritmo de compresión* que se utiliza en los ficheros .gif y eso está obligando a los *desarrolladores* de PHP a abandonar este tipo de formato.

#### Formatos PNG

El formato de imágenes **PNG** (*Portable Network Graphic*) nos permite seguir disponiendo de un formato gráfico de *difusión gratuita* con una funcionalidad similar al GIF en lo que se refiere a *transparencias* y que junto con la posibilidad de usar también el formato **JPG** va a cubrir las necesidades gráficas de esta utilidad de PHP.

### Requisitos del sistema

El manejo de imágenes dinámicas requiere que esté instalada la librería de PHP llamada **php\_gd2.dll**.

En la versión de PHP que estamos manejando se instala por defecto, pero requiere que la configuración de fichero **php.ini** tenga **activada** esta extensión.

Para activarla deberás editar el fichero **php.ini**, buscar la línea que dice: **;extensions=php\_gd2.dll** y quitar el punto y coma que lleva delante.

A esto se le llama en el argot *descomentar esa línea*. Las líneas de comentario en el fichero **php.ini** empiezan por ese signo de puntuación.

### Comprobación

Una vez que hayamos modificado **php.ini** -recuerda que debemos hacerlo con el **servidor Apache apagado**- activaremos de nuevo Apache y ejecutando **info.php** encontraremos algo similar a lo que ves en esta imagen:

gd

|                  |                             |
|------------------|-----------------------------|
| GD Support       | enabled                     |
| GD Version       | bundled (2.0.12 compatible) |
| FreeType Support | enabled                     |
| FreeType Linkage | with freetype               |
| GIF Read Support | enabled                     |
| JPG Support      | enabled                     |
| PNG Support      | enabled                     |
| WBMP Support     | enabled                     |
| XBM Support      | enabled                     |

Cuando esto ocurra nuestra configuración será la adecuada para utilizar las funciones PHP de este ámbito y estaremos en disposición de poder generar imágenes dinámicas.

### Scripts para gráficos estadísticos

Si en algún momento tienes interés en insertar en tus páginas gráficos estadísticos, en esta dirección <http://www.aditus.nu/jpgraph/index.php> podrás encontrar una interesante colección de scripts listos para usar, con licencia *gratuita para usos no comerciales*.

Anterior



Índice



Siguiente





Ver índice

# Autoinstalación de servidores



## Precauciones previas

Esta instalación no es otra cosa que una alternativa *automática* para el caso que decidas evitar o posponer la instalación manual que venimos describiendo en estas primeras páginas del curso.

Antes de efectuar esta instalación te sugerimos que desinstales cualquier versión anterior que pueda tener en tu equipo. Hacemos particular hincapié en la desinstalación de cualquier versión de Filezilla Server ya que puede ser una fuente de problemas.

## La instalación automática

La utilidad que comentamos aquí no hace otra cosa que realizar de forma automática los procesos descritos en páginas anteriores. Es decir:

- Crear el directorio de instalación (ServidoresLocales, por defecto)

- Instalar el servidor Apache.

- Instalar PHP4 y PHP5

- Copiar las librerías dll -de PHP4 y Php5- requeridas en el directorio system.

- Instalar MySQL configurando de forma automática el usuario *pepe*.

- Instalar el servidor FTP incluyendo todos los usuarios y permisos que se describen en la página de instalación manual de este servicio.

- Creación de los directorios servidorFTP (en el disco c:) tal como se describen en el proceso manual.

- Instalación del servidor de correo y configuración de cuentas de los usuarios: *juan*, *perico* y *andres* tal como se describen en el proceso de instalación manual de este servidor.

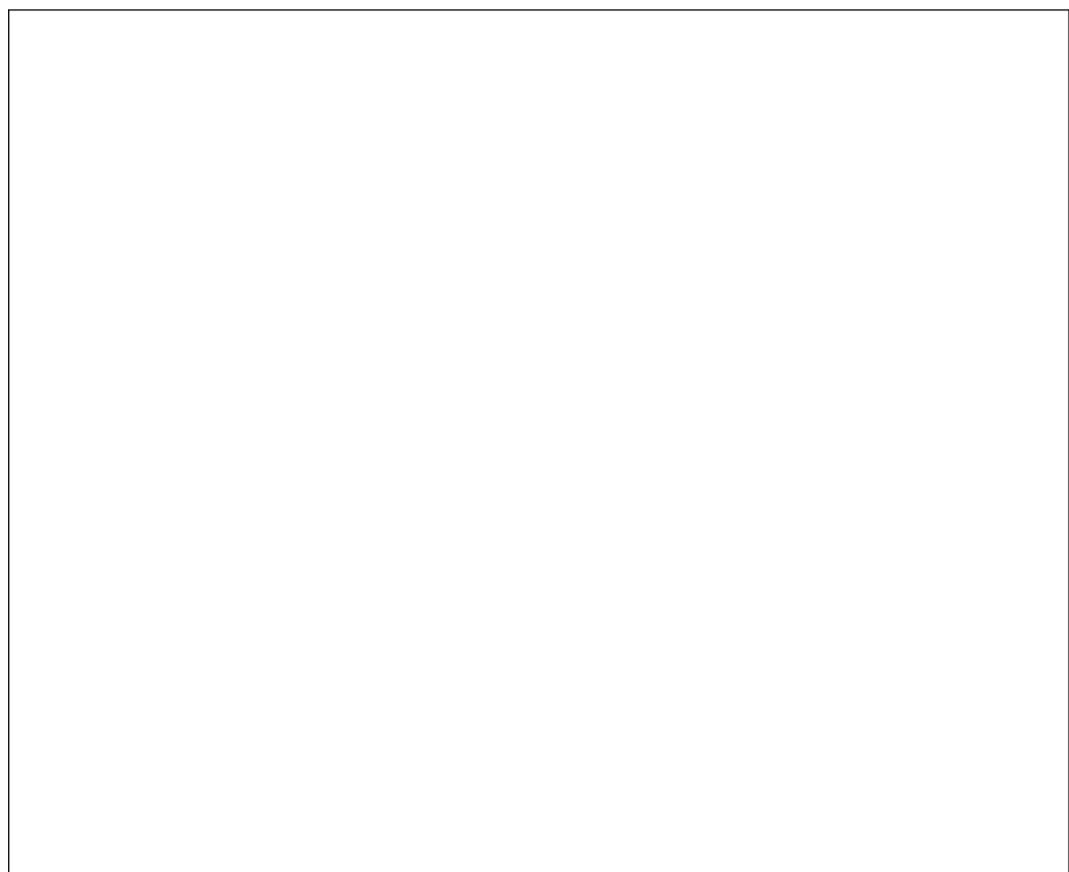
- Instalación del editor *Php-Coder*

- También instala manuales de ayuda de PHP y MySQL y una pequeña aplicación que permite gestionar cada uno de los servidores (arrancar y parar) y la posibilidad de trabajar con PHP4 o PHP5.

- Al cambiar la versión de PHP se reconfigura automáticamente la configuración del servidor.

## Instalación automática

En el directorio **software** del CD-ROM encontrarás un fichero llamado *Instalar\_servidores.exe*. Bastará con que hagas doble click sobre su ícono y comenzará el proceso de instalación automática cuyas ventanas y secuencia de instalación puedes ir viendo en la imagen.



Pulsando sobre la imagen podrás visualizar los diferentes pasos del proceso

Al ejecutar este instalador ya tendrás totalmente **instalados y configurados** en tu equipo: **Apache 2.2**, **PHP5**, **PHP4**, **MySQL**, **FileZilla Server** y el servidor de correo **Mercury**, además del **editor PHP-Coder**.

Una vez realizada esta instalación ya estaremos en condiciones de comenzar los desarrollos de los contenidos del curso tanto en su versión Iniciación como en la de Profundización.

### ¡Cuidado!

Recuerda que si utilizas **Windows Vista** como sistema operativo deberás ejecutar la aplicación de gestión de servidores (también el desinstalador) **con privilegios de Administrador**. En vez del clásico doble click sobre el ícono deberás pulsar con el botón derecho y elegir Ejecutar como Administrador.

De no hacerlo así -puede que en principio todo tenga apariencia de funcionar sin ese requisito- se te «*colgará*» el equipo cuando intentes poner en marcha el servidor FTP.

Durante el proceso de desinstalación del conjunto Servidores Locales también será necesarios los privilegios de Administrador para que pueda desinstalarse correctamente el servicio Filezilla Server.

Anterior



Índice



Siguiente





Ver índice

## Formatos soportados



### Formatos de imágenes

Pese a que **info.php** nos devuelve información sobre los tipos de imágenes soportados por la versión en uso de PHP, existe una función que permite determinar cuales de esos tipos son soportados por PHP.

#### imagetypes()

Devuelve un campo de bits correspondiente a los formatos soportados por la versión de **GD** que estamos utilizando.

Los formatos de imagen que PHP soporta actualmente son: GIF, JPG, PNG y WBMP.

En la parte derecha tienes el código fuente de un fichero que permite obtener información sobre los formatos soportados por tu versión de PHP.

El conocimiento de estas posibilidades gráficas puede sernos muy útil a la hora de elegir entre los diferentes formatos gráficos disponibles.

```
<?
if (imagetypes() & IMG_GIF) {
    echo "El tipo GIF es soportado<br>";
    }else{
    echo "El tipo GIF NO ES SOPORTADO<BR>";
}
if (imagetypes() & IMG_PNG) {
    echo "El tipo PNG es soportado<br>";
    }else{
    echo "El tipo PNG NO ES SOPORTADO<BR>";
}
if (imagetypes() & IMG_JPG) {
    echo "El tipo JPG es soportado<br>";
    }else{
    echo "El tipo JPG NO ES SOPORTADO<BR>";
}
if (imagetypes() & IMG_WBMP) {
    echo "El tipo WBMP es soportado<br>";
    }else{
    echo "El tipo WBMP NO ES SOPORTADO<BR>";
}

?>
```

ejemplo102.php

Anterior

Índice

Siguiente



Ver índice

## Creando imágenes



### Creación de imágenes dinámicas

Una imagen dinámica es tan sólo un **script** que contiene las instrucciones para la creación de esa imagen.

Para visualizar una imagen dinámica desde una página web basta con invocar el fichero que contiene el script desde la etiqueta clásica de inserción de imágenes de HTML

``

donde **imgxx.php** será el nombre del script que genera la imagen.

#### Primera etiqueta

Una vez conocidos los formatos que soporta nuestra versión, ya podemos generar imágenes utilizando cualquiera de esos formatos.

Trabajaremos con dos de ellos: **JPG** y **PNG**.

La primera instrucción que ha de contener cualquier script que deba generar imágenes ha de ser la siguiente:

**Header("Content-type:  
image/jpeg")**

si se trata de crear una imagen **JPG** o:

**Header("Content-type:  
image/png")**

si pretendemos que la imagen tenga formato **PNG**.

#### ¡Cuidado!

Cualquier etiqueta **header** (cabecera) ha de incluirse obligatoriamente **al comienzo del script** antes que ninguna otra instrucción y sin ninguna línea en blanco que la preceda.

Pondremos siempre estas instrucciones inmediatamente debajo de **<?** sin que las separe ninguna línea en blanco.

### Creación de imágenes

Definida la etiqueta anterior tenemos que: **crear** la imagen, **dibujarla** y luego **enviarla** al navegador para que pueda ser visualizada y, por último, (no es imprescindible pero si muy conveniente) **borrarla**, con el fin de liberar la memoria del servidor

### Una primera imagen

```
<?
    Header ("Content-type: image/jpeg");

    $im = imagecreate(200,200);

    Imagejpeg ($im);

    Imagedestroy ($im);

?>
```

[Ver img1.php](#)

#### ¡Cuidado!

**No dejes NUNCA líneas en blanco** entre la etiqueta **<?** de comienzo del script y la línea que contiene **Header**

Si escribiéramos el script anterior sustituyendo **image/jpeg** por **image/png** e **Imagejpeg(\$im)** por **Imagepng(\$im)** no visualizaremos nada.

El formato **jpg** –a falta de especificaciones– considera la imagen con negro como color de fondo, pero **png** requiere que ese color sea especificado.

### Añadir un color de fondo

```
<?
    Header ("Content-type: image/jpeg");

    $im = imagecreate(200,200);

    $fondo=imagecolorallocate ($im, 0, 0, 200);

    Imagefill ($im, 0, 0, $fondo);

    Imagejpeg ($im);

    Imagedestroy ($im);

?>
```

[Ver img2.php](#)

```
<?
    Header ("Content-type: image/png");

    $im = imagecreate(200,200);

    $fondo=imagecolorallocate ($im, 0, 0, 200);

    Imagefill ($im, 0, 0, $fondo);

    Imagepng ($im);

    Imagedestroy ($im);

?>
```

ocupada durante el proceso de generación de la misma.

Estas son las funciones PHP para esos procesos:

#### \$nom = imagecreate(anc,al)

Con esta función se **crea** una imagen del tamaño indicado en los parámetros **anc** y **al** (en **pixels**) que será *recogida* en la variable **nom**.

Esta función es idéntica para cualquier formato de imagen.

#### Envío de imágenes al navegador

Para enviar imágenes al navegador (visualización) se usan **funciones diferentes** según el tipo de imagen definida en **Header**.

Si pretendemos que la imagen tenga formato **JPG** habremos puesto en **Header** la indicación **jpeg** (*¡cuidado! observa la sintaxis... jpeg*). En este caso la función de visualización será:

#### Imagejpeg(\$nom)

Si se tratara de una imagen en formato **PNG** (recuerda que debe estar definido en Header) la sintaxis sería:

#### Imagepng(\$nom)

#### Eliminando imágenes de la memoria

Para borrar imágenes de la memoria del servidor (que no del navegador) se utiliza la siguiente sintaxis:

#### Imagedestroy(\$nom)

#### Creando colores

PHP permite crear una *paleta* de colores. Para ello se pueden crear **variables de color** (con independencia del formato utilizado) mediante la siguiente función:

#### \$color=imagecolorallocate (\$nom,R,G,B)

donde la variable recoge el color resultante de mezclar los colores primarios indicados en **R**, **G** y **B** que serán **números enteros** comprendidos entre **0** y **255** y que especifican la *intensidad* de las luces **roja**, **verde** y **azul** utilizadas para la obtención del color.

Se pueden definir tantos colores como se deseen tan sólo con utilizar nombres de variables distintos para cada uno de ellos.

[Ver img3.php](#)

## Dibujar un rectángulo sin relleno

```
<?
Header("Content-type: image/jpeg");
$im = imagecreate(200,200);
$fondo=imagecolorallocate ($im, 0, 0, 200);
$blanco=imagecolorallocate ($im, 255, 255, 255);
Imagefill ($im, 0, 0, $fondo);

imagerectangle ($im, 10, 10, 190, 190, $blanco);

Imagejpeg($im);

Imagedestroy($im);
?>
```

[Ver img4.php](#)

```
<?
Header("Content-type: image/png");
$im = imagecreate(200,200);
$fondo=imagecolorallocate ($im, 0, 0, 200);
$blanco=imagecolorallocate ($im, 255, 255, 255);

Imagefill ($im, 0, 0, $fondo);

imagerectangle ($im, 10, 10, 190, 190, $blanco);

Imagepng($im);

Imagedestroy($im);
?>
```

[Ver img5.php](#)

## Dibujando un rectángulo con relleno

```
<?
Header("Content-type: image/jpeg");
$im = imagecreate(200,200);
$fondo=imagecolorallocate ($im, 0, 0, 200);
$blanco=imagecolorallocate ($im, 255, 255, 255);
$amarillo=imagecolorallocate ($im, 255, 255, 0);
Imagefill ($im, 0, 0, $fondo);
imagerectangle ($im, 10, 10, 190, 190, $blanco);

imagefilledrectangle ($im, 20, 20, 180, 180, $amarillo);

Imagejpeg($im);

Imagedestroy($im);
?>
```

[Ver img6.php](#)

Para aplicar un color de fondo a una imagen (no importa el tipo del formato) se utiliza la siguiente función:

### Imagefill(\$nom,x,y,\$col)

Aquí **\$nom** es la *variable* que contiene la imagen, **x** e **y** son las coordenadas del punto de la imagen a partir del cual se aplica el relleno y **\$col** el color (previamente definido) que se pretende aplicar a la imagen.

Mediante esta función todos los puntos **colindantes** con el de coordenadas **x,y** que tengan su mismo color serán rellenados con el color especificado en la variable **\$col**.

### Rectángulos sin relleno

Para dibujar un rectángulo sin relleno (solo las líneas) se utiliza la siguiente función:

### Imagerectangle()

**\$nom, \$x0, \$y0, \$x1, \$y1, \$col**

Donde **\$nom** es el nombre de la imagen, **x0, y0** son las coordenadas del **vértice superior izquierdo** y **x1, y1** las coordenadas del **vértice inferior derecho** y **\$col** el color que pretendemos asignar a las líneas del rectángulo.

El punto (0,0) siempre es la esquina superior izquierda de la imagen y recuerda que si no usas –en las líneas– un color distinto al del fondo no se visualizará el rectángulo.

### Rectángulos con relleno

Para dibujar un rectángulo *con relleno* se utiliza la siguiente función:

### Imagefilledrectangle( \$nom, \$x0, \$y0, \$x1, \$y1, \$col)

Los parámetros son idénticos a los del caso anterior con la única diferencia de que en este caso el rectángulo aparecerá relleno con el color elegido.

### Polígonos con relleno

Para colorear el fondo de un polígono son necesarias dos operaciones:

Crear un **array** con las coordenadas de cada uno de sus vértices.

Aplicar la función que dibuja polígonos de este tipo.

La creación del array podría hacerse así:

**\$v=(x0, y0, x1, y1,... xn, yn )**

donde se irían introduciendo las coordenadas de los **sucesivos vértices del polígono** (x e y de cada

```
<?
Header("Content-type: image/png");

$im = imagecreate(200,200);

$fondo=imagecolorallocate ($im, 0, 0, 200);
$blanco=imagecolorallocate ($im, 255, 255, 255);
$amarillo=imagecolorallocate ($im, 255, 255,0);
Imagefill ($im, 0, 0, $fondo);
imagerectangle ($im, 10, 10, 190, 190, $blanco);

imagefilledrectangle ($im, 20, 20, 180, 180, $amarillo);

Imagepng($im);
Imagedestroy($im);

?>
```

[Ver img7.php](#)

### Dibujando un polígono relleno

```
<?
Header("Content-type: image/jpeg");

$esquinas=array(20,100,100,180,180,100,100,20);

$im = imagecreate(200,200);

$fondo=imagecolorallocate ($im, 0, 0, 200);
$blanco=imagecolorallocate ($im, 255, 255, 255);
$amarillo=imagecolorallocate ($im, 255, 255,0);
Imagefill ($im, 0, 0, $fondo);
imagerectangle ($im, 10, 10, 190, 190, $blanco);
imagefilledrectangle ($im, 20, 20, 180, 180, $amarillo);

imagefilledpolygon ($im, $esquinas, 4, $blanco);

Imagejpeg($im);
Imagedestroy($im);

?>
```

[Ver img8.php](#)

```
<?
Header("Content-type: image/png");

$esquinas=array(20,100,100,180,180,100,100,20);

$im = imagecreate(200,200);

$fondo=imagecolorallocate ($im, 0, 0, 200);
$blanco=imagecolorallocate ($im, 255, 255, 255);
$amarillo=imagecolorallocate ($im, 255, 255,0);
Imagefill ($im, 0, 0, $fondo);
imagerectangle ($im, 10, 10, 190, 190, $blanco);
imagefilledrectangle ($im, 20, 20, 180, 180, $amarillo);

imagefilledpolygon ($im, $esquinas, 4, $blanco);

Imagepng($im);
Imagedestroy($im);
```

vértice).

La creación de un polígono de este tipo requiere la siguiente función:

**imagefilledpolygon(\$nom, \$vert, nº vert , \$col)**

donde **\$nom** es el nombre de la imagen, **\$vert** es el **array** que contiene las coordenadas de los vértices, **nº vert** es el número de vértices y **\$col** es el color de relleno.

### Polígonos sin relleno

Su funcionamiento es idéntico al anterior en tanto requiere que se defina el **array** de coordenadas de los vértices y los parámetros de la función son los mismos indicados en el caso anterior. Sólo se modifica el nombre de la función que en este caso es:

**imagepolygon(\$nom, \$vert, nº vert , \$col)**

### Elipses, circunferencias y arcos

Una misma función nos permite dibujar elipses, circunferencias y arcos. Es la siguiente:

**imagearc(\$nom, Xc, Yc , a, b, Gi, Gf, \$col)**

Los parámetros de esta función son los siguientes:

**\$nom** es el nombre de la imagen.

**Xc** e **Yc** las *coordenadas del centro* de la ellipse.

**a** es la *longitud del eje horizontal de la ellipse*.

**b** es la *longitud del eje vertical de la ellipse*.

**Gi** es el *punto inicial del arco* y se expresa en *grados sexagesimales*.

**Gf** es el *punto final del arco* también en *grados sexagesimales*.

**\$col** es el *color* con el que se dibujará la línea.

Respecto a los ángulos, CERO GRADOS coincide con el *cero trigonométrico* pero el sentido es *contrario*, es decir, el de las *agujas del reloj*.

Obviamente, para dibujar una circunferencia *basta* con hacer iguales los valores de **a** y de **b** y fijar los puntos inicial y final en  $0^\circ$  y  $360^\circ$  respectivamente.

### Dibujando sobre una imagen de fondo

PHP permite crear imágenes utilizando como fondo una preexistente. Para ello basta con **crear una variable** indicando el **path** y el **nombre de la imagen**, por ejemplo:

?>

[Ver img9.php](#)

## Dibujando un polígono sin relleno

```
<?
Header("Content-type: image/jpeg");

$esquinas=array(20,100,100,180,180,100,100,20);

$im = imagecreate(200,200);

$fondo=imagecolorallocate ($im, 0, 0, 200);
$blanco=imagecolorallocate ($im, 255, 255, 255);
$amarillo=imagecolorallocate ($im, 255, 255, 0);
Imagefill ($im, 0, 0, $fondo);
imagerectangle ($im, 10, 10, 190, 190, $blanco);
imagefilledrectangle ($im, 20, 20, 180, 180, $amarillo);

imagepolygon ($im, $esquinas, 4, $blanco);

Imagejpeg($im);

Imagedestroy($im);

?>
```

[Ver img10.php](#)

```
<?
Header("Content-type: image/png");

$esquinas=array(20,100,100,180,180,100,100,20);

$im = imagecreate(200,200);

$fondo=imagecolorallocate ($im, 0, 0, 200);
$blanco=imagecolorallocate ($im, 255, 255, 255);
$amarillo=imagecolorallocate ($im, 255, 255, 0);
Imagefill ($im, 0, 0, $fondo);
imagerectangle ($im, 10, 10, 190, 190, $blanco);
imagefilledrectangle ($im, 20, 20, 180, 180, $amarillo);

imagepolygon ($im, $esquinas, 4, $blanco);

Imagepng($im);

Imagedestroy($im);

?>
```

[Ver img11.php](#)

## Dibujando circunferencias, elipses y arcos

```
<?
Header("Content-type: image/jpeg");
$esquinas=array(20,100,100,180,180,100,100,20);
$im = imagecreate(200,200);
$fondo=imagecolorallocate ($im, 0, 0, 200);
$blanco=imagecolorallocate ($im, 255, 255, 255);
```

**\$b="../images/cruz.jpg"**

El formato de esta imagen **debe coincidir** con el de la que pretendemos construir.

Una vez definida esta variable bastará sustituir la instrucción de *creación de imagen*

**\$nom = imagecreate(x,y)**

por:

**\$nom= imagecreatefrompng (\$b)**

en el caso de imágenes **PNG** o por:

**\$nom= imagecreatefromjpeg (\$b)**

si se tratara de imágenes en formato **JPG**.

El resultado del cambio en el primero de los casos es **este** y aquí tienes también la imagen obtenida utilizando un fondo en formato **JPG**.

Una advertencia al respecto. Como puedes comprobar en los ejemplos, *el tamaño de la imagen es el mismo de la utilizada como fondo*.

## Guardando imágenes

Las imágenes que son creadas mediante la sintaxis anterior *no se guardan en servidor*.

Si se pretende guardarlas hay que modificar la sintaxis de las etiquetas:

**Imagepng(\$nombre)**

o

**Imagejpeg(\$nombre)**

añadiendo otro parámetro con el nombre y la extensión del fichero que vamos de guardar. Así por ejemplo:

**Imagepng(  
\$nombre, "mi\_imagen.png")**

o

**Imagejpeg(  
\$nombre, "mi\_imagen.jpg")**

guardarán en el servidor las imágenes creadas con los nombres **mi\_imagen.png** o **mi\_imagen.jpg**

```
$amarillo=imagecolorallocate ($im, 255, 255,0);
Imagefill ($im, 0, 0, $fondo);
imagerectangle ($im, 10, 10, 190, 190, $blanco);
imagefilledrectangle ($im, 20, 20, 180, 180, $amarillo);
imagepolygon ($im, $esquinas, 4, $blanco);

imagearc ($im, 100, 100, 160, 160, 0, 360, $fondo);
imagearc ($im, 100, 100, 160, 100, 0, 360, $rojo);

Imagejpeg($im);
Imagedestroy($im);
```

?>

[Ver img12.php](#)

<?

```
Header ("Content-type: image/png");
$esquinas=array(20,100,100,180,180,100,100,20);
$im = imagecreate(200,200);
$ffondo=imagecolorallocate ($im, 0, 0, 200);
$blanco=imagecolorallocate ($im, 255, 255, 255);
$amarillo=imagecolorallocate ($im, 255, 255, 0);
Imagefill ($im, 0, 0, $fondo);
imagerectangle ($im, 10, 10, 190, 190, $blanco);
imagefilledrectangle ($im, 20, 20, 180, 180, $amarillo);
imagepolygon ($im, $esquinas, 4, $blanco);

imagearc ($im, 100, 100, 160, 160, 0, 360, $fondo);
imagearc ($im, 100, 100, 160, 100, 0, 360, $rojo);

Imagepng ($im);
Imagedestroy ($im);
```

?>

[Ver img13.php](#)

## Utilizando imágenes dinámicas

En todos los ejemplos anteriores hemos podido visualizar las imágenes con sólo llamarlas desde el navegador, de la misma forma que podríamos visualizar cualquier otra imagen.

Pero las imágenes dinámicas pueden ser insertadas en una página web de la misma forma que cualquier otra imagen.

Aquí tienes un ejemplo donde se recogen en una página web todas las imágenes dinámicas creadas anteriormente.

[ejemplo103.php](#)

Si observas el **código fuente** verás que es exactamente el mismo que se utiliza para insertar una imagen *normal*, con la única diferencia de que aquí el **nombre de la imagen** será el mismo que el del **fichero PHP** que la genera.

[Anterior](#)

[Índice](#)

[Siguiente](#)



[Ver índice](#)

## Imágenes con líneas y textos



### Trazando segmentos

La función PHP que permite dibujar segmentos rectilíneos es la siguiente:

**imagineLine**  
(\$nom, \$x0, \$y0, \$x1, \$y1, \$col)

donde: **\$nom** es el nombre de la variable definida mediante *imagecreate*, **x0** e **y0** son las coordenadas de uno de los extremos; **x1** e **y1** son las coordenadas del otro extremo y **\$col** es la variable de color con el que será dibujada la línea.

### Colores de fondo

Tal como puedes observar en los primeros ejemplos de esta página, PHP utiliza como fondo de la imagen el primer color definido por la función:

*ImageColorAllocate*.

Esta opción de PHP nos obliga a definir dos colores distintos para conseguir la visibilidad de las líneas.

### Crear transparencias

Si deseamos que un color determinado se comporte como si fuera transparente debemos utilizar la función:

**imagecolorTransparent** (\$nom, \$col).

donde: **\$nom** es el nombre de la variable definida mediante *imagecreate*, y **\$color** es el color que pretendemos hacer transparente.

No olvides estos dos pequeños detalles:

- Si pretendes lograr un fondo transparente debes hacer transparente el primero de los colores definidos.
- Esta función sólo tiene sentido en imágenes **PNG** que son las únicas que permiten zonas transparentes. Recuerda que **JPG** no las permite.

### Insertando textos

Para insertar textos dentro de una imagen hemos de recurrir a una de estas funciones:

**imagechar** (\$im, n, x, y, \$txt, \$col)

## Dibujando segmentos

Aunque presentaremos únicamente el código fuente de una de las imágenes, para hacer la comprobación de las funciones y las diferencias de visualización insertaremos dos ejemplos, uno en formato PGN y otro en JPG.

Recuerda que las únicas diferencias entre ambos radican en utilizar: *Header("Content-type: image/png")* o *Header("Content-type: image/jpeg")* y en las funciones *Imagepng* ó *Imagejpeg*.

```
<?
Header("Content-type: image/png");
$im = imagecreate(200,200);
$fondo=ImageColorAllocate ($im,0,0,255);
$linea=ImageColorAllocate ($im,255,255,255);

imageline($im,0,0,200,200,$linea);

Imagepng($im);
Imagedestroy($im);
?>
```

[Ver img\\_a1.php](#)  
Formato PNG

[Ver img\\_a2.php](#)  
Formato JPG

## Fondos transparentes

```
<?
Header("Content-type: image/png");
$im = imagecreate(200,200);
$fondo=ImageColorAllocate ($im,0,0,255);
$linea=ImageColorAllocate ($im,255,0,0);
imagecolorTransparent ($im , $fondo);
imageline($im,0,0,200,200,$linea);
Imagepng($im);
Imagedestroy($im);
?>
```

[Ver img\\_a3.php](#)  
Formato PNG

[Ver img\\_a4.php](#)  
Formato JPG

Aquí tienes una página -con un color de fondo- en la que puedes visualizar las diferencias entre los dos formatos.

[ejemplo104.php](#)

## Insertando caracteres

```
<?
Header("Content-type: image/png");
$im = imagecreate(150,150);
$t1="Tamaño 1";
$t2="Tamaño 2";
$t3="Tamaño 3";
$t4="Tamaño 4";
$t5="Tamaño 5";

$fondo=imagecolorallocate ($im, 0, 0, 200);
$amarillo=imagecolorallocate ($im, 255, 255, 0);
imagechar ($im, 1, 0, 0, $t1, $amarillo);
imagechar ($im, 2, 20, 20, $t2, $amarillo);
imagechar ($im, 3, 40, 40, $t2, $amarillo);
```

Requiere que la variable **\$txt** contenga una **cadena** definida con anterioridad. Mediante esta función se inserta el *primer carácter de la cadena con orientación horizontal*.

Los parámetros de la función son los siguientes:

**\$nom** el nombre de la variable con la que fue definida por *imagecreate*

**n** es un número comprendido entre **UNO** y **CINCO** que asigna el tamaño de la letra de menor a mayor.

**x** e **y** son las coordenadas del punto donde se colocará la **esquina superior izquierda** del carácter a representar.

**\$txt** es la cadena de texto de la que se extraerá el *primer carácter*, el único que se verá en la imagen.

**\$col** es el color del carácter a representar.

**imagecharup (\$im, n, x, y, \$txt, \$col)**

Su funcionamiento es similar al de la función anterior, con la única diferencia de que *inserta el carácter con orientación vertical*.

Las *coordenadas de inserción* también se corresponden con las de la **esquina superior izquierda** del carácter pero, recuerda que ahora estará girado y que, por lo tanto, ese punto coincidirá con parte *inferior izquierda* de la imagen del carácter.

**imagestring (\$im, n, x, y, \$txt, \$col)**

Esta función se comporta de forma similar a **imagechar**. La única diferencia entre ambas es que mientras **imagechar** inserta sólo el *primer carácter*, en el caso de **imagestring** se inserta la *cadena completa*.

Los parámetros de ambas funciones son los mismos.

Si la cadena **desborda** los límites de la imagen sólo se visualizará la parte de la misma contenida dentro de éstos.

**imagestringup (\$im, n, x, y, \$txt, \$col)**

Inserta una *cadena completa con orientación vertical* y sus parámetros son idénticos a los comentados cuando nos hemos referido a **imagecharup**.

### Tipos de letra

Todas las funciones anteriores utilizan **siempre** la **fuente predefinida** por PHP y sólo permiten los **cinco tamaños** que hemos

```
imagechar ($im, 4, 60, 60, $t2, $amarillo);
imagechar ($im, 5, 80, 80, $t2, $amarillo);
    Imagepng($im);
    imagedestroy($im);
?>
```

[Ver img17.php](#)  
Formato PNG

[Ver img18.php](#)  
Formato JPG

```
<?
Header("Content-type: image/png");
$im = imagecreate(150,150);
$t1="Tamaño 1";
$t2="Tamaño 2";
$t3="Tamaño 3";
$t4="Tamaño 4";
$t5="Tamaño 5";
$fondo=imagecolorallocate ($im, 0, 0, 200);
$amarillo=imagecolorallocate ($im, 255, 255,0);
imagecharup ($im, 1, 10, 10, $t1, $amarillo);
imagecharup ($im, 2, 20, 20, $t2, $amarillo);
imagecharup ($im, 3, 40, 40, $t2, $amarillo);
imagecharup ($im, 4, 60, 60, $t2, $amarillo);
imagecharup ($im, 5, 80, 80, $t2, $amarillo);
    Imagepng($im);
    imagedestroy($im);
?>
```

[Ver img19.php](#)  
Formato PNG

[Ver img20.php](#)  
Formato JPG

```
<?
Header("Content-type: image/png");
$im = imagecreate(150,150);
$t1="Tamaño 1";
$t2="Tamaño 2";
$t3="Tamaño 3";
$t4="Tamaño 4";
$t5="Tamaño 5";
$fondo=imagecolorallocate ($im, 0, 0, 200);
$amarillo=imagecolorallocate ($im, 255, 255,0);
imagestring ($im, 1, 10, 20, $t1, $amarillo);
imagestring ($im, 2, 10, 40, $t2, $amarillo);
imagestring ($im, 3, 10, 60, $t3, $amarillo);
imagestring ($im, 4, 10, 80, $t4, $amarillo);
imagestring ($im, 5, 10, 100, $t5, $amarillo);
    Imagepng($im);
    imagedestroy($im);
?>
```

[Ver img21.php](#)  
Formato PNG

[Ver img22.php](#)  
Formato JPG

```
<?
Header("Content-type: image/png");
$im = imagecreate(150,150);
$t1="Tamaño 1";
$t2="Tamaño 2";
$t3="Tamaño 3";
$t4="Tamaño 4";
$t5="Tamaño 5";
$fondo=imagecolorallocate ($im, 0, 0, 200);
$amarillo=imagecolorallocate ($im, 255, 255,0);
imagestringup ($im, 1, 10, 100, $t1, $amarillo);
imagestringup ($im, 2, 20, 100, $t2, $amarillo);
imagestringup ($im, 3, 40, 100, $t3, $amarillo);
imagestringup ($im, 4, 60, 100, $t4, $amarillo);
imagestringup ($im, 5, 80, 100, $t5, $amarillo);
```

podido ver en los ejemplos.

Afortunadamente —lo veremos en la página siguiente— PHP también permite usar **fuentes TrueType** y aplicarlas en la creación de imágenes.

```
?> Imagepng($im);  
imagedestroy($im);
```

[Ver img23.php  
Formato PNG](#)

[Ver img24.php  
Formato JPG](#)

Anterior



Índice



Siguiente





[Ver índice](#)

# Manejando fuentes



## Utilizando fuentes TrueType

Si has leído los comentarios de la página anterior recordarás que para usar estas funciones es preciso que estén instaladas las librerías **FreeType** y que, además, conozcamos el *path* de directorio que contiene las fuentes TrueType.

## Instalación de fuentes

Hemos creado un *subdirectorio* llamado **fuentes** y lo hemos incluido en directorio donde están alojadas estas páginas.

En ese subdirectorrio hemos incluido dos fuentes TrueType manteniendo en una de ellas el nombre original (*arial.ttf*) y renombrado la otra como *fuente2.ttf*.

Como podrás comprobar en los ejemplos, no hay problema alguno por el hecho de renombrar las fuentes.

## Escribiendo con fuentes TrueType

La función PHP que nos permite insertar este tipo de textos en imágenes dinámicas es la siguiente:

**Imagettfttext(\$nom, tam, ang, x, y, \$col, \$fuente, \$txt)**

donde:

**\$nom** es, como siempre, el nombre de la imagen.

**tam** es un **número entero** que indica el **el tamaño de la fuente**.

**ang** es el **giro** expresado en **grados sexagesimales** que pretendemos que tenga la cadena de texto. Si **ang=0** el texto aparecerá escrito en horizontal.

**x** e **y** son las coordenadas del **punto de inicio** de la inserción del texto. Ese punto coincide con la **esquina inferior izquierda** del rectángulo imaginario que contiene el texto.

**\$col** es el color a utilizar en el texto.

**\$fuente** es una cadena de texto que contiene el **path** y el nombre de la fuente. Observa los ejemplos.

**\$txt** es el nombre de la variable que contiene el **texto** a insertar.

## Dibujando segmentos

Aunque presentaremos únicamente el código fuente de una de las imágenes, para hacer la comprobación de las funciones y las diferencias de visualización insertaremos dos ejemplos. Uno en formato PGN y otro en JPG.

Recuerda que las únicas diferencias entre ambos radican en utilizar: *Header("Content-type: image/png")* o *Header("Content-type: image/jpeg")* y en las funciones *Imagepng* ó *Imagejpeg*.

## Texto TrueType horizontal

```
<?
Header("Content-type: image/png");
$im = imagecreate(400,300);
$fondo=imagecolorallocate ($im, 255, 255, 210);
$rojo=imagecolorallocate ($im, 255, 0, 0);

$texto="PHP";

Imagettfttext($im, 40, 0, 100, 270, $rojo,
        "./fuentes/fuente2.ttf", $texto);

Imagepng($im);
imagedestroy($im);
?>
```

[Ver img25.php](#)  
Formato PNG

[Ver img26.php](#)  
Formato JPG

## Texto TrueType girado

```
<?
Header("Content-type: image/png");
$im = imagecreate(400,300);
$fondo=imagecolorallocate ($im, 255, 255, 210);
$rojo=imagecolorallocate ($im, 255, 0, 0);

$texto="Me gusta PHP";

Imagettfttext($im, 40, 30, 100, 270, $rojo,
        "./fuentes/fuente2.ttf", $texto);

Imagepng($im);
imagedestroy($im);
?>
```

[Ver img27.php](#)  
Formato PNG

[Ver img28.php](#)  
Formato JPG

## Centrando textos

Aquí tienes un ejemplo donde utilizando **ImageTTFBox** e **ImageTTFTText** se puede **centrar un texto** -tanto si es horizontal como si está girado- con relación a un punto.

En este ejemplo, el punto de referencia para el centrado es **(200,150)** que es el centro de la imagen.

Las coordenadas de **ImageTTFTText**, como puedes ver, están calculadas usando las coordenadas de ese punto de referencia y los valores del array generado por **ImageTTFBox**

- Si la variable que contiene el array generado por **ImageTTFBox** se llama **\$pepa**, las coordenadas

## Colocando textos

PHP dispone de una función que permite determinar las **dimensiones** de una **caja de texto** (el rectángulo imaginario que rodea el texto).

**\$rec= ImageTTFBBox (tam, ang, \$fuente, \$txt)**

donde: **tam** es el **tamaño de la fuente** a utilizar.

**ang** es el **ángulo de rotación** del texto que tendría valor **cero** en el caso de orientación horizontal.

**\$fuente** es el **path y nombre de la fuente** a utilizar.

**\$txt** es el nombre de la **variable** que contiene el **texto** a incluir.

La variable **\$rec** recoge un **array escalar** cuyos valores son las coordenadas de las cuatro esquinas de la **caja de texto**.

Los índices correspondientes a cada uno de los elementos de ese array son los siguientes:

### Inferior izquierdo

Sus coordenadas son:  
**\$rec[0],\$rec[1]**

### Inferior derecho

Sus coordenadas son:  
**\$rec[2],\$rec[3]**

### Superior derecho

Sus coordenadas son:  
**\$rec[4],\$rec[5]**

### Superior izquierdo

Sus coordenadas son:  
**\$rec[6],\$rec[7]**

Respecto a estas coordenadas, habremos de tener en cuenta lo siguiente:

Las correspondientes al vértice **inferior izquierdo** son siempre **(0,0)**.

Los puntos situados *por encima* del **(0,0)** tienen **ordenada negativa**.

Las **abscisas** de los puntos situados a la **izquierda** del **(0,0)** son **negativas**.

del centro del **rectángulo imaginario** son **\$pepa[4]/2** y **\$pepa[5]/2**

Partiendo de esos valores, si queremos **centrar el texto** sobre un punto de la imagen cuyas coordenadas son **(X,Y)** nos basta con escribir como **parámetros** de la función **ImageTTFTText** los siguientes:

**Abscisa= X - \$pepa[4]/2**

**Abscisa= Y - \$pepa[5]/2**

Este procedimiento es válido tanto para textos horizontales como para textos girados.

Aquí lo tienes:

```
<?
    Header("Content-type: image/png");

    $im = imagecreate(400,300);
    $fondo=imagecolorallocate ($im, 255, 255, 210);
    $gris=imagecolorallocate ($im, 160, 160,160);
    $rojo=imagecolorallocate ($im, 255, 0, 0);

    $texto="El mundo del PHP";
    $textol="lleno de posibilidades";

    $marco= ImageTTFBBox (40, 0, "./fuentes/arial.ttf", $texto);

    Imagettfttext($im, 40, 0, 200-$marco[4]/2, 150-$marco[5]/2, $gris,
                  "./fuentes/arial.ttf", $texteo);

    $marcol= ImageTTFBBox (30, 30, "./fuentes/fuente2.ttf", $textol);

    Imagettfttext($im, 30, 30, 200-$marcol[4]/2, 150-$marcol[5]/2, $rojo,
                  "./fuentes/fuente2.ttf", $textol);

    Imagepng($im);
    imagedestroy($im);
?>
```

[Ver img29.php](#)  
Formato PNG

[Ver img30.php](#)  
Formato JPG

## Un ejemplo resumen

Aquí tienes un ejemplo bastante completo de generación de imágenes dinámicas.

[Ver ejemplo105.php](#)

[Ver código fuente](#)

### ¡Cuidado!

En servidores UNIX y LINUX la forma en la que indicamos el path del directorio que contiene las fuentes varía sustancialmente respecto a la que hemos utilizado en estos ejemplos.

En aquellos, además de *invertir* los separadores, se requiere indicar siempre rutas absolutas, con lo cual habría que hacer las modificaciones oportunas si se tratara de publicar en un hosting que utilice alguno de estos sistemas operativos.

Anterior



Índice



Siguiente



[Ver índice](#)

## Diagramas de sectores



### Diagramas de sectores

Esta posibilidad gráfica de tratamiento de información estadística la proporciona la función ***imagefilledarc()*** que requiere -por el orden que se indica- los parámetros:

**Xc** y **Yc** que son las coordenadas del centro de la elipse cuyo arco (o porción) tratamos de representar.

**A** y **B** que son las longitudes (expresadas en pixels) de los ejes horizontal y vertical de la elipse. Si ambos fueran iguales el resultado sería circular.

**Pi** y **Pf** son las posiciones (en grados sexagesimales) de los radios (inicial y final) que delimitan el sector que se trata de representar.

Los *cero grados* coinciden con el semieje horizontal positivo y el sentido del recorrido angular es el de las *agujas del reloj*.

**\$color** es la variable -ha de ser definida previamente mediante ***imagecolorallocate***- que indica el color que ha de utilizarse en el gráfico.

El último parámetro es un constante PHP que puede tomar uno de los siguientes valores:

**IMG\_ARC\_PIE**  
**IMG\_ARC\_NOFILL**  
**IMG\_ARC\_EDGED**  
**IMG\_ARC\_CHORD**

Con la primera de las constantes dibuja el *sector de elipse* delimitado por los radios indicados relleno con el color especificado.

El segundo (**IM\_ARC\_NOFILL**) únicamente dibuja la porción de arco, pero no incluye los radios en el dibujo.

La opción **IMG\_ARC\_EDGED** se comporta de forma idéntica a **IMG\_ARC\_PIE** cuando se utiliza de forma aislada aunque tiene una opción muy interesante que veremos un poco más abajo.

Con **IMG\_ARC\_CHORD** el resultado es un triángulo -relleno con el color indicado- formado por los dos radios y la *cuerda* correspondiente al arco que delimitan.

### Diagramas de sectores

```
<?
$im = imagecreate (400, 400);
$fondo = imagecolorallocate($im, 226, 226, 226);
$col1=imagecolorallocate($im,255,255,0);
$col2=imagecolorallocate($im,255,0,0);
imagefilledarc($im, 200, 200, 350, 300, 20, 240, $col1, IMG_ARC_PIE);
imagefilledarc($im, 200, 200, 350, 300, 10, 150, $col2, IMG_ARC_NOFILL);
header('Content-type: image/png');
imagepng($im);
imagedestroy($im);
?>
```

[Ver ejemplo.jpg](#)[Ver ejemplo.png](#)[Ver ejemplo.gif](#)

```
<?
$im = imagecreate (400, 400);
$fondo = imagecolorallocate($im, 226, 226, 226);
$col1=imagecolorallocate($im,255,255,0);
$col2=imagecolorallocate($im,255,0,0);
imagefilledarc($im, 200, 200, 350, 300, 20, 240, $col1, IMG_ARC_EDGED);
imagefilledarc($im, 200, 200, 350, 300, 10, 150, $col2, IMG_ARC_NOFILL);
header('Content-type: image/png');
imagepng($im);
imagedestroy($im);
?>
```

[Ver ejemplo.jpg](#)[Ver ejemplo.png](#)[Ver ejemplo.gif](#)

```
<?
$im = imagecreate (400, 400);
$fondo = imagecolorallocate($im, 226, 226, 226);
$color1=imagecolorallocate($im,255,0,0);
imagefilledarc ($im, 200, 200, 350, 300, 20, 240, $color1,
                IMG_ARC_NOFILL|IMG_ARC_EDGED);
header('Content-type: image/gif');
imagegif($im);
imagedestroy($im);
?>
```

[Ver ejemplo.jpg](#)[Ver ejemplo.png](#)[Ver ejemplo.gif](#)

```
<?
$im = imagecreate (400, 400);
$fondo = imagecolorallocate($im, 226, 226, 226);
$color1=imagecolorallocate($im,255,0,0);
imagefilledarc ($im, 200, 200, 350, 300, 50, 200, $color1, IMG_ARC_CHORD);
header('Content-type: image/gif');
imagegif($im);
imagedestroy($im);
?>
```

[Ver ejemplo.jpg](#)[Ver ejemplo.png](#)[Ver ejemplo.gif](#)

### Combinar dos constantes

Cuando utilizamos como último parámetro de la función `imagefilledarc()` una expresión del tipo: `IMG_ARC_NOFILL|IMG_ARC_EDGED` (fíjate en el signo `|` que separa ambas constantes) lo que obtenemos es la representación gráfica del contorno del sector (incluye los radios que lo delimitan). Mediante esta opción -con dos llamadas a la función- tenemos la posibilidad de representar el sector con un color de relleno (usando `IMG_ARC_PIE`) y, luego, superponerle un contorno de distinto color. Puedes verlo en los ejemplos.

## Efecto tridimensional

Tal como puedes ver en los ejemplos, resulta fácil lograr un efecto tridimensional en el dibujo de los sectores. Basta con crear un bucle que dibuje arcos sucesivos (separados verticalmente por un pixel) y posteriormente superponer un sector relleno con un color distinto.

```
<?
$im = imagecreate (400, 400);
$fondo = imagecolorallocate($im, 226, 226, 226);
$color1=imagecolorallocate($im,200,0,0);
$color2=imagecolorallocate($im,255,0,0);
$color3=imagecolorallocate($im,255,255,255);
for($i=200;$i<225;$i++) {
imagefilledarc($im, 200, $i, 370, 270, 50, 330, $color1,
IMG_ARC_NOFILL|IMG_ARC_EDGED);
}
imagefilledarc($im, 200, 200, 370, 270, 50, 330, $color2, IMG_ARC_EDGED);
imagefilledarc($im, 200,200, 370, 270, 50, 330, $color3,
IMG_ARC_NOFILL|IMG_ARC_EDGED);

header('Content-type: image/gif');
imagegif($im);
imagedestroy($im);
?>
```

[Ver ejemplo .jpg](#)

[Ver ejemplo .png](#)

[Ver ejemplo .gif](#)

## Ejercicio nº 33

Diseña un formulario mediante el cual puedas transferir 4 valores numéricos a un script que cree un diagrama de cuatro sectores cuyos ángulos centrales sumen  $360^\circ$  y sean proporcionales a los valores numéricos transferidos.

[Anterior](#)

[Índice](#)

[Siguiente](#)

## Lectura de imágenes externas

La visualización de imágenes no presenta ningún problema —lo hacemos habitualmente mediante etiquetas HTML— cuando se encuentran en el espacio del servidor, bien sea propio o ajeno.

El problema puede surgir cuando tratemos de almacenar esas imágenes fuera del *root* del servidor (una forma de impedir la accesibilidad desde otras webs) y eso puede conseguirse mediante las funciones que veremos en este capítulo.

El primer paso será establecer la ruta y el nombre de la imagen. Al margen tienes comentados algunos detalles al respecto.

El paso siguiente será extraer el formato de la imagen. Lo hice leyendo la parte de la cadena comprendida entre el último punto (.) y el final de la cadena que contiene el nombre de la imagen.

Mediante el **switch** elijo las instrucciones para cada tipo de imagen que son similares pero con *matices* según del formato.

La visualización de la imagen contiene tres instrucciones: *Header*, *imagecreatefrom* e *imageXXX*.

En **Header** hay que incluir un **Content-type** acorde con el tipo de imagen. Puede ser, entre otros valores: **image/jpeg**, **image/png** ó **image/gif** según la imagen tenga formato: **.jpg**, **.png** ó **.gif**.

Con idéntico criterio, la función que **crea** la imagen ha de ser una de estas:

\$f=imagecreatefromjpeg(\$i)  
ó  
\$f=imagecreatefrompng(\$i)  
ó  
\$f=imagecreatefromgif(\$i)

siendo **\$i** la variable que recoge el *nombre* y el *path* de la imagen original y **\$f** la variable que contiene el resultado de la ejecución de esta función.

La función *image* (la que permite visualizar la nueva imagen) también puede tener una de las tres variantes que ya conocemos de ejemplos anteriores. Pueden ser:  
**imagedine(\$f)**    **imagedona(\$f)**    ó

## Lectura de imágenes externas

Uno de los problemas que puede presentarse es la forma de indicar dónde están la imagen a visualizar. Utilizaremos la variable predefinida **\$\_SERVER['DOCUMENT\_ROOT']** para ubicarnos en el directorio root del servidor Apache. De esta forma no tendremos problemas de visualización si es que fuere el directorio de instalación de servidor web.

```
<?
# indicar la ruta de la imagen
$original=$_SERVER['DOCUMENT_ROOT']."/cursophp/images/caballos.jpg";;

# extraer el tipo de imagen según su la extensión del fichero
for($i=strlen($original)-1;$i>0;$i--) {
    if (substr($original,$i,1)==".") {
        $tipo=substr($original,$i+1);
        break;
    }
}

# las diferentes opciones dependiendo del formato de la imagen
switch($tipo){
    case "jpg":
        Header("Content-type:image/jpeg");
        $nueva=imagecreatefromjpeg($original);
        imagejpeg($nueva);
        break;

    case "png":
        Header("Content-type:image/png");
        $nueva=imagecreatefrompng($original);
        imagepng($nueva);
        break;

    case "gif":
        Header("Content-type:image/gif");
        $nueva=imagecreatefromgif($original);
        imagegif($nueva);
        break;
}
ImageDestroy();
?>
```

[Ver ejemplo .jpg](#)[Ver ejemplo .png](#)[Ver ejemplo .gif](#)

### ¡Cuidado!

Observa que en las imágenes en formato **png** se visualizan con deficiencias en los bordes de las áreas transparentes.

Con el método que vemos a continuación ese problema se reduce considerablemente.

## Lectura y redimensionado de imágenes externas

```
<?
# indicar la ruta de la imagen
$original=$_SERVER['DOCUMENT_ROOT']."/cursophp/images/caballos.jpg";
```

**imagegif(\$f)**, donde **\$f** es la variable que recoge el resultado de la función anterior.

## Redimensionado de imágenes externas

Tampoco parece *ninguna* utilidad. ¿Verdad? A fin de cuentas con etiquetas HTML podemos asignar el ancho y el alto de una imagen. Pero... ya verás como no es tan trivial esta opción.

El proceso es el siguiente:

1º.- Determinamos cuales son las dimensiones de la imagen externa que vamos a utilizar. Para ello, usaremos la función:

**\$dim=getimagesize(\$r)**

donde **\$r** es la variable que contiene el path y nombre del fichero que contiene la imagen y **\$dim** es un **array escalar** que contiene las dimensiones de la imagen analizada.

El elemento del array **\$dim[0]** contiene el ancho y **\$dim[1]** el alto, ambos expresados en **pixels**.

Cuidado...!

Esta función solo funciona cuando se trata de imágenes externas. Para determinar las dimensiones de imágenes generadas por PHP tendrás que utilizar otra distinta. De nada;-)

2º.- Creamos una **copia** de la imagen original por medio de la función **imagecreate** adecuada al **tipo** de imagen que deseamos importar. Es exactamente lo que hemos visto en el párrafo anterior.

3º.- Creamos una **nueva imagen** -podemos trabajar con varias imágenes, en eso no hay problema- mediante la función:

**\$d=imagecreatetruecolor(x,y)**

dónde **\$d** es el identificador de la imagen, y **x** e **y** son las dimensiones de esta nueva imagen.

Dado que esta imagen va a ser el **soprote** -una imagen en color verdadero (de ahí lo de *truecolor*) con fondo negro, algo muy similar al papel fotográfico que se usa en los laboratorios- sobre el que se va a *impresionar* esa especie de *negativo* que es la *imagen original* es necesario que sus dimensiones sean las deseadas para la imagen resultante.

4º.-Ahora **toca positivar** la nueva foto. Para hacerlo disponemos de la función

**imagecopyresampled()** que debe incluir -dentro del paréntesis- un

```

for($i=strlen($original)-1;$i>0;$i--) {
    if (substr($original,$i,1)==".") {
        $tipo=substr($original,$i+1);
        break;
    }
}
# dimesiones del original
$stamano=getimagesize($original);
$orig_Ancho = $stamano[0];
$orig_Alto = $stamano[1];
# factores de ampliación, distintos para provocar una distorsión
# en la imagen resultante
$ampliacion_X=2;
$ampliacion_Y=1.5;
# dimesiones de la imagen resultante. Vamos a dejarla a sangre
# (sin márgenes en blanco) y vamos a reproducir el original
# sin reencuadrar así que las esquinas superiores izquierdas de
# ambas imágenes estarán en 0,0.
$resultado_Ancho=$orig_Ancho*$ampliacion_X;
$resultado_Alto= $orig_Alto*$ampliacion_Y;
#creamos una imagen a partir de la original. Debemos elegir
#la función adecuada al tipo de imagen original
switch($tipo){
    case "jpg":
        $importada=imagecreatefromjpeg($original);
        break;
    case "png":
        $importada=imagecreatefrompng($original);
        break;
    case "gif":
        $importada=imagecreatefromgif($original);
        break;
}
# insertamos la cabecera de la nueva imagen
Header("Content-type:image/jpeg");
#creamos una imagen nueva en color verdadero
$im_base=imagecreatetruecolor($resultado_Ancho,$resultado_Alto);
#aplicamos un color de fondo a la nueva imagen
#para poder visualizar que incluye la transparencia del png o del gif
if($tipo=="png" OR $tipo=="gif"){
    $fondo=imagecolorAllocate($im_base,255,255,200);
    imagefill($im_base,0,0,$fondo);
}
#superponemos la imagen importada sobre la que acabamos de crear
imagecopyresampled($im_base,$importada,0,0,0,
                    $resultado_Ancho, $resultado_Alto,
                    $orig_Ancho,$orig_Alto);
# visualizamos la imagen resultante
imagejpeg($im_base);
ImageDestroy();
?>
```

[Ver ejemplo .jpg](#)

[Ver ejemplo .png](#)

[Ver ejemplo .gif](#)

Observa que –tanto en el ejemplo anterior como en el siguiente– solo hemos utilizado la extensión de la imagen original para elegir la función **imagecreatefrom....**. En el Header hemos puesto **image/jpeg** y, como es obvio, hemos utilizado la función asociada a este formato (**imagejpeg**). Si sustituimos **ambos valores** por los correspondientes a otro formato (gif, png) obtendríamos resultados similares.

## Recortar imágenes externas

```

<?
# obtener la imagen
$original=$_SERVER['DOCUMENT_ROOT']."/cursophp/images/aviones4.jpg";
for($i=strlen($original)-1;$i>0;$i--) {
    if (substr($original,$i,1)==".") {
        $tipo=substr($original,$i+1);
        break;
    }
}
# tamaño del original
```

este orden):

**\$d** que es el identificador de la imagen destino, es decir el papel fotográfico que hemos *creado* en el paso anterior.

**\$f** que es el identificador de la imagen original (*negativo*) obtenido en el punto 2º.

**X<sub>d</sub>** e **Y<sub>d</sub>** son las coordenadas de un punto situado en la *esquina superior izquierda del papel* a partir del que queremos que se *impresione la fotografía*. Si queremos una *foto a sangre* pondremos **0,0** y, si quieres dejar **márgenes en blanco**, habrá que poner los anchos de esos márgenes (izquierdo y superior) respectivamente.

**X<sub>f</sub>** e **Y<sub>f</sub>** nos servirán para *reencuadrar* la foto original *recortando* por la izquierda y por arriba, respectivamente, los anchos que se indiquen aquí en pixels.

**D<sub>x</sub>** e **D<sub>y</sub>** indican el ancho y el alto (por este orden) que va a tener la mancha de imagen en el positivo. Ten en cuenta que *no puedes salirte del papel* así que esos valores sumados con los márgenes (izquierdo y superior) no podrán ser mayores que las dimensiones que has elegido para el *papel fotográfico* en el punto 2º.

**F<sub>x</sub>** e **F<sub>y</sub>** indican el ancho y el alto de la porción del *original* que tratamos de reproducir. Sumados con **X<sub>f</sub>** e **Y<sub>f</sub>** no pueden exceder el tamaño del *negativo*.

Con estos parámetros la función ya se encarga de redimensionar la imagen (incluso distorsionarla si no hay proporcionalidad entre los anchos y altos del *original* y del *soporte*).

```
$tamano=getimagesize($original);
$orig_Ancho = $tamano[0];
$orig_Alto = $tamano[1];
# estableceremos un margen en blanco alrededor de la imagen de 5 pixels
# igual por los cuatro lados
$margen=10;
# establecemos recortes para reencuadrar la imagen
$recorte_izq=50;
$recorte_sup=80;
$recorte_der=40;
$recorte_inf=60;
# calculamos las dimensiones para utilizar como parámetros
# en la función imagecopyresampled
# ancho y alto original recortado
$Ancho_recortado=$orig_Ancho-$recorte_izq-$recorte_der;
$Alto_recortado=$orig_Alto-$recorte_sup-$recorte_inf;
# factores de ampliación en este caso iguales
# sin distorsión de imagen
$ampliacion_X=1;
$ampliacion_Y=1;
# dimensiones del soporte
$papel_Ancho=$Ancho_recortado*$ampliacion_X+ 2*$margen;
$papel_Alto=$Alto_recortado*$ampliacion_Y+2*$margen;
# dimensiones de la mancha de imagen al positivar
# hay que quitar los márgenes
$resultado_Ancho=$papel_Ancho -2*$margen;
$resultado_Alto=$papel_Alto -2*$margen;
switch($tipo){
    case "jpg":
        $importada=imagecreatefromjpeg($original);
        break;
    case "png":
        $importada=imagecreatefrompng($original);
        break;
    case "gif":
        $importada=imagecreatefromgif($original);
        break;
}
Header("Content-type:image/jpeg");
$im_base=imagecreatetruecolor($papel_Ancho,$papel_Alto);
$fondo=imagecolorAllocate($im_base,255,255,200);
imagefill($im_base,0,0,$fondo);
imagecopyresampled($im_base,$importada,$margen,$margen,
    $recorte_izq,$recorte_sup,
    $resultado_Ancho,$resultado_Alto,
    $Ancho_recortado,$Alto_recortado);
imagejpeg($im_base);
ImageDestroy();
?>
```

Ver imágenes original y resultante

Anterior

Índice

Siguiente



[Ver índice](#)

# Superponer, rotar y dar transparencia a imágenes



## Colores transparentes

PHP permite crear colores con determinado color de transparencia. Para ello se utiliza la función:

**ImageColorAllocateAlpha()** que debe contener dentro del paréntesis los siguientes parámetros (separados por comas):

**\$im** que es el identificador de la imagen que ha sido creada previamente.

**R,G,B** que son valores numéricos (o variables) que contienen -en una escala de 0 a 255- los la intensidad lumínosa de cada uno de los tres colores primarios (rojo, verde y azul).

**trans** es un valor numérico (comprendido entre 0 y 127) que indica el grado de transparencia de la tinta. El valor 0 indica *opacidad total*, mientras que 127 establece la transparencia total de ese color.

En el ejemplo que tienes a la derecha hemos incluido dos escalas de transparencias superpuestas a intervalos del 10% (desde 0 hasta 100%) transformados a la escala 0-127.

El orden de superposición -similar a las capas de otros programas gráficos- se corresponde con el orden de las instrucciones de creación. Los resultados de las últimas funciones se superponen siempre a los obtenidos como consecuencia de la ejecución de las anteriores.

## Transparencia en imágenes externas

Mediante la utilización de la función **imagecopymerge()** es posible ajustar el grado de transparencia de una imagen externa.

La función **imagecopymerge()** requiere que se incluyan (dentro del paréntesis y separados por comas y por el orden en el que los incluimos) los siguientes parámetros:

**\$destino** que es el identificador de la imagen sobre la que se va a colocar la transparencia. Como es lógico, deberá haber sido creada antes de incluir la función.

**\$origen** es el identificador de la imagen que pretendemos incluir con un determinado grado de

## Superposición de áreas transparentes

```
<?
/* Creamos una imagen en color verdadero, le aplicamos un color
   de fondo (para evitar el negro por defecto) y creamos un nuevo
   color que utilizaremos para los bordes de rectángulos posteriores*/
Header("Content-type:image/jpeg");
$im_base=imagecreatetruecolor(610,140);
$fondo=imagecolorAllocate($im_base,255,255,200);
$negro=imagecolorAllocate($im_base,0,0,0);
imagefill($im_base,0,0,$fondo);
# definimos las componentes de un nuevo color
$R=255; $G=00; $B=00;
/* vamos a construir una escala de transparencias
   de 0 a 10 que correspondería con valores de transparencia
   de 0% al 100%.
   Crearemos un bucle que dibuje rectángulos llenos
   con el color definido en la variable trans que irá aplicando
   al color básico los diferentes grados de transparencia
   y le pondremos un contorno negro para encuadrarlos*/
for($i=0;$i<=10;$i++){
    $trans=ImageColorAllocateAlpha($im_base,$R,$G,$B,(int)($i*127/10));
    imagefilledrectangle($im_base, 10+55*$i, 20, 50+55*$i, 80, $trans);
    imagerectangle($im_base, 10+55*$i, 20, 50+55*$i, 80, $negro);
}
# creamos un nuevo color y repetimos el proceso con nuevos rectángulos
# superpuestos a los anteriores y con las transparencias en sentido opuesto
# es decir, de 100 a 0%
$R=0; $G=0; $B=255;
for($i=0;$i<=10;$i++){
    $trans=ImageColorAllocateAlpha($im_base,$R,$G,$B,127-(int)($i*127/10));
    imagefilledrectangle($im_base, 10+55*$i, 60, 50+55*$i, 120, $trans);
    imagerectangle($im_base, 10+55*$i, 60, 50+55*$i, 120, $negro);
}
# visualizamos el resultado
imagejpeg($im_base);
ImageDestroy();
?>
```

[Ver ejemplo.jpg](#)

[Ver ejemplo.png](#)

[Ver ejemplo.gif](#)

## Transparencia de imágenes externas

```
<?
# obtener la imagen
$original=$_SERVER['DOCUMENT_ROOT']."/cursophp/images/aviones4.jpg";
# buscar el formato de la imagen mediante su extensión
for($i=strlen($original)-1;$i>0;$i--){
    if (substr($original,$i,1)==".") {
        $tipo=substr($original,$i+1);
        break;
    }
}
# tamaño del original extraido del array devuelto por getimagesize
$stamano=getimagesize($original);
$orig_Ancho = $stamano[0];
$orig_Alto = $stamano[1];
# estableceremos un margen en blanco alrededor de la imagen de 10 pixels
# igual por los cuatro lados
$borde=10;
$Ancho=$orig_Ancho+2*$borde;
$Alto=$orig_Alto+2*$borde;
# creamos la imagen según el formato original
switch($tipo){
```

**X<sub>d</sub>** e **Y<sub>d</sub>** son las coordenadas de un punto situado en la *esquina superior izquierda de la imagen destino* a partir del que queremos que se *impresione la nueva imagen*. Si queremos una *imagen a sangre* pondremos **0,0** y, si quieres dejar **márgenes**, habrá que poner los anchos de esos márgenes (izquierdo y superior) respectivamente.

**X<sub>f</sub>** e **Y<sub>f</sub>** nos servirán para *reencuadrar* la foto original *recortando* por la izquierda y por arriba, respectivamente, los anchos que se indiquen aquí en pixels.

**D<sub>x</sub>** e **D<sub>y</sub>** indican el ancho y el alto (por este orden) que va a tener la mancha de imagen en el positivo. Ten en cuenta que *no puedes salirte del papel* así que esos valores sumados con los márgenes (izquierdo y superior) no podrán ser mayores que las dimensiones que has elegido para la *imagen destino*.

**opacidad** es el último de los parámetros de la función al que puede asignársele un valor comprendido entre **0** y **100**.

Representa el porcentaje de opacidad de la imagen superpuesta. Con un valor 100 sería totalmente opaca y con 0 la transparencia sería total.

### La función **imagecopy**

Mediante esta función se puede *copiar* sobre una imagen una parte de otra. Permite extraer porciones de imágenes con su tamaño original sin que permita ampliarlas ni reducirlas. Su sintaxis es la siguiente:

**imagecopy(\$d,\$o,\$x,\$y,\$X,\$Y,\$A,\$H)**  
donde:

**\$d** el identificador de la imagen destino, **\$o** el identificador de la imagen original, **\$x** y **\$y** las coordenadas donde se posicionará –en la imagen destino– la esquina superior izquierda de la porción copiada.

**\$X** y **\$Y** son los anchos de los *recortes* izquierdo y superior de la imagen a copiar y **\$A** y **\$H** el ancho y el alto del área de imagen que pretendemos copiar.

### Rotación de imágenes

Mediante la función:

**imagerotate(\$im,ang,\$fondo)** es posible presentar imágenes rotadas.

El parámetro **\$im** es el identificador de la imagen a rotar, **ang** es el ángulo de rotación (expresado en **grados** y tomado en *sentido trigonométrico*) y **\$fondo** es un color de fondo asociado a la imagen a rotar que puede ser definido mediante la función

```

----- PHP -----
    $importada=imagecreatefromjpeg($original);
    break;
  case "png":
    $importada=imagecreatefrompng($original);
    break;
  case "gif":
    $importada=imagecreatefromgif($original);
    break;
}
Header("Content-type:image/jpeg");
# creamos una imagen nueva, un color de fondo y la rellenamos con él
$im_base=imagecreatetruecolor($Ancho,$Alto);
$fondo=imagecolorAllocate($im_base,255,255,200);
imagefill($im_base,0,0,$fondo);
# superponemos la imagen importada posicionandola y aplicandole
# una trasparencia de 50
imagecopymerge( $im_base, $importada, $borde, $borde ,
               0, 0, $orig_Ancho, $orig_Alto ,50 );

imagejpeg($im_base);
ImageDestroy();
?>
```

[Ver ejemplo .jpg](#)

[Ver ejemplo .png](#)

[Ver ejemplo .gif](#)

Si observas los resultados obtenidos en el ejemplo en el que intentamos dar transparencia a una imagen en formato **png** podrás observar que *deja bastante que desear* y produce un efecto indeseado por el recorte de las zonas *presuntamente transparentes*.

Esta situación nos obliga a replantear la situación para prever esta circunstancia y recurrir a un *truco* que parece solventar ese problema. La modificación del código fuente es la incluimos aquí debajo.

```

Header("Content-type:image/jpeg");
$im_base=imagecreatetruecolor($Ancho,$Alto);
$fondo=imagecolorAllocate($im_base,255,255,200);
imagefill($im_base,0,0,$fondo);
/* el truco consiste en crear una segunda imagen (im_truco) cuyas
dimensiones coincidan con las de la imagen transparente
que pretendemos colocar. Le asignamos como color el fondo el mismo
de la imagen destino y hacemos transparente ese color en esa imagen.
Después hacemos una copia de la imagen original sobre la imagen
im_truco y sustituimos en la función imagecopymerge la
imagen original por la obtenida mediante esta chapucilla */
$im_truco=imagecreatetruecolor($orig_Ancho, $orig_Alto);
$fondo1=imagecolorAllocate($im_truco,255,0,200);
imagefill($im_truco,0,0,$fondo1);
imagecolorTransparent ($im_truco,$fondo1);
imagecopy($im_truco, $importada, 0, 0, 0, 0, $orig_Ancho, $orig_Alto);
imagecopymerge( $im_base , $im_truco, $borde , $borde ,
               0, 0, $orig_Ancho, $orig_Alto ,60 );
imagejpeg($im_base);
ImageDestroy();
```

[Ver el nuevo resultado](#)

La función **imagecolorTransparent (\$imagen,\$color)** permite hacer transparente –en la imagen indicada mediante la variable **\$imagen**– el color señalado en la variable **\$color**.

La variable **\$color** deberá estar definida previamente mediante la función **imagecolorAllocate** u alguna otra que permita identificar un color determinado.

### Rotación de imágenes

```

<?
# obtener la imagen
```

*imagecolorallocate* u otra función que permita asociar colores a imágenes.

## Transparencia en la rotación de imágenes

Hemos intentado explorar la posibilidad de lograr imágenes rotadas con fondo transparente. Y la cosa resulta, cuando menos complicada. Cuando se trata de insertarlas sobre un *fondo plano* la situación puede *emularse* fácilmente sin más que asignar como color de fondo de la rotación uno idéntico al del *fondo plano* sobre el que ha de situarse la imagen.

Otra de las alternativas probadas fué tratar de usar *imagecolortransparent* en el color de fondo de rotación de la imagen. Es evidente que ese grado de transparencia solo lo lograremos con un formato **png** ó **jpg**. Pero... no funciona.

La alternativa siguiente fué tratar de crear una imagen con fondo transparente e insertar en ella la imagen rotada asignándole transparencia a la capa que contiene la imagen rotada. Ahí nos encontramos con algunas dificultades.

La primera de ellas es que *fondo transparente* sólo lo permiten las imágenes que son creadas mediante la función *imagecreate*. Si se crean mediante la función *imagecreatetruecolor* esa opción no funciona.

Otra diferencia entre ambas funciones tiene también relación con los colores de fondo. Mientras que en el caso de *imagecreate* se asigna como color de fondo el que se haya definido inmediatamente después de crear la imagen, cuando se trata de *imagecreatetruecolor* se asignará siempre un fondo negro y para cambiar ese color será necesario recurrir a la función *imagefill*.

Pero la *felicidad completa* parece que no existe. Al intentar explorar la primera de estas opciones hemos podido observar que el precio a pagar por la dichosa transparencia es obtener una imagen final de no demasiado buena calidad.

Mejor lo compruebas tu mism@ en el ejemplo que tienes a la derecha.

```
$original=$_SERVER['DOCUMENT_ROOT']."/cursophp/images/aviones3.jpg";
for($i=strlen($original)-1;$i>0;$i--) {
    if (substr($original,$i,1)==".") {
        $tipo=substr($original,$i+1);
        break;
    }
}
switch($tipo) {
    case "jpg":
        $importada=imagecreatefromjpeg($original);
        break;
    case "png":
        $importada=imagecreatefrompng($original);
        break;
    case "gif":
        $importada=imagecreatefromgif($original);
        break;
}
Header("Content-type:image/jpeg");
$fondo=imagecolorallocatealpha($importada,255,255,0,40);
$im_base=imagerotate($importada,30,$fondo);
imagejpeg($im_base);
ImageDestroy();
?>
```

[Ver ejemplo](#)

## Diferencias entre *imagecreate* e *imagecreatetruecolor*

Este es el código fuente de un script que lee una imagen externa y la copia íntegra sobre otra imagen creada mediante PHP. Si visualizas el ejemplo podrás observar las diferencias entre usar la función *imagecreate* o utilizar *imagecreatetruecolor*.

```
<?
$original=$_SERVER['DOCUMENT_ROOT']."/cursophp/images/aviones3.jpg";

for($i=strlen($original)-1;$i>0;$i--) {
    if (substr($original,$i,1)==".") {
        $tipo=substr($original,$i+1);
        break;
    }
}
switch($tipo) {
    case "jpg":
        $importada=imagecreatefromjpeg($original);
        break;
    case "png":
        $importada=imagecreatefrompng($original);
        break;
    case "gif":
        $importada=imagecreatefromgif($original);
        break;
}
$dimensiones=getimagesize($original);
$Ancho_original=$dimensiones[0];
$Alto_original=$dimensiones[1];
Header("Content-type:image/png");
$im_base=imagecreate($Ancho_original+20,$Alto_original+20);
$fondo=imagecolorallocate($im_base,255,0,0);
imagecolortransparent($im_base,$fondo);
imagecopy($im_base,$importada,10,10,0,0,$Ancho_original,$Alto_original);
imagepng($im_base);
ImageDestroy();
?>
```

[Ver ejemplo](#)

## Ejemplo resumen

Aquí tienes un ejemplo en el que hemos utilizado superposiciones de imágenes, con giros, recortes y diferentes grados de transparencia.

[Ver ejemplo resumen](#)

[Ver código fuente](#)

de los fondos de rotación no plantea ningún problema.

Es suficiente usar la función `imagecolortransparent`, eso sí, aplicándola a la imagen correspondiente *antes* de insertarla mediante la opción `copy` en la imagen final.

### ¡Cuidado!

La función `imagerotate()` no funciona con la versión de la librería GD que se instala con Ubuntu o Debian que es diferente de la que utilizan otras distribuciones de PHP (la de Windows por ejemplo). La forma de resolverlo puedes encontrarla [aquí](#)

### Ejercicio nº 34

Diseña un script que genere una imagen dinámica a partir de una fotografía. El script debe encuadrarla (recortándola por los cuatro bordes) y colocar en la parte inferior derecha un texto que diga: «Todos los derechos reservados».

---

Anterior

Índice

Siguiente





[Ver índice](#)

## Las Cookies



### Las cookies

De igual modo que ocurría con la función `mail`, no todos los *hosting* tienen habilitada la opción de envío de **cookies**.

Como sabes, las **cookies** son pequeños ficheros que se escriben en el ordenador del cliente.

Si utilizas *Internet Explorer* podrás ver que se almacenan como ficheros de texto en un directorio llamado *Archivos temporales de Internet* y que su nombre es de este tipo:

`xxx@nombre[z].txt`

donde **xxx** suele ser el nombre que figura en el registro de Windows como nombre del equipo (el que se pone al instalar Windows); **nombre** suele ser el nombre del directorio de servidor desde el que se envió la cookie y el número **z** suele ser el ordinal del números de accesos a la página que envía la **cookie**.

*Netscape* trata las **cookies** de distinta forma ya que las almacena en un único fichero llamado **cookies.txt** que está en el subdirectorio del usuario que se crea en la instalación del programa y que está dentro de otro llamado **users**.

### ¿Cómo enviar cookies?

La instrucción para el envío de **cookies** debe insertarse al principio de la página y antes de cualquier etiqueta HTML o línea en blanco. Esto es muy importante, no lo olvides.

La sintaxis es la siguiente:

**setcookie(nom, val, exp)**

donde:

**nom** es una cadena que contiene el nombre de la variable que recoge el valor de la cookie.

**val** es el valor que se asignará a la variable anterior. Puede ser tanto numérico como de tipo cadena.

**exp** indica cuál ha de ser la fecha de caducidad de la cookie.

Suele escribirse: `time()` (hora actual) más un número que representa los segundos que han de transcurrir hasta que la cookie **expire**.

### Una cookie muy simple

Aquí tienes un ejemplo, el más simple, del uso de esta función.

Si ejecutas el ejemplo por primera vez observarás que solo aparecerá el texto *Esto es la galletita*: sin ningún valor. Sin embargo, si actualizas el navegador o ejecutas más tarde el ejemplo siguiente aparecerá *Mi regalito* como valor de la variable.

La explicación es la siguiente: las instrucciones PHP se ejecutan en el servidor antes de enviar la página al cliente. Eso significa que, al ejecutar por primera vez, se inserta la orden de escritura y se comprueba el valor de la variable, que aun no ha sido creada y por ello aparece en blanco. Será en la actualización –ya se habría producido un envío al navegador y ya se habría escrito la cookie– cuando si se leerá el valor anterior.

Siempre que tratemos de visualizar el valor de una cookie estaremos viendo el valor asignado en la petición anterior

```
<?
# setcookie escribe el contenido de la cookie
# en el ordenador del cliente
setcookie("cookie1","Mi regalito",time()+3600);
# escribe el valor leído en la cookie
echo "Esta es la galletita:",$_COOKIE['cookie1'];
?>
```

**ejemplo108.php**

Si hemos ejecutado el ejemplo anterior este script ya podrá leer el contenido de la cookie escrita a través de aquel.

```
<?
echo "Esta es la galletita:",$_COOKIE['cookie1'];
?>
```

**ejemplo109.php**

### Una cookie con valores asignados mediante una variable

```
<?
$z="Mi regalito";
setcookie("cookie2",$z,time()+3600);
echo "Esta es la galletita:",$_COOKIE['cookie2'];
?>
```

**ejemplo110.php**

Este segundo ejemplo ya visualizaría el valor de esta cookie siempre que hubiéramos ejecutado previamente el script anterior.

```
<?
echo "Esta es la galletita:",$_COOKIE['cookie2'];
?>
```

**ejemplo111.php**

Los valores contenidos en las *cookies* pueden ser leídos por el servidor a partir de variables predefinidas o recogidas a través de un formulario.

Tal como puedes ver en el ejemplo [ejemplo112.php](#), es posible crear *cookies* en las que la variable contiene un *array* de tipo *escalar* o de tipo *asociativo*.

Como puedes ver en el ejemplo, es necesario incluir un *setcookie* por cada uno de los *valores del array* aunque a la hora de devolver esa variable el *navegador del cliente* envía el *array completo*.

### ¿Cómo leer cookies?

Al invocar la variable mediante la que fue escrita la *cookie* –desde cualquier página que esté alojada en el *subdirectorio del servidor* desde el que fué creada– tomará -de forma automática- el valor contenido en la *cookie*.

Esto sólo sería posible en el caso de que la *cookie* hubiera sido creada con anterioridad y que no hubiera *expirado*.

Tal como comentábamos a estudiar los formularios, las *cookies* pueden leerse directamente (imprimir la variable PHP con nombre idéntico a la *cookie*, sólo en el caso de que la opción *register\_globals=on*.

Si no fuera así (de forma similar al caso de los formularios) para poder leer su contenido tendríamos que recurrir al uso de la variable superglobal *\$\_COOKIE* (caso de que la versión de PHP las soporte) o, alternativamente, de *\$HTTP\_COOKIE\_VARS* teniendo en cuenta que esta última no tiene carácter superglobal.

Puedes comprobar que en los ejemplos se visualizan los contenidos de las *cookies* usando la instrucción *echo* (o *print*) y utilizando el array superglobal *\$\_COOKIE* para garantizar el funcionamiento con cualquier configuración de *register globals*.

## Una cookie definida como array

```
<?
$valores=Array("Verde","Verano","Rolls-Royce","Millonario");
# a diferencia de lo que ocurre al definir elementos de array asociativos
# en este caso los indices asociativos (color, estación, etc.) no van
# entre comillas
setcookie("cookie3[color]",$valores[0],time()+3600);
setcookie("cookie3[estacion]",$valores[1],time()+3600);
setcookie("cookie3[coche]",$valores[2],time()+3600);
setcookie("cookie3[finanzas]",$valores[3],time()+3600);
# la variable superglobal $_COOKIE['cookie3'] contiene un array, por ello
# la lectura de sus valores debe hacerse considerando que se trata de un
# array bidimensional
if (isset($_COOKIE['cookie3'])) {
    while( list($indice, $valor) = each($_COOKIE['cookie3']) ) {
        echo "$indice == $valor\n";
    }
}
?>
```

[ejemplo112.php](#)

Como en los casos anteriores también este script permite leer los contenidos de la *cookie* después de haber accedido a la página anterior.

```
<?
if (isset($_COOKIE['cookie3'])) {
    while( list($indice, $valor) = each($_COOKIE['cookie3']) ) {
        echo "$indice == $valor\n";
    }
}
?>
```

[ejemplo113.php](#)

## Un contador como aplicación práctica

```
<?
$numero=$_COOKIE['visitante'];
$numero+=1;
setcookie("visitante",$numero,time()+86400);

if($numero==1){print "Es la $numero vez que visitas esta página";}
if($numero>1){print "Es la $numero ª vez que visitas esta página";}
?>
```

[ejemplo114.php](#)

[Anterior](#)

[Índice](#)

[Siguiente](#)



Ver índice

# Gestión de directorios



## Funciones con directorios

PHP dispone de funciones que permiten obtener información sobre los contenidos de los directorios del servidor.

Algunas de estas funciones son las siguientes:

### Pseudo-objeto dir

Mediante la expresión:

#### \$b= dir (path)

en la que **path** es la **ruta absoluta**:

(..**/dir/subdir/subsubdir**)

**o relativa**

(**./subdir**)

hasta el directorio del que vamos a obtener información, recogemos en la variable **\$b** información sobre el directorio en cuestión de una forma un tanto peculiar.

**\$b** se comporta como **objeto** y, como tal *objeto*, posee los **métodos y funciones** que indicamos:

#### handle

Devuelve una cadena con la descripción del identificador del recurso.

#### path

Devuelve la **ruta** del directorio especificado.

#### read()

Realiza una lectura secuencial de los nombres y extensiones de los ficheros contenidos en el directorio especificado.

#### rewind()

Posiciona el puntero en la posición inicial.

#### close()

Cierra el *identificador* de directorio.

## Otras funciones

#### \$f= opendir (path)

Recoge en la cadena **\$f** un identificador que permitirá utilizar las restantes funciones.

#### \$z= readdir (\$f1)

Hace una lectura **secuencial** del

## El pseudo-objeto \$b=dir("path")

Este es un ejemplo de utilización de los métodos de éste pseudo-objeto. Observa que tanto el método **handle** como el método **path** se invocan sin utilizar () mientras que tanto **read()** como **rewind()** como **close()** sí utilizan esos paréntesis al invocar sus métodos.

```
<?
$directorio = dir("./images");
# en el caso de los objetos la manera de invocar
# uno de sus métodos y/o funciones requiere una sintaxis
# específica con la que vera a lo largo de este ejemplo:
# es $objeto->metodo que equivale a la tradicional llamada
# a una variable en la forma $variable.
echo "Handle: ".$directorio->handle."<br>\n";
echo "Path: ".$directorio->path."<br>\n";
while($fichero=$directorio->read()) {
    echo $fichero."<br>\n";
}
$directorio->rewind();
echo "nuevo listado del directorio despues de rebobinar<br>" ;
while($fichero=$directorio->read()) {
    echo $fichero."<br>";
}
$directorio->close();
?>
```

ejemplo106.php

## Otras funciones de directorios

Existen otras funciones -indicadas al margen- que permiten obtener la misma información que la que obtuvimos en el ejemplo anterior.

Antes de efectuar la lectura de un directorio es necesario *abrirlo* con la función **opendir**, y una vez finalizada, es aconsejable *cerrarlo* utilizando la función **closedir**.

```
<?
#abrimos el identificador de directorio
$f = opendir("./images");
#leemos el primer fichero que será ".." (recuerda la estructura
# de los directorios de MS-DOS)
$fichero=readdir($f);
echo $fichero,"<br>";

#leemos el fichero siguiente que será "..." (recuerda la estructura
# de los directorios de MS-DOS)
$fichero=readdir($f);
echo $fichero,"<br>";

#leemos el fichero siguiente (el primer fichero "real")
$fichero=readdir($f);
echo $fichero,"<br>";

#rebobinamos, enviando el puntero al primer fichero
rewinddir($f);
```

directorio indicado por el **identificador \$f1**.

A medida que efectúa la lectura secuencial *el puntero de lectura* va desplazándose al fichero que sigue al último leído.

#### **rewinddir(\$f1)**

Rebobina haciendo que el puntero apunte al **primer fichero** del directorio.

#### **closedir(\$f1)**

Cierra el identificador del directorio.

```
echo "Lista de TODOS los ficheros usando un bucle while<br>";
#leemos todos los ficheros
while($fichero=readdir($f)) {
    echo $fichero."<br>";
}
closedir($f);
?>
```

**ejemplo107.php**

[Anterior](#)

[Índice](#)

[Siguiente](#)



Ver índice

## Protección de directorios



### Protección de directorios

Una vez que tengamos modificada la configuración del servidor Apache estaremos en condiciones de poder proteger directorios –restringir o limitar el acceso a ellos– y también podremos redirigir *peticiones* a páginas predeterminadas en casos concretos.

Este control ha de realizarse mediante ficheros que tienen por nombre **.htaccess** (el primer carácter es el punto) y que no pueden llevar ningún tipo de extensión.

Los ficheros **.htaccess** pueden ser incluidos en cualquier directorio o subdirectorio del espacio del servidor.

### El fichero .htaccess

Las misiones más importantes que puede realizar este fichero son: *redireccionar* y *restringir accesos*.

Veamos cada una de ellas por separado.

#### Errores y redirecciónamiento

Los mensajes de error más frecuentes al intentar acceder a páginas web –y sus causas– son los siguientes:

##### Error 401

El subdirectorio está protegido por número IP o por password y el intento de acceder a él no ha tenido éxito.

##### Error 403

El acceso al documento solicitado está prohibido.

##### Error 404

El documento solicitado no ha sido hallado.

##### Error 500

Error del servidor. Usualmente este error se da cuando se ha intentado ejecutar de forma incorrecta un CGI, o bien debido a problemas en el servidor.

Los errores de los tipos 403 y 404 suelen producirse en la mayoría de las ocasiones por direcciones incorrectas y –aparte de causar un pésimo efecto– suelen provocar el abandono de la visita.

Un fichero **.htaccess** con este contenido:

### Configuración del servidor Apache

Para poder proteger de directorios –mediante un fichero llamado **.htaccess**– es necesario realizar algunas modificaciones en la configuración de Apache.

Abriremos nuestro fichero **httpd.conf** y buscaremos las líneas en las que aparece: **AllowOverride None** y reemplazaremos **None** por **All** guardaremos los cambios en **httpd.conf** y reiniciaremos nuestro servidor Apache.

En caso de que estemos usando como sistema operativo **Linux Ubuntu** no es necesario realizar ninguna modificación. Bastará mantener la configuración establecida durante el proceso de instalación.

### Comprobación de la configuración

Empezaremos escribiendo en la barra de direcciones de nuestro navegador (también puedes hacerlo desde este enlace) esta dirección:

**http://localhost/cursoPHP/noexisto.html** y nos aparecerá un mensaje de error diciendo que no existe ninguna página con ese nombre. Algo lógico, porque realmente no existe.

#### Ejercicio nº 35

Abre tu editor –no utilices el *block de notas* porque te dará muchísimos problemas en este caso– y escribe la siguiente línea:

**ErrorDocument 404 http://localhost/cursoPHP/index.php**

y guarda el documento en el directorio **cursoPHP** con el nombre (aunque te parezca extraño, no lleva nada delante del punto) **.htaccess**

Pulsa de nuevo en el enlace que tienes aquí arriba –o escribe la dirección en el navegador– y observarás que ahora no dice *página no encontrada* sino que se abre la página principal del Curso.

Edita de nuevo el fichero **.htaccess** y añádele las siguientes líneas:

**ErrorDocument 401 http://localhost/cursoPHP/index.php**

**ErrorDocument 403 http://localhost/cursoPHP/index.php**

guardándolo después de haber hecho los cambios.

### Crear el fichero de claves y contraseñas

Las restricciones de usuarios mediante **.htaccess** requieren un fichero de claves y contraseñas.

Para crearlo basta con abrir el *block de notas* y escribir **la clave** seguida de **dos puntos (:) y a continuación escribir la contraseña**.

Podemos poner tantas como deseemos sin más limitaciones que escribir cada bloque **clave:contraseña** en una línea distinta. Este puede ser un ejemplo:

```
pepe:Pepito  
pepa:Pepita
```

Podemos guardar este fichero en el *sitio que deseemos* sin que sea necesario que pertenezca al root del servidor.

El directorio **seguridad** que hemos creado cuando tratábamos de **INCLUDE** (ver página) puede ser un buen sitio. Podemos ponerle cualquier nombre sin que importe que tenga extensión o no la tenga.

### Crear un fichero de contraseñas encriptadas

ErrorDocument 401 pagX  
ErrorDocument 403 pagY  
ErrorDocument 404 pagZ

donde pagX, pagY y pagZ sean direcciones (completas) de las páginas a las que deseamos *redireccionar* el navegador, conseguiría que esos errores *llevaran* al visitante a la página que nosotros deseáramos.

### Herencias

El archivo .htaccess provoca **herencia**. Eso significa que las especificaciones incluidas en un directorio –sean restricciones o redirecciones– son efectivas en todos los subdirectorios que contiene, incluso en el caso de que esos subdirectorios tengan su propio .htaccess y que en él se establezcan condiciones distintas a las anteriores.

Al crear el fichero del ejercicio nº 22 y guardarlo en *cursoPHP* las condiciones establecidas afectarán a todos sus subdirectorios (puedes probar a abrir una página con un nombre cualquiera en el subdirectorio *images*). Por eso, si pretendemos que desde subdirectorios distintos se redireccione a páginas distintas tendremos que incluir un .htaccess en cada uno de ellos y omitirlo en el directorio que los contenga.

### Protección de directorios

Son muchas las posibilidades que ofrece .htaccess a la hora de restringir el acceso a un directorio determinado.

Entre otras opciones, se puede denegar el acceso a todos los usuarios; denegar el acceso con *excepciones*, autorizar a todos (equivale a no restringir), autorizar con *excepciones* o requerir clave y contraseña.

Lo que hemos denominado *excepciones* también permite una serie de alternativas tales como: una IP determinada, un rango de IP's, nombres de dominio, etcétera.

Sólo comentaremos la forma de protección de directorios mediante claves de usuario y contraseña.

### Restricción de acceso a usuarios no autorizados

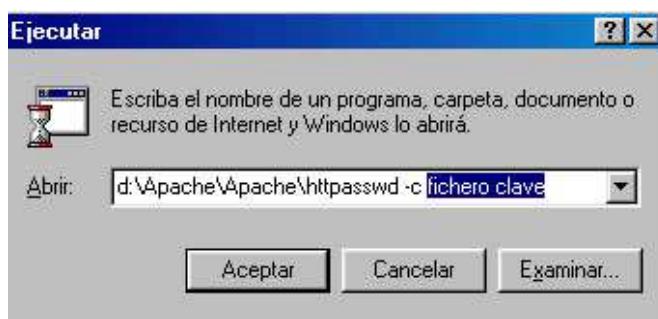
Este tipo de protección requiere crear un *fichero de claves* y *contraseñas* y configurar de forma adecuada .htaccess

Lo relativo a la creación de los primeros lo tenemos detallado aquí a la derecha en sus dos opciones:

Apache posee una utilidad que permite la creación de ficheros de claves con contraseñas **encriptadas**. Se trata de un programa llamado **htpasswd.exe** que está en el subdirectorio **bin** del servidor.

Para **crear un nuevo fichero** el procedimiento sería el siguiente:

En la línea de comandos: **Inicio->Ejecutar**



debemos escribir:

**path htpasswd -c nombre y path del fichero de claves usuario**

si se trata de un sistema operativo **Linux Ubuntu** habría que escribir en la consola

**htpasswd -c nombre y path del fichero de claves usuario**

En nuestra configuración y para **crear un fichero** con la palabra clave **pepe** escribiríamos:

C:/ServidoresLocales/Apache/bin/htpasswd -c C:/ServidoresLocales/Apache/seguridad/misclaves.txt pepe

y aparecería una ventana de MS-DOS en la que deberemos escribir la **password** para ese usuario.



### Añadir usuarios a un fichero de contraseñas encriptadas

Porcederíamos de la misma forma. Volveríamos a ejecutar **htpasswd** con la nueva clave pero sin utilizar **-c**.

**¡Cuidado!**

El modificador **-c** **destruiría el fichero anterior, si existiera** y crearía uno nuevo.

El proceso ahora sería:

encriptadas o sin encriptar.

Para el caso específico de nuestro servidor Apache, el fichero **.htaccess** ha de contener:

#### **AuthType Basic**

No permite modificación e indica el tipo de autentificación requerida.

#### **AuthName "nuestro texto"**

El texto que escribamos aquí aparecerá como mensaje en la ventana en la que nos pedirá la clave

#### **AuthUserFile "path"**

Entre esas comillas debes escribir el *nombre del fichero de contraseñas* especificando su path completo.

#### **require valid-user**

Este texto indica que para acceder se requiere un usuario válido.

Con nuestra configuración de Apache no es necesario especificar en **.htaccess** la forma de encriptación de contraseñas. El propio servidor interpreta el contenido del fichero y aplica u omite los criterios de encriptación.

A riesgo de parecerte pesados tenemos que volver a insistir que no todos los **hosting** tienen habilitada esta opción, pero además hemos de hacer mención a otro detalle muy importante.

La configuración que hemos comentado **no es válida** para todos los servidores.

Según como esté configurado el servidor, la versión del software que utilice, etcétera no sería extraño que se necesitara esta otra sintaxis:

```
AuthType Basic  
AuthName "Texto"  
AuthUserFile fichero  
required valid-user  
AuthTextCrypt On/Off
```

u otras similares que pueden inducirnos al error. Lo mejor, en caso de servidores ajenos, es *consultar* al administrador del sistema sobre estos aspectos y recabarle detalles sobre la sintaxis específica de su configuración.



C:/ServidoresLocales/Apache/bin/htpasswd C:/ServidoresLocales/Apache/seguridad/misclaves.txt luis

Habríamos creado así nuestro fichero con claves encriptadas. Si pretendiéramos visualizarlo nos aparecería lo siguiente:

```
pepe:$apr1$EC4.....$7Z3.p2tv2QpzrZbo4bI2j0  
luis:$apr1$SU4.....$iU8a.YTo.ZvYyRggDAvTC.
```

#### **¡Cuidado!**

Bajo **Linux**, antes de añadir el usuario *luis* tendríamos que asegurarnos de que el fichero **misclaves.txt** tenga permisos para poder efectuar modificaciones en su contenido.

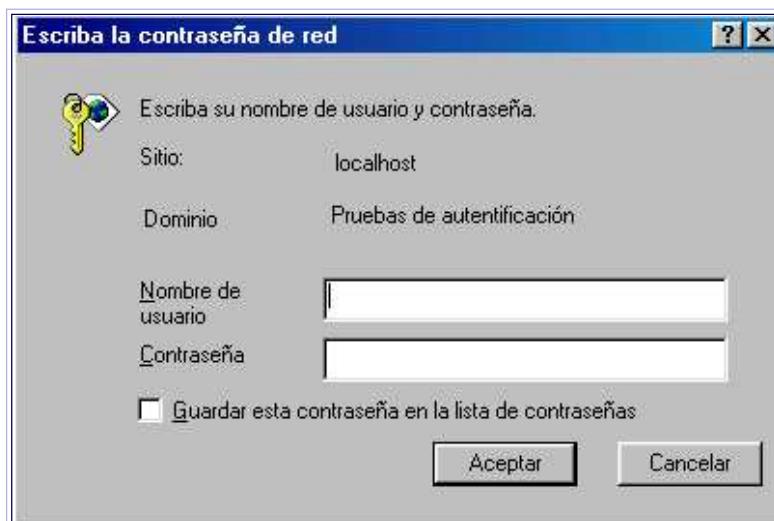
## **Un ejemplo de .htaccess**

Supongamos que tenemos un directorio llamado **protegido** en cualquier parte del servidor (por ejemplo dentro de **htdocs**) la forma de protegerlo sería crear un fichero con nombre **.htaccess** con un contenido como este:

```
AuthType Basic  
AuthName "Pruebas de autentificación"  
AuthUserFile "C:/ServidoresLocales/Apache/seguridad/misclaves.txt"  
require valid-user
```

y guardarlo en ese directorio.

Al acceder al directorio **protegido** aparecerá una ventana como esta:



y si al cabo de *tres intentos* no escribimos *la clave y contraseña adecuadas* se producirá un **Error 401**.



Ver índice

## Sesiones (I)



### ¿Qué son las sesiones?

Suponemos que habrás estado alguna vez en un hotel y que recuerdas que al inscribirnos como huéspedes nos facilitan una *tarjetita identificativa* que teóricamente habrás presentado a la hora de solicitar cualquier servicio del hotel (cafetería, restaurante, etc.).

Al registrarnos en ese hotel estaremos **ini ciando una sesión** (nuestro período de estancia) y al recibir la tarjeta identificativa se nos estará facilitando un **identificador de sesión**, que tiene validez **temporal**, ya que **expirará** en la **fecha de salida** indicada en ella.

Imaginemos ahora que vamos al **restaurante**.

Pueden ocurrir dos cosas: que decidamos **efectuar el pago directamente**, o que pidamos que el importe de la factura se incluya en **nuestra cuenta**. En el segundo de los casos, se **reiniciará la sesión** y se registrará una nueva **variable de sesión** –el importe del servicio– al firmar la *nota de cargo* del mismo.

El responsable del restaurante guardará esa *nota de cargo* –**una variable de sesión**– de forma **temporal** ya que una vez abonado su importe, en el momento que abandonemos el hotel, expirará la sesión y dejará de tener validez.

Se requiere –por tanto– un **directorio temporal** en el que **almacenar las variables de sesión**.

En PHP la sesiones funcionan de forma muy similar y de esa similitud surge la necesidad de habilitar un directorio temporal antes de utilizar **sesiones**. Lo hacemos cuando modificamos la configuración tal como comentamos en la columna de la derecha.

### Funciones de sesión

Para la gestión de **sesiones** se utilizan estas funciones:

#### **session\_start()**

Crea una sesión o continúa con la actual. En el segundo caso el identificador de sesión debe ser *transferido* por medio de una variable GET o a través de una cookie.

#### **session\_name()**

### Requisitos de configuración

Antes de poder utilizar este tipo de funciones es preciso comprobar el fichero de configuración de php. si visualizamos el fichero **info.php** y buscamos la directiva **session.save\_path** veremos que tiene el valor (en el caso de Windows) siguiente:

```
session.save_path="C:/ServidoresLocales/tmp"
```

eso es consecuencia de la creación del directorio **tmp** y de la modificación que hicimos en la **línea 992** de **php.ini** cuando detallábamos el proceso de configuración de php ([puedes verlo aquí](#)).

En el caso de la instalación sobre Ubuntu se configura de forma automática. Al observar **info.php** podrías ver que:

```
session.save_path="/var/lib/php5"
```

### Iniciando y propagando sesiones

Aquí tenemos un ejemplo de como *iniciar una sesión* y también podremos comprobar como **se propaga** al llamar a la misma página si está activada la opción **aceptar cookies**.

Si ejecutamos reiteradamente el script –pulsando en *volver a llamar esta página*– podremos ver que no se modifica el valor del **identificador de sesión**.

Si **bloqueamos las cookies** (en este enlace está descrito el procedimiento para hacerlo) podremos comprobar que **ahora la sesión no se propaga** y que cada vez que *volvemos a llamar a la página* nos aparece un nuevo valor en el **identificador de sesión**, es decir, se crea una nueva sesión.

[Ver índice](#)

[Búsqueda rápida](#)

[◀ Página anterior](#)

[Página siguiente ▶](#)

```
<?
echo session_id(),"<br>";
echo session_name(),"<br>";
?>
<A Href="ejemplo116.php">Volver a llamar esta página</A>
```

[ejemplo116.php](#)

En este otro ejemplo solo hemos hecho el *añadido* (señalado en magenta) para el caso de que PHP no esté compilado para **--enable-trans-sid** (una opción bastante habitual en los hosting que trabajan bajos sistemas operativos UNIX ó LINUX).

Si lo ejecutamos podremos comprobar que ocurre *exactamente lo mismo* que en el ejemplo anterior, tanto con **cookies** activadas como desactivadas.

```
<?
session_start();
#pedimos que escriba el identificador único y el nombre de la sesión
echo session_id(),"<br>";
echo session_name(),"<br>";
?>
<A Href="ejemplo117.php?<?=SID ?>">Volver a llamar esta página</A>
```

[ejemplo117.php](#)

Ahora haremos una **modificación importante** en el script. Al **incluir en la llamada a la página el nombre y el identificador de sesión** estamos transfiriendo –mediante el método GET– el valor de esa variable. De esta forma, la propagación de la sesión estará asegurada sea cual fuere la configuración del navegador del cliente.

Recoge el nombre de la sesión.

Si no se asigna uno de forma explícita, utiliza como nombre de sesión el contenido de la directiva **session.name** del fichero **php.ini**.

Por defecto ese nombre suele ser **PHPSESSID**, pero no está de más comprobarlo mirando el valor actual de la directiva **session.name** en **info.php**.

#### **session\_name('nombre')**

Esta función permite asignar un **nuevo nombre** a la sesión actual.

Debemos tener en cuenta que si **cambiamos de página** y queremos mantener el mismo **identificador** (conservar la sesión anterior) esta función debe ser escrita, con el mismo nombre, en la nueva página, y además, ha de ser *insertada antes de llamar* a la función **session\_start()** para que se inicie la sesión.

#### **session\_cache\_limiter()**

El limitador de caché controla las **cabeceras** HTTP enviadas al cliente.

Estas **cabeceras** determinan las reglas mediante las que se habilita la opción de que los **contenidos** de las páginas puedan ser guardados en la caché local del cliente o se impida tal almacenamiento.

En este último modo –no caché– cada petición de página requeriría una nueva llamada al servidor, lo cual tiene –como todo en la vida– ventajas e inconvenientes

Entre las ventajas está la garantía de que en cada acceso estamos viendo la versión actualizada de la página, cosa que podría no ocurrir de otro modo. El inconveniente es que requiere una nueva petición que puede significar un tiempo de espera, mientras el servidor produce la respuesta.

Esta opción viene configurada *por defecto* en las directivas de configuración del **php.ini** como **nocache**.

Para evitar –sea cual fuera la configuración de **php.ini**– el almacenamiento de las páginas en la caché del cliente hemos de utilizar:

#### **session\_cache\_limiter ('nocache,private').**

De igual forma que ocurría con **session\_name**, si utilizamos esta función debemos escribirla **antes** que **session\_name** y –también igual que en aquel caso– deberemos repetirla en cada uno de los documentos.

**¡Advertencia!**

Podemos comprobar –activando/desactivando cookies– que en las sucesivas llamadas a la página **se mantiene** el identificador de sesión.

```
<?
session_start();
#pedimos que escriba el identificador único y el nombre de la sesión
echo session_id(),"<br>";
echo session_name(),"<br>";
?>
<A Href="ejemplo118.php?<?echo session_name()."=".session_id() ?>">
    Volver a llamar esta página</A>
```

**ejemplo118.php**

## Sesiones con nombre propio

En los ejemplos siguientes se crean y propagan **sesiones con nombre propio** sin que ello altere las condiciones de respuesta con las opciones **aceptar /no aceptar cookies**. También hemos incluido **session\_cache\_limiter** con las restricciones de **nocache** con lo cual el navegador no guardará las páginas de ejemplo en la **caché**.

Si cambiáramos los parámetros (**nocache, private** por **public**) podríamos comprobar se almacena en la caché del navegador la página web devuelta por el servidor.

```
<?
session_cache_limiter('nocache,private');
session_name('leocadia');
session_start();
#pedimos que escriba el identificador único
echo session_id(),"<br>";
echo session_name(),"<br>";
?>
<A Href="ejemplo119.php">Volver a llamar esta página</A>
```

**ejemplo119.php**

```
<?
session_cache_limiter('nocache,private');
session_name('leocadia');
session_start();
#pedimos que escriba el identificador único
echo session_id(),"<br>";
echo session_name(),"<br>";
?>
<A Href="ejemplo120.php?<?=$SID ?>">Volver a llamar esta página</A>
```

**ejemplo120.php**

```
<?
session_cache_limiter('nocache,private');
session_name('leocadia');
session_start();
#pedimos que escriba el identificador único
echo session_id(),"<br>";
echo session_name(),"<br>";
?>
<A Href="ejemplo121.php?<?echo session_name()."=".session_id() ?>">
    Volver a llamar esta página</A>
```

**ejemplo121.php**

## Las cookies de sesión y sus parámetros

Para el caso de que el navegador del cliente tenga activada la opción de **aceptar cookies** PHP dispone de una función que permite *leer* los parámetros de esa **cookie**. Para ello dispone de la función

`session_get_cookie_params()` que devuelve un **array asociativo** con los siguientes índices:

El *orden* de escritura de estas instrucciones sería el siguiente:

```
<?
session_cache_limiter();
session_name('nombre');
session_start();
.....
?>
```

Es muy importante mantener ese orden y que este bloque de instrucciones sea el *primer elemento de la página* -antes de cualquier otra etiqueta- y que *no haya líneas en blanco* ni antes de la etiqueta `<?` ni entre ella y las llamadas a estas funciones.

### Propagación de las sesiones

La verdadera utilidad de las sesiones es la posibilidad de que sean propagadas, es decir, que tanto el *identificador de sesión* como los *valores de las variables de sesión* -luego hablaremos de estas variables- puedan ir pasando de *una página a otra* sin necesidad de recurrir al uso de formularios.

Para entenderlo, se trata de *dar validez y utilizar la misma tarjeta* para movernos en las diferentes secciones del *hotel* que hemos utilizado como ejemplo.

La forma habitual de *propagar* las sesiones es a través de *cookies*, pero como quiera que el usuario tiene la posibilidad de activar la opción *no aceptar cookies* y eso es algo que no podemos prever, **PHP** dispone una opción alternativa que permite la *propagación* a través de la URL.

#### Caso de que el cliente tenga activada la opción aceptar cookies

Si queremos que las sesiones se propaguen *únicamente* en el caso de que esté activada la opción *aceptar cookies* en el navegador bastará con *hacer la llamada* a la nueva página siguiendo el método tradicional, es decir:

```
<A href="pagxx.php">
```

y que esa nueva página contenga sin que lo preceda ninguna línea en blanco el script siguiente:

```
<?
session_cache_limiter();
session_name('nombre');
session_start();
.....
?>
```

donde `session_cache_limiter` no es un elemento imprescindible y podemos incluirlo o no, de acuerdo

**lifetime**: Indica el tiempo de duración de la cookie

**path**: Indica la ruta -en el ordenador del cliente- en la que se almacenan los datos de la sesión

**domain**: Indica el dominio de procedencia de la cookie

**secure**: Indica si la cookie solo puede ser enviada a través de *conexiones seguras* (1) o si no considera esa posibilidad (0).

Los valores **por defecto** para estos parámetros se pueden establecer en la configuración del fichero `php.ini`. Si visualizamos `info.php` podremos ver -en las directivas `session.cookie_*` nuestros valor por defecto- y observaremos que `session.cookie_lifetime` tiene valor 0, razón por la cual si -con la opción aceptar cookies activada- ejecutamos cualquiera de los script anteriores y miramos el directorio *Temporal Internet Files*, no encontraremos ninguna de estas cookies, dado que su plazo de expiración es cero.

```
<?
session_name('mi_sesion');
session_start();
echo session_id(),"<br>";
echo session_name(),"<br>";
# recogemos en la variable $a el array con los datos de la sesión
$a=session_get_cookie_params();
foreach($a as $c=>$v){
    echo $c,"--->",$v,"<br>";
}
?>
<A href="ejemplo122.php">Volver a llamar esta página</A>
```

ejemplo122.php

```
<?
session_name('mi_sesion');
session_start();
echo session_id(),"<br>";
echo session_name(),"<br>";
# recogemos en la variable $a el array con los datos de la sesión
$a=session_get_cookie_params();
foreach($a as $c=>$v){
    echo $c,"--->",$v,"<br>";
}
?>
<A href="ejemplo123.php?<?=SID ?>">Volver a llamar está página</A>
```

ejemplo123.php

```
<?
session_name('mi_sesion');
session_start();
echo session_id(),"<br>";
echo session_name(),"<br>";
# recogemos en la variable $a el array con los datos de la sesión
$a=session_get_cookie_params();
foreach($a as $c=>$v){
    echo $c,"--->",$v,"<br>";
}
?>
<A href="ejemplo124.php?<?echo session_name()."=".session_id() ?>">
    Volver a llamar esta página</A>
```

ejemplo124.php

Las configuraciones por defecto de `session.cookie` pueden cambiarse sin necesidad de modificar el fichero `php.ini`.

Mediante la función `session_set_cookie_params(duración , 'path', 'dominio', 'segura')` se pueden configurar esos parámetros de forma temporal-*únicamente para la página en la que está insertado*- y que como en los casos anteriores tiene tres posibilidades de script.

En los ejemplos siguientes, se escribirá -siempre que esté configurada la opción *aceptar cookies*

con nuestro interés.

Con `session_name` ocurre algo similar. Si la sesión fue iniciada con un nombre distinto al que le asigna por defecto PHP, debemos escribir en el script anterior la función `session_name`, y además, debe contener el *mismo nombre* que le habíamos asignado en la página de procedencia.

Si no se asigna ningún nombre tomará `PHPSESSID` –nombre por defecto– en todas las páginas y no será necesaria la instrucción `session_name`.

## Caso de cookies desahabilitadas

Para garantizar la propagación de las sesiones –aún cuando esté activada la opción *no aceptar cookies*– tendremos que *pasar el identificador de sesión* junto con las llamadas a las páginas siguientes. Para ello se requiere la siguiente sintaxis:

```
<A href="pagxx.php?
<? echo session_name(). "="
.session_id()?>
```

Con esta sintaxis, después de escribir el ? (recordemos que es la forma de indicar que añadimos variables y valores para que sean transferidos junto con la petición de la página) estaremos pidiendo a PHP que escriba como nombre de variable el nombre de la sesión y como valor de esa variable, después de insertar el signo igual, el identificador de sesión.

Es evidente que la utilización de esta opción nos permite asegurar que las sesiones y sus variables podrán ser usadas sin riesgos de que una configuración inadecuada por parte del cliente produzca resultados imprevisibles.

### Ejercicio nº 36

Desarrolla un formulario en el que el usuario pueda escribir su nombre y elegir un color de fondo. Al enviar este formulario los valores de ambos campos se registrarán en variables de sesión y se visualizará una nueva página que tendrá el color de fondo elegido y que presentará el nombre de usuario. Además, esta página, contendrá un enlace a una segunda página a la que deberán propagarse

en el navegador del cliente- una cookie que tendrá una caducidad de 10 minutos (el valor 10 de `session_set_cookie_params`).

Esa cookie se guardará en el mismo directorio (/) donde se guardan las páginas web visitadas (el famoso C:\WINDOWS\Temporary Internet Files, en la configuración por defecto de IE), pero..., eso solo ocurrirá si -tal como ves en el ejemplo- pones en el parámetro **dominio** el **nombre real del dominio donde está alojada la web**.

Si el nombre de dominio no coincide con que alberga la página –razonable criterio de seguridad– **no se guardará la cookie**.

```
<?
session_set_cookie_params (10,"/","localhost", 0);
session_name('mi_sesion');
session_start();
echo session_id(),"<br>";
echo session_name(),"<br>";
$a=session_get_cookie_params();
foreach($a as $c=>$v){
    echo $c,"--->",$v,"<br>";
}
?>
<A Href="ejemplo125.php">Volver a llamar esta página</A>
```

ejemplo125.php

```
<?
session_set_cookie_params (10,"/","localhost", 0);
session_name('mi_sesion');
session_start();
echo session_id(),"<br>";
echo session_name(),"<br>";
$a=session_get_cookie_params();
foreach($a as $c=>$v){
    echo $c,"--->",$v,"<br>";
}
?>
<A Href="ejemplo126.php?<?=$SID ?>">Volver a llamar esta página</A>
```

ejemplo126.php

```
<?
session_set_cookie_params (10,"/","localhost", 0);
session_name('mi_sesion');
session_start();
echo session_id(),"<br>";
echo session_name(),"<br>";
$a=session_get_cookie_params();
foreach($a as $c=>$v){
    echo $c,"--->",$v,"<br>";
}
?>
<A Href="ejemplo127.php?<?echo session_name()."=".session_id() ?>">
    Volver a llamar esta página</A>
```

ejemplo127.php

### ¡Cuidado!

Antes de empezar a desarrollar el ejercicio que te proponemos al margen, te sugerimos que leas los contenidos que hemos incluido en la página siguiente.

Anterior



Índice



Siguiente



los valores anteriores



Ver índice

## Sesiones (II)



### Variables de sesión

La verdadera utilidad de trabajar con sesiones estriba en la posibilidad de propagar junto con ellas los valores de las variables de sesión.

Se trata de ir añadiendo y propagando variables con sus valores, y de la posibilidad de utilizarlas de la misma forma que se utilizarían las variables externas enviadas a través de un formulario.

Igual que ocurría en caso de los formularios, también las variables de sesión pueden ser tratadas de forma distinta según como estén configurado PHP (`php.ini`) y cual sea la versión de PHP que utilicemos.

### Register globals

Las variables de sesión tienen, como ocurría en otros casos, una sintaxis común (que no depende de la configuración de la directiva register globals) y una específica –añadida a la anterior– para el caso en que register globals esté activado.

A diferencia de los casos que hemos visto anteriormente, cuando se trata de sesiones no existe la similitud sintáctica que existía en aquellos casos.

Esa es la razón por la que no vamos a incluirla. Su utilidad práctica es nula y podría crearnos cierta confusión.

El hecho de que PHP mantenga activas las funciones de esa opción obedece únicamente a razones de tipo práctico. La decisión de eliminarlas de las nuevas versiones podría ocasionar serios perjuicios a programadores que tienen desarrolladas sus aplicaciones con la antigua sintaxis y que se verían obligados a modificar el código fuente de todos sus scripts anteriores.

### Manejo de variables

Las funciones más importantes para el manejo de variables de sesión son las siguientes:

#### `$_SESSION['var']`

es una de las formas de definir una variable de sesión.

El índice var debe contener otra

### Con cualquier opción de `register_globals`

Aquí tenemos un ejemplo en el que utilizamos las funciones PHP específicas para el tratamiento de sesiones.

En el caso de que la versión de PHP no admitiera **superglobales** habría que sustituir `$_SESSION` por `$HTTP_SESSION_VARS` y tener presente el carácter no global de esta última variable.

```
<?
# iniciamos la sesión
session_start();
# visualizamos el identificador de sesión
echo "Este es el identificador de sesión: ",session_id(),"<br>";
# registraremos una variable de sesión asignandole un nombre
$_SESSION['variable1'];
#asignamos un valor a esa variable de sesión
$_SESSION['variable1']="Filiberto Gómez";
# registraremos una nueva variable de sesión
#asignandole directamente un valor
$_SESSION['variable2']="Otro filiberto, este Pérez";
#comprobamos la existencia de la variables de sesión
echo "Mi_variable1 esta registrada: ",
isset($_SESSION['variable1']),"<br>";
#leemos el contenido de esa variable
print "Su valor es: ".$_SESSION['variable1']."<br>";
#comprobamos la existencia de la otra variable y la visualizamos
echo "Mi_variable2 esta registrada :",
isset($_SESSION['variable2']),"<br>";
print $_SESSION['variable2']."<br>";
#destruimos la variable1
unset($_SESSION['variable1']);
echo "La variable1 ha sido destruida:",
isset($_SESSION['variable1']),"<br>";
print $_SESSION['variable1']."<br>";
#destruimos todas las variables restantes
unset($_SESSION);
#comprobamos que han sido destruidas

echo "La variable1 ya estaba vacia:",
isset($_SESSION['variable1']),"<br>";
print $_SESSION['variable1']."<br>";

echo "También ha sido destruida la variable2: ",
$_SESSION['variable2'], "<br>";
print $_SESSION['variable2']."<br>";

?>
```

ejemplo128.php

### Propagación de sesiones

Los tres scripts siguientes son un ejemplo del uso de sesiones para la propagación de sesiones.

Funcionan bajo cualquier forma de `register_globals` y también en el caso en que las cookies estuvieran desactivadas en el navegador del cliente

```
<?
/* recuerda que entre <? y la primera linea no puede haber
línneas en blanco ni tampoco puede haberla encima de <?
aunque como en este caso, si admite líneas de comentario
pero no líneas en blanco */
```

comillas— el nombre que pretendamos asignarle a esa *variable de sesión*.

Si la variable ya existiera le reasignaría un valor nulo.

### \$HTTP\_SESSION\_VARS['v']

Es complementaria de la anterior en el caso de que PHP acepte variables *superglobales*.

En el caso de que no fueran aceptadas sería la opción alternativa a aquella.

Ambas se comportan igual que cualquier otra variable. Tanto si existieran previamente como si no hubieran sido creadas anteriormente le asignaría el *valor* indicado.

**unset(\$\_SESSION);**

La función **unset** destruye las variables contenidas en el paréntesis. En este caso, al contener el array **\$\_SESSION** destruiría todas las variables contenidas en él.

**unset(\$\_SESSION['var']);**

Es similar a la anterior. En este caso solo sería destruida la variable de sesión indicada en *var*.

**isset(\$\_SESSION['var']);**

La función **isset** —no es específica del tratamiento de sesiones— devuelve un valor *booleano* (UNO ó NUL) según que exista o no exista la variable contenida en el paréntesis. De hecho, se comporta con las variables de sesión de forma idéntica a como lo haría con cualquier otro tipo de variable.

## Propagación de las variables

Las variables **\$\_SESSION['var']** creadas en cualquier página, se propagan a todas las demás páginas a las que se propague la sesión, sin que para ello sea necesaria ninguna actuación específica.

Bastará con propagar la sesión siguiendo los procedimientos descritos en la página anterior para que las variables sean transferidas automáticamente.

Esa es la verdadera utilidad de este tipo de variables.

Recuerda que para transferir otros tipos de variables teníamos que recurrir a formularios o a incluir todas las parejas *nombre=valor* en la dirección de la página que habría de recibirlas. No cabe duda que este método añade un gran factor de comodidad y utilidad a la transferencia de información entre páginas del mismo espacio de servidor.

```
# deactivamos la opcion de que las páginas puedan guardarse
# en la cache del navegador del cliente
session_cache_limiter('nocache,private');
# le asignamos un nombre a la sesión
# aunque lo habitual sería dejar el nombre por defecto
# que le asigna la configuración de php.ini
session_name('pruebas');
# iniciamos la sesión
session_start();
# creamos variables de sesión y les asignamos valores
$_SESSION['valor1']=25;
$_SESSION['valor2']="Ambrosio de Morales";
$_SESSION['variable3']="Una prueba más";
/* cerramos el script e insertamos un enlace a otra página
y propagamos la sesión incluyendo en la llamada
el nombre de la sesión y su identificador
En esta página no se visualizaría nada. Solo el enlace */
?><A Href="ejemplo130.php?<?echo session_name()." .=".session_id() ?>">
Propagar la sesión</A>
```

ejemplo129.php

```
<?
/* pese a que la sesión viene de la página anterior
tenemos que poner nuevamente session_cache_limiter
ya que esta instrucción no se conserva
solo es válida para la página en la que está definida
También tenemos que poner en session_name el mismo
nombre de la página anterior, de no hacerlo
PHP entendería que se trata de iniciar una sesión distinta
Por último también debemos iniciar la sesión
es obligatorio iniciarla */
session_cache_limiter('nocache,private');
session_name('pruebas');
session_start();
/* comprobaremos que la sesión se ha propagado
visualizando el array asociativo $_SESSION
que contiene todas las variables de sesión */

foreach($_SESSION as $indice=>$valor){
    print("Variable: ".$indice." Valor: ".$valor."<br>");
}
/* modificamos los valores de las variables de sesión
de igual forma que si fueran variables de cualquier otro tipo

$_SESSION['valor1']+87;
$_SESSION['valor2'] .=" bonito nombre";
/* destruimos la tercera variable
unset($_SESSION['variable3']);

/* propagamos la sesión a la página siguiente
con idéntico proceso al del script anterior
?><A Href="ejemplo131.php?<?echo session_name()." .=".session_id() ?>">
Propagar la sesión</A>
```

```
<?
# idénticos comentarios a los anteriores
session_cache_limiter('nocache,private');
session_name('pruebas');
session_start();
# este bucle nos confirmará que se han propagado
# los nuevos valores y que la tercera variable ha sido destruida
foreach($_SESSION as $indice=>$valor){
    print("Variable: ".$indice." Valor: ".$valor."<br>");
}
?>
```

de más comentar que las variables de sesión no se transmiten más que entre páginas alojadas en el mismo espacio de servidor.

Anterior



Índice



Siguiente





Ver índice

# Crear ficheros PDF



## Algunas posibilidades

Existen diferentes posibilidades de creación de ficheros PDF mediante el uso de funciones de PHP. La distribución de PHP incluye funciones para la creación de ficheros usando las bibliotecas PDFlib de *Thomas Merz* que están disponibles en <http://www.pdflib.com/>.

La distribuciones de PHP para Windows incluyen estas librerías restringiendo su uso a actividades no comerciales. Cuando se trata de Linux/Unix la compilación y utilización de esta biblioteca requiere *su compra*. Por esa razón son muchos los hostings que **no incluyen** entre sus servicios la posibilidad de uso de esta librería.

Una solución alternativa y **gratuita** es el uso de la **clase FPDF** que permite generar documentos PDF directamente desde PHP sin necesidad de utilización de ninguna librería externa.

En <http://www.fpdf.org> podrás encontrar las últimas versiones de esta clase así como información y documentación relacionada con su utilización.

Dado el carácter **freeware** de esta clase y las posibilidades que ofrece para ser modificada y complementada, será la que utilizaremos para desarrollar los contenidos de este tema.

## Creación de ficheros PDF

La creación de ficheros PDF requiere que el script incluya la clase **fpdf.php**. Para ello bastará con que contenga la siguiente instrucción:

```
include("fpdf.php")
```

Si el fichero fpdf.php estuviera en un directorio distinto del actual habría de escribirse la ruta completa del mismo en la instrucción anterior.

Además de esto, para poder usar la tipografía propia de la clase será necesario asignar a una **constante**, con nombre **FPDF\_FONTPATH** (el nombre ha de mantenerse ya que es que usa en la clase **fpdf**), la ruta absoluta hasta el directorio que contiene las fuentes y que en nuestro caso será el directorio **fotspdf** tal como comentamos a la derecha.

La asignación de valores a la

## Usando las librerías PDFLib

Ver índice

Búsqueda rápida

◀ Página anterior

Página siguiente ▶

Hemos de **descomentar** la línea que dice:

```
;extension=php_pdf.dll
```

y como **descomentar** no es otra cosa que quitar el punto y coma que va delante de una línea, habremos de dejarla así:

```
extension=php_pdf.dll
```

Una vez guardados los cambios habrá que –como siempre en estos casos– volver a iniciar el servidor Apache.

Para comprobar si está activa la modificación del php.ini que comentamos a la izquierda- bastaría con visualizar el **famosísimo info.php** y comprobar que aparece lo siguiente:

PDF Support	enabled
PDFlib GmbH Version	4.0.2
Revision	\$Revision: 1.112.2.7 \$

Si es así, ya estamos en disposición de empezar a trabajar con este nuevo tipo de funciones PHP.

La visualización de ficheros con extensión PDF requiere tener instalado el programa **Adobe Acrobat Reader**. En caso de que nos dispusieras de él puedes descargar desde el enlace incluido en este párrafo.

Desde enlace, <http://www.pdflib.com/products/pdflib/download/index.html> tienes acceso a la descarga de estas librerías y también a la documentación relativa a la forma de utilización de las mismas.

### ¡Cuidado!

Para el desarrollo de los contenidos de este curso no utilizaremos la librería PDFLib. Por esa razón no es necesario que efectuemos los cambios de configuración descritos en el párrafo anterior.

La clase FPDF -que será la que utilizaremos- **no requiere** ninguna modificación en la configuración de php.ini.

## La clase FPDF

Al realizar la descarga del fichero **fpdf16.zip** desde <http://www.fpdf.org> podremos observar que el zip incluye tres directorios: **doc**, **font** y **tutorial** y una serie de ficheros entre los que podremos encontrar **fpdf.php** que es el que realmente contiene la clase.

Hemos incluido, en el directorio **cursophp** de estos materiales, el fichero **fpdf.php** de esa distribución y también el directorio **font** que hemos renombrado como **fotspdf** al efecto de facilitar la identificación.

El resto de los ficheros, son meramente informativos, no los hemos incluido ya que no van a resultarnos necesarios para el uso de esta clase. Si tratáramos de utilizar esta clase en un servidor remoto deberíamos transferir al mismo, con estos o diferentes nombres, los mismos ficheros y directorios que hemos incluido junto con estos materiales.

constante la haremos mediante la sintaxis:

```
define('FPDF_FONTPATH','ruta')
```

## El constructor FPDF

La clase incluida en el fichero **fpdf.php** tiene por nombre **FPDF**. Por tanto su uso requerirá la creación de un nuevo objeto mediante la sintaxis:

```
$objeto= new FPDF()
```

Al crear el nuevo objeto se ejecuta siempre el *constructor* (recuerda que un constructor es una función con idéntico nombre que la clase que lo contiene -en este caso FPDF- que se ejecuta de forma automática el momento en que es creado un nuevo objeto) que utiliza tres parámetros: *orientacion*, *unidad de medida* y *formato*.

## Orientación, medidas y formato

Al crear un nuevo objeto pueden incluirse los valores de todos o parte de estos parámetros. En ese caso la sintaxis sería:

```
$obj=new FPDF(ort,unds,tam)
```

La *orientacion*(*ort*) permite dos valores: **P** (**normal** ó *Portrait*) y **L** (**apaisado** ó *Landscape* en denominación inglesa). El valor por defecto es **P** (**normal**).

El *unds* (unidad de medida a utilizar) permite dos valores: **in** (**pulgadas**), **pt** (**puntos**), **mm** (**milímetros**) y también **cm** (**centímetros**). El valor por defecto es **mm** (**milímetros**).

Recuerda que **una pulgada** equivale a **25,4 milímetros** y que **un punto** equivale a **1/72 pulgadas** (0,35277 mm.).

Respecto a los formatos, los valores pre establecidos son los siguientes:

Valor	Dimensiones (mm.)
A4	210x297
A5	148,5x210
A3	297x420
Legal	215,9x355,6
Letter	215,9x279,4

Pueden crearse documentos PDF con dimensiones distintas a las anteriores. Para ello sería necesario crear un array con las dimensiones pretendidas

```
$medidas=array(ancho,alto)
```

## Los tipos de letra

Por defecto el directorio **font** que se incluye en la distribución de <http://www.fpdf.org> -que nosotros hemos renombrado como **fontspdf**- incluye las tipografías: *Courier*, *Helvetica* y *Times*, (permitiendo utilizar el nombre *Arial* como sinónimo de *Helvetica*) y las fuentes de símbolos: *Symbol* y *Zapfdingbats*.

Sin embargo, es posible usar cualquier otra tipografía. En páginas siguientes veremos la forma en que pueden generarse e incluirse nuevos tipos de letra mediante la utilidad **makefont** incluida en el propio directorio **fontspdf**.

## El primer PDF

Como primer ejemplo crearemos un fichero con dos páginas en *blanco*.

Una vez creado el objeto se **añaden las páginas** mediante la función (incluida en la clase FPDF) **AddPage()** que debe ser invocada mediante la sintaxis:

```
$objeto->AddPage();
```

```
<?
# incluimos la clase fpdf que está en este mismo directorio
include("fpdf.php");
# y definimos la constante FPDF_FONTPATH como la ruta absoluta
# hasta el directorio que contiene las fuentes tipográficas
define('FPDF_FONTPATH',$_SERVER['DOCUMENT_ROOT'].'/cursophp/fontspdf/');
# creamos un nuevo objeto (MiPDF) utilizando la clase FPDF
$MiPDF=new FPDF();
# creamos una página en blanco
$MiPDF->Addpage();
# creamos una segunda página en blanco
$MiPDF->Addpage();
# visualizamos el documento
$MiPDF->Output();
?>
```

ejemplo132.php

## Dimensionado y orientación de documentos PDF

En este ejemplo establecemos dimensiones, unidades de medida y orientación del documento. Comprobaremos que podemos cambiar la orientación (de normal a apaisada o viceversa) de cada una las páginas del documento.

Las modificaciones de orientación de las páginas pueden realizarse incluyendo el tipo de orientación (L ó P) de esa página concreta en la llamada al método **AddPage()**. La sintaxis sería: **\$objeto->AddPage('L')** para el caso de apaisado ó **\$objeto->AddPage('P')** si se tratara de orientación *normal*.

Cuando se añade una página sin especificar la orientación en **AddPage()** la nueva página tendrá la orientación indicada en la llamada al constructor del nuevo objeto (**\$objeto= new FPDF('orientacion','unidades','tamaño')**).

```
<?
# incluimos la clase fpdf que está en este mismo directorio
include("fpdf.php");
# y definimos la constante FPDF_FONTPATH como la ruta absoluta
# hasta el directorio que contiene las fuentes tipográficas
define('FPDF_FONTPATH',$_SERVER['DOCUMENT_ROOT'].'/cursophp/fontspdf/');
/* vamos a configurar el documento como apaisado (P), utilizando
las pulgadas como unidad de medida y unas dimensiones "no estandar"
de 10 x 20 pulgadas */

# creamos un array con las dimensiones (ancho y alto);
$dimensiones=array (10,20);
# creamos un nuevo objeto (MiPDF) utilizando la clase FPDF
# incluyendo en este caso los valores a utilizar por el constructor
$MiPDF=new FPDF('P','in',$dimensiones);
# creamos una página en blanco. Incluimos, para esta primera página
# un cambio de orientación respecto a la inicial
```

e incluir esa variable como parámetro -puedes verlo en el ejemplo de la derecha- en la creación del nuevo objeto.

## El método Output()

La clase FPDF incluye la función Output para poder visualizar o guardar el documento creado. Utiliza dos parámetros: *nombre del fichero PDF* y *destino del documento*. La sintaxis podría ser la siguiente:

```
$obj->Output(nomb,dest)
```

Puede asignarse cualquier nombre, con o sin extensión, tal como puede verse en el ejemplo. Cuando no se indica el nombre, la función asigna por defecto el de **doc.pdf**.

El parámetro destino puede tomar uno de los siguientes valores:

**I** permite visualizar el fichero directamente en el navegador si el plugin si está disponible. La opción «Guardar como...» asignaría, por defecto, el nombre especificado en el parámetro *nomb*.

**D** envía el fichero al navegador y muestra ventana de opción que permite elegir entre *Abrir* o *Descargar*. Si se elige esta última guardará el fichero con el nombre especificado en el parámetro *nomb*.

**F** guarda el fichero en el directorio actual del servidor con el nombre asignado en la opción **nomb**.

## El método SetDisplayMode()

Este método -sólo afecta a la forma en la se visualiza el documento en la pantalla del cliente- permite configurar dos parámetros: *tamaño* y *forma de visualización*.

El primero puede usar una de estas opciones: **fullpage** (el zoom del visor se adapta de modo que encaje la página completa en la pantalla); **fullwidth** (ajusta el zoom de forma que se visualize el documento ajustando el ancho al de la pantalla); **real** (ajusta el zoom de visor al 100%) ó **default** que usa el modo por defecto del visor.

Mediante el segundo parámetro se puede establecer la forma de visualización. La opción **single** muestra las páginas separadas de una en una; **continuous** va mostrándolas una detrás de otra de forma continua; **two** muestra dos columnas (con dos páginas) simultáneamente, y **default** que, como en el caso anterior, usa el

```
$MiPDF->Addpage('L');
# creamos una segunda página en blanco
# en la que, al no incluir el parámetro de orientación
# utilizará el valor utilizado por el constructor.
$MiPDF->Addpage();
# visualizamos el documento
$MiPDF->Output();
?>
```

ejemplo133.php

## Un ejemplo un poco más completo

Aunque no tienen excesiva utilidad práctica, vamos a comentar someramente aquí -también los incluimos en el ejemplo- algunos de los métodos, de carácter básicamente informativo, que incluye la clase FPDF.

**SetAuthor("nombre del autor")** Permite incluir una cadena con el nombre del autor.

**SetCreator("nombre de la aplicación")** Permite incluir una cadena con el nombre del programa utilizado para crear el documento.

**Title("título del documento")** Permite incluir una cadena con el título del documento.

**SetKeywords("palabras clave")** Permite incluir una cadena con una lista de palabras clave separadas por espacios.

Los datos incluidos en el documento mediante los métodos anteriores no son presentados directamente en el documento. Sólo son visualizables cuando se exploran los *metadatos* del documento.

```
<?
# incluimos la clase fpdf que está en este mismo directorio
include("fpdf.php");
# y definimos la constante FPDF_FONTPATH como la ruta absoluta
# hasta el directorio que contiene las fuentes tipográficas
define('FPDF_FONTPATH', $_SERVER['DOCUMENT_ROOT'].'/cursophp/fontspdf/');
/* vamos a configurar el documento como apaisado (P), utilizando
los milímetros como unidad de medida y unas dimensiones "no estandar"
de 140 x 200 milímetros */
# creamos un array con las dimensiones (ancho y alto);
$dimensiones=array (140,200);
# creamos un nuevo objeto (MiPDF) utilizando la clase FPDF
# incluyendo en este caso los valores a utilizar por el constructor
$MiPDF=new FPDF('P', 'mm', $dimensiones);
# el método SetAuthor nos permite incluir el nombre del autor
$MiPDF->SetAuthor('Pepe Pérez');
# el método SetCreator nos permite incluir el nombre de la
# aplicación que genera el pdf
$MiPDF->SetCreator('clase FPDF');
# el método Title nos permite incluir un título
$MiPDF->SetTitle('Pruebas del pdf');
# el método SetKeywords nos permite incluir palabras claves
# separadas por espacios y dentro de una misma cadena
$MiPDF->SetKeywords('palabra1 palabra2');
# el método SetDisplayMode nos permite incluir palabras claves
# separadas por espacios y dentro de una misma cadena
$MiPDF->SetDisplayMode('fullpage','two');
# creamos una página en blanco. Incluimos, para esta primera página
# un cambio de orientación respecto a la inicial
$MiPDF->Addpage('L');
# creamos una segunda página en blanco
# en la que, al no incluir el parámetro de orientación
# utilizará el valor utilizado por el constructor.
$MiPDF->Addpage();
# visualizamos el documento
$MiPDF->Output('donpepito.pdf', 'I');
?>
```

Los ejemplos siguientes son similares. La única modificación que contienen respecto al código

modo por defecto del visor.

fuente anterior es la correspondiente al segundo parámetro (destino) del método Output.

<a href="#">Destino="I"</a>	<a href="#">Destino="D"</a>	<a href="#">Destino="F"</a>
-----------------------------	-----------------------------	-----------------------------

---

[Anterior](#)

[Índice](#)

[Siguiente](#)



Ver índice

## Textos en PDF



### Fijación de márgenes

Algunas funciones de inserción de textos permiten colocar los mismos dentro de lo que podría llamarse *caja de texto* que no es otra cosa que la superficie de la página delimitada por los márgenes de la misma.

Por defecto, estos márgenes miden **un centímetro**, por la izquierda, la parte superior y la derecha del documento. El tratamiento del margen inferior requiere una función específica que veremos en otro de los epígrafes de esta página.

#### **\$obj->SetMargins(iz,su,de)**

Los parámetros **iz**, **su** y **de** son los valores numéricos que permiten establecer, -en la unidad de medida que se indique en el constructor- los márgenes izquierdo, superior y derecho, por este orden, del documento.

Si se omite el tercer parámetro (margen derecho) se le asignará el valor que se haya indicado para el izquierdo.

### Margen inferior y salto de página

La función:

#### **\$obj->SetAutoPageBreak(a,b)**

permite activar o desactivar el salto de página automático y establecer el margen inferior (mínimo) de las páginas. El parámetro **a** puede tomar dos valores: **1** ó **0** (cierto ó falso), que se corresponden con la condición **activo/inactivo**.

Cuando está **activo** se van añadiendo nuevas páginas, de forma automática, hasta que se incluya la totalidad del texto especificado. En el caso de estar **inactivo**, los textos que excedieran los límites de la página no se visualizarían ya que no se añadiría una nueva página.

Por defecto está opción está **activada**.

El segundo parámetro, el valor **b**, permite indicar el margen inferior mínimo de la página. Su valor -numérico- deberá indicarse en las unidades establecidas por el constructor. Si no se indica asumirá, por defecto, un valor igual a **dos centímetros**.

*Color del texto*

### Inserción de textos

La forma más simple de inserción de textos –poco utilizada por el número de inconvenientes que conlleva– es la que utiliza la función:

#### **\$obj->Text(x,y,texto)**

donde **x** e **y** son los valores numéricos que representan las coordenadas del punto de la página en el que pretende realizarse la inserción y **texto** es una cadena (o una variable conteniendo una cadena) con el texto que se pretende insertar.

Esta función no considera ni los eventuales *saltos de línea* que pudiera haber en la cadena (inserta todo el texto en una sola línea) ni tampoco los márgenes del documento. Por ello puede ocurrir, tal como puedes comprobar en el ejemplo, que, por desbordar los márgenes del documento, no aparezcan los textos completos.

Una función mucho más útil es la siguiente:

#### **\$obj->Write(interlinea,texto)**

en la que **interlinea** es el interlineado (expresado en la misma unidad utilizada por el constructor) y **texto** la cadena o variable que contiene el texto a insertar.

Se comporta de la siguiente forma:

Comienza a escribir el texto en la **posición actual de punto de inserción**.

La posición actual del punto de inserción. puede ser una de las siguientes:

La esquina superior izquierda del área de impresión de la página (en el caso en que se ha producido un salto de página y no se hubieran insertado elementos anteriormente).

La posición resultante del final de la inserción del el elemento anterior.

La posición establecida como posición actual mediante las funciones **SetX**, **SetY** ó **SetXY**

La inserción de textos mediante la función **Text()** no modifica la localización del punto de inserción ya que utiliza sus propias coordenadas de posicionamiento.

Una vez que la línea alcanza el margen derecho -establecido de forma explícita o por defecto- de la caja de texto produce un salto de línea automático.

Incluye los saltos de línea que encuentra en el propio texto (caso, por ejemplo, de inclusión de ficheros de texto) y también aquellos incluidos en la propia cadena mediante la secuencia de escape **\n**.

Si la opción **AutoPageBreak** (comentada al margen) está activada produce un salto de página automático al alcanzar el margen inferior de la página.

Alinea los textos a la izquierda sin permitir modificar la configuración de esta opción

```
<?
#incluimos el fichero con la clase y definimos la variable FPDF_FONTPATH
# con el mismo criterio comentado en el ejemplo anterior
include("fpdf.php");
define('FPDF_FONTPATH', $_SERVER['DOCUMENT_ROOT'].'/cursophp/fontspdf/');
/* establecemos las dimensiones del documento
   creamos un nuevo objeto y añadimos una página*/
$dimensiones=array (140,200);
$MiPDF=new FPDF('P','mm',$dimensiones);
# ajustamos al 100% la visualización
$MiPDF->SetDisplayMode('real');

#insertamos una página en blanco
$MiPDF->Addpage();
# establecemos el color de la letra
$MiPDF->SetTextColor(255,0,0);
```

El color del texto se establece utilizando la siguiente función:

**\$obj->SetTextColor(R,G,B)**

donde R, G, B son valores numéricos comprendidos entre **0** y **255** y que representan las respectivas componentes de los colores primarios: **rojo, verde y azul**. Si se utiliza con un sólo parámetro:

**\$obj->SetTextColor(K)**

el valor numérico, también comprendido entre **0** y **255**, establece colores de la *escala de grises* desde el negro (cero) hasta el blanco (255).

Si no se especifica color los textos aparecerán en negro.

Una vez establecido un color su valor se mantiene hasta que no sea invocada nuevamente la función **SetColor** para su modificación.

## Tipo de letra, estilo y tamaño

Podemos asignar estos tres valores mediante la función:

**\$obj->SetFont(nom,est,tam)**

donde **nom** es una cadena que debe indicar el nombre del tipo de letra elegido (**Helvetica, Arial, Times, Zapfdingbats, Courier** ó **Symbol** son las fuentes por defecto).

El estilo (est) puede indicarse mediante la *cadena vacía ("")* para el tipo **normal** o mediante **B, I, U** (**negrita, cursiva, subrayada**) o sus combinaciones, por ejemplo **BU** para **negrita subrayada**.

Las fuentes de símbolos sólo permiten el estilo **normal**. El parámetro **tam** especifica el tamaño de la letra expresado en **puntos**. Si no se especifica, el valor por defecto es 12 puntos.

Más adelante, dedicamos un apartado a la creación y utilización de nuevas fuentes tipográficas.

## Posición actual

La clase FPDF dispone de funciones que determinan el valor actual de las coordenadas del punto de inserción (lugar donde se va a posicionar el texto, imagen o elemento gráfico que se incluya) y otras que permiten modificar esos puntos de inserción.

Son las siguientes:

**\$obj->GetX()**

y

**\$obj->GetY()**

que devuelven los valores la

```
# establecemos la fuente a utilizar, estilo
# y tamaño en puntos. Al no incluir estilo
# considerará el estilo normal
$MiPDF->SetFont('Arial','','11');
# creamos una variable en la que incluimos un texto
$texto="Este es el texto que tendremos que escribir aquí dentro.\n";
$texto.="Esto habría de ir detrás de un salto de línea. Veremos si funciona
$texto.="Estamos alargando la línea para comprobar el funcionamiento de ";
$texto.="las diferentes funciones";
# usamos el método Text para presentar el contenido de la variable
# situamos las coordenadas de inserción en el punto (30,40) mm.
# podremos observar que la función Text inserta todo el texto en
# una linea sin tener en cuenta si sobrepasa o no el borde del papel
# ni tampoco los caracteres de salto de linea
$MiPDF->Text(30,40,$texto);
# cambiamos el color de la letra. Al pasar un solo parámetro
# será interpretado como escala de grises 0 -255 (0 negro, 255 blanco)
$MiPDF->SetTextColor(200);
# cambiamos el tipo de letra (Helvetica y Arial son alias de la misma fuente)
$MiPDF->SetFont('Helvetica','B',11);
# utilizaremos la función Write para escribir el texto. Lo hará con
# alineación a la izquierda, interpretando los saltos de línea
# y efectuando uno automático cuando alcanza el margen.
# El valor de la interlinea se expresa en milímetros.

# Dado que la única inserción ha sido mediante Text() que no modifica el
# punto de inserción el resultado de este Write se incluirá al comienzo
# de la página, respetando los márgenes por defecto.
$MiPDF->Write(4,$texto);

# Comprobaremos los tipos de letra existentes en el directorio fonts.
$MiPDF->SetFont('Symbol','','11');
/* insertamos un salto de línea de 8 unidades que producirá
un desplazamiento del punto de inserción hasta el margen izquierdo
de una línea situada 8 unidades por debajo de la posición del final
de la cadena insertada anteriormente */
$MiPDF->Ln(8);
# escribimos el texto anterior con la letra Symbol
$MiPDF->Write(4,$texto);
# cambiamos nuevamente el tipo de letra
$MiPDF->SetFont('zapfdingbats','','11');
# insertamos un nuevo salto de párrafo de 8 unidades
# y repetimos la impresión del texto con la nueva tipografía
$MiPDF->Ln(8);
$MiPDF->Write(4,$texto);
# establecemos márgenes (izquierda, superior, derecha)
# expresados en la unidad usada por el constructor (mm).
# Serán utilizados en las páginas siguientes
$MiPDF->SetMargins(30,40,25);
# establecemos el modo de inserción automática de página
# indicando el margen inferior mínimo. El salto automático
# no sería necesario ya que, si no se especifica, se configura
# por defecto como tal
$MiPDF->SetAutoPageBreak(1,8);

# Insertamos una nueva página. En adelante
# los contenidos aparecerán en esa nueva página
$MiPDF->Addpage();
# cambiamos el color y tipo de letra
$MiPDF->SetTextColor(0,0,200);
$MiPDF->SetFont('Times','I',12);
# insertamos texto mediante Write y observaremos
# que ahora respeta los nuevos márgenes
$MiPDF->Write(4,$texto);
# posicionamos el punto de inserción en las coordenadas
# (2,2) y escribimos de nuevo el texto.

$MiPDF->SetXY(2,2);
# al visualizar el documento podremos observar que solo
# respeta estas coordenadas la primera línea, ya que las siguientes
# se ajustan a los márgenes establecidos para la página
$MiPDF->Write(4,$texto);
# redimensionamos nuevamente los márgenes
$MiPDF->SetMargins(20,5,15);

# cambiamos a una nueva página
$MiPDF->Addpage();
```

posiciones actuales, horizontal (X) y vertical (Y). El resultado estará expresado en las unidades establecidas por el *constructor* considerando como origen de coordenadas la esquina superior izquierda del documento y valores positivos -horizontal y vertical- los colocados a la derecha (X) y debajo (Y) del origen.

Para definir una nuevo punto de inserción se pueden usar cualquiera de estas funciones:

```
$obj->SetX(valor X)  
$obj->SetY(valor Y)  
$obj->SetXY(valor X, valor Y)
```

que permiten establecer el *punto de inserción* mediante sus coordenadas: horizontal, vertical o ambas.

Cuando el valor X (en el caso SetX) es negativo se considera con referencia al margen derecho de la página. De igual modo, cuando se asigna un valor negativo en SetY se considera respecto a la parte inferior de la página.

## Saltos de línea

Mediante **\$obj->Ln (valor)** se ejecuta un salto de línea. La posición actual regresa al margen izquierdo y la vertical se desplaza en el número de unidades indicado en el parámetro *valor*.

```
# cambiamos el estilo de fuente a "normal"  
$MiPDF->SetFont('Times','',12);  
# leemos el fichero de texto y lo recogemos en una variables  
$f1=fopen('regenta.txt','r');  
$regental=fread($f1,filesize('regenta.txt'));  
fclose($f1);  
# insertamos el fichero mediante write  
# podremos observar como se realiza el salto de página  
# de forma automática respetando los márgenes inferiores  
$MiPDF->Write(4,$regental);  
  
# inseraremos una nueva página  
$MiPDF->Addpage();  
# desactivamos el salto de página automatico  
# con lo cual no será respetado el margen inferior  
# y el texto rebasará ahora los límites del papal  
$MiPDF->SetAutoPageBreak(0);  
#ponemos el texto en negro y cursiva para poder  
#diferenciar los resultados al visualizar el documento  
$MiPDF->SetTextColor(0);  
# reescribimos el texto anterior  
$MiPDF->Write(4,$regental);  
# visualizamos el documento  
$MiPDF->Output();  
?>
```

ejemplo137.php

Anterior

Índice

Siguiente

## Textos recuadrados

La función **Cell** ofrece un montón de posibilidades a la hora de insertar *celdas* conteniendo textos. Su sintaxis es la siguiente:

`$obj->Cell(a,h,text,b,q,p,r,e)`

dónde los parámetros son los siguientes: **a** es el ancho de la celda; **h** su altura; **text** la variable o cadena que contiene el texto a insertar; **b** un parámetro que establece los bordes de la celda y que puede tomar los siguientes valores:

Valor	Opción
0	Sin bordes
1	Con bordes
L	Borde por la izquierda
T	Borde superior
R	Borde por la derecha
B	Borde inferior

El valor por defecto es cero. Las especificaciones de bordes laterales pueden agruparse (LR, TL, etcétera sin que importe el orden en que se haga la agrupación).

El parámetro **q** permite establecer en qué posición se habrá de insertar el elemento posterior a la celda actual (otra celda, un texto mediante Write, un gráfico, etcétera). Permite los siguientes valores:

Valor	Opción
0	A la derecha de la celda actual
1	En el margen izquierdo de la línea siguiente
2	Debajo de la celda actual alineado a su borde izquierdo

Por medio del parámetro **p** se puede establecer la alineación del texto (contenido en la celda) respecto a sus bordes. Permite usar **L**, **C** y **R** como indicadores de alineaciones del texto a la izquierda, centro ó derecha.

El parámetro **r** es un valor booleano (**1** ó **0**) que especifica si la celda ha de llenarse con un **color de fondo** (**1**) o ha de ser **transparente** (**0**).

El parámetro **e** -es opcional- permite establecer la celda como un

## Enlaces internos

En el ejemplo que tienes a continuación hemos insertado algunas de las opciones de las funciones comentadas al margen. En los párrafos siguientes incluimos algunos comentarios sobre la manera de utilizar las funciones de inclusión *enlaces internos* en el documento. Esta opción requiere tres pasos: Crear una referencia interna y recogerla en una variable identificadora.

La sintaxis sería la siguiente: `$enlace=$obj->AddLink()`

Definir la zona del documento (área, imagen o texto) en la que, al pulsar sobre ella, se activará el redireccionamiento. Podría hacerse de varias formas, entre otras:

`$obj->Cell(a,h,text,b,q,p,r,$enlace)`

Mediante esta opción se añade como último parámetro de la función Cell la variable creada mediante **AddLink()**. Si se tratara de un enlace a una URL no sería necesario AddLink y bastaría con incluir -entre comillas- la dirección de la URL. Convertiría la cadena de texto en un *hiperenlace*.

`$obj->Link(X,Y,ancho,alto,$enlace)`

Permite definir como *hiperenlace* un área de la página actual definida por un rectángulo cuya esquina superior izquierda está definida por los parámetros **X** y **Y** y cuyo ancho y alto se establecen a través de los parámetros del mismo nombre. La variable \$enlace se comporta de forma idéntica a la indicada en el caso anterior.

`$obj->Write(interlinea,'texto',$enlace)`

En este caso, añadimos un tercer parámetro a la función Write (es opcional) que contiene la variable indicadora del enlace. El comportamiento sería idéntico a los supuestos anteriores.

Establecer la posición a la que redirigiría el enlace. Será necesario indicar a dónde habrá de redirigirse el cliente en el momento que se pulse sobre uno de los enlaces anteriores. Eso se indica mediante:

`$obj->SetLink($enlace, pos_vertical, pagina)`

La variable \$enlace será el indicador mencionado en los párrafos anteriores, **pos\_vertical** la distancia del margen superior de la página de destino en la que se inicia la visualización y **pagina** el número de la página a la que se redirige mediante el enlace. Si se omiten estos parámetros se entenderá que la posición vertical es 0 y que la página es la página actual.

```

<?
#incluimos el fichero con la clase y definimos la variable FPDF_FONTPATH
# con el mismo criterio comentado en el ejemplo anterior
include("fpdf.php");
define('FPDF_FONTPATH', $_SERVER['DOCUMENT_ROOT'].'/cursophp/fontspdf/');
/* establecemos las dimensiones del documento
   creamos un nuevo objeto y añadimos una página*/
$dimensiones=array (210,297);
$MiPDF=new FPDF('P','mm',$dimensiones);
# ajustamos al 100% la visualización
$MiPDF->SetDisplayMode('real');

#insertamos una página en blanco
$MiPDF->Addpage();
# estableceremos los colores para bordes, fondos y textos
# color de borde
$MiPDF->SetDrawColor(255,0,0);
# color del relleno (gris)
$MiPDF->SetFillColor(200);
# color del texto
$MiPDF->SetTextColor(0,0,255);
# establecemos espesores de lineas y tipo y tamaño de letra
# espesor de linea 1 milímetro
$MiPDF->SetLineWidth(1);
# fuente y tamaño: Arial, negrita de 12 puntos
$MiPDF->SetFont("Arial","B",12);
# establecemos el texto para la primera celda
$celdal="Esto irá en la celda 1";

```

Si se especifica una URL se establecerá un enlace a la misma y, además, es posible establecer **enlaces internos** de la forma que puedes ver comentada en el ejemplo.

## Tamaño de una cadena de texto

La función:

**\$obj->GetStringWidth('cad')**

devuelve un número decimal que indica la longitud de la cadena indicada en **cad** con la fuente y tamaño de letra actuales. Es una función muy útil para dimensionar las celdas de modo que los textos que contienen no sobrepasen sus bordes.

## Espesor de líneas

Puede especificarse el grosor de las líneas -en los elementos gráficos que las utilizan(rectángulos, celdas, etcétera)- mediante la función:

**\$obj->SetLineWidth('grosor')**

dónde **grosor** es un valor numérico que especifica el espesor de las líneas en las unidades asignadas en el constructor. Por defecto -si no se asigna espesor mediante esta función- el ancho de línea sería de 0,2 milímetros.

## Números de página

La función:

**\$obj->PageNo()**

devuelve el número de la página actual.

## Color de las líneas

También es posible establecer el color de las líneas mediante la función:

**\$obj->SetDrawColor('R,G,B')**

dónde **R,G,B** son valores numéricos comprendidos entre **0** y **255** que asignan los valores de las componentes de los colores primarios rojo, verde y azul respectivamente. Si se incluye un único parámetro será interpretado como escala de grises. El valor **0** sería el negro y **255** representaría el blanco.

## Color de relleno

Algunos elementos gráficos (celdas de texto, rectángulos) permiten que se les aplique un color de fondo. Para establecer tal color basta con ejecutar la función:

**\$obj->SetFillColor('R,G,B')**

El criterio de asignación de colores

```
# determinamos el tamaño de esta cadena y lo recogemos
# en la variable ancho
$ancho=$MiPDF->GetStringWidth($celda1);
/* definimos la celda estableciendo:
   ancho--- igual al de la cadena que va a contener + 6 mm.
   alto --- 6 milímetros
   texto--- el contenido de la variable $celda1
   borde--- 1 (para que ponga los cuatro bordes
   celda siguiente--- 0 (para que la incluya a continuación de la actual)
   alineación --- C para que centre el texto en la celda horizontalmente
   relleno --- 1 para que aplique el fondo a la celda
   enlace--- pondremos un enlace al buscador google */

# como aun no hemos insertado ningún elemento en la página
# la celda aparecerá en la parte superior de la página y apoyada
# sobre su borde izquierdo

$MiPDF->Cell($ancho+6,6,$celda1,1,0,C,1,"http://www.google.com");

/* pero modificando insertamos una nueva celda con el mismo contenido
   modificando:
   bordes--- ahora los pondremos solo por la parte superior T e inferior E
   posición de la siguiente celda: 2 (inmediatamente debajo)
   relleno --0 */
# la nueva inserción se realizará a la derecha de la anterior
# ya que así lo especificamos al definir allí la celda siguiente
$MiPDF->Cell($ancho+6,6,$celda1,TB,2,C,0,"http://www.google.com");

# cambiamos el color del borde y el del relleno
# color de borde
$MiPDF->SetDrawColor(255,255,0);
# color del relleno (gris)
$MiPDF->SetFillColor(0,255,255);

/* una nueva inserción de una celda similar modificando
   identica a la primera celda salvo el parámetro de posición
   de la celda siguiente que cambiamos a 1 para que la celda
   siguiente aparezca en el margen izquierdo del documento.
   La actual, aparecerá inmediatamente debajo de la anterior
   ya que se así se especificó en su Cell correspondiente */

$MiPDF->Cell($ancho+6,6,$celda1,1,1,C,1,"http://www.google.com");

# la misma función anterior que ahora debería aparecer en el margen izquierdo
# modificando la especificación para que inserte la potencial celda siguiente
# inmediatamente a la derecha

$MiPDF->Cell($ancho+6,6,$celda1,1,0,C,1,"http://www.google.com");

# insertamos un salto de línea de 10 mm.
$MiPDF->Ln(10);
# una nueva celda que por efecto del salto de linea
# perdería la referencia de situarse a la derecha de la anterior
# se desplazaría 10 unidades hacia abajo y se insertaría en el margen izquierdo
$MiPDF->Cell($ancho+6,6,$celda1,1,0,C,1,"http://www.google.com");

# modificaremos ahora el punto de inserción mediante SetXY a las coordenadas
# -100, -120 mm. valores negativos
# con lo cual las referencias serán al margen derecho y al inferior

$MiPDF->SetXY(-100,-120);

# vamos a crear un enlace interno mediante la función AddLink()
# y vamos a recoger el resultado en la variable $salta
$salta=$MiPDF->AddLink();
# estableceremos la referencia del enlace
$MiPDF->SetLink($salta,0,2);
# crearemos ahora una zona (transparente) de enlace
# será el rectángulo definido como parámetros en la función Link
# es decir un rectángulo de 297mm. de ancho, 30 de alto
# que tendrá su esquina superior izquierda en el punto (0,40)
# este enlace nos llevará al lugar del documento
# que se especifique mediante SetLink($salta);
# que indique
$MiPDF->Link(0,40,297,30,$salta);
# comprobaremos que allí se inserta el texto mediante el cell correspondiente
$MiPDF->Cell($ancho+6,6,$celda1,1,0,C,1,$salta);

# insertemos un salto de página y comprobaremos la función nº de página
```

es idéntico al del párrafo anterior.

Una vez asignado un color de relleno (lo mismo ocurre con los colores y espesores de línea) se mantiene a lo largo del documento en tanto no sea modificado por la ejecución del método con unos parámetros distintos.

## Número de páginas del documento

La clase FPDF incluye la posibilidad de determinar el número total de páginas del documento. Para ello es preciso ejecutar (después de crear el objeto) la función:

`$obj ->AliasNbPages('idn')`

donde `idn` es una palabra cualquiera. Si se omite se considerará su valor por defecto que es: `{nb}`.

Para imprimir este valor en el documento será suficiente incluir el identificador utilizado en una cadena de texto. Puedes verlo comentado en el ejemplo.

```
$MiPDF->Addpage();
#añadimos a la cadena de texto el valor del número de pagina actual
$MiPDF->Cell(50,6,"pagina nº".$MiPDF->PageNo(),1,0,C,1);
# activamos el valor por defecto del señalados de número total de páginas
# al no incluir nada como parámetro de la función AliasNbPages();
# deberemos recurrir al señalador por defecto {nb}
$MiPDF->AliasNbPages();
# insertamos una nueva celda y añadimos a la cadena de texto
# anterior "de {nb}". Como puedes ver
# el señalador {nb} forma parte de la cadena como un texto más
$MiPDF->Cell(50,6,"pagina nº ".$MiPDF->PageNo()." de {nb}",1,0,C,1);

$MiPDF->Output();
?>
```

[ejemplo138.php](#)

## Ejemplo de creación de tablas

A continuación incluimos un ejemplo de utilización de estas funciones para la creación de tablas a partir de un fichero de texto.

En este ejemplo, utilizaremos saltos de página manuales y -dada la dimensión del fichero de texto- tendremos la oportunidad de comprobar el tiempo de generación de un documento de más de 200 páginas.

En páginas posteriores veremos como confeccionar la misma tabla utilizando encabezados, pies de página e inserción de páginas automáticas.

[Ver código fuente](#)

[Ver fichero de texto](#)

[Crear el PDF](#)

[Anterior](#)

[Índice](#)

[Siguiente](#)





Ver índice

# Inserción de imágenes y elementos gráficos



## Un ejemplo con gráficos e imágenes

### Líneas y rectángulos

La clase FPDF incluye métodos para el dibujo de segmentos rectilíneos y de rectángulos.

Las funciones que utiliza para estos menesteres son las siguientes:

**\$obj->Line(X1, Y1, X2, Y2)**

Si no se ha especificado ningún color -usando **SetDrawColor**- o un espesor de línea - por medio de **SetLineWidth** - se usarán los valores por defecto (color **negro** y **0,2 mm.** de espesor).

Cuando se trata de dibujar rectángulos, hemos de utilizar la función:

**\$obj->Rect(X1, Y1, A, H, 'estilo')**

donde **X1** e **Y1** son las coordenadas de la esquina superior izquierda del mismo, **A** el ancho, **H** el alto y **estilo** que puede ser una de estas tres cadenas: **D**, **F**, **DF** que significan: **dibujar líneas de borde, llenar, y dibujar líneas y llenar**. Si no se especifica estilo se interpretará por defecto la opción **D**.

### Inserción de imágenes

Se pueden incluir imágenes mediante la función:

**\$obj->Image('nombre', X1, Y1, A, H, 'tipo', 'enl')**

donde **X1** e **Y1** son las coordenadas dónde se situará la esquina superior izquierda de la imagen, **A** es el ancho con el que se visualizará la imagen, **H** su altura, tipo es el formato de la imagen original que puede ser: **JPG**, **JPEG**, **PNG** ó **GIF**. Por último, el parámetro **enl** permite -tal como ocurría con **CELL** ó con **WRITE** y con los mismos criterios allí utilizados- establecer un enlace externo o una referencia interna.

El parámetro *nombre* debe especificar la ruta, el nombre y la extensión de la imagen a incluir. La clase no soporta ni *entrelazados* en las imágenes **gif** ni transparencias (*canales alfa*) en las imágenes **png**.

Las dimensiones de la imagen pueden omitirse (incluyendo en su lugar una cadena vacía). En ese caso incluiría la imagen original con una resolución de 72 puntos por pulgada.

Si se especifica una sola de las

```
<?
#incluimos el fichero resultante de las modificaciones anteriores
include ("fpdf_con.gif.php");
define('FPDF_FONTPATH','c:/Apache/htdocs/cursoPHP/fontsPDF/');
/* establecemos las dimensiones del documento en mm.
creamos un nuevo objeto y A-4 apaisado.
Hemos modificado algunas funciones de la clase pero no le hemos cambiado
el nombre, por ello el constructor es el mismo */
$MiPDF=new FPDF('L','mm','A4');
# ajustamos la visualización para ver la página completa en pantalla
$MiPDF->SetDisplayMode('fullpage');
#añadimos una página
$MiPDF->AddPage();
# sin haber definido previamente ni un color ni un ancho de linea
# aparecerá en negro y con un espesor de 0.2 mm
$MiPDF->Line(5,5,287,5);
#ensayamos las diferentes opciones de rectangulos
#con los colores y espesores por defecto
$MiPDF->Rect(10,10,50,50);
$MiPDF->Rect(70,10,50,50,'D');
$MiPDF->Rect(140,10,50,50,'F');
$MiPDF->Rect(200,10,50,50,'DF');
# modificamos los colores y espesores de linea
$MiPDF->SetDrawColor(255,0,0);
$MiPDF->SetFillColor(0,0,255);
$MiPDF->SetLineWidth(3);
# dibujamos nuevos rectángulos con los nuevos valores
$MiPDF->Rect(10,65,50,50);
$MiPDF->Rect(70,65,50,50,'D');
$MiPDF->Rect(140,65,50,50,'F');
$MiPDF->Rect(200,65,50,50,'FD');
# insertamos una imagen (png) sin especificar dimensiones
$MiPDF->Image('./images/cruz.png',10,118,'','','png');
# otra imagen(jpg) en la que únicamente especificamos el alto
$MiPDF->Image('./images/cabina.jpg',90,118,'',90,'jpg');
# otra imagen(gif) en la que especificamos ancho y alto y provocamos
# una distorsión
$MiPDF->Image('./images/peligro.gif',215,118,80,50,'gif');
#añadimos una nueva página
$MiPDF->AddPage();
# asignamos un nombre a la imagen dinámica que vamos a generar
# e incluir en el documento PDF
$Imagen="ladinamica.jpg";
# ejecutamos la función que crea la nueva imagen
imagen1($Imagen);
# insertamos la nueva imagen y generamos la salida
$MiPDF->Image($Imagen, 45, 35 , 150, '', 'jpg','http://www.google.es');
$MiPDF->Output();
# ya podemos borrar la imagen dinámica que hemos creado
unlink($Imagen);
# esta es la función que crea la imagen dinámica
# le asignamos un nombre para que sea guardada temporalmente
# en el directorio actual
function imagen1($Imagen){
    Header("Content-type: image/jpeg");
    $im = imagecreate(200,200);
    $fondo=imagecolorallocate ($im, 0, 0, 200);
    $blanco=imagecolorallocate ($im, 255, 255, 255);
    Imagefill ($im, 0, 0, $fondo);
    Imagerectangle ($im, 10, 10, 190, 190, $blanco);
    Imagejpeg($im,$Imagen);
    ImageDestroy($im);
}
?>
```

ejemplo140.php

dimensiones la otra se calcula de forma automática y se mantienen las proporciones. Si se insertan valores de largo y ancho pueden generarse, a voluntad, efectos de distorsión.

---

[Anterior](#)

[Índice](#)

[Siguiente](#)



Ver índice

# Celdas múltiples, encabezados y pies de página



## Celdas con saltos de línea

La función **Cell**, estudiada en la página anterior, no interpreta los saltos de línea y, en el caso de que la cadena de texto sea más larga que el ancho de la celda, escribirá fuera de sus márgenes.

La clase FPDF dispone de la función:

**\$obj->MultiCell(a,h,text,b,aln,r)**

donde **a** es el ancho de la celda (si se indica cero ocupará hasta el margen derecho de la página), **h** es el alto de cada una de las celdas aunque, en la práctica, se comporta de forma idéntica al interlineado de un procesador de textos), **text** es la cadena de texto (o variable que lo contiene) a insertar.

**b** es un parámetro que puede valer: **0** (sin bordes); **1** (con bordes), y también: **L** (borde por el lateral izquierdo de la celda); **T** (borde por la parte superior), **R** (línea de borde en el margen derecho), **B** (línea de borde en la parte inferior) ó agrupaciones de estos últimos valores, en cualquier orden, tales como: **LR** ó **TB**, etcétera.

El parámetro **aln** indica la alineación horizontal que han de tener los textos y permite los valores: **L** (izquierda); **C** (centro); **R** (derecha) ó **J** (justificado).

Por último, el parámetro **r** (relleno) especifica si a las celdas se les aplicará (valor 1) un color de fondo o si se va a tratar (valor 0) de un fondo transparente.

El comportamiento de esta función tiene las siguientes particularidades:

- Inserta los saltos de línea contenidos en el fichero origen y los incluidos en la cadena de texto mediante la secuencia de escape **\n**.
- Inserta saltos de línea automáticos en el momento en que el texto alcanza el borde derecho de la celda.
- Si AutoPageBreak está activado inserta saltos de página automáticos en el momento en que el texto alcanza el margen inferior de la página.

## Encabezados, pies de página y saltos automáticos

```
<?
# incluye y define idénticos a los anteriores
include("fpdf.php");
define('FPDF_FONTPATH',$_SERVER['DOCUMENT_ROOT'].'/cursophp/fontspdf/');
# creamos una clase extendida de la clase FPDF
class GranPDF extends FPDF {
    # incluimos la función Header (debe ser definida con este nombre)
    # que se ejecuta cada vez que se añade una página, sea en forma
    # manual o en forma automática.
    # Nos permite insertar los encabezados de todas las páginas del
    # documento
    function Header(){
        #insertamos un salto de línea de 2 mm. a partir
        #del margen superior
        $this->Ln(2);
        # establecemos color y estilo de letra del encabezado
        $this->SetTextColor(0,0,0);
        $this->SetFont("Times","I", 11);
        # establecemos una celda cuyo ancho es cero
        # de esta forma se extenderá hasta el margen derecho
        # ocupando toda la caja de texto de la página.
        # le ponemos únicamente brode inferior, texto centrado
        # sin relleno y le adjudicamos como parámetro de posición
        # de la celda siguiente 1 de forma que comience en el margen
        # izquierdo de una línea nueva
        $this->Cell(0,8,"La Regenta",'B',1,'C',0);
        # colocamos un salto de líneas de 3 milímetros para separar el
        # encabezado de los textos de la página
        $this->Ln(3);
    }
    # incluimos la función Footer (debe ser definida con este nombre)
    # para insertar pies de página cada vez que cree una página nueva
    function Footer(){
        $this->Ln(2);
        $this->SetTextColor(0,0,0);
        $this->SetFont("Arial","I", 9);
        # en este caso incluimos el número de página con un borde superior
        # de la celda, texto centrado y tambien activando la celda siguiente
        # de modo que se produzca un saldo de línea
        $this->Cell(0,5,"Página ".$this->PageNo(),'T',1,'C',0);
    }
}
# acabada la inserción de la clase extendida continuamos con el código
# establecemos las dimensiones del documento
$dimensiones=array (140,200);
# creamos un nuevo objeto pero ¡cuidado! utilizaremos
# la clase extendida GranPDF
$MiPDF=new GranPDF('P','mm',$dimensiones);
# ajustamos al 100% la visualización
$MiPDF->SetDisplayMode('fullpage');
# añadimos la primera página del documento. La ejecución de esta función
# disparará la ejecución de las funciones Header() y Footer() de la
# clase extendida y, por tanto, incluirá en el documento los encabezados
# y pies de página allí establecidos
$MiPDF->Addpage();
# cambiamos el estilo de fuente a "normal"
$MiPDF->SetFont('Times','','12');
# leemos un fichero de texto y lo recogemos en una variables
    $f1=fopen('regenta.txt','r');
    $regental=fread($f1,filesize('regenta.txt'));
    fclose($f1);
# insertamos el fichero mediante Multicell
# el ancho 0 establece que la celda ocupará desde el margen
# izquierdo hasta el derecho. La interlinea será de 4mm.
# el texto que se incluirá (con salto de línea automático
# e inserción automática de nuevas páginas) será el recogido
```

## y pies de página

La clase FPDF contiene dos métodos (funciones) llamados: **Header()** y **Footer()**.

Si editamos el fichero fpdf.php podemos ver que ambas están vacías (no contienen ninguna instrucción) presentando una sintaxis como esta:

```
function Header(){  
// comentario  
}  
  
y  
function Footer(){  
// comentario  
}
```

Ambas son invocadas de forma automática cada vez que se ejecuta la función **AddPage()** y, eso permite crear otras funciones *a medida*, con igual nombre, e incluirlas en una **clase extendida**.

Recuerda que los métodos de la clase extendida prevalecen sobre los que pudieran existir en la clase padre con su mismo nombre. Por esta razón, las nuevas funciones de la clase extendida nos permitirían incluir encabezados y pies de página en cada una de las nuevas páginas añadidas mediante **AddPage()**.

El procedimiento sería este:

- Creamos una clase extendida de la clase original **FPDF**.

- Incluimos en esta nueva clase funciones con nombres **Header()** y **Footer()** (no podemos modificar el nombre ya que han de coincidir con los nombres de las funciones vacías de la clase padre) en las que indicamos los contenidos que han de insertarse (y su posición) mediante cada una de ellas.
- Al crear el nuevo objeto mediante:

```
$obj= new nombre_clase()
```

utilizaremos el nombre el de la **clase extendida** que hemos creado para incluir estas funciones.

De esta forma, cada vez que se ejecuta AddPage() (añadir una nueva página) se ejecutarán también Header() y Footer() que, al no ser ya funciones vacías, realizarán una inclusión de contenidos en cada una de las páginas del documento.

## Gestión de la inserción automática de páginas

La clase FPDF contiene esta función:

```
function AcceptPageBreak(){  
// comentario  
return $this->AutoPageBreak;  
}
```

que igual que ocurría con Header() y

```
# del fichero de texto. La celda no tendrá bordes  
# el texto estará justificado y no tendrá relleno de fondo.  
$MiPDF->Multicell(0,4,$regental,0,'J',0);  
# establecemos la visualización del documento  
$MiPDF->Output();?>
```

ejemplo141.php

El ejemplo siguiente es una adaptación del [ejemplo139](#) al uso de encabezados y pies de página.

[Ver código fuente](#)

[ejemplo142.php](#)

```
<?  
# incluimos la clase fpdf.php y la constante FPDF_FONTPATH  
include("fpdf.php");  
define('FPDF_FONTPATH',$_SERVER['DOCUMENT_ROOT'].'/cursophp/fontspdf/');  
# creamos la clase GranPDF extendida de fpdf.php  
class GranPDF extends FPDF {  
# definimos dos nuevas variables internas  
# el identificador de la columna actual  
# y el valor de la ordenada de inicio de las columnas  
    var $columna_actual=0;  
    var $ordenada_inicio=21;  
# la función Header se comporta de forma idéntica al ejemplo anterior  
    function Header(){  
        $this->SetTextColor(0,0,0);  
        $this->SetFont("Times","I", 11);  
        $this->Cell(0,8,"La Regenta",'B',1,'C',0);  
        $this->Ln(3);  
    }  
# Footer es similar al caso anterior con la única diferencia que  
    function Footer(){  
        $this->SetY(-20);  
        $this->Ln(2);  
        $this->SetTextColor(0,0,0);  
        $this->SetFont("Arial","I", 9);  
        $this->Cell(0,5,"Página ".$this->PageNo(),'T',1,'C',0);  
    }  
    function AcceptPageBreak(){  
        # al alcanzar el margen inferior se activa esta función  
        # que comprueba si el punto de inserción está en la columna  
        # primera (0) ó en la segunda  
        if($this->columna_actual==0){  
            # cambia el valor del número de columna a 1 (pasa a la segunda)  
            $this->columna_actual=1;  
            # cambia el punto de inserción a la coordenada de inicio  
            # que es la parte superior de la página  
            $this->SetY($this->ordenada_inicio);  
            # cambia el margen izquierdo de modo que el texto  
            # aparezca en la columna derecha (a partir de los 72.5 mm).  
            # más abajo está comentada la razon de estos valores  
            $this->SetLeftMargin(72.5);  
            # cambia el punto de insercion al nuevo margen izquierdo  
            $this->SetX(72.5);  
            # establecemos que la función devuelva false  
            # con ello no se produce un salto de página  
            # y el punto de inserción se translada a las  
            # coordenadas indicadas en las instrucciones anteriores  
            return false;  
        }else{  
            # si la columna es la segunda (no es la cero)  
            # cambia a la primera, asigna el valor cero  
            $this->columna_actual=0;  
            # cambia el punto de inserción a la coordenada de inicio  
            # que es la parte superior de la página  
            $this->SetY($this->ordenada_inicio);  
            # cambia el margen izquierdo de modo que el texto  
            # aparezca en la columna izquierda (a partir de los 10 mm).  
            $this->SetLeftMargin(10);  
            # cambia el punto de insercion al nuevo margen izquierdo  
            $this->SetX(10);  
            # establecemos que la función devuelva true  
            # con ello genera un salto de página  
            # y la insercion de texto continua en la pagina siguiente
```

Footer() es invocada en el momento en que encuentra un salto de página y dependiendo del valor que devuelva (true ó false) se ejecuta ó no la función AddPage().

Tal como está diseñada esta clase devolverá el valor que tenga asignado AutoPageBreak() (true ó false) pero, igual que ocurría con Header() y Footer() permite incluir otra función con el mismo nombre en una clase extendida. La condición de la nueva función es que, al igual que ocurre con esta, habrá de devolver un valor booleano (cierto ó falso).

Esta posibilidad añade nuevas e interesantes funcionalidades al uso de la clase FPDF. Una de las posibilidades podría ser la de crear documentos con varias columnas (estilo periodístico). La forma de proceder (una de las posibles) la tienes descrita en el ejemplo que hemos incluido a la derecha.

## Otras opciones

Además de las funciones que hemos descrito a lo largo de esta página, existen clases extendidas que incluyen nuevas, y en algunos casos muy interesantes, funciones que resuelven problemas específicos concretos a la hora de crear ficheros PDF. Algunas de ellas están disponibles desde el propio sitio FPDF de Olivier PLATHEY.

Una de las utilidades más interesantes -desde luego que ni mucho menos únicas- de este tipo de formatos podría ser la paginación de los resultados de las consultas en bases de datos.

Cuando nos encontramos en esa situación podremos utilizar un procedimiento casi idéntico al usado en el ejemplo sobre las tablas de alimentos.

No importaría el tamaño del fichero de texto (número de páginas que pudiera ocupar) ni el número de registros obtenidos como resultado de una consulta a una base de datos. Sería la propia clase FPDF quien se encargaría de crear las páginas necesarias de un documento con un formato preestablecido.

```
        return true;
    }
}

/* establecemos el tamaño de la página
que tendrá 140 mm. de ancho.
Si establecemos un margen por la izquierda de 10 mm.
un margen por la derecha de otros 10 mm.
y un espacio entre columnas de 5 mm. nos restan
140-25=115 mm. que repartidos entre dos columnas
les darían un ancho de 115/2=57,5 mm.
Las primera columna comenzará en 10 y acabará en
10 +57,5=67.5 mm. La segunda deberá empezar en
72.5 (añadiendo 5 mm. de espacio entre columnas */
$dimensiones=array (140,200);
# creamos el nuevo objeto partiendo de la clase ampliada
$MiPDF=new GranPDF('P','mm',$dimensiones);
# ajustamos al 100% la visualización
$MiPDF->SetDisplayMode('fullpage');
# insertamos la primera página del documento
$MiPDF->Addpage();
# cambiamos el estilo de fuente a "normal"
$MiPDF->SetFont('Times','','12');
# establecemos un color de fondo para las celdas de texto
# y el color de la tipografía
$MiPDF->SetFillColor(240,240,240);
$MiPDF->SetTextColor(0,0,0);
# leemos un fichero de texto y lo recogemos en una variables
$f1=fopen('regenta.txt','r');
$regental=fread($f1,filesize('regenta.txt'));
fclose($f1);
# insertamos el fichero mediante Multicell
# el ancho 57.5 estable el ancho de columna igual
# para la derecha que para la izquierda La interlinea será de 4mm.
# el texto que se incluirá (con salto de línea automático
# e inserción automática de nuevas páginas cuando se hayan completado
# las dos columnas) será el recogido
# del fichero de texto. La celda no tendrá bordes
# el texto estará justificado y SI tendrá relleno de fondo.
$MiPDF->Multicell(57.5,4,$regental,0,'J',1);
# establecemos la visualización del documento
$MiPDF->Output();?>
```

ejemplo143.php

## Ejercicio nº 37

Crea un documento PDF en formato A-5 con orientación vertical. Debe incluir un encabezado (a modo de logotipo) compuesto por un dibujo creado mediante la funciones PDF. Además, habrá de incluir un texto enmarcado y una fotografía centrada en el documento.

Anterior



Índice



Siguiente



## Programas necesarios

Para generar las nuevas fuentes es necesario disponer del programa **ttf2pt1.exe**. Si accedemos a [este enlace](#) podemos ver, en la zona de descargas, una opción donde dice *Complete package, except sources* desde el que podemos descargar el fichero **ttf2pt1-3.4.4.exe**.

Al ejecutar este instalador, con la opción por defecto, nos creará un directorio llamado **GnuWin32** (en Archivos de Programa) dentro del cual encontramos un directorio llamado **bin** en el que se encuentra el ejecutable **ttf2pt1**.

El resto de las opciones de esta distribución no son necesarias para nuestros fines.

## Creación del fichero .afm

Este fichero, cuya extensión es el acrónimo de **Adobe Font Metric** contiene las información sobre ancho, alto y kerning de cada carácter de una fuente.

El primer paso ha de ser la creación de ese fichero. El procedimiento para hacerlo es el que tienes descrito al margen.

### ¡Cuidado!

En el directorio **fuentes** tienes los ficheros **cayge.afm** y **caygr.afm** generados según el proceso descrito al margen.

## Definición de las fuentes

El directorio **fontsPDF** de estos materiales (recuerda que es el fruto de renombrar el directorio **font** extraído del **fichero de instalación** de la clase **FPDF**) contiene otro subdirectorio (**makefont**) que incluye un script llamado **makefont.php**. La definición de fuentes requiere la ejecución de una de las funciones que contiene ese script llamada **MakeFont()** y que utiliza los siguientes parámetros:

**MakeFont(f1, f2, c, p, tipo)**

donde **f1** es el nombre y path de la fuente TrueType a utilizar (fichero **amano.ttf** en el nombre del fichero **amano**)

## Proceso de generación de fuentes

Esta primera imagen es la visualización del directorio fuentes de estos materiales. Podrás observar que contiene fuentes TrueType y el ejecutable **ttf2pt1**.

Hemos incluido dos fuentes TrueType llamadas **bobcayge.ttf** y **bobcaygr.ttf**. Serán las que utilizaremos para añadir nuevas opciones de fuentes a nuestros PDF's.

	arial.ttf	267 KB	Archivo de fuentes ...	31/12/2002 11:20
	fuente2.ttf	109 KB	Archivo de fuentes ...	31/12/2002 11:21
	ttf2pt1.exe	162 KB	Aplicación	01/01/2004 11:06
	bobcayge.ttf	28 KB	Archivo de fuentes ...	12/03/2002 19:22
	bobcaygr.ttf	27 KB	Archivo de fuentes ...	12/03/2002 20:32

### ¡Cuidado!

No es necesario que realices la instalación de **ttf2pt1-3.4.4.exe** comentada al margen. En el subdirectorio **fuentes** de estos materiales ya dispones del fichero **ttf2pt1**.

En el caso de Linux Ubuntu puedes instalarlo desde el terminal escribiendo:

```
sudo apt-get install ttf2pt1
```

## Creación del fichero .afm

Desde la ventana de MS-DOS y una vez elegido el directorio que contiene el ejecutable **ttf2pt1** podemos crear el fichero **.afm** usando la siguiente sintaxis:

```
ttf2pt1 -anombre_de_la_fuente_TrueType.ttf nombre_del_nuevo_fichero
```

```
C:\> C:\WINDOWS\system32\cmd.exe
C:\> C:\ServidoresLocales\Apache\htdocs\cursophp\fuentes>ttf2pt1 -a amano.ttf grafi
Auto-detected front-end parser 'ttf'
<use ttf2pt1 -p? to get the full list of available front-ends>
Processing file amano.ttf
Some font name strings are in Unicode, may not show properly
Creating file grafi.tia
numglyphs = 219
Found Unicode Encoding
Guessed italic angle: -50.000000
FontName aaaaight-
No Kerning data
Finished - font files created
C:\> C:\ServidoresLocales\Apache\htdocs\cursophp\fuentes>
```

Nos generará dos ficheros con el nombre asignado al nuevo fichero y extensiones **.afm** y **.t1a** tal como puedes ver en la imagen.

	amano.ttf	94 KB	Archivo de fuentes T...	31/12/2006 17:50
	grafi.afm	13 KB	Archivo AFM	07/06/2009 8:27
	grafi.t1a	305 KB	Archivo T1A	07/06/2009 8:27

## Creación del fichero de definición de fuente

Este es un ejemplo de script que permitiría definir nuevas fuentes mediante la utilidad **makefont**.

creado para esta misma fuente mediante `ttf2pt1`. Estos dos parámetros son obligatorios.

Mediante el parámetro `c` (opcional) se especifica el tipo de codificación (el mapa de fuentes a utilizar). Si no se indica el valor por defecto sería `cp1252`.

El parámetro opcional `c` permite cambiar la codificación de algunos caracteres, y, mediante `tipo` puede indicarse el tipo de fuente original (por defecto se asigna TrueType).

La definición de nuevas fuentes requiere la ejecución de la función `MakeFont`. Para ello, una de las opciones, es utilizar un script como el que ves a la derecha. En una primera instrucción se incluye el fichero `makefont.php` y en la segunda se hace una llamada a la función únicamente con los parámetros obligatorios.

La ejecución de ese script daría como resultado la creación de dos ficheros con el mismo nombre. Uno de ellos con extensión `.z` y el otro con extensión `.php`. Ambos, para que puedan ser usados por la clase `FPDF`, deberán ser incluidos en el directorio `fonsPDF`.

## Diferentes tipografías

Lo habitual es que entre las fuentes TrueType de un tipo determinado existan varios ficheros. Uno para la fuente normal, otro para la cursiva, etcétera. Cuando se trabaja con este tipo de fuentes y se pretende usar estilos distintos lo aconsejable es definir una fuente para cada tipo y tratarlos, a la hora de escribir el código de los script como fuentes independientes.

De esta forma podríamos tener parejas de ficheros del estilo: `fx_normal.z` y `fx_normal.php`; `fx_cursiva.z` y `fx_cursiva.php`, etcétera. A la hora de utilizarlas se haría como si se tratara de fuentes de distintas familias y habría de evitarse incluir el estilo en `SetFont`.

La sintaxis de la asignación de fuentes habría de ser:

```
$obj->SetFont("nombre","", 9);
```

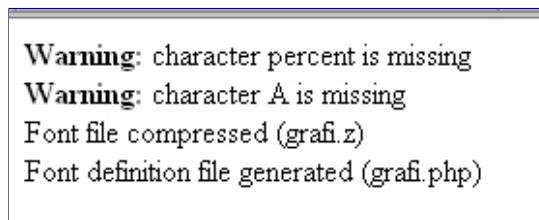
dejando el valor del segundo parámetro (estilo) como una cadena vacía. La única excepción a esta norma sería el caso del subrayado que no requiere otra cosa que asignar el valor U al parámetro estilo.

## Precauciones con las fuentes

El uso de fuentes externas puede producir efectos indeseados si, por alguna circunstancia, tal como ocurre en los ejemplos la fuente

```
<?
# incluimos el fichero makefont.php que está
# en la ruta que se especifica (si lo tuvieramos en otra
# ubicación bastaría cambiar el path del include
include($_SERVER['DOCUMENT_ROOT']."cursophp/fontspdf/makefont/makefont.php")
# ejecutamos la función MakeFont indicando como parametros
# la ruta y el fichero ttf a convertir (la fuente trueType)
# y como segundo parámetro la ruta y el nombre del fichero afm
# creado en el proceso anterior
MakeFont($_SERVER['DOCUMENT_ROOT']."/cursophp/fuentes/amano.ttf",
        $_SERVER['DOCUMENT_ROOT']."/cursophp/fuentes/grafi.afm")
?>
```

Al ejecutar el script anterior puede ocurrir que, dependiendo de la fuente ttf que estemos utilizando, aparezcan mensajes de advertencia como los que ves en esta imagen.



Significaría que algunos tipos (generalmente los asociados a los caracteres ASCII ampliados) no estarían disponibles y que, en consecuencia, o no se visualizarían o se visualizarían de forma incorrecta.

Una vez creados los ficheros como las nuevas fuentes (se crean en el mismo directorio en que hayamos ejecutado el script) es necesario incluirlos en el directorio `fontspdf` de modo que estén disponibles a la hora de crear los documentos PDF.

cayge.php	4 KB
cayge.z	13 KB
grafi.php	4 KB
grafi.z	23 KB

## Ejemplo de utilización

La utilización de nuevas fuentes requiere que

```
objeto->AddFont(nombre_a_utilizar, estilo, fichero_fuente.php)
```

donde el `nombre_a_utilizar` puede ser cualquiera, el parámetro `estilo` puede ser la cadena vacía y el tercer parámetro es el nombre y extensión del fichero `.php` que contiene la definición de la fuente y que fué creado por medio de `makefont`.

Si se omite el tercer parámetro será asignado, de forma automática, la cadena formada por la **unión** las cadenas `nombre_a_utilizar` y `estilo`, agregándole la extensión `.php`.

```
<?
#incluimos el fichero con la clase y definimos la variable FPDF_FONTPATH
# con el mismo criterio comentado en el ejemplo anterior
include("fpdf.php");
define('FPDF_FONTPATH',$_SERVER['DOCUMENT_ROOT'].'/cursophp/fontspdf/');
/* establecemos las dimensiones del documento
   creamos un nuevo objeto y añadimos una página*/
$dimensiones=array (140,200);
$MiPDF=new FPDF('P','mm',$dimensiones);
# ajustamos al 100% la visualización
$MiPDF->SetDisplayMode('real');
#insertamos una página en blanco
$MiPDF->Addpage();
# establecemos el color de la letra
$MiPDF->SetTextColor(255,0,0);
# añadimos la nueva fuente. Podemos ponerle un nombre cualquiera
# Al no incluir estilo considerará el estilo normal
$MiPDF->AddFont('fuentenueva1','','cayge.php');
```

carácteres (eñes, letras con tilde, símbolos). Conviene tenerlo en cuenta para evitar que la elección de una tipografía pueda afectar a los contenidos del documento.

### ¡Cuidado!

En el caso de Ubuntu el proceso es idéntico.

Bastará con abrir el terminal, situarse en el directorio **fuentes**.

Desde allí se ejecuta el comando **ttf2pt1** de forma idéntica a la descrita para MS-DOS.

```
# a indicar SetFont habremos de utilizar el mismo nombre de fuente
# establecido mediante AddFont
$MiPDF->SetFont('fuentenueva1','','24');
$MiPDF->Multicell(0,10,"Prueba de la fuente que hemos
                           llamado fuente nueva1",1,'L',0);
$MiPDF->Ln(10);
# incluimos la fuente caygr.php con nombre fuentenueva2
$MiPDF->AddFont('fuentenueva2','','grafi.php');
$MiPDF->SetFont('fuentenueva2','','24');
$MiPDF->Multicell(0,10,"Prueba de la fuente que hemos
                           llamado fuente nueva2",1,'R',0);
$MiPDF->Output();
?>
```

[ejemplo144.php](#)

En estos dos ejemplos pueden verse los resultados -con algunos problemas gráficos- de la sustitución de las fuentes en algunos de los ejemplos de páginas anteriores.

[ejemplo145.php](#)

[ejemplo146.php](#)

[Anterior](#)

[Índice](#)

[Siguiente](#)



Ver índice

## Campos y conexiones



### Conexión con el servidor de bases de datos

Antes de empezar a trabajar con bases de datos es **imprescindible** que ambos servidores –Apache y MySQL– estén **activos**.

Como paso inmediato hemos de **interconectar los dos servidores** de forma que sea posible transferir información de uno a otro.

Para ello es necesario utilizar siempre una función PHP con la siguiente sintaxis:

**\$c=mysql\_connect(h, u, p)**

donde **\$c** es la variable que recoge el **identificador** del enlace, **h** es la dirección del servidor de bases de datos, ("localhost"), **u** es el nombre de uno de los usuarios registrados en la tabla user ("pepe" o "root") y **p** la contraseña (en nuestro caso "pepa" ó "").

Estos tres valores –cadenas de texto– deben ir escritos entre comillas.

Para **cerrar** la conexión, tenemos que insertar:

**\$c=mysql\_close (\$c)**

donde **\$c** es el nombre de la variable en la que se recogió el **identificador** del enlace en el momento de la apertura.

Aquí tienes el código de un *script* que realiza la apertura de una conexión y después la cierra.

Y aquí puedes comprobar el funcionamiento del *script* anterior.

Si realizáramos una segunda conexión (con los mismos argumentos) sin haber cerrado la anterior **no se efectuará un nuevo enlace** sino que nos devolverá el ya abierto.

### Sintaxis alternativa

Otra forma de efectuar la conexión es utilizar los valores registrados en el fichero **mysql.inc.php** –lo hemos creado en la página anterior– y eso requiere que insertemos un **include("c:...")**, indicando la ruta completa de **seguridad** y el nombre del fichero en el que hemos guardado las claves y que era **mysql.inc.php**.

### Automatizar la conexión

Con nuestros conocimientos sobre PHP ya estamos en condiciones de hacer más cómoda la conexión. Creemos una función que realice de forma automática la conexión con MySQL y guardémosla en nuestro fichero **mysql.inc.php**

```
<?
# estas son las variables anteriores
$mysql_server="localhost";
$mysql_login="pepe";
$mysql_pass="pepa";

# creemos una nueva variable $c sin asignarle ningún valor
# para que pueda recoger el identificador de conexión
# una vez que se haya establecido esta

$c;
# escribamos la función que hace la conexión
# como pretendemos que el valor del identificador
# sea usado fuera de la función, para recuperar su valor
# pasaremos ese valor por referencia anteponiendo & al
# nombre de la variable
function conecta1(&$c){
# para usar las variables anteriores en la función
# hemos de definirlas como globales
    global $mysql_server, $mysql_login, $mysql_pass;
    if($c=mysql_connect($mysql_server,$mysql_login,$mysql_pass)){
        print "<br>Conexión establecida<br>";
    }else{
        print "<br>No ha podido realizarse la conexión<br>";
        # el exit lo incluimos para que deje de ejecutarse
        # el script si no se establece la conexión
        exit();
    }
}

# esta función asignará a $c el valor del identificador

# repetimos la misma función con otro nombre
# ahora quitaremos el mensaje de conexión establecida
# consideraremos que si no hay mensaje se ha establecido
# así quedará limpia nuestra página

function conecta2(&$c){
global $mysql_server, $mysql_login, $mysql_pass;
if($c=mysql_connect($mysql_server,$mysql_login,$mysql_pass)){
}else{
    print "<br>No ha podido realizarse la conexión<br>";
    exit();
}
?>
```

Si sustituyes el contenido de tu **mysql.inc.php** por el que tienes aquí arriba –puedes eliminar las líneas de comentario al hacerlo– estaremos en disposición de ejecutar scripts como este.

En este ejemplo utilizaremos la primera función:

[Ver código fuente](#)

[Ejecutar ejemplo](#)

y ahora haremos uso de la segunda

[Ver código fuente](#)

[Ejecutar ejemplo](#)

En este supuesto como valores de los parámetros **h**, **u** y **p** pondremos los nombres de las variables:

`$mysql_server`  
`$mysql_login` y  
`$mysql_pass`  
sin encerrarlas entre comillas.

Aquí tienes el código de un *script* que realiza la apertura de una conexión y después la cierra.

Desde este enlace puedes comprobar su funcionamiento.

## La instrucción OR DIE

Es esta una buena ocasión para hablar de una instrucción PHP que no hemos mencionado hasta el momento.

Es una opción alternativa a `exit()` que, como acabamos de ver en un ejemplo, interrumpe la ejecución de un *script* en el momento de ser ejecutada.

Cuando se produce un error en la ejecución de un *script* –no poder establecer conexión con MySQL, por ejemplo– no tiene sentido seguir ejecutándolo. Lo razonable será *interrumpir* el proceso y advertir del error.

Si añadimos a la instrucción `$c=mysql_connect('h','u','p')` (sin paréntesis, ni comas, ni punto y coma, sólo separado por un espacio):

**or die ('mensaje')**

y ponemos el *punto y coma* de fin de instrucción después de cerrar este último paréntesis, en el caso de que se produzca un error se interrumpirá la ejecución del *script* y aparecerá en la ventana del navegador el texto incluido en *mensaje*.

Este es el código fuente de un *script* que produce un error –la contraseña es incorrecta– y que utiliza esta nueva sintaxis. Pero si lo ejecutas verás que aparece un mensaje de error generado por PHP.

Este tipo de mensajes pueden deshabilitarse haciendo una modificación en `php.ini`. Pero hay una técnica mucho más fácil. Bastará con insertar delante de la función *una arroba* (@) para evitar que aparezcan. En este otro *script* lo hemos incorporado y puedes comprobarlo [aqui](#).

## Lista de bases de datos existentes

Antes de **crear** y/o **borrar** una *base de datos* puede ser conveniente y útil comprobar si ya existe.

# Tipos de campos en MySQL

MySQL tiene habilitados diversos tipos de **campos** que en una primera aproximación podrían clasificarse en tres grupos:

- Campos numéricos**
- Campos de fecha**
- Campos de cadenas de caracteres**

## Campos numéricos

MySQL soporta los tipos numéricos **exactos**(INTEGER, NUMERIC, DECIMAL, y SMALLINT) y los tipos numéricos **aproximados** (FLOAT, DOUBLE precision y REAL).

Los campos que contienen **números enteros** admiten el parámetro **UNSIGNED**, que implica que **no admite signos**, por lo que solo aceptaría **enteros positivos**.

Todos los campos **numéricos** admiten el parámetro **ZEROFILL** cuya función es completar el campo **con ceros a la izquierda** hasta su longitud máxima.

### Tipos de campos numéricos enteros

Estos son los distintos tipos de campos numéricos enteros que admite MySQL. Los parámetros señalados entre corchetes son opcionales.

#### **TINYINT [(M)] [UNSIGNED] [ZEROFILL]**

Número entero *muy pequeño*. Con la opción **UNSIGNED** puede tomar valores entre **0** y **255**. En caso contrario, puede estar comprendido entre **-128** y **127**.

El parámetro **ZEROFILL** sólo tiene sentido junto con la opción **UNSIGNED** ya que *no es habitual* llenar los números negativos con ceros a la izquierda del signo.

El valor por defecto de parámetro **M** (número de cifras) es **4** si no está activada la opción **UNSIGNED**. Si esta opción estuviera activada el valor por defecto sería **M=3**. Para valores de **M > valor por defecto** reajusta el tamaño al **valor por defecto**.

Si se asigna a M un valor **menor que cuatro** limita el número de caracteres al tamaño especificado considerando el signo sólo en los números negativos.

Por ejemplo, si **M=3** admitiría **148**, pero si intentamos insertar **-148** recortaría por la izquierda y solo insertaría **-14**.

Si intentamos insertar un valor **fuerza de rango** registraría el **valor dentro del rango más próximo a él**.  
P. ej.: Si tratamos de insertar el valor **437** escribiría **127** ó **255**, este último en el caso de tener la opción **UNSIGNED**. Si pretendiéramos insertar **-837** con la opción **UNSIGNED** escribiría **0** y sin ella pondría **-128**.

El tamaño de un campo TINYINT es de **1 byte**.

#### **SMALLINT [(M)] [UNSIGNED] [ZEROFILL]**

Número entero *pequeño*. Con la opción **UNSIGNED** puede tomar valores entre **0** y **65 535**. En caso contrario, puede estar comprendido entre **-32 768** y **32 767**.

Son válidos los comentarios hechos para **TINYINT**, excepto los relativos a los valores **por defecto** de **M** que en este caso serían **6** ó **5**. Su tamaño es de **2 bytes**.

#### **MEDIUMINT [(M)] [UNSIGNED] [ZEROFILL]**

Número entero *mediano*. Con la opción **UNSIGNED** puede tomar valores entre **0** y **16 777 215**. En caso contrario, puede estar comprendido entre **-8 388 608** y **8 388 607**.

También son válidos los comentarios hechos para **TINYINT**, excepto los relativos al valor **por defecto** de **M** que en este caso serían **8**. Su tamaño es de **3 bytes**.

#### **INT [(M)] [UNSIGNED] [ZEROFILL]**

Número entero. Con la opción **UNSIGNED** puede tomar valores entre **0** y **4 294 967 295**. En caso contrario, puede estar comprendido entre **-2 147 483 648** y **2 147 483 647**.

Son válidos todos los comentarios de los casos anteriores. Su tamaño es de **4 bytes**.

#### **INTEGER [(M)] [UNSIGNED] [ZEROFILL]**

PHP dispone de herramientas para conocer el **número** de bases de datos existentes en el servidor, así como sus nombres. Todas ellas requieren que se haya establecido una conexión con el servidor.

**\$p=mysql\_list\_dbs(\$c)**

La variable **\$p** es un **nuevo identificador imprescindible** y previo a la determinación del número y los nombres de las bases de datos existentes en el enlace abierto (identificado por **\$c**).

**\$n=mysql\_num\_rows(\$p)**

Esta función devuelve el *número de bases de datos existentes* en el servidor.

Utiliza como parámetro (**\$p**) el resultado obtenido mediante la función anterior.

Ese número puede recogerse en una variable (en este caso **\$n**).

**mysql\_db\_name(\$p, i)**

Esta nueva función devuelve el nombre de una de las bases de datos, identificada por un número **i** que debe pertenecer al intervalo [0,\$n].

Fíjate que **i** tiene que ser **i<\$n** porque si, por ejemplo, **\$n=5** los cinco valores posibles de **i** serían: 0,1,2,3 y 4.

Una **lista completa** de todas las bases de datos existentes en el servidor podría hacerse mediante el siguiente proceso:

- Abrir la conexión.
- Invocar a **mysql\_list\_dbs**.
- Contar el número de bases de datos con **mysql\_num\_rows**
- Insertar un bucle:  
**for (\$i=0;\$i<\$num,\$i++)**
- Presentar la *lista de nombres* mediante un bucle que lea los diferentes valores de **\$i** en:  
**mysql\_db\_name(\$p,\$i)**

Aquí tienes el código fuente de un ejemplo completo y desde aquí puedes ejecutarlo

## Crear una base de datos

La creación de una **base de datos** también requiere una conexión previa y utiliza la siguiente sintaxis:

**mysql\_query  
("CREATE DATABASE nom")**

donde **nom** es el nombre de la **nueva base de datos**.

Esta función devuelve TRUE si la base de datos es creada, y FALSE si no es posible hacerlo.

Es un sinónimo de **INT**

### **BIGINT [(M)] [UNSIGNED] [ZEROFILL]**

Número entero grande. Con la opción **UNSIGNED** puede tomar valores entre 0 y 18 446 744 073 709 551 615. En caso contrario, puede estar comprendido entre -9 223 372 036 854 775 808 y 21 474 839 223 372 036 854 775 807 647, pero al usarlo desde PHP estará sujeto a las limitaciones máximas de los valores numéricos de este.

Son válidos todos los comentarios de los casos anteriores. Su tamaño es de 8 bytes.

## Números de coma flotante

Por la estructura binaria de los microprocesadores y habida cuenta de que algunos números **no enteros** -sin ir más lejos, el 0.1- requerirían *infinitos caracteres binarios* para su representación exacta, se hace necesario introducir un **redondeo** en su tratamiento informático y como consecuencia de ello asumir que se generan **errores de medida**.

Esta circunstancia obligó al tratamiento de los números decimales mediante el llamado **Standar de Aritmética de Punto Flotante**, un algoritmo definido por la **IEEE** (Institute of Electrical and Electronics Engineers) que unificó los procesos de representación de números en ordenadores con lo que son uniformemente controlables los errores introducidos.

El **Standar de Aritmética de Punto Flotante** estableció **dos niveles de precisión**:

Precisión **Simple**, en la que **todo número debe ser almacenado en 32 bits** (4 bytes)

**Doble** precisión, en la que los **números se almacenan en 64 bits** (8 bytes).

MySQL admite los siguientes tipos de números de coma flotante:

### **FLOAT(x) [ZEROFILL]**

Número de **coma flotante**. Ignora la opción **UNSIGNED**, pero sí acepta **ZEROFILL**, por lo que debe prestarse atención a estas opciones ya que no sería demasiado habitual una presentación como esta: **000-3.47**

El valor de **x** especifica la **precisión**. Si **x<=24** será de **precisión simple**, cuando **24 <x <=53** lo convertirá automáticamente a **doble precisión**.

Cuando no se especifica el valor de **x** considera el campo como de **precisión simple**. Su tamaño es de 4 bytes si **x<=24** y de 8 bytes cuando **24 <x <=53**

### **FLOAT [(M,D)] [ZEROFILL]**

Número de **coma flotante de precisión simple**. Son válidos los comentarios relativos a las opciones **UNSIGNED** y **ZEROFILL** del caso anterior.

Toma valores en los intervalos siguientes:  
**-3.402823466E+38 a -1.175494351E-38**

**0 y 1.175494351E-38 a 3.402823466E+38.**

**M** es la anchura máxima de visualización y **D** es el número de decimales. Si **M > 24** se convierte automáticamente a doble precisión

**FLOAT** sin argumentos representa un número de **coma flotante y precisión simple**.

### **DOUBLE [(M,D)] [ZEROFILL]**

Número de **coma flotante de doble precisión**. Siguen siendo válidos los comentarios relativos a las opciones **UNSIGNED** y **ZEROFILL** del caso anterior.

Toma valores en los intervalos siguientes:  
**-1.7976931348623157E+308 a -2.2250738585072014E-308**

**0 y 2.2250738585072014E-308 a 1.7976931348623157E+308**

**M** es la anchura máxima de visualización y **D** es el número de decimales.

**DOUBLE** sin argumentos representa un número de **coma flotante y precisión doble**.

### **REAL [(M,D)] [ZEROFILL]**

Es sinónimo de **DOUBLE**.

### **DECIMAL [(M[,D])] [ZEROFILL]**

Es un número de coma flotante y doble precisión que se almacena como un campo de tipo CHAR.

Si intentamos crear una base de datos con un nombre ya existente la función nos devolverá FALSE.

Aquí tienes el código de un ejemplo de creación de una base de datos. Si lo ejecutas dos veces podrás comprobar que en la segunda oportunidad te aparece el mensaje diciendo que no ha sido posible crearla.

## Borrar una base de datos

Para borrar una base de datos se requiere el uso de la siguiente función PHP:

```
mysql_query  
("DROP DATABASE nom")
```

donde nom es el nombre de la base de datos y debiendo ponerse toda la cadena del paréntesis entre comillas.

Esta función devuelve TRUE cuando se ejecuta con éxito, y FALSE en el caso contrario.

Este es el código de un script que puede borrar la base creada anteriormente.

Igual que ocurría al tratar de crearla, si intentamos borrar una base de datos inexistente la función mysql\_drop\_db nos devolverá FALSE.

## Depurando los procesos de creación y borrado de bases de datos

Cuando intentamos crear una base de datos ya existente o borrar una inexistente las funciones mysql\_query nos devuelven FALSE pero esa respuesta no nos dice la causa por la que no ha sido posible la ejecución de la instrucción.

Sería mucho más interesante comprobar la existencia o inexistencia de una base de datos antes de ejecutar esas instrucciones y que después de la comprobación se nos presentara un mensaje informativo.

MySQL dispone de una sentencia para este fin, pero –aunque la vamos ver más adelante– olvidémosnos de su existencia e intentemos crear nuestro propio script de comprobación.

Combinando las instrucciones anteriores no resulta difícil hacerlo. Aquí tienes un ejemplo de código para efectuar esa comprobación al crear una base de datos y este otro código es para el caso de borrado.

Experimenta con estos scripts, sustítuyelos por otros propios en los

El valor es guardado como una cadena donde cada carácter representa una cifra. La coma y el signo menos de los números negativos no son tenidos en cuenta en el valor de M -anchoa máxima de visualización- aunque si se reserva -automáticamente- espacio para ellos en campo.

Si D vale 0 no tendrá parte decimal. Los números toman valores en el mismo intervalo especificado para DOUBLE.

Los valores por defecto de M y D son respectivamente 10 y 0.

Ocupan M+2 bytes si D > 0; M+1 bytes si D = 0 ó D+2 bytes si M < D

### NUMERIC(M,D) [ZEROFILL]

Se comporta de forma idéntica a DECIMAL.

## Campos de fecha

MySQL dispone de campos específicos para el almacenamiento de fechas. Son los siguientes:

### DATE

Recoge una fecha dentro del intervalo 01-01-1000 a 31-12-9999. MySQL guarda los valores DATE con formato AAAA-MM-DD (año-mes-día). Su tamaño es de 3 bytes.

### DATETIME

Recoge una combinación de fecha y hora dentro del intervalo 00:00:00 del día 01-01-1000 y las 23:59:59 del día 31-12-9999. MySQL guarda los valores DATETIME con formato AAAA-MM-DD HH:MM:SS (año-mes-día hora:minutos:segundos). Su tamaño es de 8 bytes.

### TIME

Recoge una hora dentro del intervalo -838:59:59 a 838:59:59. MySQL guarda los valores TIME con formato HH:MM:SS (horas:minutos:segundos). Su tamaño es de 3 bytes.

### YEAR 0 YEAR(2) o YEAR(4)

Recoge un año en formato de cuatro cifras (YEAR o YEAR(4)) o en formato de dos cifras (YEAR(2)) dentro del intervalo 1901 a 2155 en el caso de cuatro cifras o de 1970 a 2069 si se trata de dos cifras. Su tamaño es de 1 byte.

### TIMESTAMP [(M)]

Recoge un tiempo UNIX. El intervalo válido va desde 01-01-1970 00:00:00 a cualquier fecha del año 2037.

El parámetro M puede tomar los valores: 14 (valor por defecto), 12, 8, o 6 que se corresponden con los formatos AAAAMMDDHHMMSS, AAMMDDHHMMSS, AAAAMMDD, o AAMMDD.

Si se le asigna la opción NUL guardará la hora actual. Cuando se asigna 8 o 14 como parámetros es considerado como un número y para las demás opciones como una cadena.

Independientemente del valor del parámetro, un campo TIMESTAMP siempre ocupa 4 bytes.

## Campos tipo cadena de caracteres

### CHAR (M) [B/NARY]

Es una cadena de tamaño fijo que se completa a la derecha por espacios si es necesario.

El parámetro M puede valer de 1 a 255 caracteres.

Los espacios finales son suprimidos cuando la cadena es insertada en el registro.

Los valores de tipo CHAR son elegidos y comparados sin tener en cuenta Mayúsculas / Minúsculas y utilizan el juego de caracteres por defecto.

Se puede utilizar el operador BINARY para hacer la cadena sensible a Mayúsculas / Minúsculas.

Se puede utilizar un campo tipo CHAR(0) con el atributo NULL para almacenar una valor booleano. En este caso ocupará un solo byte y podrá tener únicamente dos valores: NUL ó "".

Su tamaño es de M bytes siendo 1 <= M <= 255 .

### VARCHAR(M) [B/NARY]

que utilices las funciones que hemos incluido –a la derecha tienes el código fuente– dentro del fichero **mysql.inc.php** y comprueba, listando las bases de datos existentes, que sólo queden: **mysql** y **test**.

## Tipos de campos MySQL

Conocidos los procesos de creación, listado y borrado de bases de datos ya estamos en disposición en empezar a tratar lo relativo a las **tablas**.

Es muy necesario conocer los **diferentes tipos de campos** que pueden contener las **tablas** de MySQL. Aquí a la derecha los tienes.

Conocer las posibilidades de cada uno será fundamental a la hora de diseñar una tabla. En ese momento tendremos que decidir qué campos son necesarios, cuál es tipo requerido, cuáles han de ser sus dimensiones y también cuáles de ellos requerirán ser tratados como índices.

Tipos de campo bien elegidos y un tamaño adecuado a las necesidades reales de nuestro proyecto son las mejores garantías para optimizar el tamaño de la tabla y para hacerla realmente eficaz.

El **grado de eficiencia** de una base de datos suele ser **directamente** proporcional al **tiempo invertido** en el análisis de la estructura de sus tablas.

## CREACIÓN DE LAS BASES DE DATOS NECESARIAS PARA EL CURSO

Una vez hayas hecho todas las pruebas necesarias con las funciones anteriores, llega el momento de utilizarlas para crear las bases de datos que vamos a utilizar en el Curso.

Pulsa en este enlace para que cree automáticamente la base de datos que va a contener los sucesivos ejemplos que hemos incluido en estos materiales.

**Crear base de datos EJEMPLOS**

Una vez hecho esto, escribe tu propio script y crea una segunda base de datos como el nombre **prácticas**. Esta será la que habrás de utilizar en ejercicios que tendrás que ir haciendo en el resto del curso.

Es una **cadena de caracteres de longitud variable**. Su tamaño máximo –especificado en el parámetro **M**– puede estar comprendido entre 1 y 255 caracteres. Con la opción **BINARY** es capaz de discriminar entre Mayúsculas / minúsculas.

### TINYBLOB o TINYTEXT

**TINYBLOB** y **TINYTEXT** son **cadenas de caracteres de longitud variable** con un tamaño máximo de 255 ( $2^8 - 1$ ) caracteres.

La diferencia entre ambas es que **TINYBLOB** **discrimina** entre Mayúsculas / minúsculas, mientras que **TINYTEXT** no lo hace.

Ninguno de los campos **BLOB** y **TEXT** admite **valores por DEFECTO**

Las versiones de **MySQL** anteriores a 3.23.2 permiten utilizar estos campos para indexar.

Si se intenta guardar en un campo de este tipo una cadena de **mayor longitud que la especificada** solamente se guardarán los **M** primeros caracteres de la cadena.

### BLOB o TEXT

**BLOB** y **TEXT** son **cadenas de caracteres de longitud variable** con un tamaño máximo de 65535 ( $2^{16} - 1$ ) caracteres.

La diferencia entre ambas es que **BLOB** **si discrimina** entre Mayúsculas / minúsculas, mientras que **TEXT** no lo hace.

Ninguno de los campos: **BLOB** y **TEXT** admite **valores por DEFECTO**

### MEDIUMBLOB o MEDIUMTEXT

**MEDIUMBLOB** y **MEDIUMTEXT** son **cadenas de caracteres de longitud variable** con una longitud máxima de 16.777.215 ( $2^{24} - 1$ ) caracteres.

Son válidas las especificaciones hechas en el apartado anterior.

El tamaño máximo de los campos de este tipo está sujeto a limitaciones externas tales como la memoria disponible y el **tamaño del buffer de comunicación servidor/cliente**.

### LONGBLOB o LONGTEXT

Su única diferencia con la anterior es el tamaño máximo de la cadena, que en este caso es 4.294.967.295 ( $2^{32} - 1$ ) caracteres.

### ENUM('valor1','valor2',...)

Es una **cadena de caracteres** que contiene **uno solo** de los valores de la lista (valor1, valor2, etc. etc.).

A la hora de insertar un nuevo registro en una tabla, el valor a especificar para un campo de este tipo ha de ser **una cadena que contenga uno de los valores** especificados en la tabla. Si se tratara de insertar un valor distinto de ellos insertaría una cadena vacía.

### SET('valor1','valor2','valor3',...)

Es una cadena de caracteres formados por la **unión** de ninguno, uno o varios de los valores de una lista. El máximo de elementos es 64.

Los valores que **deben escribirse** en los **campos** de este tipo han de ser **numéricos**, expresados en forma **binaria** o en forma decimal.

En el supuesto de que contuviera **tres valores** los posibles valores a **insertar** en un campo de este tipo a la hora de añadir un registro serían los siguientes.

Incluir valores	Código valor	Cadena binaria	Equiv. decimal				
val1	val2	val3	val1	val2	val3		
Si	Sí	Sí	1	1	1	111	7
Si	Sí	No	1	1	0	011	3
Si	No	Sí	1	0	1	101	5
No	Sí	Sí	0	1	1	110	6
No	No	Sí	0	0	1	100	4
No	Sí	No	0	1	0	010	2
Si	No	No	1	0	0	001	1
No	No	No	0	0	0	000	0

### **¡Cuidado!**

Antes de continuar, es conveniente comprobar desde Windows que el subdirectorio **mysql\data** contiene las bases de datos **ejemplos** y **practicas**

Anterior



Índice



Siguiente



## La aplicación MySql

Una de las opciones más útiles de PHP es la posibilidad de gestionar **bases de datos** en ordenadores remotos.

Existen multitud de programas de **servidores de bases de datos** y PHP dispone de funciones para el manejo de muchos de ellos tales como:

FilePro  
dBase  
DBM  
Microsoft SQL  
PostgreSQL  
mSQL  
InterBase  
MySQL

Nuestra opción por **MySQL** obedece a que es uno de los gestores de bases de datos SQL más populares, es muy eficiente y, además, es gratuito (*Open Source*).

El sintagma **SQL** es el acrónimo de **Structured Query Language**, es decir, *Lenguaje estructurado de consultas*.

## Bases de datos y tablas

Como habrás podido ver en la columna de la derecha, una **base de datos** no es otra cosa que un directorio que contiene **tablas**.

Las **tablas** son las que contienen los datos y en consecuencia son el elemento verdaderamente importante, dado que no es demasiado relevante que esos datos estén en uno u otro directorio, lo verdaderamente importante es que estén y que sean accesibles y manejables.

Si es este tu primer contacto con el mundo de las bases de datos, quizás sea importante conocer su argot ya que en adelante tendremos que referirnos a tablas, campos, registros e índices y quizás no esté de más explicar un poco esas ideas.

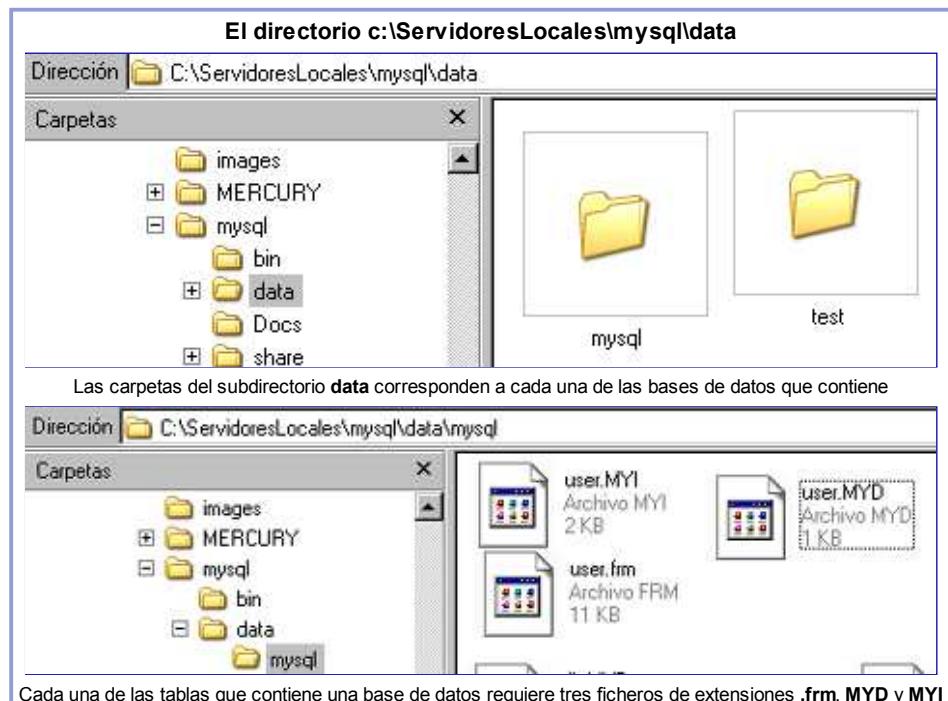
Pensemos en la base de datos –directorio– como un armario archivador de nuestro Centro e imaginemos que dentro de él hay una serie de cajones –tablas– en los que se puede guardar una buena cantidad de documentos con formato idéntico –registros– tales como: matrículas, fichas de alumnos, etcétera.

Cada uno de estos tipos de

## Organización de la información

Hemos instalado MySQL en el directorio **c:\mysql** y durante el proceso de instalación se creó un subdirectorio llamado **data** que es el destinado a albergar todas las bases de datos que vayamos creando.

Como puedes ver en la tabla siguiente, durante la instalación se crearon dos bases de datos llamadas **mysql** y **test**.



Los ficheros con extensión **.frm** contienen la estructura de cada tabla, los **MYD** los datos y los que tienen extensión **.MYI** contienen los índices de esa tabla.

En la instalación también se crearon veintitrés tablas en la base de datos mysql. Una de ellas (puedes verla en la imagen) es **user**.

La tabla **user** –la más importante para nuestros fines– contiene información sobre los usuarios, desde qué máquinas pueden acceder a nuestro servidor MySQL, sus claves y contraseñas y los permisos de acceso de cada uno de ellos.

Las restantes contienen información sobre las máquinas que pueden acceder al sistema, las bases de datos a las que tendrá acceso cada usuario y las limitaciones o restricciones que eventualmente pudieran establecerse.

### ¡Cuidado!

El uso de los tres ficheros anteriores, es condición de las tablas tipo **MyISAM** que es el tipo, por defecto, que utiliza MySQL 5.1. Otros tipos, que veremos en temas posteriores, tales como **InnoDB** almacenan la información con una estructura diferente.

## La tabla user

Activemos nuestro servidor MySQL de la forma que se indica aquí si tu sistema operativo es **Linux Ubuntu**, o, aquí si se trata de **Windows**.

documento tendría las mismas **casillas** –**campos**–, con la única diferencia de que los **datos** contenidos en esos **campos** –de igual forma, dimensión y tamaño en todos los **registros**– serían los que diferenciarían un documento de otro.

En una tabla MySQL el fichero con extensión **.frm** contiene precisamente el **documento original** de los registros y, de la misma forma que ocurre con los impresos originales, contiene la forma, dimensión y tamaño de cada una de las **casillas** (**campos**) de cada **formulario** (**registro**).

El fichero **.MYD** contiene los **datos**, es decir todo lo que hay escrito en cada una de las casillas de cada uno de los impresos de ese cajón (**tabla**) concreto.

No hemos aludido aún a los ficheros con extensión **.MYI** y son un elemento importante. Contienen los **índices**, que –como su propio nombre indica– cumplen una función idéntica a los índices de los libros. A través de ellos resulta mucho más rápido encontrar una determinada información y al igual que ocurre con los libros –índice general, onomástico, etcétera– pueden ser varios y con contenidos distintos.

## ¿Cómo empezar con las bases de datos?

De igual modo que ocurriría en el símil anterior, al instalar MySQL ya creamos el armario archivador –el directorio **data**– y también dos cajoncitos –las bases de datos **mysql** y **test**– pero por pura cuestión de orden vamos a necesitar algún otro armario para poder guardar nuestras cosas.

Lo razonable será empezar creando nuestros armarios (**bases de datos**) para que posteriormente podamos ir diseñando los documentos –los campos y la estructura– de cada uno de los tipos de impreso (tablas) que vayamos a manejar.

## Nuestras bases de datos

Utilizaremos dos bases de datos distintas. Una de ellas –a la que llamaremos **ejemplos**– contendrá todos los ejemplos que vayamos desarrollando y la otra –**prácticas**– será donde habrás de insertar las tablas que irás elaborando en los ejercicios y en la Actividad Final del Curso.

Los ejemplos aún no están creados y serás tú quien lo vaya haciendo medida que avancemos en esta parte del curso. Lo haremos de forma muy similar a la que hemos visto en páginas anteriores.

Una vez **activos el servidor MySQL y Apache** ya podemos utilizar **phpMyAdmin**. Lo habíamos instalado en un subdirectorio de `htdocs` llamado **phpmyadmin**. Así que accedamos a través de la dirección: <http://localhost/phpmyadmin/> y se nos abrirá una página como la que sigue:

Elegiremos la base de datos **mysql**. Al seleccionar **mysql** en el menú de la izquierda y pulsar sobre **mysql** (en negro en la parte superior de la lista de tablas) aparecerá la lista que está a la derecha de esta imagen.

Allí vemos la tabla **user** y un enlace activo que dice *Examinar*. Si pulsamos sobre él podremos ver un contenido similar a este que vemos aquí debajo.

	Host	User	Password
<input type="checkbox"/>	localhost	root	
<input type="checkbox"/>	127.0.0.1	root	
<input type="checkbox"/>	localhost		
<input type="checkbox"/>	localhost	pepe	*C36E402D61D6E5B30487546DBED41C181C4267E1

↑ Marcar todos/as / Desmarcar todos Para los elementos que están marcados:

Como ves, hay cuatro usuarios y dos de ellos como nombre **root** y ninguna contraseña han sido creados automáticamente durante la instalación. El cuarto de ellos –el usuario **pepe**– es el que hemos creado durante el proceso de instalación

Esta configuración es insegura ya que con los nombres de usuario por defecto –**root**– y sin contraseña cualquiera podría acceder y manipular las bases de datos.

Más adelante podremos borrar esos usuarios pero, por el momento, dejémoslos así y añadamos un nuevo usuario.

Si pulsamos la opción editar (el icono en forma de lápiz que hay a la derecha de la casilla de verificación del usuario sin nombre se nos abrirá una página como esta:

Campo	Tipo	Función	Nulo
Host	char(60)	<input type="text"/>	localhost
User	char(16)	<input type="text"/>	jose
Password	char(41)	<input type="password"/> PASSWORD	josefa
Select_priv	enum	--	<input checked="" type="radio"/> N <input type="radio"/> Y
Insert_priv	enum	--	<input checked="" type="radio"/> N <input type="radio"/> Y
Update_priv	enum	--	<input checked="" type="radio"/> N <input type="radio"/> Y
Delete_priv	enum	--	<input checked="" type="radio"/> N <input type="radio"/> Y
Create_priv	enum	--	<input checked="" type="radio"/> N <input type="radio"/> Y
Drop_priv	enum	--	<input checked="" type="radio"/> N <input type="radio"/> Y
Reload_priv	enum	--	<input checked="" type="radio"/> N <input type="radio"/> Y
Shutdown_priv	enum	--	<input checked="" type="radio"/> N <input checked="" type="radio"/> Y
Process_priv	enum	--	<input checked="" type="radio"/> N <input type="radio"/> Y
File_priv	enum	--	<input checked="" type="radio"/> N <input type="radio"/> Y

Según vayamos desarrollando los contenidos, insertaremos ejemplos de código fuente –de forma similar a la de temas anteriores– y cuando sea necesario crear bases de datos o tablas pondremos un **recuadro rojo** sobre el que **deberás ir pulsando para auto generar los ejemplos**.

En tales casos, no olvides hacerlo pues esa será la forma en la que progresivamente vayamos creando elementos necesarios para ser utilizados en los ejemplos posteriores.

## Servidores activos

De ahora en adelante, para seguir el Curso vamos a necesitar tener siempre activos tanto el servidor Apache como MySQL. De no hacerlo así, nos aparecerá un mensaje de error diciendo:

Can't connect to MySQL server on 'localhost' (10061)

Si ello te ocurriera, comprueba el *semáforo*. Probablemente esa será la causa del error.

Escribamos **localhost** en el campo *Host*, **jose** en el campo *User*, **josefa** en el campo *Password* y marquemos **todas las opciones** –una lista bastante larga por cierto– como YES (Y) y **–muy importante**– seleccionemos la función **PASSWORD** para el campo del mismo nombre.

Una vez realizado el proceso anterior deberá quedarnos como aparece en la imagen. Pulsaremos en el botón *Continúe* que hay al final de la página y habremos dado de alta al usuario **jose** con todos los privilegios para gestionar las bases de datos.

Si regresamos de nuevo a *Examinar* veremos que ya ha sido incluido el nuevo usuario y que el campo contraseña aparece encriptado como consecuencia de haber aplicado la función **PASSWORD** para **garantizar** la privacidad del usuario. MySQL requiere esta encriptación.

Respecto a los YES, la explicación es sencilla. Esas opciones permiten habilitar permisos para determinadas operaciones dentro de las bases de datos y lo único que hemos hecho ha sido conceder todas la facultades de gestión al usuario **pepe**.

## Creación de un fichero INCLUDE

En los scripts PHP con los que manejemos las bases de datos vamos a necesitar insertar continuamente: nombre del servidor, nombre de usuario y contraseña.

Tanto la comodidad como la privacidad que hemos mencionado en páginas anteriores aconsejan *guardar los datos de usuario en lugar seguro*. Así que vamos a crear un fichero –llámémosle **mysql.inc.php**– idéntico al que tenemos aquí debajo,(podemos *copiar y pegar*) y guardémoslo en nuestro directorio de seguridad que –como recordarás– estaba en:

C:/ServidoresLocales/Apache/seguridad.

```
<?
$mysql_server="localhost";
$mysql_login="pepe";
$mysql_pass="pepa";
?>
```

Anterior

Índice

Siguiente



Ver índice

## Creación de tablas



### Creación de tablas

Las tablas son *elementos* de las base de datos. Por esa razón *nos resultará imposible crear una tabla* sin tener *creada y seleccionada una base de datos*.

Es por eso que para la creación de una tabla se necesitan los siguientes requisitos:

- Tener abierta una conexión con el servidor MySQL.
- Tener seleccionada una base de datos.

#### Conexión con el servidor

La hemos comentado en la página anterior, recuerda que requería la función:

`$c=mysql_connect(h, u, p)`

Esa conexión ha de ser establecida antes de cualquier otra intervención relacionada con accesos a bases de datos y tablas.

#### Selección de la base de datos

Dado que podemos manejar bases de datos distintas –en este curso usaremos *ejemplos* y *prácticas*– es preciso *dicir* a MySQL con qué base queremos trabajar.

`mysql_select_db("n", $c);`

donde *n* es el nombre de la base de datos (puede ser una cadena entrecomillada o el nombre de una variable previa que contenga ese nombre). En este último caso, como es habitual, el nombre de la variable no llevaría comillas.

El segundo parámetro *\$c* es el *identificador de conexión*. Es decir, la variable creada al establecer la conexión con MySQL.

Este valor debe insertarse siempre. La razón es que MySQL permite mantener abiertas, de forma simultánea, varias conexiones (podríamos manejar más de un servidor de bases de datos) y en esas condiciones sería necesaria una *conexión distinta* para cada servidor.

#### Creación de una tabla

En todas las transacciones PHP-MySQL habremos de utilizar instrucciones de ambas lenguajes

### Crear una tabla MySQL

La creación de tablas MySQL requiere una de estas dos sentencias:

**CREATE TABLE IF NOT EXISTS tabla (campo1, campo2,... )**

**CREATE TABLE tabla (campo1, campo2,... )**

La única diferencia entre ambas opciones es que la segunda daría un error si tratáramos de crear una tabla preexistente (deberíamos recurrir al procedimiento que hemos visto cuando creábamos bases de datos) mientras que la primera no da ese mensaje de error.

Aunque no lo hemos indicado, CREATE DATABASE también permite esta sintaxis alternativa.

### Definición de campos en una tabla MySQL

Cada uno de los campos que vayamos a crear en una tabla requiere una definición que debe contener lo siguiente:

#### – nombre del campo

Es una palabra cualquiera –distinta para campo de la tabla y que normalmente suele elegirse aludiendo al contenido. Por ejemplo, *fec\_nac*, *nom\_perro*, etcétera.

No va entre comillas nunca y MySQL **diferencia** mayúsculas/minúsculas.

Para utilizar como nombres de campo **palabras reservadas** –por ejemplo, **create**– del lenguaje MySQL debemos escribirla entre ` `. Observa que no son comillas sino acentos graves. Lo más aconsejable es evitar esta situación.

#### – tipo y dimensiones

Los tipos de campos –los hemos visto en la página anterior– tienen que ajustarse a uno de los soportados por MySQL. Se escriben a continuación del nombre **sin otra separación que un espacio** y requieren la sintaxis –estricta– correspondiente a cada tipo.

**– flags del campo ( son opcionales)** Puede utilizarse cualquiera de los permitidos para cada tipo de campo –puedes verlos encerrados entre corchetes– al lado de cada tipo de campo.

### Ejemplo de creación de una tabla

```

<?
/* nos conectamos con el servidor
recogiendo en $c el identificador de conexión */
$c=mysql_connect ("localhost","pepe","pepa") or die ("Imposible conectar")
# seleccionamos una base de datos existente
# de lo contrario nos daría un error
# pondremos como nombre ejemplos nuestra base de datos
# creada en la página anterior y usaremos $c
# importante no olvidarlo
mysql_select_db ("ejemplos", $c);
/* ahora ya estamos en condiciones de crear la tabla
podríamos escribir ya la instrucción mysql_query y meter
detrás la sentencia MySQL pero, por razones de comodidad
crearemos antes una variable que recoja toda la sentencia
y será luego cuando la ejecutemos.
Definiremos una variable llamada $crear e iremos añadiendo cosas */
# la primera parte de la instrucción es esta (espacio final incluido
$crear="CREATE TABLE IF NOT EXISTS ";
# añadiremos el nombre de la tabla que será ejemplo1
# fijate en el punto (concatenador de cadenas) que permite
# ir añadiendo a la cadena anterior
$crear .="ejemplo1 ";

```

La forma habitual –hay algunas excepciones– en la que PHP se comunica con MySQL es la función:

```
mysql_query("sent", $c);
```

donde la cadena **sent** contiene las instrucciones propias de MySQL –las comentamos al margen– y **\$c** sigue siendo la variable que contiene el identificador de conexión.

La sintaxis de las sentencias MySQL que crean tablas la tenemos en la columna de la derecha.

## DIMENSIONES

Cuando se establezca una **dimensión** como parámetro opcional de un campo deben tenerse en cuenta algunos detalles.

Si en un campo numérico introdujéramos valores que exceden los límites su valor no se registraría en la tabla sino el valor del *límite más próximo* correspondiente a ese tipo de campo.

Supongamos un campo tipo TINYINT que permite valores comprendidos entre -128 y 127. Si asignáramos en uno de sus registros un valor igual a **234** se escribiría en la tabla **127** (el límite superior) y si ponemos **-834** escribiría el límite inferior, es decir, -128.

En caso de cadenas, si el valor introducido sobrepasara la longitud máxima permitida, la cadena sería *recortada* y únicamente se registraría el número máximo de caracteres permitidos.

## Consideraciones sobre IF NOT EXISTS

Esta variante de CREATE –aplicable tanto en tablas como en bases de datos– tiene la ventaja de no dar mensajes de error en caso de intentar crear una tabla –o base– ya existente, pero puede darnos algún sobresalto porque **no advierte** que la tabla no ha sido creada y puede darnos la sensación de que puede haber reescrito una tabla anterior.

Si pulsas reiteradamente en el enlace *Crear tabla ejemplo1* comprobarás que *no aparece ningún mensaje de error*.

**Crear la TABLA EJEMPLO1**

```
#ahora pongamos el paréntesis (con un espacio delante)
#aunque el espacio también podría detrás de ejemplo1
$crear .= "(" ;
# insertemos el primer campo y llamemoslo num1
# hagamoslo de tipo TINYINT sin otras especificamos
# sabiendo que solo permitira valores numéricos
# comprendidos entre -128 y 127
$crear .="num1 TINYINT , ";
# LOS CAMPOS SE SEPARAN CON COMAS por eso
# la hemos incluido al final de la instrucción anterior

# ahora num2 del mismo tipo con dimensión 3 y el flag UNSIGNED
# Y ZEROFILL que: cambiará los límites de valores
# al intervalo 0 - 255, y rellenará con ceros por la izquierda
# en el caso de que el número de cifras significativas
# sea menor de 3.
# Fijate que los flags van separado únicamente por espacios
$crear .="num2 TINYINT (3) UNSIGNED ZEROFILL, ";
# en num3 identico al anterior añadiremos un valor por defecto
# de manera que cuando se añadan registros a la tabla
# se escriba automaticamente ese valor 13 en el caso
# de que no le asignemos ninguno a ese campo
# por ser numérico 13 no va entre comillas
$crear .="num3 TINYINT (7) UNSIGNED ZEROFILL DEFAULT 13, ";
# ahora un número decimal num4 tipo REAL con 8 digitos en total
# de los cuales tres serán decimales y también rellenaremos con ceros
# Pondremos como valor por defecto 3.14
$crear .="num4 REAL (8,3) ZEROFILL DEFAULT 3.14, ";
# añadamos una fecha
$crear .="fecha DATE, ";
/* una cadena con un limite de 32 carácter con BINARY
   para que diferencie Pepe de PEPE */
$crear .="cadena VARCHAR(32) BINARY, ";
/* un ultimo campo –opcion– del tipo ENUM que solo admite
   como valores SI, NO, QUIZA
   fijate en las comillas y en el parentesis
   ¡¡cuidado....!! aqui no ponemos coma al final
   es el último campo que vamos a insertar y no necesita
   ser separado. Si la pones dará un ERROR */
$crear .="opcion ENUM('Si','No','Quiza') ";
# solo nos falta añadir el paréntesis contenido toda la instrucción
$crear .="";
/* tenemos completa la sentencia MySQL
   solo falta ejecutarla mediante mysql_query
   ya que la conexión está abierta
   y la base de datos ya está seleccionada */

/* pongamos un condicional de comprobación */
if(mysql_query($crear,$c)){
    print "Se ha creado la base de datos<br>";
    print "La sentencia MySQL podríamos haberla escrito asi:<br>";
    print "mysql_query(\"".\"".$crear."\", $c);";
} else{
    print "Se ha producido un error al crear la tabla";
}
?>
```

## Ejercicio nº 38

Como práctica de creación de tablas, deberás crear –en tu base de datos **practicas**– una tabla a la que llamaremos **tabla1** que recoja –al menos– en diferentes campos, los siguientes datos de tus alumnos: DNI (con letra incluida), nombre, apellidos (en campos diferentes), fecha de nacimiento, y –en un campo tipo ENUM– su condición de repetidor o no.

Anterior



Índice



Siguiente





Ver índice

## Ver y modificar estructuras



### Visualizar la estructura de una tabla

Es posible visualizar la estructura de una tabla de dos maneras: utilizando nuevas **funciones** de PHP, o aplicando nuevas **sentencias MySQL** con las funciones ya conocidas.

### Lectura de resultados de sentencias MySQL

La sentencia SHOW FIELDS – como prácticamente ocurre con todas las sentencias MySQL– no devuelve los resultados en formato *legible*.

Los resultados devueltos por estas sentencias requieren ser convertidos a un formato que sea interpretable por PHP.

Esa *traducción* se realiza de la siguiente forma:

El resultado devuelto por MySQL a través de una llamada **mysql\_query()** es recogido en una variable, de la forma siguiente:

**\$r=mysql\_query(sent, \$c)**

El resultado recogido en la variable **\$r**, está *estructurado en líneas* y la función:

**\$t =mysql\_fetch\_row (\$r)**

recoge en una variable (**\$t**) el contenido de la *primera línea* y coloca su *puntero interno* al *comienzo de la línea siguiente*. Por esta razón la *lectura completa* del contenido de la variable **\$r** requiere llamadas sucesivas a **mysql\_fetch\_row** hasta que haya sido leída la *última línea* del resultado de la llamada a MySQL.

La variable **\$t** tiene estructura de *array escalar* siendo cero el primero de sus índices.

Cuando el puntero interno de **mysql\_fetch\_row()** alcance el final de la *última línea* del resultado devolverá FALSE.

Por esa razón, la visualización de los resultados de una sentencia MySQL suele requerir dos bucles. Este es el esquema de la lectura:

```
$r=mysql_query(inst,$c);
while($r=mysql_fetch_row($r)){
    foreach ($r as $valor){
        print $valor;
    }
}
```

### Ver la estructura de una tabla utilizando MySQL

La sentencia MySQL que permiten visualizar la estructura de una tabla es la siguiente:

**SHOW FIELDS from nombre de la tabla**

```
<?
# asignamos a una variable el nombre de la base de datos
$base="ejemplos";
# esta otra recoge el nombre de la tabla
$tabla="ejemplo1";
# establecemos la conexión con el servidor
$c=mysql_connect ("localhost","pepe","pepa");
# seleccionamos la base de datos
mysql_select_db ($base, $c);

#ejecutamos mysql_query llamando a la sentencia
# SHOW FIELDS

$resultado=mysql_query( "SHOW FIELDS from $tabla",$c);

# determinamos el número campos de la tabla

$numero=mysql_num_rows($resultado);

# Presentamos ese valor numérico

print "La tabla tiene $numero campos<br>";

# ejecutamos los bucles que comentamos al margen
while($v=mysql_fetch_row ($resultado)){
foreach($v as $valor) {
    echo $valor,"<br>";
}
}

#####
#      REPETIMOS LA CONSULTA ANTERIOR USANDO AHORA LA      #
#          función mysql_fetch_array                         #
#####

#tenemos que VOLVER a EJECUTAR LA SENTENCIA MySQL
# porque el puntero está AL FINAL de la ultima linea
# de los resultados

$resultado=mysql_query( "SHOW FIELDS from $tabla",$c);

print("<BR> Los resultados con mysql_fetch_array<br>");

while($v=mysql_fetch_array($resultado)){
foreach($v as $clave=>$valor) {
    print ("El indice es: ".$clave." y el valor es: ".$valor."<br>");
}
}

#####
# la tercera posibilidad comentada
#####

$resultado=mysql_query( "SHOW FIELDS from $tabla",$c);
# intentaremos explicar este doble bucle con calma
/* En los procesos anteriores a cada paso del bucle
foreach leímos un array, lo imprimimos y sustituimos
ese array por uno nuevo

Ahora trataremos de recoger todos esos resultados en array
```

```
    }  
}
```

o también

```
while($r=mysql_fetch_row($r){  
    $g[]=$t;  
}
```

con lo que estaríamos creando un array bidimensional con el contenido de los resultados de cada línea.

En este caso el *primer índice* del array **\$g** seguiría las normas de creación de arrays y se iría autoincrementando en cada ciclo del bucle *while*.

El ejemplo tiene desarrollados ambos métodos.

Existe una función alternativa que mejora las prestaciones de la anterior. Se trata de:

```
$t=mysql_fetch_array($r)
```

es idéntica en cuanto a su funcionamiento y, además, añade una nueva posibilidad ya que los arrays que devuelve pueden ser leídos como escalares y también como asociativos. En este último caso incorporan como índice el nombre del campo de la tabla del que se han extraído cada resultado.

Respecto a la forma en la que sea asignan los índices a los array obtenidos mediante consultas a tablas, puedes verla en este enlace.

## Otras funciones informativas

**mysql\_num\_fields (\$res)**

Está función -en la que **\$res** es el *identificador de resultado* - devuelve el **número de campos** de la tabla.

**mysql\_num\_rows (\$res)**

Devuelve el **número de registros** que contiene la tabla. Si la tabla no contiene datos devolverá CERO.

**mysql\_field\_table(\$res, 0)**

Devuelve el **nombre de la tabla**. Observa que se pasa con índice **0** ya que esta información parece ser la primera que aparece en la tabla.

**mysql\_field\_type(\$rs, \$i)**

Nos devuelve el **tipo de campo** correspondiente a la posición en la tabla señalada por el índice **\$i**. Dado que la información de *primer campo* está en el índice **0**, el último valor válido de **\$i** será igual al **número de campos menos uno**.

**mysql\_field\_flags(\$res, \$i)**

Nos devuelve los **flags del campo**

para ello usamos un array bidimensional que renueva el primer indice en cada ciclo del bucle while de ahí que pongamos el \$contador para asignarle el primer indice Los segundos indices serán los valores de los indices del array \$v que recogemos de la petición MySQL pero mysql\_fetch\_array genera dos indices para cada uno de los valores, uno numérico y otro asociativo así que filtramos y si el indice es numérico guardamos en el array llamado numérico y si no lo es guardamos el llamado asociativo Tendremos separados en dos array ambos resultados ¿Complicado... ? Es cuestión de analizar con calma \*/

```
# ponemos el contador a cero para asegurar  
# no hay variables anteriores con el mismo nombre  
# y valores distintos  
  
$contador=0;  
  
while($v=mysql_fetch_array($resultado)) {  
    foreach ($v as $indice1=>$valor1) {  
        if(is_int($indice1)){  
            $numerica[$contador][$indice1]=$valor1;  
        }else{  
            $asociativa[$contador][$indice1]=$valor1;  
        }  
        $contador++;  
    }  
/* vamos a leer los array resultantes  
empecemos por el numérico que al tener  
dos indices requiere dos foreach anidados  
el valor del primero será un array  
que extraemos en el segundo */  
  
foreach($numerica as $i=>$valor){  
    foreach ($valor as $j=>$contenido){  
        print ("numérico[".$i."][".$j."]='".$contenido."  
    }  
}  
  
foreach($asociativa as $i=>$valor){  
    foreach ($valor as $j=>$contenido){  
        print ("asociativo[".$i."][".$j."]='".$contenido."  
    }  
}  
  
# liberamos memoria borrando de ella el resultado  
mysql_free_result ($resultado);  
  
# cerramos la conexión con el servidor  
mysql_close($c);  
?>
```

[Ver código fuente](#)

## Borrar una tabla

Las sentencias MySQL que permite borrar una tabla son las siguientes:

**DROP TABLE IF EXISTS from nombre de la tabla**

**DROP TABLE from nombre de la tabla**

la diferencia entre ambas radica en que usando la primera no se generaría ningún error en el caso de que tratáramos de borrar una tabla inexistente.

Aquí tienes el *código fuente* de un ejemplo:

[Ver código fuente](#)

## Borrar uno de los campos de una tabla

correspondientes a la posición en la tabla señalada por el índice `$i`. Se comporta igual que la anterior en lo relativo a los índices.

#### **mysql\_field\_len(\$res, \$i)**

Nos devuelve **la longitud del campo** correspondiente a la posición en la tabla señalada por el índice `$i`. Igual que las anteriores en lo relativo a los índices.

#### **mysql\_field\_name(\$rs, \$i)**

Nos devuelve **el nombre del campo** correspondiente a la posición en la tabla señalada por el índice `$i`. En lo relativo a los índices su comportamiento es idéntico a las anteriores.

### **Liberando memoria**

Si queremos *liberar* la parte de la memoria que contiene un identificador de resultado bastará con que insertemos la siguiente instrucción:

#### **mysql\_free\_result(\$res)**

Este proceso debe ser **posterior** a la visualización de los datos.

La sentencia MySQL que permite borrar uno de los campos de una tabla es la siguiente:

#### **ALTER TABLE nombre de la tabla DROP nombre del campo**

Es posible modificar la estructura de una tabla -en este caso **borrar un campo**- siguiendo un procedimiento similar a los anteriores.

La única diferencia estriba en utilizar la sentencia MySQL adecuada.

Resulta obvio que el campo **debe existir** para que pueda ser borrado y si no existiera, es obvio también que se produciría un error.

Aquí tienes el **código fuente** de un *script* que borra uno de los campos de una tabla.

[Ver código fuente](#)

### **Añadir un nuevo campo a una tabla**

Las sentencia MySQL que permite añadir un nuevo campo a una tabla es la siguiente:

#### **ALTER TABLE nombre de la tabla ADD nombre del campo tipo [flags]**

El procedimiento es similar a los casos anteriores utilizando esta nueva sentencia MySQL.

La sintaxis es similar a la de la creación de tablas: el **nombre del campo** debe ir seguido del **tipo** sin otra separación que el espacio

Aquí tienes el **código fuente** de un *script* que añade uno de los campos de una tabla.

[Ver código fuente](#)

---

Anterior

Índice

Siguiente



Ver índice

# Añadir registros e índices



## Manejando índices

Es muy frecuente la utilización de índices en las bases de datos. Pero... ¿qué son los índices? ¿para qué sirven?.

Si nos planteamos cuál es la utilidad del **índice** de un libro la respuesta sería:

Facilitar la búsqueda de datos  
Agilizar esa búsqueda  
La utilidad de los índices en las bases de datos es la misma.

## Tipos de índices

Si seguimos con el ejemplo del libro veremos que caben las posibilidades de que tenga:  
Un solo índice  
Varios índices  
Algunos libros sólo contienen un **índice de contenidos**. Sin embargo, es frecuente que también dispongan de otros, tales como: **índices analíticos**, **índices onomásticos**, etcétera.

Cuando existe **un solo índice** es obvio que puede decirse de él que es **único** y cuando existen varios podemos decir que el **índice de contenidos** es el **índice principal** y los demás son **índices** sin más o **índices auxiliares**.

Coincidirás con nosotros en que en el **Índice de contenidos** de un libro de texto sólo existe **una referencia** al **Tema XIII** en la que puede decir: **Tema XIII página 314**.

También coincidirás en que si en ese índice, además de lo anterior, dijera: **Tema XIII página 714** nos encontraríamos en una **situación confusa** que nos obligaría a preguntarnos: **¿dónde está el Tema XIII? ¿En la página 314? ¿En la 714? ¿En ambas?**.

Es por eso que las **tablas** de las bases de datos **no admiten nunca valores duplicados** ni en los **índices únicos** ni tampoco en los **índices principales**.

Los **índices auxiliares** tienen un comportamiento distinto. El índice onomástico de un libro puede hacer referencia a varias páginas y puede tener **duplicados**.

Por ejemplo: en un manual de Word puede existir un índice onomástico en el que se asocie la palabra

## Añadir registros a una tabla

Las sentencias MySQL que permiten añadir registros a una tabla son las siguientes:

**INSERT tabla (campo1,campo2,..) VALUES (valor1,valor2,..)**

Vamos a desarrollar un ejemplo completo de creación de una tabla e inserción de registros utilizando diversos métodos. Para ello seguiremos el siguiente proceso:

### 1.- Creación de la tabla

Empezaremos **creando una tabla** a la que llamaremos **demo4** y que contendrá los siguientes campos:

**Contador**, que será de tipo **TINYINT(8)** (número entero con un máximo de 8 dígitos) que contenga solo **valores positivos**, que se rellene automáticamente con **ceros por la izquierda** y que **se autoincremente** cada vez que añadimos un registro.

**DNI**, destinado a recoger valores de números de DNI y que debe poder contener *un máximo de ocho caracteres*.

Lo definiremos de tipo **CHAR** porque sus valores, pese a ser numéricos, *nunca van a requerir ningún tratamiento aritmético*.

**Nombre**, **Apellido1** y **Apellido2** serán tres campos tipo **VHAR** de tamaños máximos respectivos de **20, 15 y 15** caracteres.

Su finalidad la evidencian los *nombres de campo*. Les asignaremos el **flag NOT NULL** aunque no sea *totalmente correcto* dado que existen países en los que se utiliza un solo apellido.

**Nacimiento** será un campo tipo **DATE** al que asignaremos como **valor por defecto** el de **1970-12-21**. Recuerda que MySQL trata las fechas en ese orden (año-mes-día) y que admite como separadores tanto - como / por lo que son válidas entradas con cualquiera de estos formatos: **1970-12-21** ó **1970/12/21**, aunque esta segunda es convertida automáticamente al primer formato por MySQL.

**Hora** será un campo tipo **TIME** al que asignaremos como **valor por defecto** el de **00:00:00**. Recuerda que MySQL trata las horas en ese orden (hh:mm:ss) y que sólo admite como separador :.

Este campo está destinado a recoger *la hora de nacimiento* que aunque no tiene demasiado sentido es una *buenas excusa* para introducir este tipo de campo.

**Sexo** será un campo tipo **ENUM** que tendrá **dos opciones**: **M** (masculino) y **F**(femenino) y al que asignaremos **M** como **valor por defecto**.

**Fumador** será un campo tipo **CHAR(0)** que por su estructura -cadena de longitud cero- tendrá **dos únicas opciones** de valor: **NULL** ó "", que, como veremos, son **valores distintos** para MySQL.

**Idiomas** será un campo tipo **SET** definido para los valores: **Castellano, Francés, Inglés, Alemán, Búlgaro y Chino** y que podrá contener, como ocurre con los campos de este tipo, **ninguno, uno o varios** de los valores de la lista.

**Índice primario** será tratado como tal el campo **DNI** ya que se trata de un valor **único** para cada persona y que, como tal, **no puede tener duplicados**

**Índices auxiliares**. Para ejemplificar su tratamiento consideraremos el campo **Contador** como **índice secundario**.

El **código fuente** del fichero que genera esta tabla puedes verlo aquí debajo

[Ver código fuente](#)

**Crear la TABLA  
DEMO4**

### 2.1.- Añadir un registro

Cuando se añade un registro en una tabla los valores pueden añadirse en la propia sentencia MySQL o ser recogidos de los valores de variables PHP previamente definidas.

En este ejemplo tienes el **código fuente** del primero de los casos, en el que se añade a la tabla anterior un registro cuyos valores son:

DNI	Nombre	Apellido1	Apellido2	Nacimiento	Sexo	Hora	Fumador	Idiomas
1234	Lupicinio	Servidor	Servido	1954-11-23	M	16:24:52	NULL	3

**macros** con las páginas 37, 234 y 832 siempre que en esas páginas existan contenidos que aluden a la palabra *macro*.

Es por eso que las **tablas** de las **bases de datos** también **admiten duplicados** cuando se trata de **índices auxiliares**.

## Sintaxis de la definición de índices MySQL

Tanto al *crear una tabla* como al *modificarla* se pueden *añadir índices* de la misma forma que también se pueden *insertar campos*.

La sintaxis (dentro de la sentencia MySQL que crea o modifica una tabla) es la siguiente:

### PRIMARY KEY(campo)

donde **campo** es el *nombre del campo* que se establece como **índice principal** de la tabla.

El *nombre del campo* no va entrecerrillado y PRIMARY KEY (*campo*) se añade dentro de la sentencia CREATE como si se tratara de un campo más, delimitado por comas, salvo que estuviera al final de la sentencia CREATE, en cuyo caso se omitiría la coma final.

En el código fuente de la creación de la tabla que tienes a la derecha puedes ver un ejemplo práctico de la sintaxis.

Solo puede definirse **un índice primario por tabla** y el campo utilizado ha de ser un campo **no nulo**.

### UNIQUE nombre (campo)

Similar a PRIMARY KEY, en cuanto a que no admite valores duplicados, pero con dos diferencias importantes.

UNIQUE *permite la creación de más de un índice de este tipo por tabla* y además *no requiere* que los campos sobre los que se define sean *no nulos*.

### INDEX nombre (campo)

Con esta sintaxis se crea un *índice secundario* que debe tener un *nombre* (la posibilidad de que existan varios obliga a identificarlos de esta forma) y -como en los casos anteriores- el campo que va a ser utilizado como índice.

### INDEX nombre (campo(n))

Es un caso particular del anterior utilizable sólo en el caso de que el campo índice sea tipo CHAR y VARCHAR que permite indexar por los *n primeros caracteres* de esas cadenas.

Presta atención a los siguientes aspectos:

En la sentencia **no se alude** al campo **Contador**. La razón es que se trata un campo AUTOINCREMENTAL y en ese tipo de campos los valores de los registros **se escriben automáticamente** cada vez que se añade uno nuevo.

El registro **no se añadiría** si el valor de DNI **coincidiera con otra ya existente** en la tabla. Recuerda que habíamos definido ese campo como **índice primario**.

Si no hubiéramos incluido **el aviso de error** no tendríamos ninguna referencia sobre el **éxito de la inserción** y no detectaríamos el **problema de duplicidad**. Sencillamente ocurriría que **el registro no se añadiría** pero no nos enteraríamos de tal circunstancia.

Si en los valores de **nombre y apellidos** hubiéramos insertado textos **más largos** del tamaño establecido para ellos al crear la tabla, las cadenas **se recortarían** y sólo se añadirían -de izquierda a derecha- los **n primeros caracteres** de la cadena.

Si en la fecha de nacimiento hubiéramos introducido una **cadena vacía** nos habría puesto el **valor por defecto**, pero **si hubiéramos introducido un valor no válido** nos habría escrito **0000-00-00**.

Serían valores **no válidos** en este caso:

Los que tuvieran como valor de mes alguno no perteneciente al intervalo **[1,12]**.

Los que tuvieran como valor de día un valor **no válido** en concordancia con el mes y el año. Admitiría **29** en un mes de **febrero** sólo en el caso de que el año fuera **bisiesto**.

Cuando la secuencia **no coincidiera** con esta **AAAA-MM-DD**.

Cuando los **separadores** no fueran (-) o (/)

En el campo **Sexo** si hubiéramos introducido una **cadena vacía** nos habría puesto **M** (valor por defecto) pero si hubiéramos intentado introducir un valor que no fuera **M** ni **F** nos habría insertado una **cadena vacía**.

El campo **hora** se comportaría de idéntica forma al de fecha salvo que aquí la secuencia es: **hh:mm:ss**, que la **hora** debe pertenecer al intervalo **[0,23]**, los **minutos** y los **segundos** a **[0,59]** y que el único separador válido en este caso es (:).

El campo **Fumador** requiere particular atención ya que se trata de un campo **CHAR(0)** que sólo admite dos valores: **NULL** o **cadena vacía**, que *como recordarás* son distintos para MySQL.

Para introducir el valor **NULL** -utilizando el procedimiento de inserción de este ejemplo- tienes dos posibilidades, o escribir **NULL sin ponerlo entre comillas** -tal como lo he hecho en el ejemplo o escribir '**\n**' que como ves, es el carácter especial **\n** esta vez **colocado entre comillas**.

Para introducir **una cadena vacía** en este campo bastaría con que pusieramos "**(\n)**" (¡jojo no es una comilla doble es la comilla sencilla () repetida dos veces!)

Date cuenta de que si pusieramos **comillas dobles** tendríamos un **error** ya que hay unas comillas dobles delante del **INSERT** que se cierran al final de la sentencia MySQL y que **no podemos volver a escribirlas** entre ambas para evitar un **falso cierre** de la cadena que contienen.

En este caso el valor del campo **Idiomas** puede contener valores decimales comprendidos entre **0** y **64** o entre sus equivalentes binarios que son **0** y **111111**.

El valor **64** lo justifica el hecho de que **son seis los elementos** que puede contener el campo (hemos definido: Castellano, Francés, Inglés, Alemán, Búlgaro y Chino que **son seis** y que el caso de insertarlos todos requeriría el número binario **111111**, cuyo valor decimal es precisamente 64).

El valor **3** significa lo mismo que su equivalente **binario (11)** o mejor **(000011)** lo cual quiere decir que, **como el primer carácter de la derecha es uno** el campo toma **el primer elemento de la lista** (Castellano), como el segundo (de derecha a izquierda) también es **uno** tomará también el segundo elemento de la lista (Francés) y por ser **cero** todos los demás no tomará ningún otro valor de la lista, con lo que la cadena resultante sería en este caso **Castellano, Francés**.

Fíjate también en que el valor **3** no lo hemos puesto entre comillas porque se trata de una expresión decimal. ¿Qué ocurriría si hubiera puesto **11**? ¿Lo habría interpretado como **once** (decimal) o como **tres** (binario)?

La solución es simple: '**11**' (entre comillas) sería interpretado como binario, sin comillas como decimal.

**No es necesario introducir valores en todos los campos**, es decir, que la lista de campos de la sentencia **INSERT** puede no contenerlos a todos.

**No es necesario** escribir el nombre de los campos **en el mismo orden** en el que fueron creados pero si es **imprescindible** que **campos y valores** estén escritos en la sentencia **INSERT exactamente en el mismo orden** y también que en esa sentencia el **número de campos** y el **número de valores** sea el mismo.

Recuerda que los **values** tipo **numérico** se incluyen entre comillas, mientras que los **no numéricos** tienen que **estar contenidos entre comillas** incluso en el caso de no se inserten directamente sino a través de una variable PHP.

## 2.2- Añadir un registro a partir de datos contenidos en variables

También es posible añadir registros a partir de valores contenidos en variables PHP. Esta opción es, sin ninguna duda, la más utilizada ya que lo habitual será **escribir** el contenido a añadir en un **form** y después -a través del **method** (POST o GET)- pasar al **script** de inserción esos valores como **variables PHP**.

Aquí tienes el código fuente de un ejemplo con la **tabla anterior**.

El valor de **n** ha de ser:

**n <= 256**

dado que el *tamaño máximo de un índice* está limitado en MySQL a **256 bytes**.

Otra limitación de MySQL es el *número máximo de índices de una tabla* que no puede ser mayor de **dieciséis**.

## Los errores MySQL

PHP dispone de dos funciones que nos permiten detectar si una *sentencia MySQL* se ha ejecutado *correctamente* o si se ha producido *algun error*.

Son las siguientes:

**mysql\_errno(\$enl)**

Indica el *número de error* que se ha producido en la *transacción MySQL* realizada a través del *identificador de enlace \$enl*.

Cuando el *número de error* es **CERO** significa que **no se ha producido error**.

Otros valores bastante usuales son los siguientes:

### Error número 1050

Indica que hemos tratado de crear una tabla ya existente.

### Error número 1062

Indica que hemos tratado de introducir un valor con clave duplicada. Aparecerá cuando tratemos de introducir un valor ya existente en un campo con índice único o principal.

**mysql\_error(\$enl)**

Devuelve la *descripción del error*. Cuando el número de error es CERO devuelve una **cadena vacía**.

Resulta de muchísima utilidad para depurar *scripts*.

[Ver código fuente](#)

[Añadir registro](#)

Quizá te resulten de alguna utilidad estos comentarios

Observa en el código fuente que al **insertar las variables** en los **VALUES** de la sentencia MySQL ponemos *cuando se trata de valores tipo cadena '\$variable'* (el nombre de la variable entre comillas) y cuando se trata de *valores numéricos sin comillas*.

Si quieres introducir el valor **NULL** en un campo tipo **VAR(0)** define la variable así: **\$var="\ln"** y si quieres introducir una **cadena vacía** defínela de este otro modo: **\$var="""** -comillas dobles ("") seguidas de **dos** comillas sencillas ('') y para terminar otras comillas dobles ("").

## 3.- Variantes de la sentencia INSERT

La sentencia **INSERT** cuya sintaxis se indica más arriba como:

**INSERT tabla (campo1,campo2,..) VALUES (valor1,valor2,..)**

permite algunos modificadores opciones tales como:

**INSERT [LOW\_PRIORITY | DELAYED] [IGNORE] tabla (campo1,..) VALUES (valor1,..)**

de ellos **LOW\_PRIORITY** y **DELAYED** son incompatibles por lo que solo cabe **uno u otro pero ambos a la vez**.

Veamos su utilidad. La **inserción de registros** y la **lectura de una tabla** son procesos incompatibles, pero cabe la posibilidad de que se **intenten ejecutar** simultáneamente. No olvides que *estamos en Internet* y es perfectamente posible que desde dos ordenadores distintos, dos personas distintas **estén accediendo a la misma tabla** simultáneamente y que uno de los accesos sea de **escritura**, es decir: **añadir, modificar o borrar** campos en la tabla

¿Quién tiene *preferencia de paso*? ¿Quién tiene que esperar?. Si la opción **LOW\_PRIORITY** está activada, el proceso de escritura **esperará a que terminen los procesos de lectura activos** pero si está activa la opción **DELAYED** el proceso de lectura **se interrumpirá automáticamente para ceder el paso** al de escritura.

Respecto a la opción **IGNORE** tiene utilidad cuando se trata de **realizar una secuencia de inserciones**. Si no está activa en el momento en el que aparezca una clave duplicada se **interrumpirá el proceso de inserción**, por el contrario, si estuviera activa el proceso de inserción **continuará** con los siguientes registros de la secuencia, aunque -como es lógico- seguirán sin insertarse los registros con clave duplicada.

## 4.- Tablas para pruebas

Aquí tienes comentado un *script* que permite agregar **aleatoriamente** y de forma **automática** registros a la tabla **demo4**.

Dado que en las páginas siguientes trataremos de consultas va a resultarnos muy cómodo poder rellenarlas de forma automática.

[Ver código fuente](#)

**Insertar datos en  
DEMO4**

### Ejercicio nº 39

Como otra práctica de creación de tablas trata de crear una tabla con los –supuestos– datos más significativos de todos los compañeros de este Curso.

Anterior

Índice

Siguiente



[Ver índice](#)

## Añadir a través de formularios



### Los valores de SELECT MULTIPLE

La opción **SELECT MULTIPLE** dentro de un **form** típico de HTML permite elegir **ninguno**, **uno** o **varios** de los elementos de la lista.

Basta con *pulsar* con el *ratón* sobre cada uno de los *valores elegidos* manteniendo pulsada la tecla **Ctrl**, es decir, puro *Windows*.

Para recoger los valores de esa opción se define -dentro de la etiqueta **SELECT**- un **name** tipo **array**. Bastaría con escribir:

```
<SELECT MULTIPLE  
name=var[] SIZE=6>
```

Como ves, **var** es el nombre de la variable (esta vez sin **\$** delante, recuerda que no estamos en PHP sino en puro HTML) y va **seguido** de **[]** precisamente para indicar que es un **array**.

Lo de **SIZE=6** no es otra cosa que el **parámetro** que indica cuántos elementos de la lista de opciones queremos que se visualicen simultáneamente en la página.

El *truco* está en los **values** de cada **option** dentro de ese **select**:

Los hemos escrito así:

```
<option value=1>Castellano  
<option value=2>Francés  
<option value=4>Inglés  
<option value=8>Alemán  
<option value=16>Búlgaro  
<option value=32>Chino
```

Fíjate que hemos mantenido **exactamente** el mismo **orden** en el que han sido definidos en el campo **SET** de la tabla.

Y fíjate también en los valores: **1**, **2**, **4**, **8**, **16** y **32** que son precisamente las **potencias de 2**:

**2<sup>0</sup>**, **2<sup>1</sup>**, **2<sup>2</sup>**, **2<sup>3</sup>**, **2<sup>4</sup>**, **2<sup>5</sup>**, y **2<sup>6</sup>**

Al ir seleccionando valores, van añadiéndose al **array**. Por ejemplo. Si seleccionamos **Francés** y **Búlgaro** el array sería este: **var[0]=2, var[1]=16**

Si sumamos esos valores (**2 + 16**) el resultado sería **18**, y al convertir a **binario** este valor, resultará:

### Creación del formulario

El caso más frecuente -casi el único- es que los registros de una tabla se añadan utilizando un formulario y **enviando** desde él los datos a un **script PHP** que ejecute la opción de añadir. Si no recuerdas el funcionamiento de este método, [pulsa aquí](#)

En el ejemplo hemos desarrollado un formulario para añadir registros a la tabla **demo4** con las siguientes peculiaridades:

Para los campos **DNI**, **nombre** y **apellidos** hemos utilizado **input** tipo **texto** y hemos recogido mediante la opción **name** cada uno de los campos en una **variable independiente**.

Para los campos **Fecha de nacimiento** y **hora de nacimiento** hemos utilizado **tres** opciones **select** en cada una de ellas.

La finalidad de estas opciones no es otra que **impedir** la introducción de **fechas no válidas** (en realidad no lo impedimos totalmente ya que, tal como está confeccionado, podría introducirse 31 de febrero, o 31 de abril). Ese aspecto es **mejorable**, pero para hacerlo -desde el *propio formulario*- tendríamos de recurrir a un *lenguaje del lado del cliente (JavaScript)* por ejemplo.

Los valores de esos **tres campos** (tanto en fecha como en hora) los recogemos en variables que son elementos de **dos array escalares**.

El campo **sexo** la recogemos en **input** tipo **radio** y les asignamos valores **M** ó **F** que coinciden con los valores del campo **ENUM** de la tabla.

Con el campo **Fumador** -opción **Fumador/No fumador** - hacemos exactamente lo mismo, pero asignándoles valores **1** o **0** ya que el formulario no permite la opción **NULL** ó **cadena vacía**.

En el **script** posterior será cuando modifiquemos los valores de esas variables.

Para el campo **Idiomas** utilizamos una opción **select** de tipo **múltiple** y para los **values** un **pequeño truco** que describimos aquí la izquierda.

Pues bien, aquí tienes, **código fuente** del formulario que hemos diseñado

[Ver código fuente](#)

### Añadir nuevo registro con datos del formulario

Como recordarás, cuando se **envía** el contenido de un **formulario** mediante el **method=POST** y se indica como **action** un **fichero PHP** los valores enviados son recogidos en este último fichero en **variables de PHP** que tienen como nombre **\$\_POST['var']** -o **\$HTTP\_POST\_VARS['var']**- donde cada una de los índices asociativos de los array (var) coinciden con los **name** de los diferentes campos del formulario.

A partir de ahí, bastaría con **depurar los valores recibos, recoger** en variables los valores **depurados e incluirlos** en la **sentencia MySQL INSERT** -la hemos visto en la página anterior- para añadirlos a la tabla correspondiente. Aquí tienes -comentado- el script:

[Código fuente del script](#)

[Añadir regitros](#)

En realidad, tal como habrás podido ver en el **código fuente**, la depuración ha sido la siguiente: Hemos creado un valor de fecha y hora en formatos MySQL válidos de la forma que describimos un poco más arriba.

Hemos **sumado** todos los valores numéricos recibidos en el **array** obtenido del **SELECT MULTIPLE** y hemos asignado el resultado a la **variable depurada** que recoge el valor a escribir en el campo **Idiomas**. La justificación de esa **suma** la tienes al *margen*.

La variable **Fumador** es la que tiene un poquito más de complicación. Veámosla con calma:

Los valores que recibimos desde formulario son **1** o **0** y hemos de transformarlos en **una cadena vacía** o en **NULL**. Hemos insertado un **operador condicional** (un **if... else**) para convertir esos valores en: **\$var=""\N""** (comilla doble, comilla simple, barra invertida, N, comilla simple y comilla doble) ó **\$var="""** (comillas dobles, **dos** comillas simples y unas comillas dobles)

Asignados los nuevos valores tenemos que recurrir a un pequeño *truco*. Venimos repitiendo que en la sentencia **INSERT** los **nombres de las variables no numéricas** que contienen los **values** hay que escribirlos **dentro de comillas simples**, pero en el caso de un campo tipo **CHAR(0)** hemos de hacer una excepción que sería **no poner esas comillas** al nombre de la variable. Al hacerlo así, se escribirían como valores -en la sentencia de inserción- uno de estos: **='N'** o **' '** (los valores de la variable) que al contener **comillas** ya son interpretados por MySQL como una cadena.

### Ejercicio nº 40

**010010**

que es como decirle a MySQL (mirando la cadena de derecha a izquierda, ¿lo recuerdas?) que incluya los valores *segundo* (Francés) y *quinto* (Búlgaro) del SELECT MULTIPLE que corresponden a las posiciones en las que la cadena binaria contiene un uno.

Diseña un formulario –llámalo **altas1.php**– y un script (**ejercicio40.php**) que permitan añadir datos a la **tabla1** que has creado en tu base de datos –**practicas**– en el ejercicio nº 38.

---

[Anterior](#)



[Índice](#)



[Siguiente](#)





Ver índice

## Consultas en tablas



### Sintaxis MySQL de selección de registros

Las *sentencias* de selección de registros requieren utilizar -entre otras- *palabras clave* como las que enumeramos a continuación.

Observa que hay dos tipos: *obligatorias* y *opcionales*, y que algunas de las palabras clave son *alternativas* y por lo tanto, incompatibles en una misma sentencia.

El uso de estas palabras clave *requiere* que sean insertadas en un *determinado orden* tal y como se enumera aquí debajo.

Si alteráramos ese orden (p. ejemplo: colocando GROUP BY antes de WHERE) nos daría un **error** y no se ejecutaría la sentencia.

#### **SELECT**

Es la **primera palabra** de la sentencia de búsqueda y tiene carácter **obligatorio**.

#### **[STRAIGHT\_JOIN]**

Es una *palabra clave* de uso **opcional** (la marcamos con corchetes para indicar su condición de **opcional**) que fuerza al *optimizador MySQL* a organizar las tablas en el mismo orden en el que han sido especificados los campos en la cláusula FORM.

Sirve para mejorar -en casos muy concretos- la velocidad de gestión de tablas de gran tamaño.

#### **[SQL\_BIG\_RESULT]**

Es una *cláusula opcional* que se usa para indicar al *optimizador* que el resultado va a tener **una gran cantidad de registros**.

En ese caso, MySQL utilizará *tablas temporales* cuando sea necesario para optimizar la velocidad de gestión de la información.

Esta *cláusula* también puede ser utilizada dentro de GROUP BY.

#### **[SQL\_BUFFER\_RESULT]**

Es **opcional** y su finalidad es la de **forzar** a MySQL a tratar el resultado en un *fichero temporal*.

Ese tratamiento ayuda a MySQL a ...

### Consultar los registros de una tabla

Las consultas de los datos y registros contenidos en una tabla ofrecen un amplísimo abanico de posibilidades a partir de las opciones que tienes descritas al margen. Veamos algunas de las posibilidades.

#### La consulta más simple

Si utilizamos la sentencia

**SELECT \* FROM tabla**

obtendremos información sobre **todos los campos (\*)** y la salida estará en el mismo orden en el que fueron añadidos los datos. Si visualizas este ejemplo, verás que aparecen ordenados por el valor **autonumérico** del campo **Contador** lo cual, como ves, resulta coherente con la afirmación anterior.

[Ver código fuente](#)

[Ejecutar la consulta](#)

#### Consultando sólo *algunos campos*

Ahora utilizaremos la sentencia

**SELECT campo1,campo2, ... FROM tabla**

y tendremos como resultado una lista completa, por el mismo orden que la anterior, pero sólo **mostrando** los campos indicados.

[Ver código fuente](#)

[Ejecutar la consulta](#)

#### ¡Cuidado!

En los comentarios contenidos en estos ejemplos puedes ver la forma en la que **mysql\_fetch\_row** y **mysql\_fetch\_array** tratan los índices escalares de los resultados que producen los SELECT de MySQL.

Los valores de los índices se asignan a los contenidos de los campos por el mismo orden en el que estos se escriben en la sentencia SELECT. El **campo1** (primero que se escribe) será recogido por el elemento de índice **cero** del array, el **campo2** será recogido con índice **uno** y así sucesivamente

#### Consultando sólo *algunos campos* y limitando la salida a *n* registros

Ahora utilizaremos la sentencia

**SELECT campo1,campo2, ... FROM tabla LIMIT (n, m)**

y tendremos como resultado una lista que contendrá **m** registros a partir del **n+1**, por el mismo orden que la anterior, y **mostrando** los campos indicados.

[Ver código fuente](#)

[Ejecutar la consulta](#)

#### Consultando sólo *algunos campos* y ordenando la salida

Utilizaremos la sentencia MySQL de esta forma

**SELECT campo1,campo2, ... FROM tabla ORDER BY campo\_n [ASC|DESC], campo\_m [ASC|DESC]**

y tendremos como resultado una lista ordenada por el primero de los campos indicados en **ORDER BY**, y en caso de **coincidencia** de valores en ese campo, utilizaríamos el criterio de ordenación señalado en segundo lugar.

es de gran utilidad (siempre desde el punto de vista de la rapidez) cuando es necesario un largo proceso de cálculo antes de enviar los resultados al cliente.

#### **[HIGH\_PRIORITY]**

Esta cláusula, **opcional** da prioridad al comando SELECT sobre otros comandos que simultáneamente pudieran estar intentando acceder a la tabla para **escribir** en ella (añadir o modificar registros).

Si esta opción está activa, los intentos de escritura que pudieran producirse de forma simultánea deberían esperar al final de este proceso para ejecutarse.

**campo1, campo2, ...**

Tienen carácter **obligatorio** y **señalan los campos de la tabla** que deben incluirse en la consulta.

La función SELECT sólo devolverá información de aquellos campos que estén enumerados aquí.

Si se desea que la consulta **incluya a todos campos** bastará con incluir en esta posición un **\***, que es el carácter **comodín** que indica a MySQL que se desea incluir todos los campos en la consulta.

Los **campos numéricos** tienen la opción de llevar asociadas **funciones** MySQL que devuelven información **estadística**.

Algunas de esas funciones son las siguientes:

#### **MAX(campo..)**

Devuelve el valor **máximo** de ese campo en todos los registros de la tabla, salvo que tenga la opción GROUP BY, en cuyo caso devolverá el máximo de cada grupo, o cuando tenga activada la opción WHERE, en cuyo caso la función sólo será aplicada a los registros que resulten de tal **filtrado**.

#### **MIN(campo..)**

Idéntica a la anterior en cuanto a criterios de selección, esta función devuelve el **mínimo**.

#### **AVG(campo..)**

Devuelve el valor **promedio** de todos los registros numéricos seleccionados con los mismos criterios del caso anterior.

#### **SUM(campo..)**

Devuelve la **suma** de los valores del **campo** y sigue idénticos criterios de selección de campos que en los casos anteriores.

#### **STDDEV(campo..)**

Devuelve la estimación de la **desviación típica** de la población.

#### **COUNT(campo..)**

Devuelve el número de registros que cumplen la condición indicada.

[Ver código fuente](#)

[Ejecutar la consulta](#)

## Consulta **seleccionando registros**

Utilizaremos la sentencia MySQL de esta forma

**SELECT campo1, ... FROM tabla WHERE condición**

que nos devolverá la lista de registros que **cumplen la condición indicada**. Aquí tienes un ejemplo muy sencillo.

[Ver código fuente](#)

[Ejecutar la consulta](#)

La cláusula **WHERE** permite un variado abanico de **condiciones**, que trataremos de resumir aquí. Algunos de ellas son los siguientes:

Operador	Tipo de campo	Sintaxis	Descripción	Código fuente	Ver ejemplo
=	Numérico	WHERE campo=num	Selecciona los registros que contienen en el <b>campo</b> un <b>valor igual a num</b>	<a href="#">Ver</a>	<a href="#">Probar</a>
=	Cadena	WHERE campo="cadena"	Selecciona los registros que contienen en el <b>campo</b> una <b>cadena idéntica a cadena</b> (*)	<a href="#">Ver</a>	<a href="#">Probar</a>
<	Numérico	WHERE campo<num	Selecciona los registros que contienen en el <b>campo</b> un <b>valor menor a num</b>	<a href="#">Ver</a>	<a href="#">Probar</a>
<	Cadena	WHERE campo<"cadena"	Selecciona los registros que contienen en el <b>campo</b> una <b>cadena cuyos n primeros caracteres son menores</b> que los de la <b>cadena</b> , siendo <b>n</b> el número de caracteres que contiene <b>cadena</b> . (**)	<a href="#">Ver</a>	<a href="#">Probar</a>
<=	Numérico	WHERE campo<=num	Selecciona los registros que contienen en el <b>campo</b> un <b>valor menor O igual a num</b>	<a href="#">Ver</a>	<a href="#">Probar</a>
<=	Cadena	WHERE campo<="cadena"	Selecciona los registros que contienen en el <b>campo</b> una <b>cadena cuyos n primeros caracteres son menores</b> que los de la <b>cadena</b> , siendo <b>n</b> el número de caracteres que contiene <b>cadena</b> y añade respecto al caso anterior la opción de que en caso de que <b>ambos valores fueran iguales</b> también los presentaría (**)	<a href="#">Ver</a>	<a href="#">Probar</a>
>	Numérico	WHERE campo>num	Selecciona los registros que contienen en el <b>campo</b> un <b>valor mayor a num</b>	<a href="#">Ver</a>	<a href="#">Probar</a>
>	Cadena	WHERE campo>"cadena"	Selecciona los registros que contienen en el <b>campo</b> una <b>cadena cuyos n primeros caracteres son mayores</b> que los de la <b>cadena</b> , siendo <b>n</b> el número de caracteres que contiene <b>cadena</b> . (**)	<a href="#">Ver</a>	<a href="#">Probar</a>
>=	Numérico	WHERE campo>=num	Selecciona los registros que contienen en el <b>campo</b> un <b>valor mayor o igual a num</b>	<a href="#">Ver</a>	<a href="#">Probar</a>
>=	Cadena	WHERE campo>="cadena"	Selecciona los registros que contienen en el <b>campo</b> una <b>cadena cuyos n primeros caracteres son mayores</b> que los de la <b>cadena</b> , siendo <b>n</b> el número de caracteres que contiene <b>cadena</b> y añade respecto al caso anterior la opción de que en caso de que <b>ambos valores fueran iguales</b> también los presentaría (**)	<a href="#">Ver</a>	<a href="#">Probar</a>
IN	Numérico o Cadena	WHERE campo IN (valor1,valor2..)	Selecciona los registros que contienen en el <b>campo</b> valores que coinciden con alguno de los especificados dentro del paréntesis. Cuando se trata de valores <b>no numéricos</b> han de ir <b>entre comillas</b>	<a href="#">Ver</a>	<a href="#">Probar</a>
BETWEEN	Numérico o Cadena	WHERE campo BETWEEN valor1 AND valor2	Selecciona los registros en los que los valores contenidos en el <b>campo</b> seleccionado están comprendidos en el intervalo <b>valor1</b> (mínimo) – <b>valor2</b> (máximo) incluyendo en la selección ambos extremos. Cuando los contenidos de los campos son <b>cadenas</b> sigue los mismos criterios que se indican para los demás operadores de comparación	<a href="#">Ver</a>	<a href="#">Probar</a>
IS NULL	Cadena	WHERE campo IS NULL	Selecciona los registros en los que los valores contenidos en el <b>campo</b> seleccionado son <b>NULOS</b>	<a href="#">Ver</a>	<a href="#">Probar</a>
IS NOT NULL	Cadena	WHERE campo IS NOT NULL	Selecciona los registros en los que los valores contenidos en el <b>campo</b> seleccionado son <b>NO NULOS</b>	<a href="#">Ver</a>	<a href="#">Probar</a>

(\*) Cuando se trata de cadenas de caracteres, el concepto **menor que** significa **anterior** en la ordenación de los caracteres según su código ASCII y **mayor que** significa **posterior** en esa misma ordenación.

(\*\*) La discriminación de Mayúsculas/Minúsculas dependerá del tipo de campo.

Recuerda que los tipo **BLOB** hacen esa discriminación, mientras que los de tipo **TEXT** son insensibles a Mayúsculas/Minúsculas.

Cuando se trata de comparar **cadenas** MySQL dispone de una potente instrucción (**LIKE**) que permite establecer los criterios de selección **a toda o parte** de la cadena. Su sintaxis contempla

campo indicado. En el caso de aplicar estas funciones, el **resultado de la consulta** contiene una sola línea, salvo que active la opción GROUP BY, en cuyo caso devolverá **tantas líneas como grupos resulten**.

#### **FROM tabla**

Esta **expresión** -que aunque no tiene carácter obligatorio podría tomarse como tal- indica a MySQL el **nombre de la tabla** en el que debe efectuarse la consulta.

#### **WHERE definición**

Esta **instrucción** tiene carácter opcional y su utilidad es la de **filtrar** la consulta estableciendo los criterios de selección de los **registros** que debe devolver.

Si se omite WHERE, la consulta devolverá **todos** los registros de la tabla.

En la parte derecha tienes información sobre la manera de **definir** los criterios de selección de esta opción.

#### **GROUP BY definición**

Tiene carácter **opcional** y su finalidad es la de presentar los resultados de la consulta **agrupados** según el criterio establecido en su **definición**.

Resulta de gran utilidad cuando se pretende obtener **valores estadísticos** de los registros que cumplen **determinadas condiciones** (las condiciones del **agrupamiento**).

#### **ORDER BY definición**

También tiene carácter **opcional** y su utilidad es la de presentar la información de la consulta **ordenada** por los contenidos de **uno o varios campos**.

Siempre tiene como opción complementaria de que **en cada campo utilizado para la ordenación** puede establecerse uno de estos criterios **ASC** (ascendente, es el valor por defecto) o **DESC**.

Si no se establece ningún orden, los resultados de la consulta aparecerán en el mismo orden en el que fueron añadidos los registros.

#### **LIMIT m, n**

Esta cláusula es **opcional** y permite establecer **cuántos** y **cuáles** registros han de presentarse en la salida de la consulta.

Por ejemplo: **LIMIT 4, 8** indicaría a MySQL que la consulta debería **mostrar OCHO** registros contados a partir del **quinto** (es el quinto porque

distintas posibilidades utilizando **dos comodines**: **%** (que se comporta de forma similar al (\*) en las búsquedas de Windows) y **\_** (de comportamiento similar a (?) en Windows). Aquí tienes algunas de sus posibilidades:

Sintaxis	Descripción	Código fuente	Ver ejemplo
WHERE campo LIKE '%cadena%'	Selecciona todos los registros que contengan la <b>cadena</b> en el <b>campo</b> indicado sea cual fuere su posición	Ver	Probar
WHERE campo LIKE 'cadena%'	Selecciona todos los registros en los que el <b>campo</b> indicado que contengan la <b>cadena exactamente al principio del campo</b>	Ver	Probar
WHERE campo LIKE '%cadena'	Selecciona todos los registros en los que el <b>campo</b> indicado que contengan la <b>cadena exactamente al final del campo</b>	Ver	Probar
WHERE campo LIKE '_cadena%'	Selecciona todos los registros en los que el primer carácter del <b>campo</b> puede ser cualquiera pero los siguientes han de ser <b>exactamente</b> los indicados en <b>cadena</b> pudiendo ir seguidos de <b>cualesquier otros caracteres</b>	Ver	Probar

El comodín **(\_)** puede ir tanto al principio como al final y puede repetirse tantas veces como sea necesario. Sería correcto **LIKE '\_\_\_es%'** y también **LIKE 'a\_\_\_es%'** así como: **LIKE '%a\_\_\_es'**.

Como ves, *un montón* de posibilidades.

Aún tiene más opciones **WHERE** ya que acepta múltiples condiciones vinculadas por los operadores lógicos **AND, OR, NOT** o sus **sintaxis equivalentes: &&, || y !**

El comportamiento de estos operadores es idéntico al descrito para sus homónimos de PHP. ¿Los recuerdas?... Aquí los tienes... por si acaso.

Un ejemplo de sintaxis puede ser:

**WHERE (campo1=valor AND campo2 LIKE '\_cadena%)**

## **Utilizando funciones sobre campos**

La sintaxis

**SELECT MAX(campo1), MIN (campo2), ... FROM tabla**

nos devolvería UNA SOLA FILA cuyos valores serían los resultados de la aplicación de las funciones **a todos los registros** del campo indicado.

Aquí tienes un ejemplo que determina todos los valores de esos estadísticos aplicados al campo **Contador** de nuestra famosa tabla **demo4**.

Aquí está el ejemplo

[Ver código fuente](#)

[Ejecutar la consulta](#)

## **Aplicando la opción GROUP BY**

Tal como señalamos al margen, las funciones anteriores pueden aplicarse a **grupos** de registros seleccionados mediante un criterio **GROUP BY (nombre del campo)**

En este ejemplo obtendremos los mismos parámetros estadísticos que en el anterior, pero ahora agrupados por **sexo**, lo que significaría que obtendremos **dos filas** de resultados. Aquí tienes el ejemplo

[Ver código fuente](#)

[Ejecutar la consulta](#)

Como habrás podido observar, la opción SELECT tiene un sinfín de posibilidades.

## **Creación de tablas a partir de la consulta de otra tabla**

Es frecuente -podría decirse que es lo habitual- **relacionar tablas** mediante campos con **idéntico contenido**.

Supongamos que **entre los individuos** de nuestra tabla **demo4** se pretende establecer un **proceso de selección** para elegir entre ellos un número determinado de **astronautas**, pongamos por caso.

LIMIT considera el primer registro como CERO).

El criterio límite se aplica sobre los resultados de la salida, es decir, sobre los resultados **seleccionados, ordenados y filtrados** siguiendo los criterios establecidos por las cláusulas anteriores.

Si se escribe como un solo parámetro (LIMIT k), MySQL lo interpretará como que k es el segundo de ellos y que el primero es CERO, es decir:  
LIMIT 0, k

## Recuento de resultados

PHP dispone de dos funciones que permiten conocer el número de registros de la tabla afectados por una sentencia MySQL.

### mysql\_num\_rows (\$c )

Esta función devuelve un valor numérico que recoge el número de registros que cumplen las condiciones establecidas en una **consulta**. Sólo es válido para sentencia tipo SELECT

### mysql\_affected\_rows(\$c )

En este caso la función devuelve también el número de registros afectados, pero sólo en el caso de que la sentencia MySQL haya producido modificaciones en los contenidos de la tabla. Es decir, sólo recoge resultados de sentencias que: **añaden, modifican o borran** registros.

## Manejo de fechas en las consultas

MySQL dispone de algunas cláusulas de gestión de fechas que pueden tener una gran utilidad a la hora de gestionar consultas. Son las siguientes:

### DATE\_FORMAT( campo,formato)

Las diferentes opciones de formato las tienes en la tabla de la derecha. Es importante tener en cuenta que la sintaxis correcta es **%Y** (sin espacio) ya que si hubiera un espacio **% Y** interpretaría la letra Y como un texto a incluir.

### CURDATE()

Dentro de DATE\_FORMAT se puede incluir -en vez del nombre del campo- una cadena en la que se indique una fecha en formato **YYYY-MM-DD hh:mm:ss**. Puedes verlo en los ejemplos. De igual modo es posible sustituir el nombre del campo -o la cadena- por la función CURDATE() que recoge la **fecha actual del sistema** (únicamente día, mes y año). A efectos de horas, minutos y segundos CURDATE() va

Supongamos también, que la selección va a constar de **tres** pruebas que serán juzgadas y calificadas por **tres tribunales** distintos.

Una primera opción sería crear tres tablas -una para cada tribunal- e incluir en ellas todos los datos de cada uno de los individuos.

Esa opción es factible pero no es *ni la más cómoda*, ni tampoco es la más *rápida* ni la que *menos espacio de almacenamiento* necesita. No debemos olvidar que una tabla puede tener una enorme cantidad de registros.

Una opción alternativa sería crear **tres nuevas tablas** que sólo contuvieran **dos campos** cada una. Por ejemplo el campo **DNI** y el campo **Calificación**.

Como quiera que el campo **DNI** ha de contener los mismos valores en las **cuatro** tablas y además es un campo **único** podrían crearse las nuevas tablas y luego **copiar** en cada una de ellas **todos los DNI** de la tabla original.

Nos garantizaría que *no habría errores* en los DNI y además nos garantizaría que **se incluyeran todos** los aspirantes en esas nuevas tablas.

Aquí tienes el *código fuente* de un *script* que crea esas tres tablas (a las que hemos llamado **demodat1, demodat2 y demodat3**).



## Una consulta conjunta de varias tablas

MySQL permite realizar consultas simultáneas en registros situados en varias tablas.

Para ese menester se usa la siguiente sintaxis:

**SELECT tabla1.campo1, tabla2.campo2, ... FROM tabla1, tabla2**

en la que, como ves, modificamos *ligeramente* la sintaxis ya que **anteponemos el nombre de la tabla** al del **campo** correspondiente separando ambos nombres por un punto, con lo cual no hay posibilidad de error de identificación del campo **incluso cuando campos de distinta tabla tengan el mismo nombre**.

Otra *innovación* -respecto a los ejemplos anteriores- es que detrás de la **cláusula FROM** escribimos los nombres de todas las tablas que está usando **SELECT**.

A partir de ahí se pueden establecer todo tipo de relaciones para las sentencias **WHERE, ORDER BY** y **GROUP BY** utilizando para ello **campos de cualquiera de las tablas** sin otra particularidad más que **poner cuidado al aludir a los campos** utilizando siempre la sintaxis **nombre\_tabla.nombre\_campo**.

A modo de ejemplo -hemos procurado comentarlo línea a línea- aquí tienes un *script PHP* que hace una **consulta conjunta** de las tablas **demo4, demodat1, demodat2 y demodat3** y nos presenta una tabla con los datos personales y las puntuaciones de las **tres pruebas** así como las **suma de puntos** de las tres y, además, ordena los resultados -de mayor a menor- según la **suma de las tres puntuaciones**.

[Ver código fuente](#) [Ejecutar la consulta](#)

## Formatos de fechas en consultas MySQL

Los formatos soportados por la función DATE\_FORMAT format son los siguientes:

Formato	Descripción	Sintaxis	Ver código	Ver ejemplo
%d	Día del mes en formato de dos dígitos	DATE_FORMAT(Nacimiento,'%d')	<a href="#">Ver</a>	<a href="#">Probar</a>
%e	Día del mes en formato de uno ó dos dígitos	DATE_FORMAT(Nacimiento,'%e')	<a href="#">Ver</a>	<a href="#">Probar</a>

a tomar el *mediodía* de la fecha actual.

#### CURTIME()

Se comporta de forma similar a CURDATE().

Devuelve la hora actual del sistema que alberga el servidor MySQL en formato **hh:mm:ss**

#### CURRENT\_TIMESTAMP()

Se comporta de forma similar a CURDATE().

Devuelve la fecha y hora actual del sistema en formato **YYYY-MM-DD hh:mm:ss**

#### NOW()

Es un *alias* de

#### CURRENT\_TIMESTAMP().

#### **mysql\_result(\$resultado,num,campo)**

Esta función PHP permite obtener un solo campo de uno solo de los registros obtenidos como resultado de una consulta MySQL.

El parámetro **\$resultado** es la variable que recoge en resultado obtenido de la ejecución de **mysql\_query** de forma idéntica a como lo hacíamos en otras consultas.

El valor **num** es un número entero que indica el número de fila de la que queremos extraer el valor contenido en uno de sus campos.

El valor **campo** indica el *número del campo* que tratamos de extraer. Este número (la primera posición siempre es **cero**) indica el número de orden del campo tal como está especificado en la sentencia SELECT. Si en esta sentencia se incluyera \* (extraer todos los campos) consideraría el orden en el que está creada la estructura de la tabla que los contiene.

Este es el **código fuente** de un ejemplo comentado y este un enlace de prueba del script.

%D	Número de dia seguido del sufijo en inglés	DATE_FORMAT(Nacimiento,'%D')	Ver	Probar
%m	Número del mes en formato de dos dígitos	DATE_FORMAT(Nacimiento,'%m')	Ver	Probar
%c	Número del mes en formato de uno o dos dígitos	DATE_FORMAT(Nacimiento,'%c')	Ver	Probar
%M	Nombre del mes (en inglés)	DATE_FORMAT(Nacimiento,'%M')	Ver	Probar
%b	Nombre del mes abreviado (en inglés)	DATE_FORMAT(Nacimiento,'%b')	Ver	Probar
%y	Número del año en formato de dos dígitos	DATE_FORMAT(Nacimiento,'%y')	Ver	Probar
%Y	Número del año en formato de cuatro dígitos	DATE_FORMAT(Nacimiento,'%Y')	Ver	Probar
%w	Número de día de la semana 0=Domingo ... 6=Sábado	DATE_FORMAT(Nacimiento,'%w')	Ver	Probar
%W	Nombre del día de la semana (en inglés)	DATE_FORMAT(Nacimiento,'%W')	Ver	Probar
%W	Nombre abreviado del día de la semana (en inglés)	DATE_FORMAT(Nacimiento,'%W')	Ver	Probar
%j	Número de día del año en formato de 3 dígitos	DATE_FORMAT(Nacimiento,'%j')	Ver	Probar
%U	Número de semana del año considerando el DOMINGO como primer día de la semana (en formato de dos dígitos)	DATE_FORMAT(Nacimiento,'%U')	Ver	Probar
%u	Número de semana del año considerando el LUNES como primer día de la semana (en formato de dos dígitos)	DATE_FORMAT(Nacimiento,'%u')	Ver	Probar

La fecha para los ejemplos siguientes la extraemos de una variable del tipo:

**\$fecha="2005-10-12 14:23:42"**

ya que la tabla no contiene campos de fecha que incluyan horas, minutos y segundos

%H	Hora con dos dígitos (formato 0 a 24 horas)	DATE_FORMAT(\$fecha,'%H')
%k	Hora con uno ó dos dígitos (formato 0 a 24 horas)	DATE_FORMAT(\$fecha,'%k')
%h	Hora con dos dígitos (formato 0 a 12 horas)	DATE_FORMAT(\$fecha,'%h')
%l	Hora con uno ó dos dígitos (formato 0 a 12 horas)	DATE_FORMAT(\$fecha,'%l')
%i	Minutos con dos dígitos	DATE_FORMAT(\$fecha,'%i')
%s	Segundos con dos dígitos	DATE_FORMAT(\$fecha,'%s')
%r	Hora completa (HH:mm:ss) en formato de 12 horas indicando AM ó PM	DATE_FORMAT(\$fecha,'%r')
%T	Hora completa (HH:mm:ss) en formato de 24 horas	DATE_FORMAT(\$fecha,'%T')
% texto	Incluye el texto que se indica detrás del %	DATE_FORMAT(\$fecha,'% texto')
%p	Añade AM ó PM dependiendo de la Hora	DATE_FORMAT(\$fecha,'%p')

Se pueden combinar a voluntad varias opciones utilizando una sintaxis de este tipo:  
**'% Hoy es: %d - %m - %Y % es %W % estamos en el mes de %M % <br>y van transcurridos %j % días de este año.<br>Son las %r'**

Ver	Probar

#### Ejercicio nº 41

En esta actividad debes elaborar varios scripts –puedes llamarlos **ejercicio41\_1.php**, etcétera– que permitan realizar consultas en la base de datos que has creado en el ejercicio nº 38.

Previamente, tendrás que añadirle datos, bien manualmente o bien modificando el ejemplo de generación de registros aleatorios que hemos incluido en la página anterior.

#### Ejercicio nº 42

Construye una nueva tabla –**tabla2**– con los mismos campos que tu **tabla1** pero añadiendo el carácter de clave principal al campo que recoge el DNI, con lo cual podrás impedir que puedan repetirse dos alumnos con el mismo DNI.

A partir de ella, crea tablas auxiliares (transfiriendo los datos de **tabla1**) –de calificaciones de materias, por ejemplo– que contengan dos campos: DNI y calificación. Por último, tendrás que crear todo lo necesario para que el profesor de cada materia, pudiera insertar sus calificaciones y, además, crea un documento final que permita visualizar simultáneamente las calificaciones del alumno en todas la materias.

Anterior



Índice



Siguiente





[Ver índice](#)

## Modificar registros



### Sintaxis MySQL de modificación de registros

Las sentencias MySQL que permiten la modificación de registros en las tablas pueden incluir algunas de las siguientes *cláusulas* que, al igual que ocurría en casos anteriores, pueden tener categoría de **obligatorias u opcionales**.

El orden en que deben estar indicadas ha de seguir la misma secuencia en la que están descritas aquí.

#### **UPDATE**

Tiene carácter **obligatorio**, debe ser la **primera palabra de la sentencia** e indica a MySQL que vamos realizar una **modificación**.

#### **[LOW\_PRIORITY]**

Es **opcional** e indica a MySQL **espere a que se terminen de hacer** las consultas que en ese momento pudiera haber en proceso antes realizar la actualización.

#### **[IGNORE]**

Es **opcional**. Cuando se incluye en una sentencia el proceso de actualización *no se interrumpe* si aparece un conflicto de clave duplicada en uno de los registros en proceso. Simplemente ignora ese registro y continúa con los siguientes.

Si no se incluye, el proceso de modificación **se interrumpe** en el momento en que encuentre **un conflicto de clave duplicada**.

Tanto con **ignore** como sin esa *cláusula*, en el caso de duplicidad de clave **NUNCA** se efectúan las modificaciones.

#### **tabla**

Es **obligatoria** y contiene el **nombre** de la tabla que pretendemos modificar.

#### **SET**

Tiene carácter **obligatorio** y debe estar *delante* de las definiciones de **campo** y **valor**.

#### **campo1 = valor1**

Es **obligatoria** al menos una definición. Indica el **nombre del campo** a modificar (**campo1**) y el **valor** que se asignará a ese campo.

### Modificar un campo en todos los registros de una tabla

La sentencia MySQL, que permite **modificar** uno o varios campos en **todos** los registros de una tabla, es la siguiente:

[Ver índice](#)

[Búsqueda rápida](#)

[Página anterior](#)

[Página siguiente](#)

*Cuidado con esta sentencia!* Hay que tener muy presente que con esta sentencia -en la que no aparece WHERE- se modificarán **TODOS LOS REGISTROS DE LA TABLA** y por lo tanto **los campos modificados** tendrán el **mismo valor** en todos los registros.

### Algunas consideraciones sobre la sintaxis

Siempre que manejes PHP y MySQL debes tener muy presente lo siguiente:

MySQL requiere **SIEMPRE** que los valores tipo **cadena** que incluyen campos de fecha vayan **entre comillas**. Por el contrario, los **numéricos no deben llevar comillas**.

Presta mucha atención a esto cuando **escribas** los valores directamente en la sentencia MySQL. Cuando **pases valores** desde una variable PHP debes tener muy en cuenta las consideraciones anteriores y si el contenido de la variable es una cadena que va a ser tratada como tal por MySQL tienes dos opciones para evitar el error:

Definir la variable así: **\$variable = "valor"** (comillas dobles, comilla simple *al principio* y comilla simple, comilla doble *al final*) y poner en la sentencia MySQL el nombre de la variable sin entrecomillar, o Definir la variable PHP así: **\$variable = "valor"** y al escribir el nombre de esa variable en la sentencia MySQL escribirlo entre **comillas sencillas**, es decir, así: **'\$variable'**

No pienses que es caprichoso el *orden* que hemos puesto en las comillas. Recuerda que al llamar a la sentencia MySQL, el contenido de la sentencia **va entre comillas** (que por costumbre son **comillas dobles**, por esa razón **todo entrecomillado** que vaya dentro de esa sentencia ha de usar **comillas simples** para evitar un **error seguro**).

De ahí que al definir una variable PHP en la forma **\$variable = "valor"** las comillas dobles exteriores indican a PHP que se trata de una cadena, por lo que, al pasar la variable a MySQL éste recibirá el contenido de la cadena que es, lógicamente: **'valor'** y en este caso las comillas forman parte del valor, razón por el que no es necesario escribir -en la sentencia MySQL- el nombre de la variable entrecomillado.

En este primer ejemplo, hemos incluido una actualización de tablas que pondrá *puntuación* 7 en la primera de las pruebas a todos los *aspirantes a astronautas* de nuestro ejemplo.

Es un caso de actualización sin la condición WHERE y tiene el código comentado.

[Ver código fuente](#)

[Ejecutar la modificación](#)

### Selección y modificación de un solo registro

Es una de las opciones más habituales. Es el caso en el que –mediante un formulario– asignamos una condición a WHERE y simultáneamente asignamos los nuevos valor del campo o campos elegidos. Requiere la siguiente sintaxis:

#### **UPDATE tabla SET campo1=valor1, campo2=valor2 WHERE condición**

La **condición** es fundamental en esta opción y normalmente aludirá a un campo índice (clave principal o única), de modo que sea un solo registro el que cumpla la condición. Podría ser el caso, en nuestro ejemplo, del campo DNI que por su unicidad garantizaría que la modificación solamente va a afectar a uno solo de los registros.

El ejemplo siguiente nos permitirá hacer modificaciones de este tipo en la tabla **deomodat2**. Observa el código fuente y verás que mediante un simple recurso JavaScript, el script que realiza la modificación nos reenvía al formulario con un mensaje de confirmación de la modificación.

Si se pretende modificar **más de un campo** se repetirá esta definición tantas veces como sea necesario, separando cada una de ellas por **una coma**.

#### WHERE

Es un campo **opcional** y su comportamiento es idéntico a señalado al mencionar el proceso de consultas.

#### ORDER BY

Tiene idéntica funcionalidad a la descrita al referirnos a consultas

[Ver código «formulario»](#)[Ver código del «script»](#)[Ejecutar la modificación](#)

## Modificación simultánea de un campo en cualquiera de los registros

Aquí tienes un ejemplo que permite visualizar **el valor actual** de todas las puntuaciones de la **prueba 2** de los **astronautas** así como sus nombres y apellidos y DNI y en la cual se pueden modificar **ninguno, uno, varios** o todos los valores y posteriormente actualizarlos todos con los nuevos valores.

[Ver código «formulario»](#)[Ver código del «script»](#)[Ejecutar la modificación](#)

### Ejercicio nº 43

Crea los formularios y scripts necesarios para poder elegir un alumno cualquiera mediante su DNI –en la **tabla1**– y modificar cualquiera de sus datos personales.

[Anterior](#)[Índice](#)[Siguiente](#)



[Ver índice](#)

## Borrar registros y salvar datos



### Sintaxis MySQL para borrado de registros

La sintaxis MySQL para las sentencias de borrado de registros de una tabla puede contener las siguientes cláusulas que, al igual que ocurría en casos anteriores, pueden tener categoría de obligatorias u opcionales.

La secuencia en la que deben estar indicadas en la sentencia es idéntica al orden en que están descritas aquí.

#### **DELETE**

Tiene carácter obligatorio. Debe ser la primera palabra de la sentencia e indica a MySQL que tratamos de borrar uno o más registros.

#### **LOW\_PRIORITY**

Es opcional e indica a MySQL que espere para realizar la actualización a que terminen las consultas del fichero (en el caso de haber alguna en proceso).

#### **FROM**

Tiene carácter obligatorio y debe preceder a la definición de la tabla en la que se pretende eliminar registros.

#### **tabla**

Es obligatoria e indica el nombre de la tabla en la que pretendemos efectuar el borrado o eliminación de los registros.

#### **WHERE**

Es un campo opcional y su comportamiento es idéntico al señalado en al mencionar el proceso de consultas.

#### **LIMIT n**

La opción LIMIT es opcional y propia de MySQL.

Su finalidad es limitar el tiempo de ejecución del comando DELETE ya que cuando está activada devuelve el control al potencial cliente después de borrar n registros, con lo que en procesos de borrados muy largos (ficheros de gran tamaño) no obliga a esperar a borrado total para proceder a la consulta de la tabla.

Cuando se utiliza esta opción, la sentencia DELETE debe repetirse hasta que el número de registros

### Borrar todos los registros de una tabla

La sentencia MySQL que permite borrar todos los registros de una tabla es la siguiente:

**DELETE FROM tabla**

Ten muy presente que con esta sentencia -en la que no aparece WHERE- se BORRARÁN TODOS LOS REGISTROS DE LA TABLA.

Respecto a otras posibles opciones no difiere en nada de lo indicado en la página anterior. Simplemente habría que sustituir en aquellos script UPDATE por DELETE. Borrar un registro no es otra cosa que un caso particular de modificación.

### Integridad referencial tras el borrado de una tabla

¿Recuerdas el ejemplo de las pruebas de selección de astronautas? ¿Recuerdas que las tres tablas de puntuaciones habían sido creadas a partir de la tabla de datos de los aspirantes? ¿Qué ocurriría si borrásemos uno o varios registros de una de ellas? ¿Qué ocurriría se después de crear esas tablas añadiésemos nuevos aspirantes a la lista de candidatos?

Es obvio que si no hacemos algo para evitarlo se perdería la integridad referencial - la relación uno a uno - entre los registros de esas tablas.

Ocurriría que no todos los individuos que están incluidos en una de esas tablas lo estarían en las demás y por tanto, al ejecutar consultas o modificaciones posteriores correríamos el riesgo de que se produjeran errores.

Esa situación es fácilmente evitable modificando ligeramente los scripts con los que se realizan los procesos de altas y bajas.

Bastaría con añadirles algunas sentencias que cada vez que se efectúa un alta o baja en el fichero de datos personales efectúen el mismo proceso en todos los demás ficheros relacionados con aquél.

Aquí tienes comentado el código fuente de la modificación añadida al script que registra los nuevos aspirantes en el fichero de altas de la tabla demo4. Con esta modificación se actualizarían automáticamente los ficheros demodat1, demodat2 y demodat3 cada vez que se añadiera un nuevo aspirante.

El formulario no requiere ninguna modificación, los cambios sólo es necesario realizarlos en el script que realiza la inserción.

[Ver código fuente](#)

[Añadir un nuevo aspirante](#)

Hecho este pequeño inciso -creemos que importante y necesario - continuaremos con la referencia al borrado de registros.

En este ejemplo, tienes el código fuente de un script que realiza el borrado de un registro -mediante un formulario en el que se inserta el DNI- tanto en la tabla demo4 como demodat1, demodat2 y demodat3 manteniendo la integridad referencial entre los cuatro ficheros.

[Ver script](#)

[Borrar un registro](#)

### Borrar registros seleccionándolos de una lista

En el ejemplo siguiente tienes el código para utilizar la cláusula WHERE en un proceso de borrado de registros que presenta un formulario que contiene una lista con todos los registros actuales y una casilla de verificación por cada uno.

Al marcar las casillas y enviar el formulario el script que recibe los datos procede al borrado de todos los registros marcados en todas las tablas afectadas.

pendientes de borrado sea inferior al valor de *n*.

ver formulario

ver script

Ejecutar ejemplo

## Optimización de tablas

Cuando se ejecuta la sentencia `DELETE` -pese a que son eliminados los valores de los campos- se **conservan las posiciones** de los registros borrados, con lo cual **no se reduce el tamaño** de la tabla.

Esas *posiciones de registro* serán **utilizadas** por MySQL para **escribir** los registros que se vayan **añadiendo** después del proceso de borrado.

Para eliminar esos *registros vacíos* y **reducir** el tamaño de una tabla, MySQL dispone de una sentencia que es la siguiente:

**OPTIMIZE TABLE** *tabla*

Esta sentencia -que debe usarse después de un proceso de borrado **amplio**- **depura** la tabla eliminando los *registros inutilizados* por el proceso `DELETE`, con lo que logra una **reducción** del tamaño de la tabla a su *dimensión óptima*.

## Los arrays de la sentencia SELECT

Aunque están comentados en los *códigos fuente* de los scripts queremos reiterar aquí -aprovechando este espacio que la maquetación nos concede- para hacer algunas precisiones sobre los resultados de las consultas de tablas.

Se trata de los índices de los arrays que se obtienen mediante las funciones:

`mysql_fetch_array()`  
y  
`mysql_fetch_row()`

Los índices escalares, en ambos casos, cuanto tratan información obtenida mediante una sentencia `SELECT` coinciden con el orden en el que han sido establecidos los campos en esa instrucción concreta. De modo que el primer de esos nombres de campos sería asociado con el índice cero de estos array, el segundo con el índice 1 y así sucesivamente.

En el caso del array asociativo devuelto por la primera de estas funciones, los índices coinciden siempre con los nombres de los campos de los que han sido extraídos los datos.

En el caso de que la consulta afecte a varias tablas (recuerda que los campos se asignan poniendo `tabla.campo` (nombre de la tabla y nombre del campo) el índice del

## Guardar y recuperar bases de datos y/o tablas

Aunque es perfectamente factible desarrollar scripts propios que permitan guardar y recuperar tanto las estructuras como los datos de una tabla ó de la base de datos completa, mencionaremos aquí una de las posibilidades más cómodas de hacerlos.

PhpMyAdmin es una magnífica herramienta para hacer y recuperar copias de seguridad.

Si abrimos esta utilidad <http://localhost/phpmyadmin/> podremos ver los dos enlaces que ves en la imagen -SQL y Exportar- que permiten *importar* y *exportar* tanto estructuras como datos y estructuras.

Al pulsar sobre *Exportar* nos aparecerá una página como esta:

demo4  
demodat1  
demodat2  
demodat3  
ejemplo1

Seleccione todo / Deseleccione todo

SQL  
 LaTeX  
 CSV para datos de MS Excel  
 Datos CSV  
 XML

Estructura  
 Añadir 'drop table'  
 Añada el valor AUTO\_INCREMENT  
 Usar "backquotes" con tablas y nombre:  
Añada en los comentarios  
 Fechas de creación/actualización/revertir

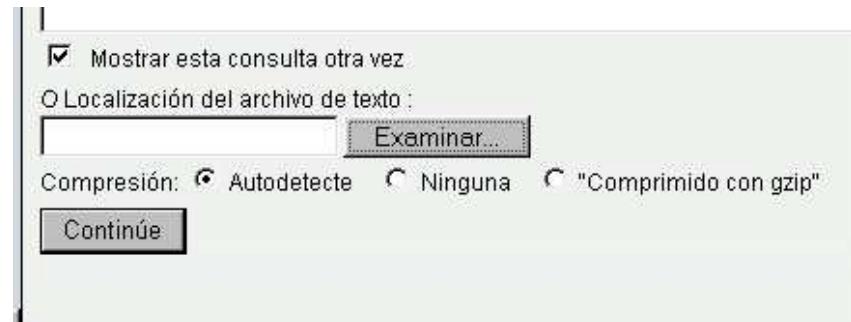
Datos  
 Completar los "Inserts"  
 "Inserts" extendidos  
 Use "inserts" con retraso  
Tipo de exportación: **INSERT**

Enviar (genera un archivo descargable)

Plantilla del nombre del archivo: **ejemplos**  recuerde la plantilla\*

donde podremos elegir una, varias o todas las tablas y que según las opciones elegidas nos permite exportar estructuras y/o datos, según las casillas de verificación que tengamos marcadas. Además nos permite elegir el formato en el que queremos guardar la copia -en nuestro caso elegiríamos SQL- y también según esté o no activada la casilla de verificación *Enviar* visualizar el fichero generado o guardarlo con el nombre que hayamos consignado en la caja de texto *Plantilla del nombre del archivo*.

array asociativo sería esa expresión con el punto incluido.



Para restaurar datos y/o estructuras desde un fichero de seguridad creado mediante el proceso anterior usaríamos la opción SQL de la primera imagen. A través de ella accederíamos a una página cuyo contenido estamos visualizando en esta última imagen.

Bastaría pulsar en examinar, buscar el fichero de seguridad y pulsar continúe. MySQL se encargaría de restaurar –en la base de datos a la que pertenezcan– todas las tablas contenidas en esa copia.

---

[Anterior](#)

[Índice](#)

[Siguiente](#)



## Peculiaridades de las tablas

Las tablas que han de contener imágenes deben tener campos del tipo **BLOB**, **MEDIUMBLOB** o **LONGBLOB**, pudiendo elegir aquel de ellos que más se aadecue al tamaño, en bytes, de las imágenes que se desean guardar en la tabla.

Por si te has olvidado de los tipos de campos, [aquí tienes un enlace para recordarlos](#).

En el ejemplo la hemos creado con un campo BLOB insertando también campos para recoger su nombre, su tamaño (en bytes), su formato (el tipo de fichero transferido) así como un campo autoincremental.

Desde este enlace -has de tener activo el servidor MySQL- podrás crear la tabla **fotos** e insertar automáticamente algunas imágenes de ejemplo.

**Crear tabla con IMAGENES**

## Transferencia de la imagen

El formulario para realizar la transferencia de la imagen no tiene particularidades. Es un formulario como los de *toda la vida*. Lo único reseñable sería incluir un campo oculto en el que pudiera especificarse una restricción en cuanto al tamaño máximo permitido para cada imagen y que debe estar acorde con el tipo de campo utilizado en la tabla.

## Comprobación del tipo de imagen

Al transferir imágenes **jpg** ó **png** el type MIME que recibía el servidor es distinto según el navegador que se utilice para hacer la transferencia.

Aquí a la derecha, en el código fuente del script que actualiza la base de datos, tienes los nombres de esos tipos asociados a los navegadores más usuales.

Hay otro aspecto a tener en cuenta. Esa discriminación de tipos se plantea únicamente cuando Apache recibe una transferencia. Cuando se visualiza un contenido las cabeceras tipo de contenido (`header("content-type: xx")`) pueden ser las mismas para todos los

## Creación de una tabla ejemplo

Lo primero de todo será disponer de una tabla en la que puedan guardarse imágenes. Aquí tienes un ejemplo.

```
<?
#el nombre de la tabla
$base="ejemplos";
#definimos otra variable con el NOMBRE QUE QUEREMOS DAR A LA TABLA
$tabla="fotos";
# establecemos la conexión con el servidor
$connexion=mysql_connect ("localhost","pepe","pepa");
#Seleccionamos la BASE DE DATOS en la que PRETENDEMOS CREAR LA TABLA
mysql_select_db ($base, $conexion);

$crear="CREATE TABLE IF NOT EXISTS $tabla (
";
$crear.="num_ident INT(10) unsigned NOT NULL AUTO_INCREMENT,
";
$crear.="imagen BLOB NOT NULL,
";
$crear.="nombre VARCHAR(255) NOT NULL DEFAULT '',
";
$crear.="tamano VARCHAR(15) NOT NULL DEFAULT '',
";
$crear.="formato VARCHAR(10) NOT NULL DEFAULT '',
";
$crear.="PRIMARY KEY (num_ident)
";

#Creamos la cadena, comprobamos si esa instrucción devuelve
# VERDADERO o FALSO
# y dependiendo de ellos insertamos el mensaje de éxito o fracaso

if(mysql_db_query ($base,$crear , $conexion)) {
echo "<h2> Tabla $tabla creada con EXITO </h2><br>";
} else{
echo "<h2> La tabla $tabla NO HA PODIDO CREARSE</h2><br>";
};

# cerramos la conexión... y listo...
mysql_close($conexion);
?>
```

## Formulario para la transferencia de las imágenes

```
<FORM ENCTYPE="multipart/form-data" ACTION="ejemplo211.php" METHOD="post">
#con este input "oculto" establecemos el límite máximo
# del tamaño del fichero a transferir. En este ejemplo 65.000 bytes
<INPUT type="hidden" name="lim_tamano" value="65000">
<p><b>Selecciona la imagen a transferir</b><br>
<INPUT type="file" name="foto"><br>
<p><b>Título la imagen</b><br>
<INPUT type="text" name="titulo"><br></p>
<p><INPUT type="submit" name="enviar" value="Aceptar"></p>
</FORM>
```

Ejemplo de transferencia de imagen

## Script para actualizar la base de datos

```
<?
$foto_name= $_FILES['foto']['name'];
$foto_size= $_FILES['foto']['size'];
$foto_type= $_FILES['foto']['type'];
?>
```

navegadores. Esa es la razón por la que a la hora de incluir el formato en la tabla utilizamos *image/jpg*, *image/gif* o *image/png*.

## ¿Cómo guardamos la imagen?

La información recibida a través del formulario requiere un *ligero retoque* antes de incluirla en el campo BLOB de la tabla. Esa reconversión requiere abrir la imagen en modo *binario (rb)* -parece que solo en el caso de Windows- leer el fichero completo y añadirle \ antes de las comillas mediante *addslashes*.

Una vez hecho el *retoque* ya puede guardarse sin más problema.

## PNG con transparencias en Internet Explorer

Internet Explorer no permite visualizar de forma automática las transparencias de las imágenes con formato **png**. Existen en la red algunos recursos que permiten solventar ese problema.

Hemos elegido uno de ellos -[pngfix.js](#)- que puedes ver en [este enlace](#).

Se trata de un fichero *JavaScript* que basta incluir en la cabecera HMTL de la página de la forma que ves en el ejemplo de la parte derecha. Cuando un navegador IE es detectado se ejecuta una función contenida en ese fichero que analiza la página, busca imágenes con extensión **png** y les aplica la transparencia adecuada.

Por esa razón, es probable que inicialmente (al cargar la página) se visualice la imagen opaca y que, posteriormente, adquiera la transparencia.

## Ver las imágenes

La lectura de una imagen utiliza solo dos instrucciones. Incluir la cabecera *Header* en el que se indica el tipo de contenido (el famoso nombre MIME de la imagen) y luego imprimir el contenido del campo.

Pero (por aquello de que *header* debe ir incluida en el script antes que cualquier otra salida) si pretendemos incluir en una página algo más que una imagen tendremos que invocar esas dos funciones, de forma independiente, para cada una de ellas.

Por esa razón, en el ejemplo que tienes al margen, al desarrollar el ejemplo que permite visualizar todas las imágenes de la tabla hemos tenido que incluir un parántesis que va

```
$foto_temporal= $_FILES['foto']['tmp_name'];
$lim_tamano= $_POST['lim_tamano'];
$foto_titulo= $_POST['titulo'];
/* limitamos los formatos de imagen admitidos a:
   png que segun del navegador que utilicemos puede ser:
   en IE image/x-png en Firefox y Mozilla image/png
   jpg que puede tener como tipo
   en IE image/pjpeg en Firefox y Mozilla image/jpeg
   gif que tiene como tipo image/gif en todos los navegadores
   Mira los comentarios al margen sobre la variable $extensión */
if ($foto_type=="image/x-png" OR $foto_type=="image/png"){
$extension="image/png";
}
if ($foto_type=="image/pjpeg" OR $foto_type=="image/jpeg"){
$extension="image/jpeg";
}
if ($foto_type=="image/gif" OR $foto_type=="image/gif"){
$extension="image/gif";
}
# condicionamos la inserción a que la foto tenga nombre,
# un tamaño distinto de cero y menor de límite establecido
# en el formulario y que la variable extensión sea no nula

if ($foto_name != "" AND $foto_size != 0
    AND $foto_titulo != '' AND
    $foto_size<=$lim_tamano AND $extension != ''){
/*reconversion de la imagen para meter en la tabla
abrimos el fichero temporal en modo
lectura "r" binaria "b"*/
$f1= fopen($foto_temporal,"rb");
#leemos el fichero completo limitando
# la lectura al tamaño de fichero
$foto_reconvertida = fread($f1, $foto_size);
#anteponemos \ a las comillas que pudiera contener el fichero
# para evitar que sean interpretadas como final de cadena
$foto_reconvertida=addslashes($foto_reconvertida);
# abrimos la base de datos y escribimos las intrucciones de inserción
# en el campo BLOB insertaremos la foto_reconvertida
$base="ejemplos";
$tabla="fotos";
$conexion=mysql_connect ("localhost","pepe","pepa");
mysql_select_db ($base, $conexion);
$meter="INSERT INTO ".$tabla;
$meter .= (num_ident, imagen, nombre, tamano, formato) ";
$meter .= VALUES('','$foto_reconvertida','$foto_titulo',";
$meter .= "$foto_size, '$extension')";
if (@mysql_query($meter,$conexion)){
print "Foto guardada en la tabla";
} else{
print "Ha habido un error al guardar la foto";
}
} else{
echo "<h2>No ha podido transferirse el fichero</h2>";
}
mysql_close();
?>
```

## Script para leer la base de datos

```
<html>
<head>
<!-- al margen te comentamos la razón por la que --&gt;
<!-- se incluyen estas líneas en rojo --&gt;
&lt;!--[if IE ]&gt;
&lt;script type="text/javascript" src="pngfix.js"&gt;&lt;/script&gt;
&lt;![endif]--&gt;
&lt;/head&gt;
&lt;body&gt;
&lt;?
$base="ejemplos";
$tabla="fotos";
$conexion=mysql_connect ("localhost","pepe","pepa");
mysql_select_db ($base, $conexion);
$sacar ="SELECT * FROM ".$tabla;
$resultado = mysql_query($sacar,$conexion);
while ($registro = mysql_fetch_array($resultado)){
print "&lt;center&gt;Título de la imagen: ".$registro['nombre']."&lt;br&gt;";</pre>

```

leyendo la tabla que contiene las imágenes para extraer los campos informativos y a la hora de ver la *imagen* hemos de recurrir a la misma técnica que se utilizaba para ver las imágenes dinámicas.

Es decir, poner una etiqueta de imagen de las de HTML pero -en vez de escribir el nombre de la imagen-poniendo incluyendo como nombre el del script que las visualiza y pasándole el *número* (valor del campo autoincremental) de la imagen que pretendí visualizar.

## El problema de los PNG en IE

El JavaScript que asigna la transparencia a las imágenes en formato png las identifica buscando la coincidencia de los tres últimos caracteres del nombre de la imagen con la extensión **png**.

Cuando se trata de imágenes dinámicas el nombre de la imagen coinciden con el nombre de la llamada al script que se utiliza para su visualización. Por eso, para advertir a JavaScript de que se trata de una imagen **png** hemos incluido el condicional que puedes ver en el ejemplo. De esa forma, cuando se trata de una imagen en este formato incluimos en la petición una variable con el valor **png** de forma que pueda ser reconocida por **pngfix.js** y aplicada la transparencia requerida.

```
/* la inclusión de este condicional obedece a los problemas que plantea la visualización de las transparencias de las imágenes png en Internet Explorer.  
Al margen justificamos las razones de su inclusión */  
if($registro['formato']=="image/png") {  
print "<img src='ver_foto.php?n=".$registro['num_ident']."'&v=png'><br>";  
} else {  
print "<img src='ver_foto.php?n=".$registro['num_ident']."'><br>";  
}  
print "Tamaño de la imagen: ".$registro['tamano']." bytes  
</center>";  
  
}  
mysql_close();  
?>  
</body>  
</html>
```

## Script para leer imágenes de la base datos

```
<?  
$numero=$_REQUEST['n'];  
$base="ejemplos";  
$tabla="fotos";  
$conexion=mysql_connect ("localhost","pepe","pepa");  
mysql_select_db ($base, $conexion);  
$sacar = "SELECT * FROM ".$tabla." WHERE (num_ident=$numero) " ;  
$resultado = mysql_query($sacar,$conexion);  
while ($registro = mysql_fetch_array($resultado)) {  
    $tipo_foto=$registro['formato'];  
    header("Content-type: $tipo_foto");  
    echo $registro['imagen'];  
}  
mysql_close();  
?>
```

[Ver imágenes guardadas](#)

Anterior



Índice



Siguiente





## Tipos de tablas

Aunque en los temas anteriores no hemos hecho alusión a ello, MySQL permite usar diferentes tipos de tablas tales como:

ISAM  
MyISAM  
InnoDB

Las tablas **ISAM** son las de formato más antiguo. Están limitadas a tamaños que no superen los 4 gigas y no permite copiar tablas entre máquinas con distinto sistema operativo.

Las tablas **MySAM** son el resultado de la evolución de las anteriores, ya que resuelven el problema que planteaban las anteriores y son el **formato por defecto** de MySQL a partir de su versión 3.23.

Las tablas del tipo **InnoDB** tienen una estructura distinta que MyISAM, ya que utilizan **un sólo archivo** por tabla en vez de los tres habituales en los tipos anteriores.

Incorporan un par de ventajas importantes, ya que permiten **realizar transacciones** y definir reglas de **integridad referencial**.

## Creación y uso de tablas InnoDB

La creación de tablas de este tipo no presenta ninguna dificultad añadida. El proceso es idéntico a las tablas habituales sin más que añadir **Type=InnoDB** después de cerrar el paréntesis de la sentencia de creación de la tabla.

Una vez creadas, las tablas InnoDB se comportan –a efectos de uso– exactamente igual que las que hemos venido utilizando en las páginas anteriores. No es preciso hacer ningún tipo de modificación en la sintaxis. Por tanto, es totalmente válido todo lo ya comentado respecto a: altas, modificaciones, consultas y bajas.

## Las transacciones

Uno de los riesgos que se plantean en la gestión de bases de datos es que pueda producirse una interrupción del proceso mientras se está actualizando una o varias tablas. Pongamos como ejemplo el cobro de nuestra nómina. Son necesarias dos anotaciones simultáneas: **El cargo en la cuenta del Organismo pagador y el abono en nuestra cuenta bancaria**.

## Creación de una tabla InnoDB

La creación de tablas tipo InnoDB requiere una de estas dos sentencias:

**CREATE TABLE IF NOT EXISTS tabla (campo1, campo2,... ) Type=InnoDB**

**CREATE TABLE tabla (campo1, campo2,... ) Type=InnoDB**

Este script crear una tabla *InnoDB* con idénticos campos a los utilizados en el caso de la tabla **demo4** con la que hemos venido trabajando hasta ahora. La sintaxis, muy similar a la utilizada allí es esta:

```
<?
$base="ejemplos";
$tabla="demoINNO";
$c=mysql_connect ("localhost","pepe","pepa");
mysql_select_db ($base, $c);

$crear="CREATE TABLE $tabla (
$crear.="Contador TINYINT(8) UNSIGNED ZEROFILL NOT NULL AUTO_INCREMENT,";
$crear.="DNI CHAR(8) NOT NULL, ";
$crear.= "Nombre VARCHAR (20) NOT NULL, ";
$crear.= "Apellido1 VARCHAR (15) not null, ";
$crear.= "Apellido2 VARCHAR (15) not null, ";
$crear.= "Nacimiento DATE DEFAULT '1970-12-21', ";
$crear.= "Hora TIME DEFAULT '00:00:00', ";
$crear.= "Sexo Enum('M','F') DEFAULT 'M' not null, ";
$crear.= "Fumador CHAR(0) , ";
$crear.= "Idiomas SET(' Castellano',' Francés','Inglés',
      ' Alemán',' Búlgaro',' Chino'), ";
$crear.= " PRIMARY KEY(DNI), ";
$crear.= " UNIQUE auto (Contador)";
$crear.=")";
# esta es la única diferencia con el proceso de
# creación de tablas MyISAM
$crear.= " Type=InnoDB";

if(mysql_query ($crear ,$c)) {
echo "<h2> Tabla $tabla creada con EXITO </h2><br>";
} else{
echo "<h2> La tabla $tabla NO HA PODIDO CREARSE ";
# echo mysql_error ($c).<br>;
$numerror=mysql_errno ($c);
if ($numerror==1050){echo "porque YA EXISTE</h2>";}
};
mysql_close($c);
?>
```

Crear tabla tipo  
InnoDB

## ¡Cuidado!

Bajo Windows, al crear una base de datos o tabla InnoDB el nombre de la misma aparecerá en minúsculas independientemente de la sintaxis que hayamos utilizado en su creación.

Si observas el ejemplo anterior, hemos puesto **demoINNO** como nombre de la tabla. Sin embargo, si miras el directorio c:\mysql verás que aparece el fichero **demoinno.frm** con minúsculas.

proceso en el intermedio de las dos operaciones podría darse la circunstancia de que apareciera registrado el pago sin que se llegaran a anotar los haberes en nuestra cuenta.

Las transacciones evitan este tipo de situaciones ya que los registros de los datos se registran de manera provisional y no toman carácter definitivo hasta que una instrucción confirme que esas anotaciones tienen carácter definitivo. Para ello, MySQL dispone de tres sentencias: **BEGIN**, **COMMIT** y **ROLLBACK**.

## Sintaxis de las transacciones

Existen tres sentencias para gestionar las transacciones. Son las siguientes:

**mysql\_query("BEGIN",\$c)**

Su ejecución requiere que este activa la conexión **\$c** con el servidor de base de datos e indica a MySQL que **comienza una transacción**.

Todas las sentencias que se ejecuten a partir de ella tendrán carácter **provisional** y no se ejecutarán de forma efectiva hasta que encuentre una sentencia de finalización.

**mysql\_query("ROLLBACK",\$c)**

Mediante esta sentencia **advertimos** a MySQL que **finaliza la transacción** pero que **no debe hacerse efectiva** ninguna de las modificaciones incluidas en ella.

**mysql\_query("COMMIT",\$c)**

Esta sentencia **advierte** a MySQL que ha finalizado la transacción y que **debe hacer efectivos** todos los cambios incluidos en ella.

## Precauciones a tener en cuenta

Cuando se utilizan campos autoincrementales en tablas InnoDB los contadores se van incrementando al añadir registros (incluso de forma provisional) con lo cual si se **aborta** la inclusión con un **ROLLBACK** ese contador mantiene el incremento y en inserciones posteriores partirá de ese valor acumulado.

Por ejemplo. Si partimos de una tabla vacía y hacemos una transacción de dos registros (número 1 y número 2 en el campo autoincremental) y la finalizamos con **ROLLBACK**, no se insertarán pero en una inserción posterior el contador autoincremental comenzará a partir del valor 2.

MySQL anuncia que a partir de la versión 5.0.3 se incluirá una nueva sentencia para permitir que se **renumerar** los campos

## Las primeras transacciones

```
<?
$base="ejemplos";
# escribimos el nombre de la tabla en MINUSCULAS
# para asegurar la compatibilidad entre plataformas
$tabla="demoInno";
$conexion=mysql_connect("localhost","pepe","pepa");
mysql_select_db ($base, $conexion);
# insertamos la sentencia BEGIN para indicar el comienzo
# de una transacción
mysql_query("BEGIN",$conexion);
/* hasta que no aparezca una sentencia que finalice la transacción
(ROLLBACK ó COMMIT) las modificaciones en la tabla serán registradas
de forma "provisional") */
mysql_query("INSERT $tabla (DNI,Nombre,Apellido1,Apellido2,
Nacimiento,Sexo,Hora,Fumador,Idiomas)
VALUES
('111111','Alvaro','Alonso','Azcárate','1954-11-23',
'M','16:24:52',NULL,3)",$conexion);
if (mysql_errno($conexion)==0){
    echo "Registro AÑADIDO<br>";
} else{
    if (mysql_errno($conexion)==1062){
        echo "No ha podido añadirse el registro <br>";
        echo "Ya existe un registro con este DNI<br>";
    } else{
        $numerror=mysql_errno($conexion);
        $descrierror=mysql_error($conexion);
        echo "Se ha producido un error nº $numerror<br>";
        echo "<br>que corresponde a: $descrierror<br>";
    }
}
# indicamos el final de la transacción, en este caso con ROLLBACK
# por lo tanto el registro con DNI 111111 no será insertado en la tabla
mysql_query("ROLLBACK",$conexion);
# incluyamos una nueva transacción
mysql_query("BEGIN",$conexion);
mysql_query("INSERT $tabla (DNI,Nombre,Apellido1,Apellido2,
Nacimiento,Sexo,Hora,Fumador,Idiomas)
VALUES
('222222','Genoveva','Zarabozo','Zitrón','1964-01-14',
'F','16:18:20',NULL,2)",$conexion);
if (mysql_errno($conexion)==0){
    echo "Registro AÑADIDO";
} else{
    if (mysql_errno($conexion)==1062){
        echo "No ha podido añadirse el registro <br>";
        echo "Ya existe un registro con este DNI";
    } else{
        $numerror=mysql_errno($conexion);
        $descrierror=mysql_error($conexion);
        echo "Se ha producido un error nº $numerror";
        echo "<br>que corresponde a: $descrierror";
    }
}
# indicamos el final de la transacción, en este caso con COMMIT
# por lo tanto el registro con DNI 222222 si será insertado en la tabla
mysql_query("COMMIT",$conexion);
# leamos el contenido de la tabla para ver el resultado
$resultado= mysql_query("SELECT * FROM $tabla ",$conexion);
print "<br>Lectura de la tabla depués del commit<br>";
while ($registro = mysql_fetch_row($resultado)){
    foreach($registro as $clave){
        echo $clave,"<br>";
    }
}
mysql_close();
?>
```

Ejecutar el ejemplo

## Integridad referencial en tablas InnoDB

autoincrementales.

## Elementos necesarios para la integridad referencial

La integridad referencial ha de establecerse siempre entre dos tablas. Una de ellas ha de comportarse como **tabla principal** (suele llamarse *tabla padre*) y la otra sería la **tabla vinculada** ó *tabla hijo*.

Es imprescindible:

Que la **tabla principal** tenga un índice primario (PRIMARY KEY)  
Que la **tabla vinculada** tenga un índice (no es necesario que sea ni único ni primario) asociado a campos de tipo idéntico a los que se usen para índice de la tabla principal.

Si observas el código fuente del ejemplo que tienes a la derecha podrás observar que utilizamos el número del DNI (único para alumno) como elemento de vinculación de la tabla de datos personales con la que incluye las direcciones.

## Borrado de tablas vinculadas

Si pretendemos eliminar una tabla principal recibiremos un mensaje de error tal como puedes ver si ejecutas este ejemplo cuyo código fuente tienes aquí. cómo es lógico, antes de ejecutarlo habrás de tener creada la tabla cuyo código fuente tienes a la derecha.

Las tablas vinculadas si permiten el borrado y una vez que éstas ya han sido eliminadas (o quitada la vinculación) ya podrán borrarse sin problemas las tablas principales. Si ejecutas este ejemplo podrás observar que borramos ambas tablas siguiendo el orden que permite hacerlo. Primero se borra la vinculada y luego la principal. Este es el código fuente del script.

### ¡Cuidado!

Si has borrado las tablas con los ejemplos anteriores no olvides crearlas de nuevo para poder visualizar los ejemplos siguientes.

## Modificación o borrado de campos vinculados

Las sentencias MySQL que deban modificar o eliminar campos utilizados para establecer vínculos entre tablas requieren de un parámetro especial (CONSTRAINT) -puede ser distinto en cada una de las tablas- que es necesario conocer previamente.

La forma de visualizarlo es ejecutar

Cuando se trabaja con varias tablas que tienen algún tipo de vínculo resulta interesante disponer de mecanismos que protejan o impidan acciones no deseadas. Supongamos, como veremos en los ejemplos posteriores que pretendemos utilizar una tabla con datos de alumnos y otra tabla distinta para las calificaciones de esos alumnos. Si no tomamos ninguna precaución (bien sea mediante los script o mediante el diseño de las tablas) podría darse la circunstancia de que incluyéramos calificaciones a alumnos inexistentes, en materias de las que no están matriculados, etcétera. También podría darse la circunstancia de que diéramos de baja a un alumno pero que se mantuvieran las calificaciones vinculadas a él. Todas estas circunstancias suelen producir efectos indeseados y las tablas InnoDB pueden ser diseñadas para prever este tipo de situaciones.

## Sintaxis para la vinculación de tablas

Los vínculos entre tablas suelen establecer en el momento de la creación de la *tabla vinculada*.

**CREATE TABLE tabla (campo1, campo2,...  
KEY nombre (campo de vinculacion ),  
FOREIGN KEY (campo de vinculacion )  
REFERENCES nombre\_de\_la tabla principal (Indice primario de la tabla principal)  
) Type=InnoDB**

donde el *campo de vinculacion* ha de ser un índice (no es necesario que sea PRIMARY KEY ni UNIQUE) y donde *Indice primario de la tabla principal* ha de ser un índice primario (PRIMARY KEY) de la tabla principal. Debe haber plena coincidencia (tanto en tipos como contenidos) entre ambos índices.

```
<?
$base="ejemplos";
$tabla1="principal";
$tabla2="vinculada";
$c=mysql_connect ("localhost","pepe","pepa");
mysql_select_db ($base, $c);
# creación de la tabla principal type InnoDB
$crear="CREATE TABLE IF NOT EXISTS $tabla1 ";
$crear.="DNI CHAR(8) NOT NULL, ";
$crear.="Nombre VARCHAR (20) NOT NULL, ";
$crear.="Apellido1 VARCHAR (15) not null, ";
$crear.="Apellido2 VARCHAR (15) not null, ";
# el indice primario es imprescindible. Recuerda que debe
# estar definido sobre campos NO NULOS
$crear.=" PRIMARY KEY(DNI) ";
$crear.="";
$crear.=" Type=InnoDB";
# creamos la tabla principal comprobando el resultado
if(@mysql_query ($crear ,$c)){
    print "La tabla ".$tabla1." ha sido creada<br>";
}else{
    print "No se ha creado ".$tabla1." ha habido un error<br>";
}
# crearemos la tabla vinculada
$crear="CREATE TABLE IF NOT EXISTS $tabla2 ";
$crear.="IDENTIDAD CHAR(8) NOT NULL, ";
$crear.="calle VARCHAR (20), ";
$crear.="poblacion VARCHAR (20), ";
$crear.="distrito VARCHAR(5), ";
# creamos el índice (lo llamamos asociador) para la vinculación
# en este caso no será ni primario ni único
# Observa que el campo IDENTIDAD de esta tabla CHAR(8)
# es idéntico al campo DNI de la tabla principal
$crear.=" KEY asociador(IDENTIDAD) , ";
#establecemos la vinculación de ambos índices
$crear.=" FOREIGN KEY (IDENTIDAD) REFERENCES $tabla1(DNI) ";
$crear.=") TYPE = INNODB";
# creamos (y comprobamos la creación) la tabla vinculada
if(@mysql_query ($crear ,$c)){
    print "La tabla ".$tabla2." ha sido creada<br>";
}else{
    print "No se ha creado ".$tabla2." ha habido un error<br>";
}
mysql_close();
?>
```

Crear tablas vinculadas

## Modificación de estructuras

**TABLE** *nombre tabla* que devuelve como resultado un array asociativo con dos índices. Uno de ellos -llamado *Table*- que contiene el nombre de la tabla y el otro -Create Table- que contiene la estructura con la que ha sido creada la tabla pero incluyendo el parámetro **CONSTRAINT** seguido de su valor. Ese valor es precisamente el que necesitamos para hacer modificaciones en los campos asociados de las tablas vinculadas.

Pulsando en este enlace cuyo código fuente tienes aquí podrás visualizar el resultado de la ejecución de esa sentencia.

Conocido el valor de parámetro anterior el proceso de **borrado** del vínculo actual requiere la siguiente sintaxis:

**ALTER TABLE** *nombre de la tabla*  
**DROP FOREIGN KEY** *parametro*

Cuando se trata de añadir un nuevo vínculo con una tabla principal habremos de utilizar la siguiente sentencia:

**ALTER TABLE** *nombre de la tabla*  
**ADD [CONSTRAINT** *parametro*]  
**FOREIGN KEY** *parametro*  
**REFERENCES** *tabla principal(clave primaria)*

El parámetro **CONSTRAINT** (encerrado en corchetes en el párrafo anterior) es **OPCIONAL** y solo habrá de utilizarse en el caso de que existiera ya una vinculación previa de esa tabla.

### La función preg\_match

En el ejemplo de la derecha utilizamos esta función cuya sintaxis es la siguiente:

**preg\_match(** *pat, cad, coin* **)**

donde **pat** es un patrón de búsqueda, **cad** es la cadena que la han de realizarse las búsquedas y **coin** es un arreglo que recoge todas las coincidencias encontradas en la cadena.

El patrón de búsqueda que hemos utilizado en el ejemplo **/CONSTRAINT.\*FOREIGN KEY/** debe interpretarse de la siguiente forma. Los caracteres / indican el comienzo y el final del patrón, el . indica que entre **CONSTRAINT** y **FOREIGN** se permite cualquier carácter para admitir la coincidencia y, además, \* indica que ese carácter cualquier puede repetirse cero o más veces.

#### ¡Cuidado!

Es posible que el script de la derecha te de un error como consecuencia

La modificación de estructuras en tablas vinculadas puede hacerse de forma idéntica a la estudiada para los casos generales de MySQL siempre que **esas modificaciones no afecten a los campos mediante los que se establecen las vinculaciones entre tablas**.

Aquí tienes un ejemplo en se borran y añaden campos en ambas tablas. Como puedes ver la sintaxis es exactamente la misma que utilizamos en temas anteriores.

```
<?
$base="ejemplos";
$stabla="principal";
$stabla1="vinculada";
$c=mysql_connect ("localhost","pepe","pepa");
mysql_select_db ($base, $c);

if(mysql_query("ALTER TABLE $stabla ADD Segundo_Apellido VARCHAR(40)",$c)){
    print "Se ha creado el nuevo campo en ".$stabla."<br>";
}
if(mysql_query("ALTER TABLE $stabla DROP Apellido2",$c)){
    print "Se ha borrado el campo Apellido 2 en ".$stabla."<br>";
}

if(mysql_query("ALTER TABLE $stabla1 ADD DP VARCHAR(5)",$c)){
    print "Se ha creado el nuevo campo en ".$stabla1."<br>";
}
if(mysql_query("ALTER TABLE $stabla1 DROP distrito",$c)){
    print "Se ha borrado el campo distrito en ".$stabla1."<br>";
}

mysql_close();
?>
```

Ejecutar el ejemplo

En este otro ejemplo determinaremos el valor de **CONSTRAINT** y modificaremos campos asociados de la tabla vinculada.

```
<?
$base="ejemplos";
$stabla="vinculada";
$stabla1="principal";
$c=mysql_connect ("localhost","pepe","pepa");
mysql_select_db ($base, $c);
$resultado=mysql_query( "SHOW CREATE TABLE $stabla",$c);
$sv=mysql_fetch_array($resultado);
# extreamos de Create Table la cadena que empiza por CONSTRAINT
# y que acaba por FOREIGN KEY lo guardamos en el array $coin
preg_match ('/CONSTRAINT.*FOREIGN KEY/', $sv['Create Table'], $coin);
# extraemos el parametro quitando el CONSTRAINT que lleva delante
# y el FOREIGN KEY que lleva al final
$para=str_replace("FOREIGN KEY",'',str_replace("CONSTRAINT",'',$coin[0]));
print "El valor de CONSTRAINT es: ".$para."<br>";
# eliminamos el vínculo con la clave externa incluyendo en la sentencia
# el valor del parametro obtenido el proceso anterior
if(mysql_query("ALTER TABLE $stabla DROP FOREIGN KEY $para",$c)){
    print "Se ha realizado con éxito el borrado del vínculo<br>";
}
# añadimos el nuevo vínculo (en este caso rehacemos el anterior
# pero el proceso es idéntico)
if(mysql_query("ALTER TABLE $stabla ADD CONSTRAINT $para
FOREIGN KEY(IDENTIDAD) REFERENCES $stabla1(DNI) ",$c)){
    print "Se ha reestablecido con éxito vínculo<br>";
}

mysql_close();
?>
```

Ejecutar el ejemplo

### Añadir registros a la tabla vinculada

Si ejecutas este ejemplo habiendo seguido la secuencia de estos materiales verás que se produce un error nº **1216**. Es lógico que así sea porque estamos intentando añadir un registro a la tabla

de que hemos podido modificar campos en los ejemplos anteriores. Si eso ocurre, borra aquí las tablas y generales de nuevo pulsando aquí.

### ¡Cuidado!

Los resultados que obtengas al ejecutar los ejemplos de borrado y modificación de datos pueden arrojar resultados distintos según los contenidos de las tablas que, a su vez, serán consecuencia de los ejemplos que hayas ejecutado anteriormente y de la secuencia de los mismos. Siempre puedes volver a las condiciones iniciales de los enlaces de la advertencia anterior.

### Opciones adicionales de FOREIGN KEY

La cláusula FOREIGN KEY permite añadirte -detrás de la definición ya comentada y sin poner coma separándola de ella- los parámetros ON DELETE y ON UPDATE en las que se permite especificar una de las siguientes opciones:

#### ON DELETE RESTRICT

Esta condición (es la condición por defecto de MySQL y no es preciso escribirla) indica a MySQL que **interrumpa el proceso de borrado** y de un mensaje de error cuando se intente borrar un registro de la tabla principal cuando en la tabla vinculada existan registros asociados al valor que se pretende borrar.

#### ON DELETE NO ACTION

Es un sinónimo de la anterior.

#### ON DELETE CASCADE

Cuando se especifica esta opción, al borrar un registro de la tabla principal se borrarán de forma automática todos los de la tabla vinculada que estuvieran asociados al valor de la clave foránea que se trata de borrar. Con ello se conseguiría una actualización automática de la segunda tabla y se mantendría la identidad referencial.

#### ON DELETE SET NULL

Con esta opción, al borrar el registro de la tabla principal **no se borrarían** los que tuviera asociados la tabla secundaria pero tomarían valor **NULL** todos los índices de ella

vinculada y ello requeriría que en el campo **DNI de la tabla principal** existiera un registro con un valor igual al que pretendemos introducir en la tabla vinculada.

[Ver script](#)

[Insertar en tabla vinculada](#)

Añadiremos un registro a la tabla principal con el DNI anterior:

[Ver script](#)

[Insertar en tabla principal](#)

y ahora ya podremos ejecutar -sin errores- el script que inserta datos en la tabla vinculada. Podremos ejecutar aquel script tantas veces como queramos ya que -el campo IDENTIDAD está definido como KEY y por tanto permite duplicados- no hemos establecido la condición de índice PRIMARIO ó UNICO.

### Borrar o modificar registros en la tabla principal

Sin intentamos borrar un registro de la tabla principal mediante un script como el que tienes en este ejemplo verás que se produce un error nº **1217** advirtiéndonos de que no se realiza el borrado porque existen registros en la tabla vinculada con valores asociados al índice del campo que pretendemos borrar y, de permitir hacerlo, se rompería la integridad referencial ya que quedarían registros *huérfanos* en la tabla vinculada.

[Ver script](#)

[Borrar en tabla principal](#)

Sin tratamos de modificar un registro de la tabla principal y la modificación afecta al índice que realiza la asociación con la tabla (o tablas) vinculadas se produciría -por las mismas razones y en las mismas circunstancias- un error nº **1217** que impediría la modificación.

[Ver script](#)

[Modificar DNI en tabla principal](#)

Si -al tratar de borrar o modificar un registro de la tabla principal- no existieran en la tabla (o tablas vinculadas) registros asociados con él, el proceso de modificación se realizaría sin ningún problema y sin dar ningún mensaje de error o advertencia.

### Automatización de procesos

Creamos dos tablas idénticas a las anteriores incluyendo algunos datos en ellas.

```
<?
$base="ejemplos";
$tabla1="principal1";
$tabla2="vinculada1";
$c=mysql_connect ("localhost","pepe","pepa");
mysql_select_db ($base, $c);
# creación de la tabla principal type InnoDB
$crear="CREATE TABLE IF NOT EXISTS $tabla1 (
$crear.="DNI CHAR(8) NOT NULL, ";
$crear.="Nombre VARCHAR (20) NOT NULL, ";
$crear.="Apellido1 VARCHAR (15) not null, ";
$crear.="Apellido2 VARCHAR (15) not null, ";
$crear.= " PRIMARY KEY(DNI) ";
$crear.=")";
$crear.= " Type=InnoDB";
# creamos la tabla principal comprobando el resultado
if(@mysql_query ($crear , $c)){
    print "La tabla ".$tabla1." ha sido creada<br>";
} else{
    print "No se ha creado ".$tabla1." ha habido un error<br>";
}
# crearemos la tabla vinculada
$crear="CREATE TABLE IF NOT EXISTS $tabla2 (
$crear.="IDENTIDAD CHAR(8) NOT NULL, ";
$crear.="calle VARCHAR (20), ";
$crear.="poblacion VARCHAR (20), ";
$crear.="distrito VARCHAR(5), ";
#creamos la tabla vinculada las opciones de DELETE y UPDATE
$crear.= " KEY asociador(IDENTIDAD), ";
#establecemos la vinculación de ambos índices
$crear.= " FOREIGN KEY (IDENTIDAD) REFERENCES $tabla1(DNI) ";
$crear.= " ON DELETE CASCADE ";
$crear.= " ON UPDATE CASCADE ";
$crear.=") TYPE = INNODB";
# creamos (y comprobamos la creación) la tabla vinculada
if(@mysql_query ($crear , $c)) {
```

coincidentes con la clave primaria de la tabla principal.

Para el caso de ON UPDATE las opciones son estas:

**ON UPDATE RESTRICT**  
**ON UPDATE CASCADE**  
**ON UPDATE SET NULL**

Su comportamiento es idéntico a sus homónimas del caso anterior.

#### ¡Cuidado!

El uso de la opción SET NULL requiere que el campo indicado en FOREIGN KEY esté permita valores nulos. Si está definido con flag NOT NULL (como ocurre en los ejemplos que tienes al margen) daría un mensaje de error.

#### ¡Cuidado!

Al incluir ON DELETE y ON UPDATE (si se incluyen ambas) han de hacerse por este mismo orden.

Si se cambiara este orden daría un mensaje de error y no se ejecutarían.

```
        print "La tabla ".$tabla2." ha sido creada<br>";
} else{
        print "No se ha creado ".$tabla2." ha habido un error<br>";
}
# añadimos registros a la tabla principal
mysql_query("INSERT $stabla1 (DNI,Nombre,Apellido1,Apellido2)
            VALUES ('111111','Robustiano','Iglesias','Pérez')",$c);
mysql_query("INSERT $stabla1 (DNI,Nombre,Apellido1,Apellido2)
            VALUES ('222222','Ambrosio','Morales','Gómez')",$c);
# añadimos registros a la tabla vinculada1
mysql_query("INSERT $stabla2 (IDENTIDAD,calle,poblacion,distrito)
            VALUES ('111111','Calle Asturias,3','Oviedo','33001')",$c);
mysql_query("INSERT $stabla2 (IDENTIDAD,calle,poblacion,distrito)
            VALUES ('111111','Calle Palencia,3','Logroño','78541')",$c);
mysql_query("INSERT $stabla2 (IDENTIDAD,calle,poblacion,distrito)
            VALUES ('222222','Calle Anunciación,3','Algeciras','21541')",$c);
mysql_close();
?>
```

[Crear tablas y datos](#)

[Ver contenidos de tablas](#)

[Ver código fuente](#)

## Modificar registros en cascada

```
<?
$base="ejemplos";
$tabla="principal1";
$conexion=mysql_connect("localhost","pepe","pepa");
mysql_select_db($base,$conexion);
# modificamos un registro
mysql_query("UPDATE $tabla SET DNI='123456' WHERE DNI='111111'",$conexion);
# borramos un registro
mysql_query("DELETE FROM $tabla WHERE (DNI='222222')",$conexion);
if (mysql_errno($conexion)==0){echo "<h2>Tablas actualizadas</h2></H2>";
} else{
        print "Ha habido un error al actualizar";
}
mysql_close();
?>
```

[Actualizar en cascada](#)

[Ver contenido de la tabla](#)

Para que puedas retornar a las condiciones iniciales, desde este enlace podrás borrar las tablas creadas para actualización en cascada. De esta forma podrás volver a crearlas, cuando deseas, en las condiciones iniciales.

[Anterior](#)

[Índice](#)

[Siguiente](#)



Ver índice

## Otras consultas



### Definición de tablas

En este ejemplo de tablas vinculadas con integridad relacional vamos a proponer la situación siguiente.

Crearemos una primera tabla (**alumnos**) que va a contener datos personales de un grupo de personas.

Para evitar duplicar un mismo alumno y alumnos sin DNI, vamos a utilizar como índice primario (PRIMARY KEY) el campo DNI (que es único para cada persona).

La condición de que el campo DNI sea PRIMARY KEY nos obliga a definirlo con el *flag* NOT NULL, dado que esta es una condición necesaria para la definición de índices primarios.

Crearemos una segunda tabla (**domicilios**) con los domicilios de cada uno de los alumnos, identificándolos también por su DNI. Para evitar que un mismo alumno pueda tener dos domicilios asignamos a al campo DNI de esta tabla la condición de índice único (UNIQUE).

El índice UNIQUE podríamos haberlo creado también como PRIMARY KEY ya que la única diferencia de comportamiento entre ambos es el hecho que aquel admitiría valores NULOS pero, dado que debemos evitar *domicilios sin alumnos*, insertaremos el *flag* NOT NULL en este campo.

El hecho de utilizar dos tablas no tiene otro sentido que la ejemplificación ya que lo habitual sería que todos los datos estuvieran en una misma tabla.

Vincularemos ambas tablas de modo que no puedan crearse direcciones de alumnos inexistentes y les pondremos la opción de *actualización y borrado* en cascada. De esta forma las acciones en la tabla de alumno se reflejarán automáticamente en la tabla de domicilios.

La tercera de las tablas (**evaluaciones**) tiene como finalidad definir las distintas evaluaciones que puede realizarse a cada alumno. Tendrá dos campos. Uno descriptivo (el nombre de la evaluación) y otro identificativo (no nulo y único) que será tratado como PRIMARY KEY para evitar duplicidades y porque, además, va a ser utilizado como

### Estructura de tablas y relaciones

Para desarrollar los ejemplos de este capítulo vamos a crear las tablas, cuyas estructuras e interrelaciones que puedes ver en el código fuente siguiente:

```
<?
$base="ejemplos";
$c=mysql_connect ("localhost","pepe","pepa");
mysql_select_db ($base, $c);
#####
# Creación de la tabla nombres con indice primario DNI
# tabla de nombres con indice primario en DNI
#####
$crear="CREATE TABLE IF NOT EXISTS alumnos (";
$crear.="DNI CHAR(8) NOT NULL, ";
$crear.="Nombre VARCHAR (20) NOT NULL, ";
$crear.="Apellido1 VARCHAR (15) not null, ";
$crear.="Apellido2 VARCHAR (15) not null, ";
$crear.=" PRIMARY KEY(DNI) ";
$crear.=")";
$crear.=" Type=InnoDB";
if(mysql_query ($crear ,$c)){
    print "tabla <b>nombres</b> creada<BR>";
} else{
    print "ha habido un error al crear la tabla <b>alumnos</b><BR>";
}
#####
# Creación de la tabla direcciones
# tabla de nombres con índice único en DNI
# para evitar dos direcciones al mismo alumno
# y clave foránea nombres(DNI)
# para evitar direcciones no asociadas a un alumno
# concreto. Se activa la opción de actualizar
# en cascada y de borrar en cascada
#####
$crear="CREATE TABLE IF NOT EXISTS domicilios (";
$crear.="DNI CHAR(8) NOT NULL, ";
$crear.="calle VARCHAR (20), ";
$crear.="poblacion VARCHAR (20), ";
$crear.="distrito VARCHAR(5), ";
$crear.=" UNIQUE identidad (DNI) , ";
$crear.="FOREIGN KEY (DNI) REFERENCES alumnos(DNI) ";
$crear.="ON DELETE CASCADE ";
$crear.="ON UPDATE CASCADE ";
$crear.=") TYPE = INNODB";
if(mysql_query ($crear ,$c)){
    print "tabla <b>domicilios</b> creada<br>";
} else{
    print "ha habido un error al crear la tabla <b>domicilios</b><BR>";
}
#####
# Creación de la tabla evaluaciones con indice primario EVALUACIONES
# tabla de nombres con índice primario en NUMERO
#####
$crear="CREATE TABLE IF NOT EXISTS evaluaciones (";
$crear.="NUMERO CHAR(1) NOT NULL, ";
$crear.="nombre_evaluacion VARCHAR (20) NOT NULL, ";
$crear.=" PRIMARY KEY(NUMERO) ";
$crear.=")";
$crear.=" Type=InnoDB";
if(mysql_query ($crear ,$c)){
    print "tabla <b>evaluaciones</b> creada<BR>";
} else{
    print "ha habido un error al crear la tabla <b>evaluaciones</b><BR>";
}
#####
# Creación de la tabla notas
# indice UNICO para los campos DNI y evaluacion
# con ello se impide calificar dos veces al mismo
#####
```

clave foránea en la tabla **notas**.

La tabla **notas** va a tener tres campos: DNI, nº de evaluación y calificación.

Estableceremos restricciones para **evitar que**:

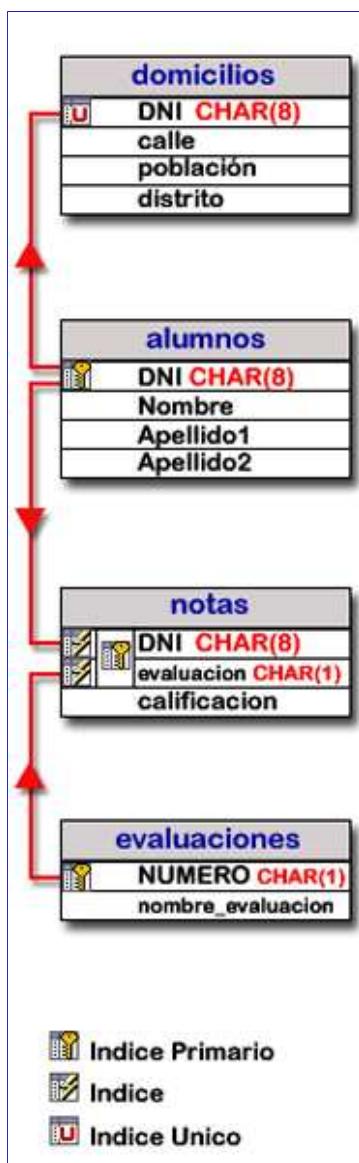
Podamos calificar a un alumno inexistente.

Podamos calificar una evaluación no incluida entre las previstas.

Podamos poner más a cada alumnos más de una calificación por evaluación.

Creando una índice primario formado por los campos DNI y evaluación evitaremos la última de las situaciones y añadiendo una vinculación con las tablas alumnos y evaluaciones estaremos en condiciones de evitar las dos primeras.

En este gráfico puedes ver un esquema de la definición de estas tablas.



## Importación y exportación de datos

Es esta una opción interesante por la posibilidad que ofrece de

```
# alumno en la misma evaluacion
# claves foráneas (DOS)
# el DNI de nombres para evitar calificar a alumnos inexistentes
# el NUMERO de la tabla evaluaciones para evitar calificar
# DOS VECES en una evaluación a un alumno
#####
$crear="CREATE TABLE IF NOT EXISTS notas (";
$crear.="DNI CHAR(8) NOT NULL, ";
$crear.="evaluacion CHAR (1) NOT NULL, ";
$crear.="calificacion TINYINT (2), ";
/* observa que este indice primario está formado
   por dos campos (DNI y evaluacion) y que, como siempre
   en el caso de PRIMARY KEY ambos son de tipo NOT NULL */
$crear.=" PRIMARY KEY vemaos(DNI,evaluacion), ";
/* Fijate en la secuencia siguiente:
   1º.- Creamos el índice
   2º.- Establecemos la clave foránea
   3º.- Establecemos las condiciones ON DELETE
   4º.- Establecemos las condiciones ON UPDATE
   Es muy importante mantener esta secuencia para evitar
   errores MySQL */
$crear.=" INDEX identico (DNI), ";
$crear.="FOREIGN KEY (DNI) REFERENCES alumnos (DNI) ";
$crear.="ON DELETE CASCADE ";
$crear.="ON UPDATE CASCADE ";
/* Esta tabla tiene dos claves foráneas asociadas a dos tablas
   la anterior definida sobre alumnos como tabla principal
   y esta que incluimos a continuación asociada con evaluaciones
   Como ves repetimos la secuencia descrita anteriormente
   Es importante establecer estas definiciones de una en una
   (tal como ves en este ejemplo) y seguir la secuencia
   comentada anteriormente */
$crear.=" INDEX evalua (evaluacion), ";
$crear.="FOREIGN KEY (evaluacion) REFERENCES evaluaciones (NUMERO) ";
$crear.="ON DELETE CASCADE ";
$crear.="ON UPDATE CASCADE ";
$crear.=") TYPE = INNODB";
if(mysql_query ($crear , $c)){
    print "tabla <b>notas</b> creada <BR>";
} else{
    print "ha habido un error al crear la tabla <b>notas</b><BR>";
    echo mysql_error ($c). "<br>";
    echo mysql_errno ($c);
}

mysql_close();
?>
```

Crear ejemplos  
tablas vinculadas

## ¡Cuidado!

Como puedes observar en la imagen de la izquierda, al definir la estructura de las tablas es muy importante prestar atención a que los campos vinculados sean del **mismo tipo y dimensión**.

Observa también que los campos de referencia de los vínculos que se establecen (en las tablas primarias) tienen que ser definidos como PRIMARY KEY y que, por tanto, han de establecerse como no nulos (NOT NULL).

## Inserción de datos en tablas

MySQL permite **importar** ficheros externos utilizando la siguiente sintaxis:

```
LOAD DATA INFILE "nombre del fichero' [REPLACE | IGNORE]
INTO TABLE nombre de la tabla
FIELDS
  TERMINATED BY 'indicador de final de campo'
  ENCLOSED BY 'caracteres delimitadores de campos'
  LINES
```

intercambiar datos entre diferentes fuentes y aplicaciones.

## Importación de ficheros

MySQL permite insertar en sus tablas los contenidos de ficheros de texto. Para ello utiliza la sentencia que tienes al margen y que detallaremos a continuación.

### LOAD DATA INFILE

Es un contenido obligatorio que define la opción de insertar datos desde un fichero externo.

#### nombre del fichero

Se incluye inmediatamente después de la anterior, es obligatorio y debe contener (entre comillas) la ruta, el nombre y la extensión del fichero que contiene los datos a insertar.

#### [REPLACE|IGNORE]

Es opcional. Si se omite se producirá un mensaje de error si el fichero contiene valores iguales a los contenidos en los campos de la tabla que no admiten duplicados. Con la opción **REPLACE** sustituiría los valores existentes en la tabla y con la opción **IGNORE**

#### INTO TABLE nombre

Tiene carácter obligatorio y debe incluir como *nombre* el de la tabla a la que se pretende agregar los registros.

#### FIELDS

Tiene carácter OPCIONAL y permite incluir especificaciones sobre cuales son los caracteres delimitadores de campos y los que indican el final del campo. Si se omite **FIELDS** no podrán incluirse los **ENCLOSED BY** ni **TERMINATED BY** de campo.

#### ENCLOSED BY

Permite especificar (encerrados entre comillas) los caracteres delimitadores de los campos. Estos caracteres deberán encontrarse en el fichero original **al principio y al final** de los contenidos de cada uno de los campos (por ejemplo, si el carácter fueran *comillas* los campos deberían aparecer así en el fichero a importar algo como esto: "32.45"). Si se omite esta especificación (o se omite **FIELDS**) se entenderá que los campos **no tienen caracteres delimitadores**.

Cuando se incluyen como delimitadores de campo las comillas (dobles o sencillas) es necesario utilizar una sintaxis como esta: \" ó \' de forma que no queda la ambigüedad de si se trata de un carácter o de las comillas de cierre de una cadena previamente abierta.

#### TERMINATED BY

Se comporta de forma similar al anterior.

Permite qué caracteres son usados en el fichero original como

**STARTING BY** 'caracteres indicadores de comienzo de registro'

**TERMINATED BY** 'caracteres indicadores del final de registro'

En este ejemplo pude un caso práctico de inserción de datos en las tablas creadas anteriormente.

```
<?
$base="ejemplos";
$c=mysql_connect ("localhost","pepe","pepa");
mysql_select_db ($base, $c);
# hemos creado un fichero de texto (datos_alumnos.txt)
# que contiene datos de algunos alumnos. Los diferentes
# campos están entre comillas y separados unos de otros
# mediante un punto y coma.
# Cada uno de los registros comienza por un asterisco
# y los registros están separados por un salto de líneas (\r\n)
# Incluimos estás especificaciones en la sentencia de inserción
$query="LOAD DATA INFILE ";
$query.= "'.$_SERVER['DOCUMENT_ROOT'].'/cursophp/datos_alumnos.txt'.'";
$query.= " REPLACE INTO TABLE alumnos";
$query.= " FIELDS ENCLOSED BY '\"' TERMINATED BY ';' ";
$query.= " LINES STARTING BY '*' TERMINATED BY '\r\n' ";
if(mysql_query($query,$c)){
    print "Datos de alumnos cargados<br>";
    }else{
    print mysql_error ($c)."<br>".mysql_errno ($c);
}
# Para esta tabla usaremos el fichero datos_evaluaciones.txt
# Los diferentes
# campos están entre comillas y separados unos de otros
# mediante una coma.
# Cada uno de los registros comienza por un espacio
# y los registros están separados por un salto de líneas (\r\n)
# Incluimos estás especificaciones en la sentencia de inserción
$query="LOAD DATA INFILE ";
$query.= "'.$_SERVER['DOCUMENT_ROOT'].'/cursophp/datos_evaluaciones.txt'.'";
$query.= " REPLACE INTO TABLE evaluaciones";
$query.= " FIELDS ENCLOSED BY '\"' TERMINATED BY ',' ";
$query.= " LINES STARTING BY ' ' TERMINATED BY '\r\n' ";
if(mysql_query($query,$c)){
    print "Datos de evaluaciones cargados<br>";
    }else{
    print mysql_error ($c)."<br>".mysql_errno ($c);
}
/* En este caso no incluimos especificación alguna.
Bajo este supuesto MySQL interpreta los valores por defecto
que son: los campos no van encerrados, las líneas no tienen
ningún carácter indicador de comienzo, los campos están separados
mediante tabulaciones (carácter de escape \t) y el final de línea
está señalado por un carácter de nueva línea (\n) */
$query="LOAD DATA INFILE ";
$query.= "'.$_SERVER['DOCUMENT_ROOT'].'/cursophp/datos_notas.txt'.'";
$query.= " IGNORE INTO TABLE notas";
if(mysql_query($query,$c)){
    print "Datos de notas cargados<br>";
    }else{
    print mysql_error ($c)."<br>".mysql_errno ($c);
}
/* Se comporta como los casos anteriores con distintos caracteres
para los diferentes eventos, tal como puedes ver en el código */
$query="LOAD DATA INFILE ";
$query.= "'.$_SERVER['DOCUMENT_ROOT'].'/cursophp/datos_domicilios.txt'.'";
$query.= " IGNORE INTO TABLE domicilios";
$query.= " FIELDS ENCLOSED BY '\"' TERMINATED BY '*' ";
$query.= " LINES STARTING BY '#' TERMINATED BY ')' ";
if(mysql_query($query,$c)){
    print "Datos de domicilios cargados
";
    }else{
    print mysql_error ($c)."<br>".mysql_errno ($c);
}
} mysql_close();
?>
```

Cargar datos

Consultar tablas

## Guardar datos en ficheros

separadores de campos (indicador de final de campo). Si se omite, se interpretará con tal el carácter *tabulador* (\t).

El uso de esta opción no requiere que se especifique previamente **ENCLOSED BY** pero si necesita que se haya incluido **FIELDS**.

Si el fichero de datos contiene caracteres (distintos de los valores por defecto) para señalar el comienzo de un registro, el final del mismo o ambos, debe incluirse este parámetro y, después de él, las especificaciones de esos valores correspondientes a:

#### **STARTING BY**

Permite especificar una carácter como indicador de comienzo de un registro. Si se omite o se especifica como " se interpretará que no hay ningún carácter que señale en comienzo de línea.

#### **TERMINATED BY**

Es el indicador del **final de un registro**. Si se omite será considerado como un *salto de línea* (\n).

### **Exportación de ficheros**

Se comporta de forma similar al supuesto anterior. Utiliza la sintaxis siguiente:

#### **SELECT \* INTO OUTFILE**

Inicia la consulta de los campos especificados después de **SELECT** (si se indica \* realiza la consulta sobre todos los campos y por el orden en el que fue creada la tabla) y redirige la salida a un fichero.

#### *nombre del fichero*

Es la ruta, nombre y extensión del fichero en el que serán almacenados los resultados de la consulta.

#### **FIELDS**

#### **ENCLOSED BY** **TERMINATED BY**

#### **LINES**

#### **STARTING BY** **TERMINATED BY**

Igual que ocurría en el caso de importación de datos, estos parámetros son opcionales. Si no se especifican se incluirán los valores por defecto.

#### **FROM nombre**

Su inclusión tiene carácter obligatorio. El valor de *nombre* ha de ser el de la tabla sobre la que se realiza la consulta.

#### **¡Cuidado!**

Al importar ficheros habrá de utilizarse el mismo formato con el que fueron creados tanto **FIELDS** como **LINES**.

MySQL permite los contenidos de sus tablas a ficheros de texto. Para ello utiliza la siguiente sintaxis:

```
SELECT * INTO OUTFILE "nombre del fichero"
FIELDS
TERMINATED BY 'indicador de final de campo'
ENCLOSED BY 'caracteres delimitadores de campos'
LINES
STARTING BY 'caracteres indicadores de comienzo de registro'
TERMINATED BY 'caracteres indicadores del final de registro'
FROM nombre de la tabla
```

```
<?
$base="ejemplos";
$c=mysql_connect ("localhost","pepe","pepa");
mysql_select_db ($base, $c);
$query="SELECT * INTO OUTFILE ";
$query.= "'.$_SERVER['DOCUMENT_ROOT'].'/cursophp/alumnos.txt'.''";
$query.= " FIELDS ENCLOSED BY '\'' TERMINATED BY ';' ";
$query.= " LINES STARTING BY '*' TERMINATED BY '\r\n' ";
$query.= " FROM alumnos";
if(mysql_query($query,$c)){
    print "fichero alumnos.txt creado<br>";
}else{
    print mysql_error ($c). "<br>". mysql_errno ($c);
}
$query="SELECT * INTO OUTFILE ";
$query.= "'.$_SERVER['DOCUMENT_ROOT'].'/cursophp/domicilios.txt'.''";
$query.= " FIELDS ENCLOSED BY '|' TERMINATED BY '*;' ";
$query.= " LINES STARTING BY '#' TERMINATED BY '} ' ";
$query.= " FROM domicilios";
if(mysql_query ($query,$c)){
    print "fichero domicilios.txt creado<br>";
}else{
    print mysql_error ($c). "<br>".mysql_errno ($c);
}
$query="SELECT * INTO OUTFILE ";
$query.= "'.$_SERVER['DOCUMENT_ROOT'].'/cursophp/notas.txt'.''";
$query.= " FROM notas";
if(mysql_query ($query,$c)){
    print "fichero notas.txt creado<br>";
}else{
    print mysql_error ($c). "<br>".mysql_errno ($c);
}
$query="SELECT * INTO OUTFILE ";
$query.= "'.$_SERVER['DOCUMENT_ROOT'].'/cursophp/evaluaciones.txt'.''";
$query.= " FIELDS ENCLOSED BY '\'' TERMINATED BY ',' ";
$query.= " LINES STARTING BY '' TERMINATED BY '\r\n' ";
$query.= " FROM evaluaciones";
if(mysql_query ($query,$c)){
    print "fichero evaluaciones.txt creado<br>";
}else{
    print mysql_error ($c). "<r>".mysql_errno ($c);
}
mysql_close();
?>
```

Crear ficheros de datos

#### **¡Cuidado!**

Al exportar ficheros en entornos Windows, si se pretende que en el fichero de texto aparezca un salto de línea no basta con utilizar la opción por defecto de **LINES TERMINATED BY '\n'** sino **LINES TERMINATED BY '\r\n'** (salto de línea y retorno) que son los caracteres que necesita Windows para producir ese efecto.

Habrá de seguirse este mismo criterio cuando se trata de importar datos desde un fichero de texto.

### **Consultas de unión (JOIN)**

## Consultas usando JOIN

La cláusula JOIN es opción aplicable a consultas en tablas que tiene diversas opciones de uso. Iremos viéndolas de una en una. Todas ellas han de ir incluidas como parámetros de una consulta. Por tanto han de ir precedidas de:

**SELECT \***

o de

**SELECT nom\_tab.nom\_camp,..**

donde *nom\_tab* es un nombre de tabla y *nom\_camp* es el nombre del campo de esa tabla que pretendemos visualizar para esa consulta. Esta sintaxis es idéntica a la ya comentada en páginas anteriores cuando tratábamos de consultas en varias tablas.

Ahora veremos las diferentes posibilidades de uso de **JOIN**

**FROM tb1 JOIN tb2**

Suele definirse como el *producto cartesiano* de los elementos de la primera tabla (*tb1*) por lo de la segunda (*tb2*).

Dicho de una forma más vulgar, esta consulta devuelve con resultado una lista de cada uno de los registros de los registros de la primera tabla asociados sucesivamente con todos los correspondientes a la segunda. Es decir, aparecerá una línea conteniendo el primer registro de la primera tabla seguido del primero de la segunda. A continuación ese mismo registro de la primera tabla acompañado del segundo de la segunda tabla, y así, sucesivamente hasta acabar los registros de esa segunda tabla. En ese momento, repite el proceso con el segundo registro de la primera tabla y, nuevamente, todos los de la segunda. Así, sucesivamente, hasta llegar al último registro de la primera tabla asociado con el último de la segunda.

En total, devolverá un número de líneas igual al resultado de multiplicar el número de registros de la primera tabla por los de la segunda.

**FROM tb2 JOIN tb1**

Si permutamos la posición de las tablas, tal como indicamos aquí, obtendremos el mismo resultado que en el caso anterior pero, como es lógico pensar, con una ordenación diferente de los resultados.

**FROM tb2 JOIN tb1 ON cond**

El parámetro ON permite añadir una condición (*cond*) la consulta de unión. Su comportamiento es idéntico al de WHERE en las consultas ya estudiadas y permite el uso de las mismas procedimientos de establecimiento de condiciones que aquel operador.

```
<?
$base="ejemplos";
$c=mysql_connect ("localhost","pepe","pepa");
mysql_select_db ($base, $c);
# vamos a crear un array con las diferentes consultas
# posteriormente lo leeremos y la ejecutaremos secuencialmente
/* Devuelve todos los campos de ambas tablas.
   Cada registro de alumnos es asociado con todos los de domicilios*/
$query[]="SELECT * FROM alumnos JOIN domicilios";
/* Devuelve todos los campos de ambas tablas. Cada registro de domicilios
   es asociado con todos los de alumnos */
$query[]="SELECT * FROM domicilios JOIN alumnos";
/* Devuelve todos los campos de los registros de ambas tablas
   en los que coinciden los números del DNI*/
$query[]="SELECT * FROM alumnos JOIN domicilios
          ON domicilios.DNI=alumnos.DNI";
/* Idéntica a la anterior. Solo se diferencia en que ahora
   se visualizan antes los campos domicilios*/
$query[]="SELECT * FROM domicilios JOIN alumnos
          ON domicilios.DNI=alumnos.DNI";
/* Devuelve cada uno de los registros de la tabla alumnos. Si existe
   un domicilio con igual DNI lo insertará. Si no existiera
   insertará valores nulos en esos campos
$query[]="SELECT * FROM alumnos LEFT JOIN domicilios
          ON domicilios.DNI=alumnos.DNI";
/* Se comporta de forma idéntica al anterior.
   Ahora insertará todos los registros de domicilios
   y los alumnos coincidentes o en su defecto campos nulos.*/
$query[]="SELECT * FROM domicilios LEFT JOIN alumnos
          ON domicilios.DNI=alumnos.DNI";
/* Al utilizar RIGHT será todos los registros de la tabla de la derecha
   (domicilios) los que aparezcan junto con las coincidencias o
   junto a campos nulos. Aparecerán primero los campos de alumnos
   y detrás los de domicilios*/
$query[]="SELECT * FROM alumnos RIGHT JOIN domicilios
          ON (domicilios.DNI=alumnos.DNI AND alumnos.Nombre LIKE 'A%')";
/* Consulta de nombre, apellido y localidad de todos los alumnos
   cuyo nombre empieza por A */
$query[]="SELECT alumnos.Nombre, alumnos.Apellido1, alumnos.Apellido2,
          domicilios.poblacion FROM alumnos JOIN domicilios
          ON (domicilios.DNI=alumnos.DNI
              AND alumnos.Nombre LIKE 'A%');

# una consulta resumen de nos permitirá visualizar una lista con nombre
# y apellidos de alumnos su dirección y localidad del domicilio
# el nombre de la evaluación y su calificación.
# Si no hay datos de población insertará ---- en vez del valor nulo
# y si no hay calificación en una evaluación aparecerá N.P.
# La consulta aparecerá agrupada por evaluaciones
/* iniciamos el select especificando los campos de las diferentes
   tablas que pretendemos visualizar
$q="(SELECT alumnos.Nombre, alumnos.Apellido1,
      alumnos.Apellido2, domicilios.calle,
      # al incluir IFNULL visualizaremos ---- en los campos cuyo resultado
      # sea nulo
      $q.= IFNULL(domicilios.poblacion,'----'),";
$q.= " evaluaciones.nombre_evaluacion,";
# con este IFNULL aparecerá N.P. en las evaluaciones no calificadas.
$q.= IFNULL(notas.calificacion,'N.P.');
# especificamos el primer JOIN con el que tendremos como resultado una lista
# de todos los alumnos con sus direcciones correspondientes
# por efecto de la cláusula ON.
# Al poner LEFT se incluirían los alumnos que no tuvieran
# su dirección registrada en la tabla de dirección
$q.= " FROM (alumnos LEFT JOIN domicilios",
$q.= "       ON alumnos.DNI=domicilios.DNI)";
# al unir por la izquierda con notas tendríamos todos los resultados
# del JOIN anterior asociados con todas sus calificaciones
# por efecto de la cláusula ON
$q.= "      LEFT JOIN notas ON notas.DNI=alumnos.DNI";
# al añadir esta nueva unión por la DERECHA con la tabla evaluaciones
# se asociaría cada uno de los resultados de las uniones anteriores
# con todos los campos de la tabla evaluaciones con lo que resultaría
# una lista de todos los alumnos con todas las calificaciones
# incluyendo un campo en blanco (sería sustituido por N.P.)
# en aquellas que no tuvieran calificación registrada
$q.= "      RIGHT JOIN evaluaciones";
$q.= "      ON evaluaciones.NUMERO=notas.evalucion";
/* la cláusula WHERE nos permite restringir los resultados a los valores
```

### **FROM tbl1 LEFT JOIN tbl2 ON cond**

Cuando se incluye la cláusula **LEFT** delante de **JOIN** el resultado de la consulta es el siguiente:

– Devolvería cada uno los registros de la tabla especificada a la izquierda de **LEFT JOIN** -sin considerar las restricciones que puedan haberse establecido en las cláusulas **ON** para los valores de esa tabla- asociándolos con aquellos de la otra tabla que cumplan las condiciones establecidas en la cláusula **ON**. Si ningún registro de la segunda tabla cumpliera la condición devolvería valores nulos.

### **FROM tb1 RIGHT JOIN tb2 ON cond**

Se comporta de forma similar al anterior. Ahora los posibles valores nulos serán asignados a la tabla indicada a la izquierda de **RIGHT JOIN** y se visualizarían todos los registros de la tabla indicada a la derecha.

## **JOIN múltiples**

Tal como puedes observar en el ejemplo, es perfectamente factible utilizar conjuntamente varios **JOIN**, **LEFT JOIN** y **RIGHT JOIN**. Las diferentes uniones irán ejecutándose de izquierda a derecha (según el orden en el que estén incluidos en la sentencia) y el resultado del primero será utilizado para la segunda unión y así sucesivamente.

En cualquier caso, es posible alterar ese orden de ejecución estableciendo otras prioridades mediante paréntesis.

## **UNION de consultas**

MySQL permite juntar en una sola salida los resultados de varias consultas. La sintaxis es la siguiente:

**(SELECT ...)**  
**UNION ALL**  
**(SELECT ...)**  
**UNION ALL**  
**(SELECT ...)**

Cada uno de los **SELECT** ha de ir encerrado entre paréntesis.

```

correspondientes únicamente a la evaluación número 1*/  

$qr.=" WHERE evaluaciones.NUMERO=1)";  

# cerramos la consulta anterior con el paréntesis. Observa que lo  

# hemos abierto delante del SELECT e insertamos UNION ALL  

# para que el resultado de la consulta anterior aparezca  

# seguido del correspondiente a la incluida después de UNION ALL  

$qr.=" UNION ALL";  

#iniciamos (también con paréntesis) la segunda consulta  

# que será identica a la anterior salvo el WHERE  

# será modificado para extraer datos de la evaluación nº2  

$qr.="(SELECT alumnos.Nombre,alumnos.Apellido1,";  

$qr.=" alumnos.Apellido2,domicilios.calle,";  

$qr.=" IFNULL(domicilios.poblacion,'----'),";  

$qr.=" evaluaciones.nombre_evaluacion,";  

$qr.=" IFNULL(notas.calificacion,'N.P.')";  

$qr.=" FROM (alumnos LEFT JOIN domicilios";  

$qr.=" ON alumnos.DNI=domicilios.DNI")";  

$qr.=" LEFT JOIN notas ON notas.DNI=alumnos.DNI";  

$qr.=" RIGHT JOIN evaluaciones";  

$qr.=" ON evaluaciones.NUMERO=notas.evalucion";  

$qr.=" WHERE evaluaciones.NUMERO=2");  

# hemos cerrado el parentesis de la consulta anterior  

# e incluimos un nuevo UNION ALL para consultar los datos  

# correspondientes a la tercera evaluación  

$qr.=" UNION ALL";  

$qr.="(SELECT alumnos.Nombre,alumnos.Apellido1,";  

$qr.=" alumnos.Apellido2,domicilios.calle,";  

$qr.=" IFNULL(domicilios.poblacion,'----'),";  

$qr.=" evaluaciones.nombre_evaluacion,";  

$qr.=" IFNULL(notas.calificacion,'N.P.')";  

$qr.=" FROM (alumnos LEFT JOIN domicilios";  

$qr.=" ON alumnos.DNI=domicilios.DNI")";  

$qr.=" LEFT JOIN notas ON notas.DNI=alumnos.DNI";  

$qr.=" RIGHT JOIN evaluaciones";  

$qr.=" ON evaluaciones.NUMERO=notas.evalucion";  

$qr.=" WHERE evaluaciones.NUMERO=3");  

# incluimos la variable $qr en el array de consultas  

$queries[]=$qr;  

# leemos el array y visualizamos el resultado  

# cada consulta a traves de la llamada a la funcion visualiza  

# a la que pasamos el resultado de la consulta  

# y la cadena que contiene las sentencias de dicha consulta  

foreach($query as $v){  

    visualiza(mysql_query($v,$c),$v);  

}  

function visualiza($resultado,$query){  

    PRINT "<BR><BR><i>Resultado de la sentencia:</i><br>";  

    print "<b><font color=#ff0000>";  

    print $query."</font></b><br><br>";  

    PRINT "<table align=center border=2>";  

    while ($registro = mysql_fetch_row($resultado)){  

        echo "<tr>";  

        foreach($registro as $valor){  

            echo "<td>",$valor,"</td>";  

        }  

    }  

    echo "</table><br>";  

}
?>

```

Ejecutar script

Anterior

Índice

Siguiente



## Borrado de usuarios

### Borrado

### de usuarios MySQL

Hasta el momento no hemos modificado ni los usuarios de MySQL ni la configuración de PHPMyAdmin.

Aún tenemos los usuarios por defecto (`root sin contraseña`) y PHPMyAdmin sigue dándonos el mensaje de advertencia relativo al fallo de seguridad de la configuración.

El primer paso para corregir esa vulnerabilidad será borrar los usuarios dejando sólo el usuario **pepe** que es el que hemos creado al instalar MySQL.

Para ello, con el servidor MySQL activo, abriremos [PhpMyAdmin](#) editaremos la tabla `user` correspondiente a la base de datos `mysql` y borraremos los usuarios, dejando únicamente al usuario **pepe**.

Una vez realizados estos cambios, deberíamos cerrar el servidor MySQL y reiniciarlo con la nueva configuración. A partir de aquí, cuando tratamos de acceder de nuevo a PHPMyAdmin nos aparecerá un mensaje de error tal como el que vemos en la imagen de la derecha.

## Modificación de la configuración de PHPMyAdmin

Hemos de modificar el fichero `config.inc.php` que está en el subdirectorio `libraires` de MyAdmin.

Durante el proceso de instalación habíamos hecho algunos cambios en este fichero (algunos de ellos para evitarnos tener que teclear continuamente nombres de usuario y contraseña) que ahora vamos a deshacer.

Al cambiar **config** por **http** (tal como se indica en la tabla de la derecha) estaremos estableciendo este último valor como método de autenticación de los usuarios.

Los otros dos cambios (que ahora hacemos a `false`) significan que no vamos a permitir el acceso sin contraseña a ningún usuario, incluido un eventual usuario `root` sin contraseña que pudiera quedar registrado desde el proceso inicial de instalación.

Borrado de usuarios				
Mostrar: <input type="text" value="30"/> filas empezando de <input type="text" value="0"/> en modo <input type="button" value="Horizontal"/> y repite encabezad				
	Host	User	password	Select_priv
<input type="checkbox"/> <input checked="" type="checkbox"/> <input type="checkbox"/>	localhost	pepe	26629bdc0057ef04	Y
Con marca: <input checked="" type="checkbox"/> <input type="checkbox"/>				

La tabla user debería quedarnos con un único usuario

**Bienvenido a phpMyAdmin 2.8.1**

La razón más probable es que usted no creó un archivo de configuración. Utilice [setup script](#)

**Error**

**X** MySQL ha dicho: [?](#)

#1045 - Access denied for user: 'root@localhost' (Using password: NO)

PHPMyAdmin nos dará este mensaje de error porque su configuración por defecto utiliza el usuario `root` que ya no existe. ¡Acabamos de borrarlo!

## Modificación de config.inc.php

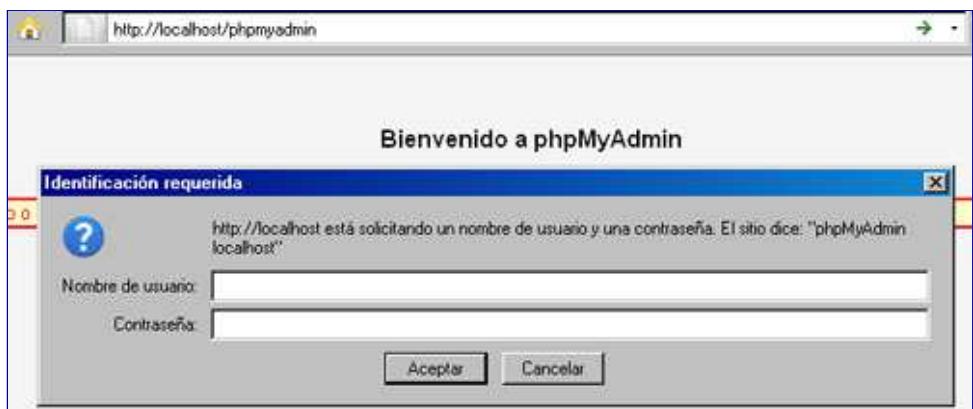
Fichero inicial	<b>config.inc.php</b>
Guardar como	<b>config.inc.php</b>
Línea	Modificaciones en el fichero config.inc.php
	Cambios
168	Donde dice:  cambiar por:  Where it says:  Change to:
	\$cfg['Servers'][\$i]['auth_type'] = 'config';  \$cfg['Servers'][\$i]['auth_type'] = 'http';
218	Donde dice:  cambiar por:  Where it says:  Change to:
	\$cfg['Servers'][\$i]['nopassword'] = true;  \$cfg['Servers'][\$i]['nopassword'] = false;
345	Donde dice:  cambiar por:  Where it says:  Change to:
	\$cfg['Servers'][\$i]['AllowNoPasswordRoot'] = true;

## Acceso a PHPmyAdmin

A partir del momento que hayamos hecho los cambios anteriores, cada vez que tratemos de acceder a <http://localhost/phpmyadmin/> va a aparecer una ventana como la que vemos en la imagen. Será necesario introducir un nombre de usuario válido (pepe en nuestro caso).

Nos desaparecerá el mensaje de advertencia y ya podremos gestionar nuestras bases de datos de forma un poco más segura.

```
$cfg["Servers"][$i]["AllowNoPasswordRoot"] =false;
```



Anterior



Índice



Siguiente





Ver índice

# Índice de actividades



## Índice de instalaciones

Instalación	Curso
Instalación del editor Dev-PHP	Ambos
Instalación del servidor Apache	Ambos
Instalación de PHP	Ambos
Configuración de Apache 1.3.33	Ambos
Configuración de PHP 4.3.11	Ambos
Instalación de MySQL 4.0	Prof.
Instalación de PhpMyAdmin	Prof.
Instalación del servidor FTP	Prof.
Instalación del servidor Correo	Prof.

## Índice de actividades propuestas

Nº	Actividad	Curso
1	Cronómetro autoactualizable en JavaScript	Ambos
2	Práctica de uso de la función print()	Inic.
3	Echo y print() con líneas de comentario	Inic.
4	Constantes numéricas y de cadena	Inic.
5	Distintos valores de las variables	Inic.
6	Número de versión y etiquetas HTML desde PHP	Inic.
7	Variables de variables y caracteres especiales	Inic.
8	Formulario que asigna color de fondo	Inic.
9	Formulario y visor de contenidos	Inic.
10	Formulario con cuestionarios y visor de contenidos	Inic.
11	Formulario con calculadora	Inic.
12	Formulario con calculadora formateando números	Inic.
13	Manejo de cadenas y caracteres especiales	Inic.
14	Formulario que devuelve el resto de la división entre doce en forma literal	Inic.
15	Formateo de números y textos trasnferidos a través de un formulario	Inic.
16	Inserción de saltos de línea automáticos en un texto	Inic.
17	Prácticas con operadores de comparación	Inic.
18	Prácticas con operadores lógicos	Inic.
19	Comprobar si un número es positivo o negativo	Inic.
20	Comprobar el valor de una opción	Inic.
21	Comprobar los valores de una opción y de casillas de verificación	Inic.
22	Tabla con una escala de grises	Inic.
23	Construir una tabla con cinco filas y siete columnas mediante bucles	Inic.
24	Seis números de la primitiva	Inic.
25	Listado de calificaciones a partir de los datos de un array bidimensional	Inic.
26	Contruir una tabla mediante una función	Inic.
27	Ejemplo de include	Inic.

<b>28</b>	Fecha y hora de la última visita	Inic.
<b>29</b>	Transferencia de imágenes y datos	Inic.
<b>30</b>	Compresión y visualización de una imagen comprimida	Prof.
<b>31</b>	Transferir ficheros -mediante formularios- al servidor FTP	Prof.
<b>32</b>	Envío de un mensaje de correo automático al cargarse una página	Prof.
<b>33</b>	Diagrama de sectores a partir de los datos de un formulario	Prof.
<b>34</b>	Recortar una imagen y colocar un texto sobre ella	Prof.
<b>35</b>	Crear un fichero .htaccess para <i>página no encontrada</i>	Prof.
<b>36</b>	Propagar mediante variables de sesión el nombre y color.	Prof.
<b>37</b>	Crear documentos PDF con imágenes y textos	Prof.
<b>38</b>	Crear tabla MySQL con datos indicados	Prof.
<b>39</b>	Crear una tabla MySQL definiendo sus propios campos	Prof.
<b>40</b>	Crear formulario para añadir registros a una tabla	Prof.
<b>41</b>	Desarrollar diversas consultas en la tabla de alumnos	Prof.
<b>42</b>	Crear formularios para insertar calificaciones por materias	Prof.
<b>43</b>	Formulario y script para modificar datos de alumnos	Prof.

---

Anterior

Índice

Siguiente